사용자 가이드

Amazon Athena

Amazon Athena: 사용자 가이드

Table of Contents

Amazon Athena란 무엇인가요?

Amazon Athena는 표준 SQL을 사용하여 Amazon S3(Amazon Simple Storage Service)에 있는 데이터를 직접 간편하게 분석할 수 있는 대화형 쿼리 서비스입니다. AWS Management Console에서 몇 가지 작업을 수행하면 Athena에서 Amazon S3에 저장된 데이터를 지정하고 표준 SQL을 사용하여 임시 쿼리를 실행하여 몇 초 안에 결과를 얻을 수 있습니다.

자세한 내용은 시작하기 단원을 참조하십시오.

또한 Amazon Athena를 사용하면 리소스를 계획, 구성 또는 관리할 필요 없이 Apache Spark를 사용하여 데이터 분석을 대화식으로 쉽게 실행할 수 있습니다. Athena에서 Apache Spark 애플리케이션을 실행하는 경우 처리를 위해 Spark 코드를 제출하고 결과를 직접 수신합니다. Amazon Athena 콘솔의 간소화된 노트북 환경을 활용하면 Python 또는 Athena 노트북 API를 통해 Apache Spark 애플리케이션을 개발할 수 있습니다.

자세한 내용은 Amazon Athena에서 Apache Spark 시작하기 단원을 참조하십시오.

Amazon Athena의 Athena SQL 및 Apache Spark는 서버리스 서비스이므로 설정하거나 관리할 인프라가 없으며 실행한 쿼리에 대해서만 비용을 지불하면 됩니다. Athena는 자동으로 확장되어 쿼리를 병렬로 실행하여 대규모 데이터 세트과 복잡한 쿼리에서도 빠르게 결과를 얻을 수 있습니다.

주제

Athena는 언제 사용해야 하나요?

Amazon Athena와 같은 쿼리 서비스, Amazon Redshift와 같은 데이터 웨어하우스, Amazon EMR과 같은 정교한 데이터 처리 프레임워크는 모두 다양한 요구와 사용 사례를 다룹니다. 다음 지침은 요구 사항에 따라 하나 이상의 서비스를 선택하는 데 도움이 될 수 있습니다.

Amazon Athena

Athena는 Amazon S3에 저장된 비정형, 반정형 및 정형 데이터를 분석하는 데 도움을 줍니다. 예를 들면 CSV, JSON 또는 컬럼 방식 데이터 형식(예: Apache Parquet 및 Apache ORC)이 해당됩니다.

Athena를 사용하면 데이터를 집계하거나 Athena로 로드할 필요 없이 ANSI SQL을 사용한 임의 쿼리를 실행할 수 있습니다.

Athena는 간편한 데이터 시각화를 위해 Amazon QuickSight와 통합되었습니다. Athena를 사용하여 보고서를 생성하거나 JDBC 또는 ODBC 드라이버를 통해 연결된 비즈니스 인텔리전스 도구 또는 SQL 클라이언트로 데이터를 탐색할 수 있습니다. 자세한 내용은 Amazon QuickSight 사용 설명서의 Amazon QuickSight란 무엇입니까? 및 ODBC 및 JDBC 드라이버로 Amazon Athena에 연결를 참조하세요.

Athena는 Amazon S3에서 데이터에 대한 지속적 메타데이터 스토어를 제공하는 AWS Glue Data Catalog와 통합됩니다. 이렇게 하면 Amazon Web Services 계정 전체에서 사용할 수 있는 중앙 메타데이터 스토어를 기반으로 Athena에서 테이블과 쿼리 데이터를 생성하고 AWS Glue의 ETL 및 데이터 검색 기능을 통합할 수 있습니다. 자세한 내용은 AWS Glue 개발자 안내서의 AWS Glue와 통합 및 AWS Glue란 무엇입니까?를 참조하세요.

Amazon Athena를 사용하면 데이터를 포맷하거나 인프라를 관리할 필요 없이 Amazon S3 직접 데이터에 대해 대화형 쿼리를 쉽게 실행할 수 있습니다. 예를 들어, Athena는 웹 로그에서 빠른 쿼리를 실행하여 사이트의 성능 문제를 해결하려는 경우에 유용합니다. Athena를 사용하면 빠르게 시작할 수 있습니다. 데이터에 대한 테이블을 정의하고 표준 SQL을 사용하여 쿼리를 시작하기만 하면 됩니다.

인프라 또는 클러스터를 관리할 필요 없이 Amazon S3의 데이터에 대해 대화형 임시 SQL 쿼리를 실행하려면 Amazon Athena를 사용해야 합니다. Amazon Athena는 서버를 설정하거나 관리할 필요 없이 Amazon S3 데이터에 대한 임시 쿼리를 실행할 수 있는 가장 쉬운 방법을 제공합니다.

Athena가 활용하거나 통합하는 AWS 서비스의 목록은 the section called “Athena와 AWS 서비스 통합” 섹션을 참조하세요.

Amazon EMR

Amazon EMR을 사용하면 온프레미스 배포와 비교할 때 Hadoop, Spark 및 Presto와 같이 고도로 분산된 처리 프레임워크를 간단하고 비용 효율적으로 실행할 수 있습니다. Amazon EMR은 유연합니다. 사용자 지정 애플리케이션과 코드를 실행하고 특정 컴퓨팅, 메모리, 스토리지 및 애플리케이션 파라미터를 정의하여 분석 요구 사항을 최적화할 수 있습니다.

Amazon EMR은 SQL 쿼리를 실행하는 것 외에도 기계 학습, 그래프 분석, 데이터 변환, 데이터 스트리밍 및 사실상 코딩 가능한 거의 모든 업무의 애플리케이션을 위한 다양한 확장 데이터 처리 작업을 실행할 수 있습니다. 사용자 지정 코드를 사용하여 Spark, Hadoop, Presto 또는 Hbase 등의 최신 빅 데이터 처리 프레임워크를 사용하여 매우 큰 데이터 세트를 처리하고 분석하는 경우 Amazon EMR을 사용해야 합니다. Amazon EMR을 사용하면 클러스터와 클러스터에 설치된 소프트웨어의 구성을 완벽하게 제어할 수 있습니다.

Amazon Athena를 사용하면 Amazon EMR을 사용하여 처리하는 데이터를 쿼리할 수 있습니다. Amazon Athena는 Amazon EMR과 동일한 여러 데이터 형식을 지원합니다. Athena의 데이터 카탈로그는 Hive 메타스토어와 호환됩니다. EMR을 사용하고 이미 Hive 메타스토어가 있는 경우, Amazon Athena에서 DDL 문을 실행하고 Amazon EMR 작업에 영향을 주지 않고 즉시 데이터를 쿼리할 수 있습니다.

Amazon Redshift

Amazon Redshift와 같은 데이터 웨어하우스는 인벤토리 시스템, 금융 시스템, 소매 판매 시스템 등 다양한 소스의 데이터를 공통 형식으로 가져와서 장기간 저장해야 할 때 가장 좋은 선택입니다. 과거 데이터에서 정교한 비즈니스 보고서를 작성하려는 경우 Amazon Redshift와 같은 데이터 웨어하우스가 최상의 선택입니다. Amazon Redshift의 쿼리 엔진은 다수의 매우 큰 데이터베이스 테이블을 조인하는 복잡한 쿼리를 실행할 때 특히 효과적으로 수행되도록 최적화되었습니다. 다수의 매우 큰 테이블의 조인이 필요한 고도로 구조화된 데이터에 대해 쿼리를 실행해야 하는 경우 Amazon Redshift Redshift를 선택하는 것이 좋습니다.

Athena를 사용하는 상황에 대한 자세한 내용은 다음 리소스를 참조하세요.

- 시작하기 리소스 센터의 Decision guide for analytics services on AWS
- Amazon Athena FAQ의 Athena와 여타 빅 데이터 서비스의 사용 사례 비교
- Amazon Athena 개요
- Amazon Athena 기능
- Amazon Athena FAQ
- Amazon Athena 블로그 게시물

Athena와 AWS 서비스 통합

Athena를 사용하여 이 섹션에 나열된 AWS 서비스에서 데이터를 쿼리할 수 있습니다. 각 서비스에서 지원하는 리전을 확인하려면 Amazon Web Services 일반 참조의 Regions and endpoints를 참조하세요.

Athena와 통합된 AWS 서비스

- AWS CloudFormation
- Amazon CloudFront
- AWS CloudTrail

- Amazon DataZone
- Elastic Load Balancing
- Amazon EMR Studio
- AWS Glue Data Catalog
- AWS Identity and Access Management(IAM)
- Amazon QuickSight
- Amazon S3 인벤토리
- AWS Step Functions
- AWS Systems Manager 인벤토리
- Amazon Virtual Private Cloud

각 통합에 대한 자세한 내용은 다음 단원을 참조하세요.

AWS CloudFormation

용량 예약

참조 주제: AWS CloudFormation 사용 설명서의 AWS::Athena::CapacityReservation

제공된 이름 및 요청된 데이터 처리 장치 수를 사용하여 용량 예약을 지정합니다. 자세한 내용은 Amazon Athena 사용 설명서의 쿼리 처리 용량 관리 섹션 및 Amazon Athena API 참조의 CreateCapacityReservation을 참조하세요.

데이터 카탈로그

참조 주제: AWS CloudFormation 사용 설명서의 AWS::Athena::DataCatalog

이름, 설명, 유형, 파라미터 및 태그를 포함하여 Athena 데이터 카탈로그를 지정합니다. 자세한 내용은 Amazon Athena 사용 설명서의 테이블, 데이터베이스, 데이터 카탈로그 이해 섹션 및 Amazon Athena API 참조의 CreateDataCatalog를 참조하세요.

명명된 쿼리

참조 주제: AWS CloudFormation 사용 설명서의 AWS::Athena::NamedQuery

AWS CloudFormation을(를) 사용하여 명명된 쿼리를 생성하고 Athena에서 실행할 수 있습니다. 명명된 쿼리를 사용하면 쿼리 이름을 쿼리에 매핑한 다음 Athena 콘솔에서 저장된 쿼리로 이를 실행할 수 있습니다. 자세한 내용은 Amazon Athena 사용 설명서의 저장된 쿼리 사용 섹션 및 Amazon Athena API 참조의 CreateNamedQuery를 참조하세요.

준비된 문

참조 주제: AWS CloudFormation 사용 설명서의 AWS::Athena::PreparedStatement

Athena에서 SQL 쿼리와 함께 사용할 준비된 문을 지정합니다. 준비된 문 에는 실행 시 값이 제공되는 파라미터 자리 표시자가 포함됩니다. 자세한 내용은 Amazon Athena 사용 설명서의 파라미터화된 쿼리 사용 섹션 및 Amazon Athena API 참조의 CreatePreparedStatement를 참조하세요.

작업 그룹

참조 주제: AWS CloudFormation 사용 설명서의 AWS::Athena::WorkGroup

AWS CloudFormation을(를) 사용하여 Athena 작업 그룹을 지정합니다. Athena 작업 그룹을 사용하여 사용자 또는 사용자 그룹에 대한 쿼리를 동일한 계정의 다른 쿼리와 분리합니다. 자세한 내용은 Amazon Athena API 참조에서 Amazon Athena 사용 설명서 및 CreateWorkGroup의 작업 그룹을 사용하여 쿼리 액세스 및 비용 제어를 참조하세요.

Amazon CloudFront

참조 항목: Amazon CloudFront 로그 쿼리

Athena를 사용하여 Amazon CloudFront 로그에 쿼리합니다. CloudFront 사용 방법에 대한 자세한 내용은 Amazon CloudFront 개발자 안내서를 참조하세요.

AWS CloudTrail

참조 항목: AWS CloudTrail 로그 쿼리

CloudTrail 로그와 함께 Athena를 사용하면 AWS 서비스 활동에 대한 분석 기능을 확실하게 향상시킬 수 있습니다. 예를 들어 쿼리를 사용하여 트렌드를 식별하고 소스 IP 주소나 사용자 등의 속성별로 활동을 추가로 격리할 수 있습니다. CloudTrail 콘솔에서 직접 로그를 쿼리할 수 있도록 테이블을 자동 생성하고 이러한 테이블을 사용하여 Athena에서 쿼리를 실행할 수 있습니다. 자세한 내용은 CloudTrail 콘솔을 사용하여 CloudTrail 로그용 Athena 테이블 생성 단원을 참조하십시오.

Amazon DataZone

참조 항목: Athena에서 Amazon DataZone 사용

Amazon DataZone을 사용하여 조직 경계를 넘어 대규모로 데이터를 공유, 검색 및 발견합니다. DataZone은 Athena, AWS Glue 및 AWS Lake Formation과 같은 AWS 분석 서비스 전반의 경험을 단순화합니다. 다양한 데이터 소스에 대량의 데이터가 있는 경우 Amazon DataZone을 사용하여 사람, 데이터 및 도구의 그룹화를 기반으로 비즈니스 사용 사례를 구축할 수 있습니다.

Athena에서는 쿼리 편집기를 사용하여 DataZone 환경에 액세스하고 쿼리할 수 있습니다. 자세한 내용은 Athena에서 Amazon DataZone 사용 단원을 참조하십시오.

Elastic Load Balancing

참조 항목: Application Load Balancer 로그 쿼리

Application Load Balancer 로그를 쿼리하면 Elastic Load Balancing 인스턴스 및 백엔드 애플리케이션과 주고 받는 트래픽, 지연 시간 및 바이트의 소스를 볼 수 있습니다. 자세한 내용은 Application Load Balancer 로그 쿼리 단원을 참조하십시오.

참조 항목: Classic Load Balancer 로그 쿼리

Classic Load Balancer 로그를 쿼리하여 Elastic Load Balancing 인스턴스 및 백엔드 애플리케이션과의 송수신 트래픽 패턴을 분석하고 이해합니다. 트래픽 소스, 지연 시간 및 전송된 바이트를 확인할 수 있습니다. 자세한 내용은 ELB 로그의 테이블 생성을 참조하세요.

Amazon EMR Studio

참조 주제: Use the Amazon Athena SQL editor in EMR Studio

EMR Studio에서 Athena를 사용하여 대화형 쿼리를 개발하고 실행할 수 있습니다. 이를 통해 Spark, Scala 및 기타 워크로드에 사용하는 것과 동일한 Amazon EMR 인터페이스에서 Athena의 SQL 분석에 EMR Studio를 사용할 수 있습니다. EMR Studio에 Athena 통합을 통해 다음 작업을 수행할 수 있습니다.

- Athena SQL 쿼리 수행
- 쿼리 결과 보기
- 쿼리 기록 보기
- 저장된 쿼리 보기
- 파라미터화된 쿼리 수행
- 데이터 카탈로그의 데이터베이스, 테이블 및 뷰 보기

Amazon EMR Studio에서는 다음 Athena 기능을 사용할 수 없습니다.

- Athena 작업 그룹, 데이터 소스 또는 용량 예약 생성 또는 업데이트와 같은 관리 기능
- Athena for Spark 또는 Spark 노트북
- DataZone 통합
- Step Functions

Athena와의 EMR Studio 통합은 EMR Studio 및 Athena를 사용할 수 있는 모든 AWS 리전에서 사용할 수 있습니다. EMR Studio에서 Athena를 사용하는 방법에 대한 자세한 내용은 Amazon EMR 관리 안내서의 Use the Amazon Athena SQL editor in EMR Studio를 참조하십시오.

AWS Glue Data Catalog

참조 항목: AWS Glue와의 통합

Athena는 Amazon S3에서 데이터에 대한 지속적 메타데이터 스토어를 제공하는 AWS Glue Data Catalog와 통합됩니다. 이렇게 하면 Amazon Web Services 계정 전체에서 사용할 수 있는 중앙 메타데이터 스토어를 기반으로 Athena에서 테이블과 쿼리 데이터를 생성하고 AWS Glue의 ETL 및 데이터 검색 기능을 통합할 수 있습니다. 자세한 내용은 AWS Glue 개발자 안내서의 AWS Glue이란 무엇입니까? 및 AWS Glue와의 통합 단원을 참조하세요.

AWS Identity and Access Management(IAM)

참조 항목: Amazon Athena에 사용되는 작업

IAM 권한 정책에서 Athena API 작업을 사용할 수 있습니다. 자세한 내용은 Amazon Athena에 사용되는 작업 및 Athena의 자격 증명 및 액세스 관리 단원을 참조하세요.

Amazon QuickSight

참조 항목: ODBC 및 JDBC 드라이버로 Amazon Athena에 연결

Athena는 간편한 데이터 시각화를 위해 Amazon QuickSight와 통합되었습니다. Athena를 사용하여 보고서를 생성하거나 JDBC 또는 ODBC 드라이버를 통해 연결된 비즈니스 인텔리전스 도구 또는 SQL 클라이언트로 데이터를 탐색할 수 있습니다. Amazon QuickSight에 대한 자세한 내용은 Amazon QuickSight 사용 설명서의 Amazon QuickSight란 무엇입니까?를 참조하세요. Athena와 함께 JDBC 및 ODBC를 사용하는 방법에 대한 내용은 ODBC 및 JDBC 드라이버로 Amazon Athena에 연결을 참조하세요.

Amazon S3 인벤토리

참조 항목: Amazon Simple Storage Service 개발자 가이드에서 Athena로 인벤토리 쿼리를 참조하세요.

Amazon Athena를 사용하여 표준 SQL로 Amazon S3 인벤토리를 쿼리할 수 있습니다. Amazon S3 인벤토리를 사용하면 비즈니스, 규정 준수 및 규제 요건에 관하여 객체의 복제 및 암호화 상태를 감사하고 보고할 수 있습니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 Amazon S3 인벤토리를 참조하세요.

AWS Step Functions

참조 항목: AWS Step Functions 개발자 가이드의 Step Functions로 Athena 호출

AWS Step Functions를 사용하여 Athena를 호출합니다. AWS Step Functions는 Amazon States Language를 사용하여 AWS 서비스를 직접 제어할 수 있습니다. Athena와 함께 Step Functions를 사용하면 쿼리 실행을 시작 및 중지하거나, 쿼리 결과를 가져오거나, 임시 또는 예약된 데이터 쿼리를 실행하거나, Amazon S3의 데이터 레이크에서 결과를 검색할 수 있습니다. Step Functions 역할에 Athena를 사용할 수 있는 권한이 있어야 합니다. 자세한 내용은 AWS Step Functions 개발자 안내서를 참조하세요.

비디오: AWS Step Functions를 사용한 Amazon Athena 쿼리 오케스트레이션

다음 동영상에서는 Amazon Athena를 사용하는 방법과, AWS Step Functions를 사용하여 정기적으로 예약된 Athena 쿼리를 실행하고 해당 보고서를 생성하는 방법을 보여줍니다.

AWS Step Functions를 사용한 Amazon Athena 쿼리 오케스트레이션

AWS Glue DataBrew, Athena 및 Amazon QuickSight의 오케스트레이션을 위해 Step Functions 및 Amazon EventBridge를 사용하는 예제는 AWS 빅데이터 블로그에서 AWS Step Functions를 사용한 AWS Glue DataBrew 작업과 Amazon Athena 쿼리 오케스트레이션을 참조하세요.

AWS Systems Manager 인벤토리

참조 항목: AWS Systems Manager 사용 설명서의 여러 리전 및 계정에서 인벤토리 데이터 쿼리

AWS Systems Manager 인벤토리가 Amazon Athena와 통합되어 여러 AWS 리전 및 계정에서 인벤토리 데이터를 쿼리하는 데 도움이 됩니다. 자세한 내용은 AWS Systems Manager 사용 설명서를 참조하세요.

Amazon Virtual Private Cloud

참조 항목: Amazon VPC 흐름 로그 쿼리

Amazon Virtual Private Cloud 흐름 로그는 VPC의 네트워크 인터페이스에서 송수신되는 IP 트래픽에 대한 정보를 수집합니다. Athena에서 로그를 쿼리하여 네트워크 트래픽 패턴을 조사하고 Amazon VPC 네트워크에서 위협 및 위험을 식별하세요. Amazon VPC에 대한 자세한 내용은 Amazon VPC 사용 설명서를 참조하세요.

설정

Amazon Web Services에 이미 가입한 경우 Amazon Athena를 즉시 사용할 수 있습니다. AWS에 가입하지 않았거나 시작하는 데 도움이 필요하다면 다음 작업을 완료하세요.

AWS 계정에 등록

AWS 계정이 없는 경우 다음 절차에 따라 계정을 생성합니다.

AWS 계정에 등록하려면

1. https://portal.aws.amazon.com/billing/signup을 여세요.
2. 온라인 지시 사항을 따르세요.

   등록 절차 중에는 전화를 받고 키패드로 인증 코드를 입력하는 과정이 있습니다.

   AWS 계정에 가입하면 AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스 액세스 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS는 가입 절차 완료된 후 사용자에게 확인 이메일을 전송합니다. 언제든지 https://aws.amazon.com/으로 가서 내 계정(My Account)을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리 액세스 권한이 있는 사용자 생성

AWS 계정에 가입하고 AWS 계정 루트 사용자에 보안 조치를 한 다음, AWS IAM Identity Center를 활성화하고 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 생성합니다.

귀하의 AWS 계정 루트 사용자 보호

1. 루트 사용자를 선택하고 AWS 계정이메일 주소를 입력하여 AWS Management Console에 계정 소유자로 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정루트 사용자용 가상 MFA 디바이스 활성화(콘솔)를 참조하세요.

관리 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center 설정을 참조하세요.
2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   IAM Identity Center 디렉터리를 ID 소스로 사용하는 방법에 대한 자습서는 AWS IAM Identity Center 사용 설명서의 기본 IAM Identity Center 디렉터리로 사용자 액세스 구성을 참조하세요.

관리 액세스 권한을 가진 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM IDentity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

  IAM Identity Center 사용자로 로그인하는 데 도움이 필요한 경우 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요.

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 권한 세트 생성을 참조하세요.
2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 그룹 추가를 참조하세요.

프로그래밍 방식 액세스 권한 부여

사용자가 AWS Management Console 외부에서 AWS와 상호 작용하려면 프로그래밍 방식의 액세스가 필요합니다. 프로그래밍 방식으로 액세스를 부여하는 방법은 AWS에 액세스하는 사용자 유형에 따라 다릅니다.

사용자에게 프로그래밍 방식 액세스 권한을 부여하려면 다음 옵션 중 하나를 선택합니다.

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
작업 인력 ID (IAM Identity Center가 관리하는 사용자)	임시 보안 인증 정보를 사용하여 AWS CLI, AWS SDK 또는 AWS API에 대한 프로그래밍 요청에 서명합니다.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • AWS CLI에 대해서는 AWS Command Line Interface 사용 설명서에서 AWS IAM Identity Center을 사용하도록 AWS CLI 구성을 참조하세요. • AWS SDK, 도구,AWS API에 대해서는 AWS SDK 및 도구 참조 가이드에서 IAM Identity Center 인증을 참조하세요.
IAM	임시 보안 인증 정보를 사용하여 AWS CLI, AWS SDK 또는 AWS API에 대한 프로그래밍 요청에 서명합니다.	IAM 사용 설명서의 AWS 리소스와 함께 임시 보안 인증 정보 사용에 나와 있는 지침을 따르세요.
IAM	(권장되지 않음) 장기 보안 인증 정보를 사용하여 AWS CLI, AWS SDK 또는 AWS API에 대한 프로그래밍 요청에 서명합니다.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • AWS CLI에 대해서는 AWS Command Line Interface 사용 설명서에서 IAM 사용자 보안 인증 정보를 사용한 인증을 참조하세요. • AWS SDK와 도구에 대해서는 AWS SDK 및 도구 참조 가이드에서 장기 보안 인증 정보를 사용한 인증을 참조하세요.

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
		• AWS API에 대해서는 IAM 사용 설명서에서 IAM 사용자의 액세스 키 관리를 참조하세요.

Athena에 대한 관리형 정책 연결

Athena 관리형 정책은 Athena 기능을 사용할 수 있는 권한을 부여합니다. 사용자가 Athena를 사용하기 위해 맡을 수 있는 하나 이상의 IAM 역할에 이러한 관리형 정책을 연결할 수 있습니다.

IAM 역할은 계정에 생성할 수 있는, 특정 권한을 지닌 IAM 자격 증명입니다. IAM 역할은 AWS에서 자격 증명으로 할 수 있는 것과 없는 것을 결정하는 권한 정책을 갖춘 AWS 자격 증명이라는 점에서 IAM 사용자와 유사합니다. 그러나 역할은 한 사람하고만 연관되지 않고 해당 역할이 필요한 사람이라면 누구든지 맡을 수 있어야 합니다. 또한 역할에는 그와 연관된 암호 또는 액세스 키와 같은 표준 장기 자격 증명이 없습니다. 대신에 역할을 맡은 사람에게는 해당 역할 세션을 위한 임시 보안 자격 증명이 제공됩니다.

역할에 대한 자세한 내용은 IAM 사용 설명서에서 IAM 역할 및 IAM 역할 만들기를 참조하세요.

Athena에 대한 액세스 권한을 부여하는 역할을 생성하려면 Athena 관리형 정책을 해당 역할에 연결해야 합니다. Athena에 대한 두 가지 관리형 정책으로 `AmazonAthenaFullAccess`와 `AWSQuicksightAthenaAccess`가 있습니다. 이러한 정책은 Amazon S3를 쿼리하고, 사용자를 대신하여 별도의 버킷에 쿼리의 결과를 작성할 수 있는 권한을 Athena에 부여합니다. Athena에 대한 이러한 정책의 내용을 확인하려면 Amazon Athena의 AWS 관리형 정책 단원을 참조하세요.

Athena 관리형 정책을 역할에 연결하는 단계는 IAM 사용 설명서의 IAM 자격 증명 권한 추가(콘솔)를 따릅니다. 이에 따라 `AmazonAthenaFullAccess` 및 `AWSQuicksightAthenaAccess` 관리형 정책을 생성한 IAM 역할에 연결합니다.

> **Note**
>
> Amazon S3의 기본 데이터 세트에 액세스하려면 추가 권한이 필요할 수 있습니다. 계정 소유자가 아니거나 다른 방법으로 버킷에 대한 액세스를 제한한 경우 리소스 기반 버킷 정책을 사용하여 액세스를 부여하려면 버킷 소유자에게 연락하고, 역할 기반 정책을 사용하여 액세스를 부여하려면 계정 관리자에게 문의하세요. 자세한 내용은 Amazon S3에 액세스 단원을 참조하십

시오. 데이터 세트 또는 Athena 쿼리 결과가 암호화되어 있다면 추가 권한이 필요할 수 있습니다. 자세한 내용은 저장 중 암호화 단원을 참조하십시오.

Athena 액세스

AWS Management Console, JDBC 또는 ODBC 연결, Athena API, Athena CLI, AWS SDK 또는 AWS Tools for Windows PowerShell를 사용해 Athena에 액세스할 수 있습니다.

- 콘솔에서 Athena SQL을 사용하려면 시작하기 단원을 참조하세요.
- Python을 사용하는 Jupyter 호환 노트북 및 Apache Spark 애플리케이션을 생성하려면 Amazon Athena에서 Apache Spark 사용 섹션을 참조하세요.
- JDBC 또는 ODBC 드라이버를 사용하는 방법은 JDBC로 Amazon Athena에 연결 및 ODBC로 Amazon Athena에 연결 단원을 참조하세요.
- Athena API를 사용하려면 Amazon Athena API 참조를 참조하세요.
- CLI를 사용하려면 AWS CLI를 설치한 다음, 명령줄에서 `aws athena help`를 입력하여 사용 가능한 명령을 확인합니다. 사용 가능한 명령에 대한 내용은 Amazon Athena 명령줄 레퍼런스를 참조하세요.
- AWS SDK for Java 2.x을(를) 사용하려면 AWS SDK for Java 2.x API 참조의 Athena 단원, GitHub.com의 Athena Java V2 Examples, AWS SDK for Java 2.x 개발자 안내서를 참조하세요.
- AWS SDK for .NET을(를) 사용하려면 AWS SDK for .NET API 참조의 `Amazon.Athena` 네임스페이스, GitHub.com의 .NET Athena examples, AWS SDK for .NET 개발자 안내서를 참조하세요.
- AWS Tools for Windows PowerShell을(를) 사용하려면 AWS Tools for PowerShell - Amazon Athena cmdlet 참조, AWS Tools for PowerShell 포털 페이지, AWS Tools for Windows PowerShell 사용 설명서를 참조하세요.
- 프로그래밍 방식으로 연결할 수 있는 Athena 서비스 엔드포인트에 대한 자세한 내용은 Amazon Web Services 일반 참조의 Amazon Athena endpoints and quotas를 참조하세요.

Athena SQL 사용

Athena SQL을 사용하면 다른 데이터 소스에 대해 사전 구축된 다양한 커넥터를 통해 AWS Glue Data Catalog, 외부 Hive 메타스토어 또는 연합 쿼리를 사용하여 Amazon S3의 현재 위치에서 데이터를 쿼리할 수 있습니다.

다른 방법:

- Athena의 JDBC 및 ODBC 드라이버를 사용해 비즈니스 인텔리전스 도구 및 기타 애플리케이션에 연결합니다.
- AWS 서비스 로그를 쿼리합니다.
- Apache Iceberg 테이블(시간 이동 쿼리 포함) 및 Apache Hudi 데이터 세트를 쿼리합니다.
- 공간 데이터를 쿼리합니다.
- Amazon SageMaker에서 기계 학습 추론을 사용하여 쿼리합니다.
- 사용자 정의 함수를 사용하여 쿼리합니다.
- 파티션 프로젝션을 사용하여 고도로 분할된 테이블의 쿼리 처리 속도를 높이고 파티션 관리를 자동화합니다.

주제

- Athena의 문제 해결
- 코드 샘플

테이블, 데이터베이스, 데이터 카탈로그 이해

Athena에서 카탈로그, 데이터베이스 및 테이블은 기본 소스 데이터에 대한 스키마를 정의하는 메타데이터 정의를 위한 컨테이너입니다.

Athena는 다음 용어를 사용하여 데이터 객체의 계층 구조를 참조합니다.

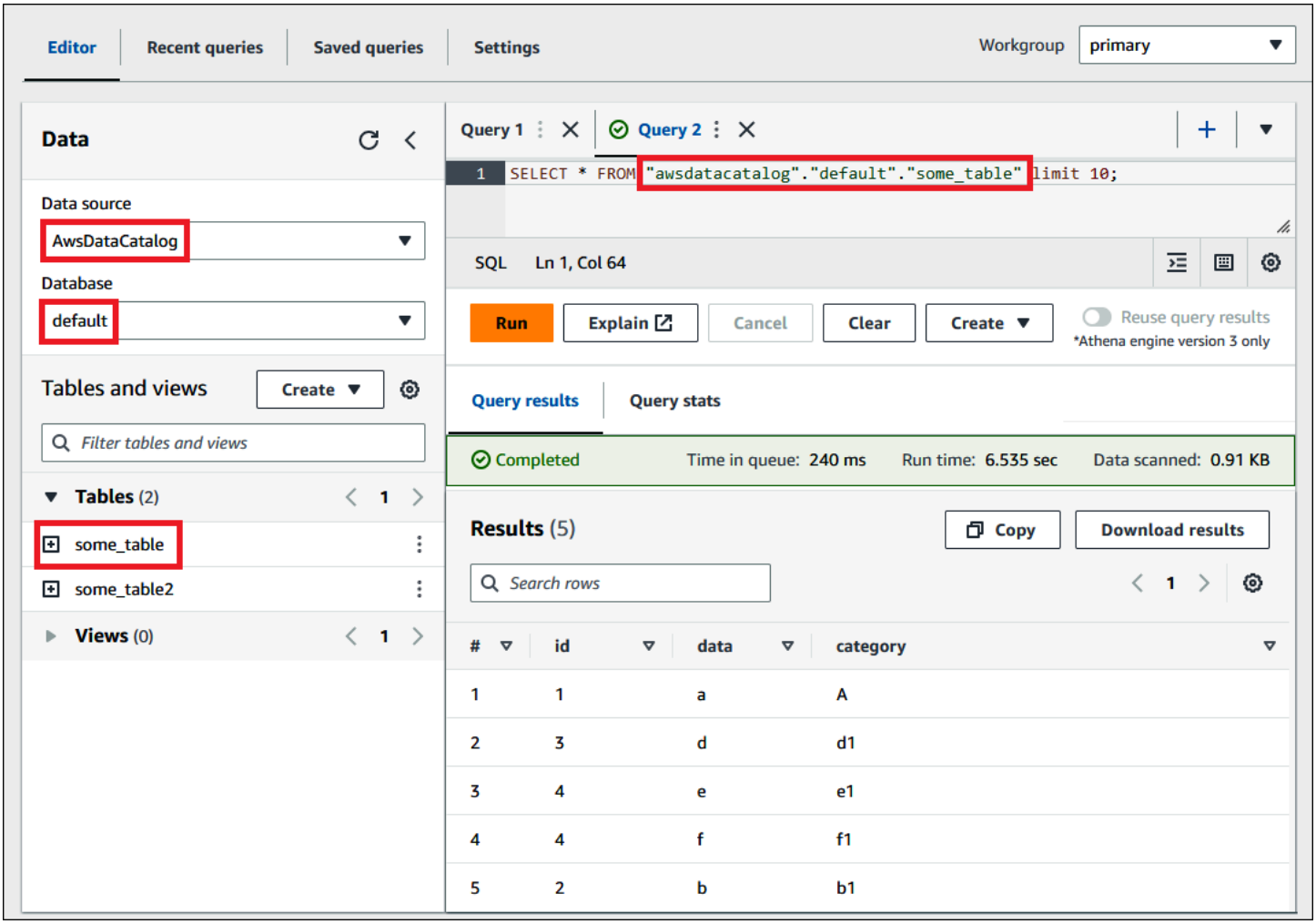
- 데이터 소스 - 데이터베이스 그룹
- 데이터베이스 - 테이블 그룹
- 테이블 - 행 또는 열 그룹으로 구성된 데이터

때때로 이 객체는 다음과 같이 동등한 대체 이름으로 참조되기도 합니다.

- 데이터 소스를 카탈로그라고도 합니다.
- 데이터베이스를 스키마라고도 합니다.

> **Note**
>
> 이 용어는 Athena에서 사용하는 페더레이션된 데이터 소스에 따라 다를 수 있습니다. 자세한 내용은 Athena 및 페더레이션된 테이블 이름 한정자 단원을 참조하십시오.

Athena 콘솔의 다음 쿼리 예제는 `awsdatacatalog` 데이터 소스, `default` 데이터베이스 및 `some_table` 테이블을 사용합니다.



각 데이터 세트에 대한 테이블이 Athena에 있어야 합니다. 테이블의 메타데이터는 Amazon S3의 데이터 위치를 Athena에 알려주고, 데이터의 구조(예: 열 이름, 데이터 유형 및 테이블 이름)를 지정합니다. 데이터베이스는 테이블의 논리적 그룹이며 데이터 세트에 대한 메타데이터와 스키마 정보만 보유합니다.

쿼리하려는 각 데이터 세트에 대해 쿼리 결과를 얻고 반환하는 데 사용할 기본 테이블이 Athena에 있어야 합니다. 따라서 데이터를 쿼리하기 전에 테이블이 Athena에 등록되어 있어야 합니다. 자동 또는 수동으로 테이블을 만들면 등록이 이루어집니다.

AWS Glue 크롤러를 사용하여 테이블을 자동으로 생성할 수 있습니다. AWS Glue 및 크롤러에 대한 자세한 내용은 AWS Glue와 통합을 참조하세요. AWS Glue는 테이블을 생성할 때 자신의 AWS Glue 데이터 카탈로그에 해당 테이블을 등록합니다. Athena는 AWS Glue 데이터 카탈로그를 사용하여 이 메타데이터를 저장하고 검색하며, 기본 데이터 세트를 분석하기 위해 쿼리를 실행할 때 사용합니다.

테이블 생성 방법에 관계없이 테이블 생성 프로세스는 Athena를 통해 데이터 세트를 등록합니다. AWS Glue Data Catalog에서 이 등록이 이루어지며, 등록하면 Athena가 데이터에 대해 쿼리를 실행할

수 있게 됩니다. Athena 쿼리 편집기에서 이 카탈로그(또는 데이터 소스)는 `AwsDataCatalog` 레이블로 참조됩니다.

테이블을 만든 후에는 SQL SELECT 문을 사용하여 테이블을 쿼리해서 원본 데이터의 특정 파일 위치 가져오기 등을 수행할 수 있습니다. 쿼리 결과는 Amazon S3의 지정된 쿼리 결과 위치에 저장됩니다.

AWS Glue 데이터 카탈로그는 Amazon Web Services 계정 전반에서 액세스할 수 있습니다. 다른 AWS 서비스도 AWS Glue 데이터 카탈로그를 공유할 수 있으므로 Athena를 사용하여 조직 전체에서 생성된 데이터베이스와 테이블을 볼 수 있으며 그 반대의 경우도 마찬가지입니다.

- 수동으로 테이블을 생성하려면
  - Athena 콘솔을 사용하여 테이블 만들기 마법사를 실행합니다.
  - Athena 콘솔을 사용하여 쿼리 편집기에서 Hive DDL 문을 작성합니다.
  - Athena API 또는 CLI를 사용해 DDL 문으로 SQL 쿼리 문자열을 실행합니다.
  - Athena JDBC 또는 ODBC 드라이버를 사용합니다.

테이블 및 데이터베이스를 수동으로 생성하는 경우 Athena는 내부적으로 `CREATE TABLE`, `CREATE DATABASE`, `DROP TABLE` 등의 HiveQL 데이터 정의 언어(DDL) 문을 사용하여 AWS Glue Data Catalog에 테이블과 데이터베이스를 생성합니다.

시작하려면 Athena 콘솔에서 자습서를 사용하거나 Athena 설명서의 단계별 안내서를 살펴보세요.

- Athena 콘솔에서 자습서를 사용하려면 콘솔 오른쪽 상단에 있는 정보 아이콘을 선택하고 자습서 탭을 선택합니다.
- Athena 쿼리 편집기에서 테이블을 생성하고 쿼리를 작성하는 방법에 대한 단계별 자습서는 시작하기 섹션을 참조하세요.

시작하기

이 자습서에서는 Amazon Athena를 이용한 데이터 쿼리를 안내합니다. Amazon Simple Storage Service에 저장된 샘플 데이터를 기반으로 테이블을 만들고 테이블을 쿼리한 다음 쿼리 결과를 확인할 것입니다.

이 자습서는 라이브 리소스를 사용하므로 실행하는 쿼리에 대한 요금이 부과됩니다. 이 자습서에서 사용하는 위치의 샘플 데이터에 대해서는 요금이 청구되지 않지만, Amazon S3에 자체 데이터 파일을 업로드하면 요금이 발생합니다.

필수 조건

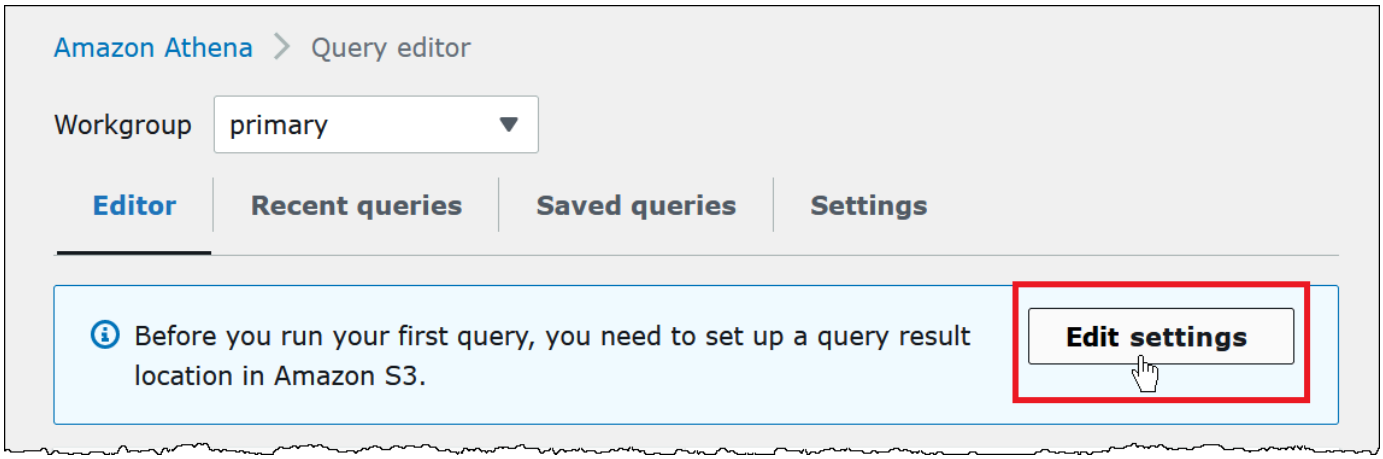
- 아직 계정에 가입하지 않았다면 AWS 계정에 가입하세요.
- Athena에 사용하는 것과 같은 AWS 리전(예: 미국 서부(오레곤) 및 계정을 사용하여 Amazon S3에 버킷을 생성하는 단계에 따라 Athena 쿼리 결과를 보관합니다. 이 버킷을 쿼리 출력 위치로 구성합니다.

1단계: 데이터베이스 생성

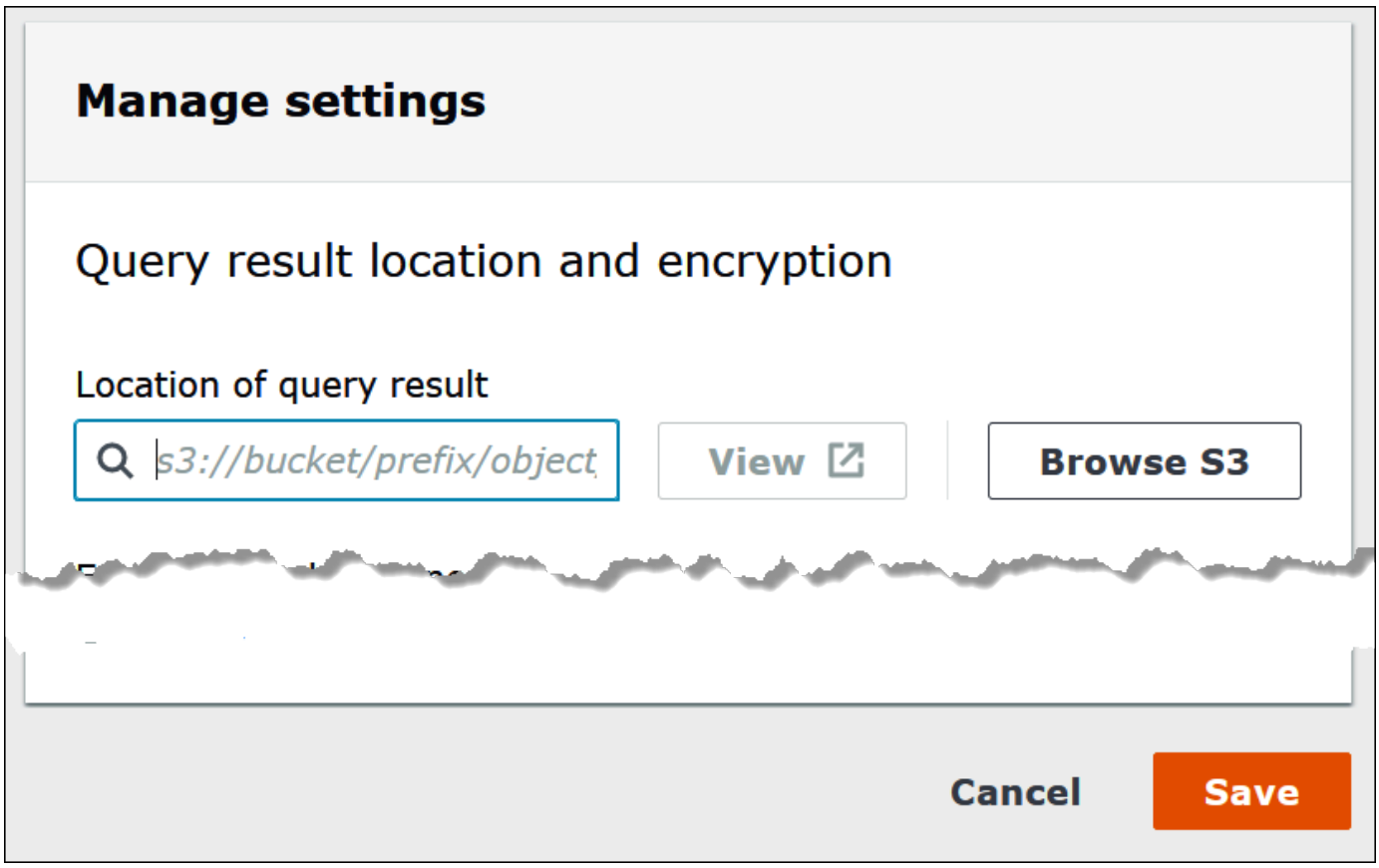
먼저 Athena에서 데이터베이스를 생성해야 합니다.

Athena 데이터베이스 생성

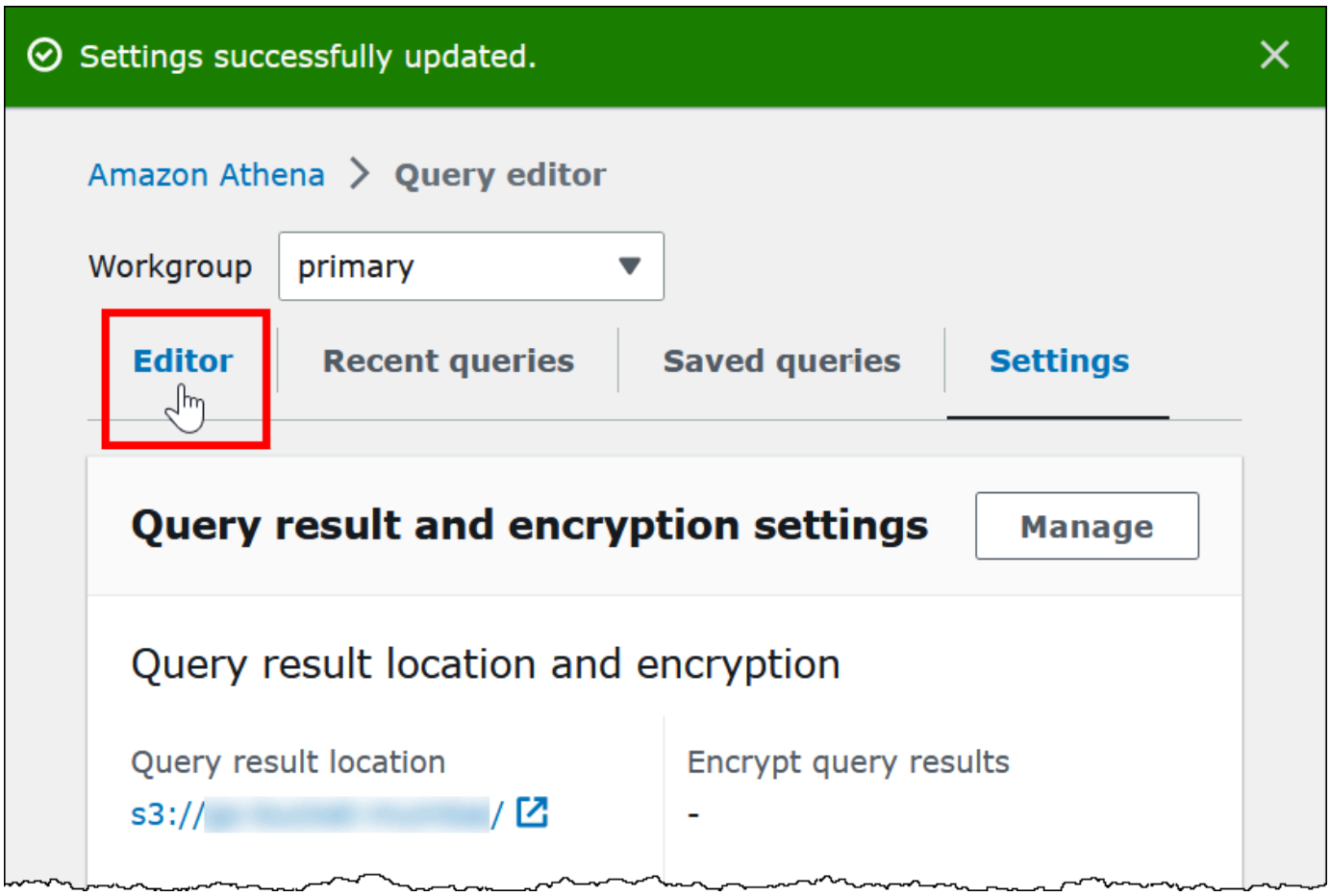
1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 현재 AWS 리전의 Athena 콘솔을 처음 방문하는 경우 쿼리 편집기 검색(Explore the query editor)을 선택해 쿼리 편집기를 엽니다. 그렇지 않으면 Athena가 쿼리 편집기에서 열립니다.
3. Amazon S3에서 쿼리 결과 위치를 설정하려면 Edit Settings(설정 편집)를 선택합니다.



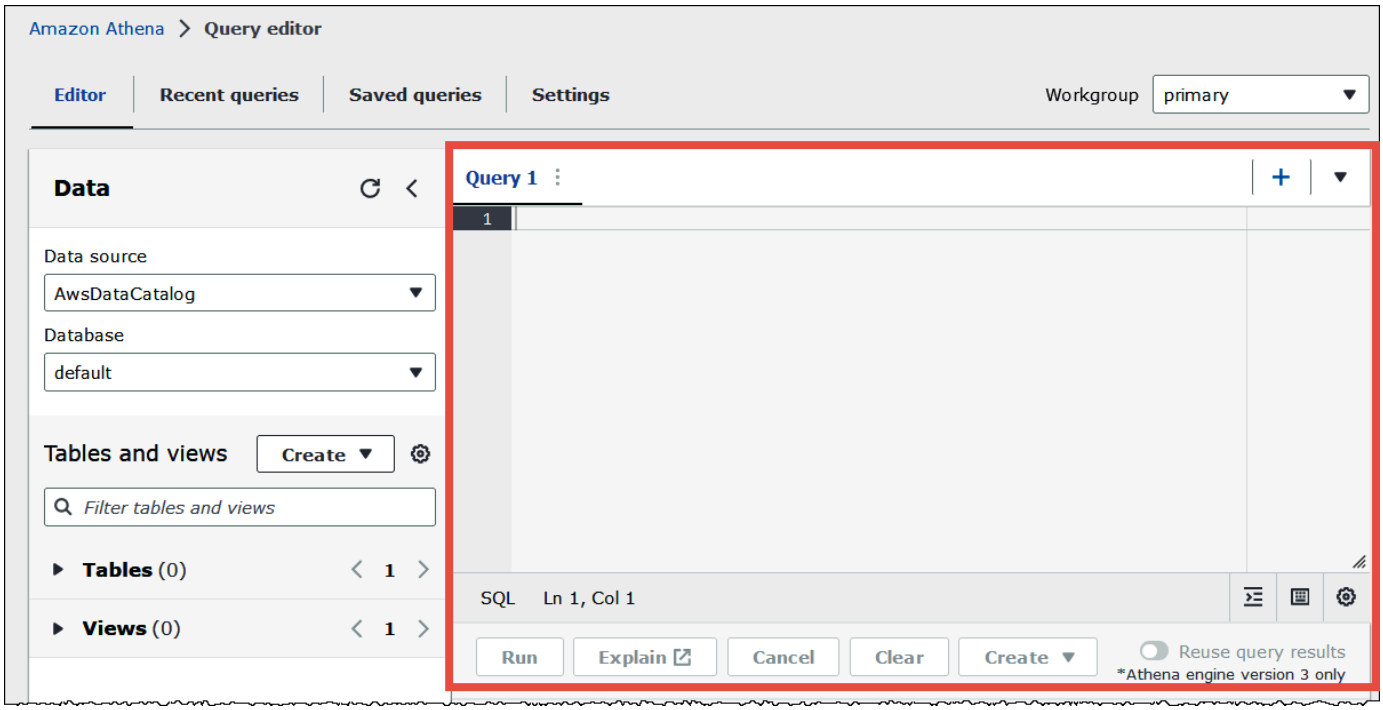
4. 설정 관리(Manage settings)에서 다음 중 하나를 수행합니다.
   - 쿼리 결과 위치(Location of query result) 상자에서 Amazon S3에서 쿼리 결과를 위해 생성한 버킷의 경로를 입력합니다. 경로 앞에 `s3://`를 붙입니다.
   - S3 검색(Browse S3) 아이콘을 선택하고 현재 리전에서 생성한 Amazon S3 버킷을 선택한 다음 선택(Choose)을 선택합니다.



5. Save(저장)를 선택합니다.
6. 편집기(Editor)를 선택하여 쿼리 편집기로 전환합니다.



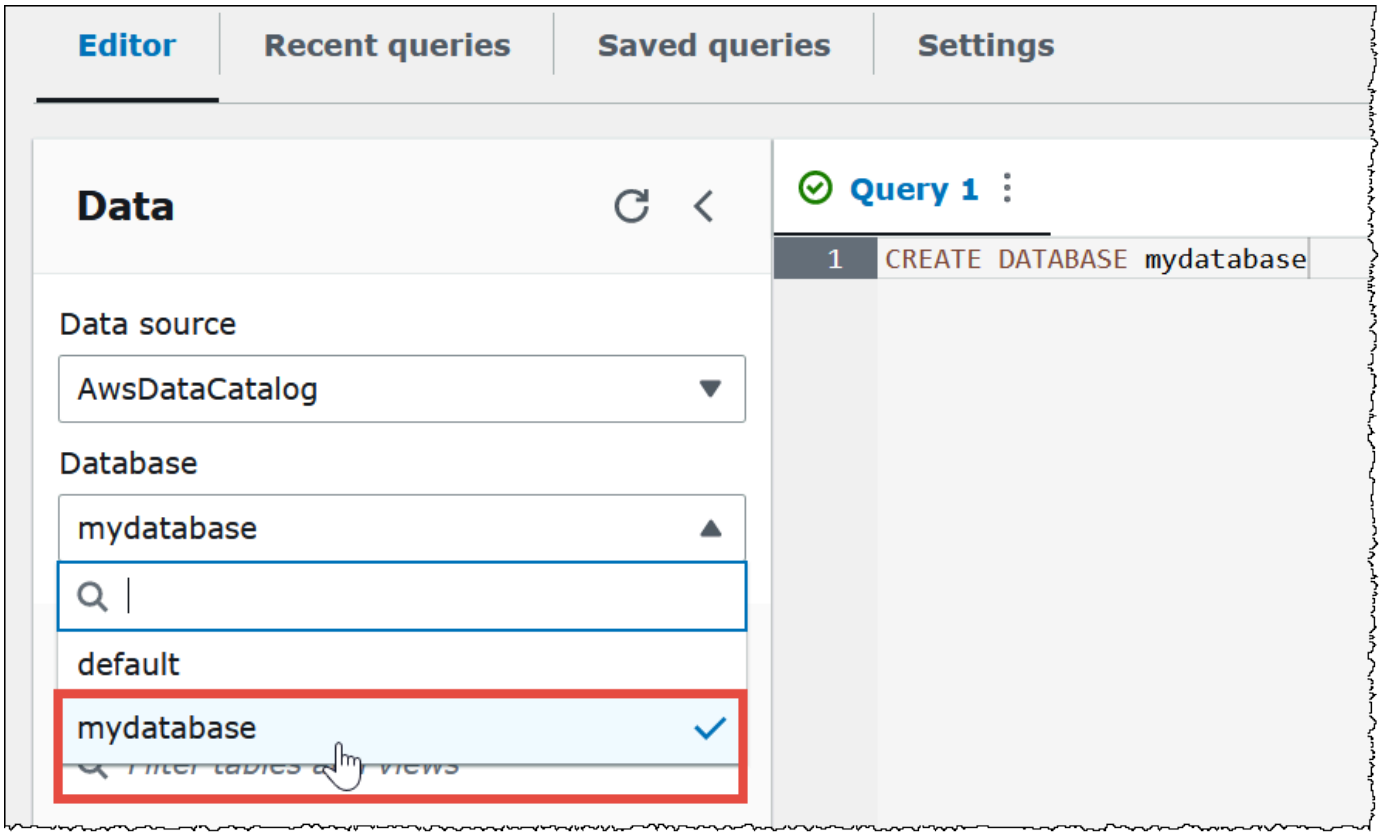
7. 탐색 창의 오른쪽에서 Athena 쿼리 편집기를 사용하여 쿼리와 문을 입력하고 실행할 수 있습니다.



8. mydatabase라는 데이터베이스를 생성하려면 다음 CREATE DATABASE 문을 입력합니다.

```
CREATE DATABASE mydatabase
```

9. 실행(Run)을 선택하거나 **Ctrl+ENTER**를 누릅니다.
10. 왼쪽의 데이터베이스(Database) 목록에서 mydatabase를 선택하면 현재 데이터베이스로 만들 수 있습니다.



2단계: 테이블 생성

이제 데이터베이스가 생겼으므로 이를 위한 Athena 테이블을 만들 수 있습니다. 생성하는 테이블은 s3://athena-examples-*myregion*/cloudfront/plaintext/ 위치에 있는 샘플 Amazon CloudFront 로그 데이터를 기반으로 합니다. 여기서 *myregion*은 자신의 현재 AWS 리전입니다.

샘플 로그 데이터는 탭으로 구분된 값(TSV) 형식입니다. 즉, 필드를 구분하는 구분 기호로 탭 문자가 사용됩니다. 데이터는 다음 예제와 같습니다. 읽기 쉽도록 발췌 부분의 탭은 공백으로 변환되었고 마지막 필드는 줄였습니다.

```
2014-07-05 20:00:09 DFW3 4260 10.0.0.15 GET eabcd12345678.cloudfront.net /test-
image-1.jpeg 200 - Mozilla/5.0[...]
```

```
2014-07-05 20:00:09 DFW3 4252 10.0.0.15 GET eabcd12345678.cloudfront.net /test-
image-2.jpeg 200 - Mozilla/5.0[...]
2014-07-05 20:00:10 AMS1 4261 10.0.0.15 GET eabcd12345678.cloudfront.net /test-
image-3.jpeg 200 - Mozilla/5.0[...]
```

Athena가 이 데이터를 읽도록 다음과 같이 직관적인 CREATE EXTERNAL TABLE 문을 만들 수 있습니다. 테이블을 생성하는 문으로 데이터에 매핑되는 열을 정의하고, 데이터의 구분 방식을 지정하고, 샘플 데이터가 포함되는 Amazon S3 위치를 지정합니다. Athena에서 폴더 내 모든 파일을 스캔할 것으로 예상하므로 LOCATION 절에서는 특정 파일이 아니라 Amazon S3 폴더 위치를 지정합니다.

이 예제에는 곧 설명하겠지만 중요한 제한 사항이 있으므로 아직 사용하지 마세요.

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
  `Date` DATE,
  Time STRING,
  Location STRING,
  Bytes INT,
  RequestIP STRING,
  Method STRING,
  Host STRING,
  Uri STRING,
  Status INT,
  Referrer STRING,
  ClientInfo STRING
  )
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t'
  LINES TERMINATED BY '\n'
  LOCATION 's3://athena-examples-my-region/cloudfront/plaintext/';
```

이 예제에서는 cloudfront_logs라는 테이블을 생성하고 각 필드의 이름과 데이터 유형을 지정합니다. 이러한 필드는 테이블의 열이 됩니다. date는 예약된 단어이므로 백틱(`) 문자로 이스케이프합니다. ROW FORMAT DELIMITED는 Athena가 LazySimpleSerDe라는 기본 라이브러리를 사용해 실제 데이터 구문 분석 작업을 수행할 것임을 의미합니다. 또한 이 예제에서는 필드를 탭(FIELDS TERMINATED BY '\t')으로 구분하고 파일의 각 레코드가 줄 바꿈 문자(LINES TERMINATED BY '\n)로 끝나도록 지정합니다. 마지막으로 LOCATION 절은 읽어 올 실제 데이터가 위치한 Amazon S3의 경로를 지정합니다.

탭 또는 쉼표로 구분된 자체 데이터가 있는 경우 (필드에 중첩된 정보가 포함되지 않는 한) 방금 제시된 예제에서와 같이 CREATE TABLE 문을 사용할 수 있습니다. 하지만 다른 구분 기호를 사용하는 중첩된 정보가 포함된 ClientInfo와 같은 열이 있는 경우 다른 접근 방식이 필요합니다.

ClientInfo 필드에서 데이터 추출

샘플 데이터를 살펴보면 여기에 최종 필드 `ClientInfo`의 전체 예제가 있습니다.

```
Mozilla/5.0%20(Android;%20U;%20Windows%20NT%205.1;%20en-US;
%20rv:1.9.0.9)%20Gecko/2009040821%20IE/3.0.9
```

보시다시피 이 필드는 다중 값입니다. 예제인 `CREATE TABLE` 문에서 필드 구분 기호로 탭을 지정하기 때문에 `ClientInfo` 필드 내의 구성 요소를 별도의 열로 나눌 수 없습니다. 따라서 새 `CREATE TABLE` 문이 필요합니다.

`ClientInfo` 필드 내부의 값에서 열을 생성하기 위해 정규식 그룹을 포함한 정규 표현식을 사용할 수 있습니다. 지정한 정규 표현식 그룹은 별도의 테이블 열이 됩니다. `CREATE TABLE` 문에 정규 표현식을 사용하려면 다음과 같은 구문을 사용합니다. 이 구문은 Athena에게 Regex SerDe 라이브러리와, 사용자가 지정한 정규 표현식을 사용하도록 지시합니다.

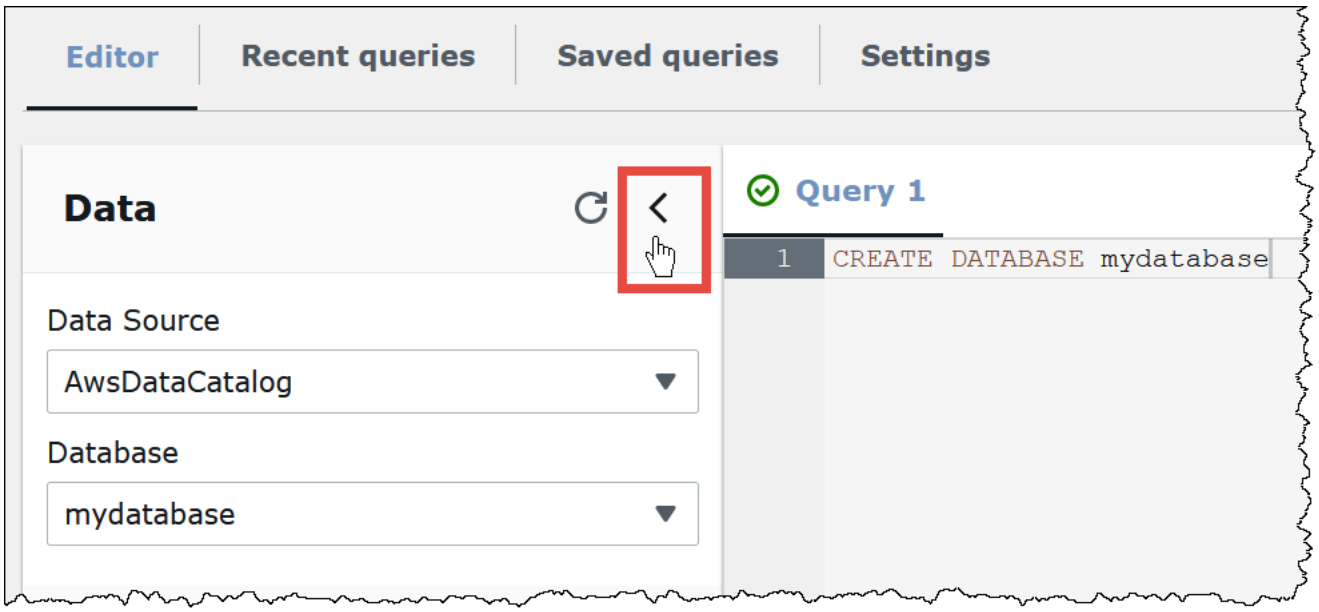
```
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
  WITH SERDEPROPERTIES ("input.regex" = "regular_expression")
```

정규 표현식은 복잡한 CSV 또는 TSV 데이터에서 테이블을 만드는 데 유용하지만 작성하고 관리하기가 어려울 수 있습니다. 다행히 JSON, Parquet, ORC와 같은 형식을 사용할 수 있는 다른 라이브러리들이 있습니다. 자세한 내용은 지원되는 SerDes 및 데이터 형식 단원을 참조하세요.

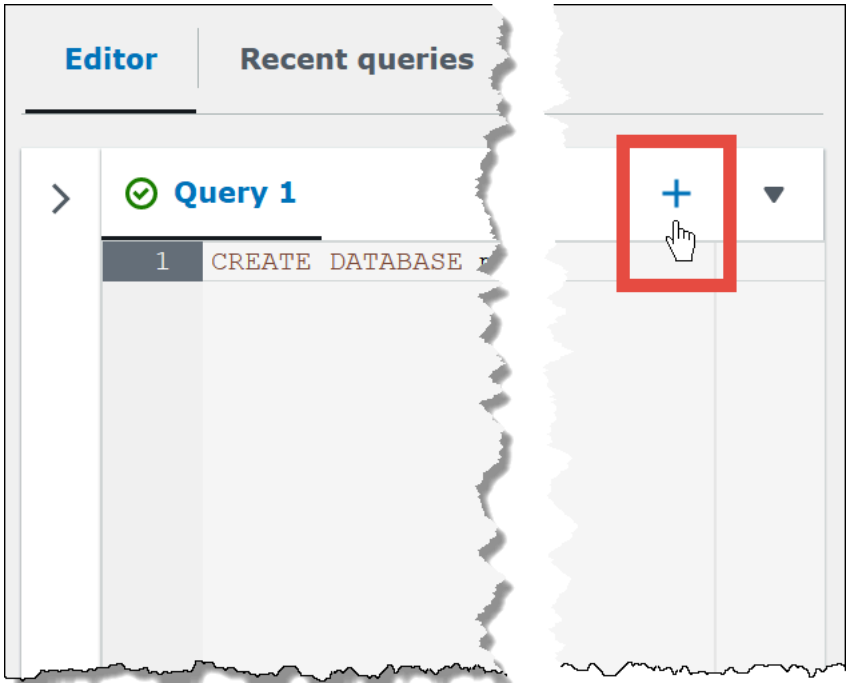
이제 Athena 쿼리 편집기에서 테이블을 생성할 준비가 되었습니다. 귀하를 위해 `CREATE TABLE` 문과 정규식이 제공되어 있습니다.

Athena에서 테이블 생성

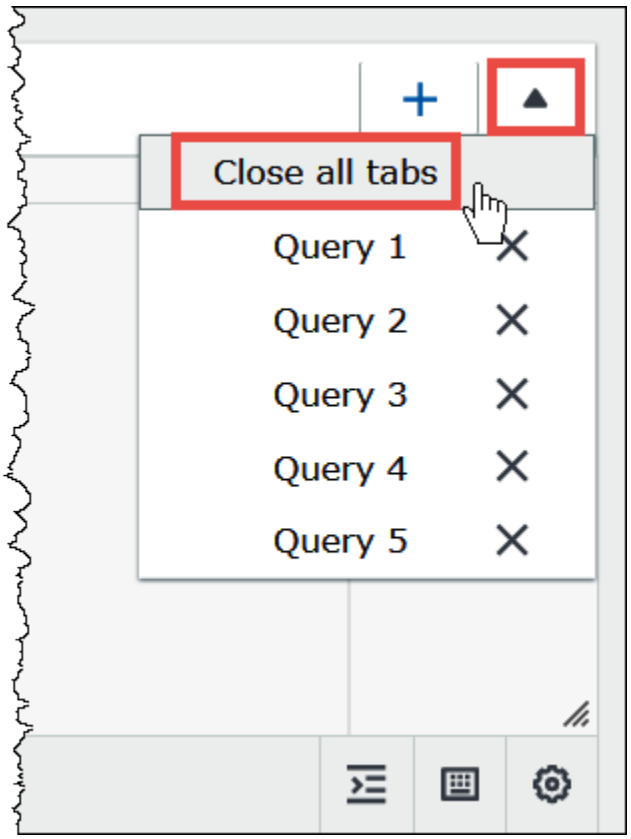
1. 탐색 창에서 데이터베이스(Database)에 대해 `mydatabase`가 선택되어 있는지 확인합니다.
2. 쿼리 편집기에서 더 많은 공간을 확보하려면 화살표 아이콘을 선택하여 탐색 창을 축소합니다.



3. 새로운 쿼리를 생성하려면 쿼리 편집기에서 더하기(+) 기호를 선택합니다. 한 번에 최대 10개의 쿼리 탭을 열 수 있습니다.



4. 하나 이상의 쿼리 탭을 닫으려면 더하기 기호 옆에 있는 화살표를 선택합니다. 모든 탭을 한 번에 닫으려면 화살표를 선택한 다음 모든 탭 닫기(Close all tabs)를 선택합니다.



5. 쿼리 창에 다음 CREATE EXTERNAL TABLE 문을 입력합니다. 이 정규 표현식은 로그 데이터에 있는 ClientInfo 필드의 운영 체제, 브라우저 및 브라우저 버전 정보를 구분합니다.

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
  `Date` DATE,
  Time STRING,
  Location STRING,
  Bytes INT,
  RequestIP STRING,
  Method STRING,
  Host STRING,
  Uri STRING,
  Status INT,
  Referrer STRING,
  os STRING,
  Browser STRING,
  BrowserVersion STRING
  )
  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
  WITH SERDEPROPERTIES (
```

```
  "input.regex" = "^(?!#)([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s
+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+[^\(]+[\(]([^\;]+).*\%20([^
\/]+)[\/](.*)$"
  ) LOCATION 's3://athena-examples-myregion/cloudfront/plaintext/';
```

6. LOCATION 문에서 *myregion*을 현재 사용 중인 AWS 리전(예: us-west-1)으로 바꿉니다.
7. Run(실행)을 선택합니다.

   cloudfront_logs 테이블이 생성되어 mydatabase 데이터베이스의 테이블 목록에 나타납니다.

3단계: 데이터 쿼리

이제 Amazon S3의 데이터를 기반으로 Athena에서 생성한 cloudfront_logs 테이블이 있으므로 테이블에서 SQL 쿼리를 실행하고 Athena에서 결과를 볼 수 있습니다. Athena에서의 SQL 사용에 대한 자세한 내용은 Athena에 대한 SQL 참조 섹션을 참조하세요.

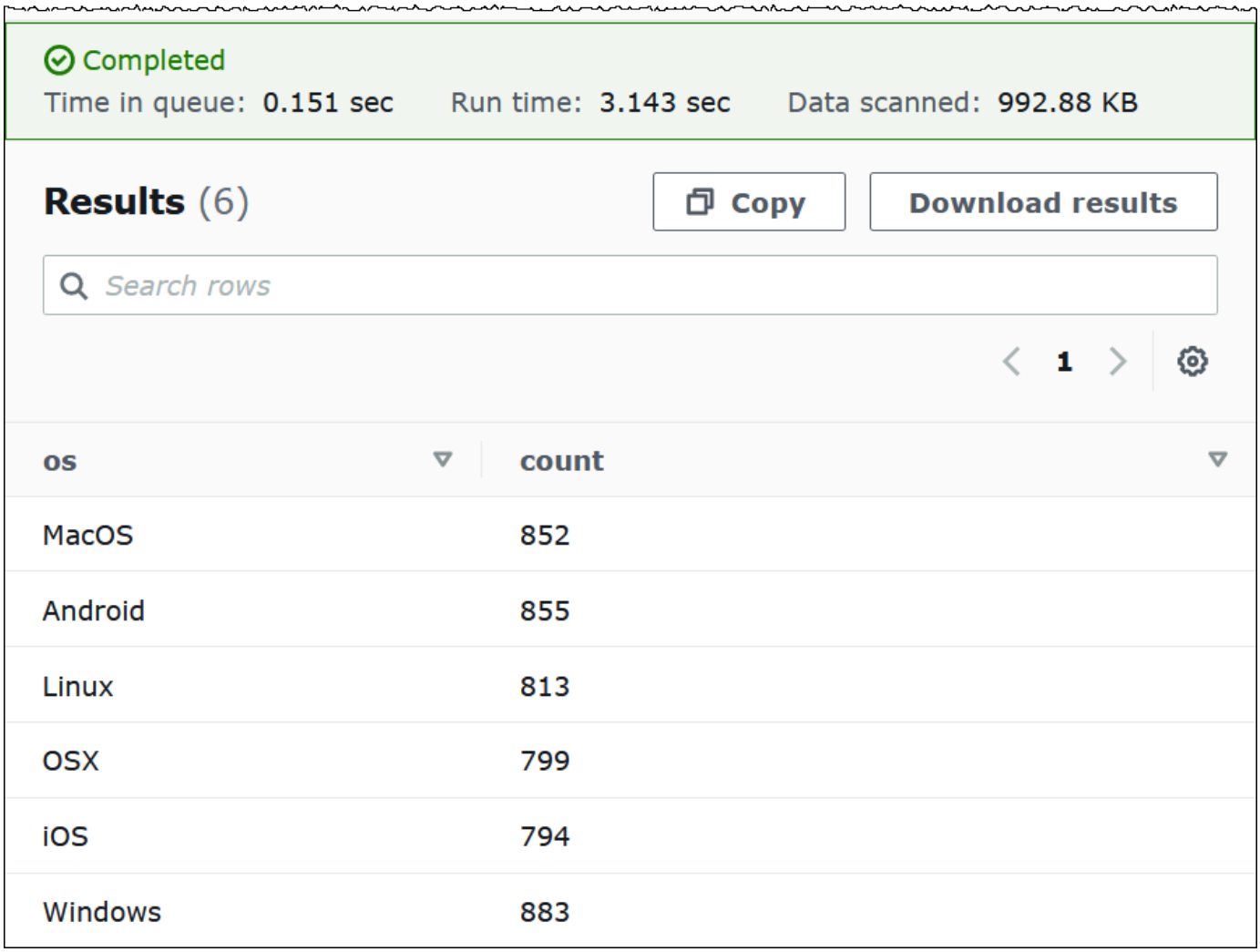
쿼리를 실행하려면

1. 더하기(+) 기호를 선택해 새 쿼리 탭을 열고 쿼리 창에 다음 SQL 문을 입력합니다.

```
SELECT os, COUNT(*) count
FROM cloudfront_logs
WHERE date BETWEEN date '2014-07-05' AND date '2014-08-05'
GROUP BY os
```

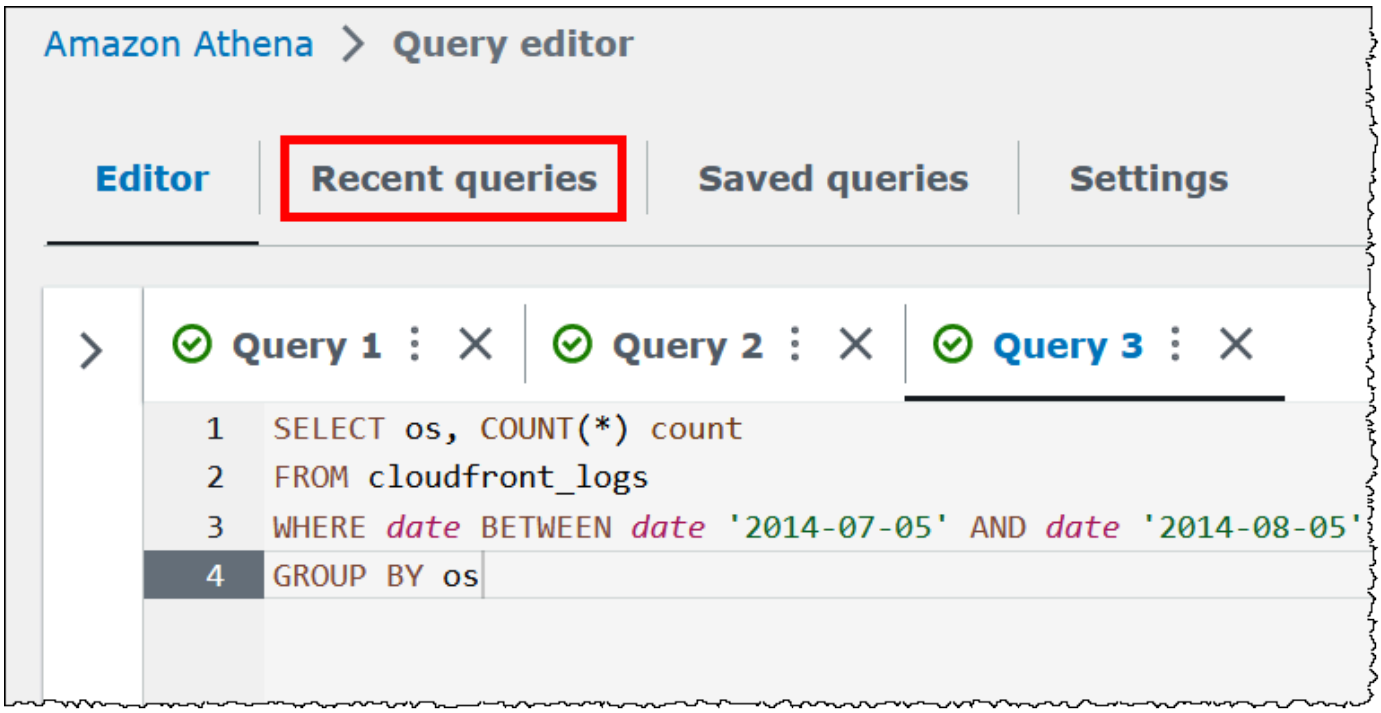
2. Run(실행)을 선택합니다.

   결과는 다음과 같이 나타납니다.

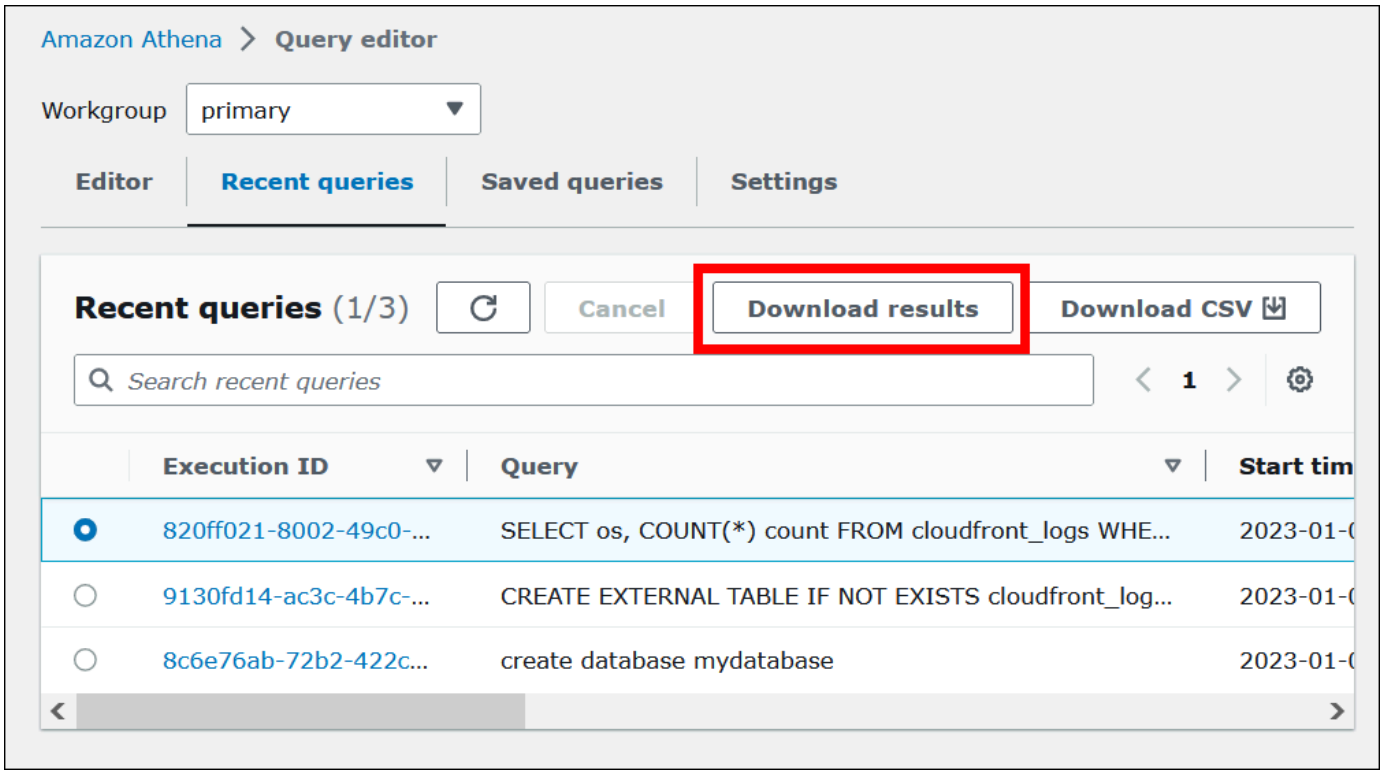


3. 쿼리의 결과를 `.csv` 파일에 저장하려면 결과 다운로드(Download results)를 선택합니다.

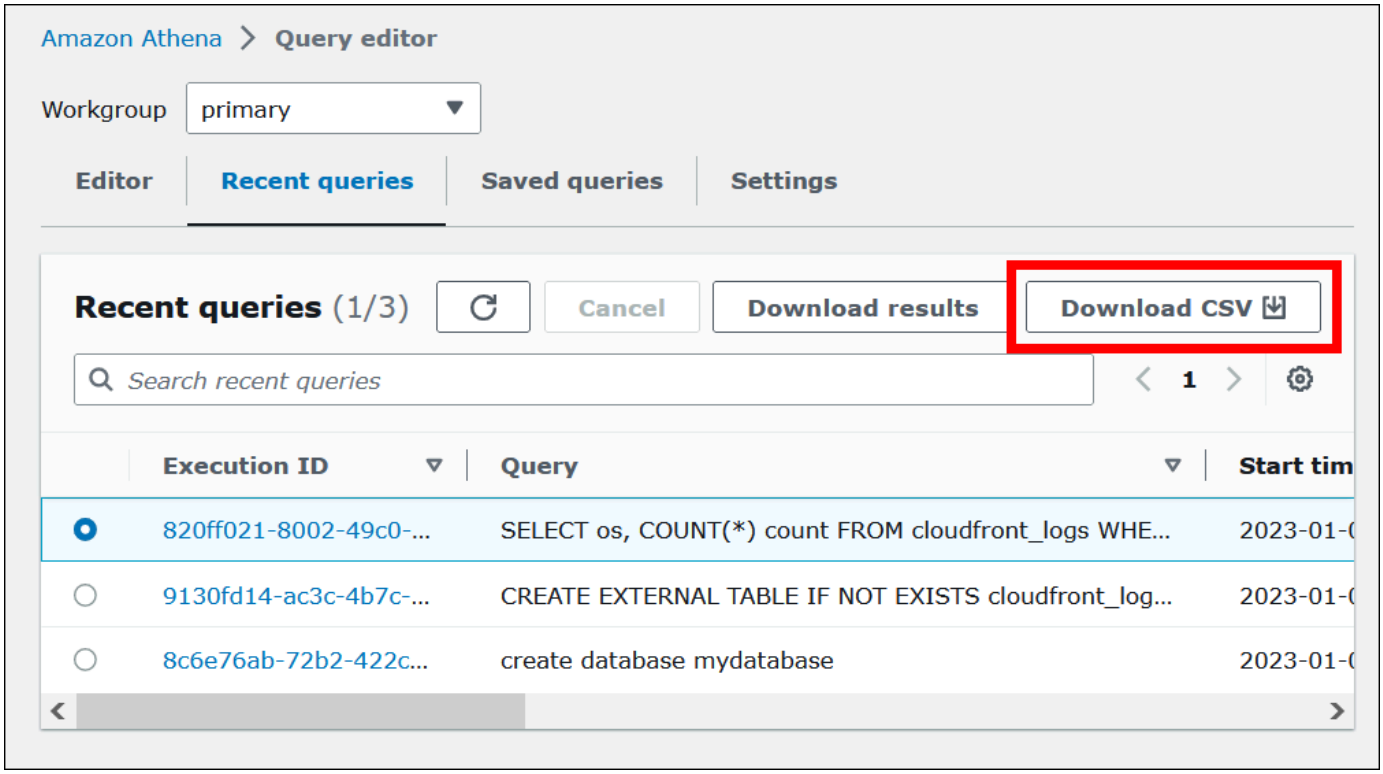
4. 이전 쿼리를 보거나 실행하려면 최신 쿼리(Recent queries) 탭을 선택합니다.



5. 최신 쿼리(Recent queries) 탭에서 이전 쿼리 결과를 다운로드하려면 쿼리를 선택한 다음 결과 다운로드(Download results)를 선택합니다. 쿼리는 45일 동안 보존됩니다.



6. 하나 이상의 최근 SQL 쿼리 문자열을 CSV 파일로 다운로드하려면 Download CSV(CSV 다운로드)를 선택합니다.



자세한 내용은 쿼리 결과, 최근 쿼리, 출력 파일 작업 단원을 참조하십시오.

쿼리 저장

쿼리 편집기에서 만들거나 편집한 쿼리를 이름으로 저장할 수 있습니다. Athena는 이러한 쿼리를 저장된 쿼리(Saved queries) 탭에 저장합니다. 이 저장된 쿼리(Saved queries)를 통해 저장한 쿼리를 불러오거나, 실행하거나, 이름을 바꾸거나, 삭제할 수 있습니다. 자세한 내용은 저장된 쿼리 사용 단원을 참조하십시오.

키보드 바로 가기 및 미리 입력하기 제안

Athena 쿼리 편집기에서는 쿼리 실행, 쿼리 형식 지정, 줄 작업, 찾기 및 바꾸기와 같은 작업을 위한 다양한 키보드 바로 가기를 제공합니다. 자세한 내용과 전체 바로 가기 목록은 AWS 빅 데이터 블로그의 Improve productivity by using keyboard shortcuts in Amazon Athena query editor를 참조하세요.

Athena 쿼리 편집기에서는 더 빠른 쿼리 작성 경험을 위해 코드 미리 입력하기 제안을 지원합니다. 정확성과 효율성이 향상된 SQL 쿼리를 작성할 수 있도록 다음 기능을 제공합니다.

- 입력할 때 키워드, 로컬 변수, 스니펫 및 카탈로그 항목에 대한 제안이 실시간으로 표시됩니다.

- 데이터베이스 이름이나 테이블 이름을 입력하고 점을 입력하면 편집기에 선택할 수 있는 테이블 또는 열 목록이 편리하게 표시됩니다.
- 스니펫 제안에 마우스를 대면 스니펫의 구문 및 사용법에 대한 간략한 개요가 시놉시스에 표시됩니다.
- 코드 가독성을 높이기 위해 키워드와 강조 표시 규칙도 Trino 및 Hive의 최신 구문에 맞게 업데이트되었습니다.

이 기능은 기본적으로 활성화되어 있습니다. 이 기능을 활성화하거나 비활성화하려면 쿼리 편집기 창의 오른쪽 하단에 있는 코드 편집기 기본 설정(톱니 아이콘)을 사용합니다.

다른 데이터 소스에 연결

이 자습서에서는 CSV 형식의 Amazon S3의 데이터 소스를 사용했습니다. AWS Glue의 Athena 사용 방법에 대한 자세한 내용은 AWS Glue를 사용하여 Amazon S3의 데이터 원본에 연결 단원을 참조하세요. ODBC 및 JDBC 드라이버, 외부 Hive 메타스토어, Athena 데이터 소스 커넥터를 사용하여 다양한 데이터 소스에 Athena를 연결할 수도 있습니다. 자세한 내용은 데이터 원본에 연결 단원을 참조하세요.

데이터 원본에 연결

Amazon Athena를 사용하여 데이터 세트의 다른 위치 및 형식으로 저장된 데이터를 쿼리할 수 있습니다. 이 데이터 세트는 CSV, JSON, Avro, Parquet 또는 기타 형식일 수 있습니다.

쿼리를 실행하기 위해 Athena에서 작업하는 테이블과 데이터베이스는 메타데이터를 기반으로 합니다. 메타데이터는 데이터 세트의 기본 데이터에 대한 데이터입니다. 해당 메타데이터가 데이터 세트를 설명하는 방법을 스키마라고 합니다. 예를 들어 테이블 이름, 테이블의 열 이름 및 각 열의 데이터 유형은 기본 데이터 세트를 설명하는 메타데이터로 저장된 스키마입니다. Athena에서 사용자는 메타데이터를 데이터 카탈로그 또는 메타스토어로 구성하기 위한 시스템을 호출합니다. 데이터 세트와 이를 설명하는 데이터 카탈로그의 조합을 데이터 원본이라고 합니다.

메타데이터와 기본 데이터 세트의 관계는 작업하는 데이터 원본 유형에 따라 달라집니다. MySQL, PostgreSQL, SQL Server와 같은 관계형 데이터 원본은 데이터 세트와 메타데이터를 밀접하게 통합합니다. 이러한 시스템에서는 데이터가 작성될 때 메타데이터가 가장 자주 작성됩니다. Hive를 사용하여 구축된 것과 같은 다른 데이터 원본을 사용하면 데이터 세트를 읽을 때 즉석에서 메타데이터를 정의할 수 있습니다. 데이터 세트는 예를 들어 CSV, JSON, Parquet 또는 Avro처럼 다양한 형식일 수 있습니다.

Athena는 기본적으로 AWS Glue Data Catalog를 지원합니다. AWS Glue Data Catalog는 Amazon S3, Amazon Redshift, Amazon DynamoDB 등의 다른 데이터 세트 및 데이터 원본을 바탕으로 구축된 데이터 카탈로그입니다. 다양한 커넥터를 사용하여 Athena를 다른 데이터 원본에 연결할 수도 있습니다.

주제

AWS Glue와의 통합

교차 계정 액세스AWS Glue는 완전관리형 추출, 변환, 로드 AWS 서비스입니다. 주요 기능 중 하나는 데이터를 분석하고 분류하는 것입니다. AWS Glue 크롤러를 사용하면 Amazon S3의 데이터로부터 데이터베이스 및 테이블 스키마를 자동으로 추론하고 관련된 메타데이터를 AWS Glue Data Catalog에 저장할 수 있습니다.

Athena는 AWS Glue Data Catalog를 사용해 Amazon Web Services 계정의 Amazon S3 데이터에 대한 테이블 메타데이터를 저장하고 검색합니다. Athena 쿼리 엔진은 데이블 메타데이터를 통해 쿼리하려는 데이터를 찾고, 읽고, 처리할 방법을 파악합니다.

AWS Glue Data Catalog에서 데이터베이스 및 테이블 스키마를 만들려면 Athena 내에서 데이터 원본에 대해 AWS Glue 크롤러를 실행하거나 Athena 쿼리 편집기에서 데이터 정의 언어(DDL) 쿼리를 직접 실행합니다. 그런 다음 생성한 데이터베이스 및 테이블 스키마를 사용하여 Athena의 DML(데이터 조작 언어) 쿼리로 데이터를 쿼리할 수 있습니다.

자신의 계정이 아닌 다른 계정에서 AWS Glue Data Catalog를 등록할 수 있습니다. AWS Glue에 요구되는 IAM 권한을 구성하면 Athena를 사용해 계정 간 쿼리를 실행할 수 있습니다. 자세한 내용은 AWS Glue 데이터 카탈로그에 대한 교차 계정 액세스 단원을 참조하십시오.

AWS Glue Data Catalog에 대한 자세한 내용은 AWS Glue 개발자 안내서의 AWS Glue의 데이터 카탈로그 및 크롤러를 참조하세요.

AWS Glue에는 별도 요금이 적용됩니다. 자세한 내용은 AWS Glue 요금을 참조하십시오.

주제

- AWS Glue를 사용하여 Amazon S3의 데이터 원본에 연결
- 다른 계정의 AWS Glue Data Catalog 등록
- Athena와 AWS Glue를 함께 사용하는 모범 사례
- AWS CLI를 사용하여 AWS Glue 데이터베이스 및 해당 테이블 다시 생성

AWS Glue를 사용하여 Amazon S3의 데이터 원본에 연결

Athena는 AWS Glue Data Catalog를 사용해 Amazon S3에 저장된 데이터에 연결하여 테이블 및 열 이름과 같은 메타데이터를 저장할 수 있습니다. 연결되면 데이터베이스, 테이블 및 뷰가 Athena의 쿼리 편집기에 나타납니다.

사용할 AWS Glue의 스키마 정보를 정의하려면 해당 정보를 자동으로 검색하도록 AWS Glue 크롤러를 생성하거나 수동으로 테이블을 추가하고 스키마 정보를 입력하면 됩니다.

AWS Glue 크롤러 생성

Athena 콘솔에서 크롤러를 시작한 다음 통합된 방식으로 AWS Glue 콘솔을 사용하여 크롤러를 생성할 수 있습니다. 크롤러를 생성할 때 크롤링할 Amazon S3의 데이터 위치를 지정합니다.

Athena 콘솔에서 시작하여 AWS Glue에서 크롤러 생성

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 쿼리 편집기에서 테이블 및 뷰(Tables and views) 옆에 있는 생성(Create)을 선택한 다음 AWS Glue 크롤러(Glue crawler)를 선택합니다.
3. AWS Glue 콘솔의 크롤러 추가 페이지에서 단계에 따라 크롤러를 생성합니다. 자세한 내용은 이 설명서의 AWS Glue 크롤러 사용과 AWS Glue 개발자 안내서의 AWS Glue Data Catalog 채우기를 참조하세요.

> **Note**
>
> Athena는 AWS Glue 크롤러에 지정한 제외 패턴을 인식하지 못합니다. 예를 들어 `.csv` 및 `.json` 파일이 모두 포함된 Amazon S3 버킷이 있는데 `.json` 파일을 크롤러에서 제외한다면 Athena는 두 파일 그룹을 모두 쿼리합니다. 이 문제를 방지하려면 제외할 파일을 다른 위치에 배치하면 됩니다.

양식을 사용하여 테이블 추가

다음 절차에서는 Athena 콘솔을 사용하여 S3 버킷 데이터에서 테이블 생성(Create Table From S3 bucket data) 양식을 사용하여 테이블을 추가하는 방법을 보여줍니다.

테이블 추가 및 양식을 사용하여 스키마 정보 입력

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 쿼리 편집기에서 테이블 및 뷰(Tables and views) 옆에 있는 생성(Create)을 선택한 다음 S3 버킷 데이터(S3 bucket data)를 선택합니다.
3. S3 버킷 데이터에서 테이블 생성(Create Table From S3 bucket data) 양식에서 테이블 이름(Table name)에 테이블 이름을 입력합니다.
4. 데이터베이스 구성(Database configuration)에서 기존 데이터베이스를 선택하거나 새 데이터베이스를 생성합니다.
5. 입력 데이터 세트 위치(Location of Input Data Set)에는 처리할 데이터 세트가 들어 있는 폴더의 Amazon S3 경로를 지정합니다. 경로에 파일 이름을 포함하지 마세요. 지정하는 폴더에 있는 파일을 Athena에서 모두 스캔합니다. 데이터가 이미 파티셔닝된 경우(예:

   s3://DOC-EXAMPLE-BUCKET/logs/year=2004/month=12/day=11/), 기본 경로(예: s3://DOC-EXAMPLE-BUCKET/logs/)만 입력합니다.
6. Data Format(데이터 형식)의 경우 다음과 같은 옵션 중에서 선택합니다.

   - Table type(테이블 유형)에서 Apache Hive, Apache Iceberg 또는 Delta Lake를 선택합니다. Athena에서는 Apache Hive 테이블 유형을 기본값으로 사용합니다. Athena의 Apache Iceberg 테이블 쿼리에 대한 내용은 Apache Iceberg 테이블 사용 단원을 참조하세요. Athena의 Delta Lake 테이블 사용에 대한 내용은 Linux Foundation Delta Lake 테이블 쿼리 단원을 참조하세요.
   - File format(파일 형식)의 경우 데이터가 들어 있는 파일 또는 로그 형식을 선택합니다.
     - 사용자 지정 구분 기호가 있는 텍스트 파일 옵션의 경우 필드 종결자(즉, 열 구분 기호)를 지정합니다. 선택 사항으로, 배열 유형의 끝을 표시하는 Collection terminator(컬렉션 종결자) 또는 맵 데이터 유형의 끝을 표시하는 Collection terminator(컬렉션 종결자)를 지정할 수 있습니다.
   - SerDe library(SerDe 라이브러리) – SerDe(serializer-deserializer) 라이브러리에서는 Athena에서 특정 데이터 형식의 테이블을 생성할 수 있도록 구문을 분석합니다. 대다수 형식에 기본 SerDe 라이브러리가 선택되어 있습니다. 다음과 같은 형식의 경우 요구 사항에 따라 라이브러리를 선택합니다.
     - Apache Web Logs(Apache 웹 로그) – RegexSerDe 또는 GrokSerDe 라이브러리를 선택합니다. RegexSerDe의 경우 Regex definition(정규식 정의) 상자에 정규 표현식을 입력합니다.

GrokSerDe의 경우 `input.format` SerDe 속성에 대해 이름이 지정된 일련의 정규 표현식을 입력합니다. 이름이 지정된 정규 표현식은 정규 표현식보다 읽고 유지 관리하기 쉽습니다. 자세한 내용은 Amazon S3에 저장된 Apache 로그 쿼리 단원을 참조하십시오.

- CSV – 쉼표로 구분된 데이터에 큰따옴표로 묶인 값이 없거나 `java.sql.Timestamp` 형식을 사용하는 경우 LazySimpleSerDe를 선택합니다. 데이터에 따옴표가 있거나 `TIMESTAMP`에 UNIX 숫자 형식을 사용하는(예:1564610311) 경우 OpenCSVSerDe를 선택합니다. 자세한 내용은 CSV, TSV, 사용자 지정 구분 기호로 구분된 파일에 대한 LazySimpleSerDe 및 CSV 처리를 위한 OpenCSVSerDe 섹션을 참조하세요.
- JSON – OpenX 또는 Hive JSON SerDe 라이브러리를 선택합니다. 두 형식 모두 각 JSON 문서가 한 줄의 텍스트에 있으며 해당 필드가 줄 바꿈 문자로 구분되지 않을 것으로 예상합니다. OpenX SerDe에서는 몇 가지 추가 속성을 제공합니다. 이러한 속성에 대한 자세한 내용은 OpenX JSON SerDe 단원을 참조하십시오. Hive SerDe에 대한 내용은 Hive JSON SerDe 섹션을 참조하세요.

Athena의 SerDe 라이브러리 사용에 대한 자세한 내용은 지원되는 SerDes 및 데이터 형식 섹션을 참조하세요.

7. SerDe properties(SerDe 속성)의 경우 사용 중인 SerDe 라이브러리와 자신의 요구 사항에 따라 속성과 값을 추가, 편집 또는 제거합니다.

   - SerDe 속성을 추가하려면 Add SerDe property(SerDe 속성 추가)를 선택합니다.
   - Name(이름) 필드에 속성의 이름을 입력합니다.
   - Value(값) 필드에 속성 값을 입력합니다.
   - SerDe 속성을 제거하려면 Remove(제거)를 선택합니다.

8. Table properties(테이블 속성)의 경우 자신의 요구 사항에 따라 테이블 속성을 선택하거나 편집합니다.

   - Write compression(쓰기 압축)의 경우 압축 옵션을 선택합니다. 쓰기 압축 옵션과 사용 가능한 압축 옵션의 가용성은 데이터 형식에 따라 다릅니다. 자세한 내용은 Athena 압축 지원 단원을 참조하십시오.
   - Encryption(암호화)의 경우 Amazon S3에서 기본 데이터가 암호화되어 있으면 Encrypted data set(암호화된 데이터 세트)를 선택합니다. 이 옵션을 통해 `CREATE TABLE` 명령문의 `has_encrypted_data` 테이블 속성이 true로 설정됩니다.

9. Column details(열 세부 정보)의 경우 테이블에 추가할 열의 이름과 데이터 유형을 입력합니다.

   - 열을 한 번에 하나씩 추가하려면 열 추가를 선택합니다.

- 열을 빠르게 추가하려면 열 일괄 추가를 선택합니다. 텍스트 상자에 쉼표로 구분된 열 목록을 *column_name data_type*, *column_name data_type*[, ...], 형식으로 입력한 다음 추가(Add)를 선택합니다.

10. (선택 사항) 파티션 세부 정보(Partition details)에 대해 하나 이상의 열 이름과 데이터 형식을 추가합니다. 파티셔닝은 열 값을 기준으로 관련 데이터가 함께 유지되므로 쿼리당 스캔되는 데이터의 양을 줄이는 데 도움이 될 수 있습니다. 파티셔닝에 대한 자세한 내용은 Athena에서 데이터 분할 섹션을 참조하세요.
11. (선택 사항) Bucketing(버킷팅) 경우 함께 그룹화하려는 행이 있는 하나 이상의 열을 지정한 다음에 해당 행을 여러 버킷에 넣을 수 있습니다. 이렇게 하면 버킷 열 값을 지정할 때 읽으려는 버킷만 쿼리할 수 있습니다.

    - Buckets(버킷)의 경우 고유 값(예: 프라이머리 키)이 많이 있고 쿼리의 데이터 필터링에 자주 사용되는 열을 하나 이상 선택합니다.
    - Number of buckets(버킷 수)의 경우 파일의 최적 크기가 허용되는 숫자를 입력합니다. 자세한 내용은 AWS 빅 데이터 블로그의 Top 10 Performance Tuning Tips for Amazon Athena(Amazon Athena의 성능 튜닝을 위한 10가지 팁)를 참조하세요.
    - 버킷된 열을 지정하기 위해 `CREATE TABLE` 명령문에서는 다음과 같은 구문을 사용합니다.

    ```
    CLUSTERED BY (bucketed_columns) INTO number_of_buckets BUCKETS
    ```

    > **Note**
    >
    > Iceberg 테이블 유형에는 Bucketing(버킷팅) 옵션을 사용할 수 없습니다.

12. 미리 보기 테이블 쿼리(Preview table query) 상자에는 양식에 입력한 정보로 생성된 `CREATE TABLE` 문이 표시됩니다. 미리 보기 문은 직접 편집할 수 없습니다. 명령문을 변경하려면 미리 보기 위의 양식 필드를 수정하거나 양식을 사용하지 않고 쿼리 편집기에서 직접 명령문을 생성합니다.
13. 테이블 생성(Create table)을 선택하여 쿼리 편집기에서 생성된 문을 실행하고 테이블을 생성합니다.

다른 계정의 AWS Glue Data Catalog 등록

Athena의 계정 간 AWS Glue 카탈로그 기능을 사용해 자신의 계정이 아닌 다른 계정의 AWS Glue 카탈로그를 등록할 수 있습니다. AWS Glue에 요구되는 IAM 권한을 구성하고 카탈로그를 Athena

DataCatalog 리소스로 등록하면 Athena를 사용해 계정 간 쿼리를 실행할 수 있습니다. 필요한 권한 구성에 대한 자세한 내용은 AWS Glue 데이터 카탈로그에 대한 교차 계정 액세스 단원을 참조하세요.

다음 절차는 Athena 콘솔을 사용하여 자신의 계정이 아닌 Amazon Web Services 계정의 AWS Glue Data Catalog를 데이터 소스로 구성할 수 있습니다.

다른 계정의 AWS Glue Data Catalog를 등록하려면

1. AWS Glue 데이터 카탈로그에 대한 교차 계정 액세스의 단계에 따라 다른 계정의 데이터 카탈로그를 쿼리할 권한이 있는지 확인합니다.
2. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
3. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

4. 데이터 소스(Data Source)을 선택합니다.
5. 오른쪽 위에서 데이터 소스 생성(Create data source)을 선택합니다.
6. 데이터 소스 선택(Choose a data source)페이지에서 데이터 소스(Data sources)에 S3 - AWS Glue Data Catalog를 선택하고 다음(Next)을 선택합니다.
7. 데이터 소스 세부 정보 입력(Enter data source details) 페이지의 AWS Glue Data Catalog 섹션에서 AWS Glue Data Catalog 선택(Choose an AWS Glue Data Catalog)에 다른 계정의 AWS Glue Data Catalog(AWS Glue Data Catalog in another account)를 선택합니다.
8. 데이터 세트 세부 정보(Dataset details)에 다음 정보를 입력합니다.

   - 데이터 소스 이름(Data source name) – 다른 계정의 데이터 카탈로그를 참조할 때 SQL 쿼리에 사용할 이름을 입력합니다.
   - 설명(Description) - (선택 사항) 다른 계정의 데이터 카탈로그에 대한 설명을 입력합니다.

- 카탈로그 ID(Catalog ID) - 데이터 카탈로그가 속한 계정의 Amazon Web Services 계정 ID(12자리)를 입력합니다. Amazon Web Services 계정 ID는 카탈로그 ID입니다.

9. (선택 사항) 태그(Tags)에 데이터 소스와 연결할 키-값 페어를 입력합니다. 태그에 대한 자세한 내용은 Athena 리소스 태깅 단원을 참조하세요.
10. 다음을 선택합니다.
11. 검토 및 생성(Review and create) 페이지에서 제공한 정보를 검토한 다음 데이터 소스 생성(Create data source)을 선택합니다. 데이터 소스 세부 정보(Data source details) 페이지는 등록한 데이터 카탈로그에 대한 데이터베이스 및 태그를 나열합니다.
12. 데이터 소스(Data Source)을 선택합니다. 등록한 데이터 카탈로그가 데이터 소스 이름(Data source name) 열에 나열됩니다.
13. 새 데이터 카탈로그에 대한 정보를 보거나 편집하려면 카탈로그를 선택한 다음 작업(Actions), 편집(Edit)을 선택합니다.
14. 새 데이터 카탈로그를 삭제하려면 카탈로그를 선택한 다음 작업(Actions), 삭제(Delete)를 선택합니다.

자세한 내용은 AWS 빅 데이터 블로그의 Query cross-account AWS Glue Data Catalogs using Amazon Athena를 참조하세요.

Athena와 AWS Glue를 함께 사용하는 모범 사례

AWS Glue Data Catalog와 함께 Athena를 사용하는 경우 AWS Glue를 사용하여 Athena에서 쿼리할 데이터베이스와 테이블(스키마)을 생성하거나, Athena를 사용하여 스키마를 생성한 후 AWS Glue 및 관련 서비스에서 사용할 수 있습니다. 이 주제에서는 두 방법을 사용할 때 고려할 사항과 모범 사례를 제공합니다.

기본적으로 Athena는 Trino를 사용하여 DML 문을 처리하고, Hive를 사용하여 스키마를 생성하고 수정하는 DDL 문을 처리합니다. 이러한 기술을 사용할 경우 Athena와 AWS Glue의 원활한 작동을 위해 따라야 하는 몇 가지 규칙이 있습니다.

이 주제의 내용

    - "HIVE_PARTITION_SCHEMA_MISMATCH"를 피하기 위한 파티션 스키마 동기화
    - 테이블 메타데이터 업데이트
- CSV 파일 작업
    - 따옴표로 묶인 CSV 데이터
    - 헤더가 포함된 CSV 파일
- AWS Glue 파티션 인덱싱 및 필터링
- 지리 공간 데이터 작업
- Athena와 함께 ETL에 AWS Glue 작업 사용
    - AWS Glue ETL 작업에 Athena를 사용하여 테이블 생성
    - ETL 작업을 사용하여 쿼리 성능 최적화
    - ORC로 변환할 때 SMALLINT 및 TINYINT 데이터 형식을 INT로 변환
    - ETL을 위한 AWS Glue 작업 자동화

데이터베이스, 테이블 및 열 이름

Athena에서 쿼리할 AWS Glue에 스키마를 생성하려면 다음 사항을 고려하세요.

- AWS Glue의 데이터베이스 이름, 테이블 이름 및 열 이름에 사용할 수 있는 문자는 UTF-8 문자열이어야 합니다. 문자열은 1바이트 미만이거나 255바이트를 초과해서는 안 됩니다. 사용할 수 있는 문자에는 공백이 포함되고 다음과 같은 한 줄 문자열 패턴으로 정의됩니다.

  ```
  [\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\t]*
  ```

- 현재 AWS Glue 정규식 패턴에서는 이름 앞에 선행 공백을 추가할 수 있습니다. 이러한 선행 공백은 탐지하기 어렵고 생성 후 사용성 문제를 일으킬 수 있으므로 선행 공백이 있는 객체 이름을 만들지 마세요.
- AWS::Glue::Database AWS CloudFormation 템플릿을 사용하여 AWS Glue 데이터베이스를 생성하고 데이터베이스 이름을 지정하지 않으면 AWS Glue는 Athena와 호환되지 않는 *resource_name-random_string* 형식으로 데이터베이스 이름을 자동으로 생성합니다.
- AWS Glue 카탈로그 관리자를 사용하여 열 이름을 바꿀 수 있지만 테이블 이름이나 데이터베이스 이름은 바꿀 수 없습니다. 이 제한을 해결하려면 이전 데이터베이스의 정의를 사용하여 새 이름으로 데이터베이스를 생성해야 합니다. 그런 다음 이전 데이터베이스의 테이블 정의를 사용하여 새 데이터베이스에서 테이블을 다시 생성합니다. 이렇게 하려면 AWS CLI 또는 AWS Glue SDK를 사용하면 됩니다. 단계는 AWS CLI를 사용하여 AWS Glue 데이터베이스 및 해당 테이블 다시 생성를 참조하세요.

AWS Glue의 데이터베이스 및 테이블에 대한 자세한 내용은 AWS Glue 개발자 안내서의 데이터베이스 및 테이블을 참조하세요.

AWS Glue 크롤러 사용

AWS Glue 크롤러는 데이터 집합의 스키마를 검색하고 AWS Glue 데이터 카탈로그에 테이블로 등록하는 데 도움이 됩니다. 크롤러는 데이터를 살펴보고 스키마를 결정합니다. 뿐만 아니라 파티션도 찾고 등록할 수 있습니다. 자세한 내용은 AWS Glue 개발자 안내서의 크롤러 정의를 참조하세요. 성공적으로 크롤링된 데이터의 테이블은 Athena에서 쿼리할 수 있습니다.

> **Note**
>
> Athena는 AWS Glue 크롤러에 지정한 제외 패턴을 인식하지 못합니다. 예를 들어 `.csv` 및 `.json` 파일이 모두 포함된 Amazon S3 버킷이 있는데 `.json` 파일을 크롤러에서 제외한다면 Athena는 두 파일 그룹을 모두 쿼리합니다. 이 문제를 방지하려면 제외할 파일을 다른 위치에 배치하면 됩니다.

AWS Glue Data Catalog와 Amazon S3의 동기화를 유지하기 위한 크롤러 예약

AWS Glue 크롤러는 일정 또는 필요에 따라 실행되도록 설정할 수 있습니다. 자세한 내용은 AWS Glue 개발자 안내서의 크롤러와 작업을 위한 시간 기반 일정을 참조하세요.

정해진 시간에 분할된 테이블에 수신되는 데이터가 있는 경우 AWS Glue 크롤러가 일정에 따라 테이블 파티션을 검색하고 업데이트하도록 설정할 수 있습니다. 그러면 상당한 시간과 비용이 들 수 있는 `MSCK REPAIR` 명령을 실행하거나 수동으로 `ALTER TABLE ADD PARTITION` 명령을 실행할 필요가 없습니다. 자세한 내용은 AWS Glue 개발자 안내서의 테이블 파티션을 참조하세요.

크롤러와 여러 데이터 원본 사용

AWS Glue 크롤러가 Amazon S3를 스캔하고 여러 디렉터리를 발견하면 휴리스틱을 사용하여 테이블의 루트가 디렉터리 구조에서 어디에 있는지, 어떤 디렉터리가 테이블의 파티션인지 파악합니다. 간혹 둘 이상의 디렉터리에서 발견된 스키마가 유사한 경우, 크롤러가 이를 별도의 테이블 대신 파티션으로 처리할 수 있습니다. 크롤러가 개별 테이블을 찾도록 돕는 한 가지 방법은, 각 테이블의 루트 디렉터리를 크롤러의 데이터 저장소로 추가하는 것입니다.

Amazon S3에 있는 다음 파티션이 한 예입니다.

```
s3://DOC-EXAMPLE-BUCKET/folder1/table1/partition1/file.txt
s3://DOC-EXAMPLE-BUCKET/folder1/table1/partition2/file.txt
```

```
s3://DOC-EXAMPLE-BUCKET/folder1/table1/partition3/file.txt
s3://DOC-EXAMPLE-BUCKET/folder1/table2/partition4/file.txt
s3://DOC-EXAMPLE-BUCKET/folder1/table2/partition5/file.txt
```

table1과 table2의 스키마가 비슷하고 AWS Glue에서 하나의 데이터 원본이 s3://DOC-EXAMPLE-BUCKET/folder1/으로 설정된 경우, 크롤러는 두 개의 파티션 열(table1과 table2가 포함된 파티션 열과 partition1부터 partition5까지 포함하는 두 번째 파티션 열)로 구성된 하나의 테이블을 만들 수 있습니다.

AWS Glue 크롤러가 의도대로 두 개의 개별 테이블을 생성하게 하려면 크롤러가 다음 절차처럼 두 개의 데이터 원본(s3://DOC-EXAMPLE-BUCKET/folder1/table1/, s3://DOC-EXAMPLE-BUCKET/folder1/table2)으로 구성되도록 설정합니다.

AWS Glue에서 기존 크롤러에 S3 데이터 스토어를 추가하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창에서 크롤러(Crawlers)를 선택합니다.
3. 크롤러 링크를 선택한 다음 편집(Edit)을 선택합니다.
4. 2단계: 데이터 소스 및 분류자 선택(Step 2: Choose data sources and classifiers)에서 편집(Edit)을 선택합니다.
5. 데이터 소스(Data sources)에서 데이터 소스 추가(Add a data source)를 선택합니다.
6. 데이터 소스 추가(Add a data source) 대화 상자의 S3 경로(S3 path)에서 찾아보기(Browse)를 선택합니다.
7. 사용할 버킷을 선택한 다음 선택(Choose)을 선택합니다.

   추가한 데이터 소스가 데이터 소스(Data sources) 목록에 표시됩니다.
8. 다음을 선택합니다.
9. 보안 설정 구성 페이지에서 크롤러에 대한 IAM 역할을 생성 또는 선택한 다음 다음(Next)을 선택합니다.
10. S3 경로가 후행 슬래시로 끝나는지 확인한 다음 S3 데이터 소스 추가(Add an S3 data source)를 선택합니다.
11. 출력 및 예약 설정 페이지의 출력 구성(Output configuration)에서 대상 데이터베이스를 선택합니다.
12. 다음을 선택합니다.
13. 검토 및 업데이트 페이지에서 선택한 항목을 검토합니다. 단계를 편집하려면 편집(Edit)을 선택합니다.

14.업데이트를 선택합니다.

"HIVE_PARTITION_SCHEMA_MISMATCH"를 피하기 위한 파티션 스키마 동기화

파티션 열이 있는 AWS Glue 데이터 카탈로그의 각 테이블에 대해 스키마는 테이블 내에 개별 파티션별로 테이블 수준에서 저장됩니다. 파티션 스키마는 파티션 내에서 읽는 데이터 샘플을 기반으로 AWS Glue 크롤러에 의해 채워집니다. 자세한 내용은 크롤러와 여러 데이터 원본 사용 단원을 참조하십시오.

Athena가 쿼리를 실행하면 쿼리에 필요한 모든 파티션의 스키마와 테이블 스키마를 검증합니다. 검증을 통해 열 데이터 형식을 순서대로 비교하고, 중복되는 열에 대해 일치하도록 합니다. 그러면 테이블 중간에서 열을 추가하거나 제거하는 등의 예기치 않은 작업을 방지할 수 있습니다. 파티션의 스키마가 테이블의 스키마와 다름을 Athena가 감지하게 되면 Athena가 쿼리를 처리하지 못하여 `HIVE_PARTITION_SCHEMA_MISMATCH` 메시지와 함께 실패할 수도 있습니다.

이 문제는 몇 가지 방법으로 해결할 수 있습니다. 첫째, 데이터가 실수로 추가된 경우 스키마 차이를 유발하는 데이터 파일을 제거하고 파티션을 삭제한 다음 데이터를 다시 크롤링할 수 있습니다. 둘째, 개별 파티션을 삭제한 다음 Athena 내에서 `MSCK REPAIR`를 실행하여 테이블의 스키마를 사용하여 파티션을 다시 만들 수 있습니다. 이 두 번째 옵션은 적용된 스키마가 계속해서 데이터를 올바르게 읽을 것이 확실한 경우에만 유효합니다.

테이블 메타데이터 업데이트

크롤링 후 AWS Glue 크롤러는 Apache Hive, Presto 및 Spark 등 다른 외부 기술과 호환되도록 특정 테이블 메타데이터를 자동으로 할당합니다. 경우에 따라 크롤러가 메타데이터 속성을 잘못 할당할 수 있습니다. Athena를 사용하여 테이블을 쿼리하기 전에 AWS Glue의 속성을 수동으로 수정하세요. 자세한 내용은 AWS Glue 개발자 안내서의 테이블 세부 정보 보기 및 편집을 참조하세요.

CSV 파일에 각 데이터 필드를 묶는 인용 부호가 있으면 `serializationLib` 속성이 잘못되어 AWS Glue이(가) 메타데이터를 잘못 할당할 수 있습니다. 자세한 내용은 따옴표로 묶인 CSV 데이터 단원을 참조하십시오.

CSV 파일 작업

CSV 파일에는 각 열의 데이터 값이 인용 부호로 묶여 있기도 하고, 분석할 데이터의 일부가 아닌 헤더 값이 CSV 파일에 포함된 경우가 있습니다. AWS Glue을(를) 사용하여 이 파일에서 스키마를 생성하는 경우 이 단원의 지침을 따르세요.

따옴표로 묶인 CSV 데이터

다음 예시처럼 데이터 필드를 큰따옴표로 묶은 CSV 파일이 있을 수 있습니다.

```
"John","Doe","123-555-1231","John said \"hello\""
"Jane","Doe","123-555-9876","Jane said \"hello\""
```

따옴표로 묶인 값을 가진 CSV 파일에서 생성된 테이블에 대해 Athena에서 쿼리를 실행하려면 OpenCSVSerDe를 사용하기 위해 AWS Glue에서 테이블 속성을 수정해야 합니다. OpenCSV SerDe에 대한 자세한 내용은 CSV 처리를 위한 OpenCSVSerDe 단원을 참조하세요.

AWS Glue 콘솔에서 테이블 속성을 편집하려면

1. AWS Glue 콘솔 탐색 창에서 테이블을 선택합니다.
2. 편집할 테이블 링크를 선택한 다음 작업(Actions), 테이블 편집(Edit table)을 선택합니다.
3. 테이블 편집 페이지에서 다음과 같이 변경합니다.
   - 직렬화 라이브러리(Serialization lib)에 `org.apache.hadoop.hive.serde2.OpenCSVSerde`를 입력합니다.
   - Serde 파라미터(Serde parameters)에서 `escapeChar`, `quoteChar`, `separatorChar` 키에 다음 값을 입력합니다.
     - `escapeChar`에 백슬래시(****)를 입력합니다.
     - `quoteChar`에 큰따옴표(**"**)를 입력합니다.
     - `separatorChar`에 쉼표(**,**)를 입력합니다.
4. Save(저장)를 선택합니다.

자세한 내용은 AWS Glue 개발자 안내서의 테이블 세부 정보 보기 및 편집을 참조하세요.

프로그래밍 방식으로 AWS Glue 테이블 속성 업데이트

다음 JSON 예제처럼 AWS Glue UpdateTable API 작업 또는 update-table CLI 명령을 사용해 테이블 정의의 `SerDeInfo` 블록을 수정할 수 있습니다.

```
"SerDeInfo": {
   "name": "",
   "serializationLib": "org.apache.hadoop.hive.serde2.OpenCSVSerde",
   "parameters": {
      "separatorChar": ","
      "quoteChar": "\""
      "escapeChar": "\\"
      }
},
```

헤더가 포함된 CSV 파일

다음 예제와 같이 `CREATE TABLE` 문으로 Athena에서 테이블을 정의할 때 `skip.header.line.count` 테이블 속성을 사용하여 CSV 데이터의 헤더를 무시할 수 있습니다.

```
...
STORED AS TEXTFILE
LOCATION 's3://DOC-EXAMPLE-BUCKET/csvdata_folder/';
TBLPROPERTIES ("skip.header.line.count"="1")
```

또는 헤더 정보가 Athena 쿼리 결과에 포함되지 않도록 미리 CSV 헤더를 제거할 수 있습니다. 한 가지 방법은 추출, 변환 및 로드(ETL) 작업을 수행하는 AWS Glue 작업을 사용하는 것입니다. AWS Glue에서 PySpark Python 언어의 확장 언어를 사용하여 스크립트를 작성할 수 있습니다. 자세한 내용은 AWS Glue 개발자 안내서의 AWS Glue에 작업 작성을 참조하세요.

다음 예는 `from_options`를 사용하여 동적 프레임을 작성하고 `writeHeader` 형식 옵션을 false로 설정하여 헤더 정보를 제거하는 AWS Glue 스크립트의 함수를 보여줍니다.

```
glueContext.write_dynamic_frame.from_options(frame = applymapping1, connection_type =
 "s3", connection_options = {"path": "s3://DOC-EXAMPLE-BUCKET/MYTABLEDATA/"}, format =
 "csv", format_options = {"writeHeader": False}, transformation_ctx = "datasink2")
```

AWS Glue 파티션 인덱싱 및 필터링

Athena는 분할된 테이블을 쿼리할 때 사용 가능한 테이블 파티션을 검색하고 쿼리와 관련된 하위 집합으로 필터링합니다. 새 데이터와 파티션이 추가되면 파티션을 처리하는 데 더 많은 시간이 필요하며 쿼리 런타임이 늘어날 수 있습니다. 시간이 지남에 따라 확장되는 파티션이 많은 테이블이 있는 경우 AWS Glue 파티션 인덱싱 및 필터링을 고려하세요. 파티션 처리를 최적화하고 고도로 분할된 테이블에서 쿼리 성능을 향상시키기 위해 Athena는 파티션 인덱싱을 사용합니다. 테이블의 속성에서 파티션 필터링을 설정하는 과정은 두 단계로 이루어집니다.

1. AWS Glue에서 파티션 인덱스 생성
2. 테이블에 대해 파티션 필터링 사용 설정

파티션 인덱스 생성

AWS Glue에서 파티션 인덱스를 생성하는 단계는 AWS Glue 개발자 안내서의 파티션 인덱스 작업을 참조하세요. AWS Glue의 파티션 인덱스에 대한 제한 사항은 해당 페이지의 파티션 인덱스 정보 섹션을 참조하세요.

파티션 필터링 사용 설정

테이블에 대해 파티션 필터링을 사용 설정하려면 AWS Glue에서 새 테이블 속성을 설정해야 합니다. AWS Glue에서 테이블 속성을 설정하는 방법에 대한 단계는 파티션 프로젝션 설정 페이지를 참조하세요. AWS Glue에서 테이블 세부 정보를 편집할 때 테이블 속성(Table properties) 섹션에 다음 키-값 페어를 추가합니다.

- 키(Key)에서 `partition_filtering.enabled`를 추가합니다.
- 값(Value)에 `true`를 추가합니다.

`partition_filtering.enabled` 값을 `false`로 설정하여 언제든지 이 테이블에서 파티션 프로젝션을 비활성화할 수 있습니다.

위의 단계를 완료한 후 Athena 콘솔로 돌아가서 데이터를 쿼리할 수 있습니다.

파티션 인덱싱 및 필터링 사용에 대한 자세한 내용은 AWS 빅 데이터 블로그의 Improve Amazon Athena query performance using AWS Glue Data Catalog partition indexes를 참조하세요.

지리 공간 데이터 작업

AWS Glue는 기본적으로 WKT(Well-known Text), WKB(Well-Known Binary) 또는 기타 PostGIS 데이터 형식을 지원하지 않습니다. AWS Glue 분류자는 지리 공간 데이터를 구문 분석하며, CSV의 경우 `varchar`와 같이 해당 형식에 지원되는 데이터 형식을 사용하여 분류합니다. 다른 AWS Glue 테이블과 마찬가지로 Athena가 이러한 데이터 형식을 그대로 구문 분석할 수 있게 하려면 지리 공간 데이터에서 생성된 테이블의 속성을 업데이트해야 할 수 있습니다. 자세한 내용은 AWS Glue 크롤러 사용 및 CSV 파일 작업 단원을 참조하세요. Athena에서 AWS Glue 테이블의 일부 지리 공간 데이터 형식을 그대로 구문 분석하지 못할 수 있습니다. Athena에서 지리 공간 데이터 작업에 대한 자세한 내용은 지리 공간 데이터 쿼리 단원을 참조하세요.

Athena와 함께 ETL에 AWS Glue 작업 사용

AWS Glue 작업은 ETL 작업을 수행합니다. AWS Glue 작업은 소스에서 데이터를 추출하고 데이터를 변환한 다음 대상으로 로드하는 스크립트를 실행합니다. 자세한 내용은 AWS Glue 개발자 안내서의 AWS Glue에 작업 작성을 참조하세요.

AWS Glue ETL 작업에 Athena를 사용하여 테이블 생성

Athena에서 생성한 테이블에는 데이터 형식을 식별하는 `classification`이라는 테이블 속성이 추가되어야 합니다. 그러면 AWS Glue이(가) ETL 작업에 이 테이블을 사용할 수 있습니다. 분류 값은

`avro`, `csv`, `json`, `orc`, `parquet` 또는 `xml`일 수 있습니다. Athena에서의 `CREATE TABLE` 문 예제는 다음과 같습니다.

```
CREATE EXTERNAL TABLE sampleTable (
  column1 INT,
  column2 INT
  ) STORED AS PARQUET
  TBLPROPERTIES (
  'classification'='parquet')
```

테이블을 만들 때 테이블 속성을 추가하지 않았다면 AWS Glue 콘솔을 사용하여 추가할 수 있습니다.

AWS Glue 콘솔을 사용하여 분류 테이블 속성을 추가하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
2. 콘솔 탐색 창에서 테이블(Tables)을 선택합니다.
3. 편집할 테이블 링크를 선택한 다음 작업(Actions), 테이블 편집(Edit table)을 선택합니다.
4. 아래로 스크롤하여 테이블 속성 섹션을 찾습니다.
5. 추가를 선택합니다.
6. 키에 **`classification`**를 입력합니다.
7. 값(Value)에 데이터 유형(예:**`json`**)을 입력합니다.
8. 저장(Save)을 선택합니다.

   입력한 데이터 유형이 테이블 세부 정보(Table details) 섹션에서 테이블의 분류(Classification) 필드에 표시됩니다.

자세한 내용은 AWS Glue 개발자 안내서에서 테이블 관련 작업을 참조하세요.

ETL 작업을 사용하여 쿼리 성능 최적화

AWS Glue 작업은 Athena에서 쿼리 성능을 최적화하는 형식으로 데이터를 변환하는 데 도움이 됩니다. 데이터 형식은 Athena의 쿼리 성능 및 쿼리 비용에 큰 영향을 줍니다.

Parquet 및 ORC 데이터 형식 사용을 권장합니다. AWS Glue는 이 두 데이터 형식의 쓰기를 모두 지원하므로, 데이터를 Athena에 가장 적합한 형식으로 쉽고 빠르게 변환할 수 있습니다. 이러한 형식 및 성능을 개선하는 다른 방법에 대한 자세한 내용은 Amazon Athena를 위한 유용한 성능 튜닝 팁 10가지를 참조하세요.

ORC로 변환할 때 SMALLINT 및 TINYINT 데이터 형식을 INT로 변환

AWS Glue ETL 작업에서 생성한 `SMALLINT` 및 `TINYINT` 데이터 형식을 Athena가 읽지 못하게 될 가능성을 줄이려면, ETL 작업을 위해 스크립트를 작성하거나 마법사를 사용할 때 `SMALLINT` 및 `TINYINT`를 `INT`로 변환합니다.

ETL을 위한 AWS Glue 작업 자동화

트리거를 기반으로 AWS Glue ETL 작업이 자동으로 실행되도록 구성할 수 있습니다. AWS 외부의 데이터가 Athena에서 쿼리하기에 최적화되지 않은 형식으로 Amazon S3 버킷에 푸시되는 동안에는 이 기능이 이상적입니다. 자세한 내용은 AWS Glue 개발자 안내서의 AWS Glue 작업 트리거를 참조하세요.

AWS CLI를 사용하여 AWS Glue 데이터베이스 및 해당 테이블 다시 생성

AWS Glue 데이터베이스 이름을 직접 바꿀 수는 없지만 해당 정의를 복사하고 정의를 수정하며 해당 정의를 사용하여 데이터베이스를 다른 이름으로 다시 생성할 수는 있습니다. 마찬가지로 이전 데이터베이스의 테이블 정의를 복사하고 정의를 수정하며 수정된 정의를 사용하여 새 데이터베이스에서 테이블을 다시 생성할 수 있습니다.

> **Note**
>
> 제시된 방법에서는 테이블 파티셔닝을 복사하지 않습니다.

다음 Windows용 절차에서는 JSON 출력을 위해 AWS CLI가 구성되어 있다고 가정합니다. AWS CLI에서 기본 출력 형식을 변경하려면 `aws configure`를 실행합니다.

AWS CLI를 사용하여 AWS Glue 데이터베이스를 복사하려면

1. 명령 프롬프트에서 다음 AWS CLI 명령을 실행하여 복사하려는 AWS Glue 데이터베이스의 정의를 검색합니다.

   ```
   aws glue get-database --name database_name
   ```

   `get-database` 명령에 대한 자세한 내용은 get-database를 참조하세요.
2. 새 데이터베이스 이름(예:*new_database_name*`.json`)을 사용하는 데스크톱의 파일에 JSON 출력을 저장합니다.
3. 텍스트 편집기에서 *new_database_name*`.json` 파일을 엽니다.

4. JSON 파일에서 다음 단계를 수행합니다.

   a. 외부 `{ "Database":` 항목과 파일 끝에 있는 해당하는 중괄호(`}`)를 제거합니다.

   b. `Name` 항목을 새 데이터베이스 이름으로 변경합니다.

   c. `CatalogId` 필드를 제거합니다.

5. 파일을 저장합니다.

6. 명령 프롬프트에서 다음 AWS CLI 명령을 실행하여 수정된 데이터베이스 정의 파일을 사용해 새 이름으로 데이터베이스를 생성합니다.

```
aws glue create-database --database-input "file://~/Desktop\new_database_name.json"
```

   `create-database` 명령에 대한 자세한 내용은 create-database를 참조하세요. 파일에서 AWS CLI 파라미터 로드에 대한 자세한 내용은 AWS Command Line Interface 사용 설명서의 파일에서 AWS CLI 파라미터 로드를 참조하세요.

7. AWS Glue에서 새 데이터베이스가 생성되었는지 확인하려면 다음 명령을 실행합니다.

```
aws glue get-database --name new_database_name
```

이제 새 데이터베이스에 복사하려는 테이블의 정의를 가져와서 정의를 수정하고 수정된 정의를 사용하여 새 데이터베이스에서 테이블을 다시 생성할 준비가 되었습니다. 이 절차에서는 테이블 이름을 변경하지 않습니다.

AWS CLI를 사용하여 AWS Glue 테이블을 복사하려면

1. 명령 프롬프트에서 다음 AWS CLI 명령을 실행합니다.

```
aws glue get-table --database-name database_name --name table_name
```

   `get-table` 명령에 대한 자세한 내용은 get-table을 참조하세요.

2. 테이블 이름(예: *table_name*.json)을 사용하는 Windows 데스크톱의 파일에 JSON 출력을 저장합니다.

3. 텍스트 편집기에서 파일을 엽니다.

4. JSON 파일에서 외부 `{"Table":` 항목과 파일 끝에 있는 해당하는 중괄호(`}`)를 제거합니다.

5. JSON 파일에서 다음 항목과 해당 값을 제거합니다.

- `DatabaseName` - `create-table` CLI 명령에서 `--database-name` 파라미터를 사용하므로 이 항목은 필요하지 않습니다.
- `CreateTime`
- `UpdateTime`
- `CreatedBy`
- `IsRegisteredWithLakeFormation`
- `CatalogId`
- `VersionId`

6. 테이블 정의 파일을 저장합니다.
7. 명령 프롬프트에서 다음 AWS CLI 명령을 실행하여 새 데이터베이스에서 테이블을 다시 생성합니다.

```
aws glue create-table --database-name new_database_name --table-input "file://~/
Desktop\table_name.json"
```

`create-table` 명령에 대한 자세한 내용은 create-table을 참조하세요.

이제 테이블이 AWS Glue에서 새 데이터베이스에 나타나고 Athena에서 해당 테이블을 쿼리할 수 있습니다.

8. 단계를 반복하여 AWS Glue의 새 데이터베이스에 각각의 추가 테이블을 복사합니다.

외부 Hive 메타스토어용 Athena 데이터 커넥터 사용

Apache Hive 메타스토어를 사용하는 Amazon S3에서 데이터 세트를 쿼리하기 위해 외부 Hive 메타스토어에 Amazon Athena 데이터 커넥터를 사용할 수 있습니다. 메타데이터를 AWS Glue Data Catalog로 마이그레이션할 필요가 없습니다. Athena 관리 콘솔에서 프라이빗 VPC에 있는 Hive 메타스토어와 통신하는 Lambda 함수를 구성한 다음 메타스토어에 연결합니다. Lambda에서 Hive 메타스토어로의 연결은 프라이빗 Amazon VPC 채널로 보호되며 퍼블릭 인터넷을 사용하지 않습니다. 자체 Lambda 함수 코드를 제공하거나 외부 Hive 메타스토어에 대한 Athena 데이터 커넥터의 기본 구현을 사용할 수 있습니다.

주제

- 기능 개요
- 워크플로

- 고려 사항 및 제한
- Apache Hive 메타스토어에 Athena 연결
- AWS Serverless Application Repository를 사용하여 Hive 데이터 원본 커넥터 배포
- 기존 IAM 실행 역할을 사용하여 Athena를 Hive 메타스토어에 연결
- 배포된 Hive 메타스토어 커넥터를 사용하도록 Athena 구성
- 외부 Hive 메타스토어 쿼리에서 기본 데이터 원본 이름 사용
- Hive 뷰 작업
- Hive 메타스토어와 함께 AWS CLI 사용
- 참조 구현

기능 개요

외부 Hive 메타스토어에 대한 Athena 데이터 커넥터를 사용하여 다음 작업을 수행할 수 있습니다.

- Athena 콘솔을 사용하여 사용자 지정 카탈로그를 등록하고 사용자 지정 카탈로그를 사용하여 쿼리를 실행합니다.
- 여러 외부 Hive 메타스토어에 대한 Lambda 함수를 정의하고 Athena 쿼리에 조인합니다.
- 동일한 Athena 쿼리에서 외부 Hive 메타스토어 및 AWS Glue Data Catalog를 사용합니다.
- 쿼리 실행 컨텍스트의 카탈로그를 현재 기본 카탈로그로 지정합니다. 이렇게 하면 쿼리의 데이터베이스 이름에 카탈로그 이름을 접두사로 붙일 필요가 없습니다. *`catalog`*`.`*`database`*`.`*`table`* 구문을 사용하는 대신 *`database`*`.`*`table`*을 사용할 수 있습니다.
- 다양한 도구를 사용하여 외부 Hive 메타스토어를 참조하는 쿼리를 실행합니다. Athena 콘솔, AWS CLI, AWS SDK, Athena API, 업데이트된 Athena JDBC 및 ODBC 드라이버를 사용할 수 있습니다. 업데이트된 드라이버는 사용자 지정 카탈로그를 지원합니다.

API 지원

외부 Hive 메타스토어용 Athena 데이터 커넥터에는 카탈로그 등록 API 작업 및 메타데이터 API 작업에 대한 지원이 포함되어 있습니다.

- 카탈로그 등록 – 외부 Hive 메타스토어 및 연합 데이터 원본에 대한 사용자 지정 카탈로그를 등록합니다.
- 메타데이터 - 메타데이터 API를 사용하여 AWS Glue에 대한 데이터베이스 및 테이블 정보와 Athena를 통해 등록한 모든 카탈로그를 제공합니다.

- Athena JAVA SDK 클라이언트 - 업데이트된 Athena Java SDK 클라이언트에서 `StartQueryExecution` 작업의 카탈로그에 대한 지원, 메타데이터 API 및 카탈로그 등록을 사용합니다.

참조 구현

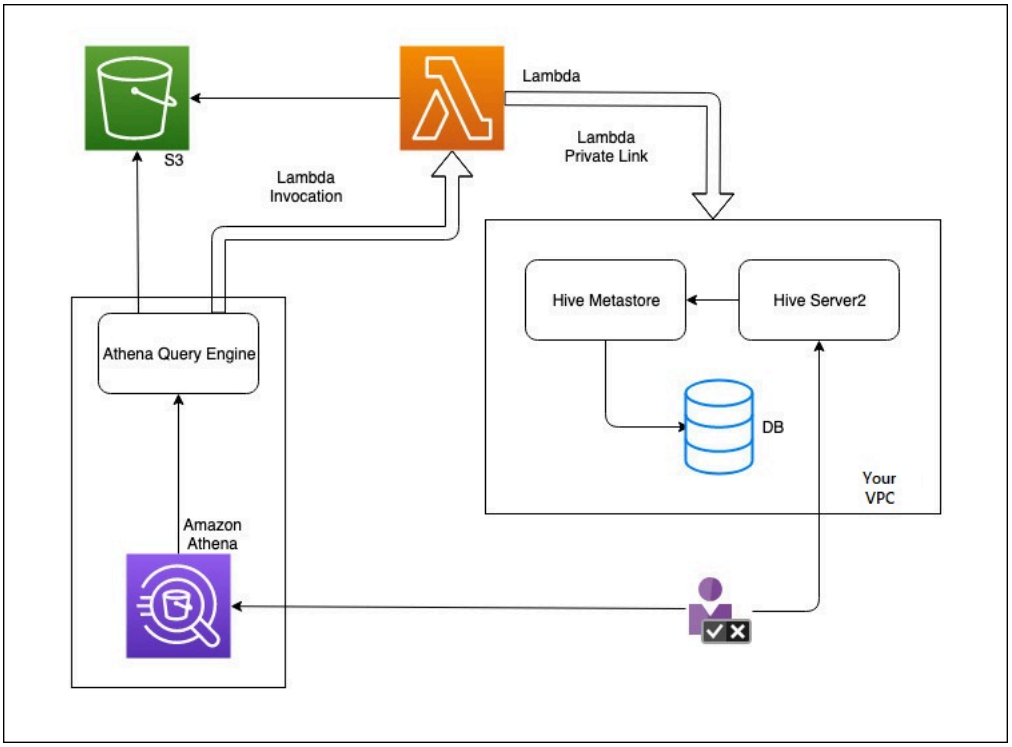
Athena는 외부 Hive 메타스토어에 연결하는 Lambda 함수에 대한 참조 구현을 제공합니다. 참조 구현은 Athena Hive 메타스토어의 오픈 소스 프로젝트로 GitHub에서 제공됩니다.

참조 구현은 AWS Serverless Application Repository(SAR)에서 다음 두 AWS SAM 애플리케이션으로 사용할 수 있습니다. SAR에서 이러한 애플리케이션 중 하나를 사용하여 Lambda 함수를 직접 만들 수 있습니다.

- **`AthenaHiveMetastoreFunction`** - Uber Lambda 함수 `.jar` 파일. "uber" JAR(fat JAR 또는 종속 항목이 있는 JAR이라고도 함)은 하나의 파일에 Java 프로그램과 해당 종속 항목을 모두 포함한 `.jar` 파일입니다.
- **`AthenaHiveMetastoreFunctionWithLayer`** - Lambda 계층 및 thin Lambda 함수 `.jar` 파일.

워크플로

다음 다이어그램은 Athena가 외부 Hive 메타스토어와 상호 작용하는 방식을 보여줍니다.



이 워크플로우에서는 데이터베이스에 연결된 Hive 메타스토어가 VPC 내에 있습니다. Hive CLI를 사용하는 Hive 메타스토어를 관리하는 데 Hive Server2를 사용합니다.

Athena의 외부 Hive 메타스토어를 사용하는 워크플로우에는 다음 단계가 포함됩니다.

1. VPC 내에 있는 Hive 메타스토어에 Athena를 연결하는 Lambda 함수를 생성합니다.
2. Hive 메타스토어에 고유한 카탈로그 이름과 계정에 해당 함수 이름을 등록합니다.
3. 카탈로그 이름을 사용하는 Athena DML 또는 DDL 쿼리를 실행하면 Athena 쿼리 엔진은 카탈로그 이름과 연결된 Lambda 함수 이름을 호출합니다.
4. Lambda 함수는 AWS PrivateLink를 사용하여 VPC의 외부 Hive 메타스토어와 통신하고 메타데이터 요청에 대한 응답을 수신합니다. Athena는 기본 AWS Glue Data Catalog의 데이데이터를 사용하는 방식과 동일하게 외부 Hive 메타스토어의 메타데이터를 사용합니다.

고려 사항 및 제한

외부 Hive 메타스토어용 Athena 데이터 커넥터를 사용하는 경우 다음 사항을 고려해야 합니다.

- CTAS를 사용하여 외부 Hive 메타스토어에서 테이블을 생성할 수 있습니다.
- INSERT INTO를 사용하여 외부 Hive 메타스토어에 데이터를 삽입할 수 있습니다.
- 외부 Hive 메타스토어에 대한 DDL 지원은 다음 문으로 제한됩니다.
  - ALTER DATABASE SET DBPROPERTIES
  - ALTER TABLE ADD COLUMNS
  - ALTER TABLE ADD PARTITION
  - ALTER TABLE DROP PARTITION
  - ALTER TABLE RENAME PARTITION
  - ALTER TABLE REPLACE COLUMNS
  - ALTER TABLE SET LOCATION
  - ALTER TABLE SET TBLPROPERTIES
  - 데이터베이스 생성
  - CREATE TABLE
  - CREATE TABLE AS
  - DESCRIBE TABLE
  - DROP DATABASE
  - DROP TABLE

  - SHOW COLUMNS
  - SHOW CREATE TABLE
  - SHOW PARTITIONS
  - SHOW SCHEMAS
  - SHOW TABLES
  - SHOW TBLPROPERTIES
- 보유할 수 있는 등록된 카탈로그의 최대 수는 1,000개입니다.
- Hive 메타스토어에 대한 Kerberos 인증은 지원되지 않습니다.
- JDBC 드라이버와 함께 연합 쿼리나 외부 Hive 메타스토어를 사용하려면 JDBC 연결 문자열에 `MetadataRetrievalMethod=ProxyAPI`를 포함해야 합니다. JDBC 드라이버에 대한 자세한 내용은 JDBC로 Amazon Athena에 연결 섹션을 참조하세요.
- Hive 숨김 열 `$path`, `$bucket`, `$file_size`, `$file_modified_time`, `$partition`, `$row_id`는 세분화된 액세스 제어 필터링에 사용할 수 없습니다.
- Hive 숨김 시스템 테이블(예: *`example_table`*`$partitions` 또는 *`example_table`*`$properties`)은 세분화된 액세스 제어에서 지원되지 않습니다.

권한

사전 구축된 사용자 지정 데이터 커넥터가 올바르게 작동하려면 다음 리소스에 액세스해야 할 수 있습니다. 사용하는 커넥터의 정보를 확인하여 VPC를 올바르게 구성했는지 확인합니다. Athena에서 쿼리를 실행하고 데이터 원본 커넥터를 만드는 데 필요한 IAM 권한에 대한 자세한 내용은 외부 Hive 메타스토어용 Athena 데이터 커넥터에 대한 액세스 허용 및 외부 Hive 메타스토어에 대한 Lambda 함수 액세스 허용 단원을 참조하세요.

- Amazon S3 – 쿼리 결과를 Amazon S3의 Athena 쿼리 결과 위치에 작성하는 것 외에도 데이터 커넥터는 Amazon S3의 유출 버킷에 작성합니다. 이 Amazon S3 위치에 대한 연결 및 권한이 필요합니다. 자세한 내용은 이 주제의 후반부에서 Amazon S3의 분산 위치 단원을 참조하세요.
- Athena – 쿼리 상태를 확인하고 오버스캔을 방지하려면 액세스가 필요합니다.
- AWS Glue - 커넥터가 보충 또는 기본 메타데이터에 AWS Glue를 사용하는 경우 액세스가 필요합니다.
- AWS Key Management Service
- 정책 - Hive 메타스토어, Athena Query Federation 및 UDF에는 AWS 관리형 정책: AmazonAthenaFullAccess 외의 정책이 필요합니다. 자세한 내용은 Athena의 자격 증명 및 액세스 관리 단원을 참조하세요.

Amazon S3의 분산 위치

Athena 함수 응답 크기에는 제한이 있기 때문에 임계값을 초과하는 응답은 Lambda 함수를 생성할 때 지정한 Amazon S3 위치로 분산됩니다. Athena는 이러한 응답을 Amazon S3로부터 직접 읽습니다.

> **Note**
>
> Athena는 Amazon S3의 응답 파일을 제거하지 않습니다. 응답 파일을 자동으로 삭제하도록 보존 정책을 설정하는 것이 좋습니다.

Apache Hive 메타스토어에 Athena 연결

Apache Hive 메타스토어에 Athena를 연결하려면 Lambda 함수를 만들고 구성해야 합니다. 기본 구현의 경우 Athena 관리 콘솔에서 시작하는 모든 필수 단계를 수행할 수 있습니다.

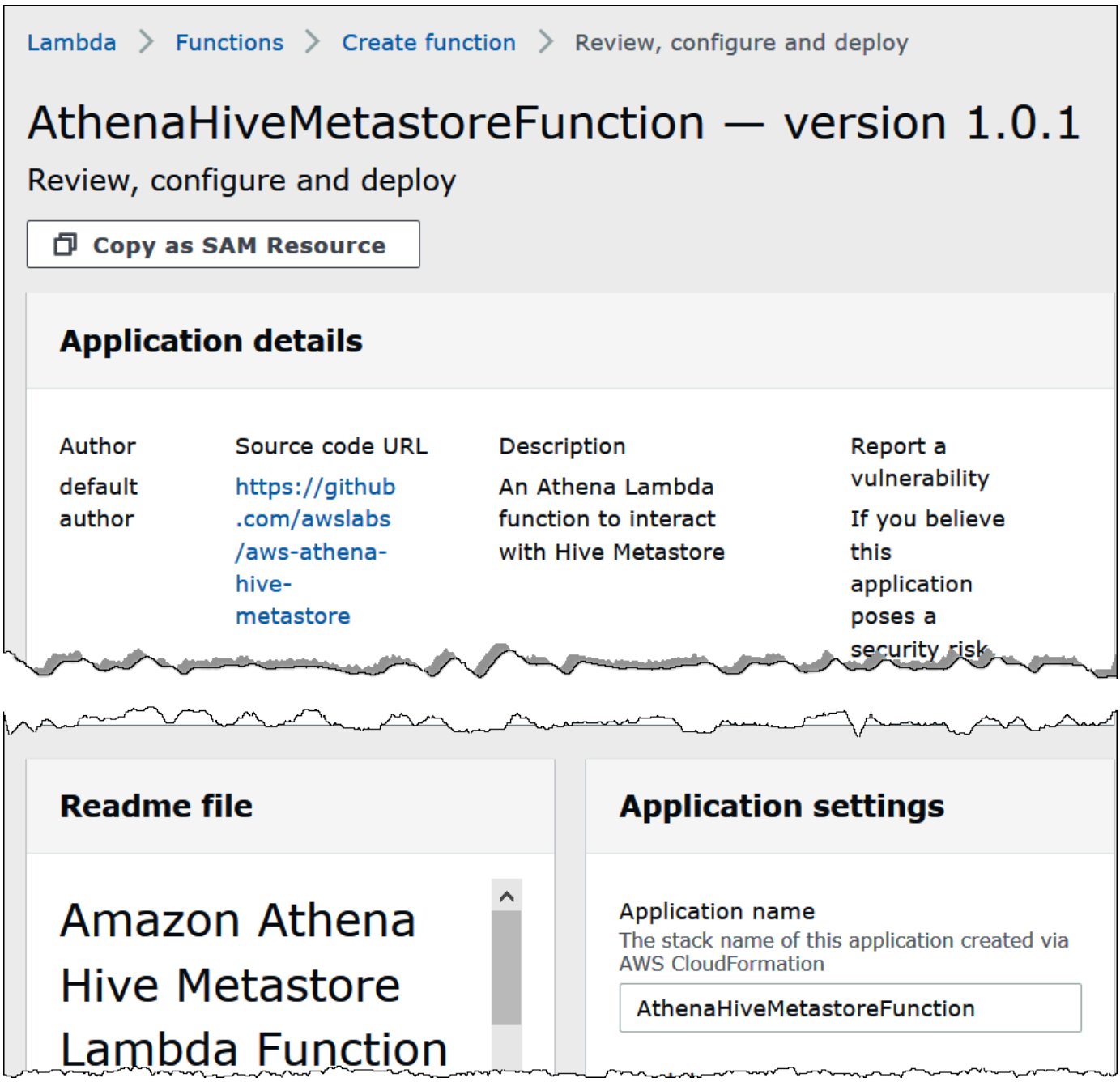
> **Note**
>
> 다음 절차를 수행하려면 Lambda 함수에 대한 사용자 지정 IAM 역할을 생성할 권한이 있어야 합니다. 사용자 지정 역할을 만들 수 있는 권한이 없는 경우 Athena 참조 구현을 사용하여 Lambda 함수를 별도로 만든 다음 AWS Lambda 콘솔에서 함수에 대한 기존 IAM 역할을 선택합니다. 자세한 내용은 기존 IAM 실행 역할을 사용하여 Athena를 Hive 메타스토어에 연결 단원을 참조하세요.

Hive 메타스토어에 Athena를 연결하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

3. 데이터 소스(Data Source)을 선택합니다.
4. 콘솔 오른쪽 위에서 데이터 원본 생성(Create data source)을 선택합니다.
5. 데이터 원본 선택(Choose data sources) 페이지에서 데이터 원본(Data source)에 대해 S3 - Apache Hive 메타스토어(S3 - Apache Hive metastore)를 선택합니다.
6. 다음을 선택합니다.
7. 데이터 원본 세부 정보(Data source details) 섹션의 데이터 원본 이름(Data source name)에 Athena에서 데이터 원본을 쿼리할 때 SQL 문에 사용할 이름을 입력합니다. 이름은 127자까지 입력할 수 있으며 계정 내에서 고유해야 합니다. 생성 후에는 변경할 수 없습니다. 유효한 문자는 a-z, A-Z, 0-9, _(밑줄), @(앰퍼샌드) 및 -(하이픈)입니다. `awsdatacatalog`, `hive`, `jmx`, `system` 이름은 Athena에 예약되어 있으므로 데이터 원본 이름에 사용할 수 없습니다.
8. Lambda 함수(Lambda function)에서 새 Lambda 함수 생성(Create a new Lambda function)을 선택한 다음AWS Lambda에서 새 Lambda 함수 생성(Create a new Lambda function)을 선택합니다.

   AWS Lambda 콘솔에서 AthenaHiveMetastoreFunction 페이지가 열립니다. 이 페이지에는 커넥터에 대한 자세한 정보가 포함되어 있습니다.

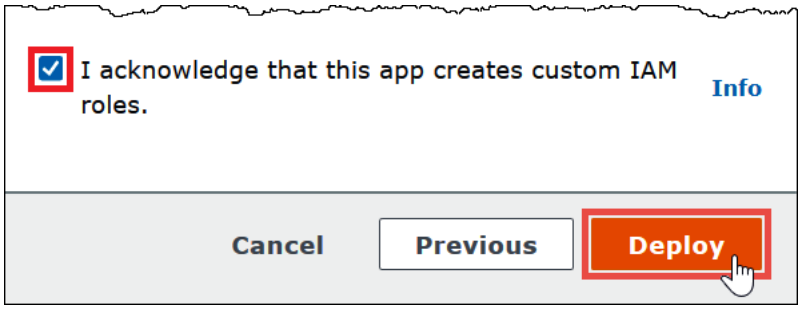


9. 애플리케이션 설정(Application settings)에서 Lambda 함수에 대한 파라미터를 입력합니다.

   - LambdaFuncName – 함수의 이름을 제공합니다. myHiveMetastore를 예로 들 수 있습니다.
   - SpillLocation – Lambda 함수 응답 크기가 4MB를 초과하는 경우 분산 메타데이터를 보관할 이 계정의 Amazon S3 위치를 지정합니다.
   - HMSUris – 포트 9083에서 Thrift 프로토콜을 사용하는 Hive 메타스토어 호스트의 URI를 입력합니다. `thrift://<host_name>:9083` 구문을 사용합니다.

- LambdaMemory - 128MB ~ 3008MB 사이의 값을 지정합니다. Lambda 함수는 사용자가 구성하는 메모리 양에 비례하여 할당된 CPU 주기입니다. 기본값은 1024입니다.
- LambdaTimeout – 허용되는 최대 Lambda 호출 실행 시간을 1초 단위로 1~900초 범위에서 지정합니다(900초는 15분). 기본값은 300초(5분)입니다.
- VPCSecurityGroupIds – Hive 메타스토어에 대한 VPC 보안 그룹 ID 목록을 입력합니다(쉼표로 구분).
- VPCSubnetIds – Hive 메타스토어에 대한 VPC 서브넷 ID 목록을 입력합니다(쉼표로 구분).

10. 이 앱은 사용자 지정 IAM 역할을 생성한다는 데 동의합니다(I acknowledge that this app creates custom IAM roles)를 선택하고 배포(Deploy)를 선택합니다.



배포가 완료되면 함수가 Lambda 애플리케이션 목록에 나타납니다. 이제 Hive 메타스토어 기능이 계정에 배포되었으므로 이를 사용하도록 Athena를 구성할 수 있습니다.

11. <shared id="ATE"/> 콘솔의 데이터 원본 입력(Enter data sources) 페이지로 돌아갑니다.
12. Lambda 함수 섹션에서 Lambda 검색 상자 옆에 있는 새로 고침 아이콘을 선택합니다. 사용 가능한 함수 목록을 새로 고치면 새로 만든 함수가 목록에 나타납니다.
13. Lambda 콘솔에서 방금 생성한 함수의 이름을 선택합니다. Lambda 함수의 ARN이 표시됩니다.
14. (선택 사항) 태그(Tags)에 대해 이 데이터 원본과 연결할 키-값 페어를 추가합니다. 태그에 대한 자세한 내용은 Athena 리소스 태깅 단원을 참조하세요.
15. 다음을 선택합니다.
16. 검토 및 생성 페이지에서 데이터 원본 세부 정보를 검토한 다음 데이터 원본 생성을 선택합니다.
17. 데이터 원본 페이지의 데이터 원본 세부 정보 섹션에는 새 커넥터에 대한 정보가 표시됩니다.

    이제 지정한 데이터 원본 이름(Data source name)을 사용하여 <shared id="ATE"/> SQL 쿼리에서 Hive 메타스토어를 참조할 수 있습니다. SQL 쿼리에서 다음 예제 구문을 사용하여 `hms-catalog-1`을 앞에서 지정한 카탈로그 이름으로 바꿉니다.

```
SELECT * FROM hms-catalog-1.CustomerData.customers
```

18. 생성한 데이터 원본 보기, 편집 또는 삭제에 대한 자세한 내용은 데이터 원본 관리 섹션을 참조하세요.

AWS Serverless Application Repository를 사용하여 Hive 데이터 원본 커넥터 배포

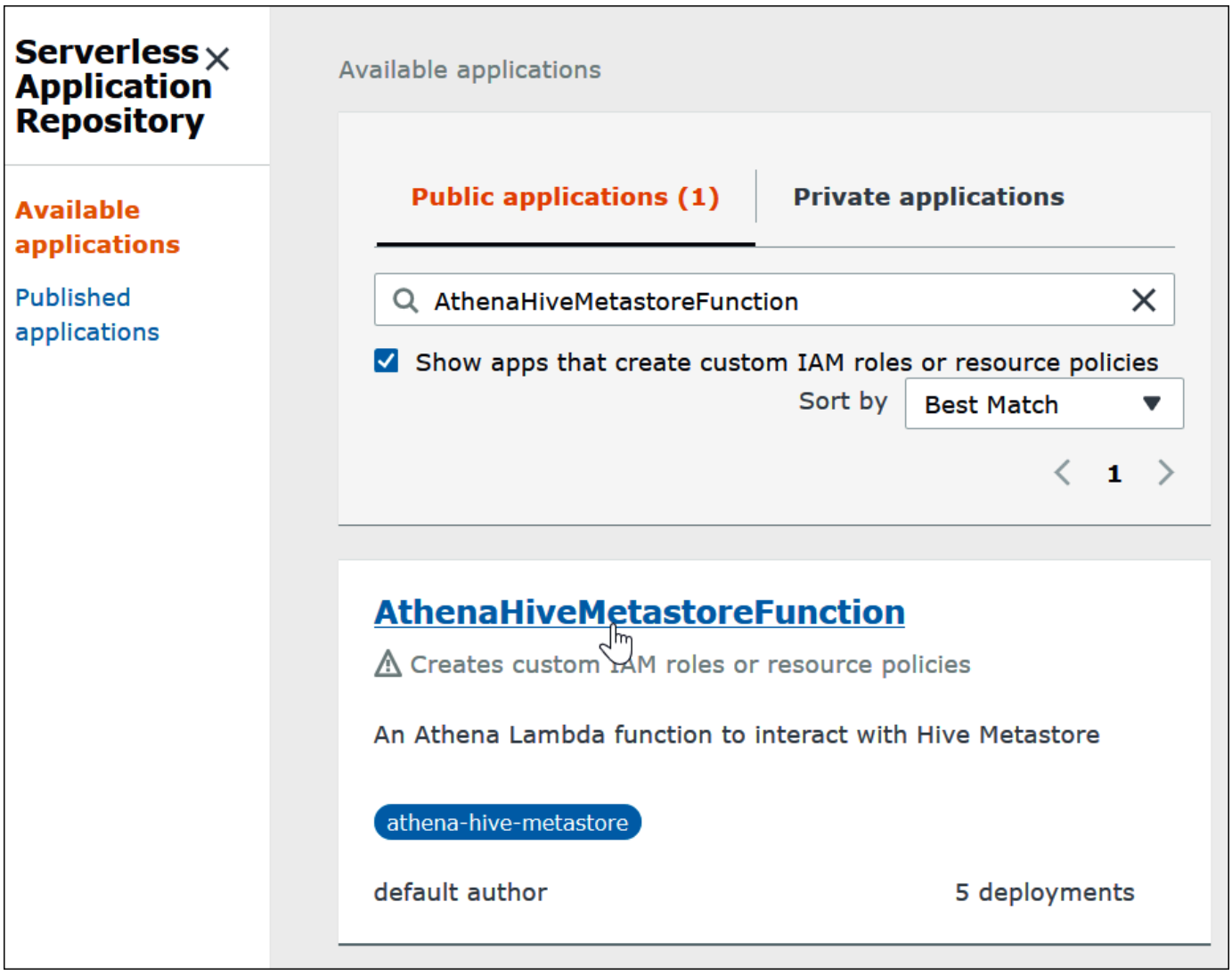
Hive용 Athena 데이터 원본 커넥터를 배포하려면 Athena 콘솔로 시작하는 대신 AWS Serverless Application Repository를 사용합니다. AWS Serverless Application Repository로 사용할 커넥터를 찾고 커넥터에 필요한 파라미터를 제공한 다음 계정에 커넥터를 배포합니다. 그런 다음 커넥터를 배포한 후 Athena 콘솔을 사용하여 Athena에서 데이터 원본을 사용할 수 있도록 합니다.

AWS Serverless Application Repository를 사용하여 Hive용 데이터 원본 커넥터를 계정에 배포하려면

1. AWS Management Console에 로그인하고 서버리스 앱 리포지토리를 엽니다.
2. 탐색 창에서 사용 가능한 애플리케이션을 선택합니다.
3. 사용자 지정 IAM 역할 또는 리소스 정책을 만드는 앱 표시(Show apps that create custom IAM roles or resource policies) 옵션을 선택합니다.
4. 검색 상자에 **`Hive`**을(를) 입력합니다. 나타나는 커넥터에는 다음 두 가지가 포함됩니다.

   - AthenaHiveMetastoreFunction – Uber Lambda 함수 `.jar` 파일.
   - AthenaHiveMetastoreFunctionWithLayer – Lambda 계층 및 thin Lambda 함수 `.jar` 파일.

   두 애플리케이션은 동일한 기능을 가지며 구현만 다릅니다. 둘 중 하나를 사용하여 Hive 메타스토어에 Athena를 연결하는 Lambda 함수를 만들 수 있습니다.
5. 삭제할 커넥터의 이름을 선택합니다. 이 자습서에서는 AthenaHiveMetastoreFunction을 사용합니다.



6. 애플리케이션 설정(Application settings)에서 Lambda 함수에 대한 파라미터를 입력합니다.

   - LambdaFuncName – 함수의 이름을 제공합니다. myHiveMetastore를 예로 들 수 있습니다.
   - SpillLocation – Lambda 함수 응답 크기가 4MB를 초과하는 경우 분산 메타데이터를 보관할 이 계정의 Amazon S3 위치를 지정합니다.
   - HMSUris – 포트 9083에서 Thrift 프로토콜을 사용하는 Hive 메타스토어 호스트의 URI를 입력합니다. `thrift://<host_name>:9083` 구문을 사용합니다.
   - LambdaMemory - 128MB ~ 3008MB 사이의 값을 지정합니다. Lambda 함수는 사용자가 구성하는 메모리 양에 비례하여 할당된 CPU 주기입니다. 기본값은 1024입니다.
   - LambdaTimeout – 허용되는 최대 Lambda 호출 실행 시간을 1초 단위로 1~900초 범위에서 지정합니다(900초는 15분). 기본값은 300초(5분)입니다.
   - VPCSecurityGroupIds – Hive 메타스토어에 대한 VPC 보안 그룹 ID 목록을 입력합니다(쉼표로 구분).

- VPCSubnetIds – Hive 메타스토어에 대한 VPC 서브넷 ID 목록을 입력합니다(쉼표로 구분).

7. 애플리케이션 세부 정보(Application details) 페이지의 오른쪽 아래에서 이 앱에서 사용자 지정 IAM 역할을 생성하는 것을 인정합니다(I acknowledge that this app creates custom IAM roles)를 선택하고 배포(Deploy)를 선택합니다.

이때 Lambda 함수를 사용하여 Hive 메타스토어에 연결하도록 Athena를 구성할 수 있습니다. 단계는 배포된 Hive 메타스토어 커넥터를 사용하도록 Athena 구성를 참조하세요.

기존 IAM 실행 역할을 사용하여 Athena를 Hive 메타스토어에 연결

기존 IAM 역할을 사용하는 Lambda 함수로 외부 Hive 메타스토어를 Athena에 연결하려면 외부 Hive 메타스토어용 Athena 커넥터의 Athena 참조 구현을 사용합니다.

3가지 주요 단계는 다음과 같습니다.

1. 복제 및 구축 - Athena 참조 구현을 복제하고 Lambda 함수 코드를 포함하는 JAR 파일을 구축합니다.
2. AWS Lambda 콘솔 - AWS Lambda 콘솔에서 Lambda 함수를 생성하고 기존 IAM 실행 역할을 할당한 다음, 생성한 함수 코드를 업로드합니다.
3. Amazon Athena 콘솔 - Amazon Athena 콘솔에서, Athena 쿼리에서 외부 Hive 메타스토어를 참조하는 데 사용할 수 있는 데이터 원본 이름을 생성합니다.

사용자 지정 IAM 역할을 생성할 수 있는 권한이 이미 있는 경우 Athena 콘솔과 AWS Serverless Application Repository를 사용한 간편한 워크플로를 사용하여 Lambda 함수를 만들고 구성합니다. 자세한 내용은 Apache Hive 메타스토어에 Athena 연결 단원을 참조하십시오.

사전 조건

- Git이 시스템에 설치되어 있어야 합니다.
- Apache Maven이 설치되어 있어야 합니다.
- Lambda 함수에 할당할 수 있는 IAM 실행 역할이 있습니다. 자세한 내용은 외부 Hive 메타스토어에 대한 Lambda 함수 액세스 허용 단원을 참조하세요.

Lambda 함수 복제 및 구축

Athena 참조 구현을 위한 함수 코드는 GitHub의 awslabs/aws-athena-hive-metastore에 있는 Maven 프로젝트입니다. 프로젝트에 대한 자세한 내용은 GitHub의 해당 README 파일 또는 본 문서의 참조 구현 주제를 참조하세요.

Lambda 함수 코드를 복제하고 빌드하려면

1. Athena 참조 구현을 복제하려면 다음 명령을 입력합니다.

   ```
   git clone https://github.com/awslabs/aws-athena-hive-metastore
   ```

2. 다음 명령을 실행하여 Lambda 함수를 위한 `.jar` 파일을 빌드합니다.

   ```
   mvn clean install
   ```

   프로젝트가 성공적으로 빌드되면 다음 `.jar` 파일이 프로젝트의 대상 폴더에 생성됩니다.

   `hms-lambda-func-1.0-SNAPSHOT-withdep.jar`

   다음 단원에서는 AWS Lambda 콘솔을 사용하여 이 파일을 Amazon Web Services 계정에 업로드합니다.

AWS Lambda 콘솔에서 Lambda 함수 생성 및 구성

이 단원에서는 AWS Lambda 콘솔을 사용하여 기존 IAM 실행 역할을 사용하는 함수를 생성합니다. 함수에 대한 VPC를 구성한 후 함수 코드를 업로드하고 함수의 환경 변수를 구성합니다.

Lambda 함수 생성

이 단원에서는 AWS Lambda 콘솔에서 기존 IAM 실행 역할을 사용하는 함수를 생성합니다.

기존 IAM 역할을 사용하는 Lambda 함수를 만들려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/lambda/에서 AWS Lambda 콘솔을 엽니다.
2. 탐색 창에서 함수를 선택합니다.
3. 함수 생성을 선택합니다.
4. 새로 작성을 선택합니다.

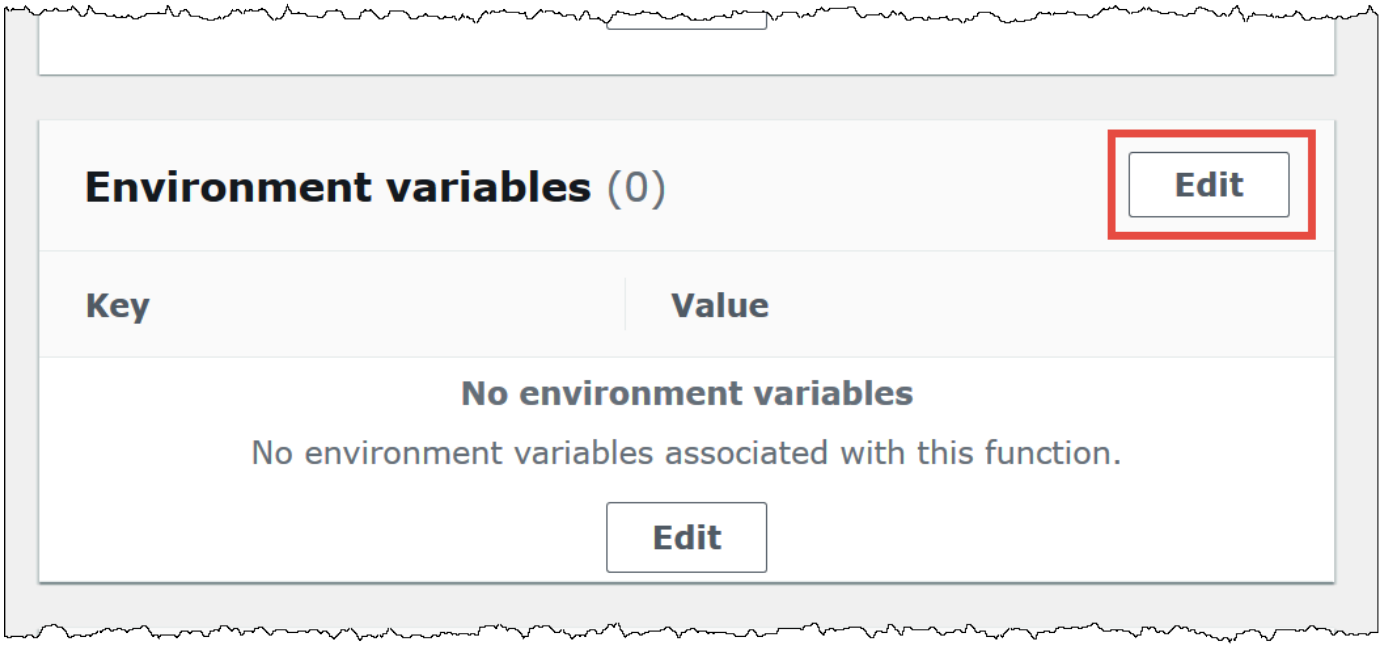
5. 함수 이름(Function name)에 Lambda 함수의 이름(예: **EHMSBasedLambda**)을 입력합니다.
6. 런타임(Runtime)에서 Java 8을 선택합니다.
7. 권한(Permissions)에서 기본 실행 역할 변경(Change default execution role)을 확장합니다.
8. 실행 역할에서 기존 역할 사용을 선택합니다.
9. 기존 역할(Existing role)에서 Lambda 함수가 Athena에 사용할 IAM 역할을 선택합니다(이 예제에서는 `AthenaLambdaExecutionRole`이라는 역할 사용).
10. Advanced settings(고급 설정)를 확장합니다.
11. 네트워크 활성화(Enable Network)를 선택합니다.
12. VPC에서 함수가 액세스할 VPC를 선택합니다.
13. 서브넷(Subnets)에서 Lambda가 사용할 VPC 서브넷을 선택합니다.
14. 보안 그룹(Security groups)에서 Lambda가 사용할 VPC 보안 그룹을 선택합니다.
15. 함수 생성을 선택합니다. AWS Lambda 콘솔에서 함수의 구성 페이지를 열고 함수 생성을 시작합니다.

코드 업로드 및 Lambda 함수 구성

콘솔에서 함수가 성공적으로 생성되었음을 알리면 함수 코드를 업로드하고 환경 변수를 구성할 준비가 된 것입니다.

Lambda 함수 코드를 업로드하고 해당 환경 변수를 구성하려면

1. Lambda 콘솔에서 지정한 함수 페이지의 코드(Code) 탭에 있는지 확인합니다.
2. 코드 소스(Code source)에서 다음에서 업로드(Upload from)를 선택한 다음 .zip 또는 .jar 파일(.zip or .jar file)을 선택합니다.
3. 이전에 생성한 `hms-lambda-func-1.0-SNAPSHOT-withdep.jar` 파일을 업로드합니다.
4. Lambda 함수 페이지에서 구성(Configuration) 탭을 선택합니다.
5. 왼쪽의 창에서 환경 변수(Environment variables)를 선택합니다.
6. 환경 변수 섹션에서 편집을 선택합니다.

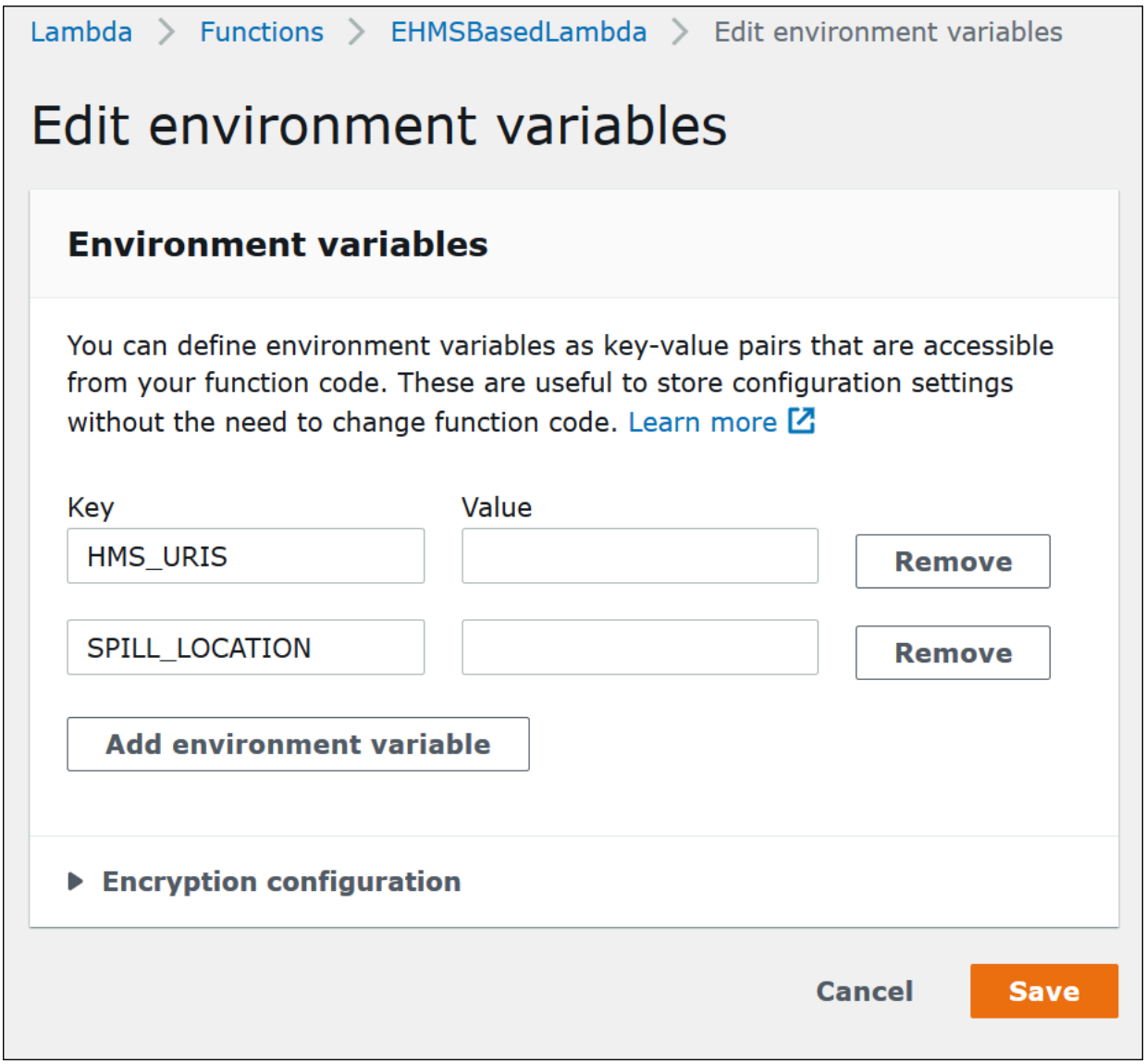


7. 환경 변수 편집(Edit environment variables) 페이지에서 환경 변수 추가(Add environment variable) 옵션을 사용하여 다음 환경 변수 키 및 값을 추가합니다.

   - HMS_URIS – 다음 구문을 사용하여 포트 9083에서 Thrift 프로토콜을 사용하는 Hive 메타스토어 호스트의 URI를 입력합니다.

     ```
     thrift://<host_name>:9083
     ```

   - SPILL_LOCATION – Lambda 함수 응답 크기가 4MB를 초과하는 경우 분산 메타데이터를 보관할 Amazon Web Services 계정의 Amazon S3 위치를 지정합니다.



8. Save(저장)를 선택합니다.

이제 Lambda 함수를 사용하여 Hive 메타스토어에 연결하도록 Athena를 구성할 준비가 되었습니다. 단계는 배포된 Hive 메타스토어 커넥터를 사용하도록 Athena 구성를 참조하세요.

배포된 Hive 메타스토어 커넥터를 사용하도록 Athena 구성

`AthenaHiveMetastoreFunction`과 같은 Lambda 데이터 원본 커넥터를 계정에 배포한 후 이를 사용하도록 Athena를 구성할 수 있습니다. 이를 위해 Athena 쿼리에 사용할 외부 Hive 메타스토어를 참조하는 데이터 원본 이름을 생성합니다.

기존 Lambda 함수를 사용하여 Hive 메타스토어에 Athena 연결

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

3. 데이터 소스(Data Source)을 선택합니다.
4. 데이터 원본(Data sources) 페이지에서 데이터 원본 연결(Connect data source)을 선택합니다.
5. 데이터 원본 선택(Choose data sources) 페이지에서 데이터 원본(Data source)에 대해 S3 - Apache Hive 메타스토어(S3 - Apache Hive metastore)를 선택합니다.
6. 다음을 선택합니다.
7. 데이터 원본 세부 정보(Data source details) 섹션의 데이터 원본 이름(Data source name)에 Athena에서 데이터 원본을 쿼리할 때 SQL 문에 사용할 이름을 입력합니다(예: `MyHiveMetastore`). 이름은 127자까지 입력할 수 있으며 계정 내에서 고유해야 합니다. 생성 후에는 변경할 수 없습니다. 유효한 문자는 a-z, A-Z, 0-9, _(밑줄), @(앰퍼샌드) 및 -(하이픈)입니다. `awsdatacatalog`, `hive`, `jmx`, `system` 이름은 Athena에 예약되어 있으므로 데이터 원본 이름에 사용할 수 없습니다.
8. 연결 세부 정보 섹션에서, Lambda 함수 선택 또는 입력 상자를 사용해 방금 만든 함수의 이름을 선택합니다. Lambda 함수의 ARN이 표시됩니다.
9. (선택 사항) 태그(Tags)에 대해 이 데이터 원본과 연결할 키-값 페어를 추가합니다. 태그에 대한 자세한 내용은 Athena 리소스 태깅 단원을 참조하세요.
10. 다음을 선택합니다.
11. 검토 및 생성 페이지에서 데이터 원본 세부 정보를 검토한 다음 데이터 원본 생성을 선택합니다.
12. 데이터 원본 페이지의 데이터 원본 세부 정보 섹션에는 새 커넥터에 대한 정보가 표시됩니다.

이제 Athena에서는 지정한 데이터 원본 이름(Data source name)을 사용하여 SQL 쿼리에서 Hive 메타스토어를 참조할 수 있습니다.

SQL 쿼리에서 다음 예제 구문을 사용하여 `ehms-catalog`를 앞에서 지정한 데이터 원본 이름으로 바꿉니다.

```
SELECT * FROM ehms-catalog.CustomerData.customers
```

13. 생성한 데이터 원본을 보거나 편집하거나 삭제하려면 데이터 원본 관리 단원을 참조하세요.

외부 Hive 메타스토어 쿼리에서 기본 데이터 원본 이름 사용

외부 Hive 메타스토어에서 DML 및 DDL 쿼리 실행 시, 쿼리 편집기에서 카탈로그 이름을 선택한 경우 해당 카탈로그 이름을 생략하여 쿼리 구문을 간소화할 수 있습니다. 이 기능에는 특정 제한 사항이 적용됩니다.

DML 문

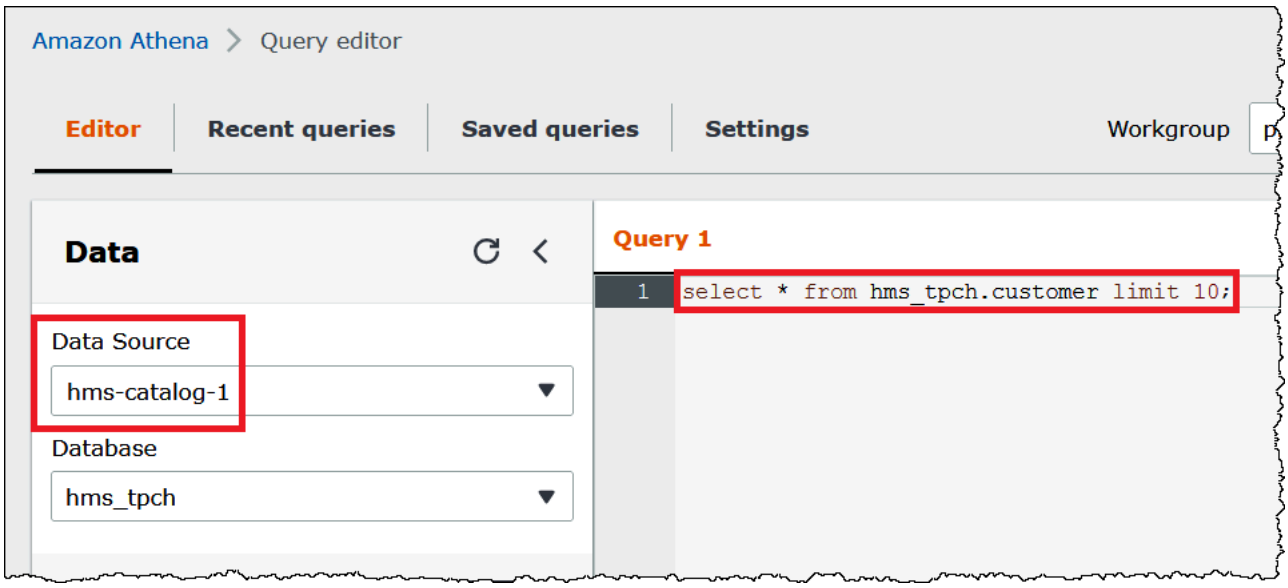
등록된 카탈로그를 사용하여 쿼리를 실행하려면

1. 다음 예와 같이 [[*data_source_name*].*database_name*].*table_name* 구문을 사용하여 데이터베이스 앞에 데이터 원본 이름을 배치할 수 있습니다.

```
select * from  "hms-catalog-1".hms_tpch.customer limit 10;
```

2. 사용하려는 데이터 원본이 쿼리 편집기에서 이미 선택되어 있으면 다음 예와 같이 쿼리에서 해당 이름을 생략할 수 있습니다.

```
select * from hms_tpch.customer limit 10:
```



3. 쿼리에서 여러 데이터 원본을 사용하는 경우 기본 데이터 원본 이름만 생략할 수 있으며 기본이 아닌 데이터 원본에 대해 전체 이름을 지정해야 합니다.

   예를 들어 쿼리 편집기에서 `AwsDataCatalog`가 기본 데이터 원본으로 선택되었다고 가정합니다. 다음 쿼리 발췌문의 `FROM` 문은 처음 두 데이터 원본 이름을 완전히 정규화하지만 세 번째 데이터 원본의 이름은 AWS Glue 데이터 카탈로그에 있으므로 생략합니다.

```
...
FROM ehms01.hms_tpch.customer,
         "hms-catalog-1".hms_tpch.orders,
         hms_tpch.lineitem
...
```

DDL 문

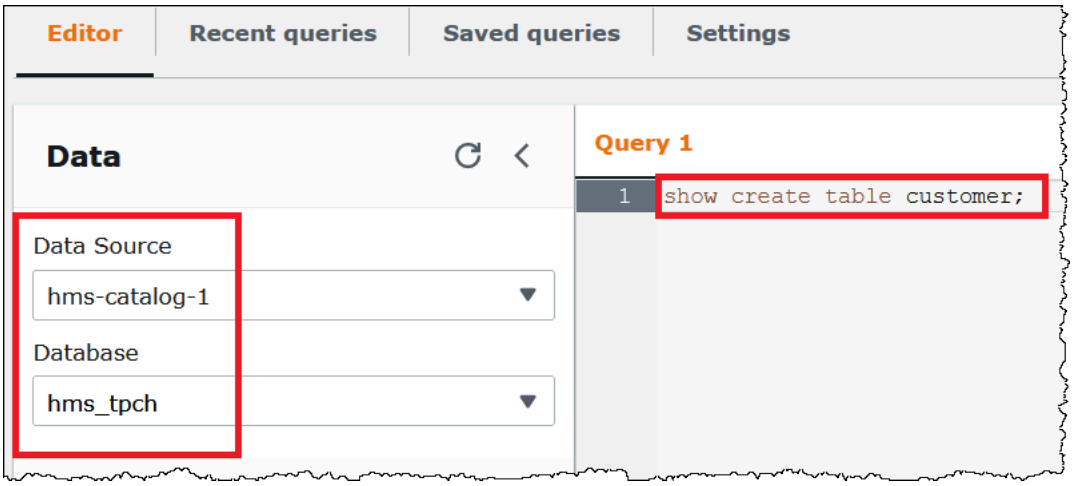
다음 Athena DDL 문은 카탈로그 이름 접두사를 지원합니다. 다른 DDL 문의 카탈로그 이름 접두사로 인해 구문 오류가 발생합니다.

```
SHOW TABLES [IN [catalog_name.]database_name] ['regular_expression']

SHOW TBLPROPERTIES [[catalog_name.]database_name.]table_name [('property_name')]

SHOW COLUMNS IN [[catalog_name.]database_name.]table_name

SHOW PARTITIONS [[catalog_name.]database_name.]table_name
```

```
SHOW CREATE TABLE [[catalog_name.][database_name.]table_name

DESCRIBE [EXTENDED | FORMATTED] [[catalog_name.][database_name.]table_name [PARTITION
 partition_spec] [col_name ( [.field_name] | [.'$elem$'] | [.'$key$'] | [.'$value$'] )]
```

DML 문과 마찬가지로 쿼리 편집기에서 데이터 원본과 데이터베이스를 선택할 때 쿼리에서 데이터 원본과 데이터베이스 접두사를 생략할 수 있습니다.

다음 이미지에서는 `hms-catalog-1` 데이터 원본 및 `hms_tpch` 데이터베이스가 쿼리 편집기에서 선택되어 있습니다. `show create table customer` 문은 `hms-catalog-1` 접두사와 `hms_tpch` 데이터베이스 이름이 쿼리 자체에서 생략된 경우에도 성공합니다.



JDBC 연결 문자열에 기본 데이터 원본 지정

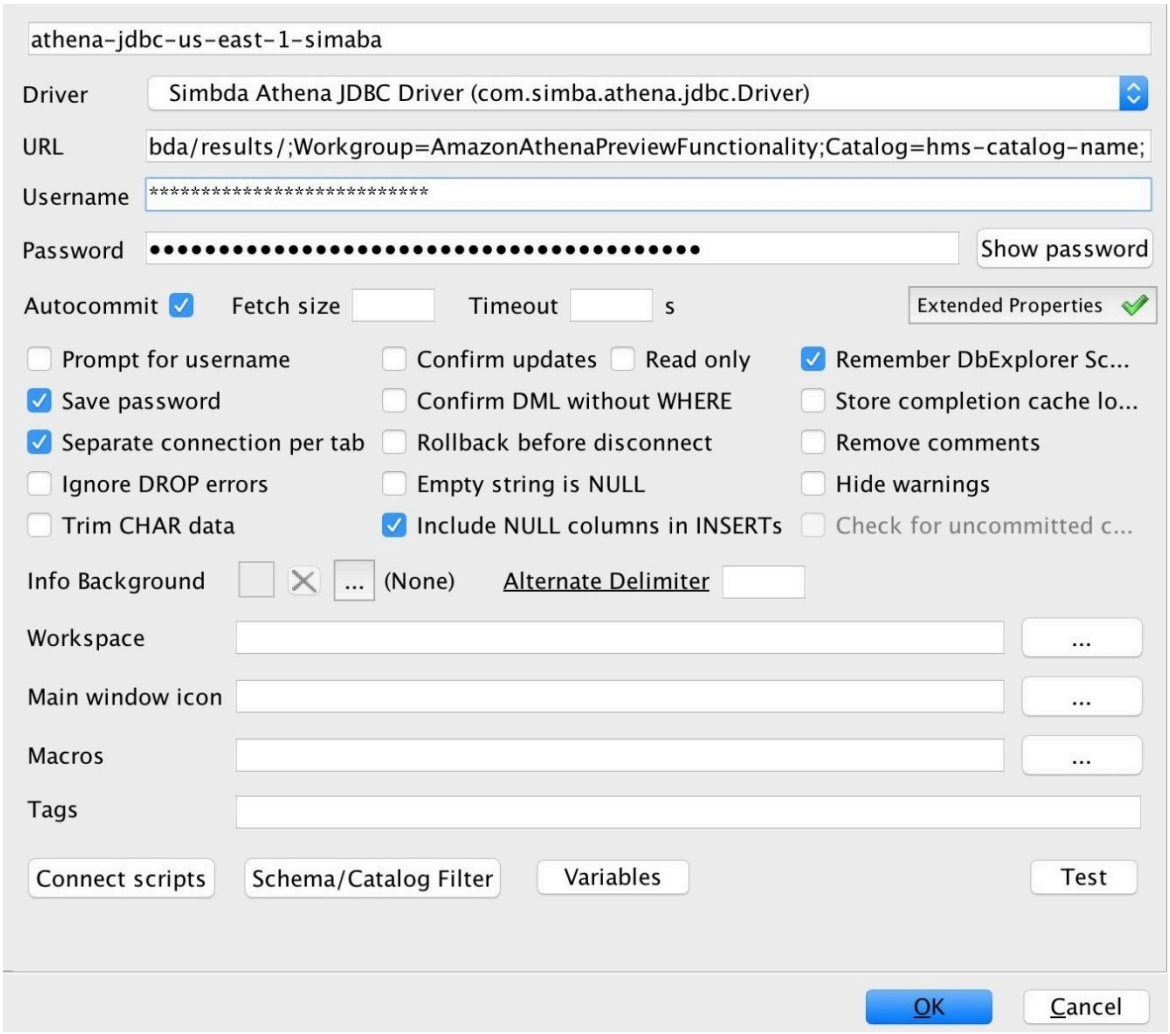
Athena JDBC 드라이버를 사용하여 Athena를 외부 Hive 메타스토어에 연결할 때 `Catalog` 파라미터를 사용하여 SQL workbench와 같은 SQL 편집기에서 연결 문자열의 기본 데이터 원본 이름을 지정할 수 있습니다.

> **Note**
>
> 최신 Athena JDBC 드라이버를 다운로드하려면 Athena와 함께 JDBC 드라이버 사용을 참조하세요.

다음 연결 문자열은 기본 데이터 원본 *`hms-catalog-name`*을 지정합니다.

```
    jdbc:awsathena://AwsRegion=us-east-1;S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/
lambda/results/;Workgroup=AmazonAthenaPreviewFunctionality;Catalog=hms-catalog-name;
```

다음 이미지는 SQL Workbench에 구성된 샘플 JDBC 연결 URL을 보여줍니다.



Hive 뷰 작업

Athena를 사용하여 외부 Apache Hive 메타스토어의 기존 뷰를 쿼리할 수 있습니다. Athena는 원본 뷰를 변경하거나 변환을 저장하지 않고 런타임에서 즉석으로 뷰를 변환합니다.

예를 들어 다음과 같이 Athena에서 지원되지 않는 구문(`LATERAL VIEW explode()` 등)을 사용하는 Hive 뷰가 있다고 가정합니다.

```
CREATE VIEW team_view AS
SELECT team, score
FROM matches
LATERAL VIEW explode(scores) m AS score
```

Athena는 Hive 뷰 쿼리 문자열을 다음과 같이 Athena가 실행할 수 있는 문으로 변환합니다.

```
SELECT team, score
FROM matches
CROSS JOIN UNNEST(scores) AS m (score)
```

외부 Hive 메타스토어를 Athena에 연결하는 방법에 대한 자세한 내용은 외부 Hive 메타스토어용 Athena 데이터 커넥터 사용 단원을 참조하세요.

고려 사항 및 제한

Athena에서 Hive 뷰를 쿼리할 때 다음 사항을 고려해 보세요.

- Athena는 Hive 뷰 생성을 지원하지 않습니다. 외부 Hive 메타스토어에 Hive 뷰를 만든 다음 Athena에서 쿼리할 수 있습니다.
- Athena는 Hive 뷰에 대한 사용자 지정 UDF를 지원하지 않습니다.
- Athena 콘솔의 알려진 문제로 인해 Hive 뷰는 뷰 목록 대신 테이블 목록 아래에 표시됩니다.
- 변환 과정은 자동이지만 특정 Hive 함수는 Hive 뷰에 대해 지원되지 않거나 특별한 처리가 필요합니다. 자세한 내용은 다음 섹션을 참조하세요.

Hive 함수 지원 제한

이 섹션에서는 Athena가 Hive 뷰에 대해 지원하지 않거나 특별한 처리가 필요한 Hive 함수에 대해 설명합니다. 현재 Athena는 주로 Hive 2.2.0의 함수를 지원하므로 상위 버전(예: Hive 4.0.0)에서만 사용할 수 있는 함수는 사용할 수 없습니다. Hive 함수의 전체 목록은 Hive 언어 설명서 UDF를 참조하십시오.

집계 함수

특별한 처리가 필요한 집계 함수

Hive 뷰에 대한 다음 집계 함수는 특별한 처리가 필요합니다.

- Avg - `avg(INT i)` 대신 `avg(CAST(i AS DOUBLE))`을 사용하세요.

지원되지 않는 집계 함수

다음 Hive 집계 함수는 Athena에서 Hive 뷰에 대해 지원되지 않습니다.

```
covar_pop
```

```
histogram_numeric
ntile
percentile
percentile_approx
```

`regr_count`, `regr_r2`, `regr_sxx`와 같은 회귀 함수는 Athena에서 Hive 뷰에 대해 지원되지 않습니다.

지원되지 않는 날짜 함수

다음 Hive 날짜 함수는 Athena에서 Hive 뷰에 대해 지원되지 않습니다.

```
date_format(date/timestamp/string ts, string fmt)
day(string date)
dayofmonth(date)
extract(field FROM source)
hour(string date)
minute(string date)
month(string date)
quarter(date/timestamp/string)
second(string date)
weekofyear(string date)
year(string date)
```

지원되지 않는 마스킹 함수

`mask()`, `mask_first_n()`와 같은 Hive 마스킹 함수는 Athena에서 Hive 뷰에 대해 지원되지 않습니다.

기타 함수

특별한 처리가 필요한 기타 함수

Hive 뷰에 대한 다음 기타 함수는 특별한 처리가 필요합니다.

- md5 - Athena는 `md5(binary)`를 지원하지만 `md5(varchar)`는 지원하지 않습니다.
- Explode - Athena는 다음 구문으로 사용되는 경우 `explode`를 지원합니다.

  ```
  LATERAL VIEW [OUTER] EXPLODE(<argument>)
  ```

- Posexplode - Athena는 다음 구문으로 사용되는 경우 `posexplode`를 지원합니다.

```
LATERAL VIEW [OUTER] POSEXPLODE(<argument>)
```

(pos, val) 출력에서 Athena는 pos 열을 BIGINT로 취급합니다. 이 때문에 오래된 뷰를 피하기 위해 pos 열을 BIGINT로 캐스팅해야 할 수 있습니다. 다음 예는 이 기법을 보여 줍니다.

```
SELECT CAST(c AS BIGINT) AS c_bigint, d
FROM table LATERAL VIEW POSEXPLODE(<argument>) t AS c, d
```

지원되지 않는 기타 함수

다음 Hive 함수는 Athena에서 Hive 뷰에 대해 지원되지 않습니다.

```
aes_decrypt
aes_encrypt
current_database
current_user
inline
java_method
logged_in_user
reflect
sha/sha1/sha2
stack
version
```

연산자

특별한 처리가 필요한 연산자

Hive 뷰에 대한 다음 연산자는 특별한 처리가 필요합니다.

- 모드(Mod) 연산자(%) - DOUBLE 형식이 암시적으로 DECIMAL(x,y)로 캐스팅되기 때문에 다음 구문은 View is stale 오류 메시지를 반환할 수 있습니다.

  ```
  a_double % 1.0 AS column
  ```

  이 문제를 해결하려면 다음 예제와 같이 CAST를 사용하세요.

  ```
  CAST(a_double % 1.0 as DOUBLE) AS column
  ```

- 나누기 연산자(/) - Hive에서 `int`를 `int`로 나누면 `double`을 생성합니다. Athena에서는 동일한 연산이 잘린 `int`을 생성합니다.

지원되지 않는 연산자

Athena는 다음 연산자를 Hive 뷰에 대해 지원하지 않습니다.

~A – 비트 NOT

A ^ b – 비트 XOR

A & b – 비트 AND

A | b – 비트 OR

A <=> b – null이 아닌 피연산자에 대해 같음(=) 연산자와 동일한 결과를 반환합니다. 둘 다 NULL인 경우 TRUE를, 둘 중 하나가 NULL인 경우 FALSE를 반환합니다.

문자열 함수

특별한 처리가 필요한 문자열 함수

Hive 뷰에 대한 다음 Hive 문자열 함수는 특별한 처리가 필요합니다.

- chr(bigint|double a) – Hive는 음수 인수를 허용하지만 Athena는 그렇지 않습니다.
- instr(string str, string substr) – `instr` 함수에 대한 Athena 매핑이 `INT` 대신 `BIGINT`를 반환하므로 다음 구문을 사용하세요.

  ```
  CAST(instr(string str, string substr) as INT)
  ```

  이 단계가 없으면 뷰가 오래된 것으로 간주됩니다.
- length(string a) – `length` 함수에 대한 Athena 매핑이 `INT` 대신 `BIGINT`를 반환하므로 뷰가 오래된 것으로 간주되지 않도록 다음 구문을 사용하세요.

  ```
  CAST(length(string str) as INT)
  ```

지원되지 않는 문자열 함수

다음 Hive 문자열 함수는 Athena에서 Hive 뷰에 대해 지원되지 않습니다.

```
ascii(string str)
character_length(string str)
decode(binary bin, string charset)
encode(string src, string charset)
elt(N int,str1 string,str2 string,str3 string,...)
field(val T,val1 T,val2 T,val3 T,...)
find_in_set(string str, string strList)
initcap(string A)
levenshtein(string A, string B)
locate(string substr, string str[, int pos])
octet_length(string str)
parse_url(string urlString, string partToExtract [, string keyToExtract])
printf(String format, Obj... args)
quote(String text)
regexp_extract(string subject, string pattern, int index)
repeat(string str, int n)
sentences(string str, string lang, string locale)
soundex(string A)
space(int n)
str_to_map(text[, delimiter1, delimiter2])
substring_index(string A, string delim, int count)
```

지원되지 않는 XPath 함수

`xpath`, `xpath_short`, `xpath_int`와 같은 Hive XPath 함수는 Athena에서 Hive 뷰에 대해 지원되지 않습니다.

문제 해결

Athena에서 Hive 뷰를 사용할 때 다음과 같은 문제가 발생할 수 있습니다.

- View ***<view name>*** is stale – 이 메시지는 일반적으로 Hive와 Athena 뷰 간의 형식 불일치를 나타냅니다. Hive LanguageManual UDF와 Presto functions and operators(Presto 함수 및 연산자) 설명서에서 동일한 함수가 서로 다른 특징을 갖는 경우 일치하지 않는 데이터 형식을 캐스팅해 보세요.
- Function not registered – Athena가 현재 함수를 지원하지 않습니다. 자세한 내용은 이 문서 앞부분의 정보를 참조하십시오.

Hive 메타스토어와 함께 AWS CLI 사용

aws athena CLI 명령을 사용하여 Athena에서 사용하는 Hive 메타스토어 데이터 카탈로그를 관리할 수 있습니다. Athena에서 사용할 카탈로그를 하나 이상 정의한 후에는 aws athena DDL 및 DML 명령에서 해당 카탈로그를 참조할 수 있습니다.

AWS CLI를 사용하여 Hive 메타스토어 카탈로그 관리

카탈로그 등록: Create-data-catalog

데이터 카탈로그를 등록하려면 create-data-catalog 명령을 사용합니다. name 파라미터를 사용하여 카탈로그에 사용할 이름을 지정합니다. Lambda 함수의 ARN을 parameters 인수의 metadata-function 옵션에 전달합니다. 새 카탈로그에 대한 태그를 만들려면 하나 이상의 공백으로 구분된 Key=*key*,Value=*value* 인수 페어와 함께 tags 파라미터를 사용합니다.

다음 예제에서는 hms-catalog-1이라는 Hive 메타스토어 카탈로그를 등록합니다. 명령은 가독성을 위해 형식이 지정되었습니다.

```
$ aws athena create-data-catalog
 --name "hms-catalog-1"
 --type "HIVE"
 --description "Hive Catalog 1"
 --parameters "metadata-function=arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-v3,sdk-version=1.0"
 --tags Key=MyKey,Value=MyValue
 --region us-east-1
```

카탈로그 세부 정보 표시: Get-data-catalog

카탈로그의 세부 정보를 표시하려면 다음 예제와 같이 카탈로그 이름을 get-data-catalog 명령에 전달합니다.

```
$ aws athena get-data-catalog --name "hms-catalog-1" --region us-east-1
```

다음 샘플 결과는 JSON 형식입니다.

```
{
    "DataCatalog": {
        "Name": "hms-catalog-1",
        "Description": "Hive Catalog 1",
        "Type": "HIVE",
        "Parameters": {
```

```
            "metadata-function": "arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-v3",
            "sdk-version": "1.0"
        }
    }
}
```

등록된 카탈로그 나열: List-data-catalogs

등록된 카탈로그를 나열하려면 다음 예제와 같이 `list-data-catalogs` 명령을 사용하고 필요에 따라 리전을 지정합니다. 나열된 카탈로그에는 항상 AWS Glue가 포함되어 있습니다.

```
$ aws athena list-data-catalogs --region us-east-1
```

다음 샘플 결과는 JSON 형식입니다.

```
{
    "DataCatalogs": [
        {
            "CatalogName": "AwsDataCatalog",
            "Type": "GLUE"
        },
        {
            "CatalogName": "hms-catalog-1",
            "Type": "HIVE",
            "Parameters": {
                "metadata-function": "arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-v3",
                "sdk-version": "1.0"
            }
        }
    ]
}
```

카탈로그 업데이트: Update-data-catalog

데이터 카탈로그를 업데이트하려면 다음 예제와 같이 `update-data-catalog` 명령을 사용합니다. 명령은 가독성을 위해 형식이 지정되었습니다.

```
$ aws athena update-data-catalog
 --name "hms-catalog-1"
 --type "HIVE"
```

```
 --description "My New Hive Catalog Description"
 --parameters "metadata-function=arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-new,sdk-version=1.0"
 --region us-east-1
```

카탈로그 삭제: Delete-data-catalog

데이터 카탈로그를 삭제하려면 다음 예제와 같이 `delete-data-catalog` 명령을 사용합니다.

```
$ aws athena delete-data-catalog --name "hms-catalog-1" --region us-east-1
```

데이터베이스 세부 정보 표시: Get-database

데이터베이스의 세부 정보를 표시하려면 다음 예제와 같이 카탈로그 및 데이터베이스의 이름을 `get-database` 명령에 전달합니다.

```
$ aws athena get-database --catalog-name hms-catalog-1 --database-name mydb
```

다음 샘플 결과는 JSON 형식입니다.

```
{
    "Database": {
        "Name": "mydb",
        "Description": "My database",
        "Parameters": {
            "CreatedBy": "Athena",
            "EXTERNAL": "TRUE"
        }
    }
}
```

카탈로그의 데이터베이스 나열: List-databases

카탈로그의 데이터베이스를 나열하려면 다음 예제와 같이 `list-databases` 명령을 사용하고 필요에 따라 리전을 지정합니다.

```
$ aws athena list-databases --catalog-name AwsDataCatalog --region us-west-2
```

다음 샘플 결과는 JSON 형식입니다.

```
{
    "DatabaseList": [
```

```
        {
            "Name": "default"
        },
        {
            "Name": "mycrawlerdatabase"
        },
        {
            "Name": "mydatabase"
        },
        {
            "Name": "sampledb",
            "Description": "Sample database",
            "Parameters": {
                "CreatedBy": "Athena",
                "EXTERNAL": "TRUE"
            }
        },
        {
            "Name": "tpch100"
        }
    ]
}
```

테이블 세부 정보 표시: Get-table-metadata

열 이름 및 데이터 형식을 포함하여 테이블의 메타데이터를 표시하려면 다음 예제와 같이 카탈로그 이름, 데이터베이스 이름 및 테이블 이름을 `get-table-metadata` 명령에 전달합니다.

```
$ aws athena get-table-metadata --catalog-name AwsDataCatalog --database-name mydb --
table-name cityuseragent
```

다음 샘플 결과는 JSON 형식입니다.

```
{
    "TableMetadata": {
        "Name": "cityuseragent",
            "CreateTime": 1586451276.0,
            "LastAccessTime": 0.0,
            "TableType": "EXTERNAL_TABLE",
            "Columns": [
                {
                    "Name": "city",
                    "Type": "string"
```

```
                },
                {
                    "Name": "useragent1",
                    "Type": "string"
                }
            ],
            "PartitionKeys": [],
            "Parameters": {
                "COLUMN_STATS_ACCURATE": "false",
                "EXTERNAL": "TRUE",
                "inputformat": "org.apache.hadoop.mapred.TextInputFormat",
                "last_modified_by": "hadoop",
                "last_modified_time": "1586454879",
                "location": "s3://DOC-EXAMPLE-BUCKET/",
                "numFiles": "1",
                "numRows": "-1",
                "outputformat":
 "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
                "rawDataSize": "-1",
                "serde.param.serialization.format": "1",
                "serde.serialization.lib":
 "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
                "totalSize": "61"
            }
        }
}
```

데이터베이스의 모든 테이블에 대한 메타데이터 표시: List-table-metadata

데이터베이스의 모든 테이블에 대한 메타데이터를 표시하려면 카탈로그 이름 및 데이터베이스 이름을 `list-table-metadata` 명령에 전달합니다. `list-table-metadata` 명령은 테이블 이름을 지정하지 않는다는 점을 제외하면 `get-table-metadata` 명령과 유사합니다. 결과 수를 제한하려면 다음 예제와 같이 `--max-results` 옵션을 사용할 수 있습니다.

```
$ aws athena list-table-metadata --catalog-name AwsDataCatalog --database-name sampledb
 --region us-east-1 --max-results 2
```

다음 샘플 결과는 JSON 형식입니다.

```
{
    "TableMetadataList": [
        {
```

```
            "Name": "cityuseragent",
            "CreateTime": 1586451276.0,
            "LastAccessTime": 0.0,
            "TableType": "EXTERNAL_TABLE",
            "Columns": [
                {
                    "Name": "city",
                    "Type": "string"
                },
                {
                    "Name": "useragent1",
                    "Type": "string"
                }
            ],
            "PartitionKeys": [],
            "Parameters": {
                "COLUMN_STATS_ACCURATE": "false",
                "EXTERNAL": "TRUE",
                "inputformat": "org.apache.hadoop.mapred.TextInputFormat",
                "last_modified_by": "hadoop",
                "last_modified_time": "1586454879",
                "location": "s3://DOC-EXAMPLE-BUCKET/",
                "numFiles": "1",
                "numRows": "-1",
                "outputformat":
"org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
                "rawDataSize": "-1",
                "serde.param.serialization.format": "1",
                "serde.serialization.lib":
"org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
                "totalSize": "61"
            }
        },
        {
            "Name": "clearinghouse_data",
            "CreateTime": 1589255544.0,
            "LastAccessTime": 0.0,
            "TableType": "EXTERNAL_TABLE",
            "Columns": [
                {
                    "Name": "location",
                    "Type": "string"
                },
                {
```

```
                    "Name": "stock_count",
                    "Type": "int"
                },
                {
                    "Name": "quantity_shipped",
                    "Type": "int"
                }
            ],
            "PartitionKeys": [],
            "Parameters": {
                "EXTERNAL": "TRUE",
                "inputformat": "org.apache.hadoop.mapred.TextInputFormat",
                "location": "s3://DOC-EXAMPLE-BUCKET/",
                "outputformat":
 "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
                "serde.param.serialization.format": "1",
                "serde.serialization.lib":
 "org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe",
                "transient_lastDdlTime": "1589255544"
            }
        }
    ],
    "NextToken":
 "eyJsYXN0RXZhbHVhdGVkS2V5Ijp7IkhBU0hfS0VZIjp7InMiOiJ0Ljk0YWZjYjk1MjJjNTQ1YmU4Y2I5OWE5NTg0MjFjY
}
```

DDL 및 DML 문 실행

AWS CLI를 사용하여 DDL 및 DML 문을 실행할 때 다음 두 가지 방법 중 하나로 Hive 메타스토어 카탈로그의 이름을 전달할 수 있습니다.

- 이를 지원하는 문으로 직접 전달
- `--query-execution-context Catalog` 파라미터를 사용하여 전달

DDL 문

다음 예제에서는 `show create table` DDL 문의 일부로 카탈로그 이름을 직접 전달합니다. 명령은 가독성을 위해 형식이 지정되었습니다.

```
$ aws athena start-query-execution
 --query-string "show create table hms-catalog-1.hms_tpch_partitioned.lineitem"
```

```
 --result-configuration "OutputLocation=s3://DOC-EXAMPLE-BUCKET/lambda/results"
```

다음 예제 DDL `show create table` 문은 `--query-execution-context`의 `Catalog` 파라미터를 사용하여 Hive 메타스토어 카탈로그 이름 `hms-catalog-1`을 전달합니다. 명령은 가독성을 위해 형식이 지정되었습니다.

```
$ aws athena start-query-execution
 --query-string "show create table lineitem"
 --query-execution-context "Catalog=hms-catalog-1,Database=hms_tpch_partitioned"
 --result-configuration "OutputLocation=s3://DOC-EXAMPLE-BUCKET/lambda/results"
```

DML 문

다음 예제 DML `select` 문은 카탈로그 이름을 쿼리에 직접 전달합니다. 명령은 가독성을 위해 형식이 지정되었습니다.

```
$ aws athena start-query-execution
 --query-string "select * from hms-catalog-1.hms_tpch_partitioned.customer limit 100"
 --result-configuration "OutputLocation=s3://DOC-EXAMPLE-BUCKET/lambda/results"
```

다음 예제 DML `select` 문은 `--query-execution-context`의 `Catalog` 파라미터를 사용하여 Hive 메타스토어 카탈로그 이름 `hms-catalog-1`을 전달합니다. 명령은 가독성을 위해 형식이 지정되었습니다.

```
$ aws athena start-query-execution
 --query-string "select * from customer limit 100"
 --query-execution-context "Catalog=hms-catalog-1,Database=hms_tpch_partitioned"
 --result-configuration "OutputLocation=s3://DOC-EXAMPLE-BUCKET/lambda/results"
```

참조 구현

Athena는 https://github.com/awslabs/aws-athena-hive-metastore에서 GitHub.com의 외부 Hive 메타스토어에 대한 커넥터의 참조 구현을 제공합니다.

참조 구현은 다음과 같은 모듈이 있는 Apache Maven 프로젝트입니다.

- **`hms-service-api`** – Lambda 함수와 Athena 서비스 클라이언트 간의 API 작업을 포함합니다. 이러한 API 작업은 `HiveMetaStoreService` 인터페이스에 정의되어 있습니다. 이는 서비스 계약이므로 이 모듈에서는 아무 것도 변경하지 마세요.

- **hms-lambda-handler** – 모든 Hive 메타스토어 API 호출을 처리하는 기본 Lambda 핸들러 집합입니다. 클래스 `MetadataHandler`는 모든 API 호출의 디스패처입니다. 이 패키지는 변경할 필요가 없습니다.
- **hms-lambda-func** – 다음 구성 요소가 있는 예제 Lambda 함수입니다.
  - **HiveMetaStoreLambdaFunc** – `MetadataHandler`를 확장하는 예제 Lambda 함수입니다.
  - **ThriftHiveMetaStoreClient** – Hive 메타스토어와 통신하는 Thrift 클라이언트입니다. 이 클라이언트는 Hive 2.3.0용으로 작성되었습니다. 다른 Hive 버전을 사용하는 경우, 응답 객체가 호환되도록 이 클래스를 업데이트해야 할 수 있습니다.
  - **ThriftHiveMetaStoreClientFactory** – Lambda 함수의 동작을 제어합니다. 예를 들어 `getHandlerProvider()` 메서드를 재정의하여 자체 핸들러 공급자 집합을 제공할 수 있습니다.
  - `hms.properties` – Lambda 함수를 구성합니다. 대부분의 경우 다음 두 속성만 업데이트가 필요합니다.
    - `hive.metastore.uris` - `thrift://<host_name>:9083` 형식의 Hive 메타스토어의 URI입니다.
    - `hive.metastore.response.spill.location`: 응답 객체의 크기가 지정된 임계값(예: 4MB)을 초과하는 경우 응답 객체를 저장할 Amazon S3 위치입니다. 임계값은 `hive.metastore.response.spill.threshold` 속성에 정의됩니다. 기본값을 변경하지 않는 것이 좋습니다.

> **Note**
>
> 이 두 속성은 Lambda 환경 변수 `HMS_URIS` 및 `SPILL_LOCATION`에 의해 재정의될 수 있습니다. 다른 Hive 메타스토어 또는 유출 위치에서 Lambda 함수를 사용하려는 경우 이 함수의 소스 코드를 다시 컴파일하는 대신 이러한 변수를 사용합니다.

- **hms-lambda-layer** – `hms-service-api`, `hms-lambda-handler` 및 해당 종속 항목을 `.zip` 파일에 넣는 Maven 어셈블리 프로젝트입니다. `.zip` 파일은 여러 Lambda 함수에서 사용할 수 있도록 Lambda 계층으로 등록됩니다.
- **hms-lambda-rnp** - Lambda 함수의 응답을 기록한 다음 이를 사용하여 응답을 재생합니다. 이 모델을 사용하여 테스트용 Lambda 응답을 시뮬레이션할 수 있습니다.

아티팩트 직접 구축

대부분의 사용 사례는 참조 구현을 수정할 필요가 없습니다. 그러나 필요한 경우 소스 코드를 수정하고 직접 아티팩트를 빌드하고 Amazon S3 위치에 아티팩트를 업로드할 수 있습니다.

아티팩트를 빌드하기 전에 hms-lambda-func 모듈의 hms.properties 파일에서 hive.metastore.uris 및 hive.metastore.response.spill.location 속성을 업데이트합니다.

아티팩트를 빌드하려면 Apache Maven이 설치되어 있어야 하며 mvn install 명령을 실행해야 합니다. 그러면 hms-lambda-layer 모듈에서 target이라는 출력 폴더에 계층 .zip 파일이 생성되고 hms-lambd-func 모듈에서 Lambda 함수 .jar 파일이 생성됩니다.

Amazon Athena 연합 쿼리 사용

Amazon S3 외의 원본에 데이터가 있는 경우 Athena 연합 쿼리를 사용하여 데이터를 제자리에 쿼리하거나 여러 데이터 원본에서 데이터를 추출하여 Amazon S3에 저장하는 파이프라인을 구축할 수 있습니다. Athena 연합 쿼리를 사용하면 관계형, 비관계형, 객체 및 사용자 지정 데이터 원본에 저장된 데이터에 대해 SQL 쿼리를 실행할 수 있습니다.

Athena는 연합 쿼리를 실행하기 위해 AWS Lambda에서 실행되는 데이터 원본 커넥터를 사용합니다. 데이터 원본 커넥터는 대상 데이터 원본과 Athena 간에 변환할 수 있는 코드입니다. 커넥터는 Athena의 쿼리 엔진 확장으로 생각할 수 있습니다. 사전 구축된 Athena 데이터 원본 커넥터는 Amazon CloudWatch Logs, Amazon DynamoDB, Amazon DocumentDB, Amazon RDS, JDBC 호환 관계형 데이터베이스(Apache 2.0 라이선스에 따른 MySQL, PostgreSQL 등)와 같은 데이터 원본을 위한 것입니다. Athena Query Federation SDK를 사용하여 사용자 지정 커넥터를 작성할 수도 있습니다. 데이터 원본 커넥터를 선택 및 구성하여 계정에 배포하려면 Athena 및 Lambda 콘솔 또는 AWS Serverless Application Repository을 사용할 수 있습니다. 데이터 원본 커넥터를 배포한 후 커넥터는 SQL 쿼리에서 지정할 수 있는 카탈로그와 연결됩니다. 여러 카탈로그의 SQL 문을 결합하고 단일 쿼리로 여러 데이터 원본에 적용할 수 있습니다.

데이터 원본에 대해 쿼리가 제출되면, Athena는 해당 커넥터를 호출하여 읽어야 하는 테이블의 일부를 식별하고, 병렬화를 관리하며, 필터 조건자를 푸시다운합니다. 쿼리를 제출하는 사용자를 기반으로 커넥터는 특정 데이터 요소에 대한 액세스를 제공하거나 제한할 수 있습니다. 커넥터는 쿼리에서 요청한 데이터를 반환하기 위한 형식으로 Apache Arrow를 사용합니다. 이렇게 하면 커넥터가 C, C++, Java, Python, Rust 같은 언어로 구현될 수 있습니다. 커넥터는 Lambda에서 처리되므로 Lambda에서 액세스할 수 있는 클라우드 또는 온프레미스의 모든 데이터 원본에서 데이터에 액세스하는 데 사용할 수 있습니다.

사용자 고유의 데이터 원본 커넥터를 작성하려면 Athena Query Federation SDK를 사용하여 Amazon Athena가 제공 및 유지 관리하는 사전 구축된 커넥터 중 하나를 사용자 지정할 수 있습니다. GitHub 리포지토리에서 소스 코드의 복사본을 수정한 다음 Connector publish tool(커넥터 게시 도구)를 사용하여 자체 AWS Serverless Application Repository 패키지를 만들 수 있습니다.

> **Note**
>
> 타사 개발자가 Athena Query Federation SDK를 사용하여 데이터 원본 커넥터를 작성한 경우도 있을 수 있습니다. 이러한 데이터 원본 커넥터와 관련한 지원 또는 라이선스 문제는 커넥터 공급업체에 문의하세요. 이러한 커넥터는 AWS에서 테스트 또는 지원되지 않습니다.

Athena에서 작성 및 테스트한 데이터 원본 커넥터 목록은 사용 가능한 데이터 소스 커넥터 섹션을 참조하세요.

사용자 고유의 데이터 원본 커넥터 작성에 대한 자세한 내용은 GitHub의 Athena 커넥터 예제를 참조하세요.

고려 사항 및 제한

- 엔진 버전 - Athena 연합 쿼리는 Athena 엔진 버전 2 이상 버전에서만 지원됩니다. Athena 엔진 버전에 대한 자세한 내용은 Athena 엔진 버전 관리 단원을 참조하세요.
- 보기 - 페더레이션된 데이터 소스에서 보기를 생성하고 쿼리할 수 있습니다. 페더레이션된 보기는 기본 데이터 소스가 아닌 AWS Glue에 저장됩니다. 자세한 내용은 페더레이션된 보기 쿼리 단원을 참조하십시오.
- 쓰기 작업 - INSERT INTO 같은 쓰기 작업은 지원되지 않습니다. 이를 실행하려고 하면 이 작업은 현재 외부 카탈로그에서 지원되지 않습니다(This operation is currently not supported for external catalogs)라는 오류 메시지가 표시될 수 있습니다.
- 요금 – 요금 정보는 Amazon Athena 요금을 참조하세요.

  JDBC 드라이버 - JDBC 드라이버와 함께 연합 쿼리나 외부 Hive 메타스토어를 사용하려면 JDBC 연결 문자열에 `MetadataRetrievalMethod=ProxyAPI`를 포함해야 합니다. JDBC 드라이버에 대한 자세한 내용은 JDBC로 Amazon Athena에 연결 섹션을 참조하세요.
- Secrets Manager – AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager에 대해 Amazon VPC 프라이빗 엔드포인트를 구성해야 합니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 Secrets Manager VPC 프라이빗 엔드포인트 생성을 참조하세요.

데이터 원본 커넥터가 올바르게 작동하려면 다음 리소스에 액세스해야 할 수 있습니다. 사전 구축된 커넥터를 사용하는 경우 커넥터의 정보를 확인하여 VPC를 올바르게 구성했는지 확인합니다. 또한 쿼리를 실행하고 커넥터를 만드는 IAM 보안 주체에게 필요한 작업에 대한 권한이 있는지 확인합니다. 자세한 내용은 Athena 연합 쿼리를 허용하는 IAM 권한 정책의 예제 단원을 참조하세요.

- Amazon S3 – 쿼리 결과를 Amazon S3의 Athena 쿼리 결과 위치에 작성하는 것 외에도 데이터 커넥터는 Amazon S3의 유출 버킷에 작성합니다. 이 Amazon S3 위치에 대한 연결 및 권한이 필요합니다.
- Athena – 데이터 원본은 쿼리 상태를 확인하고 오버스캔을 방지하기 위해 Athena와 양방향으로 연결해야 합니다.
- AWS Glue Data Catalog – 커넥터가 보충 또는 기본 메타데이터에 데이터 카탈로그를 사용하는 경우 연결 및 권한이 필요합니다.

비디오

Athena 연합 쿼리 사용에 대해 자세히 알아보려면 다음 동영상을 시청하세요.

동영상: Amazon QuickSight에서 Amazon Athena의 연합 쿼리 결과 분석

다음 동영상은 Amazon QuickSight에서 Athena 연합 쿼리의 결과를 분석하는 방법을 보여 줍니다.

Amazon QuickSight에서 Amazon Athena의 연합 쿼리 결과 분석

동영상: 게임 분석 파이프라인

다음 동영상은 Amazon Athena 연합 쿼리를 사용하여 게임 및 서비스의 원격 측정 데이터를 수집, 저장, 분석하기 위해 확장형 서버리스 데이터 파이프라인을 배포하는 방법을 보여줍니다.

게임 분석 파이프라인

사용 가능한 데이터 소스 커넥터

이 섹션에는 Athena 외부의 다양한 데이터 원본을 쿼리하는 데 사용할 수 있는 사전 구축된 Amazon S3 데이터 원본 커넥터가 나열되어 있습니다. Athena 쿼리에서 커넥터를 사용하려면 커넥터를 구성하여 계정에 배포합니다.

고려 사항 및 제한

- 일부 사전 구축된 커넥터는 커넥터를 사용하기 전에 VPC 및 보안 그룹 생성을 요구합니다. VPC 생성에 대한 자세한 내용은 데이터 원본 커넥터용 VPC 생성을 참조하세요.
- AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager에 대해 Amazon VPC 프라이빗 엔드포인트를 구성해야 합니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 Secrets Manager VPC 프라이빗 엔드포인트 생성을 참조하세요.

- 조건자 푸시다운을 지원하지 않는 커넥터의 경우 조건자를 포함하는 쿼리는 실행하는 데 시간이 오래 걸립니다. 작은 데이터 세트의 경우 매우 적은 양의 데이터가 스캔되고 쿼리에는 평균 약 2분 정도 걸립니다. 그러나 대규모 데이터 세트의 경우 많은 쿼리가 시간 초과될 수 있습니다.
- 일부 페더레이션된 데이터 소스는 Athena와는 다른 용어를 사용하여 데이터 객체를 참조합니다. 자세한 내용은 Athena 및 페더레이션된 테이블 이름 한정자 단원을 참조하십시오.
- 테이블을 나열할 때 페이지 매김을 지원하지 않는 커넥터의 경우 데이터베이스에 테이블과 메타데이터가 많으면 웹 서비스 시간이 초과될 수 있습니다. 다음 커넥터는 테이블 나열을 위한 페이지 매김 지원을 제공합니다.
  - documentdb
  - DynamoDB
  - MySQL
  - OpenSearch
  - Oracle
  - PostgreSQL
  - Redshift
  - SQL Server

추가 정보

- Athena 데이터 원본 커넥터 배포에 대한 자세한 내용은 데이터 소스 커넥터 배포 단원을 참조하세요.
- Athena 데이터 소스 커넥터를 사용하는 쿼리에 대한 자세한 내용은 페더레이션된 쿼리 실행 섹션을 참조하세요.
- Athena 데이터 원본 커넥터에 대한 자세한 내용은 GitHub의 사용 가능한 커넥터를 참조하세요.

Athena 데이터 원본 커넥터

- Amazon Athena Azure Data Lake Storage(ADLS) Gen2 커넥터
- Amazon Athena Azure Synapse 커넥터
- Amazon Athena Cloudera Hive 커넥터
- Amazon Athena Cloudera Impala 커넥터
- Amazon Athena CloudWatch 커넥터
- Amazon Athena CloudWatch 지표 커넥터

- Amazon Athena AWS CMDB 커넥터
- Amazon Athena IBM Db2 커넥터
- Amazon Athena IBM Db2 AS/400(Db2 iSeries) 커넥터
- Amazon Athena DocumentDB 커넥터
- Amazon Athena DynamoDB 커넥터
- Amazon Athena Google BigQuery 커넥터
- Amazon Athena Google Cloud Storage 커넥터
- Amazon Athena HBase 커넥터
- Amazon Athena Hortonworks 커넥터
- Amazon Athena Apache Kafka 커넥터
- Amazon Athena MSK 커넥터
- Amazon Athena MySQL 커넥터
- Amazon Athena Neptune 커넥터
- Amazon Athena OpenSearch 커넥터
- Amazon Athena Oracle 커넥터
- Amazon Athena PostgreSQL 커넥터
- Amazon Athena Redis 커넥터
- Amazon Athena Redshift 커넥터
- Amazon Athena SAP HANA 커넥터
- Amazon Athena Snowflake 커넥터
- Amazon Athena Microsoft SQL Server 커넥터
- Amazon Athena Teradata 커넥터
- Amazon Athena Timestream 커넥터
- Amazon Athena TPC Benchmark DS(TPC-DS) 커넥터
- Amazon Athena Vertica 커넥터

> **Note**
>
> AthenaJdbcConnector(최신 버전 2022.4.1)는 더 이상 사용되지 않습니다. 대신 MySQL, Redshift 또는 PostgreSQL용 커넥터와 같은 데이터베이스 관련 커넥터를 사용하세요.

Amazon Athena Azure Data Lake Storage(ADLS) Gen2 커넥터

Azure Data Lake Storage(ADLS) Gen2용 Amazon Athena 커넥터를 사용하면 Amazon Athena가 ADLS에 저장된 데이터에 대해 SQL 쿼리를 실행할 수 있습니다. Athena는 데이터 레이크에 저장된 파일에 직접 액세스할 수 없습니다.

- 워크플로 - 커넥터는 `com.microsoft.sqlserver.jdbc.SQLServerDriver` 드라이버를 사용하는 JDBC 인터페이스를 구현합니다. 커넥터는 쿼리를 Azure Synapse 엔진으로 전달하고, 이 엔진에서 데이터 레이크에 액세스합니다.
- 데이터 처리 및 S3 - 일반적으로 Lambda 커넥터는 Amazon S3로 전송하지 않고 직접 데이터를 쿼리합니다. 그러나 Lambda 함수에서 반환된 데이터가 Lambda 한도를 초과하면 Athena가 초과분을 읽을 수 있도록 사용자가 지정한 Amazon S3 유출 버킷에 데이터가 기록됩니다.
- AAD 인증 - AAD는 Azure Synapse 커넥터의 인증 방법으로 사용할 수 있습니다. AAD를 사용하려면 커넥터가 사용하는 JDBC 연결 문자열에 URL 파라미터 `authentication=ActiveDirectoryServicePrincipal`, `AADSecurePrincipalId`, `AADSecurePrincipalSecret`을 포함해야 합니다. 이러한 파라미터는 직접 또는 Secrets Manager에 의해 전달될 수 있습니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 멀티플렉서 설정에서 유출 버킷과 접두사는 모든 데이터베이스 인스턴스에서 공유됩니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.
- 필터 조건의 날짜 및 타임스탬프 데이터 형식을 적절한 데이터 형식으로 캐스팅해야 합니다.

용어

다음은 Azure Data Lake Storage Gen2 커넥터와 관련된 용어입니다.

- 데이터베이스 인스턴스 - 온프레미스, Amazon EC2 또는 Amazon RDS에 배포된 데이터베이스의 모든 인스턴스.
- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.
- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - `connection_string` 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.
- 멀티플렉싱 핸들러 - 여러 데이터베이스 연결을 수락하고 사용할 수 있는 Lambda 핸들러.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 Azure Data Lake Storage Gen2 커넥터를 구성합니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 데이터베이스 인스턴스에 연결합니다.

```
datalakegentwo://${jdbc_connection_string}
```

멀티플렉싱 핸들러 사용

멀티플렉서를 사용하여 단일 Lambda 함수로 여러 데이터베이스 인스턴스에 연결할 수 있습니다. 요청은 카탈로그 이름을 기준으로 라우팅됩니다. Lambda에서 다음 클래스를 사용합니다.

핸들러	Class
복합 핸들러	`DataLakeGen2MuxCompositeHandler`
메타데이터 핸들러	`DataLakeGen2MuxMetadataHandler`
레코드 핸들러	`DataLakeGen2MuxRecordHandler`

멀티플렉싱 핸들러 파라미터

파라미터	설명
$*catalog*_connection_string	필수 사항입니다. 데이터베이스 인스턴스 연결 문자열. Athena에서 사용되는 카탈로그의 이름을 환경 변수 앞에 붙입니다. 예를 들어, Athena에 등록된 카탈로그가 mydatalakegentwocatalog 인 경우 환경 변수 이름은 mydatalakegentwocatalog_connection_string 입니다.
default	필수 사항입니다. 기본 연결 문자열. 이 문자열은 카탈로그가 lambda:${ *AWS_LAMBDA_FUNCTION_NAME* }일 때 사용됩니다.

다음은 datalakegentwo1(기본값)과 datalakegentwo2라는 2개의 데이터베이스 인스턴스를 지원하는 DataLakeGen2 MUX Lambda 함수에 대한 예제 속성입니다.

속성	값
default	datalakegentwo://jdbc:sqlserver://adlsgentwo1. *hostname*:*port*;databaseName= *database_name* ;${*secret1_name* }
datalakegentwo_catalog1_connection_string	datalakegentwo://jdbc:sqlserver://adlsgentwo1. *hostname*:*port*;databaseName= *database_name* ;${*secret1_name* }
datalakegentwo_catalog2_connection_string	datalakegentwo://jdbc:sqlserver://adlsgentwo2. *hostname*:*port*;databaseName= *database_name* ;${*secret2_name* }

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

> **Important**
>
> 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 `username` 및 `password` 값으로 바꿉니다.

  Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

  ```
  {"username": "${username}", "password": "${password}"}
  ```

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 `${secret1_name}`입니다.

```
datalakegentwo://jdbc:sqlserver://hostname:port;databaseName=database_name;
${secret1_name}
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
datalakegentwo://
jdbc:sqlserver://
hostname:port;databaseName=database_name;user=user_name;password=password
```

단일 연결 핸들러 사용

다음과 같은 단일 연결 메타데이터 및 레코드 핸들러를 사용하여 단일 Azure Data Lake Storage Gen2 인스턴스에 연결할 수 있습니다.

핸들러 유형	Class
복합 핸들러	`DataLakeGen2CompositeHandler`
메타데이터 핸들러	`DataLakeGen2MetadataHandler`
레코드 핸들러	`DataLakeGen2RecordHandler`

단일 연결 핸들러 파라미터

파라미터	설명
`default`	필수 사항입니다. 기본 연결 문자열.

단일 연결 핸들러는 하나의 데이터베이스 인스턴스를 지원하며 `default` 연결 문자열 파라미터를 제공해야 합니다. 다른 연결 문자열은 모두 무시됩니다.

다음은 Lambda 함수에서 지원하는 단일 Azure Data Lake Storage Gen2 인스턴스에 대한 속성 예제입니다.

속성	값
`default`	`datalakegentwo://jdbc:sqlserver://` *`hostname`*`:`*`port`*`;databaseName=;${` *`secret_name`* `}`

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
`spill_bucket`	필수 사항입니다. 유출 버킷 이름.
`spill_prefix`	필수 사항입니다. 유출 버킷 키 접두사.

파라미터	설명
spill_put_request_headers	(선택 사항) 유출에 사용되는 Amazon S3 putObject 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: {"x-amz-server-side-encryption" : "AES256"}). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 ADLS Gen2와 Apache Arrow의 해당 데이터 형식이 나와 있습니다.

ADLS Gen2	화살표
bit	TINYINT
tinyint	SMALLINT
smallint	SMALLINT
int	INT
bigint	BIGINT
decimal	DECIMAL
numeric	FLOAT8
smallmoney	FLOAT8
money	DECIMAL
float[24]	FLOAT4
float[53]	FLOAT8
실제	FLOAT4
datetime	Date(MILLISECOND)

ADLS Gen2	화살표
datetime2	Date(MILLISECOND)
smalldatetime	Date(MILLISECOND)
date	Date(DAY)
시간	VARCHAR
datetimeoffset	Date(MILLISECOND)
char[n]	VARCHAR
varchar[n/max]	VARCHAR

파티션 및 분할

Azure Data Lake Storage Gen2는 데이터 파일 저장을 위해 Hadoop 호환 Gen2 blob 스토리지를 사용합니다. 이러한 파일의 데이터는 Azure Synapse 엔진에서 쿼리됩니다. Azure Synapse 엔진은 파일 시스템에 저장된 Gen2 데이터를 외부 테이블로 처리합니다. 파티션은 데이터 형식에 따라 구현됩니다. 데이터가 이미 Gen 2 스토리지 시스템 내에서 분할되고 배포된 경우 커넥터는 데이터를 단일 분할로 검색합니다.

성능

Azure Data Lake Storage Gen2 커넥터는 한 번에 여러 쿼리를 실행할 경우 쿼리 성능이 느려지고 제한이 적용될 수 있습니다.

쿼리에서 스캔하는 데이터를 줄이기 위해 Athena Azure Data Lake Storage Gen2 커넥터에서 조건자 푸시다운을 수행합니다. 스캔하는 데이터와 쿼리 실행 시간을 줄이도록 간단한 조건자와 복잡한 표현식을 커넥터로 푸시다운합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 `WHERE` 절에 사용되는 표현식입니다. Athena Azure Data Lake Storage Gen2 커넥터는 이러한 표현식을 결합하고 Azure Data Lake Storage Gen2로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena Azure Data Lake Storage Gen2 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND, OR, NOT
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, NULL_IF, IS_NULL
- 산술: ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- 기타: LIKE_PATTERN, IN

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%');
```

패스스루 쿼리

Azure Data Lake Storage Gen2 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

Azure Data Lake Storage Gen2에서 패스스루 쿼리를 사용하려면 다음 구문을 사용할 수 있습니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 Azure Data Lake Storage Gen2에 있는 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 `customer` 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 Azure Data Lake Storage Gen2 커넥터용 pom.xml 파일을 참조하세요.

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Azure Synapse 커넥터

Azure Synapse Analytics용 Amazon Athena 커넥터를 사용하면 Amazon Athena가 JDBC를 사용하여 Azure Synapse 데이터베이스에 대해 SQL 쿼리를 실행할 수 있습니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 멀티플렉서 설정에서 유출 버킷과 접두사는 모든 데이터베이스 인스턴스에서 공유됩니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.
- 필터 조건에서 `Date` 및 `Timestamp` 데이터 형식을 적절한 데이터 형식으로 캐스팅해야 합니다.
- `Real` 및 `Float` 형식의 음수 값을 검색하려면 <= 또는 >= 연산자를 사용합니다.
- `binary`, `varbinary`, `image` 및 `rowversion` 데이터 형식은 지원되지 않습니다.

용어

다음 용어는 Synapse 커넥터와 관련이 있습니다.

- 데이터베이스 인스턴스 - 온프레미스, Amazon EC2 또는 Amazon RDS에 배포된 데이터베이스의 모든 인스턴스.
- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.
- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - `connection_string` 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.
- 멀티플렉싱 핸들러 - 여러 데이터베이스 연결을 수락하고 사용할 수 있는 Lambda 핸들러.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 Synapse 커넥터를 구성합니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 데이터베이스 인스턴스에 연결합니다.

```
synapse://${jdbc_connection_string}
```

멀티플렉싱 핸들러 사용

멀티플렉서를 사용하여 단일 Lambda 함수로 여러 데이터베이스 인스턴스에 연결할 수 있습니다. 요청은 카탈로그 이름을 기준으로 라우팅됩니다. Lambda에서 다음 클래스를 사용합니다.

핸들러	Class
복합 핸들러	`SynapseMuxCompositeHandler`
메타데이터 핸들러	`SynapseMuxMetadataHandler`
레코드 핸들러	`SynapseMuxRecordHandler`

멀티플렉싱 핸들러 파라미터

파라미터	설명
$*catalog*_connection_string	필수 사항입니다. 데이터베이스 인스턴스 연결 문자열. Athena에서 사용되는 카탈로그의 이름을 환경 변수 앞에 붙입니다. 예를 들어, Athena에 등록된 카탈로그가 mysynapsecatalog 인 경우 환경 변수 이름은 mysynapsecatalog_connection_string 입니다.
default	필수 사항입니다. 기본 연결 문자열. 이 문자열은 카탈로그가 lambda:${ *AWS_LAMBDA_FUNCTION_NAME* }일 때 사용됩니다.

다음은 synapse1(기본값)과 synapse2라는 2개의 데이터베이스 인스턴스를 지원하는 Synapse MUX Lambda 함수에 대한 예제 속성입니다.

속성	값
default	synapse://jdbc:synapse://synapse1.hostname:port;databaseName= *<database_name>* ;${*secret1_name* }
synapse_catalog1_connection_string	synapse://jdbc:synapse://synapse1.hostname:port;databaseName= *<database_name>* ;${*secret1_name* }
synapse_catalog2_connection_string	synapse://jdbc:synapse://synapse2.hostname:port;databaseName= *<database_name>* ;${*secret2_name* }

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

  > **Important**
  >
  > 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 `username` 및 `password` 값으로 바꿉니다.

  Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

  ```
  {"username": "${username}", "password": "${password}"}
  ```

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 ${secret_name}입니다.

```
synapse://jdbc:synapse://hostname:port;databaseName=<database_name>;${secret_name}
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
synapse://jdbc:synapse://
hostname:port;databaseName=<database_name>;user=<user>;password=<password>
```

단일 연결 핸들러 사용

다음과 같은 단일 연결 메타데이터 및 레코드 핸들러를 사용하여 단일 Synapse 인스턴스에 연결할 수 있습니다.

핸들러 유형	Class
복합 핸들러	`SynapseCompositeHandler`
메타데이터 핸들러	`SynapseMetadataHandler`
레코드 핸들러	`SynapseRecordHandler`

단일 연결 핸들러 파라미터

파라미터	설명
`default`	필수 사항입니다. 기본 연결 문자열.

단일 연결 핸들러는 하나의 데이터베이스 인스턴스를 지원하며 `default` 연결 문자열 파라미터를 제공해야 합니다. 다른 연결 문자열은 모두 무시됩니다.

다음은 Lambda 함수에서 지원하는 단일 Synapse 인스턴스에 대한 예제 속성입니다.

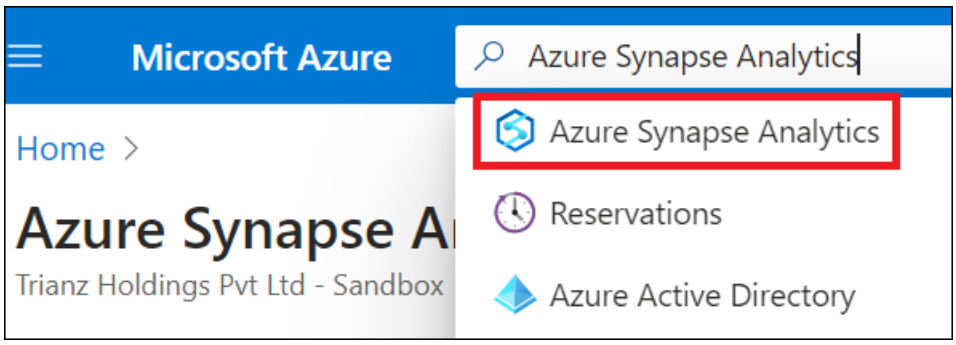
속성	값
`defa`	`synapse://jdbc:sqlserver://hostname:port;databaseName=` *`<database_name>`* `;${`*`secret_name`*`}`

Active Directory 인증 구성

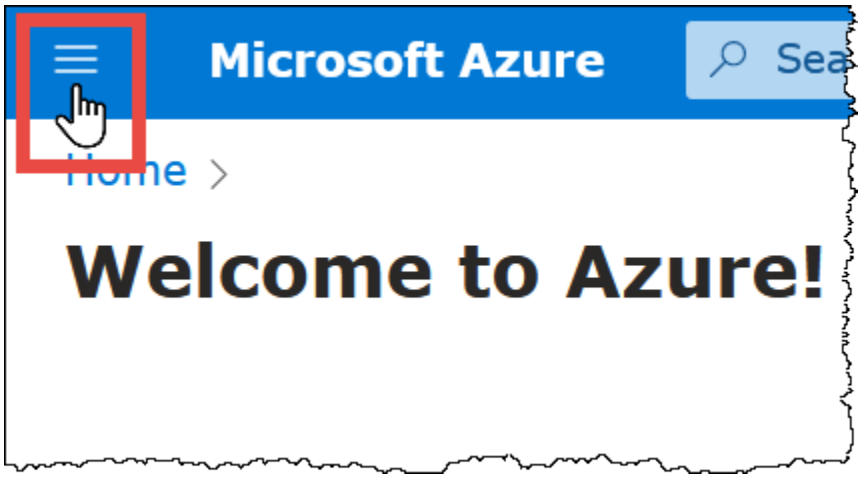
Amazon Athena Azure Synapse 커넥터는 Microsoft Active Directory 인증을 지원합니다. 시작하기 전에 먼저 Microsoft Azure 포털에서 관리 사용자를 구성한 다음 AWS Secrets Manager를 사용하여 보안 암호를 생성해야 합니다.

Active Directory 관리 사용자를 설정하려면

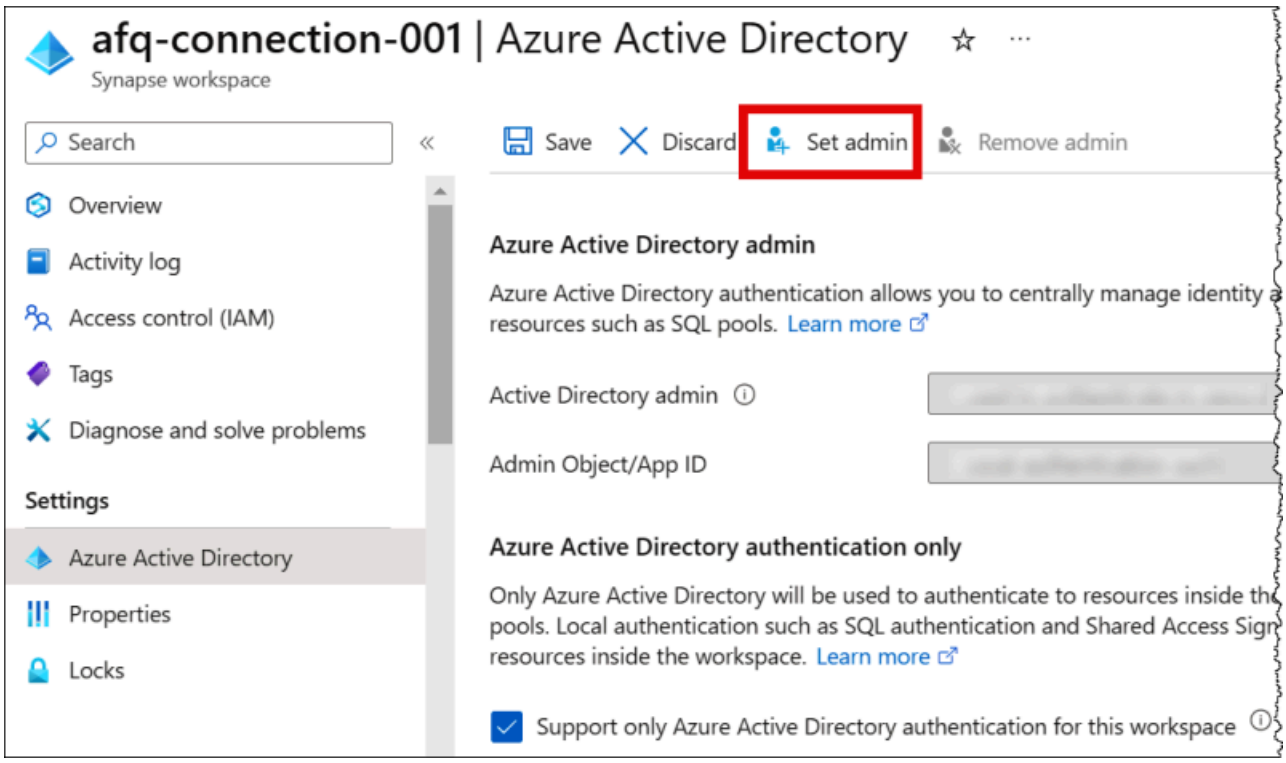
1. 관리자 권한이 있는 계정을 사용하여 Microsoft Azure 포털(https://portal.azure.com/)에 로그인합니다.
2. 검색 상자에 Azure Synapse Analytics를 입력한 다음 Azure Synapse Analytics를 선택합니다.



3. 왼쪽의 메뉴를 엽니다.



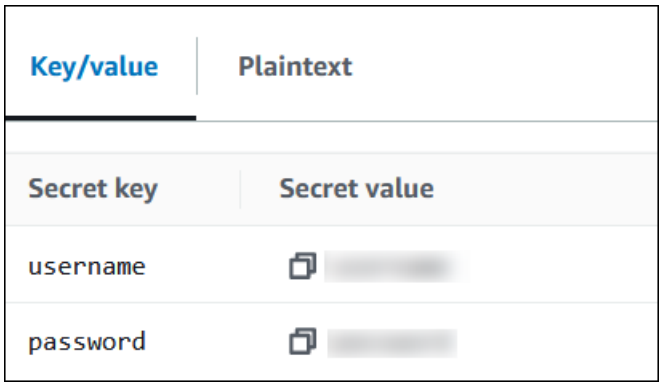
4. 탐색 창에서 Azure Active Directory를 선택합니다.
5. Set admin 탭에서 Active Directory admin을 새 사용자 또는 기존 사용자로 설정합니다.



6. AWS Secrets Manager에 관리 사용자 이름과 암호 보안 인증을 저장합니다. Secrets Manager에서 보안 암호를 생성하는 방법에 대한 자세한 내용은 AWS Secrets Manager 보안 암호 생성을 참조하세요.

Secrets Manager에서 보안 암호를 보려면

1. https://console.aws.amazon.com/secretsmanager/에서 Secrets Manager 콘솔을 엽니다.
2. 탐색 창에서 Secrets(보안 암호)를 선택합니다.
3. Secrets(보안 암호) 페이지에서 보안 암호에 대한 링크를 선택합니다.
4. 보안 암호에 대한 세부 정보 페이지에서 Retrieve secret value(보안 암호 값 검색)를 선택합니다.



연결 문자열 수정

커넥터에 대해 Active Directory 인증을 활성화하려면 다음 구문을 사용하여 연결 문자열을 수정합니다.

```
synapse://jdbc:synapse://
hostname:port;databaseName=database_name;authentication=ActiveDirectoryPassword;
{secret_name}
```

ActiveDirectoryServicePrincipal 사용

Amazon Athena Azure Synapse 커넥터는 `ActiveDirectoryServicePrincipal`도 지원합니다. 이를 활성화하려면 다음과 같이 연결 문자열을 수정합니다.

```
synapse://jdbc:synapse://
hostname:port;databaseName=database_name;authentication=ActiveDirectoryServicePrincipal;
{secret_name}
```

`secret_name`의 경우 애플리케이션 또는 클라이언트 ID를 사용자 이름으로, 암호에 서비스 보안 주체 ID의 암호를 지정합니다.

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
`spill_bucket`	필수 사항입니다. 유출 버킷 이름.
`spill_prefix`	필수 사항입니다. 유출 버킷 키 접두사.
`spill_put_request_headers`	(선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 Synapse와 Apache Arrow의 해당 데이터 형식이 나와 있습니다.

Synapse	화살표
bit	TINYINT
tinyint	SMALLINT
smallint	SMALLINT
int	INT
bigint	BIGINT
decimal	DECIMAL
numeric	FLOAT8
smallmoney	FLOAT8
money	DECIMAL
float[24]	FLOAT4
float[53]	FLOAT8
실제	FLOAT4
datetime	Date(MILLISECOND)
datetime2	Date(MILLISECOND)
smalldatetime	Date(MILLISECOND)
date	Date(DAY)
시간	VARCHAR
datetimeoffset	Date(MILLISECOND)

Synapse	화살표
char[n]	VARCHAR
varchar[n/max]	VARCHAR
nchar[n]	VARCHAR
nvarchar[n/max]	VARCHAR

파티션 및 분할

파티션은 `varchar` 형식의 단일 파티션 열로 표시됩니다. Synapse는 범위 파티셔닝을 지원하므로 파티셔닝은 Synapse 메타데이터 테이블에서 파티션 열과 파티션 범위를 추출하여 구현됩니다. 이러한 범위 값은 분할을 생성하는 데 사용됩니다.

성능

열 하위 집합을 선택하면 쿼리 런타임이 크게 느려집니다. 이 커넥터는 동시성으로 인해 상당한 제한을 나타냅니다.

쿼리에서 스캔하는 데이터를 줄이기 위해 Athena Synapse 커넥터에서 조건자 푸시다운을 수행합니다. 스캔하는 데이터와 쿼리 실행 시간을 줄이도록 간단한 조건자와 복잡한 표현식을 커넥터로 푸시다운합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 `WHERE` 절에 사용되는 표현식입니다. Athena Synapse 커넥터는 이러한 표현식을 결합하고 Synapse로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena Synapse 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND, OR, NOT
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, NULL_IF, IS_NULL
- 산술: ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- 기타: LIKE_PATTERN, IN

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%');
```

패스스루 쿼리

Synapse 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

Synapse에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 Synapse의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 customer 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

- Amazon QuickSight 및 Amazon Athena 페더레이션된 쿼리를 사용하여 Microsoft Azure Synapse 데이터베이스에 저장된 데이터에 대한 시각화와 대시보드를 작성하는 방법을 설명하는 문서는

AWS 빅 데이터 블로그의 Perform multi-cloud analytics using Amazon QuickSight, Amazon Athena Federated Query, and Microsoft Azure Synapse를 참조하세요.

- 최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 Synapse 커넥터용 pom.xml 파일을 참조하세요.
- 이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Cloudera Hive 커넥터

Cloudera Hive용 Amazon Athena 커넥터를 사용하면 Amazon Athena가 Cloudera Hive Hadoop 배포에 대해 SQL 쿼리를 실행할 수 있습니다. 커넥터는 Athena SQL 쿼리를 동일한 HiveQL 구문으로 변환합니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.
- 이 커넥터를 사용하기 전에 VPC와 보안 그룹을 설정합니다. 자세한 내용은 데이터 원본 커넥터용 VPC 생성 단원을 참조하십시오.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 멀티플렉서 설정에서 유출 버킷과 접두사는 모든 데이터베이스 인스턴스에서 공유됩니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.

용어

다음 용어는 Cloudera Hive 커넥터와 관련이 있습니다.

- 데이터베이스 인스턴스 - 온프레미스, Amazon EC2 또는 Amazon RDS에 배포된 데이터베이스의 모든 인스턴스.
- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.

- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.
- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - `connection_string` 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.
- 멀티플렉싱 핸들러 - 여러 데이터베이스 연결을 수락하고 사용할 수 있는 Lambda 핸들러.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 Cloudera Hive 커넥터를 구성합니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 데이터베이스 인스턴스에 연결합니다.

```
hive://${jdbc_connection_string}
```

멀티플렉싱 핸들러 사용

멀티플렉서를 사용하여 단일 Lambda 함수로 여러 데이터베이스 인스턴스에 연결할 수 있습니다. 요청은 카탈로그 이름을 기준으로 라우팅됩니다. Lambda에서 다음 클래스를 사용합니다.

핸들러	Class
복합 핸들러	`HiveMuxCompositeHandler`
메타데이터 핸들러	`HiveMuxMetadataHandler`
레코드 핸들러	`HiveMuxRecordHandler`

멀티플렉싱 핸들러 파라미터

파라미터	설명
$*catalog*_connection_string	필수 사항입니다. 데이터베이스 인스턴스 연결 문자열. Athena에서 사용되는 카탈로그의 이름을 환경 변수 앞에 붙입니다. 예를 들어, Athena에 등록된 카탈로그가 myhivecatalog 인 경우 환경 변수 이름은 myhivecatalog_connection_string 입니다.
default	필수 사항입니다. 기본 연결 문자열. 이 문자열은 카탈로그가 lambda:${ *AWS_LAMBDA_FUNCTION_NAME* }일 때 사용됩니다.

다음은 hive1(기본값)과 hive2라는 2개의 데이터베이스 인스턴스를 지원하는 Hive MUX Lambda 함수에 대한 예제 속성입니다.

속성	값
default	hive://jdbc:hive2://hive1:10000/default?${Test/RDS/hive1}
hive2_catalog1_connection_string	hive://jdbc:hive2://hive1:10000/default?${Test/RDS/hive1}
hive2_catalog2_connection_string	hive://jdbc:hive2://hive2:10000/default?UID=sample&PWD=sample

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

> **Important**
>
> 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 `username` 및 `password` 값으로 바꿉니다.

  Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

  ```
  {"username": "${username}", "password": "${password}"}
  ```

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 `${Test/RDS/hive1}`입니다.

```
hive://jdbc:hive2://hive1:10000/default?...&${Test/RDS/hive1}&...
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
hive://jdbc:hive2://hive1:10000/default?...&UID=sample2&PWD=sample2&...
```

현재 Cloudera Hive 커넥터는 `UID` 및 `PWD` JDBC 속성을 인식합니다.

단일 연결 핸들러 사용

다음과 같은 단일 연결 메타데이터 및 레코드 핸들러를 사용하여 단일 Cloudera Hive 인스턴스에 연결할 수 있습니다.

핸들러 유형	Class
복합 핸들러	`HiveCompositeHandler`
메타데이터 핸들러	`HiveMetadataHandler`
레코드 핸들러	`HiveRecordHandler`

단일 연결 핸들러 파라미터

파라미터	설명
`default`	필수 사항입니다. 기본 연결 문자열.

단일 연결 핸들러는 하나의 데이터베이스 인스턴스를 지원하며 `default` 연결 문자열 파라미터를 제공해야 합니다. 다른 연결 문자열은 모두 무시됩니다.

다음은 Lambda 함수에서 지원하는 단일 Cloudera Hive 인스턴스에 대한 예제 속성입니다.

속성	값
기본값	`hive://jdbc:hive2://hive1:10000/default?secret=${Test/RDS/hive1}`

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
`spill_bucket`	필수 사항입니다. 유출 버킷 이름.
`spill_prefix`	필수 사항입니다. 유출 버킷 키 접두사.
`spill_put_request_headers`	(선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: `{"x-amz-s`

파라미터	설명
	erver-side-encryption" : "AES256"}). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 JDBC, Cloudera Hive 및 Arrow의 해당 데이터 형식이 나와 있습니다.

JDBC	Cloudera Hive	화살표
불	불	Bit
Integer	TINYINT	Tiny
Short	SMALLINT	Smallint
Integer	INT	정수
Long	BIGINT	Bigint
float	"char"	Float4
Double	float8	Float8
날짜	date	DateDay
Timestamp	타임스탬프	DateMilli
String	VARCHAR	Varchar
바이트	bytes	Varbinary
BigDecimal	10진수	10진수
ARRAY	해당 사항 없음(참고 사항 참조)	나열

> **Note**
>
> 현재 Cloudera Hive는 집계 유형 ARRAY, MAP, STRUCT 또는 UNIONTYPE을 지원하지 않습니다. 집계 유형의 열은 SQL에서 VARCHAR 열로 처리됩니다.

파티션 및 분할

파티션은 커넥터에 대한 분할을 생성하는 방법을 결정하는 데 사용됩니다. Athena는 커넥터가 분할을 생성하는 데 도움이 되도록 테이블에 대한 파티셔닝 체계를 나타내는 varchar 유형의 합성 열을 생성합니다. 커넥터는 실제 테이블 정의를 수정하지 않습니다.

성능

Cloudera Hive는 정적 파티션을 지원합니다. Athena Cloudera Hive 커넥터는 이러한 파티션에서 병렬로 데이터를 검색할 수 있습니다. 파티션 배포가 균일한 초대규모 데이터 세트를 쿼리하려면 정적 파티셔닝을 사용하는 것이 좋습니다. Cloudera Hive 커넥터는 동시성으로 인한 제한에 대한 복원력이 뛰어납니다.

Athena Cloudera Hive 커넥터는 조건부 푸시다운을 수행하여 쿼리에서 스캔하는 데이터를 줄입니다. LIMIT 절, 간단한 조건자 및 복잡한 표현식을 커넥터로 푸시다운하여 스캔하는 데이터와 쿼리 실행 시간을 줄입니다.

LIMIT 절

LIMIT N 문은 쿼리로 스캔하는 데이터를 줄입니다. LIMIT N 푸시다운을 통해 커넥터는 Athena에 N개 행만 반환합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 WHERE 절에 사용되는 표현식입니다. Athena Cloudera Hive 커넥터는 이러한 표현식을 결합하고 Cloudera Hive로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena Cloudera Hive 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND, OR, NOT
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_NULL

- 산술: ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- 기타: LIKE_PATTERN, IN

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

패스스루 쿼리

Cloudera Hive 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

Cloudera Hive에서 패스스루 쿼리를 사용하려면 다음 구문을 사용할 수 있습니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 Cloudera Hive의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 `customer` 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 Cloudera Hive 커넥터용 pom.xml 파일을 참조하세요.

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Cloudera Impala 커넥터

Amazon Athena Cloudera Impala 커넥터를 사용하면 Amazon Athena가 Cloudera Impala 배포에 대해 SQL 쿼리를 실행할 수 있습니다. 커넥터는 Athena SQL 쿼리를 동일한 Impala 구문으로 변환합니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.
- 이 커넥터를 사용하기 전에 VPC와 보안 그룹을 설정합니다. 자세한 내용은 데이터 원본 커넥터용 VPC 생성 단원을 참조하십시오.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 멀티플렉서 설정에서 유출 버킷과 접두사는 모든 데이터베이스 인스턴스에서 공유됩니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.

용어

다음 용어는 Cloudera Impala 커넥터와 관련이 있습니다.

- 데이터베이스 인스턴스 - 온프레미스, Amazon EC2 또는 Amazon RDS에 배포된 데이터베이스의 모든 인스턴스.
- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.

- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.
- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - `connection_string` 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.
- 멀티플렉싱 핸들러 - 여러 데이터베이스 연결을 수락하고 사용할 수 있는 Lambda 핸들러.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 Cloudera Impala 커넥터를 구성합니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 Impala 클러스터에 연결합니다.

```
impala://${jdbc_connection_string}
```

멀티플렉싱 핸들러 사용

멀티플렉서를 사용하여 단일 Lambda 함수로 여러 데이터베이스 인스턴스에 연결할 수 있습니다. 요청은 카탈로그 이름을 기준으로 라우팅됩니다. Lambda에서 다음 클래스를 사용합니다.

핸들러	Class
복합 핸들러	`ImpalaMuxCompositeHandler`
메타데이터 핸들러	`ImpalaMuxMetadataHandler`
레코드 핸들러	`ImpalaMuxRecordHandler`

멀티플렉싱 핸들러 파라미터

파라미터	설명
$*catalog*_connection_string	필수 사항입니다. Athena 카탈로그에 대한 Impala 클러스터 연결 문자열입니다. Athena에서 사용되는 카탈로그의 이름을 환

파라미터	설명
	경 변수 앞에 붙입니다. 예를 들어, Athena에 등록된 카탈로그가 myimpalacatalog 인 경우 환경 변수 이름은 myimpalacatalog_connection_string 입니다.
default	필수 사항입니다. 기본 연결 문자열. 이 문자열은 카탈로그가 lambda:${ *AWS_LAMBDA_FUNCTION_NAME* }일 때 사용됩니다.

다음은 impala1(기본값)과 impala2라는 2개의 데이터베이스 인스턴스를 지원하는 Impala MUX Lambda 함수에 대한 예제 속성입니다.

속성	값
default	impala://jdbc:impala://some.impala.host.name:21050/?${Test/impala1}
impala_catalog1_connection_string	impala://jdbc:impala://someother.impala.host.name:21050/?${Test/impala1}
impala_catalog2_connection_string	impala://jdbc:impala://another.impala.host.name:21050/?UID=sample&PWD=sample

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

> **Important**
>
> 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 `username` 및 `password` 값으로 바꿉니다.

  Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

  ```
  {"username": "${username}", "password": "${password}"}
  ```

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 `${Test/impala1host}`입니다.

```
impala://jdbc:impala://Impala1host:21050/?...&${Test/impala1host}&...
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
impala://jdbc:impala://Impala1host:21050/?...&UID=sample2&PWD=sample2&...
```

현재 Cloudera Impala는 `UID` 및 `PWD` JDBC 속성을 인식합니다.

단일 연결 핸들러 사용

다음과 같은 단일 연결 메타데이터 및 레코드 핸들러를 사용하여 단일 Cloudera Impala 인스턴스에 연결할 수 있습니다.

핸들러 유형	Class
복합 핸들러	`ImpalaCompositeHandler`
메타데이터 핸들러	`ImpalaMetadataHandler`
레코드 핸들러	`ImpalaRecordHandler`

단일 연결 핸들러 파라미터

파라미터	설명
default	필수 사항입니다. 기본 연결 문자열.

단일 연결 핸들러는 하나의 데이터베이스 인스턴스를 지원하며 default 연결 문자열 파라미터를 제공해야 합니다. 다른 연결 문자열은 모두 무시됩니다.

다음은 Lambda 함수에서 지원하는 단일 Cloudera Impala 인스턴스에 대한 예제 속성입니다.

속성	값
default	impala://jdbc:impala://Impala1host:21050/?secret=${Test/impala1host}

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
spill_bucket	필수 사항입니다. 유출 버킷 이름.
spill_prefix	필수 사항입니다. 유출 버킷 키 접두사.
spill_put_request_headers	(선택 사항) 유출에 사용되는 Amazon S3 putObject 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: {"x-amz-server-side-encryption" : "AES256"}). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 JDBC, Cloudera Impala 및 Arrow의 해당 데이터 형식이 나와 있습니다.

JDBC	Cloudera Impala	화살표
불	불	Bit
Integer	TINYINT	Tiny
Short	SMALLINT	Smallint
Integer	INT	정수
Long	BIGINT	Bigint
float	"char"	Float4
Double	float8	Float8
날짜	date	DateDay
Timestamp	타임스탬프	DateMilli
String	VARCHAR	Varchar
바이트	bytes	Varbinary
BigDecimal	10진수	10진수
ARRAY	해당 사항 없음(참고 사항 참조)	나열

> **Note**
>
> 현재 Cloudera Impala는 집계 유형 `ARRAY`, `MAP`, `STRUCT` 또는 `UNIONTYPE`을 지원하지 않습니다. 집계 유형의 열은 SQL에서 `VARCHAR` 열로 처리됩니다.

파티션 및 분할

파티션은 커넥터에 대한 분할을 생성하는 방법을 결정하는 데 사용됩니다. Athena는 커넥터가 분할을 생성하는 데 도움이 되도록 테이블에 대한 파티셔닝 체계를 나타내는 `varchar` 유형의 합성 열을 생성합니다. 커넥터는 실제 테이블 정의를 수정하지 않습니다.

성능

Cloudera Impala는 정적 파티션을 지원합니다. Athena Cloudera Impala 커넥터는 이러한 파티션에서 병렬로 데이터를 검색할 수 있습니다. 파티션 배포가 균일한 초대규모 데이터 세트를 쿼리하려면 정적 파티셔닝을 사용하는 것이 좋습니다. Cloudera Impala 커넥터는 동시성으로 인한 제한에 대한 복원력이 뛰어납니다.

Athena Cloudera Impala 커넥터는 조건부 푸시다운을 수행하여 쿼리에서 스캔하는 데이터를 줄입니다. `LIMIT` 절, 간단한 조건자 및 복잡한 표현식을 커넥터로 푸시다운하여 스캔하는 데이터와 쿼리 실행 시간을 줄입니다.

LIMIT 절

`LIMIT N` 문은 쿼리로 스캔하는 데이터를 줄입니다. `LIMIT N` 푸시다운을 통해 커넥터는 Athena에 `N`개 행만 반환합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 `WHERE` 절에 사용되는 표현식입니다. Athena Cloudera Impala 커넥터는 이러한 표현식을 결합하고 Cloudera Impala로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena Cloudera Impala 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND, OR, NOT
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL
- 산술: ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- 기타: LIKE_PATTERN, IN

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

패스스루 쿼리

Cloudera Impala 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

Cloudera Impala에서 패스스루 쿼리를 사용하려면 다음 구문을 사용할 수 있습니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 Cloudera Impala의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 `customer` 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 Cloudera Impala 커넥터용 pom.xml 파일을 참조하세요.

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena CloudWatch 커넥터

Amazon Athena CloudWatch 커넥터를 통해 Amazon Athena는 CloudWatch와 통신할 수 있고, 이로써 SQL을 사용하여 로그 데이터를 쿼리할 수 있습니다.

커넥터는 LoggRoups를 스키마로 매핑하고 각 LogStream을 테이블로 매핑합니다. 또한 커넥터는 LogGroup에 모든 LogStream을 포함하는 특별 `all_log_streams` 뷰도 매핑합니다. 이 뷰를 사용하면 각 LogStream을 개별적으로 검색하는 대신 LogGroup의 모든 로그를 한 번에 쿼리할 수 있습니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 CloudWatch 커넥터를 구성합니다.

- spill_bucket – Lambda 함수 제한을 초과하는 데이터에 대한 Amazon S3 버킷을 지정합니다.
- spill_prefix – (선택 사항) 기본값은 `athena-federation-spill`이라는 지정된 `spill_bucket`의 하위 폴더입니다. 미리 정해진 일 수 또는 시간보다 오래된 유출을 삭제하려면 이 위치에서 Amazon S3 스토리지 수명 주기를 구성하는 것이 좋습니다.
- spill_put_request_headers – (선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵입니다(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.
- kms_key_id – (선택 사항) 기본적으로 Amazon S3로 유출된 모든 데이터는 AES-GCM 인증 암호화 모드와 임의로 생성된 키를 사용하여 암호화됩니다. Lambda 함수가 `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`과 같이 KMS에서 생성된 더 강력한 암호화 키를 사용하도록 하려면 KMS 키 ID를 지정합니다.
- disable_spill_encryption – (선택 사항) `True`로 설정하면 유출 암호화가 비활성화됩니다. S3로 유출되는 데이터가 AES-GCM을 사용하여 암호화되도록 기본값은 `False`입니다(임의로 생성된 키 또는 KMS를 사용하여 키 생성). 유출 암호화를 비활성화하면 특히 유출 위치가 서버 측 암호화를 사용하는 경우 성능이 향상될 수 있습니다.

또한 커넥터는 Amazon Athena Query Federation SDK `ThrottlingInvoker` 생성자를 통해 CloudWatch의 조절 이벤트를 처리하기 위한 AIMD 정체 제어를 지원합니다. 다음 선택적 환경 변수를 설정하여 기본 제한 동작을 조정할 수 있습니다.

- throttle_initial_delay_ms – 첫 번째 혼잡 이벤트 이후에 적용된 초기 호출 지연입니다. 기본값은 10밀리초입니다.
- throttle_max_delay_ms – 호출 간의 최대 지연입니다. TPS를 1000ms로 나누어 도출할 수 있습니다. 기본값은 1,000밀리초입니다.
- throttle_decrease_factor – Athena가 호출 속도를 줄이는 비율입니다. 기본값은 0.5입니다.

- throttle_increase_ms – Athena가 호출 지연을 줄이는 속도입니다. 기본값은 10밀리초입니다.

데이터베이스 및 테이블

Athena CloudWatch 커넥터는 LoggRoups를 스키마 즉, 데이터베이스로 매핑하고 각 LogStream을 테이블로 매핑합니다. 또한 커넥터는 LogGroup에 모든 LogStream을 포함하는 특별 `all_log_streams` 뷰도 매핑합니다. 이 뷰를 사용하면 각 LogStream을 개별적으로 검색하는 대신 LogGroup의 모든 로그를 한 번에 쿼리할 수 있습니다.

Athena CloudWatch 커넥터에 의해 매핑된 모든 테이블에는 다음 스키마가 있습니다. 이 스키마는 CloudWatch Logs에서 제공하는 필드와 일치합니다.

- log_stream – 행이 출처인 LogStream의 이름을 포함하는 `VARCHAR`입니다.
- time – 로그 라인이 생성된 시점의 epoch 시간을 포함하는 `INT64`입니다.
- message – 로그 메시지를 포함하는 `VARCHAR`입니다.

예제

다음 예제에서는 지정된 LogStream에 대해 `SELECT` 쿼리를 수행하는 방법을 보여줍니다.

```
SELECT *
FROM "lambda:cloudwatch_connector_lambda_name"."log_group_path"."log_stream_name"
LIMIT 100
```

다음 예제에서는 `all_log_streams` 보기를 사용하여 지정된 LogGroup의 모든 LogStreams에 대해 쿼리를 수행하는 방법을 보여줍니다.

```
SELECT *
FROM "lambda:cloudwatch_connector_lambda_name"."log_group_path"."all_log_streams"
LIMIT 100
```

필수 권한

이 커넥터에 필요한 IAM 정책에 대한 자세한 내용을 알아보려면 athena-cloudwatch.yaml 파일의 `Policies` 섹션을 검토하세요. 다음 목록에 필요한 권한이 요약되어 있습니다.

- Amazon S3 쓰기 액세스 - 대규모 쿼리의 결과 유출을 위해서는 커넥터에 Amazon S3 위치에 대한 쓰기 액세스 권한이 필요합니다.

- Athena GetQueryExecution - 커넥터는 업스트림 Athena 쿼리가 종료된 경우 이 권한을 사용하여 빠른 실패를 수행합니다.
- CloudWatch Logs 읽기/쓰기 – 커넥터는 이 권한을 사용하여 로그 데이터를 읽고 진단 로그를 작성합니다.

성능

Athena CloudWatch 커넥터는 쿼리에 필요한 로그 스트림 스캔을 병렬화하여 CloudWatch에 대한 쿼리를 최적화하려고 시도합니다. 특정 기간 필터에 대해 조건자 푸시다운은 Lambda 함수와 CloudWatch Logs 내에서 모두 수행됩니다.

최상의 성능을 얻으려면 로그 그룹 이름과 로그 스트림 이름에 소문자만 사용합니다. 대/소문자를 혼합하여 사용하면 커넥터에서 대소문자를 구분하지 않고 검색하므로 더욱 컴퓨팅 집약적입니다.

패스스루 쿼리

CloudWatch 커넥터는 CloudWatch Logs Insights 쿼리 구문을 사용하는 패스스루 쿼리를 지원합니다. CloudWatch Logs Insights에 대한 자세한 내용은 Amazon CloudWatch Logs 사용 설명서에서 CloudWatch Logs Insights를 사용하여 로그 데이터 분석하기를 참조하세요.

CloudWatch에서 패스스루 쿼리를 생성하려면 다음 구문을 사용하세요.

```
SELECT * FROM TABLE(
        system.query(
            STARTTIME => 'start_time',
            ENDTIME => 'end_time',
            QUERYSTRING => 'query_string',
            LOGGROUPNAMES => 'log_group-names',
            LIMIT => 'max_number_of_results'
        ))
```

다음 예제 CloudWatch 패스스루 쿼리는 필드가 1000이 아닐 때 duration 필드를 필터링합니다.

```
SELECT * FROM TABLE(
        system.query(
            STARTTIME => '1710918615308',
            ENDTIME => '1710918615972',
            QUERYSTRING => 'fields @duration | filter @duration != 1000',
            LOGGROUPNAMES => '/aws/lambda/cloudwatch-test-1',
            LIMIT => '2'
```

```
        ))
```

라이선스 정보

Amazon Athena CloudWatch 커넥터 프로젝트는 Apache-2.0 라이선스에 따라 사용이 허가됩니다.

추가적인 리소스

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena CloudWatch 지표 커넥터

Amazon Athena CloudWatch 지표 커넥터를 사용하면 Amazon Athena에서 SQL로 CloudWatch 지표 데이터를 쿼리할 수 있습니다.

Athena 자체에서 CloudWatch에 쿼리 지표를 게시하는 방법에 대한 자세한 내용은 CloudWatch 지표 및 이벤트를 사용하여 비용 관리 및 쿼리 모니터링 섹션을 참조하세요.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 CloudWatch 지표 커넥터를 구성합니다.

- spill_bucket – Lambda 함수 제한을 초과하는 데이터에 대한 Amazon S3 버킷을 지정합니다.
- spill_prefix – (선택 사항) 기본값은 `athena-federation-spill`이라는 지정된 `spill_bucket`의 하위 폴더입니다. 미리 정해진 일 수 또는 시간보다 오래된 유출을 삭제하려면 이 위치에서 Amazon S3 스토리지 수명 주기를 구성하는 것이 좋습니다.
- spill_put_request_headers – (선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵입니다(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.
- kms_key_id – (선택 사항) 기본적으로 Amazon S3로 유출된 모든 데이터는 AES-GCM 인증 암호화 모드와 임의로 생성된 키를 사용하여 암호화됩니다. Lambda 함수가 `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`과 같이 KMS에서 생성된 더 강력한 암호화 키를 사용하도록 하려면 KMS 키 ID를 지정합니다.

- disable_spill_encryption – (선택 사항) `True`로 설정하면 유출 암호화가 비활성화됩니다. S3로 유출되는 데이터가 AES-GCM을 사용하여 암호화되도록 기본값은 `False`입니다(임의로 생성된 키 또는 KMS를 사용하여 키 생성). 유출 암호화를 비활성화하면 특히 유출 위치가 서버 측 암호화를 사용하는 경우 성능이 향상될 수 있습니다.

또한 커넥터는 Amazon Athena Query Federation SDK `ThrottlingInvoker` 생성자를 통해 CloudWatch의 조절 이벤트를 처리하기 위한 AIMD 정체 제어를 지원합니다. 다음 선택적 환경 변수를 설정하여 기본 제한 동작을 조정할 수 있습니다.

- throttle_initial_delay_ms – 첫 번째 혼잡 이벤트 이후에 적용된 초기 호출 지연입니다. 기본값은 10밀리초입니다.
- throttle_max_delay_ms – 호출 간의 최대 지연입니다. TPS를 1000ms로 나누어 도출할 수 있습니다. 기본값은 1,000밀리초입니다.
- throttle_decrease_factor – Athena가 호출 속도를 줄이는 비율입니다. 기본값은 0.5입니다.
- throttle_increase_ms – Athena가 호출 지연을 줄이는 속도입니다. 기본값은 10밀리초입니다.

데이터베이스 및 테이블

Athena CloudWatch 지표 커넥터는 네임스페이스, 차원, 지표 및 지표 값을 `default`라는 단일 스키마의 두 테이블에 매핑합니다.

지표 테이블

`metrics` 테이블에는 네임스페이스, 세트 및 이름의 조합으로 고유하게 정의된 사용 가능한 지표가 포함됩니다. `metrics` 테이블에는 다음 열이 있습니다.

- namespace – 네임스페이스를 포함하는 `VARCHAR`입니다.
- metric_name – 지표 이름을 포함하는 `VARCHAR`입니다.
- dimensions – `dim_name (VARCHAR)` 및 `dim_value (VARCHAR)`로 구성된 `STRUCT` 객체의 `LIST`입니다.
- statistic – 지표에 사용할 수 있는 `VARCH` 통계의 `LIST`입니다(예: `p90`, `AVERAGE` 등).

metric_samples 테이블

`metric_samples` 테이블에는 `metrics` 테이블의 각 지표에 대해 사용 가능한 지표 샘플이 포함되어 있습니다. `metric_samples` 테이블에는 다음 열이 있습니다.

- namespace – 네임스페이스를 포함하는 VARCHAR입니다.
- metric_name – 지표 이름을 포함하는 VARCHAR입니다.
- dimensions – dim_name (VARCHAR) 및 dim_value (VARCHAR)로 구성된 STRUCT 객체의 LIST입니다.
- dim_name – 단일 차원 이름을 쉽게 필터링하는 데 사용할 수 있는 VARCHAR 편의 필드입니다.
- dim_value – 단일 차원 값을 쉽게 필터링하는 데 사용할 수 있는 VARCHAR 편의 필드입니다.
- period – 지표의 '기간'을 초 단위로 나타내는 INT 필드입니다(예: 60초 지표).
- timestamp – 지표 샘플의 epoch 시간(초)을 나타내는 BIGINT 필드입니다.
- value – 샘플 값을 포함하는 FLOAT8 필드입니다.
- statistic – 샘플의 통계 유형을 포함하는 VARCHAR입니다(예: AVERAGE 또는 p90).

필수 권한

이 커넥터에 필요한 IAM 정책에 대한 자세한 내용을 알아보려면 athena-cloudwatch-metrics.yaml 파일의 Policies 섹션을 검토하세요. 다음 목록에 필요한 권한이 요약되어 있습니다.

- Amazon S3 쓰기 액세스 - 대규모 쿼리의 결과 유출을 위해서는 커넥터에 Amazon S3 위치에 대한 쓰기 액세스 권한이 필요합니다.
- Athena GetQueryExecution - 커넥터는 업스트림 Athena 쿼리가 종료된 경우 이 권한을 사용하여 빠른 실패를 수행합니다.
- CloudWatch 지표 읽기 전용 – 커넥터는 이 권한을 사용하여 지표 데이터를 쿼리합니다.
- CloudWatch Logs 쓰기 - 커넥터는 이 액세스 권한을 사용하여 진단 로그를 작성합니다.

성능

Athena CloudWatch 지표 커넥터는 쿼리에 필요한 로그 스트림 스캔을 병렬화하여 CloudWatch 지표에 대한 쿼리를 최적화하려고 시도합니다. 특정 기간, 지표, 네임스페이스 및 차원 필터에 대해 조건자 푸시다운은 Lambda 함수와 CloudWatch Logs 내에서 모두 수행됩니다.

라이선스 정보

Amazon Athena CloudWatch 지표 커넥터 프로젝트는 Apache-2.0 라이선스에 따라 사용이 허가됩니다.

추가적인 리소스

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena AWS CMDB 커넥터

Amazon Athena AWS CMDB 커넥터를 통해 Athena는 다양한 AWS 서비스와 통신할 수 있고, 이로써 SQL을 사용하여 쿼리할 수 있습니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 AWS CMDB 커넥터를 구성합니다.

- spill_bucket – Lambda 함수 제한을 초과하는 데이터에 대한 Amazon S3 버킷을 지정합니다.
- spill_prefix – (선택 사항) 기본값은 `athena-federation-spill`이라는 지정된 `spill_bucket`의 하위 폴더입니다. 미리 정해진 일 수 또는 시간보다 오래된 유출을 삭제하려면 이 위치에서 Amazon S3 스토리지 수명 주기를 구성하는 것이 좋습니다.
- spill_put_request_headers – (선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵입니다(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.
- kms_key_id – (선택 사항) 기본적으로 Amazon S3로 유출된 모든 데이터는 AES-GCM 인증 암호화 모드와 임의로 생성된 키를 사용하여 암호화됩니다. Lambda 함수가 `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`과 같이 KMS에서 생성된 더 강력한 암호화 키를 사용하도록 하려면 KMS 키 ID를 지정합니다.
- disable_spill_encryption – (선택 사항) `True`로 설정하면 유출 암호화가 비활성화됩니다. S3로 유출되는 데이터가 AES-GCM을 사용하여 암호화되도록 기본값은 `False`입니다(임의로 생성된 키 또는 KMS를 사용하여 키 생성). 유출 암호화를 비활성화하면 특히 유출 위치가 서버 측 암호화를 사용하는 경우 성능이 향상될 수 있습니다.
- default_ec2_image_owner – (선택 사항) 설정하면 Amazon Machine Image(AMI)를 필터링하는 기본 Amazon EC2 이미지 소유자를 제어합니다. 이 값을 설정하지 않고 EC2 이미지 테이블에 대한 쿼리에 소유자 필터가 포함되어 있지 않으면 결과에 모든 공개 이미지가 포함됩니다.

데이터베이스 및 테이블

Athena AWS CMDB 커넥터를 통해 AWS 리소스 인벤토리를 쿼리하는 데 다음 데이터베이스와 테이블을 사용할 수 있습니다. 각 테이블에서 사용 가능한 열에 대한 자세한 내용을 보려면 Athena 콘솔 또는 API를 사용하여 `DESCRIBE database.table` 문을 실행합니다.

- ec2 – 이 데이터베이스는 다음을 비롯한 Amazon EC2 관련 리소스를 포함합니다.

- ebs_volumes – Amazon EBS 볼륨의 세부 정보를 포함합니다.
- ec2_instances – EC2 인스턴스의 세부 정보를 포함합니다.
- ec2_images – EC2 인스턴스 이미지의 세부 정보를 포함합니다.
- routing_tables – VPC 라우팅 테이블의 세부 정보를 포함합니다.
- security_groups – 보안 그룹의 세부 정보를 포함합니다.
- subnets – VPC 서브넷의 세부 정보를 포함합니다.
- vpcs – VPC의 세부 정보를 포함합니다.

- emr – 이 데이터베이스는 다음을 비롯한 Amazon EMR 관련 리소스를 포함합니다.

- emr_clusters – EMR 클러스터의 세부 정보를 포함합니다.

- rds – 이 데이터베이스는 다음을 비롯한 Amazon RDS 관련 리소스를 포함합니다.

- rds_instances – RDS 인스턴스의 세부 정보를 포함합니다.

- s3 – 이 데이터베이스는 다음을 비롯한 RDS 관련 리소스를 포함합니다.

- buckets – Amazon S3 버킷의 세부 정보를 포함합니다.
- objects – 콘텐츠를 제외한 Amazon S3 객체의 세부 정보를 포함합니다.

필수 권한

이 커넥터에 필요한 IAM 정책에 대한 자세한 내용을 알아보려면 athena-aws-cmdb.yaml 파일의 `Policies` 섹션을 검토하세요. 다음 목록에 필요한 권한이 요약되어 있습니다.

- Amazon S3 쓰기 액세스 - 대규모 쿼리의 결과 유출을 위해서는 커넥터에 Amazon S3 위치에 대한 쓰기 액세스 권한이 필요합니다.
- Athena GetQueryExecution - 커넥터는 업스트림 Athena 쿼리가 종료된 경우 이 권한을 사용하여 빠른 실패를 수행합니다.
- S3 List – 커넥터는 이 권한을 사용하여 Amazon S3 버킷 및 객체를 나열합니다.
- EC2 Describe – 커넥터는 이 권한을 사용하여 Amazon EC2 인스턴스, 보안 그룹, VPC 및 Amazon EBS 볼륨과 같은 리소스를 설명합니다.
- EMR Describe/List – 커넥터는 이 권한을 사용하여 EMR 클러스터를 설명합니다.
- RDS Describe – 커넥터는 이 권한을 사용하여 RDS 인스턴스를 설명합니다.

성능

현재 Athena AWS CMDB 커넥터는 병렬 스캔을 지원하지 않습니다. 조건자 푸시다운은 Lambda 함수 내에서 수행됩니다. 가능한 경우 부분 조건자는 쿼리 중인 서비스로 푸시됩니다. 예를 들어, 특정 Amazon EC2 인스턴스의 세부 정보에 대한 쿼리는 특정 인스턴스 ID로 EC2 API를 호출하여 대상 설명 작업을 실행합니다.

라이선스 정보

Amazon Athena AWS CMDB 커넥터 프로젝트는 Apache-2.0 라이선스에 따라 사용이 허가됩니다.

추가적인 리소스

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena IBM Db2 커넥터

Db2용 Amazon Athena 커넥터를 사용하면 Amazon Athena가 JDBC를 사용하여 IBM Db2 데이터베이스에 대해 SQL 쿼리를 실행할 수 있습니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.
- 이 커넥터를 사용하기 전에 VPC와 보안 그룹을 설정합니다. 자세한 내용은 데이터 원본 커넥터용 VPC 생성 단원을 참조하십시오.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 멀티플렉서 설정에서 유출 버킷과 접두사는 모든 데이터베이스 인스턴스에서 공유됩니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.
- 필터 조건의 날짜 및 타임스탬프 데이터 형식을 적절한 데이터 형식으로 캐스팅해야 합니다.

용어

다음 용어는 Db2 커넥터와 관련이 있습니다.

- 데이터베이스 인스턴스 - 온프레미스, Amazon EC2 또는 Amazon RDS에 배포된 데이터베이스의 모든 인스턴스.
- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.
- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - `connection_string` 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.
- 멀티플렉싱 핸들러 - 여러 데이터베이스 연결을 수락하고 사용할 수 있는 Lambda 핸들러.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 Db2 커넥터를 구성합니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 데이터베이스 인스턴스에 연결합니다.

```
dbtwo://${jdbc_connection_string}
```

멀티플렉싱 핸들러 사용

멀티플렉서를 사용하여 단일 Lambda 함수로 여러 데이터베이스 인스턴스에 연결할 수 있습니다. 요청은 카탈로그 이름을 기준으로 라우팅됩니다. Lambda에서 다음 클래스를 사용합니다.

핸들러	Class
복합 핸들러	`Db2MuxCompositeHandler`
메타데이터 핸들러	`Db2MuxMetadataHandler`
레코드 핸들러	`Db2MuxRecordHandler`

멀티플렉싱 핸들러 파라미터

파라미터	설명
`$`*`catalog`*`_connection_string`	필수 사항입니다. 데이터베이스 인스턴스 연결 문자열. Athena에서 사용되는 카탈로그의 이름을 환경 변수 앞에 붙입니다. 예를 들어, Athena에 등록된 카탈로그가 `mydbtwocatalog` 인 경우 환경 변수 이름은 `mydbtwocatalog_connection_string` 입니다.
`default`	필수 사항입니다. 기본 연결 문자열. 이 문자열은 카탈로그가 `lambda:${` *`AWS_LAMBDA_FUNCTION_NAME`* `}`일 때 사용됩니다.

다음은 dbtwo1(기본값)과 dbtwo2라는 2개의 데이터베이스 인스턴스를 지원하는 Db2 MUX Lambda 함수에 대한 예제 속성입니다.

속성	값
`default`	`dbtwo://jdbc:db2://dbtwo1.hostname:port/` *`database_name`* `:${`*`secret1_name`*` }`
`dbtwo_catalog1_connection_string`	`dbtwo://jdbc:db2://dbtwo1. hostname:port/` *`database_name`* `:${`*`secret1_name`*` }`

속성	값
dbtwo_catalog2_connection_string	dbtwo://jdbc:db2://dbtwo2. hostname: port/ *database_name* :${*secret2_name* }

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

  > **Important**
  >
  > 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 `username` 및 `password` 값으로 바꿉니다.

  Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

  ```
  {"username": "${username}", "password": "${password}"}
  ```

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 `${secret_name}`입니다.

```
dbtwo://jdbc:db2://hostname:port/database_name:${secret_name}
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
dbtwo://jdbc:db2://hostname:port/database_name:user=user_name;password=password;
```

단일 연결 핸들러 사용

다음과 같은 단일 연결 메타데이터 및 레코드 핸들러를 사용하여 단일 Db2 인스턴스에 연결할 수 있습니다.

핸들러 유형	Class
복합 핸들러	`Db2CompositeHandler`
메타데이터 핸들러	`Db2MetadataHandler`
레코드 핸들러	`Db2RecordHandler`

단일 연결 핸들러 파라미터

파라미터	설명
`default`	필수 사항입니다. 기본 연결 문자열.

단일 연결 핸들러는 하나의 데이터베이스 인스턴스를 지원하며 `default` 연결 문자열 파라미터를 제공해야 합니다. 다른 연결 문자열은 모두 무시됩니다.

다음은 Lambda 함수에서 지원하는 단일 Db2 인스턴스에 대한 예제 속성입니다.

속성	값
`default`	dbtwo://jdbc:db2://hostname:port/*database_name* :${secret_name}

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
spill_bucket	필수 사항입니다. 유출 버킷 이름.
spill_prefix	필수 사항입니다. 유출 버킷 키 접두사.
spill_put_request_headers	(선택 사항) 유출에 사용되는 Amazon S3 putObject 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: {"x-amz-server-side-encryption" : "AES256"}). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 JDBC와 Arrow의 해당 데이터 형식이 나와 있습니다.

Db2	화살표
CHAR	VARCHAR
VARCHAR	VARCHAR
날짜	DATEDAY
TIME	VARCHAR
TIMESTAMP	DATEMILLI
DATETIME	DATEMILLI
BOOLEAN	BOOL
SMALLINT	SMALLINT
INTEGER	INT
BIGINT	BIGINT

Db2	화살표
DECIMAL	DECIMAL
REAL	FLOAT8
DOUBLE	FLOAT8
DECFLOAT	FLOAT8

파티션 및 분할

파티션은 하나 이상의 `varchar` 형식 파티션 열로 표시됩니다. Db2 커넥터는 다음 조직 체계를 사용하여 파티션을 생성합니다.

- 해시별 배포
- 범위별 파티션
- 차원별 구성

커넥터는 하나 이상의 Db2 메타데이터 테이블에서 파티션 수, 열 이름과 같은 파티션 세부 정보를 검색합니다. 분할은 식별된 파티션 수를 기준으로 생성됩니다.

성능

Athena Db2 커넥터는 조건부 푸시다운을 수행하여 쿼리에서 스캔하는 데이터를 줄입니다. `LIMIT` 절, 간단한 조건자 및 복잡한 표현식을 커넥터로 푸시다운하여 스캔하는 데이터와 쿼리 실행 시간을 줄입니다.

LIMIT 절

`LIMIT N` 문은 쿼리로 스캔하는 데이터를 줄입니다. `LIMIT N` 푸시다운을 통해 커넥터는 Athena에 `N`개 행만 반환합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 `WHERE` 절에 사용되는 표현식입니다. Athena Db2 커넥터는 이러한 표현식을 결합하고 Db2로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena Db2 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND, OR, NOT
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, IS_NULL
- 산술: ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- 기타: LIKE_PATTERN, IN

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

패스스루 쿼리

Db2 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

Db2에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 Db2의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 `customer` 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 Db2 커넥터용 pom.xml 파일을 참조하세요.

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena IBM Db2 AS/400(Db2 iSeries) 커넥터

Db2 AS/400용 Amazon Athena 커넥터를 사용하면 Amazon Athena가 JDBC를 사용하여 IBM Db2 AS/400(Db2 iSeries) 데이터베이스에 대해 SQL 쿼리를 실행할 수 있습니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.
- 이 커넥터를 사용하기 전에 VPC와 보안 그룹을 설정합니다. 자세한 내용은 데이터 원본 커넥터용 VPC 생성 단원을 참조하십시오.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 멀티플렉서 설정에서 유출 버킷과 접두사는 모든 데이터베이스 인스턴스에서 공유됩니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.
- 필터 조건의 날짜 및 타임스탬프 데이터 형식을 적절한 데이터 형식으로 캐스팅해야 합니다.

용어

다음 용어는 Db2 AS/400 커넥터와 관련이 있습니다.

- 데이터베이스 인스턴스 - 온프레미스, Amazon EC2 또는 Amazon RDS에 배포된 데이터베이스의 모든 인스턴스.

- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.
- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - `connection_string` 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.
- 멀티플렉싱 핸들러 - 여러 데이터베이스 연결을 수락하고 사용할 수 있는 Lambda 핸들러.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 Db2 AS/400 커넥터를 구성합니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 데이터베이스 인스턴스에 연결합니다.

```
db2as400://${jdbc_connection_string}
```

멀티플렉싱 핸들러 사용

멀티플렉서를 사용하여 단일 Lambda 함수로 여러 데이터베이스 인스턴스에 연결할 수 있습니다. 요청은 카탈로그 이름을 기준으로 라우팅됩니다. Lambda에서 다음 클래스를 사용합니다.

핸들러	Class
복합 핸들러	`Db2MuxCompositeHandler`
메타데이터 핸들러	`Db2MuxMetadataHandler`
레코드 핸들러	`Db2MuxRecordHandler`

멀티플렉싱 핸들러 파라미터

파라미터	설명
$*catalog*_connection_string	필수 사항입니다. 데이터베이스 인스턴스 연결 문자열. Athena에서 사용되는 카탈로그의 이름을 환경 변수 앞에 붙입니다. 예를 들어, Athena에 등록된 카탈로그가 mydb2as400catalog 인 경우 환경 변수 이름은 mydb2as400catalog_connection_string 입니다.
default	필수 사항입니다. 기본 연결 문자열. 이 문자열은 카탈로그가 lambda:${ *AWS_LAMBDA_FUNCTION_NAME* }일 때 사용됩니다.

다음은 db2as4001(기본값)과 db2as4002라는 2개의 데이터베이스 인스턴스를 지원하는 Db2 MUX Lambda 함수에 대한 예제 속성입니다.

속성	값
default	db2as400://jdbc:as400:// *<ip_address>* ;*<properties>* ;:${*<secret name>*};
db2as400_catalog1_connection_string	db2as400://jdbc:as400://db2as4001. hostname/:${ *secret1_name* }
db2as400_catalog2_connection_string	db2as400://jdbc:as400://db2as4002. hostname/:${ *secret2_name* }
db2as400_catalog3_connection_string	db2as400://jdbc:as400:// *<ip_address>* ;user=*<username>* ;password= *<password>* ;*<properties>* ;

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

  > **Important**
  >
  > 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 `username` 및 `password` 값으로 바꿉니다.

  Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

  ```
  {"username": "${username}", "password": "${password}"}
  ```

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 `${secret_name}`입니다.

```
db2as400://jdbc:as400://<ip_address>;<properties>;:${<secret_name>};
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
db2as400://jdbc:as400://<ip_address>;user=<username>;password=<password>;<properties>;
```

단일 연결 핸들러 사용

다음과 같은 단일 연결 메타데이터 및 레코드 핸들러를 사용하여 단일 Db2 AS/400 인스턴스에 연결할 수 있습니다.

핸들러 유형	Class
복합 핸들러	`Db2CompositeHandler`
메타데이터 핸들러	`Db2MetadataHandler`
레코드 핸들러	`Db2RecordHandler`

단일 연결 핸들러 파라미터

파라미터	설명
`default`	필수 사항입니다. 기본 연결 문자열.

단일 연결 핸들러는 하나의 데이터베이스 인스턴스를 지원하며 `default` 연결 문자열 파라미터를 제공해야 합니다. 다른 연결 문자열은 모두 무시됩니다.

다음은 Lambda 함수에서 지원하는 단일 Db2 AS/400 인스턴스에 대한 예제 속성입니다.

속성	값
`default`	`db2as400://jdbc:as400://` *`<ip_address>`* `;`*`<properties>`* `::` `${`*`<secret_name>`* `};`

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
`spill_bucket`	필수 사항입니다. 유출 버킷 이름.
`spill_prefix`	필수 사항입니다. 유출 버킷 키 접두사.
`spill_put_request_headers`	(선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: `{"x-amz-s`

파라미터	설명
	`erver-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 JDBC와 Apache Arrow의 해당 데이터 형식이 나와 있습니다.

Db2 AS/400	화살표
CHAR	VARCHAR
VARCHAR	VARCHAR
날짜	DATEDAY
TIME	VARCHAR
TIMESTAMF	DATEMILLI
DATETIME	DATEMILLI
BOOLEAN	BOOL
SMALLINT	SMALLINT
INTEGER	INT
BIGINT	BIGINT
DECIMAL	DECIMAL
REAL	FLOAT8
DOUBLE	FLOAT8

Db2 AS/400	화살표
DECFLOAT	FLOAT8

파티션 및 분할

파티션은 하나 이상의 `varchar` 형식 파티션 열로 표시됩니다. Db2 AS/400 커넥터는 다음 구성 체계를 사용하여 파티션을 생성합니다.

- 해시별 배포
- 범위별 파티션
- 차원별 구성

커넥터는 하나 이상의 Db2 AS/400 메타데이터 테이블에서 파티션 수, 열 이름과 같은 파티션 세부 정보를 검색합니다. 분할은 식별된 파티션 수를 기준으로 생성됩니다.

성능

성능 개선을 위해 다음 예제와 같이 조건자 푸시다운을 사용하여 Athena에서 쿼리합니다.

```
SELECT * FROM "lambda:<LAMBDA_NAME>"."<SCHEMA_NAME>"."<TABLE_NAME>"
 WHERE integercol = 2147483647
```

```
SELECT * FROM "lambda: <LAMBDA_NAME>"."<SCHEMA_NAME>"."<TABLE_NAME>"
 WHERE timestampcol >= TIMESTAMP '2018-03-25 07:30:58.878'
```

패스스루 쿼리

Db2 AS/400 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

Db2 AS/400에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 Db2 AS/400의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 `customer` 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 Db2 AS/400 커넥터용 pom.xml 파일을 참조하세요.

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena DocumentDB 커넥터

Amazon Athena DocumentDB 커넥터를 통해 Athena는 DocumentDB와 통신할 수 있고, 이로써 SQL을 사용하여 DocumentDB를 쿼리할 수 있습니다. 커넥터는 MongoDB와 호환되는 모든 엔드포인트과 함께 작동합니다.

기존의 관계형 데이터 스토어와 달리 Amazon DocumentDB 컬렉션에는 설정된 스키마가 없습니다. DocumentDB에는 메타데이터 스토어가 없습니다. DocumentDB 컬렉션의 항목마다 필드와 데이터 형식이 다를 수 있습니다.

DocumentDB 커넥터는 테이블 스키마 정보를 생성하기 위한 두 가지 메커니즘인 기본 스키마 추론과 AWS Glue Data Catalog 메타데이터를 지원합니다.

스키마 추론이 기본값입니다. 이 옵션은 컬렉션에 있는 소수의 문서를 스캔하고, 모든 필드의 통합을 형성하고, 겹치지 않는 데이터 형식을 가진 필드를 강제 변환합니다. 이 옵션은 항목이 대부분 균일한 컬렉션에 적합합니다.

더 다양한 데이터 형식이 있는 컬렉션의 경우 커넥터는 AWS Glue Data Catalog에서 메타데이터 검색을 지원합니다. 커넥터가 DocumentDB 데이터베이스 및 컬렉션 이름과 일치하는 AWS Glue 데이터베

이스 및 테이블을 발견하면 해당 AWS Glue 테이블에서 스키마 정보를 가져옵니다. AWS Glue 테이블을 생성할 때 DocumentDB 컬렉션에서 액세스하려는 모든 필드의 상위 세트로 만드는 것이 좋습니다.

계정에서 Lake Formation을 활성화한 경우 AWS Serverless Application Repository에 배포한 Athena 페더레이션형 Lambda 커넥터의 IAM 역할은 Lake Formation에서 AWS Glue Data Catalog에 대한 읽기 액세스 권한을 가지고 있어야 합니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 DocumentDB 커넥터를 구성합니다.

- spill_bucket – Lambda 함수 제한을 초과하는 데이터에 대한 Amazon S3 버킷을 지정합니다.
- spill_prefix – (선택 사항) 기본값은 `athena-federation-spill`이라는 지정된 `spill_bucket`의 하위 폴더입니다. 미리 정해진 일 수 또는 시간보다 오래된 유출을 삭제하려면 이 위치에서 Amazon S3 스토리지 수명 주기를 구성하는 것이 좋습니다.
- spill_put_request_headers – (선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵입니다(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.
- kms_key_id – (선택 사항) 기본적으로 Amazon S3로 유출된 모든 데이터는 AES-GCM 인증 암호화 모드와 임의로 생성된 키를 사용하여 암호화됩니다. Lambda 함수가 `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`과 같이 KMS에서 생성된 더 강력한 암호화 키를 사용하도록 하려면 KMS 키 ID를 지정합니다.
- disable_spill_encryption – (선택 사항) `True`로 설정하면 유출 암호화가 비활성화됩니다. S3로 유출되는 데이터가 AES-GCM을 사용하여 암호화되도록 기본값은 `False`입니다(임의로 생성된 키 또는 KMS를 사용하여 키 생성). 유출 암호화를 비활성화하면 특히 유출 위치가 서버 측 암호화를 사용하는 경우 성능이 향상될 수 있습니다.
- disable_glue – (선택 사항) true로 설정된 경우 커넥터는 AWS Glue에서 보충 메타데이터 검색을 시도하지 않습니다.
- glue_catalog – (선택 사항) 이 옵션을 사용하여 크로스 계정 AWS Glue 카탈로그를 지정합니다. 기본적으로 커넥터는 자체 AWS Glue 계정에서 메타데이터를 가져오려고 시도합니다.

- default_docdb – 있는 경우 카탈로그 관련 환경 변수가 없을 때 사용할 DocumentDB 연결 문자열을 지정합니다.
- disable_projection_and_casing – (선택 사항) 프로젝션과 대/소문자를 비활성화합니다. 대/소문자를 구분하는 열 이름을 사용하는 Amazon DocumentDB 테이블을 쿼리하려는 경우에 사용합니다. `disable_projection_and_casing` 파라미터는 다음 값을 사용하여 대/소문자 및 열 매핑의 동작을 지정합니다.
  - false – 기본 설정입니다. 프로젝션이 활성화되며, 커넥터에서는 모든 열 이름을 소문자로 예상합니다.
  - true - 프로젝션과 대/소문자 구분을 비활성화합니다. `disable_projection_and_casing` 파라미터를 사용할 때는 다음 사항에 유의하세요.
    - 파라미터를 사용하면 대역폭 사용량이 증가할 수 있습니다. 또한 Lambda 함수가 데이터 소스와 동일한 AWS 리전에 있지 않을 경우 대역폭 사용량이 증가하여 더 높은 표준 AWS 리전 간 전송 비용이 청구됩니다. 리전 간 전송 비용에 대한 자세한 내용은 AWS 파트너 네트워크 블로그의 서버 및 서버리스 아키텍처에 대한 AWS 데이터 전송 요금을 참조하세요.
    - 더 많은 수의 바이트가 전송되고 더 많은 수의 바이트가 더 많은 역직렬화 시간을 필요로 하기 때문에 전체 지연 시간이 늘어날 수 있습니다.
- enable_case_insensitive_match - (선택 사항) `true`인 경우 Amazon DocumentDB의 스키마 및 테이블 이름에 대해 대소문자를 구분하지 않는 검색을 수행합니다. 기본값은 `false`입니다. 쿼리에 대문자 스키마 또는 테이블 이름이 포함된 경우에 사용합니다.

연결 문자열 지정

커넥터에서 사용하는 DocumentDB 인스턴스에 대한 DocumentDB 연결 세부 정보를 정의하는 속성을 하나 이상 제공할 수 있습니다. 이렇게 하려면 Athena에서 사용하려는 카탈로그 이름에 해당하는 Lambda 환경 변수를 설정합니다. 예를 들어, 다음 쿼리를 사용하여 Athena에서 서로 다른 2개의 DocumentDB 인스턴스를 쿼리하려고 한다고 가정해 보겠습니다.

```
SELECT * FROM "docdb_instance_1".database.table
```

```
SELECT * FROM "docdb_instance_2".database.table
```

이 두 SQL 문을 사용하려면 먼저 2개의 환경 변수 `docdb_instance_1`과 `docdb_instance_2`를 Lambda 함수에 추가해야 합니다. 각 값은 다음 형식의 DocumentDB 연결 문자열이어야 합니다.

```
mongodb://:@:/?ssl=true&ssl_ca_certs=rds-combined-ca-bundle.pem&replicaSet=rs0
```

보안 암호 사용

필요에 따라 연결 문자열 세부 정보 값의 일부 또는 전체에 대해 AWS Secrets Manager를 사용할 수 있습니다. Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 인터넷 액세스 또는 VPC 엔드포인트가 Lambda 함수에 연결된 VPC에 있어야 합니다.

`${my_secret}` 구문을 사용하여 연결 문자열에 Secrets Manager의 보안 암호 이름을 입력하는 경우 커넥터는 `${my_secret}`을 Secrets Manager의 일반 텍스트 값으로 정확하게 바꿉니다. 보안 암호는 *<username>*:*<password>* 값을 가진 일반 텍스트 보안 암호로 저장되어야 합니다 `{username:`*<username>*`,password:`*<password>*`}`로 저장된 보안 암호는 연결 문자열에 제대로 전달되지 않습니다.

전체 연결 문자열에도 보안 암호를 사용할 수 있으며 보안 암호 내에서 사용자 이름과 암호를 정의할 수 있습니다.

예를 들어, `docdb_instance_1`에 대한 Lambda 환경 변수를 다음 값으로 설정했다고 가정해 보겠습니다.

```
mongodb://${docdb_instance_1_creds}@myhostname.com:123/?ssl=true&ssl_ca_certs=rds-combined-ca-bundle.pem&replicaSet=rs0
```

Athena Query Federation SDK는 자동으로 Secrets Manager에서 `docdb_instance_1_creds`라는 보안 암호를 검색하고 `${docdb_instance_1_creds}` 대신 해당 값을 삽입하려고 시도합니다. `${ }` 문자 조합으로 묶은 연결 문자열 부분은 Secrets Manager의 보안 암호로 해석됩니다. 커넥터가 Secrets Manager에서 찾을 수 없는 보안 암호 이름을 지정하면 커넥터가 텍스트를 바꾸지 않습니다.

AWS Glue에서 데이터베이스 및 테이블 설정

커넥터의 기본 제공 스키마 추론 기능은 제한된 수의 문서를 스캔하고 데이터 형식의 하위 세트만 지원하므로 메타데이터에 AWS Glue를 대신 사용할 수 있습니다.

Amazon DocumentDB에서 사용할 AWS Glue 테이블을 활성화하려면 보충 메타데이터를 제공할 DocumentDB 데이터베이스 및 컬렉션에 대한 AWS Glue 데이터베이스와 테이블이 있어야 합니다.

보충 메타데이터에 AWS Glue 테이블 사용

1. AWS Glue 콘솔에서 테이블과 데이터베이스를 편집할 때 다음 테이블 속성을 추가합니다.

   - docdb-metadata-flag – 이 속성은 보충 메타데이터에 테이블을 사용할 수 있음을 DocumentDB 커넥터에 나타냅니다. `docdb-metadata-flag` 속성이 테이블 속성 목록에 있는 한 `docdb-metadata-flag`에 모든 값을 제공할 수 있습니다.

2. (선택 사항) sourceTable 테이블 속성을 추가합니다. 이 속성에 따라 Amazon DocumentDB의 소스 테이블 이름이 정의됩니다. AWS Glue 테이블 이름 지정 규칙으로 인해 Amazon DocumentDB 테이블과 동일한 이름으로 AWS Glue 테이블을 생성할 수 없는 경우 이 속성을 사용합니다. 예를 들어, AWS Glue 테이블 이름에는 대문자가 허용되지 않지만 Amazon DocumentDB 테이블 이름에는 대문자가 허용됩니다.
3. (선택 사항) columnMapping 테이블 속성을 추가합니다. 이 속성은 열 이름 매핑을 정의합니다. AWS Glue 열 이름 지정 규칙으로 인해 Amazon DocumentDB 테이블의 열과 동일한 열 이름으로 AWS Glue 테이블을 생성할 수 없는 경우 이 속성을 사용합니다. Amazon DocumentDB 열 이름에는 대문자가 허용되지만 AWS Glue 열 이름에는 대문자가 허용되지 않기 때문에 이 방법이 유용할 수 있습니다.

   `columnMapping` 속성 값은 `col1=Col1,col2=Col2` 형식의 매핑 세트여야 합니다.

   > **Note**
   >
   > 열 매핑은 상위 열 이름에만 적용되고 중첩된 필드에는 적용되지 않습니다.

   AWS Glue `columnMapping` 테이블 속성을 추가한 후 `disable_projection_and_casing` Lambda 환경 변수를 제거할 수 있습니다.
4. 이 문서에 나열된 대로 AWS Glue에 적합한 데이터 형식을 사용해야 합니다.

데이터 형식 지원

이 섹션에는 DocumentDB 커넥터가 스키마 추론에 사용하는 데이터 형식과 AWS Glue 메타데이터가 사용될 때의 데이터 형식이 나열되어 있습니다.

스키마 추론 데이터 형식

DocumentDB 커넥터의 스키마 추론 기능은 값을 다음 데이터 형식 중 하나에 속하는 것으로 추정하려고 시도합니다. 다음 표에는 Amazon DocumentDB, Java 및 Apache Arrow에 해당하는 데이터 형식이 나와 있습니다.

Apache Arrow	Java 또는 DocDB
VARCHAR	String
INT	Integer

Apache Arrow	Java 또는 DocDB
BIGINT	Long
BIT	불
FLOAT4	Float
FLOAT8	Double
TIMESTAMPSEC	날짜
VARCHAR	ObjectId
LIST	나열
STRUCT	문서

AWS Glue 데이터 유형

보충 메타데이터에 AWS Glue를 사용하는 경우 다음 데이터 형식을 구성할 수 있습니다. 다음 표에는 AWS Glue와 Apache Arrow의 해당 데이터 형식이 나와 있습니다.

AWS Glue	Apache Arrow
int	INT
bigint	BIGINT
double	FLOAT8
float	FLOAT4
boolean	BIT
이진수	VARBINARY
문자열	VARCHAR
나열	LIST

AWS Glue	Apache Arrow
구조체	STRUCT

필수 권한

이 커넥터에 필요한 IAM 정책에 대한 자세한 내용을 알아보려면 athena-docdb.yaml 파일의 `Policies` 섹션을 검토하세요. 다음 목록에 필요한 권한이 요약되어 있습니다.

- Amazon S3 쓰기 액세스 - 대규모 쿼리의 결과 유출을 위해서는 커넥터에 Amazon S3 위치에 대한 쓰기 액세스 권한이 필요합니다.
- Athena GetQueryExecution - 커넥터는 업스트림 Athena 쿼리가 종료된 경우 이 권한을 사용하여 빠른 실패를 수행합니다.
- AWS Glue Data Catalog - 스키마 정보를 가져오기 위해 DocumentDB 커넥터에 AWS Glue Data Catalog에 대한 읽기 전용 액세스 권한이 필요합니다.
- CloudWatch Logs - 로그를 저장하기 위해 커넥터 CloudWatch Logs에 대한 액세스 권한이 필요합니다.
- AWS Secrets Manager 읽기 액세스 - Secrets Manager에 DocumentDB 엔드포인트 세부 정보를 저장하기로 선택하는 경우, 커넥터에 해당 보안 암호에 대한 액세스 권한을 부여해야 합니다.
- VPC 액세스 - VPC에 연결하고 DocumentDB 인스턴스와 통신할 수 있도록 VPC에 인터페이스를 연결하고 분리하는 기능이 커넥터에 필요합니다.

성능

Athena Amazon DocumentDB 커넥터는 현재 병렬 스캔을 지원하지 않지만 DocumentDB 쿼리의 일부로 조건자 푸시다운을 시도하고, DocumentDB 컬렉션의 인덱스에 대한 조건자로 인해 스캔되는 데이터가 크게 감소합니다.

쿼리에서 스캔된는 데이터를 줄이기 위해 Lambda 함수에서 조건자 푸시다운을 수행합니다. 그러나 열의 하위 세트를 선택하면 쿼리 실행 런타임이 길어지는 경우가 있습니다. `LIMIT` 절은 스캔되는 데이터의 양을 줄이지만 조건자를 제공하지 않으면 `LIMIT` 절을 포함하는 `SELECT` 쿼리가 최소 16MB의 데이터를 스캔할 것으로 예상해야 합니다.

패스스루 쿼리

Athena Amazon DocumentDB 커넥터는 패스스루 쿼리를 지원하며 NoSQL 기반입니다. Amazon DocumentDB 쿼리에 대한 자세한 내용은 Amazon DocumentDB 개발자 가이드의 쿼리를 참조하세요.

Amazon DocumentDB에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            database => 'database_name',
            collection => 'collection_name',
            filter => '{query_syntax}'
        ))
```

다음 예제 쿼리는 TPCDS 컬렉션 내에서 example 데이터베이스를 쿼리하여 제목이 Bill of Rights인 모든 책을 필터링합니다.

```
SELECT * FROM TABLE(
        system.query(
            database => 'example',
            collection => 'tpcds',
            filter => '{title: "Bill of Rights"}'
        ))
```

추가적인 리소스

- Amazon Athena 페더레이션된 쿼리를 사용하여 MongoDB 데이터베이스를 Amazon QuickSight에 연결해 대시보드 및 시각화를 생성하는 방법에 대한 문서는 AWS 빅 데이터 블로그의 Visualize MongoDB data from Amazon QuickSight using Amazon Athena Federated Query를 참조하세요.
- 이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena DynamoDB 커넥터

Amazon DynamoDB 커넥터를 통해 Amazon Athena는 DynamoDB와 통신할 수 있고, 이로써 SQL을 사용하여 테이블을 쿼리할 수 있습니다. INSERT INTO 같은 쓰기 작업은 지원되지 않습니다.

계정에서 Lake Formation을 활성화한 경우 AWS Serverless Application Repository에 배포한 Athena 페더레이션형 Lambda 커넥터의 IAM 역할은 Lake Formation에서 AWS Glue Data Catalog에 대한 읽기 액세스 권한을 가지고 있어야 합니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 DynamoDB 커넥터를 구성합니다.

- spill_bucket – Lambda 함수 제한을 초과하는 데이터에 대한 Amazon S3 버킷을 지정합니다.
- spill_prefix – (선택 사항) 기본값은 `athena-federation-spill`이라는 지정된 `spill_bucket`의 하위 폴더입니다. 미리 정해진 일 수 또는 시간보다 오래된 유출을 삭제하려면 이 위치에서 Amazon S3 스토리지 수명 주기를 구성하는 것이 좋습니다.
- spill_put_request_headers – (선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵입니다(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.
- kms_key_id – (선택 사항) 기본적으로 Amazon S3로 유출된 모든 데이터는 AES-GCM 인증 암호화 모드와 임의로 생성된 키를 사용하여 암호화됩니다. Lambda 함수가 `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`과 같이 KMS에서 생성된 더 강력한 암호화 키를 사용하도록 하려면 KMS 키 ID를 지정합니다.
- disable_spill_encryption – (선택 사항) `True`로 설정하면 유출 암호화가 비활성화됩니다. S3로 유출되는 데이터가 AES-GCM을 사용하여 암호화되도록 기본값은 `False`입니다(임의로 생성된 키 또는 KMS를 사용하여 키 생성). 유출 암호화를 비활성화하면 특히 유출 위치가 서버 측 암호화를 사용하는 경우 성능이 향상될 수 있습니다.
- disable_glue – (선택 사항) true로 설정된 경우 커넥터는 AWS Glue에서 보충 메타데이터 검색을 시도하지 않습니다.
- glue_catalog – (선택 사항) 이 옵션을 사용하여 크로스 계정 AWS Glue 카탈로그를 지정합니다. 기본적으로 커넥터는 자체 AWS Glue 계정에서 메타데이터를 가져오려고 시도합니다.
- disable_projection_and_casing – (선택 사항) 프로젝션과 대/소문자를 비활성화합니다. 열 이름에 대/소문자가 있는 DynamoDB 테이블을 쿼리하고 AWS Glue 테이블에 `columnMapping` 속성을 지정하지 않으려는 경우 사용합니다.

  `disable_projection_and_casing` 파라미터는 다음 값을 사용하여 대/소문자 및 열 매핑의 동작을 지정합니다.

  - auto(자동) - 이전에 지원되지 않는 형식이 감지되고 테이블에 열 이름 매핑이 설정되지 않은 경우 프로젝션과 대/소문자를 비활성화합니다. 이것이 기본 설정입니다.
  - always(항상) - 프로젝션과 대/소문자를 무조건 비활성화합니다. 이는 DynamoDB 열 이름에 대/소문자가 있지만 열 이름 매핑을 지정하지 않으려는 경우에 유용합니다.

  `disable_projection_and_casing` 파라미터를 사용할 때는 다음 사항에 유의하세요.

- 파라미터를 사용하면 대역폭 사용량이 증가할 수 있습니다. 또한 Lambda 함수가 데이터 소스와 동일한 AWS 리전에 있지 않을 경우 대역폭 사용량이 증가하여 더 높은 표준 AWS 리전 간 전송 비용이 청구됩니다. 리전 간 전송 비용에 대한 자세한 내용은 AWS 파트너 네트워크 블로그의 서버 및 서버리스 아키텍처에 대한 AWS 데이터 전송 요금을 참조하세요.
- 더 많은 수의 바이트가 전송되고 더 많은 수의 바이트가 더 많은 역직렬화 시간을 필요로 하기 때문에 전체 지연 시간이 늘어날 수 있습니다.

AWS Glue에서 데이터베이스 및 테이블 설정

커넥터의 기본 제공 스키마 추론 기능이 제한되어 있으므로 메타데이터에 AWS Glue를 사용할 수 있습니다. 이렇게 하려면 AWS Glue에 데이터베이스와 테이블이 있어야 합니다. DynamoDB에서 사용할 수 있도록 하려면 해당 속성을 편집해야 합니다.

AWS Glue 콘솔에서 데이터베이스 속성을 편집하려면 다음을 수행하세요.

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
2. Databases(데이터베이스) 탭을 선택합니다.

   Databases(데이터베이스) 페이지에서 기존 데이터베이스를 편집하거나 Add database(데이터베이스 추가)를 선택하여 데이터베이스를 생성할 수 있습니다.
3. 데이터베이스 목록에서 편집하려는 데이터베이스의 링크를 선택합니다.
4. 편집을 선택합니다.
5. Update a database)데이터베이스 업데이트) 페이지에서 Location(위치)에 문자열 **dynamo-db-flag**를 추가합니다. 이 키워드는 Athena DynamoDB 커넥터가 보충 메타데이터에 사용하고 `default` 이외의 AWS Glue 데이터베이스에 필요한 테이블이 데이터베이스에 포함되어 있음을 나타냅니다. `dynamo-db-flag` 속성은 데이터베이스가 많은 계정에서 데이터베이스를 필터링하는 데 유용합니다.

AWS Glue 콘솔에서 테이블 속성을 편집하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
2. 테이블 탭을 선택합니다.

테이블 탭에서 기존 테이블을 편집합니다. 수동으로 또는 크롤러를 사용하여 테이블을 추가하는 방법에 대한 자세한 내용은 AWS Glue 개발자 안내서의 AWS Glue 콘솔에서 테이블 관련 작업을 참조하세요.

3. 테이블 목록에서 편집하려는 테이블에 대한 링크를 선택합니다.
4. Actions(작업), Edit table(테이블 편집)을 선택합니다.
5. Edit table(테이블 편집) 페이지의 Table properties(테이블 속성) 섹션에서 필요에 따라 다음 테이블 속성을 추가합니다. AWS Glue DynamoDB 크롤러를 사용하는 경우 이러한 속성이 자동으로 설정됩니다.

   - dynamodb – 테이블을 보충 메타데이터에 사용할 수 있음을 Athena DynamoDB 커넥터에 나타내는 문자열입니다. classification(분류)이라는 필드 아래의 테이블 속성에 dynamodb를 입력합니다(정확히 일치).

     > **Note**
     >
     > AWS Glue 콘솔의 테이블 작성 프로세스의 일부인 테이블 속성 설정 페이지에는 분류 필드가 있는 데이터 형식 섹션이 있습니다. 여기서는 dynamodb를 입력하거나 선택할 수 없습니다. 대신, 테이블을 생성한 후 테이블을 편집하고 테이블 속성 섹션에서 `classification`과 dynamodb를 키-값 페어로 입력하는 단계를 수행합니다.

   - sourceTable – DynamoDB에서 소스 테이블 이름을 정의하는 선택적 테이블 속성입니다. AWS Glue 테이블 이름 지정 규칙으로 인해 DynamoDB 테이블과 동일한 이름으로 AWS Glue 테이블을 생성할 수 없는 경우 이 옵션을 사용합니다. 예를 들어, AWS Glue 테이블 이름에는 대문자가 허용되지 않지만 DynamoDB 테이블 이름에는 대문자가 허용됩니다.
   - columnMapping – 열 이름 매핑을 정의하는 선택적 테이블 속성입니다. AWS Glue 열 이름 지정 규칙으로 인해 DynamoDB 테이블과 동일한 열 이름으로 AWS Glue 테이블을 생성할 수 없는 경우 이 옵션을 사용합니다. 예를 들어, AWS Glue 열 이름에는 대문자가 허용되지 않지만 DynamoDB 열 이름에는 대문자가 허용됩니다. 속성 값은 col1=Col1,col2=Col2 형식이어야 합니다. 단, 열 매핑은 최상위 열 이름에만 적용되고 중첩된 필드에는 적용되지 않습니다.
   - defaultTimeZone – 명시적 표준 시간대가 없는 `date` 또는 `datetime` 값에 적용되는 선택적 테이블 속성입니다. 이 값을 설정하면 데이터 소스 기본 표준 시간대와 Athena 세션 시간대 간의 불일치를 방지하는 것이 좋습니다.
   - datetimeFormatMapping – AWS Glue `date` 또는 `timestamp` 데이터 형식의 열에서 데이터를 구문 분석할 때 사용할 `date` 또는 `datetime` 형식을 지정하는 선택적 테이블 속성입니다. 이 속성을 지정하지 않으면 커넥터는 ISO-8601 형식을 추론하려고 시도합니다. 커넥터가 `date` 또

는 datetime 형식을 추론할 수 없거나 원시 문자열을 구문 분석할 수 없는 경우 결과에서 값이 생략됩니다.

datetimeFormatMapping 값은 col1=someformat1,col2=someformat2 형식이어야 합니다. 다음은 몇 가지 예제 형식입니다.

```
yyyyMMdd'T'HHmmss
ddMMyyyy'T'HH:mm:ss
```

열에 표준 시간대가 없는 date 또는 datetime 값이 있고 WHERE 절의 열을 사용하려는 경우 열에 대한 datetimeFormatMapping 속성을 설정합니다.

6. 열을 수동으로 정의하는 경우 적절한 데이터 형식을 사용하는지 확인합니다. 크롤러를 사용한 경우 크롤러가 검색한 열과 유형을 검증합니다.

필수 권한

이 커넥터에 필요한 IAM 정책에 대한 자세한 내용을 알아보려면 athena-dynamodb.yaml 파일의 Policies 섹션을 검토하세요. 다음 목록에 필요한 권한이 요약되어 있습니다.

- Amazon S3 쓰기 액세스 - 대규모 쿼리의 결과 유출을 위해서는 커넥터에 Amazon S3 위치에 대한 쓰기 액세스 권한이 필요합니다.
- Athena GetQueryExecution - 커넥터는 업스트림 Athena 쿼리가 종료된 경우 이 권한을 사용하여 빠른 실패를 수행합니다.
- AWS Glue Data Catalog - 스키마 정보를 가져오기 위해 DynamoDB 커넥터에 AWS Glue Data Catalog에 대한 읽기 전용 액세스 권한이 필요합니다.
- CloudWatch Logs - 로그를 저장하기 위해 커넥터 CloudWatch Logs에 대한 액세스 권한이 필요합니다.
- DynamoDB 읽기 액세스 – 커넥터는 DescribeTable, ListSchemas, ListTables, Query 및 Scan API 작업을 사용합니다.

성능

Athena DynamoDB 커넥터는 병렬 스캔을 지원하고 DynamoDB 쿼리의 일부로 조건자 푸시다운을 시도합니다. X개의 고유 값이 있는 해시 키 조건자는 DynamoDB에 대한 X개의 쿼리 호출을 생성합니다. 다른 모든 조건자 시나리오에서는 Y개의 스캔 호출이 발생합니다. 여기서 Y는 테이블 크기와 프로비저닝된 처리량을 기반으로 경험적으로 결정됩니다. 그러나 열 하위 세트를 선택할 때 쿼리 실행 런타임이 길어지는 경우가 있습니다.

LIMIT 절 및 간단한 조건자가 푸시다운되면 스캔하는 데이터가 줄어들 수 있으며 이에 따라 쿼리 실행 런타임이 감소할 수 있습니다.

LIMIT 절

LIMIT N 문은 쿼리로 스캔하는 데이터를 줄입니다. LIMIT N 푸시다운을 통해 커넥터는 Athena에 N개 행만 반환합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 WHERE 절에 사용되는 표현식입니다. Athena DynamoDB 커넥터는 이러한 표현식을 결합하고 DynamoDB로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena DynamoDB 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_NULL

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10 and col_b < 10
LIMIT 10
```

DynamoDB를 비롯한 페더레이션된 쿼리의 성능을 개선하기 위해 조건자 푸시다운을 사용하는 방법에 대한 문서는 AWS 빅 데이터 블로그의 Improve federated queries with predicate pushdown in Amazon Athena를 참조하세요.

패스스루 쿼리

DynamoDB 커넥터는 패스스루 쿼리를 지원하고 PartiQL 구문을 사용합니다. DynamoDB GetItem API 작업은 지원되지 않습니다. PartiQL을 사용하여 DynamoDB를 쿼리하는 방법에 대한 자세한 내용은 Amazon DynamoDB 개발자 안내서의 DynamoDB용 PartiQL select 문을 참조하세요.

DynamoDB에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query_string'
        ))
```

다음 DynamoDB 패스스루 쿼리 예제는 PartiQL을 사용하여 `DateWatched` 속성이 12/24/22 이후인 Fire TV Stick 디바이스 목록을 반환합니다.

```
SELECT * FROM TABLE(
        system.query(
           query => 'SELECT Devices
                       FROM WatchList
                       WHERE Devices.FireStick.DateWatched[0] > '12/24/22''
        ))
```

문제 해결

정렬 키 열의 다중 필터

오류 메시지: KeyConditionExpressions must only contain one condition per key

원인: 이 문제는 Athena 엔진 버전 3의 경우 DynamoDB 정렬 키 열에 하한 및 상한 필터가 모두 있는 쿼리에서 발생할 수 있습니다. DynamoDB는 정렬 키에 대해 둘 이상의 필터 조건을 지원하지 않으므로 커넥터가 두 조건이 모두 적용된 쿼리를 푸시다운하려고 하면 오류가 발생합니다.

솔루션: 커넥터를 버전 2023.11.1 이상으로 업데이트합니다. 커넥터 업데이트에 대한 지침은 데이터 소스 커넥터 업데이트 섹션을 참조하세요.

비용

커넥터 사용 비용은 사용되는 기본 AWS 리소스에 따라 다릅니다. 스캔을 사용하는 쿼리는 많은 읽기 용량 단위(RCU)를 사용할 수 있으므로 Amazon DynamoDB 요금 정보를 신중하게 고려하세요.

추가적인 리소스

- Amazon Athena DynamoDB 커넥터 사용에 대한 소개는 AWS Prescriptive Guidance Patterns 가이드의 Access, query, and join Amazon DynamoDB tables using Athena를 참조하세요.
- Amazon DynamoDB, Athena 및 Amazon QuickSight에서 Amazon Athena DynamoDB 커넥터를 사용하여 간단한 거버넌스 대시보드를 생성하는 방법에 대한 문서는 AWS 빅 데이터 블로그의 Query cross-account Amazon DynamoDB tables using Amazon Athena Federated Query를 참조하세요.
- 이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Google BigQuery 커넥터

Google BigQuery용 Amazon Athena 커넥터를 사용하면 Amazon Athena가 Google BigQuery 데이터에 대해 SQL 쿼리를 실행할 수 있습니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

제한 사항

- Lambda 함수의 최대 제한 시간 값은 15분입니다. 각 분할은 BigQuery에서 쿼리를 실행하며 Athena에서 읽을 결과를 저장할 수 있는 충분한 시간을 두고 완료해야 합니다. Lambda 함수가 시간 초과되면 쿼리가 실패합니다.
- Google BigQuery는 대/소문자를 구분합니다. 커넥터는 데이터세트 이름과 테이블 이름의 대/소문자를 수정하려고 시도하지만 프로젝트 ID에 대해서는 대/소문자를 수정하지 않습니다. 이는 Athena가 모든 메타데이터를 소문자로 만들기 때문에 필요합니다. 이러한 수정으로 인해 Google BigQuery에 대한 추가 호출이 많이 발생합니다.
- 바이너리 데이터 형식은 지원되지 않습니다.
- Google BigQuery 동시성 및 할당량 제한으로 인해 커넥터에 Google 할당량 제한 문제가 발생할 수 있습니다. 이러한 문제를 방지하려면 Google BigQuery에 최대한 많은 제약 조건을 푸시합니다. BigQuery 할당량에 대한 자세한 내용을 알아보려면 Google BigQuery 설명서의 할당량 및 한도를 참조하세요.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 Google BigQuery 커넥터를 구성합니다.

- spill_bucket – Lambda 함수 제한을 초과하는 데이터에 대한 Amazon S3 버킷을 지정합니다.
- spill_prefix – (선택 사항) 기본값은 `athena-federation-spill`이라는 지정된 `spill_bucket`의 하위 폴더입니다. 미리 정해진 일 수 또는 시간보다 오래된 유출을 삭제하려면 이 위치에서 Amazon S3 스토리지 수명 주기를 구성하는 것이 좋습니다.
- spill_put_request_headers – (선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵입니다(예: `{"x-amz-server-side-encryption" :`

"AES256"}). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

- kms_key_id – (선택 사항) 기본적으로 Amazon S3로 유출된 모든 데이터는 AES-GCM 인증 암호화 모드와 임의로 생성된 키를 사용하여 암호화됩니다. Lambda 함수가 a7e63k4b-8loc-40db-a2a1-4d0en2cd8331과 같이 KMS에서 생성된 더 강력한 암호화 키를 사용하도록 하려면 KMS 키 ID를 지정합니다.
- disable_spill_encryption – (선택 사항) True로 설정하면 유출 암호화가 비활성화됩니다. S3로 유출되는 데이터가 AES-GCM을 사용하여 암호화되도록 기본값은 False입니다(임의로 생성된 키 또는 KMS를 사용하여 키 생성). 유출 암호화를 비활성화하면 특히 유출 위치가 서버 측 암호화를 사용하는 경우 성능이 향상될 수 있습니다.
- gcp_project_id – 커넥터가 읽어야 하는 데이터 세트를 포함하는 프로젝트 ID(프로젝트 이름 아님)입니다(예: semiotic-primer-1234567).
- secret_manager_gcp_creds_name – JSON 형식의 BigQuery 자격 증명을 포함하는 AWS Secrets Manager 내의 보안 암호 이름입니다(예: GoogleCloudPlatformCredentials).
- big_query_endpoint - (선택 사항) BigQuery 프라이빗 엔드포인트의 URL입니다. 프라이빗 엔드포인트를 통해 BigQuery에 액세스하려는 경우 이 파라미터를 사용합니다.

분할 및 뷰

BigQuery 커넥터는 BigQuery Storage Read API를 사용하여 테이블을 쿼리하고, BigQuery Storage API는 뷰를 지원하지 않기 때문에 커넥터는 뷰의 단일 분할에 BigQuery 클라이언트를 사용합니다.

성능

테이블 쿼리를 위해 BigQuery 커넥터는 BigQuery 관리 스토리지에 빠르게 액세스할 수 있는 RPC 기반 프로토콜을 사용하는 BigQuery Storage Read API를 사용합니다. BigQuery 스토리지 읽기 API에 대한 자세한 내용은 Google Cloud 문서에서 Use the BigQuery Storage Read API to read table data를 참조하세요.

열의 하위 집합을 선택하면 쿼리 런타임 속도를 높이고 스캔되는 데이터를 줄일 수 있습니다. 동시성이 증가함에 따라 커넥터에서 쿼리 오류가 발생할 수 있으며 일반적으로 커넥터가 느립니다.

Athena Google BigQuery 커넥터는 조건부 푸시다운을 수행하여 쿼리에서 스캔하는 데이터를 줄입니다. LIMIT 절, ORDER BY 절, 간단한 조건자 및 복잡한 표현식을 커넥터로 푸시다운하여 스캔하는 데이터와 쿼리 실행 시간을 줄입니다.

LIMIT 절

`LIMIT N` 문은 쿼리로 스캔하는 데이터를 줄입니다. `LIMIT N` 푸시다운을 통해 커넥터는 Athena에 `N`개 행만 반환합니다.

상위 N개 쿼리

상위 `N`개 쿼리는 결과 세트의 순서와 반환되는 행 수에 대한 한도를 지정합니다. 이 유형의 쿼리를 사용하여 데이터 세트에 대해 상위 `N`개의 최댓값 또는 상위 `N`개의 최솟값을 결정할 수 있습니다. 상위 `N`개의 푸시다운을 통해 커넥터는 Athena에 `N`개의 정렬된 행만 반환합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 `WHERE` 절에 사용되는 표현식입니다. Athena Google BigQuery 커넥터는 이러한 표현식을 결합하고 Google BigQuery로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena Google BigQuery 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND, OR, NOT
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL
- 산술: ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- 기타: LIKE_PATTERN, IN

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
ORDER BY col_a DESC
LIMIT 10;
```

패스스루 쿼리

Google BigQuery 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

Google BigQuery에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 Google BigQuery의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 `customer` 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

라이선스 정보

Amazon Athena Google BigQuery 커넥터 프로젝트는 Apache-2.0 라이선스에 따라 사용이 허가됩니다.

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Google Cloud Storage 커넥터

Amazon Athena Google Cloud Storage 커넥터를 사용하면 Amazon Athena가 Google Cloud Storage(GCS) 버킷에 저장된 Parquet 및 CSV 파일에 대해 쿼리를 실행할 수 있습니다. GCS 버킷의 분할되지 않은 폴더나 분할된 폴더에서 하나 이상의 Parquet 또는 CSV 파일을 그룹화한 후 AWS Glue 데이터베이스 테이블에서 구성할 수 있습니다.

계정에서 Lake Formation을 활성화한 경우 AWS Serverless Application Repository에 배포한 Athena 페더레이션형 Lambda 커넥터의 IAM 역할은 Lake Formation에서 AWS Glue Data Catalog에 대한 읽기 액세스 권한을 가지고 있어야 합니다.

필수 조건

- Google Cloud Storage의 버킷 및 폴더에 해당하는 AWS Glue 데이터베이스 및 테이블을 설정합니다. 단계는 이 문서 뒷부분의 AWS Glue에서 데이터베이스 및 테이블 설정 섹션을 참조하세요.
- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.
- 현재 커넥터는 파티션 열(AWS Glue 테이블 스키마의 `string` 또는 `varchar`)에 대해 VARCHAR 유형만 지원합니다. 다른 파티션 필드 유형은 Athena에서 쿼리할 때 오류가 발생합니다.

용어

다음 용어는 GCS 커넥터와 관련이 있습니다.

- 핸들러 - GCS 버킷에 액세스하는 Lambda 핸들러입니다. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - GCS 버킷에서 메타데이터를 검색하는 Lambda 핸들러입니다.
- 레코드 핸들러 - GCS 버킷에서 데이터 레코드를 검색하는 Lambda 핸들러입니다.
- 복합 핸들러 - GCS 버킷에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러입니다.

지원되는 파일 형식

GCS 커넥터는 Parquet 및 CSV 파일 형식을 지원합니다.

> **Note**
>
> 동일한 GCS 버킷 또는 경로에 CSV 및 Parquet 파일을 모두 배치하지 않도록 합니다. 그렇게 하면 Parquet 파일을 CSV로 읽거나 그 반대로 읽으려고 할 때 런타임 오류가 발생할 수 있습니다.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 GCS 커넥터를 구성합니다.

- spill_bucket – Lambda 함수 제한을 초과하는 데이터에 대한 Amazon S3 버킷을 지정합니다.
- spill_prefix – (선택 사항) 기본값은 `athena-federation-spill`이라는 지정된 `spill_bucket`의 하위 폴더입니다. 미리 정해진 일 수 또는 시간보다 오래된 유출을 삭제하려면 이 위치에서 Amazon S3 스토리지 수명 주기를 구성하는 것이 좋습니다.
- spill_put_request_headers – (선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵입니다(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.
- kms_key_id – (선택 사항) 기본적으로 Amazon S3로 유출된 모든 데이터는 AES-GCM 인증 암호화 모드와 임의로 생성된 키를 사용하여 암호화됩니다. Lambda 함수가 `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`과 같이 KMS에서 생성된 더 강력한 암호화 키를 사용하도록 하려면 KMS 키 ID를 지정합니다.
- disable_spill_encryption – (선택 사항) `True`로 설정하면 유출 암호화가 비활성화됩니다. S3로 유출되는 데이터가 AES-GCM을 사용하여 암호화되도록 기본값은 `False`입니다(임의로 생성된 키 또는 KMS를 사용하여 키 생성). 유출 암호화를 비활성화하면 특히 유출 위치가 서버 측 암호화를 사용하는 경우 성능이 향상될 수 있습니다.
- secret_manager_gcp_creds_name – JSON 형식의 GCS 자격 증명을 포함하는 AWS Secrets Manager 내의 보안 암호 이름입니다(예: `GoogleCloudPlatformCredentials`).

AWS Glue에서 데이터베이스 및 테이블 설정

GCS 커넥터의 기본 제공 스키마 추론 기능은 제한적이기 때문에 메타데이터에 AWS Glue를 사용하는 것이 좋습니다. 다음 절차는 Athena에서 액세스할 수 있는 AWS Glue에서 데이터베이스와 테이블을 생성하는 방법을 보여줍니다.

AWS Glue에서 데이터베이스 생성

AWS Glue 콘솔을 사용하여 GCS 커넥터와 함께 사용할 데이터베이스를 생성할 수 있습니다.

AWS Glue 데이터베이스 사용자를 생성하려면 다음을 수행하세요.

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다.

3. 데이터베이스 추가(Add database)를 선택합니다.
4. Name(이름)에 GCS 커넥터와 함께 사용할 데이터베이스 이름을 입력합니다.
5. Location(위치)에 `s3://google-cloud-storage-flag.`를 지정합니다. 이 위치는 AWS Glue 데이터베이스에 Athena에서 쿼리할 GCS 데이터용 테이블이 포함되어 있음을 GCS 커넥터에 알립니다. 커넥터는 Athena에서 이 플래그가 있는 데이터베이스를 인식하고 그렇지 않은 데이터베이스는 무시합니다.
6. 데이터베이스 생성를 선택합니다.

AWS Glue에서 테이블 생성

이제 데이터베이스용 테이블을 생성할 수 있습니다. GCS 커넥터와 함께 사용할 AWS Glue 테이블을 생성할 때 추가 메타데이터를 지정해야 합니다.

AWS Glue 콘솔에서 테이블을 생성하려면 다음을 수행하세요.

1. AWS Glue 콘솔의 탐색 창에서 Tables(테이블)를 선택합니다.
2. Tables(테이블) 페이지에서 Add table(테이블 추가)을 선택합니다.
3. Set table properties(테이블 속성 설정) 페이지에 다음 정보를 입력합니다.
   - Name(이름) – 테이블의 고유한 이름입니다.
   - Database(데이터베이스) – GCS 커넥터용으로 생성한 AWS Glue 데이터베이스를 선택합니다.
   - Include path(포함 경로) – Data store(데이터 스토어) 섹션에서 Include path(포함 경로)에 `gs://`로 시작하는 GCS의 URI 위치를 입력합니다(예: `gs://gcs_table/data/`). 파티션 폴더가 하나 이상 있는 경우 해당 폴더를 경로에 포함하지 마세요.

     > **Note**
     >
     > `s3://`가 아닌 테이블 경로를 입력하면 AWS Glue 콘솔에 오류가 표시됩니다. 이 오류는 무시할 수 있습니다. 테이블이 성공적으로 생성됩니다.

   - Data format(데이터 형식) – Classification(분류)에서 CSV 또는 Parquet를 선택합니다.
4. [Next]를 선택합니다.
5. Choose or define schema(스키마 선택 또는 정의) 페이지에서 테이블 스키마를 정의하는 것이 좋지만 필수는 아닙니다. 스키마를 정의하지 않으면 GCS 커넥터가 스키마 추론을 시도합니다.

   다음 중 하나를 수행하십시오.

- GCS 커넥터가 스키마 추론을 시도하도록 하려면 Next(다음)를 선택한 다음 Create(생성)를 선택합니다.
- 스키마를 직접 정의하려면 다음 섹션의 단계를 따르세요.

AWS Glue에서 테이블 스키마 정의

AWS Glue에서 테이블 스키마를 정의하려면 더 많은 단계가 필요하지만 테이블 생성 프로세스를 더 잘 제어할 수 있습니다.

AWS Glue에서 테이블의 스키마를 정의하려면 다음을 수행하세요.

1. Choose or define schema(스키마 선택 또는 정의) 페이지에서 Add(추가)를 선택합니다.
2. Add schema entry(스키마 항목 추가) 대화 상자를 사용하여 열 이름과 데이터 유형을 제공할 수 있습니다.
3. 열을 파티션 열로 지정하려면 Set as partition key(파티션 키로 설정) 옵션을 선택합니다.
4. Save(저장)를 선택하여 열을 저장합니다.
5. Add(추가)를 선택하여 열을 추가합니다.
6. 열 추가를 마치면 Next(다음)를 선택합니다.
7. Review and create(검토 및 생성) 페이지에서 테이블을 검토한 다음 Create(생성)를 선택합니다.
8. 스키마에 파티션 정보가 포함된 경우 다음 섹션의 단계에 따라 AWS Glue에서 테이블 속성에 파티션 패턴을 추가합니다.

AWS Glue에서 테이블 속성에 파티션 패턴 추가

GCS 버킷에 파티션이 있는 경우 AWS Glue에서 테이블 속성에 파티션 패턴을 추가해야 합니다.

AWS Glue에서 테이블 속성에 파티션 정보를 추가하려면 다음을 수행하세요.

1. AWS Glue에서 생성한 테이블의 세부 정보 페이지에서 Actions(작업), Edit table(테이블 편집)을 선택합니다.
2. Edit table(테이블 편집) 페이지에서 Table properties(테이블 속성) 세션이 나타날 때까지 아래로 스크롤합니다.
3. Add(추가)를 선택하여 파티션 키를 추가합니다.
4. 키에 **`partition.pattern`**를 입력합니다. 이 키는 폴더 경로 패턴을 정의합니다.

5. Value(값)에 **StateName=${statename}/ZipCode=${zipcode}/**와 같은 폴더 경로 패턴을 입력합니다. 여기서 **${}**로 묶인 **statename** 및 **zipcode**는 파티션 열 이름입니다. GCS 커넥터는 Hive 및 비Hive 파티션 구성표를 모두 지원합니다.
6. 작업을 마쳤으면 저장을 선택합니다.
7. 방금 생성한 테이블 속성을 보려면 Advanced properties(고급 속성) 탭을 선택합니다.

이제 Athena 콘솔로 이동할 수 있습니다. AWS Glue에서 생성한 데이터베이스와 테이블은 Athena에서 쿼리할 수 있습니다.

데이터 형식 지원

다음 표에 CSV 및 Parquet에 대해 지원되는 데이터 형식이 나와 있습니다.

CSV

데이터의 특성	추론된 데이터 형식
데이터가 숫자처럼 보임	BIGINT
데이터가 문자열처럼 보임	VARCHAR
데이터가 부동 소수점(float, double 또는 decimal)처럼 보임	DOUBLE
데이터가 날짜처럼 보임	Timestamp
데이터에 true/false 값이 포함됨	BOOL

PARQUET

PARQUET	Athena(Arrow)
BINARY	VARCHAR
BOOLEAN	BOOL
DOUBLE	DOUBLE

PARQUET	Athena(Arrow)
ENUM	VARCHAR
FIXED_LEN _BYTE_ARRAY	DECIMAL
FLOAT	FLOAT(32비트)
INT32	1. INT32 2. DATEDAY(Parquet 열 논리적 유형이 DATE인 경우)
INT64	1. INT64 2. TIMESTAMP(Parquet 열 논리적 유형이 TIMESTAMP인 경우)
INT96	Timestamp
MAP	MAP
STRUCT	STRUCT
LIST	LIST

필수 권한

이 커넥터에 필요한 IAM 정책에 대한 자세한 내용을 알아보려면 athena-gcs.yaml 파일의 `Policies` 섹션을 검토하세요. 다음 목록에 필요한 권한이 요약되어 있습니다.

- Amazon S3 쓰기 액세스 - 대규모 쿼리의 결과 유출을 위해서는 커넥터에 Amazon S3 위치에 대한 쓰기 액세스 권한이 필요합니다.
- Athena GetQueryExecution - 커넥터는 업스트림 Athena 쿼리가 종료된 경우 이 권한을 사용하여 빠른 실패를 수행합니다.
- AWS Glue Data Catalog - 스키마 정보를 가져오기 위해 GCS 커넥터에 AWS Glue Data Catalog에 대한 읽기 전용 액세스 권한이 필요합니다.
- CloudWatch Logs - 로그를 저장하기 위해 커넥터 CloudWatch Logs에 대한 액세스 권한이 필요합니다.

성능

테이블 스키마에 파티션 필드가 포함되어 있고 `partition.pattern` 테이블 속성이 올바르게 구성된 경우 쿼리의 `WHERE` 절에 파티션 필드를 포함할 수 있습니다. 이러한 쿼리의 경우 GCS 커넥터는 파티션 열을 사용하여 GCS 폴더 경로를 구체화하고 GCS 폴더에서 불필요한 파일을 스캔하지 않도록 합니다.

Parquet 데이터 세트의 경우 열 하위 세트를 선택하면 스캔되는 데이터가 줄어듭니다. 이로 인해 일반적으로 열 프로젝션이 적용될 때 쿼리 실행 런타임이 짧아집니다.

CSV 데이터 세트의 경우 열 프로젝션이 지원되지 않으며 스캔되는 데이터의 양을 줄이지 않습니다.

`LIMIT` 절은 스캔되는 데이터의 양을 줄이지만 조건자를 제공하지 않으면 `LIMIT` 절을 포함하는 `SELECT` 쿼리가 최소 16MB의 데이터를 스캔할 것으로 예상해야 합니다. GCS 커넥터는 적용된 `LIMIT` 절에 관계없이 작은 데이터 세트보다 큰 데이터 세트에 대해 더 많은 데이터를 스캔합니다. 예를 들어, 쿼리 `SELECT * LIMIT 10000`은 더 작은 기본 데이터 세트보다 더 큰 기본 데이터 세트에 대해 더 많은 데이터를 스캔합니다.

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena HBase 커넥터

Amazon Athena HBase 커넥터를 통해 Amazon Athena는 Apache HBase 인스턴스와 통신할 수 있고, 이로써 SQL을 사용하여 HBase 데이터를 쿼리할 수 있습니다.

기존의 관계형 데이터 스토어와 달리 HBase 컬렉션에는 설정된 스키마가 없습니다. HBase에는 메타데이터 스토어가 없습니다. HBase 컬렉션의 항목마다 필드와 데이터 형식이 다를 수 있습니다.

HBase 커넥터는 테이블 스키마 정보를 생성하기 위한 두 가지 메커니즘인 기본 스키마 추론과 AWS Glue Data Catalog 메타데이터를 지원합니다.

스키마 추론이 기본값입니다. 이 옵션은 컬렉션에 있는 소수의 문서를 스캔하고, 모든 필드의 통합을 형성하고, 겹치지 않는 데이터 형식을 가진 필드를 강제 변환합니다. 이 옵션은 항목이 대부분 균일한 컬렉션에 적합합니다.

더 다양한 데이터 형식이 있는 컬렉션의 경우 커넥터는 AWS Glue Data Catalog에서 메타데이터 검색을 지원합니다. 커넥터가 HBase 네임스페이스 및 컬렉션 이름과 일치하는 AWS Glue 데이터베이스 및 테이블을 발견하면 해당 AWS Glue 테이블에서 스키마 정보를 가져옵니다. AWS Glue 테이블을 생성할 때 HBase 컬렉션에서 액세스하려는 모든 필드의 상위 세트로 만드는 것이 좋습니다.

계정에서 Lake Formation을 활성화한 경우 AWS Serverless Application Repository에 배포한 Athena 페더레이션형 Lambda 커넥터의 IAM 역할은 Lake Formation에서 AWS Glue Data Catalog에 대한 읽기 액세스 권한을 가지고 있어야 합니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 HBase 커넥터를 구성합니다.

- spill_bucket – Lambda 함수 제한을 초과하는 데이터에 대한 Amazon S3 버킷을 지정합니다.
- spill_prefix – (선택 사항) 기본값은 `athena-federation-spill`이라는 지정된 `spill_bucket`의 하위 폴더입니다. 미리 정해진 일 수 또는 시간보다 오래된 유출을 삭제하려면 이 위치에서 Amazon S3 스토리지 수명 주기를 구성하는 것이 좋습니다.
- spill_put_request_headers – (선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵입니다(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.
- kms_key_id – (선택 사항) 기본적으로 Amazon S3로 유출된 모든 데이터는 AES-GCM 인증 암호화 모드와 임의로 생성된 키를 사용하여 암호화됩니다. Lambda 함수가 `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`과 같이 KMS에서 생성된 더 강력한 암호화 키를 사용하도록 하려면 KMS 키 ID를 지정합니다.
- disable_spill_encryption – (선택 사항) `True`로 설정하면 유출 암호화가 비활성화됩니다. S3로 유출되는 데이터가 AES-GCM을 사용하여 암호화되도록 기본값은 `False`입니다(임의로 생성된 키 또는 KMS를 사용하여 키 생성). 유출 암호화를 비활성화하면 특히 유출 위치가 서버 측 암호화를 사용하는 경우 성능이 향상될 수 있습니다.
- disable_glue – (선택 사항) true로 설정된 경우 커넥터는 AWS Glue에서 보충 메타데이터 검색을 시도하지 않습니다.

- glue_catalog – (선택 사항) 이 옵션을 사용하여 크로스 계정 AWS Glue 카탈로그를 지정합니다. 기본적으로 커넥터는 자체 AWS Glue 계정에서 메타데이터를 가져오려고 시도합니다.
- default_hbase – 있는 경우 카탈로그 관련 환경 변수가 없을 때 사용할 HBase 연결 문자열을 지정합니다.

연결 문자열 지정

커넥터에서 사용하는 HBase 인스턴스에 대한 HBase 연결 세부 정보를 정의하는 속성을 하나 이상 제공할 수 있습니다. 이렇게 하려면 Athena에서 사용하려는 카탈로그 이름에 해당하는 Lambda 환경 변수를 설정합니다. 예를 들어, 다음 쿼리를 사용하여 Athena에서 서로 다른 2개의 HBase 인스턴스를 쿼리하려고 한다고 가정해 보겠습니다.

```
SELECT * FROM "hbase_instance_1".database.table
```

```
SELECT * FROM "hbase_instance_2".database.table
```

이 두 SQL 문을 사용하려면 먼저 2개의 환경 변수 hbase_instance_1과 hbase_instance_2를 Lambda 함수에 추가해야 합니다. 각 값은 다음 형식의 HBase 연결 문자열이어야 합니다.

```
master_hostname:hbase_port:zookeeper_port
```

보안 암호 사용

필요에 따라 연결 문자열 세부 정보 값의 일부 또는 전체에 대해 AWS Secrets Manager를 사용할 수 있습니다. Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 인터넷 액세스 또는 VPC 엔드포인트가 Lambda 함수에 연결된 VPC에 있어야 합니다.

${my_secret} 구문을 사용하여 Secrets Manager의 보안 암호 이름을 연결 문자열에 입력하면 커넥터가 보안 암호 이름을 Secrets Manager의 사용자 이름 및 암호 값으로 바꿉니다.

예를 들어, hbase_instance_1에 대한 Lambda 환경 변수를 다음 값으로 설정했다고 가정해 보겠습니다.

```
${hbase_host_1}:${hbase_master_port_1}:${hbase_zookeeper_port_1}
```

Athena Query Federation SDK는 자동으로 Secrets Manager에서 hbase_instance_1_creds라는 보안 암호를 검색하고 ${hbase_instance_1_creds} 대신 해당 값을 삽입하려고 시도합니다.

${ } 문자 조합으로 묶은 연결 문자열 부분은 Secrets Manager의 보안 암호로 해석됩니다. 커넥터가 Secrets Manager에서 찾을 수 없는 보안 암호 이름을 지정하면 커넥터가 텍스트를 바꾸지 않습니다.

AWS Glue에서 데이터베이스 및 테이블 설정

커넥터의 기본 제공 스키마 추론은 HBase에서 문자열로 직렬화되는 값(예: `String.valueOf(int)`)만 지원합니다. 커넥터의 기본 제공 스키마 추론 기능이 제한되어 있으므로 메타데이터에 AWS Glue를 대신 사용할 수 있습니다. HBase에서 사용할 AWS Glue 테이블을 활성화하려면 보충 메타데이터를 제공할 HBase 네임스페이스 및 테이블과 이름이 일치하는 AWS Glue 데이터베이스 및 테이블이 있어야 합니다. HBase 열 패밀리 이름 지정 규칙의 사용은 선택 사항이며 필수는 아닙니다.

보충 메타데이터에 AWS Glue 테이블 사용

1. AWS Glue 콘솔에서 테이블과 데이터베이스를 편집할 때 다음 테이블 속성을 추가합니다.

   - hbase-metadata-flag – 이 속성은 보충 메타데이터에 테이블을 사용할 수 있음을 HBase 커넥터에 나타냅니다. `hbase-metadata-flag` 속성이 테이블 속성 목록에 있는 한 `hbase-metadata-flag`에 모든 값을 제공할 수 있습니다.
   - hbase-native-storage-flag - 이 플래그를 사용하여 커넥터에서 지원하는 2개의 값 직렬화 모드를 전환합니다. 기본적으로 이 필드가 없으면 커넥터는 모든 값이 HBase에 문자열로 저장되어 있다고 가정합니다. 따라서 HBase에서 `INT`, `BIGINT`, `DOUBLE` 등의 데이터 형식을 문자열로 구문 분석하려고 시도합니다. 이 필드가 AWS Glue의 테이블에 있는 값으로 설정된 경우 커넥터는 '네이티브' 스토리지 모드로 전환하고 다음 함수를 사용하여 `INT`, `BIGINT`, `BIT` 및 `DOUBLE`을 바이트로 읽으려고 시도합니다.

     ```
     ByteBuffer.wrap(value).getInt()
     ByteBuffer.wrap(value).getLong()
     ByteBuffer.wrap(value).get()
     ByteBuffer.wrap(value).getDouble()
     ```

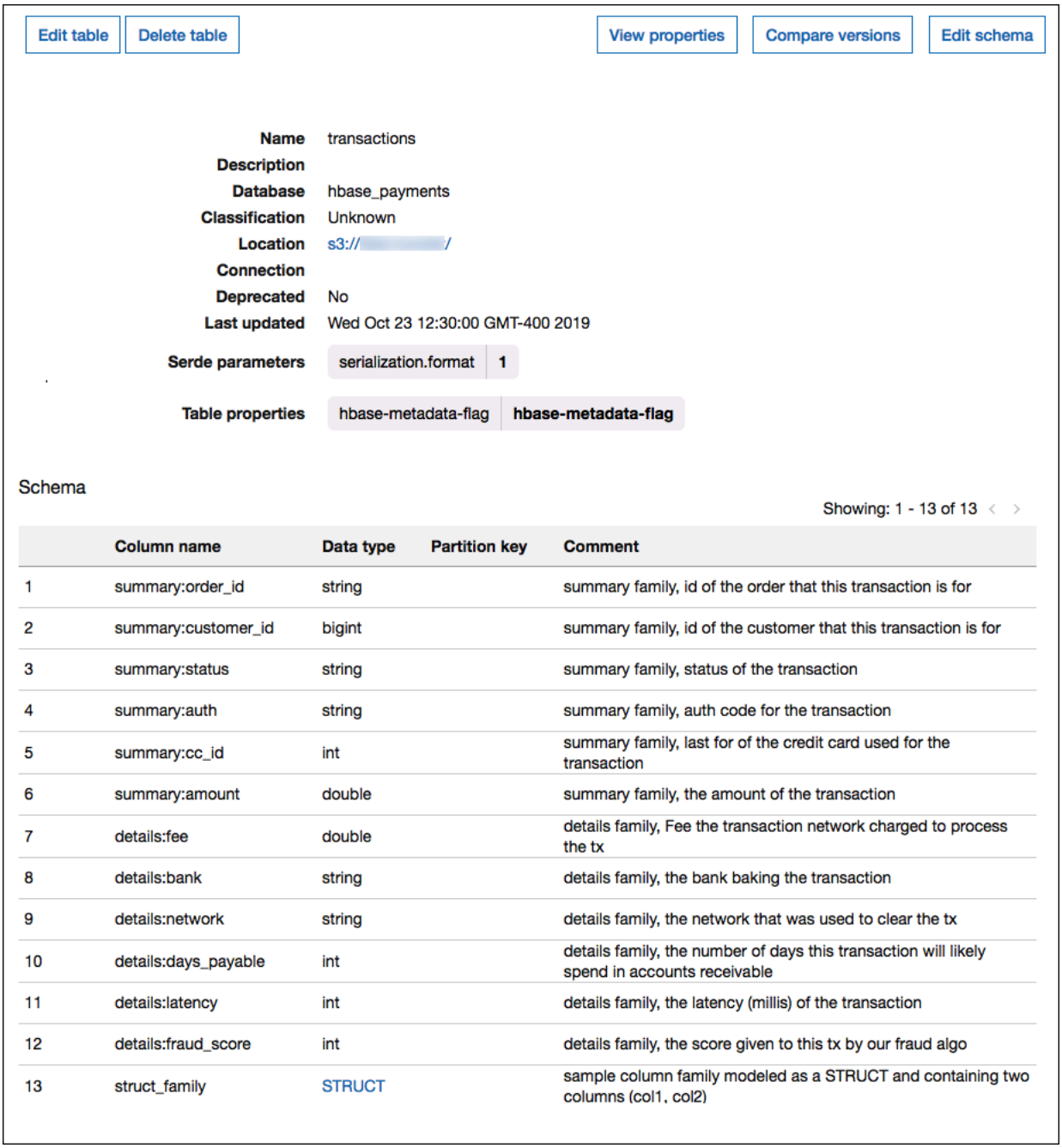
2. 이 문서에 나열된 대로 AWS Glue에 적합한 데이터 형식을 사용해야 합니다.

열 패밀리 모델링

Athena HBase 커넥터는 HBase 열 패밀리를 모델링하는 두 가지 방법, 즉 `family:column`과 같은 정규화된(평면화됨) 이름 지정 또는 `STRUCT` 객체 사용을 지원합니다.

`STRUCT` 모델에서 `STRUCT` 필드의 이름은 열 패밀리와 일치해야 하고 `STRUCT`의 하위 항목은 패밀리의 열 이름과 일치해야 합니다. 그러나 조건자 푸시다운 및 열 형식 읽기는 `STRUCT`와 같은 복합 유형에 대해 아직 완전히 지원되지 않으므로 현재 `STRUCT`를 사용하는 것은 권장되지 않습니다.

다음 이미지에서는 AWS Glue에서 두 가지 접근 방식을 조합하여 사용하도록 구성된 테이블을 보여줍니다.



	Column name	Data type	Partition key	Comment
1	summary:order_id	string		summary family, id of the order that this transaction is for
2	summary:customer_id	bigint		summary family, id of the customer that this transaction is for
3	summary:status	string		summary family, status of the transaction
4	summary:auth	string		summary family, auth code for the transaction
5	summary:cc_id	int		summary family, last for of the credit card used for the transaction
6	summary:amount	double		summary family, the amount of the transaction
7	details:fee	double		details family, Fee the transaction network charged to process the tx
8	details:bank	string		details family, the bank baking the transaction
9	details:network	string		details family, the network that was used to clear the tx
10	details:days_payable	int		details family, the number of days this transaction will likely spend in accounts receivable
11	details:latency	int		details family, the latency (millis) of the transaction
12	details:fraud_score	int		details family, the score given to this tx by our fraud algo
13	struct_family	STRUCT		sample column family modeled as a STRUCT and containing two columns (col1, col2)

데이터 형식 지원

커넥터는 모든 HBase 값을 기본 바이트 유형으로 검색합니다. 그런 다음 AWS Glue 데이터 카탈로그에서 테이블을 정의한 방법에 따라 값을 다음 테이블의 Apache Arrow 데이터 형식 중 하나로 매핑합니다.

AWS Glue 데이터 유형	Apache Arrow 데이터 형식
int	INT
bigint	BIGINT
double	FLOAT8
float	FLOAT4
boolean	BIT
이진수	VARBINARY
문자열	VARCHAR

> **Note**
>
> AWS Glue를 사용하여 메타데이터를 보충하지 않는 경우 커넥터의 스키마 추론은 데이터 형식 `BIGINT`, `FLOAT8` 및 `VARCHAR`만 사용합니다.

필수 권한

이 커넥터에 필요한 IAM 정책에 대한 자세한 내용을 알아보려면 athena-hbase.yaml 파일의 `Policies` 섹션을 검토하세요. 다음 목록에 필요한 권한이 요약되어 있습니다.

- Amazon S3 쓰기 액세스 - 대규모 쿼리의 결과 유출을 위해서는 커넥터에 Amazon S3 위치에 대한 쓰기 액세스 권한이 필요합니다.
- Athena GetQueryExecution - 커넥터는 업스트림 Athena 쿼리가 종료된 경우 이 권한을 사용하여 빠른 실패를 수행합니다.
- AWS Glue Data Catalog - 스키마 정보를 가져오기 위해 HBase 커넥터에 AWS Glue Data Catalog에 대한 읽기 전용 액세스 권한이 필요합니다.
- CloudWatch Logs - 로그를 저장하기 위해 커넥터 CloudWatch Logs에 대한 액세스 권한이 필요합니다.
- AWS Secrets Manager 읽기 액세스 - Secrets Manager에 HBase 엔드포인트 세부 정보를 저장하기로 선택하는 경우 커넥터에 해당 보안 암호에 대한 액세스 권한을 부여해야 합니다.

- VPC 액세스 - VPC에 연결하고 HBase 인스턴스와 통신할 수 있도록 VPC에 인터페이스를 연결하고 분리하는 기능이 커넥터에 필요합니다.

성능

Athena HBase 커넥터는 각 리전 서버를 병렬로 읽어 HBase 인스턴스에 대한 쿼리 병렬화를 시도합니다. 쿼리에서 스캔하는 데이터를 줄이기 위해 Athena HBase 커넥터에서 조건자 푸시다운을 수행합니다.

쿼리에서 스캔된는 데이터를 줄이기 위해 Lambda 함수에서 프로젝션 푸시다운도 수행합니다. 그러나 열의 하위 세트를 선택하면 쿼리 실행 런타임이 길어지는 경우가 있습니다. `LIMIT` 절은 스캔되는 데이터의 양을 줄이지만 조건자를 제공하지 않으면 `LIMIT` 절을 포함하는 `SELECT` 쿼리가 최소 16MB의 데이터를 스캔할 것으로 예상해야 합니다.

HBase는 쿼리 실패가 발생하기 쉽고 쿼리 실행 시간이 가변적입니다. 쿼리가 성공하려면 쿼리를 여러 번 다시 시도해야 할 수 있습니다. HBase 커넥터는 동시성으로 인한 제한에 대한 복원력이 뛰어납니다.

패스스루 쿼리

HBase 커넥터는 패스스루 쿼리를 지원하며 NoSQL 기반입니다. Apache HBase 쿼리에 대한 자세한 내용은 Apache 설명서의 Querying HBase를 참조하세요.

HBase에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            database => 'database_name',
            collection => 'collection_name',
            filter => '{query_syntax}'
        ))
```

다음 예제 HBase 패스스루 쿼리는 `default` 데이터베이스의 `employee` 컬렉션 내에서 24세 또는 30세 직원을 필터링합니다.

```
SELECT * FROM TABLE(
        system.query(
            DATABASE => 'default',
            COLLECTION => 'employee',
```

```
            FILTER => 'SingleColumnValueFilter(''personaldata'', ''age'', =,
''binary:30'')' ||
                      ' OR SingleColumnValueFilter(''personaldata'', ''age'', =,
''binary:24'')'
       ))
```

라이선스 정보

Amazon Athena HBase 커넥터 프로젝트는 Apache-2.0 라이선스에 따라 사용이 허가됩니다.

추가적인 리소스

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Hortonworks 커넥터

Hortonworks용 Amazon Athena 커넥터를 사용하면 Amazon Athena가 Cloudera Hortonworks 데이터 플랫폼에 대해 SQL 쿼리를 실행할 수 있습니다. 커넥터는 Athena SQL 쿼리를 동일한 HiveQL 구문으로 변환합니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 멀티플렉서 설정에서 유출 버킷과 접두사는 모든 데이터베이스 인스턴스에서 공유됩니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.

용어

다음 용어는 Hortonworks Hive 커넥터와 관련이 있습니다.

- 데이터베이스 인스턴스 - 온프레미스, Amazon EC2 또는 Amazon RDS에 배포된 데이터베이스의 모든 인스턴스.

- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.
- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - `connection_string` 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.
- 멀티플렉싱 핸들러 - 여러 데이터베이스 연결을 수락하고 사용할 수 있는 Lambda 핸들러.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 Hortonworks Hive 커넥터를 구성합니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 데이터베이스 인스턴스에 연결합니다.

```
hive://${jdbc_connection_string}
```

멀티플렉싱 핸들러 사용

멀티플렉서를 사용하여 단일 Lambda 함수로 여러 데이터베이스 인스턴스에 연결할 수 있습니다. 요청은 카탈로그 이름을 기준으로 라우팅됩니다. Lambda에서 다음 클래스를 사용합니다.

핸들러	Class
복합 핸들러	`HiveMuxCompositeHandler`
메타데이터 핸들러	`HiveMuxMetadataHandler`
레코드 핸들러	`HiveMuxRecordHandler`

멀티플렉싱 핸들러 파라미터

파라미터	설명
$*catalog*_connection_string	필수 사항입니다. 데이터베이스 인스턴스 연결 문자열. Athena에서 사용되는 카탈로그의 이름을 환경 변수 앞에 붙입니다. 예를 들어, Athena에 등록된 카탈로그가 myhivecatalog 인 경우 환경 변수 이름은 myhivecatalog_connection_string 입니다.
default	필수 사항입니다. 기본 연결 문자열. 이 문자열은 카탈로그가 lambda:${ *AWS_LAMBDA_FUNCTION_NAME* }일 때 사용됩니다.

다음은 hive1(기본값)과 hive2라는 2개의 데이터베이스 인스턴스를 지원하는 Hive MUX Lambda 함수에 대한 예제 속성입니다.

속성	값
default	hive://jdbc:hive2://hive1:10000/default?${Test/RDS/hive1}
hive_catalog1_connection_string	hive://jdbc:hive2://hive1:10000/default?${Test/RDS/hive1}
hive_catalog2_connection_string	hive://jdbc:hive2://hive2:10000/default?UID=sample&PWD=sample

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

> **Important**
>
> 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 `username` 및 `password` 값으로 바꿉니다.

  Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

  ```
  {"username": "${username}", "password": "${password}"}
  ```

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 `${Test/RDS/hive1host}`입니다.

```
hive://jdbc:hive2://hive1host:10000/default?...&${Test/RDS/hive1host}&...
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
hive://jdbc:hive2://hive1host:10000/default?...&UID=sample2&PWD=sample2&...
```

현재 Hortonworks Hive 커넥터는 `UID` 및 `PWD` JDBC 속성을 인식합니다.

단일 연결 핸들러 사용

다음과 같은 단일 연결 메타데이터 및 레코드 핸들러를 사용하여 단일 Hortonworks Hive 인스턴스에 연결할 수 있습니다.

핸들러 유형	Class
복합 핸들러	`HiveCompositeHandler`
메타데이터 핸들러	`HiveMetadataHandler`
레코드 핸들러	`HiveRecordHandler`

단일 연결 핸들러 파라미터

파라미터	설명
`default`	필수 사항입니다. 기본 연결 문자열.

단일 연결 핸들러는 하나의 데이터베이스 인스턴스를 지원하며 `default` 연결 문자열 파라미터를 제공해야 합니다. 다른 연결 문자열은 모두 무시됩니다.

다음은 Lambda 함수에서 지원하는 단일 Hortonworks Hive 인스턴스에 대한 예제 속성입니다.

속성	값
`default`	`hive://jdbc:hive2://hive1host:10000/default?secret=${Test/RDS/hive1host}`

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
`spill_bucket`	필수 사항입니다. 유출 버킷 이름.
`spill_prefix`	필수 사항입니다. 유출 버킷 키 접두사.
`spill_put_request_headers`	(선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: `{"x-amz-s`

파라미터	설명
	erver-side-encryption" : "AES256"}). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 JDBC, Hortonworks Hive 및 Arrow의 해당 데이터 형식이 나와 있습니다.

JDBC	Hortonworks Hive	화살표
불	불	Bit
Integer	TINYINT	Tiny
Short	SMALLINT	Smallint
Integer	INT	정수
Long	BIGINT	Bigint
float	"char"	Float4
Double	float8	Float8
날짜	date	DateDay
Timestamp	타임스탬프	DateMilli
String	VARCHAR	Varchar
바이트	bytes	Varbinary
BigDecimal	10진수	10진수
ARRAY	해당 사항 없음(참고 사항 참조)	나열

> **Note**
>
> 현재 Hortonworks Hive는 집계 유형 ARRAY, MAP, STRUCT 또는 UNIONTYPE을 지원하지 않습니다. 집계 유형의 열은 SQL에서 VARCHAR 열로 처리됩니다.

파티션 및 분할

파티션은 커넥터에 대한 분할을 생성하는 방법을 결정하는 데 사용됩니다. Athena는 커넥터가 분할을 생성하는 데 도움이 되도록 테이블에 대한 파티셔닝 체계를 나타내는 varchar 유형의 합성 열을 생성합니다. 커넥터는 실제 테이블 정의를 수정하지 않습니다.

성능

Hortonworks Hive는 정적 파티션을 지원합니다. Athena Hortonworks Hive 커넥터는 이러한 파티션에서 병렬로 데이터를 검색할 수 있습니다. 파티션 배포가 균일한 초대규모 데이터 세트를 쿼리하려면 정적 파티셔닝을 사용하는 것이 좋습니다. 열의 하위 집합을 선택하면 쿼리 런타임 속도를 높이고 스캔되는 데이터를 줄일 수 있습니다. Hortonworks Hive 커넥터는 동시성으로 인한 제한에 대한 복원력이 뛰어납니다.

Athena Hortonworks Hive 커넥터는 조건부 푸시다운을 수행하여 쿼리에서 스캔하는 데이터를 줄입니다. LIMIT 절, 간단한 조건자 및 복잡한 표현식을 커넥터로 푸시다운하여 스캔하는 데이터와 쿼리 실행 시간을 줄입니다.

LIMIT 절

LIMIT N 문은 쿼리로 스캔하는 데이터를 줄입니다. LIMIT N 푸시다운을 통해 커넥터는 Athena에 N개 행만 반환합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 WHERE 절에 사용되는 표현식입니다. Athena Hortonworks Hive 커넥터는 이러한 표현식을 결합하고 Hortonworks Hive로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena Hortonworks Hive 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND, OR, NOT
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_NULL

- 산술: ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- 기타: LIKE_PATTERN, IN

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

패스스루 쿼리

Hortonworks Hive 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

Hortonworks Hive에서 패스스루 쿼리를 사용하려면 다음 구문을 사용할 수 있습니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 Hortonworks Hive의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 `customer` 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 Hortonworks Hive 커넥터용 pom.xml 파일을 참조하세요.

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Apache Kafka 커넥터

Apache Kafka용 Amazon Athena 커넥터를 사용하면 Amazon Athena가 Apache Kafka 주제에서 SQL 쿼리를 실행할 수 있습니다. 이 커넥터를 사용하여 Athena에서 Apache Kafka 주제를 테이블로, 메시지를 행으로 볼 수 있습니다.

필수 조건

Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.
- 필터 조건의 날짜 및 타임스탬프 데이터 형식을 적절한 데이터 형식으로 캐스팅해야 합니다.
- 날짜 및 타임스탬프 데이터 형식은 CSV 파일 형식에 지원되지 않으며 varchar 값으로 처리됩니다.
- 중첩된 JSON 필드로의 매핑은 지원되지 않습니다. 커넥터는 최상위 필드만 매핑합니다.
- 커넥터는 복합 유형을 지원하지 않습니다. 복합 형식은 문자열로 해석됩니다.
- 복잡한 JSON 값을 추출하거나 사용하려면 Athena에서 제공되는 JSON 관련 함수를 사용하세요. 자세한 내용은 JSON에서 데이터 추출 단원을 참조하십시오.
- 커넥터는 Kafka 메시지 메타데이터에 대한 액세스를 지원하지 않습니다.

용어

- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.

- Kafka 엔드포인트 - Kafka 인스턴스에 연결하는 텍스트 문자열입니다.

클러스터 호환성

Kafka 커넥터는 다음과 같은 클러스터 유형과 함께 사용할 수 있습니다.

- 독립 실행형 Kafka - Kafka에 대한 직접 연결(인증된 연결 또는 인증되지 않은 연결)입니다.
- Confluent – Confluent Kafka에 대한 직접 연결입니다. Confluent Kafka 데이터와 함께 Athena를 사용하는 방법에 대한 자세한 내용은 AWS 비즈니스 인텔리전스 블로그의 Visualize Confluent data in Amazon QuickSight using Amazon Athena를 참조하세요.

Confluent에 연결

Confluent에 연결하려면 다음 단계를 수행해야 합니다.

1. Confluent에서 API 키를 생성합니다.
2. Confluent API 키의 사용자 이름과 암호를 AWS Secrets Manager에 저장합니다.
3. Kafka 커넥터에 `secrets_manager_secret` 환경 변수의 보안 암호 이름을 입력합니다.
4. 이 문서의 Kafka 커넥터 설정 섹션에 나온 단계를 수행합니다.

지원되는 인증 방법

커넥터에서 지원되는 인증 방법은 다음과 같습니다.

- SSL
- SASL/SCRAM
- SASL/PLAIN
- SASL/PLAINTEXT
- NO_AUTH
- 자체 관리형 Kafka 및 Confluent 플랫폼 – SSL, SASL/SCRAM, SASL/PLAINTEXT, NO_AUTH
- 자체 관리형 Kafka 및 Confluent 플랫폼 – SASL/PLAIN

자세한 내용은 Athena Kafka 커넥터에 대한 인증 구성 단원을 참조하십시오.

지원되는 입력 데이터 형식

커넥터에서 지원되는 입력 데이터 형식은 다음과 같습니다.

- JSON
- CSV

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 Athena Kafka 커넥터를 구성합니다.

- auth_type - 클러스터의 인증 유형을 지정합니다. 커넥터에서 지원되는 인증 유형은 다음과 같습니다.
  - NO_AUTH - Kafka에 직접 연결합니다. 예를 들어 인증을 사용하지 않는 EC2 인스턴스를 통해 배포되는 Kafka 클러스터에 연결합니다.
  - SASL_SSL_PLAIN - 이 메서드는 `SASL_SSL` 보안 프로토콜과 `PLAIN` SASL 메커니즘을 사용합니다. 자세한 내용은 Apache Kafka 설명서의 SASL configuration을 참조하세요.
  - SASL_PLAINTEXT_PLAIN - 이 메서드는 `SASL_PLAINTEXT` 보안 프로토콜과 `PLAIN` SASL 메커니즘을 사용합니다. 자세한 내용은 Apache Kafka 설명서의 SASL configuration을 참조하세요.
  - SASL_SSL_SCRAM_SHA512 - 이 인증 유형을 사용하여 Apache Kafka 클러스터에 대한 액세스를 제어할 수 있습니다. 이 메서드는 사용자 이름과 암호를 AWS Secrets Manager에 저장합니다. 보안 암호는 Kafka 클러스터와 연결해야 합니다. 자세한 내용은 Apache Kafka 설명서의 Authentication using SASL/SCRAM을 참조하세요.
  - SASL_PLAINTEXT_SCRAM_SHA512 – 이 메서드는 `SASL_PLAINTEXT` 보안 프로토콜 및 `SCRAM_SHA512` SASL 메커니즘을 사용합니다. 이 메서드는 AWS Secrets Manager에 저장된 사용자 이름과 암호를 사용합니다. 자세한 내용은 Apache Kafka 설명서의 SASL configuration 섹션을 참조하세요.
  - SSL - SSL 인증은 키 스토어 및 트러스트 스토어 파일을 사용하여 Apache Kafka 클러스터에 연결합니다. 트러스트 스토어 파일과 키 스토어 파일을 생성하여 Amazon S3 버킷에 업로드하고 커넥터를 배포할 때 Amazon S3에 대한 참조를 제공해야 합니다. 키 스토어, 트러스트 스토어 및 SSL 키는 AWS Secrets Manager에 저장됩니다. 커넥터가 배포될 때 클라이언트가 AWS 보안 암호 키를 제공해야 합니다. 자세한 내용은 Apache Kafka 설명서의 Encryption and Authentication using SSL을 참조하세요.

    자세한 내용은 Athena Kafka 커넥터에 대한 인증 구성 단원을 참조하십시오.

- certificates_s3_reference - 인증서(키 스토어 및 트러스트 스토어 파일)가 들어 있는 Amazon S3 위치입니다.
- disable_spill_encryption – (선택 사항) `True`로 설정하면 유출 암호화가 비활성화됩니다. S3로 유출되는 데이터가 AES-GCM을 사용하여 암호화되도록 기본값은 `False`입니다(임의로 생성된 키 또는 KMS를 사용하여 키 생성). 유출 암호화를 비활성화하면 특히 유출 위치가 서버 측 암호화를 사용하는 경우 성능이 향상될 수 있습니다.
- kafka_endpoint - Kafka에 제공할 엔드포인트 세부 정보입니다.
- secrets_manager_secret - 보안 인증이 저장되는 AWS 보안 암호의 이름입니다.
- 유출 파라미터 - Lambda 함수는 메모리에 맞지 않는 데이터를 Amazon S3에 임시로 저장("유출")합니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다. 다음 표의 파라미터를 사용하여 유출 위치를 지정합니다.

파라미터	설명
`spill_bucket`	필수 사항입니다. Lambda 함수가 데이터를 유출할 수 있는 Amazon S3 버킷의 이름입니다.
`spill_prefix`	필수 사항입니다. Lambda 함수가 데이터를 유출할 수 있는 유출 버킷 내 접두사입니다.
`spill_put_request_headers`	(선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

- 서브넷 ID – Lambda 함수가 데이터 소스에 액세스하는 데 사용할 수 있는 서브넷에 해당하는 하나 이상의 서브넷 ID입니다.
  - 퍼블릭 Kafka 클러스터 또는 표준 Confluent Cloud 클러스터 - 커넥터를 NAT 게이트웨이가 있는 프라이빗 서브넷에 연결합니다.
  - 프라이빗 연결을 제공하는 Confluent Cloud 클러스터 - Confluent Cloud 클러스터로 연결되는 경로가 있는 프라이빗 서브넷에 커넥터를 연결합니다.
    - AWS Transit Gateway의 경우 서브넷은 Confluent Cloud가 사용하는 것과 동일한 전송 게이트웨이에 연결된 VPC에 있어야 합니다.
    - VPC 피어링의 경우 서브넷은 Confluent Cloud VPC에 피어링된 VPC에 있어야 합니다.

- AWS PrivateLink의 경우 서브넷은 Confluent Cloud에 연결된 VPC 엔드포인트로 연결되는 경로가 있는 VPC에 있어야 합니다.

> **Note**
>
> 프라이빗 리소스에 액세스하기 위해 커넥터를 VPC에 배포하고 Confluent와 같이 공개적으로 액세스 가능한 서비스에도 연결하려는 경우 커넥터를 NAT 게이트웨이가 있는 프라이빗 서브넷과 연결해야 합니다. 자세한 정보는 Amazon VPC 사용 설명서의 NAT 게이트웨이를 참조하세요.

데이터 형식 지원

다음 표에 Kafka와 Apache Arrow에 대해 지원되는 해당 데이터 형식이 나와 있습니다.

Kafka	화살표
CHAR	VARCHAR
VARCHAR	VARCHAR
TIMESTAMP	MILLISECOND
날짜	DAY
BOOLEAN	BOOL
SMALLINT	SMALLINT
INTEGER	INT
BIGINT	BIGINT
DECIMAL	FLOAT8
DOUBLE	FLOAT8

파티션 및 분할

Kafka 주제는 파티션으로 분할됩니다. 각 파티션은 순서대로 정렬됩니다. 파티션의 각 메시지에는 offset이라는 증분 ID가 있습니다. 각 Kafka 파티션은 병렬 처리를 위해 다시 여러 영역으로 분할됩니다. 데이터는 Kafka 클러스터에 구성된 보존 기간 동안 사용할 수 있습니다.

모범 사례

다음 예와 같이 Athena를 쿼리할 때 조건자 푸시다운을 사용하는 것이 좋습니다.

```
SELECT *
FROM "kafka_catalog_name"."glue_schema_registry_name"."glue_schema_name"
WHERE integercol = 2147483647
```

```
SELECT *
FROM "kafka_catalog_name"."glue_schema_registry_name"."glue_schema_name"
WHERE timestampcol >= TIMESTAMP '2018-03-25 07:30:58.878'
```

Kafka 커넥터 설정

커넥터를 사용하려면 먼저 Apache Kafka 클러스터를 설정하고, AWS Glue Schema Registry를 사용하여 스키마를 정의하고, 커넥터에 대한 인증을 구성해야 합니다.

AWS Glue Schema Registry에서 작업할 경우 다음 사항에 유의하세요.

- AWS Glue Schema Registry의 Description(설명) 필드에 있는 텍스트에 `{AthenaFederationKafka}` 문자열이 포함되어 있는지 확인합니다. 이 마커 문자열은 Amazon Athena Kafka 커넥터와 함께 사용하는 AWS Glue 레지스트리에 필요합니다.
- 최상의 성능을 얻으려면 데이터베이스 이름과 테이블 이름에 소문자만 사용합니다. 대/소문자를 혼합하여 사용하면 커넥터에서 대소문자를 구분하지 않고 검색하므로 더욱 컴퓨팅 집약적입니다.

Apache Kafka 환경 및 AWS Glue Schema Registry를 설정하려면

1. Apache Kafka 환경을 설정합니다.
2. JSON 형식의 Kafka 주제 설명 파일, 즉, 스키마를 AWS Glue Schema Registry에 업로드합니다. 자세한 내용은 AWS Glue 개발자 안내서의 AWS Glue Schema Registry와 통합을 참조하세요. 스키마 예는 다음 단원을 참조하세요.

AWS Glue Schema Registry의 스키마 예

스키마를 AWS Glue Schema Registry에 업로드할 경우 이 단원의 예제 형식을 사용하세요.

JSON 유형 스키마 예

다음 예제에서 AWS Glue Schema Registry에서 생성할 스키마는 `json`을 `dataFormat`의 값으로 지정하고 `datatypejson`을 `topicName`에 사용합니다.

> **Note**
>
> `topicName`의 값에서는 Kafka의 주제 이름과 동일한 대/소문자를 사용해야 합니다.

```
{
  "topicName": "datatypejson",
  "message": {
    "dataFormat": "json",
    "fields": [
      {
        "name": "intcol",
        "mapping": "intcol",
        "type": "INTEGER"
      },
      {
        "name": "varcharcol",
        "mapping": "varcharcol",
        "type": "VARCHAR"
      },
      {
        "name": "booleancol",
        "mapping": "booleancol",
        "type": "BOOLEAN"
      },
      {
        "name": "bigintcol",
        "mapping": "bigintcol",
        "type": "BIGINT"
      },
      {
        "name": "doublecol",
        "mapping": "doublecol",
        "type": "DOUBLE"
```

```
      },
      {
        "name": "smallintcol",
        "mapping": "smallintcol",
        "type": "SMALLINT"
      },
      {
        "name": "tinyintcol",
        "mapping": "tinyintcol",
        "type": "TINYINT"
      },
      {
        "name": "datecol",
        "mapping": "datecol",
        "type": "DATE",
        "formatHint": "yyyy-MM-dd"
      },
      {
        "name": "timestampcol",
        "mapping": "timestampcol",
        "type": "TIMESTAMP",
        "formatHint": "yyyy-MM-dd HH:mm:ss.SSS"
      }
    ]
  }
}
```

CSV 유형 스키마 예

다음 예제에서 AWS Glue Schema Registry에서 생성할 스키마는 `csv`을 `dataFormat`의 값으로 지정하고 `datatypecsvbulk`을 `topicName`에 사용합니다. `topicName`의 값에서는 Kafka의 주제 이름과 동일한 대/소문자를 사용해야 합니다.

```
{
  "topicName": "datatypecsvbulk",
  "message": {
    "dataFormat": "csv",
    "fields": [
      {
        "name": "intcol",
        "type": "INTEGER",
        "mapping": "0"
      },
```

```
      {
        "name": "varcharcol",
        "type": "VARCHAR",
        "mapping": "1"
      },
      {
        "name": "booleancol",
        "type": "BOOLEAN",
        "mapping": "2"
      },
      {
        "name": "bigintcol",
        "type": "BIGINT",
        "mapping": "3"
      },
      {
        "name": "doublecol",
        "type": "DOUBLE",
        "mapping": "4"
      },
      {
        "name": "smallintcol",
        "type": "SMALLINT",
        "mapping": "5"
      },
      {
        "name": "tinyintcol",
        "type": "TINYINT",
        "mapping": "6"
      },
      {
        "name": "floatcol",
        "type": "DOUBLE",
        "mapping": "7"
      }
    ]
  }
}
```

Athena Kafka 커넥터에 대한 인증 구성

SSL, SASL/SCRAM, SASL/PLAIN, SASL/PLAINTEXT를 비롯한 다양한 방법을 사용하여 Apache Kafka 클러스터에 대한 인증을 수행할 수 있습니다.

다음 표에는 커넥터의 인증 유형과 각 유형에 대한 보안 프로토콜 및 SASL 메커니즘이 나와 있습니다. 자세한 내용은 Apache Kafka 설명서의 Security 섹션을 참조하세요.

auth_type	security.protocol	sasl.mechanism	클러스터 유형 호환성
SASL_SSL_PLAIN	SASL_SSL	PLAIN	• 자체 관리형 Kafka • Confluent 플랫폼 • Confluent Cloud
SASL_PLAINTEXT_PLAIN	SASL_PLAINTEXT	PLAIN	• 자체 관리형 Kafka • Confluent 플랫폼
SASL_SSL_SCRAM_SHA512	SASL_SSL	SCRAM-SHA-512	• 자체 관리형 Kafka • Confluent 플랫폼
SASL_PLAINTEXT_SCRAM_SHA512	SASL_PLAINTEXT	SCRAM-SHA-512	• 자체 관리형 Kafka • Confluent 플랫폼
SSL	SSL	N/A	• 자체 관리형 Kafka • Confluent 플랫폼

SSL

클러스터가 SSL 인증을 받은 경우 트러스트 스토어와 키 스토어 파일을 생성하여 Amazon S3 버킷에 업로드해야 합니다. 커넥터를 배포할 때 이 Amazon S3 참조를 제공해야 합니다. 키 스토어, 트러스트 스토어 및 SSL 키는 AWS Secrets Manager에 저장됩니다. 커넥터를 배포할 때 AWS 비밀 키를 제공합니다.

Secrets Manager에서 보안 암호를 생성하는 방법에 대한 자세한 내용은 AWS Secrets Manager 보안 암호 생성을 참조하세요.

이 인증 유형을 사용하려면 다음 표에 표시된 대로 환경 변수를 설정합니다.

파라미터	값
auth_type	SSL

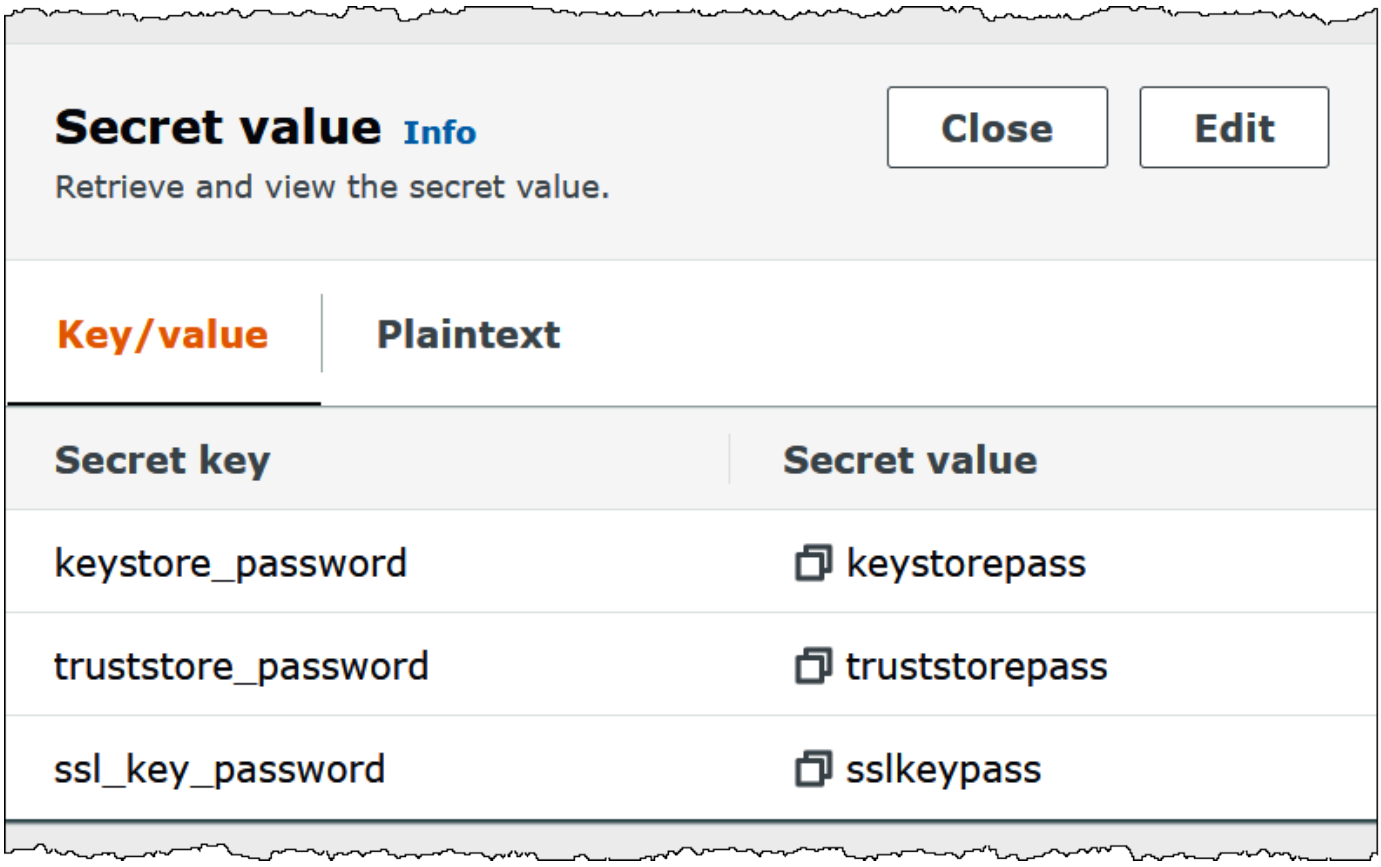
파라미터	값
`certificates_s3_reference`	인증서가 포함된 Amazon S3 위치입니다.
`secrets_manager_secret`	AWS 비밀 키의 이름입니다.

Secrets Manager에서 보안 암호를 생성한 이후에 Secrets Manager 콘솔에서 보안 암호를 볼 수 있습니다.

Secrets Manager에서 보안 암호를 보려면

1. https://console.aws.amazon.com/secretsmanager/에서 Secrets Manager 콘솔을 엽니다.
2. 탐색 창에서 Secrets(보안 암호)를 선택합니다.
3. Secrets(보안 암호) 페이지에서 보안 암호에 대한 링크를 선택합니다.
4. 보안 암호에 대한 세부 정보 페이지에서 Retrieve secret value(보안 암호 값 검색)를 선택합니다.

   다음 이미지는 세 개의 키/값 쌍(`keystore_password`, `truststore_password`, `ssl_key_password`)을 가진 보안 암호의 예를 보여줍니다.



Kafka에서 SSL을 사용하는 방법에 대한 자세한 내용은 Apache Kafka 설명서의 Encryption and Authentication using SSL을 참조하세요.

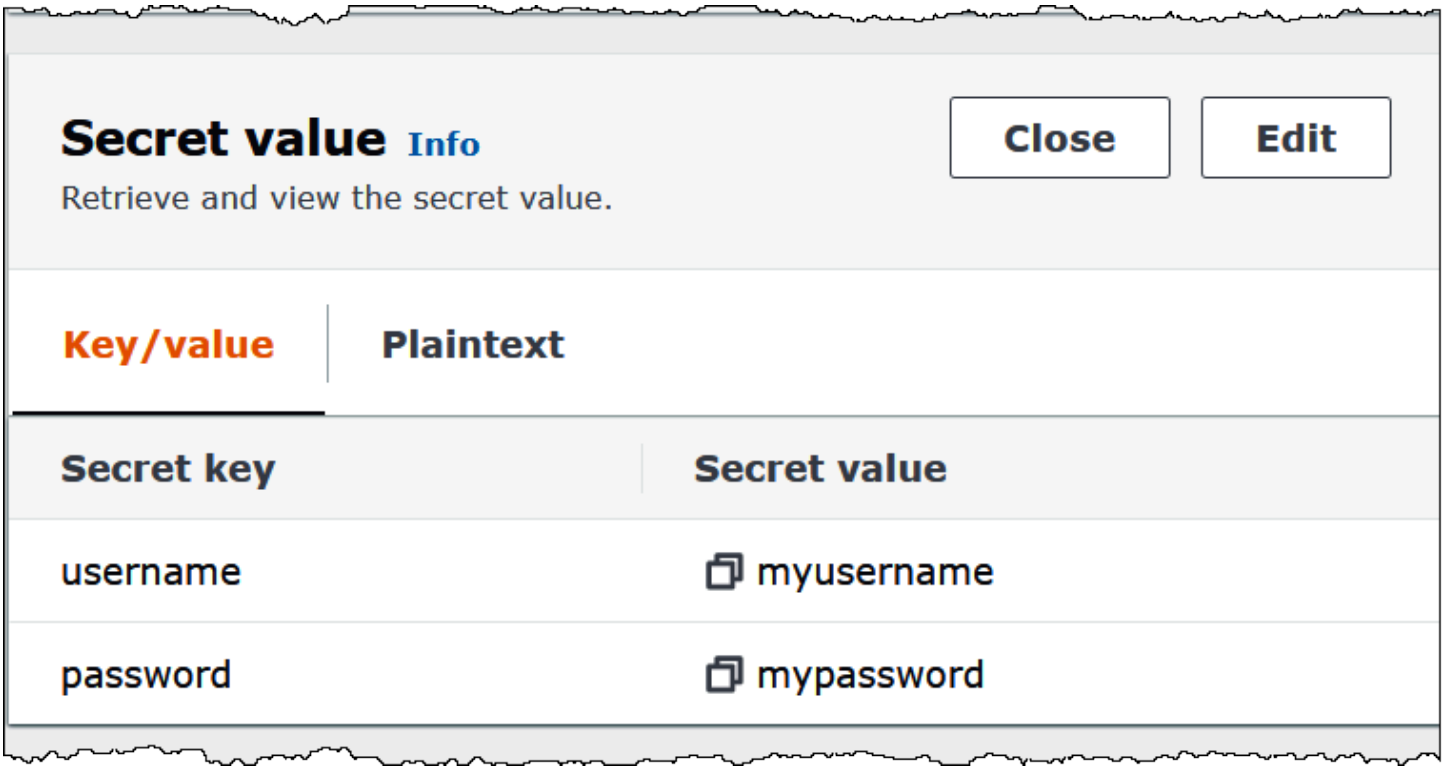
SASL/SCRAM

클러스터에서 SCRAM 인증을 사용하는 경우 커넥터를 배포할 때 클러스터와 연결되는 Secrets Manager 키를 제공합니다. 사용자의 AWS 보안 인증(비밀 키 및 액세스 키)은 클러스터에 인증하는 데 사용됩니다.

다음 표에 표시된 대로 환경 변수를 설정합니다.

파라미터	값
`auth_type`	`SASL_SSL_SCRAM_SHA512`
`secrets_manager_secret`	AWS 비밀 키의 이름입니다.

다음 이미지는 Secrets Manager 콘솔에서 `username` 및 `password`에 대해 하나씩 두 개의 키/값 쌍으로 구성된 보안 암호의 예를 보여줍니다.



Kafka에서 SASL/SCRAM을 사용하는 방법에 대한 자세한 내용은 Apache Kafka 설명서의 Authentication using SASL/SCRAM을 참조하세요.

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena MSK 커넥터

Amazon MSK용 Amazon Athena 커넥터를 사용하면 Amazon Athena가 Apache Kafka 주제에 대해 SQL 쿼리를 실행할 수 있습니다. 이 커넥터를 사용하여 Athena에서 Apache Kafka 주제를 테이블로, 메시지를 행으로 볼 수 있습니다. 자세한 내용은 AWS 빅 데이터 블로그에서 Amazon Athena를 사용하여 Amazon MSK에서 실시간 스트리밍 데이터 분석을 참조하세요.

필수 조건

Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.
- 필터 조건의 날짜 및 타임스탬프 데이터 형식을 적절한 데이터 형식으로 캐스팅해야 합니다.
- 날짜 및 타임스탬프 데이터 형식은 CSV 파일 형식에 지원되지 않으며 varchar 값으로 처리됩니다.
- 중첩된 JSON 필드로의 매핑은 지원되지 않습니다. 커넥터는 최상위 필드만 매핑합니다.
- 커넥터는 복합 유형을 지원하지 않습니다. 복합 형식은 문자열로 해석됩니다.
- 복잡한 JSON 값을 추출하거나 사용하려면 Athena에서 제공되는 JSON 관련 함수를 사용하세요. 자세한 내용은 JSON에서 데이터 추출 단원을 참조하십시오.
- 커넥터는 Kafka 메시지 메타데이터에 대한 액세스를 지원하지 않습니다.

용어

- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- Kafka 엔드포인트 - Kafka 인스턴스에 연결하는 텍스트 문자열입니다.

클러스터 호환성

MSK 커넥터는 다음과 같은 클러스터 유형과 함께 사용할 수 있습니다.

- MSK 프로비저닝 클러스터 - 클러스터 용량을 수동으로 지정, 모니터링 및 조정합니다.
- MSK 서버리스 클러스터 - 애플리케이션 I/O가 확장되면 자동으로 확장되는 온디맨드 용량을 제공합니다.
- 독립 실행형 Kafka - Kafka에 대한 직접 연결(인증된 연결 또는 인증되지 않은 연결)입니다.

지원되는 인증 방법

커넥터에서 지원되는 인증 방법은 다음과 같습니다.

- SAL/IAM
- SSL
- SASL/SCRAM
- SASL/PLAIN
- SASL/PLAINTEXT
- NO_AUTH

  자세한 내용은 Athena MSK 커넥터에 대한 인증 구성 단원을 참조하십시오.

지원되는 입력 데이터 형식

커넥터에서 지원되는 입력 데이터 형식은 다음과 같습니다.

- JSON
- CSV

파라미터

이 섹션에 언급된 Lambda 환경 변수를 사용하여 Athena MSK 커넥터를 구성합니다.

- auth_type - 클러스터의 인증 유형을 지정합니다. 커넥터에서 지원되는 인증 유형은 다음과 같습니다.
  - NO_AUTH - 인증 없이 Kafka에 직접 연결합니다. 예를 들어 인증을 사용하지 않는 EC2 인스턴스를 통해 배포되는 Kafka 클러스터에 연결합니다.
  - SASL_SSL_PLAIN - 이 메서드는 `SASL_SSL` 보안 프로토콜과 `PLAIN` SASL 메커니즘을 사용합니다.
  - SASL_PLAINTEXT_PLAIN - 이 메서드는 `SASL_PLAINTEXT` 보안 프로토콜과 `PLAIN` SASL 메커니즘을 사용합니다.

> **Note**
>
> SASL_SSL_PLAIN 및 SASL_PLAINTEXT_PLAIN 인증 유형은 Apache Kafka에서 지원되지만 Amazon MSK에서는 지원되지 않습니다.

- SASL_SSL_AWS_MSK_IAM - Amazon MSK용 IAM 액세스 제어를 사용하여 MSK 클러스터에 대한 인증과 권한 부여를 모두 처리할 수 있습니다. 사용자 AWS 보안 인증(비밀 키 및 액세스 키)은 클러스터에 연결하는 데 사용됩니다. 자세한 내용은 Amazon Managed Streaming for Apache Kafka 개발자 안내서의 IAM 액세스 제어를 참조하세요.
- SASL_SSL_SCRAM_SHA512 - 이 인증 유형을 사용하여 Amazon MSK 클러스터에 대한 액세스를 제어할 수 있습니다. 이 메서드는 사용자 이름과 암호를 AWS Secrets Manager에 저장합니다. 보안 암호를 Amazon MSK 클러스터와 연결해야 합니다. 자세한 내용은 Amazon Managed Streaming for Apache Kafka 개발자 안내서의 Amazon MSK 클러스터에 대한 SASL/SCRAM 인증 설정을 참조하세요.
- SSL - SSL 인증은 키 스토어 및 트러스트 스토어 파일을 사용하여 Amazon MSK 클러스터와 연결합니다. 트러스트 스토어 파일과 키 스토어 파일을 생성하여 Amazon S3 버킷에 업로드하고 커넥터를 배포할 때 Amazon S3에 대한 참조를 제공해야 합니다. 키 스토어, 트러스트 스토어 및 SSL 키는 AWS Secrets Manager에 저장됩니다. 커넥터가 배포될 때 클라이언트가 AWS 보안 암호 키를 제공해야 합니다. 자세한 내용은 Amazon Managed Streaming for Apache Kafka 개발자 안내서의 상호 TLS 인증을 참조하세요.

  자세한 내용은 Athena MSK 커넥터에 대한 인증 구성 단원을 참조하십시오.
- certificates_s3_reference - 인증서(키 스토어 및 트러스트 스토어 파일)가 들어 있는 Amazon S3 위치입니다.
- disable_spill_encryption – (선택 사항) True로 설정하면 유출 암호화가 비활성화됩니다. S3로 유출되는 데이터가 AES-GCM을 사용하여 암호화되도록 기본값은 False입니다(임의로 생성된 키 또는 KMS를 사용하여 키 생성). 유출 암호화를 비활성화하면 특히 유출 위치가 서버 측 암호화를 사용하는 경우 성능이 향상될 수 있습니다.
- kafka_endpoint - Kafka에 제공할 엔드포인트 세부 정보입니다. 예를 들어 Amazon MSK 클러스터의 경우 클러스터의 부트스트랩 URL을 제공합니다.
- secrets_manager_secret - 보안 인증이 저장되는 AWS 보안 암호의 이름입니다. IAM 인증에는 이 파라미터가 필요하지 않습니다.
- 유출 파라미터 - Lambda 함수는 메모리에 맞지 않는 데이터를 Amazon S3에 임시로 저장(“유출”)합니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다. 다음 표의 파라미터를 사용하여 유출 위치를 지정합니다.

파라미터	설명
`spill_bucket`	필수 사항입니다. Lambda 함수가 데이터를 유출할 수 있는 Amazon S3 버킷의 이름입니다.
`spill_prefix`	필수 사항입니다. Lambda 함수가 데이터를 유출할 수 있는 유출 버킷 내 접두사입니다.
`spill_put_request_headers`	(선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 Kafka와 Apache Arrow에 대해 지원되는 해당 데이터 형식이 나와 있습니다.

Kafka	화살표
CHAR	VARCHAR
VARCHAR	VARCHAR
TIMESTAMP	MILLISECOND
날짜	DAY
BOOLEAN	BOOL
SMALLINT	SMALLINT
INTEGER	INT
BIGINT	BIGINT
DECIMAL	FLOAT8

Kafka	화살표
DOUBLE	FLOAT8

파티션 및 분할

Kafka 주제는 파티션으로 분할됩니다. 각 파티션은 순서대로 정렬됩니다. 파티션의 각 메시지에는 offset이라는 증분 ID가 있습니다. 각 Kafka 파티션은 병렬 처리를 위해 다시 여러 영역으로 분할됩니다. 데이터는 Kafka 클러스터에 구성된 보존 기간 동안 사용할 수 있습니다.

모범 사례

다음 예와 같이 Athena를 쿼리할 때 조건자 푸시다운을 사용하는 것이 좋습니다.

```
SELECT *
FROM "msk_catalog_name"."glue_schema_registry_name"."glue_schema_name"
WHERE integercol = 2147483647
```

```
SELECT *
FROM "msk_catalog_name"."glue_schema_registry_name"."glue_schema_name"
WHERE timestampcol >= TIMESTAMP '2018-03-25 07:30:58.878'
```

MSK 커넥터 설정

커넥터를 사용하려면 먼저 Amazon MSK 클러스터를 설정하고, AWS Glue 스키마 레지스트리를 사용하여 스키마를 정의하고, 커넥터에 대한 인증을 구성해야 합니다.

> **Note**
>
> 프라이빗 리소스에 액세스하기 위해 커넥터를 VPC에 배포하고 Confluent와 같이 공개적으로 액세스 가능한 서비스에도 연결하려는 경우 커넥터를 NAT 게이트웨이가 있는 프라이빗 서브넷과 연결해야 합니다. 자세한 정보는 Amazon VPC 사용 설명서의 NAT 게이트웨이를 참조하세요.

AWS Glue Schema Registry에서 작업할 경우 다음 사항에 유의하세요.

- AWS Glue Schema Registry의 Description(설명) 필드에 있는 텍스트에 `{AthenaFederationMSK}` 문자열이 포함되어 있는지 확인합니다. 이 마커 문자열은 Amazon Athena MSK 커넥터와 함께 사용하는 AWS Glue 레지스트리에 필요합니다.
- 최상의 성능을 얻으려면 데이터베이스 이름과 테이블 이름에 소문자만 사용합니다. 대/소문자를 혼합하여 사용하면 커넥터에서 대소문자를 구분하지 않고 검색하므로 더욱 컴퓨팅 집약적입니다.

Amazon MSK 환경 및 AWS Glue Schema Registry를 설정하려면

1. Amazon MSK 환경을 설정합니다. 자세한 내용은 Amazon Managed Streaming for Apache Kafka 개발자 안내서의 Amazon MSK 설정 및 Amazon MSK 사용 시작하기를 참조하세요.
2. JSON 형식의 Kafka 주제 설명 파일, 즉, 스키마를 AWS Glue Schema Registry에 업로드합니다. 자세한 내용은 AWS Glue 개발자 안내서의 AWS Glue Schema Registry와 통합을 참조하세요. 스키마 예는 다음 단원을 참조하세요.

AWS Glue Schema Registry의 스키마 예

스키마를 AWS Glue Schema Registry에 업로드할 경우 이 단원의 예제 형식을 사용하세요.

JSON 유형 스키마 예

다음 예제에서 AWS Glue Schema Registry에서 생성할 스키마는 `json`을 `dataFormat`의 값으로 지정하고 `datatypejson`을 `topicName`에 사용합니다.

> **Note**
>
> `topicName`의 값에서는 Kafka의 주제 이름과 동일한 대/소문자를 사용해야 합니다.

```
{
  "topicName": "datatypejson",
  "message": {
    "dataFormat": "json",
    "fields": [
      {
        "name": "intcol",
        "mapping": "intcol",
        "type": "INTEGER"
      },
      {
```

```
        "name": "varcharcol",
        "mapping": "varcharcol",
        "type": "VARCHAR"
      },
      {
        "name": "booleancol",
        "mapping": "booleancol",
        "type": "BOOLEAN"
      },
      {
        "name": "bigintcol",
        "mapping": "bigintcol",
        "type": "BIGINT"
      },
      {
        "name": "doublecol",
        "mapping": "doublecol",
        "type": "DOUBLE"
      },
      {
        "name": "smallintcol",
        "mapping": "smallintcol",
        "type": "SMALLINT"
      },
      {
        "name": "tinyintcol",
        "mapping": "tinyintcol",
        "type": "TINYINT"
      },
      {
        "name": "datecol",
        "mapping": "datecol",
        "type": "DATE",
        "formatHint": "yyyy-MM-dd"
      },
      {
        "name": "timestampcol",
        "mapping": "timestampcol",
        "type": "TIMESTAMP",
        "formatHint": "yyyy-MM-dd HH:mm:ss.SSS"
      }
    ]
  }
```

```
}
```

CSV 유형 스키마 예

다음 예제에서 AWS Glue Schema Registry에서 생성할 스키마는 `csv`을 `dataFormat`의 값으로 지정하고 `datatypecsvbulk`을 `topicName`에 사용합니다. `topicName`의 값에서는 Kafka의 주제 이름과 동일한 대/소문자를 사용해야 합니다.

```
{
  "topicName": "datatypecsvbulk",
  "message": {
    "dataFormat": "csv",
    "fields": [
      {
        "name": "intcol",
        "type": "INTEGER",
        "mapping": "0"
      },
      {
        "name": "varcharcol",
        "type": "VARCHAR",
        "mapping": "1"
      },
      {
        "name": "booleancol",
        "type": "BOOLEAN",
        "mapping": "2"
      },
      {
        "name": "bigintcol",
        "type": "BIGINT",
        "mapping": "3"
      },
      {
        "name": "doublecol",
        "type": "DOUBLE",
        "mapping": "4"
      },
      {
        "name": "smallintcol",
        "type": "SMALLINT",
        "mapping": "5"
      },
```

```
      {
        "name": "tinyintcol",
        "type": "TINYINT",
        "mapping": "6"
      },
      {
        "name": "floatcol",
        "type": "DOUBLE",
        "mapping": "7"
      }
    ]
  }
}
```

Athena MSK 커넥터에 대한 인증 구성

IAM, SSL, SCRAM, 독립 실행형 Kafka를 비롯한 다양한 방법으로 Amazon MSK 클러스터에 인증할 수 있습니다.

다음 표에는 커넥터의 인증 유형과 각 유형에 대한 보안 프로토콜 및 SASL 메커니즘이 나와 있습니다. 자세한 내용은 Amazon Managed Streaming for Apache Kafka 개발자 안내서의 Apache Kafka API에 대한 인증 및 권한 부여를 참조하세요.

auth_type	security.protocol	sasl.mechanism
SASL_SSL_PLAIN	SASL_SSL	PLAIN
SASL_PLAINTEXT_PLAIN	SASL_PLAINTEXT	PLAIN
SASL_SSL_AWS_MSK_IAM	SASL_SSL	AWS_MSK_IAM
SASL_SSL_SCRAM_SHA512	SASL_SSL	SCRAM-SHA-512
SSL	SSL	N/A

> **Note**
>
> SASL_SSL_PLAIN 및 SASL_PLAINTEXT_PLAIN 인증 유형은 Apache Kafka에서 지원되지만 Amazon MSK에서는 지원되지 않습니다.

SASL/IAM

클러스터에서 IAM 인증을 사용하는 경우 클러스터를 설정할 때 사용자에 대한 IAM 정책을 구성해야 합니다. 자세한 내용은 Amazon Managed Streaming for Apache Kafka 개발자 안내서의 IAM 액세스 제어를 참조하세요.

이 인증 유형을 사용하려면 커넥터에 대한 `auth_type` Lambda 환경 변수를 `SASL_SSL_AWS_MSK_IAM`으로 설정합니다.

SSL

클러스터가 SSL 인증을 받은 경우 트러스트 스토어와 키 스토어 파일을 생성하여 Amazon S3 버킷에 업로드해야 합니다. 커넥터를 배포할 때 이 Amazon S3 참조를 제공해야 합니다. 키 스토어, 트러스트 스토어 및 SSL 키는 AWS Secrets Manager에 저장됩니다. 커넥터를 배포할 때 AWS 비밀 키를 제공합니다.

Secrets Manager에서 보안 암호를 생성하는 방법에 대한 자세한 내용은 AWS Secrets Manager 보안 암호 생성을 참조하세요.

이 인증 유형을 사용하려면 다음 표에 표시된 대로 환경 변수를 설정합니다.

파라미터	값
`auth_type`	`SSL`
`certificates_s3_reference`	인증서가 포함된 Amazon S3 위치입니다.
`secrets_manager_secret`	AWS 비밀 키의 이름입니다.

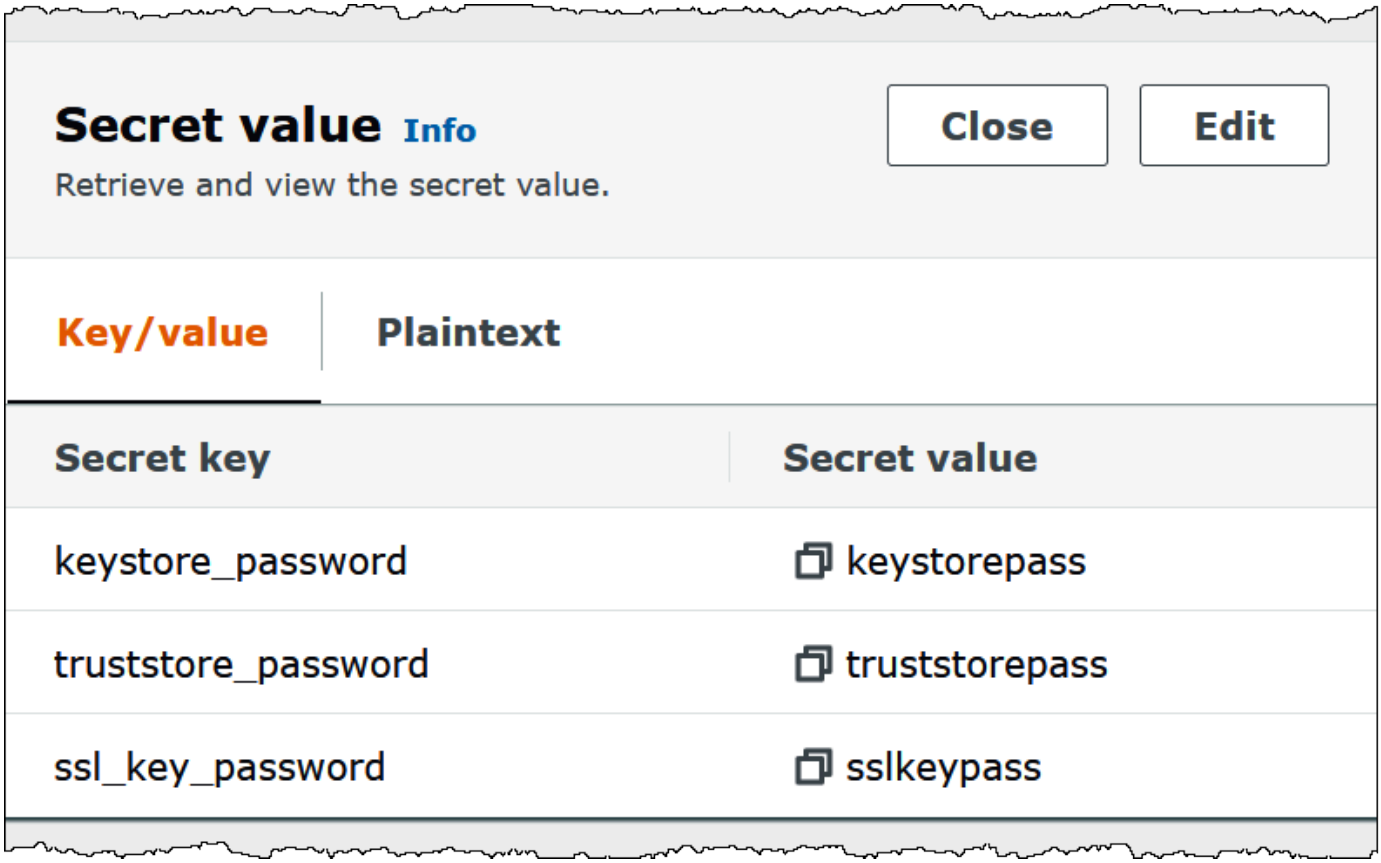
Secrets Manager에서 보안 암호를 생성한 이후에 Secrets Manager 콘솔에서 보안 암호를 볼 수 있습니다.

Secrets Manager에서 보안 암호를 보려면

1. https://console.aws.amazon.com/secretsmanager/에서 Secrets Manager 콘솔을 엽니다.
2. 탐색 창에서 Secrets(보안 암호)를 선택합니다.
3. Secrets(보안 암호) 페이지에서 보안 암호에 대한 링크를 선택합니다.

4. 보안 암호에 대한 세부 정보 페이지에서 Retrieve secret value(보안 암호 값 검색)를 선택합니다.

   다음 이미지는 세 개의 키/값 쌍(`keystore_password`, `truststore_password`, `ssl_key_password`)을 가진 보안 암호의 예를 보여줍니다.



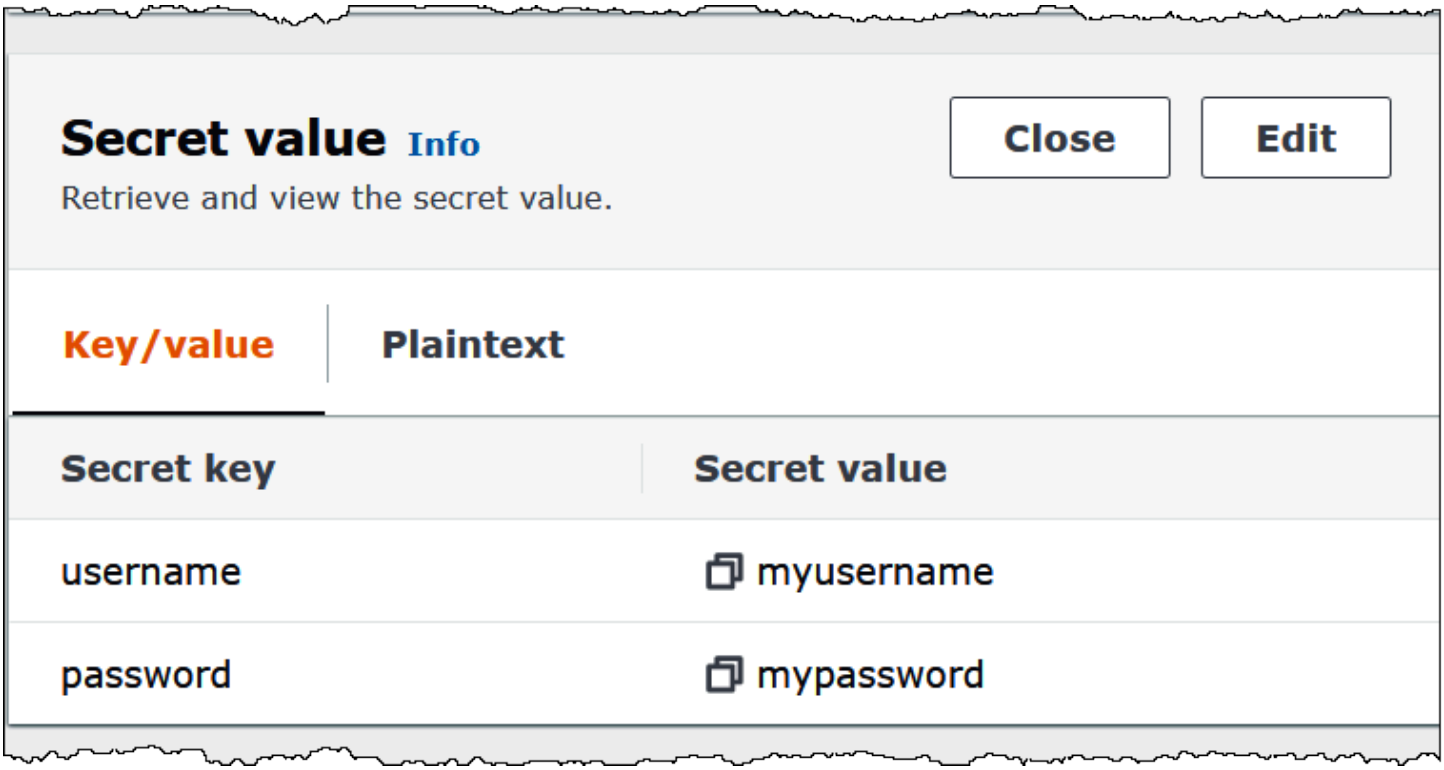
SASL/SCRAM

클러스터에서 SCRAM 인증을 사용하는 경우 커넥터를 배포할 때 클러스터와 연결되는 Secrets Manager 키를 제공합니다. 사용자의 AWS 보안 인증(비밀 키 및 액세스 키)은 클러스터에 인증하는 데 사용됩니다.

다음 표에 표시된 대로 환경 변수를 설정합니다.

파라미터	값
`auth_type`	`SASL_SSL_SCRAM_SHA512`
`secrets_manager_secret`	AWS 비밀 키의 이름입니다.

다음 이미지는 Secrets Manager 콘솔에서 `username` 및 `password`에 대해 하나씩 두 개의 키/값 쌍으로 구성된 보안 암호의 예를 보여줍니다.



라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena MySQL 커넥터

Amazon Athena는 Amazon Athena Lambda MySQL 커넥터를 통해 MySQL 데이터베이스에 액세스할 수 있습니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

- 이 커넥터를 사용하기 전에 VPC와 보안 그룹을 설정합니다. 자세한 내용은 데이터 원본 커넥터용 VPC 생성 단원을 참조하십시오.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 멀티플렉서 설정에서 유출 버킷과 접두사는 모든 데이터베이스 인스턴스에서 공유됩니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.
- Athena는 쿼리를 소문자로 변환하므로 MySQL 테이블 이름은 소문자여야 합니다. 예를 들어, `myTable`이라는 테이블에 대한 Athena 쿼리는 실패합니다.

용어

다음 용어는 MySQL 커넥터와 관련이 있습니다.

- 데이터베이스 인스턴스 - 온프레미스, Amazon EC2 또는 Amazon RDS에 배포된 데이터베이스의 모든 인스턴스.
- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.
- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - `connection_string` 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.
- 멀티플렉싱 핸들러 - 여러 데이터베이스 연결을 수락하고 사용할 수 있는 Lambda 핸들러.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 MySQL 커넥터를 구성합니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 데이터베이스 인스턴스에 연결합니다.

```
mysql://${jdbc_connection_string}
```

> **Note**
>
> MySQL 테이블에서 SELECT 쿼리를 수행할 때 java.sql.SQLException: Zero date value prohibited(java.sql.SQLException: 0 날짜 값 금지) 오류가 발생하면 연결 문자열에 다음 파라미터를 추가합니다.
>
> ```
> zeroDateTimeBehavior=convertToNull
> ```
>
> 자세한 내용을 알아보려면 GitHub.com의 Error 'Zero date value prohibited' while trying to select from MySQL table(MySQL 테이블에서 선택하는 동안 '0 날짜 값 금지' 오류)를 참조하세요.

멀티플렉싱 핸들러 사용

멀티플렉서를 사용하여 단일 Lambda 함수로 여러 데이터베이스 인스턴스에 연결할 수 있습니다. 요청은 카탈로그 이름을 기준으로 라우팅됩니다. Lambda에서 다음 클래스를 사용합니다.

핸들러	Class
복합 핸들러	`MySqlMuxCompositeHandler`
메타데이터 핸들러	`MySqlMuxMetadataHandler`
레코드 핸들러	`MySqlMuxRecordHandler`

멀티플렉싱 핸들러 파라미터

파라미터	설명
$*catalog*_connection_string	필수 사항입니다. 데이터베이스 인스턴스 연결 문자열. Athena에서 사용되는 카탈로그의 이름을 환경 변수 앞에 붙입니다. 예를 들어,

파라미터	설명
	Athena에 등록된 카탈로그가 mymysqlcatalog 인 경우 환경 변수 이름은 mymysqlcatalog_connection_string 입니다.
default	필수 사항입니다. 기본 연결 문자열. 이 문자열은 카탈로그가 lambda:${ *AWS_LAMBDA_FUNCTION_NAME* }일 때 사용됩니다.

다음은 mysql1(기본값)과 mysql2라는 2개의 데이터베이스 인스턴스를 지원하는 MySql MUX Lambda 함수에 대한 예제 속성입니다.

속성	값
default	mysql://jdbc:mysql://mysql2.host:3333/default? user=sample2&password=sample2
mysql_catalog1_connection_string	mysql://jdbc:mysql://mysql1.host:3306/default?${Test/RDS/MySql1}
mysql_catalog2_connection_string	mysql://jdbc:mysql://mysql2.host:3333/default? user=sample2&password=sample2

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

> **Important**
>
> 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용

은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 `username` 및 `password` 값으로 바꿉니다.

  Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

  ```
  {"username": "${username}", "password": "${password}"}
  ```

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 `${Test/RDS/MySql1}`입니다.

```
mysql://jdbc:mysql://mysql1.host:3306/default?...&${Test/RDS/MySql1}&...
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
mysql://jdbc:mysql://mysql1host:3306/default?...&user=sample2&password=sample2&...
```

현재 MySQL 커넥터는 `user` 및 `password` JDBC 속성을 인식합니다.

단일 연결 핸들러 사용

다음과 같은 단일 연결 메타데이터 및 레코드 핸들러를 사용하여 단일 MySQL 인스턴스에 연결할 수 있습니다.

핸들러 유형	Class
복합 핸들러	`MySqlCompositeHandler`
메타데이터 핸들러	`MySqlMetadataHandler`

핸들러 유형	Class
레코드 핸들러	`MySqlRecordHandler`

단일 연결 핸들러 파라미터

파라미터	설명
`default`	필수 사항입니다. 기본 연결 문자열.

단일 연결 핸들러는 하나의 데이터베이스 인스턴스를 지원하며 `default` 연결 문자열 파라미터를 제공해야 합니다. 다른 연결 문자열은 모두 무시됩니다.

다음은 Lambda 함수에서 지원하는 단일 MySQL 인스턴스에 대한 예제 속성입니다.

속성	값
`default`	`mysql://mysql1.host:3306/default?secret=Test/RDS/` MySql1

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
`spill_bucket`	필수 사항입니다. 유출 버킷 이름.
`spill_prefix`	필수 사항입니다. 유출 버킷 키 접두사.
`spill_put_request_headers`	(선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 JDBC와 Arrow의 해당 데이터 형식이 나와 있습니다.

JDBC	화살표
불	Bit
Integer	Tiny
Short	Smallint
Integer	정수
Long	Bigint
float	Float4
Double	Float8
날짜	DateDay
Timestamp	DateMilli
String	Varchar
바이트	Varbinary
BigDecimal	10진수
ARRAY	나열

파티션 및 분할

파티션은 커넥터에 대한 분할을 생성하는 방법을 결정하는 데 사용됩니다. Athena는 커넥터가 분할을 생성하는 데 도움이 되도록 테이블에 대한 파티셔닝 체계를 나타내는 `varchar` 유형의 합성 열을 생성합니다. 커넥터는 실제 테이블 정의를 수정하지 않습니다.

성능

MySQL은 기본 파티션을 지원합니다. Athena MySQL 커넥터는 이러한 파티션에서 병렬로 데이터를 검색할 수 있습니다. 파티션 배포가 균일한 초대규모 데이터 세트를 쿼리하려면 기본 파티셔닝을 사용하는 것이 좋습니다.

Athena MySQL 커넥터는 조건부 푸시다운을 수행하여 쿼리에서 스캔하는 데이터를 줄입니다. `LIMIT` 절, 간단한 조건자 및 복잡한 표현식을 커넥터로 푸시다운하여 스캔하는 데이터와 쿼리 실행 시간을 줄입니다.

LIMIT 절

`LIMIT N` 문은 쿼리로 스캔하는 데이터를 줄입니다. `LIMIT N` 푸시다운을 통해 커넥터는 Athena에 `N`개 행만 반환합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 `WHERE` 절에 사용되는 표현식입니다. Athena MySQL 커넥터는 이러한 표현식을 결합하고 MySQL로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena MySQL 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND, OR, NOT
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL
- 산술: ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- 기타: LIKE_PATTERN, IN

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

MySQL을 비롯한 페더레이션된 쿼리의 성능을 개선하기 위해 조건자 푸시다운을 사용하는 방법에 대한 문서는 AWS 빅 데이터 블로그의 Improve federated queries with predicate pushdown in Amazon Athena를 참조하세요.

패스스루 쿼리

MySQL 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

MySQL에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 MySQL의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 `customer` 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 MySQL 커넥터용 pom.xml 파일을 참조하세요.

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Neptune 커넥터

Amazon Neptune은 빠르고 안정적인 종합 관리형 그래프 데이터베이스 서비스로, 고도로 연결된 데이터 세트를 사용하는 애플리케이션을 쉽게 빌드하고 실행할 수 있습니다. Neptune의 특수 목적 고성능

그래프 데이터베이스 엔진은 수십억 개의 관계를 최적으로 저장하고 단 몇 밀리초의 지연 시간으로 그래프를 쿼리합니다. 자세한 내용은 Neptune 사용 설명서를 참조하세요.

Amazon Athena Neptune 커넥터를 통해 Athena는 Neptune 그래프 데이터베이스 인스턴스와 통신할 수 있고, 이로써 SQL 쿼리로 Neptune 그래프 데이터에 액세스할 수 있습니다.

계정에서 Lake Formation을 활성화한 경우 AWS Serverless Application Repository에 배포한 Athena 페더레이션형 Lambda 커넥터의 IAM 역할은 Lake Formation에서 AWS Glue Data Catalog에 대한 읽기 액세스 권한을 가지고 있어야 합니다.

필수 조건

Neptune 커넥터를 사용하려면 다음 세 단계가 필요합니다.

- Neptune 클러스터 설정
- AWS Glue Data Catalog 설정
- AWS 계정에 커넥터 배포 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요. Neptune 커넥터 배포와 관련된 자세한 내용을 알아보려면 GitHub.com의 Deploy the Amazon Athena Neptune Connector(Amazon Athena Neptune 커넥터 배포)를 참조하세요.

제한 사항

현재 Neptune 커넥터에는 다음과 같은 제한 사항이 있습니다.

- 프라이머리 키(ID)를 비롯한 예측 열은 지원되지 않습니다.

Neptune 클러스터 설정

사용하려는 기존 Amazon Neptune 클러스터 및 속성 그래프 데이터 세트가 없는 경우 하나 설정해야 합니다.

Neptune 클러스터를 호스팅하는 VPC에 인터넷 게이트웨이와 NAT 게이트웨이가 있는지 확인합니다. Neptune 커넥터 Lambda 함수가 사용하는 프라이빗 서브넷에는 이 NAT 게이트웨이를 통한 인터넷 경로가 있어야 합니다. Neptune 커넥터 Lambda 함수는 NAT 게이트웨이를 사용하여 AWS Glue와 통신합니다.

새 Neptune 클러스터를 설정하고 샘플 데이터 세트와 함께 로드하는 방법에 대한 지침은 GitHub.com의 Sample Neptune Cluster Setup(샘플 Neptune 클러스터 설정)을 참조하세요.

AWS Glue Data Catalog 설정

기존의 관계형 데이터 스토어와 달리 Neptune 그래프 DB 노드 및 엣지는 세트 스키마를 사용하지 않습니다. 항목마다 필드와 데이터 형식이 다를 수 있습니다. 그러나 Neptune 커넥터는 AWS Glue Data Catalog에서 메타데이터를 검색하기 때문에 필요한 스키마가 포함된 테이블이 있는 AWS Glue 데이터베이스를 생성해야 합니다. AWS Glue 데이터베이스 및 테이블을 생성한 후 커넥터는 Athena에서 쿼리할 수 있는 테이블 목록을 채울 수 있습니다.

대소문자를 구분하지 않는 열 일치 활성화

AWS Glue에서 열 이름이 모두 소문자인 경우에도 Neptune 테이블의 열 이름을 올바른 대소문자로 확인하려면 대소문자를 구분하지 않도록 Neptune 커넥터를 구성할 수 있습니다.

이 기능을 활성화하려면 Neptune 커넥터 Lambda 함수의 `enable_caseinsensitivematch` 환경 변수를 `true`로 설정합니다.

대소문자를 구분하는 테이블 이름의 경우 AWS Glue glabel 테이블 파라미터 지정

AWS Glue에서는 소문자 테이블 이름만 지원하므로 Neptune에 대해 AWS Glue 테이블을 생성하고 Neptune 테이블 이름이 대소문자를 구분하는 경우 `glabel` AWS Glue 테이블 파라미터를 지정해야 합니다.

AWS Glue 테이블 정의에 `glabel` 파라미터를 포함하고 값을 원래 대소문자를 사용하는 테이블 이름으로 설정합니다. 이렇게 하면 AWS Glue에서 Neptune 테이블과 상호 작용할 때 올바른 대소문자가 유지됩니다. 다음 예제에서는 `glabel`의 값을 테이블 이름 `Airport`로 설정합니다.

```
glabel = Airport
```

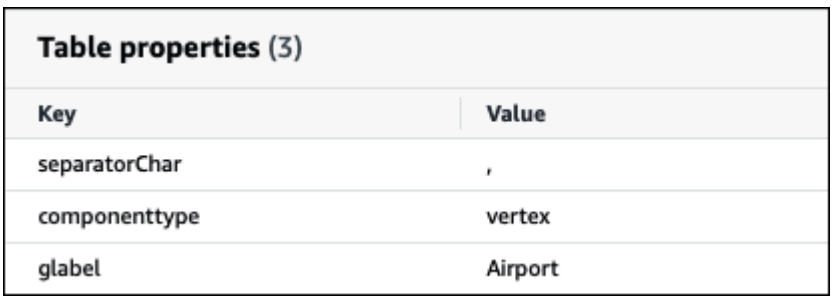


Table properties (3)

Key	Value
separatorChar	,
componenttype	vertex
glabel	Airport

Neptune과 함께 작동하도록 AWS Glue Data Catalog를 설정하는 방법에 대한 자세한 내용은 GitHub.com의 Set up AWS Glue Catalog를 참조하세요.

성능

쿼리에서 스캔하는 데이터를 줄이기 위해 Athena Neptune 커넥터에서 조건자 푸시다운을 수행합니다. 그러나 조건자에서 프라이머리 키를 사용하면 쿼리가 실패하게 됩니다. `LIMIT` 절은 스캔되는 데이터의 양을 줄이지만 조건자를 제공하지 않으면 `LIMIT` 절을 포함하는 `SELECT` 쿼리가 최소 16MB의 데이터를 스캔할 것으로 예상해야 합니다. Neptune 커넥터는 동시성으로 인한 제한에 대한 복원력이 뛰어납니다.

패스스루 쿼리

Neptune 커넥터는 패스스루 쿼리를 지원합니다. 이 특성을 사용하여 속성 그래프에서 Gremlin 쿼리를 실행하고 RDF 데이터에 대해 SPARQL 쿼리를 실행할 수 있습니다.

Neptune에서 패스스루 쿼리를 생성하려면 다음 구문을 사용하세요.

```
SELECT * FROM TABLE(
        system.query(
            DATABASE => 'database_name',
            COLLECTION => 'collection_name',
            QUERY => 'query_string'
        ))
```

다음 예시 Neptune 패스스루 쿼리는 `ATL` 코드가 있는 공항을 필터링합니다.

```
SELECT * FROM TABLE(
        system.query(
            DATABASE => 'graph-database',
            COLLECTION => 'airport',
            QUERY => 'g.V().has(\"airport\", \"code\", \"ATL\").valueMap()',
        ))
```

추가적인 리소스

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena OpenSearch 커넥터

OpenSearch Service

Amazon Athena OpenSearch 커넥터를 통해 Amazon Athena는 OpenSearch 인스턴스와 통신할 수 있고, 이로써 SQL을 사용하여 데이터를 쿼리할 수 있습니다.

> **Note**
>
> 알려진 문제로 인해 OpenSearch 커넥터는 VPC에서 사용할 수 없습니다.

계정에서 Lake Formation을 활성화한 경우 AWS Serverless Application Repository에 배포한 Athena 페더레이션형 Lambda 커넥터의 IAM 역할은 Lake Formation에서 AWS Glue Data Catalog에 대한 읽기 액세스 권한을 가지고 있어야 합니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

용어

다음 용어는 OpenSearch 커넥터와 관련이 있습니다.

- 도메인 - 이 커넥터가 OpenSearch 인스턴스의 엔드포인트와 연결하는 이름입니다. 도메인은 데이터베이스 이름으로도 사용됩니다. Amazon OpenSearch Service 내에 정의된 OpenSearch 인스턴스의 경우 도메인은 자동 검색 가능합니다. 다른 인스턴스의 경우 도메인 이름과 엔드포인트 간의 매핑을 제공해야 합니다.
- 인덱스 - OpenSearch 인스턴스에 정의된 데이터베이스 테이블입니다.
- 매핑 - 인덱스가 데이터베이스 테이블인 경우 매핑은 해당 스키마(즉, 해당 필드 및 속성의 정의)입니다.

  이 커넥터는 OpenSearch 인스턴스와 AWS Glue Data Catalog에서 메타데이터 검색을 모두 지원합니다. 커넥터가 OpenSearch 도메인 및 인덱스 이름과 일치하는 AWS Glue 데이터베이스 및 테이블을 찾으면 커넥터가 스키마 정의에 이들을 사용하려고 시도합니다. OpenSearch 인덱스에 있는 모든 필드의 상위 세트가 되도록 AWS Glue 테이블을 생성하는 것이 좋습니다.
- 문서 – 데이터베이스 테이블 내의 레코드입니다.
- 데이터 스트림 - 여러 백업 인덱스로 구성된 시간 기반 데이터입니다. 자세한 내용은 OpenSearch 설명서의 Data streams 및 Amazon OpenSearch Service 개발자 안내서의 데이터 스트림 시작하기를 참조하세요.

> **Note**
>
> 데이터 스트림 인덱스는 OpenSearch에 의해 내부적으로 생성 및 관리되므로 커넥터는 사용 가능한 첫 번째 인덱스에서 스키마 매핑을 선택합니다. 이러한 이유로 AWS Glue 테이블을 보충 메타데이터 소스로 설정하는 것이 좋습니다. 자세한 내용은 AWS Glue에서 데이터베이스 및 테이블 설정 단원을 참조하십시오.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 OpenSearch 커넥터를 구성합니다.

- spill_bucket – Lambda 함수 제한을 초과하는 데이터에 대한 Amazon S3 버킷을 지정합니다.
- spill_prefix – (선택 사항) 기본값은 `athena-federation-spill`이라는 지정된 `spill_bucket`의 하위 폴더입니다. 미리 정해진 일 수 또는 시간보다 오래된 유출을 삭제하려면 이 위치에서 Amazon S3 스토리지 수명 주기를 구성하는 것이 좋습니다.
- spill_put_request_headers – (선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵입니다(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.
- kms_key_id – (선택 사항) 기본적으로 Amazon S3로 유출된 모든 데이터는 AES-GCM 인증 암호화 모드와 임의로 생성된 키를 사용하여 암호화됩니다. Lambda 함수가 `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`과 같이 KMS에서 생성된 더 강력한 암호화 키를 사용하도록 하려면 KMS 키 ID를 지정합니다.
- disable_spill_encryption – (선택 사항) `True`로 설정하면 유출 암호화가 비활성화됩니다. S3로 유출되는 데이터가 AES-GCM을 사용하여 암호화되도록 기본값은 `False`입니다(임의로 생성된 키 또는 KMS를 사용하여 키 생성). 유출 암호화를 비활성화하면 특히 유출 위치가 서버 측 암호화를 사용하는 경우 성능이 향상될 수 있습니다.
- disable_glue – (선택 사항) true로 설정된 경우 커넥터는 AWS Glue에서 보충 메타데이터 검색을 시도하지 않습니다.
- query_timeout_cluster – 병렬 스캔 생성에 사용되는 클러스터 상태 쿼리의 제한 시간(초)입니다.
- query_timeout_search – 인덱스에서 문서를 검색하는 데 사용되는 검색 쿼리의 제한 시간(초)입니다.
- auto_discover_endpoint – 부울입니다. 기본값은 `true`입니다. Amazon OpenSearch Service를 사용하고 이 파라미터를 true로 설정하면 커넥터가 OpenSearch Service에서 적절한 설명 또는 나열 API 작업을 호출하여 도메인과 엔드포인트를 자동으로 검색할 수 있습니다. 다른 유형의 OpenSearch

인스턴스(예: 자체 호스팅)의 경우 domain_mapping 변수에 연결된 도메인 엔드포인트를 지정해야 합니다. auto_discover_endpoint=true인 경우 커넥터는 AWS 자격 증명을 사용하여 OpenSearch Service에 인증합니다. 그렇지 않으면 커넥터가 domain_mapping 변수를 통해 AWS Secrets Manager에서 사용자 이름 및 암호 자격 증명을 검색합니다.

- domain_mapping – auto_discover_endpoint가 false로 설정되고 도메인 이름과 관련 엔드포인트 간의 매핑을 정의하는 경우에만 사용됩니다. domain_mapping 변수는 다음 형식으로 여러 OpenSearch 엔드포인트를 수용할 수 있습니다.

```
domain1=endpoint1,domain2=endpoint2,domain3=endpoint3,...
```

OpenSearch 엔드포인트에 대한 인증을 위해 커넥터는 ${SecretName}: 형식을 사용하여 삽입된 대체 문자열을 지원하며, 여기에는 AWS Secrets Manager에서 검색한 사용자 이름과 암호가 포함됩니다. 표현식 끝에 있는 콜론(:)은 엔드포인트의 나머지 부분과의 구분자 역할을 합니다.

> **Important**
>
> 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

다음 예제에서는 opensearch-creds 보안 암호를 사용합니다.

```
movies=https://${opensearch-creds}:search-movies-ne...qu.us-east-1.es.amazonaws.com
```

런타임 시 ${opensearch-creds}는 다음 예제와 같이 사용자 이름과 암호로 렌더링됩니다.

```
movies=https://myusername@mypassword:search-movies-ne...qu.us-east-1.es.amazonaws.com
```

domain_mapping 파라미터에서 각 도메인-엔드포인트 페어는 다른 보안 암호를 사용할 수 있습니다. 보안 암호 자체는 *user_name*@*password* 형식으로 지정해야 합니다. 암호에 @ 기호가 포함될 수 있지만 첫 번째 @은 *user_name*과의 구분자 역할을 합니다.

쉼표(,)와 등호(=)가 이 커넥터에서 도메인-엔드포인트 페어의 구분자로 사용된다는 점도 중요합니다. 따라서 저장된 보안 암호 내의 어디에도 쉼표(,)와 등호(=)를 사용하면 안 됩니다.

AWS Glue에서 데이터베이스 및 테이블 설정

커넥터는 AWS Glue 또는 OpenSearch를 사용하여 메타데이터 정보를 가져옵니다. AWS Glue 테이블을 보충 메타데이터 정의 소스로 설정할 수 있습니다. 이 기능을 활성화하려면 보충하려는 소스의 도메인 및 인덱스와 일치하는 AWS Glue 데이터베이스 및 테이블을 정의합니다. 커넥터는 지정된 인덱스에 대한 매핑을 검색하여 OpenSearch 인스턴스에 저장된 메타데이터 정의를 활용할 수도 있습니다.

OpenSearch에서 배열에 대한 메타데이터 정의

OpenSearch에는 전용 배열 데이터 형식이 없습니다. 데이터 형식이 같으면 모든 필드에 0개 이상의 값이 포함될 수 있습니다. OpenSearch를 메타데이터 정의 소스로 사용하려면 목록 또는 배열로 간주될 필드에 대해 Athena와 함께 사용되는 모든 인덱스에 대해 `_meta` 속성을 정의해야 합니다. 이 단계를 완료하지 못하면 쿼리는 목록 필드의 첫 번째 요소만 반환합니다. `_meta` 속성을 지정할 때 필드 이름은 중첩 JSON 구조에 대해 정규화되어야 합니다(예: `address.street` 여기서 `street`는 `address` 구조 내부의 중첩 필드임).

다음 예제는 `movies` 테이블에 `actor` 및 `genre` 목록을 정의합니다.

```
PUT movies/_mapping
{
  "_meta": {
    "actor": "list",
    "genre": "list"
  }
}
```

데이터 타입

OpenSearch 커넥터는 AWS Glue 또는 OpenSearch 인스턴스에서 메타데이터 정의를 추출할 수 있습니다. 커넥터는 다음 표의 매핑을 사용하여 정의를 Apache Arrow 데이터 형식으로 변환합니다. 여기에는 다음 섹션에 언급된 사항이 포함됩니다.

OpenSearch	Apache Arrow	AWS Glue
text, keyword, binary	VARCHAR	문자열
bigint	BIGINT	bigint
scaled_float	BIGINT	SCALED_FLOAT(...)

OpenSearch	Apache Arrow	AWS Glue
정수	INT	int
bigint	SMALLINT	smallint
바이트	TINYINT	tinyint
double	FLOAT8	double
float, half_float	FLOAT4	float
boolean	BIT	boolean
date, date_nanos	DATEMILLI	타임스탬프
JSON 구조	STRUCT	STRUCT
_meta(자세한 내용을 알아보려면 OpenSearch에서 배열에 대한 메타데이터 정의 섹션 참조)	LIST	ARRAY

데이터 형식에 대한 참고 사항

- 현재 커넥터는 앞의 표에 나열된 OpenSearch 및 AWS Glue 데이터 형식만 지원합니다.
- `scaled_float`는 고정 이중 스케일링 팩터로 조정된 부동 소수점 숫자이며 Apache Arrow에서 `BIGINT`로 표시됩니다. 예를 들어, 스케일링 팩터가 100인 0.756은 76으로 반올림됩니다.
- AWS Glue에서 `scaled_float`를 정의하려면 `array` 열 유형을 선택하고 SCALED_FLOAT(*`scaling_factor`*) 형식을 사용하여 필드를 선언해야 합니다.

  다음 예제는 유효합니다.

```
SCALED_FLOAT(10.51)
SCALED_FLOAT(100)
SCALED_FLOAT(100.0)
```

  다음 예제는 유효하지 않습니다.

```
SCALED_FLOAT(10.)
```

```
SCALED_FLOAT(.5)
```

- date_nanos에서 DATEMILLI로 변환할 때 나노초는 가장 가까운 밀리초로 반올림됩니다. date 및 date_nanos에 대한 유효한 값에는 다음 형식이 포함되지만 이에 국한되지 않습니다.

```
"2020-05-18T10:15:30.123456789"
"2020-05-15T06:50:01.123Z"
"2020-05-15T06:49:30.123-05:00"
1589525370001 (epoch milliseconds)
```

- OpenSearch binary는 Base64를 사용하여 인코딩된 바이너리 값의 문자열 표현이며 VARCHAR로 변환됩니다.

SQL 쿼리 실행

다음은 이 커넥터에서 사용할 수 있는 DDL 쿼리의 예제입니다. 예제에서 *function_name*은 Lambda 함수의 이름에 해당하고, *domain*은 쿼리하려는 도메인의 이름이고, *index*는 인덱스의 이름입니다.

```
SHOW DATABASES in `lambda:function_name`
```

```
SHOW TABLES in `lambda:function_name`.domain
```

```
DESCRIBE `lambda:function_name`.domain.index
```

성능

Athena OpenSearch 커넥터는 샤드 기반 병렬 스캔을 지원합니다. 커넥터는 OpenSearch 인스턴스에서 검색된 클러스터 상태 정보를 사용하여 문서 검색 쿼리에 대한 여러 요청을 생성합니다. 요청은 각 샤드에 대해 분할되고 동시에 실행됩니다.

또한 커넥터는 문서 검색 쿼리의 일부로 조건자를 푸시다운합니다. 다음 예제 쿼리 및 조건자는 커넥터가 조건자 푸시다운을 사용하는 방법을 보여줍니다.

Query

```
SELECT * FROM "lambda:elasticsearch".movies.movies
WHERE year >= 1955 AND year <= 1962 OR year = 1996
```

조건자

```
(_exists_:year) AND year:([1955 TO 1962] OR 1996)
```

패스스루 쿼리

OpenSearch 커넥터는 패스스루 쿼리를 지원하고 Query DSL 언어를 사용합니다. Query DSL을 사용한 쿼리에 대한 자세한 내용은 Elasticsearch 설명서의 Query DSL 또는 OpenSearch 설명서의 Query DSL을 참조하세요.

OpenSearch 커넥터에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            schema => 'schema_name',
            index => 'index_name',
            query => "{query_string}"
        ))
```

다음 OpenSearch 예제 패스스루 쿼리는 `default` 스키마의 `employee` 인덱스에서 활성 고용 상태인 직원을 필터링합니다.

```
SELECT * FROM TABLE(
        system.query(
            schema => 'default',
            index => 'employee',
            query => "{ ''bool'':{''filter'':{''term'':{''status'': ''active''}}}}"
        ))
```

추가적인 리소스

- Amazon Athena OpenSearch 커넥터를 사용하여 단일 쿼리로 Amazon OpenSearch Service와 Amazon S3에서 데이터를 쿼리하는 방법에 대한 문서는 AWS 빅 데이터 블로그의 Query data in Amazon OpenSearch Service using SQL from Amazon Athena를 참조하세요.
- 이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Oracle 커넥터

Oracle용 Amazon Athena 커넥터를 사용하면 Amazon Athena가 온프레미스나 Amazon EC2 또는 Amazon RDS에서 실행 중인 Oracle에 저장된 데이터에 대해 SQL 쿼리를 실행할 수 있습니다. 커넥터를 사용하여 Oracle Exadata에 대해 데이터를 쿼리할 수도 있습니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 멀티플렉서 설정에서 유출 버킷과 접두사는 모든 데이터베이스 인스턴스에서 공유됩니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.

용어

다음 용어는 Oracle 커넥터와 관련이 있습니다.

- 데이터베이스 인스턴스 - 온프레미스, Amazon EC2 또는 Amazon RDS에 배포된 데이터베이스의 모든 인스턴스.
- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.
- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - `connection_string` 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.
- 멀티플렉싱 핸들러 - 여러 데이터베이스 연결을 수락하고 사용할 수 있는 Lambda 핸들러.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 Oracle 커넥터를 구성합니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 데이터베이스 인스턴스에 연결합니다.

```
oracle://${jdbc_connection_string}
```

> **Note**
>
> 암호에 특수 문자(예: `some.password`)가 포함된 경우 연결 문자열에 암호를 전달할 때 암호를 큰따옴표로 묶습니다(예: `"some.password"`). 이렇게 하지 않으면 Invalid Oracle URL specified 오류가 발생할 수 있습니다.

멀티플렉싱 핸들러 사용

멀티플렉서를 사용하여 단일 Lambda 함수로 여러 데이터베이스 인스턴스에 연결할 수 있습니다. 요청은 카탈로그 이름을 기준으로 라우팅됩니다. Lambda에서 다음 클래스를 사용합니다.

핸들러	Class
복합 핸들러	`OracleMuxCompositeHandler`
메타데이터 핸들러	`OracleMuxMetadataHandler`
레코드 핸들러	`OracleMuxRecordHandler`

멀티플렉싱 핸들러 파라미터

파라미터	설명
$*catalog*_connection_string	필수 사항입니다. 데이터베이스 인스턴스 연결 문자열. Athena에서 사용되는 카탈로그의 이름을 환경 변수 앞에 붙입니다. 예를 들어, Athena에 등록된 카탈로그가 `myoraclecatalog` 인 경우 환경 변수 이름은 `myoraclecatalog_connection_string` 입니다.
`default`	필수 사항입니다. 기본 연결 문자열. 이 문자열은 카탈로그가 `lambda:${` *AWS_LAMBDA_FUNCTION_NAME* `}`일 때 사용됩니다.

다음은 oracle1(기본값)과 oracle2라는 2개의 데이터베이스 인스턴스를 지원하는 Oracle MUX Lambda 함수에 대한 예제 속성입니다.

속성	값
default	oracle://jdbc:oracle:thin:${Test/RDS/Oracle1}@//oracle1.hostname:port/servicename
oracle_catalog1_connection_string	oracle://jdbc:oracle:thin:${Test/RDS/Oracle1}@//oracle1.hostname:port/servicename
oracle_catalog2_connection_string	oracle://jdbc:oracle:thin:${Test/RDS/Oracle2}@//oracle2.hostname:port/servicename

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

  > **Important**
  >
  > 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 username 및 password 값으로 바꿉니다.

Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

```
{"username": "${username}", "password": "${password}"}
```

> **Note**
>
> 암호에 특수 문자(예: `some.password`)가 포함된 경우 Secrets Manager에 암호를 저장할 때 암호를 큰따옴표로 묶습니다(예: `"some.password"`). 이렇게 하지 않으면 Invalid Oracle URL specified 오류가 발생할 수 있습니다.

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 `${Test/RDS/Oracle}`입니다.

```
oracle://jdbc:oracle:thin:${Test/RDS/Oracle}@//hostname:port/servicename
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
oracle://jdbc:oracle:thin:username/password@//hostname:port/servicename
```

현재 Oracle 커넥터는 `UID` 및 `PWD` JDBC 속성을 인식합니다.

단일 연결 핸들러 사용

다음과 같은 단일 연결 메타데이터 및 레코드 핸들러를 사용하여 단일 Oracle 인스턴스에 연결할 수 있습니다.

핸들러 유형	Class
복합 핸들러	`OracleCompositeHandler`
메타데이터 핸들러	`OracleMetadataHandler`
레코드 핸들러	`OracleRecordHandler`

단일 연결 핸들러 파라미터

파라미터	설명
default	필수 사항입니다. 기본 연결 문자열.
IsFIPSEnabled	선택 사항입니다. FIPS 모드가 활성화될 때 true로 설정합니다. 기본값은 false입니다.

단일 연결 핸들러는 하나의 데이터베이스 인스턴스를 지원하며 default 연결 문자열 파라미터를 제공해야 합니다. 다른 연결 문자열은 모두 무시됩니다.

커넥터는 Amazon RDS 인스턴스에 대한 SSL 기반 연결을 지원합니다. 지원은 전송 계층 보안(TLS) 프로토콜 및 클라이언트에 의한 서버 인증으로 제한됩니다. Amazon RDS에서는 상호 인증이 지원되지 않습니다. 아래 표의 두 번째 행은 SSL 사용 구문을 보여줍니다.

다음은 Lambda 함수에서 지원하는 단일 Oracle 인스턴스에 대한 예제 속성입니다.

속성	값
default	oracle://jdbc:oracle:thin:${Test/RDS/Oracle}@//hostname:port/servicename
	oracle://jdbc:oracle:thin:${Test/RDS/Oracle}@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCPS) (HOST=<HOST_NAME>)(PORT=))(CONNECT_DATA=(SID=))(SECURITY=(SSL_SERVER_CERT_DN=)))

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
spill_bucket	필수 사항입니다. 유출 버킷 이름.
spill_prefix	필수 사항입니다. 유출 버킷 키 접두사.

파라미터	설명
spill_put_request_headers	(선택 사항) 유출에 사용되는 Amazon S3 putObject 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: {"x-amz-server-side-encryption" : "AES256"}). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 JDBC, Oracle 및 Arrow의 해당 데이터 형식이 나와 있습니다.

JDBC	Oracle	화살표
불	boolean	Bit
Integer	N/A	Tiny
Short	smallint	Smallint
Integer	정수	정수
Long	bigint	Bigint
float	"char"	Float4
Double	float8	Float8
날짜	date	DateDay
Timestamp	타임스탬프	DateMilli
String	텍스트	Varchar
바이트	bytes	Varbinary
BigDecimal	numeric(p,s)	10진수

JDBC	Oracle	화살표
ARRAY	해당 사항 없음(참고 사항 참조)	나열

파티션 및 분할

파티션은 커넥터에 대한 분할을 생성하는 방법을 결정하는 데 사용됩니다. Athena는 커넥터가 분할을 생성하는 데 도움이 되도록 테이블에 대한 파티셔닝 체계를 나타내는 `varchar` 유형의 합성 열을 생성합니다. 커넥터는 실제 테이블 정의를 수정하지 않습니다.

성능

Oracle은 기본 파티션을 지원합니다. Athena Oracle 커넥터는 이러한 파티션에서 병렬로 데이터를 검색할 수 있습니다. 파티션 배포가 균일한 초대규모 데이터 세트를 쿼리하려면 기본 파티셔닝을 사용하는 것이 좋습니다. 열의 하위 집합을 선택하면 쿼리 런타임 속도를 높이고 스캔되는 데이터를 줄일 수 있습니다. Oracle 커넥터는 동시성으로 인한 제한에 대한 복원력이 뛰어납니다. 하지만 쿼리 런타임이 길어지는 경향이 있습니다.

쿼리에서 스캔하는 데이터를 줄이기 위해 Athena Oracle 커넥터에서 조건자 푸시다운을 수행합니다. 스캔하는 데이터와 쿼리 실행 시간을 줄이도록 간단한 조건자와 복잡한 표현식을 커넥터로 푸시다운합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 `WHERE` 절에 사용되는 표현식입니다. Athena Oracle 커넥터는 이러한 표현식을 결합하고 Oracle로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena Oracle 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND, OR, NOT
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_NULL
- 산술: ADD, SUBTRACT, MULTIPLY, DIVIDE, NEGATE
- 기타: LIKE_PATTERN, IN

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%');
```

패스스루 쿼리

Oracle 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

Oracle에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 Oracle의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리가 `customer` 테이블의 모든 열을 선택합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer'
        ))
```

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 Oracle 커넥터용 pom.xml 파일을 참조하세요.

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena PostgreSQL 커넥터

Amazon Athena PostgreSQL 커넥터를 사용하면 Athena에서 PostgreSQL 데이터베이스에 액세스할 수 있습니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 멀티플렉서 설정에서 유출 버킷과 접두사는 모든 데이터베이스 인스턴스에서 공유됩니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.

용어

다음 용어는 PostgreSQL 커넥터와 관련이 있습니다.

- 데이터베이스 인스턴스 - 온프레미스, Amazon EC2 또는 Amazon RDS에 배포된 데이터베이스의 모든 인스턴스.
- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.
- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - `connection_string` 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.

- 멀티플렉싱 핸들러 - 여러 데이터베이스 연결을 수락하고 사용할 수 있는 Lambda 핸들러.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 PostgreSQL 커넥터를 구성합니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 데이터베이스 인스턴스에 연결합니다.

```
postgres://${jdbc_connection_string}
```

멀티플렉싱 핸들러 사용

멀티플렉서를 사용하여 단일 Lambda 함수로 여러 데이터베이스 인스턴스에 연결할 수 있습니다. 요청은 카탈로그 이름을 기준으로 라우팅됩니다. Lambda에서 다음 클래스를 사용합니다.

핸들러	Class
복합 핸들러	`PostGreSqlMuxCompositeHandler`
메타데이터 핸들러	`PostGreSqlMuxMetadataHandler`
레코드 핸들러	`PostGreSqlMuxRecordHandler`

멀티플렉싱 핸들러 파라미터

파라미터	설명
`$`*`catalog`*`_connection_string`	필수 사항입니다. 데이터베이스 인스턴스 연결 문자열. Athena에서 사용되는 카탈로그의 이름을 환경 변수 앞에 붙입니다. 예를 들어, Athena에 등록된 카탈로그가 `mypostgrescatalog` 인 경우 환경 변수 이름은 `mypostgrescatalog_connection_string` 입니다.
`default`	필수 사항입니다. 기본 연결 문자열. 이 문자열은 카탈로그가 `lambda:${`*`AWS_LAMBDA_FUNCTION_NAME`*`}`일 때 사용됩니다.

다음은 postgres1(기본값)과 postgres2라는 2개의 데이터베이스 인스턴스를 지원하는 PostGreSql MUX Lambda 함수에 대한 예제 속성입니다.

속성	값
default	postgres://jdbc:postgresql://postgres1.host:5432/default?${Test/RDS/PostGres1}
postgres_catalog1_connection_string	postgres://jdbc:postgresql://postgres1.host:5432/default?${Test/RDS/PostGres1}
postgres_catalog2_connection_string	postgres://jdbc:postgresql://postgres2.host:5432/default?user=sample&password=sample

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

  > **Important**
  >
  > 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 username 및 password 값으로 바꿉니다.

Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

```
{"username": "${username}", "password": "${password}"}
```

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 `${Test/RDS/PostGres1}`입니다.

```
postgres://jdbc:postgresql://postgres1.host:5432/default?...&${Test/RDS/PostGres1}&...
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
postgres://jdbc:postgresql://postgres1.host:5432/
default?...&user=sample2&password=sample2&...
```

현재 PostgreSQL 커넥터는 `user` 및 `password` JDBC 속성을 인식합니다.

SSL 활성화

PostgreSQL 연결에서 SSL을 지원하려면 연결 문자열에 다음을 추가합니다.

```
&sslmode=verify-ca&sslfactory=org.postgresql.ssl.DefaultJavaSSLFactory
```

예

다음 연결 문자열 예제에서는 SSL을 사용하지 않습니다.

```
postgres://jdbc:postgresql://example-asdf-aurora-postgres-endpoint:5432/asdf?
user=someuser&password=somepassword
```

SSL을 활성화하려면 다음과 같이 문자열을 수정합니다.

```
postgres://jdbc:postgresql://example-asdf-aurora-postgres-
endpoint:5432/asdf?user=someuser&password=somepassword&sslmode=verify-
ca&sslfactory=org.postgresql.ssl.DefaultJavaSSLFactory
```

단일 연결 핸들러 사용

다음과 같은 단일 연결 메타데이터 및 레코드 핸들러를 사용하여 단일 PostgreSQL 인스턴스에 연결할 수 있습니다.

핸들러 유형	Class
복합 핸들러	`PostGreSqlCompositeHandler`
메타데이터 핸들러	`PostGreSqlMetadataHandler`
레코드 핸들러	`PostGreSqlRecordHandler`

단일 연결 핸들러 파라미터

파라미터	설명
`default`	필수 사항입니다. 기본 연결 문자열.

단일 연결 핸들러는 하나의 데이터베이스 인스턴스를 지원하며 `default` 연결 문자열 파라미터를 제공해야 합니다. 다른 연결 문자열은 모두 무시됩니다.

다음은 Lambda 함수에서 지원하는 단일 PostgreSQL 인스턴스에 대한 예제 속성입니다.

속성	값
`default`	`postgres://jdbc:postgresql://postgres1.host:5432/default?secret=${Test/RDS/PostgreSQL1}`

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
`spill_bucket`	필수 사항입니다. 유출 버킷 이름.

파라미터	설명
spill_prefix	필수 사항입니다. 유출 버킷 키 접두사.
spill_put_request_headers	(선택 사항) 유출에 사용되는 Amazon S3 putObject 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: {"x-amz-server-side-encryption" : "AES256"}). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 JDBC, PostGreSQL 및 Arrow의 해당 데이터 형식이 나와 있습니다.

JDBC	PostGreSQL	화살표
불	불	Bit
Integer	N/A	Tiny
Short	smallint	Smallint
Integer	정수	정수
Long	bigint	Bigint
float	"char"	Float4
Double	float8	Float8
날짜	date	DateDay
Timestamp	타임스탬프	DateMilli
String	텍스트	Varchar
바이트	bytes	Varbinary
BigDecimal	numeric(p,s)	10진수

JDBC	PostGreSQL	화살표
ARRAY	해당 사항 없음(참고 사항 참조)	나열

> **Note**
>
> ARRAY 형식은 다차원 배열(`<data_type>[][]` 또는 중첩 배열)은 지원되지 않는다는 제약 조건이 있는 PostgreSQL 커넥터에 대해 지원됩니다. 지원되지 않는 ARRAY 데이터 형식이 있는 열은 문자열 요소의 배열(`array<varchar>`)로 변환됩니다.

파티션 및 분할

파티션은 커넥터에 대한 분할을 생성하는 방법을 결정하는 데 사용됩니다. Athena는 커넥터가 분할을 생성하는 데 도움이 되도록 테이블에 대한 파티셔닝 체계를 나타내는 `varchar` 유형의 합성 열을 생성합니다. 커넥터는 실제 테이블 정의를 수정하지 않습니다.

성능

PostgreSQL은 기본 파티션을 지원합니다. Athena PostgreSQL 커넥터는 이러한 파티션에서 병렬로 데이터를 검색할 수 있습니다. 파티션 배포가 균일한 초대규모 데이터 세트를 쿼리하려면 기본 파티셔닝을 사용하는 것이 좋습니다.

Athena PostgreSQL 커넥터는 조건부 푸시다운을 수행하여 쿼리에서 스캔하는 데이터를 줄입니다. `LIMIT` 절, 간단한 조건자 및 복잡한 표현식을 커넥터로 푸시다운하여 스캔하는 데이터와 쿼리 실행 시간을 줄입니다. 그러나 열 하위 세트를 선택할 때 쿼리 실행 런타임이 길어지는 경우가 있습니다.

LIMIT 절

`LIMIT N` 문은 쿼리로 스캔하는 데이터를 줄입니다. `LIMIT N` 푸시다운을 통해 커넥터는 Athena에 `N`개 행만 반환합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 `WHERE` 절에 사용되는 표현식입니다. Athena PostgreSQL 커넥터는 이러한 표현식을 결합하고 PostgreSQL로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena PostgreSQL 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND, OR, NOT
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL
- 산술: ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- 기타: LIKE_PATTERN, IN

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

패스스루 쿼리

PostgreSQL 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

PostgreSQL에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 PostgreSQL의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 customer 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

추가적인 리소스

최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 PostgreSQL 커넥터용 pom.xml 파일을 참조하세요.

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Redis 커넥터

Amazon Redis 커넥터를 통해 Amazon Athena는 Redis 인스턴스와 통신할 수 있고, 이로써 SQL을 사용하여 Redis 데이터를 쿼리할 수 있습니다. AWS Glue Data Catalog를 사용하여 Redis 키-값 쌍을 가상 테이블에 매핑할 수 있습니다.

기존의 관계형 데이터 스토어와 달리 Redis에는 테이블이나 열의 개념이 없습니다. 대신 Redis는 키가 기본적으로 `string`이고 값이 `string`, `z-set` 또는 `hmap`인 키-값 액세스 패턴을 제공합니다.

AWS Glue Data Catalog를 사용하여 스키마를 생성하고 가상 테이블을 구성할 수 있습니다. Athena Redis 커넥터는 특수 테이블 속성을 통해 Redis 키와 값을 테이블에 매핑하는 방법을 알 수 있습니다. 자세한 내용은 이 문서의 후반부에서 AWS Glue에서 데이터베이스 및 테이블 설정 단원을 참조하세요.

계정에서 Lake Formation을 활성화한 경우 AWS Serverless Application Repository에 배포한 Athena 페더레이션형 Lambda 커넥터의 IAM 역할은 Lake Formation에서 AWS Glue Data Catalog에 대한 읽기 액세스 권한을 가지고 있어야 합니다.

Amazon Athena Redis 커넥터는 Amazon MemoryDB for Redis 및 Amazon ElastiCache for Redis를 지원합니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.
- 이 커넥터를 사용하기 전에 VPC와 보안 그룹을 설정합니다. 자세한 내용은 데이터 원본 커넥터용 VPC 생성 단원을 참조하십시오.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 Redis 커넥터를 구성합니다.

- spill_bucket – Lambda 함수 제한을 초과하는 데이터에 대한 Amazon S3 버킷을 지정합니다.

- spill_prefix – (선택 사항) 기본값은 `athena-federation-spill`이라는 지정된 `spill_bucket`의 하위 폴더입니다. 미리 정해진 일 수 또는 시간보다 오래된 유출을 삭제하려면 이 위치에서 Amazon S3 스토리지 수명 주기를 구성하는 것이 좋습니다.
- spill_put_request_headers – (선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵입니다(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.
- kms_key_id – (선택 사항) 기본적으로 Amazon S3로 유출된 모든 데이터는 AES-GCM 인증 암호화 모드와 임의로 생성된 키를 사용하여 암호화됩니다. Lambda 함수가 `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`과 같이 KMS에서 생성된 더 강력한 암호화 키를 사용하도록 하려면 KMS 키 ID를 지정합니다.
- disable_spill_encryption – (선택 사항) `True`로 설정하면 유출 암호화가 비활성화됩니다. S3로 유출되는 데이터가 AES-GCM을 사용하여 암호화되도록 기본값은 `False`입니다(임의로 생성된 키 또는 KMS를 사용하여 키 생성). 유출 암호화를 비활성화하면 특히 유출 위치가 서버 측 암호화를 사용하는 경우 성능이 향상될 수 있습니다.
- glue_catalog – (선택 사항) 이 옵션을 사용하여 크로스 계정 AWS Glue 카탈로그를 지정합니다. 기본적으로 커넥터는 자체 AWS Glue 계정에서 메타데이터를 가져오려고 시도합니다.

AWS Glue에서 데이터베이스 및 테이블 설정

Redis에서 사용할 AWS Glue 테이블을 활성화하려면 테이블에서 `redis-endpoint`, `redis-value-type` 및 `redis-keys-zset` 또는 `redis-key-prefix`와 같은 테이블 속성을 설정합니다.

또한 Redis 테이블이 포함된 모든 AWS Glue 데이터베이스에는 데이터베이스의 URI 속성에 `redis-db-flag`가 있어야 합니다. `redis-db-flag` URI 속성을 설정하려면 AWS Glue 콘솔을 사용하여 데이터베이스를 편집합니다.

다음 목록에서는 테이블 속성을 설명합니다.

- redis-endpoint – (필수) 이 테이블에 대한 데이터가 포함된 Redis 서버의 *`hostname`*`:`*`port`*`:`*`password`*(예: `athena-federation-demo.cache.amazonaws.com:6379`). 또는 ${*`Secret_Name`*}을 테이블 속성 값으로 사용하여 AWS Secrets Manager에 엔드포인트 또는 엔드포인트의 일부를 저장할 수 있습니다.

> **Note**
>
> AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Lambda 함수에 연결된 VPC에 Secrets Manager에 연결할 VPC 엔드포인트 또는 인터넷 액세스 가 있어야 합니다.

- redis-keys-zset – (`redis-key-prefix`가 사용되지 않는 경우 필수) 값이 zset인 쉼표로 구분된 키 목록입니다(예: `active-orders,pending-orders`). zset의 각 값은 테이블의 일부인 키로 처리됩니다. `redis-keys-zset` 속성 또는 `redis-key-prefix` 속성을 설정해야 합니다.
- redis-key-prefix – (`redis-keys-zset`가 사용되지 않는 경우 필수) 테이블에서 값을 스캔하기 위한 쉼표로 구분된 키 접두사 목록입니다(예: `accounts-*,acct-`). `redis-key-prefix` 속성 또는 `redis-keys-zset` 속성을 설정해야 합니다.
- redis-value-type – (필수) `redis-key-prefix` 또는 `redis-keys-zset`에 의해 정의된 키의 값이 테이블에 매핑되는 방법을 정의합니다. 리터럴은 단일 열에 매핑됩니다. zset도 단일 열에 매핑되지만 각 키는 많은 행을 저장할 수 있습니다. 해시를 사용하면 각 키가 여러 열(예: 해시, 리터럴 또는 zset)이 있는 행이 될 수 있습니다.
- redis-ssl-flag – (선택 사항) `True`이면 SSL/TLS를 사용하는 Redis 연결을 생성합니다. 기본값은 `False`입니다.
- redis-cluster-flag – (선택 사항) `True`이면 클러스터링된 Redis 인스턴스에 대한 지원을 활성화합니다. 기본값은 `False`입니다.
- redis-db-number – (선택 사항) 클러스터링되지 않은 독립 실행형 인스턴스에만 적용됩니다. 기본값이 아닌 Redis 데이터베이스에서 읽으려면 이 번호(예: 1, 2 또는 3)를 설정합니다. 기본값은 Redis 논리적 데이터베이스 0입니다. 이 번호는 Athena 또는 AWS Glue의 데이터베이스가 아니라 Redis 논리 데이터베이스를 나타냅니다. 자세한 내용을 알아보려면 Redis 설명서의 SELECT 인덱스를 참조하세요.

데이터 타입

Redis 커넥터는 다음 데이터 형식을 지원합니다. Redis 스트림은 지원되지 않습니다.

- 문자열
- 해시
- 정렬된 세트(ZSet)

모든 Redis 값은 `string` 데이터 형식으로 검색됩니다. 그런 다음 AWS Glue Data Catalog에서 테이블이 정의된 방식에 따라 다음 Apache Arrow 데이터 형식 중 하나로 변환됩니다.

AWS Glue 데이터 유형	Apache Arrow 데이터 형식
int	INT
문자열	VARCHAR
bigint	BIGINT
double	FLOAT8
float	FLOAT4
smallint	SMALLINT
tinyint	TINYINT
boolean	BIT
이진수	VARBINARY

필수 권한

이 커넥터에 필요한 IAM 정책에 대한 자세한 내용을 알아보려면 athena-redis.yaml 파일의 `Policies` 섹션을 검토하세요. 다음 목록에 필요한 권한이 요약되어 있습니다.

- Amazon S3 쓰기 액세스 - 대규모 쿼리의 결과 유출을 위해서는 커넥터에 Amazon S3 위치에 대한 쓰기 액세스 권한이 필요합니다.
- Athena GetQueryExecution - 커넥터는 업스트림 Athena 쿼리가 종료된 경우 이 권한을 사용하여 빠른 실패를 수행합니다.
- AWS Glue Data Catalog - 스키마 정보를 가져오기 위해 Redis 커넥터에 AWS Glue Data Catalog에 대한 읽기 전용 액세스 권한이 필요합니다.
- CloudWatch Logs - 로그를 저장하기 위해 커넥터 CloudWatch Logs에 대한 액세스 권한이 필요합니다.
- AWS Secrets Manager 읽기 액세스 - Secrets Manager에 Redis 엔드포인트 세부 정보를 저장하기로 선택하는 경우 커넥터에 해당 보안 암호에 대한 액세스 권한을 부여해야 합니다.

- VPC 액세스 - VPC에 연결하고 Redis 인스턴스와 통신할 수 있도록 VPC에 인터페이스를 연결하고 분리하는 기능이 커넥터에 필요합니다.

성능

Athena Redis 커넥터는 사용자가 정의한 테이블 유형(예: zset 키 또는 접두사 키)에 따라 Redis 인스턴스에 대한 쿼리 병렬화를 시도합니다.

쿼리에서 스캔하는 데이터를 줄이기 위해 Athena Redis 커넥터에서 조건자 푸시다운을 수행합니다. 그러나 프라이머리 키에 대한 조건자를 포함하는 쿼리는 시간 초과로 실패합니다. `LIMIT` 절은 스캔되는 데이터의 양을 줄이지만 조건자를 제공하지 않으면 `LIMIT` 절을 포함하는 `SELECT` 쿼리가 최소 16MB의 데이터를 스캔할 것으로 예상해야 합니다. Redis 커넥터는 동시성으로 인한 제한에 대한 복원력이 뛰어납니다.

라이선스 정보

Amazon Athena Redis 커넥터 프로젝트는 Apache-2.0 라이선스에 따라 사용이 허가됩니다.

추가적인 리소스

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Redshift 커넥터

Amazon Athena Redshift 커넥터를 사용하면 Amazon Athena에서 Amazon Redshift 데이터베이스 및 Redshift Serverless 뷰를 포함하는 Amazon Redshift Serverless 데이터베이스에 액세스할 수 있습니다. 이 페이지에 설명된 JDBC 연결 문자열 구성 설정을 사용하여 두 서비스 중 하나에 연결할 수 있습니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 멀티플렉서 설정에서 유출 버킷과 접두사는 모든 데이터베이스 인스턴스에서 공유됩니다.

- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.
- Redshift에서 외부 파티션을 지원하지 않기 때문에 쿼리에서 지정된 모든 데이터가 매번 검색됩니다.

용어

다음 용어는 Redshift 커넥터와 관련이 있습니다.

- 데이터베이스 인스턴스 - 온프레미스, Amazon EC2 또는 Amazon RDS에 배포된 데이터베이스의 모든 인스턴스.
- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.
- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - `connection_string` 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.
- 멀티플렉싱 핸들러 - 여러 데이터베이스 연결을 수락하고 사용할 수 있는 Lambda 핸들러.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 Redshift 커넥터를 구성합니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 데이터베이스 인스턴스에 연결합니다.

```
redshift://${jdbc_connection_string}
```

멀티플렉싱 핸들러 사용

멀티플렉서를 사용하여 단일 Lambda 함수로 여러 데이터베이스 인스턴스에 연결할 수 있습니다. 요청은 카탈로그 이름을 기준으로 라우팅됩니다. Lambda에서 다음 클래스를 사용합니다.

핸들러	Class
복합 핸들러	`RedshiftMuxCompositeHandler`
메타데이터 핸들러	`RedshiftMuxMetadataHandler`
레코드 핸들러	`RedshiftMuxRecordHandler`

멀티플렉싱 핸들러 파라미터

파라미터	설명
$*catalog*_connection_string	필수 사항입니다. 데이터베이스 인스턴스 연결 문자열. Athena에서 사용되는 카탈로그의 이름을 환경 변수 앞에 붙입니다. 예를 들어, Athena에 등록된 카탈로그가 `myredshiftcatalog` 인 경우 환경 변수 이름은 `myredshiftcatalog_connection_string` 입니다.
`default`	필수 사항입니다. 기본 연결 문자열. 이 문자열은 카탈로그가 `lambda:${` *AWS_LAMBDA_FUNCTION_NAME* `}`일 때 사용됩니다.

다음은 `redshift1`(기본값)과 `redshift2`라는 2개의 데이터베이스 인스턴스를 지원하는 Redshift MUX Lambda 함수에 대한 예제 속성입니다.

속성	값
`default`	`redshift://jdbc:redshift://redshift1.host:5439/dev?user=sample2&password=sample2`
`redshift_catalog1_connection_string`	`redshift://jdbc:redshift://redshift1.host:3306/default?${Test/RDS/Redshift1}`

속성	값
`redshift_catalog2_connectio n_string`	`redshift://jdbc:redshift:// redshift2.host:3333/default? user=sample2&password=sample2`

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

  > **Important**
  >
  > 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 `username` 및 `password` 값으로 바꿉니다.

  Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

  ```
  {"username": "${username}", "password": "${password}"}
  ```

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 ${Test/RDS/ `Redshift1`}입니다.

```
redshift://jdbc:redshift://redshift1.host:3306/default?...&${Test/RDS/Redshift1}&...
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
redshift://jdbc:redshift://redshift1.host:3306/
default?...&user=sample2&password=sample2&...
```

현재 Redshift 커넥터는 `user` 및 `password` JDBC 속성을 인식합니다.

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
`spill_bucket`	필수 사항입니다. 유출 버킷 이름.
`spill_prefix`	필수 사항입니다. 유출 버킷 키 접두사.
`spill_put_request_headers`	(선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 JDBC와 Apache Arrow의 해당 데이터 형식이 나와 있습니다.

JDBC	화살표
불	Bit
Integer	Tiny

JDBC	화살표
Short	Smallint
Integer	정수
Long	Bigint
float	Float4
Double	Float8
날짜	DateDay
Timestamp	DateMilli
String	Varchar
바이트	Varbinary
BigDecimal	10진수
ARRAY	나열

파티션 및 분할

Redshift는 외부 파티션을 지원하지 않습니다. 성능 관련 문제에 대한 자세한 내용을 알아보려면 성능 섹션을 참조하세요.

성능

Athena Redshift 커넥터는 조건부 푸시다운을 수행하여 쿼리에서 스캔하는 데이터를 줄입니다. `LIMIT` 절, `ORDER BY` 절, 간단한 조건자 및 복잡한 표현식을 커넥터로 푸시다운하여 스캔하는 데이터와 쿼리 실행 시간을 줄입니다. 그러나 열 하위 세트를 선택할 때 쿼리 실행 런타임이 길어지는 경우가 있습니다. Amazon Redshift는 여러 쿼리를 동시에 실행할 때 특히 쿼리 실행 속도 저하에 취약합니다.

LIMIT 절

`LIMIT N` 문은 쿼리로 스캔하는 데이터를 줄입니다. `LIMIT N` 푸시다운을 통해 커넥터는 Athena에 `N`개 행만 반환합니다.

상위 N개 쿼리

상위 N개 쿼리는 결과 세트의 순서와 반환되는 행 수에 대한 한도를 지정합니다. 이 유형의 쿼리를 사용하여 데이터 세트에 대해 상위 N개의 최댓값 또는 상위 N개의 최솟값을 결정할 수 있습니다. 상위 N개의 푸시다운을 통해 커넥터는 Athena에 N개의 정렬된 행만 반환합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 WHERE 절에 사용되는 표현식입니다. Athena Redshift 커넥터는 이러한 표현식을 결합하고 Redshift로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena Redshift 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND, OR, NOT
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL
- 산술: ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- 기타: LIKE_PATTERN, IN

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
ORDER BY col_a DESC
LIMIT 10;
```

Amazon Redshift를 비롯한 페더레이션된 쿼리의 성능을 개선하기 위해 조건자 푸시다운을 사용하는 방법에 대한 문서는 AWS 빅 데이터 블로그의 Improve federated queries with predicate pushdown in Amazon Athena를 참조하세요.

패스스루 쿼리

Redshift 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

Redshift에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 Redshift의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 `customer` 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

추가적인 리소스

최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 Redshift 커넥터용 pom.xml 파일을 참조하세요.

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena SAP HANA 커넥터

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 멀티플렉서 설정에서 유출 버킷과 접두사는 모든 데이터베이스 인스턴스에서 공유됩니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.
- SAP HANA에서 객체 이름은 SAP HANA 데이터베이스에 저장될 때 대문자로 변환됩니다. 그러나 따옴표 안의 이름은 대/소문자를 구분하기 때문에 두 테이블이 이름은 같지만 대/소문자가 다를 수 있습니다(예: `EMPLOYEE` 및 `employee`).

Athena 연합 쿼리에서 스키마 테이블 이름은 소문자로 Lambda 함수에 제공됩니다. 이 문제를 해결하려면 `@schemaCase` 쿼리 힌트를 제공하여 대/소문자를 구분하는 이름을 가진 테이블에서 데이터를 검색합니다. 다음은 쿼리 힌트가 포함된 두 가지 샘플 쿼리입니다.

```
SELECT *
FROM "lambda:saphanaconnector".SYSTEM."MY_TABLE@schemaCase=upper&tableCase=upper"
```

```
SELECT *
FROM "lambda:saphanaconnector".SYSTEM."MY_TABLE@schemaCase=upper&tableCase=lower"
```

용어

다음 용어는 SAP HANA 커넥터와 관련이 있습니다.

- 데이터베이스 인스턴스 - 온프레미스, Amazon EC2 또는 Amazon RDS에 배포된 데이터베이스의 모든 인스턴스.
- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.
- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - `connection_string` 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.
- 멀티플렉싱 핸들러 - 여러 데이터베이스 연결을 수락하고 사용할 수 있는 Lambda 핸들러.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 SAP HANA 커넥터를 구성합니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 데이터베이스 인스턴스에 연결합니다.

```
saphana://${jdbc_connection_string}
```

멀티플렉싱 핸들러 사용

멀티플렉서를 사용하여 단일 Lambda 함수로 여러 데이터베이스 인스턴스에 연결할 수 있습니다. 요청은 카탈로그 이름을 기준으로 라우팅됩니다. Lambda에서 다음 클래스를 사용합니다.

핸들러	Class
복합 핸들러	SaphanaMuxCompositeHandler
메타데이터 핸들러	SaphanaMuxMetadataHandler
레코드 핸들러	SaphanaMuxRecordHandler

멀티플렉싱 핸들러 파라미터

파라미터	설명
$*catalog*_connection_string	필수 사항입니다. 데이터베이스 인스턴스 연결 문자열. Athena에서 사용되는 카탈로그의 이름을 환경 변수 앞에 붙입니다. 예를 들어, Athena에 등록된 카탈로그가 mysaphanacatalog 인 경우 환경 변수 이름은 mysaphanacatalog_connection_string 입니다.
default	필수 사항입니다. 기본 연결 문자열. 이 문자열은 카탈로그가 lambda:${ *AWS_LAMBDA_FUNCTION_NAME* }일 때 사용됩니다.

다음은 saphana1(기본값)과 saphana2라는 2개의 데이터베이스 인스턴스를 지원하는 Saphana MUX Lambda 함수에 대한 예제 속성입니다.

속성	값
default	saphana://jdbc:sap://saphana1.host:port/?${Test/RDS/Saphana1}
saphana_catalog1_connection_string	saphana://jdbc:sap://saphana1.host:port/?${Test/RDS/Saphana1}
saphana_catalog2_connection_string	saphana://jdbc:sap://saphana2.host:port/?user=sample2&password=sample2

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

  > **Important**
  >
  > 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 `username` 및 `password` 값으로 바꿉니다.

  Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

```
{"username": "${username}", "password": "${password}"}
```

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 ${Test/RDS/Saphana1}입니다.

```
saphana://jdbc:sap://saphana1.host:port/?${Test/RDS/Saphana1}&...
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
saphana://jdbc:sap://saphana1.host:port/?user=sample2&password=sample2&...
```

현재 SAP HANA 커넥터는 user 및 password JDBC 속성을 인식합니다.

단일 연결 핸들러 사용

다음과 같은 단일 연결 메타데이터 및 레코드 핸들러를 사용하여 단일 SAP HANA 인스턴스에 연결할 수 있습니다.

핸들러 유형	Class
복합 핸들러	SaphanaCompositeHandler
메타데이터 핸들러	SaphanaMetadataHandler
레코드 핸들러	SaphanaRecordHandler

단일 연결 핸들러 파라미터

파라미터	설명
default	필수 사항입니다. 기본 연결 문자열.

단일 연결 핸들러는 하나의 데이터베이스 인스턴스를 지원하며 default 연결 문자열 파라미터를 제공해야 합니다. 다른 연결 문자열은 모두 무시됩니다.

다음은 Lambda 함수에서 지원하는 단일 SAP HANA 인스턴스에 대한 예제 속성입니다.

속성	값
default	saphana://jdbc:sap://saphana1.host:port/?secret=Test/RDS/Saphana1

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
spill_bucket	필수 사항입니다. 유출 버킷 이름.
spill_prefix	필수 사항입니다. 유출 버킷 키 접두사.
spill_put_request_headers	(선택 사항) 유출에 사용되는 Amazon S3 putObject 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: {"x-amz-server-side-encryption" : "AES256"}). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 JDBC와 Apache Arrow의 해당 데이터 형식이 나와 있습니다.

JDBC	화살표
불	Bit
Integer	Tiny
Short	Smallint
Integer	정수

JDBC	화살표
Long	Bigint
float	Float4
Double	Float8
날짜	DateDay
Timestamp	DateMilli
String	Varchar
바이트	Varbinary
BigDecimal	10진수
ARRAY	나열

데이터 형식 변환

JDBC에서 Arrow로의 변환 외에도 커넥터는 SAP HANA 소스와 Athena 데이터 형식이 호환되도록 다른 특정 변환을 수행합니다. 이러한 변환은 쿼리가 성공적으로 실행되도록 하는 데 도움이 됩니다. 다음 표에 이러한 변환이 나와 있습니다.

소스 데이터 형식(SAP HANA)	변환된 데이터 형식(Athena)
DECIMAL	BIGINT
INTEGER	INT
날짜	DATEDAY
TIMESTAMP	DATEMILLI

지원되지 않는 다른 모든 데이터 형식은 VARCHAR로 변환됩니다.

파티션 및 분할

파티션은 `Integer` 형식의 단일 파티션 열로 표시됩니다. 열에는 SAP HANA 테이블에 정의된 파티션의 이름이 들어 있습니다. 파티션 이름이 없는 테이블의 경우 단일 파티션에 해당하는 *가 반환됩니다. 파티션은 분할과 동일합니다.

명칭	유형	설명
PART_ID	Integer	SAP HANA의 명명된 파티션.

성능

SAP HANA는 기본 파티션을 지원합니다. Athena SAP HANA 커넥터는 이러한 파티션에서 병렬로 데이터를 검색할 수 있습니다. 파티션 배포가 균일한 초대규모 데이터 세트를 쿼리하려면 기본 파티셔닝을 사용하는 것이 좋습니다. 열의 하위 집합을 선택하면 쿼리 런타임 속도를 높이고 스캔되는 데이터를 줄일 수 있습니다. 이 커넥터는 동시성으로 인해 상당한 제한을 나타나며 경우에 따라 쿼리 오류가 발생할 수 있습니다.

Athena SAP HANA 커넥터는 조건부 푸시다운을 수행하여 쿼리에서 스캔하는 데이터를 줄입니다. `LIMIT` 절, 간단한 조건자 및 복잡한 표현식을 커넥터로 푸시다운하여 스캔하는 데이터와 쿼리 실행 시간을 줄입니다.

LIMIT 절

`LIMIT N` 문은 쿼리로 스캔하는 데이터를 줄입니다. `LIMIT N` 푸시다운을 통해 커넥터는 Athena에 `N`개 행만 반환합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 `WHERE` 절에 사용되는 표현식입니다. Athena SAP HANA 커넥터는 이러한 표현식을 결합하고 SAP HANA로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena SAP HANA 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND, OR, NOT
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL

- 산술: ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- 기타: LIKE_PATTERN, IN

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

패스스루 쿼리

SAP HANA 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

SAP HANA에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 SAP HANA의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 `customer` 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 SAP HANA 커넥터용 pom.xml 파일을 참조하세요.

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Snowflake 커넥터

Snowflake용 Amazon Athena 커넥터를 사용하면 Amazon Athena가 JDBC를 사용하여 Snowflake SQL 데이터베이스 또는 RDS 인스턴스에 저장된 데이터에 대해 SQL 쿼리를 실행할 수 있습니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 멀티플렉서 설정에서 유출 버킷과 접두사는 모든 데이터베이스 인스턴스에서 공유됩니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.
- 현재 단일 분할이 포함된 Snowflake 뷰가 지원됩니다.
- Snowflake에서 객체 이름은 대/소문자를 구분하기 때문에 두 테이블이 이름은 같지만 대/소문자가 다를 수 있습니다(예: `EMPLOYEE` 및 `employee`). Athena 연합 쿼리에서 스키마 테이블 이름은 소문자로 Lambda 함수에 제공됩니다. 이 문제를 해결하려면 `@schemaCase` 쿼리 힌트를 제공하여 대/소문자를 구분하는 이름을 가진 테이블에서 데이터를 검색합니다. 다음은 쿼리 힌트가 포함된 두 가지 샘플 쿼리입니다.

```
SELECT *
FROM "lambda:snowflakeconnector".SYSTEM."MY_TABLE@schemaCase=upper&tableCase=upper"
```

```
SELECT *
FROM "lambda:snowflakeconnector".SYSTEM."MY_TABLE@schemaCase=upper&tableCase=lower"
```

용어

다음 용어는 Snowflake 커넥터와 관련이 있습니다.

- 데이터베이스 인스턴스 - 온프레미스, Amazon EC2 또는 Amazon RDS에 배포된 데이터베이스의 모든 인스턴스.
- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.
- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - `connection_string` 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.
- 멀티플렉싱 핸들러 - 여러 데이터베이스 연결을 수락하고 사용할 수 있는 Lambda 핸들러.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 Snowflake 커넥터를 구성합니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 데이터베이스 인스턴스에 연결합니다.

```
snowflake://${jdbc_connection_string}
```

멀티플렉싱 핸들러 사용

멀티플렉서를 사용하여 단일 Lambda 함수로 여러 데이터베이스 인스턴스에 연결할 수 있습니다. 요청은 카탈로그 이름을 기준으로 라우팅됩니다. Lambda에서 다음 클래스를 사용합니다.

핸들러	Class
복합 핸들러	`SnowflakeMuxCompositeHandler`

핸들러	Class
메타데이터 핸들러	SnowflakeMuxMetadataHandler
레코드 핸들러	SnowflakeMuxRecordHandler

멀티플렉싱 핸들러 파라미터

파라미터	설명
$*catalog*_connection_string	필수 사항입니다. 데이터베이스 인스턴스 연결 문자열. Athena에서 사용되는 카탈로그의 이름을 환경 변수 앞에 붙입니다. 예를 들어, Athena에 등록된 카탈로그가 mysnowflakecatalog 인 경우 환경 변수 이름은 mysnowflakecatalog_connection_string 입니다.
default	필수 사항입니다. 기본 연결 문자열. 이 문자열은 카탈로그가 lambda:${ *AWS_LAMBDA_FUNCTION_NAME* }일 때 사용됩니다.

다음은 snowflake1(기본값)과 snowflake2라는 2개의 데이터베이스 인스턴스를 지원하는 Snowflake MUX Lambda 함수에 대한 예제 속성입니다.

속성	값
default	snowflake://jdbc:snowflake://snowflake1.host:port/?warehouse=warehousename&db=db1&schema=schema1&${Test/RDS/Snowflake1}
snowflake_catalog1_connection_string	snowflake://jdbc:snowflake://snowflake1.host:port/?warehouse=warehousename&db=db1&schema=schema1${Test/RDS/Snowflake1}

속성	값
snowflake_catalog2_connection_string	snowflake://jdbc:snowflake://snowflake2.host:port/?warehouse=warehousename&db=db1&schema=schema1&user=sample2&password=sample2

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

  > **Important**
  >
  > 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 `username` 및 `password` 값으로 바꿉니다.

  Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

  ```
  {"username": "${username}", "password": "${password}"}
  ```

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 `${Test/RDS/Snowflake1}`입니다.

```
snowflake://jdbc:snowflake://snowflake1.host:port/?
warehouse=warehousename&db=db1&schema=schema1${Test/RDS/Snowflake1}&...
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
snowflake://jdbc:snowflake://snowflake1.host:port/
warehouse=warehousename&db=db1&schema=schema1&user=sample2&password=sample2&...
```

현재 Snowflake는 `user` 및 `password` JDBC 속성을 인식합니다. 또한 `user` 또는 `password` 키 없이 *`username`*`/`*`password`* 형식의 사용자 이름과 암호를 허용합니다.

단일 연결 핸들러 사용

다음과 같은 단일 연결 메타데이터 및 레코드 핸들러를 사용하여 단일 Snowflake 인스턴스에 연결할 수 있습니다.

핸들러 유형	Class
복합 핸들러	`SnowflakeCompositeHandler`
메타데이터 핸들러	`SnowflakeMetadataHandler`
레코드 핸들러	`SnowflakeRecordHandler`

단일 연결 핸들러 파라미터

파라미터	설명
`default`	필수 사항입니다. 기본 연결 문자열.

단일 연결 핸들러는 하나의 데이터베이스 인스턴스를 지원하며 `default` 연결 문자열 파라미터를 제공해야 합니다. 다른 연결 문자열은 모두 무시됩니다.

다음은 Lambda 함수에서 지원하는 단일 Snowflake 인스턴스에 대한 예제 속성입니다.

속성	값
default	snowflake://jdbc:snowflake://snowflake1.host:port/?secret=Test/RDS/Snowflake1

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
spill_bucket	필수 사항입니다. 유출 버킷 이름.
spill_prefix	필수 사항입니다. 유출 버킷 키 접두사.
spill_put_request_headers	(선택 사항) 유출에 사용되는 Amazon S3 putObject 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: {"x-amz-server-side-encryption" : "AES256"}). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 JDBC와 Apache Arrow의 해당 데이터 형식이 나와 있습니다.

JDBC	화살표
불	Bit
Integer	Tiny
Short	Smallint
Integer	정수

JDBC	화살표
Long	Bigint
float	Float4
Double	Float8
날짜	DateDay
Timestamp	DateMilli
String	Varchar
바이트	Varbinary
BigDecimal	10진수
ARRAY	나열

데이터 형식 변환

JDBC에서 Arrow로의 변환 외에도 커넥터는 Snowflake 소스와 Athena 데이터 형식이 호환되도록 다른 특정 변환을 수행합니다. 이러한 변환은 쿼리가 성공적으로 실행되도록 하는 데 도움이 됩니다. 다음 표에 이러한 변환이 나와 있습니다.

소스 데이터 형식(Snowflake)	변환된 데이터 형식(Athena)
TIMESTAMP	TIMESTAMPMILLI
날짜	TIMESTAMPMILLI
INTEGER	INT
DECIMAL	BIGINT
TIMESTAMP_NTZ	TIMESTAMPMILLI

지원되지 않는 다른 모든 데이터 형식은 VARCHAR로 변환됩니다.

파티션 및 분할

파티션은 커넥터에 대한 분할을 생성하는 방법을 결정하는 데 사용됩니다. Athena는 커넥터가 분할을 생성하는 데 도움이 되도록 테이블에 대한 파티셔닝 체계를 나타내는 `varchar` 유형의 합성 열을 생성합니다. 커넥터는 실제 테이블 정의를 수정하지 않습니다.

이 합성 열과 파티션을 생성하려면 Athena에 프라이머리 키를 정의해야 합니다. 하지만 Snowflake는 프라이머리 키 제약 조건을 적용하지 않으므로 고유성을 직접 적용해야 합니다. 그렇지 않으면 Athena는 단일 분할을 기본값으로 설정하게 됩니다.

성능

최적의 성능을 위해 가능하면 쿼리에 필터를 사용합니다. 또한 파티션 배포가 균일한 방대한 데이터 세트를 검색하려면 기본 파티셔닝을 사용하는 것이 좋습니다. 열의 하위 집합을 선택하면 쿼리 런타임 속도를 높이고 스캔되는 데이터를 줄일 수 있습니다. Snowflake 커넥터는 동시성으로 인한 제한에 대한 복원력이 뛰어납니다.

Athena Snowflake 커넥터는 조건부 푸시다운을 수행하여 쿼리에서 스캔하는 데이터를 줄입니다. `LIMIT` 절, 간단한 조건자 및 복잡한 표현식을 커넥터로 푸시다운하여 스캔하는 데이터와 쿼리 실행 시간을 줄입니다.

LIMIT 절

`LIMIT N` 문은 쿼리로 스캔하는 데이터를 줄입니다. `LIMIT N` 푸시다운을 통해 커넥터는 Athena에 `N`개 행만 반환합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 `WHERE` 절에 사용되는 표현식입니다. Athena Snowflake 커넥터는 이러한 표현식을 결합하고 Snowflake로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena Snowflake 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND, OR, NOT
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL
- 산술: ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- 기타: LIKE_PATTERN, IN

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%')
LIMIT 10;
```

패스스루 쿼리

Snowflake 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

Snowflake에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 Snowflake의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 `customer` 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 Snowflake 커넥터용 pom.xml 파일을 참조하세요.

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Microsoft SQL Server 커넥터

Microsoft SQL Server용 Amazon Athena 커넥터를 사용하면 Amazon Athena가 JDBC를 사용하여 Microsoft SQL Server에 저장된 데이터에 대해 SQL 쿼리를 실행할 수 있습니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 멀티플렉서 설정에서 유출 버킷과 접두사는 모든 데이터베이스 인스턴스에서 공유됩니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.
- 필터 조건에서 `Date` 및 `Timestamp` 데이터 형식을 적절한 데이터 형식으로 캐스팅해야 합니다.
- `Real` 및 `Float` 형식의 음수 값을 검색하려면 <= 또는 >= 연산자를 사용합니다.
- `binary`, `varbinary`, `image` 및 `rowversion` 데이터 형식은 지원되지 않습니다.

용어

다음 용어는 SQL Server 커넥터와 관련이 있습니다.

- 데이터베이스 인스턴스 - 온프레미스, Amazon EC2 또는 Amazon RDS에 배포된 데이터베이스의 모든 인스턴스.
- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.

- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - connection_string 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.
- 멀티플렉싱 핸들러 - 여러 데이터베이스 연결을 수락하고 사용할 수 있는 Lambda 핸들러.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 SQL Server 커넥터를 구성합니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 데이터베이스 인스턴스에 연결합니다.

```
sqlserver://${jdbc_connection_string}
```

멀티플렉싱 핸들러 사용

멀티플렉서를 사용하여 단일 Lambda 함수로 여러 데이터베이스 인스턴스에 연결할 수 있습니다. 요청은 카탈로그 이름을 기준으로 라우팅됩니다. Lambda에서 다음 클래스를 사용합니다.

핸들러	Class
복합 핸들러	SqlServerMuxCompositeHandler
메타데이터 핸들러	SqlServerMuxMetadataHandler
레코드 핸들러	SqlServerMuxRecordHandler

멀티플렉싱 핸들러 파라미터

파라미터	설명
$*catalog*_connection_string	필수 사항입니다. 데이터베이스 인스턴스 연결 문자열. Athena에서 사용되는 카탈로그의 이름을 환경 변수 앞에 붙입니다. 예를 들어, Athena에 등록된 카탈로그가 mysqlservercatalog 인 경우 환경 변수 이름은 mysqlservercatalog_connection_string 입니다.

파라미터	설명
default	필수 사항입니다. 기본 연결 문자열. 이 문자열은 카탈로그가 lambda:${ *AWS_LAMBDA_FUNCTION_NAME* }일 때 사용됩니다.

다음은 sqlserver1(기본값)과 sqlserver2라는 2개의 데이터베이스 인스턴스를 지원하는 SqlServer MUX Lambda 함수에 대한 예제 속성입니다.

속성	값
default	sqlserver://jdbc:sqlserver://sqlserver1. *hostname*:*port*;databaseName= *<database_name>* ;${*secret1_name* }
sqlserver_catalog1_connection_string	sqlserver://jdbc:sqlserver://sqlserver1. *hostname*:*port*;databaseName= *<database_name>* ;${*secret1_name* }
sqlserver_catalog2_connection_string	sqlserver://jdbc:sqlserver://sqlserver2. *hostname*:*port*;databaseName= *<database_name>* ;${*secret2_name* }

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

> **Important**
>
> 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 `username` 및 `password` 값으로 바꿉니다.

  Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

  ```
  {"username": "${username}", "password": "${password}"}
  ```

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 `${secret_name}`입니다.

```
sqlserver://jdbc:sqlserver://hostname:port;databaseName=<database_name>;${secret_name}
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
sqlserver://
jdbc:sqlserver://
hostname:port;databaseName=<database_name>;user=<user>;password=<password>
```

단일 연결 핸들러 사용

다음과 같은 단일 연결 메타데이터 및 레코드 핸들러를 사용하여 단일 SQL Server 인스턴스에 연결할 수 있습니다.

핸들러 유형	Class
복합 핸들러	`SqlServerCompositeHandler`
메타데이터 핸들러	`SqlServerMetadataHandler`
레코드 핸들러	`SqlServerRecordHandler`

단일 연결 핸들러 파라미터

파라미터	설명
default	필수 사항입니다. 기본 연결 문자열.

단일 연결 핸들러는 하나의 데이터베이스 인스턴스를 지원하며 default 연결 문자열 파라미터를 제공해야 합니다. 다른 연결 문자열은 모두 무시됩니다.

다음은 Lambda 함수에서 지원하는 단일 SQL Server 인스턴스에 대한 예제 속성입니다.

속성	값
default	sqlserver://jdbc:sqlserver:// *hostname*:*port*;databaseName= *<database_name>* ;${*secret_name* }

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
spill_bucket	필수 사항입니다. 유출 버킷 이름.
spill_prefix	필수 사항입니다. 유출 버킷 키 접두사.
spill_put_request_headers	(선택 사항) 유출에 사용되는 Amazon S3 putObject 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: {"x-amz-server-side-encryption" : "AES256"}). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 SQL Server와 Apache Arrow의 해당 데이터 형식이 나와 있습니다.

SQL Server	화살표
bit	TINYINT
tinyint	SMALLINT
smallint	SMALLINT
int	INT
bigint	BIGINT
decimal	DECIMAL
numeric	FLOAT8
smallmoney	FLOAT8
money	DECIMAL
float[24]	FLOAT4
float[53]	FLOAT8
실제	FLOAT4
datetime	Date(MILLISECOND)
datetime2	Date(MILLISECOND)
smalldatetime	Date(MILLISECOND)
date	Date(DAY)
시간	VARCHAR
datetimeoffset	Date(MILLISECOND)
char[n]	VARCHAR
varchar[n/max]	VARCHAR

SQL Server	화살표
nchar[n]	VARCHAR
nvarchar[n/max]	VARCHAR
텍스트	VARCHAR
ntext	VARCHAR

파티션 및 분할

파티션은 `varchar` 형식의 단일 파티션 열로 표시됩니다. SQL Server 커넥터의 경우 파티션 함수에 따라 테이블에 파티션이 적용되는 방식이 결정됩니다. 파티션 함수 및 열 이름 정보는 SQL Server 메타데이터 테이블에서 검색됩니다. 그런 다음 사용자 지정 쿼리가 파티션을 가져옵니다. 분할은 수신된 개별 파티션 수를 기준으로 생성됩니다.

성능

열의 하위 집합을 선택하면 쿼리 런타임 속도를 높이고 스캔되는 데이터를 줄일 수 있습니다. SQL Server 커넥터는 동시성으로 인한 제한에 대한 복원력이 뛰어납니다.

쿼리에서 스캔하는 데이터를 줄이기 위해 Athena SQL Server 커넥터에서 조건자 푸시다운을 수행합니다. 스캔하는 데이터와 쿼리 실행 시간을 줄이도록 간단한 조건자와 복잡한 표현식을 커넥터로 푸시다운합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 `WHERE` 절에 사용되는 표현식입니다. Athena SQL Server 커넥터는 이러한 표현식을 결합하고 SQL Server로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena SQL Server 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND, OR, NOT
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, IS_DISTINCT_FROM, NULL_IF, IS_NULL
- 산술: ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- 기타: LIKE_PATTERN, IN

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%');
```

패스스루 쿼리

SQL Server 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

SQL Server에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 SQL Server의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 `customer` 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 SQL Server 커넥터용 pom.xml 파일을 참조하세요.

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Teradata 커넥터

Teradata용 Amazon Athena 커넥터를 사용하면 Athena가 Teradata 데이터베이스에 저장된 데이터에 대해 SQL 쿼리를 실행할 수 있습니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

제한 사항

- DDL 쓰기 작업은 지원되지 않습니다.
- 멀티플렉서 설정에서 유출 버킷과 접두사는 모든 데이터베이스 인스턴스에서 공유됩니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.

용어

다음 용어는 Teradata 커넥터와 관련이 있습니다.

- 데이터베이스 인스턴스 - 온프레미스, Amazon EC2 또는 Amazon RDS에 배포된 데이터베이스의 모든 인스턴스.
- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.
- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - `connection_string` 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.
- 멀티플렉싱 핸들러 - 여러 데이터베이스 연결을 수락하고 사용할 수 있는 Lambda 핸들러.

Lambda 계층 필수 구성 요소

Athena와 함께 Teradata 커넥터를 사용하려면 Teradata JDBC 드라이버를 포함하는 Lambda 계층을 생성해야 합니다. Lambda 계층은 Lambda 함수에 관한 추가 코드를 포함하는 `.zip` 파일 아카이브입니다. 계정에 Teradata 커넥터를 배포할 때 계층의 ARN을 지정합니다. 이렇게 하면 Teradata JDBC 드라이버가 있는 Lambda 계층이 Teradata 커넥터에 첨부되므로 Athena와 함께 사용할 수 있습니다.

Lambda 계층에 대한 자세한 내용은 AWS Lambda 개발자 안내서의 Lambda 계층 만들기 및 공유를 참조하세요.

Teradata 커넥터용 Lambda 계층을 생성하려면

1. Teradata JDBC 드라이버 다운로드 페이지(https://downloads.teradata.com/download/connectivity/jdbc-driver)로 이동합니다.
2. Teradata JDBC 드라이버를 다운로드합니다. 웹 사이트에서 파일을 다운로드하려면 계정을 생성하고 라이선스 계약에 동의해야 합니다.
3. 다운로드한 아카이브 파일에서 `terajdbc4.jar` 파일을 추출합니다.
4. 다음 폴더 구조를 만들고 `.jar` 파일을 안에 배치합니다.

   ```
   java\lib\terajdbc4.jar
   ```
5. `terajdbc4.jar` 파일이 포함된 전체 폴더 구조의 `.zip` 파일을 생성합니다.
6. AWS Management Console에 로그인하고 https://console.aws.amazon.com/lambda/에서 AWS Lambda 콘솔을 엽니다.
7. 탐색 창에서 계층(Layers)을 선택한 다음 계층 생성(Create layer)을 선택합니다.
8. 이름(Name)에 계층의 이름을 입력합니다(예: `TeradataJava11LambdaLayer`).
9. .zip 파일 업로드(Upload a .zip file) 옵션이 선택되어 있는지 확인하세요.
10. 업로드(Upload)를 선택하여 Teradata JDBC 드라이버가 포함된 압축 폴더를 업로드합니다.
11. 생성(Create)을 선택합니다.
12. 계층의 세부 정보 페이지에서 페이지 상단의 클립보드 아이콘을 선택하여 계층 ARN을 복사합니다.
13. 참조용으로 ARN을 저장합니다.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 Teradata 커넥터를 구성합니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 데이터베이스 인스턴스에 연결합니다.

```
teradata://${jdbc_connection_string}
```

멀티플렉싱 핸들러 사용

멀티플렉서를 사용하여 단일 Lambda 함수로 여러 데이터베이스 인스턴스에 연결할 수 있습니다. 요청은 카탈로그 이름을 기준으로 라우팅됩니다. Lambda에서 다음 클래스를 사용합니다.

핸들러	Class
복합 핸들러	`TeradataMuxCompositeHandler`
메타데이터 핸들러	`TeradataMuxMetadataHandler`
레코드 핸들러	`TeradataMuxRecordHandler`

멀티플렉싱 핸들러 파라미터

파라미터	설명
$*catalog*_connection_string	필수 사항입니다. 데이터베이스 인스턴스 연결 문자열. Athena에서 사용되는 카탈로그의 이름을 환경 변수 앞에 붙입니다. 예를 들어, Athena에 등록된 카탈로그가 `myteradatacatalog` 인 경우 환경 변수 이름은 `myteradatacatalog_connection_string` 입니다.
`default`	필수 사항입니다. 기본 연결 문자열. 이 문자열은 카탈로그가 `lambda:${` *AWS_LAMBDA_FUNCTION_NAME* `}`일 때 사용됩니다.

다음은 `teradata1`(기본값)과 `teradata2`라는 2개의 데이터베이스 인스턴스를 지원하는 Teradata MUX Lambda 함수에 대한 예제 속성입니다.

속성	값
default	teradata://jdbc:teradata://teradata2.host/TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,user=sample2&password=sample2
teradata_catalog1_connection_string	teradata://jdbc:teradata://teradata1.host/TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,${Test/RDS/Teradata1}
teradata_catalog2_connection_string	teradata://jdbc:teradata://teradata2.host/TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,user=sample2&password=sample2

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

  > **Important**
  >
  > 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 `username` 및 `password` 값으로 바꿉니다.

Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

```
{"username": "${username}", "password": "${password}"}
```

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 `${Test/RDS/Teradata1}`입니다.

```
teradata://jdbc:teradata1.host/TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,${Test/RDS/
Teradata1}&...
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
teradata://jdbc:teradata://teradata1.host/
TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,...&user=sample2&password=sample2&...
```

현재 Teradata는 `user` 및 `password` JDBC 속성을 인식합니다. 또한 `user` 또는 `password` 키 없이 *`username`*`/`*`password`* 형식의 사용자 이름과 암호를 허용합니다.

단일 연결 핸들러 사용

다음과 같은 단일 연결 메타데이터 및 레코드 핸들러를 사용하여 단일 Teradata 인스턴스에 연결할 수 있습니다.

핸들러 유형	Class
복합 핸들러	`TeradataCompositeHandler`
메타데이터 핸들러	`TeradataMetadataHandler`
레코드 핸들러	`TeradataRecordHandler`

단일 연결 핸들러 파라미터

파라미터	설명
default	필수 사항입니다. 기본 연결 문자열.

단일 연결 핸들러는 하나의 데이터베이스 인스턴스를 지원하며 default 연결 문자열 파라미터를 제공해야 합니다. 다른 연결 문자열은 모두 무시됩니다.

다음은 Lambda 함수에서 지원하는 단일 Teradata 인스턴스에 대한 예제 속성입니다.

속성	값
default	teradata://jdbc:teradata://teradata1.host/TMODE=ANSI,CHARSET=UTF8,DATABASE=TEST,secret=Test/RDS/Teradata1

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
spill_bucket	필수 사항입니다. 유출 버킷 이름.
spill_prefix	필수 사항입니다. 유출 버킷 키 접두사.
spill_put_request_headers	(선택 사항) 유출에 사용되는 Amazon S3 putObject 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: {"x-amz-server-side-encryption" : "AES256"}). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 JDBC와 Apache Arrow의 해당 데이터 형식이 나와 있습니다.

JDBC	화살표
불	Bit
Integer	Tiny
Short	Smallint
Integer	정수
Long	Bigint
float	Float4
Double	Float8
날짜	DateDay
Timestamp	DateMilli
String	Varchar
바이트	Varbinary
BigDecimal	10진수
ARRAY	나열

파티션 및 분할

파티션은 `Integer` 형식의 단일 파티션 열로 표시됩니다. 열에는 Teradata 테이블에 정의된 파티션의 이름이 들어 있습니다. 파티션 이름이 없는 테이블의 경우 단일 파티션에 해당하는 *가 반환됩니다. 파티션은 분할과 동일합니다.

명칭	유형	설명
파티션	Integer	Teradata의 명명된 파티션.

성능

Teradata는 기본 파티션을 지원합니다. Athena Teradata 커넥터는 이러한 파티션에서 병렬로 데이터를 검색할 수 있습니다. 파티션 배포가 균일한 초대규모 데이터 세트를 쿼리하려면 기본 파티셔닝을 사용하는 것이 좋습니다. 열 하위 집합을 선택하면 쿼리 런타임이 크게 느려집니다. 이 커넥터는 동시성으로 인해 약간의 제한을 나타냅니다.

쿼리에서 스캔하는 데이터를 줄이기 위해 Athena Teradata 커넥터에서 조건자 푸시다운을 수행합니다. 스캔하는 데이터와 쿼리 실행 시간을 줄이도록 간단한 조건자와 복잡한 표현식을 커넥터로 푸시다운합니다.

Predicates

조건자는 부울 값으로 평가되고 여러 조건에 따라 행을 필터링하는 SQL 쿼리의 `WHERE` 절에 사용되는 표현식입니다. Athena Teradata 커넥터는 이러한 표현식을 결합하고 Teradata로 직접 푸시하여 기능을 개선하고 스캔하는 데이터를 줄일 수 있습니다.

다음 Athena Teradata 커넥터 연산자는 조건자 푸시다운을 지원합니다.

- 부울: AND, OR, NOT
- 관계: EQUAL, NOT_EQUAL, LESS_THAN, LESS_THAN_OR_EQUAL, GREATER_THAN, GREATER_THAN_OR_EQUAL, NULL_IF, IS_NULL
- 산술: ADD, SUBTRACT, MULTIPLY, DIVIDE, MODULUS, NEGATE
- 기타: LIKE_PATTERN, IN

결합된 푸시다운 예제

쿼리 기능을 개선하기 위해 다음 예제와 같이 푸시다운 유형을 결합합니다.

```
SELECT *
FROM my_table
WHERE col_a > 10
    AND ((col_a + col_b) > (col_c % col_d))
```

```
    AND (col_e IN ('val1', 'val2', 'val3') OR col_f LIKE '%pattern%');
```

패스스루 쿼리

Teradata 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

Teradata에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 Teradata의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 `customer` 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 Teradata 커넥터용 pom.xml 파일을 참조하세요.

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Timestream 커넥터

Amazon Athena Timestream 커넥터를 사용하면 Amazon Athena가 Amazon Timestream과 통신할 수 있으므로 Amazon Athena를 통해 시계열 데이터에 액세스할 수 있습니다. 선택에 따라 AWS Glue Data Catalog를 보충 메타데이터의 소스로 사용할 수 있습니다.

Amazon Timestream은 빠르고 확장 가능한 완전 관리형 특수 목적 시계열 데이터베이스로, 매일 수조 개의 시계열 데이터 포인트를 쉽게 저장하고 분석할 수 있습니다. Timestream은 사용자 정의 정책에 따라 최근 데이터는 메모리에 보관하고 기록 데이터는 비용 최적화 스토리지 계층으로 이동하여 시계열 데이터의 수명주기를 관리하는 데 드는 시간과 비용을 절약합니다.

계정에서 Lake Formation을 활성화한 경우 AWS Serverless Application Repository에 배포한 Athena 페더레이션형 Lambda 커넥터의 IAM 역할은 Lake Formation에서 AWS Glue Data Catalog에 대한 읽기 액세스 권한을 가지고 있어야 합니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 Timestream 커넥터를 구성합니다.

- spill_bucket – Lambda 함수 제한을 초과하는 데이터에 대한 Amazon S3 버킷을 지정합니다.
- spill_prefix – (선택 사항) 기본값은 `athena-federation-spill`이라는 지정된 `spill_bucket`의 하위 폴더입니다. 미리 정해진 일 수 또는 시간보다 오래된 유출을 삭제하려면 이 위치에서 Amazon S3 스토리지 수명 주기를 구성하는 것이 좋습니다.
- spill_put_request_headers – (선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵입니다(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.
- kms_key_id – (선택 사항) 기본적으로 Amazon S3로 유출된 모든 데이터는 AES-GCM 인증 암호화 모드와 임의로 생성된 키를 사용하여 암호화됩니다. Lambda 함수가 `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`과 같이 KMS에서 생성된 더 강력한 암호화 키를 사용하도록 하려면 KMS 키 ID를 지정합니다.
- disable_spill_encryption – (선택 사항) `True`로 설정하면 유출 암호화가 비활성화됩니다. S3로 유출되는 데이터가 AES-GCM을 사용하여 암호화되도록 기본값은 `False`입니다(임의로 생성된 키 또는 KMS를 사용하여 키 생성). 유출 암호화를 비활성화하면 특히 유출 위치가 서버 측 암호화를 사용하는 경우 성능이 향상될 수 있습니다.
- glue_catalog – (선택 사항) 이 옵션을 사용하여 크로스 계정 AWS Glue 카탈로그를 지정합니다. 기본적으로 커넥터는 자체 AWS Glue 계정에서 메타데이터를 가져오려고 시도합니다.

AWS Glue에서 데이터베이스 및 테이블 설정

필요에 따라 AWS Glue Data Catalog를 보충 메타데이터의 소스로 사용할 수 있습니다. Timestream에서 사용할 AWS Glue 테이블을 활성화하려면 보충 메타데이터를 제공할 Timestream 데이터베이스 및 테이블과 이름이 일치하는 AWS Glue 데이터베이스 및 테이블이 있어야 합니다.

> **Note**
>
> 최상의 성능을 얻으려면 데이터베이스 이름과 테이블 이름에 소문자만 사용합니다. 대/소문자를 혼합하여 사용하면 커넥터에서 대소문자를 구분하지 않고 검색하므로 더욱 컴퓨팅 집약적입니다.

Timestream에서 사용할 AWS Glue 테이블을 구성하려면 AWS Glue에서 테이블 속성을 설정해야 합니다.

보충 메타데이터에 AWS Glue 테이블 사용

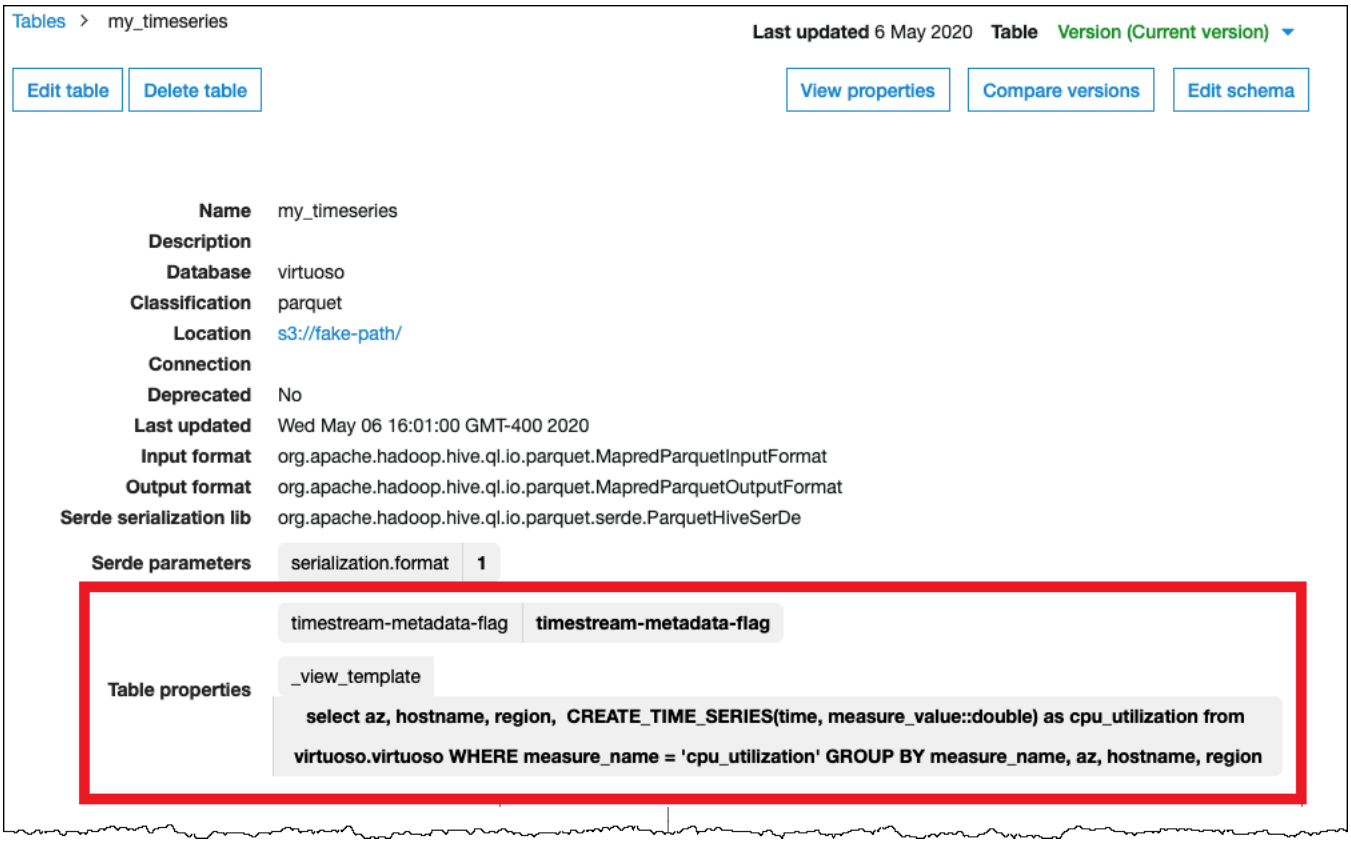
1. AWS Glue 콘솔에서 테이블을 편집하여 다음 테이블 속성을 추가합니다.
   - timestream-metadata-flag – 이 속성은 보충 메타데이터에 테이블을 사용할 수 있음을 Timestream 커넥터에 나타냅니다. `timestream-metadata-flag` 속성이 테이블 속성 목록에 있는 한 `timestream-metadata-flag`에 모든 값을 제공할 수 있습니다.
   - _view_template – 보충 메타데이터에 AWS Glue를 사용하는 경우 이 테이블 속성을 사용하고 모든 Timestream SQL을 보기로 지정할 수 있습니다. Athena Timestream 커넥터는 보기의 SQL을 Athena의 SQL과 함께 사용하여 쿼리를 실행합니다. 이는 Athena에서 사용할 수 없는 Timestream SQL의 기능을 사용하려는 경우에 유용합니다.
2. 이 문서에 나열된 대로 AWS Glue에 적합한 데이터 형식을 사용해야 합니다.

데이터 타입

현재 Timestream 커넥터는 Timestream에서 사용 가능한 데이터 형식의 하위 세트, 특히 스칼라 값 `varchar`, `double` 및 `timestamp`만 지원합니다.

`timeseries` 데이터 형식을 쿼리하려면 Timestream `CREATE_TIME_SERIES` 함수를 사용하는 AWS Glue 테이블 속성에서 보기를 구성해야 합니다. 또한 시계열 열의 유형으로 `ARRAY<STRUCT<time:timestamp,measure_value::double:double>>` 구문을 사용하는 보기에 대한 스키마를 제공해야 합니다. `double`을 테이블에 적합한 스칼라 유형으로 바꿉니다.

다음 이미지는 시계열에 대한 보기를 설정하도록 구성된 AWS Glue 테이블 속성의 예제를 보여줍니다.



필수 권한

이 커넥터에 필요한 IAM 정책에 대한 자세한 내용을 알아보려면 athena-timestream.yaml 파일의 `Policies` 섹션을 검토하세요. 다음 목록에 필요한 권한이 요약되어 있습니다.

- Amazon S3 쓰기 액세스 - 대규모 쿼리의 결과 유출을 위해서는 커넥터에 Amazon S3 위치에 대한 쓰기 액세스 권한이 필요합니다.
- Athena GetQueryExecution - 커넥터는 업스트림 Athena 쿼리가 종료된 경우 이 권한을 사용하여 빠른 실패를 수행합니다.
- AWS Glue Data Catalog - 스키마 정보를 가져오기 위해 Timestream 커넥터에 AWS Glue Data Catalog에 대한 읽기 전용 액세스 권한이 필요합니다.
- CloudWatch Logs - 로그를 저장하기 위해 커넥터 CloudWatch Logs에 대한 액세스 권한이 필요합니다.
- Timestream 액세스 - Timestream 쿼리를 실행하는 데 사용됩니다.

성능

대화형 쿼리가 제대로 작동하려면 반환되는 데이터(스캔되는 데이터 아님)를 256MB 미만으로 제한하는 LIMIT 절을 사용하는 것이 좋습니다.

Athena Timestream 커넥터는 조건자 푸시다운을 수행하여 쿼리에서 스캔하는 데이터를 줄입니다. LIMIT 절은 스캔하는 데이터를 줄이지만 조건자가 제공되지 않으면 LIMIT 절을 포함하는 SELECT 쿼리는 최소 16MB의 데이터를 스캔합니다. 열의 하위 집합을 선택하면 쿼리 런타임 속도를 높이고 스캔되는 데이터를 줄일 수 있습니다. Timestream 커넥터는 동시성으로 인한 제한에 대한 복원력이 뛰어납니다.

패스스루 쿼리

Timestream 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

Timestream에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 Timestream의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 customer 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

라이선스 정보

Amazon Athena Timestream 커넥터 프로젝트는 Apache-2.0 라이선스에 따라 사용이 허가됩니다.

추가적인 리소스

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena TPC Benchmark DS(TPC-DS) 커넥터

Amazon Athena TPC-DS 커넥터를 통해 Amazon Athena는 Athena Federation의 벤치마킹 및 기능 테스트에 사용하기 위해 무작위로 생성된 TPC 벤치마크 DS 데이터 소스와 통신할 수 있습니다. Athena

TPC-DS 커넥터는 네 가지 확장 요소 중 하나로 TPC-DS 호환 데이터베이스를 생성합니다. Amazon S3 기반 데이터 레이크 성능 테스트의 대안으로 이 커넥터를 사용하지 않는 것이 좋습니다.

필수 조건

- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.

파라미터

이 섹션의 Lambda 환경 변수를 사용하여 TPC-DS 커넥터를 구성합니다.

- spill_bucket – Lambda 함수 제한을 초과하는 데이터에 대한 Amazon S3 버킷을 지정합니다.
- spill_prefix – (선택 사항) 기본값은 `athena-federation-spill`이라는 지정된 `spill_bucket`의 하위 폴더입니다. 미리 정해진 일 수 또는 시간보다 오래된 유출을 삭제하려면 이 위치에서 Amazon S3 스토리지 수명 주기를 구성하는 것이 좋습니다.
- spill_put_request_headers – (선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵입니다(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.
- kms_key_id – (선택 사항) 기본적으로 Amazon S3로 유출된 모든 데이터는 AES-GCM 인증 암호화 모드와 임의로 생성된 키를 사용하여 암호화됩니다. Lambda 함수가 `a7e63k4b-8loc-40db-a2a1-4d0en2cd8331`과 같이 KMS에서 생성된 더 강력한 암호화 키를 사용하도록 하려면 KMS 키 ID를 지정합니다.
- disable_spill_encryption – (선택 사항) `True`로 설정하면 유출 암호화가 비활성화됩니다. S3로 유출되는 데이터가 AES-GCM을 사용하여 암호화되도록 기본값은 `False`입니다(임의로 생성된 키 또는 KMS를 사용하여 키 생성). 유출 암호화를 비활성화하면 특히 유출 위치가 서버 측 암호화를 사용하는 경우 성능이 향상될 수 있습니다.

데이터베이스 및 테이블 테스트

Athena TPC-DS 커넥터는 4개의 스케일 팩터 `tpcds1`, `tpcds10`, `tpcds100`, `tpcds250` 또는 `tpcds1000` 중 하나로 TPC-DS 호환 데이터베이스를 생성합니다.

테이블 요약

테스트 데이터 테이블 및 열의 전체 목록을 보려면 `SHOW TABLES` 또는 `DESCRIBE TABLE` 쿼리를 실행합니다. 편의를 위해 다음 표 요약이 제공됩니다.

1. call_center
2. catalog_page
3. catalog_returns
4. catalog_sales
5. customer
6. customer_address
7. customer_demographics
8. date_dim
9. dbgen_version
10. household_demographics
11. income_band
12. Inventory
13. item
14. promotion
15. reason
16. ship_mode
17. store
18. store_returns
19. store_sales
20. time_dim
21. warehouse
22. web_page
23. web_returns
24. web_sales
25. web_site

이렇게 생성된 스키마 및 데이터와 호환되는 TPC-DS 쿼리를 알아보려면 GitHub의 athena-tpcds/src/main/resources/queries/ 디렉터리를 참조하세요.

쿼리 예

다음 SELECT 쿼리 예제는 특정 군(country)의 고객 인구 통계에 대해 tpcds 카탈로그를 쿼리합니다.

```
SELECT
  cd_gender,
  cd_marital_status,
  cd_education_status,
  count(*) cnt1,
  cd_purchase_estimate,
  count(*) cnt2,
  cd_credit_rating,
  count(*) cnt3,
  cd_dep_count,
  count(*) cnt4,
  cd_dep_employed_count,
  count(*) cnt5,
  cd_dep_college_count,
  count(*) cnt6
FROM
  "lambda:tpcds".tpcds1.customer c, "lambda:tpcds".tpcds1.customer_address ca,
 "lambda:tpcds".tpcds1.customer_demographics
WHERE
  c.c_current_addr_sk = ca.ca_address_sk AND
    ca_county IN ('Rush County', 'Toole County', 'Jefferson County',
                  'Dona Ana County', 'La Porte County') AND
    cd_demo_sk = c.c_current_cdemo_sk AND
    exists(SELECT *
           FROM "lambda:tpcds".tpcds1.store_sales, "lambda:tpcds".tpcds1.date_dim
           WHERE c.c_customer_sk = ss_customer_sk AND
             ss_sold_date_sk = d_date_sk AND
             d_year = 2002 AND
             d_moy BETWEEN 1 AND 1 + 3) AND
    (exists(SELECT *
            FROM "lambda:tpcds".tpcds1.web_sales, "lambda:tpcds".tpcds1.date_dim
            WHERE c.c_customer_sk = ws_bill_customer_sk AND
              ws_sold_date_sk = d_date_sk AND
              d_year = 2002 AND
              d_moy BETWEEN 1 AND 1 + 3) OR
      exists(SELECT *
             FROM "lambda:tpcds".tpcds1.catalog_sales, "lambda:tpcds".tpcds1.date_dim
```

```
              WHERE c.c_customer_sk = cs_ship_customer_sk AND
                cs_sold_date_sk = d_date_sk AND
                d_year = 2002 AND
                d_moy BETWEEN 1 AND 1 + 3))
GROUP BY cd_gender,
  cd_marital_status,
  cd_education_status,
  cd_purchase_estimate,
  cd_credit_rating,
  cd_dep_count,
  cd_dep_employed_count,
  cd_dep_college_count
ORDER BY cd_gender,
  cd_marital_status,
  cd_education_status,
  cd_purchase_estimate,
  cd_credit_rating,
  cd_dep_count,
  cd_dep_employed_count,
  cd_dep_college_count
LIMIT 100
```

필수 권한

이 커넥터에 필요한 IAM 정책에 대한 자세한 내용을 알아보려면 athena-tpcds.yaml 파일의 `Policies` 섹션을 검토하세요. 다음 목록에 필요한 권한이 요약되어 있습니다.

- Amazon S3 쓰기 액세스 - 대규모 쿼리의 결과 유출을 위해서는 커넥터에 Amazon S3 위치에 대한 쓰기 액세스 권한이 필요합니다.
- Athena GetQueryExecution - 커넥터는 업스트림 Athena 쿼리가 종료된 경우 이 권한을 사용하여 빠른 실패를 수행합니다.

성능

Athena TPC-DS 커넥터는 선택한 스케일 팩터를 기준으로 쿼리 병렬화를 시도합니다. 조건자 푸시다운은 Lambda 함수 내에서 수행됩니다.

라이선스 정보

Amazon Athena TPC-DS 커넥터 프로젝트는 Apache-2.0 라이선스에 따라 사용이 허가됩니다.

추가적인 리소스

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트를 참조하세요.

Amazon Athena Vertica 커넥터

Vertica는 클라우드 또는 온프레미스에 배포할 수 있는 열 형식 데이터베이스 플랫폼으로서 엑사바이트 규모의 데이터 웨어하우스를 지원합니다. 연합 쿼리에 Amazon Athena Vertica 커넥터를 사용하여 Athena에서 Vertica 데이터 원본을 쿼리할 수 있습니다. 예를 들어 Vertica의 데이터 웨어하우스와 Amazon S3의 데이터 레이크에 대해 분석 쿼리를 실행할 수 있습니다.

필수 조건

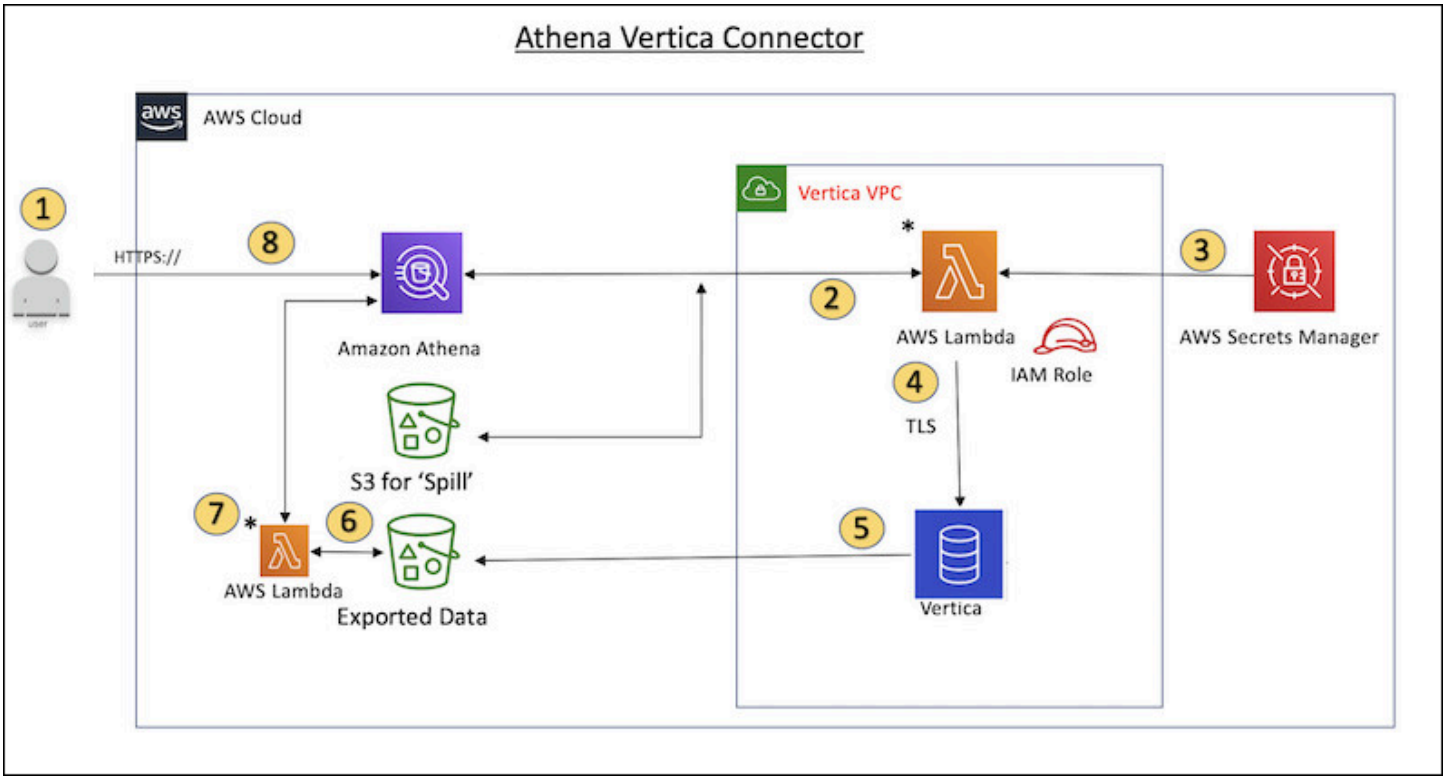
- Athena 콘솔 또는 AWS Serverless Application Repository를 사용하여 AWS 계정에 커넥터를 배포합니다. 자세한 내용은 데이터 소스 커넥터 배포 또는 AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포을 참조하세요.
- 이 커넥터를 사용하기 전에 VPC와 보안 그룹을 설정합니다. 자세한 내용은 데이터 원본 커넥터용 VPC 생성 단원을 참조하십시오.

제한 사항

- Athena Vertica 커넥터는 Amazon S3 Select를 사용하여 Amazon S3에서 Parquet 파일을 읽기 때문에 커넥터의 성능이 느려질 수 있습니다. 큰 테이블을 쿼리할 때는 CREATE TABLE AS (SELECT ...) 쿼리와 SQL 조건자를 사용하는 것이 좋습니다.
- 현재 Athena 연합 쿼리의 알려진 문제 때문에 커넥터로 인해 Vertica가 쿼리된 테이블의 모든 열을 Amazon S3로 내보내지만 쿼리된 열만 Athena 콘솔의 결과에 표시됩니다.
- DDL 쓰기 작업은 지원되지 않습니다.
- 모든 관련 Lambda 제한. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.

워크플로

다음 다이어그램은 Vertica 커넥터를 사용하는 쿼리의 워크플로를 보여줍니다.



1. Vertica에 있는 하나 이상의 테이블에 대해 SQL 쿼리가 실행됩니다.
2. 커넥터는 SQL 쿼리를 구문 분석하여 JDBC 연결을 통해 관련 부분을 Vertica로 보냅니다.
3. 연결 문자열은 AWS Secrets Manager에 저장된 사용자 이름과 암호를 사용하여 Vertica에 액세스합니다.
4. 커넥터는 다음 예와 같이 Vertica EXPORT 명령으로 SQL 쿼리를 래핑합니다.

```
EXPORT TO PARQUET (directory = 's3://DOC-EXAMPLE-BUCKET/folder_name',
   Compression='Snappy', fileSizeMB=64) OVER() as
SELECT
PATH_ID,
...
SOURCE_ITEMIZED,
SOURCE_OVERRIDE
FROM DELETED_OBJECT_SCHEMA.FORM_USAGE_DATA
WHERE PATH_ID <= 5;
```

5. Vertica는 SQL 쿼리를 처리하고 결과 세트를 Amazon S3 버킷으로 보냅니다. 처리량 향상을 위해 Vertica는 EXPORT 옵션을 사용하여 여러 Parquet 파일의 쓰기 작업을 병렬화합니다.
6. Athena는 Amazon S3 버킷을 스캔하여 결과 세트에 대해 읽을 파일 수를 결정합니다.

7. Athena는 Lambda 함수를 여러 번 호출하고 Amazon S3 Select를 사용하여 결과 세트에서 Parquet 파일을 읽습니다. 여러 번의 호출을 통해 Athena는 Amazon S3 파일 읽기를 병렬화하고 초당 최대 100GB의 처리량을 달성할 수 있습니다.
8. Athena는 데이터 레이크에서 스캔한 데이터로 Vertica에서 반환된 데이터를 처리하고 결과를 반환합니다.

용어

다음 용어는 Vertica 커넥터와 관련이 있습니다.

- 데이터베이스 인스턴스 - Amazon EC2에 배포된 Vertica 데이터베이스의 모든 인스턴스.
- 핸들러 - 데이터베이스 인스턴스에 액세스하는 Lambda 핸들러. 핸들러는 메타데이터 또는 데이터 레코드용일 수 있습니다.
- 메타데이터 핸들러 - 데이터베이스 인스턴스에서 메타데이터를 검색하는 Lambda 핸들러.
- 레코드 핸들러 - 데이터베이스 인스턴스에서 데이터 레코드를 검색하는 Lambda 핸들러.
- 복합 핸들러 - 데이터베이스 인스턴스에서 메타데이터와 데이터 레코드를 모두 검색하는 Lambda 핸들러.
- 속성 또는 파라미터 - 핸들러에서 데이터베이스 정보를 추출하는 데 사용되는 데이터베이스 속성. 이러한 속성을 Lambda 환경 변수로 구성합니다.
- 연결 문자열 - 데이터베이스 인스턴스에 대한 연결을 설정하는 데 사용되는 텍스트 문자열.
- 카탈로그 - `connection_string` 속성의 필수 접두사로서 Athena에 등록된 비 AWS Glue Glue 카탈로그.

파라미터

Amazon Athena Vertica 커넥터는 Lambda 환경 변수를 통해 구성 옵션을 제공합니다. 다음 Lambda 환경 변수를 사용하여 커넥터를 구성할 수 있습니다.

- AthenaCatalogName – Lambda 함수 이름입니다.
- ExportBucket – Vertica 쿼리 결과를 내보내는 Amazon S3 버킷입니다.
- SpillBucket – 이 함수가 데이터를 유출할 수 있는 Amazon S3 버킷의 이름입니다.
- SpillPrefix – 이 함수가 데이터를 유출할 수 있는 `SpillBucket` 위치의 접두사입니다.
- SecurityGroupIds – Lambda 함수에 적용해야 하는 보안 그룹에 해당하는 하나 이상의 ID입니다(예: `sg1`, `sg2` 또는 `sg3`).

- SubnetIds – Lambda 함수가 데이터 소스에 액세스하는 데 사용할 수 있는 서브넷에 해당하는 하나 이상의 서브넷 ID입니다(예: `subnet1` 또는 `subnet2`).
- SecretNameOrPrefix – 이 함수가 액세스할 수 있는 Secrets Manager 이름 세트의 이름 또는 접두사입니다(예: `vertica-*`).
- VerticaConnectionString – 카탈로그 관련 연결이 정의되지 않은 경우 기본적으로 사용할 Vertica 연결 세부 정보입니다. 문자열은 선택적으로 AWS Secrets Manager 구문(예: `${secret_name}`)을 사용할 수 있습니다.
- VPC ID – Lambda 함수에 연결할 VPC ID입니다.

연결 문자열

다음 형식의 JDBC 연결 문자열을 사용하여 데이터베이스 인스턴스에 연결합니다.

```
vertica://jdbc:vertica://host_name:port/database?user=vertica-
username&password=vertica-password
```

단일 연결 핸들러 사용

다음과 같은 단일 연결 메타데이터 및 레코드 핸들러를 사용하여 단일 Vertica 인스턴스에 연결할 수 있습니다.

핸들러 유형	Class
복합 핸들러	`VerticaCompositeHandler`
메타데이터 핸들러	`VerticaMetadataHandler`
레코드 핸들러	`VerticaRecordHandler`

단일 연결 핸들러 파라미터

파라미터	설명
`default`	필수 사항입니다. 기본 연결 문자열.

단일 연결 핸들러는 하나의 데이터베이스 인스턴스를 지원하며 `default` 연결 문자열 파라미터를 제공해야 합니다. 다른 연결 문자열은 모두 무시됩니다.

자격 증명 제공

JDBC 연결 문자열에서 데이터베이스의 사용자 이름과 암호를 제공하려면 연결 문자열 속성 또는 AWS Secrets Manager를 사용합니다.

- 연결 문자열 - 사용자 이름과 암호를 JDBC 연결 문자열에 속성으로 지정할 수 있습니다.

  > **Important**
  >
  > 보안 모범 사례로, 환경 변수 또는 연결 문자열에서 하드 코딩된 보안 인증은 사용하지 않습니다. 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동하는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 하드 코딩된 보안 암호를 AWS Secrets Manager로 이동을 참조하세요.

- AWS Secrets Manager - AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager 연결을 위한 VPC 엔드포인트 또는 인터넷 액세스가 Lambda 함수에 연결된 VPC에 있어야 합니다.

  JDBC 연결 문자열에 AWS Secrets Manager의 보안 암호 이름을 입력할 수 있습니다. 커넥터는 암호 이름을 Secrets Manager의 `username` 및 `password` 값으로 바꿉니다.

  Amazon RDS 데이터베이스 인스턴스의 경우 이 지원은 긴밀하게 통합됩니다. Amazon RDS를 사용하는 경우 AWS Secrets Manager 및 자격 증명 교체를 사용하는 것이 좋습니다. 데이터베이스에서 Amazon RDS를 사용하지 않는 경우 자격 증명을 다음 형식의 JSON으로 저장합니다.

  ```
  {"username": "${username}", "password": "${password}"}
  ```

보안 암호 이름이 있는 연결 문자열의 예제

다음 문자열의 보안 암호 이름은 `${vertica-username}` 및 `${vertica-password}`입니다.

```
vertica://jdbc:vertica://host_name:port/database?user=${vertica-username}&password=
${vertica-password}
```

커넥터는 다음 예제와 같이 보안 암호 이름을 사용하여 보안 암호를 검색하고 사용자 이름과 암호를 제공합니다.

```
vertica://jdbc:vertica://host_name:port/database?user=sample-user&password=sample-password
```

현재 Vertica 커넥터는 `vertica-username` 및 `vertica-password` JDBC 속성을 인식합니다.

유출 파라미터

Lambda SDK는 데이터를 Amazon S3로 유출할 수 있습니다. 동일한 Lambda 함수에서 액세스하는 모든 데이터베이스 인스턴스는 동일한 위치로 유출됩니다.

파라미터	설명
`spill_bucket`	필수 사항입니다. 유출 버킷 이름.
`spill_prefix`	필수 사항입니다. 유출 버킷 키 접두사.
`spill_put_request_headers`	(선택 사항) 유출에 사용되는 Amazon S3 `putObject` 요청에 대한 요청 헤더 및 값의 JSON 인코딩 맵(예: `{"x-amz-server-side-encryption" : "AES256"}`). 다른 가능한 헤더를 알아보려면 Amazon Simple Storage Service API Reference(Amazon Simple Storage Service API 참조)의 PutObject를 참조하세요.

데이터 형식 지원

다음 표에 Vertica 커넥터의 지원되는 데이터 형식이 나와 있습니다.

불
BigInt
Short
Integer
Long
Float

불
Double
날짜
Varchar
바이트
BigDecimal
TimeStamp as Varchar

성능

Lambda 함수는 프로젝션 푸시다운을 수행하여 쿼리에서 스캔되는 데이터를 줄입니다. `LIMIT` 절은 스캔되는 데이터의 양을 줄이지만 조건자가 제공되지 않으면 `LIMIT` 절을 포함하는 `SELECT` 쿼리가 최소 16MB의 데이터를 스캔할 것으로 예상해야 합니다. Vertica 커넥터는 동시성으로 인한 제한에 대한 복원력이 뛰어납니다.

패스스루 쿼리

Vertica 커넥터는 패스스루 쿼리를 지원합니다. 패스스루 쿼리는 테이블 함수를 사용하여 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운합니다.

Vertica에서 패스스루 쿼리를 사용하려면 다음 구문을 사용합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'query string'
        ))
```

다음 예제 쿼리는 Vertica의 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 `customer` 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10'
        ))
```

라이선스 정보

이 커넥터를 사용하면 이 커넥터에 대한 pom.xml 파일에서 목록을 찾을 수 있는 타사 구성 요소가 포함되어 있음을 인정하고 GitHub.com의 LICENSE.txt 파일에 제공된 해당 타사 라이선스의 조건에 동의하는 것으로 간주됩니다.

추가적인 리소스

최신 JDBC 드라이버 버전 정보를 알아보려면 GitHub.com의 Vertica 커넥터용 pom.xml 파일을 참조하세요.

이 커넥터에 대한 추가 정보를 알아보려면 GitHub.com의 해당 사이트 및 AWS Big Data Blog(빅 데이터 블로그)의 Querying a Vertica data source in Amazon Athena using the Athena Federated Query SDK(Athena 연합 쿼리 SDK를 사용하여 Amazon Athena에서 Vertica 데이터 소스 쿼리)를 참조하세요.

데이터 소스 커넥터 배포

연합 쿼리를 만들기 위한 준비는 Lambda 함수 데이터 원본 커넥터를 배포하고 Lambda 함수를 데이터 원본에 연결하는 두 부분으로 이루어진 프로세스입니다. 이 프로세스에서는 나중에 Athena 콘솔에서 선택할 수 있는 이름을 Lambda 함수에 지정하고 SQL 쿼리에서 참조할 수 있는 이름을 커넥터에 지정합니다.

> **Note**
>
> AWS Secrets Manager에서 Athena 연합 쿼리 기능을 사용하려면 Secrets Manager에 대해 Amazon VPC 프라이빗 엔드포인트를 구성해야 합니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 Secrets Manager VPC 프라이빗 엔드포인트 생성을 참조하세요.

주제

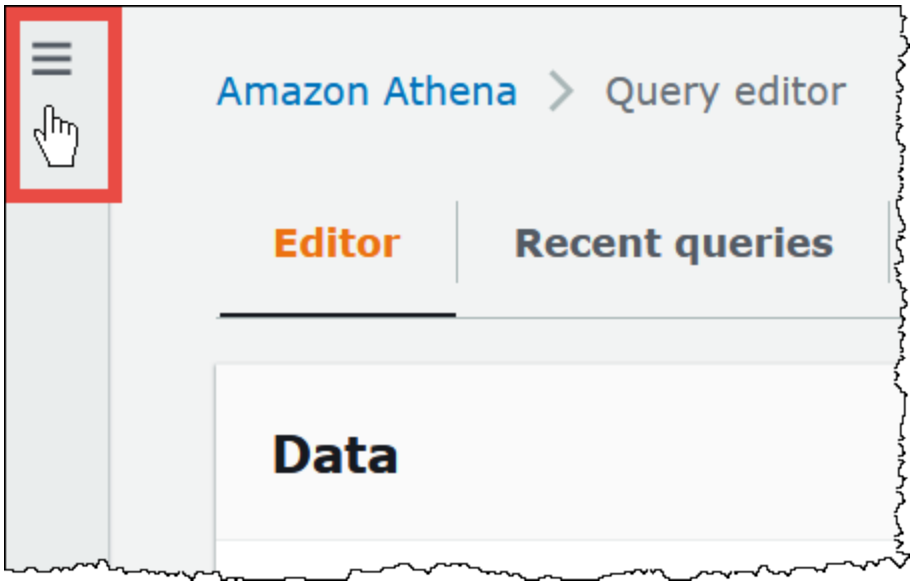
- Athena 콘솔 사용
- AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포
- 데이터 원본 커넥터용 VPC 생성
- 계정 간 연합 쿼리 사용 설정
- 데이터 소스 커넥터 업데이트

Athena 콘솔 사용

데이터 원본 커넥터를 선택하고 이름을 지정하고 배포하려면 통합 프로세스에서 Athena 및 Lambda 콘솔을 사용합니다.

데이터 원본 커넥터를 배포하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.



3. 탐색 창에서 데이터 원본(Data sources)을 선택합니다.
4. 데이터 원본(Data sources) 페이지에서 데이터 원본 생성(Create data source)을 선택합니다.
5. 데이터 원본 선택(Choose data source)에서 다음 지침을 고려하여 <shared id="ATE"/>에서 쿼리할 데이터 원본을 선택합니다.

   - 데이터 원본에 해당하는 연합 쿼리 옵션을 선택합니다. Athena에는 MySQL, Amazon DocumentDB 및 PostgreSQL을 포함한 소스에 대해 구성할 수 있는 사전 구축된 데이터 원본 커넥터가 있습니다.
   - Amazon S3에서 데이터를 쿼리하려고 하고 이 페이지에서 Apache Hive 메타스토어 또는 다른 연동 쿼리 데이터 원본 옵션 중 하나를 사용하고 있지 않은 경우 S3 - AWS Glue Data Catalog를 선택합니다. Athena는 AWS Glue Data Catalog를 사용하여 Amazon S3의 데이터 원본에 대한 메타데이터 및 스키마 정보를 저장합니다. 이것은 기본(비연합) 옵션입니다. 자세한 내용은 AWS Glue를 사용하여 Amazon S3의 데이터 원본에 연결 단원을 참조하십시오.

- S3 - Apache Hive 메타스토어(S3 - Apache Hive metastore)를 선택하여 Apache Hive 메타스토어를 사용하는 Amazon S3의 데이터 집합을 쿼리합니다. 이 옵션에 대한 자세한 내용은 Apache Hive 메타스토어에 Athena 연결을 참조하세요.
- <shared id="ATE"/>에서 사용할 고유한 데이터 원본 커넥터를 생성하려면 사용자 정의 또는 고유 데이터(Custom or shared data)을 선택합니다. 데이터 원본 커넥터 작성에 대한 정보에 대한 자세한 내용은 Athena Query Federation SDK를 사용하여 데이터 소스 커넥터 개발 섹션을 참조하세요.

이 자습서에서는 Amazon CloudWatch Logs를 연합 데이터 원본으로 선택합니다.

6. 다음을 선택합니다.
7. 데이터 원본 세부 정보 입력(Enter data source details) 섹션의 데이터 원본 이름(Data source name)에 <shared id="ATE"/>에서 데이터 원본을 쿼리할 때 SQL 문에 사용할 이름을 입력합니다(예: `CloudWatchLogs`). 이름은 127자까지 입력할 수 있으며 계정 내에서 고유해야 합니다. 생성 후에는 변경할 수 없습니다. 유효한 문자는 a-z, A-Z, 0-9, _(밑줄), @(앰퍼샌드) 및 -(하이픈)입니다. `awsdatacatalog`, `hive`, `jmx`, `system` 이름은 Athena에 예약되어 있으므로 데이터 원본 이름에 사용할 수 없습니다.
8. Lambda 함수에 대해 Lambda 함수 생성을 선택합니다. 선택한 커넥터의 함수 페이지가 AWS Lambda 콘솔에서 열립니다. 이 페이지에는 커넥터에 대한 자세한 정보가 포함되어 있습니다.
9. 애플리케이션 설정(Application settings)에서 각 애플리케이션 설정에 관한 설명을 신중히 읽은 다음 필요에 따라 값을 입력합니다.

   표시되는 애플리케이션 설정은 데이터 원본의 커넥터에 따라 다릅니다. 최소한의 필수 설정은 다음과 같습니다.

   - AthenaCatalogName – `cloudwatchlogs`와 같이 대상으로 하는 데이터 원본을 나타내는 소문자로 된 Lambda 함수의 이름입니다.
   - SpillBucket – Lambda 함수 응답 크기 제한을 초과하는 데이터를 저장하기 위한 계정의 Amazon S3 버킷입니다.

   > **Note**
   >
   > 유출된 데이터는 후속 실행에서 재사용되지 않으며 12시간 후에 안전하게 삭제할 수 있습니다. Athena는 이 데이터를 삭제하지 않습니다. Amazon S3 유출 버킷에서 이전 데이터를 삭제하는 객체 수명 주기 정책을 추가하여 이러한 객체를 관리하는 것이 좋습니다. 자세한 내용은 Amazon S3 사용 설명서의 스토리지 수명 주기 관리를 참조하세요.

10. 이 앱이 사용자 지정 IAM 역할 및 리소스 정책을 생성하는 것을 확인(I acknowledge that this app creates custom IAM roles and resource policies)을 선택합니다. 자세한 내용을 보려면 정보 링크를 선택하세요.
11. [배포]를 선택합니다. 배포가 완료되면 Lambda 콘솔의 리소스 섹션에 Lambda 함수가 나타납니다.

데이터 소스에 연결

계정에 데이터 원본 커넥터를 배포한 후 Athena를 연결할 수 있습니다.

계정에 배포한 커넥터를 사용하여 데이터 원본에 Athena 연결

1. <shared id="ATE"/> 콘솔의 데이터 원본 입력(Enter data sources) 페이지로 돌아갑니다.
2. 연결 세부 정보(Connection details)섹션에서 Lambda 함수 선택 또는 입력 검색 상자 옆에 있는 새로 고침 아이콘을 선택합니다.
3. Lambda 콘솔에서 방금 생성한 함수의 이름을 선택합니다. Lambda 함수의 ARN이 표시됩니다.
4. (선택 사항) 태그(Tags)에 대해 이 데이터 원본과 연결할 키-값 페어를 추가합니다. 태그에 대한 자세한 내용은 Athena 리소스 태깅 단원을 참조하세요.
5. 다음을 선택합니다.
6. 검토 및 생성 페이지에서 데이터 원본 세부 정보를 검토한 다음 데이터 원본 생성을 선택합니다.
7. 데이터 원본 세부 정보(Data source details) 페이지 섹션에는 새 커넥터에 대한 정보가 표시됩니다. 이제 <shared id="ATE"/> 쿼리에 커넥터를 사용할 수 있습니다.

   쿼리에서 데이터 커넥터 사용에 대한 자세한 내용은 페더레이션된 쿼리 실행 섹션을 참조하세요.

AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터 배포

데이터 원본 커넥터를 배포하기 위해 Athena 콘솔로 시작하는 대신 AWS Serverless Application Repository를 사용할 수 있습니다. AWS Serverless Application Repository로 사용할 커넥터를 찾고 커넥터에 필요한 파라미터를 제공한 다음 계정에 커넥터를 배포합니다. 그런 다음 커넥터를 배포한 후 Athena 콘솔을 사용하여 Athena에서 데이터 원본을 사용할 수 있도록 합니다.

계정에 커넥터 배포

AWS Serverless Application Repository을 사용하여 데이터 원본 커넥터를 계정에 배포하려면

1. AWS Management Console에 로그인하고 서버리스 앱 리포지토리를 엽니다.

2. 탐색 창에서 사용 가능한 애플리케이션을 선택합니다.
3. 사용자 지정 IAM 역할 또는 리소스 정책을 만드는 앱 표시(Show apps that create custom IAM roles or resource policies) 옵션을 선택합니다.
4. 검색 상자에 커넥터 이름을 입력합니다. 사전 구축된 Athena 데이터 커넥터의 목록은 사용 가능한 데이터 소스 커넥터 단원을 참조하세요.
5. 커넥터의 이름을 선택합니다. 커넥터를 선택하면 AWS Lambda 콘솔에서 Lambda 함수의 애플리케이션 세부 정보(Application details) 페이지가 열립니다.
6. 세부 정보 페이지 오른쪽의 애플리케이션 설정(Application settings)에서 필수 정보를 입력합니다. 최소 필수 설정은 다음과 같습니다. Athena에서 구축한 데이터 커넥터의 나머지 구성 가능한 옵션에 대한 자세한 내용은 GitHub의 사용 가능한 커넥터 주제를 참조하세요.

   - AthenaCatalogName – `cloudwatchlogs`와 같이 대상으로 하는 데이터 원본을 나타내는 소문자로 된 Lambda 함수의 이름입니다.
   - SpillBucket – Lambda 함수 응답 크기 제한을 초과하는 대규모 응답 페이로드에서 데이터를 수신하도록 계정에 Amazon S3 버킷을 지정합니다.
7. 이 앱이 사용자 지정 IAM 역할 및 리소스 정책을 생성하는 것을 확인(I acknowledge that this app creates custom IAM roles and resource policies)을 선택합니다. 자세한 내용을 보려면 정보 링크를 선택하세요.
8. 애플리케이션 설정(Application settings) 섹션의 오른쪽 하단에서 배포(Deploy)를 선택합니다. 배포가 완료되면 Lambda 콘솔의 리소스 섹션에 Lambda 함수가 나타납니다.

Athena에서 커넥터 사용 설정

<shared id="ATE"/> 콘솔을 사용하여 <shared id="ATE"/>에서 데이터 원본을 사용할 수 있도록 합니다.

<shared id="ATE"/> 콘솔을 사용하여 <shared id="ATE"/>에서 데이터 원본을 사용할 수 있도록 합니다.

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

3. 탐색 창에서 데이터 원본(Data sources)을 선택합니다.
4. 데이터 원본(Data sources) 페이지에서 데이터 원본 생성(Create data source)을 선택합니다.
5. 데이터 원본 선택에서는 AWS Serverless Application Repository에서 커넥터를 생성한 데이터 원본을 선택합니다. 이 자습서에서는 Amazon CloudWatch Logs를 연합 데이터 원본으로 선택합니다.
6. 다음을 선택합니다.
7. 데이터 원본 세부 정보 입력(Enter data source details) 섹션의 데이터 원본 이름(Data source name)에 <shared id="ATE"/>에서 데이터 원본을 쿼리할 때 SQL 문에 사용할 이름을 입력합니다(예: `CloudWatchLogs`). 이름은 127자까지 입력할 수 있으며 계정 내에서 고유해야 합니다. 생성 후에는 변경할 수 없습니다. 유효한 문자는 a-z, A-Z, 0-9, _(밑줄), @(앰퍼샌드) 및 -(하이픈)입니다. `awsdatacatalog`, `hive`, `jmx`, `system` 이름은 Athena에 예약되어 있으므로 데이터 원본 이름에 사용할 수 없습니다.
8. 연결 세부 정보 섹션에서, Lambda 함수 선택 또는 입력 상자를 사용해 방금 만든 함수의 이름을 선택합니다. Lambda 함수의 ARN이 표시됩니다.
9. (선택 사항) 태그(Tags)에 대해 이 데이터 원본과 연결할 키-값 페어를 추가합니다. 태그에 대한 자세한 내용은 Athena 리소스 태깅 단원을 참조하세요.
10. 다음을 선택합니다.
11. 검토 및 생성 페이지에서 데이터 원본 세부 정보를 검토한 다음 데이터 원본 생성을 선택합니다.
12. 데이터 원본 세부 정보(Data source details) 페이지 섹션에는 새 커넥터에 대한 정보가 표시됩니다. 이제 <shared id="ATE"/> 쿼리에 커넥터를 사용할 수 있습니다.

    쿼리에서 데이터 커넥터 사용에 대한 자세한 내용은 페더레이션된 쿼리 실행 섹션을 참조하세요.

데이터 원본 커넥터용 VPC 생성

일부 Athena 데이터 원본 커넥터는 VPC 및 보안 그룹이 필요합니다. 이 주제에서는 서브넷과 보안 그룹이 있는 VPC를 생성하는 방법을 보여줍니다. 이 과정의 일부로서 사용자가 생성하는 VPC, 서브넷, 보안 그룹에 대한 ID를 검색합니다. 이러한 ID는 Athena와 함께 사용하도록 커넥터를 구성할 때 필요합니다.

Athena 데이터 원본 커넥터에 대한 VPC를 생성하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.
2. 탐색 창에서 새로운 VPC 환경(New VPC Experience)이 선택되었는지 확인합니다.
3. VPC 생성을 선택합니다.
4. VPC 설정(VPC Settings)의 생성할 리소스(Resources to create)에서 VPC 등(VPC and more)을 선택합니다.
5. 자동 생성(Auto-generate)에서 VPC의 모든 리소스에 대한 이름 태그를 생성하는 데 사용할 값을 입력합니다.
6. VPC 생성을 선택합니다.
7. VPC 보기(View VPC)를 선택합니다.
8. 세부 정보(Details) 섹션의 VPC ID에서 나중에 참조할 수 있도록 VPC ID를 복사합니다.

이제 방금 생성한 VPC의 서브넷 ID를 검색할 준비가 되었습니다.

VPC 서브넷 ID를 검색하려면

1. VPC 콘솔 탐색 창에서 서브넷(Subnets)을 선택합니다.
2. 생성한 서브넷에 해당하는 이름을 선택합니다.
3. 세부 정보(Details) 섹션의 서브넷 ID(Subnet ID)에서 나중에 참조할 수 있도록 서브넷 ID를 복사합니다.

다음으로 VPC의 보안 그룹을 생성합니다.

VPC의 보안 그룹을 생성하려면

1. VPC 콘솔 탐색 창에서 보안(Security), 보안 그룹(Security Groups)을 선택합니다.
2. 보안 그룹 생성을 선택합니다.

3. 보안 그룹 생성(Create security group) 페이지에서 다음 정보를 입력합니다.
   - 보안 그룹 이름(Security group name)에 보안 그룹의 이름을 입력합니다.
   - 설명(Description)에 보안 그룹의 설명을 입력합니다. 이 필드는 필수 항목입니다.
   - VPC에 데이터 원본 커넥터로 생성한 VPC의 VPC ID를 입력합니다.
   - 인바운드 규칙(Inbound rules) 및 아웃바운드 규칙(Outbound rules)에 필요한 인바운드 및 아웃바운드 규칙을 추가합니다.
4. 보안 그룹 생성을 선택합니다.
5. 보안 그룹의 세부 정보(Details) 페이지에서 나중에 참조할 수 있도록 보안 그룹 ID(Security group ID)를 복사합니다.

계정 간 연합 쿼리 사용 설정

연합 쿼리를 사용하면 AWS Lambda에 배포된 데이터 원본 커넥터를 사용하여 Amazon S3 이외의 데이터 원본을 쿼리할 수 있습니다. 교차 계정 연합 쿼리 기능을 사용하면 Lambda 함수와 쿼리할 데이터 원본이 다른 계정에 위치하도록 할 수 있습니다.

데이터 관리자는 데이터 커넥터를 데이터 분석가의 계정과 공유하거나 데이터 분석가로서 데이터 관리자의 공유 Lambda ARN을 사용하여 계정에 추가하여 교차 계정 연동 쿼리를 활성화할 수 있습니다. 원래 계정의 커넥터에 대한 구성이 변경되면 업데이트된 구성이 다른 사용자 계정에 있는 커넥터의 공유 인스턴스에 자동으로 적용됩니다.

고려 사항 및 제한

- 교차 계정 연동 쿼리 기능은 Lambda 기반 데이터 원본을 사용하는 비 Hive 메타스토어 데이터 커넥터에 사용할 수 있습니다.
- AWS Glue Data Catalog 데이터 원본 유형에 대해 해당 기능을 사용할 수 없습니다. AWS Glue Data Catalog에 대한 교차 계정 액세스에 관한 자세한 내용은 AWS Glue 데이터 카탈로그에 대한 교차 계정 액세스 섹션을 참조하세요.
- 커넥터의 Lambda 함수에서 얻은 응답이 Lambda 응답 크기 제한(6MB)을 초과하는 경우 Athena는 자동으로 응답을 암호화하고 일괄 처리한 후 사용자가 구성한 Amazon S3 버킷으로 유출합니다. Athena 쿼리를 실행하는 엔터티가 유출 위치에 대한 액세스 권한을 보유해야 Athena에서 유출한 데이터를 읽을 수 있습니다. 쿼리를 완료한 후에는 데이터가 필요하지 않으므로 유출 위치에서 객체를 삭제하도록 Amazon S3 수명 주기 정책을 설정하는 것이 좋습니다.
- AWS 리전에서 페더레이션된 쿼리 사용은 지원되지 않습니다.

필요한 권한

- 데이터 관리자 계정 A가 데이터 분석가 계정 B와 Lambda 함수를 공유하려면 계정 B에서 Lambda가 함수 및 유출 버킷 액세스를 호출해야 합니다. 따라서 계정 A는 리소스 기반 정책을 Lambda 함수에 추가하고 보안 주체 액세스 권한을 Amazon S3 유출 버킷에 추가해야 합니다.

  1. 다음 정책은 Lambda가 계정 A의 Lambda 함수에서 계정 B에 대한 함수 호출 권한을 부여합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CrossAccountInvocationStatement",
      "Effect": "Allow",
      "Principal": {
        "AWS": ["arn:aws:iam::account-B-id:user/username"]
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:aws-region:account-A-id:function:lambda-function-
name"
    }
  ]
}
```

  2. 다음 정책은 계정 B의 보안 주체에 대한 유출 버킷 액세스를 허용합니다.

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": ["arn:aws:iam::account-B-id:user/username"]
            },
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
             ],
            "Resource": [
                "arn:aws:s3:::spill-bucket",
                "arn:aws:s3:::spill-bucket/*"
            ]
        }
```

```
    ]
}
```

3. Lambda 함수가 퍼더레이션 SDK가 제공하는 기본 암호화 대신에 AWS KMS 키를 통해 유출 버킷을 암호화하려면 다음 예와 같이 계정 A의 AWS KMS 키 정책은 계정 B의 사용자에게 액세스 권한을 부여해야 합니다.

```
{
    "Sid": "Allow use of the key",
    "Effect": "Allow",
    "Principal":
    {
       "AWS": ["arn:aws:iam::account-B-id:user/username"]
    },
    "Action": [ "kms:Decrypt" ],
    "Resource": "*" // Resource policy that gets placed on the KMS key.
}
```

- 계정 A가 해당 커넥터를 계정 B와 공유하려면 계정 B는 계정 A가 AWS Security Token Service AssumeRole API 작업을 호출함으로써 권한을 맡는 `AthenaCrossAccountCreate-account-A-id`라는 역할을 생성해야 합니다.

  `CreateDataCatalog` 작업을 허용하는 다음 정책은 계정 B에서 생성되어야 하며 계정 B가 계정 A에 대해 생성하는 `AthenaCrossAccountCreate-account-A-id` 역할에 추가되어야 합니다.

```
{
 "Effect": "Allow",
 "Action": "athena:CreateDataCatalog",
 "Resource": "arn:aws:athena:*:account-B-id:datacatalog/*"
}
```

계정 A의 데이터 원본을 계정 B와 공유

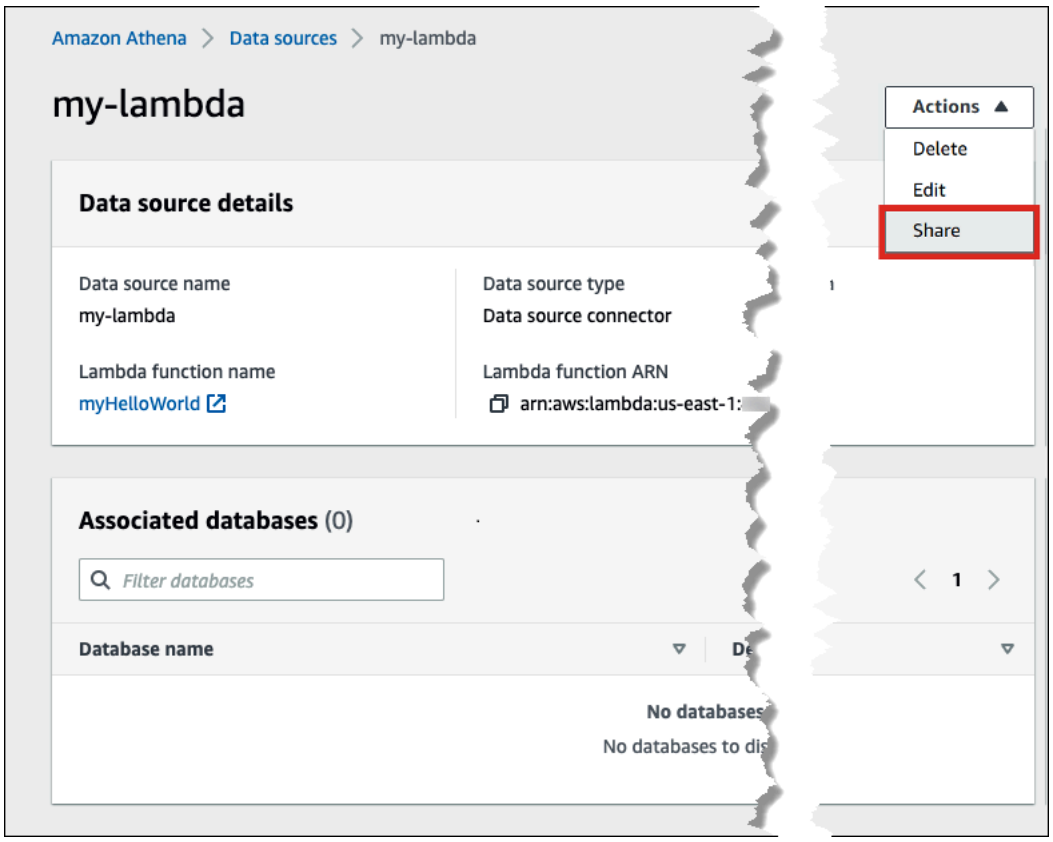
사용 권한이 설정된 후에는 Athena 콘솔의 데이터 원본(Data sources) 페이지를 사용하여 사용자 계정(계정 A)의 데이터 커넥터를 다른 계정(계정 B)과 연결할 수 있습니다. 계정 A는 커넥터의 모든 제어 및 소유권을 유지합니다. 계정 A가 커넥터에 대한 구성을 변경하면 업데이트된 구성이 계정 B의 공유 커넥터에 적용됩니다.

계정 B와 계정 A의 Lambda 데이터 원본 공유

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.

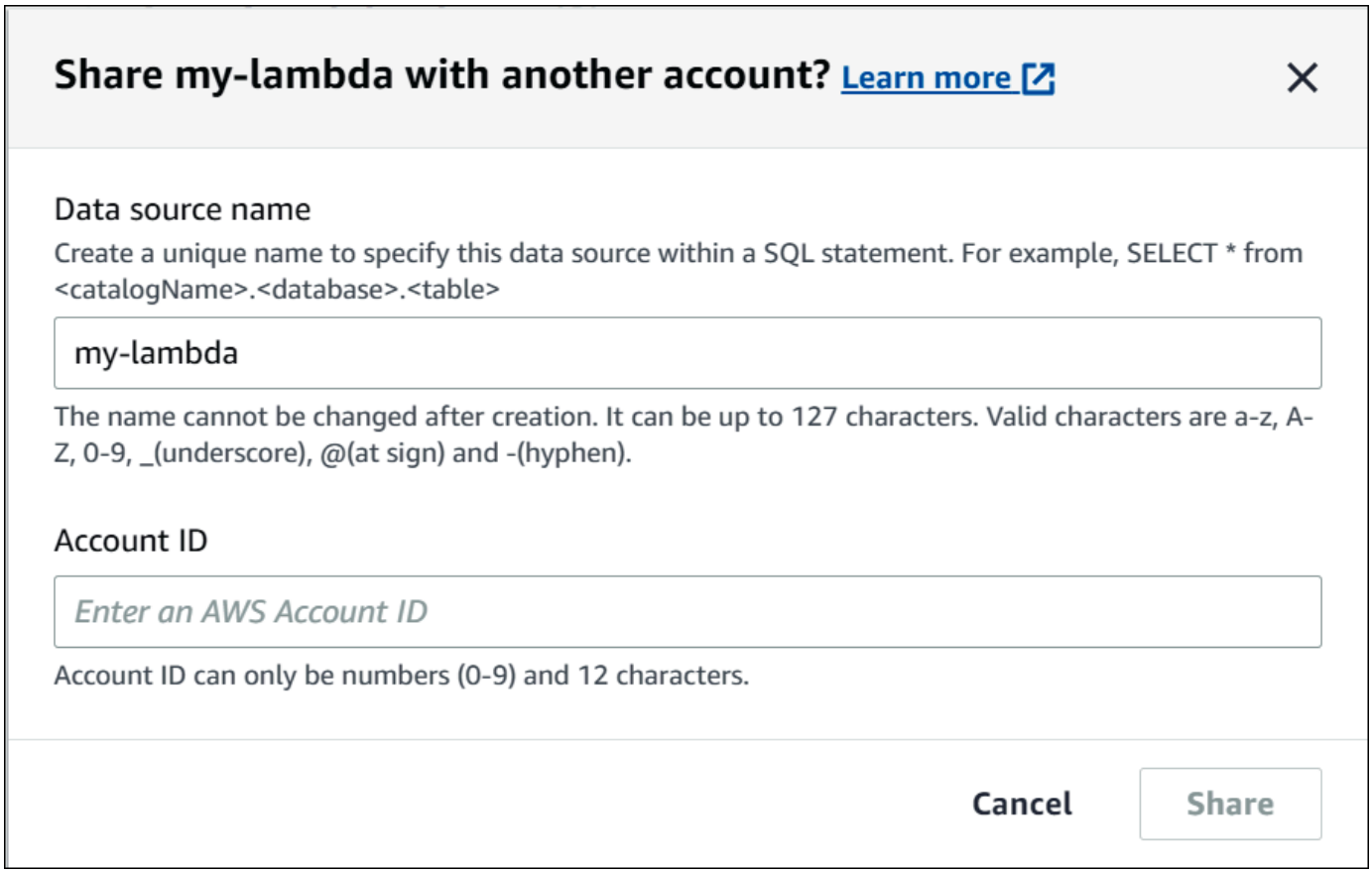
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

3. 데이터 소스(Data Source)을 선택합니다.
4. 데이터 원본(Data sources) 페이지에서 공유할 커넥터의 링크를 선택합니다.
5. Lambda 데이터 원본에 대한 세부 정보 페이지에서 오른쪽 상단 모서리에 있는 공유(Share) 옵션을 선택합니다.



6. 다른 계정과 ***Lambda-name*** 공유(Share Lambda-name with another account) 대화 상자에 필수 정보를 입력합니다.

   - 데이터 원본 이름(Data source name)에서 복사된 데이터 원본의 이름을 다른 계정에 표시하려는 대로 입력합니다.
   - 계정 ID(Account ID)에서 데이터 원본을 공유할 계정의 ID(이 경우 계정 B)를 입력합니다.



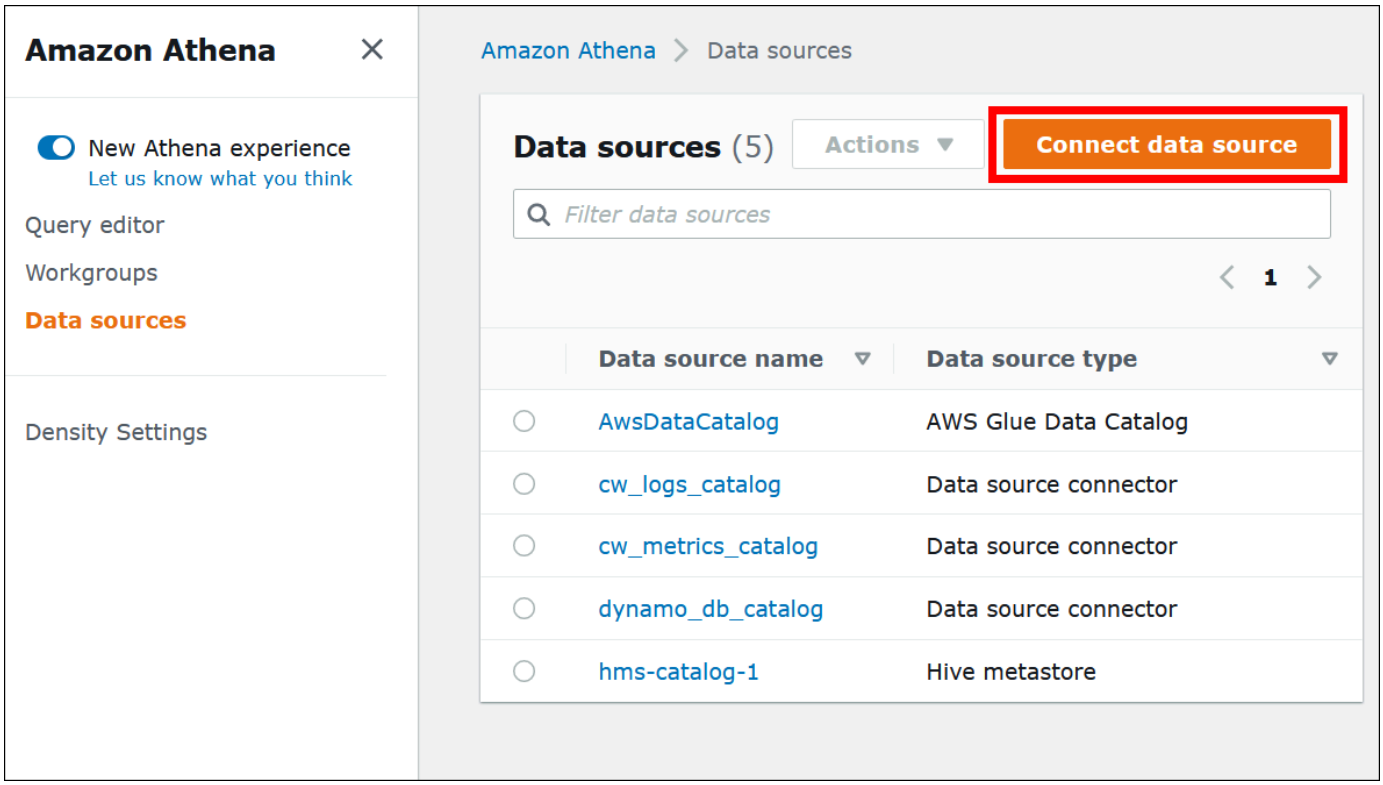
7. 공유를 선택합니다. 지정한 공유 데이터 커넥터가 계정 B에 생성됩니다. 계정 A의 커넥터에 대한 구성 변경 사항은 계정 B의 커넥터에 적용됩니다.

계정 A에서 계정 B로 공유 데이터 원본 추가

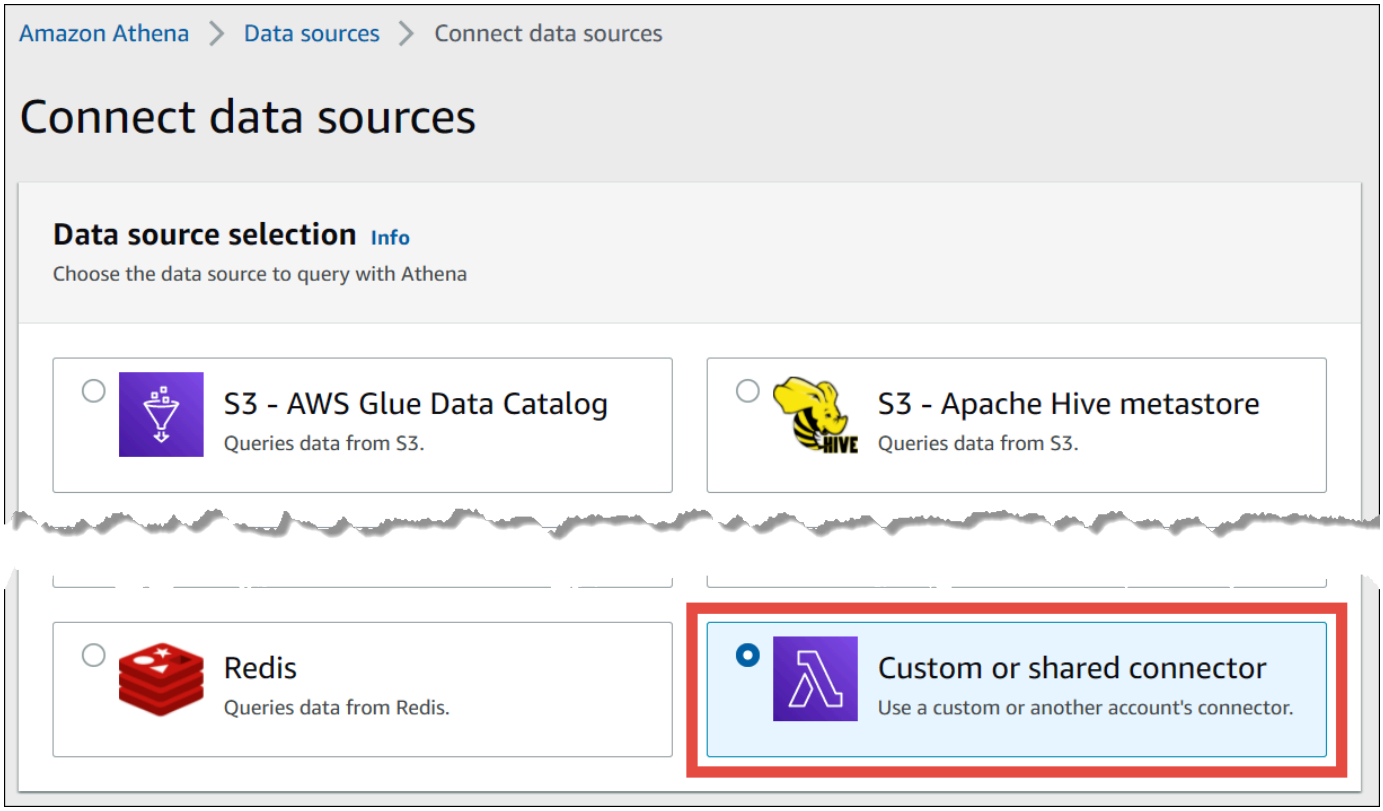
데이터 분석가는 데이터 관리자로부터 계정에 추가할 커넥터의 ARN을 제공받을 수 있습니다. 관리자가 제공한 Lambda ARN을 계정에 추가하려면 Athena 콘솔의 데이터 원본(Data sources) 페이지를 사용합니다.

계정에 공유 데이터 커넥터의 Lambda ARN 추가

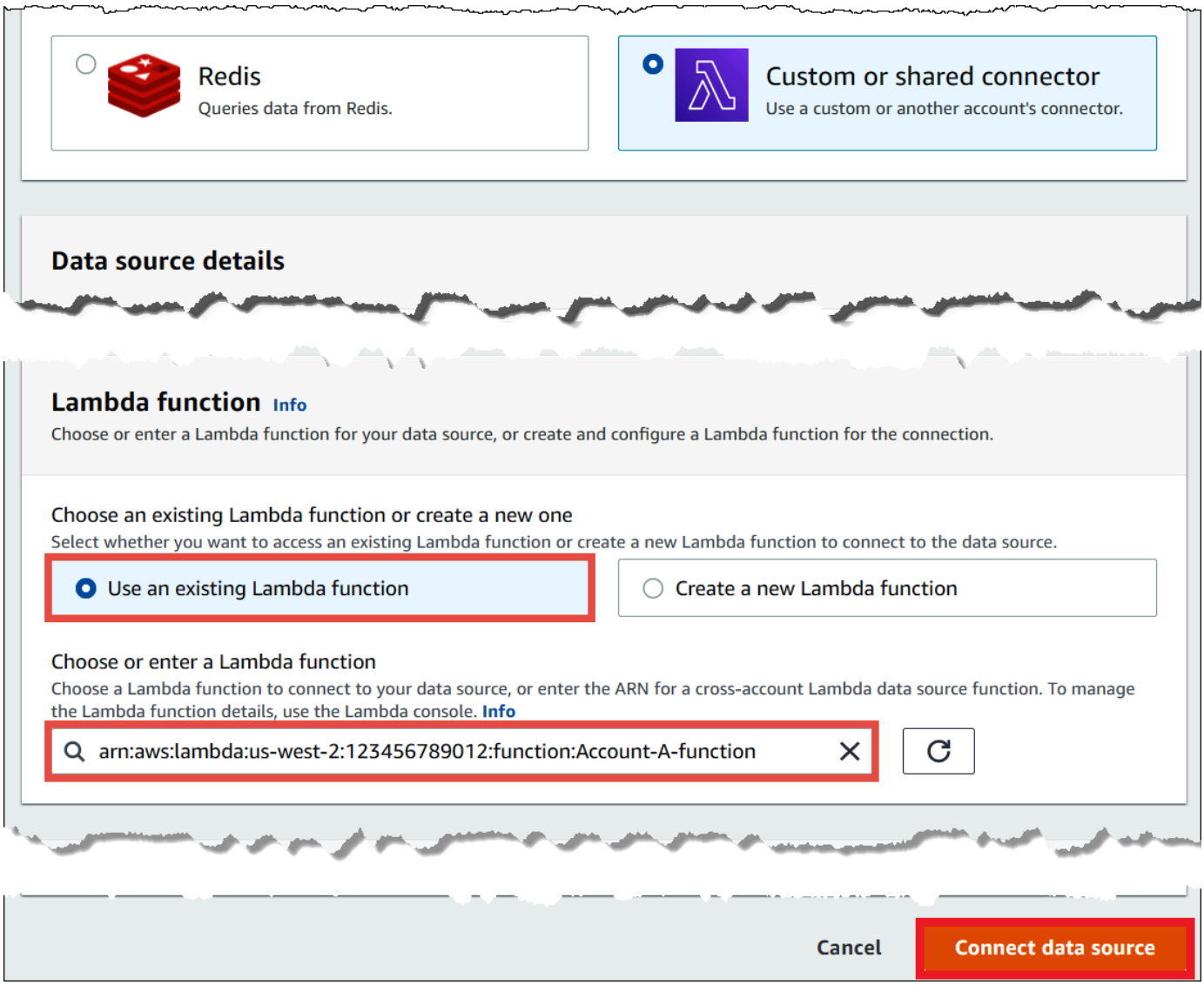
1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 새 콘솔 환경을 사용하고 있는데 탐색 창이 표시되지 않는 경우 왼쪽의 확장 메뉴를 선택합니다.
3. 데이터 소스(Data Source)을 선택합니다.
4. 데이터 원본(Data sources) 페이지에서 데이터 원본 연결(Connect data source)을 선택합니다.



5. 사용자 정의 또는 공유 커넥터(Custom or shared connector)를 선택합니다.



6. Lambda 함수(Lambda function) 섹션에서 기존 Lamba 함수 사용(Use an existing Lambda function) 옵션을 선택했는지 확인합니다.



7. Lambda 함수 선택 또는 입력(Choose or enter a Lambda function)에서 계정 A의 Lambda ARN을 입력합니다.
8. 데이터 원본 연결을 선택합니다.

문제 해결

계정 A에 계정 B에서 역할을 부여할 권한이 없다는 오류 메시지가 나타나면 계정 B에서 생성한 역할 이름의 철자가 올바른지, 올바른 정책이 연결되어 있는지 확인합니다.

데이터 소스 커넥터 업데이트

Athena는 사용하는 데이터 소스 커넥터를 최신 버전으로 주기적으로 업데이트하여 새로운 기능과 향상된 기능을 활용하도록 권장합니다. 먼저 최신 버전 번호를 찾아야 합니다.

최신 Athena Query Federation 버전 찾기

Athena 데이터 소스 커넥터의 최신 버전 번호는 최신 Athena Query Federation 버전에 해당됩니다. 경우에 따라 GitHub 릴리스가 AWS Serverless Application Repository(SAR)에서 제공되는 것보다 조금 더 최신일 수 있습니다.

최신 Athena Query Federation 버전 번호를 찾으려면

1. GitHub URL https://github.com/awslabs/aws-athena-query-federation/releases/latest를 방문하세요.
2. 기본 페이지 제목의 릴리스 번호를 다음 형식으로 기록합니다.

   Athena Query Federation의 릴리스 v*year*.*week_of_year*.*iteration_of_week*

   예를 들어 Athena Query Federation의 릴리스 v2023.8.3의 릴리스 번호는 2023.8.3입니다.

리소스 이름 찾기 및 기록

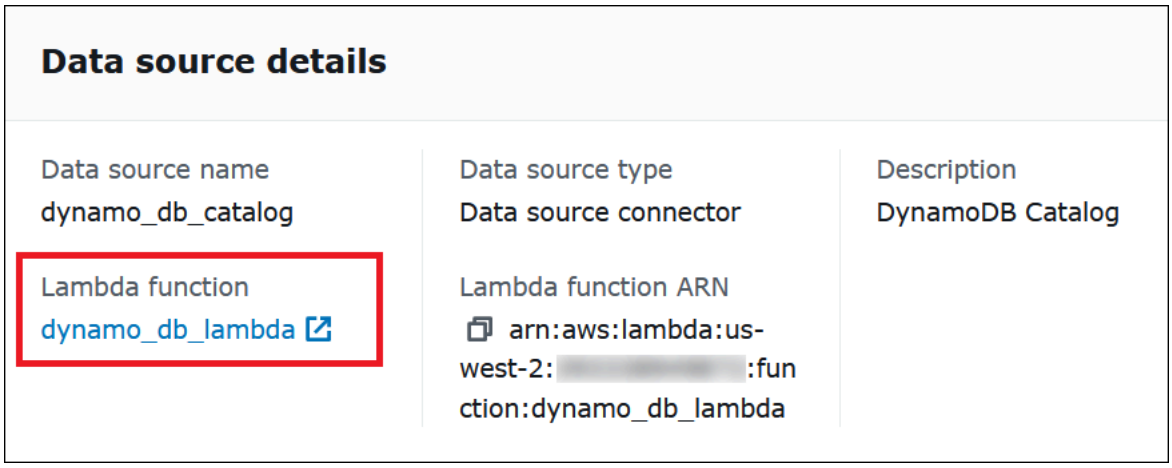
업그레이드를 준비하려면 다음 정보를 찾아 기록해야 합니다.

1. 커넥터의 Lambda 함수 이름.
2. Lambda 함수 환경 변수.
3. 커넥터의 Lambda 함수를 관리하는 Lambda 애플리케이션 이름.

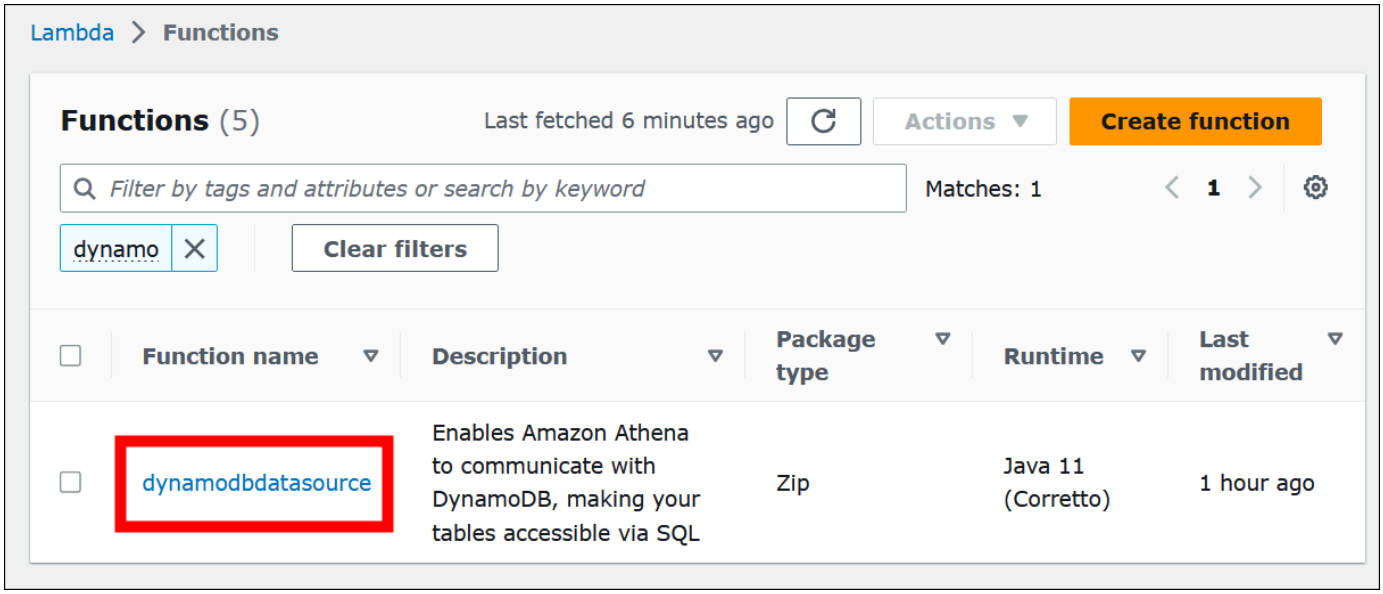
Athena 콘솔에서 리소스 이름을 찾으려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

3. 탐색 창에서 데이터 원본(Data sources)을 선택합니다.
4. 데이터 소스 이름 열에서 커넥터의 데이터 소스에 대한 링크를 선택합니다.
5. 데이터 소스 세부 정보 섹션의 Lambda 함수에서 Lambda 함수에 대한 링크를 선택합니다.



6. 함수 페이지의 함수 이름 열에서 커넥터의 함수 이름을 기록합니다.



7. 함수 이름 링크를 선택합니다.
8. 함수 개요 섹션에서 구성 탭을 선택합니다.
9. 왼쪽의 창에서 환경 변수를 선택합니다.
10. 환경 변수 섹션에서 키와 해당 값을 기록합니다.
11. 페이지 맨 위로 스크롤합니다.
12. 이 함수는 애플리케이션에 속해 있습니다. 관리하려면 여기를 클릭합니다. 메시지에서 여기를 클릭 링크를 선택합니다.
13. serverlessrepo-***your_application_name*** 페이지에서 serverlessrepo를 제외한 애플리케이션 이름을 기록합니다. 예를 들어 애플리케이션 이름이 serverlessrepo-DynamoDbTestApp인 경우 사용자 애플리케이션 이름은 DynamoDbTestApp입니다.
14. 애플리케이션의 Lambda 콘솔 페이지에서 사용 중인 커넥터 버전 찾기 단계를 계속 진행합니다.

사용 중인 커넥터 버전 찾기

다음 단계에 따라 사용 중인 커넥터 버전을 찾습니다.

사용 중인 커넥터 버전을 찾으려면

1. Lambda 애플리케이션의 Lambda 콘솔 페이지에서 배포 탭을 선택합니다.
2. 배포 탭에서 SAM 템플릿을 확장합니다.
3. CodeUri를 검색합니다.
4. 키 필드의 CodeUri에서 다음 문자열을 찾습니다.

```
applications-connector_name-
versions-year.week_of_year.iteration_of_week/hash_number
```

다음 예제는 CloudWatch 커넥터의 문자열을 보여줍니다.

```
applications-AthenaCloudwatchConnector-versions-2021.42.1/15151159...
```

5. *year.week_of_year.iteration_of_week*에 대한 값(예: 2021.42.1)을 기록합니다. 이 버전은 커넥터의 버전입니다.

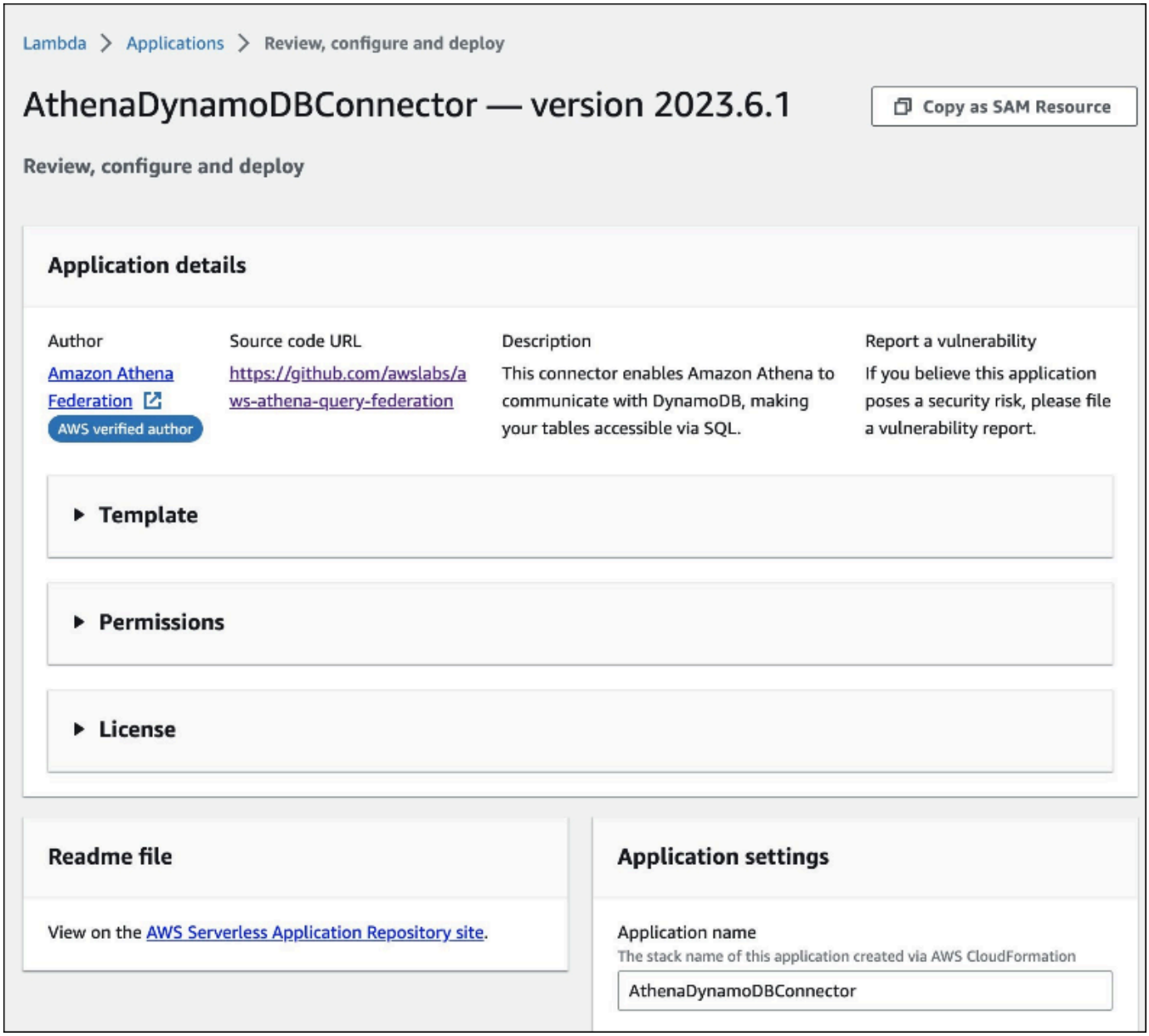
커넥터의 새 버전 배포

다음 단계에 따라 커넥터의 새 버전을 배포합니다.

커넥터의 새 버전을 배포하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

3. 탐색 창에서 데이터 원본(Data sources)을 선택합니다.
4. 데이터 원본(Data sources) 페이지에서 데이터 원본 생성(Create data source)을 선택합니다.
5. 업그레이드할 데이터 소스를 선택하고 다음을 선택합니다.
6. 연결 세부 정보 섹션에서 Lambda 함수 생성을 선택합니다. 그러면 업데이트된 애플리케이션을 배포할 수 있는 Lambda 콘솔이 열립니다.



7. 실제로 새 데이터 소스를 생성하지는 않으므로 Athena 콘솔 탭을 닫아도 됩니다.
8. 커넥터의 Lambda 콘솔 페이지에서 다음 단계를 수행합니다.

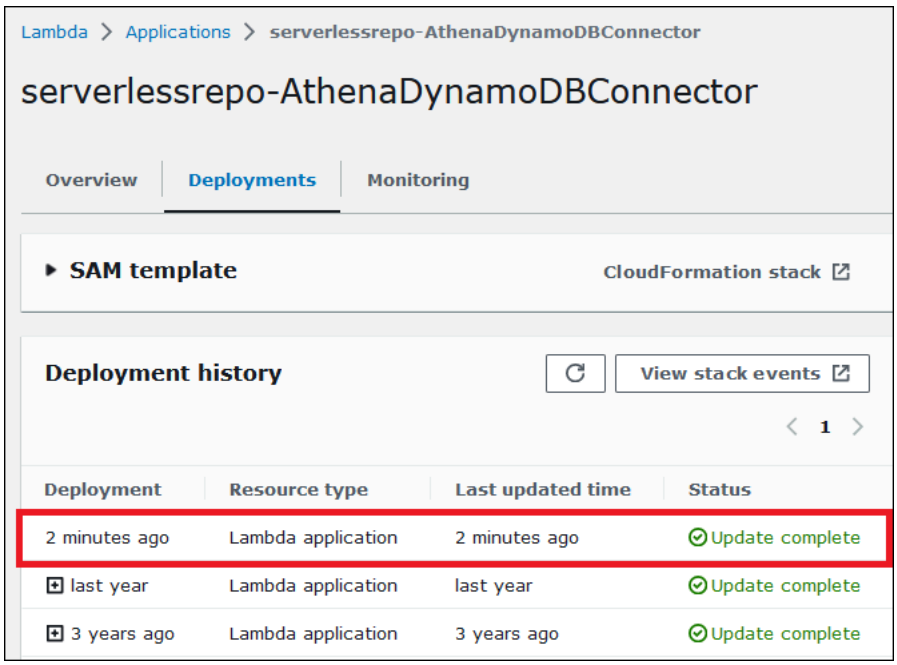
   a. 애플리케이션 이름에서 serverlessrepo- 접두사를 제거했는지 확인하고 애플리케이션 이름을 애플리케이션 이름 필드에 복사합니다.

   b. Lambda 함수 이름을 AthenaCatalogName 필드에 복사합니다. 일부 커넥터에서 이 필드는 LambdaFunctionName으로 표시됩니다.

   c. 기록했던 환경 변수를 해당 필드에 복사합니다.

9. I acknowledge that this app creates custom IAM roles and resource policies 옵션을 선택하고 배포를 선택합니다.
10. 애플리케이션이 업데이트되었는지 확인하려면 배포 탭을 선택합니다.

    배포 기록 섹션에서 업데이트가 완료되었음을 표시합니다.



11. 새 버전 번호를 확인하려면 전과 같이 SAM 템플릿을 확장하고 CodeUri를 찾은 후에 키 필드에서 커넥터 버전 번호를 확인할 수 있습니다.

이제 업데이트된 커넥터를 사용하여 Athena의 페더레이션된 쿼리를 생성할 수 있습니다.

페더레이션된 쿼리 실행

하나 이상의 데이터 커넥터를 구성하여 계정에 배포한 후 Athena 쿼리에 사용할 수 있습니다.

단일 데이터 원본 쿼리

이 섹션의 예제에서는 Amazon Athena CloudWatch 커넥터를 구성하여 계정에 배포했다고 가정합니다. 다른 커넥터를 사용할 때 동일한 방법을 사용하여 쿼리합니다.

CloudWatch 커넥터를 사용하는 Athena 쿼리를 생성하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. Athena 쿼리 편집기에서 `FROM` 절에 다음 구문을 사용하는 SQL 쿼리를 생성합니다.

```
MyCloudwatchCatalog.database_name.table_name
```

예제

다음 예제에서는 Athena CloudWatch 커넥터를 사용하여 `/var/ecommerce-engine/order-processor` CloudWatch Logs 로그 그룹의 `all_log_streams` 뷰에 연결합니다. `all_log_streams` 뷰는 로그 그룹에 있는 모든 로그 스트림의 뷰입니다. 예제 쿼리는 반환되는 행 수를 100으로 제한합니다.

```
SELECT *
FROM "MyCloudwatchCatalog"."/var/ecommerce-engine/order-processor".all_log_streams
LIMIT 100;
```

다음 예제에서는 이전 예제와 동일한 뷰에서 정보를 구문 분석합니다. 이 예제에서는 주문 ID와 로그 수준을 추출하고 `INFO` 수준이 있는 메시지를 필터링합니다.

```
SELECT
    log_stream as ec2_instance,
    Regexp_extract(message '.*orderId=(\d+) .*', 1) AS orderId,
    message AS order_processor_log,
    Regexp_extract(message, '(.*):.*', 1) AS log_level
FROM MyCloudwatchCatalog."/var/ecommerce-engine/order-processor".all_log_streams
WHERE Regexp_extract(message, '(.*):.*', 1) != 'INFO'
```

여러 데이터 원본 쿼리

좀 더 복잡한 예제로 다음 데이터 소스를 사용하여 고객 구매와 관련된 데이터를 저장하는 전자 상거래 회사를 가정합니다.

- Amazon RDS for MySQL: 제품 카탈로그 데이터 저장용
- Amazon DocumentDB: 이메일 주소 및 배송 주소와 같은 고객 계정 데이터 저장용
- Amazon DynamoDB: 주문 배송 및 추적 데이터 저장용

이 전자 상거래 애플리케이션의 데이터 분석가가 일부 지역의 배송 시간이 현지 기상 조건의 영향을 받았다는 사실을 알게 되었다고 가정합니다. 분석가는 지연된 주문 건수, 영향을 받는 고객의 위치, 영향을 가장 많이 받는 제품을 파악하려고 합니다. 분석가는 정보 소스를 개별적으로 조사하는 대신 Athena를 사용하여 하나의 페더레이션된 쿼리에서 데이터를 결합합니다.

Example

```
SELECT
     t2.product_name AS product,
     t2.product_category AS category,
     t3.customer_region AS region,
     count(t1.order_id) AS impacted_orders
FROM my_dynamodb.default.orders t1
JOIN my_mysql.products.catalog t2 ON t1.product_id = t2.product_id
JOIN my_documentdb.default.customers t3 ON t1.customer_id = t3.customer_id
WHERE
     t1.order_status = 'PENDING'
     AND t1.order_date between '2022-01-01' AND '2022-01-05'
GROUP BY 1, 2, 3
ORDER BY 4 DESC
```

페더레이션된 보기 쿼리

페더레이션된 소스를 쿼리할 때 보기를 사용하여 기본 데이터 소스를 난독화하거나 데이터를 쿼리하는 다른 분석가로부터 복잡한 조인을 숨길 수 있습니다.

고려 사항 및 제한

- 페더레이션된 보기에는 Athena 엔진 버전 3이 필요합니다.
- 페더레이션된 보기는 기본 데이터 소스가 아닌 AWS Glue에 저장됩니다.
- 페더레이션된 카탈로그로 생성된 보기는 다음 예제와 같이 정규화된 이름 구문을 사용해야 합니다.

  ```
  "ddbcatalog"."default"."customers"
  ```

- 페더레이션된 소스에서 쿼리를 실행하는 사용자는 페더레이션된 소스를 쿼리할 수 있는 권한이 있어야 합니다.
- 페더레이션된 보기에는 `athena:GetDataCatalog` 권한이 필요합니다. 자세한 내용은 Athena 연합 쿼리를 허용하는 IAM 권한 정책의 예제 단원을 참조하십시오.

예제

다음 예제에서는 페더레이션된 데이터 소스에 저장된 데이터에서 `customers` 보기를 생성합니다.

Example

```
CREATE VIEW customers AS
```

```
SELECT *
FROM my_federated_source.default.table
```

다음 쿼리 예제에서는 기본 페더레이션된 데이터 소스 대신, `customers` 보기를 참조하는 쿼리를 보여줍니다.

Example

```
SELECT id, SUM(order_amount)
FROM customers
GROUP by 1
ORDER by 2 DESC
LIMIT 50
```

다음 예제는 페더레이션된 데이터 소스와 Amazon S3 데이터 소스의 데이터를 결합하는 `order_summary` 보기를 생성합니다. Athena에서 이미 생성된 페더레이션된 소스에서 보기는 `person` 및 `profile` 테이블을 사용합니다. Amazon S3에서 보기는 `purchase` 및 `payment` 테이블을 사용합니다. Amazon S3를 참조하기 위해 명령문에서는 `awsdatacatalog` 키워드를 사용합니다. 페더레이션된 데이터 소스는 정규화된 이름 구문, *`federated_source_name`*`.`*`federated_source_database`*`.`*`federated_source_table`*을 사용합니다.

Example

```
CREATE VIEW default.order_summary AS
SELECT *
FROM federated_source_name.federated_source_database."person" p
    JOIN federated_source_name.federated_source_database."profile" pr ON pr.id = p.id
    JOIN awsdatacatalog.default.purchase i ON p.id = i.id
    JOIN awsdatacatalog.default.payment pay ON pay.id = p.id
```

추가적인 리소스

- 원래 소스와 분리되고 다중 사용자 모델에서 온디맨드 분석에 사용할 수 있는 페더레이션된 보기의 예제에 대해서는 AWS 빅 데이터 블로그의 Extend your data mesh with Amazon Athena and federated views를 참조하세요.
- Athena에서 보기 작업에 대한 자세한 내용은 뷰 작업 섹션을 참조하세요.

페더레이션 패스스루 쿼리 실행

Athena에서는 데이터 소스 자체의 쿼리 언어를 사용하여 페더레이션 데이터 소스에서 쿼리를 실행하고 실행을 위해 전체 쿼리를 데이터 소스로 푸시다운할 수 있습니다. 이러한 쿼리를 패스스루 쿼리라고 합니다. 패스스루 쿼리를 실행하려면 Athena 쿼리에서 테이블 함수를 사용합니다. 데이터 소스에서 실행할 패스스루 쿼리를 테이블 함수의 인수 중 하나에 포함시킵니다. 패스스루 쿼리는 Athena SQL을 사용하여 분석할 수 있는 테이블을 반환합니다.

지원되는 커넥터

다음 Athena 데이터 소스 커넥터는 패스스루 쿼리를 지원합니다.

- Azure 데이터 레이크 스토리지
- Azure Synapse
- Cloudera Hive
- Cloudera Impala
- CloudWatch
- Db2
- Db2 iSeries
- DocumentDB
- DynamoDB
- HBase
- Google BigQuery
- Hortonworks
- MySQL
- Neptune
- OpenSearch
- Oracle
- PostgreSQL
- Redshift
- SAP HANA
- Snowflake
- SQL Server

- Teradata
- Timestream
- Vertica

고려 사항 및 제한

Athena에서 패스스루 쿼리를 사용할 때 다음 사항을 고려하세요.

- 쿼리 패스스루는 Athena SELECT 문 또는 읽기 작업에만 지원됩니다.
- 패스스루 쿼리는 외부 쿼리의 카탈로그 컨텍스트(즉, 테이블 함수를 호출하는 쿼리) 내에서 실행되어야 합니다.
- 쿼리 성능은 데이터 소스의 구성에 따라 다를 수 있습니다.
- 패스스루 쿼리는 뷰에서 지원되지 않습니다.

구문

일반적인 Athena 쿼리 패스스루 구문은 다음과 같습니다.

```
SELECT * FROM TABLE(system.function_name(arg1 => 'arg1Value'[, arg2 =>
 'arg2Value', ...]))
```

대부분의 데이터 소스에서 첫 번째이자 유일한 인수 query이고 그 뒤에는 화살표 연산자 => 기호와 쿼리 문자열이 붙습니다.

```
SELECT * FROM TABLE(system.query(query => 'query string'))
```

간편함을 위해 선택적으로 명명된 query 인수와 화살표 연산자 => 기호를 생략할 수 있습니다.

```
SELECT * FROM TABLE(system.query('query string'))
```

데이터 소스에 쿼리 문자열 이상이 필요한 경우 데이터 소스에서 예상하는 순서대로 명명된 인수를 사용합니다. 예를 들어 *arg1* => '*arg1Value*' 표현식에는 첫 번째 인수와 해당 값이 포함됩니다. *arg1*이라는 이름은 데이터 소스에 따라 다르고 커넥터마다 다를 수 있습니다.

```
SELECT * FROM TABLE(
        system.query(
```

```
            arg1 => 'arg1Value',
            arg2 => 'arg2Value',
            arg3 => 'arg3Value'
        ));
```

특정 커넥터에 사용할 정확한 구문에 대한 자세한 내용은 개별 커넥터 페이지를 참조하세요.

따옴표 사용

전달한 쿼리 문자열을 비롯한 인수 값은 다음 예제와 같이 작은따옴표로 묶어야 합니다.

```
SELECT * FROM TABLE(system.query(query => 'SELECT * FROM testdb.persons LIMIT 10'))
```

쿼리 문자열을 큰따옴표로 묶으면 쿼리가 실패합니다. 다음 쿼리는 COLUMN_NOT_FOUND: line 1:43: Column 'select * from testdb.persons limit 10' cannot be resolved 오류 메시지와 함께 실패합니다.

```
SELECT * FROM TABLE(system.query(query => "SELECT * FROM testdb.persons LIMIT 10"))
```

작은따옴표를 이스케이프하려면 원본에 작은따옴표를 추가합니다(예: `terry''s_group`에 `terry's_group`).

예제

다음 예제 쿼리는 데이터 소스로 쿼리를 푸시다운합니다. 쿼리는 `customer` 테이블의 모든 열을 선택하여 결과를 10개로 제한합니다.

```
SELECT * FROM TABLE(
        system.query(
            query => 'SELECT * FROM customer LIMIT 10;'
        ))
```

다음 문은 동일한 쿼리를 실행하지만 선택적으로 명명된 `query` 인수와 화살표 연산자 `=>` 기호를 제거합니다.

```
SELECT * FROM TABLE(
        system.query(
            'SELECT * FROM customer LIMIT 10;'
        ))
```

Athena 및 페더레이션된 테이블 이름 한정자

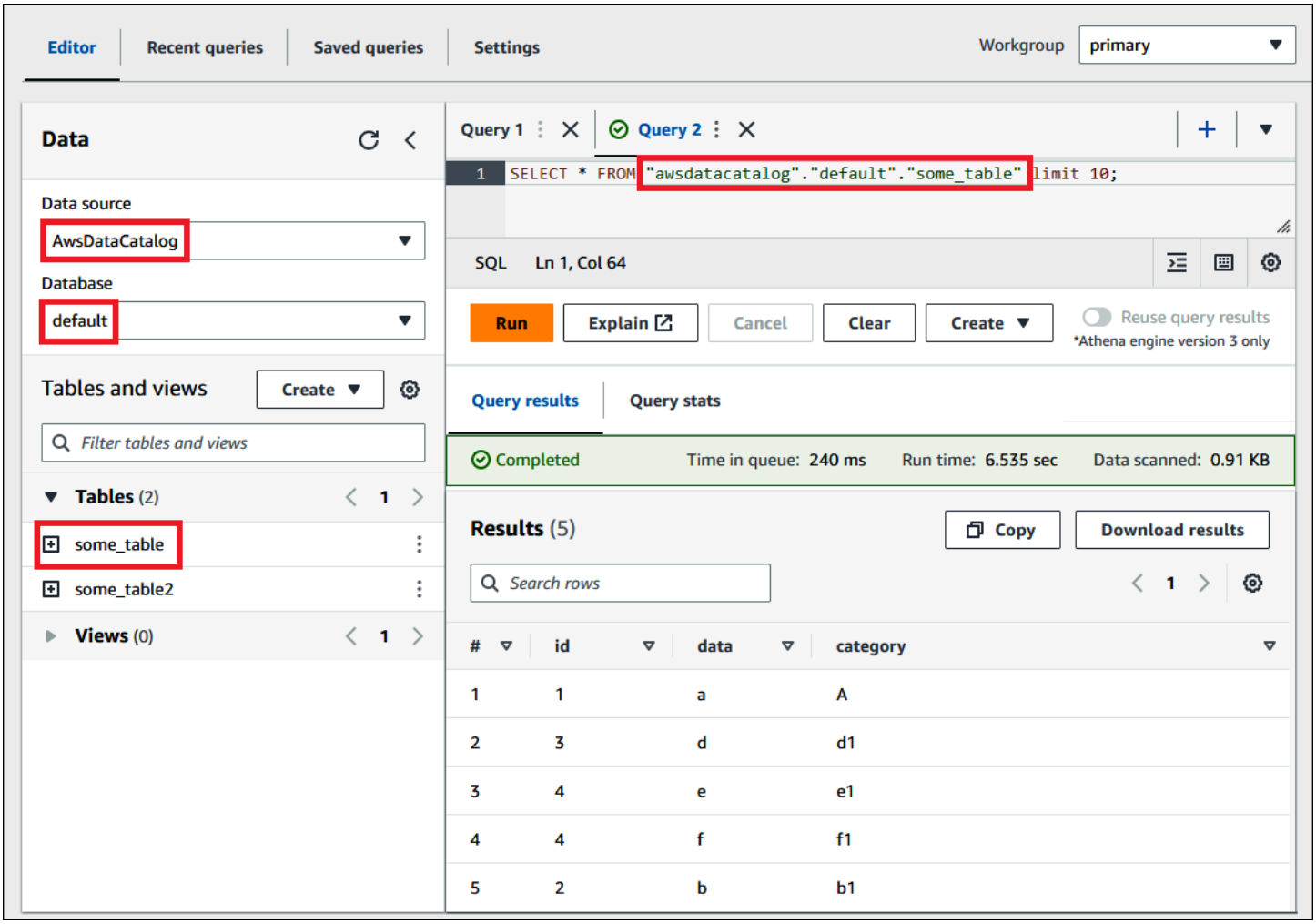
Athena는 다음 용어를 사용하여 데이터 객체의 계층 구조를 참조합니다.

- 데이터 소스 - 데이터베이스 그룹
- 데이터베이스 - 테이블 그룹
- 테이블 - 행 또는 열 그룹으로 구성된 데이터

때때로 이 객체는 다음과 같이 동등한 대체 이름으로 참조되기도 합니다.

- 데이터 소스를 카탈로그라고도 합니다.
- 데이터베이스를 스키마라고도 합니다.

Athena 콘솔의 다음 쿼리 예제는 `awsdatacatalog` 데이터 소스, `default` 데이터베이스 및 `some_table` 테이블을 사용합니다.



페더레이션된 데이터 소스의 용어

페더레이션된 데이터 소스를 쿼리하는 경우 기본 데이터 소스가 Athena와 같은 용어를 사용하지 않을 수 있다는 점에 유의합니다. 페더레이션된 쿼리를 작성할 때는 이 차이점을 염두에 두세요. 다음 섹션에서는 Athena의 데이터 객체 용어가 페더레이션된 데이터 소스의 데이터 객체 용어와 어떻게 연관되는지 설명합니다.

Amazon Redshift

Amazon Redshift 데이터베이스는 Redshift 테이블 그룹을 포함하는 Redshift 스키마 그룹입니다.

Athena	Redshift
Redshift 데이터 소스	Redshift `database`를 가리키도록 구성된 Redshift 커넥터 Lambda 함수입니다.
`data_source.database.table`	`database.schema.table`

쿼리 예

```
SELECT * FROM
Athena_Redshift_connector_data_source.Redshift_schema_name.Redshift_table_name
```

이 커넥터에 대한 자세한 내용은 Amazon Athena Redshift 커넥터 섹션을 참조하세요.

Cloudera Hive

Cloudera Hive 서버 또는 클러스터는 Cloudera Hive 테이블 그룹을 포함하는 Cloudera Hive 데이터베이스 그룹입니다.

Athena	Hive
Cloudera Hive 데이터 소스	Cloudera Hive `server`를 가리키도록 구성된 Cloudera Hive 커넥터 Lambda 함수입니다.
`data_source.database.table`	`server.database.table`

쿼리 예

```
SELECT * FROM
Athena_Cloudera_Hive_connector_data_source.Cloudera_Hive_database_name.Cloudera_Hive_table_name
```

이 커넥터에 대한 자세한 내용은 Amazon Athena Cloudera Hive 커넥터 섹션을 참조하세요.

Cloudera Impala

Impala 서버 또는 클러스터는 Impala 테이블 그룹을 포함하는 Impala 데이터베이스 그룹입니다.

Athena	Impala
Impala 데이터 소스	Impala `server`를 가리키도록 구성된 Impala 커넥터 Lambda 함수입니다.
`data_source.database.table`	`server.database.table`

쿼리 예

```
SELECT * FROM
Athena_Impala_connector_data_source.Impala_database_name.Impala_table_name
```

이 커넥터에 대한 자세한 내용은 Amazon Athena Cloudera Impala 커넥터 섹션을 참조하세요.

MySQL

MySQL 서버는 MySQL 테이블 그룹을 포함하는 MySQL 데이터베이스 그룹입니다.

Athena	MySQL
MySQL 데이터 소스	MySQL `server`를 가리키도록 구성된 MySQL 커넥터 Lambda 함수입니다.
`data_source.database.table`	`server.database.table`

쿼리 예

```
SELECT * FROM
Athena_MySQL_connector_data source.MySQL_database_name.MySQL_table_name
```

이 커넥터에 대한 자세한 내용은 Amazon Athena MySQL 커넥터 섹션을 참조하세요.

Oracle

Oracle 서버 또는 데이터베이스는 Oracle 테이블 그룹을 포함하는 Oracle 스키마 그룹입니다.

Athena	Oracle
Oracle 데이터 소스	Oracle `server`를 가리키도록 구성된 Oracle 커넥터 Lambda 함수입니다.
`data_source.database.table`	`server.schema.table`

쿼리 예

```
SELECT * FROM
Athena_Oracle_connector_data_source.Oracle_schema_name.Oracle_table_name
```

이 커넥터에 대한 자세한 내용은 Amazon Athena Oracle 커넥터 섹션을 참조하세요.

Postgres

Postgres 서버 또는 클러스터는 Postgres 데이터베이스 그룹입니다. Postgres 데이터베이스는 Postgres 테이블 그룹을 포함하는 Postgres 스키마 그룹입니다.

Athena	Postgres
Postgres 데이터 소스	Postgres `server` 및 `database`를 가리키도록 구성된 Postgres 커넥터 Lambda 함수입니다.
`data_source.database.table`	`server.database.schema.table`

쿼리 예

```
SELECT * FROM
Athena_Postgres_connector_data_source.Postgres_schema_name.Postgres_table_name
```

이 커넥터에 대한 자세한 내용은 Amazon Athena PostgreSQL 커넥터 섹션을 참조하세요.

Athena Query Federation SDK를 사용하여 데이터 소스 커넥터 개발

Athena Query Federation SDK를 사용해 자체적인 데이터 원본 커넥터를 쓸 수 있습니다. Athena Query Federation SDK는 Athena에서 사용자가 작성 및 배포하는 코드에 쿼리 실행 계획의 일부를 위임하는 데 사용할 수 있는 인터페이스 및 연결 프로토콜 집합을 정의합니다. SDK에는 커넥터 제품군과 커넥터 예제가 포함되어 있습니다.

직접 사용하기 위해 Amazon Athena의 사전 구축된 커넥터를 사용자 지정할 수도 있습니다. GitHub에서 소스 코드의 복사본을 수정한 다음 Connector publish tool을 사용하여 AWS Serverless Application Repository 패키지를 만들 수 있습니다. 이러한 방식으로 커넥터를 배포한 후에는 Athena 쿼리에 사용할 수 있습니다.

SDK를 다운로드하는 방법과 자체 커넥터 작성에 대한 자세한 지침은 GitHub의 Athena 커넥터 예제를 참조하세요.

Apache Spark용 Athena 데이터 소스 커넥터

일부 Athena 데이터 소스 커넥터는 Spark DSV2 커넥터로 사용할 수 있습니다. Spark DSV2 커넥터 이름에는 `-dsv2` 접미사(예: `athena-dynamodb-dsv2`)가 있습니다.

다음은 현재 사용 가능한 DSV2 커넥터, 해당 Spark `.format()` 클래스 이름, 해당 Amazon Athena 페더레이션된 쿼리 설명서 링크입니다.

DSV2 커넥터	Spark .format() 클래스 이름	설명서
athena-cloudwatch-dsv2	`com.amazonaws.athena.connectors.dsv2.cloudwatch.CloudwatchTableProvider`	CloudWatch
athena-cloudwatch-metrics-dsv2	`com.amazonaws.athena.connectors.dsv2.cloudwatch.metrics.CloudwatchMetricsTableProvider`	CloudWatch 지표
athena-aws-	`com.amazonaws.athena.connectors.dsv2`	CMDB

DSV2 커넥터	Spark .format() 클래스 이름	설명서
cmdb-dsv2	`.aws.cmdb.AwsCmdbTableProvider`	
athena-dynamodb-dsv2	`com.amazonaws.athena.connectors.dsv2.dynamodb.DDBTableProvider`	DynamoDB

DSV2 커넥터용 `.jar` 파일을 다운로드하려면 Amazon Athena Query Federation DSV2 GitHub 페이지를 방문하고 Releases, Release ***<version>***, Assets 섹션을 참조하세요.

Spark에 대해 jar 지정

Spark에서 Athena DSV2 커넥터를 사용하려면 사용하는 Spark 환경에 커넥터용 `.jar` 파일을 제출합니다. 다음 섹션에서는 구체적인 사례를 설명합니다.

Athena for Spark

Amazon Athena for Apache Spark에 사용자 지정 `.jar` 파일 및 사용자 지정 구성을 추가하는 방법에 대한 자세한 내용은 JAR 파일 및 사용자 지정 Spark 구성 추가 섹션을 참조하세요.

일반 Spark

커넥터 `.jar` 파일을 Spark로 전달하려면 다음 예제와 같이 `spark-submit` 명령을 사용하고 `--jars` 옵션에 `.jar` 파일을 지정합니다.

```
spark-submit \
  --deploy-mode cluster \
  --jars https://github.com/awslabs/aws-athena-query-federation-dsv2/releases/
download/some_version/athena-dynamodb-dsv2-some_version.jar
```

Amazon EMR Spark

Amazon EMR에서 `--jars` 파라미터와 함께 `spark-submit` 명령을 실행하려면 Amazon EMR Spark 클러스터에 단계를 추가해야 합니다. Amazon EMR에서 `spark-submit`을 사용하는 방법에 대한 자세한 내용은 Amazon EMR 릴리스 안내서의 Add a Spark step을 참조하세요.

AWS Glue ETL Spark

AWS Glue ETL의 경우 `.jar` 파일의 GitHub.com URL을 `aws glue start-job-run` 명령의 `--extra-jars` 인수로 전달할 수 있습니다. AWS Glue 설명서에는 Amazon S3 경로를 사용하며 `--extra-jars` 파라미터를 설명하지만, 파라미터는 HTTPS URL을 사용할 수도 있습니다. 자세한 내용은 AWS Glue 개발자 안내서의 Job parameter reference를 참조하세요.

Spark에서 커넥터 쿼리

Apache Spark에서 Athena의 기존 페더레이션된 쿼리와 동일한 쿼리를 제출하려면 `spark.sql()` 함수를 사용합니다. 예를 들어 다음과 같은 Athena 쿼리를 Apache Spark에서 사용하는 경우를 가정합니다.

```
SELECT somecola, somecolb, somecolc
FROM ddb_datasource.some_schema_or_glue_database.some_ddb_or_glue_table
WHERE somecola > 1
```

Amazon Athena DynamoDB DSV2 커넥터를 사용하여 Spark에서 동일한 쿼리를 수행하려면 다음 코드를 사용합니다.

```
dynamoDf = (spark.read
    .option("athena.connectors.schema", "some_schema_or_glue_database")
    .option("athena.connectors.table", "some_ddb_or_glue_table")
    .format("com.amazonaws.athena.connectors.dsv2.dynamodb.DDBTableProvider")
    .load())

dynamoDf.createOrReplaceTempView("ddb_spark_table")

spark.sql('''
SELECT somecola, somecolb, somecolc
FROM ddb_spark_table
WHERE somecola > 1
''')
```

파라미터 지정

Athena 데이터 소스 커넥터의 DSV2 버전은 해당 Athena 데이터 소스 커넥터와 동일한 파라미터를 사용합니다. 파라미터 정보는 해당 Athena 데이터 소스 커넥터의 설명서를 참조하세요.

PySpark 코드에서 다음 구문을 사용하여 파라미터를 구성합니다.

```
spark.read.option("athena.connectors.conf.parameter", "value")
```

예를 들어 다음 코드는 Amazon Athena DynamoDB 커넥터 `disable_projection_and_casing` 파라미터를 `always`로 설정합니다.

```
dynamoDf = (spark.read
    .option("athena.connectors.schema", "some_schema_or_glue_database")
    .option("athena.connectors.table", "some_ddb_or_glue_table")
    .option("athena.connectors.conf.disable_projection_and_casing", "always")
    .format("com.amazonaws.athena.connectors.dsv2.dynamodb.DDBTableProvider")
    .load())
```

데이터 카탈로그 액세스에 대한 IAM 정책

데이터 카탈로그에 대한 액세스를 제어하기 위해 리소스 수준 IAM 권한이나 자격 증명 기반 IAM 정책을 사용합니다.

다음은 Athena용 절차입니다.

IAM용 정보는 이 단원 끝부분에 나와 있는 링크를 참조하세요. JSON 데이터 카탈로그 정책 예제에 대한 자세한 내용은 데이터 카탈로그 예제 정책 단원을 참조하세요.

IAM 콘솔에서 시각적 편집기를 사용하려면 데이터 카탈로그 정책을 만듭니다.

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 정책을 선택한 후 정책 생성을 선택합니다.
3. 시각적 편집기(Visual editor) 탭에서 서비스 선택(Choose a service)을 선택합니다. 그런 다음 Athena를 선택하여 정책에 추가합니다.
4. 작업 선택(Select actions)을 선택한 후 정책에 추가할 작업을 선택합니다. 시각적 편집기에 Athena에서 사용할 수 있는 작업이 표시됩니다. 자세한 내용은 서비스 권한 부여 참조에서 Amazon Athena에 사용되는 작업, 리소스 및 조건 키를 참조하세요.
5. 작업 추가(add actions)를 선택하여 특정 작업을 입력하거나 와일드카드(*)를 사용하여 여러 개의 작업을 지정합니다.

   기본적으로 생성되는 정책은 사용자가 선택하는 작업을 허용합니다. Athena의 `datacatalog` 리소스에 대해 리소스 수준 권한을 지원하는 작업을 하나 이상 선택하면 편집기에 `datacatalog` 리소스가 나열됩니다.

6. 리소스를 선택하여 정책에 대한 특정 데이터 카탈로그를 지정합니다. JSON 데이터 카탈로그 정책 예제는 데이터 카탈로그 예제 정책 단원을 참조하세요.
7. 다음과 같이 datacatalog 리소스를 지정합니다.

```
arn:aws:athena:<region>:<user-account>:datacatalog/<datacatalog-name>
```

8. 정책 검토(Review policy)를 선택한 후 생성하려는 정책에 대한 이름(Name)과 설명(Description)(선택 사항)을 입력합니다. 정책 요약을 검토하여 의도한 권한을 부여했는지 확인합니다.
9. 정책 생성(Create policy)을 선택하고 새로운 정책을 저장합니다.
10. 사용자, 그룹 또는 역할에 이 자격 증명 기반 정책을 연결하고 사용자, 그룹 또는 역할이 액세스할 수 있는 datacatalog 리소스를 지정합니다.

자세한 내용은 서비스 권한 부여 참조와 IAM 사용 설명서에서 다음 주제를 참조하세요.

- Amazon Athena에 사용되는 작업, 리소스 및 조건 키
- 시각적 편집기를 사용하여 정책 생성
- IAM 정책 추가 및 제거
- 리소스에 대한 액세스 제어

JSON 데이터 카탈로그 정책 예제는 데이터 카탈로그 예제 정책 단원을 참조하세요.

AWS Glue 권한 및 AWS Glue 크롤러 권한에 대한 자세한 내용은 AWS Glue 개발자 안내서의 AWS Glue에 대한 IAM 권한 설정 및 크롤러 사전 조건을 참조하세요.

Amazon Athena 작업의 전체 목록은 Amazon Athena API 참조의 API 작업 이름을 참조하세요.

데이터 카탈로그 예제 정책

이 단원에서는 데이터 카탈로그에 다양한 작업을 허용하는 데 사용할 수 있는 예제 정책을 설명합니다.

데이터 카탈로그는 Athena에서 관리하는 IAM 리소스입니다. 따라서 해당 데이터 카탈로그 정책에서 datacatalog를 입력으로 받는 작업을 사용할 경우 다음과 같이 데이터 카탈로그의 ARN을 지정해야 합니다.

```
"Resource": [arn:aws:athena:<region>:<user-account>:datacatalog/<datacatalog-name>]
```

<datacatalog-name>은 데이터 카탈로그의 이름입니다. 예를 들어 다음과 같이 test_datacatalog라는 데이터 카탈로그를 리소스로 지정할 수 있습니다.

```
"Resource": ["arn:aws:athena:us-east-1:123456789012:datacatalog/test_datacatalog"]
```

Amazon Athena 작업의 전체 목록은 Amazon Athena API 참조의 API 작업 이름을 참조하세요. IAM 정책에 대한 자세한 내용은 IAM 사용 설명서의 시각적 편집기로 정책 생성을 참조하세요. IAM 정책을 생성하는 방법에 대한 자세한 내용은 데이터 카탈로그 액세스에 대한 IAM 정책 단원을 참조하세요.

- Example Policy for Full Access to All Data Catalogs
- Example Policy for Full Access to a Specified Data Catalog
- Example Policy for Querying a Specified Data Catalog
- Example Policy for Management Operations on a Specified Data Catalog
- Example Policy for Listing Data Catalogs
- Example Policy for Metadata Operations on Data Catalogs

Example 모든 데이터 카탈로그에 대해 전체 액세스를 허용하는 정책의 예

다음 정책은 계정에 있는 모든 데이터 카탈로그 리소스에 대해 전체 액세스를 허용합니다. 해당 계정에서 다른 모든 사용자에 대해 데이터 카탈로그를 관리해야 하는 사용자에게 이 정책을 사용하는 것이 좋습니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "athena:*"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

Example 지정된 데이터 카탈로그에 대해 전체 액세스를 허용하는 정책의 예

다음 정책은 `datacatalogA`라는 하나의 특정 데이터 카탈로그 리소스에 대해 전체 액세스를 허용합니다. 이 정책은 특정 데이터 카탈로그에 대해 전체 제어 권한을 가진 사용자에게 사용할 수 있습니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "athena:ListDataCatalogs",
            "athena:ListWorkGroups",
            "athena:GetDatabase",
            "athena:ListDatabases",
            "athena:ListTableMetadata",
            "athena:GetTableMetadata"
         ],
         "Resource":"*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "athena:StartQueryExecution",
            "athena:GetQueryResults",
            "athena:DeleteNamedQuery",
            "athena:GetNamedQuery",
            "athena:ListQueryExecutions",
            "athena:StopQueryExecution",
            "athena:GetQueryResultsStream",
            "athena:ListNamedQueries",
            "athena:CreateNamedQuery",
            "athena:GetQueryExecution",
            "athena:BatchGetNamedQuery",
            "athena:BatchGetQueryExecution",
            "athena:DeleteWorkGroup",
            "athena:UpdateWorkGroup",
            "athena:GetWorkGroup",
            "athena:CreateWorkGroup"
         ],
         "Resource":[
            "arn:aws:athena:us-east-1:123456789012:workgroup/*"
         ]
      },
      {
        "Effect":"Allow",
        "Action":[
            "athena:CreateDataCatalog",
```

```
            "athena:DeleteDataCatalog",
            "athena:GetDataCatalog",
            "athena:GetDatabase",
            "athena:GetTableMetadata",
            "athena:ListDatabases",
            "athena:ListTableMetadata",
            "athena:UpdateDataCatalog"
         ],
         "Resource":"arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA"
      }
   ]
}
```

Example 지정된 데이터 카탈로그 쿼리에 대한 정책의 예

다음 정책은 지정된 datacatalogA에서 사용자가 쿼리를 실행하도록 허용합니다. 사용자는 데이터 카탈로그에 대해 업데이트나 삭제 등의 관리 작업을 수행할 수 없습니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "athena:StartQueryExecution"
         ],
         "Resource":[
            "arn:aws:athena:us-east-1:123456789012:workgroup/*"
         ]
      },
      {
         "Effect":"Allow",
         "Action":[
            "athena:GetDataCatalog"
         ],
         "Resource":[
            "arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA"
         ]
      }
   ]
}
```

Example 지정된 데이터 카탈로그의 관리 작업을 위한 정책의 예

다음 정책은 사용자가 datacatalogA 데이터 카탈로그를 만들고, 삭제하고, 세부 정보를 조회하고, 업데이트할 수 있도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateDataCatalog",
                "athena:GetDataCatalog",
                "athena:DeleteDataCatalog",
                "athena:UpdateDataCatalog"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA"
            ]
        }
    ]
}
```

Example 데이터 카탈로그 목록 조회에 대한 정책의 예

다음 정책은 모든 사용자가 모든 데이터 카탈로그의 목록을 조회할 수 있도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListDataCatalogs"
            ],
            "Resource": "*"
        }
    ]
}
```

Example 데이터 카탈로그 메타데이터 작업에 대한 정책의 예

다음 정책은 데이터 카탈로그에 대한 메타데이터 작업을 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:GetDatabase",
                "athena:GetTableMetadata",
                "athena:ListDatabases",
                "athena:ListTableMetadata"
            ],
            "Resource": "*"
        }
    ]
}
```

데이터 원본 관리

Athena 콘솔의 데이터 원본(Data Sources) 페이지를 사용하여 생성한 데이터 원본을 관리할 수 있습니다.

데이터 원본을 보려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

3. 탐색 창에서 데이터 원본(Data sources)을 선택합니다.

4. 데이터 원본 목록에서 보려는 데이터 원본의 이름을 선택합니다.

> **Note**
>
> Data source name(데이터 소스 이름) 열의 항목은 ListDataCatalogs API 작업 및 list-data-catalogs CLI 명령의 출력에 해당합니다.

데이터 원본을 편집하려면

1. 데이터 원본(Data sources) 페이지에서 다음을 수행합니다.
   - 카탈로그 이름 옆에 있는 버튼을 선택한 다음 작업(Actions), 편집(Edit)을 선택합니다.
   - 데이터 원본의 이름을 선택합니다. 세부 정보 페이지에서 작업(Actions), 편집(Edit)을 선택합니다.
2. 편집(Edit) 페이지에서 데이터 원본에 대해 다른 Lambda 함수를 선택하거나, 설명을 변경하거나, 사용자 정의 태그를 추가할 수 있습니다. 태그에 대한 자세한 내용은 Athena 리소스 태깅 단원을 참조하세요.
3. Save(저장)를 선택합니다.
4. AwsDataCatalog 데이터 원본을 편집하려면 AwsDataCatalog 링크를 선택하여 세부 정보 페이지를 엽니다. 그런 다음 세부 정보 페이지에서 카탈로그를 편집할 수 있는 AWS Glue 콘솔에 대한 링크를 선택합니다.

데이터 원본 공유

데이터 원본 공유에 대한 자세한 내용을 보려면 다음 링크를 방문하세요.

- 비 Hive Lambda 기반 데이터 원본은 계정 간 연합 쿼리 사용 설정 섹션을 참조하세요.
- AWS Glue Data Catalog는 AWS Glue 데이터 카탈로그에 대한 교차 계정 액세스 섹션을 참조하세요.

데이터 소스를 삭제하기

1. 데이터 원본(Data sources) 페이지에서 다음을 수행합니다.
   - 카탈로그 이름 옆에 있는 버튼을 선택하고 작업(Actions), 삭제(Delete)를 선택합니다.

- 데이터 원본의 이름을 선택한 다음 세부 정보 페이지에서 작업(Actions), 삭제(Delete)를 선택합니다.

> **Note**
>
> AwsDataCatalog는 계정의 기본 데이터 원본이며 삭제할 수 없습니다.

데이터 원본을 삭제하면 해당 데이터 카탈로그, 테이블 및 뷰가 쿼리 편집기에서 제거된다는 경고가 표시됩니다. 데이터 원본을 사용한 저장된 쿼리가 더 이상 Athena에서 실행되지 않습니다.

2. 삭제를 확인하려면 데이터 원본의 이름을 입력한 다음 삭제(Delete)를 선택합니다.

Athena에서 Amazon DataZone 사용

Amazon DataZone을 사용하여 조직 경계를 넘어 대규모로 데이터를 공유, 검색 및 발견할 수 있습니다. DataZone은 Athena, AWS Glue 및 AWS Lake Formation과 같은 AWS 분석 서비스 전반의 경험을 단순화합니다. 예를 들어, 다양한 데이터 소스에 페타바이트 규모의 데이터가 있는 경우 Amazon DataZone을 사용하여 사람, 데이터 및 도구의 그룹화를 기반으로 비즈니스 사용 사례를 구축할 수 있습니다. 자세한 내용은 What is Amazon DataZone?을 참조하세요.

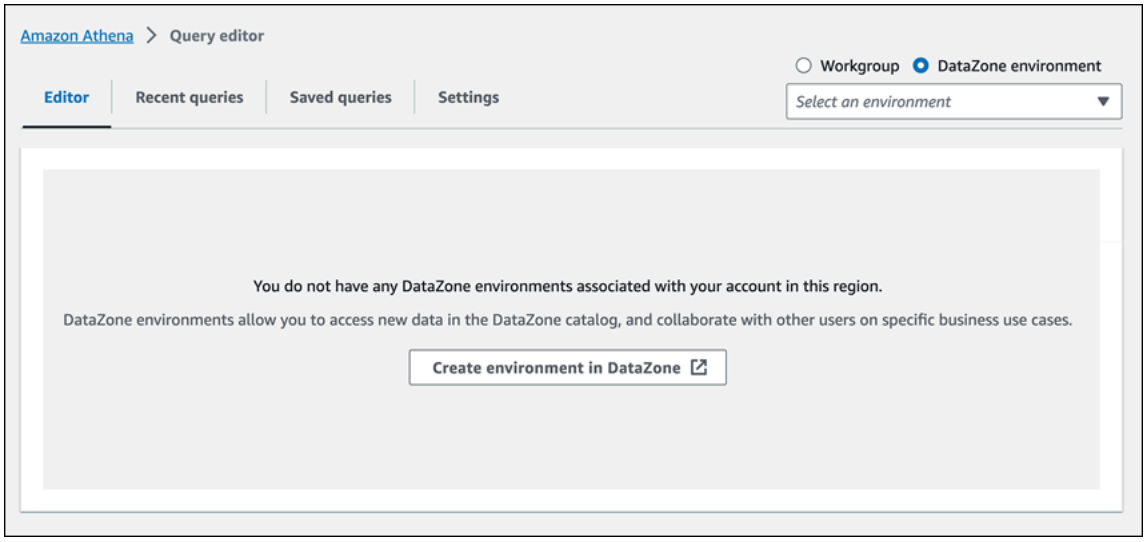
Athena에서는 쿼리 편집기를 사용하여 DataZone 환경에 액세스하고 쿼리할 수 있습니다. DataZone 환경은 DataZone 프로젝트와 도메인 조합을 지정합니다. Athena 콘솔에서 DataZone 환경을 사용하면 DataZone 환경의 IAM 역할을 수임하게 되며 해당 환경에 속하는 데이터베이스와 테이블만 표시됩니다. 권한은 DataZone에서 지정하는 역할에 따라 결정됩니다.

Athena에서는 쿼리 편집기 페이지의 DataZone 환경 선택기를 사용하여 DataZone 환경을 선택할 수 있습니다.

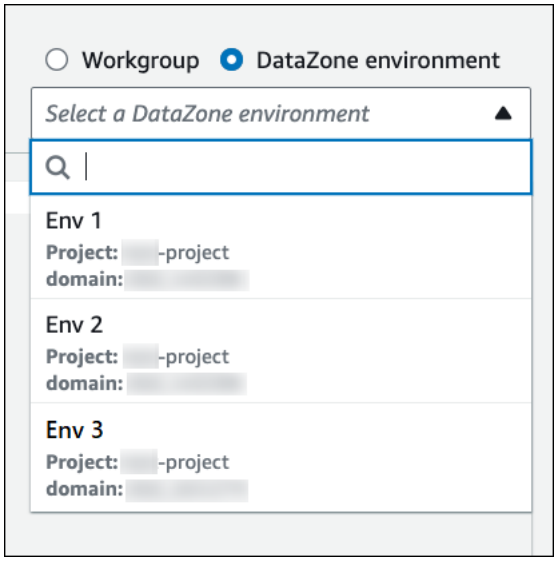
Athena에서 DataZone 환경 열기

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. Athena 콘솔 오른쪽 상단의 작업 그룹 옆에서 DataZone 환경을 선택합니다.

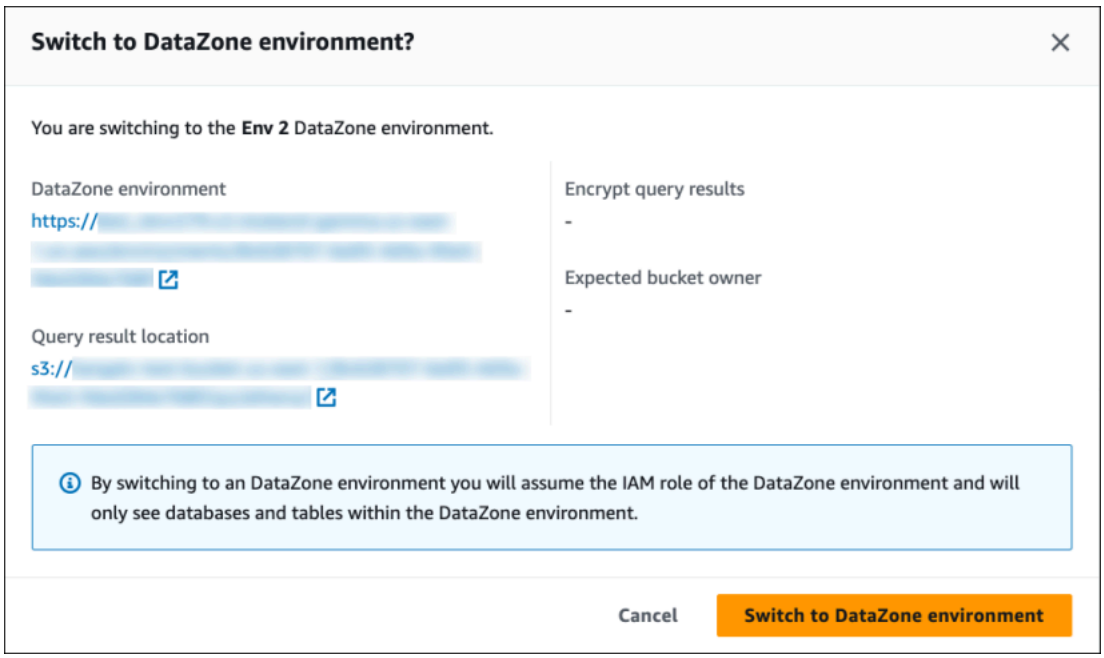
> **Note**
>
> DataZone 환경 옵션은 DataZone에서 사용 가능한 도메인이 하나 이상 있는 경우에만 제공됩니다.



3. DataZone 환경 선택기를 사용하여 DataZone 환경을 선택합니다.



4. DataZone 환경으로 전환 대화 상자에서 해당 환경이 원하는 환경인지 확인한 다음 DataZone 환경으로 전환을 선택합니다.



DataZone과 Athena를 시작하는 방법에 대한 자세한 내용은 Amazon DataZone User Guide의 Getting started 자습서를 참조하세요.

ODBC 및 JDBC 드라이버로 Amazon Athena에 연결

비즈니스 인텔리전스 도구를 사용하여 데이터를 탐색하고 시각화하려면 ODBC(Open Database Connectivity) 또는 JDBC(Java Database Connectivity) 드라이버를 다운로드, 설치하고 구성합니다.

주제

- JDBC로 Amazon Athena에 연결
- ODBC로 Amazon Athena에 연결

다음 AWS 지식 센터와 AWS Big Data Blog(빅 데이터 블로그) 주제도 참조하세요.

- JDBC 드라이버를 사용해 Athena에 연결할 때 IAM 역할 자격 증명을 사용하거나 다른 IAM 역할로 전환하는 방법은 어떻게 되나요?
- ODBC 드라이버로 Amazon Athena에 연결하기 위한 ADFS와 AWS 간의 신뢰 설정 및 Active Directory 자격 증명 사용

JDBC로 Amazon Athena에 연결

Amazon Athena는 버전 2.x와 3.x라는 두 가지 JDBC 드라이버를 제공합니다. Athena JDBC 3.x 드라이버는 더 나은 성능과 호환성을 제공하는 차세대 드라이버입니다. JDBC 3.x 드라이버는 Amazon S3에서 직접 쿼리 결과 읽기를 지원하므로 쿼리 결과를 대량으로 사용하는 애플리케이션의 성능이 개선됩니다. 또한 새 드라이버는 타사 종속성이 적기 때문에 BI 도구 및 사용자 지정 애플리케이션과의 더욱 간편한 통합이 가능합니다. 대부분의 경우 기존 구성을 전혀 변경하지 않거나 최소한의 변경으로 새 드라이버를 사용할 수 있습니다.

- JDBC 3.x 드라이버를 다운로드하려면 Athena JDBC 3.x 드라이버 섹션을 참조하세요.
- JDBC 2.x 드라이버를 다운로드하려면 Athena JDBC 2.x 드라이버 섹션을 참조하세요.

주제

Athena JDBC 3.x 드라이버

Athena JDBC 드라이버를 사용하여 여러 서드 파티 SQL 클라이언트 도구와 사용자 지정 애플리케이션에서 Amazon Athena에 연결할 수 있습니다.

시스템 사양

- Java 8 이상의 런타임 환경
- 사용 가능한 디스크 공간 20MB 이상

고려 사항 및 제한

다음은 Athena JDBC 3.x 드라이버에 대한 몇 가지 고려 사항과 제한 사항입니다.

- 로깅 - 3.x 드라이버는 런타임 시 여러 로깅 시스템 중 하나를 사용할 수 있게 하는 추상화 계층인 SLF4J를 사용합니다.
- 암호화 – `CSE_KMS` 암호화 옵션과 함께 Amazon S3 페처를 사용하는 경우 Amazon S3 클라이언트는 Amazon S3 버킷에 저장된 결과를 해독할 수 없습니다. `CSE_KMS` 암호화가 필요한 경우 스트리밍 페처를 계속 사용할 수 있습니다. Amazon S3 페처를 사용한 `CSE_KMS` 암호화 지원이 계획되어 있습니다.

JDBC 3.x 드라이버 다운로드

이 섹션에는 JDBC 3.x 드라이버의 다운로드 및 라이선스 정보가 들어 있습니다.

> **Important**
>
> JDBC 3.x 드라이버를 사용할 때는 다음 요구 사항에 유의하세요.
>
> - 포트 444 개방 – Athena가 쿼리 결과를 스트리밍하는 데 사용하는 포트 444를 유지하고 아웃바운드 트래픽에 개방합니다. PrivateLink 엔드포인트를 사용하여 Athena에 연결하는 경우, PrivateLink 엔드포인트에 연결된 보안 그룹이 포트 444의 인바운드 트래픽에 개방되어 있는지 확인합니다.
> - athena:GetQueryResultsStream 정책 – JDBC 드라이버를 사용하는 IAM 보안 주체에 대한 `athena:GetQueryResultsStream` 정책 작업을 추가합니다. 이 정책 작업은 API를 통해 직접 노출되지 않습니다. 이 정책 작업은 스트리밍 결과 지원의 일부로 ODBC 및 JDBC 드라이버에만 사용됩니다. 정책 예제는 AWS 관리형 정책: AWSQuicksightAthenaAccess을 참조하세요.

Amazon Athena 3.x JDBC 드라이버를 다운로드하려면 다음 링크를 방문하세요.

JDBC 드라이버 uber jar

다음 다운로드는 드라이버와 모든 종속 항목을 동일한 `.jar` 파일에 패키징합니다. 이 다운로드는 일반적으로 서드 파티 SQL 클라이언트에 사용됩니다.

3.2.0 uber jar

JDBC 드라이버 lean jar

다음 다운로드는 드라이버를 위한 lean `.jar`와 드라이버의 종속 항목을 위한 별도의 `.jar` 파일이 포함된 `.zip` 파일입니다. 이 다운로드는 드라이버가 사용하는 종속 항목과 충돌하는 종속 항목이 있을 수 있는 사용자 지정 애플리케이션에 주로 사용됩니다. 이 다운로드는 lean jar에 포함할 드라이버 종속 항목과 사용자 지정 애플리케이션에 이미 하나 이상의 드라이버 종속 항목이 포함되어 있는 경우 제외할 드라이버 종속 항목을 선택하려는 경우에 유용합니다.

3.2.0 lean jar

라이선스

다음 링크에는 JDBC 3.x 드라이버에 대한 라이선스 계약이 포함되어 있습니다.

[라이선스]

주제

- JDBC 3.x 드라이버 시작하기
- Amazon Athena JDBC 3.x 연결 파라미터
- 기타 JDBC 3.x 구성
- Amazon Athena JDBC 3.x 릴리스 정보
- Athena JDBC 3.x 드라이버 이전 버전

JDBC 3.x 드라이버 시작하기

이 섹션의 정보를 사용하여 Amazon Athena JDBC 3.x 드라이버를 시작하세요.

주제

- 설치 지침
- 드라이버 실행
- 드라이버 구성
- Athena JDBC v2 드라이버에서 업그레이드

설치 지침

사용자 지정 애플리케이션 또는 서드 파티 SQL 클라이언트에서 JDBC 3.x 드라이버를 사용할 수 있습니다.

사용자 지정 애플리케이션에서

드라이버 jar 및 해당 종속 항목이 포함된 `.zip` 파일을 다운로드합니다. 종속 항목마다 고유한 `.jar` 파일이 있습니다. 사용자 지정 애플리케이션에서 드라이버 jar를 종속 항목으로 추가합니다. 다른 소스에서 애플리케이션에 드라이버 jar의 종속 항목을 이미 추가했는지 여부에 따라 해당 종속 항목을 선택적으로 추가합니다.

서드 파티 SQL 클라이언트에서

드라이버 uber jar 파일을 다운로드하고 해당 클라이언트에 대한 지침에 따라 서드 파티 SQL 클라이언트에 추가합니다.

드라이버 실행

드라이버를 실행하려면 사용자 지정 애플리케이션이나 서드 파티 SQL 클라이언트를 사용합니다.

사용자 지정 애플리케이션에서

JDBC 인터페이스를 사용하여 프로그램에서 JDBC 드라이버와 상호 작용합니다. 다음 코드는 샘플 사용자 지정 Java 애플리케이션을 보여줍니다.

```
public static void main(String args[]) throws SQLException {
    Properties connectionParameters = new Properties();
    connectionParameters.setProperty("Workgroup", "primary");
    connectionParameters.setProperty("Region", "us-east-2");
    connectionParameters.setProperty("Catalog", "AwsDataCatalog");
    connectionParameters.setProperty("Database","sampledatabase");
    connectionParameters.setProperty("OutputLocation","s3://DOC-EXAMPLE-BUCKET");
    connectionParameters.setProperty("CredentialsProvider","DefaultChain");
    String url = "jdbc:athena://";
    AthenaDriver driver = new AthenaDriver();
    Connection connection = driver.connect(url, connectionParameters);
    Statement statement = connection.createStatement();
    String query = "SELECT * from sample_table LIMIT 10";
    ResultSet resultSet = statement.executeQuery(query);
    printResults(resultSet); // A custom-defined method for iterating over a
                             // result set and printing its contents
}
```

서드 파티 SQL 클라이언트에서

사용 중인 SQL 클라이언트에 대한 설명서를 따릅니다. 일반적으로 SQL 클라이언트의 그래픽 사용자 인터페이스를 사용하여 쿼리를 입력하고 제출하면 쿼리 결과가 동일한 인터페이스에 표시됩니다.

드라이버 구성

연결 파라미터를 사용하여 Amazon Athena JDBC 드라이버를 구성할 수 있습니다. 지원되는 연결 파라미터는 Amazon Athena JDBC 3.x 연결 파라미터 섹션을 참조하세요.

사용자 지정 애플리케이션에서

사용자 지정 애플리케이션에서 JDBC 드라이버의 연결 파라미터를 설정하려면 다음 중 하나를 수행합니다.

- 파라미터 이름과 해당 값을 `Properties` 객체에 추가합니다. `Connection#connect`를 직접적으로 호출할 때 해당 객체를 URL과 함께 전달합니다. 예제는 드라이버 실행의 샘플 Java 애플리케이션을 참조하세요.
- 연결 문자열(URL)에서 다음 형식을 사용하여 프로토콜 접두사 바로 뒤에 파라미터 이름과 해당 값을 추가합니다.

  ```
  <parameterName>=<parameterValue>;
  ```

  다음 예제와 같이 각 파라미터 이름/파라미터 값 페어의 끝에 세미콜론을 사용하고 세미콜론 뒤에 공백을 남기지 않습니다.

  ```
  String url = "jdbc:athena://WorkGroup=primary;Region=us-east-1;...;";AthenaDriver
   driver = new AthenaDriver();Connection connection = driver.connect(url, null);
  ```

  > **Note**
  >
  > 연결 문자열과 `Properties` 객체 모두에 파라미터가 지정된 경우 연결 문자열의 값이 우선합니다. 두 위치 모두에 동일한 파라미터를 지정하는 것은 권장되지 않습니다.

- 다음 예제와 같이 파라미터 값을 `AthenaDataSource`의 메서드에 인수로 추가합니다.

  ```
  AthenaDataSource dataSource = new AthenaDataSource();
      dataSource.setWorkGroup("primary");
      dataSource.setRegion("us-east-2");
      ...
      Connection connection = dataSource.getConnection();
      ...
  ```

서드 파티 SQL 클라이언트에서

사용 중인 SQL 클라이언트의 지침을 따릅니다. 일반적으로 클라이언트는 파라미터 이름과 해당 값을 입력할 수 있는 그래픽 사용자 인터페이스를 제공합니다.

Athena JDBC v2 드라이버에서 업그레이드

대부분의 JDBC 버전 3 연결 파라미터는 이전 버전인 버전 2(Simba) JDBC 드라이버와 호환됩니다. 즉, 버전 2 연결 문자열을 드라이버 버전 3에서 재사용할 수 있습니다. 그러나 일부 연결 파라미터가 변경

되었습니다. 이러한 변경 사항은 여기에서 설명합니다. 버전 3 JDBC 드라이버로 업그레이드할 때 필요한 경우 기존 구성을 업데이트하세요.

드라이버 클래스

일부 BI 도구는 JDBC 드라이버 `.jar` 파일에서 드라이버 클래스를 제공하도록 요청합니다. 대부분의 도구는 이 클래스를 자동으로 찾습니다. 버전 3 드라이버에서 클래스의 정규화된 이름은 `com.amazon.athena.jdbc.AthenaDriver`입니다. 버전 2 드라이버에서 클래스는 `com.simba.athena.jdbc.Driver`였습니다.

연결 문자열

버전 3 드라이버는 JDBC 연결 문자열 URL 시작 부분의 프로토콜에 `jdbc:athena://`를 사용합니다. 버전 3 드라이버는 버전 2 프로토콜 `jdbc:awsathena://`도 지원하지만 버전 2 프로토콜은 더 이상 사용되지 않습니다. 정의되지 않은 동작을 방지하기 위해 버전 3에서는 버전 2(또는 `jdbc:awsathena://`로 시작하는 연결 문자열을 허용하는 다른 드라이버)가 DriverManager 클래스에 등록된 경우 `jdbc:awsathena://`로 시작하는 연결 문자열을 허용하지 않습니다.

보안 인증 제공업체

버전 2 드라이버는 정규화된 이름(예: `com.simba.athena.amazonaws.auth.DefaultAWSCredentialsProviderChain`)을 사용하여 서로 다른 보안 인증 제공업체를 식별합니다. 버전 3 드라이버는 더 짧은 이름(예: `DefaultChain`)을 사용합니다. 새 이름은 각 보안 인증 제공업체의 해당 섹션에 설명되어 있습니다.

새로운 AWS SDK for Java의 AwsCredentialsProvider 인터페이스를 지난 AWS SDK for Java의 AWSCredentialsProvider 인터페이스 대신 구현하려면 버전 2 드라이버용으로 작성된 사용자 지정 보안 인증 제공업체를 버전 3 드라이버용으로 수정해야 합니다.

`PropertiesFileCredentialsProvider`는 JDBC 3.x 드라이버에서 지원되지 않습니다. 이 공급자는 JDBC 2.x 드라이버에서 사용되었지만 지원이 거의 종료되어가는 이전 버전의 Java용 AWS SDK에 속합니다. JDBC 3.x 드라이버에서 동일한 기능을 사용하려면 AWS 구성 프로파일 보안 인증 공급자를 대신 사용하세요.

로그 수준

다음 표에서는 JDBC 버전 2 드라이버와 버전 3 드라이버의 `LogLevel` 파라미터 차이점을 보여줍니다.

JDBC 드라이버 버전	파라미터 이름	파라미터 유형	기본값	가능한 값	연결 문자열 예제
v2	LogLevel	선택 사항	0	0~6	LogLevel=6;
v3	LogLevel	선택 사항	TRACE	OFF, ERROR, WARN, INFO, DEBUG, TRACE	LogLevel=INFO;

쿼리 ID 검색

버전 2 드라이버에서는 #getPReparedQueryId와 #getQueryId라는 두 가지 메서드가 있는 인터페이스인 com.interfaces.core.IStatementQueryInfoProvider에 대해 Statement 인스턴스를 언래핑합니다. 이러한 메서드를 사용하여 실행된 쿼리의 쿼리 실행 ID를 얻을 수 있습니다.

버전 3 드라이버에서 com.amazon.athena.jdbc.AthenaResultSet 인터페이스에 대한 Statement, PreparedStatement 및 ResultSet 인스턴스를 언래핑합니다. 인터페이스에는 #getQueryExecutionId라는 하나의 메서드가 있습니다.

Amazon Athena JDBC 3.x 연결 파라미터

지원되는 연결 파라미터는 여기서 기본 연결 파라미터, 고급 연결 파라미터 및 인증 연결 파라미터의 세 가지 섹션으로 나뉩니다. 고급 연결 파라미터 및 인증 연결 파라미터 섹션에는 관련 파라미터를 함께 그룹화하는 하위 섹션이 있습니다.

주제

- 기본 연결 파라미터
- 고급 연결 파라미터
- 인증 연결 파라미터

기본 연결 파라미터

다음 섹션에서는 JDBC 3.x 드라이버의 기본 연결 파라미터에 대해 설명합니다.

지역

쿼리가 실행될 AWS 리전입니다. 리전 목록은 Amazon Athena 엔드포인트 및 할당량을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
지역	AwsRegion (사용되지 않음)	필수(제공되지 않은 경우에는 DefaultAwsRegionProviderChain을 사용하여 검색됨)	없음

카탈로그

드라이버로 액세스할 데이터베이스와 테이블이 들어 있는 카탈로그입니다. 카탈로그에 대한 자세한 내용은 DataCatalog를 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
카탈로그	없음	선택 사항	AwsDataCatalog

데이터베이스

쿼리가 실행될 데이터베이스입니다. 데이터베이스 이름으로 명시적으로 한정되지 않은 테이블은 이 데이터베이스로 확인됩니다. 데이터베이스에 대한 자세한 내용은 데이터베이스를 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
데이터베이스	스키마	선택 사항	기본값

작업 그룹

쿼리가 실행될 작업 그룹입니다. 작업 그룹에 대한 자세한 내용은 WorkGroup을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
WorkGroup	없음	선택 사항	primary

출력 위치

쿼리 결과가 저장되는 Amazon S3의 위치입니다. 출력 위치에 대한 자세한 내용은 ResultConfiguration을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
OutputLocation	S3OutputLocation(사용되지 않음)	필수(작업 그룹이 출력 위치를 지정하지 않는 경우)	없음

고급 연결 파라미터

다음 섹션에서는 JDBC 3.x 드라이버의 고급 연결 파라미터에 대해 설명합니다.

주제

- 결과 암호화 파라미터
- 결과 가져오기 파라미터
- 쿼리 결과 재사용 파라미터
- 쿼리 실행 폴링 파라미터
- 엔드포인트 재정의 파라미터
- 프록시 구성 파라미터
- 로깅 파라미터
- 애플리케이션 이름
- 연결 테스트
- 재시도 횟수

결과 암호화 파라미터

다음 사항에 주의하세요.

- `EncryptionOption`이 `SSE_KMS` 또는 `CSE_KMS`인 경우 AWS KMS 키를 지정해야 합니다.
- `EncryptionOption`이 지정되지 않거나 `EncryptionOption`이 `SSE_S3`인 경우 AWS KMS 키를 지정할 수 없습니다.

암호화 옵션

Amazon S3에 저장되는 쿼리 결과에 사용할 암호화 유형입니다. 쿼리 결과 암호화에 대한 자세한 내용은 Amazon Athena API 참조의 EncryptionConfiguration을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값	가능한 값
EncryptionOption	S3OutputEncOption(사용되지 않음)	선택 사항	없음	SSE_S3, SSE_KMS, CSE_KMS

KMS 키

암호화 옵션으로 SSE_KMS 또는 CSE_KMS가 선택된 경우 KMS 키 ARN 또는 ID입니다. 자세한 내용은 Amazon Athena API 참조의 EncryptionConfiguration을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
KmsKey	S3OutputEncKMSKey(사용되지 않음)	선택 사항	없음

결과 가져오기 파라미터

결과 페처

쿼리 결과를 다운로드하는 데 사용될 페처입니다.

기본 결과 페처인 S3는 Athena API를 사용하지 않고 Amazon S3에서 직접 쿼리 결과를 다운로드합니다. 대부분의 경우 이것이 가장 빠른 옵션입니다. 쿼리 결과가 CSE_KMS로 암호화되거나 쿼리 결과에 대한 사용자 액세스를 허용하는 정책이 s3:CalledVia를 사용하는 Athena의 호출만 허용하는 경우에는 이 옵션을 사용할 수 없습니다.

파라미터 이름	별칭	파라미터 유형	기본값	가능한 값
ResultFetcher	없음	선택 사항	S3	S3, GetQueryResults,

파라미터 이름	별칭	파라미터 유형	기본값	가능한 값
				GetQueryResultsStream

> **Note**
>
> JDBC 2.x 드라이버에서 `UseResultsetStreaming = 1` 설정은 드라이버가 결과 세트 스트리밍 API를 사용하도록 구성합니다. JDBC 3.x 드라이버에서 이와 동일한 설정은 `ResultFetcher=GetQueryResultsStream`입니다.

가져오기 크기

이 파라미터의 값은 내부 버퍼의 최솟값 및 결과를 가져올 때 대상 페이지 크기로 사용됩니다. 값 0은 드라이버가 아래에 설명된 대로 기본값을 사용해야 함을 의미합니다. 최댓값은 100만입니다.

파라미터 이름	별칭	파라미터 유형	기본값
FetchSize	RowsToFetchPerBlock(사용되지 않음)	선택 사항	0

- `GetQueryResults` 페처는 항상 API 호출에서 지원하는 최댓값인 1,000(페이지 크기)을 사용합니다. 가져오기 크기가 1,000보다 크면 버퍼를 최솟값 이상으로 채우기 위해 여러 개의 연속 API 호출이 수행됩니다.
- `GetQueryResultsStream` 페처는 구성된 가져오기 크기를 페이지 크기로 사용하거나 기본적으로 10,000을 사용합니다.
- `S3` 페처는 구성된 가져오기 크기를 페이지 크기로 사용하거나 기본적으로 10,000을 사용합니다.

쿼리 결과 재사용 파라미터

결과 재사용 활성화

쿼리가 실행될 때 동일한 쿼리에 대한 이전 결과를 재사용할 수 있는지 여부를 지정합니다. 쿼리 결과 재사용에 대한 자세한 내용은 ResultReuseByAgeConfiguration을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
EnableResultReuseByAge	없음	선택 사항	FALSE

결과 재사용 최대 수명

Athena에서 재사용을 고려해야 하는 이전 쿼리 결과의 최대 수명(분)입니다. 결과 재사용 최대 기간에 대한 자세한 내용은 ResultReuseByAgeConfiguration을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
MaxResultReuseAgeInMinutes	없음	선택 사항	60

쿼리 실행 폴링 파라미터

최소 쿼리 실행 폴링 간격

쿼리 실행 상태에 대해 Athena를 폴링하기 전에 기다릴 최솟값(밀리초)입니다.

파라미터 이름	별칭	파라미터 유형	기본값
MinQueryExecutionPollingIntervalMillis	MinQueryExecutionPollingInterval(사용되지 않음)	선택 사항	100

최대 쿼리 실행 폴링 간격

쿼리 실행 상태에 대해 Athena를 폴링하기 전에 기다릴 최댓값(밀리초)입니다.

파라미터 이름	별칭	파라미터 유형	기본값
MaxQueryExecutionPollingIntervalMillis	MaxQueryExecutionPollingInterval(사용되지 않음)	선택 사항	5000

쿼리 실행 폴링 간격 승수

폴링 기간을 늘리는 요인입니다. 기본적으로 폴링은 `MinQueryExecutionPollingIntervalMillis` 값으로 시작하고 `MaxQueryExecutionPollingIntervalMillis` 값에 도달할 때까지 각 폴링의 두 배가 됩니다.

파라미터 이름	별칭	파라미터 유형	기본값
QueryExecutionPollingIntervalMultiplier	없음	선택 사항	2

엔드포인트 재정의의 파라미터

Athena 엔드포인트 재정의

드라이버가 Athena에 대한 API 호출을 수행하는 데 사용할 엔드포인트입니다.

다음 사항에 주의하세요.

- 제공된 URL에 `https://` 또는 `http://` 프로토콜이 지정되지 않은 경우 드라이버는 `https://` 접두사를 삽입합니다.
- 이 파라미터가 지정되지 않으면 드라이버는 기본 엔드포인트를 사용합니다.

파라미터 이름	별칭	파라미터 유형	기본값
AthenaEndpoint	EndpointOverride(사용되지 않음)	선택 사항	없음

Athena 스트리밍 서비스 엔드포인트 재정의

드라이버가 Athena 스트리밍 서비스를 사용할 때 쿼리 결과를 다운로드하는 데 사용할 엔드포인트입니다. Athena 스트리밍 서비스는 포트 444에서 사용할 수 있습니다.

다음 사항에 주의하세요.

- 제공된 URL에 `https://` 또는 `http://` 프로토콜이 지정되지 않은 경우 드라이버는 `https://` 접두사를 삽입합니다.
- 제공된 URL에 포트가 지정되지 않은 경우 드라이버는 스트리밍 서비스 포트 444를 삽입합니다.

- `AthenaStreamingEndpoint` 파라미터가 지정되지 않은 경우 드라이버는 `AthenaEndpoint` 재정의를 사용합니다. `AthenaStreamingEndpoint`와 `AthenaEndpoint` 재정의가 모두 지정되지 않은 경우 드라이버는 기본 스트리밍 엔드포인트를 사용합니다.

파라미터 이름	별칭	파라미터 유형	기본값
AthenaStreamingEndpoint	StreamingEndpointOverride(사용되지 않음)	선택 사항	없음

LakeFormation 엔드포인트 재정의

AWS Lake Formation AssumeDecoratedRoleWithSAML API를 사용하여 임시 보안 인증을 검색할 때 드라이버가 Lake Formation 서비스에 사용할 엔드포인트입니다. 이 파라미터가 지정되지 않으면 드라이버는 기본 Lake Formation 엔드포인트를 사용합니다.

다음 사항에 주의하세요.

- 제공된 URL에 `https://` 또는 `http://` 프로토콜이 지정되지 않은 경우 드라이버는 `https://` 접두사를 삽입합니다.

파라미터 이름	별칭	파라미터 유형	기본값
LakeFormationEndpoint	LfEndpointOverride(사용되지 않음)	선택 사항	없음

S3 엔드포인트 재정의

드라이버가 Amazon S3 페처를 사용할 때 쿼리 결과를 다운로드하는 데 사용할 엔드포인트입니다. 이 파라미터가 지정되지 않으면 드라이버는 기본 Amazon S3 엔드포인트를 사용합니다.

다음 사항에 주의하세요.

- 제공된 URL에 `https://` 또는 `http://` 프로토콜이 지정되지 않은 경우 드라이버는 `https://` 접두사를 삽입합니다.

파라미터 이름	별칭	파라미터 유형	기본값
S3Endpoint	None	선택 사항	없음

STS 엔드포인트 재정의

AWS STS AssumeRoleWithSAML API를 사용하여 임시 보안 인증을 검색할 때 드라이버가 AWS STS 서비스에 사용할 엔드포인트입니다. 이 파라미터가 지정되지 않으면 드라이버는 기본 AWS STS 엔드포인트를 사용합니다.

다음 사항에 주의하세요.

- 제공된 URL에 `https://` 또는 `http://` 프로토콜이 지정되지 않은 경우 드라이버는 `https://` 접두사를 삽입합니다.

파라미터 이름	별칭	파라미터 유형	기본값
StsEndpoint	StsEndpointOverrid e(사용되지 않음)	선택 사항	없음

프록시 구성 파라미터

프록시 호스트

프록시 호스트의 URL입니다. Athena 요청이 프록시를 통과해야 하는 경우 이 파라미터를 사용합니다.

> **Note**
>
> `ProxyHost`의 URL 시작 부분에 프로토콜 `https://` 또는 `http://`를 포함해야 합니다.

파라미터 이름	별칭	파라미터 유형	기본값
ProxyHost	없음	선택 사항	없음

프록시 포트

프록시 호스트에서 사용할 포트입니다. Athena 요청이 프록시를 통과해야 하는 경우 이 파라미터를 사용합니다.

파라미터 이름	별칭	파라미터 유형	기본값
ProxyPort	없음	선택 사항	없음

프록시 사용자 이름

프록시 서버에서 인증할 사용자 이름입니다. Athena 요청이 프록시를 통과해야 하는 경우 이 파라미터를 사용합니다.

파라미터 이름	별칭	파라미터 유형	기본값
ProxyUsername	ProxyUID(사용되지 않음)	선택 사항	없음

프록시 비밀번호

프록시 서버에서 인증할 암호입니다. Athena 요청이 프록시를 통과해야 하는 경우 이 파라미터를 사용합니다.

파라미터 이름	별칭	파라미터 유형	기본값
ProxyPassword	ProxyPWD(사용되지 않음)	선택 사항	없음

프록시 제외 호스트

프록시가 활성화된 경우(즉, `ProxyHost` 및 `ProxyPort` 연결 파라미터가 설정된 경우) 드라이버가 프록시를 사용하지 않고 연결하는 호스트 이름의 세트입니다. 호스트는 파이프(|) 문자로 구분되어야 합니다(예: `host1.com|host2.com`).

파라미터 이름	별칭	파라미터 유형	기본값
ProxyExemptHosts	NonProxyHosts	선택 사항	없음

ID 제공업체에 프록시 활성화

드라이버가 ID 제공업체에 연결할 때 프록시를 사용해야 하는지 여부를 지정합니다.

파라미터 이름	별칭	파라미터 유형	기본값
ProxyEnabledForIdP	UseProxyForIdP	선택 사항	FALSE

로깅 파라미터

이 섹션에서는 로깅과 관련된 파라미터를 설명합니다.

로그 수준

드라이버 로깅 수준을 지정합니다. LogPath 파라미터도 설정되지 않으면 아무 것도 로깅되지 않습니다.

> **Note**
>
> 특별한 요구 사항이 없으면 LogPath 파라미터만 설정하는 것이 좋습니다. LogPath 파라미터만 설정하면 로깅이 활성화되고 기본 TRACE 로그 수준이 사용됩니다. TRACE 로그 수준은 가장 자세한 로깅을 제공합니다.

파라미터 이름	별칭	파라미터 유형	기본값	가능한 값
LogLevel	없음	선택 사항	TRACE	OFF, ERROR, WARN, INFO, DEBUG, TRACE

로그 경로

드라이버 로그가 저장될 드라이버를 실행하는 컴퓨터의 디렉터리 경로입니다. 지정된 디렉터리 내에 고유한 이름을 가진 로그 파일이 생성됩니다. 설정하면 드라이버 로깅이 활성화됩니다.

파라미터 이름	별칭	파라미터 유형	기본값
LogPath	없음	선택 사항	없음

애플리케이션 이름

드라이버를 사용하는 애플리케이션의 이름입니다. 이 파라미터의 값이 지정되면 드라이버가 Athena에 대해 수행하는 API 호출의 사용자 에이전트 문자열에 값이 포함됩니다.

> **Note**
>
> `DataSource` 객체에서 `setApplicationName`을 직접적으로 호출하여 애플리케이션 이름을 설정할 수도 있습니다.

파라미터 이름	별칭	파라미터 유형	기본값
ApplicationName	없음	선택 사항	없음

연결 테스트

TRUE로 설정하면 드라이버는 연결에 대해 쿼리가 실행되지 않더라도 JDBC 연결이 생성될 때마다 연결 테스트를 수행합니다.

파라미터 이름	별칭	파라미터 유형	기본값
ConnectionTest	없음	선택 사항	TRUE

> **Note**
>
> 연결 테스트는 연결이 제대로 구성되었는지 확인하기 위해 Athena에 `SELECT 1` 쿼리를 제출합니다. 즉, Amazon S3에 두 개의 파일(결과 세트와 메타데이터)이 저장되며 Amazon Athena 요금 정책에 따라 추가 요금이 부과될 수 있습니다.

재시도 횟수

드라이버가 Athena에 재시도 가능한 요청을 다시 보내야 하는 최대 횟수입니다.

파라미터 이름	별칭	파라미터 유형	기본값
NumRetries	MaxErrorRetry(사용되지 않음)	선택 사항	없음

인증 연결 파라미터

Athena JDBC 3.x 드라이버는 여러 인증 방법을 지원합니다. 필요한 연결 파라미터는 사용하는 인증 방법에 따라 다릅니다.

주제

- IAM 보안 인증
- 기본 보안 인증
- AWS 구성 프로파일 보안 인증
- 인스턴스 프로파일 보안 인증
- 사용자 지정 보안 인증
- JWT 보안 인증
- Azure AD 보안 인증
- Okta 보안 인증
- Ping 보안 인증
- AD FS 자격 증명
- 브라우저 Azure AD 보안 인증
- 브라우저 SAML 보안 인증

IAM 보안 인증

다음 연결 파라미터를 설정하여 JDBC 드라이버와 함께 IAM 보안 인증을 사용하여 Amazon Athena에 연결할 수 있습니다.

User

사용자의 AWS 액세스 키 ID입니다. 액세스 키에 대한 자세한 내용은 IAM 사용 설명서의 AWS 보안 자격 증명을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
User	AccessKeyId	필수	없음

암호

AWS 보안 암호 키 ID입니다. 액세스 키에 대한 자세한 내용은 IAM 사용 설명서의 AWS 보안 자격 증명을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
암호	SecretAccessKey	선택 사항	없음

세션 토큰

임시 AWS 보안 인증을 사용하는 경우 세션 토큰을 지정해야 합니다. 임시 보안 인증에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 임시 보안 자격 증명을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
SessionToken	없음	선택 사항	없음

기본 보안 인증

다음 연결 파라미터를 설정하여 클라이언트 시스템에서 구성한 기본 보안 인증으로 Amazon Athena에 연결할 수 있습니다. 기본 보안 인증 사용에 대한 자세한 내용은 AWS SDK for Java 개발자 안내서의 Using the Default Credential Provider Chain을 참조하세요.

보안 인증 제공업체

AWS에 대한 요청을 인증하는 데 사용되는 보안 인증 제공업체입니다. 이 파라미터의 값을 `DefaultChain`으로 설정합니다.

파라미터 이름	별칭	파라미터 유형	기본값	사용할 값
CredentialsProvider	AWSCredentialsProviderClass(사용되지 않음)	필수	없음	`DefaultChain`

AWS 구성 프로파일 보안 인증

다음 연결 파라미터를 설정하여 AWS 구성 프로파일에 저장된 보안 인증을 사용할 수 있습니다. AWS 구성 프로파일은 일반적으로 `~/.aws` 디렉터리의 파일에 저장됩니다. AWS 구성 프로파일에 대한 자세한 내용은 AWS SDK for Java 개발자 안내서의 Use profiles를 참조하세요.

보안 인증 제공업체

AWS에 대한 요청을 인증하는 데 사용되는 보안 인증 제공업체입니다. 이 파라미터의 값을 `ProfileCredentials`으로 설정합니다.

파라미터 이름	별칭	파라미터 유형	기본값	사용할 값
CredentialsProvider	AWSCredentialsProviderClass(사용되지 않음)	필수	없음	`ProfileCredentials`

프로필 이름

Athena에 대한 요청을 인증하는 데 보안 인증을 사용해야 하는 AWS 구성 프로파일의 이름입니다.

파라미터 이름	별칭	파라미터 유형	기본값
ProfileName	없음	필수	없음

> **Note**
>
> 프로파일 이름을 `CredentialsProviderArguments` 파라미터의 값으로 지정할 수도 있지만 이렇게 사용하는 것은 권장되지 않습니다.

인스턴스 프로파일 보안 인증

이 인증 유형은 Amazon EC2 인스턴스에서 사용됩니다. 인스턴스 프로파일은 Amazon EC2 인스턴스에 연결된 프로파일입니다. 인스턴스 프로파일 보안 인증 제공업체를 사용하면 AWS 보안 인증 관리가 Amazon EC2 인스턴스 메타데이터 서비스에 위임됩니다. 따라서 개발자는 Amazon EC2 인스턴스에 보안 인증을 영구적으로 저장하거나 임시 보안 인증의 교체 또는 관리에 대해 걱정할 필요가 없습니다.

보안 인증 제공업체

AWS에 대한 요청을 인증하는 데 사용되는 보안 인증 제공업체입니다. 이 파라미터의 값을 `InstanceProfile`으로 설정합니다.

파라미터 이름	별칭	파라미터 유형	기본값	사용할 값
CredentialsProvider	AWSCredentialsProviderClass(사용되지 않음)	필수	없음	`InstanceProfile`

사용자 지정 보안 인증

이 인증 유형을 사용하면 AwsCredentialsProvider 인터페이스를 구현하는 Java 클래스로 자체 보안 인증을 제공할 수 있습니다.

보안 인증 제공업체

AWS에 대한 요청을 인증하는 데 사용되는 보안 인증 제공업체입니다. 이 파라미터의 값을 AWSCredentialsProvider 인터페이스를 구현하는 사용자 지정 클래스의 정규화된 클래스 이름으로 설정합니다. 런타임 시 해당 클래스가 JDBC 드라이버를 사용하는 애플리케이션의 Java 클래스 경로에 있어야 합니다.

파라미터 이름	별칭	파라미터 유형	기본값	사용할 값
CredentialsProvider	AWSCredentialsProviderClass(사용되지 않음)	필수	없음	`AwsCredentialsProvider` 사용자 지정 구현의 정규화된 클래스 이름

보안 인증 제공업체 인수

사용자 지정 보안 인증 제공업체 생성자의 쉼표로 구분된 문자열 인수 목록입니다.

파라미터 이름	별칭	파라미터 유형	기본값
CredentialsProviderArguments	AwsCredentialsProviderArguments(사용되지 않음)	선택 사항	없음

JWT 보안 인증

이 인증 유형을 사용하면 외부 ID 제공업체로부터 받은 JSON 웹 토큰(JWT)을 연결 파라미터로 사용하여 Athena로 인증할 수 있습니다. 외부 보안 인증 제공업체가 이미 AWS와 연동되어야 합니다.

보안 인증 제공업체

AWS에 대한 요청을 인증하는 데 사용되는 보안 인증 제공업체입니다. 이 파라미터의 값을 `JWT`으로 설정합니다.

파라미터 이름	별칭	파라미터 유형	기본값	사용할 값
CredentialsProvider	AWSCredentialsProviderClass(사용되지 않음)	필수	없음	`JWT`

JWT 웹 자격 증명 토큰

외부 페더레이션형 ID 제공업체로부터 얻은 JWT 토큰입니다. 이 토큰이 Athena로 인증하는 데 사용됩니다.

파라미터 이름	별칭	파라미터 유형	기본값
JwtWebIdentityToken	web_identity_token(사용되지 않음)	필수	없음

JWT 역할 ARN

수임할 역할의 Amazon 리소스 이름(ARN)입니다. 역할 수임에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
JwtRoleArn	role_arn(사용되지 않음)	필수	없음

JWT 역할 세션 이름

인증에 JWT 보안 인증을 사용할 때의 세션 이름입니다. 아무 이름이나 선택해서 사용할 수 있습니다.

파라미터 이름	별칭	파라미터 유형	기본값
JwtRoleSessionName	role_session_name(사용되지 않음)	필수	없음

Azure AD 보안 인증

Azure AD ID 제공업체를 사용하여 Athena에 대한 인증을 활성화하는 SAML 기반 인증 메커니즘입니다. 이 방법은 Athena와 Azure AD 사이에 페더레이션이 이미 설정되어 있다고 가정합니다.

> **Note**
>
> 이 섹션의 일부 파라미터 이름에는 별칭이 있습니다. 별칭은 파라미터 이름과 기능이 동일하며 JDBC 2.x 드라이버에서 이전 버전과의 호환성을 위해 제공됩니다. 더 명확하고 일관된 명명 규칙을 따르도록 파라미터 이름이 개선되었으므로 더 이상 사용되지 않는 별칭 대신 해당 이름을 사용하는 것이 좋습니다.

보안 인증 제공업체

AWS에 대한 요청을 인증하는 데 사용되는 보안 인증 제공업체입니다. 이 파라미터의 값을 `AzureAD`으로 설정합니다.

파라미터 이름	별칭	파라미터 유형	기본값	사용할 값
CredentialsProvider	AWSCredentialsProviderClass(사용되지 않음)	필수	없음	`AzureAD`

User

Azure AD로 인증에 사용할 Azure AD 사용자의 이메일 주소입니다.

파라미터 이름	별칭	파라미터 유형	기본값
User	UID(사용되지 않음)	필수	없음

암호

Azure AD 사용자의 암호입니다.

파라미터 이름	별칭	파라미터 유형	기본값
암호	PWD(사용되지 않음)	필수	없음

Azure AD 테넌트 ID

Azure AD 애플리케이션의 테넌트 ID입니다.

파라미터 이름	별칭	파라미터 유형	기본값
AzureAdTenantId	tenant_id(사용되지 않음)	필수	없음

Azure AD 클라이언트 ID

Azure AD 애플리케이션의 클라이언트 ID입니다.

파라미터 이름	별칭	파라미터 유형	기본값
AzureAdClientId	client_id(사용되지 않음)	필수	없음

Azure AD 클라이언트 보안 암호

Azure AD 애플리케이션의 클라이언트 보안 암호입니다.

파라미터 이름	별칭	파라미터 유형	기본값
AzureAdClientSecret	client_secret(사용되지 않음)	필수	없음

기본 역할

수임할 역할의 Amazon 리소스 이름(ARN)입니다. ARN 역할에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
PreferredRole	preferred_role(사용되지 않음)	선택 사항	없음

역할 세션 기간

역할 세션 지속 시간(초)입니다. 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
RoleSessionDuration	Duration(사용되지 않음)	선택 사항	3600

Lake Formation 활성화

AssumeRoleWithSAML AWS STS API 작업 대신 AssumeDecoratedRoleWithSAML Lake Formation API 작업을 사용하여 임시 IAM 보안 인증 정보를 검색할지 여부를 지정합니다.

파라미터 이름	별칭	파라미터 유형	기본값
LakeFormationEnabled	없음	선택 사항	FALSE

Okta 보안 인증

Okta ID 제공업체를 사용하여 Athena에 대한 인증을 활성화하는 SAML 기반 인증 메커니즘입니다. 이 방법은 Athena와 Okta 사이에 페더레이션이 이미 설정되어 있다고 가정합니다.

보안 인증 제공업체

AWS에 대한 요청을 인증하는 데 사용되는 보안 인증 제공업체입니다. 이 파라미터의 값을 `Okta`으로 설정합니다.

파라미터 이름	별칭	파라미터 유형	기본값	사용할 값
CredentialsProvider	AWSCredentialsProviderClass(사용되지 않음)	필수	없음	`Okta`

User

Okta로 인증에 사용할 Okta 사용자의 이메일 주소입니다.

파라미터 이름	별칭	파라미터 유형	기본값
User	UID(사용되지 않음)	필수	없음

암호

Okta 사용자의 암호입니다.

파라미터 이름	별칭	파라미터 유형	기본값
암호	PWD(사용되지 않음)	필수	없음

Okta 호스트 이름

Okta 조직의 URL입니다. Okta 애플리케이션의 Embed Link URL에서 `idp_host` 파라미터를 추출할 수 있습니다. 단계는 Okta에서 ODBC 구성 정보 검색를 참조하세요. `https://` 다음부터 `okta.com`까지(포함) 첫 번째 세그먼트가 IdP 호스트(예: `https://trial-1234567.okta.com`으로 시작하는 URL의 경우 `trial-1234567.okta.com`)입니다.

파라미터 이름	별칭	파라미터 유형	기본값
Okta 호스트 이름	IdP_Host(사용되지 않음)	필수	없음

Okta 애플리케이션 ID

애플리케이션의 ID입니다(두 부분으로 구성됨). Okta 애플리케이션의 Embed Link URL에서 애플리케이션 ID를 추출할 수 있습니다. 단계는 Okta에서 ODBC 구성 정보 검색를 참조하세요. 애플리케이션 ID는 URL의 마지막 두 세그먼트(가운데 슬래시 포함)입니다. 세그먼트는 숫자, 대문자, 소문자가 혼합된 두 개의 20자 문자열(예: `Abc1de2fghi3J45kL678/abc1defghij2klmNo3p4`)입니다.

파라미터 이름	별칭	파라미터 유형	기본값
OktaAppId	App_ID(사용되지 않음)	필수	없음

Okta 애플리케이션 이름

Okta 애플리케이션의 이름입니다.

파라미터 이름	별칭	파라미터 유형	기본값
OktaAppName	App_Name(사용되지 않음)	필수	없음

Okata MFA 유형

다중 인증(MFA)을 요구하도록 Okta를 설정한 경우 사용하려는 두 번째 요소에 따라 Okta MFA 유형 및 추가 파라미터를 지정해야 합니다.

Okta MFA 유형은 Okta로 인증하는 데 사용할 두 번째 인증 요소 유형(암호 다음)입니다. 지원되는 두 번째 요소에는 Okta Verify 앱을 통해 전송되는 푸시 알림과 Okta Verify, Google Authenticator에서 생성되거나 SMS를 통해 전송되는 임시 일회용 암호(TOTP)가 포함됩니다. 개별 조직의 보안 정책에 따라 사용자 로그인에 MFA가 필요한지 여부가 결정됩니다.

파라미터 이름	별칭	파라미터 유형	기본값	가능한 값
OktaMfaType	okta_mfa_type(사용되지 않음)	MFA를 요구하도록 Okta가 설정된 경우 필수	없음	`oktaverifywithpush` , `oktaverifywithtotp` , `googleauthenticator` , `smsauthentication`

Okta 전화번호

`smsauthentication` MFA 유형이 선택된 경우 Okta가 SMS를 사용하여 임시 일회용 암호를 보낼 전화번호입니다. 전화번호는 미국 또는 캐나다 전화번호여야 합니다.

파라미터 이름	별칭	파라미터 유형	기본값
OktaPhoneNumber	okta_phone_number(사용되지 않음)	OktaMfaType 이 `smsauthentication` 인 경우 필수	없음

Okta MFA 대기 시간

드라이버가 시간 초과 예외를 발생시키기 전에 사용자가 Okta의 푸시 알림을 확인할 때까지 기다리는 기간(초)입니다.

파라미터 이름	별칭	파라미터 유형	기본값
OktaMfaWaitTime	okta_mfa_wait_time (사용되지 않음)	선택 사항	60

기본 역할

수임할 역할의 Amazon 리소스 이름(ARN)입니다. ARN 역할에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
PreferredRole	preferred_role(사용되지 않음)	선택 사항	없음

역할 세션 기간

역할 세션 지속 시간(초)입니다. 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
RoleSessionDuration	Duration(사용되지 않음)	선택 사항	3600

Lake Formation 활성화

AssumeRoleWithSAML AWS STS API 작업 대신 AssumeDecoratedRoleWithSAML Lake Formation API 작업을 사용하여 임시 IAM 보안 인증 정보를 검색할지 여부를 지정합니다.

파라미터 이름	별칭	파라미터 유형	기본값
LakeFormationEnabled	없음	선택 사항	FALSE

Ping 보안 인증

Ping Federate ID 제공업체를 사용하여 Athena에 대한 인증을 활성화하는 SAML 기반 인증 메커니즘입니다. 이 방법은 Athena와 Ping Federate 사이에 페더레이션이 이미 설정되어 있다고 가정합니다.

보안 인증 제공업체

AWS에 대한 요청을 인증하는 데 사용되는 보안 인증 제공업체입니다. 이 파라미터의 값을 `Ping`으로 설정합니다.

파라미터 이름	별칭	파라미터 유형	기본값	사용할 값
CredentialsProvider	AWSCredentialsProviderClass(사용되지 않음)	필수	없음	`Ping`

User

Ping Federate 인증에 사용할 Ping Federate 사용자의 이메일 주소입니다.

파라미터 이름	별칭	파라미터 유형	기본값
User	UID(사용되지 않음)	필수	없음

암호

Ping Federate 사용자의 암호입니다.

파라미터 이름	별칭	파라미터 유형	기본값
암호	PWD(사용되지 않음)	필수	없음

PingHostName

Ping 서버의 주소입니다. 주소를 찾으려면 다음 URL을 방문하여 SSO 애플리케이션 엔드포인트 필드를 확인합니다.

```
https://your-pf-host-#:9999/pingfederate/your-pf-app#/spConnections
```

파라미터 이름	별칭	파라미터 유형	기본값
PingHostName	IdP_Host(사용되지 않음)	필수	없음

PingPortNumber

IdP 호스트에 연결하는 데 사용할 포트 번호입니다.

파라미터 이름	별칭	파라미터 유형	기본값
PingPortNumber	IdP_Port(사용되지 않음)	필수	없음

PingPartnerSpId

서비스 제공업체 주소입니다. 서비스 제공업체 주소를 찾으려면 다음 URL을 방문하여 SSO 애플리케이션 엔드포인트 필드를 확인합니다.

```
https://your-pf-host-#:9999/pingfederate/your-pf-app#/spConnections
```

파라미터 이름	별칭	파라미터 유형	기본값
PingPartnerSpId	Partner_SPID(사용되지 않음)	필수	없음

기본 역할

수임할 역할의 Amazon 리소스 이름(ARN)입니다. ARN 역할에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
PreferredRole	preferred_role(사용되지 않음)	선택 사항	없음

역할 세션 기간

역할 세션 지속 시간(초)입니다. 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
RoleSessionDuration	Duration(사용되지 않음)	선택 사항	3600

Lake Formation 활성화

AssumeRoleWithSAML AWS STS API 작업 대신 AssumeDecoratedRoleWithSAML Lake Formation API 작업을 사용하여 임시 IAM 보안 인증 정보를 검색할지 여부를 지정합니다.

파라미터 이름	별칭	파라미터 유형	기본값
LakeFormationEnabled	없음	선택 사항	FALSE

AD FS 자격 증명

Microsoft AD FS(Active Directory Federation Services)를 사용하여 Athena에 대한 인증을 활성화하는 SAML 기반 인증 메커니즘입니다. 이 방법은 사용자가 이미 Athena와 AD FS 사이에 페더레이션을 설정했다고 가정합니다.

보안 인증 제공업체

AWS에 대한 요청을 인증하는 데 사용되는 보안 인증 제공업체입니다. 이 파라미터의 값을 ADFS으로 설정합니다.

파라미터 이름	별칭	파라미터 유형	기본값	사용할 값
CredentialsProvider	AWSCredentialsProviderClass(사용되지 않음)	필수	없음	ADFS

User

AD FS 인증에 사용할 AD FS 사용자의 이메일 주소입니다.

파라미터 이름	별칭	파라미터 유형	기본값
User	UID(사용되지 않음)	양식 기반 인증에 필요합니다. Windows 통합 인증의 경우 선택 사항입니다.	없음

암호

AD FS 사용자의 암호입니다.

파라미터 이름	별칭	파라미터 유형	기본값
암호	PWD(사용되지 않음)	양식 기반 인증에 필요합니다. Windows 통합 인증의 경우 선택 사항입니다.	없음

ADFS 호스트 이름

AD FS 서버의 주소입니다.

파라미터 이름	별칭	파라미터 유형	기본값
AdfsHostName	IdP_Host(사용되지 않음)	필수	없음

ADFS 포트 번호

AD FS 서버에 연결하는 데 사용할 포트 번호입니다.

파라미터 이름	별칭	파라미터 유형	기본값
AdfsPortNumber	IdP_Port(사용되지 않음)	필수	없음

ADFS 신뢰 당사자

신뢰할 수 있는 신뢰 당사자입니다. 이 파라미터를 사용하여 AD FS 신뢰 당사자 엔드포인트 URL을 재정의합니다.

파라미터 이름	별칭	파라미터 유형	기본값
AdfsRelyingParty	LoginToRP(사용되지 않음)	선택 사항	`urn:amazon:webservices`

ADFS WIA 활성화

불. 이 파라미터를 사용하면 AD FS를 통한 WIA(Windows 통합 인증)를 활성화합니다.

파라미터 이름	별칭	파라미터 유형	기본값
AdfsWiaEnabled	none	선택 사항	FALSE

기본 역할

수임할 역할의 Amazon 리소스 이름(ARN)입니다. ARN 역할에 대한 자세한 내용은 AWS Security Token Service API 참조의 `AssumeRole`을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
PreferredRole	preferred_role(사용되지 않음)	선택 사항	없음

역할 세션 기간

역할 세션 지속 시간(초)입니다. 자세한 내용을 알아보려면 AWS Security Token Service API 참조의 `AssumeRole` 섹션을 참조하십시오.

파라미터 이름	별칭	파라미터 유형	기본값
RoleSessionDuration	Duration(사용되지 않음)	선택 사항	3600

Lake Formation 활성화

`AssumeRoleWithSAML` AWS STS API 작업 대신 `AssumeDecoratedRoleWithSAML` Lake Formation API 작업을 사용하여 임시 IAM 보안 인증 정보를 검색할지 여부를 지정합니다.

파라미터 이름	별칭	파라미터 유형	기본값
LakeFormationEnabled	none	선택 사항	FALSE

브라우저 Azure AD 보안 인증

브라우저 Azure AD는 Azure AD ID 제공업체에서 작동하고 다중 인증을 지원하는 SAML 기반 인증 메커니즘입니다. 표준 Azure AD 인증 메커니즘과 달리 이 메커니즘은 연결 파라미터에서 사용자 이름, 암호 또는 클라이언트 보안 암호를 요구하지 않습니다. 표준 Azure AD 인증 메커니즘과 마찬가지로 브라우저 Azure AD도 사용자가 Athena와 Azure AD 간의 페더레이션을 이미 설정했다고 가정합니다.

보안 인증 제공업체

AWS에 대한 요청을 인증하는 데 사용되는 보안 인증 제공업체입니다. 이 파라미터의 값을 `BrowserAzureAD`으로 설정합니다.

파라미터 이름	별칭	파라미터 유형	기본값	사용할 값
CredentialsProvider	AWSCredentialsProviderClass(사용되지 않음)	필수	없음	`BrowserAzureAD`

Azure AD 테넌트 ID

Azure AD 애플리케이션의 테넌트 ID

파라미터 이름	별칭	파라미터 유형	기본값
AzureAdTenantId	tenant_id(사용되지 않음)	필수	없음

Azure AD 클라이언트 ID

Azure AD 애플리케이션의 클라이언트 ID

파라미터 이름	별칭	파라미터 유형	기본값
AzureAdClientId	client_id(사용되지 않음)	필수	없음

ID 제공업체 응답 제한 시간

드라이버가 Azure AD로부터 SAML 응답 대기를 중지하기까지의 시간(초)입니다.

파라미터 이름	별칭	파라미터 유형	기본값
IdpResponseTimeout	idp_response_timeout(사용되지 않음)	선택 사항	120

기본 역할

수임할 역할의 Amazon 리소스 이름(ARN)입니다. ARN 역할에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
PreferredRole	preferred_role(사용되지 않음)	선택 사항	없음

역할 세션 기간

역할 세션 지속 시간(초)입니다. 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
RoleSessionDuration	Duration(사용되지 않음)	선택 사항	3600

Lake Formation 활성화

AssumeRoleWithSAML AWS STS API 작업 대신 AssumeDecoratedRoleWithSAML Lake Formation API 작업을 사용하여 임시 IAM 보안 인증 정보를 검색할지 여부를 지정합니다.

파라미터 이름	별칭	파라미터 유형	기본값
LakeFormationEnabled	없음	선택 사항	FALSE

브라우저 SAML 보안 인증

브라우저 SAML은 SAML 기반 ID 제공업체에서 작동하고 다중 인증을 지원할 수 있는 일반 인증 플러그인입니다.

보안 인증 제공업체

AWS에 대한 요청을 인증하는 데 사용되는 보안 인증 제공업체입니다. 이 파라미터의 값을 `BrowserSaml`으로 설정합니다.

파라미터 이름	별칭	파라미터 유형	기본값	사용할 값
CredentialsProvider	AWSCredentialsProviderClass(사용되지 않음)	필수	없음	`BrowserSaml`

Single Sign-On 로그인 URL

SAML 기반 ID 제공업체의 애플리케이션에 대한 Single Sign-On URL입니다.

파라미터 이름	별칭	파라미터 유형	기본값
SsoLoginUrl	login_url(사용되지 않음)	필수	없음

리스너 포트

SAML 응답을 수신하는 데 사용되는 포트 번호입니다. 이 값은 SAML 기반 ID 제공업체를 구성한 URL과 일치해야 합니다(예: `http://localhost:7890/athena`).

파라미터 이름	별칭	파라미터 유형	기본값
ListenPort	listen_port(사용되지 않음)	선택 사항	7890

ID 제공업체 응답 제한 시간

드라이버가 Azure AD로부터 SAML 응답 대기를 중지하기까지의 시간(초)입니다.

파라미터 이름	별칭	파라미터 유형	기본값
IdpResponseTimeout	idp_response_timeout(사용되지 않음)	선택 사항	120

기본 역할

수임할 역할의 Amazon 리소스 이름(ARN)입니다. ARN 역할에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
PreferredRole	preferred_role(사용되지 않음)	선택 사항	없음

역할 세션 기간

역할 세션 지속 시간(초)입니다. 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

파라미터 이름	별칭	파라미터 유형	기본값
RoleSessionDuration	Duration(사용되지 않음)	선택 사항	3600

Lake Formation 활성화

AssumeRoleWithSAML AWS STS API 작업 대신 AssumeDecoratedRoleWithSAML Lake Formation API 작업을 사용하여 임시 IAM 보안 인증 정보를 검색할지 여부를 지정합니다.

파라미터 이름	별칭	파라미터 유형	기본값
LakeFormationEnabled	없음	선택 사항	FALSE

기타 JDBC 3.x 구성

다음 섹션에서는 JDBC 3.x 드라이버의 몇 가지 추가 구성 설정에 대해 설명합니다.

네트워크 시간 초과

드라이버가 Athena에 대한 API 호출 시 응답을 기다리는 시간(밀리초)입니다. 이 시간이 지나면 시간 초과 예외가 발생합니다.

네트워크 시간 초과를 연결 파라미터로 설정할 수 없습니다. 설정하려면 JDBC `Connection` 객체에서 `setNetworkTimeout` 메서드를 직접적으로 호출합니다. 이 값은 JDBC 연결의 수명 주기 동안 변경될 수 있습니다. 이 파라미터의 기본값은 `infinity`입니다.

다음 예제에서는 네트워크 제한 시간을 5,000밀리초로 설정합니다.

```
...
AthenaDriver driver = new AthenaDriver();
Connection connection = driver.connect(url, connectionParameters);
connection.setNetworkTimeout(null, 5000);
...
```

쿼리 시간 초과

쿼리가 제출된 후 드라이버가 Athena에서 쿼리가 완료될 때까지 기다리는 시간(초)입니다. 이 시간이 지나면 드라이버가 제출된 쿼리 취소를 시도하고 시간 초과 예외가 발생합니다.

쿼리 시간 초과를 연결 파라미터로 설정할 수 없습니다. 설정하려면 JDBC `Statement` 객체에서 `setQueryTimeout` 메서드를 직접적으로 호출합니다. 이 값은 JDBC 문의 수명 주기 동안 변경될 수 있습니다. 이 파라미터의 기본값은 `0`입니다. 값이 `0`이면 쿼리가 완료될 때까지 실행될 수 있습니다 (Service Quotas 적용).

다음 예제에서는 쿼리 제한 시간을 5초로 설정합니다.

```
...
AthenaDriver driver = new AthenaDriver();
Connection connection = driver.connect(url, connectionParameters);
Statement statement = connection.createStatement();
statement.setQueryTimeout(5);
...
```

Amazon Athena JDBC 3.x 릴리스 정보

해당 릴리스 정보는 Amazon Athena JDBC 3.x 드라이버의 세부 개선 사항 및 수정 사항을 제공합니다.

3.2.0

2024년 4월 26일 릴리스

개선 사항

- 카탈로그 작업 성능-와일드카드 문자를 사용하지 않는 카탈로그 작업의 성능이 개선되었습니다.
- 최소 폴링 간격 변경 - 드라이버가 Athena에 보내는 API 직접 호출 횟수를 줄이도록 최소 폴링 간격 기본값이 수정되었습니다. 쿼리 완성은 최대한 빠르게 감지됩니다.
- BI 도구 발견 용이성-비즈니스 인텔리전스 도구가 드라이버를 더 쉽게 발견할 수 있게 되었습니다.
- 데이터 유형 매핑-Athena `binary`, `array` 및 `struct` DDL 데이터 유형에 대한 데이터 유형 매핑이 개선되었습니다.
- AWS SDK 버전-드라이버에 사용된 AWS SDK 버전이 2.25.34로 업데이트되었습니다.

수정 사항

- 페더레이션된 카탈로그 테이블 목록-페더레이션된 카탈로그가 빈 테이블 목록을 반환하는 문제를 수정했습니다.
- getSchemas-JDBC DatabaseMetaData#getSchemas 메서드가 모든 카탈로그가 아닌 기본 카탈로그에서만 데이터베이스를 가져오던 문제를 수정했습니다.
- getColumns-null 카탈로그 이름을 사용하여 JDBC DatabaseMetaData#getColumns 메서드를 호출했을 때 null 카탈로그가 반환되는 문제를 수정했습니다.

3.1.0

2024년 2월 15일 릴리스

개선 사항

- Microsoft AD FS(Active Directory Federation Services) Windows 통합 인증 및 양식 기반 인증에 대한 지원이 추가되었습니다.
- 버전 2.x와의 이전 버전과의 호환성을 위해 이제 awsathena JDBC 하위 프로토콜이 허용되지만 사용 중단 경고가 표시됩니다. athena JDBC 하위 프로토콜을 대신 사용하세요.
- AwsDataCatalog는 이제 카탈로그 매개변수의 기본값이고, default는 데이터베이스 매개변수의 기본값입니다. 이러한 변경을 통해 현재 카탈로그와 데이터베이스에 대해 null이 아닌 올바른 값이 반환됩니다.
- JDBC 사양을 준수하며 IS_AUTOINCREMENT 및 IS_GENERATEDCOLUMN이 이제는 NO 대신 빈 문자열을 반환합니다.
- 이제 Athena int 데이터 유형이 Athena integer와 동일한 JDBC 유형에 other 대신 매핑됩니다.
- Athena의 열 메타데이터에 선택적 precision 및 scale 필드가 없는 경우 드라이버는 이제 ResultSet 열의 해당 값에 대해 0을 반환합니다.
- AWS SDK 버전은 버전 2.21.39로 업데이트되었습니다.

수정 사항

- Athena의 일반 텍스트 결과의 열 수가 Athena 결과 메타데이터의 열 수와 일치하지 않는 경우 예외를 발생시키는 GetQueryResultsStream 문제를 수정했습니다.

3.0.0

2023년 11월 16일 릴리스

Athena JDBC 3.x 드라이버는 더 나은 성능과 호환성을 제공하는 차세대 드라이버입니다. JDBC 3.x 드라이버는 Amazon S3에서 직접 쿼리 결과 읽기를 지원하므로 쿼리 결과를 대량으로 사용하는 애플리케이션의 성능이 개선됩니다. 또한 새 드라이버는 타사 종속성이 적기 때문에 BI 도구 및 사용자 지정 애플리케이션과의 더욱 간편한 통합이 가능합니다.

Athena JDBC 3.x 드라이버 이전 버전

최신 버전의 JDBC 3.x 드라이버를 사용하는 것을 강력히 권장합니다. 최신 버전의 드라이버에는 최신 개선 사항 및 수정 사항이 포함되어 있습니다. 애플리케이션이 최신 버전과 호환되지 않는 경우에만 이전 버전을 사용하세요.

JDBC 드라이버 uber jar

다음 다운로드는 드라이버와 모든 종속 항목을 동일한 `.jar` 파일에 패키징합니다. 이 다운로드는 일반적으로 서드 파티 SQL 클라이언트에 사용됩니다.

- 3.1.0 uber jar
- 3.0.0 uber jar

JDBC 드라이버 lean jar

다음 다운로드는 드라이버를 위한 lean `.jar`와 드라이버의 종속 항목을 위한 별도의 `.jar` 파일이 포함된 `.zip` 파일입니다. 이 다운로드는 드라이버가 사용하는 종속 항목과 충돌하는 종속 항목이 있을 수 있는 사용자 지정 애플리케이션에 주로 사용됩니다. 이 다운로드는 lean jar에 포함할 드라이버 종속 항목과 사용자 지정 애플리케이션에 이미 하나 이상의 드라이버 종속 항목이 포함되어 있는 경우 제외할 드라이버 종속 항목을 선택하려는 경우에 유용합니다.

- 3.1.0 lean jar
- 3.0.0 lean jar

Athena JDBC 2.x 드라이버

JDBC 연결을 사용하여 Athena를 비즈니스 인텔리전스 도구 및 SQL 워크벤치와 같은 다른 애플리케이션에 연결할 수 있습니다. 이를 위해 이 페이지의 Amazon S3 링크를 사용해 Athena JDBC 2.x 드라이버를 다운로드, 설치 및 구성합니다. JDBC 연결 URL 구축에 대한 자세한 내용은 다운로드 가능한 JDBC 드라이버 설치 및 구성 가이드를 참조하세요. 권한에 대한 자세한 내용은 JDBC 및 ODBC 연결을 통한 액세스 단원을 참조하세요. JDBC 드라이버에 관한 피드백을 제출하려면 athena-feedback@amazon.com으로 이메일을 보내주세요. 버전 2.0.24부터는 두 가지 버전(AWS SDK가 포함된 버전과 그렇지 않은 버전)의 드라이버를 사용할 수 있습니다.

> **Important**
>
> JDBC 드라이버를 사용할 때는 다음 요구 사항에 유의하십시오.
>
> - 포트 444 개방 – Athena가 쿼리 결과를 스트리밍하는 데 사용하는 포트 444를 유지하고 아웃바운드 트래픽에 개방합니다. PrivateLink 엔드포인트를 사용하여 Athena에 연결하는 경우, PrivateLink 엔드포인트에 연결된 보안 그룹이 포트 444의 인바운드 트래픽에 개방되어 있는지 확인합니다. 포트 444가 차단되면 다음과 같은 오류 메시지가 표시될 수 있습니다. [Simba][AthenaJDBC](100123) 오류가 발생했습니다. 열 초기화 중 예외가 발생했습니다.

- athena:GetQueryResultsStream 정책 – JDBC 드라이버를 사용하는 IAM 보안 주체에 대한 `athena:GetQueryResultsStream` 정책 작업을 추가합니다. 이 정책 작업은 API를 통해 직접 노출되지 않습니다. 이 정책 작업은 스트리밍 결과 지원의 일부로 ODBC 및 JDBC 드라이버에만 사용됩니다. 정책 예제는 AWS 관리형 정책: AWSQuicksightAthenaAccess을 참조하세요.
- 여러 데이터 카탈로그에 JDBC 드라이버 사용 - Athena와 함께 여러 데이터 카탈로그에 JDBC 드라이버를 사용하려면(예를 들어 외부 Hive 메타스토어 또는 연합 쿼리 사용 시) JDBC 연결 문자열에 `MetadataRetrievalMethod=ProxyAPI`를 포함시킵니다.
- 4.1 드라이버 - 2023년부터 JDBC 버전 4.1에 대한 드라이버 지원이 중단됩니다. 추가 업데이트는 릴리스되지 않습니다. JDBC 4.1 드라이버를 사용 중인 경우 4.2 드라이버로 마이그레이션하는 것이 좋습니다.

JDBC 2.x 드라이버(AWS SDK 포함)

JDBC 드라이버 버전 2.1.5에서는 JDBC API 4.2 데이터 표준을 준수하며 JDK 8.0 이상이 필요합니다. 사용하는 JRE(Java Runtime Environment) 버전을 확인하는 방법에 대한 자세한 내용은 Java 설명서를 참조하세요.

다음 링크를 사용하여 JDBC 4.2 드라이버 `.jar` 파일을 다운로드합니다.

- AthenaJDBC42-2.1.5.1000.jar

다음 `.zip` 파일 다운로드에는 JDBC 4.2용 `.jar` 파일, AWS SDK, 해당 설명서, 릴리스 정보, 라이선스 및 계약이 포함되어 있습니다.

- SimbaAthenaJDBC-2.1.5.1000.zip

JDBC 2.x 드라이버(AWS SDK 미포함)

JDBC 드라이버 버전 2.1.5에서는 JDBC API 4.2 데이터 표준을 준수하며 JDK 8.0 이상이 필요합니다. 사용하는 JRE(Java Runtime Environment) 버전을 확인하는 방법에 대한 자세한 내용은 Java 설명서를 참조하세요.

다음 링크를 사용하여 JDBC 4.2 드라이버 `.jar` 파일(AWS SDK 제외)을 다운로드합니다.

- AthenaJDBC42-2.1.5.1001.jar

다음 `.zip` 파일 다운로드에는 JDBC 4.2용 `.jar` 파일, 해당 설명서, 릴리스 정보, 라이선스 및 계약이 포함되어 있습니다. AWS SDK는 포함하지 않습니다.

- SimbaAthenaJDBC-2.1.5.1001.zip

JDBC 2.x 드라이버 릴리스 정보, 라이선스 계약, 고지 사항

필요한 버전을 다운로드한 후, 출시 정보를 읽고 라이선스 계약 및 고지 사항을 확인하세요.

- 릴리스 정보
- 라이선스 계약
- 고지 사항
- 서드 파티 라이선스

JDBC 2.x 드라이버 설명서

드라이버용 다음 설명서를 다운로드합니다.

- JDBC 드라이버 설치 및 구성 가이드. 이 가이드를 이용해 드라이버를 설치하고 구성하세요.
- JDBC 드라이버 마이그레이션 가이드. 이 가이드를 이용해 이전 버전에서 현재 버전으로 마이그레이션하세요.

ODBC로 Amazon Athena에 연결

Amazon Athena는 버전 1.x 및 2.x라는 두 가지 ODBC 드라이버를 제공합니다. Athena ODBC 2.x 드라이버는 Linux, macOS ARM, macOS Intel 및 Windows 64비트 시스템을 지원하는 새로운 대안입니다. Athena 2.x 드라이버는 1.x ODBC 드라이버가 지원하는 모든 인증 플러그인을 지원하며 거의 모든 연결 파라미터가 이전 버전과 호환됩니다.

- ODBC 2.x 드라이버를 다운로드하려면 Amazon Athena ODBC 2.x 섹션을 참조하세요.
- ODBC 1.x 드라이버를 다운로드하려면 Athena ODBC 1.x 드라이버 섹션을 참조하세요.

주제

- Amazon Athena ODBC 2.x
- Athena ODBC 1.x 드라이버
- Amazon Athena Power BI 커넥터 사용

Amazon Athena ODBC 2.x

ODBC 연결을 사용하여 여러 서드 파티 SQL 클라이언트 도구 및 애플리케이션에서 Amazon Athena에 연결할 수 있습니다. 클라이언트 컴퓨터에서 ODBC 연결을 설정합니다.

고려 사항 및 제한

- Athena ODBC 1.x 드라이버에서 Athena 2.x ODBC 드라이버로 마이그레이션하는 방법에 대한 자세한 내용은 ODBC 2.x 드라이버로 마이그레이션 섹션을 참조하세요.
- `CSE_KMS` 암호화 옵션과 함께 S3 페처를 사용하는 경우 Amazon S3 클라이언트는 Amazon S3 버킷에 저장된 결과를 해독할 수 없습니다. 해결 방법으로 Athena 스트리밍 API 옵션을 사용하여 결과 세트를 가져옵니다.

ODBC 2.x 드라이버 다운로드

Amazon Athena 2.x ODBC 드라이버를 다운로드하려면 이 페이지에 있는 링크를 방문하세요.

> **Important**
>
> ODBC 2.x 드라이버를 사용할 때는 다음 요구 사항에 유의하세요.
>
> - 포트 444 개방 – Athena가 쿼리 결과를 스트리밍하는 데 사용하는 포트 444를 유지하고 아웃바운드 트래픽에 개방합니다. PrivateLink 엔드포인트를 사용하여 Athena에 연결하는 경우, PrivateLink 엔드포인트에 연결된 보안 그룹이 포트 444의 인바운드 트래픽에 개방되어 있는지 확인합니다.
> - athena:GetQueryResultsStream 정책 – ODBC 드라이버를 사용하는 IAM 보안 주체에 대한 `athena:GetQueryResultsStream` 정책 작업을 추가합니다. 이 정책 작업은 API를 통해 직접 노출되지 않습니다. 이 정책 작업은 스트리밍 결과 지원의 일부로 ODBC 및 JDBC 드라이버에만 사용됩니다. 정책 예제는 AWS 관리형 정책: AWSQuicksightAthenaAccess을 참조하세요.

Linux

드라이버 버전	다운로드 링크
Linux 64비트용 ODBC 2.0.3.0	Linux 64비트 ODBC 드라이버 2.0.3.0

macOS(ARM)

드라이버 버전	다운로드 링크
macOS 64비트용 ODBC 2.0.3.0(ARM)	macOS 64비트 ODBC 드라이버 2.0.3.0(ARM)

macOS(Intel)

드라이버 버전	다운로드 링크
macOS 64비트용 ODBC 2.0.3.0(Intel)	macOS 64비트 ODBC 드라이버 2.0.3.0(Intel)

Windows

드라이버 버전	다운로드 링크
Windows 64비트용 ODBC 2.0.3.0	Windows 64비트 ODBC 드라이버 2.0.3.0

주제

- ODBC 2.x 드라이버 시작하기
- Athena ODBC 2.x 연결 파라미터
- ODBC 2.x 드라이버로 마이그레이션
- ODBC 2.x 드라이버 문제 해결
- Amazon Athena ODBC 2.x 릴리스 정보

ODBC 2.x 드라이버 시작하기

이 섹션의 정보를 사용하여 Amazon Athena ODBC 2.x 드라이버를 시작하세요. 드라이버는 Windows, Linux 및 macOS 운영 체제에서 지원됩니다.

주제

- Windows
- Linux
- macOS

Windows

Windows 클라이언트 컴퓨터를 사용하여 Amazon Athena에 액세스하려면 Amazon Athena ODBC 드라이버가 필요합니다.

Windows 시스템 요구 사항

웹 브라우저를 사용하지 않고 Amazon Athena 데이터베이스에 직접 액세스하는 클라이언트 컴퓨터에 Amazon Athena ODBC 드라이버를 설치합니다.

사용하는 Windows 시스템에서 다음 요구 사항을 충족해야 합니다.

- 관리자 권한 보유
- 다음 운영 체제 중 하나:
  - Windows 11, 10 또는 8.1
  - Windows Server 2019, 2016 또는 2012
- 사용 가능한 디스크 공간 100MB 이상
- 64비트 Windows용 Microsoft Visual C++ for Visual Studio 재배포 가능 패키지가 설치됨

Amazon Athena ODBC 드라이버 설치

Windows용 Amazon Athena ODBC 드라이버를 다운로드하고 설치하려면

1. `AmazonAthenaODBC-2.x.x.x.msi` 설치 파일을 다운로드합니다.
2. 설치 파일을 실행하고 다음을 선택합니다.
3. 라이선스 계약 조건에 동의하려면 확인란을 선택하고 다음을 선택합니다.
4. 설치 위치를 변경하려면 찾아보기를 선택하고 원하는 폴더로 이동한 후 확인을 선택합니다.
5. 설치 위치를 수락하려면 다음을 선택합니다.
6. 설치를 선택합니다.
7. 설치가 완료되면 완료를 선택합니다.

드라이버 구성 옵션을 설정하는 방법

Windows에서 Amazon Athena ODBC 드라이버의 동작을 제어하기 위해 다음과 같은 방법으로 드라이버 구성 옵션을 설정할 수 있습니다.

- ODBC 데이터 원본 관리자 프로그램에서 데이터 소스 이름(DSN)을 구성하는 경우.
- 다음 위치에서 Windows 레지스트리 키를 추가하거나 변경합니다.

```
HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\YOUR_DSN_NAME
```

- 프로그래밍 방식으로 연결할 때 연결 문자열에서 드라이버 옵션을 설정합니다.

Windows에서 데이터 소스 이름 구성

ODBC 드라이버를 다운로드하여 설치한 후에는 데이터 소스 이름(DSN) 항목을 클라이언트 컴퓨터 또는 Amazon EC2 인스턴스에 추가해야 합니다. SQL 클라이언트 도구는 이 데이터 소스를 사용하여 Amazon Athena 데이터베이스에 연결하고 쿼리합니다.

시스템 DSN 항목을 만들려면

1. Windows 시작 메뉴에서 ODBC 데이터 원본(64비트)을 마우스 오른쪽 버튼을 클릭하고 더 보기, 관리자 권한으로 실행을 선택합니다.
2. ODBC 데이터 원본 관리자에서 드라이버 탭을 선택합니다.
3. 이름 열에 Amazon Athena ODBC(x64)가 있는지 확인합니다.
4. 다음 중 하나를 수행하십시오.
   - 컴퓨터의 모든 사용자를 위한 드라이버를 구성하려면 시스템 DSN 탭을 선택합니다. 다른 계정을 사용하여 데이터를 로드하는 애플리케이션은 다른 계정의 사용자 DSN을 감지하지 못할 수 있으므로 시스템 DSN 구성 옵션을 사용하는 것이 좋습니다.

     > **Note**
     >
     > 시스템 DSN 옵션을 사용하려면 관리자 권한이 필요합니다.

   - 사용자 계정용으로만 드라이버를 구성하려면 사용자 DSN 탭을 선택합니다.
5. 추가를 선택합니다. 새 데이터 원본 만들기 대화 상자가 열립니다.
6. Amazon Athena ODBC(x64)를 선택하고 마침을 선택합니다.

7. Amazon Athena ODBC 구성 대화 상자에 다음 정보를 입력합니다. 이러한 옵션에 대한 자세한 내용은 기본 ODBC 2.x 연결 파라미터 섹션을 참조하세요.

   - 데이터 소스 이름에는 데이터 소스를 식별하는 데 사용할 이름을 입력합니다.
   - 설명에는 데이터 소스를 식별하는 데 도움이 되는 설명을 입력합니다.
   - 리전에는 Athena를 사용할 AWS 리전의 이름을 입력합니다(예: **us-west-1**).
   - 카탈로그에는 Amazon Athena 카탈로그의 이름을 입력합니다. 기본값은 AWS Glue에서 사용하는 AwsDataCatalog입니다.
   - 데이터베이스에는 Amazon Athena 데이터베이스 이름을 입력합니다. 기본값은 기본값입니다.
   - 작업 그룹의 경우 Amazon Athena 작업 그룹의 이름을 입력합니다. 기본값은 프라이머리입니다.
   - S3 출력 위치의 경우 쿼리 결과가 저장될 Amazon S3의 위치를 입력합니다(예: **s3://DOC-EXAMPLE-BUCKET/**).
   - (선택 사항) 암호화 옵션에서 암호화 옵션을 선택합니다. 기본값은 NOT_SET입니다.
   - (선택 사항) KMS 키에서 필요하면 암호화 KMS 키를 선택합니다.
8. IAM 인증을 위한 구성 옵션을 지정하려면 인증 옵션을 선택합니다.
9. 다음 정보를 입력합니다.

   - 인증 유형에서 IAM 보안 인증을 선택합니다. 이 값이 기본값입니다. 사용할 수 있는 인증 유형에 대한 자세한 내용은 인증 옵션 섹션을 참조하세요.
   - 사용자 이름에는 사용자 이름을 입력합니다.
   - 암호에는 암호를 입력합니다.
   - 임시 AWS 보안 인증을 사용하려는 경우 세션 토큰에 세션 토큰을 입력합니다. 임시 보안 인증에 대한 자세한 내용은 IAM 사용 설명서의 AWS 리소스에서 임시 자격 증명 사용을 참조하세요.
10. 확인을 선택합니다.
11. Amazon Athena ODBC 구성 대화 상자 하단에서 테스트를 선택합니다. 클라이언트 컴퓨터가 Amazon Athena에 연결되면 연결 테스트 상자에 연결 성공이 보고됩니다. 그렇지 않으면 상자에 해당 오류 정보와 함께 연결 실패가 보고됩니다.
12. 확인을 선택하여 연결 테스트를 닫습니다. 이제 생성한 데이터 소스가 데이터 소스 이름 목록에 표시됩니다.

Windows에서 DSN이 없는 연결 사용

DSN이 없는 연결을 사용하면 데이터 소스 이름(DSN) 없이 데이터베이스에 연결할 수 있습니다. 다음 예제에서는 Amazon Athena에 연결하는 Amazon Athena ODBC(x64) ODBC 드라이버의 연결 문자열을 보여줍니다.

```
DRIVER={Amazon Athena ODBC (x64)};Catalog=AwsDataCatalog;AwsRegion=us-
west-1;Schema=test_schema;S3OutputLocation=
s3://DOC-EXAMPLE-BUCKET/;AuthenticationType=IAM Credentials;UID=YOUR_UID;PWD=YOUR_PWD;
```

Linux

Linux 클라이언트 컴퓨터를 사용하여 Amazon Athena에 액세스하려면 Amazon Athena ODBC 드라이버가 필요합니다.

Linux 시스템 사양

드라이버를 설치하는 각 Linux 클라이언트 컴퓨터마다 다음 요구 사항을 만족해야 합니다.

- 루트 액세스 권한이 있습니다.
- 다음 배포판 중 하나를 사용:
  - RHEL(Red Hat Enterprise Linux) 7 또는 8
  - CentOS 7 또는 8
- 100MB의 디스크 공간을 사용할 수 있습니다.
- unixODBC 버전 2.3.1 이상을 사용합니다.
- GNU C Library(glibc) 버전 2.26 이상을 사용합니다.

Linux에 ODBC 데이터 커넥터 설치

다음 절차를 따라 Linux 운영 체제에 Amazon Athena ODBC 드라이버를 설치합니다.

Linux에 Amazon Athena ODBC 드라이버 설치

1. 다음 명령 중 하나를 입력합니다.

   ```
   sudo rpm -Uvh AmazonAthenaODBC-2.X.Y.Z.rpm
   ```

   또는

```
sudo yum --nogpgcheck localinstall AmazonAthenaODBC-2.X.Y.Z.rpm
```

2. 설치가 완료되면 다음 명령 중 하나를 입력하여 드라이버가 설치되었는지 확인합니다.

   - ```
     yum list | grep amazon-athena-odbc-driver
     ```

     출력:

     ```
     amazon-athena-odbc-driver.x86_64 2.0.2.1-1.amzn2int installed
     ```

   - ```
     rpm -qa | grep amazon
     ```

     출력:

     ```
     amazon-athena-odbc-driver-2.0.2.1-1.amzn2int.x86_64
     ```

Linux에서 데이터 소스 이름 구성

드라이버가 설치된 후 다음 위치에서 예제 `.odbc.ini` 및 `.odbcinst.ini` 파일을 찾을 수 있습니다.

- `/opt/athena/odbc/ini/`.

여기에 있는 `.ini` 파일을 Amazon Athena ODBC 드라이버 및 DSN(데이터 소스 이름) 구성의 예제로 사용합니다.

> **Note**
>
> 기본적으로 ODBC 드라이버 관리자는 홈 디렉터리에 있는 숨겨진 구성 파일인 `.odbc.ini` 및 `.odbcinst.ini`를 사용합니다.

unixODBC를 사용하여 `.odbc.ini` 및 `.odbcinst.ini` 파일의 경로를 지정하려면 다음 단계를 수행합니다.

unixODBC를 사용하여 ODBC **`.ini`** 파일 위치 지정

1. 다음 예제와 같이 `ODBCINI`를 `odbc.ini` 파일의 전체 경로 및 파일 이름으로 설정합니다.

```
export ODBCINI=/opt/athena/odbc/ini/odbc.ini
```

2. 다음 예제와 같이 ODBCSYSINI를 odbcinst.ini 파일이 포함된 디렉토리의 전체 경로로 설정합니다.

```
export ODBCSYSINI=/opt/athena/odbc/ini
```

3. 다음 명령을 입력하여 unixODBC 드라이버 관리자 및 올바른 odbc*.ini 파일을 사용하고 있는지 확인합니다.

```
username % odbcinst -j
```

샘플 출력

```
unixODBC 2.3.1
DRIVERS............: /opt/athena/odbc/ini/odbcinst.ini
SYSTEM DATA SOURCES: /opt/athena/odbc/ini/odbc.ini
FILE DATA SOURCES..: /opt/athena/odbc/ini/ODBCDataSources
USER DATA SOURCES..: /opt/athena/odbc/ini/odbc.ini
SQLULEN Size.......: 8
SQLLEN Size........: 8
SQLSETPOSIROW Size.: 8
```

4. DSN(데이터 소스 이름)을 사용하여 데이터 스토어에 연결하려는 경우 odbc.ini 파일을 구성하여 DSN(데이터 소스 이름)을 정의합니다. 다음 예제와 같이 odbc.ini 파일의 속성을 설정하여 데이터 스토어에 대한 연결 정보를 지정하는 DSN을 생성합니다.

```
[ODBC Data Sources]
athena_odbc_test=Amazon Athena ODBC (x64)

[ATHENA_WIDE_SETTINGS]  # Special DSN-name to signal driver about logging
 configuration.
LogLevel=0              # To enable ODBC driver logs, set this to 1.
UseAwsLogger=0          # To enable AWS-SDK logs, set this to 1.
LogPath=/opt/athena/odbc/logs/ # Path to store the log files. Permissions to the
 location are required.

[athena_odbc_test]
Driver=/opt/athena/odbc/lib/libathena-odbc.so
AwsRegion=us-west-1
```

```
Workgroup=primary
Catalog=AwsDataCatalog
Schema=default
AuthenticationType=IAM Credentials
UID=
PWD=
S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/
```

5. 다음 예제와 같이 `odbcinst.ini` 파일을 구성합니다.

```
[ODBC Drivers]
Amazon Athena ODBC (x64)=Installed

[Amazon Athena ODBC (x64)]
Driver=/opt/athena/odbc/lib/libathena-odbc.so
Setup=/opt/athena/odbc/lib/libathena-odbc.so
```

6. Amazon Athena ODBC 드라이버를 설치하고 구성한 후에는 다음 예제와 같이 unixODBC `isql` 명령줄 도구를 사용하여 연결을 확인합니다.

```
username % isql -v "athena_odbc_test"
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
SQL>
```

macOS

macOS 클라이언트 컴퓨터를 사용하여 Amazon Athena에 액세스하려면 Amazon Athena ODBC 드라이버가 필요합니다.

macOS 시스템 요구 사항

드라이버를 설치하는 각 macOS 컴퓨터마다 다음 요구 사항을 만족해야 합니다.

- macOS 버전 14 이상을 사용합니다.
- 100MB의 디스크 공간을 사용할 수 있습니다.

- iODBC 버전 3.52.16 이상을 사용합니다.

macOS에 ODBC 데이터 커넥터 설치

macOS 운영 체제용 Amazon Athena ODBC 드라이버를 다운로드하여 설치하려면 다음 절차를 따르세요.

macOS용 Amazon Athena ODBC 드라이버 다운로드 및 설치

1. `.pkg` 패키지 파일을 다운로드합니다.
2. `.pkg` 파일을 두 번 클릭합니다.
3. 마법사의 단계를 따라 드라이버를 설치합니다.
4. 라이선스 계약 페이지에서 계속을 누른 다음 동의를 선택합니다.
5. 설치를 선택합니다.
6. 설치가 완료되면 완료를 선택합니다.
7. 다음 명령을 입력하여 드라이버가 설치되었는지 확인:

```
> pkgutil --pkgs | grep athenaodbc
```

시스템에 따라 다음 중 하나와 같은 내용이 출력될 수 있습니다.

```
com.amazon.athenaodbc-x86_64.Config
com.amazon.athenaodbc-x86_64.Driver
```

또는

```
com.amazon.athenaodbc-arm64.Config
com.amazon.athenaodbc-arm64.Driver
```

macOS에서 데이터 소스 이름 구성

드라이버가 설치된 후 다음 위치에서 예제 `.odbc.ini` 및 `.odbcinst.ini` 파일을 찾을 수 있습니다.

- Intel 프로세서 컴퓨터: `/opt/athena/odbc/x86_64/ini/`
- ARM 프로세서 컴퓨터: `/opt/athena/odbc/arm64/ini/`

여기에 있는 `.ini` 파일을 Amazon Athena ODBC 드라이버 및 DSN(데이터 소스 이름) 구성의 예제로 사용합니다.

> **Note**
>
> 기본적으로 ODBC 드라이버 관리자는 홈 디렉터리에 있는 숨겨진 구성 파일인 `.odbc.ini` 및 `.odbcinst.ini`를 사용합니다.

iODBC 드라이버 관리자를 사용하여 `.odbc.ini` 및 `.odbcinst.ini` 파일의 경로를 지정하려면 다음 단계를 수행합니다.

iODBC 드라이버 관리자를 사용하여 ODBC **`.ini`** 파일 위치 지정

1. `ODBCINI`를 `odbc.ini` 파일의 전체 경로 및 파일 이름으로 설정합니다.
   - Intel 프로세서가 탑재된 macOS 컴퓨터의 경우 다음 구문을 사용합니다.

     ```
     export ODBCINI=/opt/athena/odbc/x86_64/ini/odbc.ini
     ```
   - ARM 프로세서가 탑재된 macOS 컴퓨터의 경우 다음 구문을 사용합니다.

     ```
     export ODBCINI=/opt/athena/odbc/arm64/ini/odbc.ini
     ```
2. `ODBCSYSINI`를 `odbcinst.ini` 파일의 전체 경로 및 파일 이름으로 설정합니다.
   - Intel 프로세서가 탑재된 macOS 컴퓨터의 경우 다음 구문을 사용합니다.

     ```
     export ODBCSYSINI=/opt/athena/odbc/x86_64/ini/odbcinst.ini
     ```
   - ARM 프로세서가 탑재된 macOS 컴퓨터의 경우 다음 구문을 사용합니다.

     ```
     export ODBCSYSINI=/opt/athena/odbc/arm64/ini/odbcinst.ini
     ```
3. DSN(데이터 소스 이름)을 사용하여 데이터 스토어에 연결하려는 경우 `odbc.ini` 파일을 구성하여 DSN(데이터 소스 이름)을 정의합니다. 다음 예제와 같이 `odbc.ini` 파일의 속성을 설정하여 데이터 스토어에 대한 연결 정보를 지정하는 DSN을 생성합니다.

   ```
   [ODBC Data Sources]
   athena_odbc_test=Amazon Athena ODBC (x64)
   ```

```
[ATHENA_WIDE_SETTINGS] # Special DSN-name to signal driver about logging
 configuration.
LogLevel=0             # set to 1 to enable ODBC driver logs
UseAwsLogger=0         # set to 1 to enable AWS-SDK logs
LogPath=/opt/athena/odbc/logs/ # Path to store the log files. Permissions to the
 location are required.

[athena_odbc_test]
Description=Amazon Athena ODBC (x64)
# For ARM:
Driver=/opt/athena/odbc/arm64/lib/libathena-odbc-arm64.dylib
# For Intel:
# Driver=/opt/athena/odbc/x86_64/lib/libathena-odbc-x86_64.dylib
AwsRegion=us-west-1
Workgroup=primary
Catalog=AwsDataCatalog
Schema=default
AuthenticationType=IAM Credentials
UID=
PWD=
S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/
```

4. 다음 예제와 같이 `odbcinst.ini` 파일을 구성합니다.

```
[ODBC Drivers]
Amazon Athena ODBC (x64)=Installed

[Amazon Athena ODBC (x64)]
# For ARM:
Driver=/opt/athena/odbc/arm64/lib/libathena-odbc-arm64.dylib
Setup=/opt/athena/odbc/arm64/lib/libathena-odbc-arm64.dylib
# For Intel:
# Driver=/opt/athena/odbc/x86_64/lib/libathena-odbc-x86_64.dylib
# Setup=/opt/athena/odbc/x86_64/lib/libathena-odbc-x86_64.dylib
```

5. Amazon Athena ODBC 드라이버를 설치하고 구성한 후에는 다음 예제와 같이 `iodbctest` 명령줄 도구를 사용하여 연결을 확인합니다.

```
username@ % iodbctest
iODBC Demonstration program
This program shows an interactive SQL processor
Driver Manager: 03.52.1623.0502
```

```
Enter ODBC connect string (? shows list): ?

DSN                             | Driver
------------------------------------------------------------------------------
athena_odbc_test                | Amazon Athena ODBC (x64)

Enter ODBC connect string (? shows list): DSN=athena_odbc_test;
Driver: 2.0.2.1 (Amazon Athena ODBC Driver)

SQL>
```

Athena ODBC 2.x 연결 파라미터

Amazon Athena ODBC Configuration 대화 상자 옵션에는 Authentication Options, Advanced Options, Logging Options, Endpoint Overrides, Proxy Options가 포함되어 있습니다. 각 옵션에 대한 자세한 내용은 해당 링크를 참조하세요.

- 기본 ODBC 2.x 연결 파라미터
- 인증 옵션
- 고급 옵션
- 로깅 옵션
- 엔드포인트 재정의
- 프록시 옵션

기본 ODBC 2.x 연결 파라미터

다음 섹션에서는 각 기본 연결 파라미터를 설명합니다.

데이터 원본 이름

데이터 소스의 이름을 지정합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
DSN	DSN이 없는 연결 유형의 경우 선택 사항	none	`DSN=AmazonAthenaOdbcUsWest1;`

설명

데이터 소스에 대한 설명을 포함합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
설명	선택 사항	none	`Description=Connection to Amazon Athena us-west-1;`

카탈로그

데이터 카탈로그 이름을 지정합니다. 자세한 내용은 Amazon Athena API 참조의 DataCatalog를 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
카탈로그	선택 사항	`AwsDataCatalog`	`Catalog=AwsDataCatalog;`

지역

AWS 리전를 지정합니다. AWS 리전에 대한 자세한 내용은 리전 및 가용 영역을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
AwsRegion	필수	none	`AwsRegion =us-west-1;`

데이터베이스

데이터베이스 이름을 지정합니다. 자세한 내용은 Amazon Athena API 참조의 Database를 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
스키마	선택 사항	`default`	`Schema=default;`

작업 그룹

작업 그룹 이름을 지정합니다. 작업 그룹에 대한 자세한 내용은 Amazon Athena API 참조의 WorkGroup을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
작업 그룹	선택 사항	`primary`	`Workgroup =primary;`

출력 위치

쿼리 결과가 저장되는 Amazon S3의 위치를 지정합니다. 출력 위치에 대한 자세한 내용은 Amazon Athena API 참조의 ResultConfiguration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
S3OutputLocation	필수	none	`S3OutputL ocation=s3:// DOC-EXAMPLE-BU CKET/;`

암호화 옵션

대화 상자 파라미터 이름: 암호화 옵션

암호화 옵션을 지정합니다. 암호화 옵션에 대한 자세한 내용은 Amazon Athena API 참조의 EncryptionConfiguration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	가능한 값	연결 문자열 예제
S3OutputEncOption	선택 사항	none	NOT_SET, SSE_S3, SSE_KMS, CSE_KMS	S3OutputEncOption=SSE_S3;

KMS 키

암호화를 위한 KMS 키를 지정합니다. KMS 키의 암호화 구성에 대한 자세한 내용은 Amazon Athena API 참조의 EncryptionConfiguration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
S3OutputEncKMSKey	선택 사항	none	S3OutputEncKMSKey=your_key;

연결 테스트

ODBC 데이터 원본 관리자는 Amazon Athena에 대한 ODBC 2.x 연결을 테스트하는 데 사용할 수 있는 테스트 옵션을 제공합니다. 단계는 Windows에서 데이터 소스 이름 구성를 참조하세요. 연결을 테스트하는 경우 ODBC 드라이버는 GetWorkGroup Athena API 작업을 직접적으로 호출합니다. 직접 호출은 보안 인증 정보를 검색하기 위해 지정한 인증 유형 및 해당 보안 인증 제공업체를 사용합니다. ODBC 2.x 드라이버를 사용하는 경우 연결 테스트에 요금이 부과되지 않습니다. 테스트할 때 Amazon S3 버킷에 쿼리 결과가 생성되지 않습니다.

인증 옵션

다음 인증 유형을 사용하여 Amazon Athena에 연결할 수 있습니다. 모든 유형에서 연결 문자열 이름은 `AuthenticationType`이고, 파라미터 유형은 `Required`이며, 기본값은 `IAM Credentials`입니다. 각 인증 유형의 파라미터에 대한 자세한 내용은 해당 링크를 참조하세요. 일반 인증 파라미터에 대해서는 공통 인증 파라미터 섹션을 참조하세요.

인증 유형	연결 문자열 예제
IAM 보안 인증	AuthenticationType=IAM Credentials;

인증 유형	연결 문자열 예제
IAM 프로파일	`AuthenticationType=IAM Profile;`
AD FS	`AuthenticationType=ADFS;`
Azure AD	`AuthenticationType=AzureAD;`
브라우저 Azure AD	`AuthenticationType=BrowserAzureAD;`
브라우저 SAML	`AuthenticationType=BrowserSAML;`
브라우저 SSO OIDC	`AuthenticationType=BrowserSSOOIDC;`
기본 보안 인증	`AuthenticationType=Default Credentials;`
외부 자격 증명	`AuthenticationType=External Credentials;`
인스턴스 프로파일	`AuthenticationType=Instance Profile;`
JWT	`AuthenticationType=JWT;`
Okta	`AuthenticationType=Okta;`
Ping	`AuthenticationType=Ping;`

IAM 보안 인증

IAM 보안 인증을 통해 이 섹션에서 설명한 연결 문자열 파라미터를 사용하여 ODBC 드라이버로 Amazon Athena에 연결할 수 있습니다.

인증 유형

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
AuthenticationType	필수	`IAM Credentials`	`AuthenticationType=IAM Credentials;`

사용자 ID

사용자의 AWS 액세스 키 ID입니다. 액세스 키에 대한 자세한 내용은 IAM 사용 설명서의 AWS 보안 자격 증명을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
UID	필수	none	`UID=AKIAIOSFODNN7EXAMPLE;`

암호

사용자의 AWS 비밀 키 ID입니다. 액세스 키에 대한 자세한 내용은 IAM 사용 설명서의 AWS 보안 자격 증명을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
PWD	필수	none	`PWD=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKE;`

세션 토큰

임시 AWS 보안 인증을 사용하는 경우 세션 토큰을 지정해야 합니다. 임시 보안 인증에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 임시 보안 자격 증명을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
SessionToken	선택 사항	none	`SessionToken=AQoDYXdzEJr...<remainder of session token>;`

IAM 프로파일

ODBC 드라이버를 사용하여 Amazon Athena에 연결하도록 명명된 프로파일을 구성할 수 있습니다. 호스팅 Amazon EC2 인스턴스 프로파일에서 제공되는 보안 인증을 사용하려면 `credential_source` 파라미터를 `Ec2InstanceMetadata`로 설정합니다. 명명된 프로파일에서 사용자 지정 보안 인증 제공업체를 사용하려면 프로파일 구성에서 `plugin_name` 파라미터 값을 지정합니다.

인증 유형

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
AuthenticationType	필수	`IAM Credentials`	`AuthenticationType=IAM Profile;`

AWS 프로파일

ODBC 연결에 사용할 프로파일 이름입니다. 프로파일에 대한 자세한 내용은 AWS Command Line Interface 사용 설명서의 명명된 프로파일 사용을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
AWS 프로파일	필수	none	`AWSProfile=default;`

기본 역할

수임할 역할의 Amazon 리소스 이름(ARN)입니다. 기본 역할 파라미터는 프로파일 구성의 `plugin_name` 파라미터에서 사용자 지정 보안 인증 제공업체를 지정할 때 사용됩니다. ARN 역할에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
preferred_role	선택 사항	none	`preferred_role=arn`

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
			`:aws:IAM::123456789012:id/user1;`

세션 지속 시간

역할 세션 지속 시간(초)입니다. 세션 지속 시간에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요. 세션 기간 파라미터는 프로파일 구성의 `plugin_name` 파라미터에서 사용자 지정 보안 인증 제공업체를 지정할 때 사용됩니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
기간	선택 사항	`900`	`duration=900;`

플러그인 이름

명명된 프로파일에 사용되는 사용자 지정 보안 인증 제공업체의 이름을 지정합니다. 이 파라미터는 ODBC 데이터 원본 관리자의 인증 유형 필드 값과 값이 같을 수 있지만 `AWSProfile` 구성에서만 사용됩니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
plugin_name	선택 사항	`none`	`plugin_name=AzureAD;`

AD FS

AD FS는 Active Directory Federation Service(Active Directory Federation Service) ID 제공업체에서 작동하는 SAML 기반 인증 플러그인입니다. 플러그인은 Windows 통합 인증 및 양식 기반 인증을 지원합니다. Windows 통합 인증을 사용하는 경우 사용자 이름과 암호를 생략할 수 있습니다. AD FS 및 Athena 구성에 대한 자세한 내용은 ODBC 클라이언트를 사용하여 Microsoft AD FS 사용자를 위해 Amazon Athena에 대한 페더레이션 액세스 구성 섹션을 참조하세요.

인증 유형

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
AuthenticationType	필수	`IAM Credentials`	`AuthenticationType=ADFS;`

사용자 ID

AD FS 서버에 연결하는 데 사용하는 사용자 이름입니다. Windows 통합 인증의 경우 사용자 이름을 생략할 수 있습니다. AD FS 설정에 사용자 이름이 필요한 경우 연결 파라미터에서 제공해야 합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
UID	Windows 통합 인증의 경우 선택 사항	none	`UID=domain\username;`

암호

AD FS 서버에 연결하는 데 사용하는 암호입니다. 사용자 이름 필드와 마찬가지로 Windows 통합 인증을 사용하는 경우 사용자 이름을 생략할 수 있습니다. AD FS 설정에 암호가 필요한 경우 연결 파라미터에서 제공해야 합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
PWD	Windows 통합 인증의 경우 선택 사항	none	`PWD=password_3EXAMPLE;`

기본 역할

수임할 역할의 Amazon 리소스 이름(ARN)입니다. SAML 어설션에 여러 역할이 있는 경우 이 파라미터를 지정하여 수임할 역할을 선택할 수 있습니다. 이 역할은 SAML 어설션에 있어야 합니다. ARN 역할에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
preferred_role	선택 사항	none	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

세션 지속 시간

역할 세션 지속 시간(초)입니다. 세션 지속 시간에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
기간	선택 사항	`900`	`duration=900;`

IdP 호스트

AD FS 서비스 호스트의 이름입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
idp_host	필수	none	`idp_host=<server-name>.<company.com>;`

IdP 포트

AD FS 호스트에 연결하는 데 사용할 포트입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
idp_port	필수	none	`idp_port=443;`

LoginToRP

신뢰할 수 있는 신뢰 당사자입니다. 이 파라미터를 사용하여 AD FS 신뢰 당사자 엔드포인트 URL을 재정의합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
LoginToRP	선택 사항	`urn:amazon:webservices`	`LoginToRP=trustedparty;`

Azure AD

Azure AD는 Azure AD ID 제공업체에서 작동하는 SAML 기반 인증 플러그인입니다. 이 플러그인은 다중 인증(MFA)을 지원하지 않습니다. MFA 지원이 필요한 경우 `BrowserAzureAD` 플러그인 사용을 고려하세요.

인증 유형

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
AuthenticationType	필수	`IAM Credentials`	`AuthenticationType=AzureAD;`

기본 역할

수임할 역할의 Amazon 리소스 이름(ARN)입니다. ARN 역할에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
preferred_role	선택 사항	`none`	`preferred_role=arn:aws:iam::123456789012:id/user1;`

세션 지속 시간

역할 세션 지속 시간(초)입니다. 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
기간	선택 사항	900	duration=900;

테넌트 ID

애플리케이션 테넌트 ID를 지정합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
idp_tenant	필수	none	idp_tenant=123zz112z-z12d-1z1f-11zz-f111aa111234;

클라이언트 ID

애플리케이션 클라이언트 ID를 지정합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
client_id	필수	none	client_id=9178ac27-a1bc-1a2b-1a2b-a123abcd1234;

클라이언트 암호

클라이언트 암호를 지정합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
client_secret	필수	none	client_secret=zG12q~.xzG1xxxZ1wX1.~ZzXXX1XxkHZizeT1zzZ;

브라우저 Azure AD

브라우저 Azure AD는 Azure AD ID 제공업체에서 작동하고 다중 인증을 지원하는 SAML 기반 인증 플러그인입니다. 표준 Azure AD 플러그인과 달리 이 플러그인은 연결 파라미터에서 사용자 이름, 암호 또는 클라이언트 암호를 요구하지 않습니다.

인증 유형

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
AuthenticationType	필수	`IAM Credentials`	`AuthenticationType=BrowserAzureAD;`

기본 역할

수임할 역할의 Amazon 리소스 이름(ARN)입니다. SAML 어설션에 여러 역할이 있는 경우 이 파라미터를 지정하여 수임할 역할을 선택할 수 있습니다. 지정된 역할은 SAML 어설션에 있어야 합니다. ARN 역할에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
preferred_role	선택 사항	none	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

세션 지속 시간

역할 세션 지속 시간(초)입니다. 세션 지속 시간에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
기간	선택 사항	`900`	`duration=900;`

테넌트 ID

애플리케이션 테넌트 ID를 지정합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
idp_tenant	필수	none	`idp_tenant=123zz112z-z12d-1z1f-11zz-f111aa111234;`

클라이언트 ID

애플리케이션 클라이언트 ID를 지정합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
client_id	필수	none	`client_id=9178ac27-a1bc-1a2b-1a2b-a123abcd1234;`

제한 시간

플러그인이 Azure AD로부터 SAML 응답 대기를 중지하기까지의 시간(초)입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
제한 시간	선택 사항	`120`	`timeout=90;`

Azure 파일 캐시 활성화

임시 보안 인증 캐시를 활성화합니다. 이 연결 파라미터를 사용하면 임시 보안 인증을 캐시하여 여러 프로세스 간에 재사용할 수 있습니다. Microsoft Power BI와 같은 BI 도구를 사용할 때 열려 있는 브라우저 창 수를 줄이려면 이 옵션을 사용합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
browser_azure_cache	선택 사항	1	`browser_azure_cache=0;`

브라우저 SAML

브라우저 SAML은 SAML 기반 ID 제공업체에서 작동하고 다중 인증을 지원할 수 있는 일반 인증 플러그인입니다. 자세한 구성 정보는 ODBC, SAML 2.0 및 Okta 자격 증명 공급자를 사용하여 통합 인증 구성 섹션을 참조하세요.

인증 유형

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
AuthenticationType	필수	`IAM Credentials`	`AuthenticationType=BrowserSAML;`

기본 역할

수임할 역할의 Amazon 리소스 이름(ARN)입니다. SAML 어설션에 여러 역할이 있는 경우 이 파라미터를 지정하여 수임할 역할을 선택할 수 있습니다. 이 역할은 SAML 어설션에 있어야 합니다. ARN 역할에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
preferred_role	선택 사항	`none`	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

세션 지속 시간

역할 세션 지속 시간(초)입니다. 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
기간	선택 사항	`900`	`duration=900;`

로그인 URL

애플리케이션에 표시되는 Single Sign-On URL입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
login_url	필수	none	`login_url =https:// trial-123 4567.okta.com/ app/trial-123 4567_okta browsersaml_1/ zzz4izzzAzDFB zZz1234/sso/ saml;`

리스너 포트

SAML 응답을 수신하는 데 사용되는 포트 번호입니다. 이 값은 IdP를 구성한 IAM Identity Center URL과 일치해야 합니다(예: `http://localhost:7890/athena`).

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
listen_port	선택 사항	`7890`	`listen_po rt=7890;`

제한 시간

플러그인이 ID 제공업체로부터 SAML 응답 대기를 중지하기까지의 시간(초)입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
제한 시간	선택 사항	`120`	`timeout=90;`

브라우저 SSO OIDC

브라우저 SSO OIDC는 AWS IAM Identity Center에서 작동하는 인증 플러그인입니다. IAM Identity Center 활성화 및 사용에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 Step 1: Enable IAM Identity Center를 참조하세요.

인증 유형

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
AuthenticationType	필수	`IAM Credentials`	`AuthenticationType=BrowserSSOOIDC;`

IAM Identity Center 시작 URL

AWS 액세스 포털의 URL입니다. IAM Identity Center StartDeviceAuthorization API 작업은 `startUrl` 파라미터에 대해 이 값을 사용합니다.

AWS 액세스 포털 URL을 복사하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/singlesignon/에서 AWS IAM Identity Center 콘솔을 엽니다.
2. 탐색 창에서 설정을 선택합니다.
3. 설정 페이지의 자격 증명 소스에서 AWS 액세스 포털 URL의 클립보드 아이콘을 선택합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
sso_oidc_start_url	필수	none	`sso_oidc_start_url=https://app_id.awsapps.com/start;`

IAM Identity Center 리전

SSO가 구성된 AWS 리전입니다. `SSOOIDCClient` 및 `SSOClient` AWS SDK 클라이언트는 `region` 파라미터에 대해 이 값을 사용합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
sso_oidc_region	필수	none	`sso_oidc_region=us-east-1;`

범위

클라이언트에서 정의한 범위 목록입니다. 권한 부여 시 이 목록은 액세스 토큰이 부여될 때 권한을 제한합니다. IAM Identity Center RegisterClient API 작업은 `scopes` 파라미터에 대해 이 값을 사용합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
sso_oidc_scopes	선택 사항	none	`sso_oidc_scopes=scope1,scope2,scope3;`

계정 ID

사용자에게 할당된 AWS 계정의 식별자입니다. IAM Identity Center GetRoleCredentials API는 `accountId` 파라미터에 대해 이 값을 사용합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
sso_oidc_account_id	필수	none	`sso_oidc_account_id=123456789123;`

역할 이름

사용자에게 할당된 역할의 친숙한 이름입니다. 이 권한 세트에 지정한 이름은 AWS 액세스 포털에서 사용 가능한 역할로 표시됩니다. IAM Identity Center GetRoleCredentials API 작업은 `roleName` 파라미터에 대해 이 값을 사용합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
sso_oidc_role_name	필수	none	`sso_oidc_role_name=AthenaReadAccess;`

제한 시간

폴링 SSO API가 액세스 토큰을 확인해야 하는 시간(초)입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
sso_oidc_timeout	선택 사항	`120`	`sso_oidc_timeout=60;`

Azure 파일 캐시 활성화

임시 보안 인증 캐시를 활성화합니다. 이 연결 파라미터를 사용하면 임시 보안 인증을 캐시하여 여러 프로세스 간에 재사용할 수 있습니다. Microsoft Power BI와 같은 BI 도구를 사용할 때 열려 있는 브라우저 창 수를 줄이려면 이 옵션을 사용합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
sso_oidc_cache	선택 사항	1	`sso_oidc_cache=0;`

기본 보안 인증

클라이언트 시스템에서 구성한 기본 보안 인증을 사용하여 Amazon Athena에 연결할 수 있습니다. 기본 보안 인증 사용에 대한 자세한 내용은 AWS SDK for Java 개발자 안내서의 Using the Default Credential Provider Chain을 참조하세요.

인증 유형

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
AuthenticationType	필수	`IAM Credentials`	`AuthenticationType=DefaultCredentials;`

외부 자격 증명

외부 보안 인증은 외부 SAML 기반 ID 제공업체에 연결하는 데 사용할 수 있는 일반 인증 플러그인입니다. 플러그인을 사용하려면 SAML 응답을 반환하는 실행 파일을 전달합니다.

인증 유형

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
AuthenticationType	필수	`IAM Credentials`	`AuthenticationType=External Credentials;`

실행 가능한 경로

사용자 지정 SAML 기반 보안 인증 공급자의 로직이 포함된 실행 파일의 경로입니다. 실행 파일의 출력은 ID 제공업체의 구문 분석된 SAML 응답이어야 합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
ExecutablePath	필수	none	ExecutablePath=C:\Users*user_name* *external_credential.exe*

인수 목록

실행 파일로 전달할 인수 목록입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
ArgumentList	선택 사항	none	`ArgumentList=` *`arg1 arg2 arg3`*

인스턴스 프로파일

이 인증 유형은 EC2 인스턴스에서 사용되며 Amazon EC2 메타데이터 서비스를 통해 전달됩니다.

인증 유형

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
AuthenticationType	필수	`IAM Credentials`	`AuthenticationType=Instance Profile;`

JWT

JSON 웹 토큰(JWT) 플러그인은 JSON 웹 토큰을 사용하여 Amazon IAM 역할을 수임하는 인터페이스를 제공합니다. 구성은 ID 제공업체 구성에 따라 달라집니다. Google Cloud 및 AWS용 페더레이션을 구성하는 방법에 대한 자세한 내용은 Google Cloud 문서에서 AWS 또는 Azure와의 워크로드 아이덴티티 제휴 구성을 참조하세요.

인증 유형

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
AuthenticationType	필수	`IAM Credentials`	`AuthenticationType=JWT;`

기본 역할

수임할 역할의 Amazon 리소스 이름(ARN)입니다. ARN 역할에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
preferred_role	선택 사항	none	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

세션 지속 시간

역할 세션 지속 시간(초)입니다. 세션 지속 시간에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
기간	선택 사항	`900`	`duration=900;`

JSON 웹 토큰

AssumeRolewithWebIdentity AWS STS API 작업을 사용하여 IAM 임시 보안 인증을 검색하는 데 사용되는 JSON 웹 토큰입니다. Google Cloud Platform(GCP) 사용자를 위한 JSON 웹 토큰 생성에 대한 자세한 내용은 Google Cloud 문서의 JWT OAuth 토큰 사용을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
`web_identity_token`	필수	none	`web_identity_token=eyJhbGc...<remainder of token>;`

역할 세션 이름

세션의 이름입니다. 일반적인 기법은 애플리케이션 사용자의 이름 또는 식별자를 역할 세션 이름으로 사용하는 것입니다. 애플리케이션에서 사용하는 임시 보안 인증을 해당 사용자에게 편리하게 연결합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
role_session_name	필수	none	`role_session_name=familiarname;`

Okta

Okta는 Okta ID 제공업체에서 작동하는 SAML 기반 인증 플러그인입니다. Okta와 Amazon Athena의 페더레이션 구성에 대한 자세한 내용은 Okta 플러그인 및 Okta ID 공급자를 사용하여 ODBC에 대한 SSO 구성 섹션을 참조하세요.

인증 유형

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
AuthenticationType	필수	`IAM Credentials`	`AuthenticationType=Okta;`

사용자 ID

Okta 사용자 이름입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
UID	필수	none	`UID=jane.doe@org.com;`

암호

Okta 사용자의 암호입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
PWD	필수	none	`PWD=oktauserpasswordexample;`

기본 역할

수임할 역할의 Amazon 리소스 이름(ARN)입니다. ARN 역할에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
preferred_role	선택 사항	none	`preferred_role=arn:aws:IAM::123456789012:id/user1;`

세션 지속 시간

역할 세션 지속 시간(초)입니다. 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
기간	선택 사항	`900`	`duration=900;`

IdP 호스트

Okta 조직의 URL입니다. Okta 애플리케이션의 Embed Link URL에서 `idp_host` 파라미터를 추출할 수 있습니다. 단계는 Okta에서 ODBC 구성 정보 검색를 참조하세요. `https://` 이후 `okta.com`까지 (포함) 첫 번째 세그먼트가 IdP 호스트(예: `http://trial-1234567.okta.com`)입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
idp_host	필수	None	`idp_host=dev-99999999.okta.com;`

IdP 포트

IdP 호스트에 연결하는 데 사용할 포트 번호입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
idp_port	필수	None	`idp_port=443;`

Okta 앱 ID

애플리케이션의 ID입니다(두 부분으로 구성됨). Okta 애플리케이션의 Embed Link URL에서 `app_id` 파라미터를 추출할 수 있습니다. 단계는 Okta에서 ODBC 구성 정보 검색를 참조하세요. 애플리케이션 ID는 URL의 마지막 두 세그먼트(가운데 슬래시 포함)입니다. 세그먼트는 숫자, 대문자, 소문자가 혼합된 두 개의 20자 문자열(예: `Abc1de2fghi3J45kL678/abc1defghij2klmNo3p4`)입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
app_id	필수	None	`app_id=0oa25kx8ze9A3example/alnexamplea0piaWa0g7;`

Okta 앱 이름

Okta 애플리케이션의 이름입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
app_name	필수	None	app_name=amazon_aws_redshift;

Okta 대기 시간

다중 인증(MFA) 코드를 기다리는 시간(초 단위)을 지정합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
okta_mfa_wait_time	선택 사항	10	okta_mfa_wait_time=20;

Okata MFA 유형

MFA 팩터 유형입니다. 지원되는 유형은 Google Authenticator, SMS(Okta), Okta Verify(Push 사용), Okta Verify(TOTP 사용)입니다. 개별 조직의 보안 정책에 따라 사용자 로그인에 MFA가 필요한지 여부가 결정됩니다.

연결 문자열 이름	파라미터 유형	기본값	가능한 값	연결 문자열 예제
okta_mfa_type	Optional	None	googleauthenticator, smsauthentication, oktaverifywithpush, oktaverifywithtotp	okta_mfa_type=oktaverifywithpush;

Okta 전화번호

AWS SMS 인증에 사용할 전화번호입니다. 이 파라미터는 다중 팩터 등록에만 필요합니다. 휴대폰 번호가 이미 등록되어 있거나 보안 정책에서 AWS SMS 인증을 사용하지 않는 경우 이 필드를 무시해도 됩니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
okta_mfa_phone_number	MFA 등록 시 필수이며, 그렇지 않은 경우 선택 사항	None	`okta_mfa_phone_number=19991234567;`

Okta 파일 캐시 활성화

임시 보안 인증 캐시를 활성화합니다. 이 연결 파라미터를 사용하면 임시 보안 인증을 캐시하여 BI 애플리케이션에서 여는 여러 프로세스 간에 재사용할 수 있습니다. Okta API 제한을 방지하려면 이 옵션을 사용합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
okta_cache	선택 사항	`0`	`okta_cache=1;`

Ping

Ping은 PingFederate ID 제공업체에서 작동하는 SAML 기반 플러그인입니다.

인증 유형

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
AuthenticationType	필수	`IAM Credentials`	`AuthenticationType=Ping;`

사용자 ID

PingFederate 서버의 사용자 이름입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
UID	필수	none	`UID=pingusername@domain.com;`

암호

PingFederate 서버의 암호입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
PWD	필수	none	`PWD=pingpassword;`

기본 역할

수임할 역할의 Amazon 리소스 이름(ARN)입니다. SAML 어설션에 여러 역할이 있는 경우 이 파라미터를 지정하여 수임할 역할을 선택할 수 있습니다. 이 역할은 SAML 어설션에 있어야 합니다. ARN 역할에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
preferred_role	선택 사항	none	`preferred_role=arn:aws:iam::123456789012:id/user1;`

세션 지속 시간

역할 세션 지속 시간(초)입니다. 세션 지속 시간에 대한 자세한 내용은 AWS Security Token Service API 참조의 AssumeRole을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
기간	선택 사항	900	`duration=900;`

IdP 호스트

Ping 서버의 주소입니다. 주소를 찾으려면 다음 URL을 방문하여 SSO 애플리케이션 엔드포인트 필드를 확인합니다.

```
https://your-pf-host-#:9999/pingfederate/your-pf-app#/spConnections
```

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
idp_host	필수	none	idp_host=ec2-1-83-65-12.compute-1.amazonaws.com;

IdP 포트

IdP 호스트에 연결하는 데 사용할 포트 번호입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
idp_port	필수	None	idp_port=443;

파트너 SPID

서비스 제공업체 주소입니다. 서비스 제공업체 주소를 찾으려면 다음 URL을 방문하여 SSO 애플리케이션 엔드포인트 필드를 확인합니다.

```
https://your-pf-host-#:9999/pingfederate/your-pf-app#/spConnections
```

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
partner_spid	필수	None	partner_spid=https://us-east-1.signin.aws.amazon.com/platform/saml/<...>;

Ping URI 파라미터

인증 요청에 대한 URI 인수를 Ping에 전달합니다. Lake Formation 단일 역할 제한을 우회하려면 이 파라미터를 사용합니다. 전달된 파라미터를 인식하도록 Ping을 구성하고 전달된 역할이 사용자에게 할당된 역할 목록에 있는지 확인합니다. 그런 다음 SAML 어설션에서 단일 역할을 보냅니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
ping_uri_param	선택 사항	None	ping_uri_param=role=my_iam_role;

공통 인증 파라미터

이 섹션의 파라미터는 앞서 언급한 인증 유형에서 공통됩니다.

IdP에 대한 프록시 사용

프록시를 통해 드라이버와 IdP 간 통신을 활성화합니다. 이 옵션은 다음 인증 플러그인에서 사용할 수 있습니다.

- AD FS
- Azure AD
- 브라우저 Azure AD
- 브라우저 SSO OIDC
- JWT
- Okta
- Ping

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
UseProxyForIdP	선택 사항	0	UseProxyForIdP=1;

Lake Formation 사용

AssumeRoleWithSAML AWS STS API 작업 대신 AssumeDecoratedRoleWithSAML Lake Formation API 작업을 사용하여 임시 IAM 보안 인증 정보를 검색합니다. 이 옵션은 Azure AD, 브라우저 Azure AD, 브라우저 SAML, Okta, Ping, AD FS 인증 플러그인에서 사용할 수 있습니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
LakeformationEnabled	선택 사항	`0`	`LakeformationEnabled=1;`

SSL 비보안(IdP)

IdP와 통신할 때 SSL을 비활성화합니다. 이 옵션은 Azure AD, 브라우저 Azure AD, Okta, Ping, AD FS 인증 플러그인에서 사용할 수 있습니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
SSL_Insecure	선택 사항	`0`	`SSL_Insecure=1;`

엔드포인트 재정의

Athena 엔드포인트 재정의

`endpointOverride` `ClientConfiguration` 클래스는 이 값을 사용하여 Amazon Athena 클라이언트의 기본 HTTP 엔드포인트를 재정의합니다. 자세한 내용은 AWS SDK for C++ 개발자 안내서의 AWS Client configuration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
EndpointOverride	선택 사항	`none`	`EndpointOverride=athena.us-west-2.amazonaws.com;`

Athena 스트리밍 엔드포인트 재정의

이 `ClientConfiguration.endpointOverride` 메서드는 이 값을 사용하여 Amazon Athena 스트리밍 클라이언트의 기본 HTTP 엔드포인트를 재정의합니다. 자세한 내용은 AWS SDK for C++ 개발자 안내서의 AWS Client configuration을 참조하세요. Athena 스트리밍 서비스는 포트 444를 통해 사용할 수 있습니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
Streaming EndpointOverride	선택 사항	none	`StreamingEndpointOverride=athena.us-west-1.amazonaws.com:444;`

AWS STS 엔드포인트 재정의

이 `ClientConfiguration.endpointOverride` 메서드는 이 값을 사용하여 AWS STS 클라이언트의 기본 HTTP 엔드포인트를 재정의합니다. 자세한 내용은 AWS SDK for C++ 개발자 안내서의 AWS Client configuration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
StsEndpointOverride	선택 사항	none	`StsEndpointOverride=sts.us-west-1.amazonaws.com;`

Lake Formation 엔드포인트 재정의

이 `ClientConfiguration.endpointOverride` 메서드는 이 값을 사용하여 Lake Formation 클라이언트의 기본 HTTP 엔드포인트를 재정의합니다. 자세한 내용은 AWS SDK for C++ 개발자 안내서의 AWS Client configuration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
LakeFormationEndpointOverride	선택 사항	none	`LakeFormationEndpointOverride=lakeformation.us-west-1.amazonaws.com;`

SSO 엔드포인트 재정의

`ClientConfiguration.endpointOverride` 메서드는 이 값을 사용하여 SSO 클라이언트의 기본 HTTP 엔드포인트를 재정의합니다. 자세한 내용은 AWS SDK for C++ 개발자 안내서의 AWS Client configuration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
SSOEndpointOverride	선택 사항	none	`SSOEndpointOverride=portal.sso.us-east-2.amazonaws.com;`

SSO OIDC 엔드포인트 재정의

`ClientConfiguration.endpointOverride` 메서드는 이 값을 사용하여 SSO OIDC 클라이언트의 기본 HTTP 엔드포인트를 재정의합니다. 자세한 내용은 AWS SDK for C++ 개발자 안내서의 AWS Client configuration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
SSOOIDCEndpointOverride	선택 사항	none	`SSOOIDCEndpointOverride=oidc.us-east-2.amazonaws.com`

고급 옵션

가져오기 크기

이 요청에서 반환할 최대 결과(행) 수입니다. 파라미터 정보는 GetQuery MaxResults를 참조하세요. 스트리밍 API의 경우 최댓값은 1,000만 개입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
RowsToFetchPerBlock	선택 사항	스트리밍 이외의 경우 `1000` 스트리밍의 경우 `20000`	`RowsToFetchPerBlock=20000;`

결과 재사용 활성화

쿼리를 실행할 때 이전 쿼리 결과를 재사용할 수 있는지 여부를 지정합니다. 파라미터 정보는 ResultReuseByAgeConfiguration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
EnableResultReuse	선택 사항	`0`	`EnableResultReuse=1;`

결과 재사용 최대 수명

Athena에서 재사용을 고려해야 하는 이전 쿼리 결과의 최대 수명(분)을 지정합니다. 파라미터 정보는 ResultReuseByAgeConfiguration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
ReusedResultMaxAgeInMinutes	선택 사항	`60`	`ReusedResultMaxAgeInMinutes=90;`

스트리밍 API 활성화

Athena 스트리밍 API를 사용하여 결과 세트를 가져올지 여부를 선택합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
UseResultsetStreaming	선택 사항	`0`	`UseResultsetStreaming=1;`

S3 페처 활성화

Athena가 Amazon S3와 직접 상호 작용하여 Amazon S3 버킷에서 생성한 결과 세트를 가져옵니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
EnableS3Fetcher	선택 사항	1	`EnableS3Fetcher=1;`

여러 개의 S3 스레드 사용

여러 개의 스레드를 사용하여 Amazon S3에서 데이터를 가져옵니다. 이 옵션을 활성화하면 Amazon S3 버킷에 저장된 결과 파일을 여러 개의 스레드를 사용하여 병렬로 가져옵니다.

네트워크 대역폭이 양호한 경우에만 이 옵션을 활성화합니다. 예를 들어 EC2 c5.2xlarge 인스턴스에서 측정한 결과, 단일 스레드 S3 클라이언트는 1Gbps에 도달한 반면, 다중 스레드 S3 클라이언트에서 도달한 네트워크 처리량은 4Gbps로 나타났습니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
UseMultipleS3Threads	선택 사항	`0`	`UseMultipleS3Threads=1;`

단일 카탈로그 및 스키마 사용

기본적으로 ODBC 드라이버는 Athena에서 쿼리를 수행하여 사용 가능한 카탈로그 및 스키마 목록을 가져옵니다. 이 옵션은 드라이버가 ODBC 데이터 원본 관리자 구성 대화 상자 또는 연결 파라미터에서 지정한 카탈로그 및 스키마를 사용하도록 강제 적용됩니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
UseSingleCatalogAndSchema	선택 사항	`0`	`UseSingleCatalogAndSchema=1;`

쿼리를 사용하여 테이블 나열

`LAMBDA` 카탈로그 유형의 경우 ODBC 드라이버가 SHOW TABLES 쿼리를 제출하여 사용 가능한 테이블 목록을 가져올 수 있습니다. 이 설정이 기본값입니다. 이 파라미터가 0으로 설정되면 ODBC 드라이버가 Athena ListTableMetadata API를 사용하여 사용 가능한 테이블 목록을 가져옵니다. `LAMBDA` 카탈로그 유형의 경우 `ListTableMetadata` 사용 시 성능이 저하됩니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
UseQueryToListTables	선택 사항	`1`	`UseQueryToListTables=1;`

문자열 유형에 WCHAR 사용

기본적으로 ODBC 드라이버는 Athena 문자열 데이터 유형 `char`, `varchar`, `string`, `array`, `map<>`, `struct<>`, `row`에 `SQL_CHAR` 및 `SQL_VARCHAR`를 사용합니다. 이 파라미터를 `1`로 설정하면 드라이버에서 문자열 데이터 유형에 `SQL_WCHAR` 및 `SQL_WVARCHAR`를 강제로 사용합니다. 다양한 언어의 문자가 올바르게 저장 및 검색될 수 있도록 폭넓은 문자 및 폭넓은 가변 문자 유형을 사용하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
UseWCharForStringTypes	선택 사항	`0`	`UseWCharForStringTypes=1;`

외부 카탈로그 쿼리

드라이버가 Athena에서 외부 카탈로그를 쿼리해야 하는지 여부를 지정합니다. 자세한 내용은 ODBC 2.x 드라이버로 마이그레이션 단원을 참조하십시오.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
QueryExternalCatalogs	선택 사항	`0`	`QueryExternalCatalogs=1;`

SSL 확인

AWS SDK를 사용할 때 SSL 인증서를 확인할지 여부를 제어합니다. 이 값은 `ClientConfiguration.verifySSL` 파라미터에 전달됩니다. 자세한 내용은 AWS SDK for C++ 개발자 안내서의 AWS Client configuration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
VerifySSL	선택 사항	`1`	`VerifySSL=0;`

S3 결과 블록 크기

단일 Amazon S3 GetObject API 요청에 대해 다운로드할 블록의 크기(바이트)를 지정합니다. 기본값은 6,710만 8,864(64MB)입니다. 허용되는 최솟값 및 최댓값은 1,048만 5,760(10MB) 및 21억 4,643만 5,072(약 2GB)입니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
S3ResultBlockSize	선택 사항	`67108864`	`S3ResultBlockSize=268435456;`

문자열 열 길이

`string` 데이터 유형이 있는 열의 열 길이를 지정합니다. Athena는 정밀도가 정의되지 않은 Apache Hive 문자열 데이터 유형을 사용하기 때문에 Athena가 보고하는 기본 길이는 2147483647(`INT_MAX`)입니다. BI 도구는 대체로 열에 메모리를 미리 할당하기 때문에 메모리 사용량이 많아질 수 있습니다.

이를 방지하기 위해 Athena ODBC 드라이버는 `string` 데이터 유형의 열에 대해 보고되는 정밀도를 제한하고 기본값이 변경될 수 있도록 `StringColumnLength` 연결 파라미터를 노출합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
StringColumnLength	선택 사항	255	`StringColumnLength =65535;`

복합 유형 열 길이

`map`, `struct`, `array` 등의 복합 데이터 유형이 있는 열의 열 길이를 지정합니다. StringColumnLength와 마찬가지로 Athena는 복합 데이터 유형이 있는 열에 대해 정밀도 0을 보고합니다. Athena ODBC 드라이버는 복합 데이터 유형의 열에 대해 기본 정밀도를 설정하고 기본값이 변경될 수 있도록 `ComplexTypeColumnLength` 연결 파라미터를 노출합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
ComplexTypeColumnLength	선택 사항	65535	`ComplexTypeColumnLength=123456;`

신뢰할 수 있는 CA 인증서

HTTP 클라이언트에 SSL 인증서 신뢰 저장소를 찾을 위치를 지시합니다. 이 값은 `ClientConfiguration.caFile` 파라미터에 전달됩니다. 자세한 내용은 AWS SDK for C++ 개발자 안내서의 AWS Client configuration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
TrustedCerts	선택 사항	`%INSTALL_PATH%/bin`	`TrustedCerts=C:\\Program Files\\Amazon Athena ODBC Driver\\bin\\cacert.pem;`

최소 폴링 기간

쿼리 실행 상태에 대해 Athena를 폴링하기 전에 대기할 최솟값(밀리초)을 지정합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
MinQueryExecutionPollingInterval	선택 사항	100	MinQueryExecutionPollingInterval=200;

최대 폴링 기간

쿼리 실행 상태에 대해 Athena를 폴링하기 전에 대기할 최댓값(밀리초)을 지정합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
MaxQueryExecutionPollingInterval	선택 사항	60000	MaxQueryExecutionPollingInterval=1000;

폴링 승수

폴링 기간을 늘리는 요인을 지정합니다. 기본적으로 폴링은 최소 폴링 기간 값으로 시작되며 최대 폴링 기간 값에 도달할 때까지 폴링마다 두 배로 늘어납니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
QueryExecutionPollingIntervalMultiplier	선택 사항	2	QueryExecutionPollingIntervalMultiplier=2;

최대 폴링 기간

드라이버에서 쿼리 실행 상태에 대해 Athena를 폴링할 수 있는 최댓값(밀리초)을 지정합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
MaxPollDuration	선택 사항	1800000	MaxPollDuration=1800000;

연결 제한 시간

HTTP 연결 시 연결이 설정될 때까지 기다리는 시간(밀리초)입니다. 이 값은 `ClientConfiguration.connectTimeoutMs` Athena 클라이언트에 대해 설정됩니다. 지정하지 않은 경우 curl 기본값이 사용됩니다. 연결 파라미터에 대한 자세한 내용은 AWS SDK for Java 개발자 안내서의 Client Configuration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
ConnectionTimeout	선택 사항	`0`	`Connectio nTimeout=2000;`

요청 제한 시간

HTTP 클라이언트의 소켓 읽기 제한 시간을 지정합니다. 이 값은 Athena 클라이언트의 `ClientConfiguration.requestTimeoutMs` 파라미터에 대해 설정됩니다. 파라미터 정보는 AWS SDK for Java 개발자 안내서의 Client Configuration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
RequestTimeout	선택 사항	`10000`	`RequestTi meout=30000;`

프록시 옵션

프록시 호스트

사용자가 프록시를 거쳐야 하는 경우 이 파라미터를 사용하여 프록시 호스트를 설정합니다. 이는 AWS SDK의 `ClientConfiguration.proxyHost` 파라미터에 해당됩니다. 자세한 내용은 AWS SDK for C++ 개발자 안내서의 AWS Client configuration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
ProxyHost	선택 사항	none	`ProxyHost =127.0.0.1;`

프록시 포트

이 파라미터를 사용하여 프록시 포트를 설정합니다. 이는 AWS SDK의 `ClientConfiguration.proxyPort` 파라미터에 해당됩니다. 자세한 내용은 AWS SDK for C++ 개발자 안내서의 AWS Client configuration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
ProxyPort	선택 사항	none	`ProxyPort=8888;`

프록시 사용자 이름

이 파라미터를 사용하여 프록시 사용자 이름을 설정합니다. 이는 AWS SDK의 `ClientConfiguration.proxyUserName` 파라미터에 해당됩니다. 자세한 내용은 AWS SDK for C++ 개발자 안내서의 AWS Client configuration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
ProxyUID	선택 사항	none	`ProxyUID=username;`

프록시 비밀번호

이 파라미터를 사용하여 프록시 암호를 설정합니다. 이는 AWS SDK의 `ClientConfiguration.proxyPassword` 파라미터에 해당됩니다. 자세한 내용은 AWS SDK for C++ 개발자 안내서의 AWS Client configuration을 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
ProxyPWD	선택 사항	none	`ProxyPWD=password;`

비프록시 호스트

이 선택적 파라미터를 사용하여 드라이버가 프록시를 사용하지 않고 연결하는 호스트를 지정합니다. 이는 AWS SDK의 `ClientConfiguration.nonProxyHosts` 파라미터에 해당됩니다. 자세한 내용은 AWS SDK for C++ 개발자 안내서의 AWS Client configuration을 참조하세요.

NonProxyHost 연결 파라미터는 CURLOPT_NOPROXY curl 옵션에 전달됩니다. CURLOPT_NOPROXY 형식에 대한 자세한 내용은 curl 문서에서 CURLOPT_NOPROXY를 참조하세요.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
NonProxyHost	선택 사항	none	`NonProxyHost=.amazonaws.com,localhost,.example.net,.example.com;`

프록시 사용

지정된 프록시를 통한 사용자 트래픽을 활성화합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
UseProxy	선택 사항	none	`UseProxy=1;`

로깅 옵션

여기에서 설명하는 설정을 수정하려면 관리자 권한이 필요합니다. 변경하려는 경우 ODBC 데이터 원본 관리자 로깅 옵션 대화 상자를 사용하거나 Windows 레지스트리를 직접 수정할 수 있습니다.

로그 수준

이 옵션은 ODBC 드라이버 로그를 활성화합니다. Windows에서는 레지스트리 또는 대화 상자를 사용하여 로깅을 활성화 또는 비활성화할 수 있습니다. 옵션은 다음 레지스트리 경로에 있습니다.

```
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Amazon Athena\ODBC\Driver
```

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
LogLevel	선택 사항	0	`LogLevel=1;`

로그 경로

ODBC 드라이버 로그가 저장되는 파일의 경로를 지정합니다. 레지스트리 또는 대화 상자를 사용하여 이 값을 설정할 수 있습니다. 옵션은 다음 레지스트리 경로에 있습니다.

```
Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Amazon Athena\ODBC\Driver
```

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
LogPath	선택 사항	none	LogPath=C:\Users\ *username*\projects\internal\trunk\;

AWS 로거 사용

AWS SDK 로깅을 활성화할지 여부를 지정합니다. 활성화하려면 1을 지정하고 비활성화하려면 0을 지정합니다.

연결 문자열 이름	파라미터 유형	기본값	연결 문자열 예제
UseAwsLogger	선택 사항	0	UseAwsLogger=1;

ODBC 2.x 드라이버로 마이그레이션

대부분의 Athena ODBC 2.x 연결 파라미터는 ODBC 1.x 드라이버와 역호환되므로 대부분의 기존 연결 문자열을 Athena ODBC 2.x 드라이버에서 재사용할 수 있습니다. 하지만 다음 연결 파라미터는 수정이 필요합니다.

로그 수준

현재 ODBC 드라이버는 LOG_OFF (0)~LOG_TRACE (6)까지 다양한 사용 가능한 로깅 옵션을 제공하지만 Amazon Athena ODBC 드라이버에는 0(비활성화됨) 및 1(활성화됨)의 두 가지 값만 있습니다.

ODBC 2.x 드라이버 로깅에 대한 자세한 내용은 로깅 옵션 섹션을 참조하세요.

	ODBC 1.x 드라이버	ODBC 2.x 드라이버
연결 문자열 이름	LogLevel	LogLevel
파라미터 유형	선택 사항	선택 사항
기본값	0	0
가능한 값	0-6	0,1
연결 문자열 예제	LogLevel=6;	LogLevel=1;

MetadataRetrievalMethod

현재 ODBC 드라이버는 Athena에서 메타데이터를 검색하는 몇 가지 옵션을 제공합니다. Amazon Athena ODBC 드라이버는 MetadataRetrievalMethod를 더 이상 사용하지 않으며 항상 Amazon Athena API를 사용하여 메타데이터를 추출합니다.

Athena는 외부 카탈로그 쿼리를 위해 QueryExternalCatalogs 플래그를 도입했습니다. 현재 ODBC 드라이버로 외부 카탈로그를 쿼리하려면 MetadataRetrievalMethod를 ProxyAPI로 설정합니다. Athena ODBC 드라이버로 외부 카탈로그를 쿼리하려면 QueryExternalCatalogs를 1로 설정합니다.

	ODBC 1.x 드라이버	ODBC 2.x 드라이버
연결 문자열 이름	MetadataRetrievalMethod	QueryExternalCatalogs
파라미터 유형	선택 사항	선택 사항
기본값	Auto	0
가능한 값	Auto, AWS Glue, ProxyAPI, Query	0,1
연결 문자열 예제	MetadataRetrievalMethod=ProxyAPI;	QueryExternalCatalogs=1;

연결 테스트

ODBC 1.x 드라이버 연결을 테스트할 때 드라이버는 Amazon S3 버킷에서 두 개의 파일(결과 세트에 대한 파일 및 메타데이터에 대한 파일)을 생성하는 `SELECT 1` 쿼리를 실행합니다. 테스트 연결 요금은 Amazon Athena 요금 정책에 따라 부과됩니다.

ODBC 2.x 드라이버 연결을 테스트하는 경우 드라이버는 GetWorkGroup Athena API 작업을 직접적으로 호출합니다. 직접 호출은 보안 인증 정보를 검색하기 위해 지정한 인증 유형 및 해당 보안 인증 제공업체를 사용합니다. ODBC 2.x 드라이버를 사용하는 경우 연결 테스트에 요금이 부과되지 않으며, 테스트할 때 Amazon S3 버킷에 쿼리 결과가 생성되지 않습니다.

ODBC 2.x 드라이버 문제 해결

Amazon Athena ODBC 드라이버 관련 문제가 발생하는 경우 AWS Support에 문의할 수 있습니다 (AWS Management Console에서 지원, Support Center 선택).

이때 다음 정보를 포함하고 지원팀이 사용 사례를 이해하는 데 도움이 되는 추가 세부 정보를 제공해야 합니다.

- 설명 - (필수) 사용 사례에 대한 자세한 정보와 예상 동작과 관찰된 동작 간 차이를 포함하는 설명입니다. 지원 엔지니어가 문제를 쉽게 탐색할 수 있도록 모든 정보를 포함합니다. 문제가 간헐적으로 발생하는 경우 문제가 발생한 날짜, 타임스탬프 또는 간격을 명시합니다.
- 버전 정보 - (필수) 드라이버 버전, 운영 체제 및 사용한 애플리케이션에 대한 정보입니다. 예: 'ODBC 드라이버 버전 1.2.3, Windows 10(x64), Power BI'.
- 로그 파일 - (필수) 문제를 이해하는 데 필요한 최소한의 ODBC 드라이버 로그 파일입니다. ODBC 2.x 드라이버의 로깅 옵션에 대한 자세한 내용은 로깅 옵션 섹션을 참조하세요.
- 연결 문자열 - (필수) ODBC 연결 문자열 또는 사용한 연결 파라미터를 보여주는 대화 상자의 스크린샷입니다. 연결 파라미터에 대한 자세한 내용은 Athena ODBC 2.x 연결 파라미터 섹션을 참조하세요.
- 문제 단계 - (선택 사항) 가능하면 문제를 재현하는 데 도움이 되는 단계 또는 독립 실행형 프로그램을 포함합니다.
- 쿼리 오류 정보 - (선택 사항) DML 또는 DDL 쿼리와 관련된 오류인 경우 다음 정보를 포함합니다.
  - 실패한 DML 또는 DDL 쿼리의 전체 또는 단순화된 버전.
  - 사용한 계정 ID 및 AWS 리전, 쿼리 실행 ID.
- SAML 오류 - (선택 사항) SAML 어설션을 사용한 인증과 관련된 문제인 경우 다음 정보를 포함합니다.

- 사용한 ID 제공업체 및 인증 플러그인.
- SAML 토큰을 사용한 예제.

Amazon Athena ODBC 2.x 릴리스 정보

이 릴리스 정보에서는 Amazon Athena ODBC 2.x 드라이버의 개선 사항, 기능, 알려진 문제 및 워크플로 변경 사항에 대한 세부 정보를 제공합니다.

2.0.3.0

2024년 4월 8일 릴리스

Amazon Athena ODBC v2.0.3.0 드라이버에는 다음과 같은 개선 사항 및 수정 사항이 포함되어 있습니다.

개선 사항

- Linux 및 Mac 플랫폼에서 Okta 인증 플러그인에 대한 MFA 지원을 추가했습니다.
- 이제 Windows에 대해 `athena-odbc.dll` 라이브러리와 `AmazonAthenaODBC-2.x.x.x.msi` 설치 프로그램이 모두 서명됩니다.
- 드라이버와 함께 설치된 CA 인증서 `cacert.pem` 파일을 업데이트했습니다.
- Lambda 카탈로그에 테이블을 나열하는 데 필요한 시간을 개선했습니다. `LAMBDA` 카탈로그 유형의 경우 ODBC 드라이버가 이제 SHOW TABLES 쿼리를 제출하여 사용 가능한 테이블 목록을 가져올 수 있습니다. 자세한 내용은 쿼리를 사용하여 테이블 나열 단원을 참조하십시오.
- `SQL_WCHAR` 및 `SQL_WVARCHAR`를 사용하여 문자열 데이터 유형을 보고하는 `UseWCharForStringTypes` 연결 파라미터를 도입했습니다. 자세한 내용은 문자열 유형에 WCHAR 사용 단원을 참조하십시오.

수정 사항

- Get-OdbcDsn PowerShell 도구를 사용할 때 발생했던 레지스트리 손상 경고를 수정했습니다.
- 쿼리 문자열 시작 시 주석을 처리하도록 구문 분석 로직을 업데이트했습니다.
- 날짜 및 타임스탬프 데이터 유형에서 이제 연도 필드에 0을 사용할 수 있습니다.

새 ODBC v2 드라이버를 다운로드하려면 ODBC 2.x 드라이버 다운로드 섹션을 참조하세요. 연결 정보는 Amazon Athena ODBC 2.x 섹션을 참조하세요.

2.0.2.2

2024년 2월 13일 릴리스

Amazon Athena ODBC v2.0.2.2 드라이버에는 다음과 같은 개선 사항 및 수정 사항이 포함되어 있습니다.

개선 사항

- 문자열 및 복합 데이터 유형의 기본 열 길이를 변경하는 데 사용할 수 있는 연결 파라미터인 `StringColumnLength` 및 `ComplexTypeColumnLength`를 추가했습니다. 자세한 내용은 문자열 열 길이 및 복합 유형 열 길이 단원을 참조하세요.
- Linux 및 macOS (Intel 및 ARM) 운영 체제에 대한 지원이 추가되었습니다. 자세한 내용은 Linux 및 macOS 단원을 참조하세요.
- AWS-SDK-CPP가 1.11.245 태그 버전으로 업데이트되었습니다.
- curl 라이브러리가 8.6.0 버전으로 업데이트되었습니다.

수정 사항

- 정밀도 열의 문자열과 유사한 데이터 유형에 대한 결과 세트 메타데이터에서 잘못된 값이 보고되는 문제를 해결했습니다.

ODBC v2 드라이버를 다운로드하려면 ODBC 2.x 드라이버 다운로드 섹션을 참조하십시오. 연결 정보는 Amazon Athena ODBC 2.x 섹션을 참조하세요.

2.0.2.1

2023년 12월 7일 출시

Amazon Athena ODBC v2.0.2.1 드라이버에는 다음과 같은 개선 사항 및 수정 사항이 포함되어 있습니다.

개선 사항

- 모든 인터페이스의 ODBC 드라이버 스레드 안전이 개선되었습니다.
- 로깅이 활성화되면 이제 날짜 시간 값이 밀리초 단위로 기록됩니다.
- 브라우저 SSO OIDC 플러그인으로 인증하는 동안 이제 터미널이 열려 사용자에게 디바이스 코드가 표시됩니다.

수정 사항

- 스트리밍 API의 결과를 구문 분석할 때 발생하는 메모리 릴리스 문제가 해결되었습니다.
- 이제 `SQLTablePrivileges()`, `SQLSpecialColumns()`, `SQLProcedureColumns()` 및 `SQLProcedures()` 인터페이스에 대한 요청이 빈 결과 세트를 반환합니다.

ODBC v2 드라이버를 다운로드하려면 ODBC 2.x 드라이버 다운로드 섹션을 참조하십시오. 연결 정보는 Amazon Athena ODBC 2.x 섹션을 참조하세요.

2.0.2.0

2023년 10월 17일 출시

Amazon Athena ODBC v2.0.2.0 드라이버에는 다음과 같은 개선 사항 및 수정 사항이 포함되어 있습니다.

개선 사항

- 브라우저 Azure AD, 브라우저 SSO OIDC 및 Okta 브라우저 기반 인증 플러그인에 파일 캐시 기능이 추가되었습니다.

  Power BI와 같은 BI 도구 및 브라우저 기반 플러그인은 여러 브라우저 창을 사용합니다. 새 파일 캐시 연결 파라미터를 사용하면 임시 보안 인증을 캐시하여 BI 애플리케이션에서 여는 여러 프로세스 간에 재사용할 수 있습니다.
- 이제 애플리케이션에서 명령문이 준비된 후 결과 세트에 대한 정보를 쿼리할 수 있습니다.
- 느린 클라이언트 네트워크에서 사용할 수 있도록 기본 연결 및 요청 시간 제한이 늘어났습니다. 자세한 내용은 연결 제한 시간 및 요청 제한 시간 단원을 참조하세요.
- SSO 및 SSO OIDC에 대한 엔드포인트 재정의가 추가되었습니다. 자세한 내용은 엔드포인트 재정의 단원을 참조하십시오.
- 인증 요청을 위한 URI 인수를 Ping에 전달하는 연결 파라미터가 추가되었습니다. 이 파라미터를 사용하여 Lake Formation 단일 역할 제한을 우회할 수 있습니다. 자세한 내용은 Ping URI 파라미터 단원을 참조하십시오.

수정 사항

- 행 기반 바인딩 메커니즘을 사용할 때 발생하는 정수 오버플로 문제가 수정되었습니다.
- 브라우저 SSO OIDC 인증 플러그인의 필수 연결 파라미터 목록에서 제한 시간이 제거되었습니다.

- `SQLStatistics()`, `SQLPrimaryKeys()`, `SQLForeignKeys()` 및 `SQLColumnPrivileges()`에 대한 누락된 인터페이스가 추가되고 요청 시 빈 결과 세트를 반환하는 기능이 추가되었습니다.

새 ODBC v2 드라이버를 다운로드하려면 ODBC 2.x 드라이버 다운로드 섹션을 참조하세요. 연결 정보는 Amazon Athena ODBC 2.x 섹션을 참조하세요.

2.0.1.1

2023년 8월 10일 출시

Amazon Athena ODBC v2.0.1.1 드라이버에는 다음과 같은 개선 사항 및 수정 사항이 포함되어 있습니다.

개선 사항

- Okta 인증 플러그인에 URI 로깅을 추가했습니다.
- 외부 보안 인증 제공업체 플러그인에 기본 역할 파라미터를 추가했습니다.
- AWS 구성 파일의 프로파일 이름에서 프로파일 접두사에 대한 처리를 추가했습니다.

수정 사항

- Lake Formation 및 AWS STS 클라이언트에서 작업할 때 발생한 AWS 리전 사용 문제를 수정했습니다.
- 누락된 파티션 키를 테이블 열 목록에 복원했습니다.
- 누락된 `BrowserSSOOIDC` 인증 유형을 AWS 프로파일에 추가했습니다.

새 ODBC v2 드라이버를 다운로드하려면 ODBC 2.x 드라이버 다운로드 섹션을 참조하세요.

2.0.1.0

2023년 6월 29일 출시

Amazon Athena가 ODBC v2.0.1.0 드라이버를 출시합니다.

Athena에서 호환되는 SQL 개발 및 비즈니스 인텔리전스 애플리케이션의 데이터에 연결하고 쿼리하며 시각화하는 경험을 개선하는 새로운 ODBC 드라이버를 출시했습니다. Athena ODBC 드라이버의 최신 버전은 기존 드라이버의 기능을 지원하면서 업그레이드도 간단합니다. 새 버전에는 AWS IAM Identity

Center를 통한 사용자 인증 지원이 포함되어 있습니다. 또한 Amazon S3에서 쿼리 결과를 읽는 옵션을 제공하므로 쿼리 결과를 더 빨리 사용할 수 있습니다.

자세한 내용은 Amazon Athena ODBC 2.x 단원을 참조하십시오.

Athena ODBC 1.x 드라이버

이 페이지의 링크를 사용하여 Amazon Athena 1.x ODBC 드라이버 라이선스 계약, ODBC 드라이버 및 ODBC 설명서를 다운로드합니다. ODBC 연결 문자열에 대한 자세한 내용은 이 페이지에서 다운로드할 수 있는 ODBC 드라이버 설치 및 구성 안내서 PDF 파일을 참조하세요. 권한에 대한 자세한 내용은 JDBC 및 ODBC 연결을 통한 액세스 단원을 참조하세요.

Important

ODBC 1.x 드라이버를 사용할 때는 다음 요구 사항에 유의하세요.

- 포트 444 개방 – Athena가 쿼리 결과를 스트리밍하는 데 사용하는 포트 444를 유지하고 아웃바운드 트래픽에 개방합니다. PrivateLink 엔드포인트를 사용하여 Athena에 연결하는 경우, PrivateLink 엔드포인트에 연결된 보안 그룹이 포트 444의 인바운드 트래픽에 개방되어 있는지 확인합니다.
- athena:GetQueryResultsStream 정책 – ODBC 드라이버를 사용하는 IAM 보안 주체에 대한 `athena:GetQueryResultsStream` 정책 작업을 추가합니다. 이 정책 작업은 API를 통해 직접 노출되지 않습니다. 이 정책 작업은 스트리밍 결과 지원의 일부로 ODBC 및 JDBC 드라이버에만 사용됩니다. 정책 예제는 AWS 관리형 정책: AWSQuicksightAthenaAccess을 참조하세요.

Windows

드라이버 버전	다운로드 링크
Windows 32비트용 ODBC 1.2.3.1000	Windows 32비트 ODBC 드라이버 1.2.3.1000
Windows 64비트용 ODBC 1.2.3.1000	Windows 64비트 ODBC 드라이버 1.2.3.1000

Linux

드라이버 버전	다운로드 링크
Linux 32비트용 ODBC 1.2.3.1000	Linux 32비트 ODBC 드라이버 1.2.3.1000
Linux 64비트용 ODBC 1.2.3.1000	Linux 64비트 ODBC 드라이버 1.2.3.1000

OSX

드라이버 버전	다운로드 링크
OSX용 ODBC 1.2.3.1000	OSX ODBC 드라이버 1.2.3.1000

설명서

내용	설명서 링크
Amazon Athena ODBC 드라이버 라이선스 계약	라이선스 계약
ODBC 1.2.3.1000용 설명서	ODBC 드라이버 설치 및 구성 가이드 버전 1.2.3.1000
ODBC 1.2.3.1000용 릴리스 정보	ODBC 드라이버 릴리스 정보 버전 1.2.3.1000

ODBC 드라이버 참고

프록시를 사용하지 않고 연결

드라이버가 프록시를 사용하지 않고 연결하는 특정 호스트를 지정하려면 ODBC 연결 문자열에 선택 사항인 `NonProxyHost` 속성을 사용할 수 있습니다.

`NonProxyHost` 속성은 다음 예에서와 같이 프록시 연결이 활성화된 경우 커넥터가 프록시 서버를 통과하지 않고 액세스할 수 있는 호스트의 쉼표로 구분된 목록을 지정합니다.

```
.amazonaws.com,localhost,.example.net,.example.com
```

NonProxyHost 연결 파라미터는 CURLOPT_NOPROXY curl 옵션에 전달됩니다. CURLOPT_NOPROXY 형식에 대한 자세한 내용은 curl 문서에서 CURLOPT_NOPROXY를 참조하세요.

ODBC 클라이언트를 사용하여 Microsoft AD FS 사용자를 위해 Amazon Athena에 대한 페더레이션 액세스 구성

ODBC 클라이언트를 사용하여 Microsoft Active Directory Federation Services(AD FS) 사용자를 위해 Amazon Athena에 대한 페더레이션 액세스를 설정하려면 먼저 AD FS와 AWS 계정 간에 신뢰를 구축합니다. 이 신뢰가 구축되면 AD 사용자는 AD 보안 인증을 사용하여 AWS에 페더레이션하고 AWS Identity and Access Management(IAM) 역할의 권한을 수임하여 Athena API와 같은 AWS 리소스에 액세스할 수 있습니다.

이 신뢰를 구축하려면 AD FS를 AWS 계정에 SAML 공급자로 추가하고 페더레이션 사용자가 맡을 수 있는 IAM 역할을 생성합니다. AD FS 측에서 AWS를 신뢰 당사자로 추가하고 승인(특히 Athena 및 Amazon S3)을 위해 AWS에 올바른 사용자 특성을 전송하도록 SAML 클레임 규칙을 작성합니다.

Athena에 대한 AD FS 액세스를 구성하는 주요 단계는 다음과 같습니다.

1. IAM SAML 공급자 및 역할 설정

2. AD FS 구성

3. Active Directory 사용자 및 그룹 생성

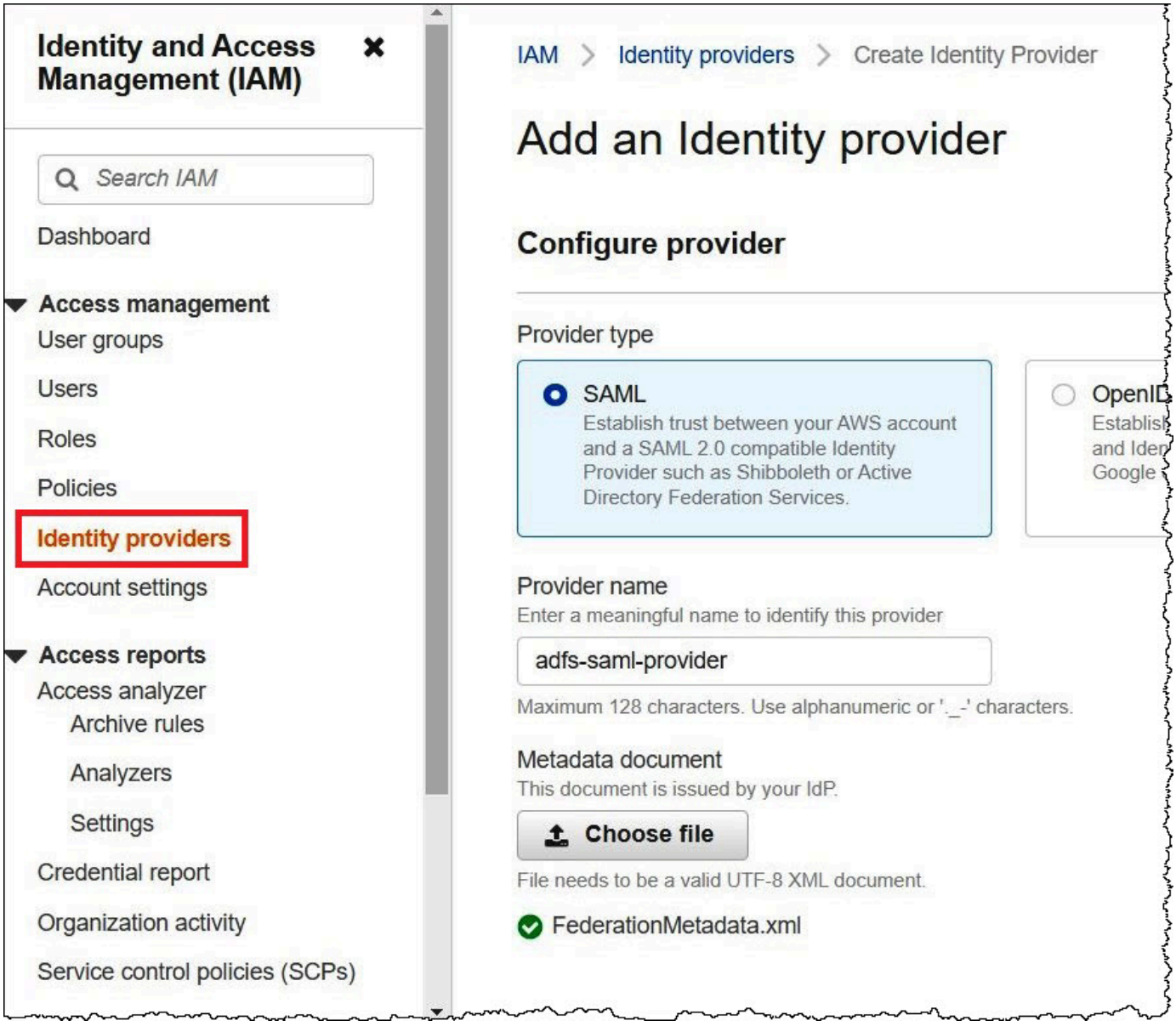
4. Athena에 대한 AD FS ODBC 연결 구성

1. IAM SAML 공급자 및 역할 설정

이 단원에서는 AD FS를 AWS 계정에 SAML 공급자로 추가하고 페더레이션 사용자가 맡을 수 있는 IAM 역할을 생성합니다.

SAML 공급자를 설정하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 [자격 증명 공급자(Identity providers)]를 선택합니다.
3. 공급자 추가(Add Provider)를 선택합니다.
4. 공급자 유형(Provider type)에 SAML을 선택합니다.



5. 공급자 이름(Provider name)에 **`adfs-saml-provider`**를 입력합니다.
6. 브라우저에서 다음 주소를 입력하여 AD FS 서버의 페더레이션 XML 파일을 다운로드합니다. 이 단계를 수행하려면 브라우저에서 AD FS 서버에 액세스할 수 있어야 합니다.

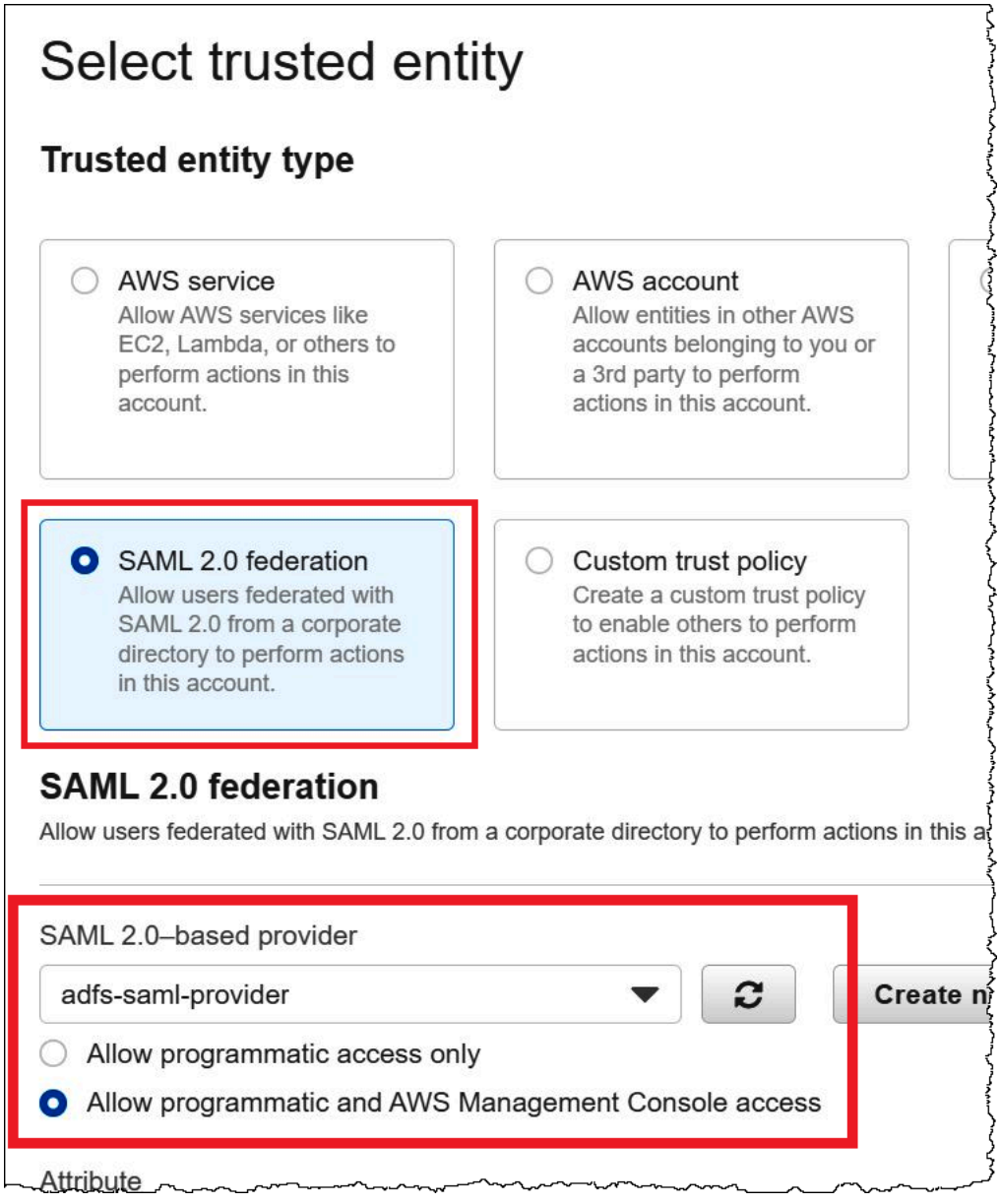
```
https://adfs-server-name/federationmetadata/2007-06/federationmetadata.xml
```

7. IAM 콘솔의 Metadata document(메타데이터 문서)에서 Choose file(파일 선택)을 선택한 다음 페더레이션 메타데이터 파일을 AWS에 업로드합니다.
8. 완료하려면 Add provider(공급자 추가)를 선택합니다.

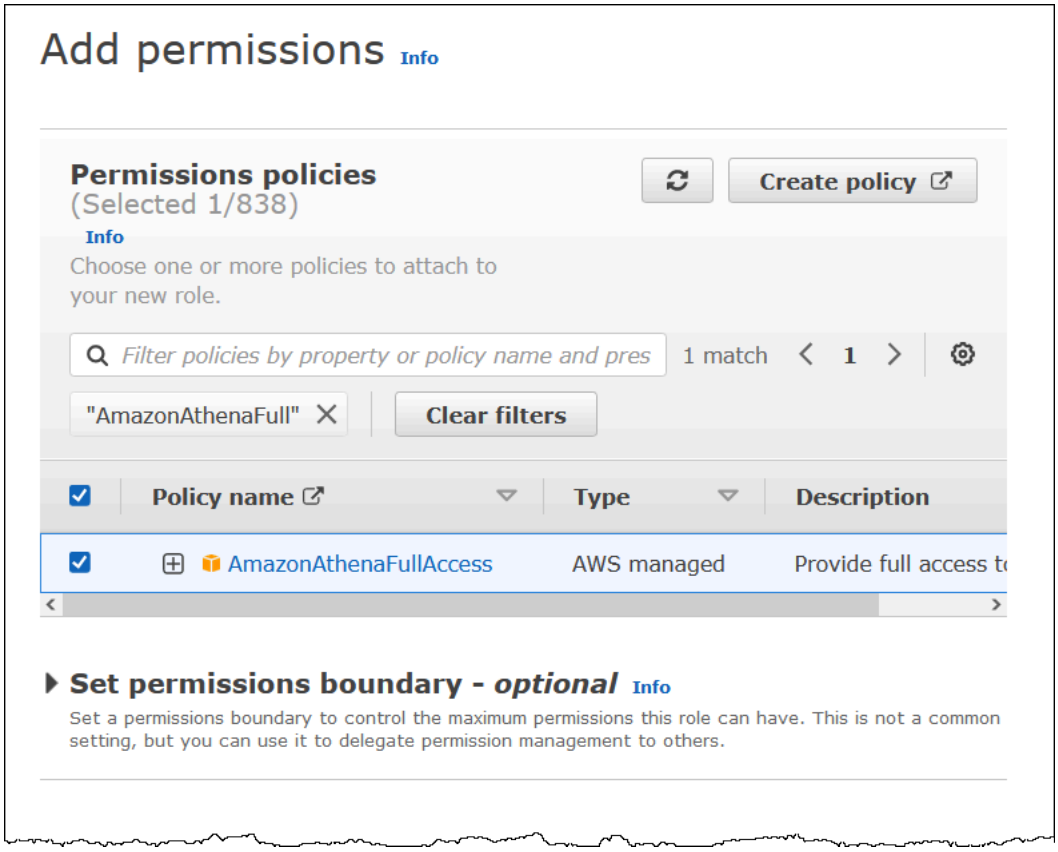
그런 다음 페더레이션 사용자가 수임할 수 있는 IAM 역할을 생성합니다.

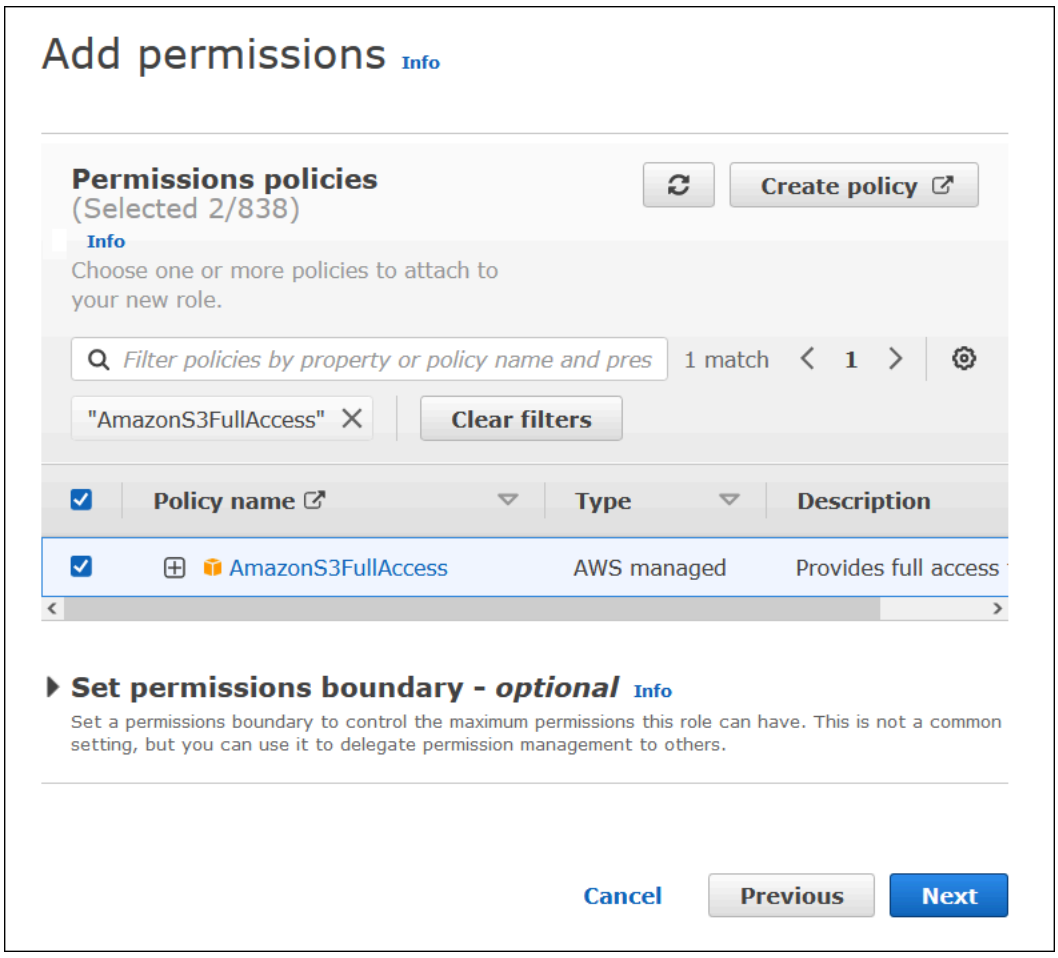
페더레이션 사용자에 대한 IAM 역할을 생성하려면

1. IAM 콘솔의 탐색 창에서 Roles(역할)를 선택합니다.
2. 역할 생성을 선택합니다.
3. Trusted entity type(신뢰할 수 있는 엔터티 유형)에서 SAML 2.0 Federation(SAML 2.0 페더레이션)을 선택합니다.
4. SAML 2.0-based provider(SAML 2.0 기반 공급자)의 경우 생성한 adfs-saml-provider 공급자를 선택합니다.
5. 프로그래밍 방식 및 AWS Management Console 액세스 허용을 선택한 후 다음을 선택합니다.



6. Add permissions(권한 추가) 페이지에서 이 역할에 필요한 IAM 권한 정책을 필터링한 다음 해당 확인란을 선택합니다. 이 자습서에서는 `AmazonAthenaFullAccess` 및 `AmazonS3FullAccess` 정책을 연결합니다.



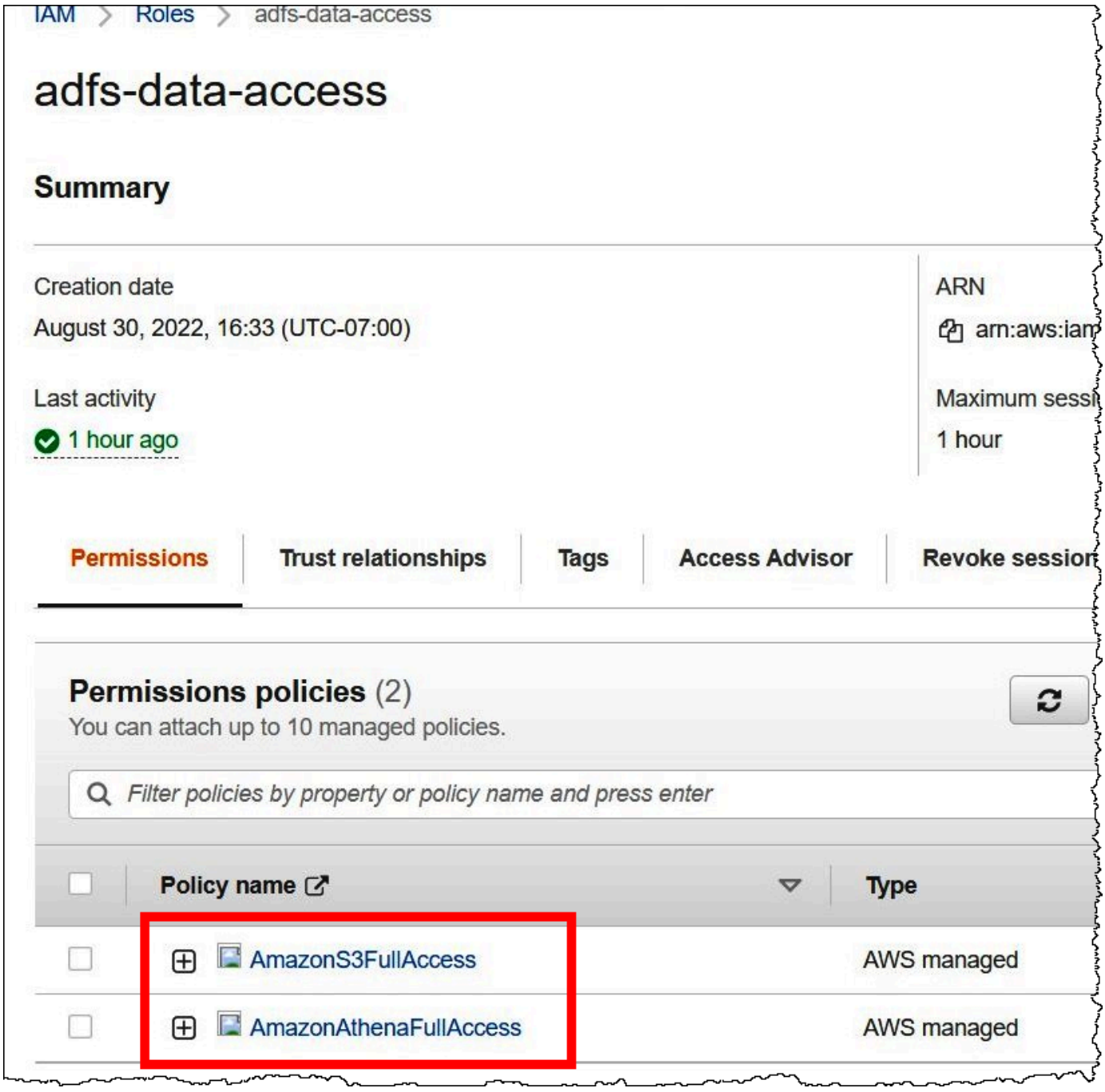


7. 다음을 선택합니다.
8. Name, review, and create(이름, 검토 및 생성) 페이지에서 Role name(역할 이름)에 역할의 이름을 입력합니다. 이 자습서에서는 adfs-data-access라는 이름을 사용합니다.

   Step 1: Select trusted entities(1단계: 신뢰할 수 있는 엔터티 선택)에서 Principal(보안 주체) 필드에 `"Federated:" "arn:aws:iam::account_id:saml-provider/adfs-saml-provider"`가 자동으로 채워져야 합니다. `Condition` 필드에는 `"SAML:aud"` 및 `"https://signin.aws.amazon.com/saml"`이 포함되어야 합니다.

Step 2: Add permissions(2단계: 권한 추가)에서는 역할에 연결된 정책을 보여줍니다.

9. 역할 생성을 선택합니다. 역할 생성을 확인하는 배너 메시지가 표시됩니다.
10. Roles(역할) 페이지에서 방금 생성한 역할의 이름을 선택합니다. 역할에 대한 요약 페이지에는 연결된 정책이 표시됩니다.



2. AD FS 구성

이제 AWS를 신뢰 당사자로 추가하고 SAML 클레임 규칙을 작성하여 인증을 위해 올바른 사용자 특성을 AWS에 전송할 수 있습니다.

SAML 기반 페더레이션에는 IdP(Active Directory) 및 신뢰 당사자(AWS)라는 두 참가 당사자가 있습니다. 이는 IdP에서 인증을 사용하는 서비스 또는 애플리케이션입니다.

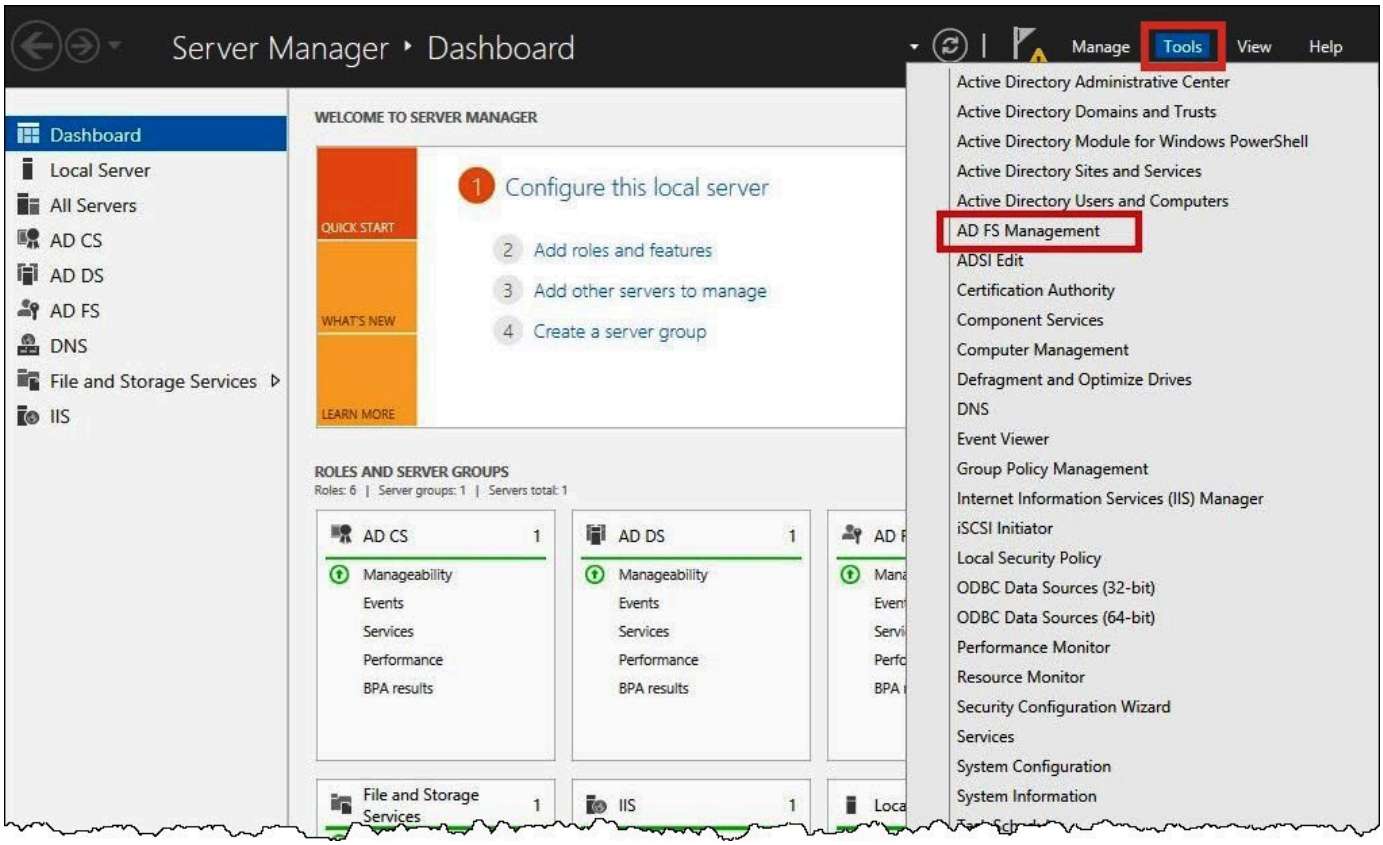
AD FS를 구성하려면 먼저 신뢰 당사자 신뢰를 추가한 다음 신뢰 당사자에 대한 SAML 클레임 규칙을 구성합니다. AD FS는 클레임 규칙을 사용하여 신뢰 당사자에게 전송되는 SAML 어설션을 형성합니다. SAML 어설션에는 AD 사용자에 대한 정보가 참이며 사용자를 인증했다고 명시되어 있습니다.

신뢰 당사자 신뢰 추가

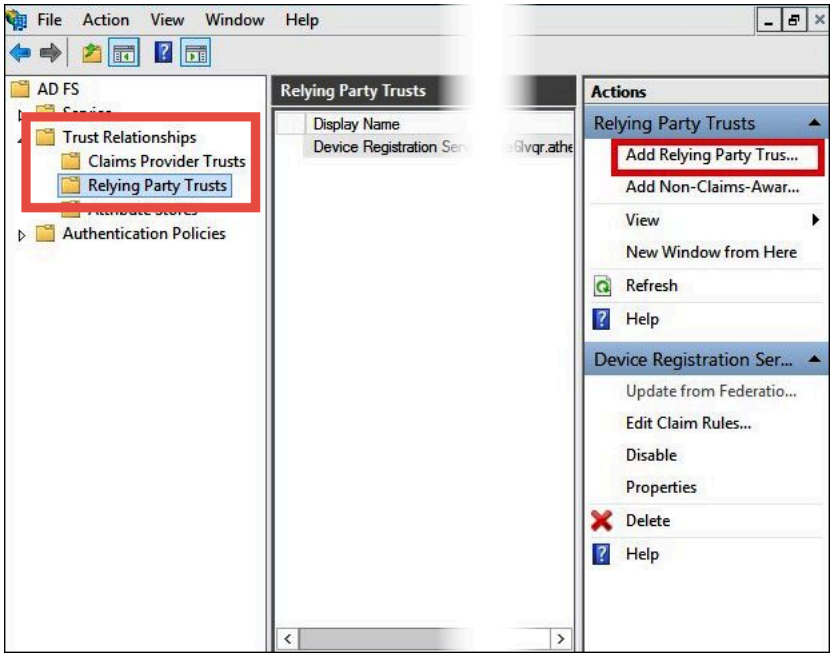
AD FS에서 신뢰 당사자 신뢰를 추가하려면 AD FS 서버 관리자를 사용합니다.

AD FS에서 신뢰 당사자 신뢰를 추가하려면

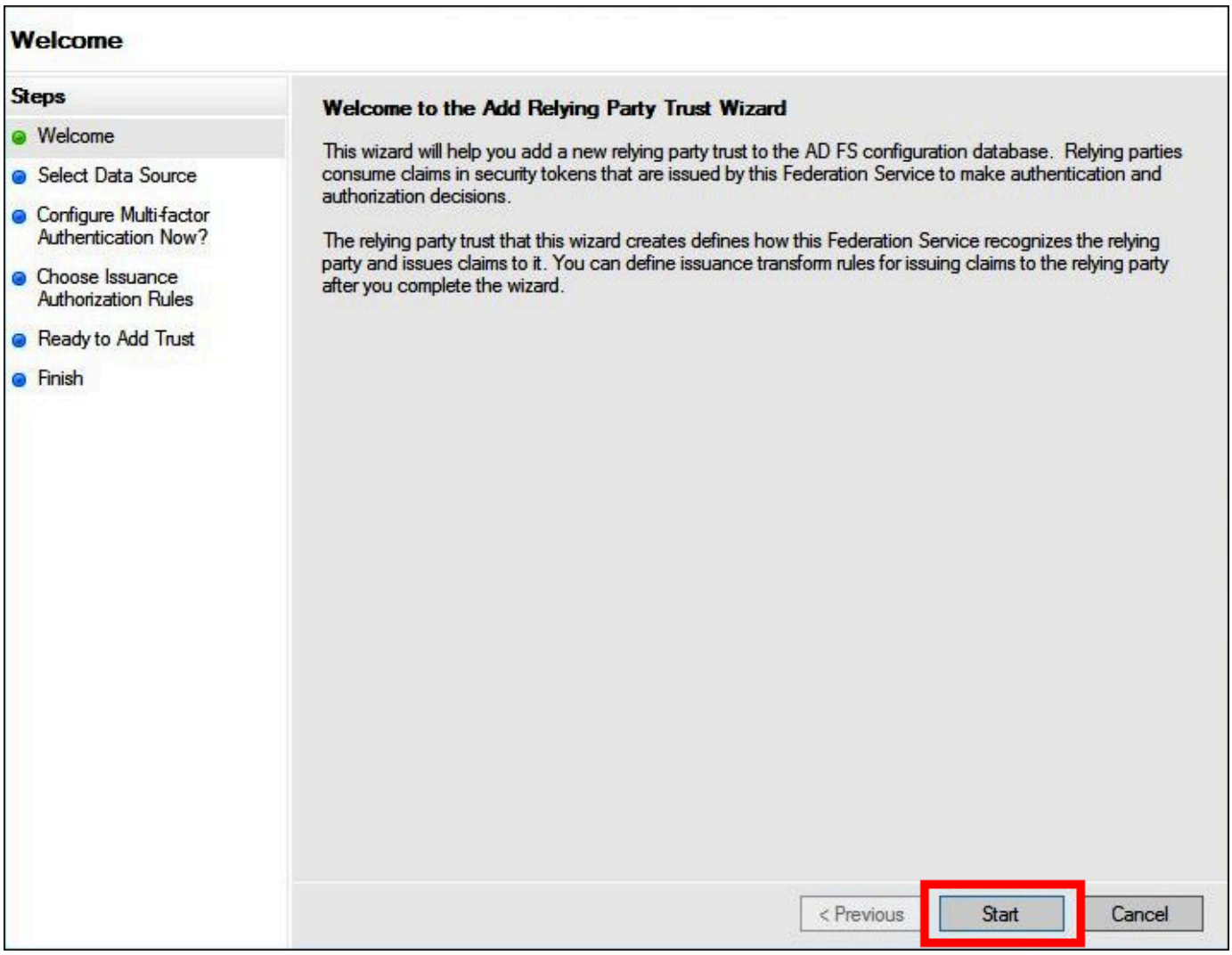
1. AD FS 서버에 로그인합니다.
2. Start(시작) 메뉴에서 Server Manager(서버 관리자)를 엽니다.
3. Tools(도구)를 선택한 다음 AD FS Management(AD FS 관리)를 선택합니다.



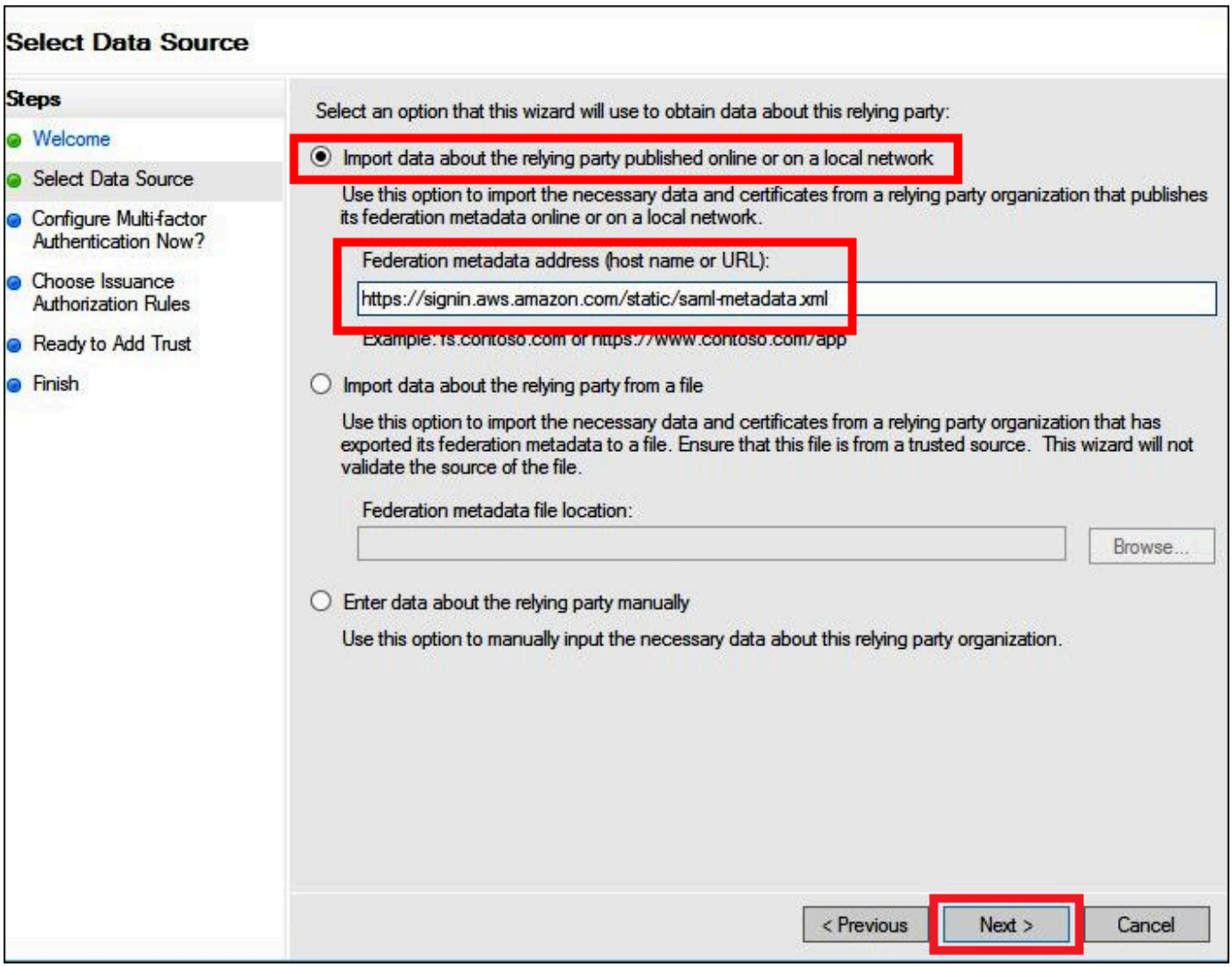
4. 탐색 창의 Trust Relationships(신뢰 관계)에서 Relying Party Trusts(신뢰 당사자 신뢰)를 선택합니다.
5. Actions(작업)에서 Add Relying Party Trust(신뢰 당사자 신뢰 추가)를 선택합니다.



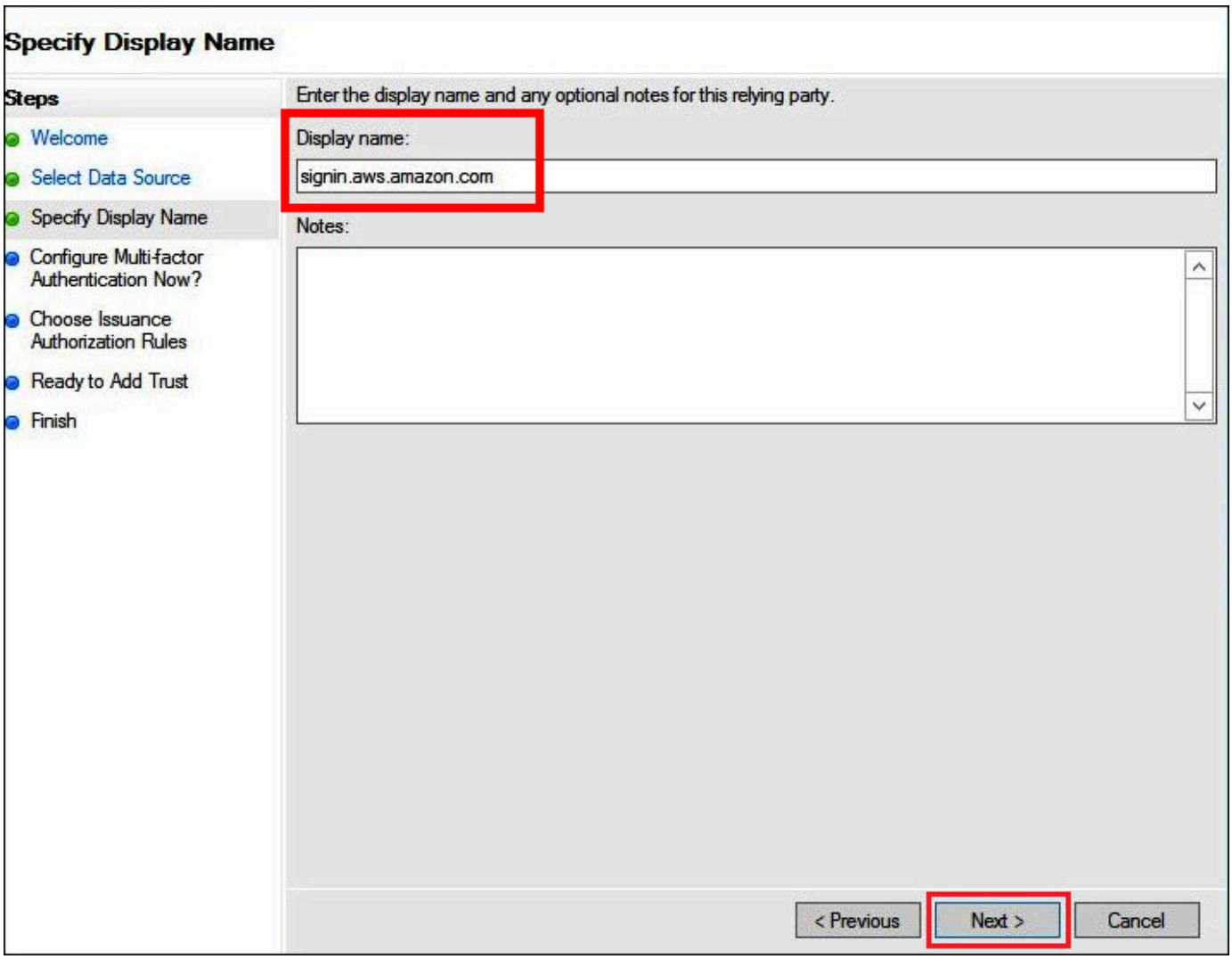
6. Add Relying Party Trust Wizard(신뢰 당사자 신뢰 추가 마법사) 페이지에서 시작을 선택합니다.



7. Select Data Source(데이터 소스 선택) 화면에서 Import data about the relying party published online or on a local network(온라인 또는 로컬 네트워크에 게시된 신뢰 당사자에 대한 데이터 가져오기) 옵션을 선택합니다.
8. Federation metadata address (host name or URL)(연동 메타데이터 주소(호스트 이름 또는 URL))에 **`https://signin.aws.amazon.com/static/saml-metadata.xml`** URL을 입력합니다.
9. [Next]를 선택합니다.

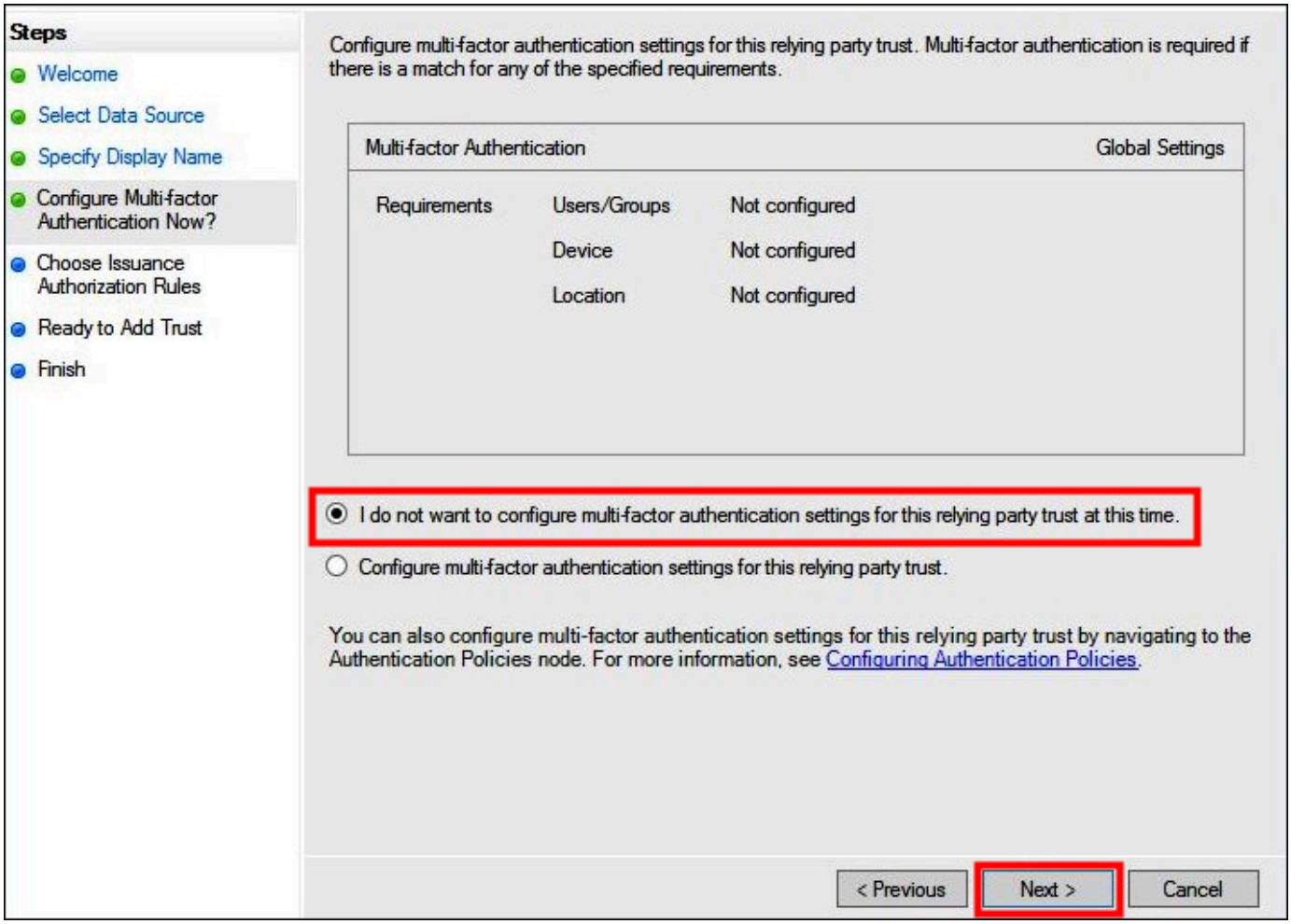


10. Specify Display Name(표시 이름 지정) 페이지의 Display name(표시 이름)에 신뢰 당사자에 대한 표시 이름을 입력한 후 Next(다음)를 선택합니다.



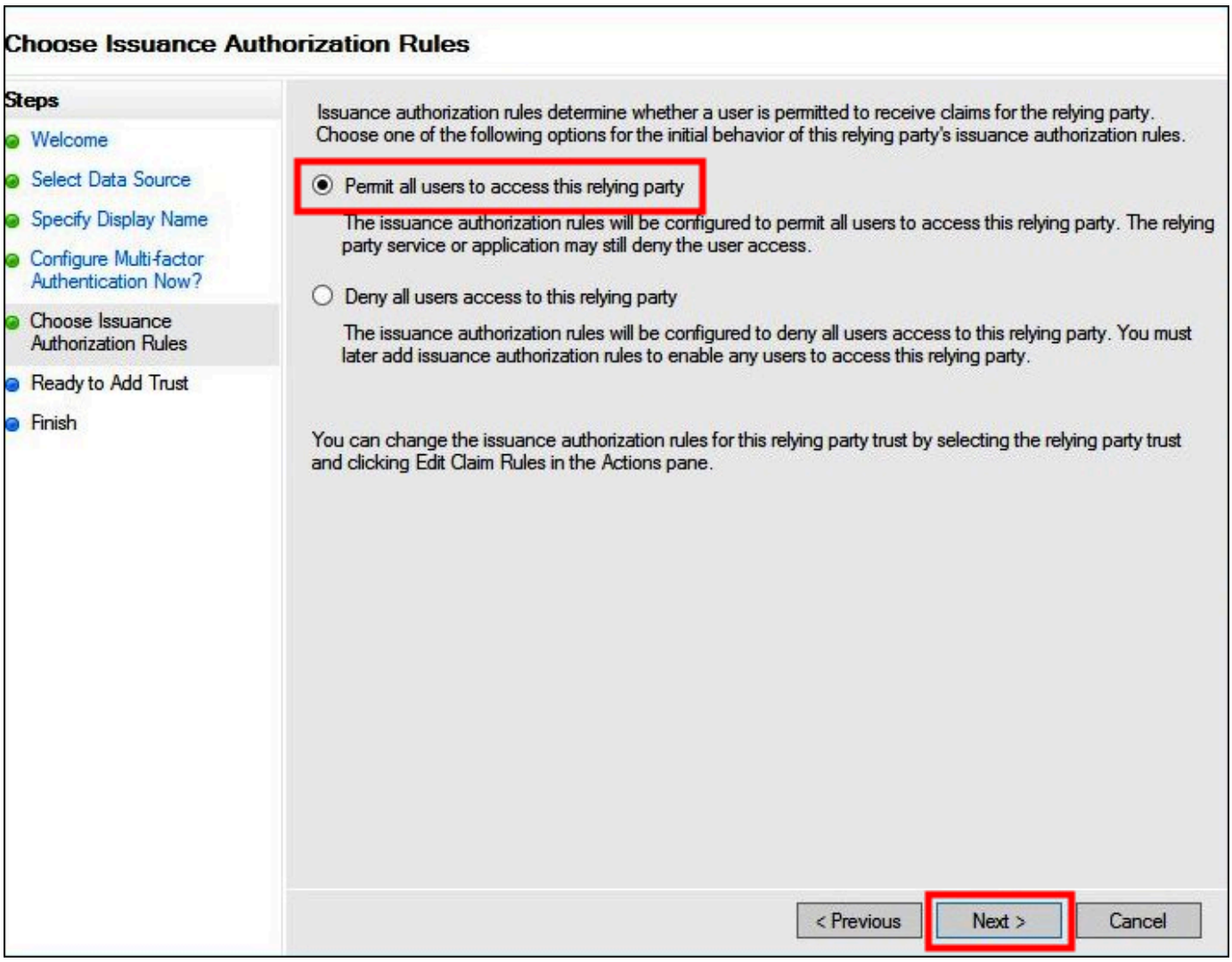
11. 이 자습서에서는 Configure Multi-factor Authentication Now(지금 멀티 팩터 인증 구성) 페이지에서 I do not want to configure multi-factor authentication for this relying party trust at this time(지금은 이 신뢰 당사자 신뢰에 대한 멀티 팩터 인증을 구성하지 않습니다)을 선택합니다.

    보안 강화를 위해 멀티 팩터 인증을 구성하여 AWS 리소스를 보호하는 것이 좋습니다. 샘플 데이터 세트를 사용하므로 이 자습서에서는 멀티 팩터 인증을 활성화하지 않습니다.

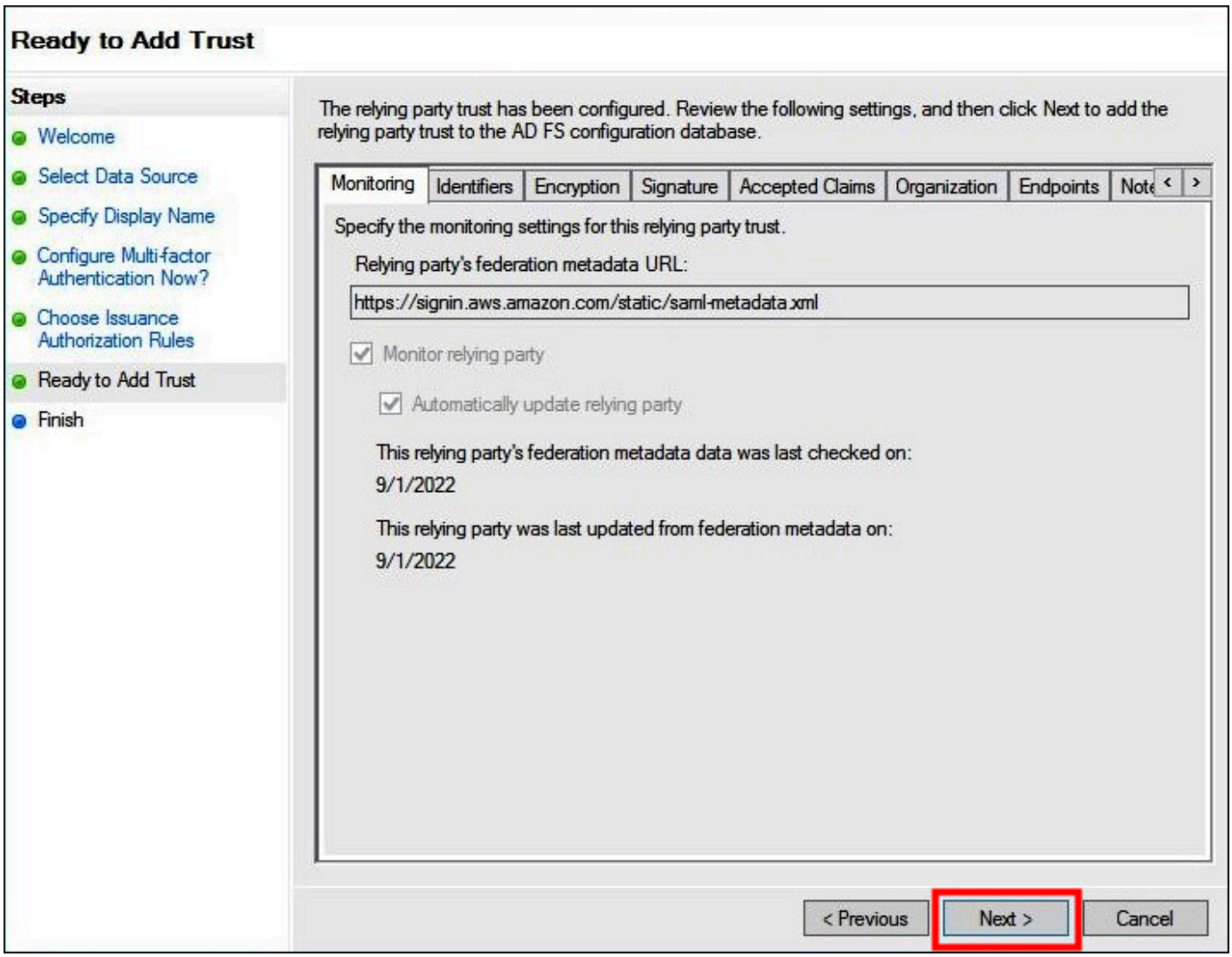


12. 다음을 선택합니다.
13. Choose Issuance Authorization Rules(발급 권한 부여 규칙 선택) 페이지에서 Permit all users to access this relying party(모든 사용자가 이 신뢰 당사자에 액세스하도록 허용)를 선택합니다.

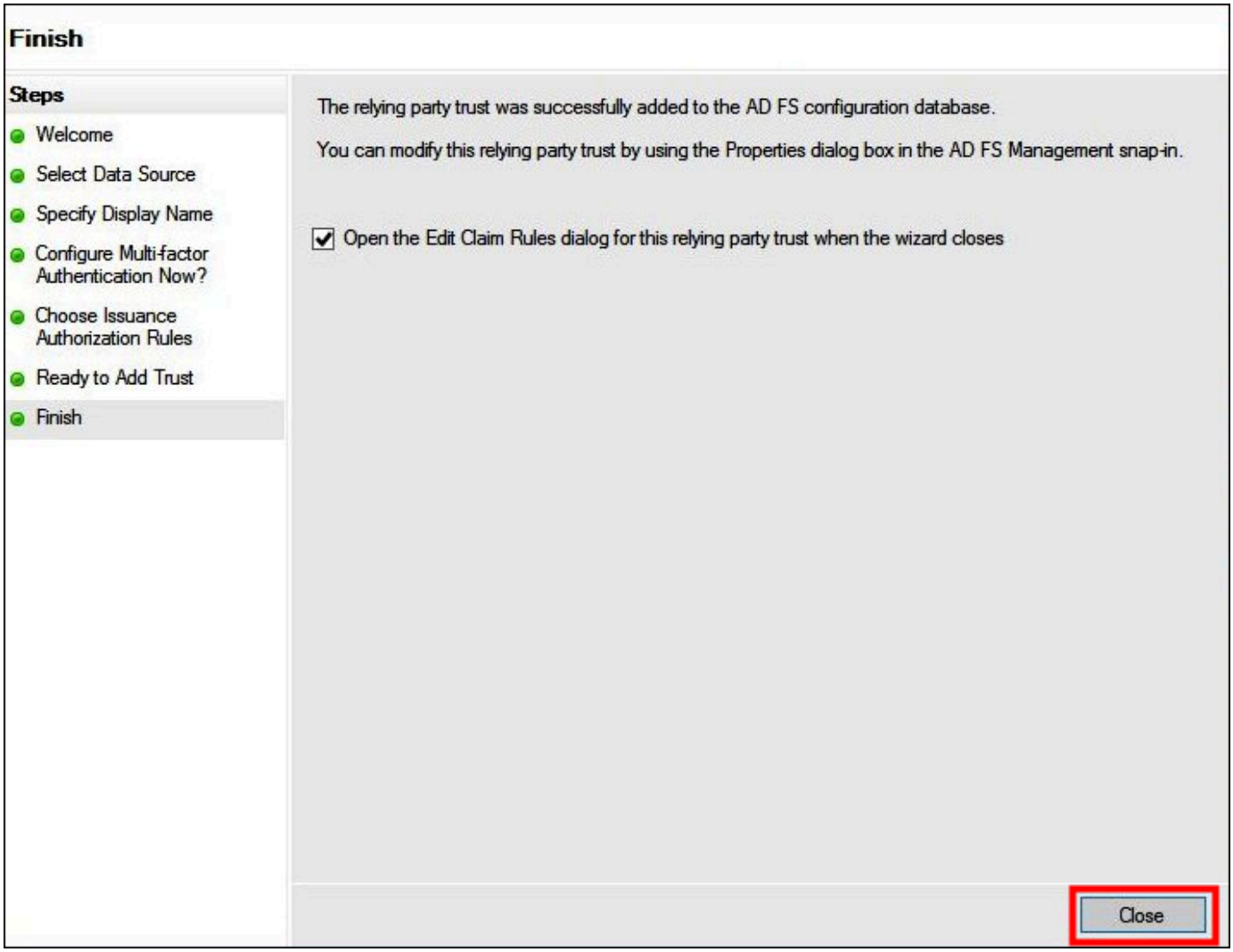
    이 옵션을 사용하면 Active Directory의 모든 사용자가 AWS를 신뢰 당사자로 하여 AD FS를 사용할 수 있습니다. 보안 요구 사항을 고려하여 이 구성을 적절히 조정해야 합니다.



14. 다음을 선택합니다.
15. Ready to Add Trust(신뢰 추가 준비 완료) 페이지에서 Next(다음)를 선택하여 신뢰 당사자 신뢰를 AD FS 구성 데이터베이스에 추가합니다.



16. Finish(마침) 페이지에서 Close(닫기)를 선택합니다.



신뢰 당사자를 위한 SAML 클레임 규칙 구성

이 작업에서는 두 클레임 규칙 세트를 생성합니다.

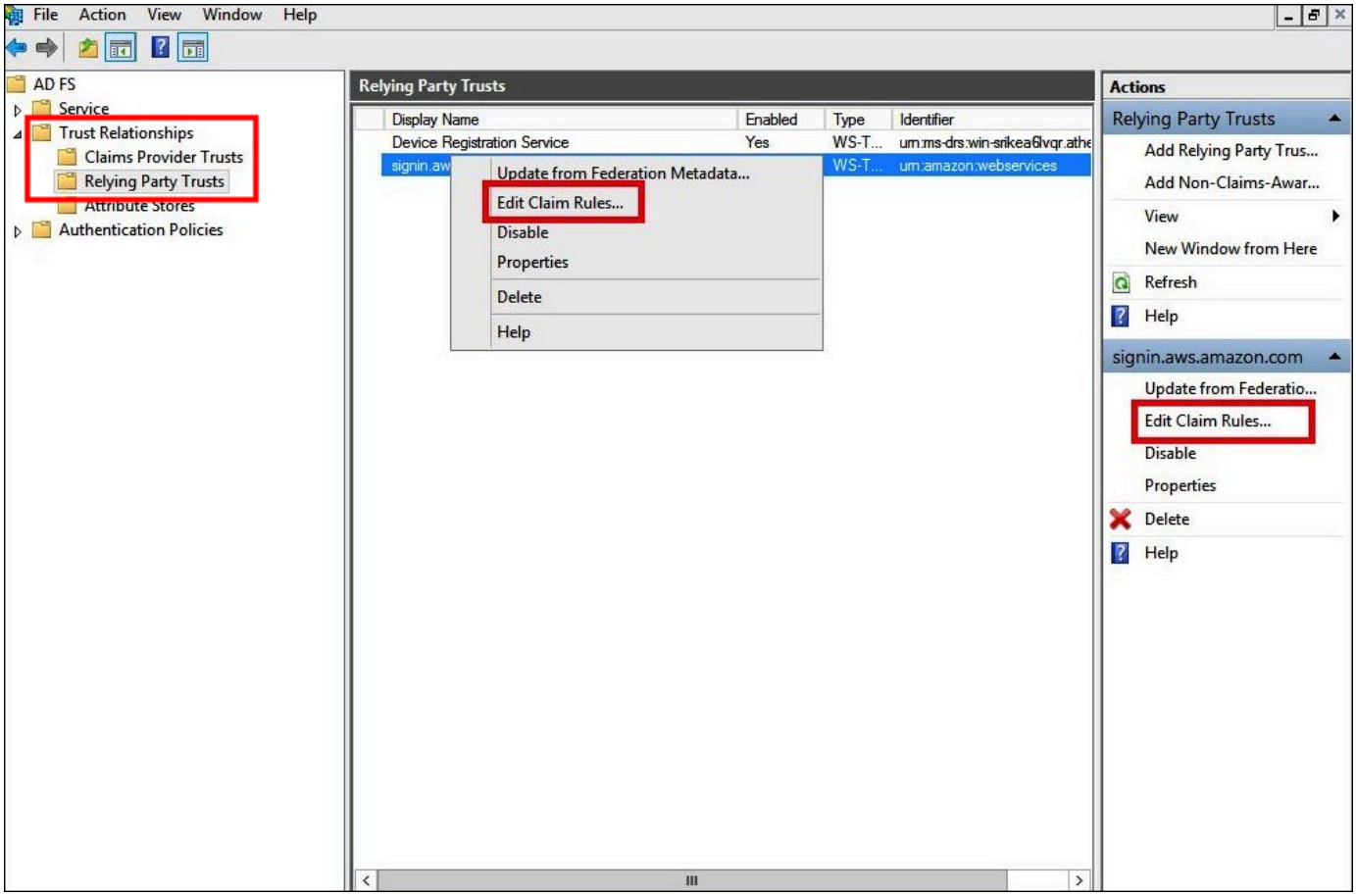
첫 번째 세트인 규칙 1~4에는 AD 그룹 구성원 자격에 따라 IAM 역할을 맡는 데 필요한 AD FS 클레임 규칙이 포함됩니다. 이는 AWS Management Console에 대한 페더레이션 액세스를 설정하려는 경우에 생성하는 것과 동일한 규칙입니다.

두 번째 세트인 규칙 5~6은 Athena 액세스 제어에 필요한 클레임 규칙입니다.

AD FS 클레임 규칙을 생성하려면

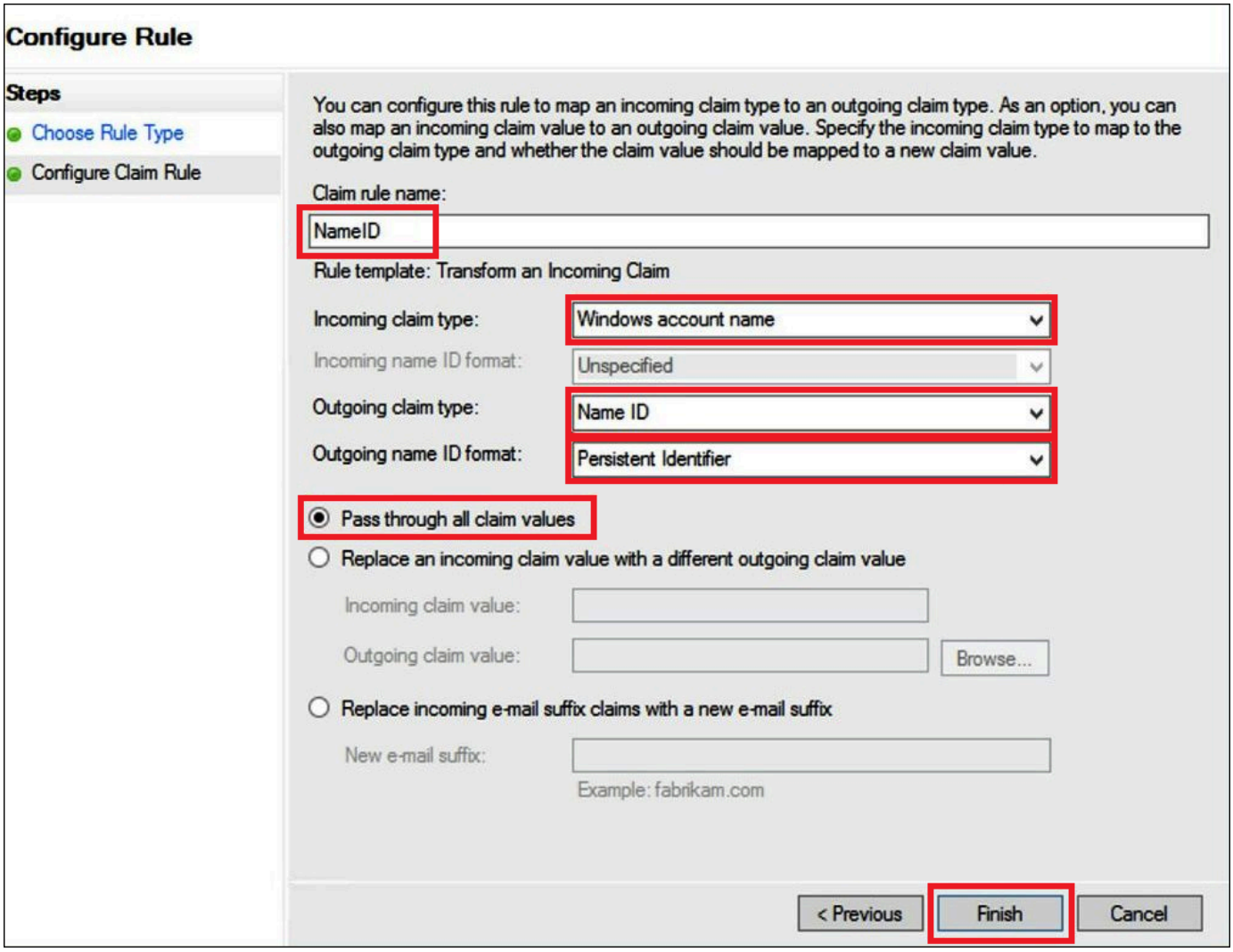
1. AD FS 관리 콘솔 탐색 창에서 Trust Relationships(신뢰 관계), Relying Party Trusts(신뢰 당사자 신뢰)를 선택합니다.

2. 이전 단원에서 생성한 신뢰 당사자를 찾습니다.
3. 신뢰 당사자를 마우스 오른쪽 버튼으로 클릭하고 Edit Claim Rules(클레임 규칙 편집)를 선택하거나 Actions(작업) 메뉴에서 Edit Claim Rules(클레임 규칙 편집)를 선택합니다.



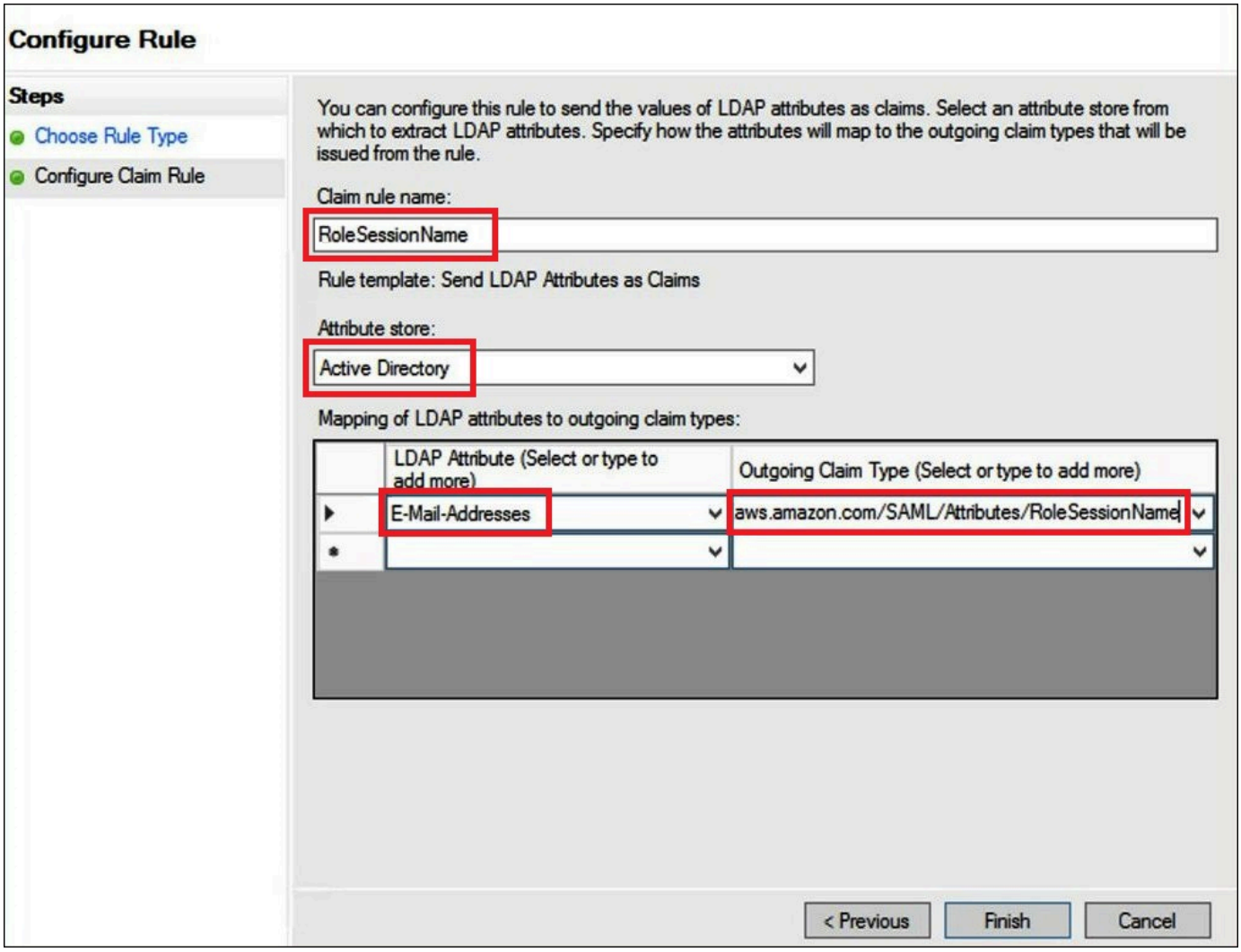
4. 규칙 추가(Add Rule)를 선택합니다.
5. 변환 클레임 규칙 추가 마법사의 Configure Rule(규칙 구성) 페이지에서 다음 정보를 입력하여 클레임 규칙 1을 생성한 다음 Finish(마침)를 선택합니다.

   - Claim Rule name(클레임 규칙 이름)에 **`NameID`**를 입력합니다.
   - Rule template(규칙 템플릿)에서 Transform an Incoming Claim(수신 클레임 변환)을 사용합니다.
   - Incoming claim name(수신 클레임 이름)에서 Windows Account Name(Windows 계정 이름)을 선택합니다.
   - Outgoing claim name(발신 클레임 이름)으로 Name ID(이름 ID)를 선택합니다.
   - Outgoing name ID format(발신 이름 ID 형식)으로 Persistent Identifier(영구 식별자)를 선택합니다.
   - Pass through all claim values(모든 클레임 값 전달)를 선택합니다.



6. Add Rule(규칙 추가)을 선택하고 다음 정보를 입력하여 클레임 규칙 2를 생성한 다음 Finish(마침)를 선택합니다.

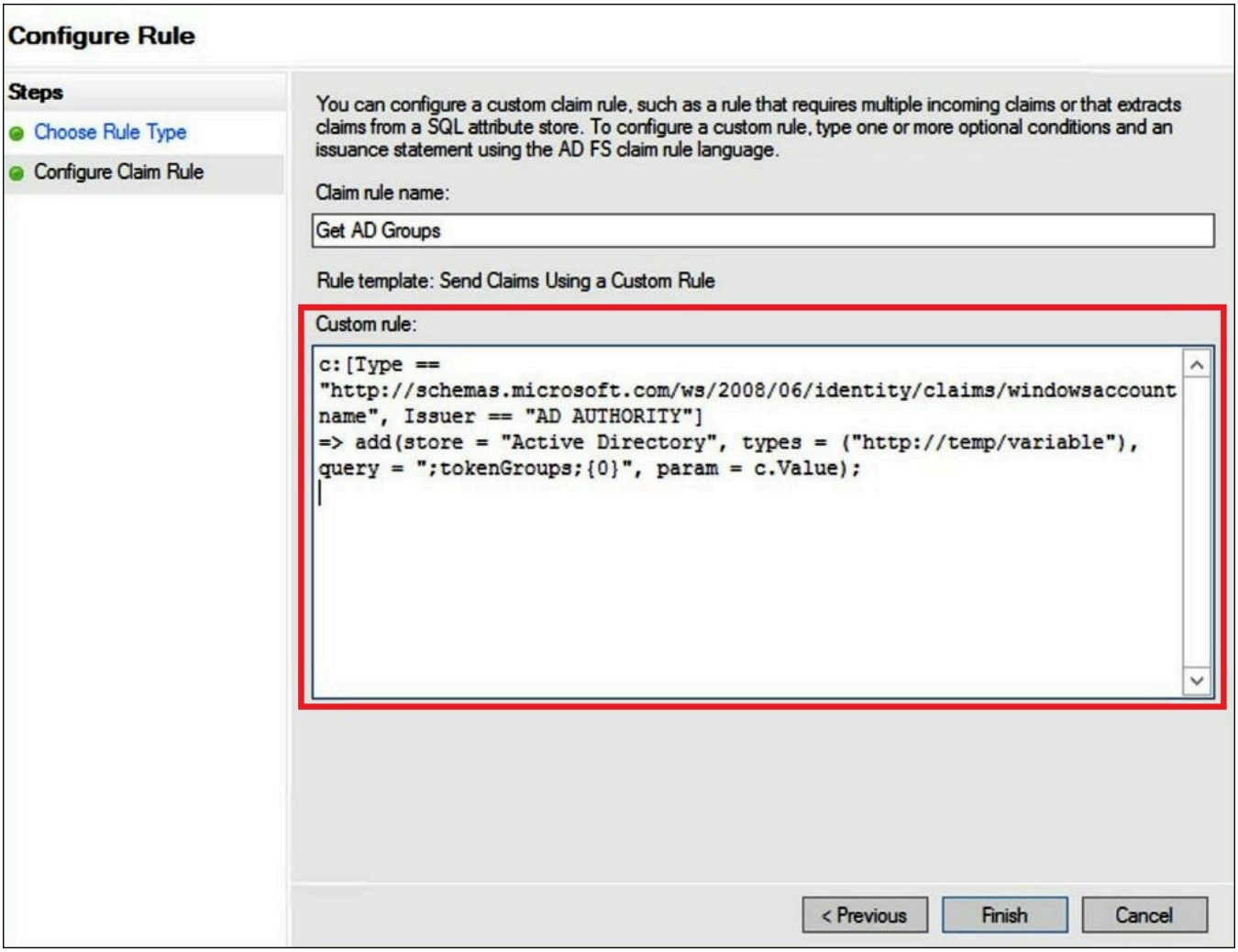
   - Claim Rule name(클레임 규칙 이름)에 **`RoleSessionName`**을 입력합니다.
   - Rule template(규칙 템플릿)으로 Send LDAP Attribute as Claims(LDAP 속성을 클레임으로 전송)를 사용합니다.
   - 속성 저장의 경우 Active Directory를 선택합니다.
   - Mapping of LDAP attributes to outgoing claim types(발신 클레임 유형에 LDAP 속성 매핑)에서 **`E-Mail-Addresses`** 속성을 추가합니다. Outgoing Claim Type(발신 클레임 유형)에 **`https://aws.amazon.com/SAML/Attributes/RoleSessionName`**을 입력합니다.



7. Add Rule(규칙 추가)을 선택하고 다음 정보를 입력하여 클레임 규칙 3을 생성한 다음 Finish(마침)를 선택합니다.

   - Claim Rule name(클레임 규칙 이름)에 **`Get AD Groups`**을 입력합니다.
   - Rule template(규칙 템플릿)에서 Send Claims Using a Custom Rule(사용자 지정 규칙을 사용하여 클레임 전송)을 사용합니다.
   - Custom rule(사용자 지정 규칙)에 다음 코드를 입력합니다.

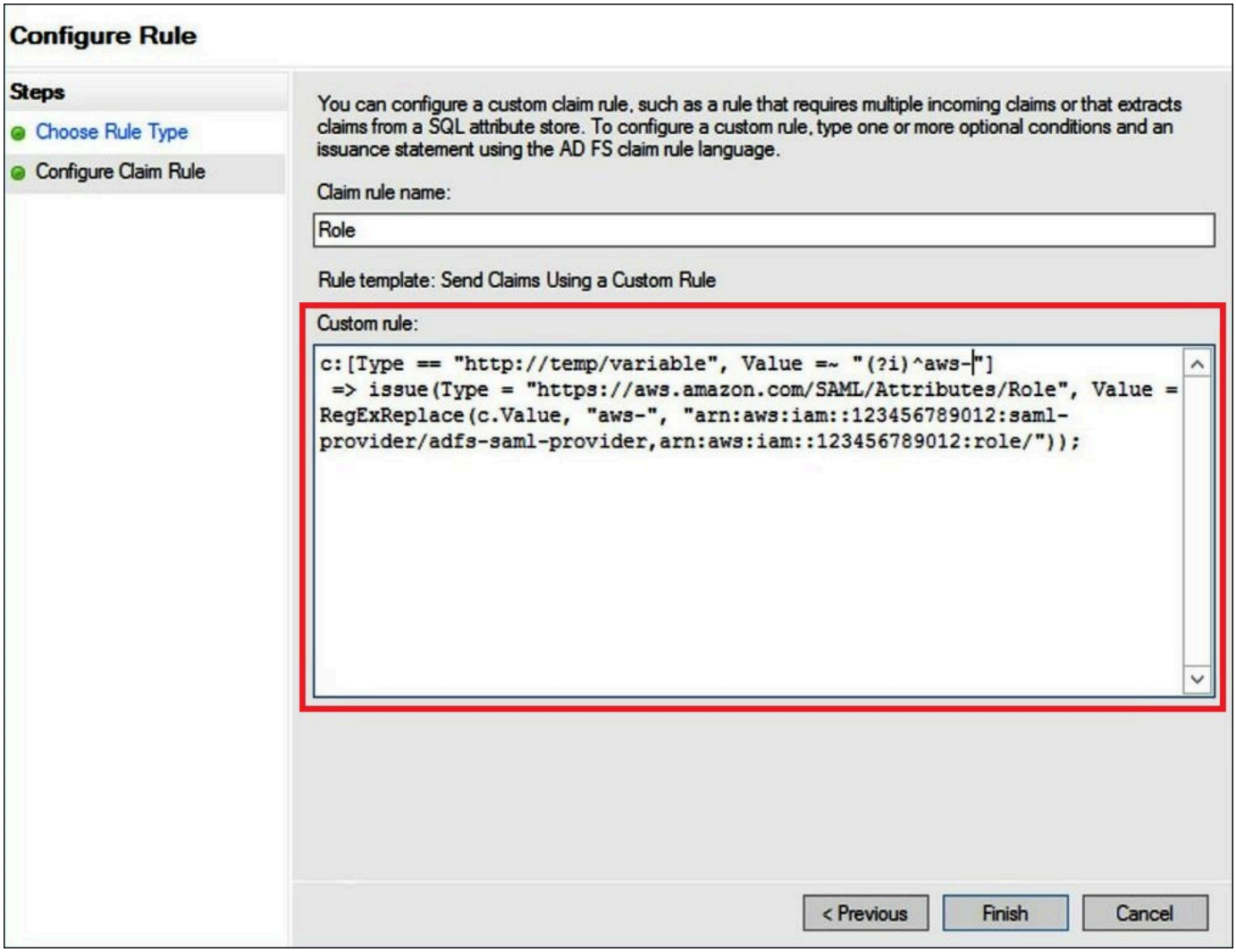
```
c:[Type == "http://schemas.microsoft.com/ws/2008/06/identity/claims/
windowsaccountname",
 Issuer == "AD AUTHORITY"]=> add(store = "Active Directory", types = ("http://
temp/variable"),
 query = ";tokenGroups;{0}", param = c.Value);
```



8. 규칙 추가(Add Rule)를 선택합니다. 다음 정보를 입력하여 클레임 규칙 4를 생성한 다음 Finish(마침)를 선택합니다.

   - Claim Rule name(클레임 규칙 이름)에 **`Role`**을 입력합니다.
   - Rule template(규칙 템플릿)에서 Send Claims Using a Custom Rule(사용자 지정 규칙을 사용하여 클레임 전송)을 사용합니다.
   - Custom rule(사용자 지정 규칙)에 이전에 생성한 SAML 공급자의 계정 번호 및 이름과 함께 다음 코드를 입력합니다.

```
c:[Type == "http://temp/variable", Value =~ "(?i)^aws-"]=> issue(Type = "https://
aws.amazon.com/SAML/Attributes/Role",
Value = RegExReplace(c.Value, "aws-", "arn:aws:iam::AWS_ACCOUNT_NUMBER:saml-
provider/adfs-saml-provider,arn:aws:iam:: AWS_ACCOUNT_NUMBER:role/"));
```

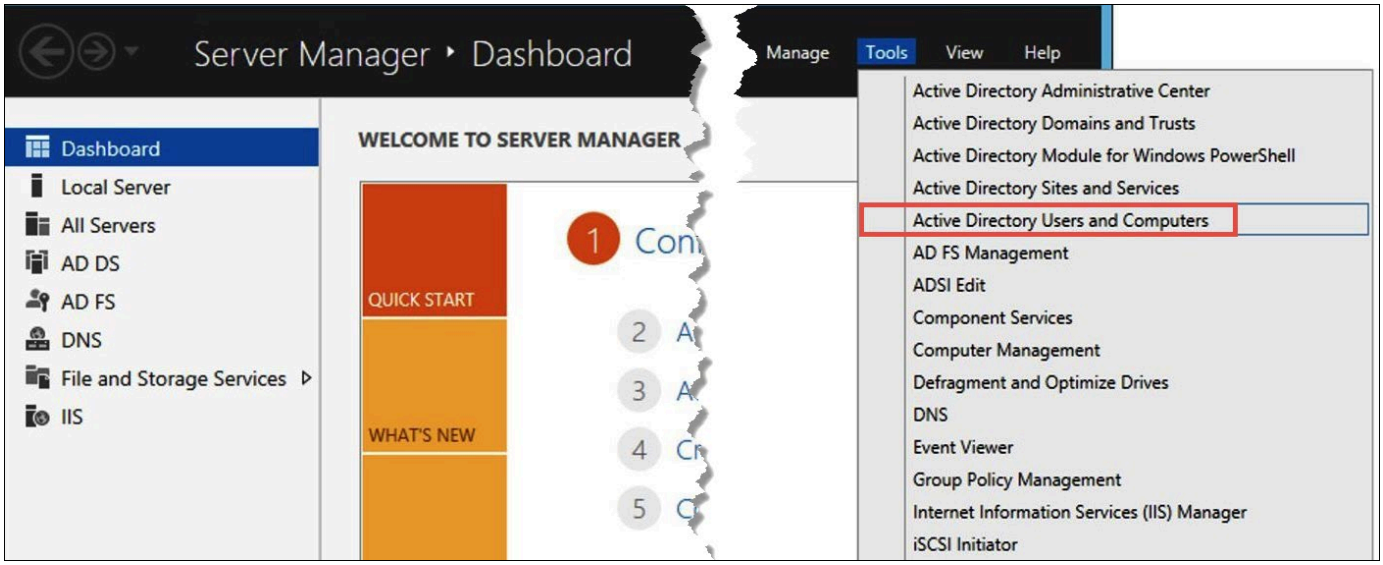


3. Active Directory 사용자 및 그룹 생성

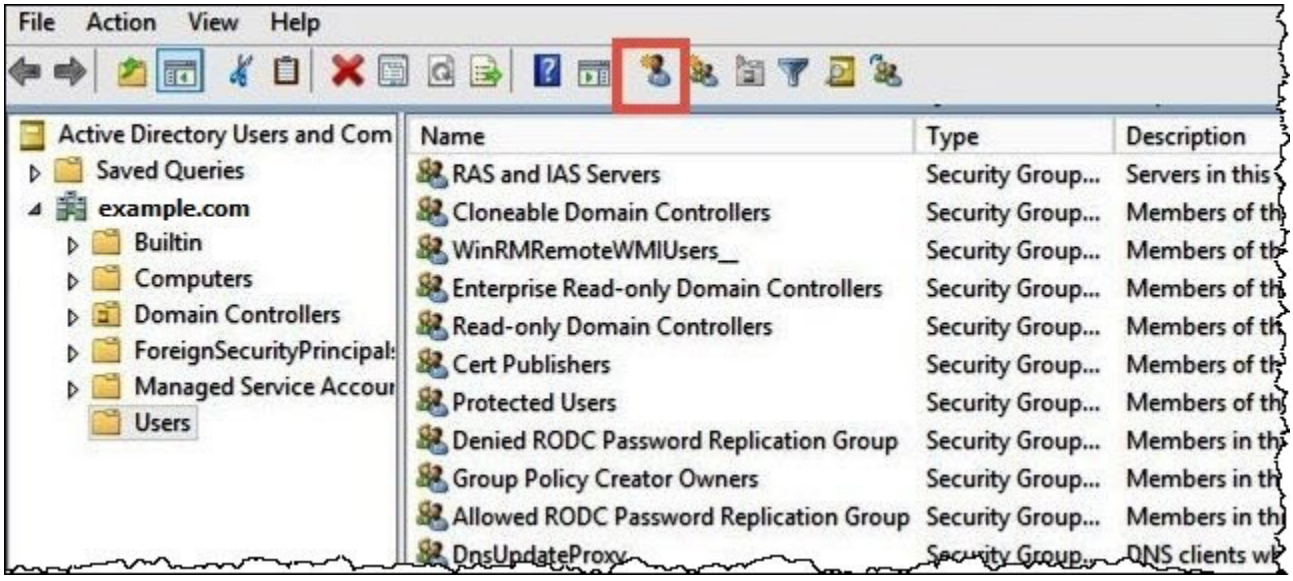
이제 Athena에 액세스할 AD 사용자와 이들을 배치할 AD 그룹을 생성하여 그룹별로 액세스 수준을 제어할 수 있습니다. 데이터 액세스 패턴을 분류하는 AD 그룹을 생성한 후 해당 그룹에 사용자를 추가합니다.

Athena에 액세스할 수 있는 AD 사용자를 생성하려면

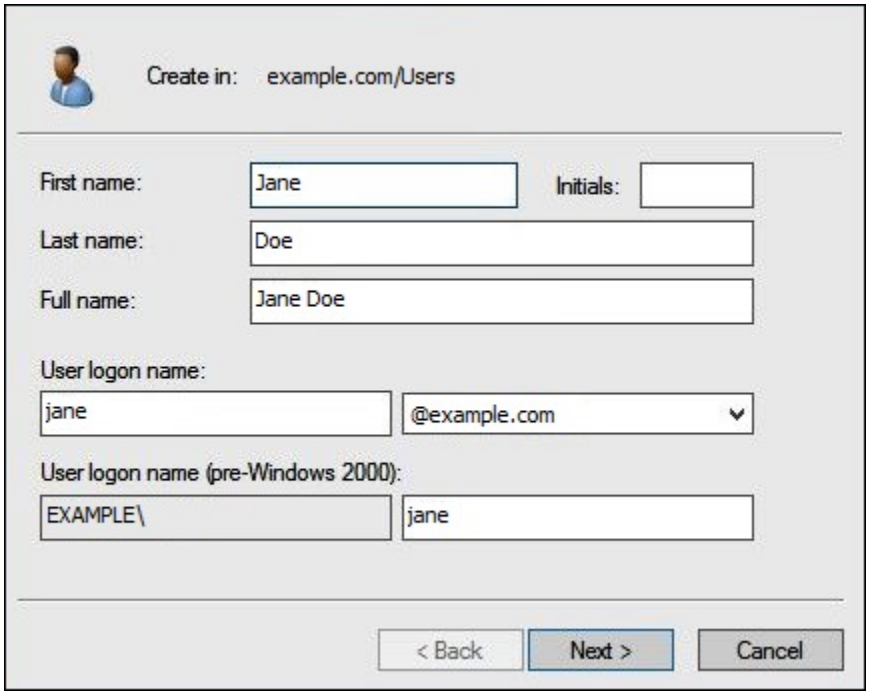
1. 서버 관리자 대시보드에서 Tools(도구)를 선택한 다음 Active Directory Users and Computers(Active Directory 사용자 및 컴퓨터)를 선택합니다.



2. 탐색 창에서 사용자를 선택합니다.
3. Active Directory Users and Computers(Active Directory 사용자 및 컴퓨터) 도구 모음에서 Create user(사용자 생성) 옵션을 선택합니다.

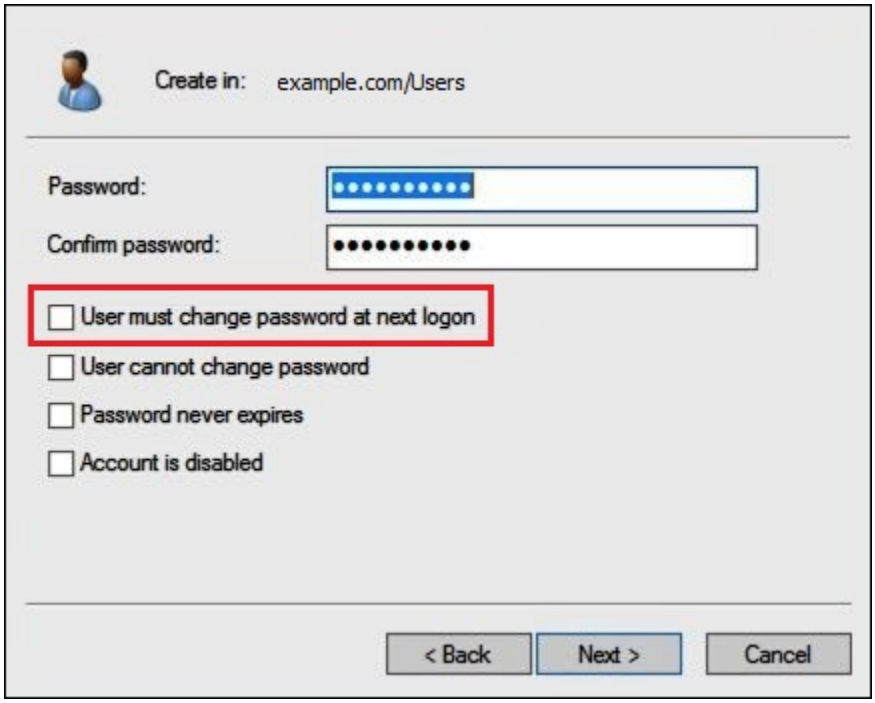


4. New Object – User(새 객체 - 사용자) 대화 상자에서 First name(이름), Last name(성), Full name(전체 이름)에 이름을 입력합니다. 이 자습서에서는 **`Jane Doe`**를 사용합니다.

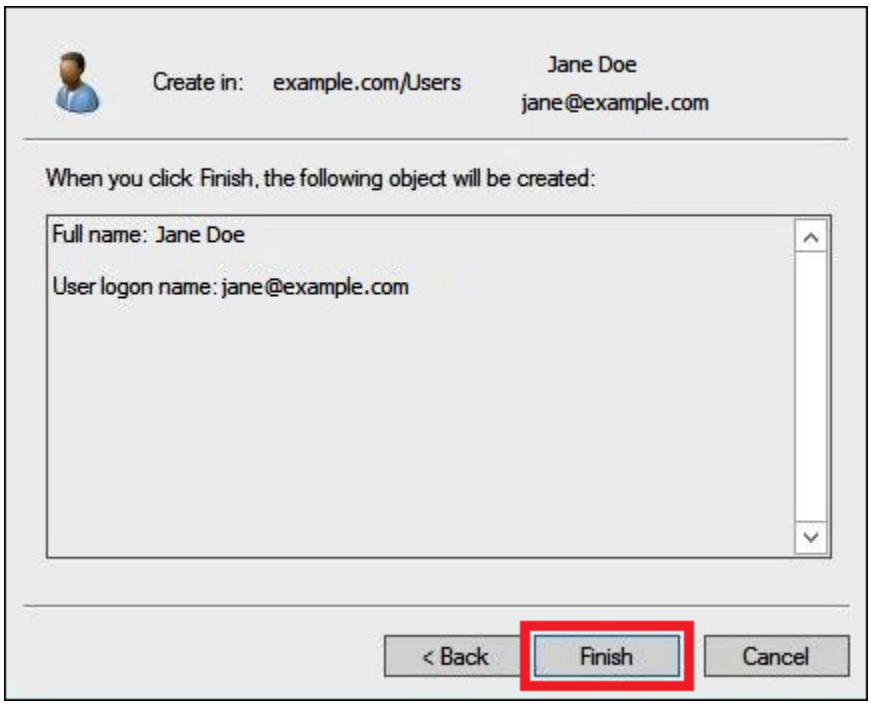


5. 다음을 선택합니다.
6. Password(암호)에 암호를 입력한 다음 확인을 위해 다시 입력합니다.

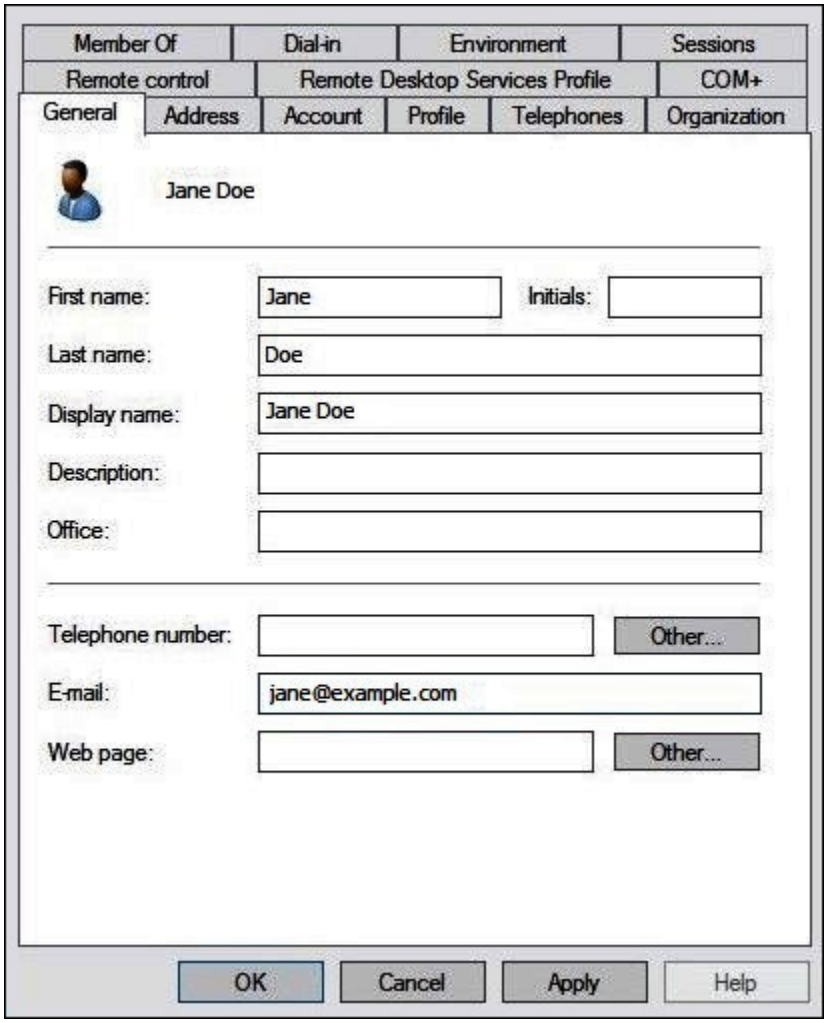
   단순화를 위해 이 자습서에서는 User must change password at next sign on(사용자가 다음 로그온 시 암호를 변경해야 함)을 선택 취소합니다. 실제 시나리오에서는 새로 생성된 사용자에게 암호를 변경하도록 요구해야 합니다.



7. 다음을 선택합니다.
8. 마침을 클릭합니다.



9. Active Directory Users and Computers(Active Directory 사용자 및 컴퓨터)에서 사용자 이름을 선택합니다.
10. 사용자에 대한 Properties(속성) 대화 상자에서 E-mail(이메일)에 이메일 주소를 입력합니다. 이 자습서에서는 **`jane@example.com`**를 사용합니다.



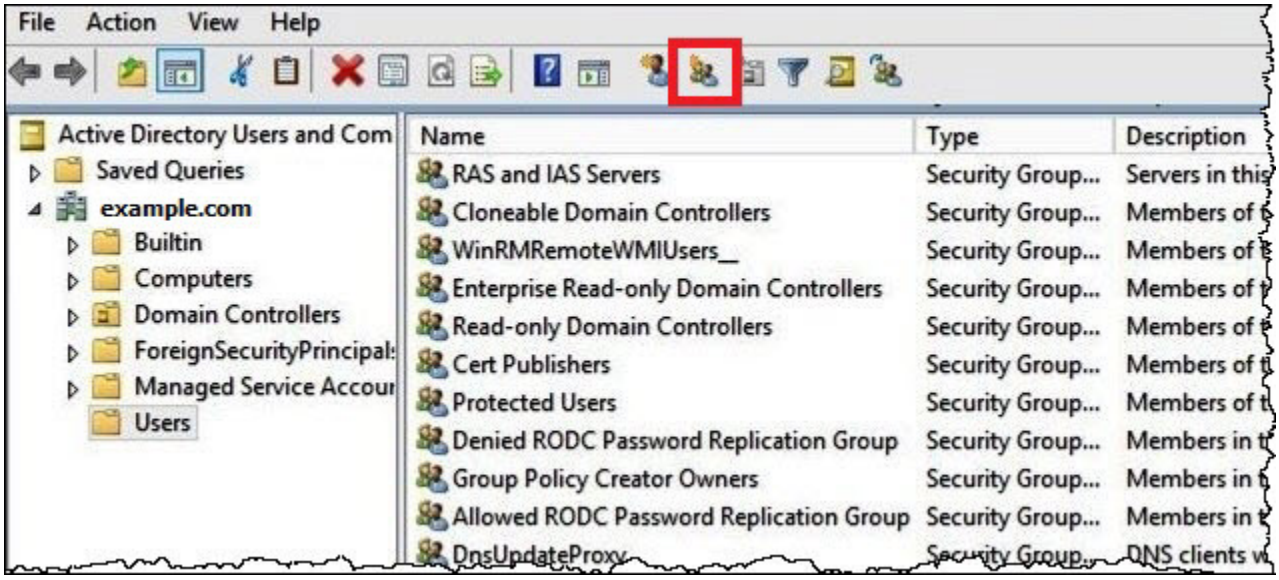
11. 확인을 선택합니다.

데이터 액세스 패턴을 나타내는 AD 그룹 생성

구성원이 AWS에 로그인할 때 `adfs-data-access` IAM 역할을 맡는 AD 그룹을 생성할 수 있습니다. 다음 예에서는 aws-adfs-data-access라는 AD 그룹을 생성합니다.

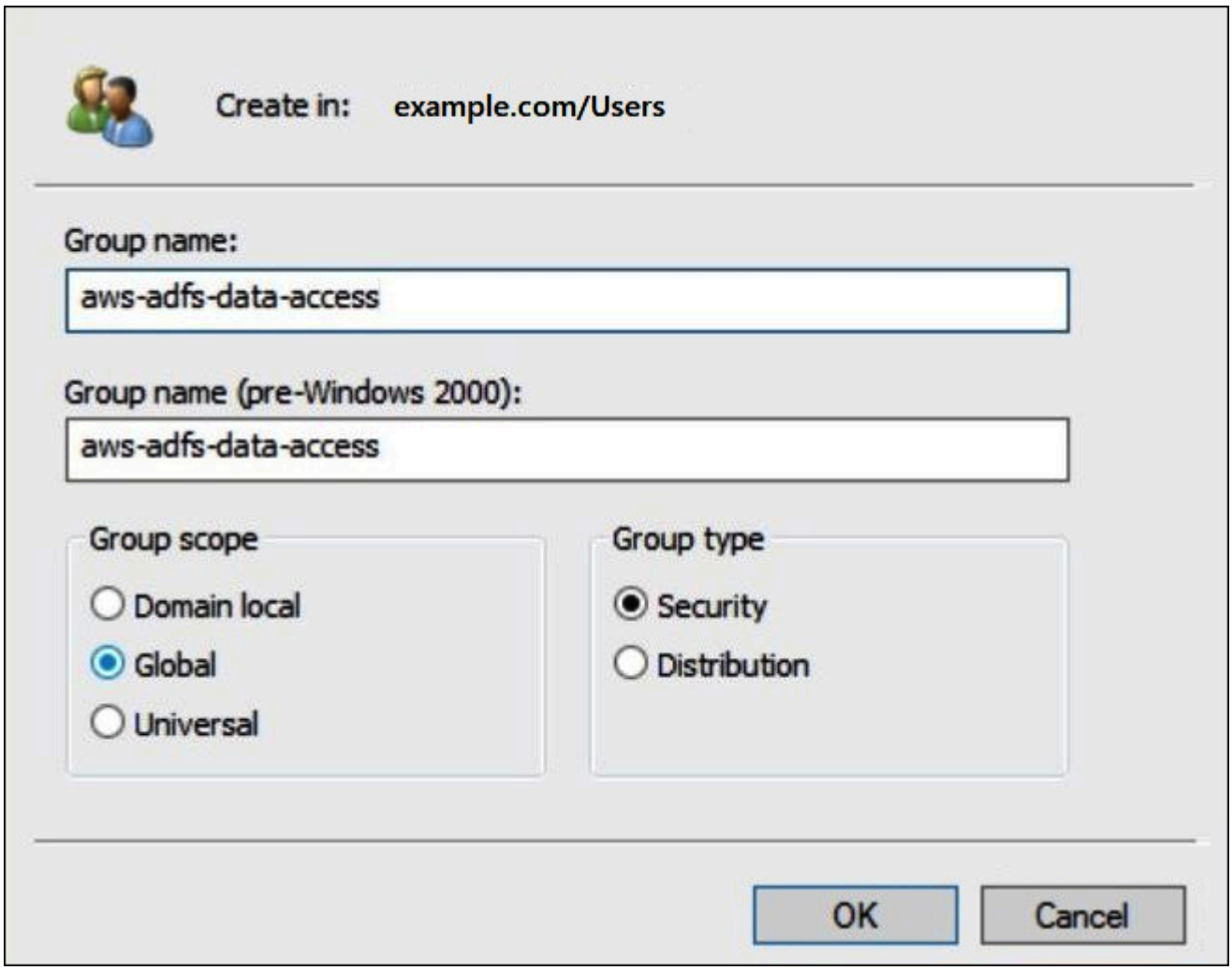
AD 그룹을 생성하려면

1. 서버 관리자 대시보드의 Tools(도구) 메뉴에서 Active Directory Users and Computers(Active Directory 사용자 및 컴퓨터)를 선택합니다.
2. 도구 모음에서 Create new group(새 그룹 생성) 옵션을 선택합니다.



3. New Object - Group(새 객체 - 그룹) 대화 상자에서 다음 정보를 입력합니다.

   - Role name(역할 이름)에 **`aws-adfs-data-access`**를 입력합니다.
   - Group scope(그룹 범위)에서 Global(글로벌)을 선택합니다.
   - Group type(그룹 유형)에서 Security(보안)를 선택합니다.



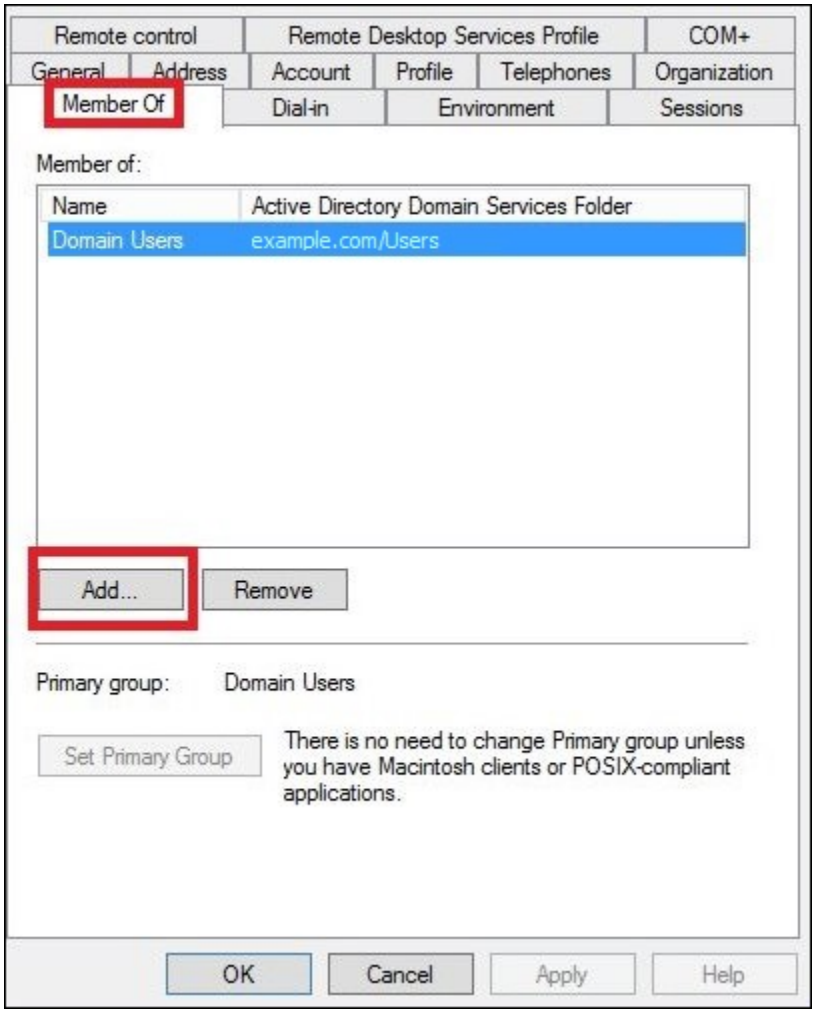
4. 확인을 선택합니다.

적절한 그룹에 AD 사용자 추가

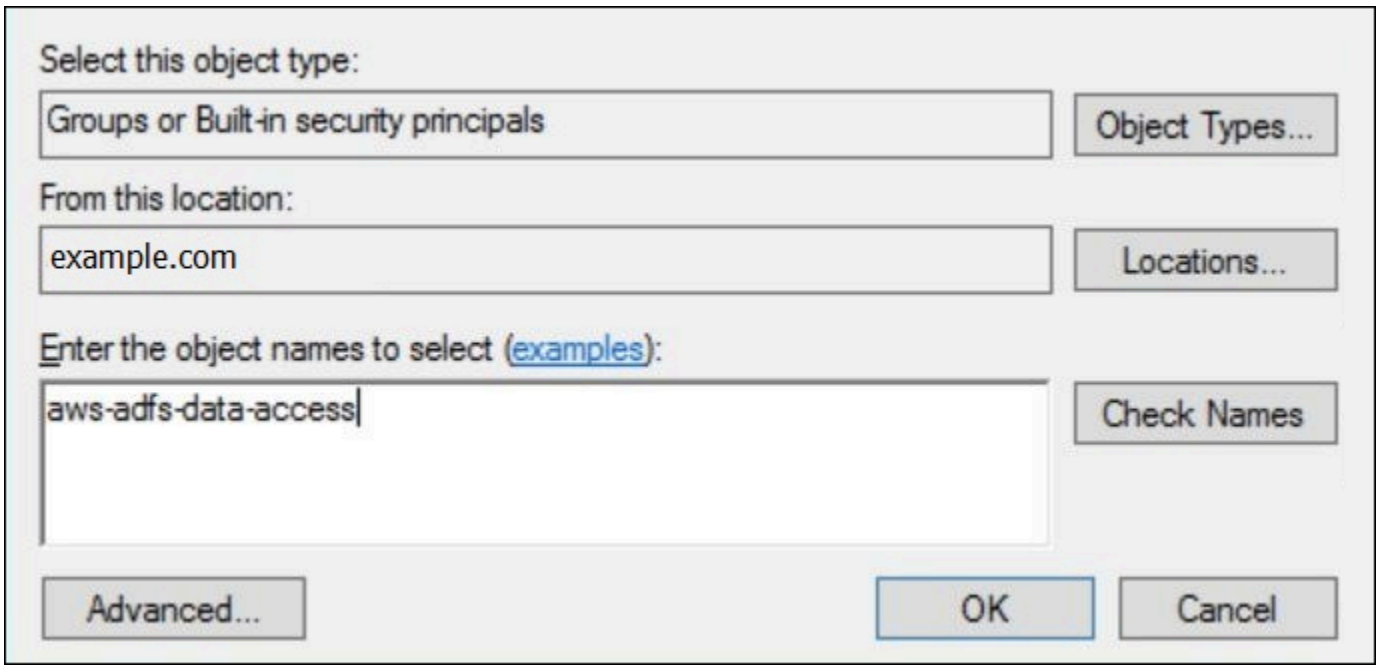
이제 AD 사용자와 AD 그룹을 모두 생성했으므로 그룹에 사용자를 추가할 수 있습니다.

AD 그룹에 AD 사용자를 추가하려면

1. 서버 관리자 대시보드의 Tools(도구) 메뉴에서 Active Directory Users and Computers(Active Directory 사용자 및 컴퓨터)를 선택합니다.
2. First name(이름) 및 Last name(성)에서 사용자(예: Jane Doe)를 선택합니다.
3. 사용자에 대한 Properties(속성) 대화 상자의 Member Of(소속 그룹) 탭에서 Add(추가)를 선택합니다.

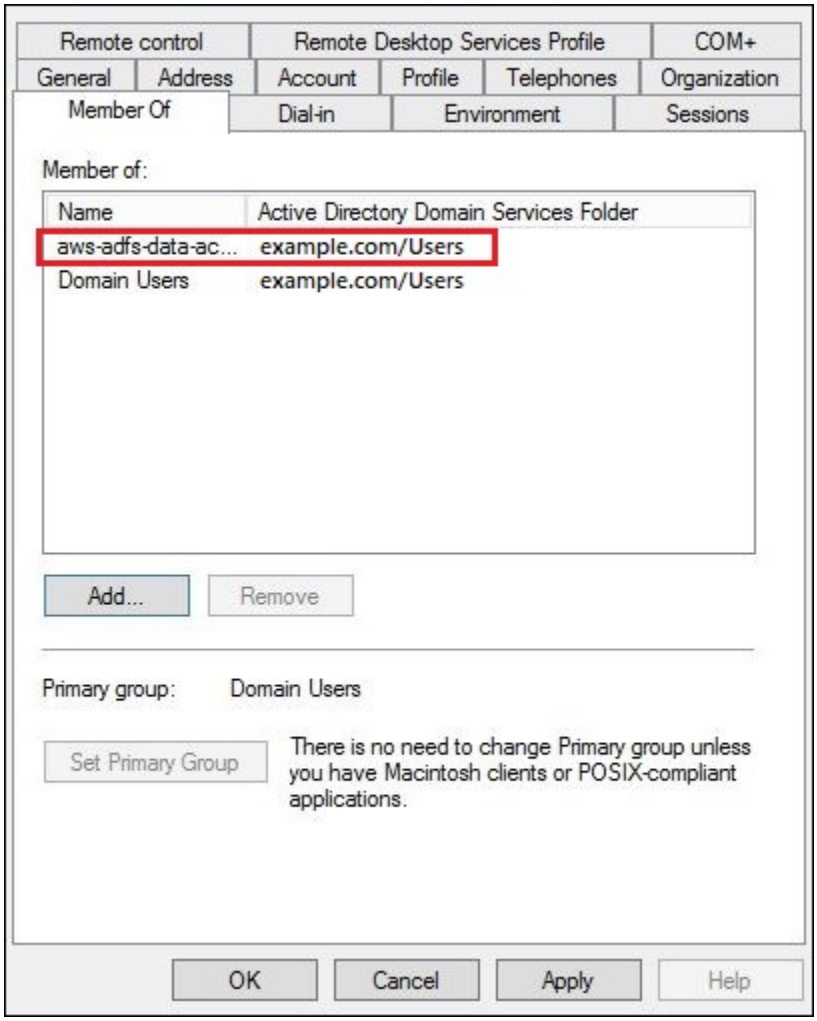


4. 요구 사항에 따라 AD FS 그룹을 하나 이상 추가합니다. 이 자습서에서는 aws-adfs-data-access 그룹을 추가합니다.
5. Select Groups(그룹 선택) 대화 상자에서 Enter the object names to select(선택할 객체 이름 입력)에 생성한 AD FS 그룹의 이름을 입력한 다음(예: **`aws-adfs-data-access`**) Check Names(이름 확인)를 선택합니다.



6. 확인을 선택합니다.

   사용자에 대한 Properties(속성) 대화 상자에서 AD 그룹의 이름이 Member of(소속 그룹) 목록에 표시됩니다.



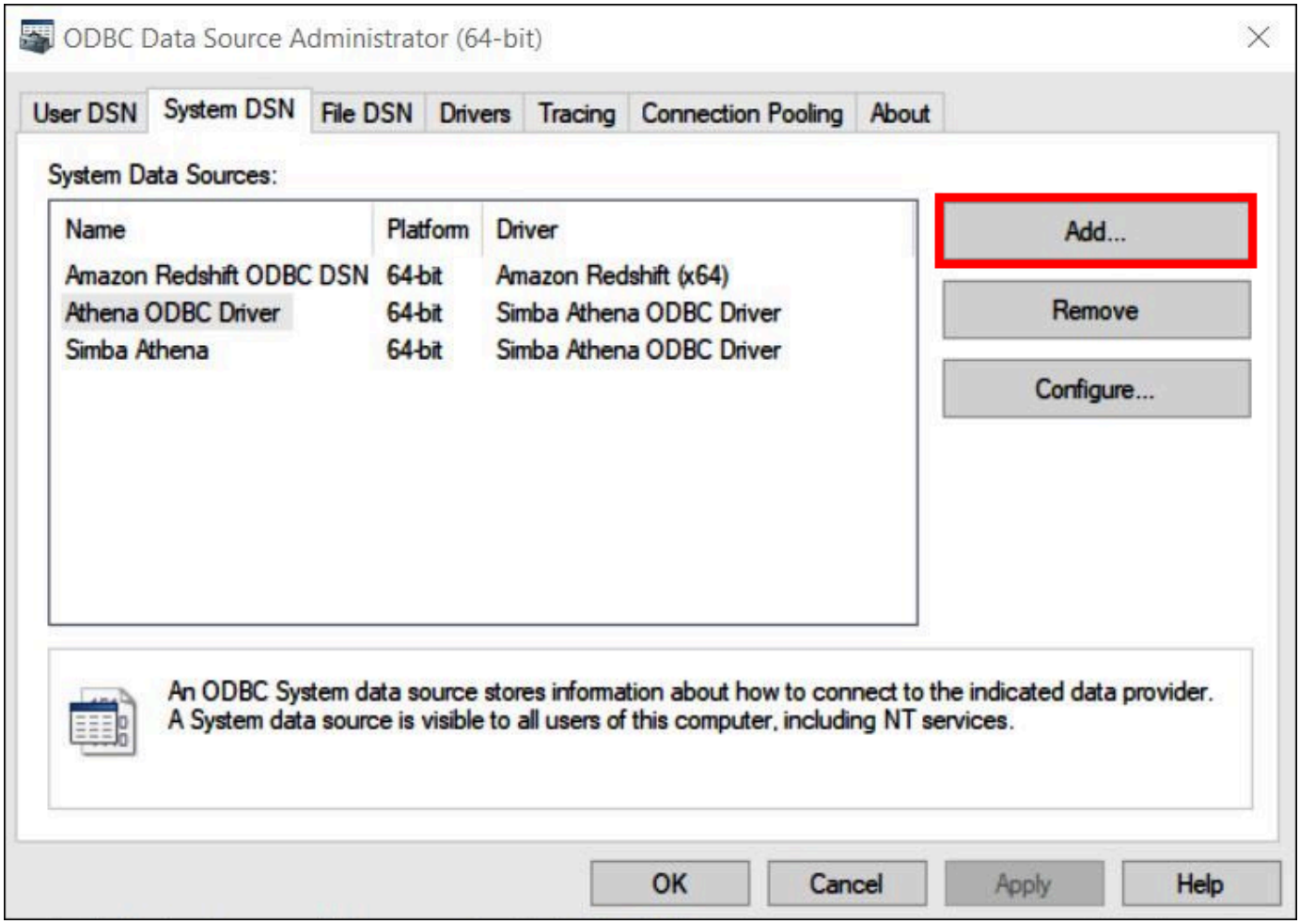
7. Apply(적용)를 선택한 다음 OK(확인)를 선택합니다.

4. Athena에 대한 AD FS ODBC 연결 구성

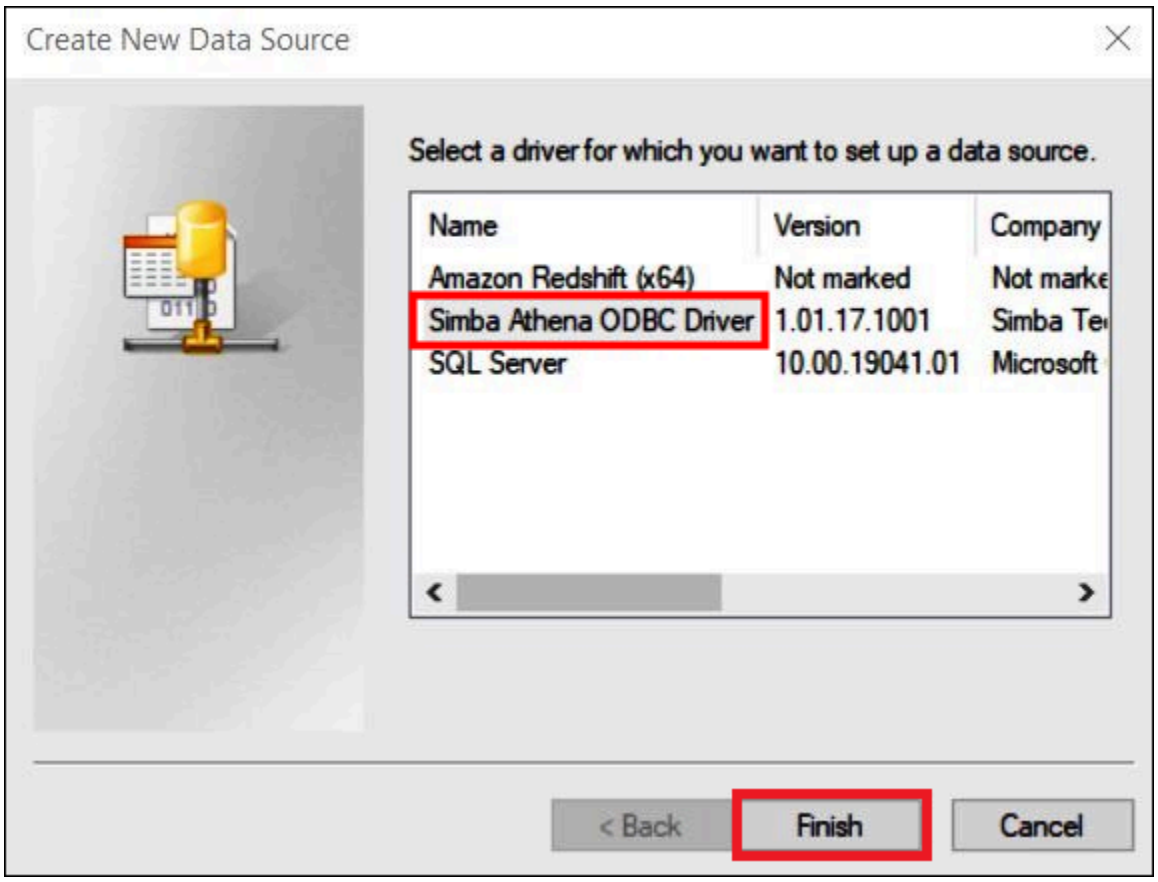
AD 사용자와 그룹을 생성한 이후에 Windows에서 ODBC 데이터 소스 프로그램을 사용하여 AD FS에 대한 Athena ODBC 연결을 구성할 수 있습니다.

Athena에 대한 AD FS ODBC 연결을 구성하려면

1. Athena용 ODBC 드라이버를 설치합니다. 다운로드 링크는 ODBC로 Amazon Athena에 연결를 참조하세요.
2. Windows에서 Start(시작), ODBC Data Sources(ODBC 데이터 소스)를 선택합니다.
3. ODBC 데이터 원본 관리자(ODBC Data Source Administrator) 프로그램에서 추가(Add)를 선택합니다.

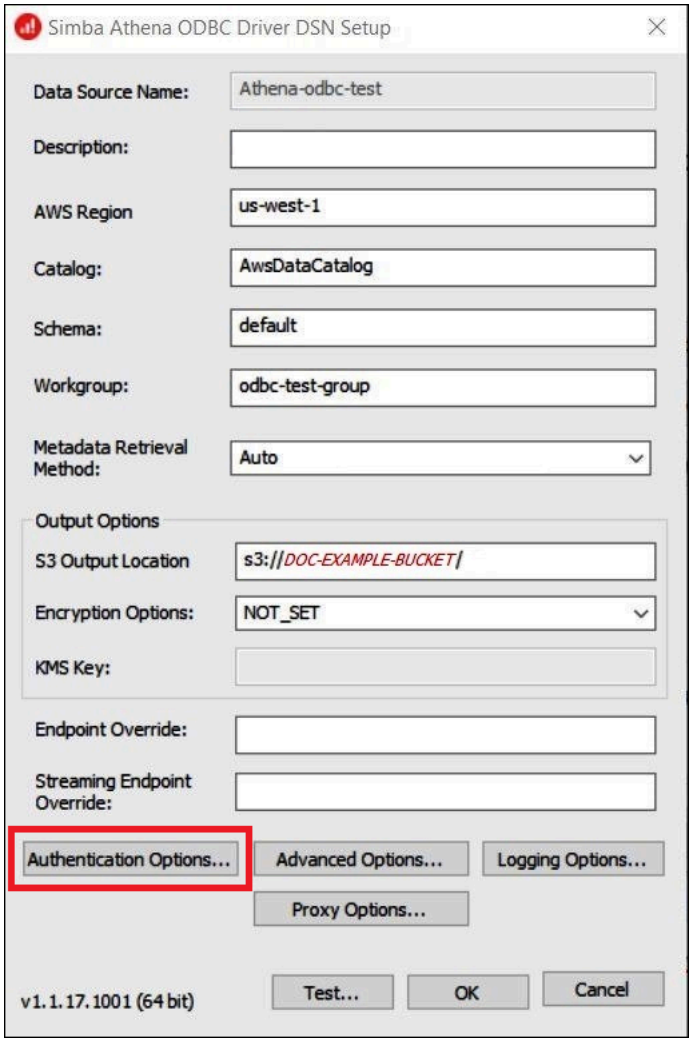


4. Create New Data Source(새 데이터 소스 생성) 대화 상자에서 Simba Athena ODBC Driver(Simba Athena ODBC 드라이버)를 선택한 다음 Finish(마침)를 선택합니다.



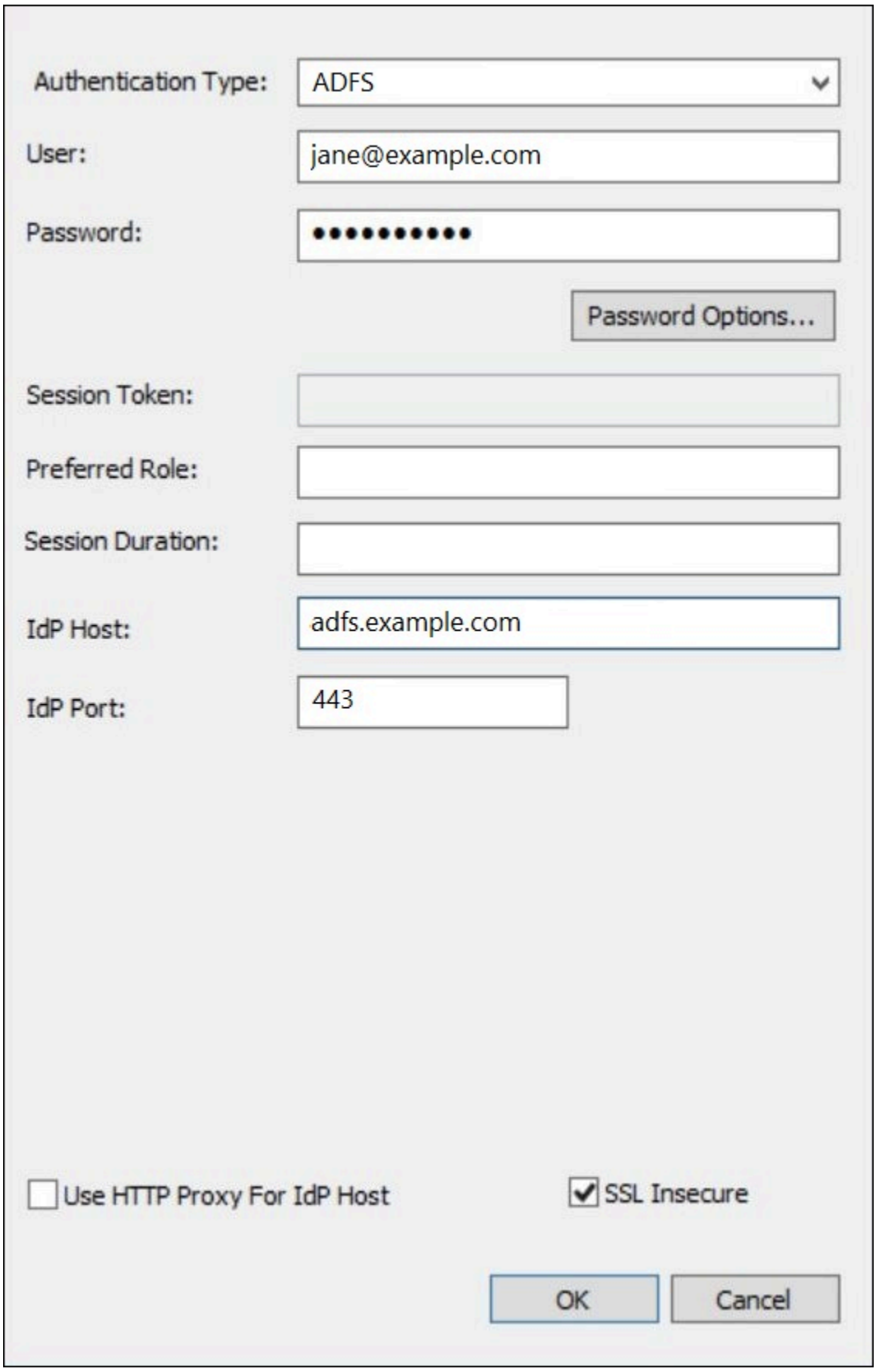
5. Simba Athena ODBC Driver DSN Setup(Simba Athena ODBC 드라이버 DSN 설치) 대화 상자에서 다음 값을 입력합니다.

   - 데이터 소스 이름(Data Source Name)에 데이터 소스 이름을 입력합니다(예: **`Athena-odbc-test`**).
   - 설명(Description)에 데이터 원본에 대한 설명을 입력합니다.
   - AWS 리전에 사용 중인 AWS 리전을 입력합니다(예: **`us-west-1`**).
   - S3 출력 위치(S3 Output Location)에 출력을 저장할 Amazon S3 경로를 입력합니다.



6. 인증 옵션(Authentication Options)을 선택합니다.
7. Authentication Options(인증 옵션) 대화 상자에서 다음 값을 지정합니다.

   - Authentication Type(인증 유형)에서 ADFS를 선택합니다.
   - User(사용자)에 사용자의 이메일 주소(예: **`jane@example.com`**)를 입력합니다.
   - Password(암호)에 사용자의 ADFS 암호를 입력합니다.
   - IdP Host(IdP 호스트트)에 AD FS 서버 이름(예: **`adfs.example.com`**)을 입력합니다.
   - IdP Port(IdP 포트)에서 기본값 443을 사용합니다.
   - SSL Insecure(안전하지 않은 SSL) 옵션을 선택합니다.


Authentication Type:
ADFS
User:
jane@example.com
Password:
Password Options...
Session Token:
Preferred Role:
Session Duration:
IdP Host:
adfs.example.com
IdP Port:
443
Use HTTP Proxy For IdP Host
SSL Insecure
OK
Cancel

8. 확인(OK)을 선택하여 인증 옵션(Authentication Options)을 닫습니다.
9. 테스트(Test)를 선택하여 연결을 테스트하거나 확인(OK)을 선택하여 마칩니다.

Okta 플러그인 및 Okta ID 공급자를 사용하여 ODBC에 대한 SSO 구성

이 페이지에서는 Okta 자격 증명 공급자를 사용하여 Single Sign-On(SSO) 기능에 대한 Amazon Athena ODBC 드라이버 및 Okta 플러그인을 구성하는 방법을 설명합니다.

필수 조건

이 자습서의 단계를 완료하려면 다음이 필요합니다.

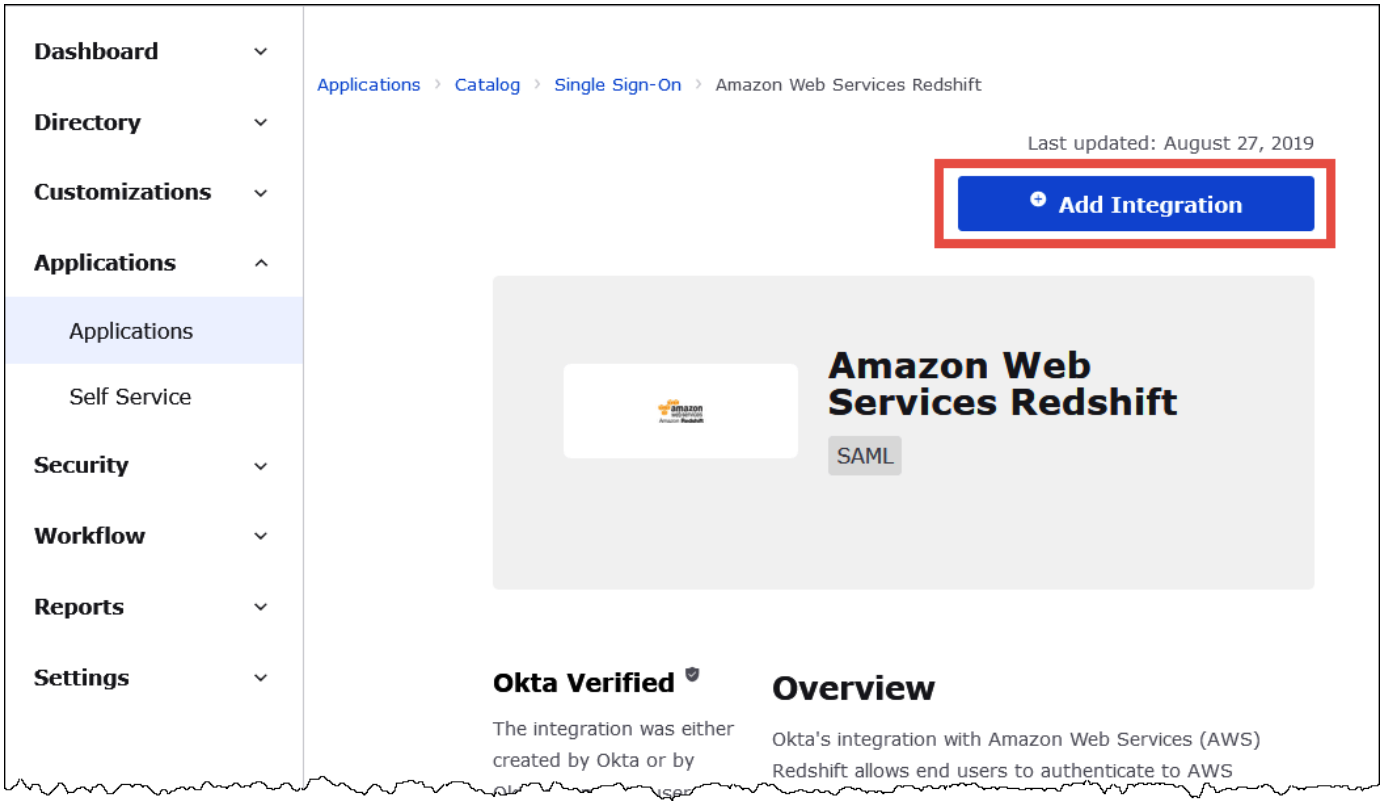
- Amazon Athena ODBC 드라이버. 다운로드 링크는 ODBC로 Amazon Athena에 연결 단원을 참조하세요.
- SAML에서 사용하려는 IAM 역할. 자세한 내용은 IAM 사용자 설명서의 SAML 2.0 페더레이션을 위한 역할 생성을 참조하세요.
- Okta 계정. 자세한 내용을 보려면 Okta.com을 방문하세요.

Okta에서 앱 통합 생성

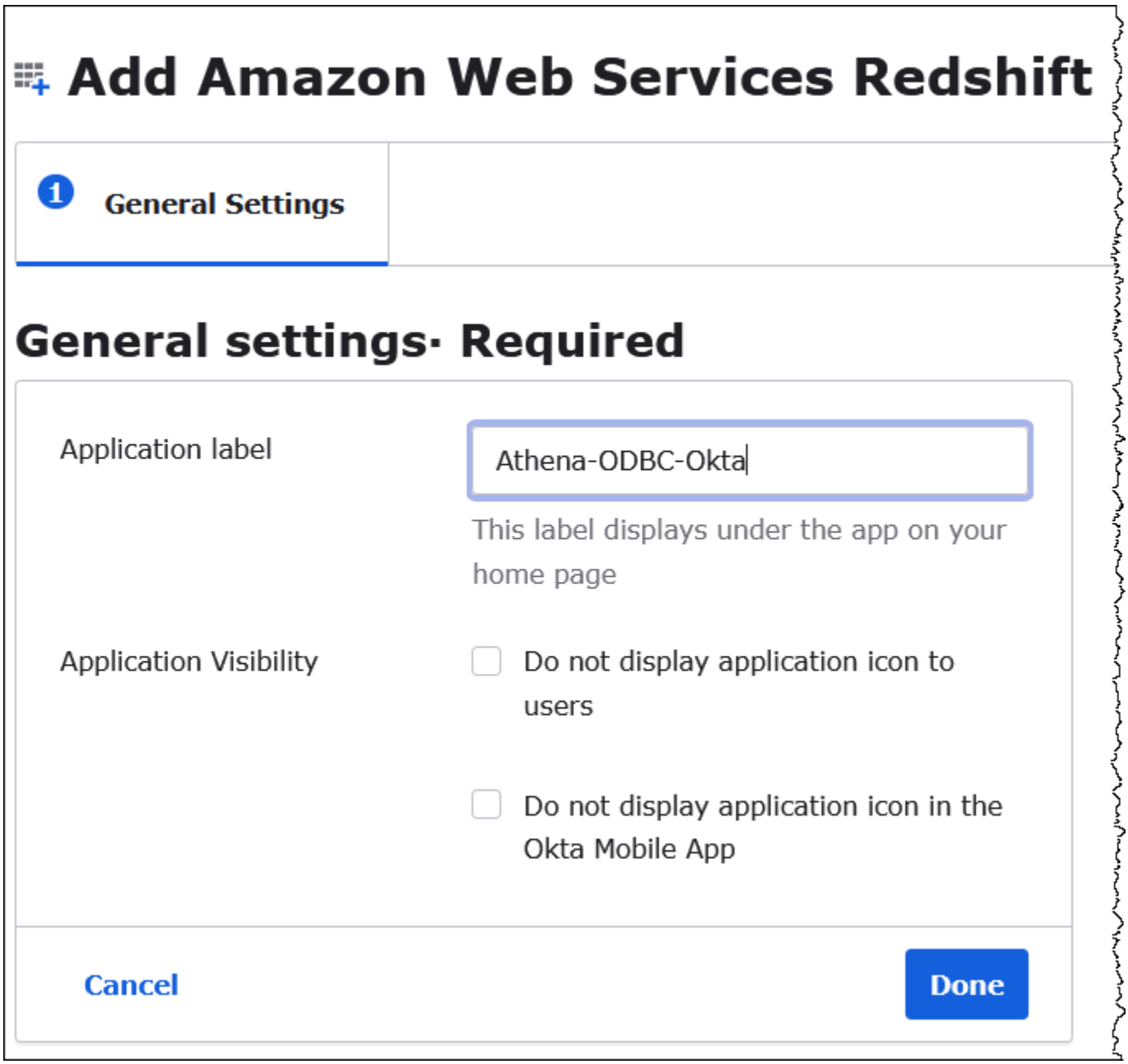
먼저 Okta 대시보드를 사용하여 Athena에 대한 통합 인증을 위한 SAML 2.0 앱을 생성하고 구성합니다. Okta의 기존 Redshift 애플리케이션을 사용하여 Athena에 대한 액세스를 구성할 수 있습니다.

Okta에서 앱 통합을 생성하려면

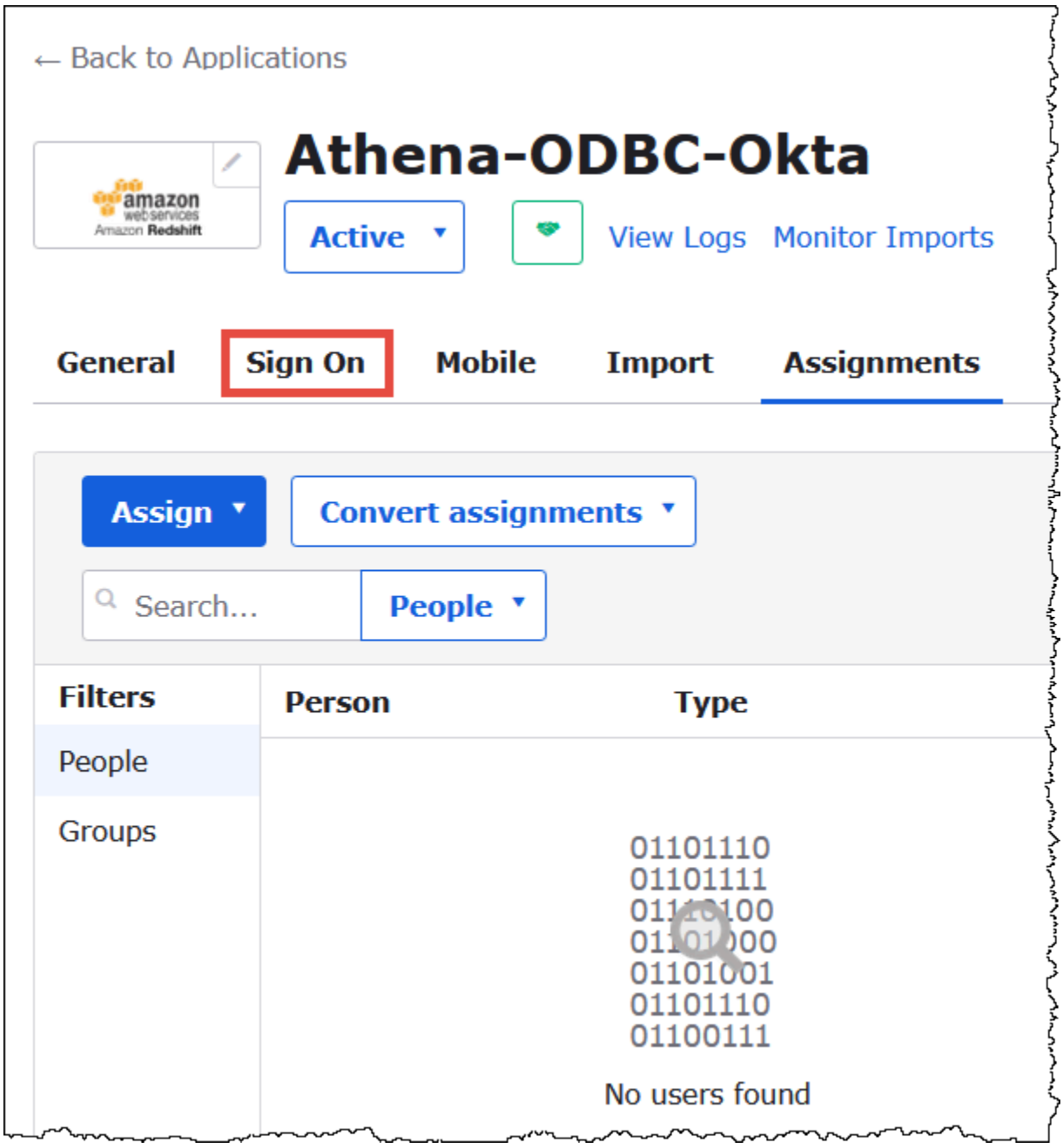
1. Okta.com에서 계정의 관리자 페이지에 로그인합니다.
2. 탐색 창에서 애플리케이션(Applications), 애플리케이션(Applications)을 선택합니다.
3. 애플리케이션(Applications) 페이지에서 앱 카탈로그 찾아보기(Browse App Catalog)를 선택합니다.
4. Browse App Integration Catalog(앱 통합 카탈로그 찾아보기) 페이지의 Use Case(사용 사례) 섹션에서 All Integrations(모든 통합)을 선택합니다.
5. 검색 상자에 Amazon Web Services Redshift를 입력한 다음 Amazon Web Services Redshift SAML을 선택합니다.
6. 통합 추가(Add Integrations)를 선택합니다.



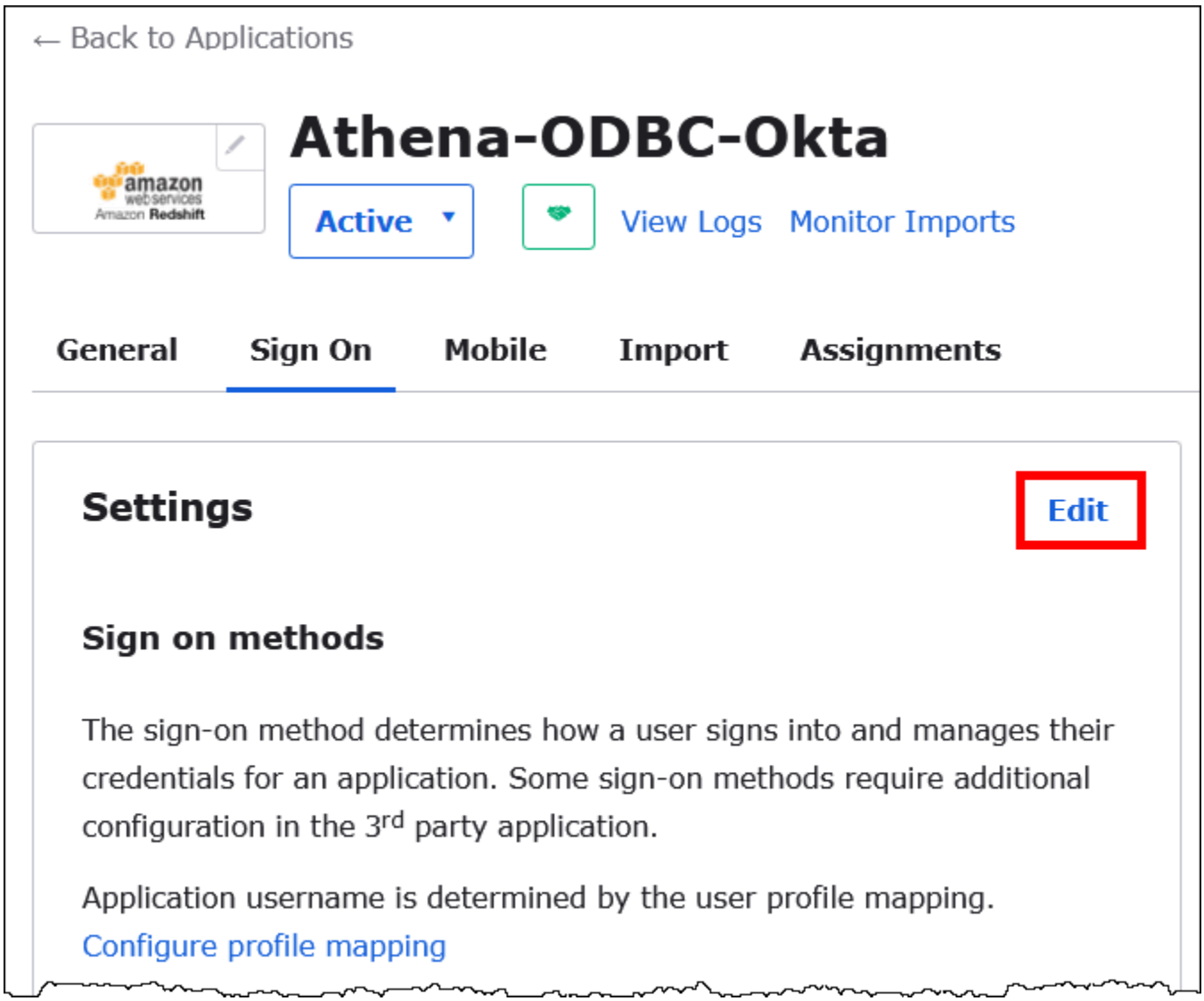
7. General Settings Required(필요한 일반 설정) 섹션에서 Application label(애플리케이션 레이블)에 애플리케이션 이름을 입력합니다. 이 자습서에서는 Athena-ODBC-Okta라는 이름을 사용합니다.



8. 완료를 선택합니다.
9. Okta 애플리케이션(예: Athena-ODBC-Okta)에 대한 페이지에서 Sign On(로그인)을 선택합니다.



10. 설정(settings) 섹션에서 편집(Edit)을 선택합니다.



11. Advanced Sign-on Settings(고급 로그인 설정) 섹션에서 다음 값을 구성합니다.

    - IdP ARN 및 역할 ARN(IdP ARN and Role ARN)에 AWS IDP ARN 및 역할 ARN을 쉼표로 분리된 값으로 입력합니다. 자세한 내용은 IAM 사용 설명서의 인증 응답을 위한 SAML 어설션 구성을 참조하세요.
    - Session Duration(세션 기간)에 900~43200초 사이의 값을 입력합니다. 이 자습서에서는 기본값인 3600(1시간)을 사용합니다.

Advanced Sign-on Settings

These fields may be required for a Amazon Web Services Redshift proprietary sign-on option or general setting.

Idp ARN and Role ARN	arn:aws:iam::1234567890:saml-provid Enter your AWS IDP ARN and Role ARN as comma separated values (e.g. "arn:aws:iam::1234567890:saml-provider/OKTA,arn:aws:iam::1234567890:role/SAML_ROLE").
Session Duration	3600 Set the user's session duration in seconds here. Valid range is 900 to 43200.
DB User Format (Redshift)	${user.username} EL expression to get DB User value (e.g. "${user.username}", "${user.firstName}${user.lastName}@acme.com")
Auto Create (Redshift)	☑ AutoCreate Redshift property (Create a new database user if one does not exist)
Allowed DB Groups (Redshift)	 Comma separated list of allowed user groups. Use "*" to allow all groups, "\," to escape comma in group name

DbUser Format(DbUser 형식), AutoCreate 및 Allowed DBGroups(허용된 DBGroups) 설정은 Athena에서 사용되지 않습니다. 따라서 해당 설정을 구성할 필요가 없습니다.

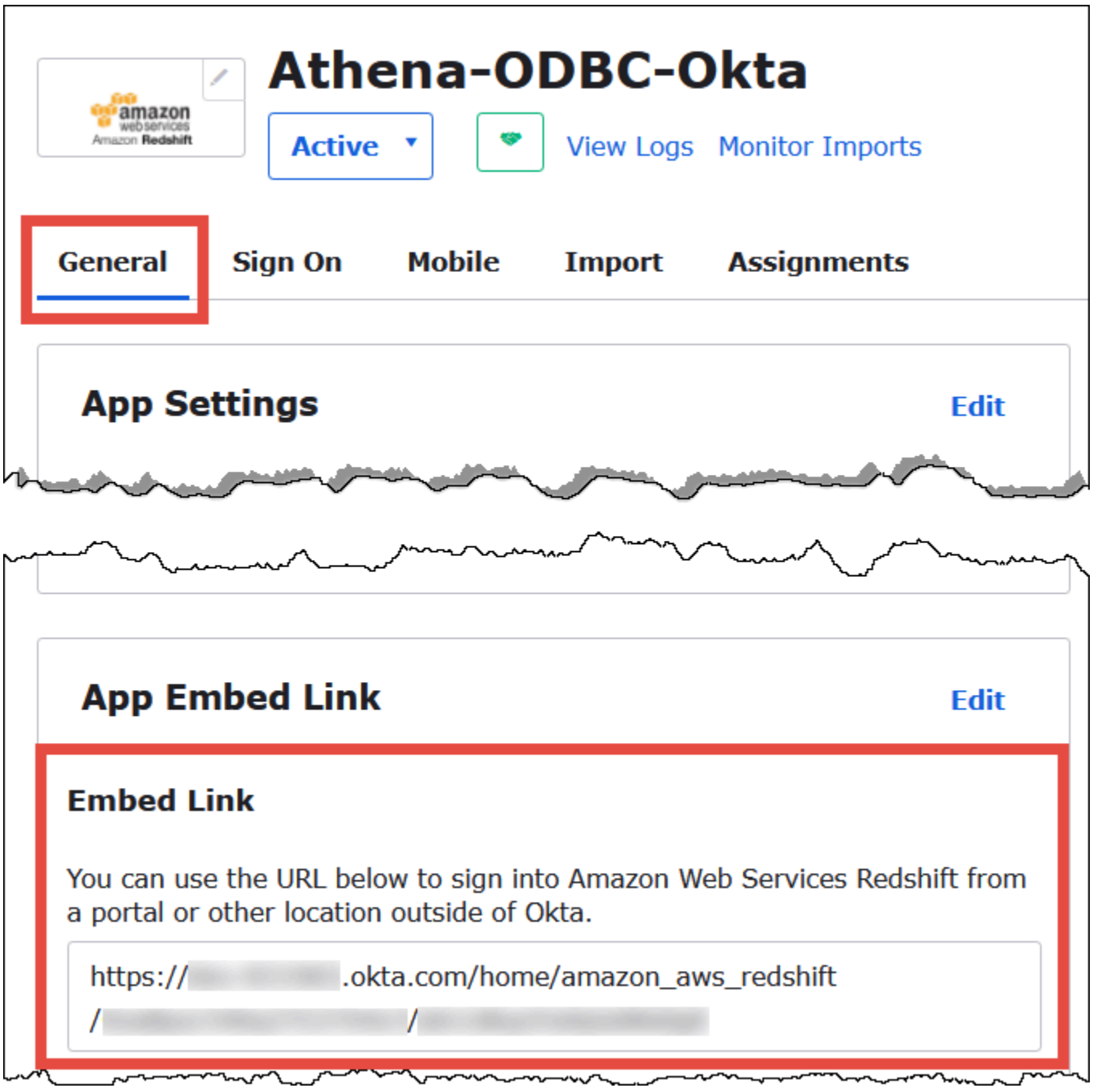
12. Save(저장)를 선택합니다.

Okta에서 ODBC 구성 정보 검색

이제 Okta 애플리케이션을 생성했으므로 애플리케이션의 ID와 IdP 호스트 URL을 검색할 준비가 되었습니다. 나중에 Athena에 연결하기 위해 ODBC를 구성할 때 필요합니다.

Okta에서 ODBC 구성 정보를 검색하려면

1. Okta 애플리케이션의 General(일반) 탭을 선택한 다음 아래로 스크롤하여 App Embed Link(앱 포함 링크)로 이동합니다.



Embed Link(포함 링크) URL의 형식은 다음과 같습니다.

```
https://trial-1234567.okta.com/home/amazon_aws_redshift/Abc1de2fghi3J45kL678/
abc1defghij2klmNo3p4
```

2. Embed Link(포함 링크) URL에서 다음 부분을 추출하여 저장합니다.

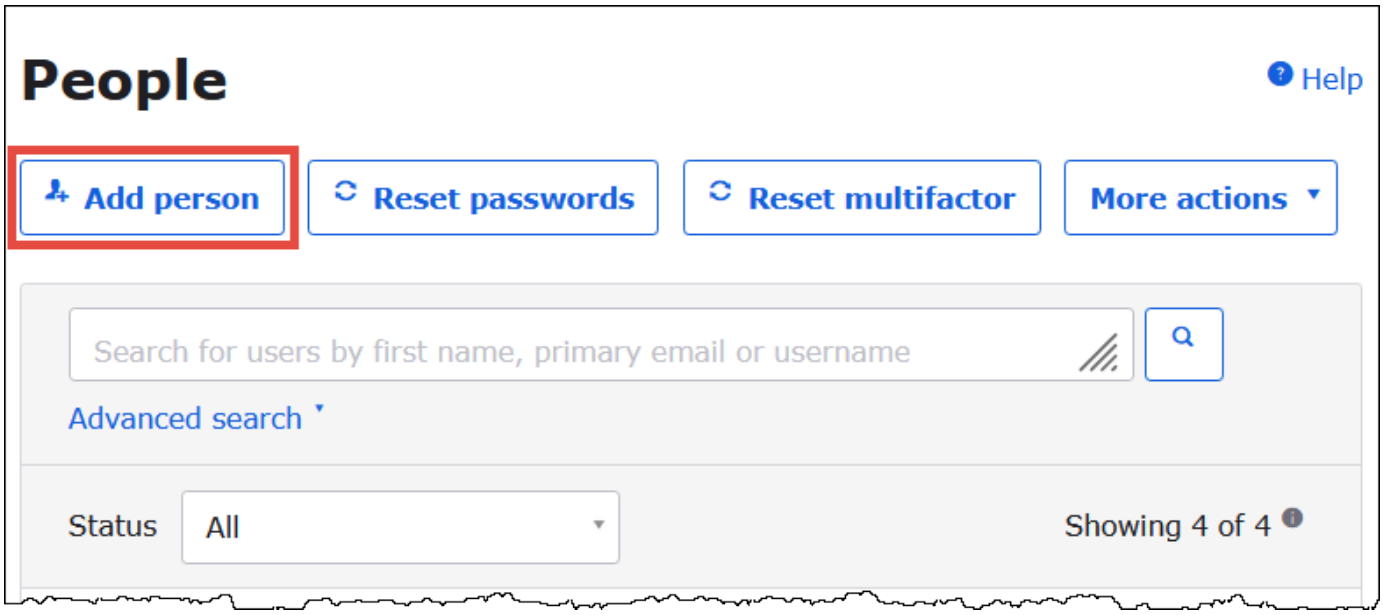
- `https://` 다음 첫 번째 세그먼트, 최대 `okta.com`까지(예: trial-1234567.okta.com). IdP 호스트입니다.
- URL의 마지막 두 세그먼트(가운데 슬래시 포함) 세그먼트는 숫자, 대문자, 소문자가 혼합된 두 개의 20자 문자열(예: ABC1DE2FGHI3J45KL678/ABC1DEFGHIJ2KLMNO3P4)입니다. 애플리케이션 ID입니다.

Okta 애플리케이션에 사용자 추가

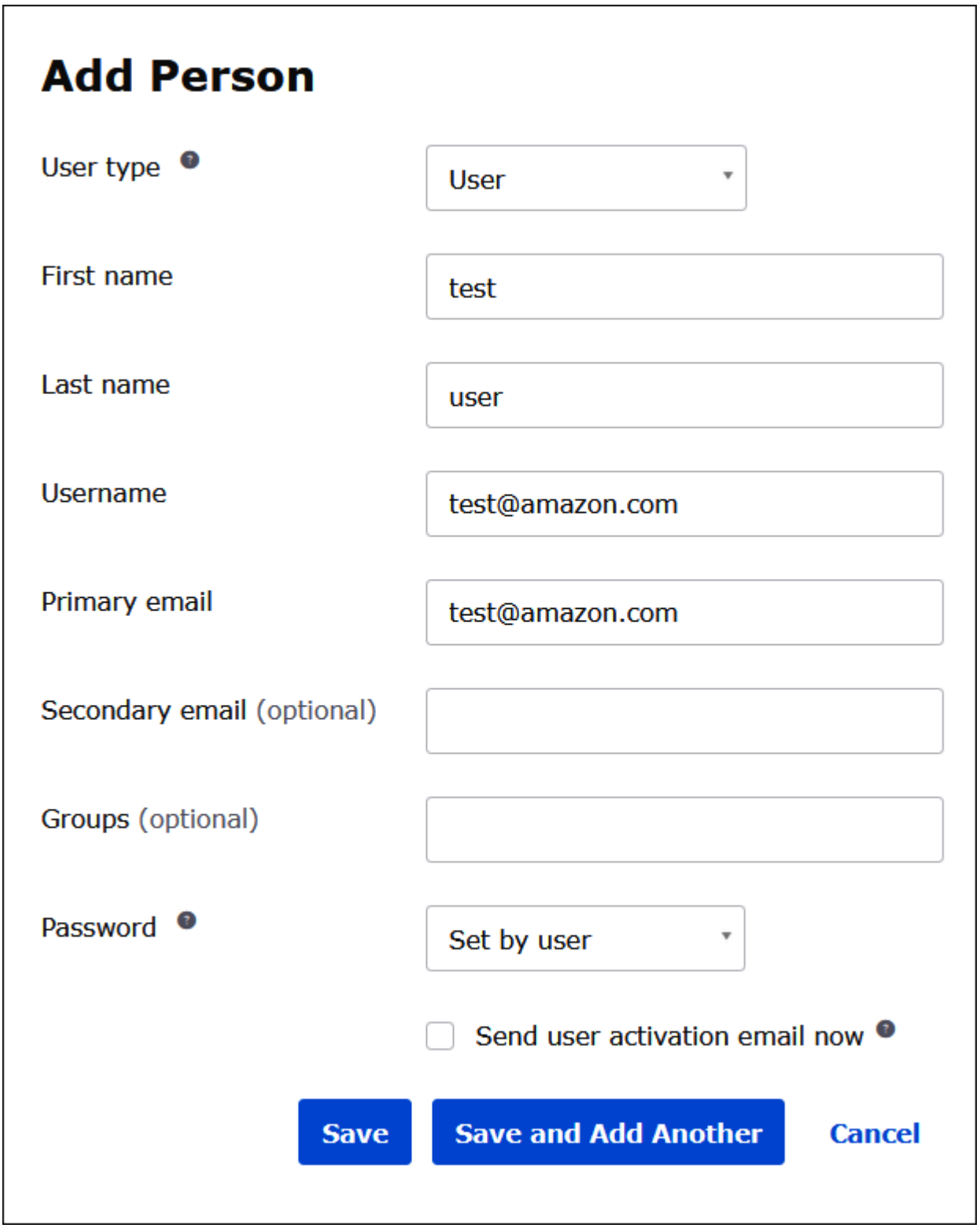
이제 Okta 애플리케이션에 사용자를 추가할 준비가 되었습니다.

Okta 애플리케이션에 사용자를 추가하려면

1. 왼쪽 탐색 창에서 디렉터리(Directory)와 사람(People)을 차례로 선택합니다.
2. 사람 추가(Add Person)를 선택합니다.



3. 사람 추가(Add Person) 대화 상자에 다음 정보를 입력합니다.
   - 이름(First name)과 성(Last name)에 값을 입력합니다. 이 자습서에서는 **`test user`**를 사용합니다.
   - 사용자 이름(Username) 및 기본 이메일(Primary email) 값을 입력합니다. 이 자습서에서는 두 항목 모두 **`test@amazon.com`**을 사용합니다. 암호에 대한 보안 요구 사항은 다를 수 있습니다.

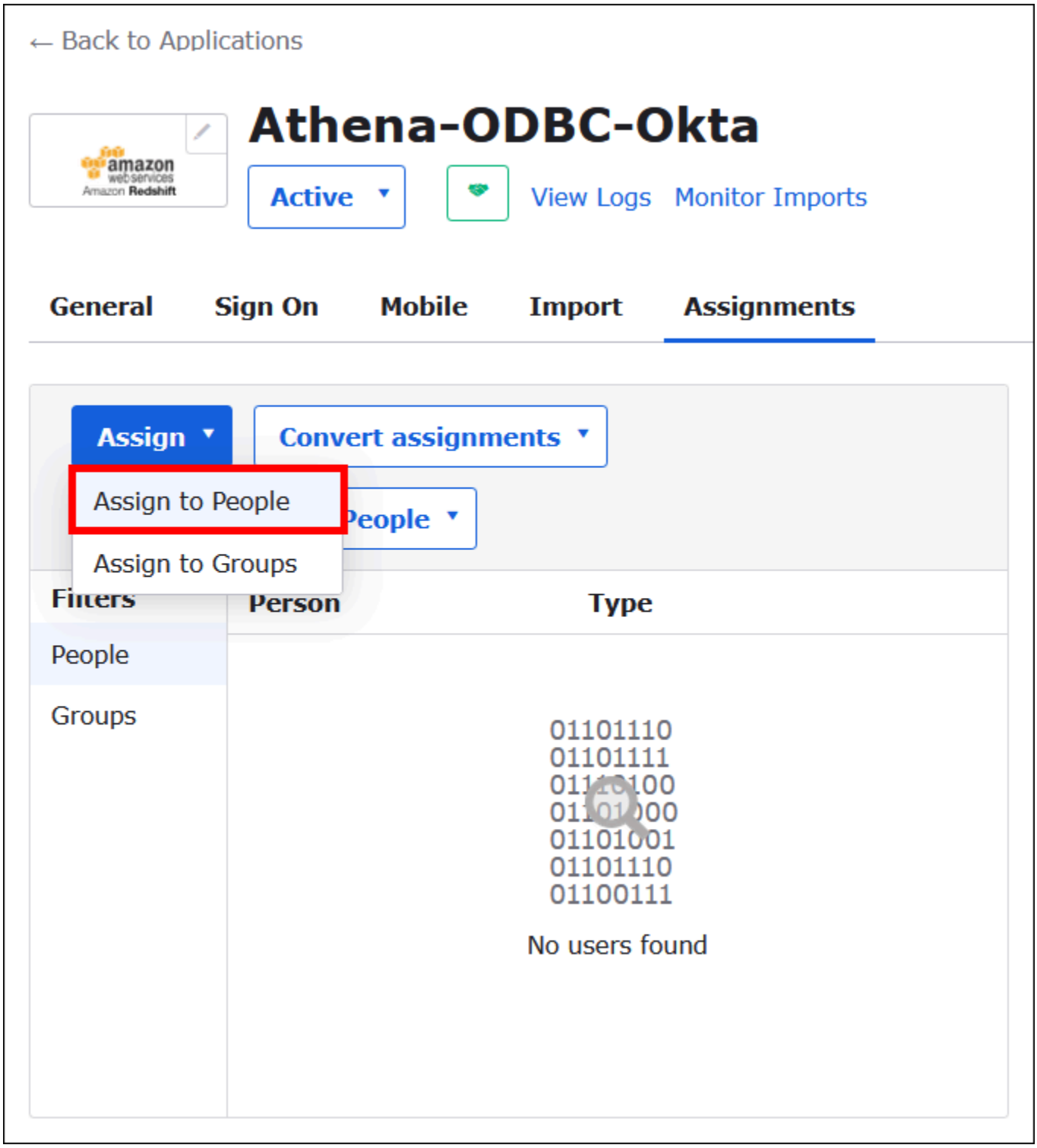


4. Save(저장)를 선택합니다.

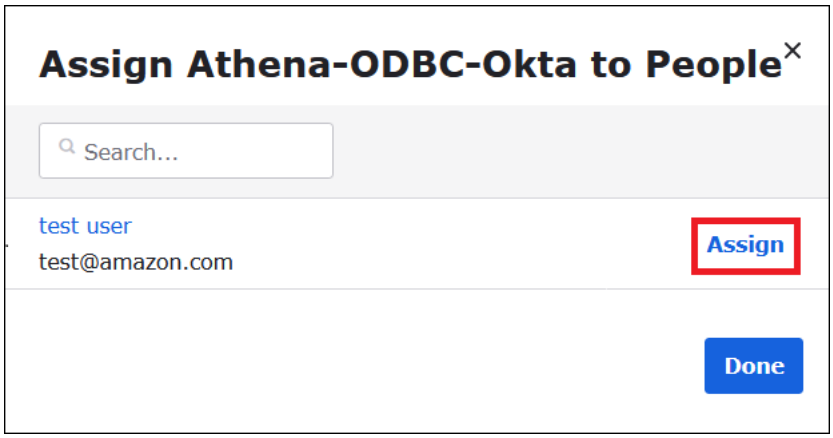
이제 생성된 사용자를 애플리케이션에 할당할 준비가 되었습니다.

애플리케이션에 사용자를 할당하려면

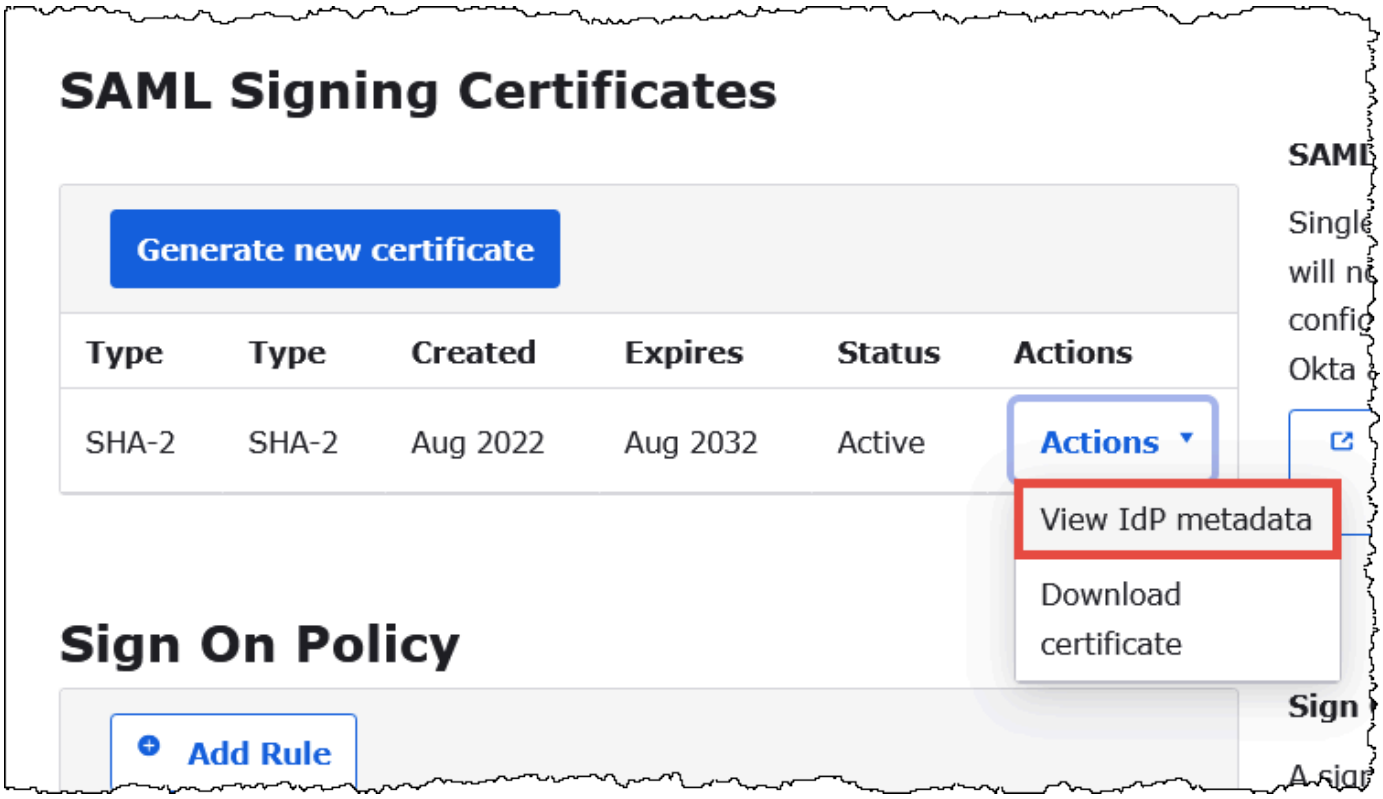
1. 탐색 창에서 Applications(애플리케이션), Applications(애플리케이션)를 선택한 다음 애플리케이션 이름(예: Athena-ODBC-Okta)을 선택합니다.
2. Assign(할당)을 선택한 다음 Assign to People(사람에게 할당)을 선택합니다.



3. 사용자에 대한 Assign(할당) 옵션을 선택한 다음 Done(완료)을 선택합니다.



4. 프롬프트에서 Save and Go Back(저장하고 돌아가기)을 선택합니다. 대화 상자에 사용자의 상태가 Assigned(할당됨)로 표시됩니다.
5. 완료(Done)를 선택합니다.
6. Sign On(로그인) 탭을 선택합니다.
7. 아래로 스크롤하여 SAML 서명 인증서(SAML Signing Certificates) 섹션으로 이동합니다.
8. 작업을 선택합니다.
9. IdP 메타데이터 보기(View IdP metadata)에 대한 컨텍스트(마우스 오른쪽 버튼 클릭) 메뉴를 연 다음 파일을 저장할 브라우저 옵션을 선택합니다.
10. 파일을 `.xml` 확장명으로 저장합니다.

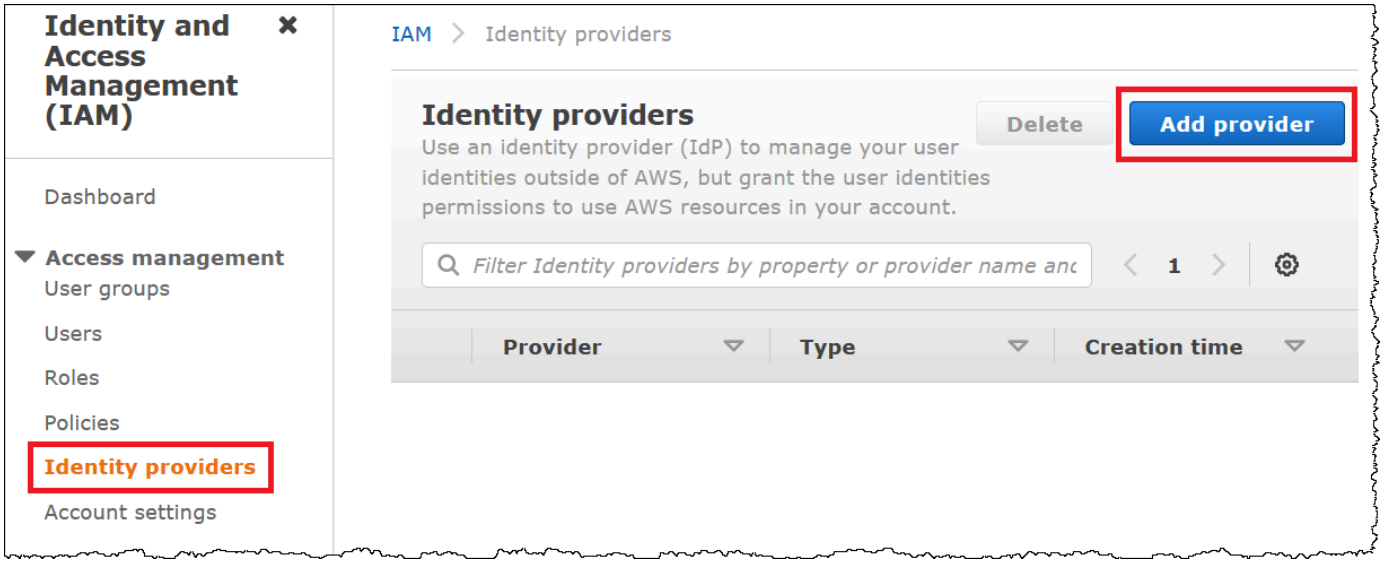


AWS SAML 자격 증명 공급자 및 역할 생성

이제 AWS의 IAM 콘솔에 메타데이터 XML 파일을 업로드할 준비가 되었습니다. 이 파일을 사용하여 AWS SAML 자격 증명 공급자 및 역할을 생성합니다. AWS 서비스 관리자 계정을 사용하여 이러한 단계를 수행합니다.

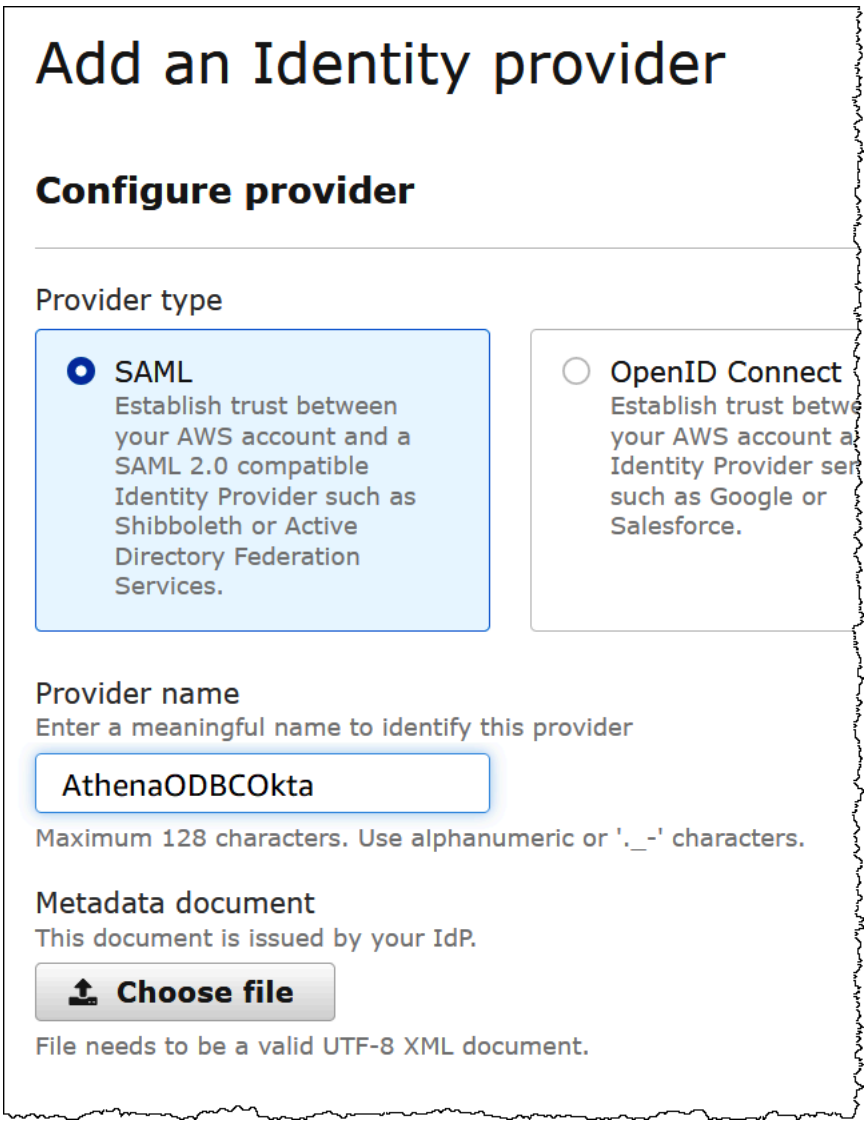
AWS에서 SAML 자격 증명 공급자 및 역할을 생성하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/IAM/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 [자격 증명 공급자(Identity providers)]를 선택한 다음 [공급자 추가(Add provider)]를 선택합니다.



3. 자격 증명 공급자 추가(Add an Identity provider) 창에서 공급자 구성(Configure provider)에 다음 정보를 입력합니다.

   - 공급자 유형(Provider type)에 SAML을 선택합니다.
   - 공급자 이름(Provider name)에 공급자의 이름을 입력합니다(예: **`AthenaODBCOkta`**).
   - 메타데이터 문서(Metadata document)에서 파일 선택(Choose file) 옵션을 사용하여 다운로드한 자격 증명 공급자(IdP) 메타데이터 XML 파일을 업로드합니다.



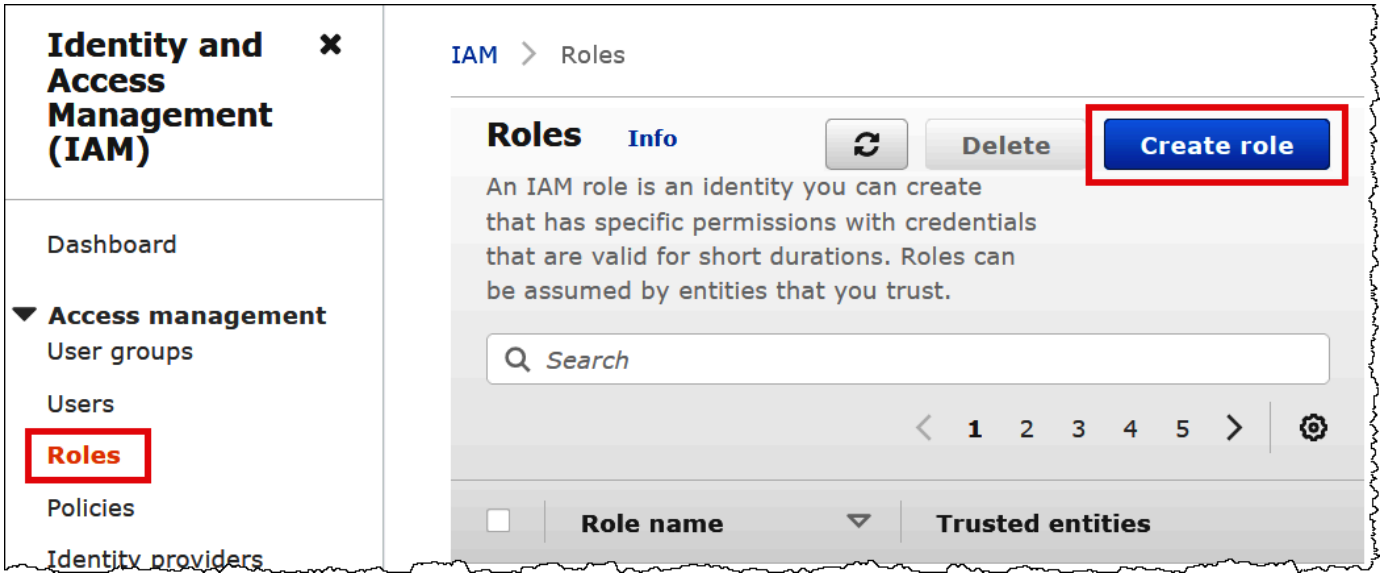
4. 공급자 추가(Add Provider)를 선택합니다.

Athena 및 Amazon S3 액세스를 위한 IAM 역할 생성

이제 Aurora 및 Amazon S3 액세스를 위한 IAM 역할을 생성할 준비가 되었습니다. 이 역할을 사용자에게 할당합니다. 이렇게 하면 사용자에게 Athena에 대한 Single Sign-On 액세스 권한을 제공할 수 있습니다.

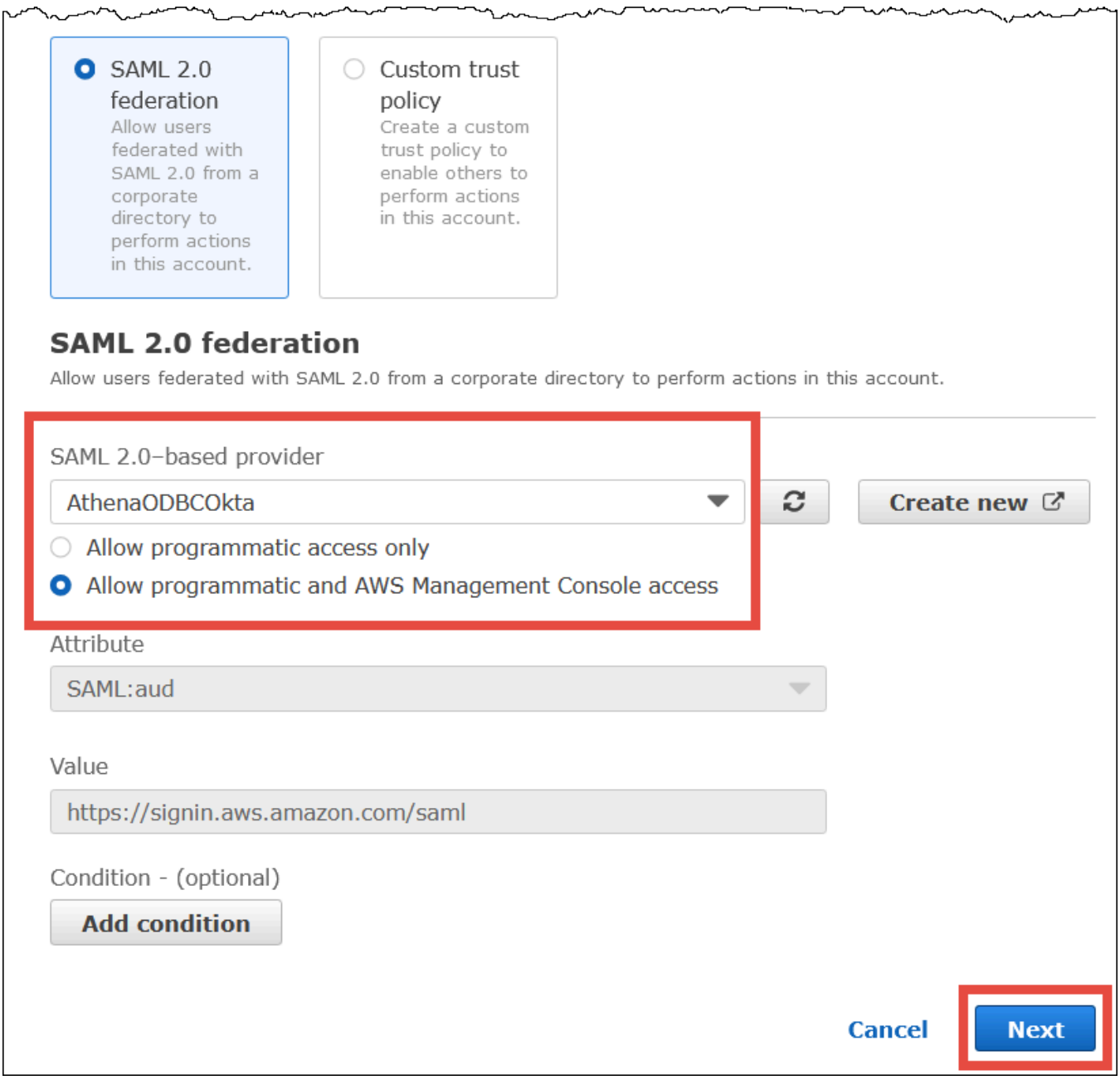
사용자에 대한 IAM 역할을 생성하려면

1. IAM 콘솔의 탐색 창에서 역할(Roles)을 선택하고 역할 생성(Create role)을 선택합니다.

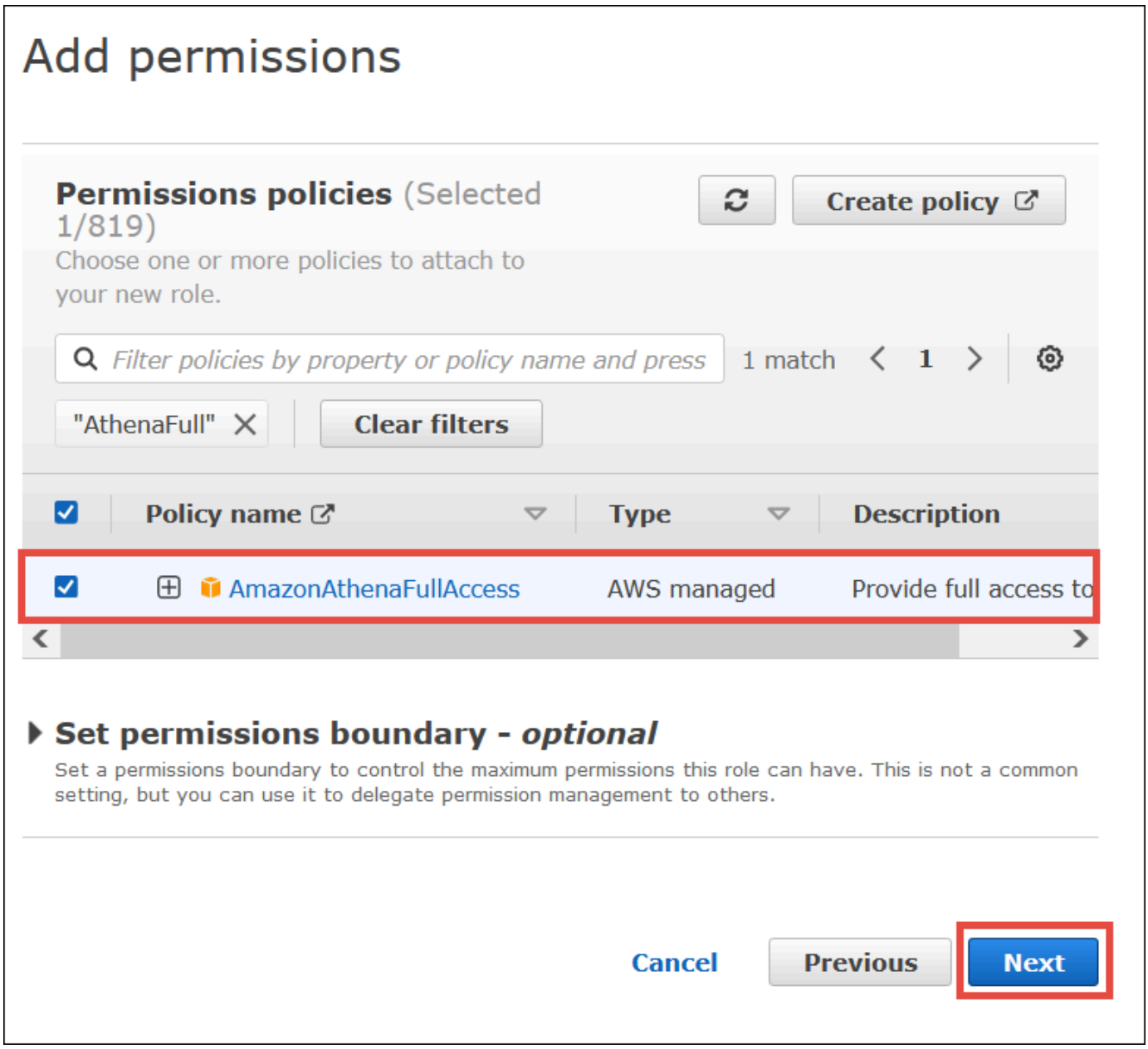


2. 역할 생성(Create role) 페이지에서 다음 옵션을 선택합니다.

   - 신뢰할 수 있는 엔터티 유형 선택(Select type of trusted entity)에서 SAML 2.0 연합(SAML 2.0 Federation)을 선택합니다.
   - SAML 2.0 기반 공급자(SAML 2.0–based provider)에서 생성한 SAML 자격 증명 공급자(예: AthenaODBCOkta)를 선택합니다.
   - Allow programmatic and AWS Management Console access를 선택합니다.



3. 다음(Next)을 선택합니다.
4. 권한 추가(Add Permissions) 페이지에서 필터 정책(Filter policies)에 **`AthenaFull`**를 입력한 다음 Enter 키를 누릅니다.
5. `AmazonAthenaFullAccess` 관리형 정책을 선택하고 Next(다음)를 선택합니다.



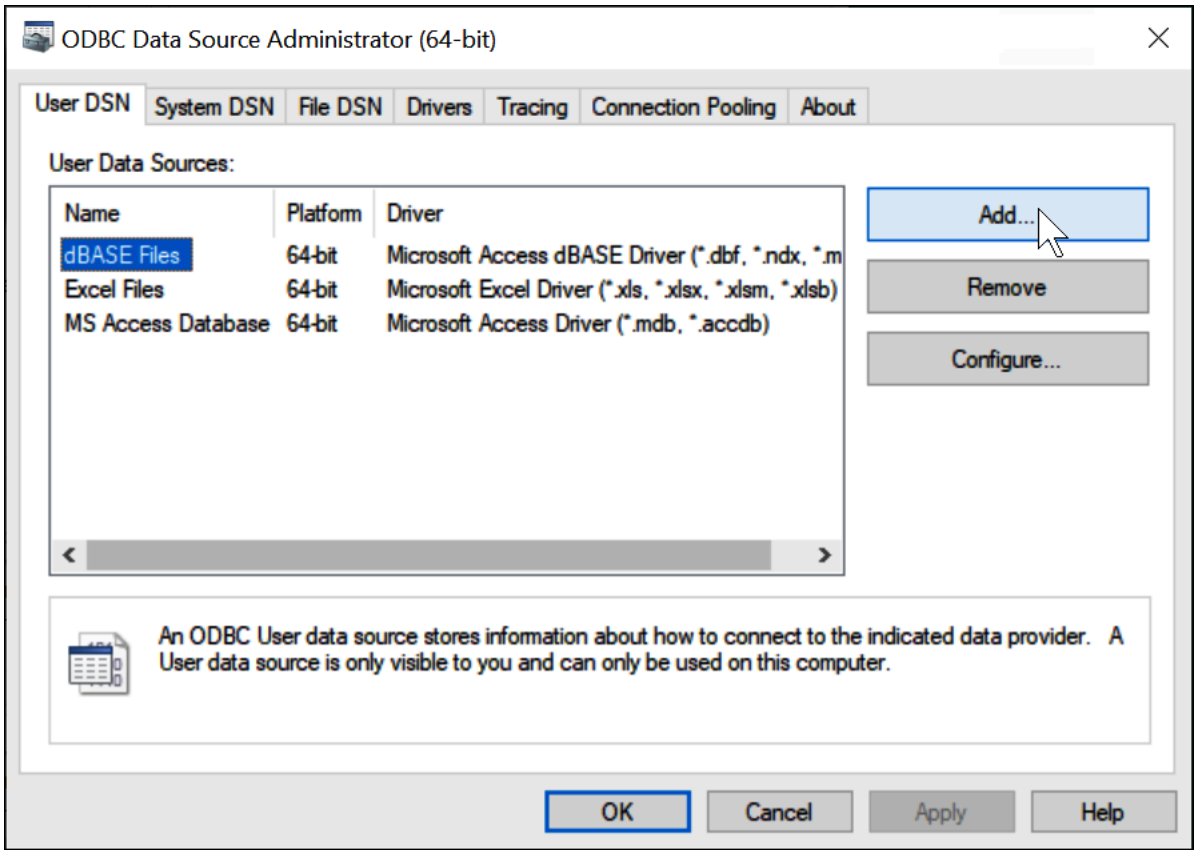
6. 이름 지정, 검토 및 생성(Name, review, and create) 페이지의 역할 이름(Role name)에 역할의 이름(예: **`Athena-ODBC-OktaRole`**)을 입력한 후 역할 생성(Create role)을 선택합니다.

Athena에 대한 SAML ODBC 연결 구성

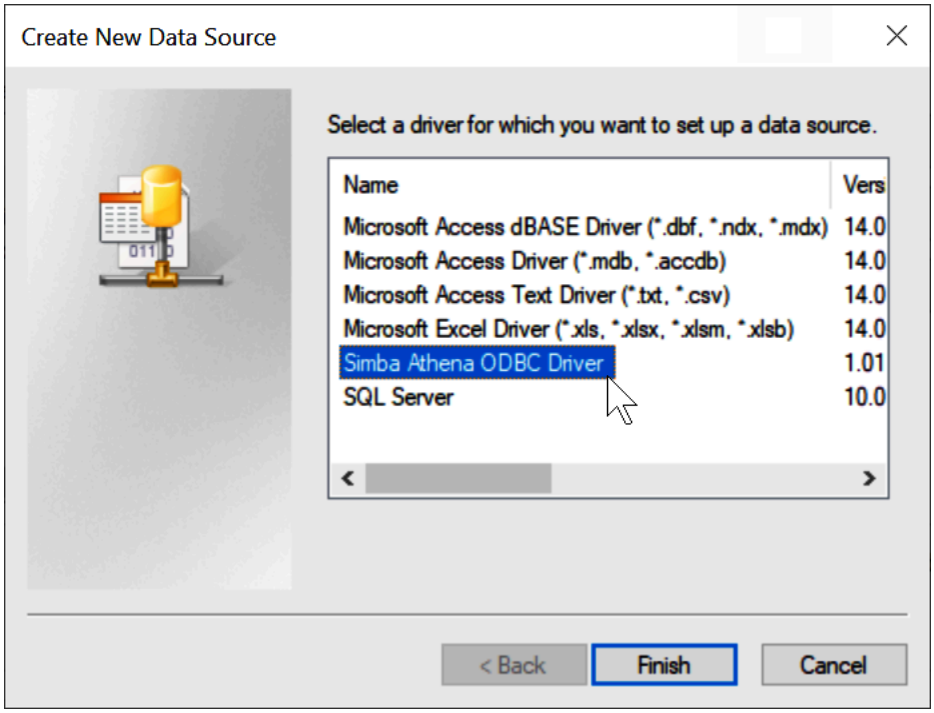
이제 Windows의 ODBC 데이터 소스 프로그램을 사용하여 Athena 대한 Okta ODBC 연결을 구성할 준비가 되었습니다.

Athena에 대한 Okta ODBC 연결을 구성하려면

1. Windows에서 ODBC 데이터 원본(ODBC Data Sources) 프로그램을 시작합니다.
2. ODBC 데이터 원본 관리자(ODBC Data Source Administrator) 프로그램에서 추가(Add)를 선택합니다.

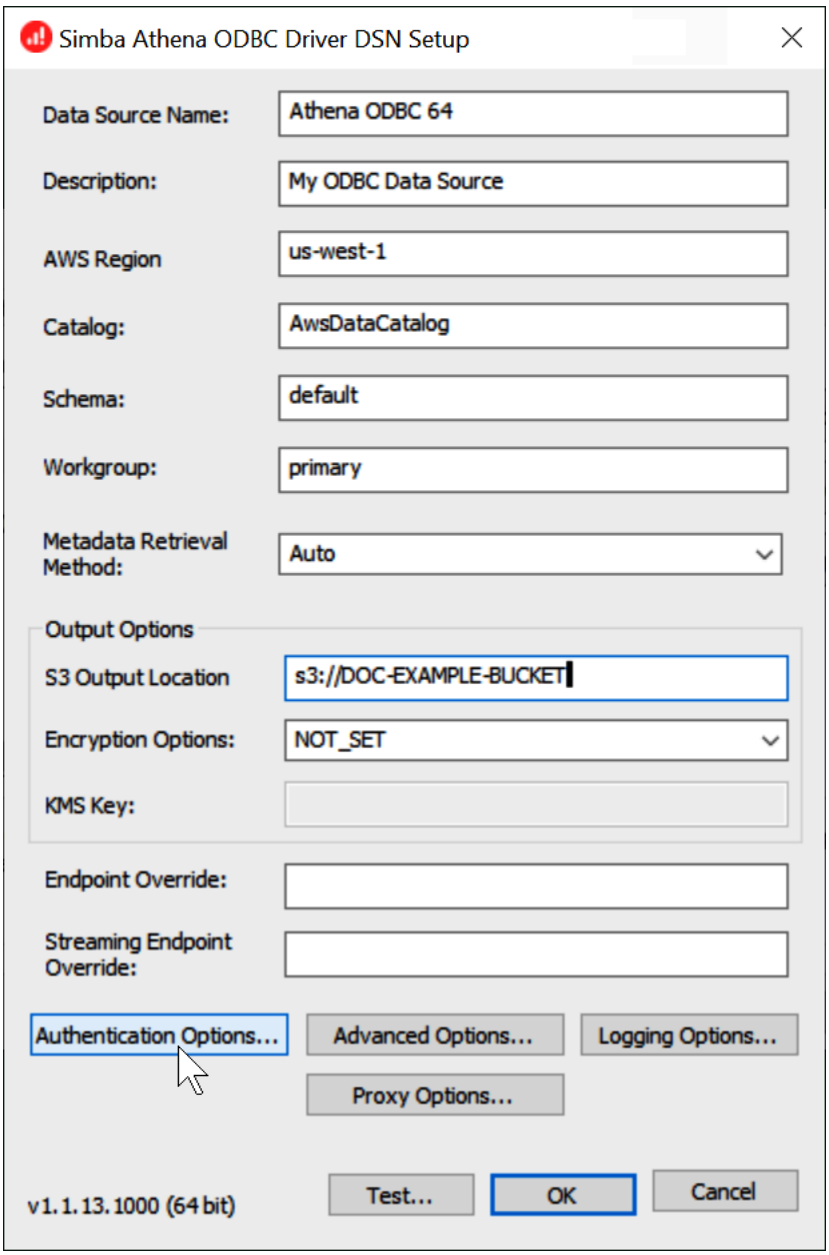


3. Simba Athena ODBC 드라이버(Simba Athena ODBC Driver)를 선택하고 완료(Finish)를 선택합니다.



4. Simba Athena ODBC 드라이버 DSN 설치(Simba Athena ODBC Driver DSN Setup) 대화 상자에서 설명된 값을 입력합니다.

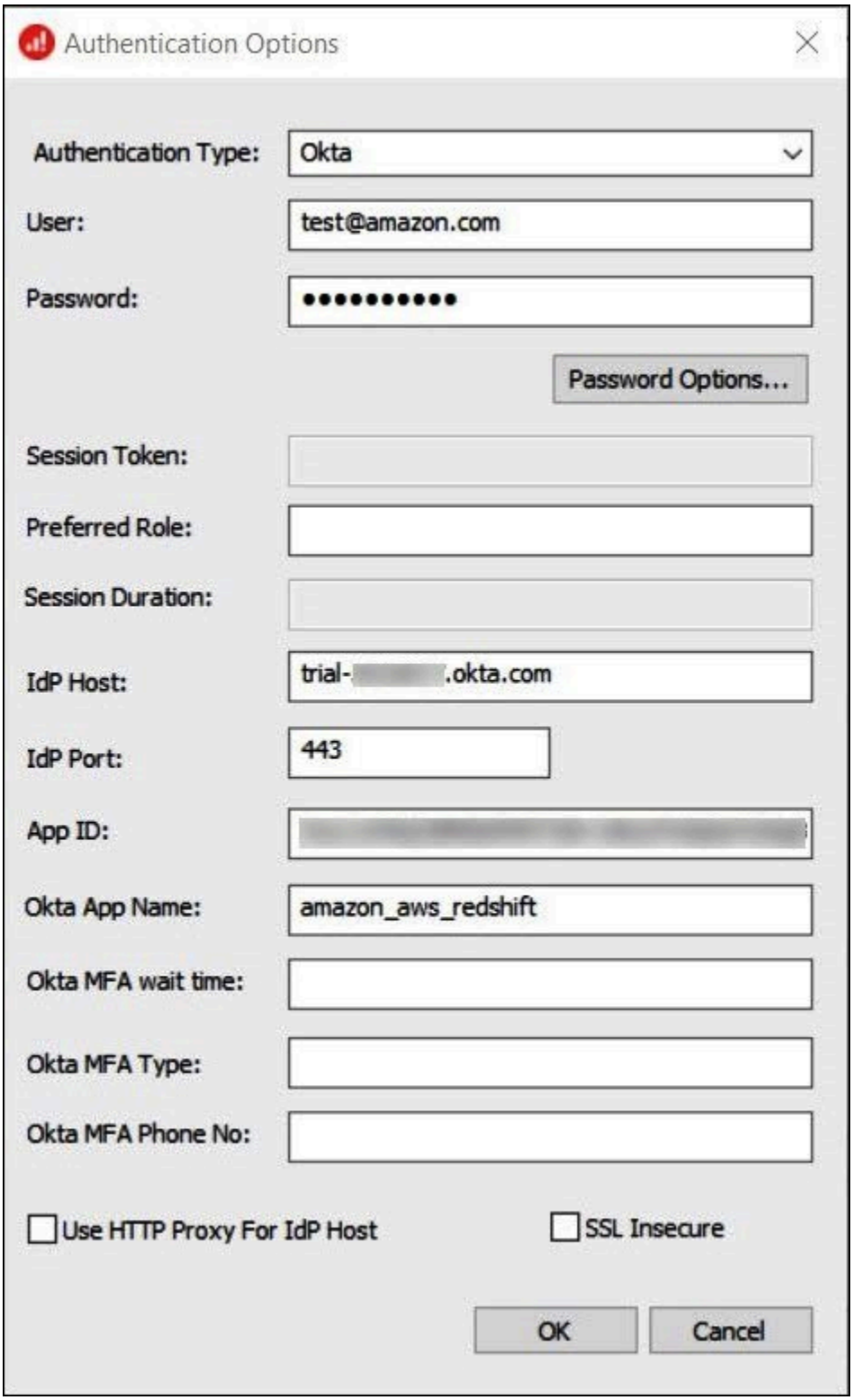
   - 데이터 소스 이름(Data Source Name)에 데이터 소스 이름을 입력합니다(예: **`Athena ODBC 64`**).
   - 설명(Description)에 데이터 원본에 대한 설명을 입력합니다.
   - AWS 리전에서 사용 중인 AWS 리전을 입력합니다(예: **`us-west-1`**).
   - S3 출력 위치(S3 Output Location)에 출력을 저장할 Amazon S3 경로를 입력합니다.



5. 인증 옵션(Authentication Options)을 선택합니다.
6. 인증 옵션(Authentication Options) 대화 상자에서 다음 값을 선택하거나 입력합니다.

   - 인증 유형(Authentication Type)에서 Okta를 선택합니다.
   - 사용자에 Okta 사용자 이름을 입력합니다.
   - 암호에 Okta 암호를 입력합니다.
   - IdP 호스트(IdP Host)에서 이전에 기록한 값(예: **`trial-1234567.okta.com`**)을 입력합니다.
   - IdP 포트(IdP Port)에 **`443`**을 입력합니다.
   - 앱 ID(App ID)에서 이전에 기록한 값(Okta 포함 링크의 마지막 두 세그먼트)을 입력합니다.

- Okta 앱 이름(Okta App Name)에 **`amazon_aws_redshift`**를 입력합니다.



7. 확인(OK)을 선택합니다.
8. 테스트(Test)를 선택하여 연결을 테스트하거나 확인(OK)을 선택하여 마칩니다.

ODBC, SAML 2.0 및 Okta 자격 증명 공급자를 사용하여 통합 인증 구성

데이터 원본에 연결하기 위해 Amazon Athena를 PingOne, Okta, OneLogin 등의 자격 증명 공급자(IdP)와 함께 사용할 수 있습니다. Athena ODBC 드라이버 버전 1.1.13 및 Athena JDBC 드라이버 버전 2.0.25부터 모든 SAML 2.0 공급자와 함께 작동하도록 구성할 수 있는 브라우저 SAML 플러그 인이 포함되어 있습니다. 이 항목에서는 Okta 자격 증명 공급자를 사용하여 통합 인증(SSO) 기능을 추가하기 위해 Amazon Athena ODBC 드라이버와 브라우저 기반 SAML 플러그인을 구성하는 방법을 보여줍니다.

필수 조건

이 자습서의 단계를 완료하려면 다음이 필요합니다.

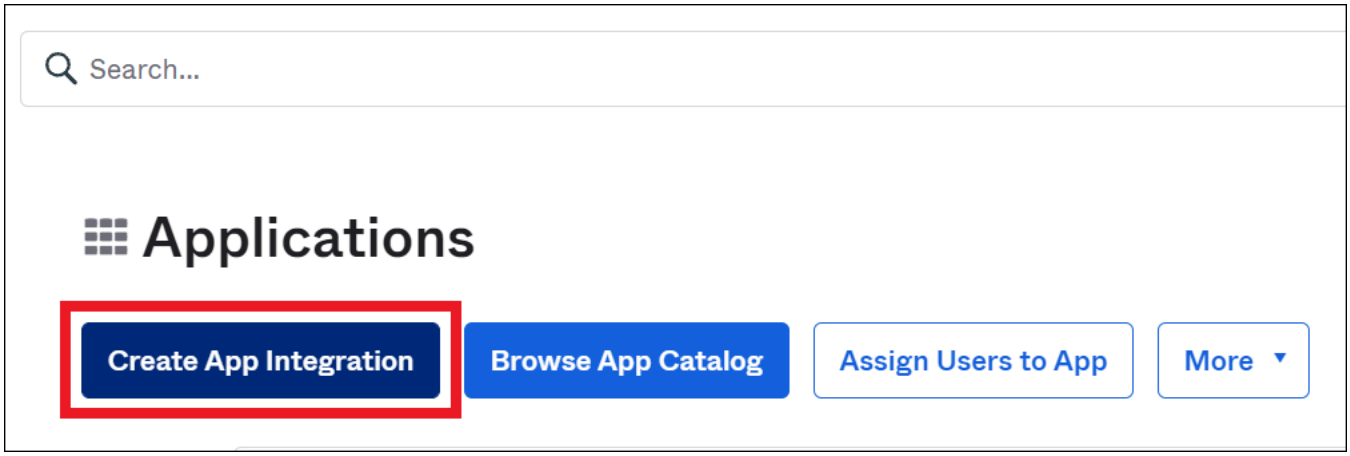
- Athena ODBC 드라이버 버전 1.1.13 이상. 버전 1.1.13 이상에는 브라우저 SAML 지원이 포함되어 있습니다. 자세한 내용은 ODBC로 Amazon Athena에 연결을 참조하세요.
- SAML에서 사용하려는 IAM 역할. 자세한 내용은 IAM 사용자 설명서의 SAML 2.0 페더레이션을 위한 역할 생성을 참조하세요.
- Okta 계정. 자세한 내용을 보려면 okta.com을 방문하세요.

Okta에서 앱 통합 생성

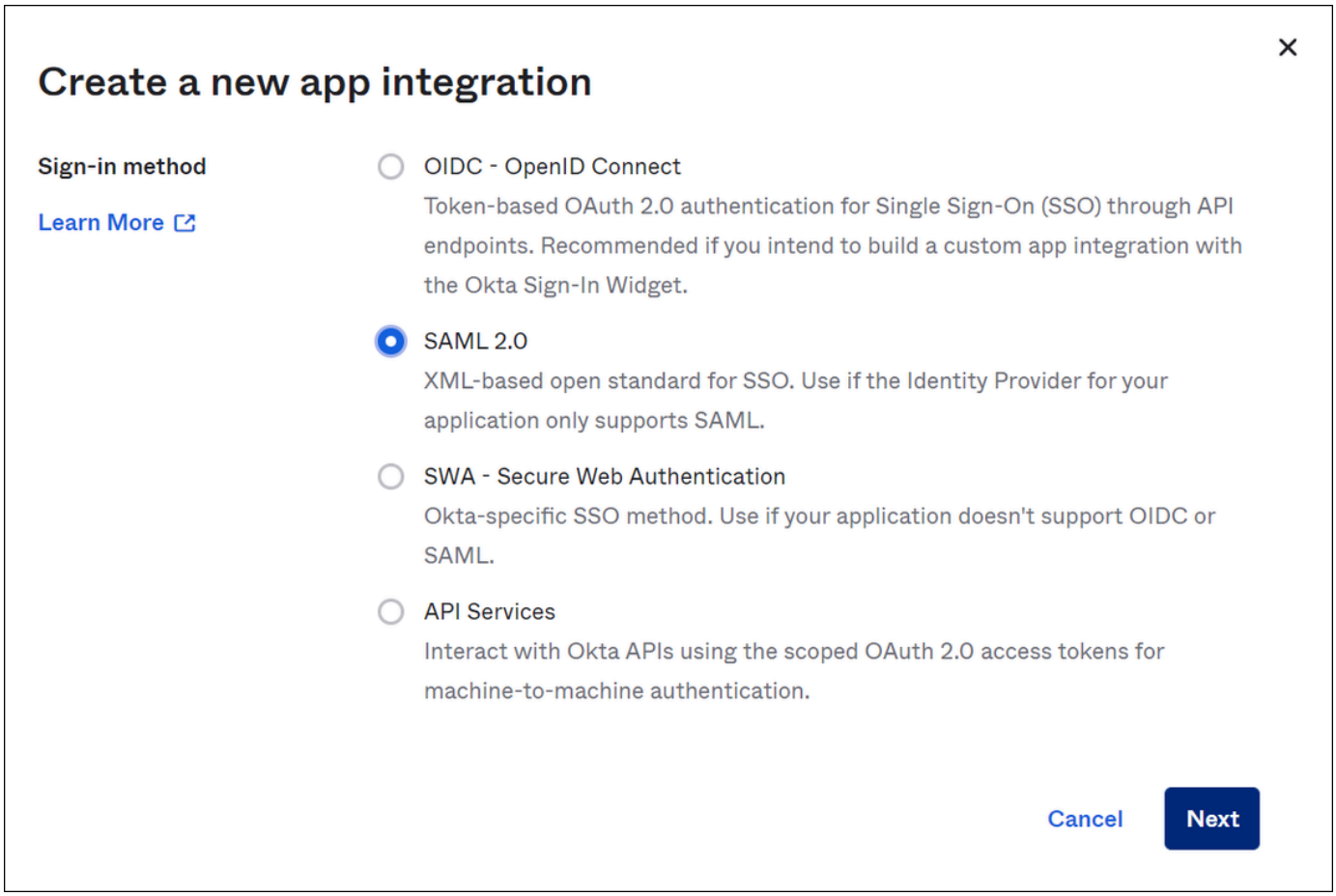
먼저 Okta 대시보드를 사용하여 Athena에 대한 통합 인증을 위한 SAML 2.0 앱을 생성하고 구성합니다.

Okta 대시보드를 사용하여 Athena용 통합 인증 설정

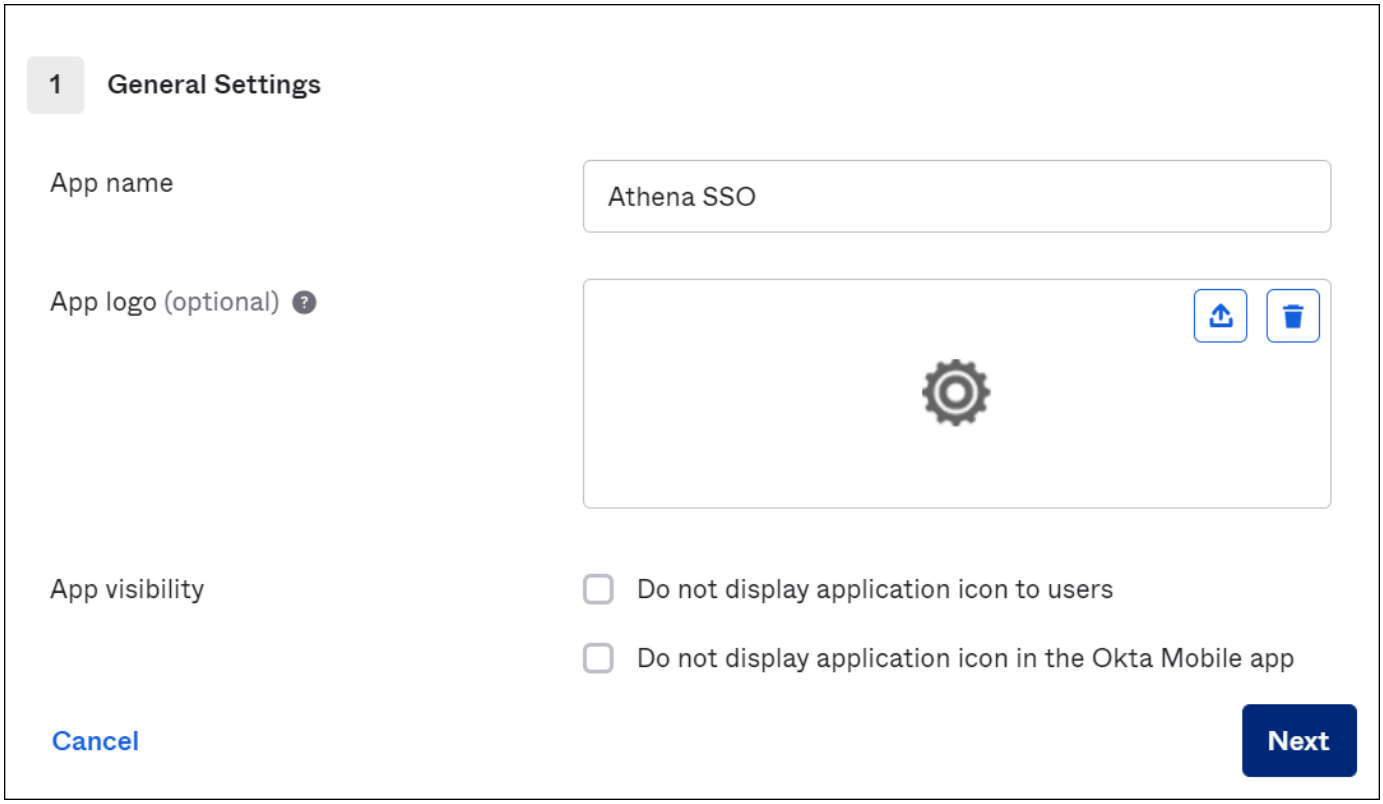
1. `okta.com`에서 Okta 관리 페이지에 로그인합니다.
2. 탐색 창에서 애플리케이션(Applications), 애플리케이션(Applications)을 선택합니다.
3. 애플리케이션 페이지에서 애플리케이션 통합 생성(Create App Integration)을 선택합니다.



4. 새 앱 통합 생성(Create a new app integration) 대화 상자에서 로그인 방법(Sign-in method)에 대해 SAML 2.0을 선택하고 다음(Next)을 선택합니다.

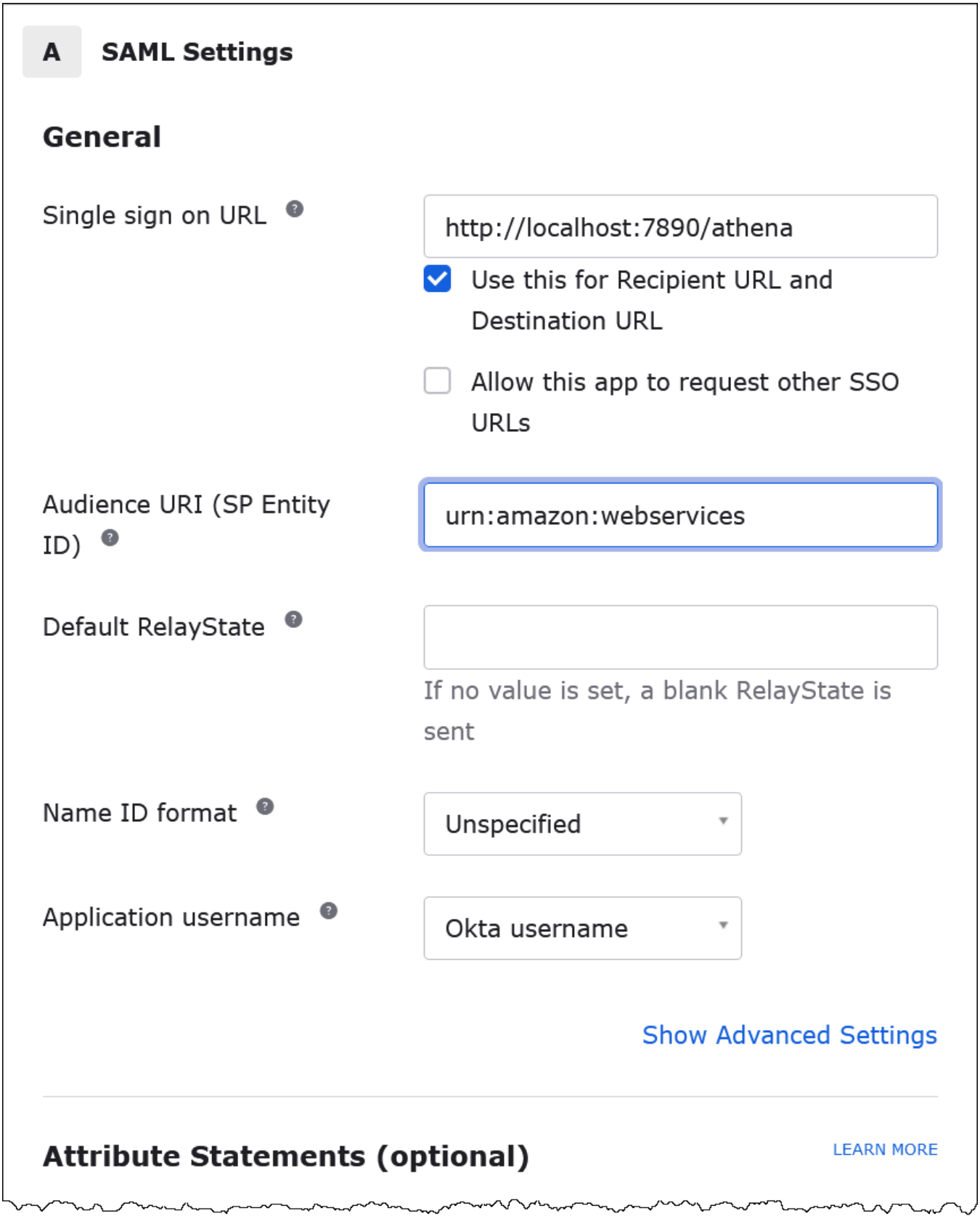


5. SAML 통합 생성(Create SAML Integration) 페이지의 일반 설정(General Settings) 섹션에서 애플리케이션 이름을 입력합니다. 이 자습서에서는 Athena SSO라는 이름을 사용합니다.



6. 다음을 선택합니다.
7. SAML 구성(Configure SAML) 페이지의 SAML 설정(SAML Settings) 섹션에서 다음 값을 입력합니다.

   - 통합 인증 URL(Single sign on URL)에 **`http://localhost:7890/athena`**를 입력합니다.
   - 대상 URI(Audience URI)에 **`urn:amazon:webservices`**를 입력합니다.


A
SAML Settings
General
Single sign on URL
http://localhost:7890/athena
Use this for Recipient URL and Destination URL
Allow this app to request other SSO URLs
Audience URI (SP Entity ID)
urn:amazon:webservices
Default RelayState
If no value is set, a blank RelayState is sent
Name ID format
Unspecified
Application username
Okta username
Show Advanced Settings
Attribute Statements (optional)
LEARN MORE

8. 속성 문(Attribute Statements)(선택 사항)에 다음 2개의 이름/값 페어를 입력합니다. 필수 매핑 속성입니다.

   - 이름(Name)에 다음 URL을 입력합니다.

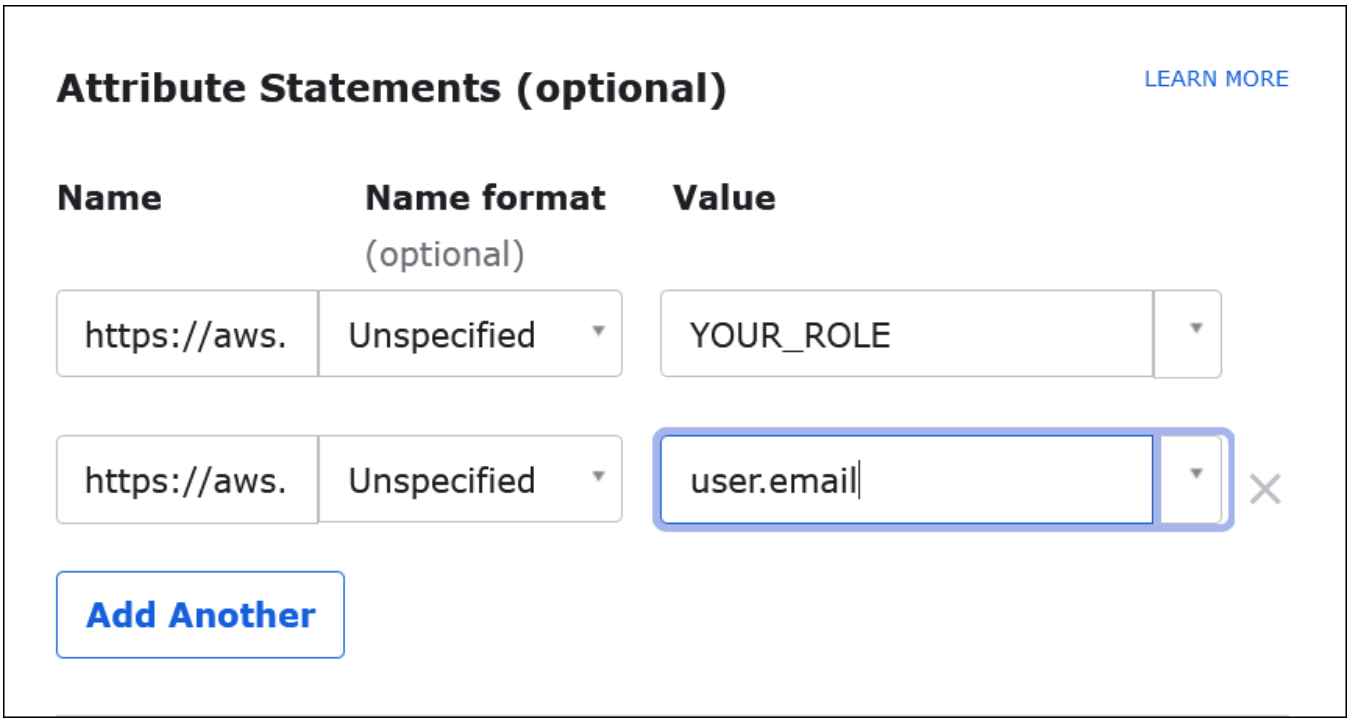
     `https://aws.amazon.com/SAML/Attributes/Role`

     값(Value)에 IAM 역할의 이름을 입력합니다. 자세한 내용은 IAM 사용 설명서의 인증 응답을 위한 SAML 어설션 구성을 참조하세요.

   - 이름(Name)에 다음 URL을 입력합니다.

     `https://aws.amazon.com/SAML/Attributes/RoleSessionName`

     값에 **`user.email`**를 입력합니다.



9. 다음을 선택한 후 완료를 선택합니다.

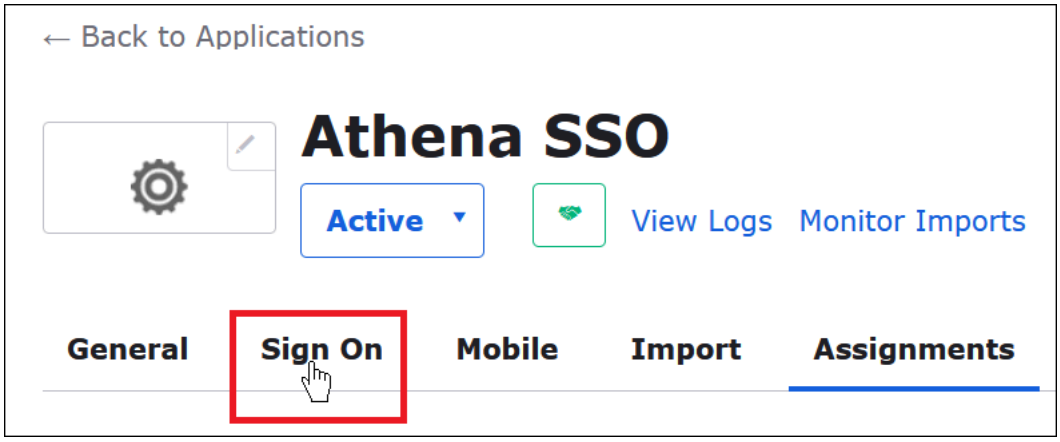
   Okta는 애플리케이션을 생성할 때 다음에 검색할 로그인 URL도 생성합니다.

Okta 대시보드에서 로그인 URL 가져오기

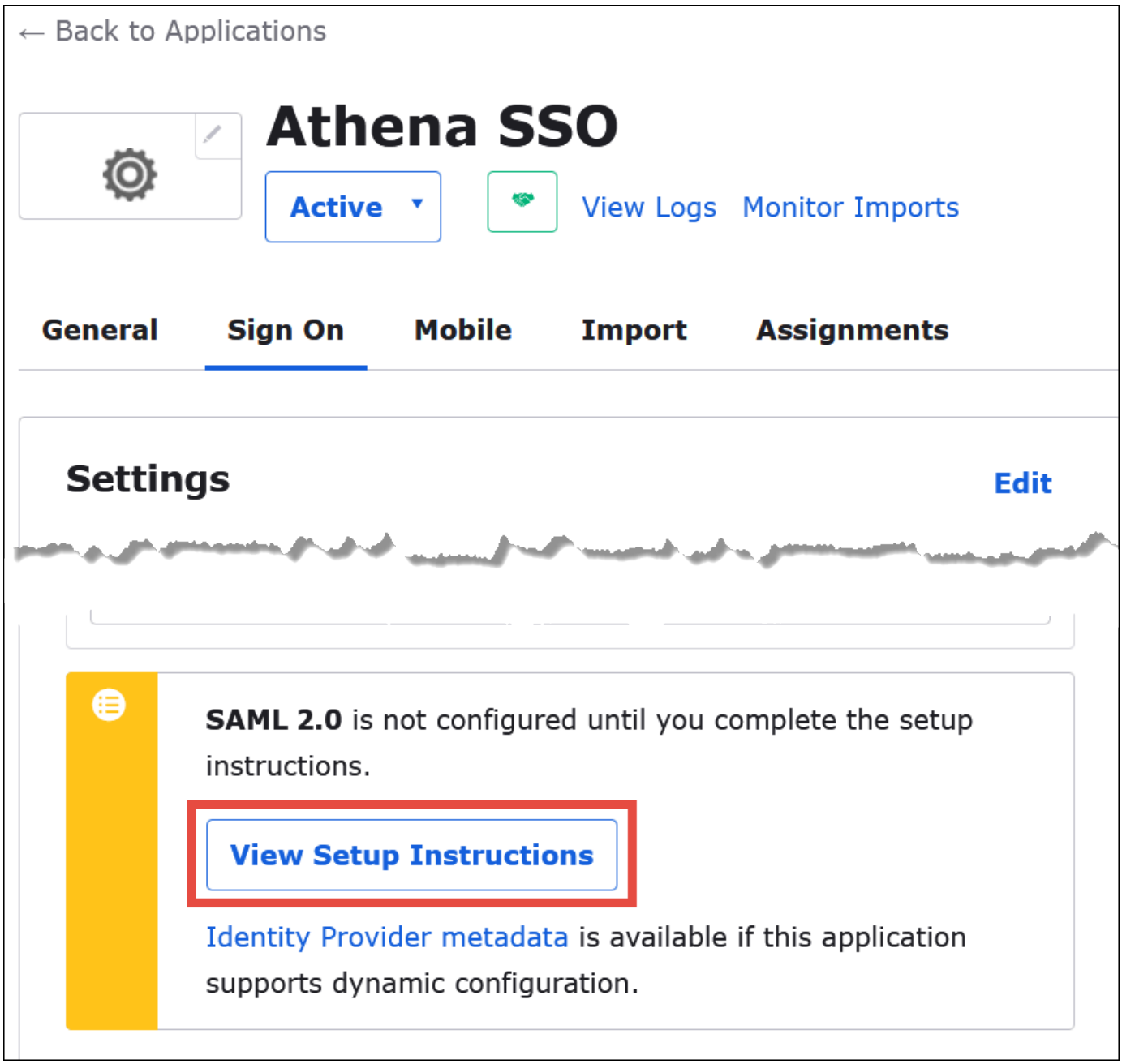
애플리케이션이 생성되었으므로 Okta 대시보드에서 로그인 URL과 기타 메타데이터를 얻을 수 있습니다.

Okta 대시보드에서 로그인 URL 가져오기

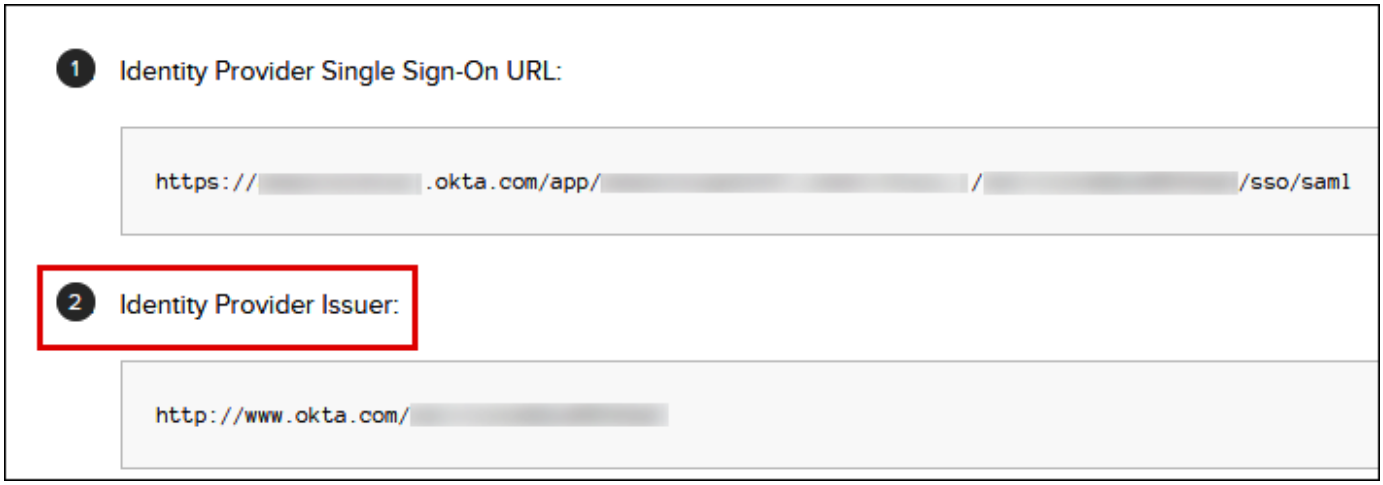
1. Okta 탐색 창에서 애플리케이션(Applications), 애플리케이션(Applications)을 선택합니다.
2. 로그인 URL을 찾을 애플리케이션을 선택합니다(예: AthenaSSO).
3. 애플리케이션 페이지에서 사인온(Sign On)을 선택합니다.



4. 설정 지침 보기(View Setup Instructions)를 선택합니다.



5. Athena SSO를 위한 SAML 2.0을 구성하는 방법(How to Configure SAML 2.0 for Athena SSO) 페이지에서 자격 증명 공급자 발급자(Identity Provider Issuer)의 URL을 찾습니다. Okta 대시보드의 일부 위치는 이 URL을 SAML 발급자 ID(SAML issuer ID)로 참조합니다.



6. ID 공급자 Single Sign-On의 값을 복사하거나 저장합니다.

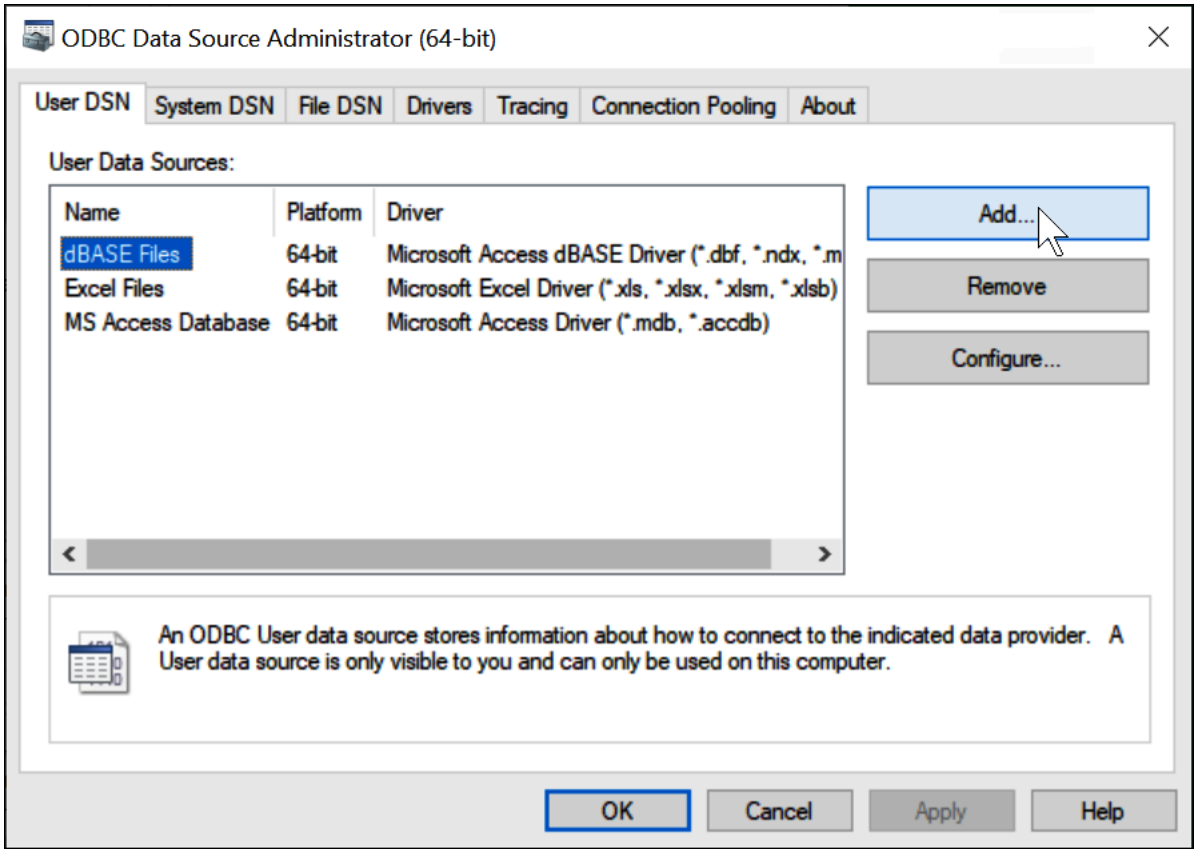
   다음 섹션에서는 ODBC 연결을 구성할 때 이 값을 브라우저 SAML 플러그 인에 대한 로그인 URL(Login URL) 연결 파라미터로 제공합니다.

Athena에 대한 브라우저 SAML ODBC 연결 구성

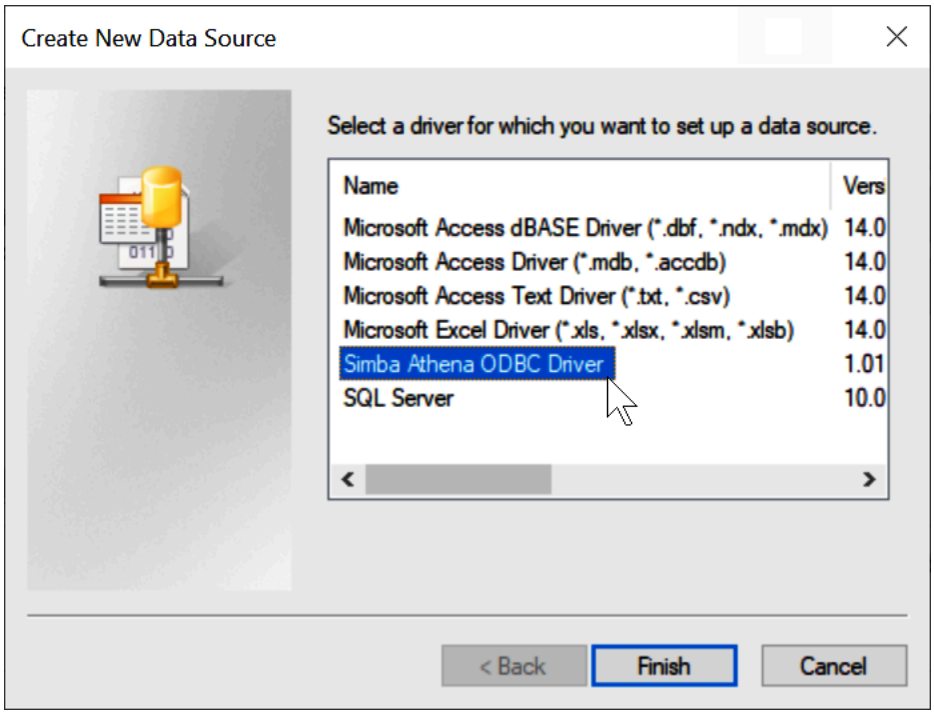
이제 Windows의 ODBC 데이터 원본 프로그램을 사용하여 Athena 대한 브라우저 SAML 연결을 구성할 준비가 되었습니다.

Athena에 대한 브라우저 SAML ODBC 연결 구성

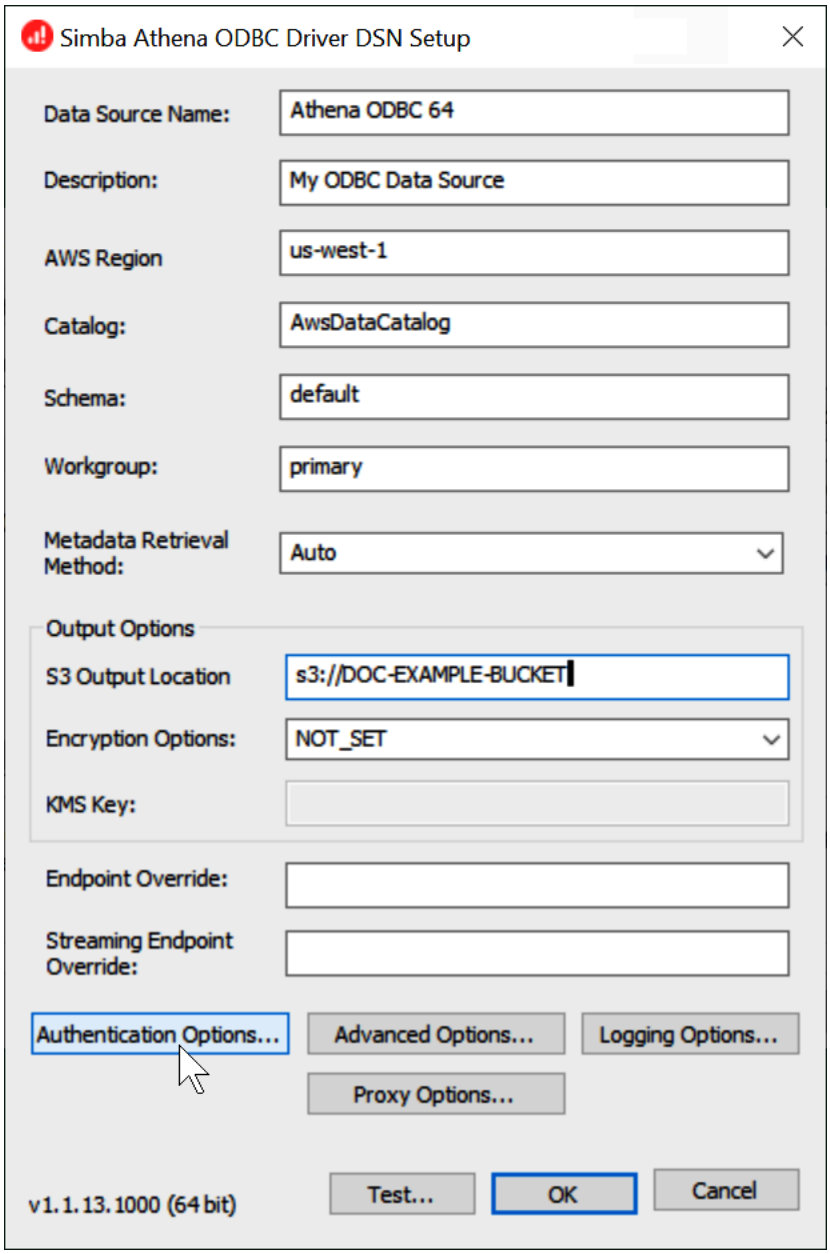
1. Windows에서 ODBC 데이터 원본(ODBC Data Sources) 프로그램을 시작합니다.
2. ODBC 데이터 원본 관리자(ODBC Data Source Administrator) 프로그램에서 추가(Add)를 선택합니다.



3. Simba Athena ODBC 드라이버(Simba Athena ODBC Driver)를 선택하고 완료(Finish)를 선택합니다.



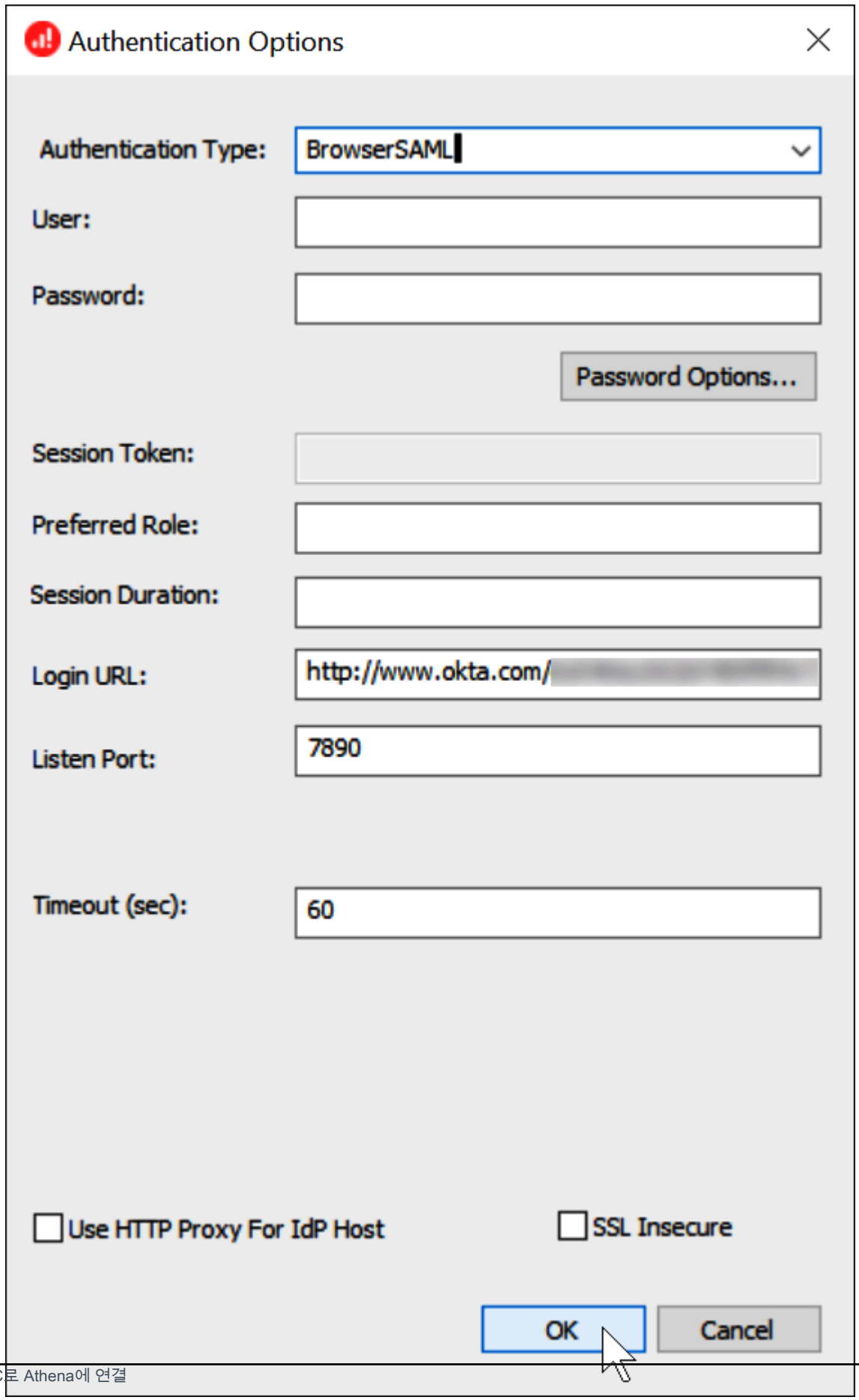
4. Simba Athena ODBC 드라이버 DSN 설치(Simba Athena ODBC Driver DSN Setup) 대화 상자에서 설명된 값을 입력합니다.



- 데이터 원본 이름(Data Source Name)에 데이터 원본 이름을 입력합니다(예: Athena ODBC 64).
- 설명(Description)에 데이터 원본에 대한 설명을 입력합니다.
- AWS 리전에 사용 중인 AWS 리전을 입력합니다(예: **`us-west-1`**).
- S3 출력 위치(S3 Output Location)에 출력을 저장할 Amazon S3 경로를 입력합니다.

5. 인증 옵션(Authentication Options)을 선택합니다.

6. 인증 옵션(Authentication Options) 대화 상자에서 다음 값을 선택하거나 입력합니다.


Authentication Options
Authentication Type:
BrowserSAML
User:
Password:
Password Options...
Session Token:
Preferred Role:
Session Duration:
Login URL:
http://www.okta.com/
Listen Port:
7890
Timeout (sec):
60
Use HTTP Proxy For IdP Host
SSL Insecure
OK
Cancel

- 인증 유형(Authentication Type)에서 BrowserSAML을 선택합니다.
- 로그인 URL(Login URL)에 Okta 대시보드에서 얻은 자격 증명 공급자 통합 인증 URL(Identity Provider Single Sign-On URL)을 입력합니다.
- 수신 대기 포트(Listen Port)에 7890을 입력합니다.
- 제한 시간(초)(Timeout (sec))에 연결 제한 시간 값을 초 단위로 입력합니다.

7. 확인(OK)을 선택하여 인증 옵션(Authentication Options)을 닫습니다.
8. 테스트(Test)를 선택하여 연결을 테스트하거나 확인(OK)을 선택하여 마칩니다.

Amazon Athena Power BI 커넥터 사용

Windows 운영 체제에서 Amazon Athena용 Microsoft Power BI connector를 사용하여 Microsoft Power BI Desktop에서 Amazon Athena의 데이터를 분석할 수 있습니다. Power BI에 대한 자세한 내용은 Microsoft power BI를 참조하세요. Power BI 서비스에 콘텐츠를 게시한 뒤 2021년 7월 이후 릴리스의 Power BI 게이트웨이를 사용하여 온디맨드 또는 예약된 새로고침을 통해 콘텐츠를 최신 상태로 유지할 수 있습니다.

필수 조건

시작하기 전에 자신의 환경이 다음 요구 사항을 충족하는지 확인하세요. Amazon Athena ODBC 드라이버가 필요합니다.

- AWS 계정
- Athena 사용 권한
- Amazon Athena ODBC 드라이버
- Power BI 데스크톱

지원되는 기능

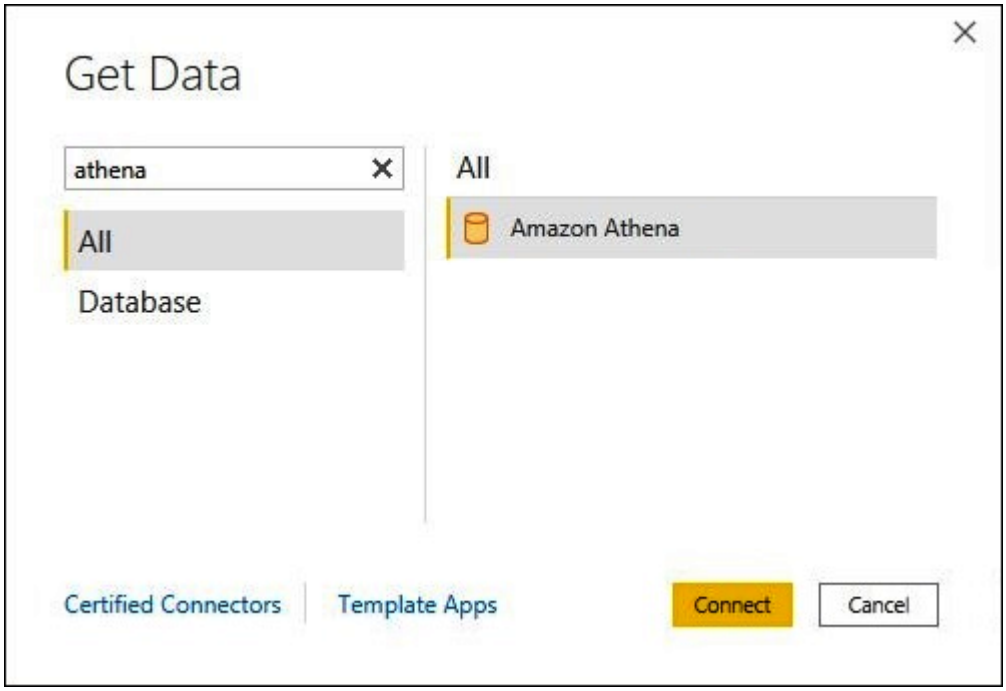
- 가져오기 - 쿼리를 위해 선택한 테이블 및 열을 Power BI Desktop으로 가져옵니다.
- DirectQuery - Power BI Desktop으로 데이터를 가져오거나 복사하지 않습니다. Power BI Desktop은 기본 데이터 원본을 직접 쿼리합니다.
- Power BI 게이트웨이 - Microsoft Power BI 서비스와 Athena 사이의 다리처럼 작동하는 AWS 계정의 온프레미스 데이터 게이트웨이입니다. 이 게이트웨이는 Microsoft Power BI 서비스에서 데이터를 확인하는 데 필요합니다.

Amazon Athena에 연결

Power BI Desktop을 Amazon Athena 데이터에 연결하려면 다음 단계를 수행합니다.

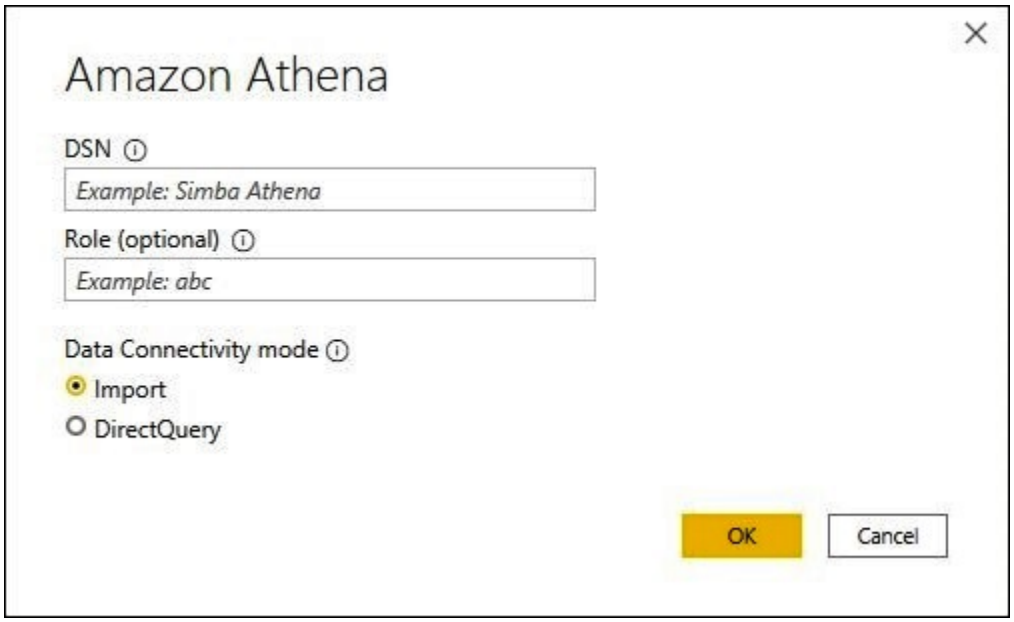
Power BI 데스크톱에서 Athena 데이터에 연결하려면

1. Power BI Desktop을 실행합니다.
2. 다음 중 하나를 수행하세요.
   - 파일, 데이터 가져오기 선택
   - 홈 리본 메뉴에서 데이터 가져오기를 선택합니다.
3. 검색 상자에 Athena를 입력합니다.
4. Amazon Athena를 선택한 다음 연결을 선택합니다.

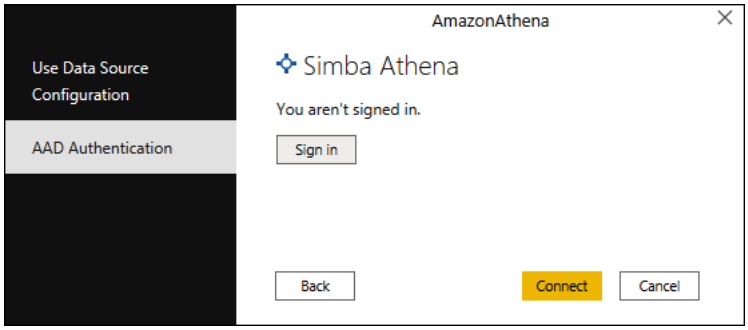


5. Amazon Athena 연결 페이지에서 다음 정보를 입력합니다.
   - 사용할 ODBC DSN의 이름을 DSN에 입력합니다. DSN 구성에 대한 지침은 ODBC 드라이버 설명서를 참조하세요.
   - 데이터 연결 모드의 경우 다음 일반 지침에 따라 사용 사례에 적합한 모드를 선택합니다.
     - 데이터 집합이 비교적 작은 경우 가져오기를 선택합니다. 가져오기 모드를 사용하면 Power BI는 Athena와 함께 작동하여 시각화에 사용할 전체 데이터 집합의 콘텐츠를 가져옵니다.
     - 데이터 집합이 비교적 큰 경우 DirectQuery를 선택합니다. DirectQuery 모드에서는 데이터가 워크스테이션에 다운로드되지 않습니다. 시각화를 만들거나 이와 상호작용하는 동안

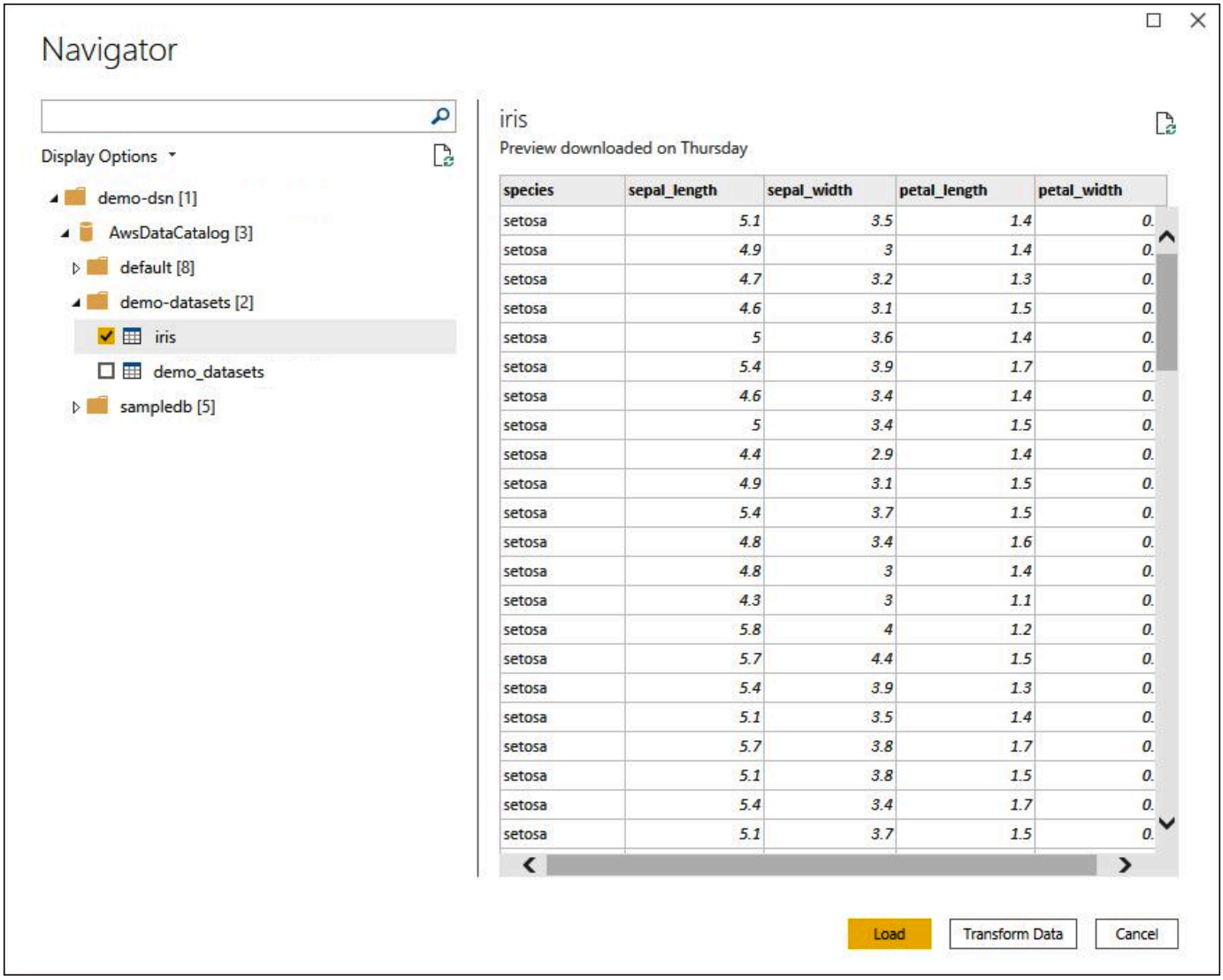
Microsoft Power BI는 Athena와 함께 작동하여 항시 현재 데이터를 볼 수 있도록 기본 데이터 원본을 동적으로 쿼리합니다. DirectQuery에 대한 자세한 내용은 Microsoft 설명서의 Power BI Desktop에서 DirectQuery 사용을 참조하세요.



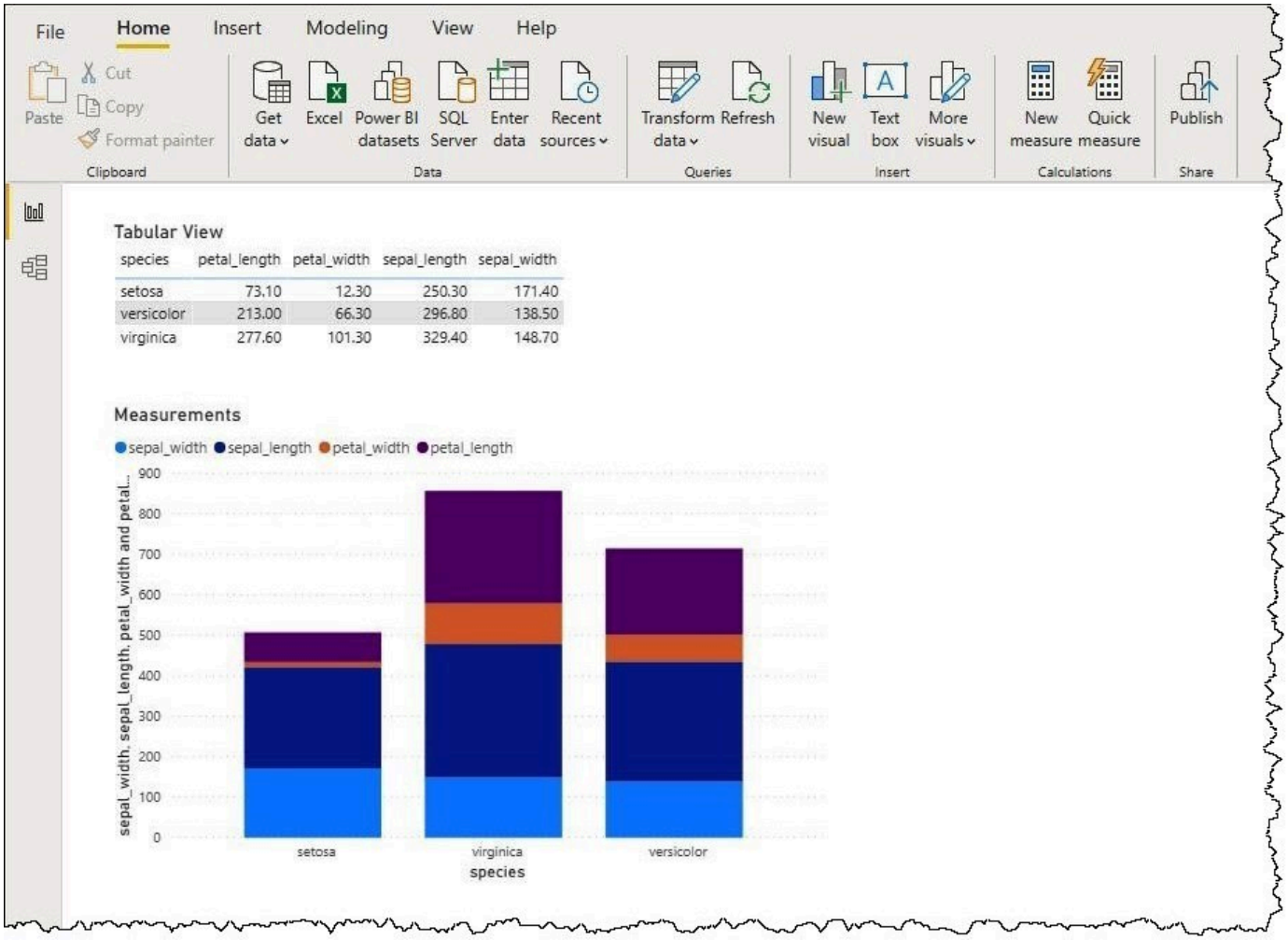
6. 확인을 선택합니다.
7. 데이터 원본 인증을 구성하라는 메시지가 나타나면 데이터 원본 구성 사용 또는 AAD 인증을 선택한 다음 연결을 선택합니다.



데이터 카탈로그, 데이터베이스 및 테이블이 네비게이터 대화 상자에 나타납니다.



8. 표시 옵션 창에서 사용하려는 데이터 집합의 확인란을 선택합니다.
9. 데이터 집합을 가져오기 전에 변환하려면 대화 상자 아래쪽으로 이동하여 데이터 변환을 선택합니다. 이렇게 하면 파워 쿼리 편집기가 열려 사용하려는 데이터 집합을 필터링하고 구체화할 수 있습니다.
10. 로드(Load)를 선택합니다. 로드가 완료되면 다음 이미지와 같은 시각화를 만들 수 있습니다. 가져오기 모드로 DirectQuery를 선택한 경우 Power BI는 요청된 시각화에 대해 Athena에 쿼리를 보냅니다.



온프레미스 게이트웨이 설정

다른 사용자가 웹, 모바일 및 임베디드 앱을 통해 상호작용할 수 있도록 Power BI 서비스에 대시보드 및 데이터 집합을 게시할 수 있습니다. Microsoft Power BI 서비스에서 데이터를 보려면 Microsoft Power BI 온프레미스 데이터 게이트웨이를 AWS 계정에 설치합니다. 게이트웨이는 Microsoft Power BI 서비스와 Athena 사이의 다리처럼 작동합니다.

온프레미스 데이터 게이트웨이를 다운로드, 설치 및 테스트하려면

1. Microsoft Power BI 게이트웨이 다운로드 페이지를 방문하여 개인 모드 또는 표준 모드를 선택합니다. 개인 모드는 Athena 커넥터를 로컬로 테스트하는 데 유용합니다. 표준 모드는 다중 사용자 프로덕션 환경에 적합합니다.
2. 온프레미스 게이트웨이(개인 모드 또는 표준 모드)를 설치하려면 Microsoft 문서에서 온프레미스 데이터 게이트웨이 설치를 참조하세요.

3. 게이트웨이를 테스트하려면 Microsoft 문서에서 온프레미스 데이터 게이트웨이와 함께 사용자 지정 데이터 커넥터 사용을 참조하세요.

온프레미스 데이터 게이트웨이에 대한 자세한 내용은 다음 Microsoft 리소스를 참조하세요.

- 온프레미스 데이터 게이트웨이란 무엇입니까?
- Power BI용 데이터 게이트웨이를 배포하기 위한 지침

Athena에서 사용할 Power BI 게이트웨이를 구성하는 예제는 AWS 빅 데이터 블로그 문서 Creating dashboards quickly on Microsoft power BI using amazon Athena를 참조하세요.

데이터베이스 및 테이블 생성

Amazon Athena는 Amazon Simple Storage Service에 데이터가 존재하는 외부 테이블을 정의하고 쿼리할 수 있도록 데이터 정의 언어(DDL) 문과 ANSI SQL 함수 및 연산자의 하위 집합을 지원합니다.

Athena에서 데이터베이스와 테이블을 생성할 때는 테이블의 데이터가 실시간 쿼리가 가능한 상태가 되도록 데이터의 스키마와 위치를 설명합니다.

쿼리 성능을 향상시키고 비용을 줄이려면 데이터를 분할하고 Amazon S3 스토리지에 Apache parquet나 ORC 같은 오픈 소스 열 기반 형식을 사용하는 것이 좋습니다.

주제

- Athena에서 데이터베이스 생성
- Athena에서 테이블 생성
- 테이블, 데이터베이스 및 열의 이름
- 예약어
- Amazon S3에서 테이블 위치
- 열 기반 스토리지 형식
- 열 기반 형식으로 변환
- Athena에서 데이터 분할
- Amazon Athena를 사용한 파티션 프로젝션

Athena에서 데이터베이스 생성

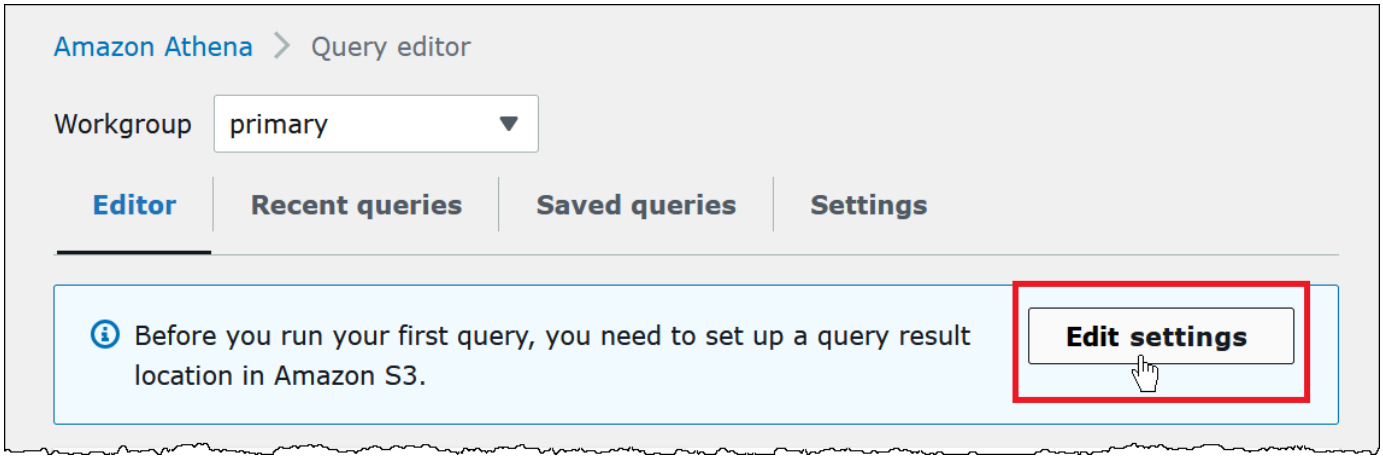
Athena의 데이터베이스는 사용자가 작성한 테이블에 대한 논리적 그룹입니다.

필수 조건

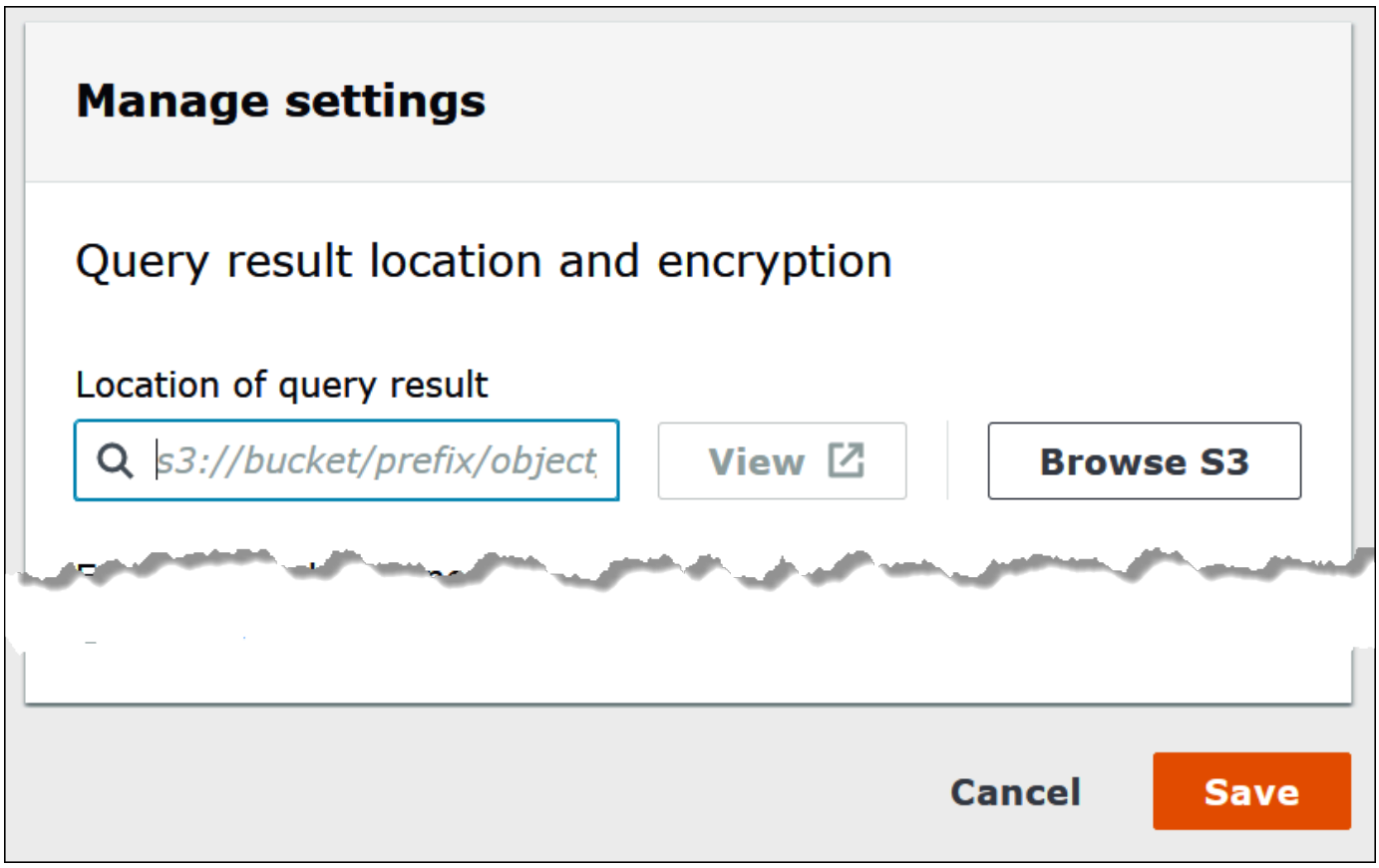
Amazon S3에 쿼리 출력 위치를 아직 설정하지 않은 경우 다음 사전 필수 단계를 수행하여 설정합니다.

쿼리 출력 위치를 생성하려면

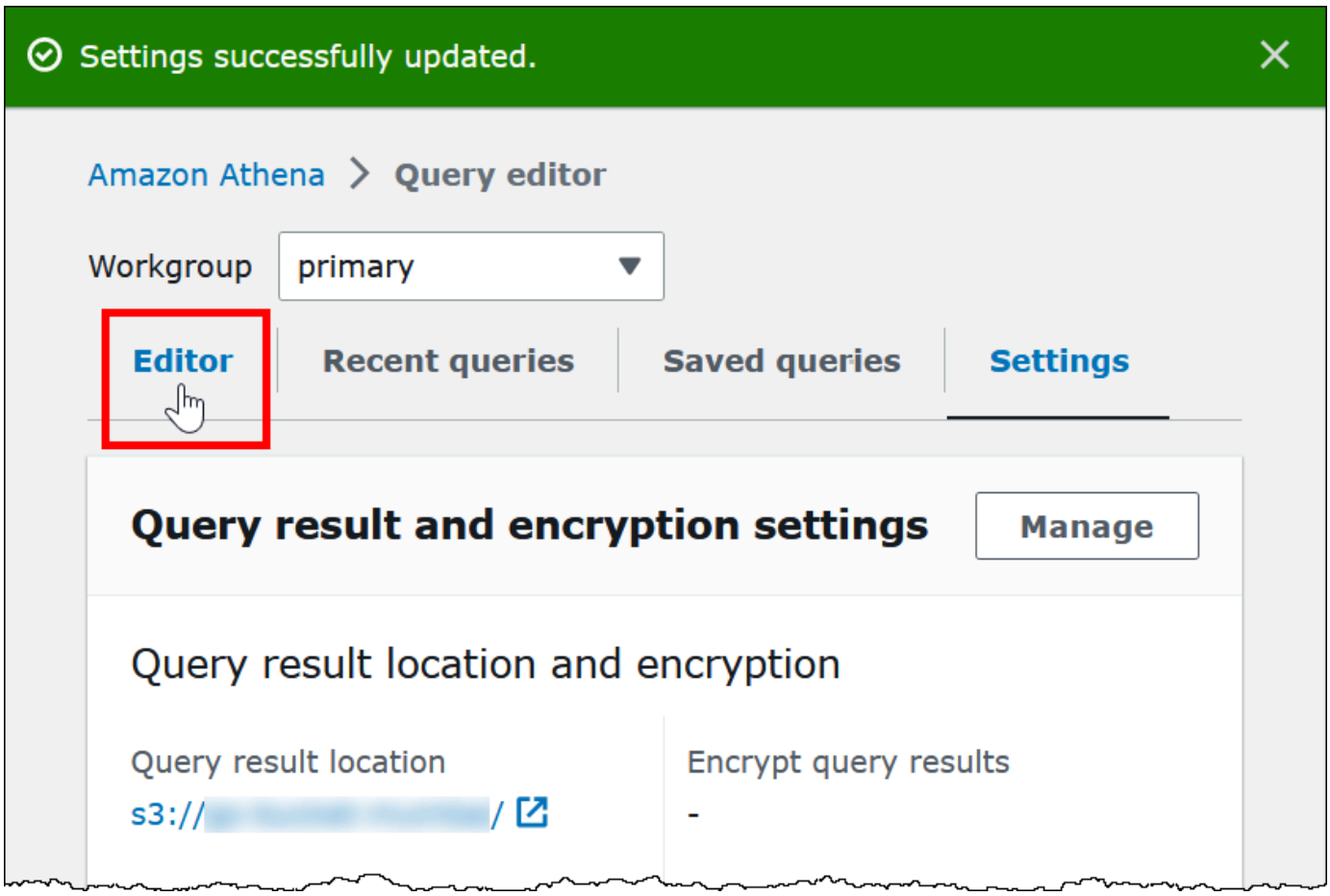
1. Athena에 사용하는 것과 같은 AWS 리전 및 계정을 사용하여(예: Amazon S3 콘솔 사용) Amazon S3에 버킷을 생성하는 단계에 따라 Athena 쿼리 결과를 보관합니다. 이 버킷을 쿼리 출력 위치로 구성합니다.
2. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
3. 이 AWS 리전에서 Athena 콘솔을 처음 방문하는 경우 쿼리 편집기 탐색을 선택하여 쿼리 편집기를 엽니다. 그렇지 않으면 Athena가 쿼리 편집기에서 열립니다.
4. Amazon S3에서 쿼리 결과 위치를 설정하려면 Edit Settings(설정 편집)를 선택합니다.



5. 설정 관리(Manage settings)에서 다음 중 하나를 수행합니다.
   - 쿼리 결과 위치(Location of query result) 상자에서 Amazon S3에서 쿼리 결과를 위해 생성한 버킷의 경로를 입력합니다. 경로 앞에 `s3://`를 붙입니다.
   - S3 검색(Browse S3) 아이콘을 선택하고 현재 리전에서 생성한 Amazon S3 버킷을 선택한 다음 선택(Choose)을 선택합니다.



6. Save(저장)를 선택합니다.
7. 편집기(Editor)를 선택하여 쿼리 편집기로 전환합니다.



데이터베이스 생성

쿼리 결과 위치를 설정한 후 Athena 콘솔 쿼리 편집기에서 데이터베이스를 생성하는 작업은 간단합니다.

Athena 쿼리 편집기를 사용하여 데이터베이스 생성

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 쿼리 편집기의 편집기 탭에서 Hive 데이터 정의 언어(DDL) 명령 `CREATE DATABASE myDataBase`를 입력합니다. *myDatabase*를 사용하려는 이름으로 바꿉니다. 데이터베이스 이름에 대한 제한 사항은 테이블, 데이터베이스 및 열의 이름 섹션을 참조하세요.
3. 실행(Run)을 선택하거나 **Ctrl+ENTER**를 누릅니다.
4. 데이터베이스를 현재 데이터베이스로 만들려면 쿼리 편집기 왼쪽의 데이터베이스(Database) 메뉴에서 데이터베이스를 선택합니다.

Athena 데이터베이스 권한 제어에 대한 자세한 내용은 AWS Glue Data Catalog의 데이터베이스와 테이블에 대한 세분화된 액세스 단원을 참조하세요.

Athena에서 테이블 생성

JDBC 또는 ODBC 드라이버를 사용하거나 Athena 테이블 추가 양식을 사용하여 Athena 콘솔에서 DDL 문을 실행할 수 있습니다.

Athena에서 새 테이블 스키마를 만들면 Athena가 데이터 카탈로그에 스키마를 저장하고 쿼리 실행 시 사용합니다.

Athena는 스키마-온-리드(schema-on-read)라는 접근 방식을 사용합니다. 즉, 쿼리를 실행할 때 스키마가 사용자의 데이터에 프로젝션됩니다. 따라서 데이터를 로드하거나 변환할 필요가 없습니다.

Athena는 Amazon S3의 데이터를 수정하지 않습니다.

Athena는 Apache Hive를 사용하여 테이블을 정의하고 데이터베이스를 생성하며, 이는 테이블의 논리적 네임스페이스입니다.

Athena에서 데이터베이스와 테이블을 생성할 때, 읽기 시간 쿼리를 위해 Amazon S3에서 테이블 데이터가 있는 스키마와 위치를 간단히 설명합니다. 따라서 데이터베이스와 테이블이 데이터베이스와 테이블의 스키마 정의와 함께 저장되지 않으므로 기존 관계형 데이터베이스 시스템에서와 의미가 약간 다릅니다.

쿼리할 때 표준 SQL을 사용하여 테이블을 쿼리하면 그 시간에 데이터가 읽힙니다. Apache Hive 설명서를 사용하면 데이터베이스와 테이블을 만드는 방법을 배울 수 있지만, 다음과 같이 Athena에만 해당하는 지침이 있습니다.

최대 쿼리 문자열 길이는 256KB입니다.

Hive는 SerDe(serializer-deserializer) 라이브러리를 사용하여 다양한 데이터 형식을 지원합니다. 정규 표현식을 사용하여 복잡한 스키마를 정의할 수도 있습니다. 지원되는 SerDe 라이브러리 목록은 지원되는 SerDes 및 데이터 형식 단원을 참조하세요.

고려 사항 및 제한

Athena의 테이블에 대한 몇 가지 중요한 제한 사항 및 고려 사항은 다음과 같습니다.

Athena의 테이블 및 Amazon S3의 데이터에 대한 요구 사항

테이블을 만들 때 `LOCATION` 절을 사용하여 기본 데이터의 Amazon S3 버킷 위치를 지정합니다. 다음을 고려하세요.

- Athena는 버전이 지정된 Amazon S3 버킷에 있는 최신 버전의 데이터만 쿼리할 수 있으며, 이전 버전의 데이터는 쿼리할 수 없습니다.
- 이를 위해서는 Amazon S3 위치에서 데이터를 처리할 수 있는 적정한 권한이 있어야 합니다. 자세한 내용은 Amazon S3에 액세스 단원을 참조하세요.
- Athena는 LOCATION 절에 지정된 동일한 버킷의 여러 스토리지 클래스에 저장된 객체에 대한 쿼리를 지원합니다. 예를 들어 Amazon S3의 서로 다른 스토리지 클래스(Standard, Standard-IA, Intelligent-Tiering)에 저장된 객체의 데이터를 쿼리할 수 있습니다.
- Athena는 요청자 지불 버킷을 지원합니다. Athena에서 쿼리하려고 하는 소스 데이터가 포함된 버킷에 대해 요청자 지불을 활성화하는 방법에 대한 자세한 내용은 작업 그룹 만들기 단원을 참조하세요.
- Athena는 S3 Glacier Flexible Retrieval 또는 S3 Glacier Deep Archive 스토리지 클래스의 데이터에 대한 쿼리를 지원하지 않습니다. S3 Glacier Flexible Retrieval 또는 S3 Glacier Deep Archive 스토리지 클래스의 객체는 무시됩니다. 또는 Athena가 쿼리할 수 있는 Amazon S3 Glacier Instant Retrieval 스토리지 클래스를 사용할 수 있습니다. 자세한 내용은 Amazon S3 Glacier Instant Retrieval 스토리지 클래스를 참조하세요.

  스토리지 클래스에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서에서 스토리지 클래스, Amazon S3에서 객체의 스토리지 클래스 변경, GLACIER 스토리지 클래스로 전환(객체 아카이브) 및 요청자 지불 버킷을 참조하세요.
- 객체가 많은 Amazon S3 버킷에 대해 쿼리를 실행하는데 데이터가 분할되지 않은 경우, 그러한 쿼리는 Amazon S3의 Get 요청 빈도 제한에 영향을 주고 Amazon S3 예외를 발생시킬 수 있습니다. 오류를 방지하려면 데이터를 분할하세요. 또한 Amazon S3 요청 빈도를 조정합니다. 자세한 내용은 요청 빈도 및 성능 고려 사항을 참조하세요.
- TableType 속성을 지정하지 않고 AWS Glue CreateTable API 작업 또는 AWS CloudFormation AWS::Glue::Table 템플릿을 사용하여 Athena에서 사용할 테이블을 만든 다음 SHOW CREATE TABLE 또는 MSCK REPAIR TABLE 같은 DDL 쿼리를 실행하면, 실패: NullPointerException Name이 null임(FAILED: NullPointerException Name is null)이라는 오류 메시지가 표시될 수 있습니다.

  이 오류를 해결하려면 AWS Glue CreateTable API 호출 또는 AWS CloudFormation 템플릿의 일부로 TableInput TableType 속성의 값을 지정하세요. TableType의 가능한 값은 EXTERNAL_TABLE 또는 VIRTUAL_VIEW입니다.

  이 요구 사항은 AWS Glue CreateTable API 작업 또는 AWS::Glue::Table 템플릿을 사용하여 테이블을 만들 때만 적용됩니다. DDL 문이나 AWS Glue 크롤러를 사용하여 Athena용 테이블을 생성할 경우 TableType 속성이 자동으로 정의됩니다.

지원되는 함수

Athena 쿼리에서 지원되는 함수는 Trino 및 Presto의 함수에 해당합니다. 개별 함수에 대한 자세한 내용을 알아보려면 Trino 또는 Presto 설명서의 함수 및 연산자 섹션을 참조하세요.

트랜잭션 데이터 변환은 지원되지 않음

Athena는 테이블 데이터에 대한 트랜잭션 기반 작업(예: Hive 또는 Presto에 있는 작업)을 지원하지 않습니다. 지원되지 않는 키워드 전체 목록은 지원되지 않는 DDL을 참조하세요.

테이블 상태를 변경하는 작업은 ACID

테이블을 생성하고 업데이트하거나 삭제할 때 이러한 작업은 ACID를 준수합니다. 예를 들어, 여러 사용자 또는 클라이언트가 동시에 테이블을 생성하거나 기존 테이블을 변경하려고 할 경우 그 중 하나만 성공합니다.

테이블은 EXTERNAL

Iceberg 테이블을 생성할 때를 제외하고 항상 `EXTERNAL` 키워드를 사용하세요. Iceberg가 아닌 테이블에 대해 `EXTERNAL` 키워드 없이 `CREATE TABLE`을 사용하는 경우 Athena에서 오류가 발생합니다. Athena에 테이블을 놓으면 테이블 메타데이터만 제거됩니다. 데이터는 Amazon S3에 유지됩니다.

AWS Glue 또는 Athena 콘솔을 사용하여 테이블 생성

Athena에서 AWS Glue 또는 테이블 추가 양식을 사용하거나 Athena 쿼리 편집기에서 DDL 문을 실행하여 테이블을 생성할 수 있습니다.

AWS Glue 크롤러를 사용하여 테이블 생성

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 쿼리 편집기에서 테이블 및 뷰(Tables and views) 옆에 있는 생성(Create)을 선택한 다음 AWS Glue 크롤러(Glue crawler)를 선택합니다.
3. AWS Glue 콘솔의 크롤러 추가(Add crawler) 페이지에 있는 단계에 따라 크롤러를 추가합니다.

   자세한 내용은 AWS Glue 크롤러 사용 단원을 참조하십시오.

Athena 테이블 생성 양식을 사용하여 테이블 생성

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.

2. 쿼리 편집기에서 테이블 및 뷰(Tables and views) 옆에 있는 생성(Create)을 선택한 다음 S3 버킷 데이터(S3 bucket data)를 선택합니다.
3. S3 버킷 데이터에서 테이블 생성(Create Table From S3 bucket data) 양식에 테이블 생성을 위한 정보를 입력한 다음 테이블 생성(Create table)을 선택합니다. 양식의 필드에 대한 자세한 내용은 양식을 사용하여 테이블 추가 섹션을 참조하세요.

Hive DDL을 사용하여 테이블을 생성하려면

1. Database(데이터베이스) 메뉴에서 테이블을 생성할 데이터베이스를 선택합니다. CREATE TABLE 문에서 데이터베이스를 지정하지 않으면 쿼리 편집기에서 현재 선택되어 있는 데이터베이스에 테이블이 생성됩니다.
2. 쿼리 편집기에 다음과 같은 문을 입력한 다음 실행(Run)을 선택하거나 **Ctrl+ENTER**를 누릅니다.

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
    `Date` Date,
    Time STRING,
    Location STRING,
    Bytes INT,
    RequestIP STRING,
    Method STRING,
    Host STRING,
    Uri STRING,
    Status INT,
    Referrer STRING,
    OS String,
    Browser String,
    BrowserVersion String
) ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
"input.regex" = "^(?!#)([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s
+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+[^\(]+[\(]([^\;]+).*\%20([^
\/]+)[\/](.*)$"
) LOCATION 's3://athena-examples-MyRegion/cloudfront/plaintext/';
```

테이블 정보 표시

Athena에서 테이블을 생성하면 해당 이름이 왼쪽의 테이블(Tables) 목록에 표시됩니다. 테이블에 대한 정보를 표시하고 관리하려면 Athena 콘솔에서 테이블 이름 옆에 있는 세로 점 3개를 선택합니다.

- 테이블 미리 보기 - Athena 쿼리 편집기에서 `SELECT * FROM "database_name"."table_name" LIMIT 10` 문을 실행하여 모든 열의 첫 10개 행을 표시합니다.
- 테이블 DDL 생성 - Athena 쿼리 편집기에서 `SHOW CREATE TABLE table_name` 문을 실행하여 테이블을 다시 생성하는 데 사용할 수 있는 DDL 문을 생성합니다.
- 파티션 로드 - Athena 쿼리 편집기에서 `MSCK REPAIR TABLE table_name` 문이 실행됩니다. 이 옵션은 테이블에 파티션이 있는 경우에만 사용할 수 있습니다.
- 편집기에 삽입 - 현재 편집 위치의 쿼리 편집기에 테이블 이름을 삽입합니다.
- 테이블 삭제 - 테이블을 삭제할지 묻는 확인 대화 상자를 표시합니다. 동의하면 Athena 쿼리 편집기에서 `DROP TABLE table_name` 문이 실행됩니다.
- 테이블 속성 - 테이블 이름, 데이터베이스 이름, 생성 시간 및 테이블에 암호화된 데이터가 있는지 여부를 표시합니다.

테이블, 데이터베이스 및 열의 이름

Athena에서 데이터베이스 객체 명명에 대해 다음 팁을 사용하세요.

데이터베이스, 테이블 및 열 이름 요구 사항

- AWS Glue의 데이터베이스 이름, 테이블 이름 및 열 이름에 사용할 수 있는 문자는 UTF-8 문자열이어야 합니다. 문자열은 1바이트 미만이거나 255바이트를 초과해서는 안 됩니다. 이 한도를 초과하면 Value at 'name' failed to satisfy constraint: Member must have length less than or equal to 255와 같은 오류가 발생합니다. 사용할 수 있는 문자에는 공백이 포함되고 다음과 같은 한 줄 문자열 패턴으로 정의됩니다.

```
[\u0020-\uD7FF\uE000-\uFFFD\uD800\uDC00-\uDBFF\uDFFF\t]*
```

- 현재 AWS Glue 정규식 패턴에서는 이름 앞에 선행 공백을 추가할 수 있습니다. 이러한 선행 공백은 탐지하기 어렵고 생성 후 사용성 문제를 일으킬 수 있으므로 선행 공백이 있는 객체 이름을 만들지 마세요.
- AWS::Glue::Database AWS CloudFormation 템플릿을 사용하여 AWS Glue 데이터베이스를 생성하고 데이터베이스 이름을 지정하지 않으면 AWS Glue는 Athena와 호환되지 않는 *resource_name-random_string* 형식으로 데이터베이스 이름을 자동으로 생성합니다.
- AWS Glue 카탈로그 관리자를 사용하여 열 이름을 바꿀 수 있지만 테이블 이름이나 데이터베이스 이름은 바꿀 수 없습니다. 이 제한을 해결하려면 이전 데이터베이스의 정의를 사용하여 새 이름으로 데이터베이스를 생성해야 합니다. 그런 다음 이전 데이터베이스의 테이블 정의를 사용하여 새 데이터

베이스에서 테이블을 다시 생성합니다. 이렇게 하려면 AWS CLI 또는 AWS Glue SDK를 사용하면 됩니다. 단계는 AWS CLI를 사용하여 AWS Glue 데이터베이스 및 해당 테이블 다시 생성를 참조하세요.

Athena의 테이블 이름과 테이블 열 이름에는 소문자를 사용합니다.

Athena는 DDL 및 DML 쿼리에서 대소문자 혼용을 허용하지만 쿼리를 실행할 때 이름을 소문자로 사용합니다. 이러한 이유로 테이블 또는 열 이름에 대소문자를 혼용하지 말고, Athena에서 대소문자를 통해서만 이러한 이름을 구별하지 마세요. 예를 들어 DDL 문을 사용하여 `Castle`이라는 열을 만드는 경우 생성된 열은 `castle`과 같이 소문자로 표시됩니다. 그런 다음 DML 쿼리에서 열 이름을 `Castle` 또는 `CASTLE`로 지정할 경우 Athena는 쿼리를 실행할 이름을 소문자로 표시하지만 쿼리에서 선택한 대소문자를 사용하여 열 머리글을 표시합니다.

데이터베이스, 테이블 및 열 이름은 255자보다 작거나 같아야 합니다.

밑줄로 시작하는 이름

테이블을 만들 때 밑줄로 시작하는 테이블, 뷰 또는 열 이름은 백틱(`)으로 묶습니다. 다음 예를 참조하세요.

```
CREATE EXTERNAL TABLE IF NOT EXISTS `_myunderscoretable`(
  `_id` string, `_index` string)
LOCATION 's3://DOC-EXAMPLE-BUCKET/'
```

숫자로 시작하는 테이블, 뷰 또는 열 이름

`SELECT`, `CTAS` 또는 `VIEW` 쿼리를 실행하는 경우 숫자로 시작하는 테이블, 뷰 또는 열 이름과 같은 식별자를 인용 부호로 묶습니다. 다음 예를 참조하세요.

```
CREATE OR REPLACE VIEW "123view" AS
SELECT "123columnone", "123columntwo"
FROM "234table"
```

열 이름 및 복합 유형

복합 유형의 경우 영숫자, 밑줄(_) 및 기간(.)만 열 이름에 허용됩니다. 제한된 문자가 있는 키에 대한 테이블 및 매핑을 만들 때 사용자 지정 DDL 문을 사용할 수 있습니다. 자세한 내용은 AWS 빅 데이터 블로그의 Create tables in Amazon Athena from nested JSON and mappings using JSONSerDe를 참조하세요.

예약어

Athena의 특정 예약어는 이스케이프를 해야 합니다. DDL 문에서 예약어를 이스케이프하려면 해당 문을 악센트 부호(`)로 묶습니다. 뷰에 대한 SQL SELECT 문과 쿼리에서 예약어를 이스케이프하려면 예약어를 큰 따옴표(")로 묶습니다.

자세한 내용은 예약어 단원을 참조하십시오.

추가 리소스

전체 데이터베이스 및 테이블 생성 구문은 다음 페이지를 참조하세요.

- CREATE DATABASE
- CREATE TABLE

AWS Glue의 데이터베이스 및 테이블에 대한 자세한 내용은 AWS Glue 개발자 안내서의 데이터베이스 및 테이블을 참조하세요.

예약어

Athena에서 예약어가 포함된 쿼리를 실행할 때는 쿼리를 특수 문자로 묶어서 이스케이프해야 합니다. 어떤 키워드가 Athena에 예약되어 있는지 확인하려면 이 주제에 있는 목록을 사용하세요.

DDL 문에서 예약어를 이스케이프하려면 해당 문을 악센트 부호(`)로 묶습니다. 뷰에 대한 SQL SELECT 문과 쿼리에서 예약어를 이스케이프하려면 예약어를 큰 따옴표(")로 묶습니다.

- DDL 문의 예약어 목록
- SQL SELECT 문의 예약어 목록
- 예약어가 포함된 쿼리의 예

DDL 문의 예약어 목록

Athena는 DDL 문에서 다음과 같은 예약어 목록을 사용합니다. 예약어를 이스케이프하지 않고 사용할 경우, Athena에 오류가 표시됩니다. 예약어를 이스케이프하려면 악센트 부호(`)로 묶습니다.

DDL 예약어를 악센트 부호(`)로 묶지 않고는 DDL 문에 식별자 이름으로 사용할 수 없습니다.

```
ALL, ALTER, AND, ARRAY, AS, AUTHORIZATION, BETWEEN, BIGINT,
BINARY, BOOLEAN, BOTH, BY, CASE, CASHE, CAST, CHAR, COLUMN,
```

```
CONF, CONSTRAINT, COMMIT, CREATE, CROSS, CUBE, CURRENT,
CURRENT_DATE, CURRENT_TIMESTAMP, CURSOR, DATABASE, DATE,
DAYOFWEEK, DECIMAL, DELETE, DESCRIBE, DISTINCT, DOUBLE, DROP,
ELSE, END, EXCHANGE, EXISTS, EXTENDED, EXTERNAL, EXTRACT,
FALSE, FETCH, FLOAT, FLOOR, FOLLOWING, FOR, FOREIGN, FROM,
FULL, FUNCTION, GRANT, GROUP, GROUPING, HAVING, IF, IMPORT,
IN, INNER, INSERT, INT, INTEGER, INTERSECT, INTERVAL, INTO,
IS, JOIN, LATERAL, LEFT, LESS, LIKE, LOCAL, MACRO, MAP, MORE,
NONE, NOT, NULL, NUMERIC, OF, ON, ONLY, OR, ORDER, OUT,
OUTER, OVER, PARTIALSCAN, PARTITION, PERCENT, PRECEDING,
PRECISION, PRESERVE, PRIMARY, PROCEDURE, RANGE, READS,
REDUCE, REGEXP, REFERENCES, REVOKE, RIGHT, RLIKE, ROLLBACK,
ROLLUP, ROW, ROWS, SELECT, SET, SMALLINT, START,TABLE,
TABLESAMPLE, THEN, TIME, TIMESTAMP, TO, TRANSFORM, TRIGGER,
TRUE, TRUNCATE, UNBOUNDED,UNION, UNIQUEJOIN, UPDATE, USER,
USING, UTC_TIMESTAMP, VALUES, VARCHAR, VIEWS, WHEN, WHERE,
WINDOW, WITH
```

SQL SELECT 문의 예약어 목록

Athena는 뷰에 대한 SQL SELECT 문 및 쿼리에서 다음과 같은 예약어 목록을 사용합니다.

이 키워드를 식별자로 사용할 경우 쿼리 문에서 큰따옴표(")를 사용하여 묶어야 합니다.

```
ALTER, AND, AS, BETWEEN, BY, CASE, CAST, CONSTRAINT, CREATE,
CROSS, CUBE, CURRENT_CATALOG, CURRENT_DATE, CURRENT_PATH,
CURRENT_SCHEMA, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_USER,
DEALLOCATE, DELETE, DESCRIBE, DISTINCT, DROP, ELSE, END, ESCAPE,
EXCEPT, EXECUTE, EXISTS, EXTRACT, FALSE, FIRST, FOR, FROM,
FULL, GROUP, GROUPING, HAVING, IN, INNER, INSERT, INTERSECT,
INTO, IS, JOIN, JSON_ARRAY, JSON_EXISTS, JSON_OBJECT,
JSON_QUERY, JSON_TABLE, JSON_VALUE, LAST, LEFT, LIKE,
LISTAGG, LOCALTIME, LOCALTIMESTAMP, NATURAL, NORMALIZE,
NOT, NULL, OF, ON, OR, ORDER, OUTER, PREPARE, RECURSIVE, RIGHT,
ROLLUP, SELECT, SKIP, TABLE, THEN, TRIM, TRUE, UESCAPE, UNION,
UNNEST, USING, VALUES, WHEN, WHERE, WITH
```

예약어가 포함된 쿼리의 예

다음 예제의 쿼리는 백틱(`)을 사용하여 테이블 이름과 열 이름 중 하나로 사용되는 DDL 관련 예약어 partition과 date를 이스케이프합니다.

```
CREATE EXTERNAL TABLE `partition` (
```

```
`date` INT,
col2 STRING
)
PARTITIONED BY (year STRING)
STORED AS TEXTFILE
LOCATION 's3://DOC-EXAMPLE-BUCKET/test_examples/';
```

다음 예제 쿼리는 ALTER TABLE ADD PARTITION 및 ALTER TABLE DROP PARTITION 문에 DDL 관련 예약어가 있는 열 이름을 포함합니다. DDL 예약어는 백틱(`)으로 묶여 있습니다.

```
ALTER TABLE test_table
ADD PARTITION (`date` = '2018-05-14')
```

```
ALTER TABLE test_table
DROP PARTITION (`partition` = 'test_partition_value')
```

다음 예제 쿼리는 SELECT 문에 예약어(end)를 식별자로 포함합니다. 키워드는 큰따옴표로 이스케이프되어 있습니다.

```
SELECT *
FROM TestTable
WHERE "end" != nil;
```

다음 예제 쿼리는 SELECT 문에 예약어(first)를 포함합니다. 키워드는 큰따옴표로 이스케이프되어 있습니다.

```
SELECT "itemId"."first"
FROM testTable
LIMIT 10;
```

Amazon S3에서 테이블 위치

Athena에서 CREATE TABLE 쿼리를 실행하면 Athena는 메타데이터를 저장하는 AWS Glue 데이터 카탈로그에 테이블을 등록합니다.

Amazon S3의 데이터에 대한 경로를 지정하려면 다음 예와 같이 LOCATION 속성을 사용합니다.

```
CREATE EXTERNAL TABLE `test_table`(
...
)
```

```
ROW FORMAT ...
STORED AS INPUTFORMAT ...
OUTPUTFORMAT ...
LOCATION s3://DOC-EXAMPLE-BUCKET/folder/
```

- 버킷 이름 지정에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버킷 규제 및 제한을 참조하세요.
- Amazon S3에서의 폴더 사용에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 폴더 사용을 참조하세요.

Amazon S3의 LOCATION은 테이블을 나타내는 모든 파일을 지정합니다.

> **Important**
>
> Athena는 지정한 Amazon S3 폴더에 저장된 모든 데이터를 읽습니다. Athena가 읽지 않도록 할 데이터가 있는 경우 Athena가 읽어야 할 데이터가 있는 Amazon S3 폴더와 동일한 폴더에 해당 데이터를 저장하지 않아야 합니다. 파티셔닝을 활용하는 경우 Athena가 파티션 내의 데이터를 스캔하도록 하려면 WHERE 필터에 파티션을 포함해야 합니다. 자세한 내용은 테이블 위치 및 파티션 단원을 참조하십시오.

LOCATION 문에서 CREATE TABLE을 지정하는 경우 다음 지침을 따르세요.

- 후행 슬래시를 사용하세요.
- Amazon S3 폴더 또는 Amazon S3 액세스 포인트 별칭에 대한 경로를 사용할 수 있습니다. Amazon S3 액세스 포인트 별칭에 대한 자세한 내용은 Amazon S3 사용 설명서의 액세스 포인트에 버킷 스타일 별칭 사용을 참조하세요.

다음 사용:

```
s3://DOC-EXAMPLE-BUCKET/folder/
```

```
s3://DOC-EXAMPLE-BUCKET-metadata-s3alias/folder/
```

데이터에 대해 LOCATION을 지정하는 데 다음 항목을 사용하지 마세요.

- 파일 위치를 지정할 때 파일 이름, 밑줄, 와일드카드 또는 glob 패턴을 사용하지 마세요.

- Amazon S3 버킷 경로에 `s3.amazon.com` 등의 전체 HTTP 표기법을 추가하지 마세요.
- `//`와 같이 경로에서 `S3://DOC-EXAMPLE-BUCKET/folder//folder/` 같은 빈 폴더를 사용하지 마세요. 유효한 Amazon S3 경로일 경우 Athena는 이를 허용하지 않으며 `s3://DOC-EXAMPLE-BUCKET/folder/folder/`로 변경하고 여분의 `/`를 삭제합니다.

  다음 사용 안 함:

```
s3://DOC-EXAMPLE-BUCKET
s3://DOC-EXAMPLE-BUCKET/*
s3://DOC-EXAMPLE-BUCKET/mySpecialFile.dat
s3://DOC-EXAMPLE-BUCKET/prefix/filename.csv
s3://DOC-EXAMPLE-BUCKET.s3.amazon.com
S3://DOC-EXAMPLE-BUCKET/prefix//prefix/
arn:aws:s3:::DOC-EXAMPLE-BUCKET/prefix
s3://arn:aws:s3:<region>:<account_id>:accesspoint/<accesspointname>
https://<accesspointname>-<number>.s3-accesspoint.<region>.amazonaws.com
```

테이블 위치 및 파티션

원본 데이터는 열 집합을 기준으로 파티션이라는 Amazon S3 폴더에 그룹화될 수 있습니다. 예를 들어, 이러한 열은 특정 레코드가 생성된 연도, 월, 일을 나타낼 수 있습니다.

테이블을 생성할 때 테이블을 분할하도록 선택할 수 있습니다. Athena가 분할되지 않은 테이블에 대해 SQL 쿼리를 실행하면 테이블 정의의 `LOCATION` 속성을 기본 경로로 사용하여 나열하고 사용 가능한 모든 파일을 스캔합니다. 그러나 분할된 테이블을 쿼리하기 전에 먼저 파티션 정보로 AWS Glue 데이터 카탈로그를 업데이트해야 합니다. 이 정보는 특정 파티션 내의 파일 스키마와 해당 파티션의 Amazon S3에 있는 파일의 `LOCATION`을 나타냅니다.

- AWS Glue 크롤러가 어떻게 파티션을 추가하는지 알아보려면 AWS Glue 개발자 안내서의 크롤러는 파티션 생성 시기를 어떻게 결정합니까?를 참조하세요.
- 크롤러가 기존 파티션의 데이터에 대한 테이블을 만들도록 구성하는 방법은 크롤러와 여러 데이터 원본 사용 단원을 참조하세요.
- Athena에서 직접 테이블에 파티션을 만들 수도 있습니다. 자세한 내용은 Athena에서 데이터 분할 단원을 참조하십시오.

Athena가 분할된 테이블에 대해 쿼리를 실행하는 경우 먼저 쿼리의 `WHERE` 절에 분할된 열이 사용되는지 확인합니다. 분할된 열이 사용되는 경우 Athena는 지정된 파티션 열과 일치하는 파티션 사양을 반환하도록 AWS Glue 데이터 카탈로그에 요청합니다. 파티션 사양에는 Athena가 데이터를 읽을 때 사

용할 Amazon S3 접두사를 지정하는 LOCATION 속성이 포함되어 있습니다. 이 경우 이 접두사에 저장된 데이터만 스캔됩니다. WHERE 절에서 분할된 열을 사용하지 않는 경우 Athena는 테이블의 파티션에 속한 모든 파일을 검색합니다.

Athena로 파티셔닝을 사용하여 쿼리 성능을 높이고 쿼리 비용을 줄이는 예제는 Amazon Athena를 위한 유용한 성능 튜닝 팁 10가지를 참조하세요.

열 기반 스토리지 형식

Apache Parquet 및 ORC는 빠른 데이터 검색에 최적화되어 있고, AWS 분석 애플리케이션에서 사용되는 열 기반 스토리지 형식입니다.

열 기반 스토리지 형식에는 Athena에서 사용하는 데 적합하게 만드는 다음과 같은 특징이 있습니다.

- 열 기준 압축, 열 데이터 유형에 대해 선택한 압축 알고리즘 사용: Amazon S3에서 스토리지 공간을 절약하고 쿼리 처리 중 디스크 공간 및 I/O를 줄입니다.
- Parquet 및 ORC의 조건자 푸시다운: Athena에서 필요한 블록만 가져오도록 쿼리할 수 있어 쿼리 성능을 높입니다. Athena 쿼리가 데이터에서 특정 열 값을 얻으면 데이터 블록 조건자의 통계(예: 최대 값/최소값)를 사용해 블록을 읽거나 건너뛸지 결정합니다.
- Parquet 및 ORC의 데이터 분할: Athena에서 쿼리 처리 중 데이터 읽기를 여러 리더에게 분할해 병렬 처리를 늘립니다.

기존 원시 데이터를 다른 스토리지 형식에서 Parquet 또는 ORC로 변환하려면 Athena에서 CREATE TABLE AS SELECT (CTAS) 쿼리를 실행하고 데이터 스토리지 형식을 Parquet 또는 ORC로 지정하거나 AWS Glue 크롤러를 사용합니다.

Parquet 및 ORC 중에서 선택

Optimized Row Columnar(ORC) 및 Parquet 중에서의 선택은 특정 사용 요구 사항에 따라 다릅니다.

Apache Parquet은 효율적인 데이터 압축 및 인코딩 체계를 제공하며 복잡한 쿼리를 실행하고 대량의 데이터를 처리하는 데 적합합니다. Parquet은 Apache Arrow와 함께 사용하도록 최적화되어 있으므로 Arrow 관련 도구를 사용하는 경우 더 유용합니다.

ORC는 Hive 데이터를 저장하는 효율적인 방법을 제공합니다. ORC 파일은 종종 Parquet 파일보다 크기가 작으며 ORC 인덱스를 사용하면 쿼리 속도를 높일 수 있습니다. 또한 ORC는 구조체, 맵, 목록과 같은 복잡한 유형을 지원합니다.

Parquet 및 ORC 중에서 선택할 때는 다음을 고려합니다.

쿼리 성능 - Parquet은 더 광범위한 쿼리 유형을 지원하므로 복잡한 쿼리를 수행하려는 경우 Parquet이 더 적합합니다.

복잡한 데이터 유형 - 복잡한 데이터 유형을 사용하는 경우 다양한 복잡한 데이터 유형을 지원하는 ORC가 더 적합합니다.

파일 크기 - 디스크 공간이 우려되는 경우 일반적으로 ORC에서는 파일 크기가 더 작으므로 스토리지 비용을 절감할 수 있습니다.

압축 - Parquet 및 ORC 모두 뛰어난 압축률을 제공하지만 최적의 형식은 특정 사용 사례에 따라 달라질 수 있습니다.

진화 - Parquet과 ORC 모두 스키마 진화를 지원하므로 시간이 지남에 따라 열을 추가, 제거 또는 수정할 수 있습니다.

Parquet 및 ORC 모두 빅 데이터 애플리케이션에 적합하지만 선택하기 전에 시나리오의 요구 사항을 고려합니다. 데이터 및 쿼리에 대한 벤치마크를 수행하여 사용 사례에 더 적합한 형식을 확인하는 것이 좋습니다.

열 기반 형식으로 변환

데이터를 Apache parquet 또는 ORC 같은 오픈 소스 열 기반 형식으로 변환하면 Amazon Athena 쿼리 성능이 개선됩니다.

JSON 또는 CSV와 같은 소스 데이터를 열 형식으로 쉽게 변환할 수 있는 옵션을 통해 CREATE TABLE AS 쿼리를 사용하거나 AWS Glue의 작업을 실행할 수 있습니다.

- `CREATE TABLE AS`(CTAS) 쿼리를 통해 한 번에 데이터를 Parquet 또는 ORC로 변환할 수 있습니다. 예는 CTAS 쿼리 예제 페이지의 예제: 쿼리 결과를 다른 형식으로 쓰기를 참조하세요.
- AWS Glue 작업을 실행하여 CSV 데이터를 Parquet으로 변환하는 방법에 대한 자세한 내용은 AWS 빅 데이터 블로그 게시물 Amazon S3 및 AWS Glue를 이용한 데이터 레이크 구축하기의 'CSV에서 Parquet 형식으로 데이터 변환' 섹션을 참조하세요. AWS Glue에서는 동일한 기술을 사용하여 CSV 데이터를 ORC로 변환하거나 JSON 데이터를 Parquet 또는 ORC로 변환할 수 있습니다.

Athena에서 데이터 분할

데이터를 분할하면 각 쿼리가 스캔하는 데이터의 양을 제한하여 성능을 향상시키고 비용을 절감할 수 있습니다. 어떤 키를 기준으로도 데이터를 분할할 수 있습니다. 일반적으로 시간을 기준으로 데이터가 분할되어, 다중 레벨 파티셔닝 체계가 형성되는 경우가 많습니다. 예를 들어, 매시간 데이터를 수집하

는 고객은 연, 월, 일, 시를 기준으로 분할할 수 있습니다. 다양한 소스로부터 데이터를 수집하되 하루에 한 번만 로드하는 고객의 경우 데이터 원본 식별자 및 날짜별로 분할할 수 있습니다.

Athena는 Apache Hive 스타일 파티션을 사용할 수 있습니다. 이 파티션의 데이터 경로에 등호로 연결된 키 값 페어가 포함되어 있습니다(예: `country=us/...` 또는 `year=2021/month=01/day=26/...`). 따라서 경로에는 파티션 키의 이름과 각 경로가 나타내는 값이 모두 포함됩니다. 파티션을 나눈 테이블에 새 Hive 파티션을 로드하려면 MSCK REPAIR TABLE 명령을 사용할 수 있으며 이 명령은 Hive 스타일 파티션에서만 작동합니다.

Athena는 비 Hive 스타일 파티션 체계를 사용할 수도 있습니다. 예를 들어 CloudTrail 로그 및 Firehose 전송 스트림은 `data/2021/01/26/us/6fc7845e.json`와 같은 날짜 부분에 대해 별도의 경로 구성 요소를 사용합니다. Hive와 호환되지 않는 데이터의 경우 ALTER TABLE ADD PARTITION을(를) 사용하여 파티션을 수동으로 추가합니다.

고려 사항 및 제한

파티셔닝을 사용할 때는 다음 사항에 유의하세요.

- `WHERE` 절에서 분할된 테이블을 쿼리하고 파티션을 지정하면 Athena는 해당 파티션의 데이터만 검색합니다. 자세한 내용은 테이블 위치 및 파티션 단원을 참조하십시오.
- 객체가 많은 Amazon S3 버킷에 대해 쿼리를 실행하는데 데이터가 분할되지 않은 경우, 그러한 쿼리는 Amazon S3의 `GET` 요청 빈도 제한에 영향을 주고 Amazon S3 예외를 발생시킬 수 있습니다. 오류를 방지하려면 데이터를 분할하세요. 또한 Amazon S3 요청 빈도를 조정합니다. 자세한 내용은 모범 사례 설계 패턴: Amazon S3 성능 최적화를 참조하세요.
- Athena에 사용될 파티션 위치는 `s3` 프로토콜(예: `s3://DOC-EXAMPLE-BUCKET/folder/`)을 사용해야 합니다. Athena에서, 다른 프로토콜(예: `s3a://DOC-EXAMPLE-BUCKET/folder/`)을 사용하는 위치는 포함 테이블에서 `MSCK REPAIR TABLE` 쿼리를 실행할 때 쿼리 실패를 초래하게 됩니다.
- Amazon S3 경로가 camel 표기 대신 소문자인지 확인합니다(예: `userId` 대신 `userid`). S3 경로가 camel 표기인 경우 `MSCK REPAIR TABLE`은 AWS Glue Data Catalog에 파티션을 추가하지 않습니다. 자세한 내용은 MSCK REPAIR TABLE 단원을 참조하십시오.
- `MSCK REPAIR TABLE`은 일치하는 파티션 스키마를 찾기 위해 폴더와 하위 폴더를 모두 스캔하기 때문에 별도의 폴더 계층 구조에 있는 별도의 테이블에 데이터를 보관해야 합니다. 예를 들어 테이블 1의 데이터를 `s3://DOC-EXAMPLE-BUCKET1`에 두고 테이블 2에 대한 데이터을 `s3://DOC-EXAMPLE-BUCKET1/table-2-data`에 두었다고 가정합시다. 두 테이블이 모두 문자열로 분할된 경우 `MSCK REPAIR TABLE`은 테이블 2의 파티션을 테이블 1에 추가합니다. 이를 방지하려면 대신에 `s3://DOC-EXAMPLE-BUCKET1` 및 `s3://DOC-EXAMPLE-BUCKET2`와 같은 별도의 폴더 구조를 사용하세요. 이 동작은 Amazon EMR과 Apache Hive에서도 동일합니다.

- Athena와 함께 AWS Glue Data Catalog를 사용하는 경우 계정 및 테이블당 파티션의 서비스 할당량은 AWS Glue 엔드포인트 및 할당량을 참조하세요.
  - Athena는 1천만 개의 파티션이 있는 AWS Glue 테이블에 대한 쿼리를 지원하지만, 단일 스캔으로 1백만 개 이상의 파티션을 읽을 수는 없습니다. 이러한 시나리오에서는 파티션 인덱싱이 유용할 수 있습니다. 자세한 내용을 알아보려면 AWS Big Data Blog(빅 데이터 블로그) 문서인 Improve Amazon Athena query performance using AWS Glue Data Catalog partition indexes(파티션 인덱스를 사용하여 Amazon Athena 쿼리 성능 향상)를 참조하세요.
- AWS Glue Data Catalog를 사용하는 경우 파티션 할당량 증가를 요청하려면 AWS Glue의 Service Quotas 콘솔을 방문하세요.

분할된 데이터로 테이블 생성 및 로드

파티션을 사용하는 테이블을 생성하려면 CREATE TABLE 문에 `PARTITIONED BY` 절을 사용합니다. `PARTITIONED BY` 절은 다음 예와 같이 데이터를 분할하는 키를 정의합니다. `LOCATION` 절은 분할된 데이터의 루트 위치를 지정합니다.

```
CREATE EXTERNAL TABLE users (
first string,
last string,
username string
)
PARTITIONED BY (id string)
STORED AS parquet
LOCATION 's3://DOC-EXAMPLE-BUCKET'
```

테이블을 생성한 후 쿼리를 위해 파티션에 데이터를 로드합니다. Hive 스타일 파티션의 경우, MSCK REPAIR TABLE을(를) 실행합니다. Hive 스타일이 아닌 파티션의 경우, ALTER TABLE ADD PARTITION을(를) 사용하여 파티션을 수동으로 추가합니다.

쿼리를 위해 Hive 스타일 및 비 Hive 스타일 데이터 준비

다음 섹션에서는 Athena에서 쿼리하기 위해 Hive 스타일 및 비 Hive 스타일 데이터를 준비하는 방법을 보여줍니다.

시나리오 1: Hive 형식으로 Amazon S3에 저장된 데이터

이 시나리오에서는 Amazon S3에 있는 별도의 폴더에 파티션이 저장됩니다. 예를 들어 다음은 지정된 접두사 아래에 S3 객체를 나열하는 `aws s3 ls` 명령으로 출력된 샘플 광고 노출에 대한 부분 목록입니다.

```
aws s3 ls s3://elasticmapreduce/samples/hive-ads/tables/impressions/

    PRE dt=2009-04-12-13-00/
    PRE dt=2009-04-12-13-05/
    PRE dt=2009-04-12-13-10/
    PRE dt=2009-04-12-13-15/
    PRE dt=2009-04-12-13-20/
    PRE dt=2009-04-12-14-00/
    PRE dt=2009-04-12-14-05/
    PRE dt=2009-04-12-14-10/
    PRE dt=2009-04-12-14-15/
    PRE dt=2009-04-12-14-20/
    PRE dt=2009-04-12-15-00/
    PRE dt=2009-04-12-15-05/
```

여기서 로그는 날짜, 시간 및 분 단위로 설정된 열 이름(dt)으로 저장됩니다. 분할된 열의 상위 폴더 위치, 스키마 및 이름을 DDL에 포함하면 Athena가 해당 하위 폴더의 데이터를 쿼리할 수 있습니다.

테이블 생성

이 데이터로 테이블을 생성하려면 다음 Athena DDL 문과 같이 'dt'를 따라 파티션을 생성합니다.

```
CREATE EXTERNAL TABLE impressions (
    requestBeginTime string,
    adId string,
    impressionId string,
    referrer string,
    userAgent string,
    userCookie string,
    ip string,
    number string,
    processId string,
    browserCookie string,
    requestEndTime string,
    timers struct<modelLookup:string, requestTime:string>,
    threadId string,
    hostname string,
    sessionId string)
PARTITIONED BY (dt string)
ROW FORMAT  serde 'org.apache.hive.hcatalog.data.JsonSerDe'
LOCATION 's3://elasticmapreduce/samples/hive-ads/tables/impressions/' ;
```

이 테이블은 Hive의 기본 JSON serializer-deserializer를 사용하여 Amazon S3에 저장된 JSON 데이터를 읽습니다. 지원되는 형식에 대한 자세한 내용은 지원되는 SerDes 및 데이터 형식 단원을 참조하세요.

MSCK REPAIR TABLE 실행

CREATE TABLE 쿼리 실행 후 Athena 쿼리 편집기에서 MSCK REPAIR TABLE 명령을 실행하여 다음 예와 같이 파티션을 로드합니다.

```
MSCK REPAIR TABLE impressions
```

이 명령을 실행하면 데이터를 쿼리할 준비가 됩니다.

데이터 쿼리

파티션 열을 사용하여 노출 테이블의 데이터를 쿼리합니다. 다음은 그 예입니다.

```
SELECT dt,impressionid FROM impressions WHERE dt<'2009-04-12-14-00' and
 dt>='2009-04-12-13-00' ORDER BY dt DESC LIMIT 100
```

이 쿼리는 다음과 비슷한 결과를 표시합니다.

```
2009-04-12-13-20    ap3HcVKAWfXtgIPu6WpuUfAfL0DQEc
2009-04-12-13-20    17uchtodoS9kdeQP1x0XThKl5IuRsV
2009-04-12-13-20    JOUf1SCtRwviGw8sVcghqE5h0nkgtp
2009-04-12-13-20    NQ2XP0J0dvVbCXJ0pb4XvqJ5A4QxxH
2009-04-12-13-20    fFAItiBMsgqro9kRdIwbeX60SROaxr
2009-04-12-13-20    V4og4R9W6G3QjHHwF7gI1cSqig5D1G
2009-04-12-13-20    hPEPtBwk45msmwWTxPVVo1kVu4v11b
2009-04-12-13-20    v0SkfxegheD90gp31UCr6FplnKpx6i
2009-04-12-13-20    1iD9odVgOIi4QWkwHMcOhmwTkWDKfj
2009-04-12-13-20    b31tJiIA25CK8eDHQrHnbcknfSndUk
```

시나리오 2: 데이터가 Hive 형식으로 분할되지 않음

다음 예에서 aws s3 ls 명령은 Amazon S3에 저장된 ELB 로그를 보여줍니다. 데이터 레이아웃이 key=value 페어를 사용하지 않으므로 Hive 형식이 아니라는 점에 유의하세요. (aws s3 ls 명령에 대한 --recursive 옵션은 지정된 디렉터리 또는 접두사 아래의 모든 파일 또는 객체를 나열하도록 지정합니다.)

```
aws s3 ls s3://athena-examples-myregion/elb/plaintext/ --recursive
```

```
2016-11-23 17:54:46   11789573 elb/plaintext/2015/01/01/part-r-00000-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:46    8776899 elb/plaintext/2015/01/01/part-r-00001-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:46    9309800 elb/plaintext/2015/01/01/part-r-00002-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47    9412570 elb/plaintext/2015/01/01/part-r-00003-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47   10725938 elb/plaintext/2015/01/01/part-r-00004-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:46    9439710 elb/plaintext/2015/01/01/part-r-00005-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47          0 elb/plaintext/2015/01/01_$folder$
2016-11-23 17:54:47    9012723 elb/plaintext/2015/01/02/part-r-00006-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47    7571816 elb/plaintext/2015/01/02/part-r-00007-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:47    9673393 elb/plaintext/2015/01/02/part-r-00008-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48   11979218 elb/plaintext/2015/01/02/part-r-00009-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48    9546833 elb/plaintext/2015/01/02/part-r-00010-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48   10960865 elb/plaintext/2015/01/02/part-r-00011-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48          0 elb/plaintext/2015/01/02_$folder$
2016-11-23 17:54:48   11360522 elb/plaintext/2015/01/03/part-r-00012-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48   11211291 elb/plaintext/2015/01/03/part-r-00013-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:48    8633768 elb/plaintext/2015/01/03/part-r-00014-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:49   11891626 elb/plaintext/2015/01/03/part-r-00015-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:49    9173813 elb/plaintext/2015/01/03/part-r-00016-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:49   11899582 elb/plaintext/2015/01/03/part-r-00017-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:49          0 elb/plaintext/2015/01/03_$folder$
2016-11-23 17:54:50    8612843 elb/plaintext/2015/01/04/part-r-00018-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:50   10731284 elb/plaintext/2015/01/04/part-r-00019-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
```

```
2016-11-23 17:54:50    9984735 elb/plaintext/2015/01/04/part-r-00020-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:50    9290089 elb/plaintext/2015/01/04/part-r-00021-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:50    7896339 elb/plaintext/2015/01/04/part-r-00022-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    8321364 elb/plaintext/2015/01/04/part-r-00023-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51          0 elb/plaintext/2015/01/04_$folder$
2016-11-23 17:54:51    7641062 elb/plaintext/2015/01/05/part-r-00024-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51   10253377 elb/plaintext/2015/01/05/part-r-00025-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    8502765 elb/plaintext/2015/01/05/part-r-00026-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51   11518464 elb/plaintext/2015/01/05/part-r-00027-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    7945189 elb/plaintext/2015/01/05/part-r-00028-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    7864475 elb/plaintext/2015/01/05/part-r-00029-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51          0 elb/plaintext/2015/01/05_$folder$
2016-11-23 17:54:51   11342140 elb/plaintext/2015/01/06/part-r-00030-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:51    8063755 elb/plaintext/2015/01/06/part-r-00031-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    9387508 elb/plaintext/2015/01/06/part-r-00032-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    9732343 elb/plaintext/2015/01/06/part-r-00033-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52   11510326 elb/plaintext/2015/01/06/part-r-00034-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    9148117 elb/plaintext/2015/01/06/part-r-00035-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52          0 elb/plaintext/2015/01/06_$folder$
2016-11-23 17:54:52    8402024 elb/plaintext/2015/01/07/part-r-00036-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52    8282860 elb/plaintext/2015/01/07/part-r-00037-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:52   11575283 elb/plaintext/2015/01/07/part-r-00038-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:53    8149059 elb/plaintext/2015/01/07/part-r-00039-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
```

```
2016-11-23 17:54:53   10037269 elb/plaintext/2015/01/07/part-r-00040-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:53   10019678 elb/plaintext/2015/01/07/part-r-00041-ce65fca5-
d6c6-40e6-b1f9-190cc4f93814.txt
2016-11-23 17:54:53          0 elb/plaintext/2015/01/07_$folder$
2016-11-23 17:54:53          0 elb/plaintext/2015/01_$folder$
2016-11-23 17:54:53          0 elb/plaintext/2015_$folder$
```

ALTER TABLE ADD PARTITION 실행

데이터가 Hive 형식이 아니므로 `MSCK REPAIR TABLE` 명령을 사용하여 테이블을 생성한 후 테이블에 파티션을 추가합니다. 그 대신 ALTER TABLE ADD PARTITION 명령을 사용하여 각 파티션을 수동으로 추가할 수 있습니다. 예를 들어 s3://athena-examples-*myregion*/elb/plaintext/2015/01/01/에 데이터를 로드하려면 다음 쿼리를 실행합니다. 각 Amazon S3 폴더에 대해 별도의 파티션 열이 필요하지 않으며, 파티션 키 값이 Amazon S3 키와 다를 수 있습니다.

```
ALTER TABLE elb_logs_raw_native_part ADD PARTITION (dt='2015-01-01') location 's3://
athena-examples-us-west-1/elb/plaintext/2015/01/01/'
```

파티션이 이미 있는 경우 오류(파티션이 이미 있습니다.)가 수신됩니다. 이 오류를 방지하기 위해 `IF NOT EXISTS` 절을 사용할 수 있습니다. 자세한 내용은 ALTER TABLE ADD PARTITION 단원을 참조하십시오. 파티션을 제거하려면 ALTER TABLE DROP PARTITION을 사용합니다.

파티션 프로젝션

파티션을 관리하지 않으려면 파티션 프로젝션을 사용합니다. 파티션 프로젝션은 구조가 미리 알려진 고도로 분할된 테이블에 대한 옵션입니다. 파티션 프로젝션에서 파티션 값 및 위치는 메타데이터 리포지토리에서 읽지 않고 구성하는 테이블 속성에서 계산됩니다. 인메모리 계산은 원격 조회보다 빠르기 때문에 파티션 프로젝션을 사용하면 쿼리 런타임을 크게 줄일 수 있습니다.

자세한 내용은 Amazon Athena를 사용한 파티션 프로젝션 단원을 참조하십시오.

추가 리소스

- Firehose 데이터의 분할 옵션에 대한 자세한 내용은 Amazon Data Firehose 예제 섹션을 참조하세요.
- JDBC 드라이버를 사용하여 파티션 추가를 자동화할 수도 있습니다.
- CTAS 및 INSERT INTO를 사용하여 데이터 세트를 분할할 수 있습니다. 자세한 내용은 ETL 및 데이터 분석에 CTAS 및 INSERT INTO 사용 단원을 참조하십시오.

Amazon Athena를 사용한 파티션 프로젝션

Athena에서 파티션 프로젝션을 사용하여 고도로 분할된 테이블의 쿼리 처리 속도를 높이고 파티션 관리를 자동화할 수 있습니다.

파티션 프로젝션에서 Athena는 AWS Glue에서 테이블에 직접 구성하는 테이블 속성에서 파티션 값 및 위치를 계산합니다. 테이블 속성을 사용하면 Athena는 필요한 파티션 정보를 '프로젝션'하거나 결정할 수 있으므로 AWS Glue Data Catalog에서 시간이 많이 걸리는 메타데이터를 검색하지 않아도 됩니다. 메모리 내 작업은 주로 원격 작업보다 빠르기 때문에 파티션 프로젝션은 고도로 분할된 테이블에 대한 쿼리의 실행 시간을 줄일 수 있습니다. 쿼리 및 기본 데이터의 특정 특성에 따라 파티션 프로젝션은 파티션 메타데이터 검색에 제한되는 쿼리에 대한 쿼리 실행 시간을 크게 줄일 수 있습니다.

고도로 분할된 테이블에 대한 정리 및 프로젝션

파티션 정리는 메타데이터를 수집하여 쿼리에 적용되는 파티션으로만 해당 메타데이터를 “정리”합니다. 이렇게 하면 종종 쿼리 속도가 높아집니다. Athena는 파티션 프로젝션용으로 구성된 테이블을 포함하여 파티션 열이 있는 모든 테이블에 대해 파티션 정리를 사용합니다.

일반적으로 쿼리를 처리할 때 Athena는 파티션 정리를 수행하기 전에 AWS Glue Data Catalog에 대한 `GetPartitions` 호출을 수행합니다. 테이블에 많은 수의 파티션이 있는 경우 `GetPartitions`를 사용하면 성능에 부정적인 영향을 줄 수 있습니다. 이를 방지하기 위해 파티션 프로젝션을 사용할 수 있습니다. 파티션 프로젝션을 사용하면 파티션 프로젝션 구성이 파티션 자체를 빌드하는 데 필요한 모든 정보를 Athena에 제공하므로 Athena는 `GetPartitions`를 호출하지 않아도 됩니다.

파티션 프로젝션 사용

파티션 프로젝션을 사용하려면 AWS Glue Data Catalog 또는 외부 Hive 메타스토어의 테이블 속성에서 각 파티션 열에 대한 파티션 값 및 프로젝션 형식의 범위를 지정합니다. 테이블의 이러한 사용자 지정 속성을 사용하면 Athena가 테이블에서 쿼리를 실행할 때 예상되는 파티션 패턴을 알 수 있습니다. 쿼리를 실행하는 동안 Athena는 AWS Glue Data Catalog 또는 외부 Hive 메타스토어에서 파티션 값을 검색하는 대신 이 정보를 사용하여 파티션 값을 프로젝션합니다. 이렇게 하면 쿼리 실행 시간이 단축될 뿐만 아니라 Athena, AWS Glue 또는 외부 Hive 메타스토어에 파티션을 수동으로 만들 필요가 없으므로 파티션 관리도 자동화할 수 있습니다.

> **Important**
>
> 테이블에서 파티션 프로젝션을 활성화하면 Athena는 AWS Glue Data Catalog 또는 Hive 메타스토어의 테이블에 등록된 모든 파티션 메타데이터를 무시합니다.

사용 사례

파티션 프로젝션이 유용한 시나리오는 다음과 같습니다.

- 고도로 분할된 테이블에 대한 쿼리는 원하는 만큼 빨리 완료되지 않습니다.
- 데이터에 새 날짜 또는 시간 파티션이 생성될 때 정기적으로 테이블에 파티션을 추가합니다. 파티션 프로젝션을 사용하면 새 데이터가 도착할 때 사용할 수 있는 상대 날짜 범위를 구성할 수 있습니다.
- Amazon S3에 고도로 분할된 데이터가 있습니다. AWS Glue Data Catalog 또는 Hive 메타스토어에서 데이터를 모델링하는 것은 실용적이지 않으며 쿼리는 데이터의 작은 부분만 읽습니다.

프로젝션 가능한 파티션 구조

파티션 프로젝션은 파티션이 다음과 같은(이에 국한되지 않음) 예측 가능한 패턴을 따를 때 가장 쉽게 구성할 수 있습니다.

- 정수 – `[1, 2, 3, 4, ..., 1000]` 또는 `[0500, 0550, 0600, ..., 2500]`과 같은 정수의 연속 시퀀스입니다.
- 날짜 – `[20200101, 20200102, ..., 20201231]` 또는 `[1-1-2020 00:00:00, 1-1-2020 01:00:00, ..., 12-31-2020 23:00:00]`과 같은 날짜 또는 날짜/시간의 연속 시퀀스입니다.
- 열거형 값 - 공항 코드 또는 AWS 리전과 같은 열거형 값의 유한 집합입니다.
- AWS 서비스 로그 - AWS 서비스 로그에는 일반적으로 AWS Glue에서 지정할 수 있고 Athena가 파티션 프로젝션에 사용할 수 있는 파티션 체계를 가진 알려진 구조가 있습니다.

파티션 경로 템플릿 사용자 지정

기본적으로 Athena는 `s3://DOC-EXAMPLE-BUCKET/<table-root>/partition-col-1=<partition-col-1-val>/partition-col-2=<partition-col-2-val>/` 양식을 사용하여 파티션 위치를 작성하지만 데이터가 다르게 구성된 경우 Athena는 이 경로 템플릿을 사용자 지정할 수 있는 메커니즘을 제공합니다. 단계는 사용자 지정 S3 스토리지 위치 지정를 참조하세요.

고려 사항 및 제한

다음 사항을 고려하세요.

- 파티션 프로젝션을 사용할 경우 AWS Glue 또는 외부 Hive 메타스토어에서 수동으로 파티션을 지정할 필요가 없습니다.
- 테이블에서 파티션 프로젝션을 활성화하면 Athena는 해당 테이블에 대한 AWS Glue Data Catalog 또는 외부 Hive 메타스토어의 파티션 메타데이터를 무시합니다.

- Amazon S3에 프로젝션된 파티션이 없는 경우 Athena는 계속해서 파티션을 프로젝션합니다. Athena가 오류를 발생시키지는 않지만 데이터가 반환되지 않습니다. 그러나 너무 많은 파티션이 비어 있으면 기존 AWS Glue 파티션에 비해 성능이 느려질 수 있습니다. 프로젝션된 파티션의 절반 이상이 비어 있으면 기존 파티션을 사용하는 것이 좋습니다.
- 파티션 프로젝션에 대해 정의된 범위 한계를 벗어난 값에 대한 쿼리는 오류를 반환하지 않습니다. 그 대신 쿼리가 실행되지만 0개의 행을 반환합니다. 예를 들어 2,020으로 시작하고 `'projection.timestamp.range'='2020/01/01,NOW'`로 정의된 시간 관련 데이터가 있는 경우 `SELECT * FROM table-name WHERE timestamp = '2019/02/02'` 같은 쿼리는 성공적으로 완료되지만 0개의 행을 반환합니다.
- Athena를 통해 테이블을 쿼리할 때만 파티션 프로젝션을 사용할 수 있습니다. Amazon Redshift Spectrum, Athena for Spark, Amazon EMR 등의 다른 서비스를 통해 동일한 테이블을 읽는 경우 표준 파티션 메타데이터가 사용됩니다.
- 파티션 프로젝션은 DML 전용 기능이므로 `SHOW PARTITIONS`에서는 Athena에 의해 프로젝션되었지만 AWS Glue 카탈로그 또는 외부 Hive 메타스토어에 등록되지 않은 파티션은 나열하지 않습니다.
- Athena는 파티션 프로젝션을 위한 구성으로 뷰의 테이블 속성을 사용하지 않습니다. 이 제한을 해결하려면 뷰가 참조하는 테이블의 테이블 속성에서 파티션 프로젝션을 구성하고 활성화합니다.
- Lake Formation 데이터 필터는 Athena의 파티션 프로젝션에 사용할 수 없습니다.

비디오

다음 비디오는 Athena에서 쿼리의 성능을 향상시키기 위해 파티션 프로젝션을 사용하는 방법을 보여줍니다.

Amazon Athena를 사용한 파티션 프로젝션

주제

- 파티션 프로젝션 설정
- 파티션 프로젝션에 지원되는 형식
- 동적 ID 분할
- Amazon Data Firehose 예제

파티션 프로젝션 설정

테이블의 속성에서 파티션 프로젝션을 설정하는 과정은 두 단계로 이루어집니다.

1. 각 파티션 열에 대한 데이터 범위 및 관련 패턴을 지정하거나 사용자 지정 템플릿을 사용합니다.

2. 테이블에 대해 파티션 프로젝션을 활성화합니다.

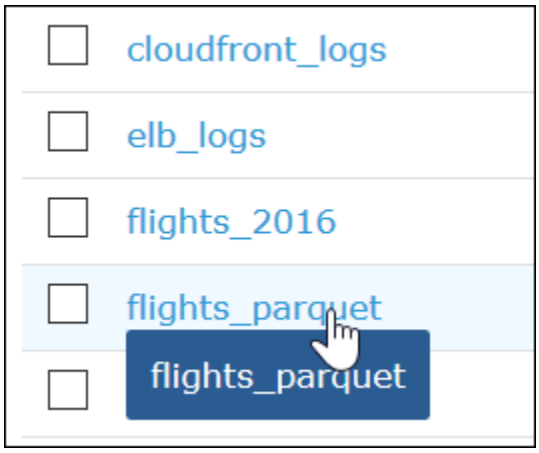
> **Note**
>
> 기존 테이블에 파티션 프로젝션 속성을 추가하기 전에 파티션 프로젝션 속성을 설정하려는 파티션 열이 테이블 스키마에 이미 있어야 합니다. 파티션 열이 아직 없는 경우 기존 테이블에 파티션 열을 수동으로 추가해야 합니다. AWS Glue에서는 이 단계를 자동으로 수행하지 않습니다.

이 단원에서는 AWS Glue에 대해 테이블 속성을 설정하는 방법을 보여줍니다. 이러한 속성을 설정하기 위해 AWS Glue 콘솔, Athena CREATE TABLE 쿼리 또는 AWS Glue API 작업을 사용할 수 있습니다. 다음 절차에서는 AWS Glue 콘솔에서 속성을 설정하는 방법을 보여줍니다.

AWS Glue 콘솔을 사용하여 파티션 프로젝션을 구성하고 활성화하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
2. 테이블 탭을 선택합니다.

   테이블 탭에서 기존 테이블을 편집하거나 테이블 추가를 선택하여 새 테이블을 생성할 수 있습니다. 수동으로 또는 크롤러를 사용하여 테이블을 추가하는 방법에 대한 자세한 내용은 AWS Glue 개발자 안내서의 AWS Glue 콘솔에서 테이블 관련 작업을 참조하세요.
3. 테이블 목록에서 편집하려는 테이블에 대한 링크를 선택합니다.



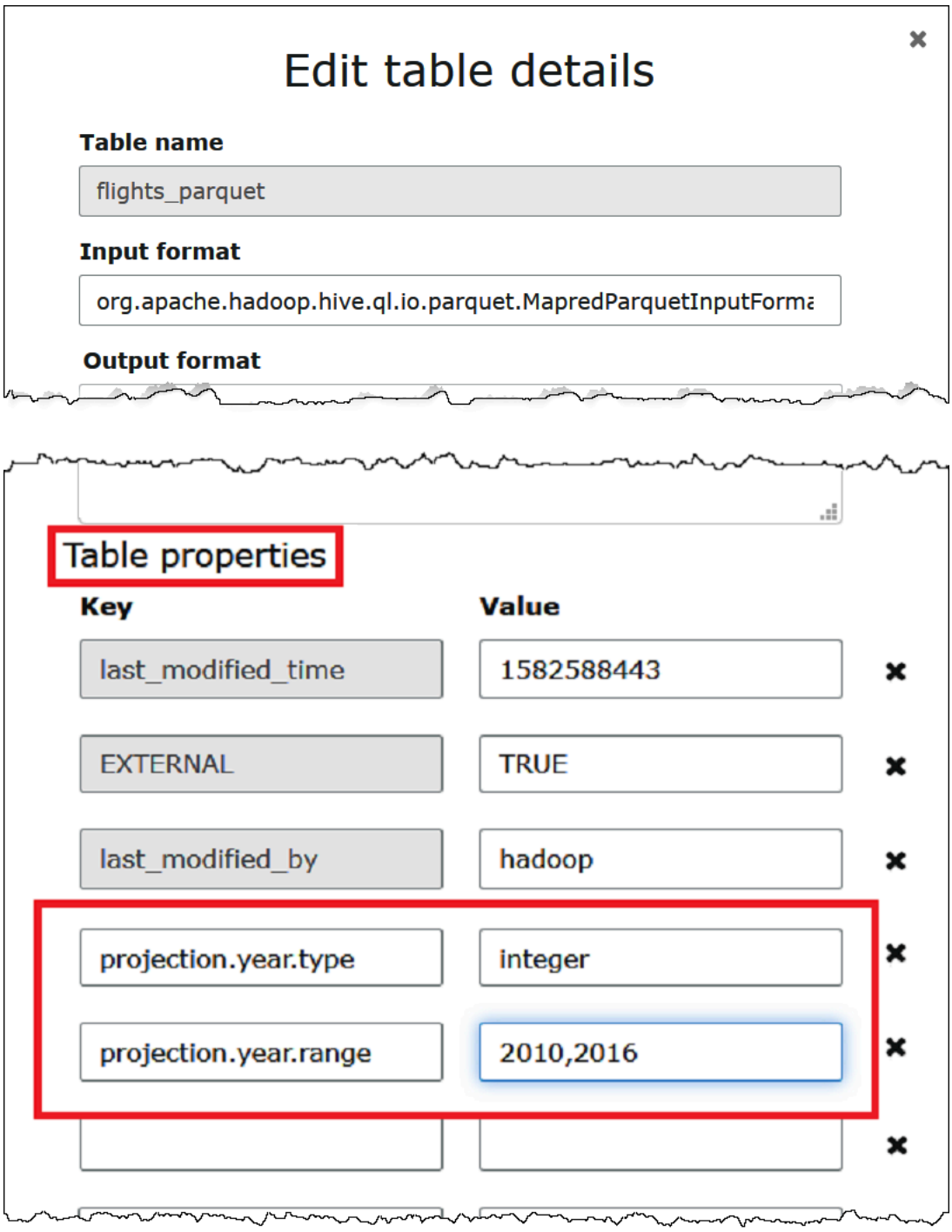
4. Actions(작업), Edit table(테이블 편집)을 선택합니다.
5. Edit table(테이블 편집) 페이지의 Table properties(테이블 속성) 섹션에서 분할된 각 열에 대해 다음 키-값 페어를 추가합니다.

a. 키에 `projection.columnName.type`을 추가합니다.

b. 값에 지원되는 형식 `enum`, `integer`, `date`, `injected` 중 하나를 추가합니다. 자세한 내용은 파티션 프로젝션에 지원되는 형식 단원을 참조하세요.

6. 파티션 프로젝션에 지원되는 형식의 지침을 따르고, 구성 요구 사항에 따라 추가적인 키-값 페어를 추가합니다.

   다음 예제 테이블 구성은 파티션 프로젝션에 대한 `year` 열을 구성하여 반환할 수 있는 값을 2010에서 2016까지의 범위로 제한합니다.

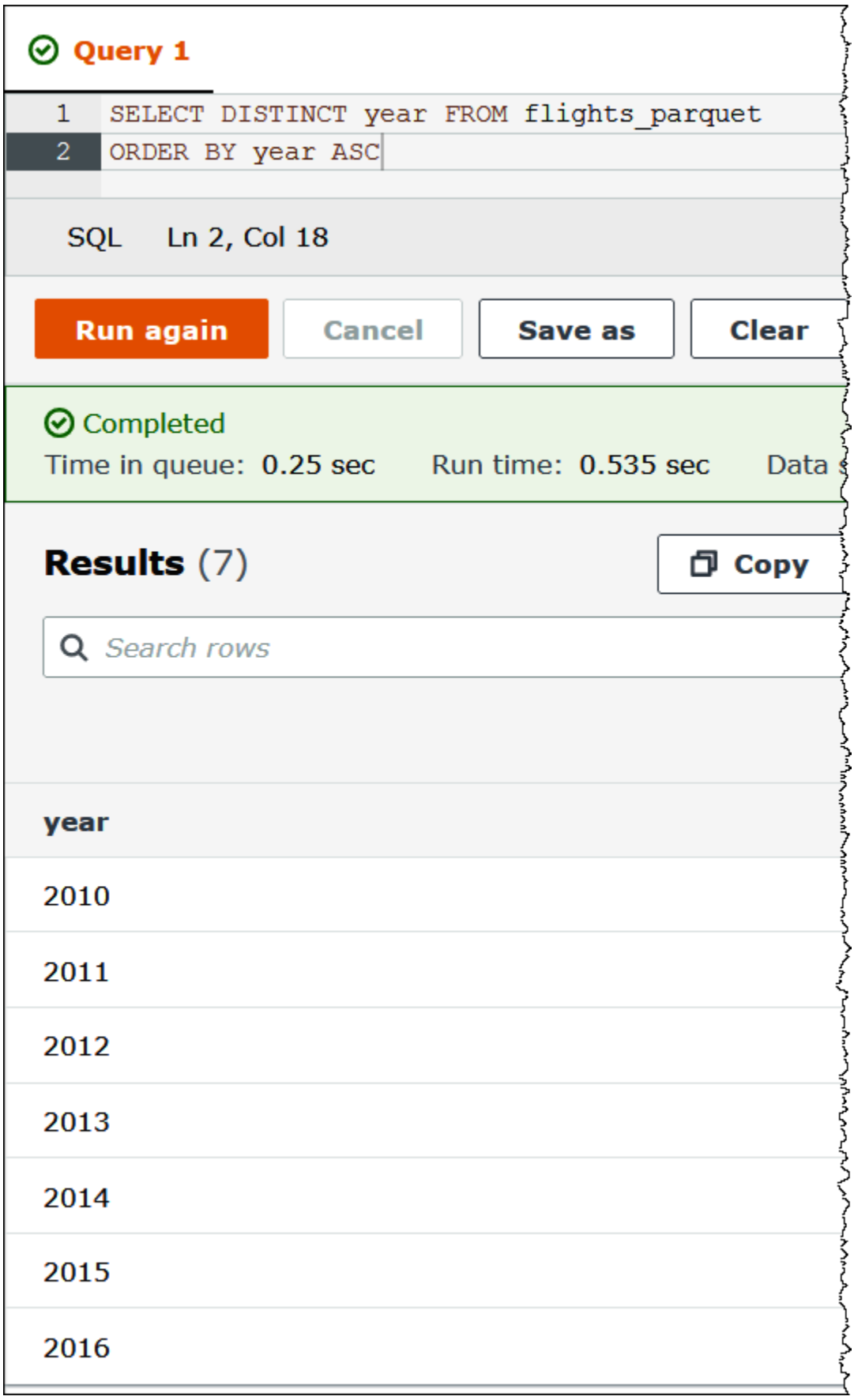


7. 키-값 페어를 추가하여 파티션 프로젝션을 활성화합니다. 키에 `projection.enabled`를 입력하고 값에 `true`를 입력합니다.

> **Note**
>
> `projection.enabled`을 `false`로 설정하여 언제든지 이 테이블에서 파티션 프로젝션을 비활성화할 수 있습니다.

8. 작업을 마쳤으면 저장을 선택합니다.
9. Athena 쿼리 편집기에서 테이블에 대해 구성한 열을 테스트 쿼리합니다.

   다음 예제 쿼리는 `SELECT DISTINCT`를 사용하여 `year` 열에서 고유한 값을 반환합니다. 데이터베이스에는 1987년부터 2016년까지 데이터가 포함되어 있지만 `projection.year.range` 속성은 반환되는 값을 2010년에서 2016년까지로 제한합니다.


Query 1
1 SELECT DISTINCT year FROM flights_parquet
2 ORDER BY year ASC
SQL Ln 2, Col 18
Run again
Cancel
Save as
Clear
Completed
Time in queue: 0.25 sec
Run time: 0.535 sec
Data
Results (7)
Copy
Search rows
year
2010
2011
2012
2013
2014
2015
2016

> **Note**
>
> `projection.enabled`를 `true`로 설정했지만 하나 이상의 파티션 열을 구성하지 못하면 다음과 같은 오류 메시지가 나타납니다.
> `HIVE_METASTORE_ERROR: Table database_name.table_name is configured for partition projection, but the following partition columns are missing projection configuration: [column_name] (table database_name.table_name).`

사용자 지정 S3 스토리지 위치 지정

AWS Glue에서 테이블 속성을 편집할 때 프로젝션된 파티션에 대한 사용자 지정 Amazon S3 경로 템플릿을 지정할 수도 있습니다. 사용자 지정 템플릿을 사용하면 Athena가 일반적인 `.../column=value/...` 패턴을 따르지 않는 사용자 지정 Amazon S3 파일 위치에 파티션 값을 올바르게 매핑할 수 있습니다.

사용자 지정 템플릿을 사용하는 것은 선택 사항입니다. 그러나 사용자 지정 템플릿을 사용하는 경우 템플릿에 각 파티션 열에 대한 자리 표시자를 포함시켜야 합니다. 분할된 데이터 파일이 파티션별 '폴더'에 상주하도록 템플릿 지정된 위치는 슬래시로 끝나야 합니다.

사용자 지정 파티션 위치 템플릿을 지정하려면

1. AWS Glue 콘솔을 사용하여 파티션 프로젝션을 구성하고 활성화하는 단계에 따라 다음과 같이 사용자 지정 템플릿을 지정하는 키-값 페어를 추가합니다.

   a. 키에 `storage.location.template`를 입력합니다.

   b. 값에서 모든 파티션 열에 대한 자리 표시자를 포함하는 위치를 지정합니다. 각 자리 표시자(및 S3 경로 자체)가 단일 슬래시로 종료되었는지 확인합니다.

      다음 예제 템플릿 값은 테이블에 파티션 열 a, b 및 c가 있다고 가정합니다.

      ```
      s3://DOC-EXAMPLE-BUCKET/table_root/a=${a}/${b}/some_static_subdirectory/${c}/
      ```

      ```
      s3://DOC-EXAMPLE-BUCKET/table_root/c=${c}/${b}/some_static_subdirectory/${a}/
      ${b}/${c}/${c}/
      ```

같은 테이블의 경우 다음 예제 템플릿 값은 열 c에 대한 자리 표시자가 포함되어 있지 않기 때문에 유효하지 않습니다.

```
s3://DOC-EXAMPLE-BUCKET/table_root/a=${a}/${b}/some_static_subdirectory/
```

2. 적용을 선택합니다.

파티션 프로젝션에 지원되는 형식

테이블에는 enum, integer, date, 또는 injected 파티션 열 형식의 조합이 있을 수 있습니다.

열거형 형식

값이 열거형 집합의 멤버(예: 공항 코드 또는 AWS 리전)인 파티션 열의 enum 형식을 사용합니다.

테이블에서 다음과 같이 파티션 속성을 정의합니다.

속성 이름	예제 값	설명
projection. *columnName* .type	enum	필수 사항입니다. *columnName* 열에 사용할 프로젝션 형식입니다. 열거형 형식의 사용을 나타내기 위해서는 값이 enum이어야 합니다(대소문자 구분 안 함). 선행 및 후행 공백을 사용할 수 있습니다.
projection. *columnName* .values	A,B,C,D,E,F,G,Unknown	필수 사항입니다. *ColumnName* 열에 대한 열거형 파티션 값의 쉼표로 구분된 목록입니다. 공백은 열거형 값의 일부로 간주됩니다.

> **Note**
>
> 가장 좋은 방법은 enum 기반 파티션 프로젝션 사용을 수십 개 이하로 제한하는 것입니다 enum 프로젝션에 대해 구체적인 제한은 없지만 테이블의 메타데이터 총 크기는 gzip 압축 시 약 1MB

의 AWS Glue 제한을 초과할 수 없습니다. 이 제한은 열 이름, 위치, 스토리지 형식 등과 같은 테이블의 주요 부분에서 공유됩니다. 자신이 enum 프로젝션에 수십 개 이상의 고유 ID를 사용하고 있다면 대리 필드에서 더 적은 수의 고유 값으로 버킷팅하는 등의 대안적 방법을 고려하는 것이 좋습니다. 카디널리티를 포기함으로써 enum 필드에서 고유 값의 수를 제어할 수 있습니다.

정수 형식

가능한 값을 정의된 범위 내의 정수로 해석할 수 있는 파티션 열에 대해 정수 형식을 사용합니다. 프로젝션 정수 열은 현재 Java 부호 있는 long(-2^{63}~2^{63}-1 포함)의 범위로 제한됩니다.

속성 이름	예제 값	설명
`projection.`*`columnName`*`.type`	`integer`	필수 사항입니다. *`columnName`* 열에 사용할 프로젝션 형식입니다. 정수 형식의 사용을 나타내기 위해서는 값이 `integer`여야 합니다(대소문자 구분 안 함). 선행 및 후행 공백을 사용할 수 있습니다.

`projection.`*`columnName`*`.range`	`0,10`
	
	`-1,8675309`
	
	`0001,9999`
		필수 사항입니다. *`columnName`* 열에 대한 쿼리에서 반환할 최소 및 최대 범위 값을 제공하는 두 요소의 쉼표로 구분된 목록입니다. 값을 하이픈이 아닌 쉼표로 구분해야 합니다. 이 값은 포함되며 음수일 수 있으며 선행 0을 포함할 수 있습니다. 선행 및 후행 공백을 사용할 수 있습니다.

`projection.`*`columnName`*`.interval`	`1`
	
	`5`
		선택 사항입니다. *`columnName`* 열에 대한 연속 파티션 값 사이의 간격을 지정하는 양의 정수입니다. 예를 들어 `interval` 값이 "1"인 `range`

속성 이름	예제 값	설명
		값 “1,3"은 1, 2, 3 값을 생성합니다. `interval` 값이 “2"인 동일한 `range` 값은 값 1과 3을 생성하며 2를 건너뜁니다. 선행 및 후행 공백을 사용할 수 있습니다. 기본 값은 1입니다.

`projection.` *`columnName`* `.digits`	`1`
	
	`5`
		선택 사항입니다. *`columnName`* 열에 대한 파티션 값의 최종 표시에 포함할 자릿수를 지정하는 양의 정수입니다. 예를 들어 `digits` 값이 “1"인 `range` 값 “1,3"은 1, 2, 3 값을 생성합니다. `digits` 값이 “2"인 동일한 `range` 값은 01, 02 및 03 값을 생성합니다. 선행 및 후행 공백을 사용할 수 있습니다. 기본 값은 고정 자릿수와 선행 0이 없습니다.

날짜 형식

정의된 범위 내에서 값을 날짜(선택적 시간 포함)로 해석할 수 있는 파티션 열에 대해 날짜 형식을 사용합니다.

> **Important**
>
> 프로젝션 날짜 열은 쿼리 실행 시 UTC(협정 세계 표준시)로 생성됩니다.

속성 이름	예제 값	설명
`projection.` *`columnName`* `.type`	`date`	필수 사항입니다. *`columnName`* 열에 사용할 프로젝션 형식입니다. 날짜 형식의 사용을 나타내기 위해서는 값이 `date`여야 합니다(대소문자 구

속성 이름	예제 값	설명
		분 안 함). 선행 및 후행 공백을 사용할 수 있습니다.
projection. *columnName* .range	201701,201812 01-01-2010,12-31-2018 NOW-3YEARS,NOW 201801,NOW+1MONTH	필수 사항입니다. *columnName* 열에 대한 최소 및 최대 range 값을 제공하는 두 요소의 쉼표로 구분된 목록입니다. 이 값은 포함되며 Java java.time.* 날짜 형식과 호환되는 모든 형식을 사용할 수 있습니다. 최소값과 최대값 모두 동일한 형식을 사용해야 합니다. .format 특성에 지정된 형식은 이러한 값에 사용되는 형식이어야 합니다. 이 열에는 이 정규식 패턴으로 서식이 지정된 상대 날짜 문자열도 포함될 수 있습니다. \s*NOW\s*(([\+\-])\s*([0-9]+)\s*(YEARS?|MONTHS?|WEEKS?|DAYS?|HOURS?|MINUTES?|SECONDS?)\s*)? 공백은 허용되지만 날짜 리터럴은 날짜 문자열 자체의 일부로 간주됩니다.
projection. *columnName* .format	yyyyMM dd-MM-yyyy dd-MM-yyyy-HH-mm-ss	필수 사항입니다. Java 날짜 형식 DateTimeFormatter를 기반으로 하는 날짜 형식 문자열입니다. 지원되는 모든 Java.time.* 형식일 수 있습니다.

속성 이름	예제 값	설명
projection. *columnNam e* .interval	1 5	*columnName#* 열에 대한 연속 파티션 값 사이의 간격을 지정하는 양의 정수입니다. 예를 들어 interval 값이 1이고 interval.unit 값이 MONTHS인 range값 2017-01,2018-12 는 2017-01, 2017-02, 2017-03 등의 값을 생성합니다. interval 값이 2이고 interval.unit 값이 MONTHS인 동일한 range 값은 2017-01, 2017-03, 2017-05 등의 값을 생성합니다. 선행 및 후행 공백을 사용할 수 있습니다. 제공된 날짜가 하루 또는 한 달 정밀도인 경우 interval은 선택 사항이며 기본값은 각각 1일 또는 1개월입니다. 그렇지 않으면 interval이 필요합니다.
projection. *columnNam e* .interval.unit	YEARS MONTHS WEEKS DAYS HOURS MINUTES SECONDS MILLIS	ChronoUnit의 직렬화된 형식을 나타내는 시간 단위 단어입니다. 가능한 값은 YEARS, MONTHS, WEEKS, DAYS, HOURS, MINUTES, SECONDS 또는 MILLIS입니다. 이러한 값은 대/소문자를 구분합니다. 제공된 날짜가 하루 또는 한 달 정밀도인 경우 interval.unit 은 선택 사항이며 기본값은 각각 1일 또는 1개월입니다. 그렇지 않으면 interval.unit 이 필요합니다.

삽입 형식

일부 논리적 범위 내에서 절차에 따라 생성될 수 없지만 쿼리의 WHERE 절에서 단일 값으로 제공되는 가능한 값을 가진 파티션 열에 대해 삽입된 형식을 사용하세요.

다음 사항을 명심해야 합니다.

- 삽입된 각 열에 대해 필터 식이 제공되지 않으면 삽입된 열에 대한 쿼리가 실패합니다.
- 주입된 열의 필터 식에 대해 여러 값이 있는 쿼리는 값이 분리된 경우에만 성공합니다.
- `string` 형식의 열만 지원됩니다.

속성 이름	값	설명
`projection.` *`columnNam e`* `.type`	`inje`	필수 사항입니다. *`columnName`* 열에 사용할 프로젝션 형식입니다. `string` 형식만 지원됩니다. 지정된 값은 `injected`여야 합니다(대소문자 구분 안 함). 선행 및 후행 공백을 사용할 수 있습니다.

자세한 내용은 [injected 프로젝션 유형 사용](#) 단원을 참조하세요.

동적 ID 분할

데이터가 카디널리티가 높은 속성으로 파티션되거나 값을 미리 알 수 없는 경우 `injected` 프로젝션 유형을 사용할 수 있습니다. 이러한 속성의 예로 사용자 이름과 디바이스 또는 제품의 ID가 있습니다. `injected` 프로젝션 유형을 사용하여 파티션 키를 구성하면 Athena는 쿼리 자체의 값을 사용하여 읽을 파티션 세트를 계산합니다.

다음 조건을 충족해야 Athena가 `injected` 프로젝션 유형으로 구성된 파티션 키가 있는 테이블에서 쿼리를 실행할 수 있습니다.

- 쿼리에는 파티션 키 값이 하나 이상 포함해야 합니다.
- 값은 데이터를 읽지 않고도 평가할 수 있는 리터럴 또는 표현식이어야 합니다.

이러한 기준 중 하나라도 충족되지 않으면 쿼리가 실패하고 다음 오류가 발생합니다.

CONSTRAINT_VIOLATION: 삽입된 프로젝션 파티션 열 *`column_name`*에는 WHERE 절에 하나 이상의 정적 같음 조건만 포함되어야 합니다.

injected 프로젝션 유형 사용

IoT 디바이스의 이벤트로 구성되고 디바이스 ID로 파티션된 데이터 세트가 있다고 가정해 보겠습니다. 다음은 이 데이터 세트의 특성입니다.

- 디바이스 ID는 무작위로 생성됩니다.

- 새 디바이스는 자주 프로비저닝됩니다.
- 현재 수십만 개의 디바이스가 있으며 미래에는 수백만 개가 될 것입니다.

기존 메타스토어로는 이 데이터 세트를 관리하기 어렵습니다. 데이터 스토리지와 메타스토어 간에 파티션을 동기화된 상태로 유지하기가 어렵고 쿼리를 계획하는 동안 파티션 필터링이 느려질 수 있습니다. 그러나 파티션 프로젝션을 사용하고 `injected` 프로젝션 유형을 사용하도록 테이블을 구성하면 메타스토어에서 파티션을 관리할 필요가 없고 쿼리에서 파티션 메타데이터를 조회할 필요가 없다는 두 가지 이점이 있습니다.

다음 `CREATE TABLE` 예제에서는 방금 설명한 디바이스 이벤트 데이터 세트에 대한 테이블을 생성합니다. 테이블에서는 injected 프로젝션 유형을 사용합니다.

```
CREATE EXTERNAL TABLE device_events (
   event_time TIMESTAMP,
  data STRING,
  battery_level INT
)
PARTITIONED BY (
  device_id STRING
)
LOCATION "s3://DOC-EXAMPLE-BUCKET/prefix/"
TBLPROPERTIES (
  "projection.enabled" = "true",
  "projection.device_id.type" = "injected",
  "storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/prefix/${device_id}"
)
```

다음 예제 쿼리는 12시간 동안 3개의 특정 디바이스에서 수신한 이벤트 수를 조회합니다.

```
SELECT device_id, COUNT(*) AS events
FROM device_events
WHERE device_id IN (
  '4a770164-0392-4a41-8565-40ed8cec737e',
  'f71d12cf-f01f-4877-875d-128c23cbde17',
  '763421d8-b005-47c3-ba32-cc747ab32f9a'
)
AND event_time BETWEEN TIMESTAMP '2023-11-01 20:00' AND TIMESTAMP '2023-11-02 08:00'
GROUP BY device_id
```

이 쿼리를 실행하면 Athena는 device_id 파티션 키에 대한 3개의 값을 확인하고 이들 값을 사용하여 파티션 위치를 계산합니다. Athena는 storage.location.template 속성 값을 사용하여 다음 위치를 생성합니다.

- s3://DOC-EXAMPLE-BUCKET/*prefix*/4a770164-0392-4a41-8565-40ed8cec737e
- s3://DOC-EXAMPLE-BUCKET/*prefix*/f71d12cf-f01f-4877-875d-128c23cbde17
- s3://DOC-EXAMPLE-BUCKET/*prefix*/763421d8-b005-47c3-ba32-cc747ab32f9a

파티션 프로젝션 구성에서 storage.location.template 속성을 제외하면 Athena는 Hive 스타일 파티셔닝을 사용하여 LOCATION의 값(예: s3://DOC-EXAMPLE-BUCKET/*prefix*/device_id=4a770164-0392-4a41-8565-40ed8cec737e)을 기준으로 파티션 위치를 예상합니다.

Amazon Data Firehose 예제

Firehose를 사용하여 데이터를 Amazon S3로 전송하는 경우 기본 구성은 다음 예제와 같은 키를 사용하여 객체를 작성합니다.

```
s3://DOC-EXAMPLE-BUCKET/prefix/yyyy/MM/dd/HH/file.extension
```

새 데이터가 수신될 때 AWS Glue Data Catalog에 추가하는 대신에 쿼리할 때 파티션을 자동으로 찾는 Athena 테이블을 생성하려면 파티션 프로젝션을 사용할 수 있습니다.

다음 CREATE TABLE 예제에서는 기본 Firehose 구성을 사용합니다.

```
CREATE EXTERNAL TABLE my_ingested_data (
 ...
)
...
PARTITIONED BY (
 datehour STRING
)
LOCATION "s3://DOC-EXAMPLE-BUCKET/prefix/"
TBLPROPERTIES (
 "projection.enabled" = "true",
 "projection.datehour.type" = "date",
 "projection.datehour.format" = "yyyy/MM/dd/HH",
 "projection.datehour.range" = "2021/01/01/00,NOW",
 "projection.datehour.interval" = "1",
```

```
 "projection.datehour.interval.unit" = "HOURS",
 "storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/prefix/${datehour}/"
)
```

CREATE TABLE 문의 TBLPROPERTIES 절은 Athena에 다음을 알려줍니다.

- 테이블을 쿼리할 때 파티션 프로젝션 사용
- 파티션 키 datehour는 date 형식입니다(선택적 시간 포함).
- 날짜 형식 지정 방법
- 날짜 시간 범위입니다. 값을 하이픈이 아닌 쉼표로 구분해야 합니다.
- Amazon S3에 대한 데이터를 찾을 수 있는 위치

테이블을 쿼리할 때 Athena는 datehour에 대한 값을 계산하고 스토리지 위치 템플릿을 사용하여 파티션 위치 목록을 생성합니다.

date 형식 사용

프로젝션된 파티션 키에 대해 date 형식을 사용하는 경우 범위를 지정해야 합니다. Firehose 전송 스트림이 생성되기 전의 날짜에 대한 데이터가 없으므로 생성 날짜를 시작으로 사용할 수 있습니다. 향후 날짜에 대한 데이터가 없으므로 특수 토큰 NOW를 종료로 사용할 수 있습니다.

CREATE TABLE 예에서 시작 날짜는 2021년 1월 1일 UTC 자정으로 지정됩니다.

> **Note**
>
> Athena가 기존 파티션만 찾을 수 있도록 데이터가 최대한 가깝게 일치하는 범위를 구성합니다.

예제 테이블에서 쿼리가 실행되면 Athena는 값을 생성하는 범위와 조합된 datehour 파티션 키 조건을 사용합니다. 다음과 같은 쿼리를 가정합니다.

```
SELECT *
FROM my_ingested_data
WHERE datehour >= '2020/12/15/00'
AND datehour < '2021/02/03/15'
```

SELECT 쿼리의 첫 번째 조건은 CREATE TABLE 문에 의해 지정된 날짜 범위의 시작 이전 날짜를 사용합니다. 파티션 프로젝션 구성에서는 2021년 1월 1일 이전 날짜의 파티션을 지정하지 않으므로 Athena는 다음 위치에서만 데이터를 찾고 쿼리의 이전 날짜는 무시합니다.

```
s3://DOC-EXAMPLE-BUCKET/prefix/2021/01/01/00/
s3://DOC-EXAMPLE-BUCKET/prefix/2021/01/01/01/
s3://DOC-EXAMPLE-BUCKET/prefix/2021/01/01/02/
...
s3://DOC-EXAMPLE-BUCKET/prefix/2021/02/03/12/
s3://DOC-EXAMPLE-BUCKET/prefix/2021/02/03/13/
s3://DOC-EXAMPLE-BUCKET/prefix/2021/02/03/14/
```

마찬가지로 쿼리가 2021년 2월 3일 15:00 이전의 날짜와 시간에 실행된 경우 마지막 파티션은 쿼리 조건의 날짜와 시간이 아니라 현재 날짜와 시간을 반영합니다.

가장 최근 데이터를 쿼리하려는 경우, 다음 예와 같이 Athena가 미래 날짜를 생성하지 않고 시작 datehour만 지정한다는 사실을 활용할 수 있습니다.

```
SELECT *
FROM my_ingested_data
WHERE datehour >= '2021/11/09/00'
```

파티션 키 선택

파티션 프로젝션이 파티션 위치를 파티션 키에 매핑하는 방법을 지정할 수 있습니다. 이전 섹션의 CREATE TABLE 예제에서 날짜와 시간은 datehour라는 하나의 파티션 키로 결합되었지만 다른 체계도 가능합니다. 예를 들어 년, 월, 일 및 시간에 대해 별도의 파티션 키를 사용하여 테이블을 구성할 수도 있습니다.

하지만 날짜를 년, 월, 일로 분할하면 date 파티션 프로젝션 유형을 사용할 수 없습니다. 또 다른 방법은 시간에서 날짜를 분리하여 date 파티션 프로젝션 유형을 계속 활용하면서 시간 범위를 지정하는 쿼리를 더 읽기 쉽게 만드는 것입니다.

이를 염두에 두고 다음 CREATE TABLE 예에서는 날짜와 시간을 구분합니다. SQL에서는 date가 예약어이기 때문에 이 예제에서는 날짜를 나타내는 파티션 키의 이름으로 day를 사용합니다.

```
CREATE EXTERNAL TABLE my_ingested_data2 (
 ...
)
...
PARTITIONED BY (
 day STRING,
 hour INT
```

```
)
LOCATION "s3://DOC-EXAMPLE-BUCKET/prefix/"
TBLPROPERTIES (
 "projection.enabled" = "true",
 "projection.day.type" = "date",
 "projection.day.format" = "yyyy/MM/dd",
 "projection.day.range" = "2021/01/01,NOW",
 "projection.day.interval" = "1",
 "projection.day.interval.unit" = "DAYS",
 "projection.hour.type" = "integer",
 "projection.hour.range" = "0,23",
 "projection.hour.digits" = "2",
 "storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/prefix/${day}/${hour}/"
)
```

예제 CREATE TABLE 문에서 시간은 정수로 구성된 별도의 파티션 키입니다. 시간 파티션 키의 구성은 0 ~ 23의 범위를 지정하고 Athena가 파티션 위치를 생성할 때 시간을 두 자리 숫자로 형식화해야 합니다.

my_ingested_data2 테이블에 대한 쿼리는 다음과 같을 수 있습니다.

```
SELECT *
FROM my_ingested_data2
WHERE day = '2021/11/09'
AND hour > 3
```

파티션 키 유형 및 파티션 프로젝션 유형

첫 번째 CREATE TABLE 예의 datehour 키는 파티션 프로젝션 구성에서 date로 구성되지만 파티션 키의 유형은 string입니다. 두 번째 예의 day도 마찬가지입니다. 파티션 프로젝션 구성의 유형은 파티션 위치를 생성할 때 값의 형식을 지정하는 방법만 Athena에 알려줍니다. 지정한 형식은 파티션 키의 유형을 변경하지 않습니다. 쿼리에서 datehour 및 day는 string 형식입니다.

쿼리에 day = '2021/11/09'와 같은 조건이 포함된 경우 Athena는 파티션 프로젝션 구성에 지정된 날짜 형식을 사용하여 표현식의 오른쪽에 있는 문자열을 구문 분석합니다. Athena는 날짜가 구성된 범위 내에 있는지 확인한 후 날짜 형식을 다시 사용하여 저장 위치 템플릿에 날짜를 문자열로 삽입합니다.

마찬가지로 day > '2021/11/09'와 같은 쿼리 조건의 경우 Athena는 오른쪽을 구문 분석하고 구성된 범위 내에서 일치하는 모든 날짜 목록을 생성합니다. 그런 다음 날짜 형식을 사용하여 각 날짜를 스토리지 위치 템플릿에 삽입하여 파티션 위치 목록을 생성합니다.

day > '2021-11-09' 또는 day > DATE '2021-11-09'와 같은 조건을 쓰는 작업은 적용되지 않습니다. 첫 번째 경우에는 날짜 형식이 일치하지 않으며(포워드 슬래시 대신 하이픈에 유의) 두 번째 경우에는 데이터 형식이 일치하지 않습니다.

사용자 지정 접두사 및 동적 분할 사용

Firehose는 사용자 지정 접두사와 동적 파티셔닝으로 구성할 수 있습니다. 이러한 기능을 사용하여 Amazon S3 키를 구성하고 사용 사례를 더 잘 지원하는 분할 체계를 설정할 수 있습니다. 이러한 분할 체계와 함께 파티션 프로젝션을 사용하여 적절하게 구성할 수도 있습니다.

예를 들어 사용자 지정 접두사 기능을 사용하여 기본 yyyy/MM/dd/HH 체계 대신 ISO 형식의 날짜가 있는 Amazon S3 키를 가져올 수 있습니다.

사용자 지정 접두사를 동적 파티셔닝과 결합하여 다음 예제와 같이 Firehose 메시지의 customer_id와 같은 속성을 추출할 수 있습니다.

```
prefix/!{timestamp:yyyy}-!{timestamp:MM}-!{timestamp:dd}/!
{partitionKeyFromQuery:customer_id}/
```

Amazon S3 접두사를 사용하면 Firehose 전송 스트림에서 s3://DOC-EXAMPLE-BUCKET/prefix/2021-11-01/customer-1234/file.extension과 같은 키에 객체를 씁니다. 값을 미리 알 수 없으며 customer_id와 같은 속성의 경우 파티션 프로젝션 유형 injected를 사용하고 다음과 같이 CREATE TABLE 문을 사용할 수 있습니다.

```
CREATE EXTERNAL TABLE my_ingested_data3 (
 ...
)
...
PARTITIONED BY (
 day STRING,
 customer_id STRING
)
LOCATION "s3://DOC-EXAMPLE-BUCKET/prefix/"
TBLPROPERTIES (
 "projection.enabled" = "true",
 "projection.day.type" = "date",
 "projection.day.format" = "yyyy-MM-dd",
 "projection.day.range" = "2021-01-01,NOW",
 "projection.day.interval" = "1",
 "projection.day.interval.unit" = "DAYS",
 "projection.customer_id.type" = "injected",
 "storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/prefix/${day}/${customer_id}/"
```

```
)
```

injected 유형의 파티션 키가 있는 테이블을 쿼리할 때 쿼리에 해당 파티션 키에 대한 값이 포함되어야 합니다. my_ingested_data3 테이블에 대한 쿼리는 다음과 같을 수 있습니다.

```
SELECT *
FROM my_ingested_data3
WHERE day BETWEEN '2021-11-01' AND '2021-11-30'
AND customer_id = 'customer-1234'
```

ISO 형식의 날짜

day 파티션 키의 값은 ISO 형식이므로 다음 예와 같이 날짜 파티션 키에 STRING 대신 DATE 유형을 사용할 수도 있습니다.

```
PARTITIONED BY (day DATE, customer_id STRING)
```

쿼리할 때 이 전략을 사용하면 다음 예와 같이 구문 분석 또는 캐스팅하지 않고 파티션 키에 날짜 함수를 사용할 수 있습니다.

```
SELECT *
FROM my_ingested_data3
WHERE day > CURRENT_DATE - INTERVAL '7' DAY
AND customer_id = 'customer-1234'
```

Note

DATE 유형의 파티션 키를 지정하는 경우 사용자 지정 접두사 기능을 사용하여 날짜가 ISO 형식인 Amazon S3 키를 생성하는 것으로 가정합니다. yyyy/MM/dd/HH의 Firehose 기본 형식을 사용하는 경우 다음 예제와 같이 해당 테이블 속성이 date 유형이더라도 파티션 키를 string 유형으로 지정해야 합니다.

```
PARTITIONED BY (
  `mydate` string)
TBLPROPERTIES (
  'projection.enabled'='true',
   ...
  'projection.mydate.type'='date',
  'storage.location.template'='s3://DOC-EXAMPLE-BUCKET/prefix/${mydate}')
```

쿼리 결과에서 테이블 생성(CTAS)

CREATE TABLE AS SELECT(CTAS) 쿼리는 다른 쿼리의 SELECT 문 결과로부터 Athena의 새 테이블을 만듭니다. Athena는 CTAS 문에서 생성한 데이터 파일을 Amazon S3의 지정된 위치에 저장합니다. 구문은 CREATE TABLE AS 단원을 참조하세요.

CREATE TABLE AS에서는 CREATE TABLE DDL 문을 SELECT DML 문과 결합하므로 기술적으로 DDL과 DML을 모두 포함합니다. 그러나 서비스 할당량 측면에서 Athena의 CTAS 쿼리는 DML로 처리됩니다. Athena에서 서비스 할당량에 대한 자세한 내용은 Service Quotas 섹션을 참조하세요.

CTAS 쿼리를 사용해 다음 작업을 수행할 수 있습니다.

- 원리 데이터 세트를 반복적으로 쿼리하지 않고 쿼리 결과에서 한 번에 테이블을 생성합니다. 따라서 원시 데이터 세트를 더 쉽게 사용할 수 있습니다.
- 쿼리 결과를 변환하고 테이블을 Apache Iceberg와 같은 다른 테이블 형식으로 마이그레이션합니다. 이렇게 하면 쿼리 성능이 개선되고 Athena 쿼리 비용이 줄어듭니다. 자세한 설명은 Iceberg 테이블 생성을 참조하세요.
- 쿼리 결과를 Parquet 및 ORC 등과 같은 스토리지 형식으로 변환합니다. 이렇게 하면 쿼리 성능이 개선되고 Athena 쿼리 비용이 줄어듭니다. 자세한 설명은 열 기반 스토리지 형식을 참조하세요.
- 필요한 데이터만 포함된 기존 테이블의 복사본을 생성합니다.

주제

CTAS 쿼리에 대한 고려 사항 및 제한 사항

다음 섹션에서는 Athena에서 CREATE TABLE AS SELECT(CTAS) 쿼리를 사용할 때 유의해야 할 고려 사항과 제한 사항을 설명합니다.

CTAS 쿼리 구문

CTAS 쿼리 구문은 테이블 생성에 사용되는 `CREATE [EXTERNAL] TABLE`의 구문과 다릅니다. CREATE TABLE AS 섹션을 참조하세요.

CTAS 쿼리와 뷰

CTAS 쿼리는 Amazon S3의 지정된 위치에 새 데이터를 쓰는 반면에 뷰는 어떠한 데이터도 쓰지 않습니다.

CTAS 쿼리 결과의 위치

작업 그룹이 쿼리 결과 위치에 대한 클라이언트 측 설정을 재정의하는 경우 Athena는 `s3://DOC-EXAMPLE-BUCKET/tables/<query-id>/` 위치에 테이블을 생성합니다. 작업 그룹에 지정된 쿼리 결과 위치를 보려면 작업 그룹 세부 정보를 조회하세요.

작업 그룹이 쿼리 결과 위치를 재정의하지 않으면 CTAS 쿼리에서 `WITH (external_location ='s3://DOC-EXAMPLE-BUCKET/')` 구문을 사용하여 CTAS 쿼리 결과가 저장되는 위치를 지정할 수 있습니다.

> **Note**
>
> `external_location` 속성은 비어 있는 위치를 지정해야 합니다. CTAS 쿼리는 버킷의 경로 위치(접두사)가 비어 있는지 확인하고 해당 위치에 이미 데이터가 있는 경우 해당 데이터를 절대 덮어쓰지 않습니다. 동일한 위치를 다시 사용하려면 버킷의 키 접두사 위치에서 데이터를 삭제하세요.

`external_location` 구문을 생략하고 작업 그룹 설정을 사용하지 않으면 Athena는 쿼리 결과 위치에 대해 클라이언트 측 설정을 사용하고 `s3://DOC-EXAMPLE-BUCKET/<Unsaved-or-query-name>/<year>/<month/<date>/tables/<query-id>/` 위치에 테이블을 생성합니다.

분리된 파일 찾기

CTAS 또는 `INSERT INTO` 문이 실패하는 경우 분리된 데이터가 데이터 위치에 남아있을 가능성이 있습니다. 일부 경우에서 Athena는 버킷에서 데이터 또는 부분 데이터를 삭제하지 않으므로 후속 쿼리에서 이 부분 데이터를 읽을 수도 있습니다. 검사 또는 삭제할 분리된 파일을 찾으려면 Athena에서 제공하는 데이터 매니페스트 파일을 사용하여 작성할 파일 목록을 추적할 수 있습니다. 자세한 내용은 쿼리 출력 파일 식별 및 DataManifestLocation를 참조하세요.

ORDER BY 절 무시됨

CTAS 쿼리에서 Athena는 쿼리의 `SELECT` 부분에 있는 `ORDER BY` 절을 무시합니다.

SQL 사양(ISO 9075 Part 2)에 따라, `ORDER BY` 절을 바로 포함하는 쿼리 표현식에서만 쿼리 표현식에서 지정한 테이블의 행 순서가 보장됩니다. 어떤 경우든 SQL의 테이블은 내재적으로 정렬되지 않기 때문에 하위 쿼리에서 `ORDER BY`를 구현하면 쿼리가 잘못 수행되고 정렬된 출력을 얻을 수 없습니다. 따라서 Athena CTAS 쿼리에서는 데이터를 쓸 때 `ORDER BY` 절에서 지정한 순서가 유지되지 않을 수 있습니다.

쿼리 결과 저장 형식

데이터 스토리지 형식을 지정하지 않은 경우 CTAS 쿼리 결과는 기본적으로 Parquet에 저장됩니다. CTAS 결과는 `PARQUET`, `ORC`, `AVRO`, `JSON` 및 `TEXTFILE`에 저장할 수 있습니다. CTAS `TEXTFILE` 형식에는 다중 문자 구분 기호가 지원되지 않습니다. CTAS 쿼리를 실행할 때 형식 변환을 해석하기 위해 SerDe를 지정할 필요는 없습니다. Example: Writing query results to a different format 섹션을 참조하세요.

압축 형식

`GZIP` 압축은 JSON 및 TEXTFILE 형식의 CTAS 쿼리 결과에 사용됩니다. Parquet의 경우 `GZIP` 또는 `SNAPPY`를 사용할 수 있으며 기본값은 `GZIP`입니다. ORC의 경우 `LZ4`, `SNAPPY`, `ZLIB` 또는 `ZSTD`를 사용할 수 있으며 기본값은 `ZLIB`입니다. 압축을 지정하는 CTAS 예제는 Example: Specifying data storage and compression formats단원을 참조하십시오. Athena의 압축에 대한 자세한 내용은 Athena 압축 지원 섹션을 참조하세요.

파티션 및 버킷 한도

CTAS 쿼리의 결과 데이터를 분할하고 버킷화할 수 있습니다. 자세한 내용은 Athena에서 파티셔닝 및 버킷팅 단원을 참조하십시오. CTAS를 사용하여 파티셔닝된 테이블을 만들 때 Athena의 쓰기 제한은 파티션 100개입니다.

대상 테이블의 속성을 지정하는 `WITH` 절 끝에 분할 및 버킷팅 조건자를 포함합니다. 자세한 내용은 Example: Creating bucketed and partitioned tables 및 Athena에서 파티셔닝 및 버킷팅 단원을 참조하세요.

100개 파티션 제한을 해결하는 방법에 대한 자세한 내용은 CTAS 및 INSERT INTO를 사용하여 100개 파티션 한도 해결 단원을 참조하세요.

암호화(Encryption)

Amazon S3에서 다른 쿼리 결과를 암호화하는 것과 유사한 방식으로 Athena에서 CTAS 쿼리 결과를 암호화할 수 있습니다. 자세한 내용은 Amazon S3에 저장된 Athena 쿼리 결과 암호화 단원을 참조하십시오.

예상 버킷 소유자

CTAS 문의 경우 예상 버킷 소유자 설정이 Amazon S3의 대상 테이블 위치에 적용되지 않습니다. 예상 버킷 소유자 설정은 Athena 쿼리 결과에 대해 지정한 Amazon S3 출력 위치에만 적용됩니다. 자세한 내용은 Athena 콘솔을 사용하여 쿼리 결과 위치 지정 단원을 참조하십시오.

데이터 타입

CTAS 쿼리의 열 데이터 유형은 원본 쿼리에 지정된 유형과 동일합니다.

콘솔에서 CTAS 쿼리 실행

Athena 콘솔을 사용하면 다른 쿼리에서 CTAS 쿼리를 생성할 수 있습니다.

다른 쿼리에서 CTAS 쿼리를 생성하려면

1. Athena 콘솔 쿼리 편집기에서 쿼리를 실행합니다.
2. 쿼리 편집기 하단에서 생성(Create) 옵션을 선택한 다음 쿼리에서 테이블(Table from query)을 선택합니다.
3. Create table as select 양식에서 다음과 같이 필드를 작성합니다.

    a. Table name(테이블 이름)에는 새 테이블의 이름을 입력합니다. 소문자와 밑줄만 사용합니다(예: `my_select_query_parquet`).

    b. Database configuration(데이터베이스 구성)에는 기존 데이터베이스를 선택하거나 데이터베이스를 생성하는 옵션을 사용합니다.

    c. (선택 사항) Result configuration(결과 구성)의 Location of CTAS query results(CTAS 쿼리 결과 위치)에서는 작업 그룹 쿼리 결과 위치 설정에 따라 이 옵션이 재정의되지 않으면 다음 중 하나를 수행합니다.

    - 검색 상자에 기존 S3 위치의 경로를 입력하거나 Browse S3(S3 찾아보기)를 선택하여 목록에서 위치를 선택합니다.
    - View(보기)를 선택하여 기존 버킷에 대한 자세한 내용을 보고 버킷을 선택하거나 자체 설정으로 생성할 수 있는 Amazon S3 콘솔의 Buckets(버킷) 페이지를 엽니다.

데이터가 출력될 Amazon S3의 빈 위치를 지정해야 합니다. 지정한 위치에 데이터가 이미 있는 경우 쿼리가 오류와 함께 실패합니다.

작업 그룹 쿼리 결과 위치 설정에 따라 이 위치 설정이 재정의되는 경우 Athena에서는 위치 `s3://DOC-EXAMPLE-BUCKET/tables/query_id/`에 테이블을 생성합니다.

d. Data format(데이터 형식)에는 데이터의 형식을 지정합니다.

- Table type(테이블 유형) – Athena의 기본 테이블 유형은 Apache Hive입니다.
- File format(파일 형식) - CSV, TSV, JSON, Parquet 또는 ORC와 같은 옵션 중에서 선택합니다. Parquet 및 ORC 형식에 대한 자세한 내용은 열 기반 스토리지 형식 섹션을 참조하세요.
- Write compression(쓰기 압축) – (선택 사항) 압축 형식을 선택합니다. Athena는 여러 압축 형식을 사용하는 테이블에서 데이터를 읽는 것을 비롯하여, 데이터 읽기 및 쓰기를 위한 다양한 압축 형식을 지원합니다. 예를 들어 Athena는 일부 Parquet 파일이 Snappy로 압축되고 다른 Parquet 파일이 GZIP으로 압축된 경우 Parquet 파일 형식을 사용하는 테이블의 데이터를 성공적으로 읽을 수 있습니다. ORC, 텍스트 파일, JSON 스토리지 형식에도 동일한 원칙이 적용됩니다. 자세한 내용은 Athena 압축 지원 단원을 참조하십시오.
- Partitions(파티션) – (선택 사항) 파티셔닝하려는 열을 선택합니다. 데이터를 파티셔닝하면 각 쿼리에서 스캔하는 데이터의 양이 제한되어 성능이 향상되고 비용이 절감됩니다. 어떤 키를 기준으로도 데이터를 분할할 수 있습니다. 자세한 내용은 Athena에서 데이터 분할 단원을 참조하십시오.
- Buckets(버킷) – (선택 사항) 버킷팅하려는 열을 선택합니다. 버킷팅은 단일 파티션 내에서 특정 열을 기준으로 데이터를 그룹화하는 기법입니다. 이러한 열을 버킷 키라고 합니다. 관련 데이터를 단일 버킷(파티션 내 파일)으로 그룹화하면 Athena에서 스캔하는 데이터의 양이 대폭 감소하여 쿼리 성능이 향상되고 비용이 절감됩니다. 자세한 내용은 Athena에서 파티셔닝 및 버킷팅 단원을 참조하십시오.

e. Preview table query(미리 보기 테이블 쿼리)의 경우 쿼리를 검토합니다. 쿼리 구문은 CREATE TABLE AS 단원을 참조하세요.

f. 테이블 생성을 선택합니다.

템플릿을 사용하여 CTAS 쿼리를 생성하는 방법

`CREATE TABLE AS SELECT` 템플릿을 사용하여 쿼리 편집기에서 CTAS 쿼리를 생성합니다.

1. Athena 콘솔에서 테이블 및 뷰(Tables and views) 옆에 있는 테이블 생성(Create table)을 선택한 다음 CREATE TABLE AS SELECT를 선택합니다. 그러면 쿼리 편집기가 자리 표시자 값이 있는 CTAS 쿼리로 채워집니다.
2. 쿼리 편집기에서 필요에 따라 쿼리를 편집합니다. 쿼리 구문은 CREATE TABLE AS 단원을 참조하세요.
3. Run(실행)을 선택합니다.

예를 보려면 CTAS 쿼리 예제을 참조하세요.

Athena에서 파티셔닝 및 버킷팅

파티셔닝과 버킷팅은 Athena에서 쿼리를 실행할 때 스캔해야 하는 데이터의 양을 줄이는 두 가지 방법입니다. 파티셔닝 및 버킷팅은 상호 보완적이며 함께 사용할 수 있습니다. 스캔하는 데이터의 양을 줄이면 성능이 향상되고 비용이 절감됩니다. Athena 쿼리 성능에 대한 일반적인 지침은 Top 10 performance tuning tips for Amazon Athena를 참조하세요.

파티셔닝이란 무엇인가요?

파티셔닝이란 데이터의 특정 속성을 기반으로 Amazon S3의 디렉터리(또는 '접두사')로 데이터를 구성하는 작업을 의미합니다. 이러한 속성을 파티션 키라고 합니다. 일반적인 파티션 키는 날짜 또는 기타 시간 단위(예: 연도 또는 월)입니다. 하지만 데이터 세트는 둘 이상의 키로 파티셔닝할 수 있습니다. 예를 들어 제품 판매에 대한 데이터를 날짜, 제품 카테고리 및 마켓에 따라 파티셔닝할 수 있습니다.

파티셔닝 방법 결정

쿼리에 항상 또는 자주 사용되고 카디널리티가 낮은 속성이 파티션 키로 사용하기에 적합합니다. 파티셔닝이 너무 많은 경우와 너무 적은 경우 사이에는 상충 관계가 있습니다. 파티션이 너무 많으면 파일 수가 증가하여 오버헤드가 발생합니다. 파티션 자체를 필터링하는 데 약간의 오버헤드도 발생합니다. 파티션이 너무 적으면 쿼리에서 더 많은 데이터를 스캔해야 하기도 합니다.

파티셔닝된 테이블 생성

데이터 세트가 파티셔닝되면 Athena에서 파티셔닝된 테이블을 생성할 수 있습니다. 파티셔닝된 테이블은 파티션 키가 있는 테이블입니다. `CREATE TABLE`을 사용하는 경우 테이블에 파티션을 추가합니다. `CREATE TABLE AS`을 사용하는 경우 Amazon S3에 생성된 파티션이 테이블에 자동으로 추가됩니다.

`CREATE TABLE` 문에서 `PARTITIONED BY (`*`column_name data_type`*`)` 절에 파티션 키를 지정합니다. `CREATE TABLE AS`에서는 `WITH (partitioned_by = ARRAY['`*`partition_key`*`'])`

절에 파티션 키를 지정하거나 Iceberg 테이블의 경우 `WITH (partitioning = ARRAY['partition_key'])` 절에 지정합니다. 성능을 고려하여 파티션 키는 항상 `STRING` 유형이어야 합니다. 자세한 내용은 파티션 키의 데이터 형식으로 문자열 사용 단원을 참조하십시오.

추가적인 `CREATE TABLE` 및 `CREATE TABLE AS` 구문 세부 정보는 CREATE TABLE 및 CTAS 테이블 속성 섹션을 참조하세요.

파티셔닝된 테이블 쿼리

파티셔닝된 테이블을 쿼리하면 Athena는 쿼리에서 조건자를 사용하여 파티션 목록을 필터링합니다. 그런 다음 일치하는 파티션의 위치를 사용하여 찾은 파일을 처리합니다. Athena는 쿼리 조건자와 일치하지 않는 파티션의 데이터를 읽지 않음으로써 스캔하는 데이터의 양을 효율적으로 줄일 수 있습니다.

예제

`sales_date` 및 `product_category`를 기준으로 파티셔닝된 테이블이 있고 특정 카테고리에서 1주일 간의 총 수익을 알고자 합니다. Athena가 다음 예제와 같이 최소의 데이터만 스캔하도록 하기 위해 `sales_date` 및 `product_category` 열에 조건자를 포함합니다.

```
SELECT SUM(amount) AS total_revenue
FROM sales
WHERE sales_date BETWEEN '2023-02-27' AND '2023-03-05'
AND product_category = 'Toys'
```

데이터 세트가 날짜를 기준으로 파티셔닝되었지만 정밀한 타임스탬프도 있다고 가정합니다.

Iceberg 테이블을 사용하면 열과 관계를 설정하도록 파티션 키를 선언할 수 있지만, Hive 테이블을 사용하면 쿼리 엔진이 열과 파티션 키 간 관계를 인식하지 못합니다. 따라서 쿼리에서 필요한 것보다 많은 데이터를 스캔하지 않도록 하려면 쿼리에서 열과 파티션 키 모두에 조건자를 포함해야 합니다.

예를 들어 이전 예제의 `sales` 테이블에도 해당 `TIMESTAMP` 데이터 형식의 `sold_at` 열이 있다고 가정합니다. 특정 시간 범위의 매출만 원하는 경우 다음과 같이 쿼리를 작성합니다.

```
SELECT SUM(amount) AS total_revenue
FROM sales
WHERE sales_date = '2023-02-28'
AND sold_at BETWEEN TIMESTAMP '2023-02-28 10:00:00' AND TIMESTAMP '2023-02-28
 12:00:00'
AND product_category = 'Toys'
```

Hive 테이블과 Iceberg 테이블 쿼리 사이의 이러한 차이에 대한 자세한 내용은 시간을 기준으로 파티셔닝된 타임스탬프 필드에 대한 쿼리를 작성하는 방법 섹션을 참조하세요.

버킷팅이란 무엇인가요?

버킷팅은 데이터 세트의 레코드를 버킷이라는 카테고리로 구성하는 방법입니다.

버킷 및 버킷팅의 의미는 서로 다르며, Amazon S3 버킷과도 혼동해서는 안 됩니다. 데이터 버킷팅에서 속성 값이 동일한 레코드는 동일한 버킷으로 이동합니다. 레코드는 각 버킷이 대략 같은 양의 데이터를 보유하도록 버킷 사이에서 최대한 균등하게 분산됩니다.

실제로 버킷은 파일이고 해시 함수에서 레코드를 배치할 버킷을 결정합니다. 버킷팅된 데이터 세트에는 파티션 1개에 버킷당 하나 이상의 파일이 있습니다. 파일이 속한 버킷은 파일 이름에서 인코딩됩니다.

버킷팅의 장점

버킷팅은 데이터 세트가 특정 속성을 기준으로 버킷팅되고 해당 속성에 특정 값이 있는 레코드를 검색하려는 경우에 유용합니다. 데이터가 버킷팅되어 있기 때문에 Athena는 값을 사용하여 찾을 파일을 결정할 수 있습니다. 예를 들어 데이터 세트가 `customer_id`를 기준으로 버킷팅되었고 특정 고객에 대한 모든 레코드를 찾고 싶다고 가정합니다. Athena는 이러한 레코드를 포함하는 버킷을 확인하고 해당 버킷의 파일만 읽습니다.

카디널리티가 높고(즉, 개별 값이 많음) 열이 균등하게 분산되었으며 특정 값을 자주 쿼리하는 열이 있을 때 버킷팅에 적합합니다.

> **Note**
>
> Athena는 버킷팅된 테이블에 새 레코드를 추가할 때 `INSERT INTO` 사용을 지원하지 않습니다.

버킷팅된 열에 대한 필터링에 지원되는 데이터 형식

특정 데이터 형식으로 버킷팅된 열에 필터를 추가할 수 있습니다. Athena는 다음과 같은 데이터 형식의 버킷팅된 열에서 필터링을 지원합니다.

- BOOLEAN
- BYTE
- 날짜
- DOUBLE
- FLOAT

- INT
- LONG
- SHORT
- STRING
- VARCHAR

Hive 및 Spark 지원

Athena 엔진 버전 2는 Hive 버킷 알고리즘을 사용하여 버킷팅된 데이터 세트를 지원하며 Athena 엔진 버전 3은 Apache Spark 버킷팅 알고리즘도 지원합니다. Hive 버킷팅이 기본값입니다. Spark 알고리즘을 사용하여 데이터 세트를 버킷팅하는 경우 TBLPROPERTIES 절을 사용하여 `bucketing_format` 속성 값을 `spark`로 설정합니다.

> **Note**
>
> Athena는 CREATE TABLE AS SELECT(CTAS) 쿼리당 파티션을 100개로 제한합니다. 마찬가지로, INSERT INTO 문을 통해 대상 테이블에 최대 100개의 파티션만 추가할 수 있습니다. 이 100개의 한도는 테이블이 버킷팅 및 파티셔닝된 경우에만 적용됩니다.
> 이 제한을 초과하면 "HIVE_TOO_MANY_OPEN_파티션: 파티션/버킷에 대해 열린 작성자 100개를 초과했습니다."라는 오류 메시지가 표시될 수 있습니다. 이러한 제한은 CTAS 문(최대 100개의 파티션 생성) 및 일련의 INSERT INTO 문(각각 최대 100개의 파티션 삽입)을 사용하여 해결할 수 있습니다. 자세한 내용은 CTAS 및 INSERT INTO를 사용하여 100개 파티션 한도 해결 단원을 참조하십시오.

CREATE TABLE 버킷팅 예제

기존의 버킷팅된 데이터 세트에 대한 테이블을 생성하려면 CLUSTERED BY (*column*) 절 및 INTO *N* BUCKETS절을 차례로 사용합니다. INTO *N* BUCKETS 절은 데이터가 버킷팅되는 버킷 수를 지정합니다.

다음 CREATE TABLE 예제에서는 Spark 알고리즘을 사용하여 `customer_id`를 기준으로 `sales` 데이터 세트를 8개의 버킷으로 버킷팅합니다. CREATE TABLE 문에서는 CLUSTERED BY 및 TBLPROPERTIES 절을 사용하여 속성을 적절하게 설정합니다.

```
CREATE EXTERNAL TABLE sales (...)
...
```

```
CLUSTERED BY (`customer_id`) INTO 8 BUCKETS
...
TBLPROPERTIES (
  'bucketing_format' = 'spark'
)
```

CREATE TABLE AS(CTAS) 버킷팅 예제

CREATE TABLE AS를 사용하여 버킷팅을 지정하려면 다음 예제와 같이 bucketed_by 및 bucket_count 파라미터를 사용합니다.

```
CREATE TABLE sales
WITH (
  ...
  bucketed_by = ARRAY['customer_id'],
  bucket_count = 8
)
AS SELECT ...
```

버킷팅 쿼리 예제

다음 쿼리 예제는 특정 고객이 1주일 동안 구매한 제품의 이름을 찾습니다.

```
SELECT DISTINCT product_name
FROM sales
WHERE sales_date BETWEEN '2023-02-27' AND '2023-03-05'
AND customer_id = 'c123'
```

이 테이블이 sales_date를 기준으로 파티셔닝되고 customer_id를 기준으로 버키팅된 경우 Athena는 고객 레코드가 들어 있는 버킷을 계산할 수 있습니다. Athena는 파티션당 최대 하나의 파일 정도만 읽습니다.

추가적인 리소스

버킷팅 및 파티셔닝된 테이블 모두를 생성하는 CREATE TABLE AS 예제는 예제: 버킷팅 및 파티셔닝된 테이블 생성을 참조하세요.

CTAS 쿼리 예제

다음 예제를 사용하여 CTAS 쿼리를 생성합니다. CTAS 구문에 대한 자세한 정보는 CREATE TABLE AS 단원을 참조하세요.

이 섹션:

- Example: Duplicating a table by selecting all columns
- Example: Selecting specific columns from one or more tables
- Example: Creating an empty copy of an existing table
- Example: Specifying data storage and compression formats
- Example: Writing query results to a different format
- Example: Creating unpartitioned tables
- Example: Creating partitioned tables
- Example: Creating bucketed and partitioned tables
- Example: Creating an Iceberg table with Parquet data
- Example: Creating an Iceberg table with Avro data

Example 예제: 모든 열을 선택하여 테이블 복제

다음 예제에서는 테이블의 모든 열을 복사해 테이블을 만듭니다.

```
CREATE TABLE new_table AS
SELECT *
FROM old_table;
```

동일한 예제의 다음 변형에서 SELECT 문에는 WHERE 절이 포함되어 있습니다. 이 경우, 쿼리는 테이블에서 WHERE 절을 충족하는 행만 선택합니다.

```
CREATE TABLE new_table AS
SELECT *
FROM old_table
WHERE condition;
```

Example 예제: 하나 이상의 테이블에서 특정 열 선택

다음 예제에서는 다른 테이블의 열 세트에 대해 실행할 새 쿼리를 생성합니다.

```
CREATE TABLE new_table AS
SELECT column_1, column_2, ... column_n
```

```
FROM old_table;
```

동일한 예제의 이번 변형에서는 여러 테이블의 특정 열을 바탕으로 새 테이블을 생성합니다.

```
CREATE TABLE new_table AS
SELECT column_1, column_2, ... column_n
FROM old_table_1, old_table_2, ... old_table_n;
```

Example 예제: 기존 테이블의 빈 복사본 생성

다음 예제에서는 `WITH NO DATA`를 사용해 비어 있지만 원본 테이블과 스키마가 동일한 새 테이블을 생성합니다.

```
CREATE TABLE new_table
AS SELECT *
FROM old_table
WITH NO DATA;
```

Example 예제: 데이터 스토리지 및 압축 형식 지정

CTAS를 사용하면 한 스토리지 형식의 소스 테이블을 사용하여 스토리지 형식이 다른 또 다른 테이블을 만들 수 있습니다.

`format` 속성을 사용하여 `ORC`, `PARQUET`, `AVRO`, `JSON` 또는 `TEXTFILE`을 새 테이블의 스토리지 형식으로 지정합니다.

`PARQUET`, `ORC`, `TEXTFILE` 및 `JSON` 스토리지 형식의 경우 `write_compression` 속성을 사용하여 새 테이블의 데이터에 대한 압축 형식을 지정합니다. 각 파일 형식에서 지원하는 압축 유형에 대한 자세한 내용은 Athena 압축 지원 섹션을 참조하세요.

다음 예제에서는 `new_table` 테이블의 데이터를 Parquet 형식으로 Snappy 압축을 사용하여 저장하도록 지정합니다. Parquet의 기본 압축은 `GZIP`입니다.

```
CREATE TABLE new_table
WITH (
      format = 'Parquet',
      write_compression = 'SNAPPY')
AS SELECT *
FROM old_table;
```

다음 예제에서는 new_table 테이블의 데이터를 ORC 형식으로 Snappy 압축을 사용하여 저장하도록 지정합니다. ORC의 기본 압축은 ZLIB입니다.

```
CREATE TABLE new_table
WITH (format = 'ORC',
      write_compression = 'SNAPPY')
AS SELECT *
FROM old_table ;
```

다음 예제에서는 new_table 테이블의 데이터를 텍스트 파일 형식으로 Snappy 압축을 사용하여 저장하도록 지정합니다. 텍스트 파일과 JSON 형식 모두의 기본 압축은 GZIP입니다.

```
CREATE TABLE new_table
WITH (format = 'TEXTFILE',
      write_compression = 'SNAPPY')
AS SELECT *
FROM old_table ;
```

Example 예제: 쿼리 결과를 다른 형식으로 쓰기

다음 CTAS 쿼리는 CSV 또는 다른 형식으로 저장할 수 있는 모든 레코드를 old_table에서 선택하고 ORC 형식으로 Amazon S3에 저장된 기본 데이터로 새 테이블을 만듭니다.

```
CREATE TABLE my_orc_ctas_table
WITH (
      external_location = 's3://DOC-EXAMPLE-BUCKET/my_orc_stas_table/',
      format = 'ORC')
AS SELECT *
FROM old_table;
```

Example 예제: 분할되지 않은 테이블 생성

다음 예제에서는 분할되지 않은 테이블을 생성합니다. 테이블 데이터가 다른 형식으로 저장됩니다. 이러한 예제 중 일부에서는 외부 위치를 지정합니다.

다음 예제에서는 결과를 텍스트 파일로 저장하는 CTAS 쿼리를 생성합니다.

```
CREATE TABLE ctas_csv_unpartitioned
WITH (
     format = 'TEXTFILE',
     external_location = 's3://DOC-EXAMPLE-BUCKET/ctas_csv_unpartitioned/')
```

```
AS SELECT key1, name1, address1, comment1
FROM table1;
```

다음 예제에서는 결과가 Parquet에 저장되고 기본 결과 위치가 사용됩니다.

```
CREATE TABLE ctas_parquet_unpartitioned
WITH (format = 'PARQUET')
AS SELECT key1, name1, comment1
FROM table1;
```

다음 쿼리에서는 테이블이 JSON으로 저장되고 원본 테이블 결과에서 특정 열을 선택합니다.

```
CREATE TABLE ctas_json_unpartitioned
WITH (
     format = 'JSON',
     external_location = 's3://DOC-EXAMPLE-BUCKET/ctas_json_unpartitioned/')
AS SELECT key1, name1, address1, comment1
FROM table1;
```

다음 예제에서 형식은 ORC입니다.

```
CREATE TABLE ctas_orc_unpartitioned
WITH (
     format = 'ORC')
AS SELECT key1, name1, comment1
FROM table1;
```

다음 예제에서 형식은 Avro입니다.

```
CREATE TABLE ctas_avro_unpartitioned
WITH (
     format = 'AVRO',
     external_location = 's3://DOC-EXAMPLE-BUCKET/ctas_avro_unpartitioned/')
AS SELECT key1, name1, comment1
FROM table1;
```

Example 예제: 분할된 테이블 생성

다음 예제에서는 `partitioned_by` 및 `WITH` 절의 기타 속성을 사용하여 다른 스토리지 형식으로 분할된 테이블에 대한 `CREATE TABLE AS SELECT` 쿼리를 보여줍니다. 구문은 CTAS 테이블 속성 단원

을 참조하세요. 분할할 열 선택에 대한 자세한 정보는 Athena에서 파티셔닝 및 버킷팅 단원을 참조하세요.

> **Note**
>
> SELECT 문의 열 목록 끝에 파티션 열을 나열합니다. 둘 이상의 열을 기준으로 분할할 수 있으며 최대 100개의 고유한 파티션과 버킷 조합을 가질 수 있습니다. 예를 들어 버킷이 지정되지 않은 경우 100개의 파티션을 가질 수 있습니다.

```
CREATE TABLE ctas_csv_partitioned
WITH (
     format = 'TEXTFILE',
     external_location = 's3://DOC-EXAMPLE-BUCKET/ctas_csv_partitioned/',
     partitioned_by = ARRAY['key1'])
AS SELECT name1, address1, comment1, key1
FROM tables1;
```

```
CREATE TABLE ctas_json_partitioned
WITH (
     format = 'JSON',
     external_location = 's3://DOC-EXAMPLE-BUCKET/ctas_json_partitioned/',
     partitioned_by = ARRAY['key1'])
AS select name1, address1, comment1, key1
FROM table1;
```

Example 예제: 버킷팅 및 분할된 테이블 생성

다음 예제는 Amazon S3에 쿼리 결과를 저장하기 위해 분할 및 버킷팅을 둘 다 사용하는 CREATE TABLE AS SELECT 쿼리를 보여줍니다. 테이블 결과가 다른 열을 기준으로 분할 및 버킷팅됩니다. Athena는 최대 100개의 고유한 파티션 및 버킷 조합을 지원합니다. 예를 들어 5개의 버킷이 있는 테이블을 생성하는 경우 각각 5개의 버킷이 있는 20개의 파티션이 지원됩니다. 구문은 CTAS 테이블 속성 단원을 참조하세요.

버킷팅할 열 선택에 대한 자세한 정보는 Athena에서 파티셔닝 및 버킷팅 단원을 참조하세요.

```
CREATE TABLE ctas_avro_bucketed
WITH (
      format = 'AVRO',
```

```
      external_location = 's3://DOC-EXAMPLE-BUCKET/ctas_avro_bucketed/',
      partitioned_by = ARRAY['nationkey'],
      bucketed_by = ARRAY['mktsegment'],
      bucket_count = 3)
AS SELECT key1, name1, address1, phone1, acctbal, mktsegment, comment1, nationkey
FROM table1;
```

Example 예: Parquet 데이터를 사용하여 Iceberg 테이블 생성

다음 예에서는 Parquet 데이터 파일을 사용하여 Iceberg 테이블을 생성합니다. 파일은 `table1`의 `dt` 열을 사용하여 월별로 분할됩니다. 이 예에서는 테이블의 모든 브랜치에 기본적으로 10개의 스냅샷이 보존되도록 테이블의 보존 속성을 업데이트합니다. 지난 7일 이내의 스냅샷도 유지됩니다. Athena에서 Iceberg 테이블 속성에 대한 자세한 내용은 테이블 속성 섹션을 참조하세요.

```
CREATE TABLE ctas_iceberg_parquet
WITH (table_type = 'ICEBERG',
      format = 'PARQUET',
      location = 's3://DOC-EXAMPLE-BUCKET/ctas_iceberg_parquet/',
      is_external = false,
      partitioning = ARRAY['month(dt)'],
      vacuum_min_snapshots_to_keep = 10,
      vacuum_max_snapshot_age_seconds = 604800
   )
AS SELECT key1, name1, dt FROM table1;
```

Example 예: Avro 데이터를 사용하여 Iceberg 테이블 생성

다음 예에서는 `key1`에 의해 분할된 Avro 데이터 파일을 사용하여 Iceberg 테이블을 생성합니다.

```
CREATE TABLE ctas_iceberg_avro
WITH ( format = 'AVRO',
       location = 's3://DOC-EXAMPLE-BUCKET/ctas_iceberg_avro/',
       is_external = false,
       table_type = 'ICEBERG',
       partitioning = ARRAY['key1'])
AS SELECT key1, name1, date FROM table1;
```

ETL 및 데이터 분석에 CTAS 및 INSERT INTO 사용

Athena에서 Create Table as Select(CTAS) 및 INSERT INTO 문을 사용하면 데이터 처리를 위해 Amazon S3에 데이터를 추출, 변환 및 로드(ETL)할 수 있습니다. 이 주제에서는 이러한 문을 사용하여

데이터 세트를 분할하고 열 기반 데이터 형식으로 변환함으로써 데이터 분석에 최적화되도록 하는 방법을 보여줍니다.

CTAS 문은 표준 SELECT 쿼리를 사용하여 새 테이블을 생성합니다. CTAS 문을 사용하여 분석할 데이터의 하위 세트를 만들 수 있습니다. 하나의 CTAS 문에서, 데이터를 분할하고 압축을 지정하고 데이터를 Apache Parquet 또는 Apache ORC와 같은 열 기반 형식으로 변환할 수 있습니다. CTAS 쿼리를 실행하면 생성한 테이블 및 파티션은 AWS Glue Data Catalog에 자동으로 추가됩니다. 이렇게 하면 생성한 새 테이블 및 파티션을 후속 쿼리에 즉시 사용할 수 있습니다.

INSERT INTO 문은 원본 테이블에서 실행되는 SELECT 쿼리 문을 기반으로 대상 테이블에 새 행을 삽입합니다. INSERT INTO 문을 사용하면 CTAS가 지원하는 모든 변환을 사용하여 CSV 형식의 원본 테이블 데이터를 대상 테이블 데이터로 변환하고 로드할 수 있습니다.

개요

Athena에서 CTAS 문을 사용하여 데이터의 초기 배치 변환을 수행합니다. 그런 다음, 여러 INSERT INTO 문을 사용하여 CTAS 문에 의해 생성된 테이블에 증분 업데이트를 만듭니다.

단계

- 1단계: 원본 데이터 집합을 기반으로 테이블 생성
- 2단계: CTAS를 사용하여 데이터 분할, 변환 및 압축
- 3단계: INSERT INTO를 사용하여 데이터 추가
- 4단계: 성능 및 비용 차이 측정

1단계: 원본 데이터 집합을 기반으로 테이블 생성

이 주제의 예제에서는 공개적으로 이용 가능한 NOAA Global Historical Climatology Network Daily(GHCN-D) 데이터 집합의 Amazon S3 읽기 가능 하위 집합을 사용합니다. Amazon S3의 데이터에는 다음과 같은 특성이 있습니다.

```
Location: s3://aws-bigdata-blog/artifacts/athena-ctas-insert-into-blog/
Total objects: 41727
Size of CSV dataset: 11.3 GB
Region: us-east-1
```

원본 데이터는 파티션 없이 Amazon S3에 저장됩니다. 데이터는 다음과 유사한 파일에서 CSV 형식입니다.

```
2019-10-31 13:06:57  413.1 KiB artifacts/athena-ctas-insert-into-blog/2010.csv0000
2019-10-31 13:06:57  412.0 KiB artifacts/athena-ctas-insert-into-blog/2010.csv0001
2019-10-31 13:06:57   34.4 KiB artifacts/athena-ctas-insert-into-blog/2010.csv0002
2019-10-31 13:06:57  412.2 KiB artifacts/athena-ctas-insert-into-blog/2010.csv0100
2019-10-31 13:06:57  412.7 KiB artifacts/athena-ctas-insert-into-blog/2010.csv0101
```

이 샘플의 파일 크기는 상대적으로 작습니다. 파일을 더 큰 파일로 병합하면 총 파일 수를 줄일 수 있으므로 쿼리 성능이 향상됩니다. CTAS 및 INSERT INTO 문을 사용하여 쿼리 성능을 개선할 수 있습니다.

샘플 데이터 세트를 기반으로 데이터베이스 및 테이블을 생성하려면

1. Athena 콘솔에서 미국 동부(버지니아 북부)(US East (N. Virginia)) AWS 리전을 선택합니다. `us-east-1`에서 이 자습서의 모든 쿼리를 실행합니다.
2. Athena 쿼리 편집기에서 CREATE DATABASE 명령을 실행하여 데이터베이스를 생성합니다.

   ```
   CREATE DATABASE blogdb
   ```

3. 다음 문을 실행하여 테이블을 생성합니다.

   ```
   CREATE EXTERNAL TABLE `blogdb`.`original_csv` (
     `id` string,
     `date` string,
     `element` string,
     `datavalue` bigint,
     `mflag` string,
     `qflag` string,
     `sflag` string,
     `obstime` bigint)
   ROW FORMAT DELIMITED
     FIELDS TERMINATED BY ','
   STORED AS INPUTFORMAT
     'org.apache.hadoop.mapred.TextInputFormat'
   OUTPUTFORMAT
     'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
   LOCATION
     's3://aws-bigdata-blog/artifacts/athena-ctas-insert-into-blog/'
   ```

2단계: CTAS를 사용하여 데이터 분할, 변환 및 압축

테이블을 생성한 후에는 단일 CTAS 문을 사용하여 데이터를 Snappy 압축을 사용하는 Parquet 형식으로 변환하고 연도별로 데이터를 분할할 수 있습니다.

1단계에서 생성한 테이블에는 날짜 형식이 YYYYMMDD(예: 20100104)인 date 필드가 있습니다. 새 테이블이 year에 분할되므로 다음 절차의 샘플 문은 Presto 함수 substr("date",1,4)을 사용하여 date 필드에서 year 값을 추출합니다.

연도별로 분할하기 위해 데이터를 Snappy 압축을 사용하는 Parquet 형식으로 변환하려면

- 다음 CTAS 문을 실행하여 ##을 Amazon S3 버킷 위치로 바꿉니다.

```
CREATE table new_parquet
WITH (format='PARQUET',
parquet_compression='SNAPPY',
partitioned_by=array['year'],
external_location = 's3://DOC-EXAMPLE-BUCKET/optimized-data/')
AS
SELECT id,
         date,
         element,
         datavalue,
         mflag,
         qflag,
         sflag,
         obstime,
         substr("date",1,4) AS year
FROM original_csv
WHERE cast(substr("date",1,4) AS bigint) >= 2015
        AND cast(substr("date",1,4) AS bigint) <= 2019
```

> **Note**
>
> 이 예제에서 생성한 테이블에는 2015년부터 2019년까지의 데이터만 포함됩니다. 3단계에서는 INSERT INTO 명령을 사용하여 이 테이블에 새 데이터를 추가합니다.

쿼리가 완료되면 다음 절차를 사용하여 CTAS 문에서 지정한 Amazon S3 위치의 출력을 확인합니다.

CTAS 문에 의해 생성된 파티션 및 Parquet 파일을 확인하려면

1. 생성된 파티션을 표시하려면 다음 AWS CLI 명령을 실행합니다. 끝에 슬래시(/)가 있어야 합니다.

```
aws s3 ls s3://DOC-EXAMPLE-BUCKET/optimized-data/
```

출력이 파티션에 표시됩니다.

```
      PRE year=2015/
      PRE year=2016/
      PRE year=2017/
      PRE year=2018/
      PRE year=2019/
```

2. Parquet 파일을 확인하려면 다음 명령을 실행합니다. 출력을 처음 5개의 결과로 제한하는 | head -5 옵션은 Windows에서 사용할 수 없습니다.

```
aws s3 ls s3://DOC-EXAMPLE-BUCKET/optimized-data/ --recursive --human-readable |
 head -5
```

다음과 유사하게 출력됩니다.

```
2019-10-31 14:51:05    7.3 MiB optimized-data/
year=2015/20191031_215021_00001_3f42d_1be48df2-3154-438b-b61d-8fb23809679d
2019-10-31 14:51:05    7.0 MiB optimized-data/
year=2015/20191031_215021_00001_3f42d_2a57f4e2-ffa0-4be3-9c3f-28b16d86ed5a
2019-10-31 14:51:05    9.9 MiB optimized-data/
year=2015/20191031_215021_00001_3f42d_34381db1-00ca-4092-bd65-ab04e06dc799
2019-10-31 14:51:05    7.5 MiB optimized-data/
year=2015/20191031_215021_00001_3f42d_354a2bc1-345f-4996-9073-096cb863308d
2019-10-31 14:51:05    6.9 MiB optimized-data/
year=2015/20191031_215021_00001_3f42d_42da4cfd-6e21-40a1-8152-0b902da385a1
```

3단계: INSERT INTO를 사용하여 데이터 추가

2단계에서는 CTAS를 사용하여 2015년부터 2019년까지 파티션이 있는 테이블을 생성했습니다. 하지만 원본 데이터 세트에는 2010년부터 2014년까지의 데이터도 포함되어 있습니다. 이제 INSERT INTO 문을 사용하여 해당 데이터를 추가합니다.

하나 이상의 INSERT INTO 문을 사용하여 테이블에 데이터를 추가하려면

1. WHERE 절에 2015년 이전의 연도를 지정하여 다음 INSERT INTO 명령을 실행합니다.

```
INSERT INTO new_parquet
SELECT id,
         date,
         element,
         datavalue,
         mflag,
         qflag,
         sflag,
         obstime,
         substr("date",1,4) AS year
FROM original_csv
WHERE cast(substr("date",1,4) AS bigint) < 2015
```

2. 다음 구문을 사용하여 `aws s3 ls` 명령을 다시 실행합니다.

```
aws s3 ls s3://DOC-EXAMPLE-BUCKET/optimized-data/
```

출력이 새 파티션에 표시됩니다.

```
      PRE year=2010/
      PRE year=2011/
      PRE year=2012/
      PRE year=2013/
      PRE year=2014/
      PRE year=2015/
      PRE year=2016/
      PRE year=2017/
      PRE year=2018/
      PRE year=2019/
```

3. Parquet 형식에서 압축 및 열 기반 스토리지를 사용하여 얻은 데이터 세트의 크기 감소를 확인하려면 다음 명령을 실행합니다.

```
aws s3 ls s3://DOC-EXAMPLE-BUCKET/optimized-data/ --recursive --human-readable --
summarize
```

다음 결과는 Snappy 압축을 사용하는 Parquet 형식으로 처리한 후 데이터 세트의 크기가 1.2GB 임을 보여줍니다.

```
...
2020-01-22 18:12:02 2.8 MiB optimized-data/
year=2019/20200122_181132_00003_nja5r_f0182e6c-38f4-4245-afa2-9f5bfa8d6d8f
2020-01-22 18:11:59 3.7 MiB optimized-data/
year=2019/20200122_181132_00003_nja5r_fd9906b7-06cf-4055-a05b-f050e139946e
Total Objects: 300
     Total Size: 1.2 GiB
```

4. 더 많은 CSV 데이터가 원본 테이블에 추가되면 INSERT INTO 문을 사용하여 해당 데이터를 Parquet 테이블에 추가할 수 있습니다. 예를 들어 2020년에 대한 새 데이터가 있는 경우 다음 INSERT INTO 문을 실행할 수 있습니다. INSERT INTO 문은 데이터 및 관련 파티션을 `new_parquet` 테이블에 추가합니다.

```
INSERT INTO new_parquet
SELECT id,
         date,
         element,
         datavalue,
         mflag,
         qflag,
         sflag,
         obstime,
         substr("date",1,4) AS year
FROM original_csv
WHERE cast(substr("date",1,4) AS bigint) = 2020
```

> **Note**
>
> INSERT INTO 문은 대상 테이블에 최대 100개의 파티션 쓰기를 지원합니다. 또한 100개 이상의 파티션을 추가하려면 여러 INSERT INTO 문을 실행합니다. 자세한 내용은 CTAS 및 INSERT INTO를 사용하여 100개 파티션 한도 해결 단원을 참조하세요.

4단계: 성능 및 비용 차이 측정

데이터를 변환한 후에는 새 테이블 및 이전 테이블에서 동일한 쿼리를 실행하고 결과를 비교하여 성능 향상 및 비용 절감을 측정할 수 있습니다.

> **Note**
>
> Athena 쿼리별 비용에 대한 자세한 내용은 Amazon Athena 요금을 참조하세요.

성능 향상 및 비용 차이를 측정하려면

1. 원본 테이블에서 다음 쿼리를 실행합니다. 쿼리는 연도의 모든 값에 대해 고유 ID 수를 찾습니다.

```
SELECT substr("date",1,4) as year,
       COUNT(DISTINCT id)
FROM original_csv
GROUP BY 1 ORDER BY 1 DESC
```

2. 쿼리가 실행된 시간과 스캔 데이터의 양을 적어둡니다.
3. 새 테이블에서 동일한 쿼리를 실행하고 쿼리 런타임 및 스캔 데이터의 양을 적어둡니다.

```
SELECT year,
  COUNT(DISTINCT id)
FROM new_parquet
GROUP BY 1 ORDER BY 1 DESC
```

4. 결과를 비교하고 성능 및 비용 차이를 계산합니다. 다음 샘플 결과는 새 테이블의 테스트 쿼리가 이전 테이블의 쿼리보다 빠르고 저렴하다는 것을 보여줍니다.

표	런타임	스캔 데이터
원본	16.88초	11.35GB
신규	3.79초	428.05MB

5. 원본 테이블에서 다음 샘플 쿼리를 실행합니다. 이 쿼리에서는 2018년 지구의 평균 최고 기온(섭씨), 평균 최저 기온(섭씨) 및 평균 강우량(mm)을 계산합니다.

```
SELECT element, round(avg(CAST(datavalue AS real)/10),2) AS value
```

```
FROM original_csv
WHERE element IN ('TMIN', 'TMAX', 'PRCP') AND substr("date",1,4) = '2018'
GROUP BY 1
```

6. 쿼리가 실행된 시간과 스캔 데이터의 양을 적어둡니다.
7. 새 테이블에서 동일한 쿼리를 실행하고 쿼리 런타임 및 스캔 데이터의 양을 적어둡니다.

```
SELECT element, round(avg(CAST(datavalue AS real)/10),2) AS value
FROM new_parquet
WHERE element IN ('TMIN', 'TMAX', 'PRCP') and year = '2018'
GROUP BY 1
```

8. 결과를 비교하고 성능 및 비용 차이를 계산합니다. 다음 샘플 결과는 새 테이블의 테스트 쿼리가 이전 테이블의 쿼리보다 빠르고 저렴하다는 것을 보여줍니다.

표	런타임	스캔 데이터
원본	18.65초	11.35GB
신규	1.92초	68MB

요약

이 주제에서는 Athena에 CTAS 및 INSERT INTO 문을 사용하여 ETL 작업을 수행하는 방법을 알아보았습니다. CTAS 문을 사용하여 데이터를 Snappy 압축을 사용하는 Parquet 형식으로 변환하는 첫 번째 변환 세트를 수행했습니다. 또한 CTAS 문을 사용하여 분할되지 않은 데이터 세트에서 분할된 데이터 세트로 변환했습니다. 이를 통해 데이터 세트의 크기를 줄이고 쿼리 실행 비용을 낮출 수 있었습니다. 새 데이터를 사용할 수 있게 되면 CTAS 문을 통해 생성한 테이블에 데이터를 변환하고 로드하는 데 INSERT INTO 문을 사용할 수 있습니다.

CTAS 및 INSERT INTO를 사용하여 100개 파티션 한도 해결

Athena는 `CREATE TABLE AS SELECT`(CTAS) 쿼리당 파티션을 100개로 제한합니다. 마찬가지로, INSERT INTO 문을 통해 대상 테이블에 최대 100개의 파티션을 추가할 수 있습니다.

이 제한을 초과하면 “HIVE_TOO_MANY_OPEN_파티션: 파티션/버킷에 대해 열린 작성자 100개를 초과했습니다.”라는 오류 메시지가 표시될 수 있습니다. 이러한 제한은 CTAS 문(최대 100개의 파티션 생성) 및 일련의 `INSERT INTO` 문(각각 최대 100개의 파티션 삽입)을 사용하여 해결할 수 있습니다.

이 주제의 예제에서는 Amazon S3 버킷 위치(s3://DOC-EXAMPLE-BUCKET/)에 데이터가 상주하는 `tpch100`이라는 데이터베이스를 사용합니다.

CTAS 및 INSERT INTO를 사용하여 100개 이상의 파티션이 있는 테이블을 생성하려면

1. `CREATE EXTERNAL TABLE` 문을 사용하여 원하는 필드에 분할된 테이블을 생성합니다.

   다음 예제 문은 `l_shipdate` 열을 기준으로 데이터를 분할합니다. 테이블에는 2525개의 파티션이 있습니다.

```
CREATE EXTERNAL TABLE `tpch100.lineitem_parq_partitioned`(
  `l_orderkey` int,
  `l_partkey` int,
  `l_suppkey` int,
  `l_linenumber` int,
  `l_quantity` double,
  `l_extendedprice` double,
  `l_discount` double,
  `l_tax` double,
  `l_returnflag` string,
  `l_linestatus` string,
  `l_commitdate` string,
  `l_receiptdate` string,
  `l_shipinstruct` string,
  `l_comment` string)
PARTITIONED BY (
  `l_shipdate` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe' STORED AS
 INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat' OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat' LOCATION
 's3://DOC-EXAMPLE-BUCKET/lineitem/'
```

2. 다음과 유사한 `SHOW PARTITIONS <table_name>` 명령을 실행하여 파티션을 나열합니다.

```
SHOW PARTITIONS lineitem_parq_partitioned
```

   다음은 샘플 결과의 일부입니다.

```
/*
l_shipdate=1992-01-02
```

```
l_shipdate=1992-01-03
l_shipdate=1992-01-04
l_shipdate=1992-01-05
l_shipdate=1992-01-06

...

l_shipdate=1998-11-24
l_shipdate=1998-11-25
l_shipdate=1998-11-26
l_shipdate=1998-11-27
l_shipdate=1998-11-28
l_shipdate=1998-11-29
l_shipdate=1998-11-30
l_shipdate=1998-12-01
*/
```

3. CTAS 쿼리를 실행하여 분할된 테이블을 생성합니다.

   다음 예제에서는 my_lineitem_parq_partitioned라는 테이블을 만들고 WHERE 절을 사용하여 DATE를 1992-02-01 이전으로 제한합니다. 샘플 데이터 세트는 1992년 1월로 시작하므로 1992년 1월에 대한 파티션만 생성됩니다.

```
CREATE table my_lineitem_parq_partitioned
WITH (partitioned_by = ARRAY['l_shipdate']) AS
SELECT l_orderkey,
         l_partkey,
         l_suppkey,
         l_linenumber,
         l_quantity,
         l_extendedprice,
         l_discount,
         l_tax,
         l_returnflag,
         l_linestatus,
         l_commitdate,
         l_receiptdate,
         l_shipinstruct,
         l_comment,
         l_shipdate
FROM tpch100.lineitem_parq_partitioned
WHERE cast(l_shipdate as timestamp) < DATE ('1992-02-01');
```

4. SHOW PARTITIONS 명령을 실행하여 원하는 파티션이 테이블에 포함되어 있는지 확인합니다.

```
SHOW PARTITIONS my_lineitem_parq_partitioned;
```

이 예제의 파티션은 1992년 1월부터입니다.

```
/*
l_shipdate=1992-01-02
l_shipdate=1992-01-03
l_shipdate=1992-01-04
l_shipdate=1992-01-05
l_shipdate=1992-01-06
l_shipdate=1992-01-07
l_shipdate=1992-01-08
l_shipdate=1992-01-09
l_shipdate=1992-01-10
l_shipdate=1992-01-11
l_shipdate=1992-01-12
l_shipdate=1992-01-13
l_shipdate=1992-01-14
l_shipdate=1992-01-15
l_shipdate=1992-01-16
l_shipdate=1992-01-17
l_shipdate=1992-01-18
l_shipdate=1992-01-19
l_shipdate=1992-01-20
l_shipdate=1992-01-21
l_shipdate=1992-01-22
l_shipdate=1992-01-23
l_shipdate=1992-01-24
l_shipdate=1992-01-25
l_shipdate=1992-01-26
l_shipdate=1992-01-27
l_shipdate=1992-01-28
l_shipdate=1992-01-29
l_shipdate=1992-01-30
l_shipdate=1992-01-31
*/
```

5. INSERT INTO 문을 사용하여 테이블에 파티션을 추가합니다.

다음 예제에서는 1992년 2월의 날짜에 대한 파티션을 추가합니다.

```
INSERT INTO my_lineitem_parq_partitioned
SELECT l_orderkey,
         l_partkey,
         l_suppkey,
         l_linenumber,
         l_quantity,
         l_extendedprice,
         l_discount,
         l_tax,
         l_returnflag,
         l_linestatus,
         l_commitdate,
         l_receiptdate,
         l_shipinstruct,
         l_comment,
         l_shipdate
FROM tpch100.lineitem_parq_partitioned
WHERE cast(l_shipdate as timestamp) >= DATE ('1992-02-01')
AND cast(l_shipdate as timestamp) < DATE ('1992-03-01');
```

6. SHOW PARTITIONS를 다시 실행합니다.

```
SHOW PARTITIONS my_lineitem_parq_partitioned;
```

이제 샘플 테이블에는 1992년 1월 및 2월의 파티션이 있습니다.

```
/*
l_shipdate=1992-01-02
l_shipdate=1992-01-03
l_shipdate=1992-01-04
l_shipdate=1992-01-05
l_shipdate=1992-01-06

...

l_shipdate=1992-02-20
l_shipdate=1992-02-21
l_shipdate=1992-02-22
l_shipdate=1992-02-23
l_shipdate=1992-02-24
l_shipdate=1992-02-25
l_shipdate=1992-02-26
```

```
l_shipdate=1992-02-27
l_shipdate=1992-02-28
l_shipdate=1992-02-29
*/
```

7. 각 파티션에 100개 이하의 파티션을 읽고 추가하는 `INSERT INTO` 문을 계속 사용합니다. 필요한 파티션 수에 도달할 때까지 계속합니다.

> **Important**
>
> `WHERE` 조건을 설정할 때 쿼리가 중첩되지 않도록 해야 합니다. 그렇지 않으면 일부 파티션에 중복된 데이터가 있을 수 있습니다.

SerDe 참조

Athena는 CSV, JSON, Parquet 및 ORC 등 다양한 데이터 형식의 데이터를 구문 분석하기 위한 다양한 SerDe 라이브러리를 지원합니다. Athena는 사용자 지정 SerDes를 지원하지 않습니다.

주제

- SerDe 사용
- 지원되는 SerDes 및 데이터 형식

SerDe 사용

SerDe(Serializer/Deserializer)는 Athena가 다양한 형식의 데이터와 상호 작용하는 한 방식입니다.

테이블 스키마는 DDL이 아니라 개발자가 지정한 SerDe에 의해 정의됩니다. 즉, SerDe는 Athena에서 테이블을 생성할 때 지정한 DDL 구성을 재정의할 수 있습니다.

쿼리에 SerDe를 사용하려면

Athena에서 테이블을 생성할 때 SerDe를 사용하려면 다음 방법 중 하나를 사용합니다.

- 다음 예제와 같이 `ROW FORMAT DELIMITED`를 지정한 다음 DDL 문을 사용하여 필드 구분 기호를 지정합니다. `ROW FORMAT DELIMITED`를 지정할 때 Athena는 기본적으로 LazySimpleSerDe를 사용합니다.

```
ROW FORMAT DELIMITED
```

```
FIELDS TERMINATED BY ','
ESCAPED BY '\\'
COLLECTION ITEMS TERMINATED BY '|'
MAP KEYS TERMINATED BY ':'
```

`ROW FORMAT DELIMITED`의 예는 다음 주제를 참조하세요.

CSV, TSV, 사용자 지정 구분 기호로 구분된 파일에 대한 LazySimpleSerDe

Amazon CloudFront 로그 쿼리

Amazon EMR 로그 쿼리

Amazon VPC 흐름 로그 쿼리

ETL 및 데이터 분석에 CTAS 및 INSERT INTO 사용

- `ROW FORMAT SERDE`를 사용해 Athena가 데이터를 테이블에 읽고 쓸 때 사용해야 하는 SerDe의 형식을 명시적으로 지정합니다. 다음 예제에서는 LazySimpleSerDe를 지정합니다. 구분 기호를 지정하려면 `WITH SERDEPROPERTIES`를 사용합니다. `WITH SERDEPROPERTIES`에서 지정한 속성은 `ROW FORMAT DELIMITED` 예제에서 별도의 문(예: `FIELDS TERMINATED BY`)에 해당합니다.

```
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
'serialization.format' = ',',
'field.delim' = ',',
'collection.delim' = '|',
'mapkey.delim' = ':',
'escape.delim' = '\\'
)
```

`ROW FORMAT SERDE`의 예는 다음 주제를 참조하세요.

Avro SerDe

Grok SerDe

JSON SerDe 라이브러리

CSV 처리를 위한 OpenCSVSerDe

Regex SerDe

지원되는 SerDes 및 데이터 형식

Athena는 CSV, TSV, 사용자 지정 구분 기호 및 JSON 형식, Hadoop 관련 형식의 데이터(ORC, Apache Avro, Parquet), Logstash의 로그, AWS CloudTrail 로그, Apache WebServer 로그에서 데이터를 쿼리하고 테이블을 만들도록 지원합니다.

> **Note**
>
> 본 단원에 나열된 형식은 Athena에서 데이터를 읽는 데 사용합니다. CTAS 쿼리를 실행할 때 Athena에서 데이터를 쓰는 데 사용하는 형식에 대한 자세한 정보는 쿼리 결과에서 테이블 생성(CTAS) 단원을 참조하세요.

Athena에서 이러한 형식의 데이터를 쿼리하고 테이블을 만들려면 Athena가 사용되는 형식과 데이터 구문 분석 방법을 알 수 있도록 serializer-deserializer 클래스(SerDe)를 지정합니다.

이 테이블은 Athena 및 해당 SerDe 라이브러리에 지원되는 데이터 형식을 나열합니다.

SerDe는 Athena가 사용하는 데이터 카탈로그에 데이터 취급 방법을 알려주는 사용자 지정 라이브러리입니다. Athena에서 `CREATE TABLE` 문의 `ROW FORMAT` 파트에 명시적으로 SerDe 유형을 나열하여 지정합니다. Athena는 기본적으로 특정 유형의 데이터 형식에 일부 SerDe 유형을 사용하기 때문에, 경우에 따라 SerDe 이름을 생략할 수 있습니다.

지원되는 데이터 형식 및 SerDes

데이터 형식	설명	Athena에서 지원되는 SerDe 유형
Amazon Ion	Amazon Ion은 JSON의 상위 집합인 서식 있는 자기 기술형 데이터 형식이며, Amazon에서 개발한 오픈 소스입니다.	Amazon Ion Hive SerDe을(를) 사용합니다.
Apache Avro	레코드 값에 대해 JSON 기반 스키마를 사용하는 Hadoop에서 데이터를 저장하는 형식입니다.	Avro SerDe를 사용합니다.

데이터 형식	설명	Athena에서 지원되는 SerDe 유형
Apache Parquet	Hadoop에서 데이터의 컬럼 방식 스토리지를 위한 형식입니다.	Parquet SerDe 및 SNAPPY 압축을 사용합니다.
Apache WebServer 로그	Apache WebServer에서 로그를 저장하는 형식입니다.	Grok SerDe 또는 Regex SerDe를 사용합니다.
CloudTrail 로그	CloudTrail에서 로그를 저장하는 형식입니다.	• Hive JSON SerDe를 사용합니다. 자세한 내용은 AWS CloudTrail 로그 쿼리 단원을 참조하십시오.
CSV(쉼표로 분리된 값)	CSV 형식의 데이터에서 각 행은 데이터 레코드를 나타내며 각 레코드는 쉼표로 구분된 하나 이상의 필드로 구성됩니다.	• 데이터에 따옴표로 묶인 값이 포함되지 않거나 `java.sql.Timestamp` 형식을 사용하는 경우 CSV, TSV, 사용자 지정 구분 기호로 구분된 파일에 대한 LazySimpleSerDe를 사용합니다. • 데이터의 값에 따옴표가 있거나 `TIMESTAMP` 에 UNIX 숫자 형식을 사용하는 경우 (예: `1564610311`) CSV 처리를 위한 OpenCSVSerDe를 사용합니다.
사용자 지정 구분 기호로 구분	이 형식의 데이터에서 각 행은 데이터 레코드를 나타내며 레코드는 사용자 지정 단일 문자 구분 기호로 구분됩니다.	CSV, TSV, 사용자 지정 구분 기호로 구분된 파일에 대한 LazySimpleSerDe를 사용하고 사용자 지정 단일 문자 구분 기호를 지정합니다.

데이터 형식	설명	Athena에서 지원되는 SerDe 유형
[JSON](JavaScript Object Notation)	JSON 데이터에서 각 행은 데이터 레코드를 나타내며 각 레코드는 쉼표로 구분된 속성-값 쌍과 배열로 구성됩니다.	• Hive JSON SerDe를 사용합니다. • OpenX JSON SerDe를 사용합니다.
Logstash 로그	Logstash에서 로그를 저장하는 형식입니다.	Grok SerDe를 사용합니다.
ORC(Optimized Row Columnar)	Hive 데이터의 최적화된 컬럼 방식 스토리지를 위한 형식입니다.	ORC SerDe 및 ZLIB 압축을 사용합니다.
TSV(탭으로 구분된 값)	TSV 형식의 데이터에서 각 행은 데이터 레코드를 나타내며 각 레코드는 탭으로 구분된 하나 이상의 필드로 구성됩니다.	CSV, TSV, 사용자 지정 구분 기호로 구분된 파일에 대한 LazySimpleSerDe를 사용하고 구분자 문자를 `FIELDS TERMINATED BY '\t'` 로 지정합니다.

주제

- Amazon Ion Hive SerDe
- Avro SerDe
- Grok SerDe
- JSON SerDe 라이브러리
- CSV, TSV, 사용자 지정 구분 기호로 구분된 파일에 대한 LazySimpleSerDe
- CSV 처리를 위한 OpenCSVSerDe
- ORC SerDe
- Parquet SerDe
- Regex SerDe

Amazon Ion Hive SerDe

Amazon Ion Hive SerDe를 사용하여 Amazon Ion 형식으로 저장된 데이터를 쿼리할 수 있습니다. Amazon Ion은 서식 있는 자기 기술형 오픈 소스 데이터 형식입니다. Amazon Ion 형식은 Amazon Quantum Ledger Database(Amazon QLDB)와 같은 서비스 및 오픈 소스 SQL 쿼리 언어PartiQL에서 사용됩니다.

Amazon Ion은 상호 교환 가능한 이진 및 텍스트 형식을 갖습니다. 이 기능은 텍스트의 사용 편의성과 이진 인코딩의 효율성을 결합합니다.

Athena에서 Amazon Ion 데이터를 쿼리하려면 Amazon Ion 데이터를 직렬화 및 역직렬화하는 Amazon Ion Hive SerDe를 사용할 수 있습니다. 역직렬화를 사용하면 Parquet 또는 ORC와 같은 다른 형식으로 기록하기 위해 Amazon Ion 데이터에 쿼리를 실행하거나 읽을 수 있습니다. 직렬화를 사용하면 `CREATE TABLE AS SELECT`(CTAS) 또는 기존 테이블에서 데이터를 복사하는 `INSERT INTO` 쿼리를 사용하여 Amazon Ion 형식으로 데이터를 생성할 수 있습니다.

> **Note**
>
> Amazon Ion은 JSON의 상위 집합이므로 Amazon Ion Hive SerDe를 사용하여 Amazon Ion JSON이 아닌 데이터 집합을 쿼리할 수 있습니다. 다른 JSON SerDe 라이브러리와 달리 Amazon Ion SerDe는 각 데이터 행이 한 줄에 있을 것으로 예상하지 않습니다. 이 기능은 "예쁜 인쇄" 형식의 JSON 데이터 집합을 쿼리하려 하거나 줄 바꿈 문자로 필드를 행으로 분할하려는 경우에 유용합니다.

Athena에서 Amazon Ion을 쿼리하는 방법에 대한 추가 정보 및 예제는 Analyze Amazon Ion datasets using Amazon Athena를 참조하세요.

SerDe 이름

- com.amazon.ionhiveserde.IonHiveSerDe

고려 사항 및 제한

- 중복된 필드 - Amazon Ion 구조체는 정렬되어 있고 중복된 필드를 지원하는 반면 Hive의 `STRUCT<>` 및 `MAP<>`은 그렇지 않습니다. 따라서 Amazon Ion 구조체에서 중복된 필드를 역직렬화하면 하나의 값이 비결정론적으로 선택되고 다른 값은 무시됩니다.
- 외부 기호 테이블 지원되지 않음 - 현재 Athena는 외부 기호 테이블 또는 다음 Amazon Ion Hive SerDe 속성을 지원하지 않습니다.

  - `ion.catalog.class`
  - `ion.catalog.file`
  - `ion.catalog.url`
  - `ion.symbol_table_imports`
- 파일 확장자 - Amazon Ion은 파일 확장자를 사용하여 Amazon Ion 파일을 역직렬화하는 데 사용할 압축 코덱을 결정합니다. 따라서 압축된 파일은 사용된 압축 알고리즘에 해당하는 파일 확장자를 가져야 합니다. 예를 들어 ZSTD를 사용하는 경우 해당 파일은 `.zst` 확장자를 가져야 합니다.
- 동종 데이터 - Amazon Ion은 특정 필드의 값에 사용할 수 있는 데이터 형식에 제약을 두지 않습니다. 예를 들어 두 개의 서로 다른 Amazon Ion 문서에 서로 다른 데이터 형식을 갖는 이름이 같은 필드가 있을 수 있습니다. 그러나 Hive는 스키마를 사용하기 때문에 하나의 Hive 열에 추출한 모든 값은 동일한 데이터 형식을 가져야 합니다.
- 맵 키 형식 제한 - 다른 형식의 데이터를 Amazon Ion으로 직렬화할 때 맵 키 형식이 `STRING`, `VARCHAR` 또는`CHAR` 중 하나인지 확인합니다. Hive를 사용하면 어떤 기본 데이터 형식이라도 맵 키로 사용할 수 있지만 Amazon Ion 기호는 문자열 형식이어야 합니다.
- 공용 구조체 형식 - Athena는 현재 Hive Union 형식을 지원하지 않습니다.
- Double 데이터 유형 - Amazon Ion은 현재 `double` 데이터 유형을 지원하지 않습니다.

주제

- CREATE TABLE을 사용하여 Amazon Ion 테이블 생성
- CTAS 및 INSERT INTO를 사용하여 Amazon Ion 테이블 생성
- Amazon Ion SerDe 속성 사용
- 경로 추출기 사용

CREATE TABLE을 사용하여 Amazon Ion 테이블 생성

Athena에서 Amazon Ion 형식으로 저장된 데이터로부터 테이블을 생성하려면 CREATE TABLE 문에서 다음 기법 중 하나를 사용할 수 있습니다.

- `STORED AS ION`를 지정합니다. 이 사용법에서 명시적으로 Amazon Ion Hive SerDe를 지정할 필요는 없습니다. 이 선택 사항은 더 간단한 옵션입니다.
- `ROW FORMAT SERDE`, `INPUTFORMAT`, `OUTPUTFORMAT` 필드에서 Amazon Ion 클래스 경로를 지정합니다.

또한 CREATE TABLE AS SELECT(CTAS) 문을 사용하여 Athena에서 Amazon Ion 테이블을 생성할 수도 있습니다. 자세한 설명은 CTAS 및 INSERT INTO를 사용하여 Amazon Ion 테이블 생성을 참조하세요.

STORED AS ION 지정

다음 CREATE TABLE 문 예제는 LOCATION 절 이전에 STORED AS ION을 사용하여 Amazon Ion 형식의 비행 데이터를 기반으로 테이블을 생성합니다. LOCATION 절은 Ion 형식의 입력 파일이 있는 버킷 또는 폴더를 지정합니다. 지정된 위치의 모든 파일이 스캔됩니다.

```
CREATE EXTERNAL TABLE flights_ion (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
)
STORED AS ION
LOCATION 's3://DOC-EXAMPLE-BUCKET/'
```

Amazon Ion 클래스 경로 지정

STORED AS ION 구문을 사용하는 대신 다음과 같이 ROW FORMAT SERDE, INPUTFORMAT, OUTPUTFORMAT 절에 대한 Ion 클래스 경로 값을 명시적으로 지정할 수 있습니다.

파라미터	Ion 클래스 경로
ROW FORMAT SERDE	'com.amazon.ionhiveserde.IonHiveSerDe'
STORED AS INPUTFORMAT	'com.amazon.ionhiveserde.formats.IonInputFormat'
OUTPUTFORMAT	'com.amazon.ionhiveserde.formats.IonOutputFormat'

다음 DDL 쿼리는 이 기법을 사용하여 이전 예제와 동일한 외부 테이블을 만듭니다.

```
CREATE EXTERNAL TABLE flights_ion (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
)
ROW FORMAT SERDE
 'com.amazon.ionhiveserde.IonHiveSerDe'
STORED AS INPUTFORMAT
 'com.amazon.ionhiveserde.formats.IonInputFormat'
OUTPUTFORMAT
 'com.amazon.ionhiveserde.formats.IonOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/'
```

Athena의 CREATE TABLE 문에 대한 SerDe 속성에 관한 자세한 내용은 Amazon Ion SerDe 속성 사용을 참조하세요.

CTAS 및 INSERT INTO를 사용하여 Amazon Ion 테이블 생성

CREATE TABLE AS SELECT(CTAS) 및 INSERT INTO 문을 사용하여 Athena에서 테이블의 데이터를 Amazon Ion 형식으로 복사하거나 삽입할 수 있습니다.

CTAS 쿼리에서 다음 예제와 같이 WITH 절에서 format='ION'을 지정합니다.

```
CREATE TABLE new_table
WITH (format='ION')
AS SELECT * from existing_table
```

기본적으로 Athena는 Amazon Ion 결과를 Ion 이진 형식으로 직렬화하지만 텍스트 형식을 사용할 수도 있습니다. 텍스트 형식을 사용하려면 다음 예제와 같이 CTAS WITH 절에서 ion_encoding = 'TEXT'를 지정합니다.

```
CREATE TABLE new_table
WITH (format='ION', ion_encoding = 'TEXT')
AS SELECT * from existing_table
```

CTAS WITH 절에서 Amazon Ion 지정 속성에 대한 자세한 내용은 다음 섹션을 참조하세요.

CTAS WITH 절 Amazon Ion 속성

CTAS 쿼리에서 `WITH` 절을 사용하여 Amazon Ion 형식을 지정하고 사용할 Amazon Ion 인코딩 및/또는 쓰기 압축 알고리즘을 선택적으로 지정할 수 있습니다.

형식

`ION` 키워드를 CTAS 쿼리의 `WITH` 절에서 형식 옵션으로 지정할 수 있습니다. 이렇게 하는 경우 생성하는 테이블은 읽기에 `IonInputFormat`에 지정한 형식을 사용하고 `IonOutputFormat`에 지정한 형식으로 데이터를 직렬화합니다.

다음 예제에서는 CTAS 쿼리가 Amazon Ion 형식을 사용하도록 지정합니다.

```
WITH (format='ION')
```

ion_encoding

선택 사항

기본값: `BINARY`

값: `BINARY`, `TEXT`

데이터를 Amazon Ion 이진 형식으로 직렬화할지 아니면 Amazon Ion 텍스트 형식으로 직렬화할지 지정합니다. 다음 예제는 Amazon Ion 텍스트 형식을 지정합니다.

```
WITH (format='ION', ion_encoding='TEXT')
```

write_compression

선택 사항

기본값: `GZIP`

값: `GZIP`, `ZSTD`, `BZIP2`, `SNAPPY`, `NONE`

출력 파일을 압축하는 데 사용할 압축 알고리즘을 지정합니다.

다음 예제에서는 CTAS 쿼리가 Zstandard 압축 알고리즘을 사용하여 Amazon Ion 형식으로 출력을 쓰도록 지정합니다.

```
WITH (format='ION', write_compression = 'ZSTD')
```

Athena의 데이터 압축에 대한 자세한 내용은 Athena 압축 지원을 참조하세요.

Athena의 다른 CTAS 속성에 대한 자세한 내용은 CTAS 테이블 속성을 참조하세요.

Amazon Ion SerDe 속성 사용

이 주제는 Athena에서 `CREATE TABLE` 문에 대한 SerDe 속성에 대한 정보가 포함되어 있습니다. Amazon Ion SerDe 속성 사용에 대한 자세한 내용 및 예제는 GitHub에서 Amazon Ion Hive SerDe 설명서의 SerDe 속성을 참조하세요.

Amazon Ion SerDe 속성 지정

`CREATE TABLE` 문에서 Amazon Ion Hive SerDe에 대한 속성을 지정하려면 `WITH SERDEPROPERTIES` 절을 사용합니다. `WITH SERDEPROPERTIES`가 `ROW FORMAT SERDE` 절의 하위 필드이므로 다음 구문과 같이 `ROW FORMAT SERDE` 및 Amazon Ion Hive SerDe 클래스 경로를 먼저 지정해야 합니다.

```
...
ROW FORMAT SERDE
 'com.amazon.ionhiveserde.IonHiveSerDe'
WITH SERDEPROPERTIES (
 'property' = 'value',
 'property' = 'value',
...
)
```

`WITH SERDEPROPERTIES`를 사용하려면 `ROW FORMAT SERDE` 절이 필요하지만 `STORED AS ION` 또는 더 긴 `INPUTFORMAT` 및 `OUTPUTFORMAT` 구문을 사용하여 Amazon Ion 형식을 지정할 수 있다는 점에 유의하세요.

Amazon Ion SerDe 속성

다음은 Athena에서 `CREATE TABLE` 문에 사용할 수 있는 Amazon Ion SerDe 속성입니다.

ion.encoding

선택 사항

기본값: `BINARY`

값: `BINARY`, `TEXT`

이 속성은 추가된 새 값이 Amazon Ion 이진 또는 Amazon Ion 텍스트 형식으로 직렬화되는지 여부를 선언합니다.

다음 SerDe 속성 예제는 Amazon Ion 텍스트 형식을 지정합니다.

```
'ion.encoding' = 'TEXT'
```

ion.fail_on_overflow

선택 사항

기본값: `true`

값: `true`, `false`

Amazon Ion은 임의의 큰 숫자 형식을 허용하지만 Hive는 허용하지 않습니다. 기본적으로 Amazon Ion 값이 Hive 열에 맞지 않으면 SerDE가 실패하지만 `fail_on_overflow` 구성 옵션을 사용하여 실패 대신 값이 오버플로되도록 할 수 있습니다.

이 속성은 테이블 또는 열 수준에서 설정될 수 있습니다. 테이블 수준에서 이를 지정하려면 다음 예제와 같이 `ion.fail_on_overflow`를 지정합니다. 이렇게 하면 모든 열의 기본 동작이 설정됩니다.

```
'ion.fail_on_overflow' = 'true'
```

특정 열을 제어하려면 다음 예와 같이 마침표로 구분하여 `ion`과 `fail_on_overflow` 사이에 열 이름을 지정합니다.

```
'ion.<column>.fail_on_overflow' = 'false'
```

ion.path_extractor.case_sensitive

선택 사항

기본값: `false`

값: `true`, `false`

Amazon Ion 필드 이름을 대/소문자를 구분하여 처리할지 여부를 결정합니다. `false`이면 SerDe는 Amazon Ion 필드 이름을 구문 분석할 때 대/소문자를 무시합니다.

예를 들어 다음 예제와 같이 소문자로 `alias` 필드를 정의하는 Hive 테이블 스키마와 `alias` 필드 및 `ALIAS` 필드가 있는 Amazon Ion 문서가 있다고 가정합니다.

```
-- Hive Table Schema
alias: STRING

-- Amazon Ion Document
{ 'ALIAS': 'value1'}
{ 'alias': 'value2'}
```

다음 예제에서는 대/소문자 구분이 `false`로 설정된 경우 SerDe 속성 및 추출된 결과 테이블을 보여 줍니다.

```
-- Serde properties
'ion.alias.path_extractor' = '(alias)'
'ion.path_extractor.case_sensitive' = 'false'

--Extracted Table
| alias    |
|----------|
| "value1" |
| "value2" |
```

다음 예제에서는 대/소문자 구분이 `true`로 설정된 경우 SerDe 속성 및 추출된 결과 테이블을 보여 줍니다.

```
-- Serde properties
'ion.alias.path_extractor' = '(alias)'
'ion.path_extractor.case_sensitive' = 'true'

--Extracted Table
| alias    |
|----------|
| "value2" |
```

두 번째 경우 `ALIAS` 필드의 `value1`은 대/소문자 구분을 `true`로 설정한 경우 무시되며 경로 추출기는 `alias`로 지정됩니다.

ion.*<column>*.path_extractor

선택 사항

기본값: NA

값: 검색 경로가 있는 문자열

주어진 열에 대한 지정된 검색 경로로 경로 추출기를 생성합니다. 경로 추출기는 Amazon Ion 필드를 Hive 열에 매핑합니다. 경로 추출기를 지정하지 않으면 Athena는 열 이름을 기반으로 런타임에 동적으로 경로 추출기를 만듭니다.

다음 예제 경로 추출기는 `example_ion_field`를 `example_hive_column`에 매핑합니다.

```
'ion.example_hive_column.path_extractor' = '(example_ion_field)'
```

경로 추출기 및 검색 경로에 대한 자세한 내용은 경로 추출기 사용을 참조하세요.

ion.timestamp.serialization_offset

선택 사항

기본값: `'Z'`

값: `OFFSET`, `OFFSET` 는 *<signal>*`hh:mm`과 같이 표시됩니다. 예제 값: `01:00`, `+01:00`, `-09:30`, `Z`(UTC, 00:00과 동일)

기본 제공 시간대가 없고 UNIX Epoch에서 오프셋으로 저장되는 Apache Hive 타임스탬프와는 달리 Amazon Ion 타임스탬프는 오프셋이 있습니다. 이 속성을 사용하여 Amazon Ion으로 직렬화할 때 오프셋을 지정합니다.

다음 예제에서는 오프셋을 1시간 추가합니다.

```
'ion.timestamp.serialization_offset' = '+01:00'
```

ion.serialize_null

선택 사항

기본값: `OMIT`

값: `OMIT`, `UNTYPED`, `TYPED`

Amazon Ion SerDE는 값이 null인 열을 직렬화하거나 생략하도록 구성할 수 있습니다. 강력한 형식의 null(TYPED) 또는 형식이 지정되지 않은 null(UNTYPED)을 쓰도록 선택할 수 있습니다. 강력한 형식의 null은 기본 Amazon Ion-Hive 형식 매핑에 따라 결정됩니다.

다음 예제에서는 강력한 형식의 null을 지정합니다.

```
'ion.serialize_null'='TYPED'
```

ion.ignore_malformed

선택 사항

기본값: `false`

값: `true`, `false`

`true`이면 잘못된 형식의 항목 또는 SerDE에서 읽을 수 없는 경우 전체 파일을 무시합니다. 자세한 내용은 GitHub에서 설명서의 Ignore malformed(잘못된 형식 무시)를 참조하세요.

ion.*<column>*.serialize_as

선택 사항

기본값: 열의 기본 형식.

값: Amazon Ion 형식을 포함하는 문자열

값이 직렬화되는 Amazon Ion 데이터 형식을 결정합니다. Amazon Ion 및 Hive 형식이 항상 직접 매핑되는 것은 아니기 때문에 몇몇 Hive 형식에는 직렬화를 위한 여러 유효한 데이터 형식이 있습니다. 데이터를 기본이 아닌 데이터 형식으로 직렬화하려면 이 속성을 사용합니다. 형식 매핑에 대한 자세한 내용은 GitHub의 Amazon Ion Type mapping(형식 매핑) 페이지를 참조하세요.

기본적으로 이진 Hive 열은 Amazon Ion blob으로 직렬화되지만 Amazon Ion clob(문자 대용량 객체)으로 직렬화될 수도 있습니다. 다음 예제에서는 `example_hive_binary_column` 열을 clob으로 직렬화합니다.

```
'ion.example_hive_binary_column.serialize_as' = 'clob'
```

경로 추출기 사용

Amazon Ion은 문서 스타일 파일 형식이지만 Apache Hive는 평면 열 기반 형식입니다. `path extractors`라는 특별한 Amazon Ion SerDe 속성을 사용하여 두 형식 간을 매핑할 수 있습니다. 경로 추출기는 계층적 Amazon Ion 형식을 평면화하고 Amazon Ion 값을 Hive 열에 매핑하며 필드 이름을 바꾸는 데 사용할 수 있습니다.

Athena는 추출기를 생성할 수 있지만 필요한 경우 사용자 고유의 추출기를 정의할 수도 있습니다.

생성된 경로 추출기

기본적으로 Athena는 Hive 열 이름과 일치하는 최상위 Amazon Ion 값을 검색하고 런타임에 이러한 일치하는 값을 기반으로 경로 추출기를 생성합니다. Amazon Ion 데이터 형식이 Hive 테이블 스키마와 일

치하는 경우 Athena는 동적으로 추출기를 생성하므로 경로 추출기를 추가로 추가할 필요가 없습니다. 이러한 기본 경로 추출기는 테이블 메타데이터에 저장되지 않습니다.

다음 예제에서는 Athena가 열 이름을 기반으로 추출기를 생성하는 방법을 보여줍니다.

```
-- Example Amazon Ion Document
{
    identification: {
        name: "John Smith",
        driver_license: "XXXX"
    },

    alias: "Johnny"
}

-- Example DDL
CREATE EXTERNAL TABLE example_schema2 (
    identification MAP<STRING, STRING>,
    alias STRING
)
STORED AS ION
LOCATION 's3://DOC-EXAMPLE-BUCKET/path_extraction1/'
```

다음 예제 추출기는 Athena에 의해 생성됩니다. 첫 번째는 `identification` 필드를 `identification` 열으로 추출하고 두 번째는 `alias` 필드를 `alias` 열으로 추출합니다.

```
'ion.identification.path_extractor' = '(identification)'
'ion.alias.path_extractor' = '(alias)'
```

다음 예제에서는 추출된 테이블을 보여줍니다.

```
|                  identification                    |  alias   |
|----------------------------------------------------|----------|
|{["name", "driver_license"],["John Smith", "XXXX"]} | "Johnny" |
```

사용자 고유의 경로 추출기 지정

Amazon Ion 필드가 Hive 열에 깔끔하게 매핑되지 않는 경우 자체 경로 추출기를 지정할 수 있습니다. `CREATE TABLE` 문의 `WITH SERDEPROPERTIES` 절에서 다음 구문을 사용합니다.

```
WITH SERDEPROPERTIES (
   "ion.path_extractor.case_sensitive" = "<Boolean>",
```

```
   "ion.<column_name>.path_extractor" = "<path_extractor_expression>"
)
```

> **Note**
>
> 기본적으로 경로 추출기는 대/소문자를 구분하지 않습니다. 이 설정을 재정의하려면 ion.path_extractor.case_sensitive SerDe 속성을 `true`로 설정합니다.

경로 추출기에서 검색 경로 사용

경로 추출기의 SerDe 속성 구문은 *<path_extractor_expression>*을 포함합니다.

```
"ion.<column_name>.path_extractor" = "<path_extractor_expression>"
```

*<path_extractor_expression>*을 사용하여 Amazon Ion 문서를 구문 분석하고 일치하는 데이터를 찾는 검색 경로를 지정할 수 있습니다. 검색 경로는 괄호로 묶여 있으며 다음 구성 요소 중 하나 이상을 공백으로 구분하여 포함할 수 있습니다.

- 와일드 카드 - 모든 값과 일치합니다.
- 인덱스 - 지정된 숫자 인덱스의 값과 일치합니다. 인덱스는 0부터 시작됩니다.
- 텍스트 - 필드 이름이 지정된 텍스트와 동일한 모든 값과 일치합니다.
- 주석 - 지정된 주석을 갖는 래핑된 경로 구성 요소에 의해 지정된 값과 일치합니다.

다음 예제에서는 Amazon Ion 문서 및 일부 검색 경로 예시를 보여 줍니다.

```
-- Amazon Ion document
{
    foo: ["foo1", "foo2"] ,
    bar: "myBarValue",
    bar: A::"annotatedValue"
}

-- Example search paths
(foo 0)       # matches "foo1"
(1)           # matches "myBarValue"
(*)           # matches ["foo1", "foo2"], "myBarValue" and A::"annotatedValue"
()            # matches {foo: ["foo1", "foo2"] , bar: "myBarValue", bar:
 A::"annotatedValue"}
```

```
(bar)         # matches "myBarValue" and A::"annotatedValue"
(A::bar)      # matches A::"annotatedValue"
```

추출기 예제

필드 평면화 및 이름 바꾸기

다음 예제에서는 필드를 평면화하고 이름을 바꾸는 검색 경로 집합을 보여 줍니다. 이 예제에서는 검색 경로를 사용하여 다음을 수행합니다.

- nickname 열을 alias 필드로 매핑
- name 열을 identification 구조체에 위치한 name 하위 필드로 매핑.

다음은 Amazon Ion 문서의 예입니다.

```
-- Example Amazon Ion Document
{
    identification: {
        name: "John Smith",
        driver_license: "XXXX"
    },

    alias: "Johnny"
}
```

다음은 경로 추출기를 정의하는 CREATE TABLE 문의 예입니다.

```
-- Example DDL Query
CREATE EXTERNAL TABLE example_schema2 (
    name STRING,
    nickname STRING
)
ROW FORMAT SERDE
 'com.amazon.ionhiveserde.IonHiveSerDe'
WITH SERDEPROPERTIES (
 'ion.nickname.path_extractor' = '(alias)',
 'ion.name.path_extractor' = '(identification name)'
 )
STORED AS ION
LOCATION 's3://DOC-EXAMPLE-BUCKET/path_extraction2/'
```

다음 예제에서는 추출된 데이터를 보여줍니다.

```
-- Extracted Table
| name         |   nickname   |
|--------------|--------------|
| "John Smith" |  "Johnny"    |
```

검색 경로 및 추가 검색 경로 예제에 대한 자세한 내용은 GitHub의 Ion Java Path Extraction(Ion Java 경로 추출) 페이지를 참조하세요.

비행 데이터를 텍스트 형식으로 추출

다음 CREATE TABLE 쿼리 예제는 WITH SERDEPROPERTIES를 사용하여 비행 데이터를 추출하기 위한 경로 추출기를 추가하고 출력 인코딩을 Amazon Ion 텍스트로 지정합니다. 이 예제에서는 STORED AS ION 구문을 사용합니다.

```
CREATE EXTERNAL TABLE flights_ion (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
)
ROW FORMAT SERDE
 'com.amazon.ionhiveserde.IonHiveSerDe'
WITH SERDEPROPERTIES (
 'ion.encoding' = 'TEXT',
 'ion.yr.path_extractor'='(year)',
 'ion.quarter.path_extractor'='(results quarter)',
 'ion.month.path_extractor'='(date month)')
STORED AS ION
LOCATION 's3://DOC-EXAMPLE-BUCKET/'
```

Avro SerDe

SerDe 이름

Avro SerDe

라이브러리 이름

org.apache.hadoop.hive.serde2.avro.AvroSerDe

예제

보안상의 이유로 Athena에서는 `avro.schema.url`을 이용해 테이블 스키마를 지정할 수 없습니다. `avro.schema.literal`를 사용합니다. Avro 형식의 데이터에서 스키마를 추출하려면 `getschema` 파라미터와 함께 Apache `avro-tools-<version>.jar`을(를) 사용할 수 있습니다. 그러면 `WITH SERDEPROPERTIES` 문 내에서 사용할 수 있는 스키마가 반환됩니다. 다음 예를 참조하세요.

```
java -jar avro-tools-1.8.2.jar getschema my_data.avro
```

`avro-tools-<version>.jar` 파일은 설치된 Avro 버전의 `java` 하위 디렉터리에 있습니다. Avro를 다운로드하려면 Apache Avro releases(Apache Avro 릴리스)를 참조하세요. Apache Avro Tools를 직접 다운로드하려면 Apache Avro tools Maven repository(Apache Avro Tools Maven 리포지토리)를 참조하세요.

스키마를 얻은 후 `CREATE TABLE` 문을 이용해 Amazon S3에 저장된 기본 Avro 데이터를 기반으로 Athena 테이블을 생성합니다. Avro SerDe를 지정하려면 `ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'`를 사용합니다. 다음 예제에 설명된 대로 테이블에 대한 열 이름 및 해당 데이터 유형을 지정하는 것 외에도 `WITH SERDEPROPERTIES` 절을 사용하여 스키마를 지정해야 합니다.

> **Note**
>
> `s3://athena-examples-myregion/path/to/data/`의 *myregion*을, Athena를 실행하는 리전 식별자로 바꿉니다(예: `s3://athena-examples-us-west-1/path/to/data/`).

```
CREATE EXTERNAL TABLE flights_avro_example (
   yr INT,
   flightdate STRING,
   uniquecarrier STRING,
   airlineid INT,
   carrier STRING,
   flightnum STRING,
   origin STRING,
   dest STRING,
   depdelay INT,
   carrierdelay INT,
```

```
   weatherdelay INT
)
PARTITIONED BY (year STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
WITH SERDEPROPERTIES ('avro.schema.literal'='
{
   "type" : "record",
   "name" : "flights_avro_subset",
   "namespace" : "default",
   "fields" : [ {
      "name" : "yr",
      "type" : [ "null", "int" ],
      "default" : null
   }, {
      "name" : "flightdate",
      "type" : [ "null", "string" ],
      "default" : null
   }, {
      "name" : "uniquecarrier",
      "type" : [ "null", "string" ],
      "default" : null
   }, {
      "name" : "airlineid",
      "type" : [ "null", "int" ],
      "default" : null
   }, {
      "name" : "carrier",
      "type" : [ "null", "string" ],
      "default" : null
   }, {
      "name" : "flightnum",
      "type" : [ "null", "string" ],
      "default" : null
   }, {
      "name" : "origin",
      "type" : [ "null", "string" ],
      "default" : null
   }, {
      "name" : "dest",
      "type" : [ "null", "string" ],
      "default" : null
   }, {
      "name" : "depdelay",
      "type" : [ "null", "int" ],
```

```
      "default" : null
   }, {
      "name" : "carrierdelay",
      "type" : [ "null", "int" ],
      "default" : null
   }, {
      "name" : "weatherdelay",
      "type" : [ "null", "int" ],
      "default" : null
    } ]
}
')
STORED AS AVRO
LOCATION 's3://athena-examples-myregion/flight/avro/';
```

테이블에서 `MSCK REPAIR TABLE` 문을 실행해 파티션 메타데이터를 새로 고칩니다.

```
MSCK REPAIR TABLE flights_avro_example;
```

총 출발 횟수로 상위 10개 출발 도시를 쿼리합니다.

```
SELECT origin, count(*) AS total_departures
FROM flights_avro_example
WHERE year >= '2000'
GROUP BY origin
ORDER BY total_departures DESC
LIMIT 10;
```

> **Note**
>
> 비행 테이블 데이터의 출처는 미국 운수부 운송통계국에서 제공한 항공편입니다. 원본에서 채도를 낮췄습니다.

Grok SerDe

Logstash Grok SerDe는 비정형 텍스트 데이터(일반적으로 로그)의 역직렬화를 위한 특수 패턴 집합이 있는 라이브러리입니다. 각 Grok 패턴은 명명된 정규식입니다. 필요에 따라 이러한 역직렬화 패턴을 식별하고 재사용할 수 있습니다. 그러면 정규식을 사용하는 경우에 비해 더 쉽게 Grok을 사용할 수 있습니다. Grok은 사전 정의된 패턴 집합을 제공합니다. 사용자 지정 패턴을 만들 수도 있습니다.

Athena에서 테이블을 생성할 때 Grok SerDe를 지정하려면 `ROW FORMAT SERDE 'com.amazonaws.glue.serde.GrokSerDe'` 절 뒤에 데이터에서 일치시킬 패턴을 지정하는 `WITH SERDEPROPERTIES` 절을 사용합니다. 여기서,

- `input.format` 표현식은 데이터에서 일치시킬 패턴을 정의합니다. 이 값은 필수입니다.
- `input.grokCustomPatterns` 표현식은 명명된 사용자 지정 패턴을 정의합니다. 이 패턴은 나중에 `input.format` 표현식에서 사용할 수 있습니다. 이는 선택 사항입니다. 여러 패턴 항목을 `input.grokCustomPatterns` 표현식에 포함하려면, 줄 바꿈 이스케이프 문자(\n)를 이용해 `'input.grokCustomPatterns'='INSIDE_QS ([^\"]*)\nINSIDE_BRACKETS ([^\]]*)'`) 같은 형식으로 구분하세요.
- `STORED AS INPUTFORMAT` 및 `OUTPUTFORMAT` 절이 필요합니다.
- `LOCATION` 절은 여러 데이터 객체를 포함할 수 있는 Amazon S3 버킷을 지정합니다. 버킷의 모든 데이터 객체가 역직렬화되어 테이블을 생성합니다.

예제

이 예제는 사전 정의된 Grok 패턴 목록에 의존합니다. 사전 정의된 패턴을 참조하세요.

예 1

이 예제는 `s3://DOC-EXAMPLE-BUCKET/groksample/`에 저장된 Postfix maillog 항목의 소스 데이터를 사용합니다.

```
Feb  9 07:15:00 m4eastmail postfix/smtpd[19305]: B88C4120838: connect from
 unknown[192.168.55.4]
Feb  9 07:15:00 m4eastmail postfix/smtpd[20444]: B58C4330038:
 client=unknown[192.168.55.4]
Feb  9 07:15:03 m4eastmail postfix/cleanup[22835]: BDC22A77854: message-
id=<31221401257553.5004389LCBF@m4eastmail.example.com>
```

다음 명령문은 Athena에서 사용자 지정 패턴과 직접 지정한 사전 정의된 패턴을 사용하여 소스 데이터로 `mygroktable`이라는 테이블을 작성합니다.

```
CREATE EXTERNAL TABLE `mygroktable`(
   syslogbase string,
   queue_id string,
   syslog_message string
   )
ROW FORMAT SERDE
```

```
   'com.amazonaws.glue.serde.GrokSerDe'
WITH SERDEPROPERTIES (
   'input.grokCustomPatterns' = 'POSTFIX_QUEUEID [0-9A-F]{7,12}',
   'input.format'='%{SYSLOGBASE} %{POSTFIX_QUEUEID:queue_id}:
 %{GREEDYDATA:syslog_message}'
   )
STORED AS INPUTFORMAT
   'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
   'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
   's3://DOC-EXAMPLE-BUCKET/groksample/';
```

우선 %{NOTSPACE:column}처럼 간단한 패턴으로 시작해 열이 매핑되도록 한 다음, 필요에 따라 열을 직렬화합니다.

예제 2

다음 예에서는 Log4j 로그에 대한 쿼리를 만듭니다. 예제 로그에는 다음 형식의 항목이 있습니다.

```
2017-09-12 12:10:34,972 INFO  - processType=AZ, processId=ABCDEFG614B6F5E49,
 status=RUN,
threadId=123:amqListenerContainerPool23P:AJ|ABCDE9614B6F5E49||
2017-09-12T12:10:11.172-0700],
executionTime=7290, tenantId=12456, userId=123123f8535f8d76015374e7a1d87c3c,
 shard=testapp1,
jobId=12312345e5e7df0015e777fb2e03f3c, messageType=REAL_TIME_SYNC,
action=receive, hostname=1.abc.def.com
```

이 로그 데이터를 쿼리하려면:

- 각 열의 input.format에 Grok 패턴을 추가합니다. 예를 들어 timestamp에 %{TIMESTAMP_ISO8601:timestamp}를 추가합니다. loglevel에 %{LOGLEVEL:loglevel}을 추가합니다.
- 대시(-)와 로그 형식에서 항목을 구분하는 쉼표를 매핑하여 input.format의 패턴을 로그의 형식과 정확하게 일치시킵니다.

```
CREATE EXTERNAL TABLE bltest (
 timestamp STRING,
 loglevel STRING,
 processtype STRING,
```

```
 processid STRING,
 status STRING,
 threadid STRING,
 executiontime INT,
 tenantid INT,
 userid STRING,
 shard STRING,
 jobid STRING,
 messagetype STRING,
 action STRING,
 hostname STRING
 )
ROW FORMAT SERDE 'com.amazonaws.glue.serde.GrokSerDe'
WITH SERDEPROPERTIES (
"input.grokCustomPatterns" = 'C_ACTION receive|send',
"input.format" = "%{TIMESTAMP_ISO8601:timestamp} %{LOGLEVEL:loglevel} - processType=
%{NOTSPACE:processtype}, processId=%{NOTSPACE:processid}, status=%{NOTSPACE:status},
 threadId=%{NOTSPACE:threadid}, executionTime=%{POSINT:executiontime}, tenantId=
%{POSINT:tenantid}, userId=%{NOTSPACE:userid}, shard=%{NOTSPACE:shard}, jobId=
%{NOTSPACE:jobid}, messageType=%{NOTSPACE:messagetype}, action=%{C_ACTION:action},
 hostname=%{HOST:hostname}"
) STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/samples/';
```

예 3

Amazon S3 로그를 쿼리하는 다음 예제는 `'input.grokCustomPatterns'='INSIDE_QS ([^\"]*)\nINSIDE_BRACKETS ([^\\]]*)'` 예제 쿼리의 이 조각에서처럼 줄 바꿈 이스케이프 문자(\n)로 구분한 패턴 항목 2개를 포함하는 `'input.grokCustomPatterns'` 표현식을 보여줍니다.

```
CREATE EXTERNAL TABLE `s3_access_auto_raw_02`(
  `bucket_owner` string COMMENT 'from deserializer',
  `bucket` string COMMENT 'from deserializer',
  `time` string COMMENT 'from deserializer',
  `remote_ip` string COMMENT 'from deserializer',
  `requester` string COMMENT 'from deserializer',
  `request_id` string COMMENT 'from deserializer',
  `operation` string COMMENT 'from deserializer',
  `key` string COMMENT 'from deserializer',
  `request_uri` string COMMENT 'from deserializer',
  `http_status` string COMMENT 'from deserializer',
```

```
  `error_code` string COMMENT 'from deserializer',
  `bytes_sent` string COMMENT 'from deserializer',
  `object_size` string COMMENT 'from deserializer',
  `total_time` string COMMENT 'from deserializer',
  `turnaround_time` string COMMENT 'from deserializer',
  `referrer` string COMMENT 'from deserializer',
  `user_agent` string COMMENT 'from deserializer',
  `version_id` string COMMENT 'from deserializer')
ROW FORMAT SERDE
  'com.amazonaws.glue.serde.GrokSerDe'
WITH SERDEPROPERTIES (
  'input.format'='%{NOTSPACE:bucket_owner} %{NOTSPACE:bucket} \
\[%{INSIDE_BRACKETS:time}\\] %{NOTSPACE:remote_ip} %{NOTSPACE:requester}
 %{NOTSPACE:request_id} %{NOTSPACE:operation} %{NOTSPACE:key} \"?
%{INSIDE_QS:request_uri}\"? %{NOTSPACE:http_status} %{NOTSPACE:error_code}
 %{NOTSPACE:bytes_sent} %{NOTSPACE:object_size} %{NOTSPACE:total_time}
 %{NOTSPACE:turnaround_time} \"?%{INSIDE_QS:referrer}\"? \"?%{INSIDE_QS:user_agent}\"?
 %{NOTSPACE:version_id}',
  'input.grokCustomPatterns'='INSIDE_QS ([^\"]*)\nINSIDE_BRACKETS ([^\\]]*)')
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET'
```

JSON SerDe 라이브러리

Athena에서는 SerDe 라이브러리를 사용하여 JSON 데이터를 역직렬화할 수 있습니다. 역직렬화는 Parquet 또는 ORC와 같은 다른 형식으로 직렬화(기록)할 수 있도록 JSON 데이터를 변환합니다.

- 기본 Hive JSON SerDe
- OpenX JSON SerDe은
- Amazon Ion Hive SerDe은

> **Note**
>
> Hive 및 OpenX 라이브러리는 JSON 데이터가 줄 바꿈 문자로 구분된 레코드와 함께 한 줄(형식이 지정되지 않음)에 있을 것으로 예상합니다. Amazon Ion Hive SerDe는 이러한 요구 사항이 없으며 Ion 데이터 형식은 JSON의 상위 집합이므로 대안으로 사용할 수 있습니다.

라이브러리 이름

다음 중 하나를 사용하세요.

org.apache.hive.hcatalog.data.JsonSerDe

org.openx.data.jsonserde.JsonSerDe

com.amazon.ionhiveserde.IonHiveSerDe

Hive JSON SerDe

Hive JSON SerDe는 일반적으로 이벤트와 같은 JSON 데이터를 처리하는 데 사용됩니다. 이러한 이벤트는 새 줄로 구분된 JSON 인코딩 텍스트의 한 줄 문자열로 표시됩니다. Hive JSON SerDe는 `map` 또는 `struct` 키 이름에 중복 키를 허용하지 않습니다.

> **Note**
>
> SerDe는 각 JSON 문서가 레코드의 필드를 구분하는 줄 종료 문자가 없는 한 줄의 텍스트에 있을 것으로 예상합니다. JSON 텍스트가 가독성 좋게 꾸민 형식이면 테이블을 만든 후 쿼리하려고 할 때 HIVE_CURSOR_ERROR: 행이 유효한 JSON 객체가 아님(HIVE_CURSOR_ERROR: Row is not a valid JSON Object) 또는 HIVE_CURSOR_ERROR: JsonParseException: 예기치 않은 입력 종료: OBJECT의 닫기 마커 필요(HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT) 같은 오류 메시지가 나타날 수 있습니다. 자세한 내용은 GitHub의 OpenX SerDe 문서에서 JSON 데이터 파일을 참조하세요.

다음 예제 DDL 문은 Hive JSON SerDe를 사용하여 샘플 온라인 광고 데이터를 기반으로 테이블을 생성합니다. `LOCATION` 절에서 `s3://DOC-EXAMPLE-BUCKET.elasticmapreduce/samples/hive-ads/tables/impressions`의 *myregion*을, Athena를 실행하는 리전 식별자로 바꿉니다(예: `s3://us-west-2.elasticmapreduce/samples/hive-ads/tables/impressions`).

```
CREATE EXTERNAL TABLE impressions (
    requestbegintime string,
    adid string,
    impressionid string,
    referrer string,
    useragent string,
    usercookie string,
    ip string,
    number string,
```

```
    processid string,
    browsercookie string,
    requestendtime string,
    timers struct
                <
                 modellookup:string,
                 requesttime:string
                >,
    threadid string,
    hostname string,
    sessionid string
)
PARTITIONED BY (dt string)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
LOCATION 's3://DOC-EXAMPLE-BUCKET.elasticmapreduce/samples/hive-ads/tables/
impressions';
```

Hive JSON SerDe로 타임스탬프 형식 지정

문자열에서 타임스탬프 값을 구문 분석하려면 `ROW FORMAT SERDE`절에 `WITH SERDEPROPERTIES` 하위 필드을 추가하고 이를 사용하여 `timestamp.formats` 파라미터를 지정할 수 있습니다. 파라미터에서 아래 예제와 같이 하나 이상의 타임스탬프 패턴를 쉼표로 구분한 목록을 지정합니다.

```
...
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
WITH SERDEPROPERTIES ("timestamp.formats"="yyyy-MM-dd'T'HH:mm:ss.SSS'Z',yyyy-MM-
dd'T'HH:mm:ss")
...
```

자세한 내용은 Apache Hive 설명서의 Timestamps를 참조하세요.

쿼리를 위한 테이블 로드

테이블을 생성한 후 MSCK REPAIR TABLE을 실행하여 테이블을 로드하면 Athena에서 쿼리가 가능합니다.

```
MSCK REPAIR TABLE impressions
```

CloudTrail 로그 쿼리

Hive JSON SerDe를 사용하여 CloudTrail 로그를 쿼리할 수 있습니다. 자세한 내용과 예제 `CREATE TABLE` 문은 AWS CloudTrail 로그 쿼리 단원을 참조하세요.

OpenX JSON SerDe

Hive JSON SerDe와 마찬가지로 OpenX JSON을 사용하여 JSON 데이터를 처리할 수 있습니다. 데이터는 새 줄로 구분된 JSON 인코딩 텍스트의 한 줄 문자열로도 표시됩니다. Hive JSON SerDe와 마찬가지로 OpenX JSON SerDe는 `map` 또는 `struct` 키 이름에 중복 키를 허용하지 않습니다.

> **Note**
>
> SerDe는 각 JSON 문서가 레코드의 필드를 구분하는 줄 종료 문자가 없는 한 줄의 텍스트에 있을 것으로 예상합니다. JSON 텍스트가 가독성 좋게 꾸민 형식이면 테이블을 만든 후 쿼리하려고 할 때 HIVE_CURSOR_ERROR: 행이 유효한 JSON 객체가 아님(HIVE_CURSOR_ERROR: Row is not a valid JSON Object) 또는 HIVE_CURSOR_ERROR: JsonParseException: 예기치 않은 입력 종료: OBJECT의 닫기 마커 필요(HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT) 같은 오류 메시지가 나타날 수 있습니다. 자세한 내용은 GitHub의 OpenX SerDe 문서에서 JSON 데이터 파일을 참조하세요.

선택적 속성

Hive JSON SerDe와 달리 OpenX JSON SerDe에는 데이터의 불일치를 해결하는 데 유용한 다음과 같은 선택적 SerDe 속성이 있습니다.

ignore.malformed.json

선택 사항입니다. `TRUE`로 설정하면 형식이 잘못된 JSON 구문을 건너뛸 수 있습니다. 기본값은 `FALSE`입니다.

dots.in.keys

선택 사항입니다. 기본값은 `FALSE`입니다. `TRUE`로 설정하면 SerDe가 키 이름의 점을 밑줄로 바꿀 수 있습니다. 예를 들어 JSON 데이터 세트에 이름이 `"a.b"`인 키가 있을 경우 이 속성을 사용하여 Athena에서 열 이름이 `"a_b"`가 되도록 정의할 수 있습니다. 이 SerDe가 없으면 기본적으로 Athena는 열 이름에 점을 허용하지 않습니다.

case.insensitive

선택 사항입니다. 기본값은 `TRUE`입니다. `TRUE`로 설정하면 SerDe가 모든 대문자 열을 소문자로 변환합니다.

데이터에서 대/소문자를 구분하는 키 이름을 사용하려면 `WITH SERDEPROPERTIES ("case.insensitive"= FALSE;)`를 사용합니다. 그런 다음, 아직 모두 소문자가 아닌 모든 키에 대해 다음 구문을 사용하여 열 이름에서 속성 이름으로 매핑을 제공합니다.

```
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES ("case.insensitive" = "FALSE", "mapping.userid" = "userId")
```

URL 및 Url와 같이 소문자로 동일한 두 개의 키가 있는 경우, 다음과 같은 오류가 발생할 수 있습니다.

HIVE_CURSOR_ERROR: 행이 유효한 JSON 객체가 아님 - JSONException: 중복 키 "url"

이 문제를 해결하려면 다음 예제와 같이 case.insensitive 속성을 FALSE로 설정하고 키를 다른 이름으로 매핑합니다.

```
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES ("case.insensitive" = "FALSE", "mapping.url1" = "URL",
 "mapping.url2" = "Url")
```

매핑

선택 사항입니다. 열 이름을, 열 이름과 동일하지 않은 JSON 키에 매핑합니다. mapping 파라미터는 JSON 데이터에 키워드인 키가 있을 경우에 유용합니다. 예를 들어 timestamp라는 JSON 키가 있는 경우 다음 구문을 사용하여 키를 ts라는 열에 매핑합니다.

```
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES ("mapping.ts" = "timestamp")
```

Hive 호환 이름에 콜론이 있는 중첩된 필드 이름 매핑

struct 내에 콜론이 있는 필드 이름이 있는 경우 mapping 속성을 사용하여 Hive 호환 이름에 필드를 매핑할 수 있습니다. 예를 들어 열 유형 정의에 my:struct:field:string이 포함된 경우 WITH SERDEPROPERTIES에 다음 항목을 포함하여 my_struct_field:string에 정의를 매핑할 수 있습니다.

```
("mapping.my_struct_field" = "my:struct:field")
```

다음 예제에서는 해당 CREATE TABLE 문을 보여줍니다.

```
CREATE EXTERNAL TABLE colon_nested_field (
item struct<my_struct_field:string>)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
WITH SERDEPROPERTIES ("mapping.my_struct_field" = "my:struct:field")
```

예제: 광고 데이터

다음 예제 DDL 문은 OpenX JSON SerDe를 사용하여 Hive JSON SerDe의 예제에 사용된 것과 동일한 샘플 온라인 광고 데이터를 기반으로 테이블을 생성합니다. LOCATION 절에서 *myregion*을, Athena를 실행하는 리전 식별자로 바꿉니다.

```
CREATE EXTERNAL TABLE impressions (
    requestbegintime string,
    adid string,
    impressionId string,
    referrer string,
    useragent string,
    usercookie string,
    ip string,
    number string,
    processid string,
    browsercokie string,
    requestendtime string,
    timers struct<
       modellookup:string,
       requesttime:string>,
    threadid string,
    hostname string,
    sessionid string
)   PARTITIONED BY (dt string)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://DOC-EXAMPLE-BUCKET.elasticmapreduce/samples/hive-ads/tables/
impressions';
```

예제: 중첩 JSON 역직렬화

JSON SerDes를 사용하여 더 복잡한 JSON 인코딩 데이터를 구문 분석 할 수 있습니다. 이를 위해서는 `struct` 및 `array` 요소를 사용하여 중첩 구조를 나타내는 `CREATE TABLE` 문을 사용해야 합니다.

다음 예제에서는 중첩 구조가 있는 JSON 데이터로 Athena 테이블을 생성합니다. Athena에서 JSON 인코딩 데이터를 구문 분석하려면 각 JSON 문서가 한 줄당 하나씩 나열되며 줄 바꿈으로 구분되는지 확인합니다.

이 예제는 다음과 같은 구조의 JSON 인코딩 데이터를 가정합니다.

```
{
"DocId": "AWS",
```

```
"User": {
        "Id": 1234,
        "Username": "bob1234",
        "Name": "Bob",
"ShippingAddress": {
"Address1": "123 Main St.",
"Address2": null,
"City": "Seattle",
"State": "WA"
   },
"Orders": [
   {
     "ItemId": 6789,
     "OrderDate": "11/11/2017"
   },
   {
     "ItemId": 4352,
     "OrderDate": "12/12/2017"
   }
  ]
 }
}
```

다음 CREATE TABLE 문은 Openx-JsonSerDe와 struct 및 array 컬렉션 데이터 형식을 사용하여 개체 그룹을 설정할 수 있습니다. 각 JSON 문서는 한 줄당 하나씩 나열되며 줄 바꿈으로 구분이 됩니다. 오류를 방지하기 위해 쿼리되는 데이터에는 struct 또는 맵 키 이름의 중복 키가 포함되지 않습니다.

```
CREATE external TABLE complex_json (
   docid string,
   `user` struct<
               id:INT,
               username:string,
               name:string,
               shippingaddress:struct<
                                address1:string,
                                address2:string,
                                city:string,
                                state:string
                                >,
               orders:array<
                           struct<
                                itemid:INT,
```

```
                                orderdate:string
                                >
                            >
                >
    )
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://DOC-EXAMPLE-BUCKET/myjsondata/';
```

추가적인 리소스

Athena의 JSON 및 중첩 JSON 작업에 대한 자세한 내용은 다음 리소스를 참조하세요.

- Create tables in Amazon Athena from nested JSON and mappings using JSONSerDe(AWS 빅 데이터 블로그)
- Amazon Athena에서 JSON 데이터를 읽으려고 할 때 오류가 발생합니다.(AWS Knowledge Center 문서)
- hive-json-schema(GitHub) – 예제 JSON 문서에서 `CREATE TABLE` 문을 생성하는 Java로 작성된 도구입니다. 생성된 `CREATE TABLE` 문은 OpenX JSON Serde를 사용합니다.

CSV, TSV, 사용자 지정 구분 기호로 구분된 파일에 대한 LazySimpleSerDe

이 SerDe 지정은 선택 사항입니다. 이는 Athena가 기본적으로 사용하는 CSV, TSV 및 사용자 지정 구분 기호로 구분된 형식의 데이터에 대한 SerDe입니다. 이 SerDe는 어떤 SerDe도 지정하지 않고 `ROW FORMAT DELIMITED`만 지정한 경우에 사용됩니다. 데이터에 인용 부호로 묶인 값이 없는 경우 이 SerDe를 사용합니다.

LazySimpleSerDe에 대한 참조 문서는 Apache Hive 개발자 안내서의 Hive SerDe 단원을 참조하세요.

라이브러리 이름

LazySimpleSerDe의 클래스 라이브러리 이름은 `org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe`입니다. LazySimpleSerDe 클래스에 대한 자세한 내용은 GitHub.com의 LazySimpleSerDe.java를 참조하세요.

헤더 무시

테이블을 정의할 때 데이터의 헤더를 무시하려면 다음 예제와 같이 `skip.header.line.count` 테이블 속성을 사용합니다.

```
TBLPROPERTIES ("skip.header.line.count"="1")
```

예제는 Amazon VPC 흐름 로그 쿼리 및 Amazon CloudFront 로그 쿼리의 CREATE TABLE 문을 참조하세요.

CSV 예

다음 예제에서는 Athena에서 LazySimpleSerDe를 사용하여 CSV 데이터로 테이블을 생성하는 방법을 보여줍니다. 이 SerDe를 사용하여 사용자 지정 구분 파일을 역직렬화하려면 다음 예제의 패턴을 따라합니다. 단, FIELDS TERMINATED BY 절을 사용하여 서로 다른 단일 문자 구분 기호를 지정합니다. LazySimpleSerDe는 다중 문자 구분 기호를 지원하지 않습니다.

> **Note**
>
> s3://athena-examples-*myregion*/path/to/data/의 *myregion*을, Athena를 실행하는 리전 식별자로 바꿉니다(예: s3://athena-examples-us-west-1/path/to/data/).

CREATE TABLE 문을 사용하여 Amazon S3에 저장된 CSV 형식의 기본 데이터로 Athena 테이블을 만듭니다.

```
CREATE EXTERNAL TABLE flight_delays_csv (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
    carrier STRING,
    tailnum STRING,
    flightnum STRING,
    originairportid INT,
    originairportseqid INT,
    origincitymarketid INT,
    origin STRING,
    origincityname STRING,
    originstate STRING,
    originstatefips STRING,
    originstatename STRING,
    originwac INT,
    destairportid INT,
    destairportseqid INT,
```

```
    destcitymarketid INT,
    dest STRING,
    destcityname STRING,
    deststate STRING,
    deststatefips STRING,
    deststatename STRING,
    destwac INT,
    crsdeptime STRING,
    deptime STRING,
    depdelay INT,
    depdelayminutes INT,
    depdel15 INT,
    departuredelaygroups INT,
    deptimeblk STRING,
    taxiout INT,
    wheelsoff STRING,
    wheelson STRING,
    taxiin INT,
    crsarrtime INT,
    arrtime STRING,
    arrdelay INT,
    arrdelayminutes INT,
    arrdel15 INT,
    arrivaldelaygroups INT,
    arrtimeblk STRING,
    cancelled INT,
    cancellationcode STRING,
    diverted INT,
    crselapsedtime INT,
    actualelapsedtime INT,
    airtime INT,
    flights INT,
    distance INT,
    distancegroup INT,
    carrierdelay INT,
    weatherdelay INT,
    nasdelay INT,
    securitydelay INT,
    lateaircraftdelay INT,
    firstdeptime STRING,
    totaladdgtime INT,
    longestaddgtime INT,
    divairportlandings INT,
    divreacheddest INT,
```

```
    divactualelapsedtime INT,
    divarrdelay INT,
    divdistance INT,
    div1airport STRING,
    div1airportid INT,
    div1airportseqid INT,
    div1wheelson STRING,
    div1totalgtime INT,
    div1longestgtime INT,
    div1wheelsoff STRING,
    div1tailnum STRING,
    div2airport STRING,
    div2airportid INT,
    div2airportseqid INT,
    div2wheelson STRING,
    div2totalgtime INT,
    div2longestgtime INT,
    div2wheelsoff STRING,
    div2tailnum STRING,
    div3airport STRING,
    div3airportid INT,
    div3airportseqid INT,
    div3wheelson STRING,
    div3totalgtime INT,
    div3longestgtime INT,
    div3wheelsoff STRING,
    div3tailnum STRING,
    div4airport STRING,
    div4airportid INT,
    div4airportseqid INT,
    div4wheelson STRING,
    div4totalgtime INT,
    div4longestgtime INT,
    div4wheelsoff STRING,
    div4tailnum STRING,
    div5airport STRING,
    div5airportid INT,
    div5airportseqid INT,
    div5wheelson STRING,
    div5totalgtime INT,
    div5longestgtime INT,
    div5wheelsoff STRING,
    div5tailnum STRING
)
```

```
    PARTITIONED BY (year STRING)
    ROW FORMAT DELIMITED
      FIELDS TERMINATED BY ','
      ESCAPED BY '\\'
      LINES TERMINATED BY '\n'
    LOCATION 's3://athena-examples-myregion/flight/csv/';
```

이 테이블에 새 파티션이 추가될 때마다 `MSCK REPAIR TABLE`을(를) 실행하여 파티션 메타데이터를 새로 고칩니다.

```
MSCK REPAIR TABLE flight_delays_csv;
```

1시간 이상 지연된 상위 10개 경로를 쿼리합니다.

```
SELECT origin, dest, count(*) as delays
FROM flight_delays_csv
WHERE depdelayminutes > 60
GROUP BY origin, dest
ORDER BY 3 DESC
LIMIT 10;
```

> **Note**
>
> 비행 테이블 데이터의 출처는 미국 운수부 운송통계국에서 제공한 항공편입니다. 원본에서 채도를 낮췄습니다.

TSV 예

Amazon S3에 저장된 TSV 데이터에서 Athena 테이블을 생성하려면 `ROW FORMAT DELIMITED`를 사용하고 `\t`를 탭 필드 구분 기호로, `\n`을 줄 구분 기호로, `\`를 이스케이프 문자로 지정합니다. 다음 발췌 부분에서 이 구문을 보여줍니다. `athena-examples` 위치에 사용 가능한 샘플 TSV 플라이트 데이터가 없지만 CSV 테이블과 마찬가지로 새 파티션을 추가할 때마다 파티션 메타데이터를 새로 고치도록 `MSCK REPAIR TABLE`을 실행해야 합니다.

```
...
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
ESCAPED BY '\\'
LINES TERMINATED BY '\n'
```

```
...
```

CSV 처리를 위한 OpenCSVSerDe

CSV 데이터를 위한 Athena 테이블을 생성할 때 데이터에 포함되는 값의 유형에 따라 사용할 SerDe를 결정합니다.

- 데이터에 큰따옴표(")로 묶인 값이 포함되어 있으면 OpenCSV SerDe를 사용하여 Athena의 값을 역직렬화할 수 있습니다. 데이터에 큰따옴표(")로 묶인 값이 포함되어 있지 않으면 SerDe를 지정하지 않아도 됩니다. 이 경우 Athena는 기본 `LazySimpleSerDe`를 사용합니다. 자세한 설명은 CSV, TSV, 사용자 지정 구분 기호로 구분된 파일에 대한 LazySimpleSerDe을 참조하세요.
- 데이터에 UNIX 숫자 `TIMESTAMP` 값이 있는 경우(예: `1579059880000`) OpenCSVSerDe를 사용합니다. 데이터가 `java.sql.Timestamp` 형식을 사용하는 경우 LazySimpleSerDe를 사용합니다.

CSV SerDe(OpenCSVSerDe)

OpenCSV SerDe는 문자열 데이터에 대해 다음과 같은 특성을 갖습니다.

- 큰따옴표(")를 기본 인용 부호로 사용하고 다음과 같은 구분자, 따옴표 및 이스케이프 문자를 지정할 수 있습니다.

  ```
  WITH SERDEPROPERTIES ("separatorChar" = ",", "quoteChar" = "`", "escapeChar" = "\\" )
  ```

- `\t` 또는 `\n`을 직접 이스케이프할 수 있습니다. 이스케이프하려면 `"escapeChar" = "\\"`를 사용합니다. 이 주제의 예제를 참조하세요.
- CSV 파일에 포함된 줄 바꿈을 지원하지 않습니다.

`STRING`이 아닌 데이터 형식의 경우 OpenCSVSerDe는 다음과 같이 동작합니다.

- `BOOLEAN`, `BIGINT`, `INT`, `DOUBLE` 데이터 형식을 인식합니다.
- 숫자 데이터 형식으로 정의된 열의 빈 값이나 null 값을 인식하지 못하므로 이러한 값을 `string`으로 유지합니다. 한 가지 해결 방법은 null 값을 포함한 열을 `string`으로 생성한 다음 `CAST`를 사용하여 쿼리의 필드를 숫자 데이터 형식으로 변환하면서 null에 대해 기본값 `0`을 부여하는 것입니다. 자세한 내용은 AWS 지식 센터의 Athena에서 CSV 데이터를 쿼리하는 경우 HIVE_BAD_DATA: Error parsing field value 오류가 발생합니다를 참조하세요.
- `CREATE TABLE` 문에서 `timestamp` 데이터 형식으로 지정된 열의 경우 밀리초 단위의 UNIX 숫자 형식(예: `1579059880000`)으로 지정되어 있으면 `TIMESTAMP` 데이터를 인식합니다.

  - OpenCSVSerDe는 "YYYY-MM-DD HH:MM:SS.fffffffff"(소수점 9자리 정밀도)처럼 JDBC와 호환되는 java.sql.Timestamp 형식의 TIMESTAMP는 지원하지 않습니다.
- CREATE TABLE 문에서 DATE 데이터 형식으로 지정된 열의 경우 값이 1970년 1월 1일 이후 경과된 일 수를 나타내면 값을 인식합니다. 예를 들어 date 데이터 형식을 가진 열의 값 18276은 쿼리 시 2020-01-15로 렌더링됩니다. 이 UNIX 형식에서는 하루를 86,400초로 간주합니다.
  - OpenCSVSerDe는 다른 형식의 DATE를 직접 지원하지 않습니다. 다른 형식의 타임스탬프 데이터를 처리하려면 열을 string으로 정의한 다음 SELECT 쿼리에서 원하는 결과를 반환하도록 시간 변환 함수를 사용합니다. 자세한 내용은 AWS 지식 센터의 Amazon Athena에서 테이블을 쿼리할 때 TIMESTAMP 결과가 비어 있음 항목을 참조하세요.
- 표에서 열을 원하는 형식으로 변환하려면 테이블에 대한 뷰를 생성하고 CAST를 사용하여 원하는 형식으로 변환할 수 있습니다.

Example 예: UNIX 숫자 형식으로 지정된 TIMESTAMP 유형 및 DATE 유형 사용

쉼표로 구분된 데이터를 포함한 다음 3개 열을 생각해보겠습니다. 각 열의 값은 큰따옴표로 묶여 있습니다.

```
"unixvalue creationdate 18276 creationdatetime 1579059880000","18276","1579059880000"
```

다음 문은 Athena에서, 지정된 Amazon S3 버킷 위치로부터 테이블을 만듭니다.

```
CREATE EXTERNAL TABLE IF NOT EXISTS testtimestamp1(
 `profile_id` string,
 `creationdate` date,
 `creationdatetime` timestamp
 )
 ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
 LOCATION 's3://DOC-EXAMPLE-BUCKET'
```

이제 다음 쿼리를 실행합니다.

```
SELECT * FROM testtimestamp1
```

이 쿼리는 날짜 및 시간 데이터를 보여주는 다음과 같은 결과를 반환합니다.

```
profile_id                                                    creationdate
 creationdatetime
```

```
unixvalue creationdate 18276 creationdatetime 1579146280000       2020-01-15
 2020-01-15 03:44:40.000
```

Example 예: **\t** 또는 **\n** 이스케이프

다음 테스트 데이터를 고려하세요.

```
" \\t\\t\\n 123 \\t\\t\\n ",abc
" 456 ",xyz
```

다음 문은 Athena에서 "escapeChar" = "\\"를 지정한 테이블을 만듭니다.

```
CREATE EXTERNAL TABLE test1 (
f1 string,
s2 string)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES ("separatorChar" = ",", "escapeChar" = "\\")
LOCATION 's3://DOC-EXAMPLE-BUCKET/dataset/test1/'
```

이제 다음 쿼리를 실행합니다.

```
SELECT * FROM test1;
```

\t 또는 \n을 올바르게 이스케이프하여 이 결과를 반환합니다.

```
f1            s2
\t\t\n 123 \t\t\n            abc
456                          xyz
```

SerDe 이름

CSV SerDe

라이브러리 이름

이 SerDe를 사용하려면 ROW FORMAT SERDE 뒤에 정규화된 클래스 이름을 지정합니다. 또한 다음과 같이 SERDEPROPERTIES에 구분 기호를 지정합니다.

```
...
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
```

```
  "quoteChar"     = "`",
  "escapeChar"    = "\\"
)
```

헤더 무시

테이블을 정의할 때 데이터의 헤더를 무시하려면 다음 예제와 같이 `skip.header.line.count` 테이블 속성을 사용합니다.

```
TBLPROPERTIES ("skip.header.line.count"="1")
```

예제는 Amazon VPC 흐름 로그 쿼리 및 Amazon CloudFront 로그 쿼리의 `CREATE TABLE` 문을 참조하세요.

예

이 예에서는 다음 내용이 포함된 CSV의 데이터가 `s3://DOC-EXAMPLE-BUCKET/mycsv/`에 저장되었다고 가정합니다.

```
"a1","a2","a3","a4"
"1","2","abc","def"
"a","a1","abc3","ab4"
```

`CREATE TABLE` 문을 사용해 해당 데이터를 바탕으로 한 Athena 테이블을 만듭니다. 다음 예제처럼 `ROW FORMAT SERDE` 뒤의 OpenCSVSerDe 클래스를 참조해 `WITH SERDEPROPERTIES`에 문자 구분 기호, 따옴표, 이스케이프 문자를 지정합니다.

```
CREATE EXTERNAL TABLE myopencsvtable (
   col1 string,
   col2 string,
   col3 string,
   col4 string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
   'separatorChar' = ',',
   'quoteChar' = '"',
   'escapeChar' = '\\'
   )
STORED AS TEXTFILE
LOCATION 's3://DOC-EXAMPLE-BUCKET/mycsv/';
```

테이블의 모든 값을 쿼리합니다.

```
SELECT * FROM myopencsvtable;
```

쿼리는 다음 값을 반환합니다.

```
col1     col2    col3    col4
-----------------------------
a1       a2      a3      a4
1        2       abc     def
a        a1      abc3    ab4
```

ORC SerDe

SerDe 이름

OrcSerDe

라이브러리 이름

이 라이브러리는 ORC 형식의 데이터를 위해 Apache Hive OrcSerde.java 클래스를 사용합니다. ORC에서 리더로, ORC에서 작성기로 객체를 전달합니다.

예제

> **Note**
>
> `s3://athena-examples-myregion/path/to/data/`의 *myregion*을, Athena를 실행하는 리전 식별자로 바꿉니다(예: `s3://athena-examples-us-west-1/path/to/data/`).

다음 예에서는 ORC에 비행 지연 데이터의 테이블을 생성합니다. 테이블에 파티션이 포함됩니다.

```
DROP TABLE flight_delays_orc;
CREATE EXTERNAL TABLE flight_delays_orc (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
```

```
    airlineid INT,
    carrier STRING,
    tailnum STRING,
    flightnum STRING,
    originairportid INT,
    originairportseqid INT,
    origincitymarketid INT,
    origin STRING,
    origincityname STRING,
    originstate STRING,
    originstatefips STRING,
    originstatename STRING,
    originwac INT,
    destairportid INT,
    destairportseqid INT,
    destcitymarketid INT,
    dest STRING,
    destcityname STRING,
    deststate STRING,
    deststatefips STRING,
    deststatename STRING,
    destwac INT,
    crsdeptime STRING,
    deptime STRING,
    depdelay INT,
    depdelayminutes INT,
    depdel15 INT,
    departuredelaygroups INT,
    deptimeblk STRING,
    taxiout INT,
    wheelsoff STRING,
    wheelson STRING,
    taxiin INT,
    crsarrtime INT,
    arrtime STRING,
    arrdelay INT,
    arrdelayminutes INT,
    arrdel15 INT,
    arrivaldelaygroups INT,
    arrtimeblk STRING,
    cancelled INT,
    cancellationcode STRING,
    diverted INT,
    crselapsedtime INT,
```

```
    actualelapsedtime INT,
    airtime INT,
    flights INT,
    distance INT,
    distancegroup INT,
    carrierdelay INT,
    weatherdelay INT,
    nasdelay INT,
    securitydelay INT,
    lateaircraftdelay INT,
    firstdeptime STRING,
    totaladdgtime INT,
    longestaddgtime INT,
    divairportlandings INT,
    divreacheddest INT,
    divactualelapsedtime INT,
    divarrdelay INT,
    divdistance INT,
    div1airport STRING,
    div1airportid INT,
    div1airportseqid INT,
    div1wheelson STRING,
    div1totalgtime INT,
    div1longestgtime INT,
    div1wheelsoff STRING,
    div1tailnum STRING,
    div2airport STRING,
    div2airportid INT,
    div2airportseqid INT,
    div2wheelson STRING,
    div2totalgtime INT,
    div2longestgtime INT,
    div2wheelsoff STRING,
    div2tailnum STRING,
    div3airport STRING,
    div3airportid INT,
    div3airportseqid INT,
    div3wheelson STRING,
    div3totalgtime INT,
    div3longestgtime INT,
    div3wheelsoff STRING,
    div3tailnum STRING,
    div4airport STRING,
    div4airportid INT,
```

```
    div4airportseqid INT,
    div4wheelson STRING,
    div4totalgtime INT,
    div4longestgtime INT,
    div4wheelsoff STRING,
    div4tailnum STRING,
    div5airport STRING,
    div5airportid INT,
    div5airportseqid INT,
    div5wheelson STRING,
    div5totalgtime INT,
    div5longestgtime INT,
    div5wheelsoff STRING,
    div5tailnum STRING
)
PARTITIONED BY (year String)
STORED AS ORC
LOCATION 's3://athena-examples-myregion/flight/orc/'
tblproperties ("orc.compress"="ZLIB");
```

테이블에서 `MSCK REPAIR TABLE` 문을 실행해 파티션 메타데이터를 새로 고칩니다:

```
MSCK REPAIR TABLE flight_delays_orc;
```

이 쿼리를 사용해 1시간 이상 지연된 상위 10개 경로를 얻습니다.

```
SELECT origin, dest, count(*) as delays
FROM flight_delays_orc
WHERE depdelayminutes > 60
GROUP BY origin, dest
ORDER BY 3 DESC
LIMIT 10;
```

Parquet SerDe

SerDe 이름

ParquetHiveSerDe는 Parquet 형식으로 저장된 데이터에 사용됩니다.

> **Note**
>
> 데이터를 Parquet 형식으로 변환하려면 CREATE TABLE AS SELECT(CTAS) 쿼리를 사용합니다. 자세한 내용은 쿼리 결과에서 테이블 생성(CTAS), CTAS 쿼리 예제 및 ETL 및 데이터 분석에 CTAS 및 INSERT INTO 사용 섹션을 참조하세요.

라이브러리 이름

Athena는 Parquet에 저장된 데이터를 역직렬화해야 할 때 다음 클래스를 사용합니다. `org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe`

예: parquet에 저장된 파일 쿼리

> **Note**
>
> `s3://athena-examples-myregion/path/to/data/`의 *myregion*을, Athena를 실행하는 리전 식별자로 바꿉니다(예: `s3://athena-examples-us-west-1/path/to/data/`).

다음 `CREATE TABLE` 문을 사용하여 Amazon S3에 Parquet 형식으로 저장된 기본 데이터로 Athena 테이블을 만듭니다.

```
CREATE EXTERNAL TABLE flight_delays_pq (
    yr INT,
    quarter INT,
    month INT,
    dayofmonth INT,
    dayofweek INT,
    flightdate STRING,
    uniquecarrier STRING,
    airlineid INT,
    carrier STRING,
    tailnum STRING,
    flightnum STRING,
    originairportid INT,
    originairportseqid INT,
    origincitymarketid INT,
    origin STRING,
    origincityname STRING,
    originstate STRING,
```

```
    originstatefips STRING,
    originstatename STRING,
    originwac INT,
    destairportid INT,
    destairportseqid INT,
    destcitymarketid INT,
    dest STRING,
    destcityname STRING,
    deststate STRING,
    deststatefips STRING,
    deststatename STRING,
    destwac INT,
    crsdeptime STRING,
    deptime STRING,
    depdelay INT,
    depdelayminutes INT,
    depdel15 INT,
    departuredelaygroups INT,
    deptimeblk STRING,
    taxiout INT,
    wheelsoff STRING,
    wheelson STRING,
    taxiin INT,
    crsarrtime INT,
    arrtime STRING,
    arrdelay INT,
    arrdelayminutes INT,
    arrdel15 INT,
    arrivaldelaygroups INT,
    arrtimeblk STRING,
    cancelled INT,
    cancellationcode STRING,
    diverted INT,
    crselapsedtime INT,
    actualelapsedtime INT,
    airtime INT,
    flights INT,
    distance INT,
    distancegroup INT,
    carrierdelay INT,
    weatherdelay INT,
    nasdelay INT,
    securitydelay INT,
    lateaircraftdelay INT,
```

```
    firstdeptime STRING,
    totaladdgtime INT,
    longestaddgtime INT,
    divairportlandings INT,
    divreacheddest INT,
    divactualelapsedtime INT,
    divarrdelay INT,
    divdistance INT,
    div1airport STRING,
    div1airportid INT,
    div1airportseqid INT,
    div1wheelson STRING,
    div1totalgtime INT,
    div1longestgtime INT,
    div1wheelsoff STRING,
    div1tailnum STRING,
    div2airport STRING,
    div2airportid INT,
    div2airportseqid INT,
    div2wheelson STRING,
    div2totalgtime INT,
    div2longestgtime INT,
    div2wheelsoff STRING,
    div2tailnum STRING,
    div3airport STRING,
    div3airportid INT,
    div3airportseqid INT,
    div3wheelson STRING,
    div3totalgtime INT,
    div3longestgtime INT,
    div3wheelsoff STRING,
    div3tailnum STRING,
    div4airport STRING,
    div4airportid INT,
    div4airportseqid INT,
    div4wheelson STRING,
    div4totalgtime INT,
    div4longestgtime INT,
    div4wheelsoff STRING,
    div4tailnum STRING,
    div5airport STRING,
    div5airportid INT,
    div5airportseqid INT,
    div5wheelson STRING,
```

```
    div5totalgtime INT,
    div5longestgtime INT,
    div5wheelsoff STRING,
    div5tailnum STRING
)
PARTITIONED BY (year STRING)
STORED AS PARQUET
LOCATION 's3://athena-examples-myregion/flight/parquet/'
tblproperties ("parquet.compression"="SNAPPY");
```

테이블에서 MSCK REPAIR TABLE 문을 실행해 파티션 메타데이터를 새로 고칩니다:

```
MSCK REPAIR TABLE flight_delays_pq;
```

1시간 이상 지연된 상위 10개 경로를 쿼리합니다.

```
SELECT origin, dest, count(*) as delays
FROM flight_delays_pq
WHERE depdelayminutes > 60
GROUP BY origin, dest
ORDER BY 3 DESC
LIMIT 10;
```

> **Note**
>
> 비행 테이블 데이터의 출처는 미국 운수부 운송통계국에서 제공한 항공편입니다. 원본에서 채도를 낮췄습니다.

Parquet 통계 무시

Parquet 데이터를 읽을 때 다음과 같은 오류 메시지가 나타날 수 있습니다.

```
HIVE_CANNOT_OPEN_SPLIT: Index x out of bounds for length y
HIVE_CURSOR_ERROR: Failed to read x bytes
HIVE_CURSOR_ERROR: FailureException at Malformed input: offset=x
HIVE_CURSOR_ERROR: FailureException at java.io.IOException:
can not read class org.apache.parquet.format.PageHeader: Socket is closed by peer.
```

이 문제를 해결하려면 다음 예제와 같이 CREATE TABLE 또는 ALTER TABLE SET TBLPROPERTIES 문을 사용하여 Parquet SerDe `parquet.ignore.statistics` 속성을 `true`로 설정합니다.

CREATE TABLE 예제

```
...
ROW FORMAT SERDE
'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
WITH SERDEPROPERTIES (
'parquet.ignore.statistics'='true')
STORED AS PARQUET
...
```

ALTER TABLE 예제

```
ALTER TABLE ... SET TBLPROPERTIES ('parquet.ignore.statistics'='true')
```

Regex SerDe

Regex SerDe는 정규 표현식(정규식)을 사용하여 정규식 그룹을 테이블 열로 추출하여 데이터를 역직렬화합니다.

데이터의 행이 정규식과 일치하지 않으면 행의 모든 열이 `NULL`로 반환됩니다. 행이 정규식과 일치하지만 예상보다 적은 그룹이 있는 경우 누락된 그룹은 `NULL`입니다. 데이터의 행이 정규식과 일치하지만 정규식의 그룹보다 많은 열이 있는 경우 추가 열은 무시됩니다.

자세한 내용은 Apache Hive 설명서의 Class RegexSerDe를 참조하세요.

SerDe 이름

RegexSerDe

라이브러리 이름

RegexSerDe

예제

다음 예제에서는 RegExSerDe를 사용하여 CloudFront 로그에서 테이블을 생성합니다. `s3://athena-examples-`*`myregion`*`/cloudfront/plaintext/`의 *`myregion`*을, Athena를 실행하는 리전 식별자로 바꿉니다(예: `s3://athena-examples-us-west-1/cloudfront/plaintext/`).

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_logs (
  `Date` DATE,
  Time STRING,
  Location STRING,
  Bytes INT,
  RequestIP STRING,
  Method STRING,
  Host STRING,
  Uri STRING,
  Status INT,
  Referrer STRING,
  os STRING,
  Browser STRING,
  BrowserVersion STRING
 )
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
 WITH SERDEPROPERTIES (
 "input.regex" = "^(?!#)([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\
\s+([^ ]+)\\s+([^ ]+)\\s+([^ ]+)\\s+[^\(]+[\(]([^\;]+).*\%20([^\/]+)[\/]
(.*)$"
 )
LOCATION 's3://athena-examples-myregion/cloudfront/plaintext/';
```

Amazon Athena를 사용하여 SQL 쿼리 실행

AWS Glue Data Catalog에 등록된 데이터 원본과 Hive 메타스토어 및 Amazon DocumentDB 인스턴스와 같이 Athena 연동 쿼리를 사용하여 연결하는 데이터 원본에서 Amazon Athena를 사용하여 SQL 쿼리를 실행할 수 있습니다. 데이터 원본 작업에 대한 자세한 내용은 데이터 원본에 연결 단원을 참조하세요. 스키마를 수정하는 데이터 정의 언어(DDL) 쿼리를 실행하면 Athena가 데이터 원본과 연결된 메타스토어에 메타데이터를 작성합니다. 또한 `CREATE TABLE AS` 및 `INSERT INTO` 같은 일부 쿼리는 데이터 세트에 레코드를 쓸 수 있습니다(예: Amazon S3 위치에 CSV 레코드 추가). 쿼리를 실행하면 Athena는 사용자가 지정한 쿼리 결과 위치에 쿼리 결과를 저장합니다. 이렇게 하면 쿼리 기록을 보고 쿼리 결과 세트를 다운로드하여 볼 수 있습니다.

이 단원에서는 다양한 SQL 문을 사용하여 공통 데이터 원본 및 데이터 유형에 대한 Athena 쿼리를 실행하는 방법에 대한 지침을 제공합니다. 일반적인 구조 및 연산자 작업(예: 배열, 연결, 필터링, 평면화 및 정렬 작업)에 대한 일반 지침이 제공됩니다. 다른 예로는 중첩 구조 및 맵이 있는 테이블의 데이터 쿼리, JSON으로 인코딩된 데이터 세트을 기반으로 하는 테이블, AWS CloudTrail 로그 및 Amazon EMR 로그와 같은 AWS 서비스와 관련된 데이터 세트이 있습니다. 표준 SQL 사용에 대한 포괄적인 내

용은 이 설명서에서 다루지 않습니다. SQL에 대한 자세한 내용은 Trino 및 Presto 언어 참조 사항을 참조하세요.

주제

- SQL 쿼리에 대한 실행 계획 보기
- 쿼리 결과, 최근 쿼리, 출력 파일 작업
- 쿼리 결과 재사용
- 완료된 쿼리에 대한 통계 및 실행 세부 정보 보기
- 뷰 작업
- 저장된 쿼리 사용
- 파라미터화된 쿼리 사용
- 비용 기반 최적화 프로그램 사용
- S3 Express One Zone 데이터 쿼리
- 복원된 Amazon S3 Glacier 객체 쿼리
- 스키마 업데이트 처리
- 배열 쿼리
- 지리 공간 데이터 쿼리
- JSON 쿼리
- Machine Learning(ML) with Amazon Athena 사용
- 사용자 정의 함수를 사용한 쿼리
- 리전 간 쿼리
- AWS Glue Data Catalog 쿼리
- AWS 서비스 로그 쿼리
- Amazon S3에 저장된 웹 서버 로그 쿼리

고려 사항 및 제한 사항은 Amazon Athena의 SQL 쿼리에 대한 고려 사항 및 제한 사항 단원을 참조하세요.

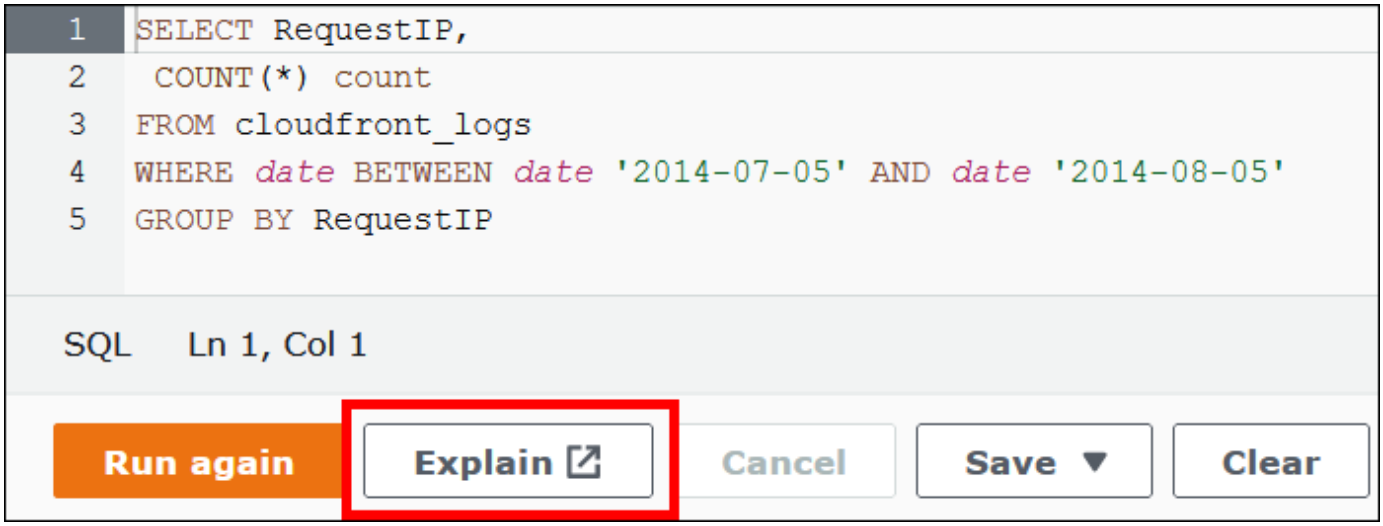
SQL 쿼리에 대한 실행 계획 보기

Athena 쿼리 편집기를 사용하여 쿼리가 실행되는 방식의 그래픽 표현을 확인할 수 있습니다. 편집기에서 쿼리를 입력하고 설명(Explain) 옵션을 선택할 경우, Athena는 쿼리에 EXPLAIN SQL 문을 사용하

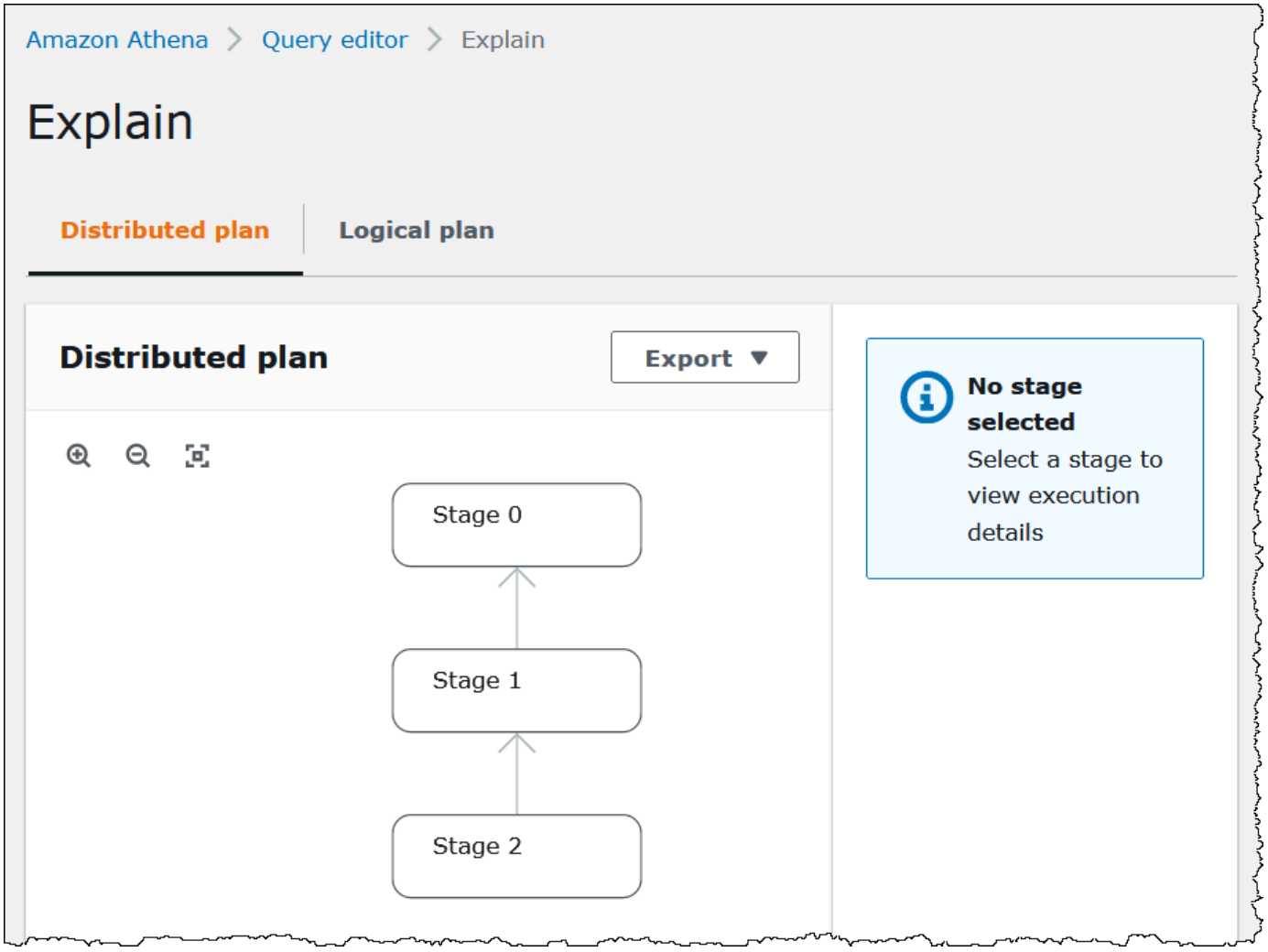
여 해당되는 두 그래프인 분산된 실행 계획 및 논리적 실행 계획을 만듭니다. 이러한 그래프를 사용하여 쿼리의 효율성을 분석, 문제 해결, 개선할 수 있습니다.

쿼리 실행 계획을 보려면

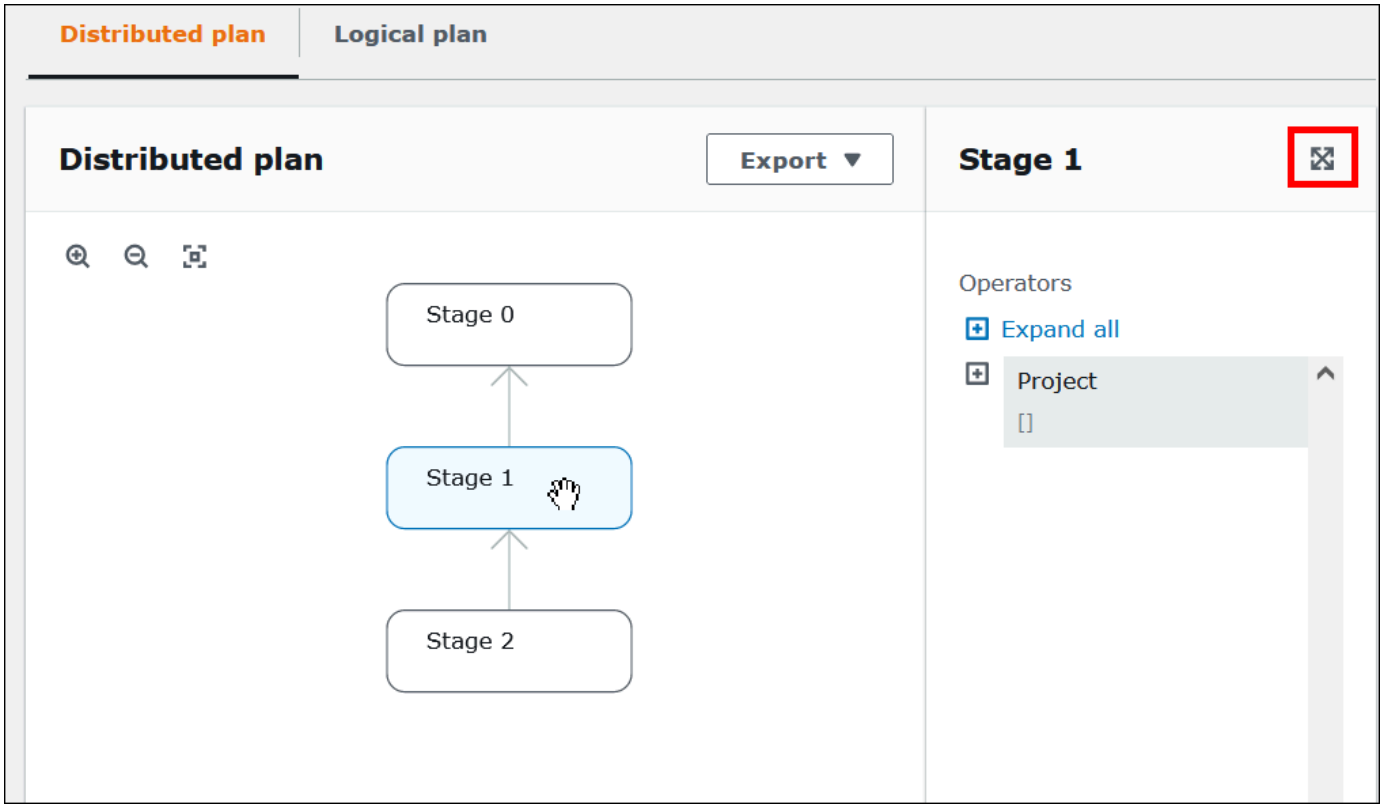
1. Athena 쿼리 편집기에서 쿼리를 입력한 다음 설명(Explain)을 선택합니다.



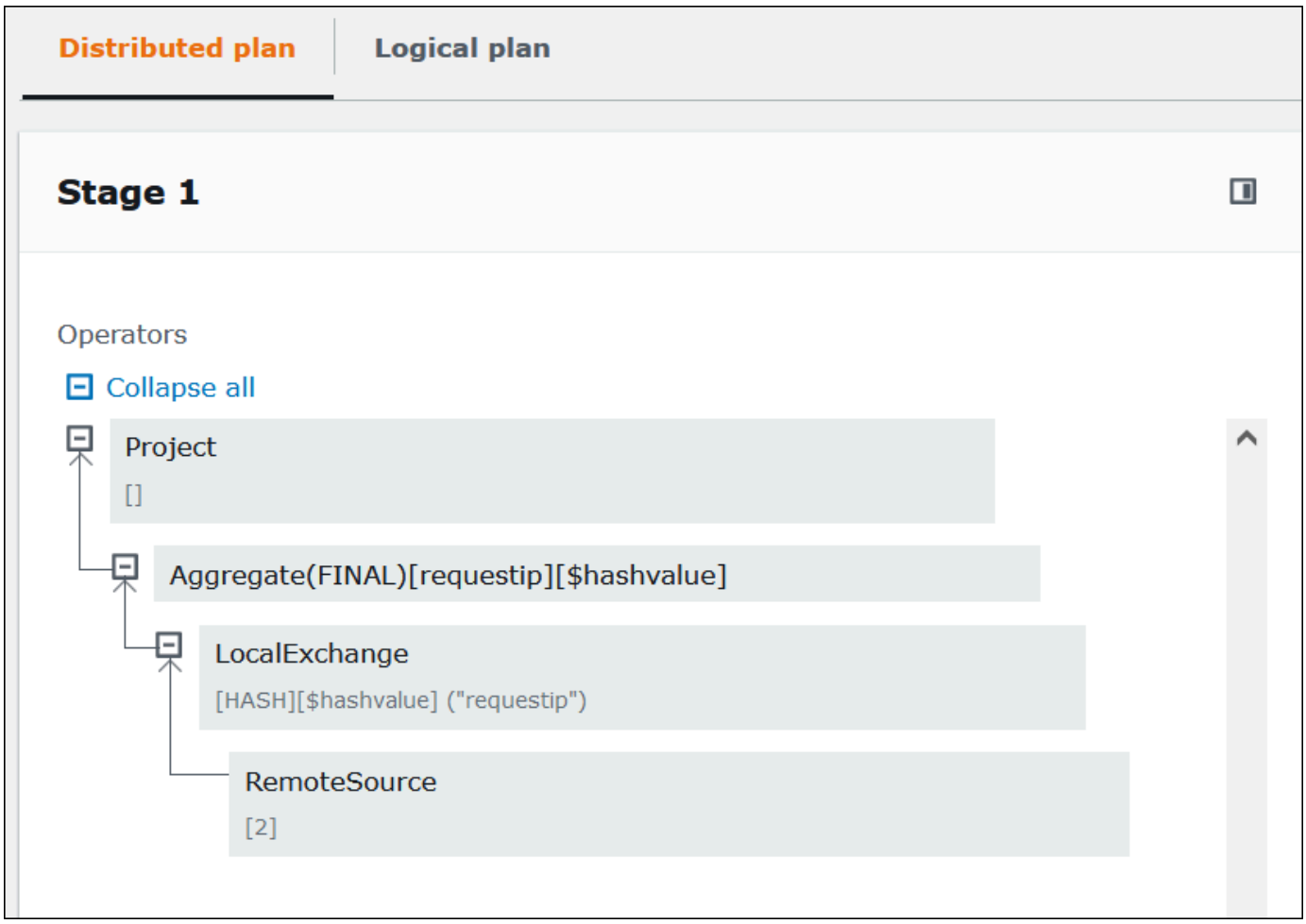
   배포된 계획(Distributed plan) 탭은 배포된 환경에서의 쿼리 실행 계획을 나타냅니다. 분산된 계획에는 처리 조각이나 단계가 있습니다. 각 단계는 0부터 시작되는 인덱스 번호가 있으며 하나 이상의 노드로 처리됩니다. 노드 간에는 데이터를 교환할 수 있습니다.



2. 그래프를 살펴보려면 다음 옵션을 사용합니다.
   - 마우스를 스크롤하여 확대 또는 축소하거나, 돋보기 아이콘을 사용합니다.
   - 화면에 맞게 그래프를 조정하려면 크기에 맞게 확대/축소(Zoom to fit) 아이콘을 선택합니다.
   - 그래프를 이동하려면 마우스 포인터를 드래그합니다.
3. 단계의 세부 정보를 보려면 단계를 선택합니다.

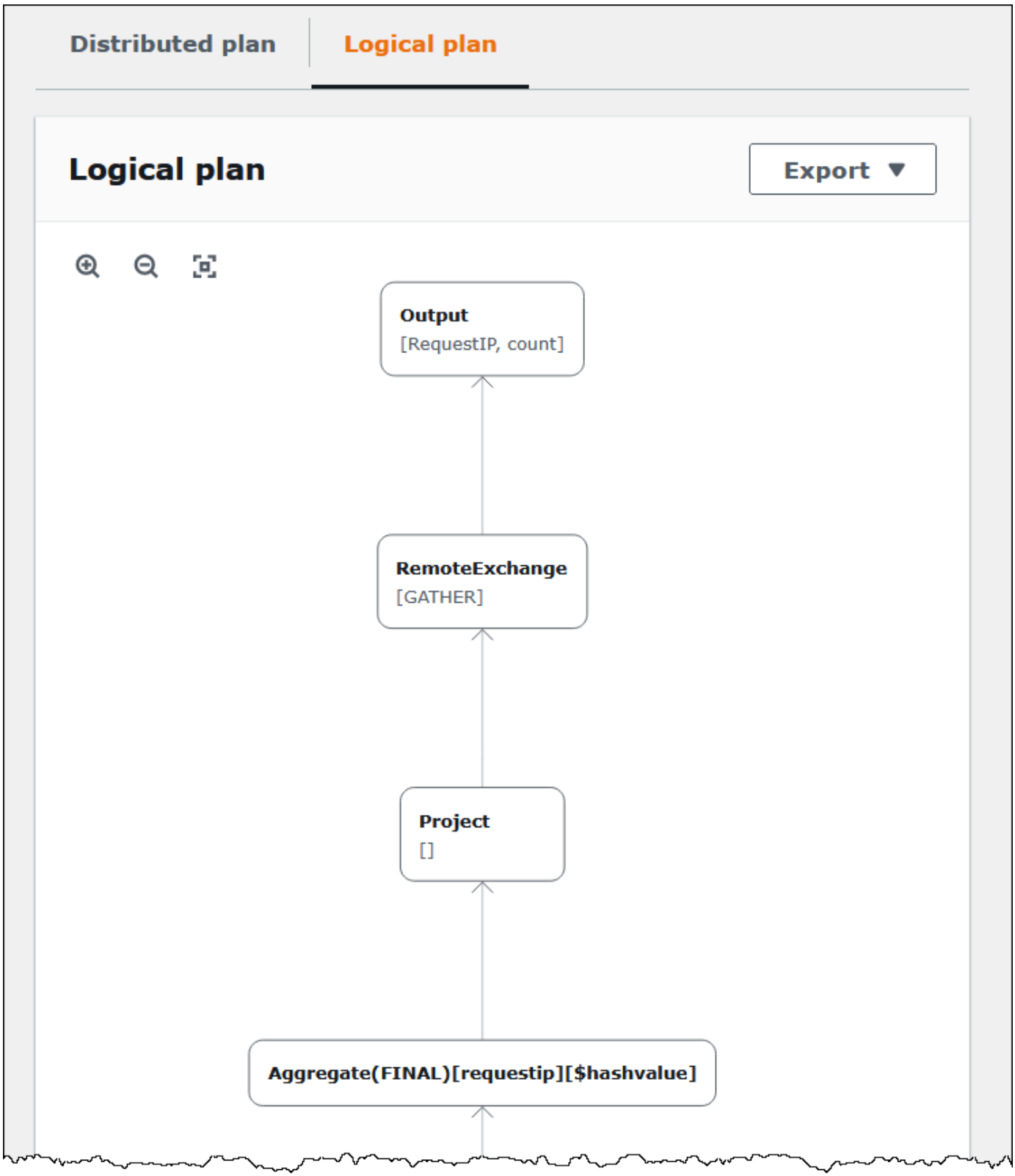


4. 단계 세부 정보를 전체 너비로 보려면 세부 정보 창의 오른쪽 상단에 있는 확장 아이콘을 선택합니다.
5. 자세한 내용을 보려면 연산자 트리에서 하나 이상의 항목을 확장합니다. 분산된 계획 조각에 대한 자세한 내용은 EXPLAIN 문 출력 유형 단원을 참조하세요.



> **Important**
>
> 현재 일부 파티션 필터는 Athena에서 쿼리에 적용하더라도 중첩 연산자 트리 그래프에 표시되지 않을 수도 있습니다. 이러한 필터의 효과를 확인하려면 쿼리에서 EXPLAIN 또는 EXPLAIN ANALYZE를 실행하고 결과를 봅니다.

6. 논리적 계획(Logical plan) 탭을 선택합니다. 그래프는 쿼리를 실행할 논리적 계획을 보여줍니다. 연산 용어에 대한 자세한 내용은 Athena EXPLAIN 문 결과의 이해 단원을 참조하세요.



7. 계획을 SVG 이미지나 PNG 이미지, 또는 JSON 텍스트로 내보내려면 내보내기(Export)를 선택합니다.

추가적인 리소스

자세한 정보는 다음 리소스를 참조하세요.

Athena에서 EXPLAIN 및 EXPLAIN ANALYZE 사용

Athena EXPLAIN 문 결과의 이해

완료된 쿼리에 대한 통계 및 실행 세부 정보 보기

쿼리 결과, 최근 쿼리, 출력 파일 작업

Amazon Athena는 Amazon S3에 지정된 쿼리 결과 위치에서 실행되는 각각의 쿼리에 대해 쿼리 결과 및 메타데이터 정보를 자동으로 저장합니다. 필요한 경우 이 위치의 파일에 액세스하여 해당 파일로 작업할 수 있습니다. 또한 Athena 콘솔에서 쿼리 결과 파일을 직접 다운로드할 수 있습니다.

Amazon S3 쿼리 결과 위치를 처음으로 설정하려면 Athena 콘솔을 사용하여 쿼리 결과 위치 지정 단원을 참조하세요.

실행되는 모든 쿼리에 대해 출력 파일은 자동으로 저장됩니다. Athena 콘솔을 사용해 쿼리 출력 파일을 액세스하고 보려면 IAM 보안 주체(사용자 및 역할)가 쿼리 결과 위치의 Amazon S3 GetObject 작업에 대한 권한뿐 아니라 Athena GetQueryResults 작업에 대한 권한도 가지고 있어야 합니다. 쿼리 결과 위치를 암호화할 수 있습니다. 위치가 암호화되는 경우 사용자는 쿼리 결과 위치를 암호화하고 암호화 해제할 수 있는 적절한 키 권한을 가지고 있어야 합니다.

> **Important**
>
> 쿼리 결과 위치의 Amazon S3 `GetObject` 작업에 대한 권한이 있는 IAM 보안 주체는 Athena `GetQueryResults` 작업에 대한 권한이 거부되더라도 Amazon S3에서 쿼리 결과를 검색할 수 있습니다.

쿼리 결과 위치 지정

Athena가 사용하는 쿼리 결과 위치는 작업 그룹 설정과 클라이언트 측 설정의 조합으로 결정됩니다. 클라이언트 측 설정은 쿼리를 실행하는 방식을 기반으로 합니다.

- Athena 콘솔을 사용하여 쿼리를 실행하는 경우 탐색 모음의 설정에 입력한 쿼리 결과 위치가 클라이언트 측 설정을 결정합니다.
- Athena API를 사용하여 쿼리를 실행하는 경우 StartQueryExecution 작업의 `OutputLocation` 파라미터가 클라이언트 측 설정을 결정합니다.

- ODBC 또는 JDBC 드라이버를 사용하여 쿼리를 실행하는 경우 연결 URL에 지정된 `S3OutputLocation` 속성에 따라 클라이언트 측 설정이 결정됩니다.

> **Important**
>
> API를 사용하거나 ODBC 또는 JDBC 드라이버를 사용하여 쿼리를 실행할 때는 콘솔 설정이 적용되지 않습니다.

각 작업 그룹 구성에는 활성화할 수 있는 Override client-side settings(클라이언트 측 설정 재정의) 옵션이 있습니다. 이 옵션을 활성화하면 해당 작업 그룹과 연결된 IAM 보안 주체가 쿼리를 실행할 때 작업 그룹 설정이 적용 가능한 클라이언트 측 설정보다 우선적으로 적용됩니다.

Athena 콘솔을 사용하여 쿼리 결과 위치 지정

쿼리를 실행하려면 먼저, Amazon S3에서 쿼리 결과 버킷 위치를 지정하거나, 버킷을 지정했고 구성이 클라이언트 설정을 재정의하는 작업 그룹을 사용해야 합니다.

Athena 콘솔을 사용하여 클라이언트 측 설정 쿼리 결과 위치를 지정하려면

1. 쿼리 결과 위치를 지정할 작업 그룹으로 전환합니다. 기본 작업 그룹의 이름은 Primary입니다.
2. 탐색 모음에서 설정을 선택합니다.
3. 탐색 모음에서 관리(Manage)를 선택합니다.
4. 설정 관리(Manage settings)에서 다음 중 하나를 수행합니다.
   - 쿼리 결과 위치(Location of query result) 상자에서 Amazon S3에서 쿼리 결과를 위해 생성한 버킷의 경로를 입력합니다. 경로 앞에 `s3://`를 붙입니다.
   - S3 검색(Browse S3) 아이콘을 선택하고 현재 리전에서 생성한 Amazon S3 버킷을 선택한 다음 선택(Choose)을 선택합니다.

   > **Note**
   >
   > 작업 그룹의 모든 사용자에 대해 쿼리 결과 위치를 지정하는 작업 그룹을 사용하는 경우 쿼리 결과 위치를 변경하는 옵션을 사용할 수 없습니다.

5. (선택 사항) View lifecycle configuration(수명 주기 구성 보기)를 선택하여 쿼리 결과 버킷에서 Amazon S3 수명 주기 규칙을 보고 구성합니다. 생성하는 Amazon S3 수명 주기 규칙은 만료 규

칙 또는 전환 규칙일 수 있습니다. 만료 규칙은 일정 시간이 지나면 쿼리 결과를 자동으로 삭제합니다. 전환 규칙은 쿼리 결과를 다른 Amazon S3 스토리지 계층으로 이동합니다. 자세한 내용은 Amazon Simple Storage Service Console 사용 설명서의 버킷에서 수명 주기 구성 설정을 참조하세요.

6. (선택 사항) 예상 버킷 소유자(Expected bucket owner)에 출력 위치 버킷의 소유자가 될 것으로 예상되는 AWS 계정의 ID를 입력합니다. 이는 추가 보안 조치입니다. 버킷 소유자의 계정 ID가 여기에서 지정한 ID와 일치하지 않으면 버킷으로의 출력 시도가 실패합니다. 자세한 내용은 Amazon S3 사용 설명서의 버킷 소유자 조건을 사용하여 버킷 소유권 확인을 참조하세요.

   > **Note**
   >
   > 예상 버킷 소유자 설정은 Athena 쿼리 결과에 대해 지정한 Amazon S3 출력 위치에만 적용됩니다. 외부 Amazon S3 버킷의 데이터 소스 위치, `CTAS` 및 `INSERT INTO` 대상 테이블 위치, `UNLOAD` 문 출력 위치, 연합 쿼리의 유출 버킷 작업 또는 다른 계정의 테이블에 대해 실행되는 `SELECT` 쿼리 등의 다른 Amazon S3 위치에는 적용되지 않습니다.

7. (선택 사항) Amazon S3 저장된 쿼리 결과를 암호화하려면 쿼리 결과 암호화(Encrypt query results)를 선택하세요. Athena에서 암호화 사용에 대한 자세한 내용은 저장 중 암호화 섹션을 참조하세요.

8. (선택 사항) 쿼리 결과 버킷에 대해 ACL이 사용 설정되어 있을 때 버킷 소유자에게 쿼리 결과에 대한 전체 제어 액세스 권한을 부여하려면 버킷 소유자에게 쿼리 결과에 대한 전체 제어 할당을 선택하세요. 예를 들어, 쿼리 결과 위치가 다른 계정이 소유한 경우 쿼리 결과에 대한 소유권 및 전체 제어 권한을 다른 계정에 부여할 수 있습니다. 자세한 내용은 Amazon S3 사용자 안내서의 객체 소유권 제어 및 버킷에 대해 ACL 사용 중지를 참조하세요.

9. Save(저장)를 선택합니다.

이전에 생성한 기본 위치

이전에 Athena에서는 쿼리 결과 위치(Query result location)의 값을 지정하지 않은 상태로 쿼리를 실행하고 작업 그룹이 쿼리 결과 위치 설정을 재정의하지 않는 경우 Athena에서 기본 위치를 자동으로 생성했습니다. 기본 위치는 `aws-athena-query-results-`*`MyAcctID`*`-`*`MyRegion`*이었습니다. 여기서 *`MyAcctID`*는 쿼리를 실행한 IAM 보안 주체의 Amazon Web Services 계정 ID이고 *`MyRegion`*은 쿼리가 실행된 리전(예: `us-west-1`)입니다.

이제 이전에 계정이 Athena를 사용한 적이 없는 리전에서 Athena 쿼리를 실행하려면 먼저, 쿼리 결과 위치를 지정하거나, 쿼리 결과 위치 설정을 재정의하는 작업 그룹을 사용해야 합니다. Athena에서 더

이상 기본 쿼리 결과 위치를 자동으로 생성하지 않지만, 이전에 생성된 기본 `aws-athena-query-results-MyAcctID-MyRegion` 위치는 유효한 상태로 유지되며 계속 사용할 수 있습니다.

작업 그룹을 사용하여 쿼리 결과 위치 지정

AWS Management Console, AWS CLI 또는 Athena API를 사용하여 작업 그룹 구성에서 쿼리 결과 위치를 지정합니다.

AWS CLI를 사용할 경우 `aws athena create-work-group` 또는 `aws athena update-work-group` 명령을 실행할 때 `--configuration` 옵션의 `OutputLocation` 파라미터를 사용하여 쿼리 결과 위치를 지정합니다.

Athena 콘솔을 사용하여 작업 그룹에 대한 쿼리 결과 위치를 지정하려면

1. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

2. 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
3. 작업 그룹 목록에서 편집할 작업 그룹의 링크를 선택합니다.
4. 편집을 선택합니다.
5. 쿼리 결과 위치 및 암호화에 대해 다음 중 하나를 수행합니다.
   - 쿼리 결과 위치(Location of query result) 상자에 쿼리 결과에 대한 Amazon S3의 버킷 경로를 입력합니다. 경로 앞에 `s3://`를 붙입니다.
   - S3 검색(Browse S3)을 선택하고 사용하고 싶은 현재 리전에 대한 Amazon S3 버킷을 선택한 다음 선택(Choose)을 선택합니다.

6. (선택 사항) 예상 버킷 소유자(Expected bucket owner)에 출력 위치 버킷의 소유자가 될 것으로 예상되는 AWS 계정의 ID를 입력합니다. 이는 추가 보안 조치입니다. 버킷 소유자의 계정 ID가 여기에서 지정한 ID와 일치하지 않으면 버킷으로의 출력 시도가 실패합니다. 자세한 내용은 Amazon S3 사용 설명서의 버킷 소유자 조건을 사용하여 버킷 소유권 확인을 참조하세요.

   > **Note**
   >
   > 예상 버킷 소유자 설정은 Athena 쿼리 결과에 대해 지정한 Amazon S3 출력 위치에만 적용됩니다. 외부 Amazon S3 버킷의 데이터 소스 위치, `CTAS` 및 `INSERT INTO` 대상 테이블 위치, `UNLOAD` 문 출력 위치, 연합 쿼리의 유출 버킷 작업 또는 다른 계정의 테이블에 대해 실행되는 `SELECT` 쿼리 등의 다른 Amazon S3 위치에는 적용되지 않습니다.

7. (선택 사항) Amazon S3 저장된 쿼리 결과를 암호화하려면 쿼리 결과 암호화(Encrypt query results)를 선택하세요. Athena에서 암호화 사용에 대한 자세한 내용은 저장 중 암호화 섹션을 참조하세요.

8. (선택 사항) 쿼리 결과 버킷에 대해 ACL이 사용 설정되어 있을 때 버킷 소유자에게 쿼리 결과에 대한 전체 제어 액세스 권한을 부여하려면 버킷 소유자에게 쿼리 결과에 대한 전체 제어 할당을 선택하세요. 예를 들어, 쿼리 결과 위치가 다른 계정이 소유한 경우 쿼리 결과에 대한 소유권 및 전체 제어 권한을 다른 계정에 부여할 수 있습니다.

   버킷의 S3 객체 소유권 설정이 버킷 소유자 기본인 경우 버킷 소유자는 이 작업 그룹에서 작성된 모든 쿼리 결과 객체도 소유합니다. 예를 들어, 외부 계정의 작업 그룹에서 이 옵션을 활성화하고 쿼리 결과 위치를 계정의 버킷 소유자 기본의 S3 객체 소유권 설정을 가진 Amazon S3 버킷으로 설정한 경우, 외부 작업 그룹의 쿼리 결과에 대한 전체 제어 권한을 갖습니다.

   쿼리 결과 버킷의 S3 객체 소유권 설정이 버킷 소유자 시행인 경우 이 옵션은 아무 영향이 없습니다. 자세한 내용은 Amazon S3 사용자 안내서의 객체 소유권 제어 및 버킷에 대해 ACL 사용 중지를 참조하세요.

9. 작업 그룹의 모든 사용자가 지정한 쿼리 결과 위치를 사용하도록 요청하려면 설정(Settings) 섹션까지 아래로 스크롤하고 고객 측 설정 재정의(Override client-side settings)를 선택합니다.

10. Save changes(변경 사항 저장)를 선택합니다.

Athena 콘솔을 사용하여 쿼리 결과 파일 다운로드

쿼리를 실행한 후 즉시 쿼리 창에서 쿼리 결과 CSV 파일을 다운로드할 수 있습니다. 또한 최근 쿼리(Recent queries) 탭에서 최근 쿼리의 쿼리 결과를 다운로드할 수 있습니다.

> **Note**
>
> Athena 쿼리 결과 파일은 개별 사용자가 구성할 수 있는 정보가 포함된 데이터 파일입니다. 이 데이터를 읽고 분석하는 일부 프로그램은 해당 데이터의 일부를 명령으로 해석할 가능성이 있습니다(CSV 주입). 이러한 이유로 쿼리 결과 CSV 데이터를 스프레드시트 프로그램으로 가져올 때 해당 프로그램에서 보안 문제에 대한 경고가 표시될 수 있습니다. 시스템을 안전하게 유지하려면 다운로드된 쿼리 결과에서 링크나 매크로를 사용하지 않도록 설정해야 합니다.

쿼리 실행 및 쿼리 결과 다운로드

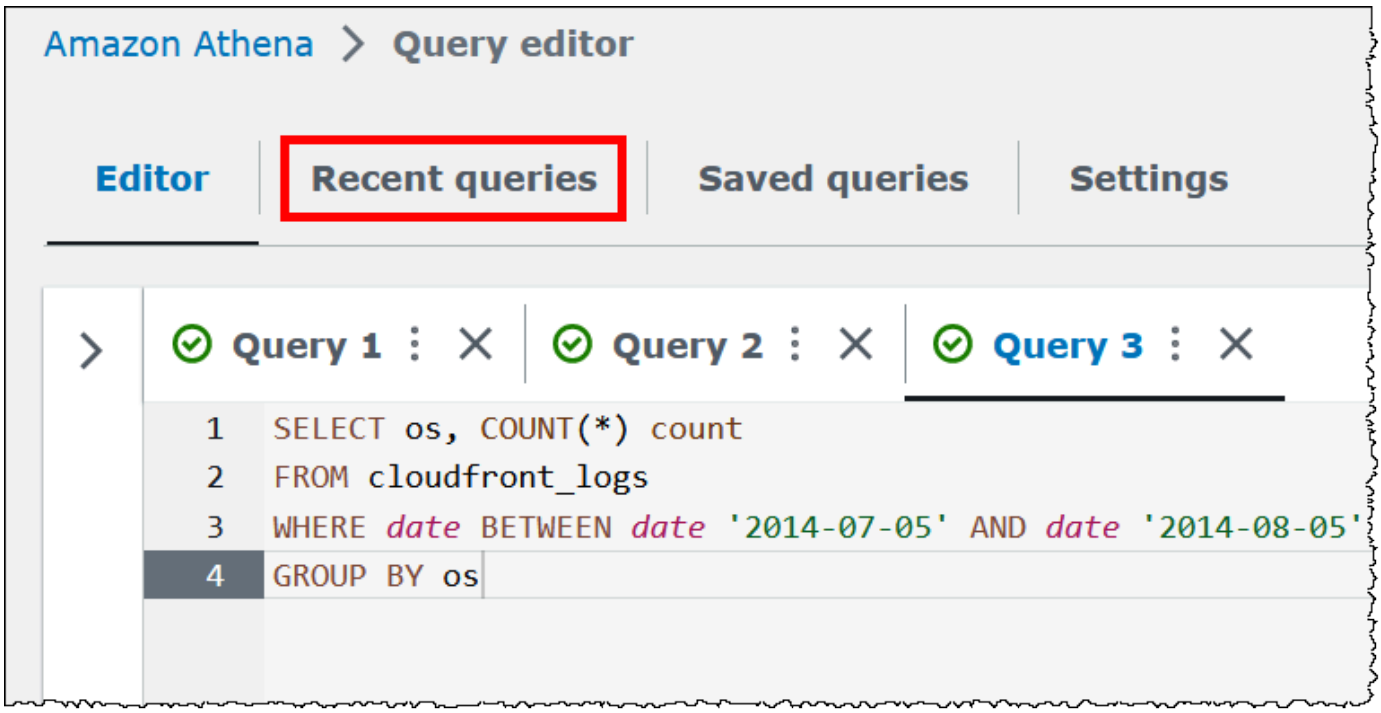
1. 쿼리 편집기에서 쿼리를 입력한 다음 실행(Run)을 선택합니다.

   쿼리 실행이 완료되면 Results(결과) 창에 쿼리 결과가 표시됩니다.

2. 쿼리 결과의 CSV 파일을 다운로드하려면 쿼리 결과 창 위에서 결과 다운로드(Download results)를 선택합니다. 사용 중인 브라우저 및 브라우저 구성에 따라 다운로드를 확인해야 할 수 있습니다.

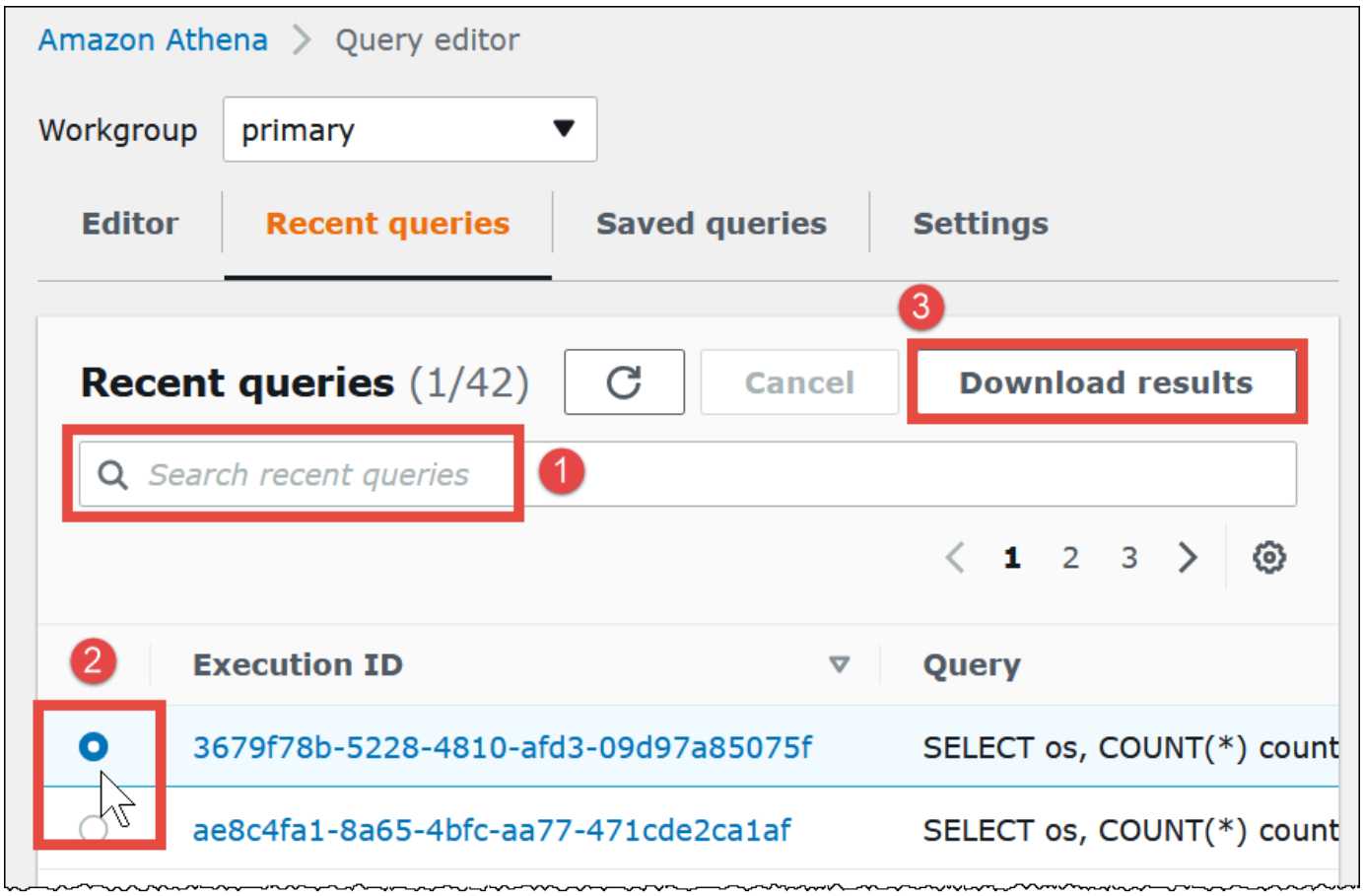
이전 쿼리에 대한 쿼리 결과 파일을 다운로드하려면

1. 최근 쿼리(Recent queries)를 선택합니다.



2. 쿼리 상자를 사용하여 쿼리를 찾고 쿼리를 선택한 다음 결과 다운로드(Download results)를 선택합니다.

> **Note**
>
> Download results(결과 다운로드) 옵션을 사용하여 수동으로 삭제된 쿼리 결과를 검색하거나, Amazon S3 수명 주기 규칙에 의해 삭제되었거나 다른 위치로 이동된 쿼리 결과를 검색할 수 없습니다.

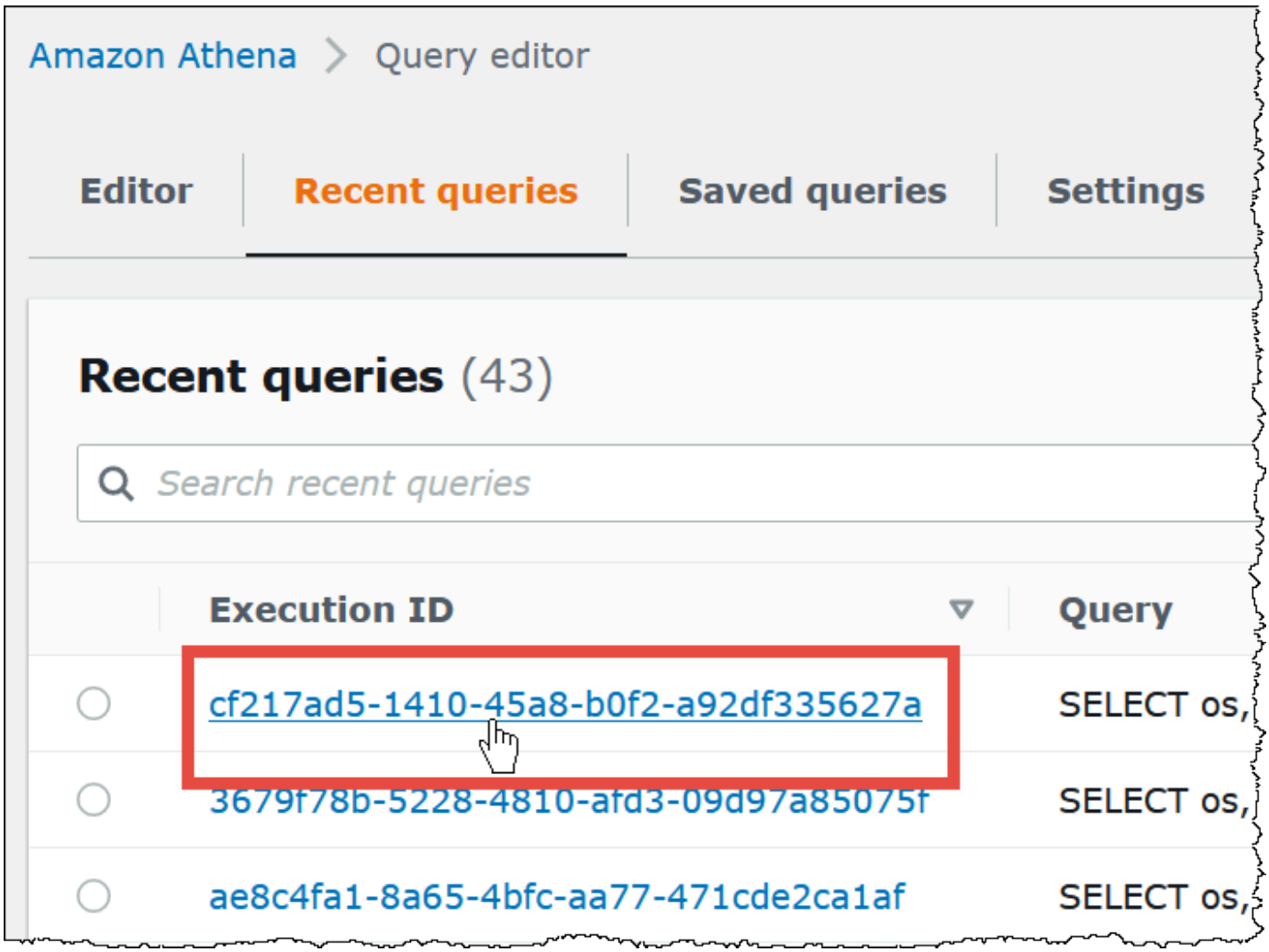


최근 쿼리 보기

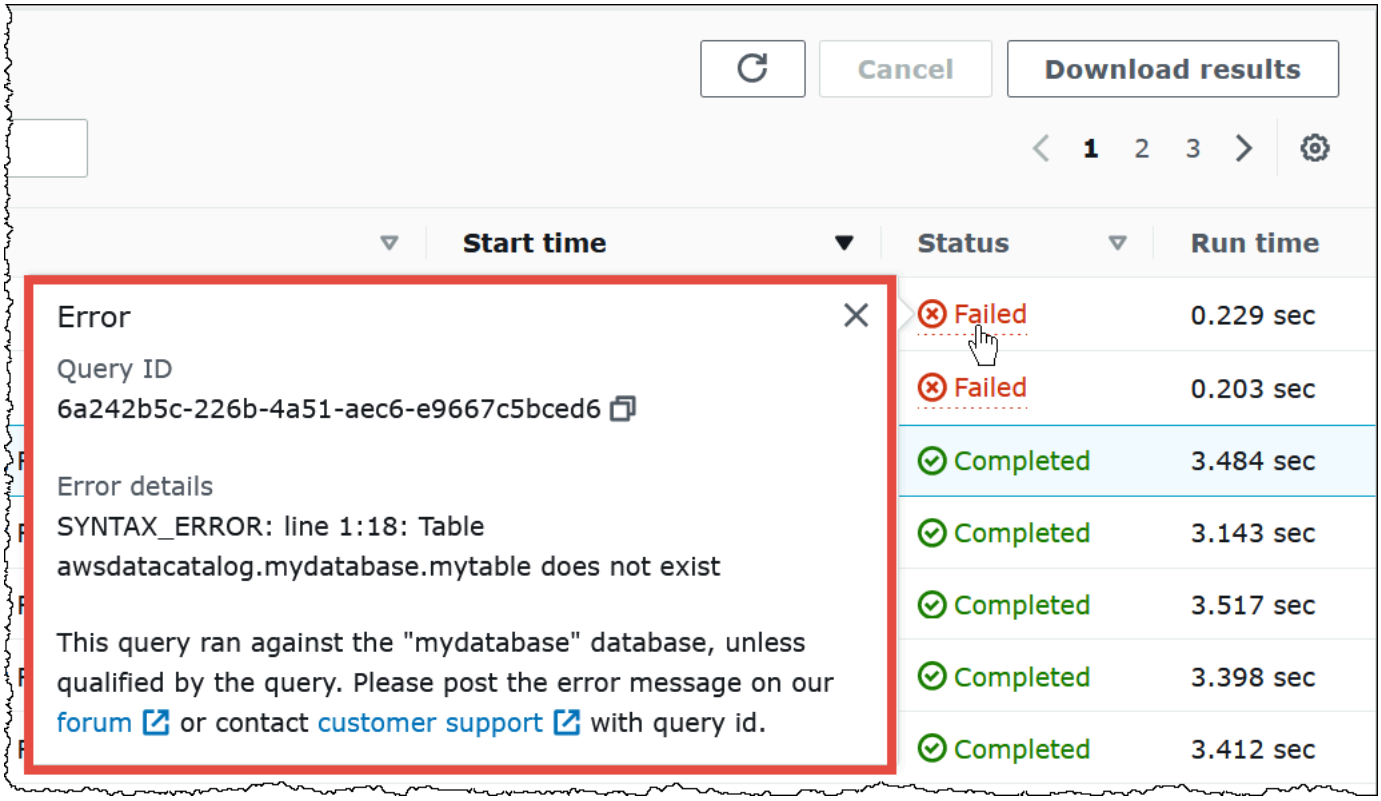
Athena 콘솔을 사용하여 성공 또는 실패한 쿼리를 확인하고 실패한 쿼리에 대한 오류 세부 정보를 볼 수 있습니다. Athena는 45일 동안 쿼리 기록을 보존합니다.

Athena 콘솔에서 최근 쿼리 보기

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 최근 쿼리(Recent queries)를 선택합니다. 최근 쿼리(Recent queries) 탭에는 실행된 각 쿼리에 대한 정보가 표시됩니다.
3. 쿼리 편집기에서 쿼리 문을 열려면 쿼리의 실행 ID를 선택합니다.



4. 실패한 쿼리에 대한 세부 정보를 보려면 쿼리에 대한 실패(Failed) 링크를 선택합니다.

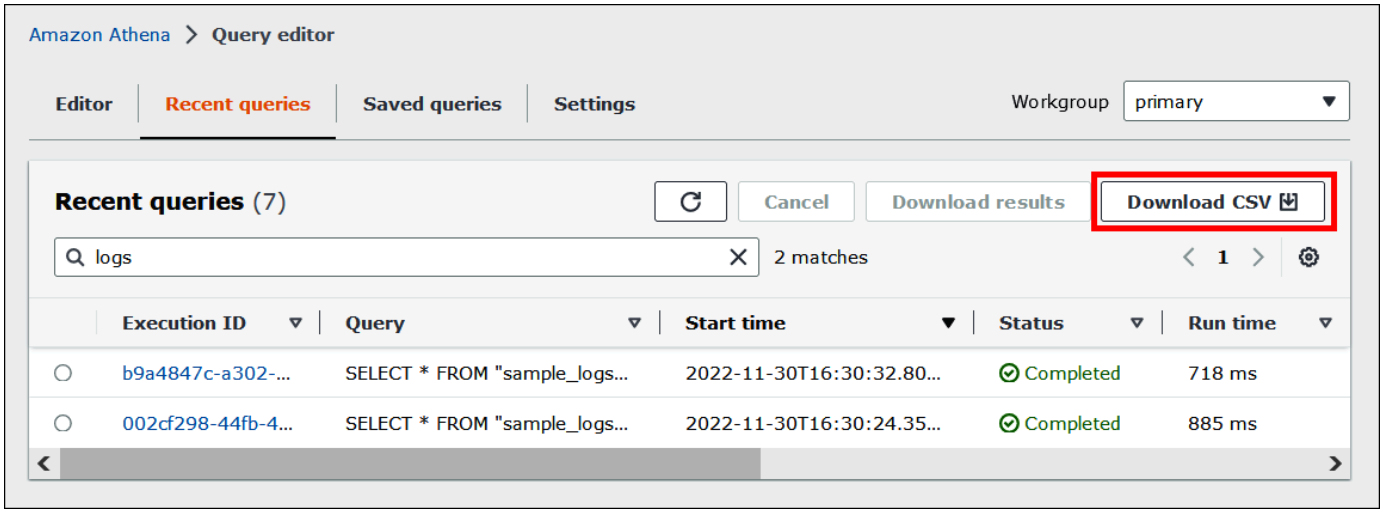


여러 개의 최근 쿼리를 CSV 파일로 다운로드

Athena 콘솔의 Recent queries(최근 쿼리) 탭을 사용하여 하나 이상의 최근 쿼리를 CSV 파일로 내보내 테이블 형식으로 볼 수 있습니다. 다운로드한 파일에는 쿼리 결과가 아니라 SQL 쿼리 문자열 자체와 쿼리에 대한 기타 정보가 들어 있습니다. 내보낸 필드에는 실행 ID, 쿼리 문자열 내용, 쿼리 시작 시간, 상태, 실행 시간, 스캔한 데이터의 양, 사용된 쿼리 엔진 버전 및 암호화 방법이 포함됩니다. 검색 상자에 입력한 조건을 사용하여 최대 500개의 최근 쿼리 또는 필터링된 쿼리를 내보낼 수 있습니다.

하나 이상의 최근 쿼리를 CSV 파일로 내보내려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 최근 쿼리(Recent queries)를 선택합니다.
3. (선택 사항) 검색 상자를 사용하여 다운로드할 최근 쿼리를 필터링합니다.
4. Download CSV를 선택합니다.



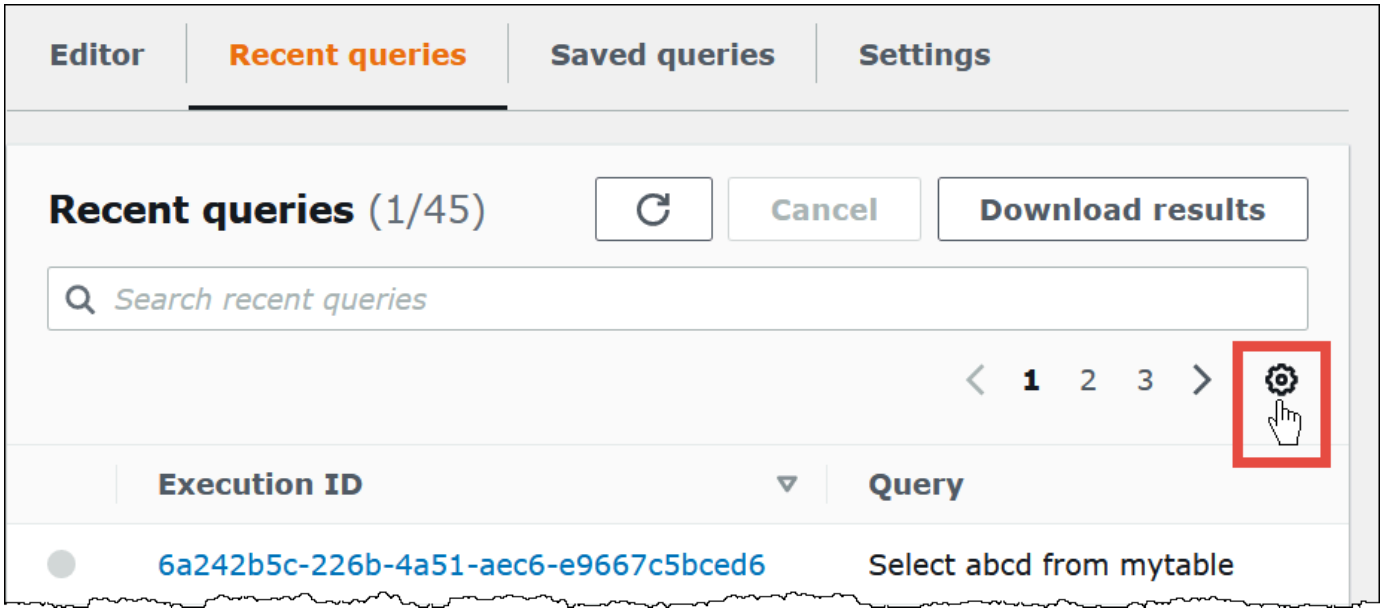
5. 파일 저장 프롬프트에서 Save(저장)를 선택합니다. 기본 파일 이름은 `Recent Queries`이며 뒤에 타임스탬프가 옵니다(예:`Recent Queries 2022-12-05T16 04 27.352-08 00.csv`)

최근 쿼리 표시 옵션 구성

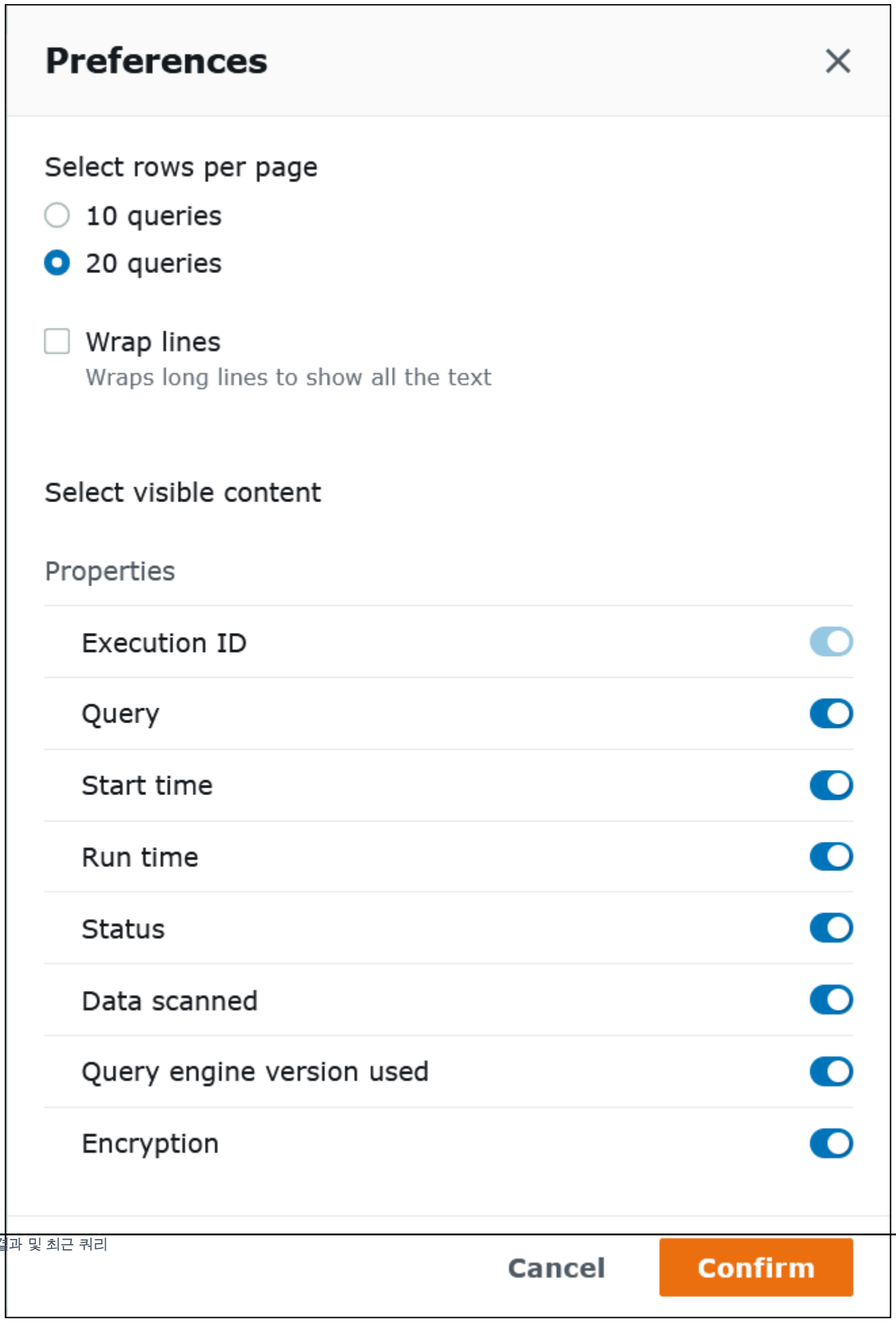
표시할 열 및 자동 줄 바꿈과 같이 Recent queries(최근 쿼리) 탭에 대한 옵션을 구성할 수 있습니다.

Recent queries(최근 쿼리) 탭에 대한 옵션을 구성하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 최근 쿼리(Recent queries)를 선택합니다.
3. 옵션 버튼(기어 모양 아이콘)을 선택합니다.



4. 기본 설정(Preferences) 대화 상자에서 페이지당 행 수, 줄 바꿈 동작 및 표시할 열을 선택합니다.


Preferences
Select rows per page
10 queries
20 queries
Wrap lines
Wraps long lines to show all the text
Select visible content
Properties
Execution ID
Query
Start time
Run time
Status
Data scanned
Query engine version used
Encryption
Cancel
Confirm

5. 확인을 선택합니다.

쿼리 기록을 45일 이상 유지

쿼리 기록을 45일 이상 보존하려면 쿼리 기록을 검색하여 Amazon S3와 같은 데이터 스토어에 저장할 수 있습니다. 이 프로세스를 자동화하려면 Athena 및 Amazon S3 API 작업과 CLI 명령을 사용할 수 있습니다. 다음 절차는 이러한 단계를 요약합니다.

프로그래밍 방식으로 쿼리 기록을 검색하고 저장하려면

1. Athena ListQueryExecutions API 작업 또는 list-query-executions CLI 명령을 사용하여 쿼리 ID를 검색합니다.
2. Athena GetQueryExecution API 작업 또는 get-query-execution CLI 명령을 사용하여 해당 ID를 기반으로 각 쿼리에 대한 정보를 검색합니다.
3. Amazon S3 PutObject API 작업 또는 put-object CLI 명령을 사용하여 Amazon S3에 정보를 저장합니다.

Amazon S3에서 쿼리 출력 파일 찾기

구성이 클라이언트 측 설정을 재정의하는 작업 그룹에서 쿼리가 발생하지 않는 한 쿼리 출력 파일은 다음 경로 패턴으로 Amazon S3의 하위 폴더에 저장됩니다. 작업 그룹 구성이 클라이언트 측 설정을 재정의하는 경우 쿼리는 작업 그룹에서 지정한 결과 경로를 사용합니다.

```
QueryResultsLocationInS3/[QueryName|Unsaved/yyyy/mm/dd/]
```

- *QueryResultsLocationInS3*는 작업 그룹 설정 또는 클라이언트 측 설정에 따라 지정된 쿼리 결과 위치입니다. 자세한 내용은 이 문서의 후반부에서 the section called “쿼리 결과 위치 지정” 단원을 참조하세요.
- 다음 하위 폴더는 작업 그룹 구성에 의해 결과 경로가 재정의되지 않은 콘솔에서 실행되는 쿼리에 대해서만 생성됩니다. AWS CLI에서 실행되거나 Athena API를 사용하는 쿼리는 *QueryResultsLocationInS3*에 직접 저장됩니다.
  - *QueryName*은 결과가 저장되는 쿼리의 이름입니다. 쿼리가 실행되었지만 저장되지 않은 경우 Unsaved가 사용됩니다.
  - *yyyy/mm/dd*는 쿼리가 실행된 날짜입니다.

CREATE TABLE AS SELECT 쿼리와 연결된 파일은 위 패턴의 tables 하위 폴더에 저장됩니다.

쿼리 출력 파일 식별

파일은 쿼리 이름, 쿼리 ID, 쿼리가 실행된 날짜를 기반으로 Amazon S3의 쿼리 결과 위치에 저장됩니다. 각 쿼리에 대한 파일은 *QueryID*를 사용하여 이름 지정됩니다. QueryID는 쿼리가 실행될 때 Athena가 각 쿼리에 할당하는 고유 식별자입니다.

다음 파일 형식이 저장됩니다.

파일 유형	파일 이름 지정 패턴	설명
쿼리 결과 파일	*QueryID*.csv *QueryID*.txt	DML 쿼리 결과 파일은 CSV(쉼표로 분리된 값) 형식으로 저장됩니다. DDL 쿼리 결과는 일반 텍스트 파일로 저장됩니다. 콘솔을 사용할 경우 콘솔의 결과 창에서 또는 쿼리 기록에서 결과 파일을 다운로드할 수 있습니다. 자세한 내용은 Athena 콘솔을 사용하여 쿼리 결과 파일 다운로드 단원을 참조하세요.
쿼리 메타데이터 파일	*QueryID*.csv.metadata *QueryID*.txt.metadata	DML 및 DDL 쿼리 메타데이터 파일은 이진 형식으로 저장되며 사람이 읽을 수 없습니다. 파일 확장명은 관련된 쿼리 결과 파일과 같습니다. Athena는 `GetQueryResults` 작업을 사용해 쿼리 결과를 읽을 때 메타데이터를 사용합니다. 이러한 파일을 삭제할 수 있지만, 쿼리에 대한 중요한 정보가 손실될 수 있기 때문에 파일을 삭제하지 않는 것이 좋습니다.

파일 유형	파일 이름 지정 패턴	설명
데이터 매니페스트 파일	*QueryID*-manifest.csv	INSERT INTO 쿼리가 실행될 때 Athena가 Amazon S3 데이터 원본 위치에서 생성하는 파일을 추적하기 위해 데이터 매니페스트 파일이 생성됩니다. 쿼리가 실패하는 경우 매니페스트는 해당 쿼리가 작성하려고 한 파일도 추적합니다. 매니페스트는 실패한 쿼리에서 발생하는 분리된 파일을 식별하는 데 유용합니다.

AWS CLI를 사용하여 쿼리 출력 위치 및 파일 식별

AWS CLI를 사용하여 쿼리 출력 위치 및 결과 파일을 식별하려면 다음 예제처럼 `aws athena get-query-execution` 명령을 실행합니다. *abc1234d-5efg-67hi-jklm-89n0op12qr34*를 쿼리 ID로 교체합니다.

```
aws athena get-query-execution --query-execution-id abc1234d-5efg-67hi-
jklm-89n0op12qr34
```

이 명령은 다음과 비슷한 출력을 반환합니다. 각 출력 파라미터에 대한 설명은 AWS CLI 명령 참조의 get-query-execution을 참조하세요.

```
{
    "QueryExecution": {
        "Status": {
            "SubmissionDateTime": 1565649050.175,
            "State": "SUCCEEDED",
            "CompletionDateTime": 1565649056.6229999
        },
        "Statistics": {
            "DataScannedInBytes": 5944497,
            "DataManifestLocation": "s3://DOC-EXAMPLE-BUCKET/athena-query-
results-123456789012-us-west-1/MyInsertQuery/2019/08/12/abc1234d-5efg-67hi-
jklm-89n0op12qr34-manifest.csv",
            "EngineExecutionTimeInMillis": 5209
```

```
        },
        "ResultConfiguration": {
            "EncryptionConfiguration": {
                "EncryptionOption": "SSE_S3"
            },
            "OutputLocation": "s3://DOC-EXAMPLE-BUCKET/athena-query-
results-123456789012-us-west-1/MyInsertQuery/2019/08/12/abc1234d-5efg-67hi-
jklm-89n0op12qr34"
        },
        "QueryExecutionId": "abc1234d-5efg-67hi-jklm-89n0op12qr34",
        "QueryExecutionContext": {},
        "Query": "INSERT INTO mydb.elb_log_backup SELECT * FROM mydb.elb_logs LIMIT
 100",
        "StatementType": "DML",
        "WorkGroup": "primary"
    }
}
```

쿼리 결과 재사용

Athena에서 쿼리를 다시 실행할 때 마지막 저장된 쿼리 결과를 재사용하도록 선택할 수 있습니다. 이 옵션을 선택하면 스캔되는 바이트 수의 측면에서 성능이 향상되고 비용이 절감됩니다. 예를 들어 지정된 기간 내에 결과가 변경되지 않을 것을 알고 있는 경우 쿼리 결과를 재사용하는 것이 좋습니다. 쿼리 결과 재사용을 위한 최대 수명을 지정할 수 있습니다. Athena에서는 지정한 기간보다 오래되지 않는 한 저장된 결과를 사용합니다. 자세한 내용은 AWS 빅 데이터 블로그의 Reduce cost and improve query performance with Amazon Athena를 참조하세요.

> **Note**
>
> 쿼리 결과 재사용 기능에는 Athena 엔진 버전 3이 필요합니다. 엔진 버전 변경에 대한 자세한 내용은 Athena 엔진 버전 변경 섹션을 참조하세요.

주요 기능

- 쿼리 결과 재사용은 쿼리별 옵트인 기능입니다. 쿼리별로 쿼리 결과 재사용을 활성화할 수 있습니다.
- 쿼리 결과 재사용을 위한 최대 수명을 분, 시간 또는 일 단위로 지정할 수 있습니다. 지정 가능한 최대 수명은 사용된 시간 단위에 상관없이 7일입니다. 기본값은 60분입니다.

- 쿼리에 대한 결과 재사용을 활성화하면 Athena에서 동일한 작업 그룹 내에서 이전에 실행된 쿼리를 찾습니다. 해당하는 저장된 쿼리 결과를 찾으면 쿼리를 다시 실행하지 않고 이전 결과 위치를 가리키거나 해당 위치에서 데이터를 가져옵니다.

- 결과 재사용 옵션을 활성화하는 쿼리에 대해 Athena에서는 다음 조건에 모두 해당하는 경우에만 작업 그룹 폴더에 저장된 마지막 쿼리 결과를 재사용합니다.
  - 쿼리 문자열이 정확히 일치합니다.
  - 데이터베이스 및 카탈로그 이름이 일치합니다.
  - 이전 결과가 지정된 최대 수명보다 오래되지 않았거나 최대 수명이 지정되지 않은 경우 60분 미만입니다.
  - Athena에서는 현재 실행과 정확히 동일한 결과 구성을 가진 실행만 재사용합니다.
  - 쿼리에서 참조되는 모든 테이블에 액세스할 수 있습니다.
  - 이전 결과가 저장된 S3 파일 위치에 액세스할 수 있습니다.

이러한 조건 중 하나라도 충족되지 않는 경우 Athena에서는 캐시된 결과를 사용하지 않고 쿼리를 실행합니다.

고려 사항 및 제한

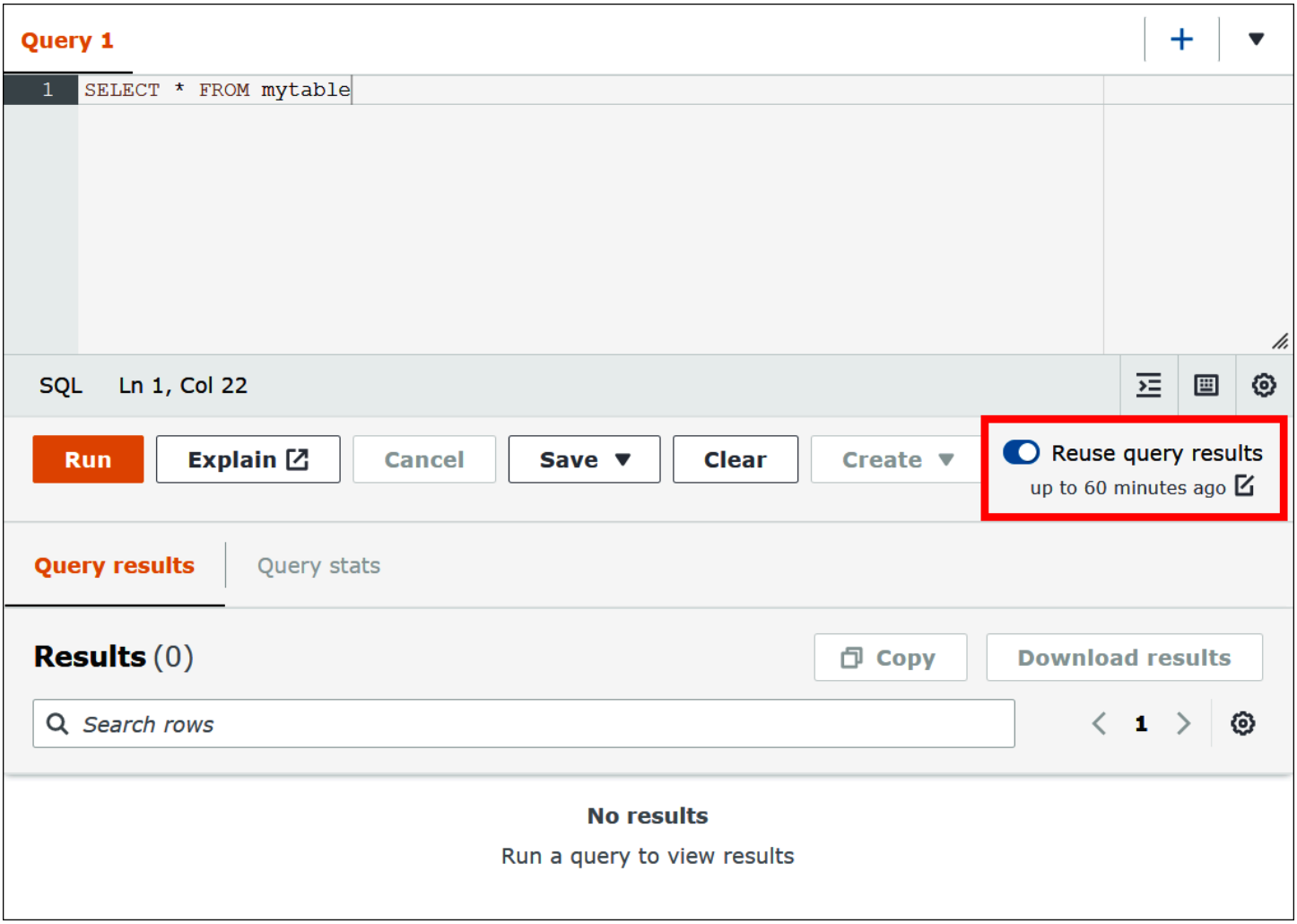
쿼리 결과 재사용 기능을 사용할 경우 다음 사항에 유의하세요.

- Athena에서는 동일한 작업 그룹 내에서만 쿼리 결과를 재사용합니다.
- 쿼리 결과 재사용 기능은 작업 그룹 구성을 준수합니다. 쿼리에 대한 결과 구성을 재정의하면 기능이 비활성화됩니다.
- AWS Glue에 등록된 Apache Hive, Apache Hudi, Apache Iceberg, Linux Foundation Delta Lake 테이블이 지원됩니다. 외부 Hive 메타스토어는 지원되지 않습니다.
- 페더레이션 카탈로그 또는 외부 Hive 메타스토어를 참조하는 쿼리는 지원되지 않습니다.
- Lake Formation 제어 테이블에서는 쿼리 결과 재사용이 지원되지 않습니다.
- 테이블 소스의 Amazon S3 위치가 Lake Formation의 데이터 위치로 등록된 경우 쿼리 결과 재사용은 지원되지 않습니다.
- 행 및 열 권한이 있는 테이블은 지원되지 않습니다.
- 세분화된 액세스 제어(예: 열 또는 행 필터링)가 있는 테이블은 지원되지 않습니다.
- 지원되지 않는 테이블을 참조하는 쿼리는 쿼리 결과를 재사용할 수 없습니다.
- Athena에서 이전에 생성한 출력 파일을 재사용하려면 Amazon S3 읽기 권한이 있어야 합니다.

- 쿼리 결과 재사용 기능에서는 이전 결과의 내용이 수정되지 않은 것으로 가정합니다. Athena에서는 이전 결과를 사용하기 전에 무결성을 검사하지 않습니다.
- 이전 실행의 쿼리 결과가 삭제되거나 Amazon S3의 다른 위치로 이동된 경우 동일한 쿼리의 후속 실행에서는 쿼리 결과를 재사용하지 않습니다.
- 잠재적으로 기한이 경과한 결과가 반환될 수 있습니다. Athena에서는 지정한 최대 재사용 수명에 도달할 때까지 소스 데이터의 변경 사항을 확인하지 않습니다.
- 재사용할 수 있는 결과가 여러 개 있는 경우 최신 결과를 사용합니다.
- 비결정적 연산자 또는 함수(예: `rand()` 또는 `shuffle()`)를 사용하는 쿼리는 캐시된 결과를 사용하지 않습니다. 예를 들어 `ORDER BY`를 포함하지 않는 `LIMIT`은 비결정적이므로 캐시되지 않지만 `ORDER BY`를 포함하는 `LIMIT`은 결정적이므로 캐시됩니다.
- 쿼리 결과 재사용은 Athena 콘솔, Athena API 및 JDBC 드라이버에서 지원됩니다. 현재 쿼리 결과 재사용에 대한 ODBC 드라이버 지원은 Windows에서만 사용할 수 있습니다.
- JDBC로 쿼리 결과 재사용 기능을 사용하기 위해 필요한 최소 드라이버 버전은 2.0.34.1000입니다. ODBC의 경우 필요한 최소 드라이버 버전은 1.1.19.1002입니다. 드라이버 다운로드 정보는 ODBC 및 JDBC 드라이버로 Amazon Athena에 연결 섹션을 참조하세요.
- 데이터 카탈로그를 2개 이상 사용하는 쿼리에서는 쿼리 결과 재사용이 지원되지 않습니다.
- 테이블이 20개를 초과하여 포함된 쿼리에서는 쿼리 결과 재사용이 지원되지 않습니다.

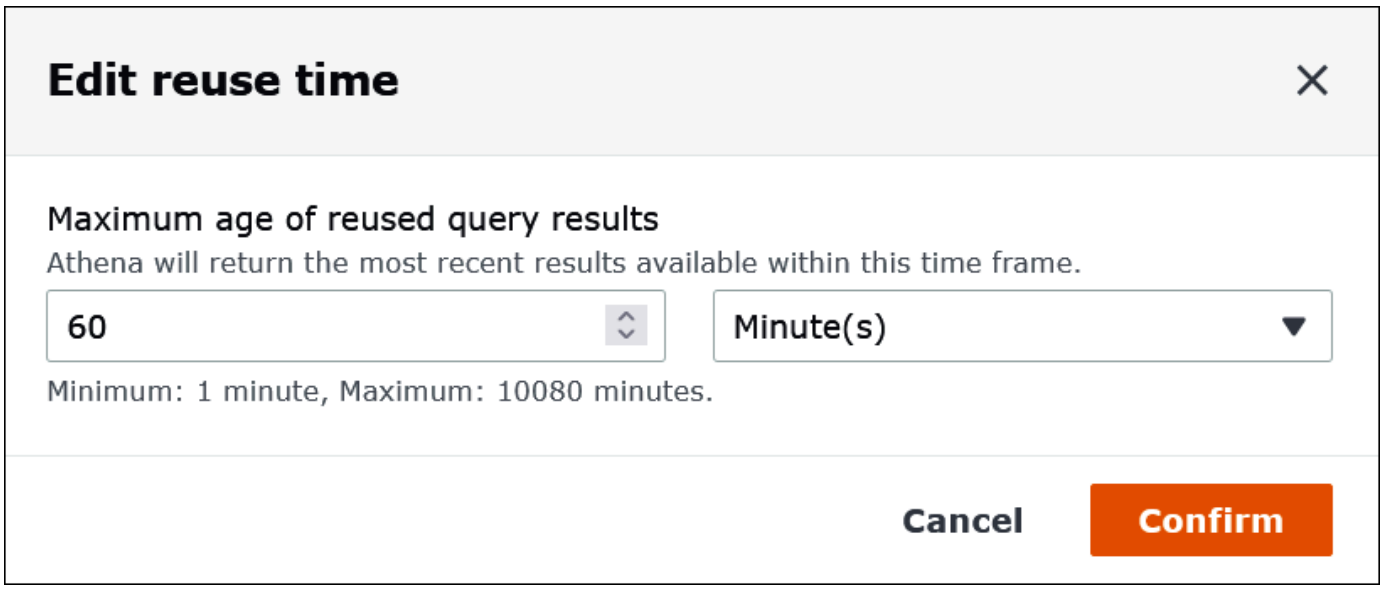
Athena 콘솔에서 쿼리 결과 재사용.

이 기능을 사용하려면 Athena 쿼리 편집기에서 Reuse query results(쿼리 결과 재사용) 옵션을 활성화합니다.

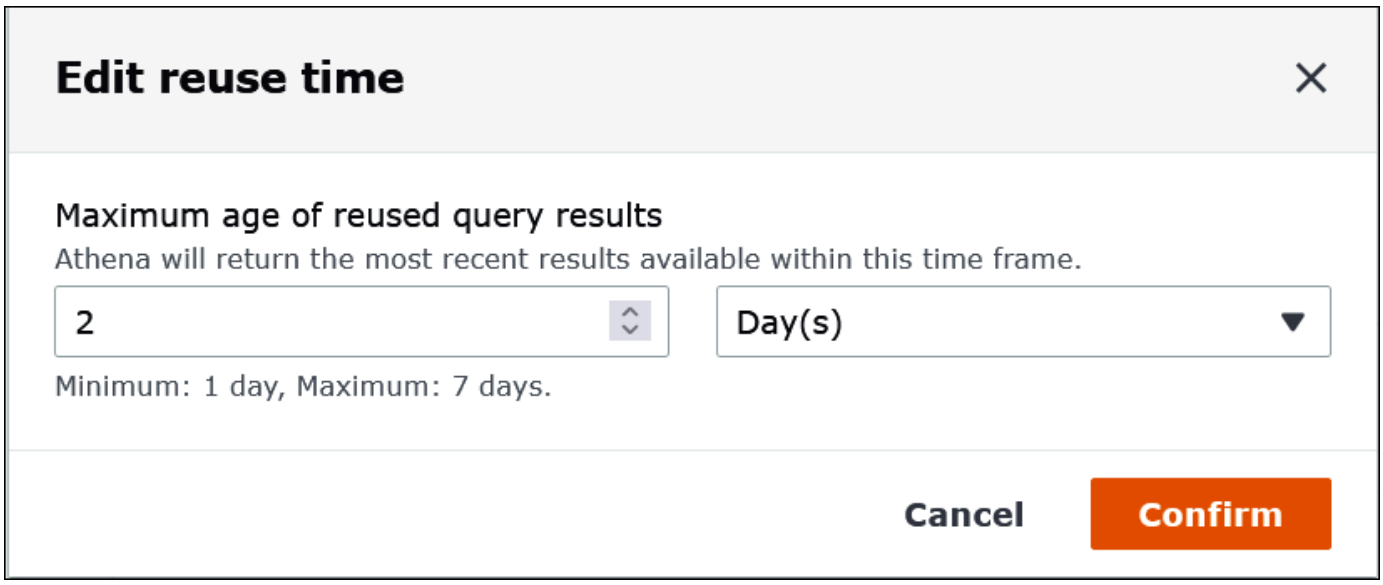


쿼리 결과 재사용 기능을 구성하려면

1. Athena 쿼리 편집기의 Reuse query results(쿼리 결과 재사용) 옵션에서 up to 60 minutes ago(최대 60분 전) 옆에 있는 편집 아이콘을 선택합니다.
2. Edit reuse time(재사용 시간 편집) 대화 상자의 오른쪽 상자에서 시간 단위(분, 시간 또는 일)를 선택합니다.
3. 왼쪽 상자에서 지정하려는 시간 단위 수를 입력하거나 선택합니다. 입력할 수 있는 최대 시간은 선택한 시간 단위에 상관없이 7일입니다.

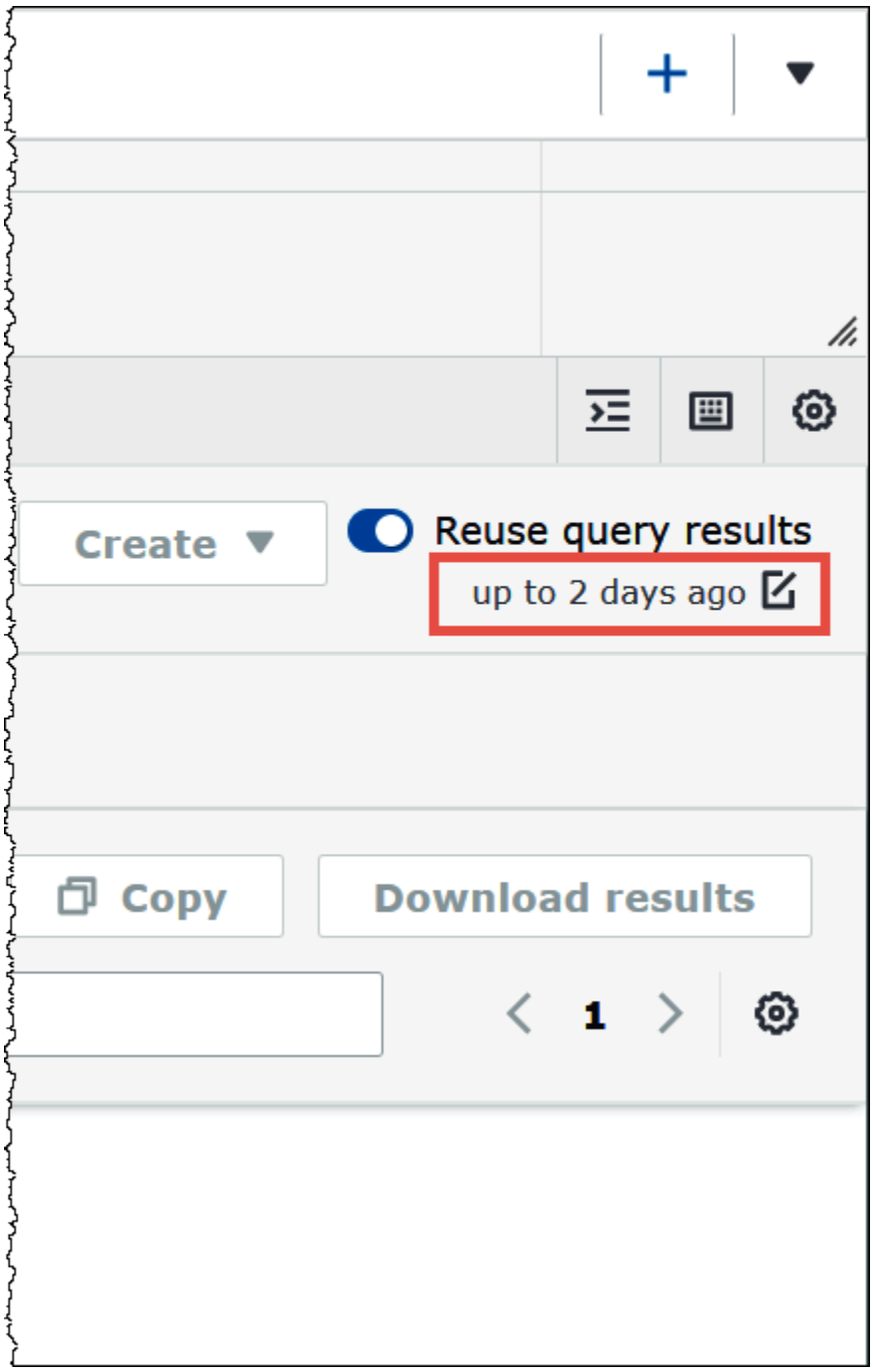


다음 예에서는 최대 재사용 시간을 2일로 지정합니다.



4. 확인을 선택합니다.

   배너에 구성 변경을 확인하는 메시지가 표시되고 Reuse query results(쿼리 결과 재사용) 옵션에 새 설정이 표시됩니다.

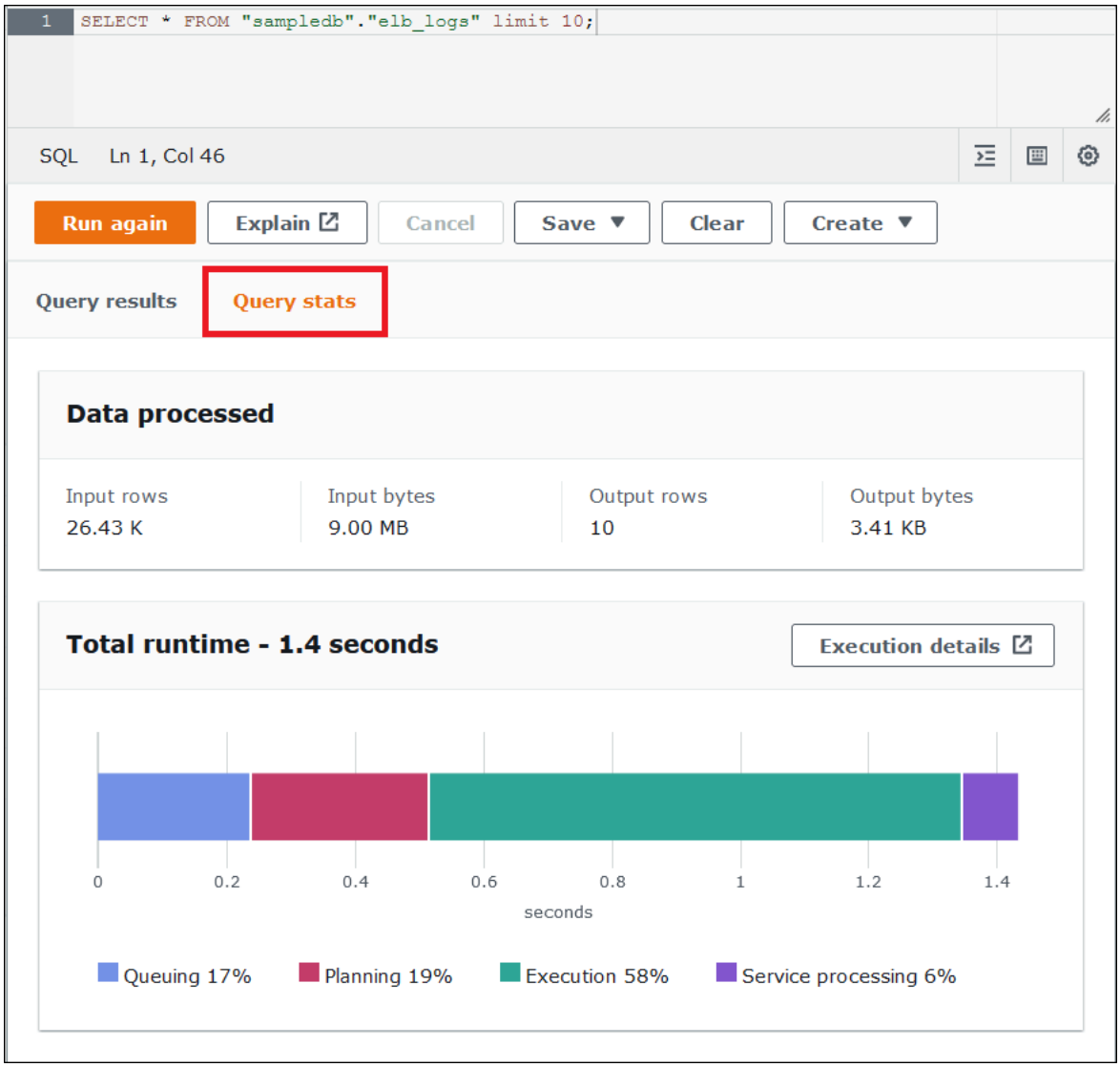


완료된 쿼리에 대한 통계 및 실행 세부 정보 보기

쿼리를 실행한 후 처리된 입력 및 출력 데이터에 대한 통계를 얻고 쿼리의 각 단계에 걸린 시간을 보여 주는 그래픽 표현을 확인하며 실행 세부 정보를 대화식으로 탐색할 수 있습니다.

완료된 쿼리에 대한 쿼리 통계를 보려면

1. Athena 쿼리 편집기에서 쿼리를 실행한 후 쿼리 통계(Query stats) 탭을 선택합니다.



쿼리 통계(Query stats) 탭은 다음 정보를 제공합니다.

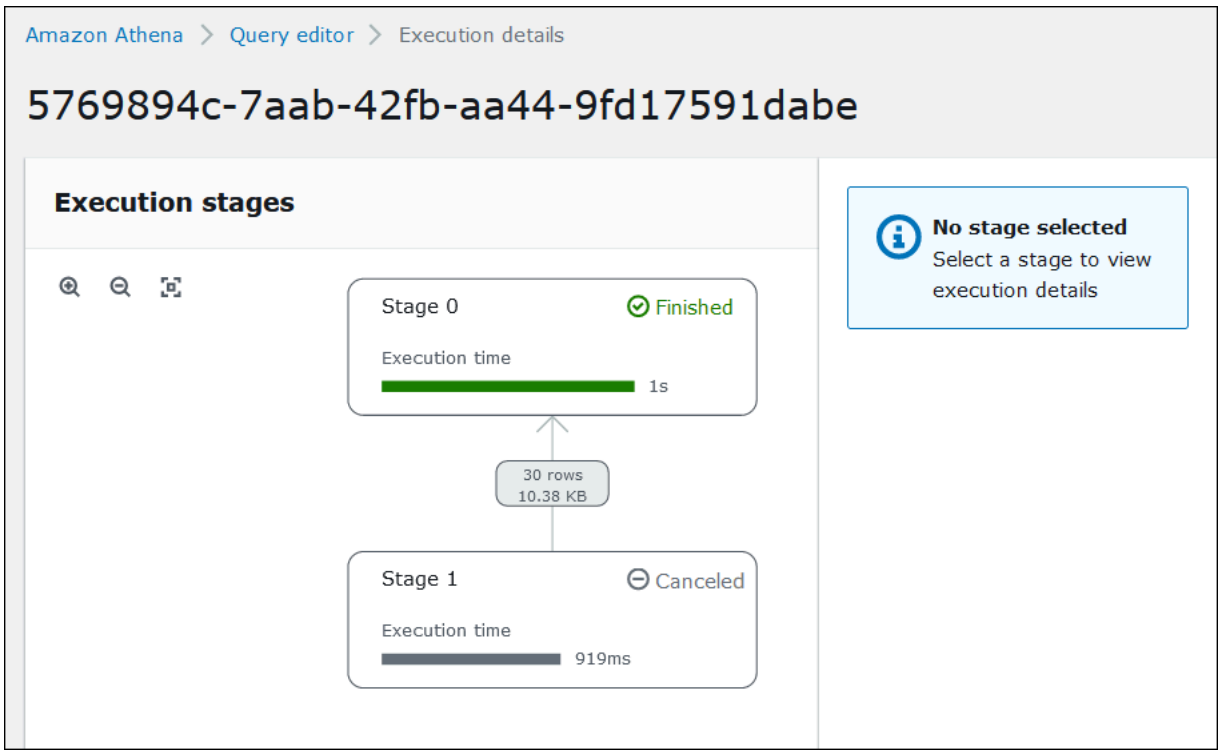
- 처리된 데이터 – 처리된 입력 행 및 바이트 수와 출력되는 행 및 바이트 수를 표시합니다.
- 총 런타임 – 쿼리를 실행하는 데 걸린 총 시간과 대기열, 계획, 실행, 서비스 처리에 소요된 시간을 보여주는 그래픽 표현을 표시합니다.

> **Note**
>
> 쿼리에서 행 수준 필터가 Lake Formation에 정의된 경우 단계 수준의 입력 및 출력 행 수와 데이터 크기 정보가 표시되지 않습니다.

2. 쿼리 실행 방식에 대한 정보를 대화식으로 탐색하려면 실행 세부 정보(Execution details)를 선택합니다.

실행 세부 정보(Execution details) 페이지에는 쿼리의 실행 ID와 0부터 시작하는 쿼리 단계의 그래프가 표시됩니다. 단계는 처음부터 끝까지 상향식으로 정렬됩니다. 각 단계의 레이블에는 단계를 실행하는 데 걸린 시간이 표시됩니다.

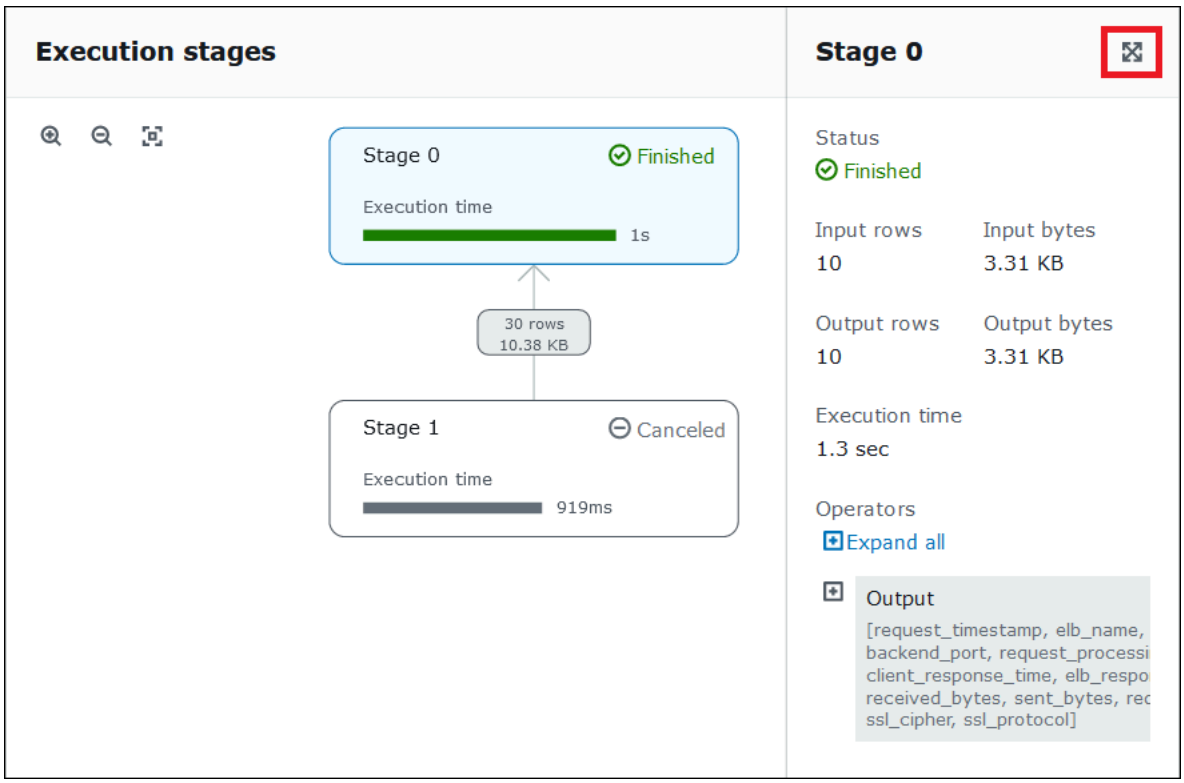
> **Note**
>
> 쿼리의 총 런타임과 실행 단계 시간은 크게 다른 경우가 많습니다. 예를 들어 총 런타임이 분 단위인 쿼리에서 단계의 실행 시간이 시간 단위로 표시될 수 있습니다. 단계는 여러 작업에서 병렬로 실행되는 논리적 계산 단위이므로 단계의 실행 시간은 모든 작업의 실행 시간을 집계한 것입니다. 이러한 불일치에도 불구하고 단계 실행 시간은 쿼리에서 컴퓨팅 집약도가 가장 높은 단계를 상대적으로 나타내는 지표로 유용할 수 있습니다.



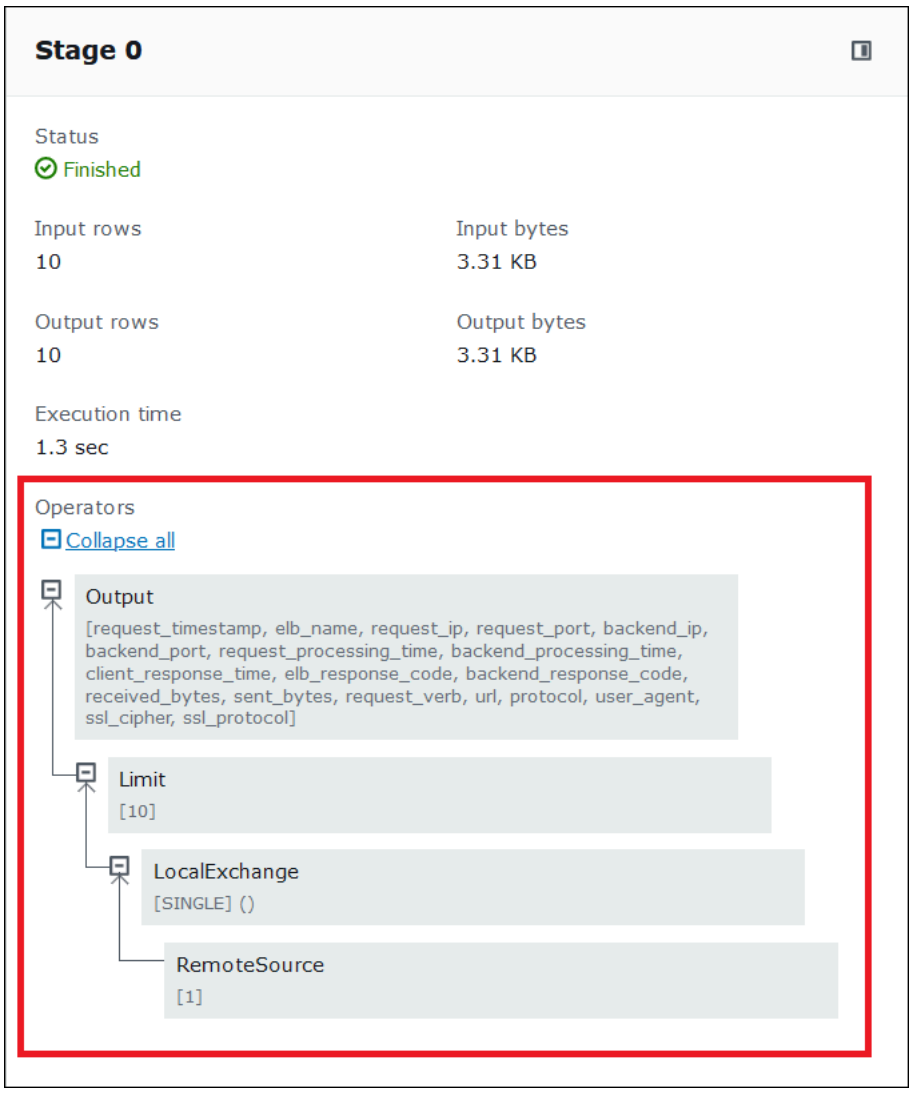
그래프를 살펴보려면 다음 옵션을 사용합니다.

- 마우스를 스크롤하여 확대 또는 축소하거나, 돋보기 아이콘을 사용합니다.
- 화면에 맞게 그래프를 조정하려면 크기에 맞게 확대/축소(Zoom to fit) 아이콘을 선택합니다.
- 그래프를 이동하려면 마우스 포인터를 드래그합니다.

3. 단계에 대한 세부 정보를 보려면 단계를 선택합니다. 오른쪽의 단계 세부 정보 창에는 입력 및 출력한 행 및 바이트 수와 연산자 트리가 표시됩니다.



4. 단계 세부 정보를 전체 너비로 보려면 세부 정보 창의 오른쪽 상단에 있는 확장 아이콘을 선택합니다.
5. 단계의 부분에 대한 정보를 가져오려면 연산자 트리에서 하나 이상의 항목을 확장합니다.



실행 세부 정보에 대한 자세한 내용은 Athena EXPLAIN 문 결과의 이해 단원을 참조하세요.

추가적인 리소스

자세한 정보는 다음 리소스를 참조하세요.

SQL 쿼리에 대한 실행 계획 보기

Athena에서 EXPLAIN 및 EXPLAIN ANALYZE 사용

뷰 작업

Amazon Athena의 뷰는 물리적 테이블이 아니라 논리적 테이블입니다. 뷰를 정의하는 쿼리는 쿼리에서 해당 뷰가 참조될 때마다 실행됩니다.

SELECT 쿼리에서 뷰를 생성한 후 향후 쿼리에 이 뷰를 참조할 수 있습니다. 자세한 내용은 CREATE VIEW 단원을 참조하세요.

주제

뷰를 사용해야 하는 경우

뷰를 생성하여 다음을 수행할 수 있습니다.

- 데이터 하위 집합을 쿼리합니다. 예를 들어 원본 테이블에서 열 하위 집합이 있는 뷰를 만들어 데이터 쿼리를 간소화할 수 있습니다.
- 여러 테이블을 쿼리 하나에 결합합니다. 테이블이 여러 개 있고 UNION ALL로 테이블을 결합하려는 경우 해당 표현식으로 뷰를 생성하여 결합된 테이블에 대한 쿼리를 간소화할 수 있습니다.
- 기존 기본 쿼리의 복잡성을 숨기고 사용자의 쿼리 실행을 간소화합니다. 기본 쿼리에는 보통 쿼리 이해 및 디버깅을 어렵게 하는 테이블 사이의 조인, 열 목록의 표현식, 여러 SQL 구문이 포함됩니다. 복잡성을 숨기고 쿼리를 간소화하는 뷰를 생성할 수 있습니다.
- 최적화 기술을 사용하여 실험하고 최적화된 쿼리를 생성합니다. 예를 들어 최고의 성능을 입증하는 WHERE 조건, JOIN 명령 또는 여러 표현식의 조합을 찾은 경우, 이러한 절과 표현식을 사용하여 뷰를 생성할 수 있습니다. 그러면 애플리케이션이 이 뷰에 대해 비교적 간단한 쿼리를 만들 수 있습니다. 원래 쿼리를 최적화할 더 좋은 방법을 나중에 찾은 경우, 뷰를 다시 생성할 때 모든 애플리케이션이 최적화된 기본 쿼리의 이점을 즉시 활용합니다.
- 기본 테이블 및 열 이름이 변경된 경우 그러한 이름을 숨기고, 유지 관리 문제를 최소화합니다. 이 경우 새 이름을 사용하여 뷰를 다시 생성합니다. 기본 테이블 대신 뷰를 사용하는 모든 쿼리는 아무런 변경 없이 계속 실행됩니다.

Athena의 뷰에 대해 지원되는 작업

Athena는 뷰에 대해 다음 작업을 지원합니다. 쿼리 편집기에서 다음 명령을 실행할 수 있습니다.

문	설명
CREATE VIEW	지정된 SELECT 쿼리에서 새 뷰를 생성합니다. 자세한 내용은 뷰 생성 단원을 참조하세요. 선택적 OR REPLACE 절을 사용하여 기존 뷰를 바꿔 업데이트할 수 있습니다.
DESCRIBE VIEW	명명된 뷰의 열 목록을 보여 줍니다. 이를 통해 복잡한 뷰의 속성을 검사할 수 있습니다.
DROP VIEW	기존 뷰를 삭제합니다. 뷰가 없는 경우 선택적 IF EXISTS 절이 오류를 억제합니다.
SHOW CREATE VIEW	지정된 뷰를 생성하는 SQL 문을 보여 줍니다.
SHOW VIEWS	지정된 데이터베이스 또는 현재 데이터베이스(데이터베이스 이름이 생략된 경우)의 뷰를 나열합니다. 선택적 LIKE 절을 정규식과 함께 사용하여 뷰 이름 목록을 제한합니다. 콘솔의 왼쪽 창에서 뷰 목록을 볼 수도 있습니다.
SHOW COLUMNS	뷰에 대한 스키마에 열을 나열합니다.

뷰 고려 사항

다음 고려 사항은 Athena에서 뷰를 생성하고 사용하는 데 적용됩니다.

- Athena에서 Athena 콘솔, AWS Glue Data Catalog(이를 사용하도록 마이그레이션한 경우) 또는 동일한 카탈로그에 연결된 Amazon EMR 클러스터에서 실행되는 Presto를 사용하여 생성된 뷰를 미리 보고 작업할 수 있습니다. 다른 방법으로 생성된 Athena 뷰는 미리 보거나 추가할 수 없습니다.
- AWS GlueData Catalog를 통해 보기를 생성하는 경우 PartitionKeys 파라미터를 포함시키고 "PartitionKeys":[]와 같이 해당 값을 빈 목록으로 설정해야 합니다. 그렇지 않으면 Athena에서 뷰 쿼리가 실패합니다. 다음 예제에서는 "PartitionKeys":[]를 사용하여 Data Catalog에서 생성된 뷰를 보여 줍니다.

```
aws glue create-table
--database-name mydb
--table-input '{
"Name":"test",
  "TableType": "EXTERNAL_TABLE",
  "Owner": "hadoop",
  "StorageDescriptor":{
     "Columns":[{
           "Name":"a","Type":"string"},{"Name":"b","Type":"string"}],
   "Location":"s3://DOC-EXAMPLE-BUCKET/Oct2018/25Oct2018/",
   "InputFormat":"org.apache.hadoop.mapred.TextInputFormat",
   "OutputFormat": "org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat",
   "SerdeInfo":{"SerializationLibrary":"org.apache.hadoop.hive.serde2.OpenCSVSerde",
   "Parameters":{"separatorChar": "|", "serialization.format":
 "1"}}},"PartitionKeys":[]}'
```

- Data Catalog에서 Athena 뷰를 생성한 경우, Data Catalog가 뷰를 테이블로 처리합니다. Data Catalog에서 테이블 수준 세분화된 액세스 제어를 사용하여 이러한 뷰에 대한 액세스를 제한할 수 있습니다.
- Athena는 재귀 뷰를 실행하지 못하도록 하고 그러한 경우 오류 메시지를 표시합니다. 재귀 뷰는 자체를 참조하는 뷰 쿼리입니다.
- Athena는 기한 경과 뷰를 감지하면 오류 메시지를 표시합니다. 다음 중 하나가 발생하면 기한 경과 뷰가 보고됩니다.
  - 뷰가 존재하지 않는 테이블 또는 데이터베이스를 참조합니다.
  - 스키마 또는 메타데이터 변경이 참조된 테이블에서 이루어집니다.
  - 참조된 테이블이 삭제되고 다른 스키마 또는 구성으로 재생성됩니다.
- 중첩 뷰 뒤의 쿼리가 유효하고 테이블과 데이터베이스가 있는 한 중첩 뷰를 만들고 실행할 수 있습니다.

뷰 제한 사항

- Athena 뷰 이름에는 밑줄 (_) 이외의 특수 문자를 사용할 수 없습니다. 자세한 내용은 테이블, 데이터베이스 및 열의 이름 단원을 참조하세요.
- 뷰 이름을 지정하는 데 예약어를 사용하지 마세요. 예약어를 사용하는 경우에는 뷰에 대한 쿼리에서 큰 따옴표를 사용하여 예약어를 묶습니다. 예약어 섹션을 참조하세요.
- 외부 Hive 메타스토어 또는 UDF를 통해 Athena에서 생성된 보기를 사용할 수 없습니다. 외부 Hive에서 작성된 뷰 작업에 대한 자세한 내용은 Hive 뷰 작업을 참조하세요.

- 지리 공간 함수에 뷰를 사용할 수 없습니다.
- 뷰를 사용하여 Amazon S3의 데이터에 대한 액세스 제어를 관리할 수 없습니다. 뷰를 쿼리하려면 Amazon S3에 저장된 데이터에 액세스할 수 있는 권한이 필요합니다. 자세한 내용은 Amazon S3에 액세스 단원을 참조하십시오.
- 여러 계정에서 보기를 쿼리하는 작업은 Athena 엔진 버전 2와 Athena 엔진 버전 3에서 모두 지원되지만 크로스 계정 AWS Glue Data Catalog를 포함하는 보기는 생성할 수 없습니다. 교차 계정 데이터 카탈로그 액세스에 관한 자세한 내용은 AWS Glue 데이터 카탈로그에 대한 교차 계정 액세스를 참조하세요.
- Athena에서는 Hive 또는 Iceberg의 숨겨진 메타데이터 열 `$bucket`, `$file_modified_time`, `$file_size` 및 `$partition`을 보기에서 지원하지 않습니다. Athena에서 `$path` 메타데이터 열 사용에 대한 자세한 내용은 Amazon S3의 소스 데이터에 대한 파일 위치 가져오기 섹션을 참조하세요.

콘솔에서 뷰 관련 작업

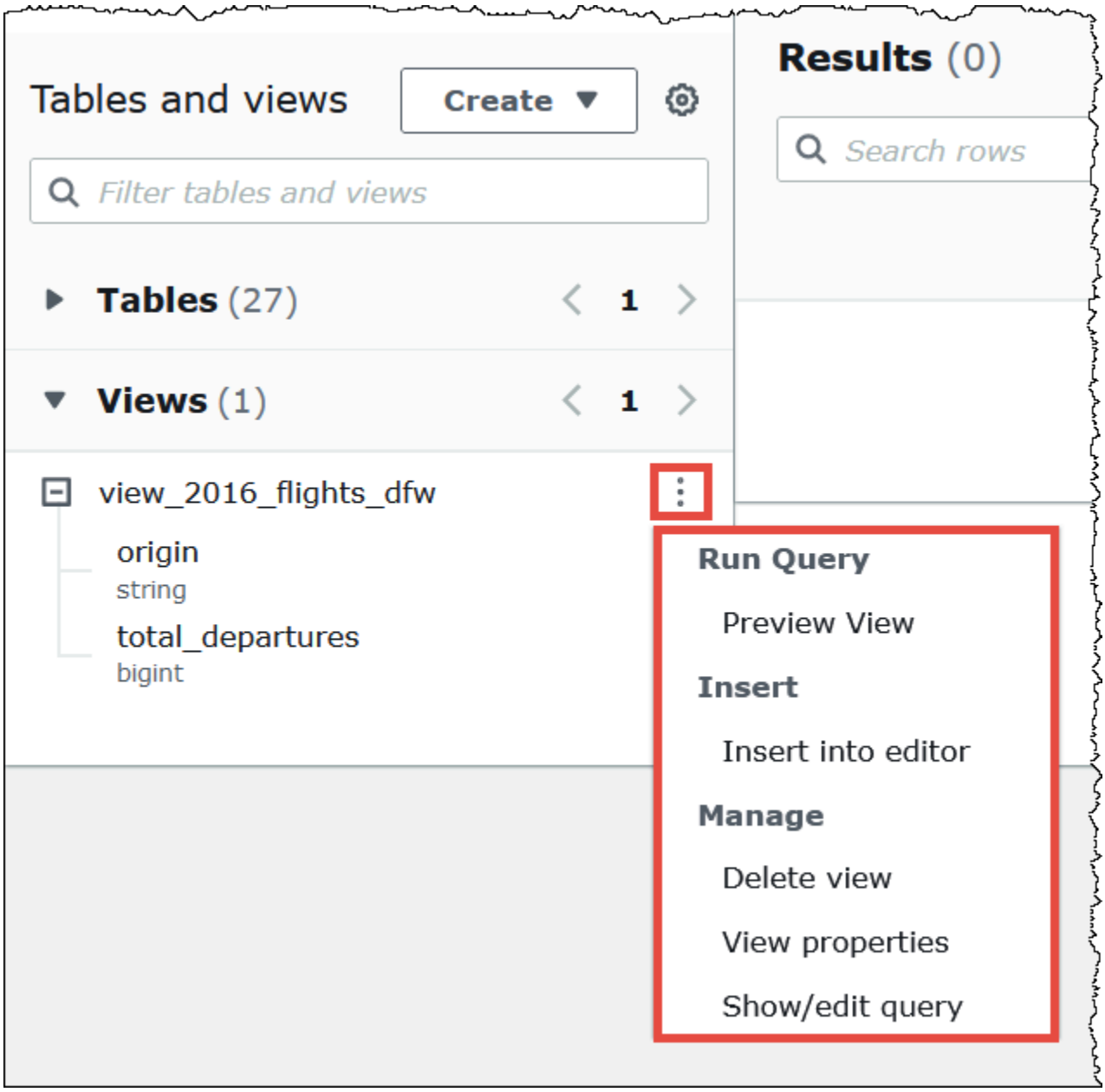
Athena 콘솔에서 다음을 수행할 수 있습니다.

- 테이블이 나열된 왼쪽 창에서 모든 뷰를 찾을 수 있습니다.
- 뷰를 필터링합니다.
- 뷰를 미리 보거나, 속성을 표시, 편집 또는 삭제합니다.

뷰에 대한 작업 표시

뷰를 이미 생성한 경우에만 콘솔에 뷰가 나타납니다.

1. Athena 콘솔에서 뷰(Views)를 선택한 다음 확장하면 뷰(Views) 아래 여러 열이 표시됩니다.
2. 뷰 옆에 있는 3개의 세로 점을 선택하여 뷰에 대한 작업 목록을 표시합니다.



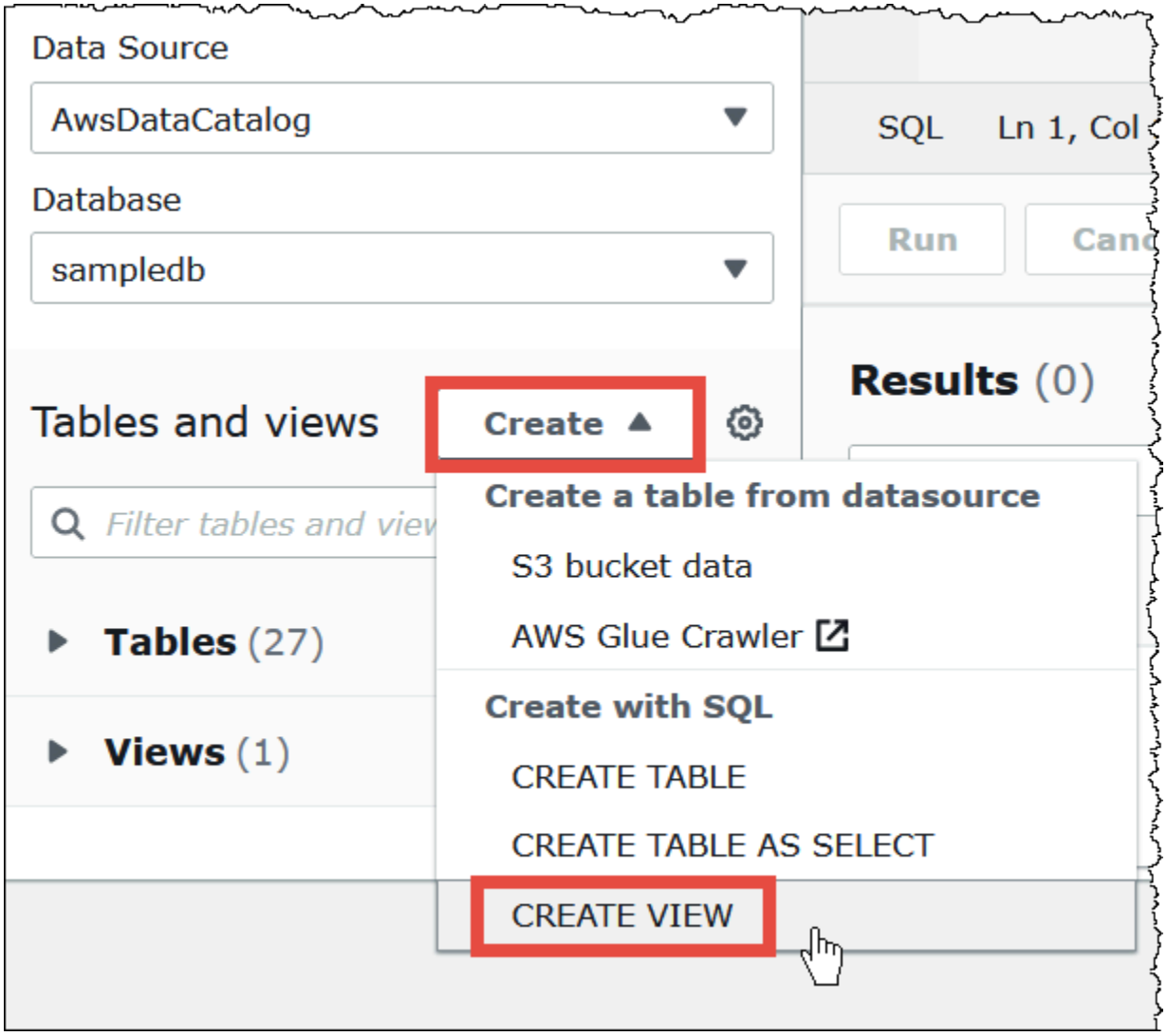
3. 뷰를 미리 보거나, 뷰 이름을 쿼리 편집기에 삽입하거나, 뷰를 삭제하거나, 뷰의 속성을 보거나, 쿼리 편집기에서 뷰를 표시하고 편집하는 작업을 선택합니다.

뷰 생성

템플릿을 사용하거나 기존 쿼리를 실행하여 Athena 콘솔에서 뷰를 생성할 수 있습니다.

템플릿을 사용하여 뷰 생성

1. Athena 콘솔에서 테이블 및 뷰(Tables and views) 옆에 있는 생성(Create)을 선택한 다음 뷰 생성(Create view)을 선택합니다.



   이 작업은 편집 가능한 뷰 템플릿을 쿼리 편집기에 배치합니다.

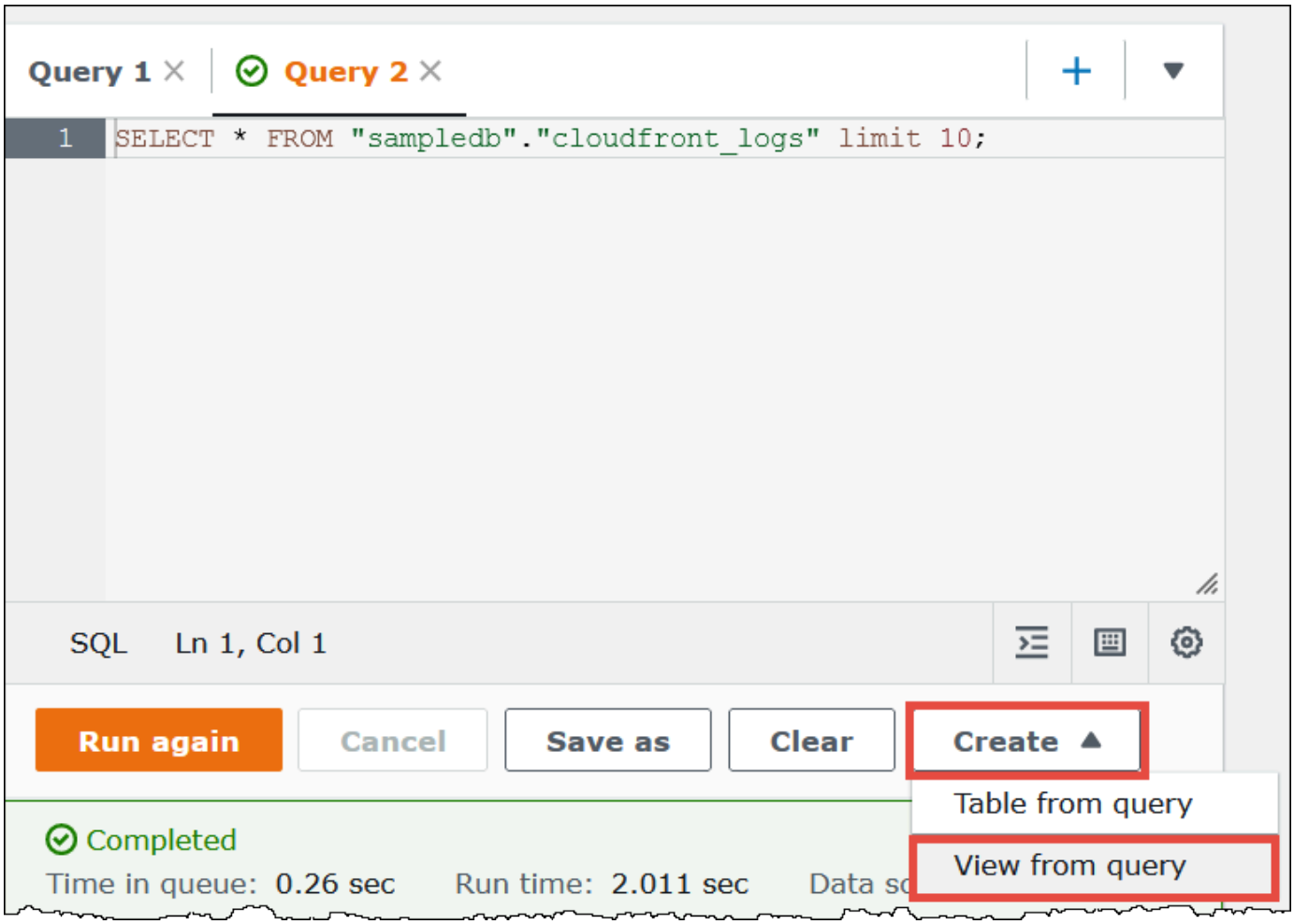
2. 요구 사항에 따라 뷰 템플릿을 편집합니다. 문에 뷰 이름을 입력할 때 뷰 이름에 밑줄(_) 이외의 특수 문자를 사용할 수 없습니다. 테이블, 데이터베이스 및 열의 이름 섹션을 참조하세요. 뷰 이름을 지정하는데 예약어를 사용하지 마세요.

   뷰 생성에 대한 자세한 내용은 CREATE VIEW 및 뷰 예제 섹션을 참조하세요.

3. 실행(Run)을 선택하여 뷰를 생성합니다. 뷰가 Athena 콘솔의 뷰 목록에 나타납니다.

기존 쿼리에서 뷰 생성

1. Athena 쿼리 편집기를 사용하여 기존 쿼리를 실행합니다.
2. 쿼리 편집기 창에서 생성(Create)을 선택한 다음 쿼리에서 뷰(View from query)를 선택합니다.



3. 뷰 생성(Create View) 대화 상자에서 뷰 이름을 입력하고 생성(Create)을 선택합니다. 뷰 이름에는 밑줄 (_) 이외의 특수 문자를 사용할 수 없습니다. 테이블, 데이터베이스 및 열의 이름 섹션을 참조하세요. 뷰 이름을 지정하는데 예약어를 사용하지 마세요.

   Athena는 콘솔의 뷰 목록에 뷰를 추가하고 쿼리 편집기에 뷰에 대한 `CREATE VIEW` 문을 표시합니다.

참고

- 테이블의 기반이 되는 테이블을 삭제한 다음 뷰를 실행하려고 하면 Athena에서 오류 메시지를 표시합니다.

- 기존 뷰의 위에 있는 뷰인 중첩 뷰(nested view)를 생성할 수 있습니다. Athena는 자신을 참조하는 재귀 뷰가 실행되지 않도록 방지합니다.

뷰 예제

뷰 쿼리 구문을 표시하려면 SHOW CREATE VIEW를 사용합니다.

Example 예 1

`id` 및 `name`이라는 열 2개가 있는 `employees` 테이블과, `id` 및 `salary`라는 열 2개가 있는 `salaries` 테이블을 가정하겠습니다.

이 예에서는 테이블 `employees` 및 `salaries`에서 급여에 매핑된 ID 목록을 가져오는 `SELECT` 쿼리로 `name_salary`라는 뷰를 생성합니다.

```
CREATE VIEW name_salary AS
SELECT
 employees.name,
 salaries.salary
FROM employees, salaries
WHERE employees.id = salaries.id
```

Example 예제 2

다음 예에서는 더 복잡한 쿼리 구문을 숨길 수 있는 `view1`이라는 뷰를 생성합니다.

이 뷰는 `table1` 및 `table2`라는 테이블 2개 위에서 실행되며, 각 테이블은 다른 `SELECT` 쿼리입니다. 이 뷰는 `table1`에서 열을 선택하고 `table2`로 결과를 조인합니다. 이 조인은 두 테이블에 있는 `a` 열을 기반으로 합니다.

```
CREATE VIEW view1 AS
WITH
  table1 AS (
         SELECT a,
         MAX(b) AS the_max
         FROM x
         GROUP BY a
         ),
  table2 AS (
         SELECT a,
```

```
         AVG(d) AS the_avg
         FROM y
         GROUP BY a)
SELECT table1.a, table1.the_max, table2.the_avg
FROM table1
JOIN table2
ON table1.a = table2.a;
```

페더레이션된 보기 쿼리에 대한 자세한 내용은 페더레이션된 보기 쿼리 섹션을 참조하세요.

AWS Glue Data Catalog 뷰 사용

이 기능은 현재 프리뷰 버전이 출시 중이기 때문에 변경될 수도 있습니다. 자세한 내용을 알아보려면 AWS 서비스 약관 문서의 베타 및 미리 보기 섹션을 참조하세요.

Amazon Athena, Amazon Redshift 등의 AWS 서비스 전반에 걸쳐 단일 공통 뷰를 원하는 경우 AWS Glue Data Catalog 뷰를 사용합니다. Data Catalog 뷰에서 액세스 권한은 뷰를 쿼리하는 사용자 대신 뷰를 만든 사용자가 정의합니다. 이러한 권한 부여 방법을 정의자 의미 체계라고 합니다.

다음은 Data Catalog 뷰를 사용할 수 있는 방법을 보여주는 사용 사례입니다.

- 액세스 제어 강화 - 사용자에게 필요한 권한 수준에 따라 데이터 액세스를 제한하는 뷰를 생성합니다. 예를 들어 Data Catalog 뷰를 사용하여 인사 관리(HR) 부서 소속이 아닌 직원은 개인 식별 정보를 보지 못하도록 할 수 있습니다.
- 완전한 기록 보장 - Data Catalog 뷰에 특정 필터를 적용하여 Data Catalog 뷰의 데이터 레코드가 항상 완전한지 확인할 수 있습니다.
- 보안 강화 - Data Catalog 뷰에서 뷰를 생성하려면 뷰를 생성하는 쿼리 정의가 그대로 유지되어야 합니다. 이렇게 하면 Data Catalog 뷰가 악의적인 행위자의 SQL 명령에 덜 취약해집니다.
- 기본 테이블에 대한 액세스 방지 - 정의자 의미 체계를 통해 사용자는 기본 테이블을 사용하지 않으면서 뷰에 액세스할 수 있습니다. 뷰를 정의하는 사용자에게만 테이블에 대한 액세스 권한이 필요합니다.

Data Catalog 뷰 정의는 AWS Glue Data Catalog에 저장됩니다. 즉, AWS Lake Formation을 사용하여 리소스 부여, 열 부여 또는 태그 기반 액세스 제어를 통해 액세스 권한을 부여할 수 있습니다. Lake Formation에서 액세스 권한 부여 및 취소에 대한 자세한 내용은 AWS Lake Formation 개발자 안내서의 Granting and revoking permissions on Data Catalog resources를 참조하십시오.

권한

Data Catalog 뷰에는 `Lake Formation Admin`, `Definer` 및 `Invoker`의 세 가지 역할이 필요합니다.

- **`Lake Formation Admin`** - 모든 Lake Formation 권한을 구성할 수 있는 액세스 권한이 있습니다.
- **`Definer`** - Data Catalog 뷰를 생성합니다. `Definer` 역할에는 뷰 정의가 참조하는 모든 기본 테이블에 대해 부여 가능한 전체 `SELECT` 권한이 있어야 합니다.
- **`Invoker`** - Data Catalog 뷰를 쿼리하거나 해당 메타데이터를 확인할 수 있습니다.

`Definer` 역할의 신뢰 관계는 다음 예제와 같이 AWS Glue 및 Lake Formation 서비스 보안 주체에 대한 `sts:AssumeRole` 작업을 허용해야 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "glue.amazonaws.com",
                    "lakeformation.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Athena 액세스를 위한 IAM 권한도 필요합니다. 자세한 내용은 Amazon Athena의 AWS 관리형 정책 단원을 참조하십시오.

제한 사항

- Data Catalog 뷰는 다른 뷰를 참조할 수 없습니다.
- 뷰 정의에서 최대 10개의 테이블을 참조할 수 있습니다.
- 기본 테이블이 Lake Formation에 등록되어 있어야 합니다.
- `DEFINER` 보안 주체는 IAM 역할만 될 수 있습니다.

- DEFINER 역할에는 기본 테이블에 대한 전체 SELECT(부여 가능) 권한이 있어야 합니다.
- UNPROTECTED Data Catalog 뷰는 지원되지 않습니다.
- 사용자 정의 함수(UDF)는 뷰 정의에서 지원되지 않습니다.
- Athena 페더레이션 데이터 소스는 Data Catalog 뷰에서 사용할 수 없습니다.
- 외부 Hive 메타스토어에는 Data Catalog 뷰가 지원되지 않습니다.
- Athena는 기한 경과 뷰를 감지하면 오류 메시지를 표시합니다. 다음 중 하나가 발생하면 기한 경과 뷰가 보고됩니다.
  - 뷰가 존재하지 않는 테이블 또는 데이터베이스를 참조합니다.
  - 스키마 또는 메타데이터 변경이 참조된 테이블에서 이루어집니다.
  - 참조된 테이블이 삭제되고 다른 스키마 또는 구성으로 재생성됩니다.

Data Catalog 뷰 생성

다음 예제 구문은 Definer 역할의 사용자가 orders_by_date Data Catalog 뷰를 생성하는 방법을 보여줍니다. 이 예제에서는 Definer 역할에 default 데이터베이스의 orders 테이블에 대한 전체 SELECT 권한이 있다고 가정합니다.

```
CREATE PROTECTED MULTI DIALECT VIEW orders_by_date
SECURITY DEFINER
AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
WHERE order_city = 'SEATTLE'
GROUP BY orderdate
```

Data Catalog 뷰 쿼리

뷰가 생성된 후 Lake Formation Admin은 Data Catalog 뷰에 대한 SELECT 권한을 Invoker 보안 주체에게 부여할 수 있습니다. 그러면 Invoker 보안 주체는 뷰에서 참조하는 기본 테이블에 액세스하지 않고도 뷰를 쿼리할 수 있습니다. 다음은 Invoker 쿼리의 예입니다.

```
SELECT * from orders_by_date where price > 5000
```

Data Catalog 뷰 업데이트

`Lake Formation Admin` 또는 `Definer`는 `ALTER VIEW UPDATE DIALECT` 구문을 사용하여 뷰 정의를 업데이트할 수 있습니다. 다음 예제에서는 `orders` 테이블 대신 `returns` 테이블에서 열을 선택하도록 뷰 정의를 수정합니다.

```
ALTER VIEW orders_by_date UPDATE DIALECT
AS
SELECT return_date, sum(totalprice) AS price
FROM returns
WHERE order_city = 'SEATTLE'
GROUP BY orderdate
```

Data Catalog 뷰 생성 및 관리 구문에 대한 자세한 내용은 Glue Data Catalog 뷰 구문 섹션을 참조하세요.

Glue Data Catalog 뷰 구문

> 이 기능은 현재 프리뷰 버전이 출시 중이기 때문에 변경될 수도 있습니다. 자세한 내용을 알아보려면 AWS 서비스 약관 문서의 베타 및 미리 보기 섹션을 참조하세요.

이 섹션에서는 AWS Glue Data Catalog 뷰를 생성하고 관리하기 위한 데이터 정의 언어(DDL) 명령을 설명합니다.

ALTER VIEW DIALECT

엔진 언어를 추가하거나 기존 엔진 언어를 업데이트 또는 삭제하여 Data Catalog 뷰를 업데이트할 수 있습니다. `Lake FormationAdmin`과 `Definer`(뷰를 생성한 사용자)만 Data Catalog 뷰에서 `ALTER VIEW DIALECT` 문을 사용할 수 있는 권한을 갖습니다.

구문

```
ALTER VIEW name [ FORCE ] [ ADD|UPDATE ] DIALECT AS query
```

```
ALTER VIEW name [ DROP ] DIALECT
```

FORCE

FORCE 키워드는 뷰에서 충돌하는 엔진 언어 정보를 새 정의로 덮어쓰게 합니다. FORCE 키워드는 Data Catalog 뷰를 업데이트하여 기존 엔진 언어 간에 뷰 정의가 충돌하는 경우에 유용합니다. Data Catalog 뷰에 Athena 언어와 Amazon Redshift 언어가 모두 있고 업데이트로 인해 보기 정의에서 Amazon Redshift와 충돌이 발생한다고 가정해 보겠습니다. 이 경우 FORCE 키워드를 사용하여 업데이트를 완료하고 Amazon Redshift 언어를 기한 경과로 표시할 수 있습니다. 기한 경과로 표시된 엔진이 뷰를 쿼리하면 쿼리가 실패합니다. 엔진에서 예외를 발생시켜 기한 경과 결과를 허용하지 않습니다. 이 문제를 해결하려면 뷰에서 기한 경과 언어를 업데이트합니다.

ADD

Data Catalog 뷰에 새 엔진 언어를 추가합니다. 지정된 엔진이 Data Catalog 뷰에 이미 존재해서는 안 됩니다.

UPDATE

Data Catalog 뷰에 이미 존재하는 엔진 언어를 업데이트합니다.

DROP

Data Catalog 뷰에서 기존 엔진 언어를 삭제합니다. Data Catalog 뷰에서 엔진을 삭제한 후에는 삭제된 엔진에서 Data Catalog 뷰를 쿼리할 수 없습니다. 뷰의 다른 엔진 언어는 여전히 뷰를 쿼리할 수 있습니다.

DIALECT AS

엔진별 SQL 쿼리를 소개합니다.

예제

```
ALTER VIEW orders_by_date FORCE ADD DIALECT
AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
GROUP BY orderdate
```

```
ALTER VIEW orders_by_date FORCE UPDATE DIALECT
AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
GROUP BY orderdate
```

```
ALTER VIEW orders_by_date DROP DIALECT
```

CREATE PROTECTED MULTI DIALECT VIEW

AWS Glue Data Catalog에서 Data Catalog 뷰를 생성합니다. Data Catalog 뷰는 Athena와 Amazon Redshift, Amazon EMR 등의 다른 SQL 엔진에서 원활하게 작동하는 단일 뷰 스키마입니다.

구문

```
CREATE [ OR REPLACE ] PROTECTED MULTI DIALECT VIEW view_name
 [ SECURITY DEFINER ]
AS query
```

PROTECTED

필수 키워드. 데이터 유출로부터 뷰를 보호하도록 지정합니다. Data Catalog 뷰는 `PROTECTED` 뷰로만 생성할 수 있습니다.

MULTI DIALECT

뷰가 다양한 쿼리 엔진의 SQL 언어의 지원하므로 해당 엔진에서 읽을 수 있도록 지정합니다.

SECURITY DEFINER

이 뷰에 정의자 의미 체계가 적용되도록 지정합니다. 정의자 의미 체계란 기본 테이블에 대한 유효 읽기 권한이 실제 읽기를 수행하는 보안 주체가 아니라 뷰를 정의한 보안 주체 또는 역할에 속함을 의미합니다.

OR REPLACE

다른 엔진의 SQL 언어가 뷰에 있는 경우 Data Catalog 뷰를 대체할 수 없습니다. 호출 엔진의 뷰에 SQL 언어만 있는 경우 뷰를 바꿀 수 있습니다.

예

다음 예제에서는 `orders` 테이블의 쿼리를 기반으로 `orders_by_date` Data Catalog 뷰를 생성합니다.

```
CREATE PROTECTED MULTI DIALECT VIEW orders_by_date
SECURITY DEFINER
AS
```

```
SELECT orderdate, sum(totalprice) AS price
FROM orders
WHERE order_city = 'SEATTLE'
GROUP BY orderdate
```

DESCRIBE

지정된 Data Catalog 뷰의 열 목록을 표시합니다. DESCRIBE 문은 Athena 뷰의 DESCRIBE 문과 유사합니다. Athena의 뷰와 달리 명령문의 출력은 Lake Formation 액세스 제어를 통해 제어됩니다. 따라서 이 쿼리의 출력은 뷰의 모든 열이 아니라 호출자가 액세스할 수 있는 열입니다.

구문

```
DESCRIBE [db_name.]view_name
```

예제

```
DESCRIBE orders
```

DROP VIEW

호출 엔진 언어가 Data Catalog 뷰에 있는 경우에만 Data Catalog 뷰를 삭제합니다. 예를 들어 사용자가 Athena에서 DROP VIEW를 호출하면 Athena의 언어가 뷰에 있는 경우에만 뷰가 삭제됩니다. 그렇지 않으면 작업이 실패합니다. Lake Formation 관리자와 뷰 정의자만 Data Catalog 뷰에서 DROP VIEW 문을 사용할 수 있는 권한이 있습니다.

구문

```
DROP VIEW [ IF EXISTS ] view_name
```

예제

```
DROP VIEW orders_by_date
```

```
DROP FORCE VIEW IF EXISTS orders_by_date
```

뷰가 없는 경우 선택적 IF EXISTS 절이 오류가 억제되도록 합니다.

SHOW COLUMNS

지정된 단일 데이터 카탈로그 뷰의 열 이름만 표시합니다. `SHOW COLUMNS` 문은 Athena 뷰의 `SHOW COLUMNS` 문과 유사합니다. Athena의 뷰와 달리 명령문의 출력은 Lake Formation 액세스 제어를 통해 제어됩니다. 따라서 이 쿼리의 출력은 뷰의 모든 열이 아니라 호출자가 액세스할 수 있는 열입니다.

구문

```
SHOW COLUMNS {FROM|IN} database_name.view_name
```

```
SHOW COLUMNS {FROM|IN} view_name [{FROM|IN} database_name]
```

SHOW CREATE VIEW

Data Catalog 뷰를 생성한 SQL 구문을 표시합니다. 반환된 SQL은 Athena에서 사용되는 뷰 생성 구문을 보여줍니다. Lake Formation 관리자 및 뷰 정의자 보안 주체만 Data Catalog 뷰에서 `SHOW CREATE VIEW`를 직접적으로 호출할 권한이 있습니다.

구문

```
SHOW CREATE VIEW view_name
```

예제

```
SHOW CREATE VIEW orders_by_date
```

SHOW VIEWS

데이터베이스에 있는 모든 뷰의 이름을 나열합니다. Athena 엔진 SQL 언어가 있는 데이터베이스의 모든 Data Catalog 뷰가 나열됩니다. 뷰에 Athena 엔진 언어가 없는 다른 Data Catalog 뷰는 필터링되어 제외됩니다.

구문

```
SHOW VIEWS [IN database_name] [LIKE 'regular_expression']
```

예제

```
SHOW VIEWS
```

```
SHOW VIEWS IN marketing_analytics LIKE 'orders*'
```

저장된 쿼리 사용

Athena 콘솔을 사용하여 Athena 쿼리 편집기에서 만든 쿼리를 저장하고, 편집하고, 실행하고, 이름을 바꾸고, 삭제할 수 있습니다.

고려 사항 및 제한

- 저장된 쿼리의 이름, 설명 및 쿼리 텍스트를 업데이트할 수 있습니다.
- 사용자는 계정의 쿼리만 업데이트할 수 있습니다.
- 쿼리가 속한 작업 그룹이나 데이터베이스는 변경할 수 없습니다.
- Athena 쿼리 수정 기록을 보관하지 않습니다. 특정 버전의 쿼리를 유지하려면 다른 이름으로 저장합니다.

Athena 콘솔에서 저장된 쿼리를 사용한 작업

쿼리 저장 및 이름 지정

1. Athena 콘솔 쿼리 편집기에서 쿼리를 입력 또는 실행합니다.
2. 쿼리 편집기 창 위의 쿼리 탭에서 세로 점 세 개를 선택한 다음 Save as(다른 이름으로 저장)를 선택합니다.
3. 이름 선택(Choose a name) 대화 상자에서 쿼리의 이름과 설명(선택 사항)을 입력합니다. 확장 가능한 SQL 쿼리 미리 보기 창을 사용하여 저장하기 전에 쿼리의 내용을 확인할 수 있습니다.
4. Save query(쿼리 저장)를 선택합니다.

   쿼리 편집기의 쿼리 탭에는 지정한 이름이 표시됩니다.

저장된 쿼리를 실행하려면

1. Athena 콘솔에서 저장된 쿼리(Saved queries) 탭을 선택합니다.
2. Saved queries(저장된 쿼리) 목록에서 실행할 쿼리의 ID를 선택합니다.

   쿼리 편집기에 선택한 쿼리가 표시됩니다.
3. Run(실행)을 선택합니다.

저장된 쿼리 편집

1. Athena 콘솔에서 저장된 쿼리(Saved queries) 탭을 선택합니다.
2. Saved queries(저장된 쿼리) 목록에서 편집할 쿼리의 ID를 선택합니다.
3. 쿼리 편집기에서 필요에 따라 쿼리를 편집합니다.
4. 다음 단계 중 하나를 수행하세요.
   - 그런 다음 실행(Run)을 선택하여 쿼리를 실행합니다.
   - 쿼리를 저장하려면 쿼리 탭에서 세로 점 세 개를 선택한 다음 Save(저장)를 선택합니다.
   - 쿼리를 다른 이름으로 저장하려면 쿼리 탭에서 세로 점 세 개를 선택한 다음 Save as(다른 이름으로 저장)를 선택합니다.

쿼리 편집기에 이미 표시된 저장된 쿼리를 삭제하거나 이름을 바꾸려면

1. 쿼리 탭에서 세로 점 세 개를 선택한 다음 Rename(이름 바꾸기) 또는 Delete(삭제)를 선택합니다.
2. 프롬프트에 따라 쿼리를 삭제하거나 이름을 바꿉니다.

쿼리 편집기에 표시되지 않은 저장된 쿼리의 이름을 바꾸려면

1. Athena 콘솔에서 저장된 쿼리(Saved queries) 탭을 선택합니다.
2. 이름을 바꾸려는 쿼리의 확인란을 선택합니다.
3. 이름 바꾸기를 선택합니다.
4. 쿼리 이름 바꾸기(Rename query) 대화 상자에서 쿼리 이름과 쿼리 설명을 편집합니다. 확장 가능한 SQL 쿼리 미리 보기 창을 사용하여 이름을 바꾸기 전에 내용을 확인할 수 있습니다.
5. 쿼리 이름 바꾸기(Rename query)를 선택합니다.

   이름이 바뀐 쿼리가 저장된 쿼리(Saved queries) 목록에 표시됩니다.

쿼리 편집기에 표시되지 않은 저장된 쿼리를 삭제하려면

1. Athena 콘솔에서 저장된 쿼리(Saved queries) 탭을 선택합니다.
2. 하나 이상의 삭제하려는 쿼리의 체크박스를 선택합니다.
3. 삭제를 선택합니다.
4. 확인 프롬프트에서 Delete(삭제)를 선택합니다.

Saved queries(저장된 쿼리) 목록에서 하나 이상의 쿼리가 제거됩니다.

Athena API를 사용하여 저장된 쿼리 업데이트

Athena API를 사용한 저장된 쿼리를 업데이트하는 방법에 대한 자세한 내용은 Athena API 참조의 UpdateNamedQuery 섹션을 참조하세요.

파라미터화된 쿼리 사용

Athena 파라미터화된 쿼리를 사용하여 실행 시 동일한 쿼리를 다른 파라미터 값으로 다시 실행하고 SQL 명령어 삽입 공격을 방지할 수 있습니다. Athena에서 파라미터화된 쿼리는 모든 DML 쿼리나 SQL 준비된 문에서 실행 파라미터 형태를 취할 수 있습니다.

- 실행 파라미터가 있는 쿼리는 한 단계로 수행할 수 있으며 작업 그룹에 특정되지 않습니다. 파라미터화하려는 값의 아무 DML 쿼리에 물음표를 넣습니다. 쿼리를 실행할 때 실행 파라미터 값을 순차적으로 선언합니다. 파라미터 선언과 파라미터 값 할당은 동일한 쿼리에서 분리된 방식으로 수행할 수 있습니다. 준비된 문과 달리 실행 파라미터를 사용하여 쿼리를 제출할 경우 작업 그룹을 선택할 수 있습니다.
- 준비된 문에는 두 개의 개별 SQL 문인 PREPARE 및 EXECUTE가 필요합니다. 먼저 PREPARE 문에서 파라미터를 정의합니다. 그런 다음 정의한 파라미터의 값을 제공하는 EXECUTE 문을 실행합니다. 준비된 문은 작업 그룹마다 다르므로 해당 문이 속한 작업 그룹의 컨텍스트 외부에서는 실행할 수 없습니다.

고려 사항 및 제한

- 파라미터화된 쿼리는 Athena 엔진 버전 2 이상에서만 지원됩니다. Athena 엔진 버전에 대한 자세한 내용은 Athena 엔진 버전 관리 단원을 참조하세요.
- 현재 파라미터화된 쿼리는 SELECT, INSERT INTO, CTAS, UNLOAD 문에만 지원됩니다.
- 파라미터화된 쿼리에서 파라미터는 위치 파라미터이며 ?로 표시됩니다. 파라미터에는 쿼리의 순서에 따라 값이 할당됩니다. 명명된 파라미터는 지원되지 않습니다.
- 현재 ? 파라미터는 WHERE절에만 배치할 수 있습니다. SELECT ? FROM table 등의 구문은 지원되지 않습니다.
- 물음표 파라미터는 큰 따옴표나 작은 따옴표로 묶을 수 없습니다(즉,'?' 및 "?"의 경우 잘못된 구문임).
- SQL 실행 파라미터를 문자열로 처리하려면 큰따옴표 대신 작은따옴표로 묶어야 합니다.

- 필요한 경우 파라미터화된 용어의 값을 입력할 때 CAST 함수를 사용할 수 있습니다. 예를 들어 쿼리에서 파라미터화한 date 유형의 열이 있고 날짜 2014-07-05에 대해 쿼리하려는 경우 파라미터 값으로 CAST('2014-07-05' AS DATE)를 입력하면 결과가 반환됩니다.
- 준비된 문은 작업 그룹에 특정하며 준비된 문 이름은 작업 그룹 내에서 고유해야 합니다.
- 준비된 문에 대한 IAM 권한이 필요합니다. 자세한 내용은 준비된 문에 대한 액세스 허용 단원을 참조하십시오.
- Athena 콘솔에서 실행 파라미터가 있는 쿼리는 최대 25개의 물음표로 제한됩니다.

실행 파라미터를 사용한 쿼리

DML 쿼리에서 물음표 자리표시자를 사용하여 준비된 문을 먼저 만들지 않고도 파라미터화된 쿼리를 만들 수 있습니다. 이러한 쿼리를 실행하려면 Athena 콘솔을 사용하거나 AWS CLI 또는 AWS SDK를 사용하고 execution-parameters 인수에 변수를 선언합니다.

Athena 콘솔에서 실행 파라미터와 함께 쿼리 실행

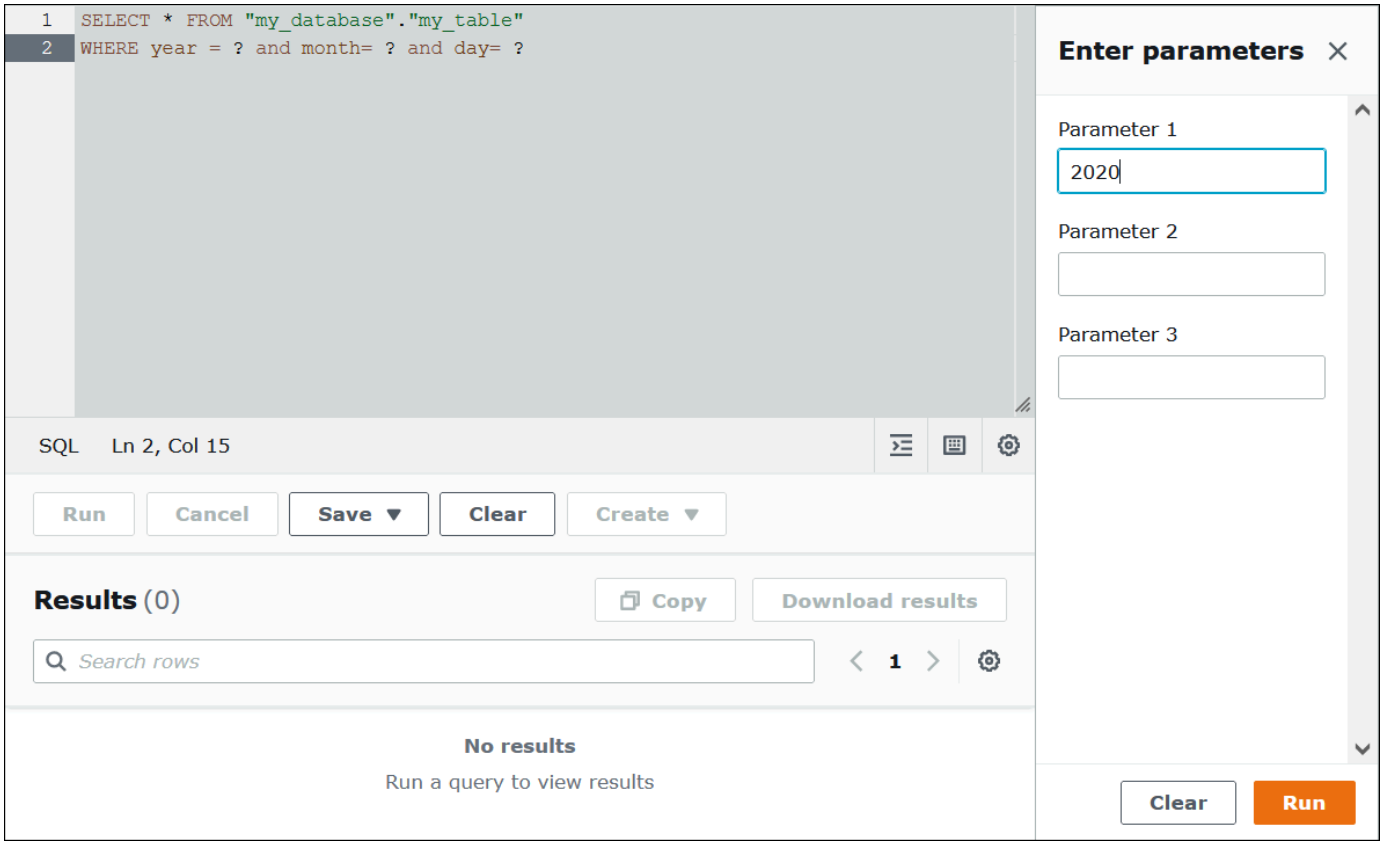
Athena 콘솔에서 실행 파라미터(물음표)가 있는 파라미터화된 쿼리를 실행하면, 쿼리에 물음표가 나타나는 순서대로 값을 입력하라는 메시지가 표시됩니다.

실행 파라미터가 있는 쿼리를 실행하려면

1. 다음 예제와 같이 Athena 편집기에서 물음표 자리표시자와 함께 쿼리를 입력합니다.

```
SELECT * FROM "my_database"."my_table"
WHERE year = ? and month= ? and day= ?
```

2. Run(실행)을 선택합니다.
3. 파라미터 입력(Enter parameters) 대화 상자에서 쿼리에 포함된 각 물음표의 순서대로 값을 입력합니다.



4. 파라미터 입력이 끝나면 실행(Run)을 선택합니다. 입력한 파라미터 값에 대한 쿼리 결과가 편집기에 표시됩니다.

이때 다음 중 한 가지를 수행할 수 있습니다.

- 동일한 쿼리에 대해 다른 파라미터 값을 입력한 다음 다시 실행(Run again)을 선택합니다.
- 입력한 모든 값을 한 번에 지우려면 지우기(Clear)를 선택합니다.
- 쿼리를 직접 편집하려면(예: 물음표 추가 또는 제거) 파라미터 입력(Enter parameters) 대화 상자를 먼저 닫습니다.
- 나중에 사용할 수 있도록 파라미터화된 쿼리를 저장하려면 저장(Save) 또는 다른 이름으로 저장(Save as)을 선택한 다음 쿼리에 이름을 지정합니다. 저장된 쿼리 사용에 관한 자세한 내용은 저장된 쿼리 사용 단원을 참조하세요.

편의상, 쿼리 편집기에서 동일한 탭을 사용하는 한 파라미터 입력(Enter parameters) 대화 상자에는 쿼리에 대해 이전에 입력한 값이 저장됩니다.

AWS CLI를 사용하여 실행 파라미터로 쿼리 실행

AWS CLI를 사용하여 실행 파라미터로 쿼리를 실행하려면 `start-query-execution` 명령을 사용하고 `query-string` 인수에 파라미터화된 쿼리를 제공합니다. 그런 다음 `execution-parameters` 인수에 실행 파라미터 값을 제공합니다. 다음 예는 이 기법을 보여 줍니다.

```
aws athena start-query-execution
--query-string "SELECT * FROM table WHERE x = ? AND y = ?"
--query-execution-context "Database"="default"
--result-configuration "OutputLocation"="s3://DOC-EXAMPLE-BUCKET;/..."
--execution-parameters "1" "2"
```

준비된 문을 사용한 쿼리

서로 다른 쿼리 파라미터로 동일한 쿼리를 반복 실행하도록 준비된 문을 사용할 수 있습니다. 준비된 문 에는 실행 시 값이 제공되는 파라미터 자리 표시자가 포함됩니다.

> **Note**
>
> 작업 그룹에서 준비된 문의 최대 개수는 1000입니다.

SQL 문

PREPARE, EXECUTE, DEALLOCATE PREPARE SQL 문을 사용하여 Athena 콘솔 쿼리 편집기에서 파라미터화된 쿼리를 실행할 수 있습니다.

- 일반적으로 리터럴 값을 사용하는 파라미터를 지정하기 위해 PREPARE문에 물음표를 사용합니다.
- 쿼리를 실행할 때 파라미터를 값으로 바꾸려면 EXECUTE 문에 USING 절을 사용합니다.
- 작업 그룹의 준비된 문에서 준비된 문을 제거하려면 DEALLOCATE PREPARE 문을 사용합니다.

다음 단원에서는 이러한 문 각각에 대한 추가 정보를 제공합니다.

PREPARE

나중에 실행할 문을 준비합니다. 준비된 문은 사용자가 지정한 이름으로 현재 작업 그룹에 저장됩니다. 이 문은 쿼리가 실행될 때 리터럴을 대신해 대체될 파라미터를 포함할 수 있습니다. 값으로 대체할 파라미터는 물음표로 표시됩니다.

구문

```
PREPARE statement_name FROM statement
```

다음 표는 이러한 파라미터에 대해 설명합니다.

파라미터	설명
statement_name	준비되는 문의 이름입니다. 이 이름은 작업 그룹 내에서 고유해야 합니다.
##	SELECT, CTAS 또는 INSERT INTO 쿼리.

PREPARE 예제

다음 예제는 PREPARE 문의 사용을 보여줍니다. 물음표는 쿼리 실행 시 EXECUTE 문이 제공할 값을 나타냅니다.

```
PREPARE my_select1 FROM
SELECT * FROM nation
```

```
PREPARE my_select2 FROM
SELECT * FROM "my_database"."my_table" WHERE year = ?
```

```
PREPARE my_select3 FROM
SELECT order FROM orders WHERE productid = ? and quantity < ?
```

```
PREPARE my_insert FROM
INSERT INTO cities_usa (city, state)
SELECT city, state
FROM cities_world
WHERE country = ?
```

```
PREPARE my_unload FROM
UNLOAD (SELECT * FROM table1 WHERE productid < ?)
TO 's3://DOC-EXAMPLE-BUCKET/'
WITH (format='PARQUET')
```

EXECUTE

준비된 문을 실행합니다. 파라미터의 값은 USING 절에서 지정합니다.

구문

```
EXECUTE statement_name [USING value1 [ ,value2, ... ] ]
```

*statement_name*은 준비된 문의 이름입니다. *value1* 및 *value2*는 문의 파라미터에 지정될 값입니다.

EXECUTE 예제

다음 예제는 파라미터를 포함하지 않는 준비된 문 my_select1을 실행합니다.

```
EXECUTE my_select1
```

다음 예제는 하나의 파라미터를 포함한 준비된 문 my_select2를 실행합니다.

```
EXECUTE my_select2 USING 2012
```

다음 예제는 2개의 파라미터를 포함한 준비된 문 my_select3을 실행합니다.

```
EXECUTE my_select3 USING 346078, 12
```

다음 예제는 준비된 문 my_insert의 파라미터에 문자열 값을 제공합니다.

```
EXECUTE my_insert USING 'usa'
```

다음 예제는 준비된 문 my_unload의 productid 파라미터에 숫자 값을 제공합니다.

```
EXECUTE my_unload USING 12
```

DEALLOCATE PREPARE

현재 작업 그룹의 준비된 문 목록에서 지정된 이름을 가진 준비된 문을 제거합니다.

구문

```
DEALLOCATE PREPARE statement_name
```

*statement_name*은 제거할 준비된 문의 이름입니다.

예

다음 예제는 현재 작업 그룹에서 준비된 문 my_select1을 제거합니다.

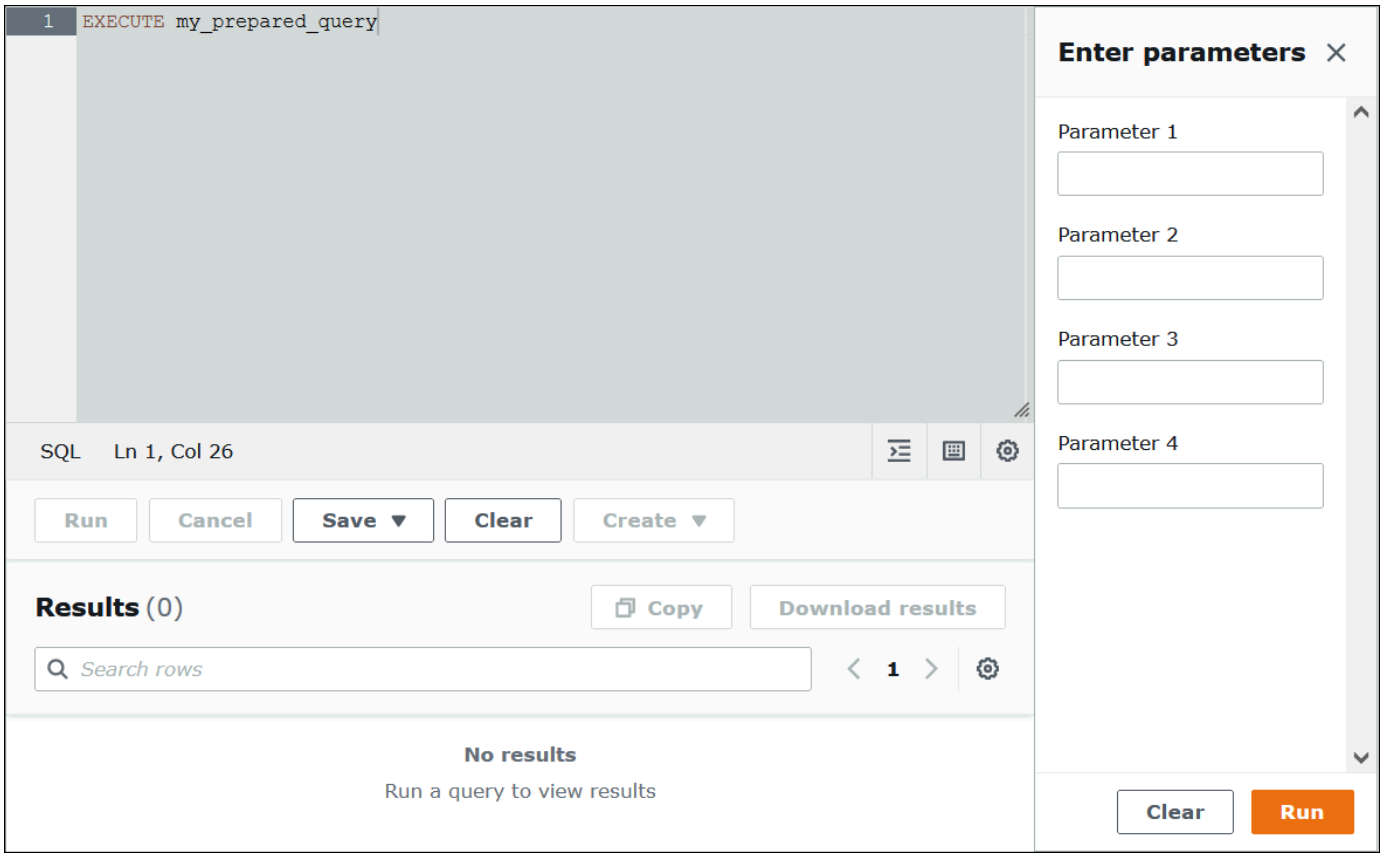
```
DEALLOCATE PREPARE my_select1
```

Athena 콘솔에서 USING 절 없이 준비된 문 실행

쿼리 편집기에서 EXECUTE *prepared_statement* 구문을 사용하여 기존의 준비된 문을 실행하는 경우, Athena에서는 파라미터 입력(Enter parameters) 대화 상자가 열려 EXECUTE ... USING 문의 USING 절에 일반적으로 입력되는 값을 입력할 수 있습니다.

파라미터 입력(Enter parameters) 대화 상자를 사용하여 준비된 문을 실행하려면

1. 쿼리 편집기에서 EXECUTE prepared_statement USING *value1*, *value2* ... 구문을 사용하는 대신 EXECUTE *prepared_statement* 구문을 사용합니다.
2. Run(실행)을 선택합니다. 파라미터 입력(Enter parameters) 대화 상자가 표시됩니다.



3. 실행 파라미터(Execution parameters) 대화 상자에 순서에 따라 값을 입력합니다. 쿼리의 원래 텍스트가 표시되지 않으므로 각 위치 파라미터의 의미를 기억하거나 준비된 문을 참조할 수 있어야 합니다.
4. Run(실행)을 선택합니다.

AWS CLI를 사용하여 준비된 문 만들기

AWS CLI를 사용해 준비된 문을 작성하는 데 다음 athena 명령 중 하나를 사용할 수 있습니다.

- `create-prepared-statement` 명령을 사용하고 실행 파라미터가 있는 쿼리 문을 제공합니다.
- `start-query-execution` 명령을 사용하고 PREPARE 구문을 사용하는 쿼리 문자열을 제공합니다.

create-prepared-statement 사용

`create-prepared-statement` 명령에서 다음 예제와 같이 `query-statement` 인수에 쿼리 텍스트를 정의합니다.

```
aws athena create-prepared-statement
--statement-name PreparedStatement1
--query-statement "SELECT * FROM table WHERE x = ?"
--work-group athena-engine-v2
```

start-query-execution 및 PREPARE 구문 사용

`start-query-execution` 명령을 사용합니다. 다음 예제와 같이 `query-string` 인수에 PREPARE 문을 넣습니다.

```
aws athena start-query-execution
--query-string "PREPARE PreparedStatement1 FROM SELECT * FROM table WHERE x = ?"
--query-execution-context '{"Database": "default"}'
--result-configuration '{"OutputLocation": "s3://DOC-EXAMPLE-BUCKET/..."}'
```

AWS CLI를 사용해 준비된 문 실행

AWS CLI를 사용해 준비된 문을 실행하는 데 다음과 같은 메서드 중 하나를 사용하여 파라미터 값을 제공할 수 있습니다.

- `execution-parameters` 인수를 사용합니다.

- `query-string` 인수에 `EXECUTE ... USING` SQL 구문을 사용합니다.

execution-parameters 인수 사용

이러한 방식에서는 `start-query-execution` 명령을 사용하고 `query-string` 인수에 기존 준비된 문의 이름을 제공합니다. 그런 다음 `execution-parameters` 인수에 실행 파라미터 값을 제공합니다. 다음 예제는 이 메서드를 보여 줍니다.

```
aws athena start-query-execution
--query-string "Execute PreparedStatement1"
--query-execution-context "Database"="default"
--result-configuration "OutputLocation"="s3://DOC-EXAMPLE-BUCKET/..."
--execution-parameters "1" "2"
```

EXECUTE 사용 중... SQL 구문 사용

`EXECUTE ... USING`구문을 사용하여 기존 준비된 문을 실행하려면, `start-query-execution` 명령을 사용하고 다음 예제와 같이 `query-string` 인수에 준비된 명령문의 이름과 파라미터 값을 모두 배치합니다.

```
aws athena start-query-execution
--query-string "EXECUTE PreparedStatement1 USING 1"
--query-execution-context '{"Database": "default"}'
--result-configuration '{"OutputLocation": "s3://DOC-EXAMPLE-BUCKET/..."}'
```

준비된 문 나열

특정 작업 그룹에 대해 준비된 문을 나열하기 위해 Athena list-prepared-statements AWS CLI 명령 또는 ListPreparedStatements Athena API 작업을 사용할 수 있습니다. `--work-group` 파라미터가 필요합니다.

```
aws athena list-prepared-statements --work-group primary
```

추가적인 리소스

AWS 빅 데이터 블로그에서 다음 관련 게시물을 참조하세요.

- Amazon Athena 파라미터화된 쿼리를 사용하여 재사용성 및 보안 개선
- Amazon Athena 파라미터화된 쿼리를 사용하여 데이터를 서비스로 제공

비용 기반 최적화 프로그램 사용

Athena SQL의 비용 기반 최적화 프로그램(CBO) 기능을 사용하여 쿼리를 최적화할 수 있습니다. AWS Glue의 테이블 중 하나에 대해 Athena가 테이블 또는 열 수준의 통계를 수집하도록 요청할 수도 있습니다. 쿼리의 모든 테이블에 통계가 있는 경우 Athena는 통계를 사용하여 가장 성능이 뛰어난 것으로 판단되는 실행 계획을 생성합니다. 쿼리 최적화 프로그램은 통계 모델을 기반으로 대체 계획을 계산한 다음 쿼리를 실행하기에 가장 빠를 것 같은 모델을 선택합니다.

AWS Glue 테이블에 대한 통계는 AWS Glue Data Catalog에 수집되고 저장되며 향상된 쿼리 계획 및 실행을 위해 Athena에 제공됩니다. 이러한 통계는 Parquet, ORC, JSON, ION, CSV, XML 등의 파일 유형에 대한 고유 수, null 수, max 및 min 값과 같은 열 수준 통계입니다. Amazon Athena는 이러한 통계를 사용하여 쿼리 처리 시 최대한 빨리 가장 제한적인 필터를 적용하여 쿼리를 최적화합니다. 이 필터링은 메모리 사용량과 쿼리 결과를 전달하기 위해 읽어야 하는 레코드 수를 제한합니다.

Athena는 CBO와 함께 규칙 기반 최적화 프로그램(RBO)이라는 기능을 사용합니다. RBO는 쿼리 성능 향상이 예상되는 규칙을 기계적으로 적용합니다. RBO의 변환은 쿼리 계획 단순화를 목표로 하기 때문에 일반적으로 유용합니다. 그러나 RBO는 비용 계산이나 계획 비교를 수행하지 않기 때문에 쿼리가 복잡할수록 RBO가 최적의 계획을 세우기 어렵습니다.

이러한 이유로 Athena는 RBO와 CBO를 모두 사용하여 쿼리를 최적화합니다. Athena는 쿼리 실행을 개선할 기회를 식별한 후 최적의 계획을 세웁니다. 실행 계획 세부 정보에 대한 자세한 내용은 SQL 쿼리에 대한 실행 계획 보기 섹션을 참조하세요. CBO 작동 방식에 대한 자세한 내용은 AWS 빅 데이터 비용 기반 최적화 프로그램 블로그 게시물을 참조하세요.

AWS Glue 카탈로그 테이블에 대한 통계를 생성하려면 Athena 콘솔, AWS Glue 콘솔 또는 AWS Glue API를 사용할 수 있습니다. Athena는 AWS Glue 카탈로그와 통합되어 있으므로 Amazon Athena에서 쿼리를 실행할 때 해당 쿼리 성능이 자동으로 향상됩니다.

고려 사항 및 제한

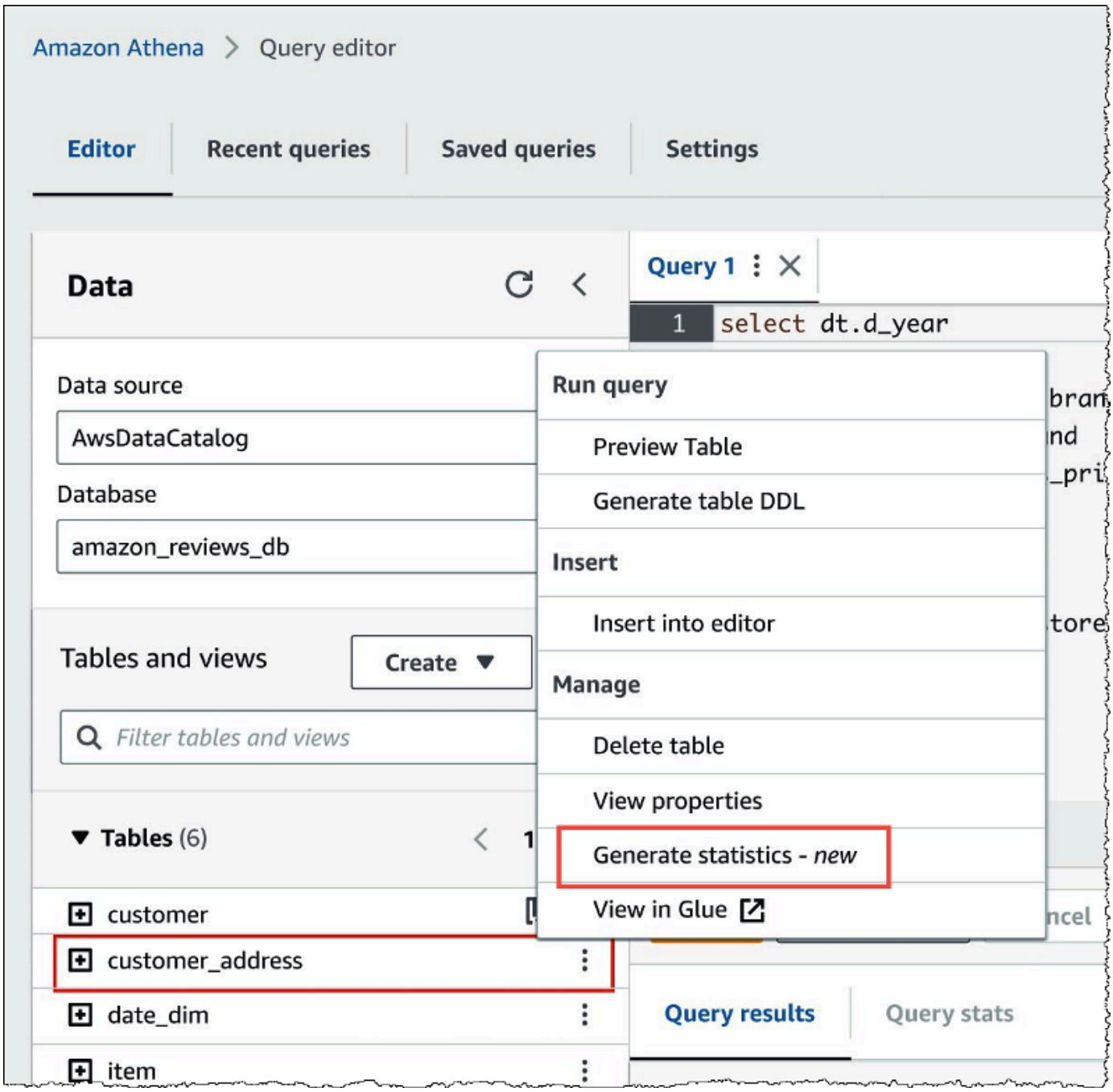
- 테이블 유형 - 현재 Athena의 CBO 기능은 AWS Glue Data Catalog에 있는 Hive 테이블만 지원합니다.
- Athena for Spark - CBO 기능은 Athena for Spark에서 사용할 수 없습니다.
- 요금 - 요금 정보는 AWS Glue 요금 페이지를 참조하세요.

Athena 콘솔을 사용하여 테이블 통계 생성

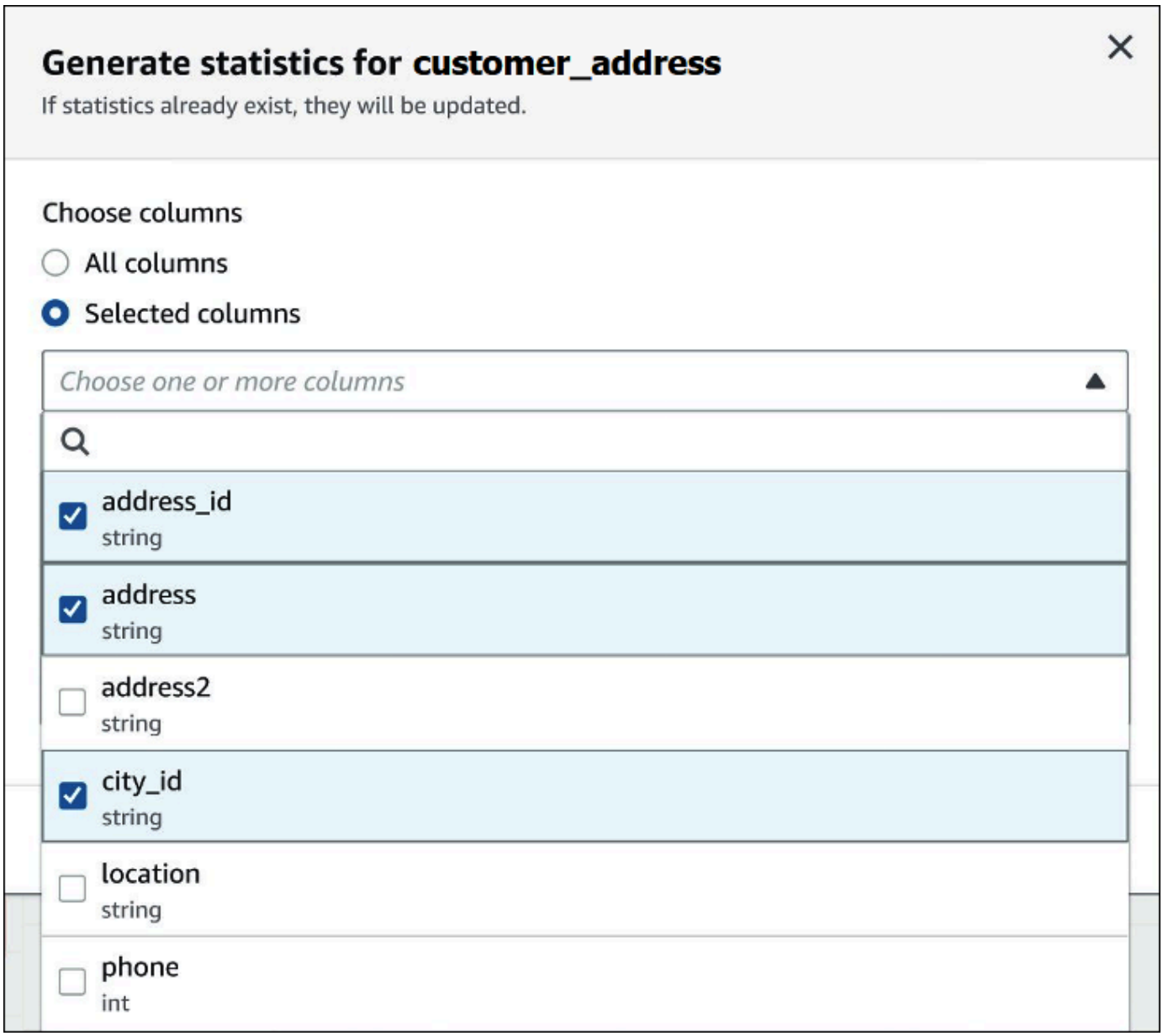
이 섹션에서는 Athena 콘솔을 사용하여 AWS Glue의 테이블에 대한 테이블 또는 열 수준 통계를 생성하는 방법을 설명합니다. AWS Glue를 사용하여 테이블 통계를 생성하는 방법에 대한 자세한 내용은 AWS Glue 개발자 안내서의 Working with column statistics를 참조하세요.

Athena 콘솔을 사용하여 테이블에 대한 통계 생성

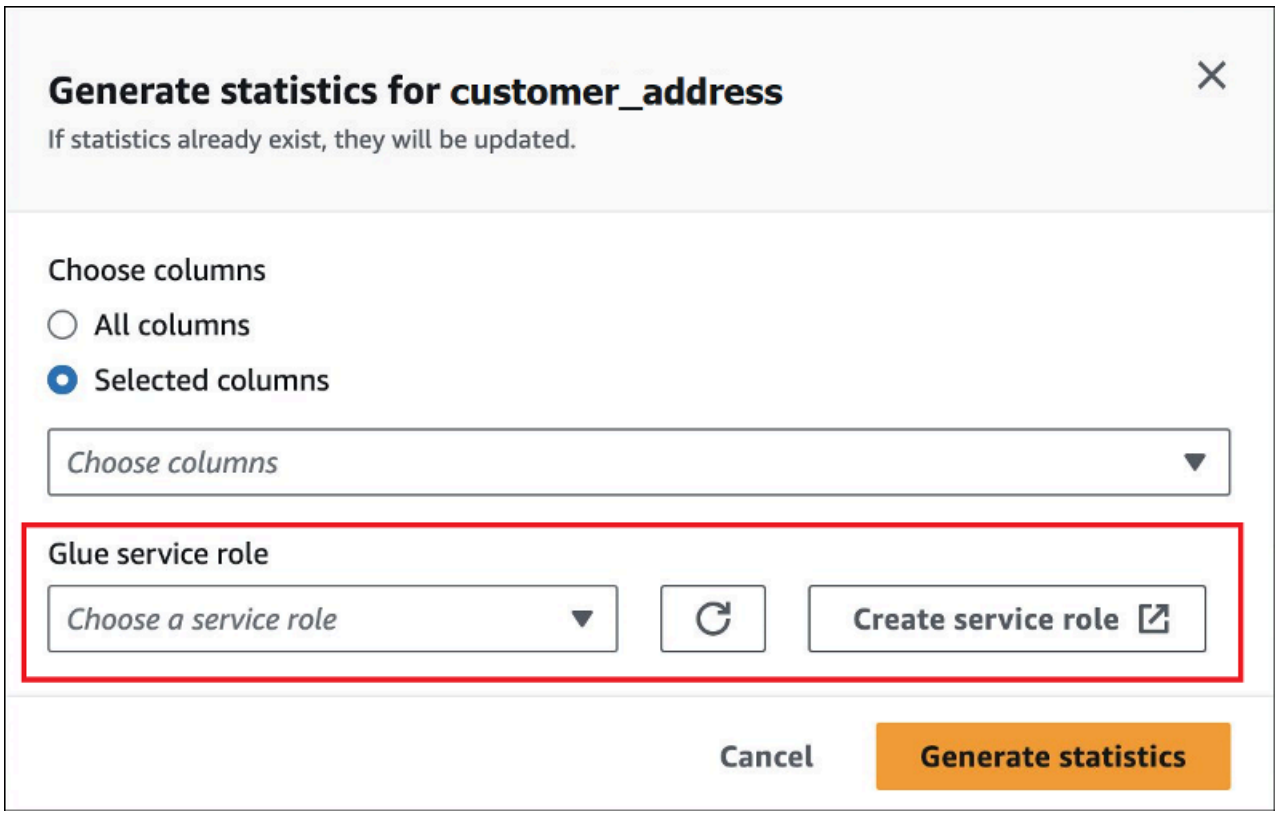
1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. Athena 쿼리 편집기 테이블 목록에서 원하는 테이블에 대한 3개의 세로 점을 선택한 다음 통계 생성을 선택합니다.



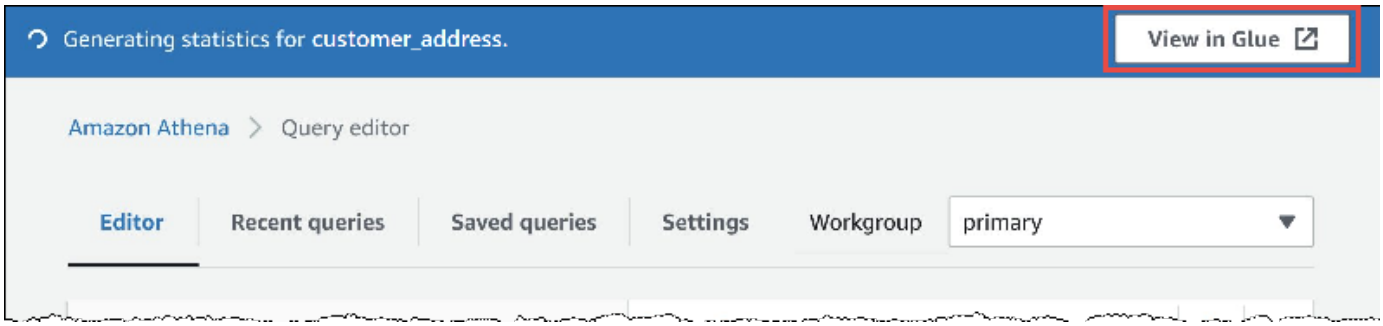
3. 통계 생성 대화 상자에서 모든 열을 선택하여 테이블의 모든 열에 대한 통계를 생성하거나 선택한 열을 선택하여 특정 열을 선택합니다. 모든 열이 기본값입니다.



4. AWS Glue 서비스 역할의 경우 기존 서비스 역할을 생성하거나 선택하여 AWS Glue에 통계 생성 권한을 부여합니다. 또한 AWS Glue 서비스 역할에는 테이블의 데이터가 포함된 Amazon S3 버킷에 대한 `S3:GetObject` 권한이 필요합니다.



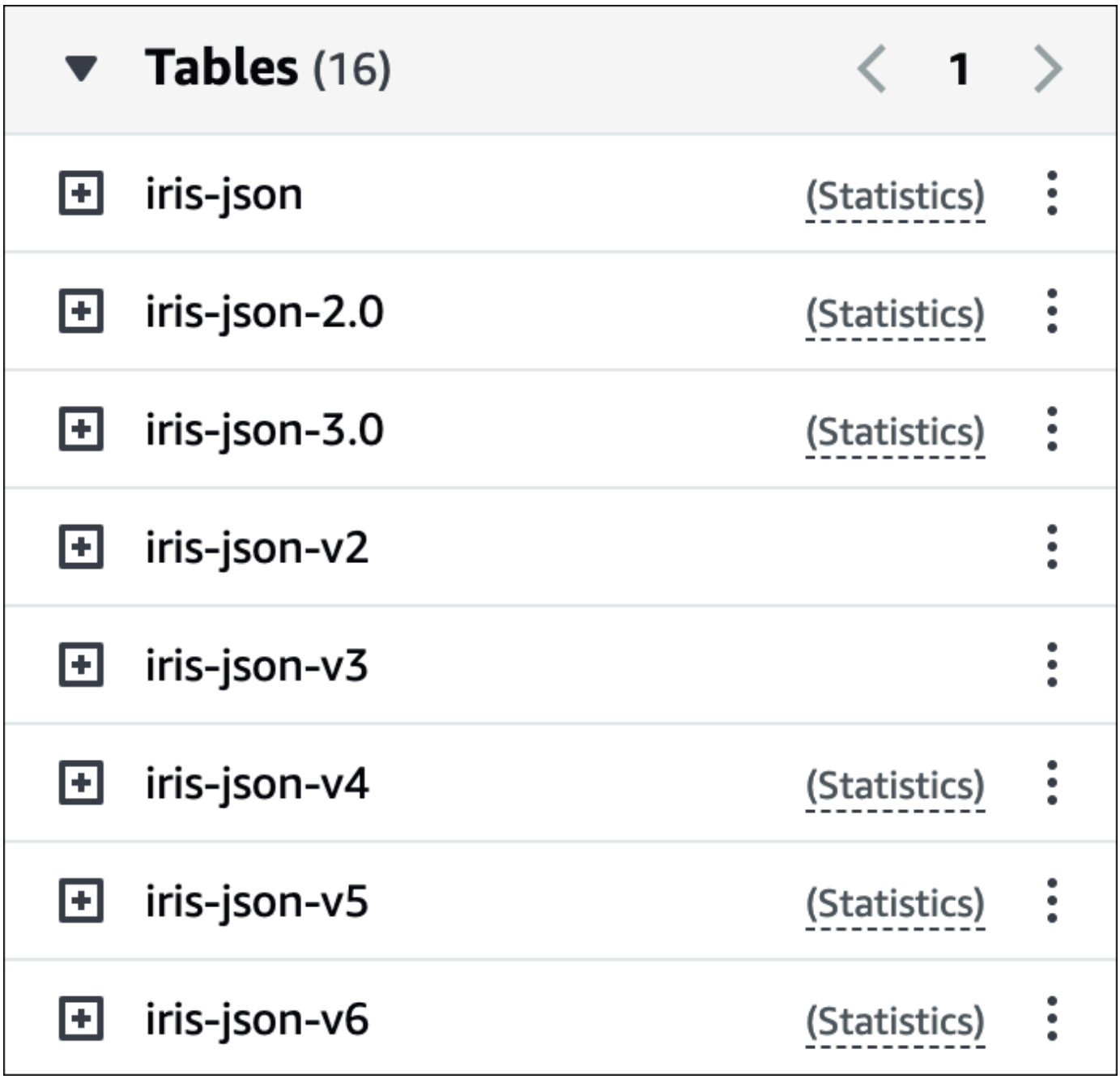
5. 통계 생성을 선택합니다. ***table_name***에 대한 통계 생성 알림 배너에 작업 상태가 표시됩니다.



6. AWS Glue 콘솔에서 세부 정보를 보려면 Glue에서 보기를 선택합니다.

   AWS Glue 콘솔에서 통계를 보는 방법에 대한 자세한 내용은 AWS Glue 개발자 안내서의 Viewing column statistics를 참조하세요.

7. 통계가 생성된 후에는 다음 이미지와 같이 통계가 있는 테이블과 열에 통계라는 단어가 괄호 안에 표시됩니다.



이제 쿼리를 실행하면 Athena는 통계가 생성된 테이블과 열에 대해 비용 기반 최적화를 수행합니다.

추가적인 리소스

자세한 내용은 다음 리소스를 참조하세요.

S3 Express One Zone 데이터 쿼리

Amazon S3 Express One Zone 스토리지 클래스는 10밀리초 미만의 응답 시간을 제공하는 고성능 Amazon S3 스토리지 클래스입니다. 따라서 초당 수십만 개의 요청으로 데이터에 자주 액세스하는 애플리케이션에 유용합니다.

S3 Express One Zone은 동일한 가용 영역 내에서 데이터를 복제하고 저장하여 속도와 비용을 최적화합니다. 이는 AWS 리전 내 최소 3개의 AWS 가용 영역에 걸쳐 데이터를 자동으로 복제하는 Amazon S3 리전별 스토리지 클래스와 다릅니다.

자세한 내용은 Amazon S3 사용 설명서의 What is S3 Express One Zone?을 참조하십시오.

필수 조건

시작하기 전에 다음 조건이 충족되는지 확인합니다.

- Athena 엔진 버전 3 - Athena SQL과 함께 S3 Express One Zone을 사용하려면 Athena 엔진 버전 3을 사용하도록 작업 그룹을 구성해야 합니다.
- S3 Express One Zone 권한 - S3 Express One Zone이 Amazon S3 객체에 대해 `GET`, `LIST` 또는 `PUT`과 같은 작업을 호출하면 스토리지 클래스가 사용자를 대신하여 `CreateSession`을 호출합니다. 따라서 IAM 정책은 Athena가 해당 API 작업을 간접적으로 호출할 수 있도록 `s3express:CreateSession` 작업을 허용해야 합니다.

고려 사항 및 제한

Athena를 사용하여 S3 Express One Zone을 쿼리할 때 다음 사항을 고려하십시오.

- S3 Express One Zone 버킷은 `SSE_S3` 암호화만 지원합니다. Athena 쿼리 결과는 쿼리 결과를 암호화하기 위해 작업 그룹 설정에서 선택한 옵션에 관계없이 `SSE_S3` 암호화를 사용하여 작성됩니다. 이 제한에는 `CREATE TABLE AS`(CTAS) 및 `INSERT INTO` 문을 포함하여 Athena가 S3 Express One Zone 버킷에 데이터를 쓰는 모든 시나리오가 포함됩니다.
- S3 Express One Zone 데이터에 대한 테이블 생성에는 AWS Glue 크롤러가 지원되지 않습니다.
- `MSCK REPAIR TABLE` 문은 지원되지 않습니다. 임시 해결책으로 ALTER TABLE ADD PARTITION을 사용합니다.
- 다음 파일 및 테이블 형식은 지원되지 않거나 지원이 제한적입니다. 목록에 없지만 Athena에 대해 지원되는 형식(예: Parquet, ORC 및 JSON)은 S3 Express One Zone 스토리지에서도 사용할 수 있습니다.

파일 또는 테이블 형식	제한
Apache Avro	지원되지 않음
CloudTrail 로그	지원되지 않음
Apache Hudi	지원되지 않음
Amazon Ion	지원되지 않음
Logstash 로그	지원되지 않음
Apache WebServer 로그	지원되지 않음
Delta Lake	DDL은 지원되지 않습니다. 더미 스키마를 사용하여 Delta Lake 테이블을 생성하는 방법에 대한 자세한 내용은 Delta Lake 메타데이터 동기화 섹션을 참조하십시오. 테이블에 대한 SELECT 쿼리가 지원됩니다.

시작하기

Athena를 사용하여 S3 Express One Zone 데이터를 쿼리하는 것은 간단합니다. 시작하려면 다음 절차를 따르세요.

Athena SQL을 사용하여 S3 Express One Zone 데이터 쿼리

1. 데이터를 S3 Express One Zone 스토리지로 전환합니다. 자세한 내용은 Amazon S3 사용 설명서의 객체의 스토리지 클래스 설정을 참조하십시오.
2. Athena에서 CREATE TABLE 문을 사용하여 AWS Glue Data Catalog에서 데이터를 카탈로그화합니다. Athena에서 테이블을 생성하는 방법에 대한 자세한 내용은 Athena에서 테이블 생성 섹션과 CREATE TABLE 문을 참조하십시오.
3. (선택 사항) Amazon S3 디렉터리 버킷을 사용하도록 Athena 작업 그룹의 쿼리 결과 위치를 구성합니다. Amazon S3 디렉터리 버킷은 일반 버킷보다 성능이 뛰어나며 일관되게 10밀리초 미만의 지연 시간이 필요한 워크로드 또는 성능이 중요한 애플리케이션용으로 설계되었습니다. 자세한 내용은 Amazon S3 사용 설명서의 디렉터리 버킷 개요를 참조하십시오.

복원된 Amazon S3 Glacier 객체 쿼리

Athena를 사용하여 S3 Glacier Flexible Retrieval(이전의 Glacier) 및 S3 Glacier Deep Archive Amazon S3 스토리지 클래스로부터 복원된 객체를 쿼리할 수 있습니다. 이 기능은 테이블별로 활성화해야 합니다. 쿼리를 실행하기 전에 테이블에서 이 기능을 활성화하지 않으면 Athena는 쿼리 실행 중에 테이블의 S3 Glacier Flexible Retrieval 및 S3 Glacier Deep Archive 객체를 모두 건너뜁니다.

고려 사항 및 제한

- 복원된 Amazon S3 Glacier 객체 쿼리는 Athena 엔진 버전 3에서만 지원됩니다.
- 이 기능은 Apache Hive 테이블에서만 지원됩니다.
- 데이터를 쿼리하기 전에 객체를 복원해야 합니다. Athena는 자동으로 객체를 복원하지 않습니다.

복원된 객체를 사용하도록 테이블 구성

복원된 객체를 쿼리에 포함하도록 Athena 테이블을 구성하려면 `read_restored_glacier_objects` 테이블 속성을 `true`로 설정해야 합니다. 이 작업을 수행하기 위해 Athena 쿼리 편집기 또는 AWS Glue 콘솔을 사용할 수 있습니다. AWS Glue CLI, AWS Glue API 또는 AWS Glue SDK도 사용할 수 있습니다.

Athena 쿼리 편집기 사용

Athena에서는 다음 예제와 같이 ALTER TABLE SET TBLPROPERTIES 명령을 사용하여 테이블 속성을 설정할 수 있습니다.

```
ALTER TABLE table_name SET TBLPROPERTIES ('read_restored_glacier_objects' = 'true')
```

AWS Glue 콘솔 사용

AWS Glue 콘솔에서 다음 단계를 수행하여 `read_restored_glacier_objects` 테이블 속성을 추가합니다.

AWS Glue 콘솔에서 테이블 속성을 구성하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
2. 다음 중 하나를 수행하십시오.
   - 데이터 카탈로그로 이동을 선택합니다.

- 탐색 창의 데이터 카탈로그 테이블을 선택합니다.

3. 테이블 페이지의 테이블 목록에서 편집하려는 테이블에 대한 링크를 선택합니다.
4. Actions(작업), Edit table(테이블 편집)을 선택합니다.
5. 테이블 편집 페이지의 테이블 속성 섹션에서 다음 키 값 페어를 추가합니다.
   - 키에 `read_restored_glacier_objects`을 추가합니다.
   - 값에 `true`을(를) 입력합니다.
6. Save(저장)를 선택합니다.

AWS CLI 사용

AWS CLI에서 AWS Glue update-table 명령과 해당 `--table-input` 인수를 사용하여 테이블을 재정의하고 이에 따라 `read_restored_glacier_objects` 속성을 추가할 수 있습니다. `--table-input` 인수에서 `Parameters` 구조를 사용하여 `read_restored_glacier_objects` 속성과 `true`의 값을 지정합니다. `--table-input`의 인수에는 공백이 없어야 하며 큰따옴표를 이스케이프 처리하려는 경우 백슬래시를 사용해야 합니다. 다음 예제에서는 *my_database* 및 *my_table*을 사용자의 데이터베이스 및 테이블 이름으로 바꿉니다.

```
aws glue update-table \
   --database-name my_database \
   --table-input={\"Name\":\"my_table\",\"Parameters\":{\"read_restored_glacier_objects
\":\"true\"}}
```

> **Important**
>
> AWS Glue `update-table` 명령은 덮어쓰기 모드에서 작동합니다. 따라서 기존 테이블 정의를 `table-input` 파라미터로 지정된 새 정의로 바꿉니다. 이런 이유로 `read_restored_glacier_objects` 속성을 추가할 때 `table-input` 파라미터에서 테이블에 포함하려는 모든 필드를 지정해야 합니다.

스키마 업데이트 처리

이 단원에서는 다양한 데이터 형식의 스키마 업데이트를 처리하는 방법을 안내합니다. Athena는 스키마-온-리드(schema-on-read) 쿼리 엔진입니다. 즉, Athena에 테이블을 만들면 데이터를 읽을 때 스키마를 적용합니다. 기본 데이터를 변경하거나 다시 작성하지 않습니다.

테이블 스키마 변경이 예측되는 경우, 필요에 적합한 데이터 형식으로 만들어 보세요. 변화하는 스키마에 대해 기존 Athena 쿼리를 다시 사용하고, 파티션이 있는 테이블을 쿼리할 때 스키마 불일치 오류를 방지하는 것이 목표입니다.

이러한 목표를 달성하려면 다음 주제의 테이블을 토대로 테이블의 데이터 형식을 선택합니다.

주제

요약: Athena의 업데이트 및 데이터 형식

다음 표에서는 데이터 스토리지 형식과 지원되는 스키마 조작을 요약합니다. 이 표를 사용하면 시간이 지남에 따라 스키마가 변경되더라도 Athena 쿼리를 계속 사용할 수 있는 형식을 선택하는 데 도움이 됩니다.

이 표에서 Parquet과 ORC가 다른 기본 열 액세스 방법이 서로 다른 열 기반 형식임을 알 수 있습니다. 기본적으로 Parquet은 이름으로 열에 액세스하고 ORC는 인덱스(서수 값)로 액세스합니다. 따라서 Athena는 테이블을 생성할 때 기본 열 액세스 방법을 전환하여 스키마 변화를 통해 유연성을 강화할 수 있도록 정의된 SerDe 속성을 제공합니다.

Parquet의 경우 `parquet.column.index.access` 속성을 `true`로 설정하여 열의 서수 번호를 사용하도록 열 액세스 방법을 설정할 수 있습니다. 이 속성을 `false`로 설정하면 열 이름을 사용하도록 열 액세스 방법이 변경됩니다. 마찬가지로 ORC의 경우 `orc.column.index.access` 속성을 사용하여 열 액세스 방법을 제어합니다. 자세한 내용은 ORC 및 Parquet에서 인덱스 액세스 단원을 참조하십시오.

CSV 및 TSV를 사용하면 열 순서 변경, 테이블 시작 부분에 열 추가를 제외한 모든 스키마 조작을 수행할 수 있습니다. 예를 들어 스키마 혁신을 위해 열을 제거하지 않고 열 이름만 바꾸면 되는 경우 CSV 또는 TSV로 테이블을 생성할 수 있습니다. 열을 제거해야 하는 경우 CSV 또는 TSV를 사용하지 않고 다른 지원되는 형식 중 Parquet, ORC 등과 같은 열 기반 형식을 사용하는 것이 좋습니다.

Athena의 스키마 업데이트 및 데이터 형식

예상 스키마 업데이트 유형	요약	CSV(헤더 포함 및 불포함) 및 TSV	JSO	AVF	PARQUE 이름으로 읽기(기본값)	PARQUE 인덱스로 읽기	ORC: 인덱스로 읽기(기본값)	ORC: 이름으로 읽기
열 이름 바꾸기	CSV 및 TSV 또는 ORC 및 Parquet(인덱스로 읽는 경우) 형식으로 데이터를 저장합니다.	Y	N	N	N	Y	Y	N
테이블의 시작 또는 중간에 열 추가	JSON, AVRO 또는 Parquet 및 ORC(이름으로 읽는 경우) 형식으로 데이터를 저장합니다. CSV 및 TSV를 사용하지 마세요.	N	Y	Y	Y	N	N	Y
테이블 끝에 열 추가	CSV나 TSV, JSON, AVRO, ORC 또는 Parquet 형식으로 데이터를 저장합니다.	Y	Y	Y	Y	Y	Y	Y
열 제거	JSON, AVRO 또는 Parquet 및 ORC(이름으로 읽는 경우) 형식으로 데이터를 저장합니다. CSV 및 TSV	N	Y	Y	Y	N	N	Y

예상 스키마 업데이트 유형	요약	CSV(헤더 포함 및 불포함) 및 TSV	JSO	AVF	PARQUE 이름으로 읽기(기본값)	PARQUE 인덱스로 읽기	ORC: 인덱스로 읽기(기본값)	ORC: 이름으로 읽기
	를 사용하지 마세요.							
열 재정렬	AVRO, JSON 또는 ORC 및 Parquet(이름으로 읽는 경우) 형식으로 데이터를 저장합니다.	N	Y	Y	Y	N	N	Y
열의 데이터 형식 변경	데이터를 어떤 형식으로든 저장하되 Athena에서 쿼리를 테스트하여 데이터 형식이 호환되는지 확인합니다. Parquet과 ORC의 경우 데이터 형식 변경은 분할된 테이블에만 적용됩니다.	Y	Y	Y	Y	Y	Y	Y

ORC 및 Parquet에서 인덱스 액세스

PARQUET 및 ORC는 인덱스 또는 이름으로 읽을 수 있는 열 기반 데이터 스토리지 형식입니다. 데이터를 이러한 형식으로 저장하면 스키마에 대한 모든 작업을 수행하고 스키마 불일치 오류 없이 Athena 쿼리를 실행할 수 있습니다.

- Athena는 `SERDEPROPERTIES ( 'orc.column.index.access'='true')`에 정의된 대로 기본적으로 인덱스를 기준으로 ORC를 읽습니다. 자세한 내용은 ORC: 인덱스로 읽기 단원을 참조하십시오.
- Athena는 `SERDEPROPERTIES ( 'parquet.column.index.access'='false')`에 정의된 대로 기본적으로 이름을 기준으로 Parquet을 읽습니다. 자세한 내용은 Parquet: 이름으로 읽기 단원을 참조하십시오.

이는 기본값이므로 `CREATE TABLE` 쿼리에서 SerDe 속성을 지정하는 것은 선택 사항이며 묵시적으로 사용됩니다. 이 방법을 사용할 경우 스키마 업데이트 작업 중 일부만 실행하고 나머지는 금지할 수 있습니다. 이러한 작업을 사용하려면 다른 `CREATE TABLE` 쿼리를 실행하고 SerDe 설정을 변경하세요.

> **Note**
>
> SerDe 속성은 각 파티션에 자동으로 전파되지 않습니다. `ALTER TABLE ADD PARTITION` 문을 사용하여 각 파티션의 SerDe 속성을 설정합니다. 이 프로세스를 자동화하려면 `ALTER TABLE ADD PARTITION` 문을 실행하는 스크립트를 작성합니다.

다음 섹션에서는 이러한 경우에 대해 상세히 설명합니다.

ORC: 인덱스로 읽기

기본적으로 ORC의 테이블은 인덱스로 읽습니다. 이는 다음 구문으로 정의됩니다.

```
WITH SERDEPROPERTIES (
  'orc.column.index.access'='true')
```

인덱스로 읽기를 사용하면 열 이름을 변경할 수 있습니다. 하지만 테이블 중간에서 열을 제거하거나 추가할 수는 없습니다.

테이블 중간에 열을 추가하거나 ORC에서 열을 제거할 수 있도록 ORC에서 이름으로 읽도록 지정하려면 `CREATE TABLE` 명령문에서 SerDe 속성 `orc.column.index.access`를 `false`로 설정합니다. 이 구성에서는 열의 이름을 바꿀 수 없습니다.

> **Note**
>
> Athena 엔진 버전 2에서는 ORC 테이블이 이름을 기준으로 읽도록 설정된 경우 Athena는 ORC 파일의 모든 열 이름을 소문자로 설정하도록 요구합니다. Apache Spark는 ORC 파일을

생성할 때 필드 이름을 소문자로 하지 않으므로 Athena가 이렇게 생성된 데이터를 읽지 못할 수 있습니다. 해결 방법은 열의 이름을 소문자로 바꾸거나 Athena 엔진 버전 3을 사용하는 것입니다.

다음 예제에서는 이름으로 읽도록 ORC를 변경하는 방법을 보여 줍니다.

```
CREATE EXTERNAL TABLE orders_orc_read_by_name (
   `o_comment` string,
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderpriority` string,
   `o_orderstatus` string,
   `o_clerk` string,
   `o_shippriority` int,
   `o_orderdate` string
)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.orc.OrcSerde'
WITH SERDEPROPERTIES (
  'orc.column.index.access'='false')
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_orc/';
```

Parquet: 이름으로 읽기

기본적으로 Parquet의 테이블은 이름으로 읽습니다. 이는 다음 구문으로 정의됩니다.

```
WITH SERDEPROPERTIES (
  'parquet.column.index.access'='false')
```

이름으로 읽기를 사용하면 테이블 중간에 열을 추가하고 열을 제거할 수 있습니다. 하지만 열의 이름을 바꿀 수는 없습니다.

열의 이름을 바꿀 수 있도록 Parquet에서 인덱스로 읽도록 지정하려면 `parquet.column.index.access` SerDe 속성을 `true`로 설정하여 테이블을 만들어야 합니다.

업데이트 유형

이 주제에서는 데이터를 실제로 변경하지 않고도 CREATE TABLE 문을 통해 스키마에서 가능한 몇 가지 변경 사항을 설명합니다. 각 스키마 업데이트 유형을 검토하고 Athena에서 이를 수행할 수 있는 데이터 형식을 지정합니다. 스키마를 업데이트하기 위해 ALTER TABLE 명령을 사용할 수도 있지만 기존 테이블을 실제로 수정하지 않는 경우도 있습니다. 대신 원래 CREATE TABLE 문에서 사용한 스키마를 수정하는 새 이름의 테이블을 생성합니다.

- 테이블의 시작 또는 중간에 열 추가
- 테이블 끝에 열 추가
- 열 제거
- 열 이름 바꾸기
- 열 재정렬
- 열의 데이터 형식 변경

예상되는 스키마의 변화에 따라 Athena 쿼리를 계속 사용하려면 호환되는 데이터 형식을 선택합니다.

CSV와 Parquet이라는 두 가지 형식으로 존재하는 orders 테이블에서 주문 정보를 읽는 애플리케이션을 살펴보겠습니다.

다음 예제에서는 Parquet 형식으로 테이블을 만듭니다.

```
CREATE EXTERNAL TABLE orders_parquet (
   `orderkey` int,
   `orderstatus` string,
   `totalprice` double,
   `orderdate` string,
   `orderpriority` string,
   `clerk` string,
   `shippriority` int
) STORED AS PARQUET
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_ parquet/';
```

다음 예제에서는 CSV 형식으로 동일한 테이블을 만듭니다.

```
CREATE EXTERNAL TABLE orders_csv (
   `orderkey` int,
   `orderstatus` string,
```

```
    `totalprice` double,
    `orderdate` string,
    `orderpriority` string,
    `clerk` string,
    `shippriority` int
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_csv/';
```

다음 단원에서는 이러한 테이블 업데이트가 Athena 쿼리에 주는 영향을 검토합니다.

테이블의 시작 또는 중간에 열 추가

열 추가는 가장 자주 발생하는 스키마 변경 중 하나입니다. 예를 들어 새 열을 추가하여 테이블을 새 데이터로 보강할 수 있습니다. 또는 기존 열의 소스가 변경된 경우 새 열을 추가하고, 이 열의 이전 버전을 유지하여 이러한 열을 사용하는 애플리케이션을 조정할 수 있습니다.

테이블의 시작 또는 중간에 열을 추가하고 기존 테이블에 대한 쿼리를 계속해서 실행하려면 Parquet 및 ORC(이름으로 읽도록 SerDe 속성을 설정한 경우), AVRO, JSON 형식을 사용합니다. 자세한 설명은 ORC 및 Parquet에서 인덱스 액세스을 참조하세요.

CSV 및 TSV 형식의 테이블 시작 또는 중간에 열을 추가하지 마세요. 이러한 형식에서는 순서가 중요합니다. 그런 경우 열을 추가하면 파티션의 스키마가 변경될 경우 스키마 불일치 오류가 발생합니다.

다음 예제에서는 JSON 데이터를 기반으로 테이블 중간에 `o_comment` 열을 추가하는 새 테이블을 생성합니다.

```
CREATE EXTERNAL TABLE orders_json_column_addition (
    `o_orderkey` int,
    `o_custkey` int,
    `o_orderstatus` string,
    `o_comment` string,
    `o_totalprice` double,
    `o_orderdate` string,
    `o_orderpriority` string,
    `o_clerk` string,
    `o_shippriority` int,
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_json/';
```

테이블 끝에 열 추가

Parquet, ORC, Avro, JSON, CSV 및 TSV와 같이 Athena가 지원하는 형식 중 하나로 테이블을 생성하는 경우, 기존 열 뒤에(즉, 파티션 열 앞에) 열을 추가하는 `ALTER TABLE ADD COLUMNS` 문을 사용할 수 있습니다.

다음 예제에서는 파티션 열 앞의 `orders_parquet` 테이블 끝에 `comment` 열을 추가합니다.

```
ALTER TABLE orders_parquet ADD COLUMNS (comment string)
```

> **Note**
>
> `ALTER TABLE ADD COLUMNS`를 실행한 후 Athena 쿼리 편집기에서 새 테이블 열을 보려면 편집기에서 테이블 목록을 수동으로 새로 고친 다음 테이블을 다시 확장합니다.

열 제거

열에 더 이상 데이터가 없는 경우 테이블에서 열을 제거하거나 열의 데이터에 대한 액세스를 제한해야 할 수 있습니다.

- JSON, Avro, Parquet 및 ORC 형식(이름으로 읽는 경우)의 테이블에서 열을 제거할 수 있습니다. 자세한 설명은 ORC 및 Parquet에서 인덱스 액세스을 참조하세요.
- Athena에서 이미 생성된 테이블을 유지하려면 CSV 및 TSV 형식의 테이블에서 열을 제거하지 않는 것이 좋습니다. 열을 제거하면 스키마가 차단되므로 제거된 열 없이 테이블을 다시 만들어야 합니다.

다음 예제에서는 Parquet 형식의 테이블에서 열 `` `totalprice` ``를 제거하고 쿼리를 실행합니다. Athena에서 Parquet은 기본적으로 이름으로 읽습니다. 따라서 이름으로 읽기를 지정하는 SERDEPROPERTIES 구성을 생략했습니다. 스키마를 변경했더라도 다음 쿼리에 성공합니다.

```
CREATE EXTERNAL TABLE orders_parquet_column_removed (
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderstatus` string,
   `o_orderdate` string,
   `o_orderpriority` string,
   `o_clerk` string,
   `o_shippriority` int,
   `o_comment` string
```

```
)
STORED AS PARQUET
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_parquet/';
```

열 이름 바꾸기

철자를 수정하거나, 열 이름에 설명을 포함시키거나, 열이 재정렬되지 않도록 기존 열을 다시 사용하기 위해 테이블의 열 이름을 바꿔야 할 수 있습니다.

데이터를 CSV 및 TSV 형식 또는 Parquet 및 ORC 형식(인덱스로 읽도록 구성된 경우)으로 저장한 경우 열 이름을 변경할 수 있습니다. 자세한 설명은 ORC 및 Parquet에서 인덱스 액세스을 참조하세요.

Athena는 스키마의 열 순서대로 CSV 및 TSV 형식의 데이터를 읽고 이를 동일한 순서로 반환합니다. 데이터를 열에 매핑할 때 열 이름을 사용하지 않기 때문에, Athena 쿼리를 중단하지 않고 CSV 또는 TSV 형식의 열 이름을 바꿀 수 있습니다.

열 이름을 바꾸는 한 가지 전략은 동일한 기본 데이터를 기반으로 새 테이블을 만들되 새 열 이름을 사용하는 것입니다. 다음 예제에서는 `orders_parquet_column_renamed`라는 새 `orders_parquet` 테이블을 만듭니다. 이 예제는 열 이름 `` `o_totalprice` ``를 `` `o_total_price` ``로 변경한 다음 Athena에서 다음 쿼리를 실행합니다.

```
CREATE EXTERNAL TABLE orders_parquet_column_renamed (
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderstatus` string,
   `o_total_price` double,
   `o_orderdate` string,
   `o_orderpriority` string,
   `o_clerk` string,
   `o_shippriority` int,
   `o_comment` string
)
STORED AS PARQUET
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_parquet/';
```

Parquet 테이블의 경우 다음 쿼리가 실행되지만 인덱스 대신 이름으로 열에 액세스(Parquet의 기본값)하므로 이름을 바꾼 열에 데이터가 표시되지 않습니다.

```
SELECT *
FROM orders_parquet_column_renamed;
```

CSV 형식의 테이블이 있는 쿼리는 비슷하게 보입니다:

```
CREATE EXTERNAL TABLE orders_csv_column_renamed (
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderstatus` string,
   `o_total_price` double,
   `o_orderdate` string,
   `o_orderpriority` string,
   `o_clerk` string,
   `o_shippriority` int,
   `o_comment` string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_csv/';
```

CSV 테이블의 경우, 다음 쿼리가 실행되고 이름이 바뀐 열을 포함한 모든 열에 데이터가 표시됩니다:

```
SELECT *
FROM orders_csv_column_renamed;
```

열 재정렬

JSON 또는 Parquet과 같이 기본적으로 이름으로 읽는 형식의 데이터가 있는 테이블의 경우에만 열을 재정렬할 수 있습니다. 필요한 경우 ORC를 이름으로 읽도록 지정할 수도 있습니다. 자세한 설명은 ORC 및 Parquet에서 인덱스 액세스을 참조하세요.

다음 예제에서는 열 순서가 다른 새 테이블을 생성합니다.

```
CREATE EXTERNAL TABLE orders_parquet_columns_reordered (
   `o_comment` string,
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderpriority` string,
   `o_orderstatus` string,
   `o_clerk` string,
   `o_shippriority` int,
   `o_orderdate` string
)
STORED AS PARQUET
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_parquet/';
```

열의 데이터 형식 변경

기존 유형에 필요한 양의 정보를 더 이상 보관할 수 없는 경우 다른 열 유형을 사용하는 것이 좋습니다. 예를 들어 ID 열의 값은 `INT` 데이터 형식의 크기를 초과할 수 있으므로 `BIGINT` 데이터 형식을 사용해야 합니다.

열에 대해 다른 데이터 형식을 사용하려는 경우 다음 사항을 고려해야 합니다.

- 대부분의 경우 열의 데이터 형식을 직접 변경할 수는 없습니다. 대신 Athena 테이블을 다시 생성하고 새 데이터 형식으로 열을 정의합니다.
- 특정 데이터 형식만 다른 데이터 형식으로 읽을 수 있습니다. 처리할 수 있는 데이터 형식은 본 섹션의 테이블을 참조하세요.
- Parquet 및 ORC의 데이터인 경우 테이블이 파티셔닝되지 않으면 열에 대해 다른 데이터 형식을 사용할 수 없습니다.
- Parquet 및 ORC 형식의 분할된 테이블의 경우 파티션의 열 형식은 다른 파티션의 열 형식과 다를 수 있으며, Athena는 가능하다면 원하는 형식으로 `CAST`합니다. 자세한 설명은 파티션이 있는 테이블의 스키마 불일치 오류 방지을 참조하세요.
- LazySimpleSerDe를 사용하여 생성한 테이블의 경우에만 `ALTER TABLE REPLACE COLUMNS` 문을 사용하여 기존 열을 다른 데이터 형식으로 바꿀 수 있지만 유지하려는 기존 열도 모두 해당 명령문에서 다시 정의해야 합니다. 그렇지 않으면 해당 열이 삭제됩니다. 자세한 내용은 ALTER TABLE REPLACE COLUMNS 단원을 참조하십시오.
- Apache Iceberg 테이블의 경우에만 ALTER TABLE CHANGE COLUMN 문을 사용하여 열의 데이터 형식을 변경할 수 있습니다. `ALTER TABLE REPLACE COLUMNS`는 Iceberg 테이블에서 지원되지 않습니다. 자세한 내용은 변화하는 Iceberg 테이블 스키마 단원을 참조하십시오.

> **Important**
>
> 데이터 형식 변환을 수행하기 전에 쿼리를 테스트 및 확인하는 것이 좋습니다. Athena에서 대상 데이터 형식을 사용할 수 없는 경우 `CREATE TABLE` 쿼리에 실패할 수 있습니다.

다음 테이블에는 다른 데이터 형식으로 처리되는 데이터 형식이 나열되어 있습니다.

호환되는 데이터 형식

원래 데이터 형식	사용 가능한 대상 데이터 형식
STRING	BYTE, TINYINT, SMALLINT, INT, BIGINT
BYTE	TINYINT, SMALLINT, INT, BIGINT
TINYINT	SMALLINT, INT, BIGINT
SMALLINT	INT, BIGINT
INT	BIGINT
FLOAT	DOUBLE

다음 예제에서는 원래 orders_json 테이블에 대해 CREATE TABLE 문을 사용하여 orders_json_bigint라는 새 테이블을 생성합니다. 새 테이블은 INT 대신 BIGINT를 `o_shippriority` 열의 데이터 형식으로 사용합니다.

```
CREATE EXTERNAL TABLE orders_json_bigint (
   `o_orderkey` int,
   `o_custkey` int,
   `o_orderstatus` string,
   `o_totalprice` double,
   `o_orderdate` string,
   `o_orderpriority` string,
   `o_clerk` string,
   `o_shippriority` BIGINT
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://DOC-EXAMPLE-BUCKET/orders_json';
```

데이터 형식이 변경되기 전에 원래 SELECT 쿼리와 비슷한 다음 쿼리가 실행됩니다:

```
Select * from orders_json
LIMIT 10;
```

파티션이 있는 테이블의 업데이트

Athena에서 테이블과 그 파티션은 동일한 데이터 형식을 사용해야 하지만 스키마는 다를 수 있습니다. 새 파티션을 만들면 파티션은 보통 테이블의 스키마를 상속합니다. 시간이 지남에 따라 스키마는 달라지기 시작할 수 있습니다. 이유는 다음과 같습니다.

- 테이블의 스키마가 변경되더라도 파티션의 스키마는 테이블의 스키마와 동기화 상태를 유지하도록 업데이트되지 않습니다.
- AWS Glue 크롤러를 사용하여 다른 스키마가 있는 파티션의 데이터를 검색할 수 있습니다. 즉 AWS Glue로 Athena에 테이블을 만든다면 크롤러가 처리를 마친 후 테이블과 그 파티션의 스키마가 다를 수 있습니다.
- AWS API를 사용하여 파티션을 직접 추가하는 경우

파티션이 있는 테이블이 다음 제약을 충족하면 Athena는 이를 처리합니다. 이러한 제약을 충족하지 않는다면 Athena는 HIVE_PARTITION_SCHEMA_MISMATCH 오류를 표시합니다.

- 각 파티션의 스키마가 테이블의 스키마와 호환됩니다.
- 테이블의 데이터 형식이 수행하려는 업데이트 유형(예: 열 추가, 삭제, 재정렬 또는 열의 데이터 형식 변경)을 허용합니다.

  예를 들어 CSV 및 TSV 형식의 경우 열 이름을 바꾸고, 테이블 끝에 새 열을 추가하고, 형식이 호환되는 경우 열의 데이터 형식을 변경할 수 있으나, 열을 제거할 수는 없습니다. 다른 형식의 경우 열을 추가 또는 제거하거나 형식이 호환되는 경우 열의 데이터 형식을 다른 형식으로 변경할 수 있습니다. 자세한 정보는 요약: Athena의 업데이트 및 데이터 형식을 참조하세요.

파티션이 있는 테이블의 스키마 불일치 오류 방지

쿼리 실행을 시작할 때 Athena는 각 열 데이터 형식이 테이블과 파티션 간에 호환되는지 확인하여 테이블의 스키마를 검증합니다.

- Parquet 및 ORC 데이터 스토리지 형식의 경우 Athena는 열 이름을 토대로 이를 열 이름 기반 스키마 검증에 사용합니다. 이렇게 하여 Parquet 및 ORC에 파티션이 있는 테이블의 `HIVE_PARTITION_SCHEMA_MISMATCH` 오류를 제거합니다. (SerDe 속성이 이름을 기준으로 인덱스에 액세스하도록 설정된 경우(`orc.column.index.access=FALSE`) 이 사항은 ORC에 해당됩니다. Parquet은 기본적으로 이름을 기준으로 인덱스를 읽습니다.)

- CSV, JSON 및 Avro의 경우 Athena는 인덱스 기반 스키마 검증을 사용합니다. 즉, 스키마 불일치 오류가 발생한다면 Athena가 실패하지 않고 쿼리할 수 있도록 스키마 불일치를 발생시키는 파티션을 삭제하고 다시 생성해야 합니다.

Athena는 테이블의 스키마를 파티션의 스키마와 비교합니다. AWS Glue 크롤러로 Athena의 CSV, JSON 및 AVRO에 테이블을 만들 경우 크롤러가 처리를 마친 후 테이블과 그 파티션의 스키마가 다를 수 있습니다. 테이블의 스키마와 파티션 스키마 간에 불일치가 있으면 다음과 유사한 스키마 검증 오류로 인해 Athena에서 쿼리가 실패합니다. 'crawler_test.click_avro'는 '문자열' 형식으로 선언되었는데, 'partition_0=2017-01-17' 파티션은 'col68'열을 'double' 형식으로 선언했습니다."

그러한 오류의 일반적인 해결 방법은 오류를 발생시키는 파티션을 삭제하고 다시 생성하는 것입니다. 자세한 내용은 ALTER TABLE DROP PARTITION 및 ALTER TABLE ADD PARTITION 단원을 참조하세요.

배열 쿼리

Amazon Athena를 사용하면 배열을 만들고, 연결하고, 다른 데이터 형식으로 변환한 다음 필터링 및 평면화하고 정렬할 수 있습니다.

주제

배열 생성

Athena에서 배열 리터럴을 구축하려면 ARRAY 키워드 뒤에 [ ] 괄호를 사용하고 쉼표로 구분된 배열 요소를 포함합니다.

예제

이 쿼리는 네 개 요소로 이루어진 배열 하나를 생성합니다.

```
SELECT ARRAY [1,2,3,4] AS items
```

다음 결과가 반환됩니다.

```
+-----------+
| items     |
+-----------+
| [1,2,3,4] |
+-----------+
```

이 쿼리는 두 개의 배열을 생성합니다.

```
SELECT ARRAY[ ARRAY[1,2], ARRAY[3,4] ] AS items
```

다음 결과가 반환됩니다.

```
+--------------------+
| items              |
+--------------------+
| [[1, 2], [3, 4]]   |
+--------------------+
```

호환되는 형식의 선택된 열에서 배열을 생성하려면 다음 예제와 같은 쿼리를 사용합니다.

```
WITH
dataset AS (
  SELECT 1 AS x, 2 AS y, 3 AS z
)
SELECT ARRAY [x,y,z] AS items FROM dataset
```

이 쿼리가 반환하는 값:

```
+-----------+
| items     |
+-----------+
| [1,2,3]   |
+-----------+
```

다음 예에서는 두 개의 배열이 선택되고 환영 메시지로 반환됩니다.

```
WITH
dataset AS (
  SELECT
    ARRAY ['hello', 'amazon', 'athena'] AS words,
    ARRAY ['hi', 'alexa'] AS alexa
)
SELECT ARRAY[words, alexa] AS welcome_msg
FROM dataset
```

이 쿼리가 반환하는 값:

```
+----------------------------------------+
| welcome_msg                            |
+----------------------------------------+
| [[hello, amazon, athena], [hi, alexa]] |
+----------------------------------------+
```

키-값 페어의 배열을 만들려면 이 예에서와 같이 키 배열 뒤에 값 배열을 가져오는 MAP 연산자를 사용하세요.

```
SELECT ARRAY[
   MAP(ARRAY['first', 'last', 'age'],ARRAY['Bob', 'Smith', '40']),
   MAP(ARRAY['first', 'last', 'age'],ARRAY['Jane', 'Doe', '30']),
   MAP(ARRAY['first', 'last', 'age'],ARRAY['Billy', 'Smith', '8'])
] AS people
```

이 쿼리가 반환하는 값:

```
+-------------------------------------------------------------------------------------------------------
+
| people
               |
```

```
+-----------------------------------------------------------------------------------------------
+
| [{last=Smith, first=Bob, age=40}, {last=Doe, first=Jane, age=30}, {last=Smith,
 first=Billy, age=8}] |
+-----------------------------------------------------------------------------------------------
+
```

문자열과 배열 연결

문자열 연결

두 문자열을 연결하려면 다음 예시처럼 이중 파이프 `||` 연산자를 사용합니다.

```
SELECT 'This' || ' is' || ' a' || ' test.' AS Concatenated_String
```

이 쿼리가 반환하는 값:

#	Concatenated_String
1	This is a test.

`concat()` 함수를 사용하여 동일한 결과를 얻을 수 있습니다.

```
SELECT concat('This', ' is', ' a', ' test.') AS Concatenated_String
```

이 쿼리가 반환하는 값:

#	Concatenated_String
1	This is a test.

`concat_ws()` 함수를 사용하여 첫 번째 인수에 지정된 구분 기호에 문자열을 연결할 수 있습니다.

```
SELECT concat_ws(' ', 'This', 'is', 'a', 'test.') as Concatenated_String
```

이 쿼리가 반환하는 값:

#	Concatenated_String
1	This is a test.

점을 사용하여 문자열 데이터 유형의 두 열을 연결하려면 큰따옴표를 사용하여 두 열을 참조하고 작은따옴표를 사용하여 점을 하드코딩된 문자열로 묶습니다. 열이 문자열 데이터 유형이 아닌 경우 CAST("*column_name*" as VARCHAR)를 사용하여 먼저 열을 캐스팅할 수 있습니다.

```
SELECT "col1" || '.' || "col2" as Concatenated_String
FROM my_table
```

이 쿼리가 반환하는 값:

#	Concatenated_String
1	*col1_string_value* .*col2_string_value*

배열 연결

동일한 기법을 사용하여 배열을 연결할 수 있습니다.

여러 배열을 연결하려면 이중 파이프 || 연산자를 사용합니다.

```
SELECT ARRAY [4,5] || ARRAY[ ARRAY[1,2], ARRAY[3,4] ] AS items
```

이 쿼리가 반환하는 값:

#	항목
1	[[4, 5], [1, 2], [3, 4]]

여러 배열을 하나의 배열으로 결합하려면 이중 파이프 연산자나 concat() 함수를 사용합니다.

```
WITH
dataset AS (
  SELECT
```

```
    ARRAY ['Hello', 'Amazon', 'Athena'] AS words,
    ARRAY ['Hi', 'Alexa'] AS alexa
)
SELECT concat(words, alexa) AS welcome_msg
FROM dataset
```

이 쿼리가 반환하는 값:

#	welcome_msg
1	[Hello, Amazon, Athena, Hi, Alexa]

concat() 기타 문자열 함수에 대한 자세한 내용은 Trino 설명서의 문자열 함수 및 연산자를 참조하세요.

배열 데이터 형식 변환

배열의 데이터를 지원되는 데이터 형식으로 변환하려면 CAST 연산자를 사용합니다(예: CAST(value AS type)). Athena는 기본 Presto 데이터 형식을 모두 지원합니다.

```
SELECT
   ARRAY [CAST(4 AS VARCHAR), CAST(5 AS VARCHAR)]
AS items
```

이 쿼리가 반환하는 값:

```
+-------+
| items |
+-------+
| [4,5] |
+-------+
```

다음 예와 같이 키-값 페어 요소로 두 개의 배열을 만들고 JSON으로 변환하여 연결합니다.

```
SELECT
   ARRAY[CAST(MAP(ARRAY['a1', 'a2', 'a3'], ARRAY[1, 2, 3]) AS JSON)] ||
   ARRAY[CAST(MAP(ARRAY['b1', 'b2', 'b3'], ARRAY[4, 5, 6]) AS JSON)]
AS items
```

이 쿼리가 반환하는 값:

```
+-------------------------------------------------+
| items                                           |
+-------------------------------------------------+
| [{"a1":1,"a2":2,"a3":3}, {"b1":4,"b2":5,"b3":6}] |
+-------------------------------------------------+
```

길이 구하기

`cardinality` 함수는 다음 예에서와 같이 배열의 길이를 반환합니다.

```
SELECT cardinality(ARRAY[1,2,3,4]) AS item_count
```

이 쿼리가 반환하는 값:

```
+------------+
| item_count |
+------------+
| 4          |
+------------+
```

배열 요소에 액세스

배열 요소에 액세스하려면 다음 예에서와 같이 `[]` 연산자(1은 첫 번째 요소를, 2는 두 번째 요소를 지정하는 방식)를 사용합니다.

```
WITH dataset AS (
SELECT
   ARRAY[CAST(MAP(ARRAY['a1', 'a2', 'a3'], ARRAY[1, 2, 3]) AS JSON)] ||
   ARRAY[CAST(MAP(ARRAY['b1', 'b2', 'b3'], ARRAY[4, 5, 6]) AS JSON)]
AS items )
SELECT items[1] AS item FROM dataset
```

이 쿼리가 반환하는 값:

```
+------------------------+
| item                   |
+------------------------+
| {"a1":1,"a2":2,"a3":3} |
+------------------------+
```

주어진 위치(인덱스 위치라고 함)에서 배열 요소에 액세스하려면 element_at() 함수를 사용하여 배열 이름과 인덱스 위치를 지정합니다.

- 인덱스가 0보다 큰 경우, element_at()은 지정된 요소를 배열의 처음부터 끝까지 계산하여 반환합니다. 이는 [] 연산자처럼 작동합니다.
- 인덱스가 0보다 작은 경우, element_at()은 배열의 끝에서부터 시작까지 계산하여 요소를 반환합니다.

다음 쿼리는 배열 words를 만들고, 거기에서 first_word로 첫 번째 요소 hello를 선택하고, middle_word로 두 번째 요소 amazon을(배열의 끝에서부터 계산), last_word로 세 번째 요소 athena를 선택합니다.

```
WITH dataset AS (
  SELECT ARRAY ['hello', 'amazon', 'athena'] AS words
)
SELECT
  element_at(words, 1) AS first_word,
  element_at(words, -2) AS middle_word,
  element_at(words, cardinality(words)) AS last_word
FROM dataset
```

이 쿼리가 반환하는 값:

```
+----------------------------------------+
| first_word  | middle_word | last_word  |
+----------------------------------------+
| hello       | amazon      | athena     |
+----------------------------------------+
```

중첩 배열 평면화

중첩 배열로 작업할 때 중첩 배열 요소를 단일 배열로 확장하거나 배열을 여러 행으로 확장해야 하는 경우가 많습니다.

예제

중첩된 배열의 요소를 여러 값의 단일 배열로 평면화하려면 flatten 함수를 사용합니다. 이 쿼리는 배열의 각 요소에 대한 행을 반환합니다.

```
SELECT flatten(ARRAY[ ARRAY[1,2], ARRAY[3,4] ]) AS items
```

이 쿼리가 반환하는 값:

```
+-----------+
| items     |
+-----------+
| [1,2,3,4] |
+-----------+
```

배열을 여러 행으로 평면화하려면 다음 예제에서와 같이 UNNEST 연산자와 함께 CROSS JOIN을 사용합니다.

```
WITH dataset AS (
  SELECT
    'engineering' as department,
    ARRAY['Sharon', 'John', 'Bob', 'Sally'] as users
)
SELECT department, names FROM dataset
CROSS JOIN UNNEST(users) as t(names)
```

이 쿼리가 반환하는 값:

```
+----------------------+
| department  | names  |
+----------------------+
| engineering | Sharon |
+----------------------|
| engineering | John   |
+----------------------|
| engineering | Bob    |
+----------------------|
| engineering | Sally  |
+----------------------+
```

키-값 페어의 배열을 평면화하려면 다음 예제에서와 같이 선택한 키를 열로 이동합니다.

```
WITH
dataset AS (
  SELECT
    'engineering' as department,
     ARRAY[
      MAP(ARRAY['first', 'last', 'age'],ARRAY['Bob', 'Smith', '40']),
```

```
      MAP(ARRAY['first', 'last', 'age'],ARRAY['Jane', 'Doe', '30']),
      MAP(ARRAY['first', 'last', 'age'],ARRAY['Billy', 'Smith', '8'])
     ] AS people
  )
SELECT names['first'] AS
 first_name,
 names['last'] AS last_name,
 department FROM dataset
CROSS JOIN UNNEST(people) AS t(names)
```

이 쿼리가 반환하는 값:

```
+---------------------------------------+
| first_name | last_name | department  |
+---------------------------------------+
| Bob        | Smith     | engineering |
| Jane       | Doe       | engineering |
| Billy      | Smith     | engineering |
+---------------------------------------+
```

직원 목록에서 합산 점수가 가장 높은 직원을 선택합니다. UNNEST은(는) 기본 조인 연산자라서 묵시적이므로 앞에 CROSS JOIN을(를) 붙이지 않고 FROM 절에 사용할 수 있습니다.

```
WITH
dataset AS (
  SELECT ARRAY[
    CAST(ROW('Sally', 'engineering', ARRAY[1,2,3,4]) AS ROW(name VARCHAR, department
 VARCHAR, scores ARRAY(INTEGER))),
    CAST(ROW('John', 'finance', ARRAY[7,8,9]) AS ROW(name VARCHAR, department VARCHAR,
 scores ARRAY(INTEGER))),
    CAST(ROW('Amy', 'devops', ARRAY[12,13,14,15]) AS ROW(name VARCHAR, department
 VARCHAR, scores ARRAY(INTEGER)))
  ] AS users
),
users AS (
 SELECT person, score
 FROM
   dataset,
   UNNEST(dataset.users) AS t(person),
   UNNEST(person.scores) AS t(score)
)
SELECT person.name, person.department, SUM(score) AS total_score FROM users
GROUP BY (person.name, person.department)
```

```
ORDER BY (total_score) DESC
LIMIT 1
```

이 쿼리가 반환하는 값:

```
+---------------------------------+
| name | department | total_score |
+---------------------------------+
| Amy  | devops     | 54          |
+---------------------------------+
```

직원 목록에서 개별 점수가 가장 높은 직원을 선택합니다.

```
WITH
dataset AS (
 SELECT ARRAY[
   CAST(ROW('Sally', 'engineering', ARRAY[1,2,3,4]) AS ROW(name VARCHAR, department
 VARCHAR, scores ARRAY(INTEGER))),
   CAST(ROW('John', 'finance', ARRAY[7,8,9]) AS ROW(name VARCHAR, department VARCHAR,
 scores ARRAY(INTEGER))),
   CAST(ROW('Amy', 'devops', ARRAY[12,13,14,15]) AS ROW(name VARCHAR, department
 VARCHAR, scores ARRAY(INTEGER)))
 ] AS users
),
users AS (
 SELECT person, score
 FROM
   dataset,
   UNNEST(dataset.users) AS t(person),
   UNNEST(person.scores) AS t(score)
)
SELECT person.name, score FROM users
ORDER BY (score) DESC
LIMIT 1
```

이 쿼리가 반환하는 값:

```
+--------------+
| name | score |
+--------------+
| Amy  | 15    |
+--------------+
```

고려 사항 및 제한

UNNEST가 쿼리에 있는 하나 이상의 배열에 사용되고 배열 중 하나가 NULL인 경우 쿼리는 행을 반환하지 않습니다. 빈 문자열인 배열에 UNNEST를 사용하면 빈 문자열이 반환됩니다.

예를 들어 다음 쿼리에서는 두 번째 배열이 null이므로 쿼리는 행을 반환하지 않습니다.

```
SELECT
    col1,
    col2
FROM UNNEST (ARRAY ['apples','oranges','lemons']) AS t(col1)
CROSS JOIN UNNEST (ARRAY []) AS t(col2)
```

다음 예제에서는 두 번째 배열이 빈 문자열을 포함하도록 수정됩니다. 각 행에 대해 쿼리는 col1의 값을 반환하고 col2의 값에 대해 빈 문자열을 반환합니다. 첫 번째 배열의 값이 반환되려면 두 번째 배열의 빈 문자열이 필요합니다.

```
SELECT
    col1,
    col2
FROM UNNEST (ARRAY ['apples','oranges','lemons']) AS t(col1)
CROSS JOIN UNNEST (ARRAY ['']) AS t(col2)
```

하위 쿼리에서 배열 만들기

행 모음에서 배열을 만듭니다.

```
WITH
dataset AS (
  SELECT ARRAY[1,2,3,4,5] AS items
)
SELECT array_agg(i) AS array_items
FROM dataset
CROSS JOIN UNNEST(items) AS t(i)
```

이 쿼리가 반환하는 값:

```
+-----------------+
| array_items     |
+-----------------+
| [1, 2, 3, 4, 5] |
```

```
+-----------------+
```

행 집합에서 고유한 값의 배열을 만들려면 `distinct` 키워드를 사용합니다.

```
WITH
dataset AS (
  SELECT ARRAY [1,2,2,3,3,4,5] AS items
)
SELECT array_agg(distinct i) AS array_items
FROM dataset
CROSS JOIN UNNEST(items) AS t(i)
```

이 쿼리는 다음 결과를 반환합니다. 순서는 보장되지 않습니다.

```
+-----------------+
| array_items     |
+-----------------+
| [1, 2, 3, 4, 5] |
+-----------------+
```

`array_agg` 함수 사용에 대한 자세한 내용을 알아보려면 Trino 설명서의 Aggregate functions(집계 함수)를 참조하세요.

배열 필터링

필터 조건과 일치하는 행 모음에서 배열을 만듭니다.

```
WITH
dataset AS (
  SELECT ARRAY[1,2,3,4,5] AS items
)
SELECT array_agg(i) AS array_items
FROM dataset
CROSS JOIN UNNEST(items) AS t(i)
WHERE i > 3
```

이 쿼리가 반환하는 값:

```
+-------------+
| array_items |
+-------------+
```

```
| [4, 5]      |
+-------------+
```

다음 예와 같이 요소 중 하나에 특정 값(예: 2)이 포함되어 있는지 여부에 따라 배열을 필터링합니다.

```
WITH
dataset AS (
  SELECT ARRAY
  [
    ARRAY[1,2,3,4],
    ARRAY[5,6,7,8],
    ARRAY[9,0]
  ] AS items
)
SELECT i AS array_items FROM dataset
CROSS JOIN UNNEST(items) AS t(i)
WHERE contains(i, 2)
```

이 쿼리가 반환하는 값:

```
+--------------+
| array_items  |
+--------------+
| [1, 2, 3, 4] |
+--------------+
```

`filter` 함수

```
 filter(ARRAY [list_of_values], boolean_function)
```

ARRAY 표현식의 `filter` 함수를 사용하여 *`boolean_function`*이 true인 *`list_of_values`*에 항목의 하위 집합인 새 배열을 만듭니다. `filter` 함수는 *`UNNEST`* 함수를 사용할 수 없는 경우에 유용할 수 있습니다.

다음 예제에서는 `[1,0,5,-1]` 배열에서 0보다 큰 값으로 필터링합니다.

```
SELECT filter(ARRAY [1,0,5,-1], x -> x>0)
```

결과

```
[1,5]
```

다음 예제에서는 [-1, NULL, 10, NULL] 배열에서 null이 아닌 값으로 필터링합니다.

```
SELECT filter(ARRAY [-1, NULL, 10, NULL], q -> q IS NOT NULL)
```

결과

```
[-1,10]
```

배열 정렬

일련의 행에서 고유한 값의 정렬된 배열을 만들려면 다음 예제와 같이 array_sort 함수를 사용합니다.

```
WITH
dataset AS (
  SELECT ARRAY[3,1,2,5,2,3,6,3,4,5] AS items
)
SELECT array_sort(array_agg(distinct i)) AS array_items
FROM dataset
CROSS JOIN UNNEST(items) AS t(i)
```

이 쿼리가 반환하는 값:

```
+--------------------+
| array_items        |
+--------------------+
| [1, 2, 3, 4, 5, 6] |
+--------------------+
```

배열을 여러 행으로 확장하는 방법에 관한 자세한 내용은중첩 배열 평면화 단원을 참조하세요.

배열과 집계 함수 사용

- 배열 내에서 값을 추가하려면 다음 예에서와 같이 SUM을 사용합니다.
- 배열 내에서 여러 행을 집계하려면 array_agg를 사용합니다. 자세한 설명은 하위 쿼리에서 배열 만들기을 참조하세요.

> **Note**
>
> Athena 엔진 버전 2부터 집계 함수에 대해 ORDER BY가 지원됩니다.

```
WITH
dataset AS (
  SELECT ARRAY
  [
    ARRAY[1,2,3,4],
    ARRAY[5,6,7,8],
    ARRAY[9,0]
  ] AS items
),
item AS (
  SELECT i AS array_items
  FROM dataset, UNNEST(items) AS t(i)
)
SELECT array_items, sum(val) AS total
FROM item, UNNEST(array_items) AS t(val)
GROUP BY array_items;
```

마지막 SELECT 문에서 sum() 및 UNNEST를 사용하는 대신 다음 예제와 같이 reduce()를 사용하여 처리 시간과 데이터 전송을 줄일 수 있습니다.

```
WITH
dataset AS (
  SELECT ARRAY
  [
    ARRAY[1,2,3,4],
    ARRAY[5,6,7,8],
    ARRAY[9,0]
  ] AS items
),
item AS (
  SELECT i AS array_items
  FROM dataset, UNNEST(items) AS t(i)
)
SELECT array_items, reduce(array_items, 0 , (s, x) -> s + x, s -> s) AS total
FROM item;
```

위 쿼리는 다음과 같은 결과를 반환합니다. 반환되는 결과의 순서는 보장되지 않습니다.

```
+----------------------+
| array_items  | total |
+----------------------+
| [1, 2, 3, 4] | 10    |
```

```
| [5, 6, 7, 8] | 26    |
| [9, 0]       | 9     |
+----------------------+
```

배열을 문자열로 변환

하나의 배열을 단일 문자열로 변환하려면 `array_join` 함수를 사용합니다. 다음의 독립형 예제에서는 별칭이 지정된 `words`라는 배열을 포함한 `dataset`라는 테이블을 생성합니다. 이 쿼리는 `array_join`을 사용해 `words`의 배열 요소를 조인하고, 공백으로 이들을 분리한 다음, 별칭이 지정된 `welcome_msg`라는 열에 최종 문자열을 반환합니다.

```
WITH
dataset AS (
  SELECT ARRAY ['hello', 'amazon', 'athena'] AS words
)
SELECT array_join(words, ' ') AS welcome_msg
FROM dataset
```

이 쿼리가 반환하는 값:

```
+---------------------+
| welcome_msg         |
+---------------------+
| hello amazon athena |
+---------------------+
```

배열을 사용하여 맵 만들기

맵(Maps)은 Athena에서 사용할 수 있는 데이터 형식으로 구성된 키 값 페어입니다. 맵을 만들려면 `MAP` 연산자를 사용하고, 그리로 두 개의 배열을 전달합니다. 첫 번째는 열(키) 이름이고 두 번째는 값입니다. 배열의 모든 값은 같은 형식이어야 합니다. 맵 값 배열 요소 중 일부가 다른 형식이어야 하는 경우, 나중에 변환하면 됩니다.

예제

이 예제는 데이터 세트에서 사용자를 선택합니다. `MAP` 연산자를 사용하고 그리로 두 개의 배열을 전달합니다. 첫 번째 배열에는 "first", "last" 및 "age"와 같은 열 이름 값이 포함됩니다. 두 번째 배열은 "Bob", "Smith", "35"와 같은 각 열의 값으로 구성됩니다.

```
WITH dataset AS (
```

```
  SELECT MAP(
    ARRAY['first', 'last', 'age'],
    ARRAY['Bob', 'Smith', '35']
  ) AS user
)
SELECT user FROM dataset
```

이 쿼리가 반환하는 값:

```
+---------------------------------+
| user                            |
+---------------------------------+
| {last=Smith, first=Bob, age=35} |
+---------------------------------+
```

이 예제에서와 같이 뒤에 [key_name]이 있는 필드 이름을 선택하여 Map 값을 검색할 수 있습니다.

```
WITH dataset AS (
 SELECT MAP(
   ARRAY['first', 'last', 'age'],
   ARRAY['Bob', 'Smith', '35']
 ) AS user
)
SELECT user['first'] AS first_name FROM dataset
```

이 쿼리가 반환하는 값:

```
+------------+
| first_name |
+------------+
| Bob        |
+------------+
```

복잡한 데이터 형식 및 중첩 구조의 배열 쿼리

소스 데이터는 흔히 복잡한 데이터 형식 및 중첩 구조의 배열로 구성됩니다. 이 단원의 예는 Athena 쿼리를 사용하여 요소의 데이터 형식을 변경하고, 배열 내의 요소를 찾고, 키워드를 찾는 방법을 보여줍니다.

- ROW 생성
- CAST를 이용해 배열의 필드 이름 변경

- . 표기법을 사용하여 배열 필터링
- 중첩된 값으로 배열 필터링
- UNNEST를 사용하여 배열 필터링
- regexp_like를 사용하여 배열에서 키워드 찾기

ROW 생성

> **Note**
>
> 이 섹션의 예제는 작업할 샘플 데이터를 만드는 수단으로 ROW를 사용합니다. Athena 내에서 테이블을 쿼리한다면 데이터 소스로부터 이미 ROW 데이터 형식을 작성했으므로 따로 작성할 필요가 없습니다. CREATE_TABLE을 사용하면 Athena가 내부에 STRUCT를 정의하고 표를 데이터로 채우고 데이터 세트의 각 행에 ROW 데이터 형식을 만듭니다. 기본 ROW 데이터 형식은 지원되는 모든 SQL 데이터 형식의 명명된 필드로 구성됩니다.

```
WITH dataset AS (
 SELECT
   ROW('Bob', 38) AS users
 )
SELECT * FROM dataset
```

이 쿼리가 반환하는 값:

```
+--------------------------+
| users                    |
+--------------------------+
| {field0=Bob, field1=38} |
+--------------------------+
```

CAST를 이용해 배열의 필드 이름 변경

ROW 값을 포함하는 배열에서 필드 이름을 변경하기 위해 ROW 선언을 CAST할 수 있습니다.

```
WITH dataset AS (
  SELECT
    CAST(
      ROW('Bob', 38) AS ROW(name VARCHAR, age INTEGER)
```

```
    ) AS users
)
SELECT * FROM dataset
```

이 쿼리가 반환하는 값:

```
+--------------------+
| users              |
+--------------------+
| {NAME=Bob, AGE=38} |
+--------------------+
```

> **Note**
>
> 위에서는 name을 Presto에서의 형식인 VARCHAR로 선언합니다. CREATE TABLE 문 내에 이 STRUCT를 선언하는 경우, Hive가 데이터 형식을 String으로 정의하므로 String 형식을 사용합니다.

. 표기법을 사용하여 배열 필터링

다음 예에서는 점 . 표기법을 사용하여 AWS CloudTrail 로그 테이블의 userIdentity 열에서 accountId 필드를 선택합니다. 자세한 내용은 AWS CloudTrail 로그 쿼리를 참조하세요.

```
SELECT
  CAST(useridentity.accountid AS bigint) as newid
FROM cloudtrail_logs
LIMIT 2;
```

이 쿼리가 반환하는 값:

```
+--------------+
| newid        |
+--------------+
| 112233445566 |
+--------------+
| 998877665544 |
+--------------+
```

값의 배열을 쿼리하려면 다음 쿼리를 실행합니다.

```
WITH dataset AS (
  SELECT ARRAY[
    CAST(ROW('Bob', 38) AS ROW(name VARCHAR, age INTEGER)),
    CAST(ROW('Alice', 35) AS ROW(name VARCHAR, age INTEGER)),
    CAST(ROW('Jane', 27) AS ROW(name VARCHAR, age INTEGER))
  ] AS users
)
SELECT * FROM dataset
```

다음 결과를 반환합니다.

```
+-----------------------------------------------------------------+
| users                                                           |
+-----------------------------------------------------------------+
| [{NAME=Bob, AGE=38}, {NAME=Alice, AGE=35}, {NAME=Jane, AGE=27}] |
+-----------------------------------------------------------------+
```

중첩된 값으로 배열 필터링

큰 배열에는 흔히 중첩된 구조가 포함되며, 그 안에서 값을 필터링하거나 검색할 수 있어야 합니다.

중첩된 BOOLEAN 값이 포함된 값의 배열에 데이터 집합을 정의하려면 다음 쿼리를 실행합니다.

```
WITH dataset AS (
  SELECT
    CAST(
      ROW('aws.amazon.com', ROW(true)) AS ROW(hostname VARCHAR, flaggedActivity
 ROW(isNew BOOLEAN))
    ) AS sites
)
SELECT * FROM dataset
```

다음 결과를 반환합니다.

```
+-------------------------------------------------------------+
| sites                                                       |
+-------------------------------------------------------------+
| {HOSTNAME=aws.amazon.com, FLAGGEDACTIVITY={ISNEW=true}}  |
+-------------------------------------------------------------+
```

그런 다음, 그 요소의 BOOLEAN 값을 필터링하고 액세스하려면 계속해서 점 . 표기법을 사용합니다.

```
WITH dataset AS (
  SELECT
    CAST(
      ROW('aws.amazon.com', ROW(true)) AS ROW(hostname VARCHAR, flaggedActivity
 ROW(isNew BOOLEAN))
    ) AS sites
)
SELECT sites.hostname, sites.flaggedactivity.isnew
FROM dataset
```

이 쿼리는 중첩된 필드를 선택하고 다음 결과를 반환합니다.

```
+------------------------+
| hostname       | isnew |
+------------------------+
| aws.amazon.com | true  |
+------------------------+
```

UNNEST를 사용하여 배열 필터링

중첩된 구조를 포함하는 배열을 하위 요소 중 하나로 필터링하려면 UNNEST 연산자로 쿼리를 실행합니다. UNNEST에 대한 자세한 내용은 중첩 배열 평면화을 참조하세요.

예를 들어 이 쿼리는 데이터 세트에서 사이트의 호스트 이름을 찾아냅니다.

```
WITH dataset AS (
  SELECT ARRAY[
    CAST(
      ROW('aws.amazon.com', ROW(true)) AS ROW(hostname VARCHAR, flaggedActivity
 ROW(isNew BOOLEAN))
    ),
    CAST(
      ROW('news.cnn.com', ROW(false)) AS ROW(hostname VARCHAR, flaggedActivity
 ROW(isNew BOOLEAN))
    ),
    CAST(
      ROW('netflix.com', ROW(false)) AS ROW(hostname VARCHAR, flaggedActivity ROW(isNew
 BOOLEAN))
    )
  ] as items
)
SELECT sites.hostname, sites.flaggedActivity.isNew
```

```
FROM dataset, UNNEST(items) t(sites)
WHERE sites.flaggedActivity.isNew = true
```

다음 결과가 반환됩니다.

```
+------------------------+
| hostname       | isnew |
+------------------------+
| aws.amazon.com | true  |
+------------------------+
```

regexp_like를 사용하여 배열에서 키워드 찾기

다음의 예는 regexp_like 함수를 사용하여 배열 내부 요소 내에서 키워드에 대한 데이터 집합을 검색하는 방법을 보여줍니다. 이 함수는 평가할 정규 표현식 패턴 또는 파이프(|)로 구분된 용어 목록을 입력으로 사용하며 패턴을 평가하고 지정된 문자열이 해당 패턴을 포함하고 있는지 판단합니다.

정규 표현식 패턴은 문자열 내에 포함되어 있어야 하고, 문자열과 반드시 일치할 필요는 없습니다. 전체 문자열을 일치시키려면 패턴의 시작 부분을 ^로, 끝부분을 $로 묶으세요(예: '^pattern$').

호스트 이름과 flaggedActivity 요소가 포함된 사이트 배열을 고려하세요. 이 요소에는 여러 개의 MAP 요소가 포함된 ARRAY가 포함되며, 각기 다른 인기 키워드와 인기도가 나열됩니다. 이 배열에서 MAP 내의 특정 키워드를 찾고자 한다고 가정해 보겠습니다.

이 데이터 세트에서 특정 키워드를 갖는 사이트를 검색하려면 유사한 SQL LIKE 연산자 대신 regexp_like을(를) 사용하는데, 다수의 키워드를 검색할 때는 regexp_like이(가) 더욱 효율적이기 때문입니다.

Example 예제 1: **regexp_like** 사용하기

이 예제의 쿼리는 regexp_like 함수를 사용하여 배열 내부의 값에 있는 'politics|bigdata'(이)라는 용어를 검색합니다.

```
WITH dataset AS (
  SELECT ARRAY[
    CAST(
      ROW('aws.amazon.com', ROW(ARRAY[
          MAP(ARRAY['term', 'count'], ARRAY['bigdata', '10']),
          MAP(ARRAY['term', 'count'], ARRAY['serverless', '50']),
          MAP(ARRAY['term', 'count'], ARRAY['analytics', '82']),
          MAP(ARRAY['term', 'count'], ARRAY['iot', '74'])
      ])
```

```
      ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
   ),
   CAST(
     ROW('news.cnn.com', ROW(ARRAY[
       MAP(ARRAY['term', 'count'], ARRAY['politics', '241']),
       MAP(ARRAY['term', 'count'], ARRAY['technology', '211']),
       MAP(ARRAY['term', 'count'], ARRAY['serverless', '25']),
       MAP(ARRAY['term', 'count'], ARRAY['iot', '170'])
     ])
     ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
   ),
   CAST(
     ROW('netflix.com', ROW(ARRAY[
       MAP(ARRAY['term', 'count'], ARRAY['cartoons', '1020']),
       MAP(ARRAY['term', 'count'], ARRAY['house of cards', '112042']),
       MAP(ARRAY['term', 'count'], ARRAY['orange is the new black', '342']),
       MAP(ARRAY['term', 'count'], ARRAY['iot', '4'])
     ])
     ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
   )
 ] AS items
),
sites AS (
  SELECT sites.hostname, sites.flaggedactivity
  FROM dataset, UNNEST(items) t(sites)
)
SELECT hostname
FROM sites, UNNEST(sites.flaggedActivity.flags) t(flags)
WHERE regexp_like(flags['term'], 'politics|bigdata')
GROUP BY (hostname)
```

이 쿼리는 다음 두 사이트를 반환합니다.

```
+----------------+
| hostname       |
+----------------+
| aws.amazon.com |
+----------------+
| news.cnn.com   |
+----------------+
```

Example 예제 2: **regexp_like** 사용하기

다음 예의 쿼리는 검색어와 일치하는 사이트의 총 인기 점수를 regexp_like 함수로 합산한 다음, 가장 높은 순서에서 가장 낮은 순서로 정렬합니다.

```
WITH dataset AS (
  SELECT ARRAY[
    CAST(
      ROW('aws.amazon.com', ROW(ARRAY[
          MAP(ARRAY['term', 'count'], ARRAY['bigdata', '10']),
          MAP(ARRAY['term', 'count'], ARRAY['serverless', '50']),
          MAP(ARRAY['term', 'count'], ARRAY['analytics', '82']),
          MAP(ARRAY['term', 'count'], ARRAY['iot', '74'])
      ])
      ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
    ),
    CAST(
      ROW('news.cnn.com', ROW(ARRAY[
        MAP(ARRAY['term', 'count'], ARRAY['politics', '241']),
        MAP(ARRAY['term', 'count'], ARRAY['technology', '211']),
        MAP(ARRAY['term', 'count'], ARRAY['serverless', '25']),
        MAP(ARRAY['term', 'count'], ARRAY['iot', '170'])
      ])
      ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
    ),
    CAST(
      ROW('netflix.com', ROW(ARRAY[
        MAP(ARRAY['term', 'count'], ARRAY['cartoons', '1020']),
        MAP(ARRAY['term', 'count'], ARRAY['house of cards', '112042']),
        MAP(ARRAY['term', 'count'], ARRAY['orange is the new black', '342']),
        MAP(ARRAY['term', 'count'], ARRAY['iot', '4'])
      ])
      ) AS ROW(hostname VARCHAR, flaggedActivity ROW(flags ARRAY(MAP(VARCHAR,
 VARCHAR)) ))
    )
  ] AS items
),
sites AS (
  SELECT sites.hostname, sites.flaggedactivity
  FROM dataset, UNNEST(items) t(sites)
)
```

```
SELECT hostname, array_agg(flags['term']) AS terms, SUM(CAST(flags['count'] AS
 INTEGER)) AS total
FROM sites, UNNEST(sites.flaggedActivity.flags) t(flags)
WHERE regexp_like(flags['term'], 'politics|bigdata')
GROUP BY (hostname)
ORDER BY total DESC
```

이 쿼리는 다음 두 사이트를 반환합니다.

```
+-----------------------------------+
| hostname       | terms    | total  |
+----------------+------------------+
| news.cnn.com   | politics |  241   |
+----------------+------------------+
| aws.amazon.com | bigdata |  10    |
+----------------+------------------+
```

지리 공간 데이터 쿼리

지리 공간 데이터에는 객체의 지리적 위치를 지정하는 식별자가 포함되어 있습니다. 이러한 형식의 데이터의 예로는 일기 예보, 지도 안내, 지리적 위치, 매장 위치 및 항공사 노선이 있는 트윗 등이 있습니다. 지리 공간 데이터는 비즈니스 분석, 보고 및 예측에 중요한 역할을 합니다.

위도, 경도와 같은 지리 공간 식별자를 사용하면 모든 우편 주소를 지리적 좌표 집합으로 변환할 수 있습니다.

주제

지리 공간 쿼리란 무엇인가요?

지리 공간 쿼리는 Athena에 지원되는 특수한 SQL 쿼리 형식입니다. 다음과 같은 점에서 비 공간 SQL 쿼리와 다릅니다.

- `point`, `line`, `multiline`, `polygon`, `multipolygon` 등 특수한 geometry 데이터 형식을 사용합니다.

- `distance`, `equals`, `crosses`, `touches`, `overlaps`, `disjoint` 등 geometry 데이터 형식 사이의 관계를 표현합니다.

Athena에서 지리 공간 쿼리를 사용하여 다음과 같은 작업을 실행할 수 있습니다.

- 두 지점 사이의 거리를 파악합니다.
- 한 영역(폴리곤)에 다른 영역이 포함되는지 확인합니다.
- 한 줄이 다른 줄이나 폴리곤과 교차하거나 접촉하는지 확인합니다.

예를 들어 Athena에서 Mount Rainier의 지리적 좌표를 위해 `double` 형식의 값으로부터 `point` geometry 데이터 형식을 얻으려면 다음 예제처럼 `ST_Point (longitude, latitude)` 지리 공간 함수를 사용합니다.

```
ST_Point(-121.7602, 46.8527)
```

입력 데이터 형식 및 geometry 데이터 형식

Athena에서 지리 공간 함수를 사용하려면 데이터를 WKT 형식으로 입력하거나 Hive JSON SerDe를 사용합니다. Athena에 지원되는 geometry 데이터 형식을 사용할 수도 있습니다.

입력 데이터 형식

지리 공간 쿼리를 처리하기 위해 Athena는 다음 데이터 형식의 입력 데이터를 지원합니다.

- WKT(Well-known Text). Athena에서 WKT는 `varchar(x)` 또는 `string` 데이터 형식으로 표현됩니다.
- JSON 인코딩 지리 공간 데이터. 지리 공간 데이터로 JSON 파일을 구문 분석하고 테이블을 생성하기 위해 Athena는 Hive JSON SerDe를 사용합니다. Athena에서 이 SerDe 사용에 대한 자세한 내용은 JSON SerDe 라이브러리 단원을 참조하세요.

Geometry 데이터 형식

지리 공간 쿼리를 처리하기 위해 Athena는 다음과 같은 특수한 geometry 데이터 형식을 지원합니다.

- `point`
- `line`
- `polygon`

- `multiline`
- `multipolygon`

지원되는 지리 공간 함수

Athena에 사용할 수 있는 지리 공간 함수는 사용하는 엔진 버전에 따라 다릅니다.

- Athena 엔진 버전 3의 지리 공간 함수에 대한 내용은 Trino 설명서의 Geospatial functions(지리 공간 함수)를 참조하세요.
- Athena 엔진 버전 2의 현재 함수 이름 변경 사항 및 새 함수 목록은 Athena 엔진 버전 2의 지리 공간 함수 이름 변경 및 새 함수 섹션을 참조하세요.

Athena 엔진 버전 관리에 대한 자세한 내용은 Athena 엔진 버전 관리 단원을 참조하세요.

주제

- Athena 엔진 버전 3의 지리 공간 함수
- Athena 엔진 버전 2의 지리 공간 함수

Athena 엔진 버전 3의 지리 공간 함수

Athena 엔진 버전 3의 지리 공간 함수에 대한 내용은 Trino 설명서의 지리 공간 함수를 참조하세요.

Athena 엔진 버전 2의 지리 공간 함수

이 주제에는 Athena 엔진 버전 2부터 지원되는 ESRI 지리 공간 함수가 나열되어 있습니다. Athena 엔진 버전에 대한 자세한 내용은 Athena 엔진 버전 관리 단원을 참조하세요.

Athena 엔진 버전 2의 변경 사항

- 일부 함수의 입력 및 출력 형식이 변경되었습니다. 가장 주목할 점은 입력에서 `VARBINARY` 유형이 더 이상 직접 지원되지 않는다는 것입니다. 자세한 내용은 지리 공간 함수 변경 사항 단원을 참조하십시오.
- 일부 지리 공간 함수의 이름이 변경되었습니다. 자세한 내용은 Athena 엔진 버전 2의 지리 공간 함수 이름 변경 단원을 참조하십시오.
- 새 기능이 추가되었습니다. 자세한 내용은 Athena 엔진 버전 2의 새 지리 공간 함수 단원을 참조하십시오.

Athena는 다음과 같은 유형의 지리 공간 함수를 지원합니다.

- 생성자 함수
- 지리 공간 관계 함수
- 작업 함수
- 접근자 함수
- 집계 함수
- Bing 타일 함수

생성자 함수

`point`, `line` 또는 `polygon` geometry 데이터 형식의 이진 표기를 구하려면 생성자 함수를 사용합니다. 또한 이러한 기능을 사용하여 이진 데이터를 텍스트로 변환하고 WKT(Well-Known Text)로 표현되는 geometry 데이터에 대한 이진 값을 얻을 수 있습니다.

`ST_AsBinary(geometry)`

지정된 geometry의 WKB 표현을 포함한 varbinary 데이터 형식을 반환합니다. 예제

```
SELECT ST_AsBinary(ST_Point(-158.54, 61.56))
```

`ST_AsText(geometry)`

지정된 각 geometry 데이터 형식을 텍스트로 변환합니다. geometry 데이터 형식의 WKT 표현인 varchar 데이터 형식의 값을 반환합니다. 예제

```
SELECT ST_AsText(ST_Point(-158.54, 61.56))
```

`ST_GeomAsLegacyBinary(geometry)`

지정된 geometry에서 레거시 varbinary를 반환합니다. 예제

```
SELECT ST_GeomAsLegacyBinary(ST_Point(-158.54, 61.56)
```

`ST_GeometryFromText(varchar)`

WKT 형식의 텍스트를 geometry 데이터 형식으로 변환합니다. geometry 데이터 형식의 값을 반환합니다. 예제

```
SELECT ST_GeometryFromText(ST_AsText(ST_Point(1, 2)))
```

ST_GeomFromBinary(varbinary)

WKB 표현에서 geometry 유형 객체를 반환합니다. 예제

```
SELECT ST_GeomFromBinary(ST_AsBinary(ST_Point(-158.54, 61.56)))
```

ST_GeomFromLegacyBinary(varbinary)

레거시 varbinary 형식에서 geometry 형식 객체를 반환합니다. 예제

```
SELECT ST_GeomFromLegacyBinary(ST_GeomAsLegacyBinary(ST_Point(-158.54, 61.56)))
```

ST_LineFromText(varchar)

geometry 데이터 형식 `line`으로 값을 반환합니다. 예제

```
SELECT ST_Line('linestring(1 1, 2 2, 3 3)')
```

ST_LineString(array(point))

point geometry 형식 배열로부터 형성된 `LineString` geometry 형식을 반환합니다. 지정된 배열에 비어 있지 않은 점이 두 개 미만이면 빈 `LineString`이 반환됩니다. 배열의 요소가 null이거나, 비어 있거나, 이전 요소와 동일한 경우 예외가 발생합니다. 반환된 geometry는 단순하지 않을 수 있습니다. 지정된 입력에 따라 반환된 geometry가 자체 교차하거나 중복된 꼭지점을 포함할 수 있습니다. 예제

```
SELECT ST_LineString(ARRAY[ST_Point(-158.54, 61.56), ST_Point(-158.55, 61.56)])
```

ST_MultiPoint(array(point))

지정된 점으로부터 형성된 `MultiPoint` geometry 객체를 반환합니다. 지정된 배열이 비어있는 경우는 null을 반환합니다. 배열의 요소가 null이거나, 비어 있는 경우 예외가 발생합니다. 반환된 geometry는 단순하지 않을 수 있으며 지정된 배열에 중복된 점이 있는 경우 중복된 점을 포함할 수 있습니다. 예제

```
SELECT ST_MultiPoint(ARRAY[ST_Point(-158.54, 61.56), ST_Point(-158.55, 61.56)])
```

ST_Point(double, double)

geometry 형식의 point 객체를 반환합니다. 이 함수에 대한 입력 데이터 값으로, UTM(Universal Transverse Mercator) 카트리지 좌표계의 값이나 십진도의 지리 지도 단위(경도와 위도)와 같은 기하학적 값을 사용하십시오. 경도와 위도 값은 WGS 1984 또는 EPSG:4326으로도 알려진 세계지구좌표시스템을 사용합니다. WGS 1984는 GPS(Global Positioning System)에서 사용하는 좌표계입니다.

예를 들어 다음 표기법에서 지도 좌표는 경도와 위도로 지정되며, 버퍼 거리인 .072284 값은 각도 단위의 십진수 각도로 지정됩니다.

```
SELECT ST_Buffer(ST_Point(-74.006801, 40.705220), .072284)
```

구문:

```
SELECT ST_Point(longitude, latitude) FROM earthquakes LIMIT 1
```

다음 예제에서는 특정 경도 및 위도 좌표를 사용합니다.

```
SELECT ST_Point(-158.54, 61.56)
FROM earthquakes
LIMIT 1
```

다음 예제에서는 특정 경도 및 위도 좌표를 사용합니다.

```
SELECT ST_Point(-74.006801, 40.705220)
```

다음 예제에서는 ST_AsText 함수를 사용하여 WKT에서 geometry를 얻습니다.

```
SELECT ST_AsText(ST_Point(-74.006801, 40.705220)) AS WKT
```

ST_Polygon(varchar)

제공된 세로 좌표 시퀀스를 시계 방향(왼쪽에서 오른쪽)으로 사용하여 geometry 데이터 형식 polygon을 반환합니다. Athena 엔진 버전 2부터 다각형만 입력으로 수락됩니다. 예제

```
SELECT ST_Polygon('polygon ((1 1, 1 4, 4 4, 4 1))')
```

to_geometry(sphericalGeography)

지정된 구형 지리 객체로부터 geometry 객체를 반환합니다. 예제

```
SELECT to_geometry(to_spherical_geography(ST_Point(-158.54, 61.56)))
```

to_spherical_geography(geometry)

지정된 geometry로부터 구형 지리 객체를 반환합니다. 이 함수를 사용하여 geometry 객체를 지구 반경을 갖는 구체의 구형 지리 객체로 변환합니다. 이 함수는 2차원 공간에 정의된 POINT, MULTIPOINT, LINESTRING, MULTILINESTRING, POLYGON, MULTIPOLYGON geometry 또는 이러한 geometry의 GEOMETRYCOLLECTION에만 사용할 수 있습니다. 지정된 geometry의 각 점에 대해 함수는 point.x가 [-180.0, 180.0] 내에 있고 point.y가 [-90.0, 90.0] 내에 있는지 확인합니다. 이 함수는 이러한 점을 경도 및 위도로 사용하여 sphericalGeography 결과의 형상을 구성합니다.

예제

```
SELECT to_spherical_geography(ST_Point(-158.54, 61.56))
```

지리 공간 관계 함수

다음 함수는 사용자가 입력 항목으로 지정한 서로 다른 두 geometry 간의 관계를 표현하고, boolean 형식의 결과를 반환합니다. geometry 페어를 지정하는 순서가 중요합니다. 첫 번째 geometry 값을 왼쪽 geometry, 두 번째 geometry 값을 오른쪽 geometry라고 합니다.

이 함수는 다음을 반환합니다.

- TRUE: 함수에 의해 설명된 관계가 만족되는 경우에만.
- FALSE: 함수에 의해 설명된 관계가 만족되지 않는 경우에만.

ST_Contains(geometry, geometry)

왼쪽 geometry에 오른쪽 geometry가 포함되어 있는 경우에만 TRUE를 반환합니다. 예시:

```
SELECT ST_Contains('POLYGON((0 2,1 1,0 -1,0 2))', 'POLYGON((-1 3,2 1,0 -3,-1 3))')
```

```
SELECT ST_Contains('POLYGON((0 2,1 1,0 -1,0 2))', ST_Point(0, 0))
```

```
SELECT ST_Contains(ST_GeometryFromText('POLYGON((0 2,1 1,0 -1,0 2))'),
 ST_GeometryFromText('POLYGON((-1 3,2 1,0 -3,-1 3))'))
```

ST_Crosses(geometry, geometry)

왼쪽 geometry가 오른쪽 geometry와 교차하는 경우에만 TRUE를 반환합니다. 예제

```
SELECT ST_Crosses(ST_Line('linestring(1 1, 2 2 )'), ST_Line('linestring(0 1, 2 2)'))
```

ST_Disjoint(geometry, geometry)

왼쪽 geometry와 오른쪽 geometry의 교차가 빈 경우에만 TRUE를 반환합니다. 예제

```
SELECT ST_Disjoint(ST_Line('linestring(0 0, 0 1)'), ST_Line('linestring(1 1, 1 0)'))
```

ST_Equals(geometry, geometry)

왼쪽 geometry가 오른쪽 geometry와 일치하는 경우에만 TRUE를 반환합니다. 예제

```
SELECT ST_Equals(ST_Line('linestring( 0 0, 1 1)'), ST_Line('linestring(1 3, 2 2)'))
```

ST_Intersects(geometry, geometry)

왼쪽 geometry와 오른쪽 geometry와 교차하는 경우에만 TRUE를 반환합니다. 예제

```
SELECT ST_Intersects(ST_Line('linestring(8 7, 7 8)'), ST_Polygon('polygon((1 1, 4 1, 4
 4, 1 4))'))
```

ST_Overlaps(geometry, geometry)

왼쪽 geometry와 오른쪽 geometry가 중첩되는 경우에만 TRUE를 반환합니다. 예제

```
SELECT ST_Overlaps(ST_Polygon('polygon((2 0, 2 1, 3 1))'), ST_Polygon('polygon((1 1, 1
 4, 4 4, 4 1))'))
```

ST_Relate(geometry, geometry, varchar)

왼쪽 geometry가 오른쪽 geometry와 지정된 DE-9IM(Dimensionally Extended nine-Intersection Model) 관계에 있는 경우에만 TRUE를 반환합니다. 세 번째(varchar) 입력은 관계입니다. 예제

```
SELECT ST_Relate(ST_Line('linestring(0 0, 3 3)'), ST_Line('linestring(1 1, 4 4)'),
 'T********')
```

ST_Touches(geometry, geometry)

왼쪽 geometry가 오른쪽 geometry와 접하는 경우에만 TRUE를 반환합니다.

예제

```
SELECT ST_Touches(ST_Point(8, 8), ST_Polygon('polygon((1  1, 1  4, 4  4, 4 1))'))
```

ST_Within(geometry, geometry)

왼쪽 geometry가 오른쪽 geometry 내에 있는 경우에만 TRUE를 반환합니다.

예제

```
SELECT ST_Within(ST_Point(8, 8), ST_Polygon('polygon((1  1, 1  4, 4  4, 4 1))'))
```

작업 함수

작업 함수를 사용하여 geometry 데이터 형식 값에 대한 작업을 수행합니다. 예를 들어 단일 geometry 데이터 형식의 경계, 두 geometry 데이터 형식 간의 교차, 왼쪽과 오른쪽 geometry 간 차이를 알아낼 수 있습니다. 각각의 geometry 데이터 형식 또는 특정 geometry 데이터 형식 주위의 링이나 외장 버퍼는 동일합니다.

geometry_union(array(geometry))

지정된 geometry의 점 집합 공용 구조체를 나타내는 geometry를 반환합니다. 예제

```
SELECT geometry_union(ARRAY[ST_Point(-158.54, 61.56), ST_Point(-158.55, 61.56)])
```

ST_Boundary(geometry)

geometry 데이터 형식 중 하나를 입력으로 취해 boundary geometry 데이터 형식을 반환합니다.

예시:

```
SELECT ST_Boundary(ST_Line('linestring(0 1, 1 0)')))
```

```
SELECT ST_Boundary(ST_Polygon('polygon((1  1, 1  4, 4  4, 4 1))'))
```

ST_Buffer(geometry, double)

geometry 데이터 형식(점, 선, 다각형, 다중 선, 다중 다각형) 중 하나를 입력으로 가져오고 거리를 double 형식으로 가져옵니다). 지정한 거리(또는 반경)만큼 버퍼링된 geometry 데이터 형식을 반환합니다. 예제

```
SELECT ST_Buffer(ST_Point(1, 2), 2.0)
```

다음 예제에서 지도 좌표는 경도와 위도로 지정되며, 버퍼 거리인 .072284 값은 각도 단위의 십진수 각도로 지정됩니다.

```
SELECT ST_Buffer(ST_Point(-74.006801, 40.705220), .072284)
```

ST_Difference(geometry, geometry)

왼쪽 geometry와 오른쪽 geometry 간 차집합의 geometry를 반환합니다. 예제

```
SELECT ST_AsText(ST_Difference(ST_Polygon('polygon((0 0, 0 10, 10 10, 10 0))'),
 ST_Polygon('polygon((0 0, 0 5, 5 5, 5 0))')))
```

ST_Envelope(geometry)

입력으로 line, polygon, multiline, multipolygon geometry 데이터 형식을 사용합니다. point geometry 데이터 형식을 지원하지 않습니다. 포락선의 geometry를 반환합니다. 여기서 포락선은 지정된 geometry 데이터 형식을 둘러싼 직사각형입니다. 예시:

```
SELECT ST_Envelope(ST_Line('linestring(0 1, 1 0)'))
```

```
SELECT ST_Envelope(ST_Polygon('polygon((1  1, 1  4, 4  4, 4 1))'))
```

ST_EnvelopeAsPts(geometry)

geometry의 경계 사각형의 왼쪽 아래 및 오른쪽 위 모서리를 나타내는 두 점의 배열을 반환합니다. 지정된 geometry가 비어 있으면 null을 반환합니다. 예제

```
SELECT ST_EnvelopeAsPts(ST_Point(-158.54, 61.56))
```

ST_ExteriorRing(geometry)

polygon 입력 형식의 외부 링의 geometry를 반환합니다. Athena 엔진 버전 2부터 다각형이 입력으로 수락되는 유일한 지오메트리입니다. 예시:

```
SELECT ST_ExteriorRing(ST_Polygon(1,1, 1,4, 4,1))
```

```
SELECT ST_ExteriorRing(ST_Polygon('polygon ((0 0, 8 0, 0 8, 0 0), (1 1, 1 5, 5 1, 1
 1))'))
```

ST_Intersection(geometry, geometry)

왼쪽 geometry와 오른쪽 geometry의 교집합의 geometry를 반환합니다. 예시:

```
SELECT ST_Intersection(ST_Point(1,1), ST_Point(1,1))
```

```
SELECT ST_Intersection(ST_Line('linestring(0 1, 1 0)'), ST_Polygon('polygon((1  1, 1
 4, 4  4, 4 1))'))
```

```
SELECT ST_AsText(ST_Intersection(ST_Polygon('polygon((2 0, 2 3, 3 0))'),
 ST_Polygon('polygon((1 1, 4 1, 4 4, 1 4))')))
```

ST_SymDifference(geometry, geometry)

왼쪽 geometry와 오른쪽 geometry 간 기하학적 대칭차집합의 geometry를 반환합니다. 예제

```
SELECT ST_AsText(ST_SymDifference(ST_Line('linestring(0 2, 2 2)'),
 ST_Line('linestring(1 2, 3 2)')))
```

ST_Union(geometry, geometry)

지정된 geometry의 점 집합 합집합을 나타내는 geometry 데이터 형식을 반환합니다. 예제

```
SELECT ST_Union(ST_Point(-158.54, 61.56),ST_LineString(array[ST_Point(1,2),
 ST_Point(3,4)]))
```

접근자 함수

접근자 함수는 다양한 geometry 데이터 형식으로부터 varchar, bigint 또는 double 형식의 값을 얻는 데 유용합니다. 여기서 geometry는 Athena에 지원되는 geometry 데이터 형식인 point, line,

polygon, multiline, multipolygon 중 하나입니다. 예를 들어, polygon geometry 데이터 형식의 영역, 지정된 geometry 데이터 형식의 최대 및 최소 X 및 Y 값을 구하거나, line 길이를 얻거나, 지정된 geometry 데이터 형식으로 점수를 얻을 수 있습니다.

geometry_invalid_reason(geometry)

지정된 geometry가 유효하지 않거나 단순하지 않은 이유를 varchar 데이터 형식으로 반환합니다. 지정된 geometry가 유효하지 않거나 단순하지 않은 경우 유효하지 않은 이유를 반환합니다. 지정된 geometry가 유효하고 단순한 경우 null을 반환합니다. 예제

```
SELECT geometry_invalid_reason(ST_Point(-158.54, 61.56))
```

great_circle_distance(latitude1, longitude1, latitude2, longitude2)

킬로미터 단위의 지구 표면 두 점 사이의 대원 거리(great-circle distance)를 double로 반환합니다. 예제

```
SELECT great_circle_distance(36.12, -86.67, 33.94, -118.40)
```

line_locate_point(lineString, point)

지정된 라인 스트링의 가장 가까운 점에서 지정된 점까지의 위치를 전체 2d 라인 길이 대비 비율로 나타내는 0에서 1 사이의 double을 반환합니다.

지정된 라인 스트링 또는 점이 비어 있거나 null인 경우 null을 반환합니다. 예제

```
SELECT line_locate_point(ST_GeometryFromText('LINESTRING (0 0, 0 1)'), ST_Point(0,
 0.2))
```

simplify_geometry(geometry, double)

Ramer-douglas-peucker algorithm(Ramer-Douglas-Peucker 알고리즘)을 사용해 지정된 geometry의 단순화 버전인 geometry 데이터 형식을 반환합니다. 유효하지 않은 파생 geometry(특히 폴리곤)를 생성하지 않아야 합니다. 예제

```
SELECT simplify_geometry(ST_GeometryFromText('POLYGON ((1 0, 2 1, 3 1, 3 1, 4 1, 1
 0))'), 1.5)
```

ST_Area(geometry)

geometry 데이터 형식을 입력으로 취해 double 형식의 면을 반환합니다. 예제

```
SELECT ST_Area(ST_Polygon('polygon((1 1, 4 1, 4 4, 1 4))'))
```

ST_Centroid(geometry)

geometry 데이터 형식 polygon을 입력으로 취해 다각형 포락선의 중심인 point geometry 데이터 형식을 반환합니다. 예시:

```
SELECT ST_Centroid(ST_GeometryFromText('polygon ((0 0, 3 6, 6 0, 0 0))'))
```

```
SELECT ST_AsText(ST_Centroid(ST_Envelope(ST_GeometryFromText('POINT (53 27)'))))
```

ST_ConvexHull(geometry)

지정된 입력의 모든 geometry를 둘러싸는 가장 작은 볼록 geometry인 geometry 데이터 형식을 반환합니다. 예제

```
SELECT ST_ConvexHull(ST_Point(-158.54, 61.56))
```

ST_CoordDim(geometry)

지원되는 geometry 데이터 형식 중 하나를 입력으로 취해 tinyint 형식으로 좌표 성분 개수를 반환합니다. 예제

```
SELECT ST_CoordDim(ST_Point(1.5,2.5))
```

ST_Dimension(geometry)

지원되는 geometry 데이터 형식 중 하나를 입력으로 가져오고, tinyint 형식으로 geometry의 공간 차원을 반환합니다. 예제

```
SELECT ST_Dimension(ST_Polygon('polygon((1 1, 4 1, 4 4, 1 4))'))
```

ST_Distance(geometry, geometry)

공간 참조에 기초해 두 geometry 간 2차원 최소 데카르트 거리(투영 단위, projected unit)를 포함한 double을 반환합니다. Athena 엔진 버전 2부터 입력 중 하나가 빈 지오메트리인 경우 null을 반환합니다. 예제

```
SELECT ST_Distance(ST_Point(0.0,0.0), ST_Point(3.0,4.0))
```

ST_Distance(sphericalGeography, sphericalGeography)

두 개의 구면 지리점 사이의 대원 거리(미터)를 double로 반환합니다. 예제

```
SELECT ST_Distance(to_spherical_geography(ST_Point(61.56,
 -86.67)),to_spherical_geography(ST_Point(61.56, -86.68)))
```

ST_EndPoint(geometry)

line geometry 데이터 형식의 마지막 점을 point geometry 데이터 형식으로 반환합니다. 예제

```
SELECT ST_EndPoint(ST_Line('linestring(0 2, 2 2)'))
```

ST_Geometries(geometry)

지정된 컬렉션의 geometry 배열을 반환합니다. 지정된 geometry가 다중 geometry가 아닌 경우 1 요소 배열을 반환합니다. 지정된 geometry가 비어 있으면 null을 반환합니다.

예를 들어, MultiLineString 객체가 주어진 경우 ST_Geometries가 LineString 객체의 배열을 생성합니다. GeometryCollection 객체가 주어지면 ST_Geometries는 구성 요소가 평탄화되지 않은 배열을 반환합니다. 예제

```
SELECT ST_Geometries(GEOMETRYCOLLECTION(MULTIPOINT(0 0, 1 1),
 GEOMETRYCOLLECTION(MULTILINESTRING((2 2, 3 3)))))
```

결과:

```
array[MULTIPOINT(0 0, 1 1),GEOMETRYCOLLECTION(MULTILINESTRING((2 2, 3 3)))]
```

ST_GeometryN(geometry, index)

지정된 정수 인덱스의 geometry 요소를 geometry 데이터 형식으로 반환합니다. 인덱스는 1에서 시작합니다. 지정된 geometry가 geometry의 컬렉션(예: GEOMETRYCOLLECTION 또는 MULTI* 객체)인 경우 지정된 인덱스의 geometry를 반환합니다. 지정된 인덱스가 1보다 작거나 컬렉션의 전체 요소 개수보다 크면 null을 반환합니다. 전체 요소 개수를 찾으려면 ST_NumGeometries를 사용합니다. 단일 geometry(예: POINT, LINESTRING 또는 POLYGON)는 한 요소의 컬렉션으로 처리됩니다. 빈 geometry는 빈 컬렉션으로 처리됩니다. 예제

```
SELECT ST_GeometryN(ST_Point(-158.54, 61.56),1)
```

ST_GeometryType(geometry)

geometry의 형식을 varchar로 반환합니다. 예제

```
SELECT ST_GeometryType(ST_Point(-158.54, 61.56))
```

ST_InteriorRingN(geometry, index)

지정된 인덱스의 내부 링 요소를 반환합니다(인덱스는 1에서 시작). 주어진 인덱스가 1보다 작거나 지정된 geometry에 있는 내부 링의 전체 요소 개수보다 크면 null을 반환합니다. 지정된 geometry가 폴리곤이 아니면 오류가 발생합니다. 전체 요소 개수를 찾으려면 ST_NumInteriorRing를 사용합니다. 예제

```
SELECT ST_InteriorRingN(st_polygon('polygon ((0 0, 1 0, 1 1, 0 1, 0 0))'),1)
```

ST_InteriorRings(geometry)

지정된 geometry에서 검색된 모든 내부 링의 geometry 배열을 반환하거나, 폴리곤에 내부 링이 없는 경우 빈 배열을 반환합니다. 지정된 geometry가 비어 있으면 null을 반환합니다. 지정된 geometry가 다각형이 아니면 오류가 발생합니다. 예제

```
SELECT ST_InteriorRings(st_polygon('polygon ((0 0, 1 0, 1 1, 0 1, 0 0))'))
```

ST_IsClosed(geometry)

입력으로 `line` 및 `multiline` geometry 데이터 형식만 가져옵니다. 라인이 닫힌 경우에만 TRUE(`boolean` 형식)를 반환합니다. 예제

```
SELECT ST_IsClosed(ST_Line('linestring(0 2, 2 2)'))
```

ST_IsEmpty(geometry)

입력으로 `line` 및 `multiline` geometry 데이터 형식만 가져옵니다. 지정된 geometry가 비어 있는 경우, 즉 `line` 시작 및 끝 값이 일치하는 경우에만 TRUE(`boolean` 형식)를 반환합니다. 예제

```
SELECT ST_IsEmpty(ST_Point(1.5, 2.5))
```

ST_IsRing(geometry)

`line` 형식이 닫히고 단순한 경우에만 TRUE(`boolean` 형식)를 반환합니다. 예제

```
SELECT ST_IsRing(ST_Line('linestring(0 2, 2 2)'))
```

ST_IsSimple(geometry)

지정된 geometry에 변칙적인 기하학적 점(예: 자체 교차 또는 자체 접점)이 없으면 true를 반환합니다. geometry가 단순하지 않은 이유를 확인하려면 geometry_invalid_reason()을 사용합니다. 예제

```
SELECT ST_IsSimple(ST_LineString(array[ST_Point(1,2), ST_Point(3,4)]))
```

ST_IsValid(geometry)

지정된 geometry가 잘 형성되어 있는 경우에만 true를 반환합니다. geometry가 잘 형성되지 않은 이유를 확인하려면 geometry_invalid_reason()을 사용합니다. 예제

```
SELECT ST_IsValid(ST_Point(61.56, -86.68))
```

ST_Length(geometry)

double 형식으로 line 길이를 반환합니다. 예제

```
SELECT ST_Length(ST_Line('linestring(0 2, 2 2)'))
```

ST_NumGeometries(geometry)

컬렉션의 geometry 개수를 정수로 반환합니다. geometry가 geometry의 컬렉션인 경우(예: GEOMETRYCOLLECTION 또는 MULTI* 객체) geometry의 개수를 반환합니다. 단일 geometry는 1을 반환하고 빈 geometry는 0을 반환합니다. GEOMETRYCOLLECTION 객체의 빈 geometry는 1개의 geometry로 계수됩니다. 예를 들어 아래의 경우에는 1로 평가됩니다.

```
ST_NumGeometries(ST_GeometryFromText('GEOMETRYCOLLECTION(MULTIPOINT EMPTY)'))
```

ST_NumInteriorRing(geometry)

polygon geometry의 내부 링 개수를 bigint 형식으로 반환합니다. 예제

```
SELECT ST_NumInteriorRing(ST_Polygon('polygon ((0 0, 8 0, 0 8, 0 0), (1 1, 1 5, 5 1, 1
 1))'))
```

ST_NumPoints(geometry)

geometry의 지점 개수를 bigint 형식으로 반환합니다. 예제

```
SELECT ST_NumPoints(ST_Point(1.5, 2.5))
```

ST_PointN(lineString, index)

지정된 정수 인덱스의 지정된 라인 스트링의 꼭지점을 point geometry 데이터 형식으로 반환합니다. 인덱스는 1에서 시작합니다. 주어진 인덱스가 1보다 작거나 컬렉션의 전체 요소 개수보다 크면 null을 반환합니다. 전체 요소 개수를 찾으려면 ST_NumPoints를 사용합니다. 예제

```
SELECT ST_PointN(ST_LineString(array[ST_Point(1,2), ST_Point(3,4)]),1)
```

ST_Points(geometry)

지정된 선 스트링 geometry 객체의 점 배열을 반환합니다. 예제

```
SELECT ST_Points(ST_LineString(array[ST_Point(1,2), ST_Point(3,4)]))
```

ST_StartPoint(geometry)

line geometry 데이터 형식의 첫 점을 point geometry 데이터 형식으로 반환합니다. 예제

```
SELECT ST_StartPoint(ST_Line('linestring(0 2, 2 2)'))
```

ST_X(point)

지점의 X 좌표를 double 형식으로 반환합니다. 예제

```
SELECT ST_X(ST_Point(1.5, 2.5))
```

ST_XMax(geometry)

double 형식으로 geometry의 최대 X 좌표를 반환합니다. 예제

```
SELECT ST_XMax(ST_Line('linestring(0 2, 2 2)'))
```

ST_XMin(geometry)

double 형식으로 geometry의 최소 X 좌표를 반환합니다. 예제

```
SELECT ST_XMin(ST_Line('linestring(0 2, 2 2)'))
```

ST_Y(point)

지점의 Y 좌표를 double 형식으로 반환합니다. 예제

```
SELECT ST_Y(ST_Point(1.5, 2.5))
```

ST_YMax(geometry)

double 형식으로 geometry의 최대 Y 좌표를 반환합니다. 예제

```
SELECT ST_YMax(ST_Line('linestring(0 2, 2 2)'))
```

ST_YMin(geometry)

double 형식으로 geometry의 최소 Y 좌표를 반환합니다. 예제

```
SELECT ST_YMin(ST_Line('linestring(0 2, 2 2)'))
```

집계 함수

convex_hull_agg(geometry)

입력으로 전달된 모든 geometry를 둘러싸는 최소 볼록 geometry를 반환합니다.

geometry_union_agg(geometry)

입력으로 전달된 모든 geometry의 점 집합 합집합을 나타내는 geometry를 반환합니다.

Bing 타일 함수

다음 함수는 Microsoft Bing maps tile system(Bing 지도 타일 시스템)에서 geometry와 타일을 상호 간에 변환합니다.

bing_tile(x, y, zoom_level)

정수 좌표 x 및 y와 지정된 확대/축소의 수준에서 Bing 타일 객체를 반환합니다. 확대/축소 수준은 1~23의 정수여야 합니다. 예제

```
SELECT bing_tile(10, 20, 12)
```

bing_tile(quadKey)

Quadkey에서 Bing 타일 객체를 반환합니다. 예제

```
SELECT bing_tile(bing_tile_quadkey(bing_tile(10, 20, 12)))
```

bing_tile_at(latitude, longitude, zoom_level)

지정된 위도, 경도, 확대/축소 수준에서 Bing 타일 객체를 반환합니다. 위도는 -85.05112878과 85.05112878 사이여야 합니다. 경도는 -180과 180 사이여야 합니다. `latitude` 및 `longitude` 값은 `double`이어야 하며, `zoom_level`은 정수여야 합니다. 예제

```
SELECT bing_tile_at(37.431944, -122.166111, 12)
```

bing_tiles_around(latitude, longitude, zoom_level)

지정된 확대/축소 수준에서 지정된 위도 및 경도 점을 둘러싸는 Bing 타일 배열을 반환합니다. 예제

```
SELECT bing_tiles_around(47.265511, -122.465691, 14)
```

bing_tiles_around(latitude, longitude, zoom_level, radius_in_km)

지정된 확대/축소 수준에서 Bing 타일 배열을 반환합니다. 배열에는 지정된 위도와 경도에서 지정된 반지름(킬로미터 단위)의 원을 덮는 최소 Bing 타일 집합이 포함됩니다. `latitude`, `longitude` 및 `radius_in_km` 값은 `double`이고, 확대/축소 수준은 `integer`입니다. 예제

```
SELECT bing_tiles_around(37.8475, 112.596667, 10, .5)
```

bing_tile_coordinates(tile)

지정된 Bing 타일의 x 및 y 좌표를 반환합니다. 예제

```
SELECT bing_tile_coordinates(bing_tile_at(37.431944, -122.166111, 12))
```

bing_tile_polygon(tile)

지정된 Bing 타일의 폴리곤 표현을 반환합니다. 예제

```
SELECT bing_tile_polygon(bing_tile_at(47.265511, -122.465691, 4))
```

bing_tile_quadkey(tile)

지정된 Bing 타일의 quadkey를 반환합니다. 예제

```
SELECT bing_tile_quadkey(bing_tile(52, 143, 10))
```

bing_tile_zoom_level(tile)

지정된 Bing 타일의 확대/축소 수준을 정수로 반환합니다. 예제

```
SELECT bing_tile_zoom_level(bing_tile(52, 143, 10))
```

geometry_to_bing_tiles(geometry, zoom_level)

지정된 확대/축소 수준에서 지정된 geometry를 완전히 덮는 최소 Bing 타일 집합을 반환합니다. 1~23의 확대/축소 수준이 지원됩니다. 예제

```
SELECT geometry_to_bing_tiles(ST_Point(61.56, 58.54), 10)
```

Athena 엔진 버전 2의 지리 공간 함수 이름 변경 및 새 함수

이 단원에는 Athena 엔진 버전 2에 지리 공간 함수 이름 변경 및 새로 도입된 지리 공간 함수가 나열되어 있습니다. Athena 엔진 버전 2의 현재 다른 변경 사항에 대한 내용은 Athena 엔진 버전 2 섹션을 참조하세요.

Athena 엔진 버전 관리에 대한 자세한 내용은 Athena 엔진 버전 관리 단원을 참조하세요.

Athena 엔진 버전 2의 지리 공간 함수 이름 변경

다음 함수의 이름이 변경되었습니다. 일부는 입력 및 출력 형식도 변경되었습니다. 자세한 내용은 해당 링크를 참조하세요.

이전 함수 이름	Athena 엔진 버전 2부터 적용되는 함수 이름
st_coordinate_dimension	ST_CoordDim
st_end_point	ST_EndPoint
st_exterior_ring	ST_ExteriorRing

이전 함수 이름	Athena 엔진 버전 2부터 적용되는 함수 이름
st_interior_ring_number	ST_NumInteriorRing
st_geometry_from_text	ST_GeometryFromText
st_is_closed	ST_IsClosed
st_is_empty	ST_IsEmpty
st_is_ring	ST_IsRing
st_max_x	ST_XMax
st_max_y	ST_YMax
st_min_x	ST_XMin
st_min_y	ST_YMin
st_point_number	ST_NumPoints
st_start_point	ST_StartPoint
st_symmetric_difference	ST_SymDifference

Athena 엔진 버전 2의 새 지리 공간 함수

다음과 같은 지리 공간 함수가 Athena 엔진 버전 2의 현재 새 함수입니다. 자세한 내용은 해당 링크를 참조하세요.

생성자 함수

- ST_AsBinary
- ST_GeomAsLegacyBinary
- ST_GeomFromBinary
- ST_GeomFromLegacyBinary
- ST_LineString
- ST_MultiPoint

- to_geometry
- to_spherical_geography

작업 함수

- geometry_union
- ST_EnvelopeAsPts
- ST_Union

접근자 함수

- geometry_invalid_reason
- great_circle_distance
- line_locate_point
- simplify_geometry
- ST_ConvexHull
- ST_Distance(구형 지리)
- ST_Geometries
- ST_GeometryN
- ST_GeometryType
- ST_InteriorRingN
- ST_InteriorRings
- ST_IsSimple
- ST_IsValid
- ST_NumGeometries
- ST_PointN
- ST_Points

집계 함수

- convex_hull_agg
- geometry_union_agg

Bing 타일 함수

- bing_tile
- bing_tile(쿼드키)
- bing_tile_at
- bing_tiles_around
- bing_tiles_around(반지름)
- bing_tile_coordinates
- bing_tile_polygon
- bing_tile_quadkey
- bing_tile_zoom_level
- geometry_to_bing_tiles

예제: 지리 공간 쿼리

이 주제의 예제에서는 GitHub에서 사용 가능한 샘플 데이터에서 두 개의 테이블을 생성하고 데이터를 기반으로 테이블을 쿼리합니다. 샘플 데이터는 단지 설명을 위한 것이며 정확하지 않을 수 있으며 다음 파일에 있습니다.

- **`earthquakes.csv`** – 캘리포니아에서 발생한 지진이 수록되어 있습니다. 예제 `earthquakes` 테이블에서는 이 데이터의 필드를 사용합니다.
- **`california-counties.json`** – ESRI 호환 GeoJSON 형식으로 캘리포니아 주의 카운티 데이터가 수록되어 있습니다. 데이터에는 `AREA`, `PERIMETER`, `STATE`, `COUNTY` 및 `NAME` 같은 여러 필드가 포함되어 있지만 예제 `counties` 테이블에서는 `Name`(문자열) 및 `BoundaryShape`(이진) 두 개만 사용합니다.

> **Note**
>
> Athena는 `com.esri.json.hadoop.EnclosedEsriJsonInputFormat`을 사용하여 JSON 데이터를 지리 공간 이진 형식으로 변환합니다.

다음 코드 예제는 `earthquakes`라는 테이블을 생성합니다.

```
CREATE external TABLE earthquakes
```

```
(
 earthquake_date string,
 latitude double,
 longitude double,
 depth double,
 magnitude double,
 magtype string,
 mbstations string,
 gap string,
 distance string,
 rms string,
 source string,
 eventid string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE LOCATION 's3://DOC-EXAMPLE-BUCKET/my-query-log/csv/';
```

다음 코드 예제는 counties라는 테이블을 생성합니다.

```
CREATE external TABLE IF NOT EXISTS counties
 (
 Name string,
 BoundaryShape binary
 )
ROW FORMAT SERDE 'com.esri.hadoop.hive.serde.EsriJsonSerDe'
STORED AS INPUTFORMAT 'com.esri.json.hadoop.EnclosedEsriJsonInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/my-query-log/json/';
```

다음 쿼리 예제는 counties 및 earthquake 테이블에 CROSS JOIN 함수를 사용합니다. 이 예제에서는 ST_CONTAINS를 사용하여 경계에 지진 장소(ST_POINT로 지정됨)가 포함된 카운티를 쿼리합니다. 쿼리는 해당 카운티를 이름별로 그룹화하고 개수별로 정렬한 다음 내림차순으로 반환합니다.

> **Note**
>
> Athena 엔진 버전 2부터 ST_CONTAINS와 같은 함수에서는 더는 VARBINARY 유형을 입력으로 지원하지 않습니다. 이러한 이유로 이 예제에서는 ST_GeomFromLegacyBinary(varbinary) 함수를 사용하여 boundaryshape 이진 값을 geometry로 변환합니다. 자세한 내용은 Athena 엔진 버전 2 참조의 지리 공간 함수 변경 사항 단원을 참조하세요.

```
SELECT counties.name,
        COUNT(*) cnt
FROM counties
CROSS JOIN earthquakes
WHERE ST_CONTAINS (ST_GeomFromLegacyBinary(counties.boundaryshape),
 ST_POINT(earthquakes.longitude, earthquakes.latitude))
GROUP BY  counties.name
ORDER BY  cnt DESC
```

이 쿼리가 반환하는 값:

```
+------------------------+
| name             | cnt |
+------------------------+
| Kern             | 36  |
+------------------------+
| San Bernardino   | 35  |
+------------------------+
| Imperial         | 28  |
+------------------------+
| Inyo             | 20  |
+------------------------+
| Los Angeles      | 18  |
+------------------------+
| Riverside        | 14  |
+------------------------+
| Monterey         | 14  |
+------------------------+
| Santa Clara      | 12  |
+------------------------+
| San Benito       | 11  |
+------------------------+
| Fresno           | 11  |
+------------------------+
| San Diego        | 7   |
+------------------------+
| Santa Cruz       | 5   |
+------------------------+
| Ventura          | 3   |
+------------------------+
| San Luis Obispo  | 3   |
+------------------------+
| Orange           | 2   |
```

```
+------------------------+
| San Mateo        | 1   |
+------------------------+
```

추가적인 리소스

지리 공간 쿼리에 대한 추가 예제는 다음 블로그 게시물을 참조하세요.

- Extend geospatial queries in Amazon Athena with UDFs and AWS Lambda
- Amazon Athena 및 Amazon QuickSight를 사용하여 200년 이상의 세계 기후 데이터를 시각화하기.
- Amazon Athena로 OpenStreetMap 쿼리

JSON 쿼리

Amazon Athena를 사용하면 JSON 인코딩 값을 구문 분석하고, JSON에서 데이터를 추출하고, 값을 검색하고, JSON 어레이의 길이와 크기를 찾을 수 있습니다.

주제

- JSON 데이터 읽기 모범 사례
- JSON에서 데이터 추출
- JSON 배열에서 값 검색
- JSON 배열의 길이와 크기 획득
- JSON 쿼리 문제 해결

JSON 데이터 읽기 모범 사례

JavaScript Object Notation(JSON)은 데이터 구조를 텍스트로 인코딩하는 일반적인 방법입니다. 많은 애플리케이션과 도구가 JSON 인코딩 데이터를 출력합니다.

Amazon Athena에서 외부 데이터로 테이블을 만들고 JSON 인코딩 데이터를 포함할 수 있습니다. 이러한 유형의 소스 데이터에는 JSON SerDe 라이브러리와 함께 Athena를 사용합니다.

JSON 인코딩 데이터를 읽을 때는 다음 팁을 참조합니다.

- 기본 JSON SerDe `org.apache.hive.hcatalog.data.JsonSerDe` 또는 OpenX SerDe `org.openx.data.jsonserde.JsonSerDe` 중 올바른 SerDe를 선택합니다. 자세한 내용은 JSON SerDe 라이브러리 단원을 참조하십시오.

- 각각의 JSON 인코딩 레코드가 가독성 좋게 꾸며서 표시되는 것이 아니라 별도의 라인에 표시되는지 확인합니다.

> **Note**
>
> SerDe는 각 JSON 문서가 레코드의 필드를 구분하는 줄 종료 문자가 없는 한 줄의 텍스트에 있을 것으로 예상합니다. JSON 텍스트가 가독성 좋게 꾸민 형식이면 테이블을 만든 후 쿼리하려고 할 때 HIVE_CURSOR_ERROR: 행이 유효한 JSON 객체가 아님(HIVE_CURSOR_ERROR: Row is not a valid JSON Object) 또는 HIVE_CURSOR_ERROR: JsonParseException: 예기치 않은 입력 종료: OBJECT의 닫기 마커 필요(HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT) 같은 오류 메시지가 나타날 수 있습니다. 자세한 내용은 GitHub의 OpenX SerDe 문서에서 JSON 데이터 파일을 참조하세요.

- 대/소문자 열에서 JSON 인코딩 데이터를 생성합니다.
- 이 예제에서와 같이 형식이 잘못된 레코드를 무시할 수 있는 옵션이 제공됩니다.

```
CREATE EXTERNAL TABLE json_table (
  column_a string,
  column_b int
 )
 ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
 WITH SERDEPROPERTIES ('ignore.malformed.json' = 'true')
 LOCATION 's3://DOC-EXAMPLE-BUCKET/path/';
```

- 스키마가 정해지지 않은 소스 데이터의 필드를 Athena의 JSON 인코딩 문자열로 변환합니다.

Athena가 JSON 데이터 기반의 테이블을 만들면 기존 스키마와 미리 정의된 스키마를 기반으로 데이터를 구문 분석합니다. 그러나 모든 데이터에 사전 정의된 스키마가 있는 것은 아닙니다. 이러한 경우 스키마 관리를 단순화하려면 보통 스키마가 정해지지 않은 소스 데이터의 필드를 Athena의 JSON 문자열로 변환한 다음 JSON SerDe 라이브러리를 사용하는 방법이 유용합니다.

예를 들어, 서로 다른 센서의 공통 필드를 사용하여 이벤트를 게시하는 IoT 애플리케이션을 가정해 보겠습니다. 이러한 필드 중 하나는 이벤트를 보내는 센서에 고유한 사용자 지정 페이로드를 저장해야 합니다. 이 경우 스키마를 알 수 없기 때문에 이 정보를 JSON 인코딩 문자열로 저장하는 것이 좋습니다. 이를 위해 다음 예에서와 같이 Athena 테이블의 데이터를 JSON으로 변환합니다. JSON 인코딩 데이터를 Athena 데이터 형식으로 변환할 수도 있습니다.

- Athena 데이터 형식을 JSON로 변환

- JSON을 Athena 데이터 형식으로 변환

Athena 데이터 형식을 JSON로 변환

Athena 데이터 형식을 JSON으로 변환하려면 CAST를 사용합니다.

```
WITH dataset AS (
  SELECT
    CAST('HELLO ATHENA' AS JSON) AS hello_msg,
    CAST(12345 AS JSON) AS some_int,
    CAST(MAP(ARRAY['a', 'b'], ARRAY[1,2]) AS JSON) AS some_map
)
SELECT * FROM dataset
```

이 쿼리가 반환하는 값:

```
+-----------------------------------------+
| hello_msg      | some_int | some_map      |
+-----------------------------------------+
| "HELLO ATHENA" | 12345    | {"a":1,"b":2} |
+-----------------------------------------+
```

JSON을 Athena 데이터 형식으로 변환

JSON 데이터를 Athena 데이터 형식으로 변환하려면 CAST를 사용합니다.

> **Note**
>
> 이 예에서 문자열을 JSON 인코딩 문자열로 나타내려면 JSON 키워드로 시작하고 JSON '12345'처럼 작은따옴표를 사용합니다.

```
WITH dataset AS (
  SELECT
    CAST(JSON '"HELLO ATHENA"' AS VARCHAR) AS hello_msg,
    CAST(JSON '12345' AS INTEGER) AS some_int,
    CAST(JSON '{"a":1,"b":2}' AS MAP(VARCHAR, INTEGER)) AS some_map
)
SELECT * FROM dataset
```

이 쿼리가 반환하는 값:

```
+-------------------------------------+
| hello_msg    | some_int | some_map  |
+-------------------------------------+
| HELLO ATHENA | 12345    | {a:1,b:2} |
+-------------------------------------+
```

JSON에서 데이터 추출

Athena의 테이블로 역직렬화할 필요가 없는 JSON 인코딩 문자열을 포함하는 원본 데이터가 있을 수 있습니다. 이 경우에도 Presto에 제공된 JSON 함수를 사용하여 이 데이터에 대해 SQL 작업을 실행할 수 있습니다.

이 JSON 문자열을 예제 데이터 세트로 간주합니다.

```
{"name": "Susan Smith",
"org": "engineering",
"projects":
    [
     {"name":"project1", "completed":false},
     {"name":"project2", "completed":true}
    ]
}
```

예제: 속성 추출

JSON 문자열에서 `name` 및 `projects` 속성을 추출하려면 다음 예제에서와 같이 `json_extract` 함수를 사용합니다. `json_extract` 함수는 JSON 문자열을 포함하는 열을 가져오고 `JSONPath`처럼 점 `.` 표기법으로 이루어진 식을 검색합니다.

> **Note**
>
> `JSONPath`가 단순한 트리 순회를 수행합니다. 그리고 `$` 기호를 사용해 JSON 문서의 루트를 나타내고, 그 뒤에는 마침표와 루트 바로 아래에 중첩되는 요소가 이어집니다(예: `$.name`).

```
WITH dataset AS (
  SELECT '{"name": "Susan Smith",
```

```
            "org": "engineering",
            "projects": [{"name":"project1", "completed":false},
            {"name":"project2", "completed":true}]}'
    AS myblob
)
SELECT
  json_extract(myblob, '$.name') AS name,
  json_extract(myblob, '$.projects') AS projects
FROM dataset
```

반환되는 값은 기본 Athena 데이터 유형이 아닌 JSON 인코딩 문자열입니다.

```
+-----------------------------------------------------------------------------------------------
+
| name           | projects
         |
+-----------------------------------------------------------------------------------------------
+
| "Susan Smith"  | [{"name":"project1","completed":false},
{"name":"project2","completed":true}] |
+-----------------------------------------------------------------------------------------------
+
```

JSON 문자열에서 스칼라 값을 추출하려면 `json_extract_scalar` 함수를 사용합니다. `json_extract`와 비슷하지만 스칼라 값(부울, 숫자 또는 문자열)만 반환합니다.

> **Note**
>
> 어레이, 맵 또는 구조체에 `json_extract_scalar` 함수를 사용하지 마세요.

```
WITH dataset AS (
  SELECT '{"name": "Susan Smith",
            "org": "engineering",
            "projects": [{"name":"project1", "completed":false},{"name":"project2",
 "completed":true}]}'
    AS myblob
)
SELECT
  json_extract_scalar(myblob, '$.name') AS name,
  json_extract_scalar(myblob, '$.projects') AS projects
```

```
FROM dataset
```

이 쿼리가 반환하는 값:

```
+---------------------------+
| name           | projects |
+---------------------------+
| Susan Smith    |          |
+---------------------------+
```

예제 어레이에서 `projects` 속성의 첫 번째 요소를 얻으려면 `json_array_get` 함수를 사용하고 인덱스 위치를 지정합니다.

```
WITH dataset AS (
  SELECT '{"name": "Bob Smith",
           "org": "engineering",
           "projects": [{"name":"project1", "completed":false},{"name":"project2",
 "completed":true}]}'
    AS myblob
)
SELECT json_array_get(json_extract(myblob, '$.projects'), 0) AS item
FROM dataset
```

JSON 인코딩 어레이에서 지정된 인덱스 위치에 있는 값을 반환합니다.

```
+---------------------------------------+
| item                                  |
+---------------------------------------+
| {"name":"project1","completed":false} |
+---------------------------------------+
```

Athena 문자열 유형이 반환되게 하려면 `JSONPath` 식 내에 `[]` 연산자를 사용한 다음 `json_extract_scalar` 함수를 사용합니다. `[]`에 대한 자세한 정보는 배열 요소에 액세스 섹션을 참조하십시오.

```
WITH dataset AS (
   SELECT '{"name": "Bob Smith",
             "org": "engineering",
             "projects": [{"name":"project1", "completed":false},{"name":"project2",
 "completed":true}]}'
```

```
     AS myblob
)
SELECT json_extract_scalar(myblob, '$.projects[0].name') AS project_name
FROM dataset
```

다음 결과를 반환합니다.

```
+--------------+
| project_name |
+--------------+
| project1     |
+--------------+
```

JSON 배열에서 값 검색

특정 값이 JSON 인코딩 배열 내에 있는지 알아보려면 `json_array_contains` 함수를 사용합니다.

다음 쿼리는 "project2"에 참여하는 사용자의 이름을 나열합니다.

```
WITH dataset AS (
  SELECT * FROM (VALUES
    (JSON '{"name": "Bob Smith", "org": "legal", "projects": ["project1"]}'),
    (JSON '{"name": "Susan Smith", "org": "engineering", "projects": ["project1",
 "project2", "project3"]}'),
    (JSON '{"name": "Jane Smith", "org": "finance", "projects": ["project1",
 "project2"]}')
  ) AS t (users)
)
SELECT json_extract_scalar(users, '$.name') AS user
FROM dataset
WHERE json_array_contains(json_extract(users, '$.projects'), 'project2')
```

이 쿼리는 사용자 목록을 반환합니다.

```
+-------------+
| user        |
+-------------+
| Susan Smith |
+-------------+
| Jane Smith  |
+-------------+
```

다음 쿼리 예제는 완료된 프로젝트의 총 개수와 함께 프로젝트를 완료한 사용자의 이름을 나열합니다. 그리고 이러한 작업을 수행합니다.

- 명확성을 위해 중첩 SELECT 설명을 사용합니다.
- 프로젝트의 배열을 추출합니다.
- CAST를 사용하여 배열을 키-값 페어의 기본 배열로 변환합니다.
- UNNEST 연산자를 사용하여 각각의 배열 요소를 추출합니다.
- 획득한 값을 완료된 프로젝트별로 필터링하고 개수를 셉니다.

> **Note**
>
> MAP에 CAST를 사용할 때 키 요소를 VARCHAR(Presto의 기본 문자열)로 지정할 수 있지만 값은 JSON으로 남겨 둡니다. MAP의 값 유형이 서로 다르기 때문입니다(첫 번째 키-값 페어는 문자열, 두 번째는 부울).

```
WITH dataset AS (
  SELECT * FROM (VALUES
    (JSON '{"name": "Bob Smith",
             "org": "legal",
             "projects": [{"name":"project1", "completed":false}]}'),
    (JSON '{"name": "Susan Smith",
             "org": "engineering",
             "projects": [{"name":"project2", "completed":true},
                          {"name":"project3", "completed":true}]}'),
    (JSON '{"name": "Jane Smith",
             "org": "finance",
             "projects": [{"name":"project2", "completed":true}]}')
  ) AS t (users)
),
employees AS (
  SELECT users, CAST(json_extract(users, '$.projects') AS
    ARRAY(MAP(VARCHAR, JSON))) AS projects_array
  FROM dataset
),
names AS (
  SELECT json_extract_scalar(users, '$.name') AS name, projects
  FROM employees, UNNEST (projects_array) AS t(projects)
)
```

```
SELECT name, count(projects) AS completed_projects FROM names
WHERE cast(element_at(projects, 'completed') AS BOOLEAN) = true
GROUP BY name
```

이 쿼리는 다음 결과를 반환합니다.

```
+----------------------------------+
| name        | completed_projects |
+----------------------------------+
| Susan Smith | 2                  |
+----------------------------------+
| Jane Smith  | 1                  |
+----------------------------------+
```

JSON 배열의 길이와 크기 획득

예제: **json_array_length**

JSON 인코딩 배열의 길이를 얻으려면 json_array_length 함수를 사용합니다.

```
WITH dataset AS (
  SELECT * FROM (VALUES
    (JSON '{"name":
            "Bob Smith",
            "org":
            "legal",
            "projects": [{"name":"project1", "completed":false}]}'),
    (JSON '{"name": "Susan Smith",
            "org": "engineering",
            "projects": [{"name":"project2", "completed":true},
                         {"name":"project3", "completed":true}]}'),
    (JSON '{"name": "Jane Smith",
             "org": "finance",
             "projects": [{"name":"project2", "completed":true}]}')
  ) AS t (users)
)
SELECT
  json_extract_scalar(users, '$.name') as name,
  json_array_length(json_extract(users, '$.projects')) as count
FROM dataset
ORDER BY count DESC
```

이 쿼리는 다음 결과를 반환합니다.

```
+---------------------+
| name        | count |
+---------------------+
| Susan Smith | 2     |
+---------------------+
| Bob Smith   | 1     |
+---------------------+
| Jane Smith  | 1     |
+---------------------+
```

예제: **json_size**

JSON 인코딩 배열이나 객체의 크기를 얻으려면 json_size 함수를 사용하고 JSON 문자열과 JSONPath 표현식이 포함된 열을 배열이나 객체에 지정합니다.

```
WITH dataset AS (
  SELECT * FROM (VALUES
    (JSON '{"name": "Bob Smith", "org": "legal", "projects": [{"name":"project1",
 "completed":false}]}'),
    (JSON '{"name": "Susan Smith", "org": "engineering", "projects":
 [{"name":"project2", "completed":true},{"name":"project3", "completed":true}]}'),
    (JSON '{"name": "Jane Smith", "org": "finance", "projects": [{"name":"project2",
 "completed":true}]}')
  ) AS t (users)
)
SELECT
  json_extract_scalar(users, '$.name') as name,
  json_size(users, '$.projects') as count
FROM dataset
ORDER BY count DESC
```

이 쿼리는 다음 결과를 반환합니다.

```
+---------------------+
| name        | count |
+---------------------+
| Susan Smith | 2     |
+---------------------+
| Bob Smith   | 1     |
+---------------------+
| Jane Smith  | 1     |
+---------------------+
```

JSON 쿼리 문제 해결

JSON 관련 쿼리 문제 해결에 대한 도움말은 JSON 관련 오류 또는 다음 리소스를 참조하세요.

- Amazon Athena에서 JSON 데이터를 읽으려고 할 때 오류가 발생합니다.
- Athena에서 AWS Config의 파일을 읽을 때 "HIVE_CURSOR_ERROR: Row is not a valid JSON Object - JSONException: Duplicate key"를 어떻게 해결해야 합니까?
- 입력 JSON 파일에 다수의 레코드가 있는 경우에도 Amazon Athena의 SELECT COUNT 쿼리가 레코드를 1개만 반환합니다.
- Athena 테이블의 행에 대한 Amazon S3 원본 파일을 확인하려면 어떻게 해야 합니까?

또한 Amazon Athena의 SQL 쿼리에 대한 고려 사항 및 제한 사항 단원도 참조하세요.

Machine Learning(ML) with Amazon Athena 사용

Machine Learning(ML) with Amazon Athena를 통해 사용자는 Athena에서 Amazon SageMaker를 사용해 기계 학습(ML) 추론을 실행하는 SQL 문을 작성할 수 있습니다. 이 기능은 데이터 분석을 위해 ML 모델에 대한 액세스를 간소화하므로 추론을 실행하기 위해 복잡한 프로그래밍 방법을 사용할 필요가 없습니다.

ML with Athena를 사용하려면 `USING EXTERNAL FUNCTION` 절로 ML with Athena 함수를 정의합니다. 함수는 사용하려는 SageMaker 모델 엔드포인트를 가리키고 모델에 전달할 변수 이름과 데이터 유형을 지정합니다. 쿼리의 후속 절은 함수를 참조하여 값을 모델에 전달합니다. 모델은 쿼리가 전달하는 값을 기반으로 추론을 실행한 다음 추론 결과를 반환합니다. SageMaker와 SageMaker 엔드포인트의 작동 방식에 대한 자세한 내용은 Amazon SageMaker 개발자 안내서를 참조하세요.

ML with Athena 및 SageMaker 추론을 사용하여 결과 집합에서 이상값을 감지하는 예제는 AWS Big Data Blog(빅 데이터 블로그) 항목 Detecting anomalous values by invoking the Amazon Athena machine learning inference function(Amazon Athena 기계 학습 추론 기능을 호출하여 이상 값 감지)을 참조하세요.

고려 사항 및 제한

- 사용 가능한 리전 - Athena ML 기능은 Athena 엔진 버전 2 이상이 지원되는 AWS 리전에서 제공됩니다.
- SageMaker 모델 엔드포인트가 **`text/csv`**를 수락하고 반환해야 함 - 데이터 형식에 대한 자세한 내용은 Amazon SageMaker 개발자 안내서의 추론을 위한 공통 데이터 형식을 참조하세요.

- Athena는 CSV 헤더를 전송하지 않음 - SageMaker 엔드포인트가 `text/csv`인 경우 입력 핸들러는 입력의 첫 번째 줄이 CSV 헤더라고 가정해서는 안 됩니다. Athena는 CSV 헤더를 전송하지 않으므로 Athena로 반환되는 출력에는 Athena가 예상한 것보다 포함된 행이 하나 적어서 오류가 발생합니다.
- SageMaker 엔드포인트 확장 – 참조된 SageMaker 모델 엔드포인트가 엔드포인트에 대한 Athena 호출을 위해 충분히 확장되었는지 확인합니다. 자세한 내용은 Amazon SageMaker 개발자 안내서의 SageMaker 모델 자동 크기 조정 및 Amazon SageMaker API 참조의 CreateEndpointConfig를 참조하세요.
- IAM 권한 - ML with Athena 함수를 지정하는 쿼리를 실행하려면 쿼리를 실행하는 IAM 보안 주체가 참조된 SageMaker 모델 엔드포인트에 대한 `sagemaker:InvokeEndpoint` 작업을 수행할 수 있어야 합니다. 자세한 내용은 ML과 Athena에 대한 액세스 허용 단원을 참조하세요.
- ML with Athena 함수는 **`GROUP BY`** 절에서 직접 사용할 수 없습니다.

ML with Athena 구문

`USING EXTERNAL FUNCTION` 절은 쿼리의 후속 `SELECT` 문에서 참조할 수 있는 ML with Athena 함수 또는 여러 함수를 지정합니다. 변수 및 반환 값에 대한 함수 이름, 변수 이름 및 데이터 유형을 정의합니다.

시놉시스

다음 구문은 ML with Athena 함수를 지정하는 `USING EXTERNAL FUNCTION` 절을 보여줍니다.

```
USING EXTERNAL FUNCTION ml_function_name (variable1 data_type[, variable2 data_type]
[,...])
RETURNS data_type
SAGEMAKER 'sagemaker_endpoint'
SELECT ml_function_name()
```

파라미터

USING EXTERNAL FUNCTION ***`ml_function_name`*** (***`variable1 data_type`***[, ***`variable2 data_type`***][,...])

*`ml_function_name`*은 후속 쿼리 절에서 사용할 수 있는 함수 이름을 정의합니다. 각 *`variable data_type`*은 SageMaker 모델이 입력으로 수락하는 명명된 변수와 해당 데이터 형식을 지정합니다. 지정된 데이터 유형은 지원되는 Athena 데이터 형식이어야 합니다.

RETURNS ***data_type***

*data_type*은 *ml_function_name*이 쿼리에 반환하는 SQL 데이터 형식을 SageMaker 모델의 출력으로 지정합니다.

SAGEMAKER '***sagemaker_endpoint***'

*sagemaker_endpoint*는 SageMaker 모델의 엔드포인트를 지정합니다.

SELECT [...] ***ml_function_name***(***expression***) [...]

함수 변수 및 SageMaker 모델에 값을 전달하여 결과를 반환하는 SELECT 쿼리입니다. *ml_function_name*은 쿼리에서 이전에 정의된 함수를 지정한 후 값을 전달하기 위해 평가되는 *expression*을 지정합니다. 전달되고 반환되는 값은 `USING EXTERNAL FUNCTION` 절의 함수에 대해 지정된 해당 데이터 유형과 일치해야 합니다.

예

다음의 예제는 ML with Athena를 사용하는 쿼리를 보여줍니다.

Example

```
USING EXTERNAL FUNCTION predict_customer_registration(age INTEGER)
    RETURNS DOUBLE
    SAGEMAKER 'xgboost-2019-09-20-04-49-29-303'
SELECT predict_customer_registration(age) AS probability_of_enrolling, customer_id
     FROM "sampledb"."ml_test_dataset"
     WHERE predict_customer_registration(age) < 0.5;
```

고객 사용 사례

Machine Learning(ML) with Amazon Athena의 미리보기 버전을 사용하는 다음 동영상은 Athena와 함께 SageMaker를 사용하는 방법을 보여줍니다.

고객 이탈 예측

다음 동영상은 Athena와 Amazon SageMaker의 기계 학습 기능을 결합하여 고객 이탈을 예측하는 방법을 보여줍니다.

Amazon Athena와 Amazon SageMaker를 사용하여 고객 이탈 예측

봇넷 감지

다음 동영상은 한 기업이 Amazon Athena와 Amazon SageMaker를 사용하여 봇넷을 어떻게 감지하는지를 보여줍니다.

Amazon Athena Amazon SageMaker를 사용하여 봇넷 감지

사용자 정의 함수를 사용한 쿼리

Amazon Athena의 사용자 정의 함수(UDF)를 사용하면 사용자 지정 함수를 만들어 레코드 또는 레코드 그룹을 처리할 수 있습니다. UDF는 파라미터를 수락하고 작업을 수행한 다음 결과를 반환합니다.

Athena에서 UDF를 사용하려면 SQL 쿼리의 `SELECT` 문 앞에 `USING EXTERNAL FUNCTION` 절을 작성합니다. `SELECT` 문은 UDF를 참조하고 쿼리가 실행될 때 UDF에 전달되는 변수를 정의합니다. SQL 쿼리는 UDF를 호출할 때 Java 런타임을 사용하여 Lambda 함수를 호출합니다. UDF는 Lambda 함수 내에 Java 배포 패키지의 메서드로 정의됩니다. Lambda 함수에 대해 동일한 Java 배포 패키지에 여러 UDF를 정의할 수 있습니다. 또한 `USING EXTERNAL FUNCTION` 절에 Lambda 함수의 이름을 지정합니다.

Athena UDF에 대한 Lambda 함수를 배포하는 데는 두 가지 옵션이 있습니다. Lambda를 사용하여 함수를 직접 배포하거나 AWS Serverless Application Repository을 사용할 수 있습니다. UDF에 대한 기존 Lambda 함수를 찾으려면 퍼블릭 AWS Serverless Application Repository 또는 프라이빗 리포지토리를 검색한 다음 Lambda에 배포하면 됩니다. 또한 Java 소스 코드를 생성 또는 수정하거나, JAR 파일로 패키지하거나, Lambda 또는 AWS Serverless Application Repository을 사용하여 배포할 수 있습니다. 시작하기 위한 Java 소스 코드 및 패키지 예제는 Lambda를 사용하여 UDF 생성 및 배포 단원을 참조하세요. Lambda에 대한 자세한 내용은 AWS Lambda 개발자 안내서를 참조하세요. AWS Serverless Application Repository에 대한 자세한 내용은 AWS Serverless Application Repository개발자 안내서 단원을 참조하세요.

Athena에서 UDF를 사용하여 텍스트를 번역하고 분석하는 예제는 AWS 기계 학습 블로그 항목 Translate and analyze text using SQL functions with Amazon Athena, Amazon Translate, and Amazon Comprehend를 참조하거나 video를 시청하세요.

Amazon Athena에서 UDF를 사용하여 지리 공간 쿼리를 확장하는 예제는 AWS 빅 데이터 블로그의 Extend geospatial queries in Amazon Athena with UDFs and AWS Lambda를 참조하세요.

고려 사항 및 제한

- 기본 제공 Athena 함수 - Athena의 기본 제공 함수는 고성능으로 설계되었습니다. 가능한 경우 UDF를 통해 기본 제공 함수를 사용하는 것이 좋습니다. 기본 제공 함수에 대한 자세한 내용은 Amazon Athena의 함수 단원을 참조하세요.
- 스칼라 UDF만 지원 - Athena는 한 번에 한 행을 처리하여 하나의 열 값을 반환하는 스칼라 UDF만 지원합니다. Athena는 Lambda를 호출할 때마다 UDF에 행 배치를 전달합니다(병렬로 전달 가능). UDF와 쿼리를 설계할 때는 이 처리의 네트워크 트래픽에 미칠 수 있는 영향을 염두에 두어야 합니다.
- UDF 핸들러 함수는 축약 형식을 사용합니다.— UDF 함수에 대해 축약 형식(전체 형식이 아님)을 사용합니다(예: `package.Class::method` 대신 `package.Class`).
- UDF 메서드는 소문자여야 합니다.— UDF 메서드는 소문자여야 합니다. 낙타 대문자는 허용되지 않습니다.
- UDF 메서드에 파라미터가 필요합니다. - UDF 메서드에 하나 이상의 파라미터가 있어야 합니다. 입력 파라미터 없이 정의된 UDF를 간접적으로 호출하려고 하면 런타임 예외가 발생합니다. UDF는 데이터 레코드에서 함수를 수행하도록 고안되었지만 인수가 없는 UDF는 데이터를 받아들이지 않으므로 예외가 발생합니다.
- Java 런타임 지원 - 현재 Athena UDF는 Lambda용 Java 8 및 Java 11 런타임을 지원합니다. 자세한 내용은 AWS Lambda 개발자 안내서의 Java로 Lambda 함수 구축을 참조하세요.
- IAM 권한 – Athena에서 UDF 쿼리 문을 실행하고 생성하려면 쿼리를 실행하는 IAM 보안 주체가 Athena 함수 외의 작업을 수행할 수 있어야 합니다. 자세한 내용은 Amazon Athena 사용자 정의 함수(UDF)를 허용하는 IAM 권한 정책의 예 단원을 참조하세요.
- Lambda 할당량 - Lambda 할당량이 UDF에 적용됩니다. 자세한 내용은 AWS Lambda 개발자 안내서에서 Lambda 할당량을 참조하십시오.
- 행 수준 필터링 - UDF에서 Lake Formation 행 수준 필터링은 지원되지 않습니다.
- 뷰 – UDF에는 뷰를 사용할 수 없습니다.
- 알려진 문제 - 알려진 문제의 최신 목록은 GitHub의 awslabs/aws-athena-query-federation 섹션에서 제한 사항 및 문제를 참조하세요.

비디오

Athena에서 UDF 사용하기에 대해 자세히 알아보려면 다음 동영상을 시청하세요.

동영상: Amazon Athena의 사용자 정의 함수(UDF) 소개

다음 동영상은 Amazon Athena에서 UDF를 사용하여 민감한 정보를 수정하는 방법을 보여줍니다.

> **Note**
>
> 이 동영상의 구문은 사전 시연용이지만 개념은 동일합니다.
> `AmazonAthenaPreviewFunctionality` 작업 그룹 없이 Athena를 사용합니다.

Amazon Athena의 사용자 정의 함수(UDF) 소개

동영상: Amazon Athena에서 SQL 쿼리를 사용하여 텍스트 필드 번역, 분석 및 수정

다음 동영상은 다른 AWS 서비스와 함께 Amazon Athena의 UDF를 사용하여 텍스트를 번역하고 분석할 수 있는 방법을 보여줍니다.

> **Note**
>
> 이 동영상의 구문은 사전 시연용이지만 개념은 동일합니다. 정확한 구문은 AWS 기계 학습 블로그에서 관련 블로그 게시물 Translate, redact, and analyze text using SQL functions with Amazon Athena, Amazon Translate, and Amazon Comprehend를 참조하세요.

Amazon Athena에서 SQL 쿼리를 사용하여 텍스트 필드 번역, 분석 및 수정

UDF 쿼리 구문

`USING EXTERNAL FUNCTION` 절은 쿼리의 후속 `SELECT` 문에서 참조할 수 있는 단일 UDF 또는 여러 UDF를 지정합니다. UDF의 메서드 이름과 UDF를 호스팅하는 Lambda 함수의 이름이 필요합니다. Lambda 함수 이름 대신 Lambda ARN을 사용할 수 있습니다. 크로스 계정 시나리오에서는 Lambda ARN이 필요합니다.

시놉시스

```
USING EXTERNAL FUNCTION UDF_name(variable1 data_type[, variable2 data_type][,...])
RETURNS data_type
LAMBDA 'lambda_function_name_or_ARN'
[, EXTERNAL FUNCTION UDF_name2(variable1 data_type[, variable2 data_type][,...])
RETURNS data_type
LAMBDA 'lambda_function_name_or_ARN'[,...]]
SELECT  [...] UDF_name(expression) [, UDF_name2(expression)] [...]
```

파라미터

USING EXTERNAL FUNCTION *UDF_name*(*variable1 data_type*[, *variable2 data_type*][,...])

*UDF_name*은 참조된 Lambda 함수 내에서 Java 메서드에 해당하는 UDF의 이름을 지정합니다. 각 *variable data_type*은 UDF가 입력으로 수락하는 명명된 변수와 해당 데이터 형식을 지정합니다. *data_type*은 다음 표에 나열된 지원되는 Athena 데이터 형식 중 하나여야 하며 해당 Java 데이터 형식에 매핑되어야 합니다.

Athena 데이터 형식	Java 데이터 유형
TIMESTAMP	java.time.LocalDateTime(UTC)
날짜	java.time.LocalDate(UTC)
TINYINT	java.lang.Byte
SMALLINT	java.lang.Short
REAL	java.lang.Float
DOUBLE	java.lang.Double
DECIMA (RETURNS 참고 사항 참조)	java.math.BigDecimal
BIGINT	java.lang.Long
INTEGER	java.lang.Int
VARCHAR	java.lang.String
VARBINARY	byte[]
BOOLEAN	java.lang.Boolean
ARRAY	java.util.List
ROW	java.util.Map<String, Object>

RETURNS ***data_type***

data_type은 UDF가 출력으로 반환하는 SQL 데이터 형식을 지정합니다. 위의 표에 나열된 Athena 데이터 형식이 지원됩니다. DECIMAL 데이터 형식의 경우 *precision*과 *scale*이 정수인 RETURNS DECIMAL(*precision*, *scale*) 구문을 사용합니다.

LAMBDA '***lambda_function***'

*lambda_function*은 UDF를 실행할 때 호출되는 Lambda 함수의 이름을 지정합니다.

SELECT [...] ***UDF_name***(***expression***) [...]

UDF에 값을 전달하고 결과를 반환하는 SELECT 쿼리입니다. *UDF_name*은 사용할 UDF를 지정하며, 값을 전달하기 위해 평가되는 *expression*이 뒤에 옵니다. 전달되고 반환되는 값은 USING EXTERNAL FUNCTION 절의 UDF에 대해 지정된 해당 데이터 유형과 일치해야 합니다.

예제

GitHub의 AthenaUDFHandler.java 코드에 기반한 쿼리 예제는 GitHub Amazon Athena UDF 커넥터페이지를 참조하세요.

Lambda를 사용하여 UDF 생성 및 배포

사용자 지정 UDF를 생성하려면 UserDefinedFunctionHandler 클래스를 확장하여 새 Java 클래스를 생성합니다. SDK의 UserDefinedFunctionHandler.java용 소스 코드는 GitHub의 awslabs/aws-athena-query-federation/athena-federation-sdk 리포지토리에서 이용할 수 있습니다. 이와 함께 사용자 지정 UDF를 생성하기 위해 검토하고 수정할 수 있는 UDF 구현 예제도 제공합니다.

이 섹션의 단계에서는 명령줄과 배포에서 Apache Maven을 사용하여 사용자 지정 UDF Jar 파일을 작성하고 구축하는 방법을 보여줍니다.

Maven을 사용하여 Athena에 대한 사용자 지정 UDF를 만드는 단계

- SDK 복제 및 개발 환경 준비
- Maven 프로젝트 만들기
- Maven 프로젝트에 종속성 및 플러그인 추가
- UDF에 대한 Java 코드 작성
- JAR 파일 구축
- AWS Lambda에 JAR 배포

SDK 복제 및 개발 환경 준비

시작하기 전에 git가 `sudo yum install git -y`를 사용하여 시스템에 설치되어 있는지 확인하세요.

AWS query federation SDK를 설치하려면

- 명령줄에 다음을 입력하여 SDK 리포지토리를 복제합니다. 이 리포지토리에는 SDK, 예제 및 데이터 소스 커넥터 제품군이 포함되어 있습니다. 데이터 소스 커넥터에 대한 자세한 내용은 Amazon Athena 연합 쿼리 사용 단원을 참고하세요.

```
git clone https://github.com/awslabs/aws-athena-query-federation.git
```

이 절차의 사전 조건을 설치하려면

이미 Apache Maven, AWS CLI 및AWS Serverless Application Model 빌드 도구가 설치된 개발 머신에서 작업하는 경우 이 단계를 건너뛸 수 있습니다.

1. 복제할 때 생성한 `aws-athena-query-federation` 디렉터리의 루트에서 개발 환경을 준비하는 prepare_dev_env.sh 스크립트를 실행합니다.
2. 을 업데이트하여 설치 프로세스에서 생성된 새 변수를 소싱하거나 터미널 세션을 다시 시작합니다.

```
source ~/.profile
```

> **Important**
>
> 이 단계를 건너뛰면 나중에 Lambda 함수를 게시할 수 없는 AWS CLI 또는 AWS SAM 빌드 도구에 대한 오류가 발생합니다.

Maven 프로젝트 만들기

다음 명령을 실행하여 Maven 프로젝트를 만듭니다. *groupId*를 고유한 조직 ID로 바꾸고 *my-athena-udf*를 애플리케이션 이름으로 바꿉니다. 자세한 내용은 Apache Maven 문서에서 첫 번째 Maven 프로젝트를 만들려면 어떻게 해야 합니까?를 참조하세요.

```
mvn -B archetype:generate \
```

```
-DarchetypeGroupId=org.apache.maven.archetypes \
-DgroupId=groupId \
-DartifactId=my-athena-udfs
```

Maven 프로젝트에 종속성 및 플러그인 추가

Maven 프로젝트 pom.xml 파일에 다음 구성을 추가합니다. 예를 들어 GitHub의 pom.xml 파일을 참조하세요.

```
<properties>
    <aws-athena-federation-sdk.version>2022.47.1</aws-athena-federation-sdk.version>
</properties>

<dependencies>
    <dependency>
        <groupId>com.amazonaws</groupId>
        <artifactId>aws-athena-federation-sdk</artifactId>
        <version>${aws-athena-federation-sdk.version}</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-shade-plugin</artifactId>
            <version>3.2.1</version>
            <configuration>
                <createDependencyReducedPom>false</createDependencyReducedPom>
                <filters>
                    <filter>
                        <artifact>*:*</artifact>
                        <excludes>
                            <exclude>META-INF/*.SF</exclude>
                            <exclude>META-INF/*.DSA</exclude>
                            <exclude>META-INF/*.RSA</exclude>
                        </excludes>
                    </filter>
                </filters>
            </configuration>
            <executions>
                <execution>
                    <phase>package</phase>
```

```
                    <goals>
                        <goal>shade</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

UDF에 대한 Java 코드 작성

UserDefinedFunctionHandler.java를 확장하여 새 클래스를 만듭니다. 클래스 내부에 UDF를 작성합니다.

다음 예제에서는 UDF에 대한 두 개의 Java 메서드인 `compress()` 및 `decompress()`가 클래스 `MyUserDefinedFunctions` 내에 만들어집니다.

```
*package *com.mycompany.athena.udfs;

public class MyUserDefinedFunctions
        extends UserDefinedFunctionHandler
{
    private static final String SOURCE_TYPE = "MyCompany";

    public MyUserDefinedFunctions()
    {
        super(SOURCE_TYPE);
    }

    /**
     * Compresses a valid UTF-8 String using the zlib compression library.
     * Encodes bytes with Base64 encoding scheme.
     *
     * @param input the String to be compressed
     * @return the compressed String
     */
    public String compress(String input)
    {
        byte[] inputBytes = input.getBytes(StandardCharsets.UTF_8);

        // create compressor
        Deflater compressor = new Deflater();
        compressor.setInput(inputBytes);
```

```
        compressor.finish();

        // compress bytes to output stream
        byte[] buffer = new byte[4096];
        ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream(inputBytes.length);
        while (!compressor.finished()) {
            int bytes = compressor.deflate(buffer);
            byteArrayOutputStream.write(buffer, 0, bytes);
        }

        try {
            byteArrayOutputStream.close();
        }
        catch (IOException e) {
            throw new RuntimeException("Failed to close ByteArrayOutputStream", e);
        }

        // return encoded string
        byte[] compressedBytes = byteArrayOutputStream.toByteArray();
        return Base64.getEncoder().encodeToString(compressedBytes);
    }

    /**
     * Decompresses a valid String that has been compressed using the zlib compression
library.
     * Decodes bytes with Base64 decoding scheme.
     *
     * @param input the String to be decompressed
     * @return the decompressed String
     */
    public String decompress(String input)
    {
        byte[] inputBytes = Base64.getDecoder().decode((input));

        // create decompressor
        Inflater decompressor = new Inflater();
        decompressor.setInput(inputBytes, 0, inputBytes.length);

        // decompress bytes to output stream
        byte[] buffer = new byte[4096];
        ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream(inputBytes.length);
        try {
```

```
            while (!decompressor.finished()) {
                int bytes = decompressor.inflate(buffer);
                if (bytes == 0 && decompressor.needsInput()) {
                    throw new DataFormatException("Input is truncated");
                }
                byteArrayOutputStream.write(buffer, 0, bytes);
            }
        }
        catch (DataFormatException e) {
            throw new RuntimeException("Failed to decompress string", e);
        }

        try {
            byteArrayOutputStream.close();
        }
        catch (IOException e) {
            throw new RuntimeException("Failed to close ByteArrayOutputStream", e);
        }

        // return decoded string
        byte[] decompressedBytes = byteArrayOutputStream.toByteArray();
        return new String(decompressedBytes, StandardCharsets.UTF_8);
    }
}
```

JAR 파일 구축

`mvn clean install`을 실행하여 프로젝트를 빌드합니다. 성공적으로 빌드되면 JAR 파일이 *`artifactId`*`-`*`version`*`.jar`이라는 프로젝트의 `target` 폴더에 생성됩니다. 여기서 *`artifactId`*는 Maven 프로젝트에서 제공한 이름입니다(예: `my-athena-udfs`).

AWS Lambda에 JAR 배포

Lambda에 코드를 배포하는 두 가지 옵션이 있습니다.

- AWS Serverless Application Repository을 사용하여 배포(권장)
- JAR 파일에서 Lambda 함수 만들기

옵션 1: AWS Serverless Application Repository에 배포

AWS Serverless Application Repository에 JAR 파일을 배포할 때 애플리케이션의 아키텍처를 나타내는 AWS SAM 템플릿 YAML 파일을 만듭니다. 그런 다음 이 YAML 파일과, 애플리케이션의 아티팩트

를 업로드하여 AWS Serverless Application Repository에 공개하는 Amazon S3 버킷을 지정합니다. 아래 절차에서는 앞서 복제했던 Athena Query Federation SDK의 `athena-query-federation/tools` 디렉터리에 위치한 publish.sh 스크립트를 사용합니다.

자세한 내용과 요구 사항은 AWS Serverless Application Repository 개발자 안내서의 애플리케이션 게시, AWS Serverless Application Model 개발자 안내서의 AWS SAM 템플릿 개념 및 AWS SAM CLI를 사용하여 서버리스 애플리케이션 게시를 참조하세요.

다음 예제에서는 YAML 파일의 파라미터를 보여줍니다. YAML 파일에 유사한 파라미터를 추가하고 프로젝트 디렉터리에 저장합니다. 전체 예제는 GitHub에서 athena-udf.yaml을 참조하세요.

```
Transform: 'AWS::Serverless-2016-10-31'
Metadata:
  'AWS::ServerlessRepo::Application':
    Name: MyApplicationName
    Description: 'The description I write for my application'
    Author: 'Author Name'
    Labels:
      - athena-federation
    SemanticVersion: 1.0.0
Parameters:
  LambdaFunctionName:
    Description: 'The name of the Lambda function that will contain your UDFs.'
    Type: String
  LambdaTimeout:
    Description: 'Maximum Lambda invocation runtime in seconds. (min 1 - 900 max)'
    Default: 900
    Type: Number
  LambdaMemory:
    Description: 'Lambda memory in MB (min 128 - 3008 max).'
    Default: 3008
    Type: Number
Resources:
  ConnectorConfig:
    Type: 'AWS::Serverless::Function'
    Properties:
      FunctionName: !Ref LambdaFunctionName
      Handler: "full.path.to.your.handler. For example,
 com.amazonaws.athena.connectors.udfs.MyUDFHandler"
      CodeUri: "Relative path to your JAR file. For example, ./target/athena-
udfs-1.0.jar"
      Description: "My description of the UDFs that this Lambda function enables."
      Runtime: java8
```

```
      Timeout: !Ref LambdaTimeout
      MemorySize: !Ref LambdaMemory
```

YAML 파일을 저장한 프로젝트 디렉터리에 publish.sh 스크립트를 복사하고 다음 명령을 실행합니다.

```
./publish.sh MyS3Location MyYamlFile
```

예를 들어 버킷 위치가 s3://DOC-EXAMPLE-BUCKET/mysarapps/athenaudf이고 YAML 파일이 my-athena-udfs.yaml로 저장된 경우:

```
./publish.sh DOC-EXAMPLE-BUCKET/mysarapps/athenaudf my-athena-udfs
```

Lambda 함수를 생성하는 방법

1. https://console.aws.amazon.com/lambda/에서 Lambda 콘솔을 열고 함수 생성을 선택한 다음 서버리스 앱 리포지토리 찾아보기를 선택합니다.
2. 프라이빗 애플리케이션을 선택하거나 목록에서 애플리케이션을 찾거나 키워드를 사용하여 애플리케이션을 검색한 다음 선택합니다.
3. 애플리케이션 세부 정보를 검토하고 제공한 다음 배포를 선택합니다.

   이제 Lambda 함수 JAR 파일에 정의된 메서드 이름을 Athena의 UDF로 사용할 수 있습니다.

옵션 2: 직접 Lambda 함수 생성

콘솔 또는 AWS CLI를 사용하여 직접 Lambda 함수를 만들 수도 있습니다. 다음은 Lambda create-function CLI 명령을 사용하는 예제입니다.

```
aws lambda create-function \
 --function-name MyLambdaFunctionName \
 --runtime java8 \
 --role arn:aws:iam::1234567890123:role/my_lambda_role \
 --handler com.mycompany.athena.udfs.MyUserDefinedFunctions \
 --timeout 900 \
 --zip-file fileb://./target/my-athena-udfs-1.0-SNAPSHOT.jar
```

리전 간 쿼리

Athena는 Athena를 사용하는 리전과 다른 AWS 리전에서 Amazon S3 데이터를 쿼리하는 기능을 지원합니다. 데이터 이동이 실용적이지 않거나 허용되지 않는 경우, 또는 여러 리전에서 데이터를 쿼리하려는 경우 리전 간 쿼리는 옵션이 될 수 있습니다. 특정 리전에서 Athena를 사용할 수 없는 경우에도 해당 리전의 데이터를 Athena를 사용할 수 있는 다른 리전에서 쿼리할 수 있습니다.

리전의 데이터를 쿼리하려면 Amazon S3 데이터가 계정에 속하지 않더라도 해당 리전에서 계정을 활성화해야 합니다. 미국 동부(오하이오)와 같은 일부 리전의 경우 계정이 생성될 때 리전에 대한 액세스가 자동으로 활성화됩니다. 다른 리전에서는 계정을 사용하기 전에 리전에 계정을 "옵트인"해야 합니다. 옵트인이 필요한 리전의 목록은 Linux 인스턴스용 Amazon EC2 사용 설명서의 사용 가능한 리전을 참조하세요. 리전에 옵트인하는 방법에 대한 구체적인 지침은 Amazon Web Services 일반 참조의 Managing AWS regions를 참조하세요.

고려 사항 및 제한

- 데이터 액세스 권한 - Athena에서 리전 간 Amazon S3 데이터를 성공적으로 쿼리하려면 계정에 데이터를 읽을 수 있는 권한이 있어야 합니다. 쿼리하려는 데이터가 다른 계정에 속한 경우 다른 계정에서 해당 데이터가 포함된 Amazon S3 위치에 대한 액세스 권한을 부여해야 합니다.
- 데이터 전송 요금 - 리전 간 쿼리에 Amazon S3 데이터 전송 요금이 적용됩니다. 쿼리를 실행하면 데이터 세트 크기보다 더 많은 데이터가 전송될 수 있습니다. 먼저 데이터의 하위 세트를 대상으로 쿼리를 테스트하고 AWS Cost Explorer에서 비용을 검토하는 것이 좋습니다.
- AWS Glue - 리전 간에 AWS Glue를 사용할 수 있습니다. 리전 간 AWS Glue 트래픽에 추가 요금이 적용될 수 있습니다. 자세한 내용은 AWS 빅 데이터 블로그의 Create cross-account and cross-region AWS Glue connections를 참조하세요.
- Amazon S3 암호화 옵션 - SSE-S3 및 SSE-KMS 암호화 옵션은 리전 간 쿼리에 대해 지원되지만 CSE-KMS는 지원되지 않습니다. 자세한 내용은 지원되는 Amazon S3 암호화 옵션 단원을 참조하십시오.
- 페더레이션된 쿼리 - AWS 리전에서 페더레이션된 쿼리 사용은 지원되지 않습니다.
- 중국 리전-리전 간 쿼리는 중국 리전에서 지원되지 않습니다.

위 조건이 충족되면 지정한 `LOCATION` 값을 가리키는 Athena 테이블을 생성하고 데이터를 투명하게 쿼리할 수 있습니다. 특별한 구문은 필요하지 않습니다. Athena 테이블 생성에 대한 자세한 내용은 Athena에서 테이블 생성 섹션을 참조하세요.

AWS Glue Data Catalog 쿼리

AWS Glue Data Catalog는 많은 AWS 서비스에서 중앙 메타데이터 리포지토리로 사용되므로 데이터 카탈로그 메타데이터를 쿼리해야 할 수 있습니다. 이를 위해 Athena에서 SQL 쿼리를 사용할 수 있습니다. Athena를 사용하여 데이터베이스, 테이블, 파티션 및 열과 같은 AWS Glue 카탈로그 메타데이터를 쿼리할 수 있습니다.

AWS Glue 카탈로그 메타데이터를 가져오려면 Athena 백엔드에서 `information_schema` 데이터베이스를 쿼리합니다. 이 주제의 예제 쿼리에서는 Athena를 사용하여 일반적인 사용 사례에 대한 AWS Glue 카탈로그 메타데이터를 쿼리하는 방법을 보여줍니다.

주제

고려 사항 및 제한

- `information_schema` 데이터베이스를 쿼리하는 대신, 개별 Apache Hive DDL 명령를 사용하여 Athena에서 특정 데이터베이스, 테이블, 뷰, 파티션 및 열에 대한 메타데이터 정보를 추출할 수 있습니다. 그러나 출력은 표가 아닌 형식을 가집니다.
- AWS Glue 메타데이터의 양이 소량에서 중간 정도라면 `information_schema` 쿼리가 가장 성능이 뛰어납니다. 메타데이터의 양이 많으면 오류가 발생할 수 있습니다.
- `CREATE VIEW`를 사용하여 `information_schema` 데이터베이스에 뷰를 생성할 수 없습니다.

데이터베이스 나열 및 지정된 데이터베이스 검색

이 단원의 예제는 스키마 이름별로 메타데이터에서 데이터베이스를 나열하는 방법을 보여줍니다.

Example - 데이터베이스 나열

다음 예제 쿼리는 `information_schema.schemata` 테이블의 데이터베이스를 나열합니다.

```
SELECT schema_name
FROM   information_schema.schemata
LIMIT  10;
```

다음 표는 샘플 결과를 보여 줍니다.

6	alb-databas1
7	alb_original_cust
8	alblogsdatabase
9	athena_db_test
10	athena_ddl_db

Example - 지정된 데이터베이스 검색

다음 예제 쿼리에서 `rdspostgresql`은 샘플 데이터베이스입니다.

```
SELECT schema_name
FROM   information_schema.schemata
WHERE  schema_name = 'rdspostgresql'
```

다음 표는 샘플 결과를 보여 줍니다.

	schema_name
1	rdspostgresql

지정된 데이터베이스의 테이블 나열 및 이름별로 테이블 검색

테이블에 대한 메타데이터를 나열하려면 테이블 스키마별로 또는 테이블 이름별로 쿼리할 수 있습니다.

Example - 스키마별로 테이블 나열

다음 쿼리는 `rdspostgresql` 테이블 스키마를 사용하는 테이블을 나열합니다.

```
SELECT table_schema,
       table_name,
       table_type
FROM   information_schema.tables
WHERE  table_schema = 'rdspostgresql'
```

다음 표는 샘플 결과를 보여줍니다.

	table_schema	table_name	table_type
1	rdspostgresql	rdspostgresqldb1_public_account	BASE TABLE

Example - 이름별로 테이블 검색

다음 쿼리는 athena1 테이블에 대한 메타데이터 정보를 가져옵니다.

```
SELECT table_schema,
       table_name,
       table_type
FROM   information_schema.tables
WHERE  table_name = 'athena1'
```

다음 표는 샘플 결과를 보여줍니다.

	table_schema	table_name	table_type
1	기본값	athena1	BASE TABLE

특정 테이블에 대한 파티션 나열

SHOW PARTITIONS *table_name*을(를) 사용하여 다음 예제처럼 지정된 테이블에 대한 파티션을 나열할 수 있습니다.

```
SHOW PARTITIONS cloudtrail_logs_test2
```

$partitions메타데이터 쿼리를 사용하여 특정 테이블의 파티션 번호와 파티션 값을 나열할 수도 있습니다.

Example – $partitions 구문을 사용하여 테이블에 대한 파티션 쿼리

다음 예제 쿼리는 $partitions 구문을 사용하여 cloudtrail_logs_test2 테이블에 대한 파티션을 나열합니다.

```
SELECT * FROM default."cloudtrail_logs_test2$partitions" ORDER BY partition_number
```

다음 표는 샘플 결과를 보여 줍니다.

	table_catalog	table_schema	table_name	연도	월	일
1	awsdatacatalog	default	cloudtrail_logs_test2	2020	08	10
2	awsdatacatalog	default	cloudtrail_logs_test2	2020	08	11
3	awsdatacatalog	default	cloudtrail_logs_test2	2020	08	12

모든 테이블에 대한 모든 열 나열

AwsDataCatalog의 모든 테이블이나 AwsDataCatalog의 특정 데이터베이스에 있는 모든 테이블의 모든 열을 나열할 수 있습니다.

- AwsDataCatalog의 모든 데이터베이스에 대한 모든 열을 나열하려면 SELECT * FROM information_schema.columns 쿼리를 사용합니다.
- 결과를 특정 데이터베이스로 제한하려면 WHERE 절에 table_schema='*database_name*'을 사용합니다.

Example - 특정 데이터베이스의 모든 테이블에 대한 모든 열 나열

다음 예제 쿼리는 데이터베이스 webdata의 모든 테이블에 대한 모든 열을 나열합니다.

```
SELECT * FROM information_schema.columns WHERE table_schema = 'webdata'
```

특정 테이블에서 공통적으로 포함하는 열 나열

데이터베이스의 특정 테이블에서 공통적으로 포함하는 열을 나열할 수 있습니다.

- `SELECT column_name FROM information_schema.columns` 구문을 사용합니다.
- `WHERE` 절의 경우 `WHERE table_name IN ('table1', 'table2')` 구문을 사용합니다.

Example - 동일한 데이터베이스에 있는 두 테이블의 공통 열 나열

다음 예제 쿼리는 `table1` 및 `table2` 테이블에서 공통적으로 포함하는 열을 나열합니다.

```
SELECT column_name
FROM information_schema.columns
WHERE table_name IN ('table1', 'table2')
GROUP BY column_name
HAVING COUNT(*) > 1;
```

지정된 테이블 또는 뷰에 대한 열 나열 또는 검색

테이블에 대한 모든 열, 뷰에 대한 모든 열을 나열하거나 지정된 데이터베이스와 테이블에서 이름별로 열을 검색할 수 있습니다.

열을 나열하려면 `SELECT * ` 쿼리를 사용합니다. `FROM` 절에서 `information_schema.columns`를 지정합니다. `WHERE` 절에서 `table_schema='`*`database_name`*`'`을(를) 사용해 데이터베이스를 지정하고 `table_name = '`*`table_name`*`'`을(를) 사용해 나열하려는 열이 있는 테이블 또는 뷰를 지정합니다.

Example - 지정된 테이블에 대한 모든 열 나열

다음 예제 쿼리는 `rdspostgresqldb1_public_account` 테이블에 대한 모든 열을 나열합니다.

```
SELECT *
FROM   information_schema.columns
WHERE  table_schema = 'rdspostgresql'
       AND table_name = 'rdspostgresqldb1_public_account'
```

다음 표는 샘플 결과를 보여 줍니다.

	table_catalog	table_schema	table_name	column_name	ordinal_position	column_default	is_nullable	data_t	comm	extra_info
1	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	비밀번호	1		예	varcha		
2	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	user_id	2		예	정수		
3	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	created_on	3		예	timest		
4	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	last_login	4		예	timest		
5	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	email	5		예	varcha		
6	awsdatacatalog	rdspostgresql	rdspostgresqldb1_public_account	username	6		예	varcha		

Example - 지정된 뷰에 대한 열 나열

다음 예제 쿼리는 `arrayview` 뷰에 대한 `default` 데이터베이스의 모든 열을 나열합니다.

```
SELECT *
```

```
FROM   information_schema.columns
WHERE  table_schema = 'default'
       AND table_name = 'arrayview'
```

다음 표는 샘플 결과를 보여 줍니다.

	table_catalog	table_schema	table_name	column_name	ordinal_position	column_default	is_nullable	data_type	comment	extra_info
1	awsdatacatalog	default	arrayview	searchdate	1		예	varchar		
2	awsdatacatalog	default	arrayview	sid	2		예	varchar		
3	awsdatacatalog	default	arrayview	btid	3		예	varchar		
4	awsdatacatalog	default	arrayview	p	4		예	varchar		
5	awsdatacatalog	default	arrayview	infantprice	5		예	varchar		
6	awsdatacatalog	default	arrayview	sump	6		예	varchar		
7	awsdatacatalog	default	arrayview	journeymaparray	7		예	array(varchar)		

Example - 지정된 데이터베이스 및 테이블에서 이름으로 열 검색

다음 예제 쿼리는 `default` 데이터베이스의 `arrayview` 뷰에서 `sid` 열에 대한 메타데이터를 검색합니다.

```
SELECT *
FROM   information_schema.columns
WHERE  table_schema = 'default'
       AND table_name = 'arrayview'
```

```
        AND column_name='sid'
```

다음 표는 샘플 결과를 보여줍니다.

	table_catalog	table_schema	table_name	column_name	ordinal_position	column_default	is_nullable	data_t	comm	extra_info
1	awsdatacatalog	default	arrayview	sid	2		예	varcha		

AWS 서비스 로그 쿼리

이 단원에는 Amazon Athena를 사용하여 AWS CloudTrail 로그, Amazon CloudFront 로그, Classic Load Balancer 로그, Application Load Balancer 로그, Amazon VPC 흐름 로그, Network Load Balancer 로그 등, 널리 사용되는 데이터 세트를 쿼리하는 다양한 절차가 나와 있습니다.

이 섹션의 작업은 Athena 콘솔을 사용하지만 Athena JDBC 드라이버, AWS CLI 또는 Amazon Athena API Reference 등의 다른 도구를 사용할 수도 있습니다.

Athena에서 AWS CloudFormation를 사용하여 AWS 서비스 로그 테이블, 파티션, 예제 쿼리를 자동으로 생성하는 방법에 관한 자세한 내용은 AWS Big Data Blog에서 Automating AWS 서비스 logs table creation and querying them with Amazon Athena를 참조하세요. Athena에서 AWS Glue용 Python 라이브러리를 사용해 AWS 서비스 로그를 처리하고 쿼리하기 위한 일반 프레임워크를 생성하는 방법에 대한 자세한 내용은 Amazon Athena를 사용하여 간단히 AWS 서비스 로그 쿼리를 참조하세요.

이 단원의 주제에서는 Athena에 액세스할 수 있는 적절한 권한과 쿼리할 데이터가 상주해야 하는 Amazon S3 버킷을 모두 설정했다고 가정합니다. 자세한 내용은 설정 및 시작하기 단원을 참조하세요.

주제

- Application Load Balancer 로그 쿼리
- Classic Load Balancer 로그 쿼리
- Amazon CloudFront 로그 쿼리
- AWS CloudTrail 로그 쿼리
- Amazon EMR 로그 쿼리
- AWS Global Accelerator 흐름 로그 쿼리

- Amazon GuardDuty 결과 쿼리
- AWS Network Firewall 로그 쿼리
- Network Load Balancer 로그 쿼리
- Amazon Route 53 Resolver 쿼리 로그의 쿼리
- Amazon SES 이벤트 로그 쿼리
- Amazon VPC 흐름 로그 쿼리
- AWS WAF 로그 쿼리

Application Load Balancer 로그 쿼리

Application Load Balancer는 컨테이너를 사용하여 마이크로서비스 분포에서 트래픽 분배를 가능하게 하는, Elastic Load Balancing을 위한 로드 밸런싱 옵션입니다. Application Load Balancer 로그를 쿼리하면 Elastic Load Balancing 인스턴스 및 백엔드 애플리케이션과 주고 받는 트래픽, 지연 시간 및 바이트의 소스를 볼 수 있습니다. 자세한 내용은 Application Load Balancers 사용 설명서의 Application Load Balancer에 대한 액세스 로그 및 Application Load Balancer에 대한 연결 로그를 참조하세요.

주제

- 필수 조건
- ALB 액세스 로그의 테이블 생성
- 파티션 프로젝션을 사용하여 Athena에서 ALB 액세스 로그의 테이블 생성
- ALB 액세스 로그의 쿼리 예제
- ALB 연결 로그의 테이블 생성
- 파티션 프로젝션을 사용하여 Athena에서 ALB 연결 로그의 테이블 생성
- ALB 연결 로그의 쿼리 예제
- 추가적인 리소스

필수 조건

- Application Load Balancer 로그를 Amazon S3 버킷에 저장할 수 있도록 액세스 로깅 또는 연결 로깅을 활성화합니다.
- Athena용으로 만들 테이블을 보관할 데이터베이스입니다. 데이터베이스를 만들려면 Athena 또는 AWS Glue 콘솔을 사용하세요. 자세한 내용은 이 가이드의 Athena에서 데이터베이스 생성 또는 AWS Glue 개발자 안내서의 AWS glue 콘솔에서 데이터베이스 관련 작업을 참조하세요.

ALB 액세스 로그의 테이블 생성

1. 다음 CREATE TABLE 명령문을 복사하여 Athena 콘솔의 쿼리 편집기에 붙여 넣습니다. Athena 콘솔 시작하기에 대한 자세한 내용은 시작하기 섹션을 참조하세요. LOCATION 절의 경로를 Amazon S3 액세스 로그 폴더의 위치로 바꿉니다. 액세스 로그 파일 위치에 대한 자세한 내용은 Application Load Balancer 사용 설명서의 액세스 로그 파일을 참조하세요. 각 로그 파일 필드에 대한 자세한 내용은 액세스 로그 항목을 참조하세요.

> **Note**
>
> 다음 CREATE TABLE 문에는 최근에 추가된 classification, classification_reason 및 traceability_id 열이 포함됩니다. 이러한 항목을 포함하지 않은 Application Load Balancer 액세스 로그에 대한 테이블을 생성하려면 CREATE TABLE 문에서 해당되는 열을 제거하고 그에 따라 정규 표현식을 수정합니다.

```
CREATE EXTERNAL TABLE IF NOT EXISTS alb_access_logs (
             type string,
             time string,
             elb string,
             client_ip string,
             client_port int,
             target_ip string,
             target_port int,
             request_processing_time double,
             target_processing_time double,
             response_processing_time double,
             elb_status_code int,
             target_status_code string,
             received_bytes bigint,
             sent_bytes bigint,
             request_verb string,
             request_url string,
             request_proto string,
             user_agent string,
             ssl_cipher string,
             ssl_protocol string,
             target_group_arn string,
             trace_id string,
             domain_name string,
             chosen_cert_arn string,
```

```
            matched_rule_priority string,
            request_creation_time string,
            actions_executed string,
            redirect_url string,
            lambda_error_reason string,
            target_port_list string,
            target_status_code_list string,
            classification string,
            classification_reason string,
            traceability_id string
            )
            ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
            WITH SERDEPROPERTIES (
            'serialization.format' = '1',
            'input.regex' =
        '([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*):([0-9]*) ([^ ]*)[:-]([0-9]*) ([-.0-9]*)
 ([-.0-9]*) ([-.0-9]*) (|[-0-9]*) (-|[-0-9]*) ([-0-9]*) ([-0-9]*) \"([^ ]*) (.*) (-
 |[^ ]*)\" \"([^\"]*)\" ([A-Z0-9-_]+) ([A-Za-z0-9.-]*) ([^ ]*) \"([^\"]*)\" \"([^
\"]*)\" \"([^\"]*)\" ([-.0-9]*) ([^ ]*) \"([^\"]*)\" \"([^\"]*)\" \"([^ ]*)\" \"([^
\s]+?)\" \"([^\s]+)\" \"([^ ]*)\" \"([^ ]*)\" ?([^ ]*)?( .*)?')
            LOCATION 's3://DOC-EXAMPLE-BUCKET/access-log-folder-path/'
```

2. Athena 콘솔에서 쿼리를 실행합니다. 쿼리가 완료된 후 Athena는 `alb_access_logs` 테이블을 등록하여 쿼리를 실행할 수 있도록 데이터를 준비합니다.

파티션 프로젝션을 사용하여 Athena에서 ALB 액세스 로그의 테이블 생성

ALB 액세스 로그에는 미리 지정할 수 있는 파티션 스키마를 가진 알려진 구조가 있기 때문에 Athena 파티션 프로젝션 기능을 사용하여 쿼리 런타임을 줄이고 파티션 관리를 자동화할 수 있습니다. 새 데이터가 추가되면 파티션 프로젝션은 자동으로 새 파티션을 추가합니다. 따라서 `ALTER TABLE ADD PARTITION`을 사용해 파티션을 수동으로 추가할 필요가 없습니다.

다음 `CREATE TABLE` 문 예제에서는 하나의 AWS 리전에 대해 지정된 날짜부터 현재까지의 ALB 액세스 로그에 파티션 프로젝션을 자동으로 사용합니다. 이 문은 이전 섹션의 예제를 기반으로 하지만 파티션 프로젝션을 사용하기 위해 `PARTITIONED BY`과 `TBLPROPERTIES` 절을 추가합니다. `LOCATION` 및 `storage.location.template` 절에서 자리 표시자를 ALB 액세스 로그의 Amazon S3 버킷 위치를 식별하는 값으로 바꿉니다. 액세스 로그 파일 위치에 대한 자세한 내용은 Application Load Balancer 사용 설명서의 액세스 로그 파일을 참조하세요. `projection.day.range`에 대해 *2022/01/01*을 사용하려는 시작 날짜로 바꿉니다. 쿼리가 성공적으로 실행된면 테이블을 쿼리할 수 있습니다. 파티션을 로드하기 위해 `ALTER TABLE ADD PARTITION`을 실행하지 않아도 됩니다. 각 로그 파일 필드에 대한 자세한 내용은 액세스 로그 항목을 참조하세요.

```
CREATE EXTERNAL TABLE IF NOT EXISTS alb_access_logs (
             type string,
             time string,
             elb string,
             client_ip string,
             client_port int,
             target_ip string,
             target_port int,
             request_processing_time double,
             target_processing_time double,
             response_processing_time double,
             elb_status_code int,
             target_status_code string,
             received_bytes bigint,
             sent_bytes bigint,
             request_verb string,
             request_url string,
             request_proto string,
             user_agent string,
             ssl_cipher string,
             ssl_protocol string,
             target_group_arn string,
             trace_id string,
             domain_name string,
             chosen_cert_arn string,
             matched_rule_priority string,
             request_creation_time string,
             actions_executed string,
             redirect_url string,
             lambda_error_reason string,
             target_port_list string,
             target_status_code_list string,
             classification string,
             classification_reason string,
             traceability_id string
             )
             PARTITIONED BY
             (
              day STRING
             )
             ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
             WITH SERDEPROPERTIES (
             'serialization.format' = '1',
```

```
            'input.regex' =
        '([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*):([0-9]*) ([^ ]*)[:-]([0-9]*) ([-.0-9]*)
 ([-.0-9]*) ([-.0-9]*) (|[-0-9]*) (-|[-0-9]*) ([-0-9]*) ([-0-9]*) \"([^ ]*) (.*) (- |
[^ ]*)\" \"([^\"]*)\" ([A-Z0-9-_]+) ([A-Za-z0-9.-]*) ([^ ]*) \"([^\"]*)\" \"([^\"]*)\"
 \"([^\"]*)\" ([-.0-9]*) ([^ ]*) \"([^\"]*)\" \"([^\"]*)\" \"([^ ]*)\" \"([^\s]+?)\"
 \"([^\s]+)\" \"([^ ]*)\" \"([^ ]*)\" ?([^ ]*)?( .*)?')
            LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/<ACCOUNT-NUMBER>/
elasticloadbalancing/<REGION>/'
            TBLPROPERTIES
            (
             "projection.enabled" = "true",
             "projection.day.type" = "date",
             "projection.day.range" = "2022/01/01,NOW",
             "projection.day.format" = "yyyy/MM/dd",
             "projection.day.interval" = "1",
             "projection.day.interval.unit" = "DAYS",
             "storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/AWSLogs/<ACCOUNT-
NUMBER>/elasticloadbalancing/<REGION>/${day}"
            )
```

파티션 프로젝션에 대한 자세한 내용은 Amazon Athena를 사용한 파티션 프로젝션 단원을 참조하세요.

ALB 액세스 로그의 쿼리 예제

다음 쿼리는 클라이언트 IP 주소별로 그룹화된 로드 밸런서가 수신한 HTTP GET 요청의 개수를 계산합니다:

```
SELECT COUNT(request_verb) AS
 count,
 request_verb,
 client_ip
FROM alb_logs
GROUP BY request_verb, client_ip
LIMIT 100;
```

다른 쿼리는 Safari 브라우저 사용자가 방문한 URL을 보여줍니다:

```
SELECT request_url
FROM alb_logs
WHERE user_agent LIKE '%Safari%'
LIMIT 10;
```

다음 쿼리는 ELB 상태 코드 값이 500보다 크거나 같은 레코드를 보여 줍니다.

```
SELECT * FROM alb_logs
WHERE elb_status_code >= 500
```

다음 예에서는 datetime을 통해 로그를 구문 분석하는 방법을 보여 줍니다.

```
SELECT client_ip, sum(received_bytes)
FROM alb_logs
WHERE parse_datetime(time,'yyyy-MM-dd''T''HH:mm:ss.SSSSSS''Z')
     BETWEEN parse_datetime('2018-05-30-12:00:00','yyyy-MM-dd-HH:mm:ss')
     AND parse_datetime('2018-05-31-00:00:00','yyyy-MM-dd-HH:mm:ss')
GROUP BY client_ip;
```

다음 쿼리는 지정된 날짜의 모든 ALB 로그에 대해 파티션 프로젝션을 사용하는 테이블을 쿼리합니다.

```
SELECT *
FROM alb_logs
WHERE day = '2022/02/12'
```

ALB 연결 로그의 테이블 생성

1. 다음 CREATE TABLE 명령문을 복사하여 Athena 콘솔의 쿼리 편집기에 붙여 넣습니다. Athena 콘솔 시작하기에 대한 자세한 내용은 시작하기 섹션을 참조하세요. LOCATION 절의 경로를 Amazon S3 연결 로그 폴더의 위치로 바꿉니다. 연결 로그 파일 위치에 대한 자세한 내용은 Application Load Balancer 사용 설명서의 연결 로그 파일을 참조하세요. 각 로그 파일 필드에 대한 자세한 내용은 연결 로그 항목을 참조하세요.

   ```
   CREATE EXTERNAL TABLE IF NOT EXISTS alb_connection_logs (
            time string,
            client_ip string,
            client_port int,
            listener_port int,
            tls_protocol string,
            tls_cipher string,
            tls_handshake_latency double,
            leaf_client_cert_subject string,
            leaf_client_cert_validity string,
            leaf_client_cert_serial_number string,
            tls_verify_status string,
            traceability_id string
   ```

```
        )
        ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
        WITH SERDEPROPERTIES (
        'serialization.format' = '1',
        'input.regex' =
         '([^ ]*) ([^ ]*) ([0-9]*) ([0-9]*) ([A-Za-z0-9.-]*) ([^ ]*) ([-.0-9]*)
\"([^\"]*)\" ([^ ]*) ([^ ]*) ([^ ]*) ?([^ ]*)?( .*)?')
        LOCATION 's3://DOC-EXAMPLE-BUCKET/connection-log-folder-path/'
```

2. Athena 콘솔에서 쿼리를 실행합니다. 쿼리가 완료된 후 Athena는 `alb_connection_logs` 테이블을 등록하여 쿼리를 실행할 수 있도록 데이터를 준비합니다.

파티션 프로젝션을 사용하여 Athena에서 ALB 연결 로그의 테이블 생성

ALB 연결 로그에는 미리 지정할 수 있는 파티션 스키마를 가진 알려진 구조가 있기 때문에 Athena 파티션 프로젝션 기능을 사용하여 쿼리 런타임을 줄이고 파티션 관리를 자동화할 수 있습니다. 새 데이터가 추가되면 파티션 프로젝션은 자동으로 새 파티션을 추가합니다. 따라서 `ALTER TABLE ADD PARTITION`을 사용해 파티션을 수동으로 추가할 필요가 없습니다.

다음 `CREATE TABLE` 문 예제에서는 하나의 AWS 리전에 대해 지정된 날짜부터 현재까지의 ALB 연결 로그에 파티션 프로젝션을 자동으로 사용합니다. 이 문은 이전 섹션의 예제를 기반으로 하지만 파티션 프로젝션을 사용하기 위해 `PARTITIONED BY`과 `TBLPROPERTIES` 절을 추가합니다. `LOCATION` 및 `storage.location.template` 절에서 자리 표시자를 ALB 연결 로그의 Amazon S3 버킷 위치를 식별하는 값으로 바꿉니다. 연결 로그 파일 위치에 대한 자세한 내용은 Application Load Balancer 사용 설명서의 연결 로그 파일을 참조하세요. `projection.day.range`에 대해 *2023*/*01*/*01*을 사용하려는 시작 날짜로 바꿉니다. 쿼리가 성공적으로 실행된면 테이블을 쿼리할 수 있습니다. 파티션을 로드하기 위해 `ALTER TABLE ADD PARTITION`을 실행하지 않아도 됩니다. 각 로그 파일 필드에 대한 자세한 내용은 연결 로그 항목을 참조하세요.

```
CREATE EXTERNAL TABLE IF NOT EXISTS alb_connection_logs (
         time string,
         client_ip string,
         client_port int,
         listener_port int,
         tls_protocol string,
         tls_cipher string,
         tls_handshake_latency double,
         leaf_client_cert_subject string,
         leaf_client_cert_validity string,
         leaf_client_cert_serial_number string,
         tls_verify_status string,
```

```
         traceability_id string
         )
            PARTITIONED BY
            (
             day STRING
            )
            ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
            WITH SERDEPROPERTIES (
            'serialization.format' = '1',
            'input.regex' =
             '([^ ]*) ([^ ]*) ([0-9]*) ([0-9]*) ([A-Za-z0-9.-]*) ([^ ]*) ([-.0-9]*)
 \"([^\"]*)\" ([^ ]*) ([^ ]*) ([^ ]*) ?([^ ]*)?( .*)?')
            LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/<ACCOUNT-NUMBER>/
elasticloadbalancing/<REGION>/'
            TBLPROPERTIES
            (
             "projection.enabled" = "true",
             "projection.day.type" = "date",
             "projection.day.range" = "2023/01/01,NOW",
             "projection.day.format" = "yyyy/MM/dd",
             "projection.day.interval" = "1",
             "projection.day.interval.unit" = "DAYS",
             "storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/AWSLogs/<ACCOUNT-
NUMBER>/elasticloadbalancing/<REGION>/${day}"
            )
```

파티션 프로젝션에 대한 자세한 내용은 Amazon Athena를 사용한 파티션 프로젝션 단원을 참조하세요.

ALB 연결 로그의 쿼리 예제

다음 쿼리는 `tls_verify_status` 값이 `'Success'`가 아닌 발생 항목 수를 세고, 클라이언트 IP 주소별로 그룹화합니다.

```
SELECT DISTINCT client_ip, count() AS count FROM alb_connection_logs
WHERE tls_verify_status != 'Success'
GROUP BY client_ip
ORDER BY count() DESC;
```

다음 쿼리는 지정된 시간 범위에서 `tls_handshake_latency` 값이 2초를 초과하는 발생 항목을 검색합니다.

```
SELECT * FROM alb_connection_logs
```

```
WHERE
  (
    parse_datetime(time, 'yyyy-MM-dd''T''HH:mm:ss.SSSSSS''Z')
    BETWEEN
    parse_datetime('2024-01-01-00:00:00', 'yyyy-MM-dd-HH:mm:ss')
    AND
    parse_datetime('2024-03-20-00:00:00', 'yyyy-MM-dd-HH:mm:ss')
  )
  AND
    (tls_handshake_latency >= 2.0);
```

추가적인 리소스

- AWS 지식 센터의 Amazon Athena를 사용하여 Application Load Balancer 액세스 로그를 분석하려면 어떻게 해야 합니까?를 참조하세요.
- Elastic Load Balancing에서 HTTP 상태 코드에 대한 자세한 내용은 Application Load Balancer 사용 설명서의 Application Load Balancer 문제 해결을 참조하세요.
- AWS 빅 데이터 블로그의 Catalog and analyze Application Load Balancer logs more efficiently with AWS Glue custom classifiers and Amazon Athena를 참조하세요.

Classic Load Balancer 로그 쿼리

Classic Load Balancer 로그를 사용하여 Elastic Load Balancing 인스턴스 및 백엔드 애플리케이션과의 송수신 트래픽 패턴을 분석하고 이해합니다. 트래픽의 소스, 지연 시간 및 전송된 바이트 수를 확인할 수 있습니다.

Elastic Load Balancing 로그를 분석하기 전에 대상 Amazon S3 버킷에 저장하기 위한 로그를 구성합니다. 자세한 내용은 Classic Load Balancer 액세스 로그 사용을 참조하세요.

- Elastic Load Balancing 로그에 대한 테이블 생성
- Elastic Load Balancing 예제 쿼리

Elastic Load Balancing 로그에 대한 테이블을 생성하려면

1. 다음 DDL 문을 복사하여 Athena 콘솔에 붙여 넣습니다. Elastic Load Balancing 로그 레코드의 구문을 확인합니다. 최신 버전 레코드의 열과 Regex 구문을 포함하도록 다음 쿼리를 업데이트해야 할 수 있습니다.

   ```
   CREATE EXTERNAL TABLE IF NOT EXISTS elb_logs (
   ```

```
 timestamp string,
 elb_name string,
 request_ip string,
 request_port int,
 backend_ip string,
 backend_port int,
 request_processing_time double,
 backend_processing_time double,
 client_response_time double,
 elb_response_code string,
 backend_response_code string,
 received_bytes bigint,
 sent_bytes bigint,
 request_verb string,
 url string,
 protocol string,
 user_agent string,
 ssl_cipher string,
 ssl_protocol string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
 'serialization.format' = '1',
 'input.regex' = '([^ ]*) ([^ ]*) ([^ ]*):([0-9]*) ([^ ]*)[:-]([0-9]*) ([-.0-9]*)
 ([-.0-9]*) ([-.0-9]*) (|[-0-9]*) (-|[-0-9]*) ([-0-9]*) ([-0-9]*) \\\"([^ ]*)
 ([^ ]*) (- |[^ ]*)\\\" (\"[^\"]*\") ([A-Z0-9-]+) ([A-Za-z0-9.-]*)$'
)
LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/AWS_account_ID/elasticloadbalancing/';
```

2. Elastic Load Balancing 로그의 대상을 지정하여 `LOCATION` Amazon S3 버킷을 수정합니다.
3. Athena 콘솔에서 쿼리를 실행합니다. 쿼리가 완료된 후 Athena는 `elb_logs` 테이블을 등록하여 쿼리를 위한 데이터를 준비합니다. 자세한 내용은 Elastic Load Balancing 예제 쿼리 단원을 참조하십시오.

Elastic Load Balancing 예제 쿼리

다음과 비슷한 방식으로 쿼리를 사용합니다. `4XX` 또는 `5XX` 오류 응답 코드를 반환한 백엔드 애플리케이션 서버가 나열됩니다. `LIMIT` 연산자를 사용하여 한 번에 쿼리할 로그 수를 제한합니다.

```
SELECT
 timestamp,
```

```
 elb_name,
 backend_ip,
 backend_response_code
FROM elb_logs
WHERE backend_response_code LIKE '4%' OR
      backend_response_code LIKE '5%'
LIMIT 100;
```

후속 쿼리를 사용하여 백엔드 IP 주소 및 Elastic Load Balancing 인스턴스 이름별로 그룹화된 모든 트랜잭션의 응답 시간을 요약합니다.

```
SELECT sum(backend_processing_time) AS
 total_ms,
 elb_name,
 backend_ip
FROM elb_logs WHERE backend_ip <> ''
GROUP BY backend_ip, elb_name
LIMIT 100;
```

자세한 내용은 Athena를 이용한 S3 데이터 분석을 참조하세요.

Amazon CloudFront 로그 쿼리

Amazon CloudFront CDN을 구성하여 웹 배포 액세스 로그를 Amazon Simple Storage Service로 내보낼 수 있습니다. 이러한 로그를 사용하여 CloudFront가 제공하는 웹 속성 전반에 걸쳐 사용자의 서핑 패턴을 탐색합니다.

로그 쿼리를 시작하기 전에 기본 CloudFront 배포에 따라 웹 배포에서 로그에 액세스할 수 있도록 설정합니다. 자세한 내용은 Amazon CloudFront 개발자 안내서의 액세스 로그를 참조하세요. 이 로그를 저장할 Amazon S3 버킷을 기록해 둡니다.

- CloudFront 표준 로그의 테이블 생성
- CloudFront 실시간 로그의 테이블 생성
- 표준 CloudFront 로그의 쿼리 예제

CloudFront 표준 로그의 테이블 생성

> **Note**
>
> 이 절차는 CloudFront의 웹 배포 액세스 로그에 적용됩니다. RTMP 배포의 스트리밍 로그에는 적용되지 않습니다.

CloudFront 표준 로그 파일 필드의 테이블 생성

1. 다음 예제 DDL 문을 복사하여 Athena 콘솔의 쿼리 편집기에 붙여 넣습니다. 예제 명령문은 Amazon CloudFront 개발자 안내서의 표준 로그 파일 필드 섹션에 설명된 로그 파일 필드를 사용합니다. 로그를 저장하는 Amazon S3 버킷의 `LOCATION`을 수정합니다. 쿼리 편집기 사용에 대한 자세한 내용은 시작하기 단원을 참조하세요.

   이 쿼리는 `ROW FORMAT DELIMITED` 및 `FIELDS TERMINATED BY '\t'`를 지정하여 필드가 탭 문자로 구분됨을 나타냅니다. `ROW FORMAT DELIMITED`의 경우 Athena는 기본적으로 LazySimpleSerDe를 사용합니다. `date` 열은 Athena에서 예약어이기 때문에 백틱(`) 키를 사용하여 이스케이프됩니다. 자세한 설명은 예약어을 참조하세요.

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_standard_logs (
  `date` DATE,
  time STRING,
  x_edge_location STRING,
  sc_bytes BIGINT,
  c_ip STRING,
  cs_method STRING,
  cs_host STRING,
  cs_uri_stem STRING,
  sc_status INT,
  cs_referrer STRING,
  cs_user_agent STRING,
  cs_uri_query STRING,
  cs_cookie STRING,
  x_edge_result_type STRING,
  x_edge_request_id STRING,
  x_host_header STRING,
  cs_protocol STRING,
  cs_bytes BIGINT,
  time_taken FLOAT,
  x_forwarded_for STRING,
```

```
  ssl_protocol STRING,
  ssl_cipher STRING,
  x_edge_response_result_type STRING,
  cs_protocol_version STRING,
  fle_status STRING,
  fle_encrypted_fields INT,
  c_port INT,
  time_to_first_byte FLOAT,
  x_edge_detailed_result_type STRING,
  sc_content_type STRING,
  sc_content_len BIGINT,
  sc_range_start BIGINT,
  sc_range_end BIGINT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION 's3://DOC-EXAMPLE-BUCKET/'
TBLPROPERTIES ( 'skip.header.line.count'='2' )
```

2. Athena 콘솔에서 쿼리를 실행합니다. 쿼리가 완료된 후 Athena는 `cloudfront_standard_logs` 테이블을 등록하여 쿼리를 실행할 수 있도록 데이터를 준비합니다.

CloudFront 실시간 로그의 테이블 생성

CloudFront 실시간 로그 파일 필드의 테이블 생성

1. 다음 예제 DDL 문을 복사하여 Athena 콘솔의 쿼리 편집기에 붙여 넣습니다. 예제 문에서는 Amazon CloudFront 개발자 안내서의 실시간 로그 섹션에 설명된 로그 파일 필드를 사용합니다. 로그를 저장하는 Amazon S3 버킷의 `LOCATION`을 수정합니다. 쿼리 편집기 사용에 대한 자세한 내용은 시작하기 단원을 참조하세요.

   이 쿼리는 `ROW FORMAT DELIMITED` 및 `FIELDS TERMINATED BY '\t'`를 지정하여 필드가 탭 문자로 구분됨을 나타냅니다. `ROW FORMAT DELIMITED`의 경우 Athena는 기본적으로 LazySimpleSerDe를 사용합니다. `timestamp` 열은 Athena에서 예약어이기 때문에 백틱(`) 키를 사용하여 이스케이프됩니다. 자세한 설명은 예약어을 참조하세요.

   다음 예제에는 사용 가능한 모든 필드가 포함되어 있습니다. 필요하지 않은 필드를 주석 처리하거나 제거할 수 있습니다.

```
CREATE EXTERNAL TABLE IF NOT EXISTS cloudfront_real_time_logs (
`timestamp` STRING,
```

```
c_ip STRING,
time_to_first_byte BIGINT,
sc_status BIGINT,
sc_bytes BIGINT,
cs_method STRING,
cs_protocol STRING,
cs_host STRING,
cs_uri_stem STRING,
cs_bytes BIGINT,
x_edge_location STRING,
x_edge_request_id STRING,
x_host_header STRING,
time_taken BIGINT,
cs_protocol_version STRING,
c_ip_version STRING,
cs_user_agent STRING,
cs_referer STRING,
cs_cookie STRING,
cs_uri_query STRING,
x_edge_response_result_type STRING,
x_forwarded_for STRING,
ssl_protocol STRING,
ssl_cipher STRING,
x_edge_result_type STRING,
fle_encrypted_fields STRING,
fle_status STRING,
sc_content_type STRING,
sc_content_len BIGINT,
sc_range_start STRING,
sc_range_end STRING,
c_port BIGINT,
x_edge_detailed_result_type STRING,
c_country STRING,
cs_accept_encoding STRING,
cs_accept STRING,
cache_behavior_path_pattern STRING,
cs_headers STRING,
cs_header_names STRING,
cs_headers_count BIGINT,
primary_distribution_id STRING,
primary_distribution_dns_name STRING,
origin_fbl STRING,
origin_lbl STRING,
asn STRING
```

```
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LOCATION 's3://DOC-EXAMPLE-BUCKET/'
TBLPROPERTIES ( 'skip.header.line.count'='2' )
```

2. Athena 콘솔에서 쿼리를 실행합니다. 쿼리가 완료된 후 Athena는 `cloudfront_real_time_logs` 테이블을 등록하여 쿼리를 실행할 수 있도록 데이터를 준비합니다.

표준 CloudFront 로그의 쿼리 예제

다음 쿼리는 2018년 6월 9일부터 6월 11일까지 CloudFront가 제공한 바이트 수를 합산합니다. 날짜 열 이름은 예약된 단어이므로 큰따옴표로 묶습니다.

```
SELECT SUM(bytes) AS total_bytes
FROM cloudfront_standard_logs
WHERE "date" BETWEEN DATE '2018-06-09' AND DATE '2018-06-11'
LIMIT 100;
```

쿼리 결과에서 중복 행(예: 중복된 빈 행)을 제거하려면 다음 예제와 같이 `SELECT DISTINCT` 문을 사용할 수 있습니다.

```
SELECT DISTINCT *
FROM cloudfront_standard_logs
LIMIT 10;
```

추가적인 리소스

Athena를 사용하여 CloudFront 로그를 쿼리하는 방법에 대한 자세한 내용은 AWS Big Data Blog의 다음 게시물을 참조하세요.

Easily query AWS 서비스 logs using Amazon Athena(2019년 5월 29일).

Analyze your Amazon CloudFront access logs at scale(대규모의 Amazon CloudFront 액세스 로그 분석)(2018년 12월 21일).

Build a serverless architecture to analyze Amazon CloudFront access logs using AWS Lambda, Amazon Athena, and Amazon Managed Service for Apache Flink(2017년 5월 26일).

AWS CloudTrail 로그 쿼리

AWS CloudTrail는 AWS API 호출 및 Amazon Web Services 계정 이벤트를 기록하는 서비스입니다.

CloudTrail 로그에는 AWS 서비스(콘솔 포함)에 수행된 API 호출에 대한 세부 정보가 포함됩니다. CloudTrail은 암호화된 로그 파일을 생성하여 Amazon S3에 저장합니다. 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

> **Note**
>
> 여러 계정, 리전 및 날짜에 걸쳐 CloudTrail 이벤트 정보에 대한 SQL 쿼리를 수행하려면 CloudTrail Lake 사용을 고려해 보세요. CloudTrail Lake는 기업의 정보를 검색 가능한 단일 이벤트 데이터 스토어로 집계하는 트레일을 생성할 수 있는 또 다른 AWS 솔루션입니다. 이 솔루션은 Amazon S3 버킷 스토리지를 사용하지 않고 대신 이벤트를 데이터 레이크에 저장하므로 더 풍부하고 빠른 쿼리가 가능합니다. 이를 사용하여 여러 조직, 지역 및 사용자 지정 시간 범위 내에서 이벤트를 검색하는 SQL 쿼리를 만들 수 있습니다. CloudTrail 콘솔 자체에서 CloudTrail Lake 쿼리를 수행하므로 CloudTrail Lake를 사용하는 경우 Athena가 필요하지 않습니다. 자세한 내용은 CloudTrail Lake 설명서를 참조하세요.

CloudTrail 로그와 함께 Athena를 사용하면 AWS 서비스 활동에 대한 분석 기능을 확실하게 향상시킬 수 있습니다. 예를 들어 쿼리를 사용하여 트렌드를 식별하고 소스 IP 주소나 사용자 등의 속성별로 활동을 추가로 격리할 수 있습니다.

일반적인 적용 사례는 CloudTrail 로그를 사용해 보안 및 규정 준수를 위한 운영 활동을 분석하는 것입니다. 자세한 예제는 AWS Big Data Blog 게시물 Analyze Security, Compliance, and Operational Activity Using AWS CloudTrail and Amazon Athena를 참조하세요.

Athena를 사용하여 로그 파일의 `LOCATION`을 지정하면 Amazon S3에서 직접 이러한 로그 파일을 쿼리할 수 있습니다. 이 작업을 두 가지 방법으로 수행할 수 있습니다.

- CloudTrail 콘솔에서 직접 CloudTrail 로그 파일용 테이블을 생성하는 방법.
- Athena 콘솔에서 CloudTrail 로그 파일용 테이블을 수동으로 생성하는 방법.

주제

- 수동 분할을 사용하여 Athena에서 CloudTrail 로그용 테이블 생성
- 수동 분할을 사용하여 조직 전체 추적을 위한 테이블 생성
- 파티션 프로젝션을 사용하여 Athena에서 CloudTrail 로그용 테이블 생성
- 중첩된 필드 쿼리
- 쿼리 예
- CloudTrail 로그 쿼리 팁

CloudTrail 로그 및 Athena 테이블 이해

테이블 생성을 시작하기 앞서 CloudTrail과 그 데이터 저장 방식을 자세히 알아봐야 합니다. 이렇게 하면 CloudTrail 콘솔 또는 Athena에서 필요한 테이블을 생성하는 데 도움이 됩니다.

CloudTrail은 압축된 gzip 형식(*.json.gzip)의 JSON 텍스트 파일로 로그를 저장합니다. 로그 파일의 위치는 추적 설정 방법, AWS 리전 또는 로깅 중인 리전 및 기타 요인에 따라 다릅니다.

로그가 저장되는 위치, JSON 구조 및 레코드 파일 내용에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 다음 주제를 참조하세요.

- CloudTrail 로그 파일 찾기
- CloudTrail 로그 파일의 예
- CloudTrail 레코드 콘텐츠
- CloudTrail 이벤트 참조

로그를 수집하여 Amazon S3에 저장하려면 AWS Management Console에서 CloudTrail을 사용하도록 설정합니다. 자세한 내용은 AWS CloudTrail 사용 설명서의 추적 생성을 참조하세요.

로그가 저장되는 대상 Amazon S3 버킷을 기록해 둡니다. `LOCATION` 절을 CloudTrail 로그 위치의 경로와 작업할 객체 집합으로 바꿉니다. 이 예에서는 특정 계정에 대해 로그의 `LOCATION` 값을 사용하지만 애플리케이션에 적합한 정도의 특정성을 사용할 수 있습니다.

다음 예를 참조하세요.

- 여러 계정의 데이터를 분석하려면 `LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/'`를 이용해 `LOCATION` 범위 지정자를 롤백하여 모든 `AWSLogs`를 표시할 수 있습니다.
- 특정 날짜, 계정 및 리전의 데이터를 분석하려면 `LOCATION 's3://DOC-EXAMPLE-BUCKET/123456789012/CloudTrail/us-east-1/2016/03/14/'.`를 사용합니다.

객체 계층 구조에서 최상위 수준을 사용하면 Athena로 쿼리할 때 가장 큰 유연성을 발휘할 수 있습니다.

CloudTrail 콘솔을 사용하여 CloudTrail 로그용 Athena 테이블 생성

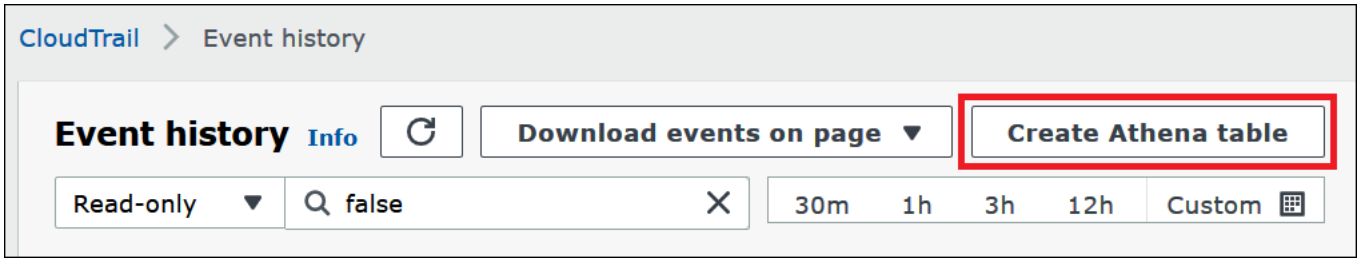
이전 CloudTrail 콘솔에서 직접 CloudTrail 로그를 쿼리하기 위해 분할되지 않은 Athena 테이블을 생성할 수 있습니다. CloudTrail 콘솔에서 Athena 테이블을 생성하려면 Athena에서 테이블을 생성할 수 있는 충분한 권한이 있는 역할로 로그인해야 합니다.

> **Note**
>
> 조직 추적 로그에 대한 Athena 테이블은 CloudTrail 콘솔을 사용해 만들 수 없습니다. 대신 Athena 콘솔을 사용하여 테이블을 수동으로 만들고 올바른 저장 위치를 지정할 수 있습니다. 조직 추적에 대한 자세한 내용은 AWS CloudTrail 사용 설명서에서 조직에 대한 추적 생성을 참조하세요.

- Athena 권한 설정에 대한 자세한 내용은 설정 단원을 참조하세요.
- 파티션이 있는 테이블 생성에 대한 자세한 내용은 수동 분할을 사용하여 Athena에서 CloudTrail 로그용 테이블 생성 단원을 참조하세요.

CloudTrail 콘솔을 사용하여 CloudTrail 추적용 Athena 테이블을 생성하려면

1. https://console.aws.amazon.com/cloudwatch/에서 CloudTrail 콘솔을 엽니다.
2. 탐색 창에서 Event history(이벤트 내역)를 선택합니다.
3. Athena 테이블 생성을 선택합니다.



4. 스토리지 위치에서, 아래쪽 화살표를 사용하여 추적이 쿼리할 로그 파일이 저장되는 Amazon S3 버킷을 선택합니다.

> **Note**
>
> 추적과 연결된 버킷의 이름을 찾으려면 CloudTrail 탐색 창에서 추적을 선택하고 추적의 S3 버킷 열을 표시합니다. 버킷의 Amazon S3 위치를 보려면 S3 버킷 열에서 버킷에 대한 링크를 선택합니다. 그러면 Amazon S3 콘솔이 CloudTrail 버킷 위치로 이동합니다.

5. 테이블 생성을 선택합니다. Amazon S3 버킷 이름을 포함하는 기본 이름을 가진 테이블이 생성됩니다.

수동 분할을 사용하여 Athena에서 CloudTrail 로그용 테이블 생성

Athena 콘솔에서 CloudTrail 로그 파일용 테이블을 수동으로 생성한 다음 Athena에서 쿼리를 실행할 수 있습니다.

Athena 콘솔을 사용하여 CloudTrail 추적에 대한 Athena 테이블을 만들려면

1. 다음 DDL 문을 복사하여 Athena 콘솔 쿼리 편집기에 붙여 넣습니다.

```
CREATE EXTERNAL TABLE cloudtrail_logs (
eventversion STRING,
useridentity STRUCT<
               type:STRING,
               principalid:STRING,
               arn:STRING,
               accountid:STRING,
               invokedby:STRING,
               accesskeyid:STRING,
               userName:STRING,
  sessioncontext:STRUCT<
    attributes:STRUCT<
               mfaauthenticated:STRING,
               creationdate:STRING>,
    sessionissuer:STRUCT<
               type:STRING,
               principalId:STRING,
               arn:STRING,
               accountId:STRING,
               userName:STRING>,
    ec2RoleDelivery:string,
    webIdFederationData: STRUCT<
               federatedProvider: STRING,
```

```
                attributes: map<string,string>
    >
  >
>,
eventtime STRING,
eventsource STRING,
eventname STRING,
awsregion STRING,
sourceipaddress STRING,
useragent STRING,
errorcode STRING,
errormessage STRING,
requestparameters STRING,
responseelements STRING,
additionaleventdata STRING,
requestid STRING,
eventid STRING,
resources ARRAY<STRUCT<
                arn:STRING,
                accountid:STRING,
                type:STRING>>,
eventtype STRING,
apiversion STRING,
readonly STRING,
recipientaccountid STRING,
serviceeventdetails STRING,
sharedeventid STRING,
vpcendpointid STRING,
eventCategory STRING,
tlsDetails struct<
  tlsVersion:string,
  cipherSuite:string,
  clientProvidedHostHeader:string>
)
PARTITIONED BY (region string, year string, month string, day string)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/Account_ID/CloudTrail/';
```

> **Note**
>
> 예제에 표시된 `org.apache.hive.hcatalog.data.JsonSerDe`를 사용하는 것이 좋습니다. `com.amazon.emr.hive.serde.CloudTrailSerde`가 존재하지만 현재 일부 최신 CloudTrail 필드는 처리하지 않습니다.

2. (선택 사항) 테이블에 필요하지 않은 필드를 제거합니다. 특정 열 집합만 읽어야 하는 경우 테이블 정의에서 다른 열을 제외할 수 있습니다.
3. 로그 데이터가 포함된 Amazon S3 버킷을 가리키도록 `s3://DOC-EXAMPLE-BUCKET/AWSLogs/Account_ID/CloudTrail/`을 수정합니다.
4. 필드가 올바르게 나열되는지 확인합니다. CloudTrail 레코드의 필드 전체 목록에 대한 자세한 내용은 CloudTrail 레코드 콘텐츠를 참조하세요.

   다음 예제에서는 Hive JSON SerDe를 사용합니다. 이 예에서 `requestparameters`, `responseelements`, `additionaleventdata` 필드는 쿼리에서 `STRING` 형식으로 나열되지만, JSON에서는 `STRUCT` 데이터 형식이 사용됩니다. 따라서 이러한 필드에서 데이터를 가져오려면 `JSON_EXTRACT` 함수를 사용합니다. 자세한 내용은 the section called “JSON에서 데이터 추출” 단원을 참조하십시오. 성능 향상을 위해 예제에서는 데이터를 AWS 리전, 연도, 월 및 일별로 분할합니다.
5. Athena 콘솔에서 `CREATE TABLE` 문을 실행합니다.
6. 다음 예제와 같이 ALTER TABLE ADD PARTITION 명령을 사용하여 파티션을 쿼리할 수 있도록 파티션을 로드합니다.

```
ALTER TABLE table_name ADD
   PARTITION (region='us-east-1',
              year='2019',
              month='02',
              day='01')
   LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/Account_ID/CloudTrail/us-
east-1/2019/02/01/'
```

수동 분할을 사용하여 조직 전체 추적을 위한 테이블 생성

Athena에서 조직 전체 CloudTrail 로그 파일에 대한 테이블을 생성하려면 수동 분할을 사용하여 Athena에서 CloudTrail 로그용 테이블 생성의 단계를 따르되, 다음 절차에 설명된 대로 수정하세요.

조직 전체 CloudTrail 로그에 대한 Athena 테이블을 생성하려면

1. CREATE TABLE 문에서 다음 예시와 같이 조직 ID를 포함하도록 LOCATION 절을 수정합니다.

```
LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/organization_id/Account_ID/CloudTrail/'
```

2. PARTITIONED BY 절에서 다음 예시와 같이 계정 ID에 대한 항목을 문자열로 추가합니다.

```
PARTITIONED BY (account string, region string, year string, month string, day
 string)
```

다음 예시는 결합된 결과를 보여줍니다.

```
...

PARTITIONED BY (account string, region string, year string, month string, day
 string)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/organization_id/Account_ID/CloudTrail/'
```

3. ALTER TABLE 문 ADD PARTITION 절에서 다음 예시와 같이 계정 ID를 포함합니다.

```
ALTER TABLE table_name ADD
PARTITION (account='111122223333',
region='us-east-1',
year='2022',
month='08',
day='08')
```

4. ALTER TABLE 문 LOCATION 절에서 다음 예시와 같이 조직 ID, 계정 ID 및 추가할 파티션을 포함합니다.

```
LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/organization_id/Account_ID/CloudTrail/us-
east-1/2022/08/08/'
```

다음 예시 ALTER TABLE 문은 결합된 결과를 보여줍니다.

```
ALTER TABLE table_name ADD
PARTITION (account='111122223333',
```

```
region='us-east-1',
year='2022',
month='08',
day='08')
LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/organization_id/111122223333/CloudTrail/
us-east-1/2022/08/08/'
```

파티션 프로젝션을 사용하여 Athena에서 CloudTrail 로그용 테이블 생성

CloudTrail 로그에는 미리 지정할 수 있는 파티션 스키마를 가진 알려진 구조가 있기 때문에 Athena 파티션 프로젝션 기능을 사용하여 쿼리 런타임을 줄이고 파티션 관리를 자동화할 수 있습니다. 새 데이터가 추가되면 파티션 프로젝션은 자동으로 새 파티션을 추가합니다. 따라서 `ALTER TABLE ADD PARTITION`을 사용해 파티션을 수동으로 추가할 필요가 없습니다.

다음 `CREATE TABLE` 문 예제에서는 하나의 AWS 리전에 대해 지정된 날짜부터 현재까지의 CloudTrail 로그에 파티션 프로젝션을 자동으로 사용합니다. `LOCATION` 및 `storage.location.template` 절에서 *bucket*, *account-id* 및 *aws-region* 자리 표시자를 상응하는 동일한 값으로 바꿉니다. `projection.timestamp.range`에 대해 *2020*/*01*/*01*을 사용하려는 시작 날짜로 바꿉니다. 쿼리가 성공적으로 실행된면 테이블을 쿼리할 수 있습니다. 파티션을 로드하기 위해 `ALTER TABLE ADD PARTITION`을 실행하지 않아도 됩니다.

```
CREATE EXTERNAL TABLE cloudtrail_logs_pp(
    eventVersion STRING,
    userIdentity STRUCT<
        type: STRING,
        principalId: STRING,
        arn: STRING,
        accountId: STRING,
        invokedBy: STRING,
        accessKeyId: STRING,
        userName: STRING,
        sessionContext: STRUCT<
            attributes: STRUCT<
                mfaAuthenticated: STRING,
                creationDate: STRING>,
            sessionIssuer: STRUCT<
                type: STRING,
                principalId: STRING,
                arn: STRING,
                accountId: STRING,
                userName: STRING>,
```

```
            ec2RoleDelivery:string,
            webIdFederationData: STRUCT<
                federatedProvider: STRING,
                attributes: map<string,string>
            >
        >
    >,
    eventTime STRING,
    eventSource STRING,
    eventName STRING,
    awsRegion STRING,
    sourceIpAddress STRING,
    userAgent STRING,
    errorCode STRING,
    errorMessage STRING,
    requestparameters STRING,
    responseelements STRING,
    additionaleventdata STRING,
    requestId STRING,
    eventId STRING,
    readOnly STRING,
    resources ARRAY<STRUCT<
        arn: STRING,
        accountId: STRING,
        type: STRING>>,
    eventType STRING,
    apiVersion STRING,
    recipientAccountId STRING,
    serviceEventDetails STRING,
    sharedEventID STRING,
    vpcendpointid STRING,
    eventCategory STRING,
    tlsDetails struct<
        tlsVersion:string,
        cipherSuite:string,
        clientProvidedHostHeader:string>
  )
PARTITIONED BY (
   `timestamp` string)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/AWSLogs/account-id/CloudTrail/aws-region'
```

```
TBLPROPERTIES (
  'projection.enabled'='true',
  'projection.timestamp.format'='yyyy/MM/dd',
  'projection.timestamp.interval'='1',
  'projection.timestamp.interval.unit'='DAYS',
  'projection.timestamp.range'='2020/01/01,NOW',
  'projection.timestamp.type'='date',
  'storage.location.template'='s3://DOC-EXAMPLE-BUCKET/AWSLogs/account-id/
CloudTrail/aws-region/${timestamp}')
```

파티션 프로젝션에 대한 자세한 내용은 Amazon Athena를 사용한 파티션 프로젝션 단원을 참조하세요.

중첩된 필드 쿼리

`userIdentity` 및 `resources` 필드는 중첩된 데이터 형식이기 때문에 이를 쿼리하려면 특수한 처리가 필요합니다.

`userIdentity` 객체는 중첩된 STRUCT 형식으로 구성됩니다. 다음 예에서처럼 점으로 필드를 구분하여 이 객체들을 쿼리할 수 있습니다.

```
SELECT
    eventsource,
    eventname,
    useridentity.sessioncontext.attributes.creationdate,
    useridentity.sessioncontext.sessionissuer.arn
FROM cloudtrail_logs
WHERE useridentity.sessioncontext.sessionissuer.arn IS NOT NULL
ORDER BY eventsource, eventname
LIMIT 10
```

`resources` 필드는 STRUCT 객체의 배열입니다. 이러한 배열의 경우 `CROSS JOIN UNNEST`를 사용해 배열의 중첩을 해제함으로써 해당 객체를 쿼리할 수 있습니다.

다음 예제는 리소스 ARN이 `example/datafile.txt`로 끝나는 모든 행을 반환합니다. replace 함수는 가독성을 위해 ARN에서 초기 `arn:aws:s3:::` 하위 문자열을 제거합니다.

```
SELECT
    awsregion,
    replace(unnested.resources_entry.ARN,'arn:aws:s3:::') as s3_resource,
    eventname,
    eventtime,
    useragent
```

```
FROM cloudtrail_logs t
CROSS JOIN UNNEST(t.resources) unnested (resources_entry)
WHERE unnested.resources_entry.ARN LIKE '%example/datafile.txt'
ORDER BY eventtime
```

다음 예제에서는 DeleteBucket 이벤트를 쿼리합니다. 이 쿼리는 resources 객체에서 버킷의 이름 및 버킷이 속한 계정 ID를 추출합니다.

```
SELECT
    awsregion,
    replace(unnested.resources_entry.ARN,'arn:aws:s3:::') as deleted_bucket,
    eventtime AS time_deleted,
    useridentity.username,
    unnested.resources_entry.accountid as bucket_acct_id
FROM cloudtrail_logs t
CROSS JOIN UNNEST(t.resources) unnested (resources_entry)
WHERE eventname = 'DeleteBucket'
ORDER BY eventtime
```

중첩 해제에 대한 자세한 내용은 배열 필터링 단원을 참조하세요.

쿼리 예

다음 예제는 CloudTrail 이벤트 로그에 대해 생성된 테이블에서 모든 익명(서명되지 않은) 요청을 반환하는 쿼리의 일부를 보여줍니다. 이 쿼리는 useridentity.accountid가 익명이고 useridentity.arn이 지정되지 않은 요청을 선택합니다.

```
SELECT *
FROM cloudtrail_logs
WHERE
    eventsource = 's3.amazonaws.com' AND
    eventname in ('GetObject') AND
    useridentity.accountid = 'anonymous' AND
    useridentity.arn IS NULL AND
    requestparameters LIKE '%[your bucket name ]%';
```

자세한 내용은 AWS Big Data Blog 게시물 Analyze Security, Compliance, and Operational Activity Using AWS CloudTrail and Amazon Athena를 참조하세요.

CloudTrail 로그 쿼리 팁

CloudTrail 로그 데이터를 탐색하려면 다음 팁을 활용하세요.

- 로그를 쿼리하기 전에 로그 테이블이 the section called “수동 분할을 사용하여 Athena에서 CloudTrail 로그용 테이블 생성”의 로그 테이블과 동일한 형태인지 확인합니다. 첫 번째 테이블이 아닌 경우 `DROP TABLE cloudtrail_logs` 명령을 사용하여 기존 테이블을 삭제합니다.
- 기존 테이블을 삭제하고 다시 만드세요. 자세한 내용은 수동 분할을 사용하여 Athena에서 CloudTrail 로그용 테이블 생성 단원을 참조하십시오.

  Athena 쿼리의 필드가 올바르게 나열되는지 확인합니다. CloudTrail 레코드의 필드 전체 목록에 대한 자세한 내용은 CloudTrail 레코드 콘텐츠를 참조하세요.

  쿼리에 STRUCT 같은 JSON 형식의 필드가 포함된 경우, JSON에서 데이터를 추출합니다. 자세한 내용은 JSON에서 데이터 추출 단원을 참조하십시오.

  다음은 CloudTrail 테이블에 대해 쿼리를 실행하기 위한 몇 가지 제안 사항입니다.
- 먼저 어떤 사용자가 어떤 API 작업을 호출했으며 어떤 소스 IP 주소에서 시작되었는지 확인합니다.
- 다음 기본 SQL 쿼리를 템플릿으로 사용합니다. 쿼리를 Athena 콘솔에 붙여 넣고 실행합니다.

```
SELECT
 useridentity.arn,
 eventname,
 sourceipaddress,
 eventtime
FROM cloudtrail_logs
LIMIT 100;
```

- 데이터를 추가 탐색하려면 쿼리를 수정합니다.
- 성능을 향상시키려면 `LIMIT` 절을 포함시켜 지정한 행 하위 집합을 반환합니다.

Amazon EMR 로그 쿼리

Amazon EMR 및 Amazon EMR에서 실행되는 빅 데이터 애플리케이션은 로그 파일을 생성합니다. 로그 파일은 마스터 노드에 작성되며 로그 파일을 Amazon S3에 자동으로 보관하도록 Amazon EMR을 구성할 수도 있습니다. Amazon Athena를 사용하여 이러한 로그를 쿼리해 애플리케이션 및 클러스터의 이벤트 및 추세를 식별할 수 있습니다. Amazon EMR의 로그 파일 형식 및 Amazon S3에 이를 저장하는 방법에 대한 자세한 내용은 Amazon EMR 관리 안내서의 로그 파일 보기를 참조하세요.

Amazon EMR 로그 파일 기반 기본 테이블 생성 및 쿼리

다음 예제에서는 `s3://aws-logs-`*`123456789012`*`-`*`us-west-2`*`/elasticmapreduce/`*`j-2ABCDE34F5GH6`*`/elasticmapreduce/`에 저장된 로그 파일을 기반으로

기본 테이블 myemrlogs를 생성합니다. 아래 예제에 사용된 Amazon S3 위치는 *us-west-2* 리전의 Amazon Web Services 계정 *123456789012*에서 생성한 EMR 클러스터의 기본 로그 위치 패턴을 반영합니다. 사용자 지정 위치를 사용하는 경우 패턴은 s3://DOC-EXAMPLE-BUCKET/*ClusterID*입니다.

잠재적으로 쿼리 성능을 향상시키고 데이터 전송을 줄이기 위해 분할된 테이블을 생성하는 방법에 대한 자세한 내용은 Amazon EMR 로그 기반 분할된 테이블 생성 및 쿼리 단원을 참조하세요.

```
CREATE EXTERNAL TABLE `myemrlogs`(
  `data` string COMMENT 'from deserializer')
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '\n'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6'
```

다음 예제 쿼리는 이전 예제에서 생성된 myemrlogs 테이블에서 실행할 수 있습니다.

Example - ERROR, WARN, INFO, EXCEPTION, FATAL 또는 DEBUG 발생에 대한 단계 로그 쿼리

```
SELECT data,
        "$PATH"
FROM "default"."myemrlogs"
WHERE regexp_like("$PATH",'s-86URH188Z6B1')
        AND regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

Example - ERROR, WARN, INFO, EXCEPTION, FATAL 또는 DEBUG에 대한 특정 인스턴스 로그 i-00b3c0a839ece0a9c 쿼리

```
SELECT "data",
        "$PATH" AS filepath
FROM "default"."myemrlogs"
WHERE regexp_like("$PATH",'i-00b3c0a839ece0a9c')
        AND regexp_like("$PATH",'state')
        AND regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

Example - ERROR, WARN, INFO, EXCEPTION, FATAL 또는 DEBUG에 대한 Presto 애플리케이션 로그 쿼리

```
SELECT "data",
        "$PATH" AS filepath
FROM "default"."myemrlogs"
WHERE regexp_like("$PATH",'presto')
        AND regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

Example - ERROR, WARN, INFO, EXCEPTION, FATAL 또는 DEBUG에 대한 Namenode 애플리케이션 로그 쿼리

```
SELECT "data",
        "$PATH" AS filepath
FROM "default"."myemrlogs"
WHERE regexp_like("$PATH",'namenode')
        AND regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

Example - ERROR, WARN, INFO, EXCEPTION, FATAL 또는 DEBUG에 대해 날짜 및 시간별 모든 로그 쿼리

```
SELECT distinct("$PATH") AS filepath
FROM "default"."myemrlogs"
WHERE regexp_like("$PATH",'2019-07-23-10')
        AND regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

Amazon EMR 로그 기반 분할된 테이블 생성 및 쿼리

이러한 예제에서는 동일한 로그 위치를 사용하여 Athena 테이블을 생성하지만 테이블은 분할되고 각 로그 위치에 대해 파티션이 생성됩니다. 자세한 내용은 Athena에서 데이터 분할 단원을 참조하세요.

다음 쿼리는 `mypartitionedemrlogs`라는 분할된 테이블을 생성합니다.

```
CREATE EXTERNAL TABLE `mypartitionedemrlogs`(
  `data` string COMMENT 'from deserializer')
 partitioned by (logtype string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '\n'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
```

```
OUTPUTFORMAT
 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6'
```

다음 쿼리 문은 Amazon EMR이 Amazon S3에서 생성하는 다양한 로그 유형에 대한 하위 디렉터리를 기반으로 테이블 파티션을 생성합니다.

```
ALTER TABLE mypartitionedemrlogs ADD
     PARTITION (logtype='containers')
     LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/
containers/'
```

```
ALTER TABLE mypartitionedemrlogs ADD
     PARTITION (logtype='hadoop-mapreduce')
     LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/
hadoop-mapreduce/'
```

```
ALTER TABLE mypartitionedemrlogs ADD
     PARTITION (logtype='hadoop-state-pusher')
     LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/
hadoop-state-pusher/'
```

```
ALTER TABLE mypartitionedemrlogs ADD
     PARTITION (logtype='node')
     LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/
node/'
```

```
ALTER TABLE mypartitionedemrlogs ADD
     PARTITION (logtype='steps')
     LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6/
steps/'
```

파티션을 생성한 후 테이블에서 `SHOW PARTITIONS` 쿼리를 실행하여 다음 사항을 확인할 수 있습니다.

```
SHOW PARTITIONS mypartitionedemrlogs;
```

다음 예제에서는 특정 로그 항목에 대한 쿼리가 위 예제에서 생성된 테이블 및 파티션을 사용한다는 것을 보여줍니다.

Example - ERROR 또는 WARN에 대한 컨테이너 파티션의 애플리케이션 application_1561661818238_0002 로그 쿼리

```
SELECT data,
        "$PATH"
FROM "default"."mypartitionedemrlogs"
WHERE logtype='containers'
        AND regexp_like("$PATH",'application_1561661818238_0002')
        AND regexp_like(data, 'ERROR|WARN') limit 100;
```

Example - 작업 job_1561661818238_0004 및 Failed Reduces에 대한 hadoop-Mapreduce 파티션 쿼리

```
SELECT data,
        "$PATH"
FROM "default"."mypartitionedemrlogs"
WHERE logtype='hadoop-mapreduce'
        AND regexp_like(data,'job_1561661818238_0004|Failed Reduces') limit 100;
```

Example - 쿼리 ID 056e0609-33e1-4611-956c-7a31b42d2663에 대한 노드 파티션의 Hive 로그 쿼리

```
SELECT data,
        "$PATH"
FROM "default"."mypartitionedemrlogs"
WHERE logtype='node'
        AND regexp_like("$PATH",'hive')
        AND regexp_like(data,'056e0609-33e1-4611-956c-7a31b42d2663') limit 100;
```

Example - 애플리케이션 1567660019320_0001_01_000001에 대한 노드 파티션의 resourcemanager 로그 쿼리

```
SELECT data,
        "$PATH"
FROM "default"."mypartitionedemrlogs"
WHERE logtype='node'
        AND regexp_like(data,'resourcemanager')
        AND regexp_like(data,'1567660019320_0001_01_000001') limit 100
```

AWS Global Accelerator 흐름 로그 쿼리

AWS Global Accelerator를 사용하면 AWS 글로벌 네트워크를 통해 네트워크 트래픽을 최적의 엔드포인트로 보내는 액셀러레이터를 생성할 수 있습니다. Global Accelerator에 대한 자세한 내용은 AWS Global Accelerator란 무엇인가요?를 참조하세요.

Global Accelerator 흐름 로그를 사용하면 액셀러레이터의 네트워크 인터페이스에서 송수신되는 IP 주소 트래픽에 대한 정보를 캡처할 수 있습니다. 데이터를 검색하고 확인할 수 있는 흐름 로그 데이터가 Amazon S3에 게시됩니다. 자세한 내용은 AWS Global Accelerator의 흐름 로그를 참조하세요.

Athena를 사용하여 Amazon S3에서 해당 위치를 지정하는 테이블을 만들어 Global Accelerator 흐름 로그를 쿼리할 수 있습니다.

Global Accelerator 흐름 로그에 대한 테이블을 생성하려면

1. 다음 DDL 문을 복사하여 Athena 콘솔에 붙여 넣습니다. 이 쿼리는 ROW FORMAT DELIMITED를 지정하고 SerDe를 지정하지 않습니다(즉, `LazySimpleSerDe` 사용). 이 쿼리에서 필드는 공백으로 끝납니다.

```
CREATE EXTERNAL TABLE IF NOT EXISTS aga_flow_logs (
  version string,
  account string,
  acceleratorid string,
  clientip string,
  clientport int,
  gip string,
  gipport int,
  endpointip string,
  endpointport int,
  protocol string,
  ipaddresstype string,
  numpackets bigint,
  numbytes int,
  starttime int,
  endtime int,
  action string,
  logstatus string,
  agasourceip string,
  agasourceport int,
  endpointregion string,
  agaregion string,
  direction string
```

```
)
PARTITIONED BY (dt string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
LOCATION 's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/account_id/globalaccelerator/
region/'
TBLPROPERTIES ("skip.header.line.count"="1");
```

2. 로그 데이터가 포함된 Amazon S3 버킷을 가리키도록 LOCATION 값을 수정합니다.

```
's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/account_id/globalaccelerator/region_code/'
```

3. Athena 콘솔에서 쿼리를 실행합니다. 쿼리가 완료된 후 Athena는 데이터를 쿼리에 사용할 수 있도록 aga_flow_logs 테이블을 등록합니다.
4. 다음 샘플 쿼리와 같이 데이터를 읽을 수 있도록 파티션을 생성합니다. 이 쿼리는 지정한 날짜에 대한 단일 파티션을 생성합니다. 날짜와 위치의 자리 표시자를 교체합니다.

```
ALTER TABLE aga_flow_logs
ADD PARTITION (dt='YYYY-MM-dd')
LOCATION 's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/account_id/
globalaccelerator/region_code/YYYY/MM/dd';
```

AWS Global Accelerator 흐름 로그에 대한 쿼리 예제

Example - 특정 엣지 로케이션을 전달하는 요청 나열

다음 예제 쿼리는 LHR 엣지 로케이션을 전달한 요청을 나열합니다. LIMIT 연산자를 사용하여 한 번에 쿼리할 로그 수를 제한합니다.

```
SELECT
  clientip,
  agaregion,
  protocol,
  action
FROM
  aga_flow_logs
WHERE
  agaregion LIKE 'LHR%'
LIMIT
  100;
```

Example - 가장 많은 HTTPS 요청을 수신하는 엔드포인트 IP 주소 나열

가장 많은 수의 HTTPS 요청을 수신하는 엔드포인트 IP 주소를 확인하려면 다음 쿼리를 사용합니다. 이 쿼리는 HTTPS 포트 443에서 수신한 패킷 수를 계산하고, 대상 IP 주소별로 그룹화한 다음, 상위 10개 IP 주소를 반환합니다.

```
SELECT
  SUM(numpackets) AS packetcount,
  endpointip
FROM
  aga_flow_logs
WHERE
  endpointport = 443
GROUP BY
  endpointip
ORDER BY
  packetcount DESC
LIMIT
  10;
```

Amazon GuardDuty 결과 쿼리

Amazon GuardDuty는 AWS 환경에서 예상치 못한 활동, 잠재적으로 승인되지 않은 활동 또는 악의적 활동을 식별할 수 있도록 지원하는 보안 모니터링 서비스입니다. GuardDuty는 예상치 못한 활동 및 잠재적으로 악의적인 활동을 탐지할 경우 보안 결과를 생성합니다. 사용자는 이 결과를 Amazon S3에 내보내어 저장하고 분석할 수 있습니다. 또한 결과를 Amazon S3에 내보낸 후 Athena를 사용하여 결과를 쿼리할 수 있습니다. 이 문서에서는 Athena에서 GuardDuty 결과에 대한 테이블을 만들고 쿼리하는 방법을 보여 줍니다.

Amazon GuardDuty에 대한 자세한 내용은 Amazon GuardDuty 사용 설명서를 참조하세요.

필수 조건

- 결과를 Amazon S3에 내보내는 GuardDuty 기능을 활성화합니다. 실행 단계는 Amazon GuardDuty 사용 설명서의 결과 내보내기를 참조하세요.

Athena에서 GuardDuty 결과에 대한 테이블 생성

Athena에서 GuardDuty 결과를 쿼리하려면 해당 결과에 대한 테이블을 생성해야 합니다.

Athena에서 GuardDuty 결과에 대한 테이블을 생성하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 다음 DDL 문을 Athena 콘솔에 붙여 넣습니다. Amazon S3의 GuardDuty 결과를 가리키도록 `LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/account-id/GuardDuty/'`의 값을 수정합니다.

```
CREATE EXTERNAL TABLE `gd_logs` (
  `schemaversion` string,
  `accountid` string,
  `region` string,
  `partition` string,
  `id` string,
  `arn` string,
  `type` string,
  `resource` string,
  `service` string,
  `severity` string,
  `createdat` string,
  `updatedat` string,
  `title` string,
  `description` string)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
 LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/account-id/GuardDuty/'
 TBLPROPERTIES ('has_encrypted_data'='true')
```

> **Note**
>
> SerDe는 각 JSON 문서가 레코드의 필드를 구분하는 줄 종료 문자가 없는 한 줄의 텍스트에 있을 것으로 예상합니다. JSON 텍스트가 가독성 좋게 꾸민 형식이면 테이블을 만든 후 쿼리하려고 할 때 HIVE_CURSOR_ERROR: 행이 유효한 JSON 객체가 아님(HIVE_CURSOR_ERROR: Row is not a valid JSON Object) 또는 HIVE_CURSOR_ERROR: JsonParseException: 예기치 않은 입력 종료: OBJECT의 닫기 마커 필요(HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT) 같은 오류 메시지가 나타날 수 있습니다. 자세한 내용은 GitHub의 OpenX SerDe 문서에서 JSON 데이터 파일을 참조하세요.

3. Athena 콘솔에서 `gd_logs` 테이블을 등록하는 쿼리를 실행합니다. 이 쿼리가 완료되면 결과를 Athena에서 쿼리할 수 있습니다.

쿼리 예제

다음 예제에서는 Athena에서 GuardDuty 결과를 쿼리하는 방법을 보여 줍니다.

Example - DNS 데이터 유출

다음 쿼리는 DNS 쿼리를 통해 데이터를 유출할 수 있는 Amazon EC2 인스턴스에 대한 정보를 반환합니다.

```
SELECT
    title,
    severity,
    type,
    id AS FindingID,
    accountid,
    region,
    createdat,
    updatedat,
    json_extract_scalar(service, '$.count') AS Count,
    json_extract_scalar(resource, '$.instancedetails.instanceid') AS InstanceID,
    json_extract_scalar(service, '$.action.actiontype') AS DNS_ActionType,
    json_extract_scalar(service, '$.action.dnsrequestaction.domain') AS DomainName,
    json_extract_scalar(service, '$.action.dnsrequestaction.protocol') AS protocol,
    json_extract_scalar(service, '$.action.dnsrequestaction.blocked') AS blocked
FROM gd_logs
WHERE type = 'Trojan:EC2/DNSDataExfiltration'
ORDER BY severity DESC
```

Example - 무단 IAM 사용자 액세스

다음 쿼리는 모든 리전에서 IAM 보안 주체에 대한 모든 `UnauthorizedAccess:IAMUser` 결과 유형을 반환합니다.

```
SELECT title,
         severity,
         type,
         id,
         accountid,
         region,
         createdat,
         updatedat,
         json_extract_scalar(service, '$.count') AS Count,
         json_extract_scalar(resource, '$.accesskeydetails.username') AS IAMPrincipal,
```

```
        json_extract_scalar(service,'$.action.awsapicallaction.api') AS
 APIActionCalled
FROM gd_logs
WHERE type LIKE '%UnauthorizedAccess:IAMUser%'
ORDER BY severity desc;
```

GuardDuty 결과 쿼리 팁

쿼리를 작성할 때 다음 사항에 유의하세요.

- 중첩된 JSON 필드에서 데이터를 추출하려면 Presto `json_extract` 또는 `json_extract_scalar` 함수를 사용합니다. 자세한 내용은 JSON에서 데이터 추출 단원을 참조하세요.
- JSON 필드의 모든 문자가 소문자인지 확인합니다.
- 쿼리 결과 다운로드에 대한 자세한 내용은 Athena 콘솔을 사용하여 쿼리 결과 파일 다운로드 단원을 참조하세요.

AWS Network Firewall 로그 쿼리

AWS Network Firewall는 Amazon Virtual Private Cloud 인스턴스를 위한 필수 네트워크 보안을 배포할 수 있는 관리형 서비스입니다. AWS Network Firewall는 AWS Firewall Manager와 함께 작동하기 때문에 AWS Network Firewall 규칙에 기반해 정책을 빌드한 다음 중앙에서 이러한 정책을 VPC 및 계정 전반에 적용할 수 있습니다. AWS Network Firewall에 대한 자세한 내용은 AWS Network Firewall 단원을 참조하세요.

방화벽의 상태 저장 규칙 엔진에 전달하는 트래픽에 대해 AWS Network Firewall 로깅을 구성할 수 있습니다. 로깅은 상태 저장 엔진이 패킷을 받은 시간, 패킷에 대한 상세 정보, 패킷에 대해 수행된 상태 저장 규칙 동작을 비롯해 네트워크 트래픽에 대한 자세한 정보를 제공합니다. 로그는 구성한 로그 대상에 게시되며, 여기서 로그를 검색하고 확인할 수 있습니다. 자세한 내용은 AWS Network Firewall 개발자 안내서의 AWS Network Firewall에서 네트워크 트리팩 로깅을 참조하세요.

알림 로그용 테이블 생성

1. 다음 샘플 DDL 문을 알림 로그의 구조에 맞게 수정하세요. 최신 버전 로그의 열을 포함하도록 문을 업데이트해야 할 수 있습니다. 자세한 내용은 AWS Network Firewall 개발자 안내서의 방화벽 로그의 내용을 참조하세요.

   ```
   CREATE EXTERNAL TABLE network_firewall_alert_logs (
     firewall_name string,
   ```

```
  availability_zone string,
  event_timestamp string,
  event struct<
    timestamp:string,
    flow_id:bigint,
    event_type:string,
    src_ip:string,
    src_port:int,
    dest_ip:string,
    dest_port:int,
    proto:string,
    app_proto:string,
    tls_inspected:boolean,
    alert:struct<
      alert_id:string,
      alert_type:string,
      action:string,
      signature_id:int,
      rev:int,
      signature:string,
      category:string,
      severity:int,
      rule_name:string,
      alert_name:string,
      alert_severity:string,
      alert_description:string,
      file_name:string,
      file_hash:string,
      packet_capture:string,
      reference_links:array<string>
    >,
    src_country:string,
    dest_country:string,
    src_hostname:string,
    dest_hostname:string,
    user_agent:string,
    url:string
   >
)
 ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
 LOCATION 's3://DOC-EXAMPLE-BUCKET/path_to_alert_logs_folder/';
```

2. LOCATION 절을 수정하여 Amazon S3의 로그 폴더를 지정합니다.

3. Athena 쿼리 편집기에서 사용자의 CREATE TABLE 쿼리를 실행합니다. 쿼리가 완료된 후 Athena는 network_firewall_alert_logs 테이블을 등록하여 쿼리 준비를 나타내는 데이터를 만듭니다.

알림 로그 샘플 쿼리

이 섹션의 샘플 알림 로그 쿼리는 TLS 검사가 수행된 이벤트 중 심각도 수준이 2 이상인 알림이 있는 이벤트를 필터링합니다.

쿼리는 별칭을 사용하여 해당 열이 속한 struct를 보여주는 출력 열 제목을 생성합니다. 예를 들어 event.alert.category 필드의 열 제목은 event_alert_category이며 그냥 category가 아닙니다. 열 이름을 추가로 사용자 지정하려면 원하는 대로 별칭을 수정할 수 있습니다. 예를 들어 밑줄이나 기타 구분 기호를 사용하여 struct 이름과 필드 이름을 구분할 수 있습니다.

테이블 정의와 쿼리 결과에 표시할 필드를 기반으로 열 이름과 struct 참조를 수정해야 합니다.

```
SELECT
  firewall_name,
  availability_zone,
  event_timestamp,
  event.timestamp AS event_timestamp,
  event.flow_id AS event_flow_id,
  event.event_type AS event_type,
  event.src_ip AS event_src_ip,
  event.src_port AS event_src_port,
  event.dest_ip AS event_dest_ip,
  event.dest_port AS event_dest_port,
  event.proto AS event_protol,
  event.app_proto AS event_app_proto,
  event.tls_inspected AS event_tls_inspected,
  event.alert.alert_id AS event_alert_alert_id,
  event.alert.alert_type AS event_alert_alert_type,
  event.alert.action AS event_alert_action,
  event.alert.signature_id AS event_alert_signature_id,
  event.alert.rev AS event_alert_rev,
  event.alert.signature AS event_alert_signature,
  event.alert.category AS event_alert_category,
  event.alert.severity AS event_alert_severity,
  event.alert.rule_name AS event_alert_rule_name,
  event.alert.alert_name AS event_alert_alert_name,
  event.alert.alert_severity AS event_alert_alert_severity,
  event.alert.alert_description AS event_alert_alert_description,
```

```
  event.alert.file_name AS event_alert_file_name,
  event.alert.file_hash AS event_alert_file_hash,
  event.alert.packet_capture AS event_alert_packet_capture,
  event.alert.reference_links AS event_alert_reference_links,
  event.src_country AS event_src_country,
  event.dest_country AS event_dest_country,
  event.src_hostname AS event_src_hostname,
  event.dest_hostname AS event_dest_hostname,
  event.user_agent AS event_user_agent,
  event.url AS event_url
FROM
  network_firewall_alert_logs
WHERE
  event.alert.severity >= 2
  AND event.tls_inspected = true
LIMIT 10;
```

넷플로우 로그용 테이블 생성

1. 다음 샘플 DDL 문을 넷플로우 로그의 구조에 맞게 수정하세요. 최신 버전 로그의 열을 포함하도록 문을 업데이트해야 할 수 있습니다. 자세한 내용은 AWS Network Firewall 개발자 안내서의 방화벽 로그의 내용을 참조하세요.

```
CREATE EXTERNAL TABLE network_firewall_netflow_logs (
  firewall_name string,
  availability_zone string,
  event_timestamp string,
  event struct<
    timestamp:string,
    flow_id:bigint,
    event_type:string,
    src_ip:string,
    src_port:int,
    dest_ip:string,
    dest_port:int,
    proto:string,
    app_proto:string,
    netflow:struct<
      pkts:int,
      bytes:bigint,
      start:string,
      `end`:string,
      age:int,
```

```
      min_ttl:int,
      max_ttl:int,
      tcp_flags:struct<
        syn:boolean,
        fin:boolean,
        rst:boolean,
        psh:boolean,
        ack:boolean,
        urg:boolean
      >,
      tls_inspected:boolean
    >
  >
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://DOC-EXAMPLE-BUCKET/path_to_netflow_logs_folder/';
```

2. LOCATION 절을 수정하여 Amazon S3의 로그 폴더를 지정합니다.
3. Athena 쿼리 편집기에서 CREATE TABLE 쿼리를 실행합니다. 쿼리가 완료된 후 Athena는 network_firewall_netflow_logs 테이블을 등록하여 쿼리 준비를 나타내는 데이터를 만듭니다.

넷플로우 로그 샘플 쿼리

이 섹션의 샘플 넷플로우 로그 쿼리는 TLS 검사가 수행된 이벤트를 필터링합니다.

쿼리는 별칭을 사용하여 해당 열이 속한 struct를 보여주는 출력 열 제목을 생성합니다. 예를 들어 event.netflow.bytes 필드의 열 제목은 event_netflow_bytes이며 그냥 bytes가 아닙니다. 열 이름을 추가로 사용자 지정하려면 원하는 대로 별칭을 수정할 수 있습니다. 예를 들어 밑줄이나 기타 구분 기호를 사용하여 struct 이름과 필드 이름을 구분할 수 있습니다.

테이블 정의와 쿼리 결과에 표시할 필드를 기반으로 열 이름과 struct 참조를 수정해야 합니다.

```
SELECT
  event.src_ip AS event_src_ip,
  event.dest_ip AS event_dest_ip,
  event.proto AS event_proto,
  event.app_proto AS event_app_proto,
  event.netflow.pkts AS event_netflow_pkts,
  event.netflow.bytes AS event_netflow_bytes,
  event.netflow.tcp_flags.syn AS event_netflow_tcp_flags_syn,
  event.netflow.tls_inspected AS event_netflow_tls_inspected
```

```
FROM network_firewall_netflow_logs
WHERE event.netflow.tls_inspected = true
```

Network Load Balancer 로그 쿼리

Athena를 사용하여 Network Load Balancer의 로그를 분석하고 처리합니다. 이러한 로그는 Network Load Balancer로 전송된 TLS(전송 계층 보안) 요청에 대한 세부 정보를 수신합니다. 이러한 액세스 로그를 사용하여 트래픽 패턴을 분석하고 문제를 해결할 수 있습니다.

Network Load Balancer 액세스 로그를 분석하기 전에 대상 Amazon S3 버킷에 저장하기 위한 로그를 활성화하고 구성합니다. 자세한 내용과 각 Network Load Balancer 액세스 로그 항목에 대한 정보는 Network Load Balancer의 액세스 로그를 참조하세요.

- Network Load Balancer 로그에 대한 테이블 생성
- Network Load Balancer 쿼리 예제

Network Load Balancer 로그에 대한 테이블을 생성하려면

1. 다음 DDL 문을 복사하여 Athena 콘솔에 붙여 넣습니다. Network Load Balancer 로그 레코드의 구문을 검사합니다. 최신 버전 레코드의 열과 Regex 구문을 포함하도록 다음 쿼리를 업데이트해야 할 수 있습니다.

```
CREATE EXTERNAL TABLE IF NOT EXISTS nlb_tls_logs (
            type string,
            version string,
            time string,
            elb string,
            listener_id string,
            client_ip string,
            client_port int,
            target_ip string,
            target_port int,
            tcp_connection_time_ms double,
            tls_handshake_time_ms double,
            received_bytes bigint,
            sent_bytes bigint,
            incoming_tls_alert int,
            cert_arn string,
            certificate_serial string,
            tls_cipher_suite string,
            tls_protocol_version string,
```

```
            tls_named_group string,
            domain_name string,
            alpn_fe_protocol string,
            alpn_be_protocol string,
            alpn_client_preference_list string,
            tls_connection_creation_time string
            )
            ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
            WITH SERDEPROPERTIES (
            'serialization.format' = '1',
            'input.regex' =
        '([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*):([0-9]*) ([^ ]*):([0-9]*)
 ([-.0-9]*) ([-.0-9]*) ([-0-9]*) ([-0-9]*) ([-0-9]*) ([^ ]*) ([^ ]*) ([^ ]*)
 ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*) ([^ ]*)$')
            LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/AWS_account_ID/
elasticloadbalancing/region';
```

2. LOCATION Amazon S3 버킷을 수정하여 Network Load Balancer 로그의 대상을 지정합니다.
3. Athena 콘솔에서 쿼리를 실행합니다. 쿼리가 완료된 후 Athena는 `nlb_tls_logs` 테이블을 등록하여 쿼리를 위한 데이터를 준비합니다.

Network Load Balancer 쿼리 예제

인증서가 사용되는 횟수를 확인하려면 이 예제와 비슷한 방식으로 쿼리를 사용합니다.

```
SELECT count(*) AS
         ct,
         cert_arn
FROM "nlb_tls_logs"
GROUP BY  cert_arn;
```

다음 쿼리는 1.3 이전의 TLS 버전을 사용하고 있는 사용자 수를 보여 줍니다.

```
SELECT tls_protocol_version,
         COUNT(tls_protocol_version) AS
         num_connections,
         client_ip
FROM "nlb_tls_logs"
WHERE tls_protocol_version < 'tlsv13'
GROUP BY tls_protocol_version, client_ip;
```

다음 쿼리를 사용하여 TLS 핸드셰이크 시간이 오래 걸리는 연결을 식별합니다.

```
SELECT *
FROM "nlb_tls_logs"
ORDER BY  tls_handshake_time_ms DESC
LIMIT 10;
```

다음 쿼리를 사용하여 지난 30일 동안 협상된 TLS 프로토콜 버전 및 암호 제품군을 식별하고 계산합니다.

```
SELECT tls_cipher_suite,
         tls_protocol_version,
         COUNT(*) AS ct
FROM "nlb_tls_logs"
WHERE from_iso8601_timestamp(time) > current_timestamp - interval '30' day
        AND NOT tls_protocol_version = '-'
GROUP BY tls_cipher_suite, tls_protocol_version
ORDER BY ct DESC;
```

Amazon Route 53 Resolver 쿼리 로그의 쿼리

Amazon Route 53 Resolver 쿼리 로그에 대한 Athena 테이블을 생성하고 Athena에서 쿼리할 수 있습니다.

Route 53 Resolver 쿼리 로깅은 인바운드 해석기(resolver) 엔드포인트를 사용하는 온프레미스 리소스인 VPC 내의 리소스에 의해 수행된 DNS 쿼리, 재귀적 DNS 확인(recursive DNS resolution)을 위해 아웃바운드 해석기 엔드포인트를 사용하는 쿼리, 그리고 도메인 목록을 차단, 허용 또는 모니터링하기 위해 Route 53 Resolver DNS 방화벽 규칙을 사용하는 쿼리를 로그하기 위한 것입니다. Resolver 쿼리 로깅에 대한 자세한 내용은 Amazon Route 53 개발자 안내서의 Resolver 쿼리 로깅을 참조하세요. 로그의 각 필드에 대한 자세한 내용은 Amazon Route 53 개발자 안내서의 Resolver 쿼리 로그에 표시되는 값을 참조하세요.

Resolver 쿼리 로그에 대한 테이블 생성

Athena 콘솔에서 쿼리 편집기를 사용하여 Route 53 Resolver 쿼리 로그에 대한 테이블을 만들고 쿼리할 수 있습니다.

Route 53 Resolver 쿼리 로그에 대해 Athena 테이블을 만들고 쿼리하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. Athena 쿼리 편집기에 다음 `CREATE TABLE` 문을 입력합니다. `LOCATION` 절의 값을 Amazon S3의 Resolver 로그 위치에 상응하는 값으로 바꿉니다.

```
CREATE EXTERNAL TABLE r53_rlogs (
  version string,
  account_id string,
  region string,
  vpc_id string,
  query_timestamp string,
  query_name string,
  query_type string,
  query_class
    string,
  rcode string,
  answers array<
    struct<
      Rdata: string,
      Type: string,
      Class: string>
    >,
  srcaddr string,
  srcport int,
  transport string,
  srcids struct<
    instance: string,
    resolver_endpoint: string
    >,
  firewall_rule_action string,
  firewall_rule_group_id string,
  firewall_domain_list_id string
 )

ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://DOC-EXAMPLE-BUCKET/AWSLogs/aws_account_id/vpcdnsquerylogs/{vpc-id}/'
```

Resolver 쿼리 로그 데이터는 JSON 형식이기 때문에 CREATE TABLE 문은 JSON SerDe 라이브러리를 사용하여 데이터를 분석합니다.

> **Note**
>
> SerDe는 각 JSON 문서가 레코드의 필드를 구분하는 줄 종료 문자가 없는 한 줄의 텍스트에 있을 것으로 예상합니다. JSON 텍스트가 가독성 좋게 꾸민 형식이면 테이블을 만든 후 쿼리하려고 할 때 HIVE_CURSOR_ERROR: 행이 유효한 JSON 객체가 아님(HIVE_CURSOR_ERROR: Row is not a valid JSON Object) 또는

HIVE_CURSOR_ERROR: JsonParseException: 예기치 않은 입력 종료: OBJECT의 닫기 마커 필요(HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT) 같은 오류 메시지가 나타날 수 있습니다. 자세한 내용은 GitHub의 OpenX SerDe 문서에서 JSON 데이터 파일을 참조하세요.

3. 쿼리 실행을 선택합니다. 이 문은 `r53_rlogs`라는 Athena 테이블을 만듭니다. 이 테이블의 열은 Resolver 로그 데이터의 각 필드를 나타냅니다.
4. Athena 콘솔 쿼리 편집기에서 다음 쿼리를 실행하여 테이블이 생성되었는지 확인합니다.

```
SELECT * FROM "r53_rlogs" LIMIT 10
```

파티셔닝 예제

다음 예제는 파티션 프로젝션을 사용하고 vpc와 날짜를 기준으로 파티셔닝된 Resolver 쿼리 로그에 대한 `CREATE TABLE` 문을 보여줍니다. 파티션 프로젝션에 대한 자세한 내용은 Amazon Athena를 사용한 파티션 프로젝션 단원을 참조하세요.

```
CREATE EXTERNAL TABLE r53_rlogs (
  version string,
  account_id string,
  region string,
  vpc_id string,
  query_timestamp string,
  query_name string,
  query_type string,
  query_class string,
  rcode string,
  answers array<
    struct<
      Rdata: string,
      Type: string,
      Class: string>
    >,
  srcaddr string,
  srcport int,
  transport string,
  srcids struct<
    instance: string,
    resolver_endpoint: string
    >,
```

```
  firewall_rule_action string,
  firewall_rule_group_id string,
  firewall_domain_list_id string
)
PARTITIONED BY (
`date` string,
`vpc` string
)
ROW FORMAT SERDE      'org.openx.data.jsonserde.JsonSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT          'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION              's3://DOC-EXAMPLE-BUCKET/route53-query-logging/
AWSLogs/aws_account_id/vpcdnsquerylogs/'
TBLPROPERTIES(
'projection.enabled' = 'true',
'projection.vpc.type' = 'enum',
'projection.vpc.values' = 'vpc-6446ae02',
'projection.date.type' = 'date',
'projection.date.range' = '2023/06/26,NOW',
'projection.date.format' = 'yyyy/MM/dd',
'projection.date.interval' = '1',
'projection.date.interval.unit' = 'DAYS',
'storage.location.template' = 's3://DOC-EXAMPLE-BUCKET/route53-query-logging/
AWSLogs/aws_account_id/vpcdnsquerylogs/${vpc}/${date}/'
)
```

쿼리 예제

다음 예제는 Athena에서 Resolver 쿼리 로그에 대해 수행할 수 있는 몇 가지 쿼리를 보여 줍니다.

예제 1 - query_timestamp의 내림차순으로 로그 쿼리

다음 쿼리는 `query_timestamp`의 내림차순으로 로그 결과를 표시합니다.

```
SELECT * FROM "r53_rlogs"
ORDER BY query_timestamp DESC
```

예제 2 - 지정된 시작 및 종료 시간 내에 로그 쿼리

다음 쿼리는 2020년 9월 24일 자정과 오전 8시 사이의 로그를 쿼리합니다. 자신의 필요에 따라 시작 시간과 종료 시간을 대체합니다.

```
SELECT query_timestamp, srcids.instance, srcaddr, srcport, query_name, rcode
```

```
FROM "r53_rlogs"
WHERE (parse_datetime(query_timestamp,'yyyy-MM-dd''T''HH:mm:ss''Z')
     BETWEEN parse_datetime('2020-09-24-00:00:00','yyyy-MM-dd-HH:mm:ss')
     AND parse_datetime('2020-09-24-00:08:00','yyyy-MM-dd-HH:mm:ss'))
ORDER BY query_timestamp DESC
```

예제 3 - 지정된 DNS 쿼리 이름 패턴을 기반으로 로그 쿼리

다음 쿼리는 쿼리 이름에 “example.com” 문자열을 포함한 레코드를 선택합니다.

```
SELECT query_timestamp, srcids.instance, srcaddr, srcport, query_name, rcode, answers
FROM "r53_rlogs"
WHERE query_name LIKE '%example.com%'
ORDER BY query_timestamp DESC
```

예제 4 - 응답 없는 요청 로그 쿼리

다음 쿼리는 응답이 없는 요청의 로그 항목을 선택합니다.

```
SELECT query_timestamp, srcids.instance, srcaddr, srcport, query_name, rcode, answers
FROM "r53_rlogs"
WHERE cardinality(answers) = 0
```

예제 5 - 특정 답변이 포함된 로그 쿼리

다음 쿼리는 `answer.Rdata` 값에 특정 IP 주소가 있는 로그를 보여 줍니다.

```
SELECT query_timestamp, srcids.instance, srcaddr, srcport, query_name, rcode,
 answer.Rdata
FROM "r53_rlogs"
CROSS JOIN UNNEST(r53_rlogs.answers) as st(answer)
WHERE answer.Rdata='203.0.113.16';
```

Amazon SES 이벤트 로그 쿼리

Amazon Athena를 사용하여 Amazon Simple Email Service(Amazon SES) 이벤트 로그를 쿼리할 수 있습니다.

Amazon SES는 사용자의 이메일 주소와 도메인을 사용해 이메일을 보내고 받기 위한 경제적이고 편리한 방법을 제공하는 이메일 플랫폼입니다. 이벤트, 지표, 통계를 사용하여 세분화된 수준으로 Amazon SES 전송 활동을 모니터링할 수 있습니다.

정의한 특성에 따라 Amazon CloudWatch, Amazon Data Firehose 또는 Amazon Simple Notification Service에 Amazon SES 이벤트를 게시할 수 있습니다. Amazon S3에 정보를 저장한 이후 Amazon Athena Athena에서 쿼리할 수 있습니다.

Firehose, Amazon Athena, Amazon QuickSight를 사용하여 Amazon SES 이메일 이벤트를 분석하는 방법에 대한 자세한 내용은 AWS 메시징 및 타겟팅 블로그의 AWS 분석 서비스를 이용한 Amazon SES 이벤트 데이터 분석 섹션을 참조하세요.

Amazon VPC 흐름 로그 쿼리

Amazon Virtual Private Cloud 흐름 로그는 VPC의 네트워크 인터페이스에서 송수신되는 IP 트래픽에 대한 정보를 수집합니다. 로그를 사용하여 네트워크 트래픽 패턴을 조사하고 VPC 네트워크에서 위협 및 위험을 식별하세요.

Amazon VPC 흐름 로그를 쿼리하려는 경우 다음 두 가지 옵션이 있습니다.

- Amazon VPC 콘솔 - Amazon VPC 콘솔의 Athena 통합 기능을 사용하여 분할로 Athena 데이터베이스, 작업 그룹 및 흐름 로그 테이블을 생성하는 AWS CloudFormation 템플릿을 생성합니다. 또한 템플릿에서는 VPC 통해 흐르는 트래픽에 대한 인사이트를 얻기 위해 사용할 수 있는 미리 정의된 흐름 로그 쿼리 집합을 생성할 수 있습니다.

  자세한 내용은 Amazon VPC 사용 설명서의 Amazon Athena를 사용하여 흐름 로그 쿼리를 참조하세요.
- Amazon Athena 콘솔 - Athena 콘솔에서 직접 테이블과 쿼리를 생성합니다. 자세히 알아보려면 이 페이지를 계속 읽어 보세요.

사용자 지정 VPC 흐름 로그에 대한 테이블 생성 및 쿼리

Athena에서 로그 쿼리를 시작하기 전에 VPC 흐름 로그를 활성화하고 Amazon S3 버킷에 저장하도록 구성합니다. 로그를 생성한 후 몇 분 동안 실행하여 일부 데이터를 수집합니다. 로그는 GZIP 압축 형식으로 생성되어 Athena를 이용해 직접 쿼리할 수 있습니다.

VPC 흐름 로그를 생성하는 경우 흐름 로그에서 반환할 필드와 해당 필드가 나타나는 순서를 지정할 때 사용자 지정 형식을 사용할 수 있습니다. 흐롬 로그 레코드에 대한 자세한 내용은 Amazon VPC 사용 설명서의 흐름 로그 레코드를 참조하세요.

일반적인 고려 사항

Athena에서 Amazon VPC 흐름 로그용 테이블을 생성할 때 다음 사항을 기억하세요.

- 기본적으로 Athena에서 Parquet은 이름으로 열에 액세스합니다. 자세한 내용은 스키마 업데이트 처리 단원을 참조하십시오.
- Athena의 열 이름에 대한 흐름 로그 레코드의 이름을 사용합니다. Athena 스키마의 열 이름은 Amazon VPC 흐름 로그의 필드 이름과 정확히 일치해야 하며 다음과 같은 차이점이 있습니다.
  - Amazon VPC 로그 필드 이름의 하이픈을 Athena 열 이름의 밑줄로 바꿉니다. Athena에서 데이터베이스 이름, 테이블 이름 및 열 이름에 사용할 수 있는 유일한 문자는 소문자, 숫자 및 밑줄 문자입니다. 자세한 내용은 데이터베이스, 테이블 및 열 이름 단원을 참조하십시오.
  - Athena의 예약 키워드인 흐름 로그 레코드 이름을 백틱(backtick)으로 둘러싸서 이스케이프합니다.
- VPC 흐름 로그는 AWS 계정에 특정적입니다. 로그 파일을 Amazon S3에 게시할 때 Amazon VPC가 Amazon S3에 생성하는 경로에는 흐름 로그를 생성하는 데 사용된 AWS 계정 ID가 포함됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 Amazon S3에 흐름 로그 게시를 참조하세요.

Amazon VPC 흐름 로그에 대한 CREATE TABLE 문

다음은 VPC 흐름 로그에 대한 Amazon VPC 테이블을 생성하는 절차입니다. 사용자 지정 형식을 사용하여 흐름 로그를 생성하는 경우 흐름 로그를 생성할 때 지정한 필드와 일치하는 필드가 해당 필드를 지정할 때와 동일한 순서로 포함되는 테이블을 생성합니다.

Amazon VPC 흐름 로그에 대한 Athena 테이블 생성

1. 일반적인 고려 사항 섹션의 다음 지침에 따라 Athena 콘솔 쿼리 편집기에 다음과 같은 DDL 문을 입력합니다. 다음 문 샘플은 흐름 로그 레코드에 설명된 대로 Amazon VPC 흐름 로그 버전 2 ~ 5의 열을 갖는 테이블을 생성합니다. 다른 열 집합이나 열 순서를 사용하는 경우 그에 맞게 문을 수정합니다.

```
CREATE EXTERNAL TABLE IF NOT EXISTS `vpc_flow_logs` (
  version int,
  account_id string,
  interface_id string,
  srcaddr string,
  dstaddr string,
  srcport int,
  dstport int,
  protocol bigint,
  packets bigint,
  bytes bigint,
  start bigint,
  `end` bigint,
```

```
  action string,
  log_status string,
  vpc_id string,
  subnet_id string,
  instance_id string,
  tcp_flags int,
  type string,
  pkt_srcaddr string,
  pkt_dstaddr string,
  region string,
  az_id string,
  sublocation_type string,
  sublocation_id string,
  pkt_src_aws_service string,
  pkt_dst_aws_service string,
  flow_direction string,
  traffic_path int
)
PARTITIONED BY (`date` date)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
LOCATION 's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/{account_id}/
vpcflowlogs/{region_code}/'
TBLPROPERTIES ("skip.header.line.count"="1");
```

다음 사항에 주의하세요.

- 이 쿼리는 `ROW FORMAT DELIMITED`를 지정하며 SerDe 지정은 생략합니다. 즉, 쿼리는 CSV, TSV, 사용자 지정 구분 기호로 구분된 파일에 대한 LazySimpleSerDe을 사용합니다. 이 쿼리에서 필드는 공백으로 끝납니다.
- `PARTITIONED BY` 절은 `date` 유형을 사용합니다. 따라서 쿼리에서 수학 연산자를 사용하여 특정 날짜보다 오래되었거나 최신인 항목을 선택할 수 있습니다.

> **Note**
>
> `date`는 DDL 문에 예약된 키워드이므로 백틱 문자로 이스케이프됩니다. 자세한 내용은 예약어 단원을 참조하십시오.

- 다른 사용자 지정 형식을 사용하는 VPC 흐름 로그의 경우 흐름 로그를 생성할 때 지정한 필드와 일치하도록 필드를 수정합니다.

2. 로그 데이터가 포함된 Amazon S3 버킷을 가리키도록 LOCATION 's3://DOC-EXAMPLE-BUCKET/*prefix*/AWSLogs/*{account_id}*/vpcflowlogs/*{region_code}*/'을 수정합니다.
3. Athena 콘솔에서 쿼리를 실행합니다. 쿼리가 완료된 후 Athena는 vpc_flow_logs 테이블을 등록하여 쿼리를 실행할 수 있도록 데이터를 준비합니다.
4. 다음 샘플 쿼리와 같이 데이터를 읽을 수 있도록 파티션을 생성합니다. 이 쿼리는 지정한 날짜에 대한 단일 파티션을 생성합니다. 필요하다면 날짜와 위치의 자리 표시자를 교체하세요.

> **Note**
>
> 이 쿼리는 사용자가 지정한 날짜에 대한 단일 파티션만 생성합니다. 프로세스를 자동화하려면 이 쿼리를 실행하고 year/month/day에 대해 이 방법으로 파티션을 생성하는 스크립트를 사용하거나 파티션 프로젝션을 지정하는 CREATE TABLE 문을 사용합니다.

```
ALTER TABLE vpc_flow_logs
ADD PARTITION (`date`='YYYY-MM-dd')
LOCATION 's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/{account_id}/
vpcflowlogs/{region_code}/YYYY/MM/dd';
```

vpc_flow_logs 테이블에 대한 쿼리 예

Athena 콘솔의 쿼리 편집기를 사용하여 생성한 테이블에서 SQL 문을 실행합니다. 쿼리를 저장하거나 이전 쿼리를 보거나 쿼리 결과를 CSV 형식으로 다운로드할 수 있습니다. 다음 예에서는 vpc_flow_logs를 테이블의 이름으로 바꿉니다. 사용자의 요구 사항에 따라 열 값과 기타 변수를 수정합니다.

다음 예제 쿼리는 지정된 날짜에 대해 최대 100개의 흐름 로그를 나열합니다.

```
SELECT *
FROM vpc_flow_logs
WHERE date = DATE('2020-05-04')
LIMIT 100;
```

다음 쿼리는 거절된 모든 TCP 연결을 나열하며 새로 생성한 날짜 파티션 열인 date를 이용해 이러한 이벤트가 발생한 요일을 추출합니다.

```
SELECT day_of_week(date) AS
```

```
  day,
  date,
  interface_id,
  srcaddr,
  action,
  protocol
FROM vpc_flow_logs
WHERE action = 'REJECT' AND protocol = 6
LIMIT 100;
```

어떤 서버가 가장 많은 수의 HTTPS 요청을 수신하는지 알아보려면 다음 쿼리를 사용합니다. 이 쿼리는 지난 주부터 HTTPS 포트 443에서 수신한 패킷 수를 계산하고, 대상 IP 주소별로 그룹화한 다음, 상위 10개를 반환합니다.

```
SELECT SUM(packets) AS
  packetcount,
  dstaddr
FROM vpc_flow_logs
WHERE dstport = 443 AND date > current_date - interval '7' day
GROUP BY dstaddr
ORDER BY packetcount DESC
LIMIT 10;
```

Apache Parquet 형식으로 흐름 로그에 대한 테이블 생성

다음은 Apache Parquet 형식으로 Amazon VPC 흐름 로그에 대한 Amazon VPC 테이블을 생성하는 절차입니다.

Parquet 형식으로 Amazon VPC 흐름 로그에 대한 Athena 테이블 생성

1. 일반적인 고려 사항 섹션의 다음 지침에 따라 Athena 콘솔 쿼리 편집기에 다음과 같은 DDL 문을 입력합니다. 다음 샘플 문은 흐름 로그 레코드에 설명된 대로 Parquet 형식으로 Amazon VPC 흐름 로그 버전 2 ~ 5의 열을 포함하는 테이블을 만들며 Hive는 매시간 분할됩니다. 시간당 파티션이 없는 경우 hour 절에서 PARTITIONED BY를 제거합니다.

   ```
   CREATE EXTERNAL TABLE IF NOT EXISTS vpc_flow_logs_parquet (
     version int,
     account_id string,
     interface_id string,
     srcaddr string,
     dstaddr string,
     srcport int,
   ```

```
  dstport int,
  protocol bigint,
  packets bigint,
  bytes bigint,
  start bigint,
  `end` bigint,
  action string,
  log_status string,
  vpc_id string,
  subnet_id string,
  instance_id string,
  tcp_flags int,
  type string,
  pkt_srcaddr string,
  pkt_dstaddr string,
  region string,
  az_id string,
  sublocation_type string,
  sublocation_id string,
  pkt_src_aws_service string,
  pkt_dst_aws_service string,
  flow_direction string,
  traffic_path int
)
PARTITIONED BY (
  `aws-account-id` string,
  `aws-service` string,
  `aws-region` string,
  `year` string,
  `month` string,
  `day` string,
  `hour` string
)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/'
TBLPROPERTIES (
  'EXTERNAL'='true',
  'skip.header.line.count'='1'
```

```
)
```

2. 로그 데이터가 포함된 Amazon S3 경로를 가리키도록 샘플 `LOCATION 's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/'`을 수정합니다.
3. Athena 콘솔에서 쿼리를 실행합니다.
4. 데이터가 Hive 호환 형식인 경우 Athena 콘솔에서 다음 명령을 실행하여 메타스토어의 Hive 파티션을 업데이트 및 로드합니다. 쿼리가 완료되면 `vpc_flow_logs_parquet` 테이블에서 데이터를 쿼리할 수 있습니다.

   ```
   MSCK REPAIR TABLE vpc_flow_logs_parquet
   ```

   Hive 호환 데이터를 사용하지 않는 경우 ALTER TABLE ADD PARTITION을 실행하여 파티션을 로드합니다.

Athena를 사용하여 Parquet 형식으로 Amazon VPC 흐름 로그를 쿼리하는 방법에 대한 자세한 내용은 AWS Big Data Blog의 Optimize performance and reduce costs for network analytics with VPC Flow Logs in Apache Parquet format을 참조하세요.

파티션 프로젝션을 사용하여 Amazon VPC 흐름 로그에 대한 테이블 생성 및 쿼리

다음과 같이 `CREATE TABLE`을 사용하여 테이블을 생성하고, 테이블을 분할하고, 파티션 프로젝션을 사용하여 자동으로 파티션을 채웁니다. 예에서 테이블 이름 `test_table_vpclogs`를 테이블 이름으로 바꿉니다. `LOCATION` 절을 편집하여 Amazon VPC 로그 데이터가 포함된 Amazon S3 버킷을 지정합니다.

다음 `CREATE TABLE` 문은 Hive 스타일이 아닌 분할 형식으로 전달되는 VPC 흐름 로그입니다. 이 예제에서는 다중 계정 집계를 허용합니다. 여러 계정의 VPC 흐름 로그를 하나의 Amazon S3 버킷으로 중앙 집중화하려는 경우 Amazon S3 경로에 계정 ID를 입력해야 합니다.

```
CREATE EXTERNAL TABLE IF NOT EXISTS test_table_vpclogs (
  version int,
  account_id string,
  interface_id string,
  srcaddr string,
  dstaddr string,
  srcport int,
  dstport int,
  protocol bigint,
  packets bigint,
  bytes bigint,
```

```
  start bigint,
  `end` bigint,
  action string,
  log_status string,
  vpc_id string,
  subnet_id string,
  instance_id string,
  tcp_flags int,
  type string,
  pkt_srcaddr string,
  pkt_dstaddr string,
  az_id string,
  sublocation_type string,
  sublocation_id string,
  pkt_src_aws_service string,
  pkt_dst_aws_service string,
  flow_direction string,
  traffic_path int
)
PARTITIONED BY (accid string, region string, day string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ' '
LOCATION '$LOCATION_OF_LOGS'
TBLPROPERTIES
(
"skip.header.line.count"="1",
"projection.enabled" = "true",
"projection.accid.type" = "enum",
"projection.accid.values" = "$ACCID_1,$ACCID_2",
"projection.region.type" = "enum",
"projection.region.values" = "$REGION_1,$REGION_2,$REGION_3",
"projection.day.type" = "date",
"projection.day.range" = "$START_RANGE,NOW",
"projection.day.format" = "yyyy/MM/dd",
"storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/AWSLogs/${accid}/vpcflowlogs/
${region}/${day}"
)
```

test_table_vpclogs에 대한 쿼리 예

다음 예제 쿼리는 선행하는 CREATE TABLE 문에 의해 생성된 test_table_vpclogs를 쿼리합니다. 쿼리에서 test_table_vpclogs를 자체 테이블 이름으로 바꿉니다. 사용자의 요구 사항에 따라 열 값과 기타 변수를 수정합니다.

지정된 기간 동안 처음 100개의 액세스 로그 항목을 시간순으로 반환하려면 다음과 같은 쿼리를 실행합니다.

```
SELECT *
FROM test_table_vpclogs
WHERE day >= '2021/02/01' AND day < '2021/02/28'
ORDER BY day ASC
LIMIT 100
```

지정된 기간 동안 상위 10개 HTTP 패킷을 수신하는 서버를 보려면 다음과 같은 쿼리를 실행합니다. 이 쿼리는 지난 주부터 HTTPS 포트 443에서 수신한 패킷 수를 계산하고, 대상 IP 주소별로 그룹화한 다음, 상위 10개를 반환합니다.

```
SELECT SUM(packets) AS packetcount,
       dstaddr
FROM test_table_vpclogs
WHERE dstport = 443
  AND day >= '2021/03/01'
  AND day < '2021/03/31'
GROUP BY dstaddr
ORDER BY packetcount DESC
LIMIT 10
```

지정된 기간 동안 생성된 로그를 반환하려면 다음과 같은 쿼리를 실행합니다.

```
SELECT interface_id,
       srcaddr,
       action,
       protocol,
       to_iso8601(from_unixtime(start)) AS start_time,
       to_iso8601(from_unixtime("end")) AS end_time
FROM test_table_vpclogs
WHERE DAY >= '2021/04/01'
  AND DAY < '2021/04/30'
```

지정된 기간 사이에 소스 IP 주소에 대한 액세스 로그를 반환하려면 다음과 같은 쿼리를 실행합니다.

```
SELECT *
FROM test_table_vpclogs
WHERE srcaddr = '10.117.1.22'
  AND day >= '2021/02/01'
```

```
  AND day < '2021/02/28'
```

거부된 TCP 연결을 나열하려면 다음과 같은 쿼리를 실행합니다.

```
SELECT day,
       interface_id,
       srcaddr,
       action,
       protocol
FROM test_table_vpclogs
WHERE action = 'REJECT' AND protocol = 6 AND day >= '2021/02/01' AND day < '2021/02/28'
LIMIT 10
```

10.117로 시작하는 IP 주소 범위에 대한 액세스 로그를 반환하려면 다음과 같은 쿼리를 실행합니다.

```
SELECT *
FROM test_table_vpclogs
WHERE split_part(srcaddr,'.', 1)='10'
  AND split_part(srcaddr,'.', 2) ='117'
```

지정된 기간 사이에 대상 IP 주소에 대한 액세스 로그를 반환하려면 다음과 같은 쿼리를 실행합니다.

```
SELECT *
FROM test_table_vpclogs
WHERE dstaddr = '10.0.1.14'
  AND day >= '2021/01/01'
  AND day < '2021/01/31'
```

파티션 프로젝션을 사용하여 Apache Parquet 형식으로 흐름 로그에 대한 테이블 생성

VPC 흐름 로그에 대한 다음 파티션 프로젝션 CREATE TABLE 문은 Apache Parquet 형식이고 Hive와 호환되지 않으며 일 대신 시간 및 날짜를 기준으로 파티셔닝됩니다. 예에서 테이블 이름 `test_table_vpclogs_parquet`를 테이블 이름으로 바꿉니다. `LOCATION` 절을 편집하여 Amazon VPC 로그 데이터가 포함된 Amazon S3 버킷을 지정합니다.

```
CREATE EXTERNAL TABLE IF NOT EXISTS test_table_vpclogs_parquet (
  version int,
  account_id string,
  interface_id string,
  srcaddr string,
  dstaddr string,
  srcport int,
```

```
  dstport int,
  protocol bigint,
  packets bigint,
  bytes bigint,
  start bigint,
  `end` bigint,
  action string,
  log_status string,
  vpc_id string,
  subnet_id string,
  instance_id string,
  tcp_flags int,
  type string,
  pkt_srcaddr string,
  pkt_dstaddr string,
  az_id string,
  sublocation_type string,
  sublocation_id string,
  pkt_src_aws_service string,
  pkt_dst_aws_service string,
  flow_direction string,
  traffic_path int
)
PARTITIONED BY (region string, date string, hour string)
ROW FORMAT SERDE
'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/{account_id}/vpcflowlogs/'
TBLPROPERTIES (
"EXTERNAL"="true",
"skip.header.line.count" = "1",
"projection.enabled" = "true",
"projection.region.type" = "enum",
"projection.region.values" = "us-east-1,us-west-2,ap-south-1,eu-west-1",
"projection.date.type" = "date",
"projection.date.range" = "2021/01/01,NOW",
"projection.date.format" = "yyyy/MM/dd",
"projection.hour.type" = "integer",
"projection.hour.range" = "00,23",
"projection.hour.digits" = "2",
```

```
"storage.location.template" = "s3://DOC-EXAMPLE-BUCKET/prefix/AWSLogs/${account_id}/
vpcflowlogs/${region}/${date}/${hour}"
)
```

추가적인 리소스

Athena를 사용하여 VPC 흐름 로그를 분석하는 방법에 대한 자세한 내용은 다음 AWS 빅 데이터 블로그 게시물을 참조하세요.

- 포인트 앤 클릭 방식 Amazon Athena 통합을 통한 VPC 흐름 로그 분석
- Amazon Athena 및 Amazon QuickSight를 사용한 VPC 흐름 로그 분석
- Apache Parquet 형식의 VPC 흐름 로그를 사용하여 네트워크 분석 성능을 최적화하고 비용을 절감합니다.

AWS WAF 로그 쿼리

AWS WAF는 보호되는 웹 애플리케이션이 클라이언트로부터 수신하는 HTTP 및 HTTPS 요청을 모니터링하고 제어할 수 있는 웹 애플리케이션 방화벽입니다. AWS WAF WAF 웹 액세스 제어 목록(ACL) 내에서 규칙을 구성하여 웹 요청을 처리하는 방법을 정의합니다. 그런 다음 웹 ACL을 연결하여 웹 애플리케이션을 보호합니다. AWS WAF로 보호할 수 있는 웹 애플리케이션 리소스의 예로는 Amazon CloudFront 배포, Amazon API Gateway REST API 및 Application Load Balancer가 있습니다. AWS WAF에 대한 자세한 내용은 AWS WAF 개발자 안내서의 AWS WAF 섹션을 참조하세요.

AWS WAF 로그에는 AWS WAF가 AWS 리소스로부터 요청을 받은 시간, 요청에 대한 세부 정보 및 각 요청이 부합되는 규칙에 대한 작업과 같이 웹 ACL에 의해 분석되는 트래픽에 대한 정보가 포함됩니다.

로그를 쿼리하고 볼 수 있는 여러 대상 중 하나에 로그를 게시하도록 AWS WAF 웹 ACL을 구성할 수 있습니다. 웹 ACL 로깅 및 AWS WAF 로그 내용 구성에 대한 자세한 내용은 AWS WAF 개발자 안내서의 Logging AWS WAF web ACL traffic을 참조하세요.

AWS WAF 로그를 중앙 데이터 레이크 리포지토리에 집계하고 <shared id="ATE"/>로 쿼리하는 방법의 예시는 AWS Big Data Blog 게시물 Analyzing AWS WAF logs with OpenSearch Service, Amazon Athena and Amazon QuickSight를 참조하세요.

이 주제에서는 분할을 사용하는 것과 사용하지 않는 두 가지 `CREATE TABLE` 문을 예제로 제공합니다.

> **Note**
>
> 이 주제의 `CREATE TABLE` 문은 v1 및 v2 AWS WAF 로그에 모두 사용할 수 있습니다. v1에서는 `webaclid` 필드에 ID가 포함되어 있습니다. v2에서는 `webaclid` 필드에 전체 ARN이 포함

되어 있습니다. 이곳의 CREATE TABLE 문은 string 데이터 유형을 사용하여 애그노스틱 방식으로 이 콘텐츠를 취급합니다.

주제

- 파티션 프로젝션을 사용하여 Athena에서 AWS WAF S3 로그에 대한 테이블 생성
- 분할 없이 AWS WAF 로그의 테이블 생성
- AWS WAF 로그에 대한 쿼리 예제

파티션 프로젝션을 사용하여 Athena에서 AWS WAF S3 로그에 대한 테이블 생성

AWS WAF 로그에는 미리 지정할 수 있는 파티션 스키마를 가진 알려진 구조가 있기 때문에 Athena 파티션 프로젝션 기능을 사용하여 쿼리 런타임을 줄이고 파티션 관리를 자동화할 수 있습니다. 새 데이터가 추가되면 파티션 프로젝션은 자동으로 새 파티션을 추가합니다. 따라서 ALTER TABLE ADD PARTITION을 사용해 파티션을 수동으로 추가할 필요가 없습니다.

다음 CREATE TABLE 문 예제에서는 네 개의 각기 다른 AWS 리전에 대해 지정된 날짜부터 현재까지의 AWS WAF 로그에 대해 파티션 프로젝션을 자동으로 사용합니다. 이 예제의 PARTITION BY 절은 리전 및 날짜별로 분할되지만 요구 사항에 따라 수정할 수 있습니다. 필요에 따라 필드를 로그 출력과 일치하도록 수정합니다. LOCATION 및 storage.location.template 절에서 *bucket* 및 *accountID* 자리 표시자를 AWS WAF 로그의 Amazon S3 버킷 위치를 식별하는 값으로 바꿉니다. projection.day.range에 대해 *2021*/*01*/*01*을 사용하려는 시작 날짜로 바꿉니다. 쿼리가 성공적으로 실행되면 테이블을 쿼리할 수 있습니다. 파티션을 로드하기 위해 ALTER TABLE ADD PARTITION을 실행하지 않아도 됩니다.

```
CREATE EXTERNAL TABLE `waf_logs`(
  `timestamp` bigint,
  `formatversion` int,
  `webaclid` string,
  `terminatingruleid` string,
  `terminatingruletype` string,
  `action` string,
  `terminatingrulematchdetails` array <
                                    struct <
                                        conditiontype: string,
                                        sensitivitylevel: string,
                                        location: string,
                                        matcheddata: array < string >
                                          >
```

```
                                    >,
  `httpsourcename` string,
  `httpsourceid` string,
  `rulegrouplist` array <
                      struct <
                          rulegroupid: string,
                          terminatingrule: struct <
                                              ruleid: string,
                                              action: string,
                                              rulematchdetails: array <
                                                                  struct <
                                                                      conditiontype:
string,

sensitivitylevel: string,
                                                                      location:
string,
                                                                      matchedda
array < string >
                                                                         >
                                                                  >
                                           >,
                          nonterminatingmatchingrules: array <
                                                               struct <
                                                                   ruleid: string,
                                                                   action: string,
                                                                   overriddenaction:
string,
                                                                   rulematchdetails:
array <

struct <

   conditiontype: string,

   sensitivitylevel: string,

   location: string,

   matcheddata: array < string >

      >
                                                                  >,
```

```
                                                                challengeresponse:
 struct <

 responsecode: string,

 solvetimestamp: string
                                                                              >,
                                                                captcharesponse:
 struct <

 responsecode: string,

 solvetimestamp: string
                                                                              >
                                                                        >
                                                                >,
                          excludedrules: string
                            >
                       >,
`ratebasedrulelist` array <
                         struct <
                             ratebasedruleid: string,
                             limitkey: string,
                             maxrateallowed: int
                               >
                          >,
  `nonterminatingmatchingrules` array <
                                    struct <
                                        ruleid: string,
                                        action: string,
                                        rulematchdetails: array <
                                                             struct <
                                                                 conditiontype: string,
                                                                 sensitivitylevel:
 string,
                                                                 location: string,
                                                                 matcheddata: array <
 string >
                                                                    >
                                                             >,
                                        challengeresponse: struct <
                                                            responsecode: string,
                                                            solvetimestamp: string
                                                             >,
```

```
                                    captcharesponse: struct <
                                                        responsecode: string,
                                                        solvetimestamp: string
                                                         >
                                        >
                                    >,
`requestheadersinserted` array <
                                struct <
                                    name: string,
                                    value: string
                                      >
                                 >,
`responsecodesent` string,
`httprequest` struct <
                  clientip: string,
                  country: string,
                  headers: array <
                              struct <
                                  name: string,
                                  value: string
                                    >
                               >,
                  uri: string,
                  args: string,
                  httpversion: string,
                  httpmethod: string,
                  requestid: string
                    >,
`labels` array <
             struct <
                 name: string
                   >
              >,
`captcharesponse` struct <
                      responsecode: string,
                      solvetimestamp: string,
                      failureReason: string
                        >,
`challengeresponse` struct <
                      responsecode: string,
                      solvetimestamp: string,
                      failureReason: string
                      >,
`ja3Fingerprint` string,
```

```
  `oversizefields` string,
  `requestbodysize` int,
  `requestbodysizeinspectedbywaf` int
)
PARTITIONED BY (
`region` string,
`date` string)
ROW FORMAT SERDE
  'org.openx.data.jsonserde.JsonSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/AWSLogs/accountID/WAFLogs/region/DOC-EXAMPLE-WEBACL/'
TBLPROPERTIES(
 'projection.enabled' = 'true',
 'projection.region.type' = 'enum',
 'projection.region.values' = 'us-east-1,us-west-2,eu-central-1,eu-west-1',
 'projection.date.type' = 'date',
 'projection.date.range' = '2021/01/01,NOW',
 'projection.date.format' = 'yyyy/MM/dd',
 'projection.date.interval' = '1',
 'projection.date.interval.unit' = 'DAYS',
 'storage.location.template' = 's3://DOC-EXAMPLE-BUCKET/AWSLogs/accountID/WAFLogs/
${region}/DOC-EXAMPLE-WEBACL/${date}/')
```

Note

예제에서 LOCATION 절의 경로 형식은 표준이지만 구현한 AWS WAF 구성에 따라 달라질 수 있습니다. 예를 들어 다음 예제 AWS WAF 로그 경로는 CloudFront 배포에 적용됩니다.

```
s3://DOC-EXAMPLE-BUCKET/AWSLogs/12345678910/WAFLogs/cloudfront/
cloudfronyt/2022/08/08/17/55/
```

AWS WAF 로그 테이블을 생성하거나 쿼리하는 동안 문제가 발생하는 경우 로그 데이터의 위치를 확인하거나 AWS Support에 문의하세요.

파티션 프로젝션에 대한 자세한 내용은 Amazon Athena를 사용한 파티션 프로젝션 단원을 참조하세요.

분할 없이 AWS WAF 로그의 테이블 생성

이 단원에서는 분할이나 파티션 프로젝션 없이 AWS WAF 로그의 테이블을 생성하는 방법을 설명합니다.

> **Note**
>
> 성능 및 비용상의 이유로 쿼리에는 파티셔닝되지 않은 스키마를 사용하지 않는 것이 좋습니다. 자세한 내용은 AWS 빅 데이터 블로그의 Top 10 Performance Tuning Tips for Amazon Athena(Amazon Athena의 성능 튜닝을 위한 10가지 팁)를 참조하세요.

AWS WAF 테이블을 생성하려면

1. 다음 DDL 문을 복사하여 Athena 콘솔에 붙여 넣습니다. 필요에 따라 필드를 로그 출력과 일치하도록 수정합니다. 로그를 저장하는 것과 일치하도록 Amazon S3 버킷의 `LOCATION`을 수정합니다.

   이 쿼리는 OpenX JSON SerDe를 사용합니다.

   > **Note**
   >
   > SerDe는 각 JSON 문서가 레코드의 필드를 구분하는 줄 종료 문자가 없는 한 줄의 텍스트에 있을 것으로 예상합니다. JSON 텍스트가 가독성 좋게 꾸민 형식이면 테이블을 만든 후 쿼리하려고 할 때 HIVE_CURSOR_ERROR: 행이 유효한 JSON 객체가 아님(HIVE_CURSOR_ERROR: Row is not a valid JSON Object) 또는 HIVE_CURSOR_ERROR: JsonParseException: 예기치 않은 입력 종료: OBJECT의 닫기 마커 필요(HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT) 같은 오류 메시지가 나타날 수 있습니다. 자세한 내용은 GitHub의 OpenX SerDe 문서에서 JSON 데이터 파일을 참조하세요.

```
CREATE EXTERNAL TABLE `waf_logs`(
  `timestamp` bigint,
  `formatversion` int,
  `webaclid` string,
  `terminatingruleid` string,
  `terminatingruletype` string,
  `action` string,
  `terminatingrulematchdetails` array <
                                  struct <
```

```
                                    conditiontype: string,
                                    sensitivitylevel: string,
                                    location: string,
                                    matcheddata: array < string >
                                      >
                                  >,
 `httpsourcename` string,
 `httpsourceid` string,
 `rulegrouplist` array <
                     struct <
                         rulegroupid: string,
                         terminatingrule: struct <
                                             ruleid: string,
                                             action: string,
                                             rulematchdetails: array <
                                                                  struct <
                                                                      conditiontype:
string,

sensitivitylevel: string,
                                                                      location:
string,
                                                                      matcheddata:
array < string >
                                                                         >
                                                                >
                                           >,
                         nonterminatingmatchingrules: array <
                                                               struct <
                                                                   ruleid: string,
                                                                   action: string,
                                                                   overriddenaction:
string,
                                                                   rulematchdetails:
array <

 struct <

     conditiontype: string,

     sensitivitylevel: string,

     location: string,
```

```
     matcheddata: array < string >

        >
                                                                    >,
                                                                   challengeresponse:
 struct <

 responsecode: string,

 solvetimestamp: string
                                                                        >,
                                                                   captcharesponse:
 struct <

 responsecode: string,

 solvetimestamp: string
                                                                        >
                                                                     >
                                                               >,
                         excludedrules: string
                           >
                      >,
`ratebasedrulelist` array <
                        struct <
                            ratebasedruleid: string,
                            limitkey: string,
                            maxrateallowed: int
                              >
                         >,
  `nonterminatingmatchingrules` array <
                                  struct <
                                      ruleid: string,
                                      action: string,
                                      rulematchdetails: array <
                                                          struct <
                                                              conditiontype:
 string,
                                                              sensitivitylevel:
 string,
                                                              location: string,
                                                              matcheddata: array <
 string >
```

```
                                                                    >
                                                                  >,
                                        challengeresponse: struct <
                                                                  responsecode: string,
                                                                  solvetimestamp: string
                                                                  >,
                                        captcharesponse: struct <
                                                                  responsecode: string,
                                                                  solvetimestamp: string
                                                                  >
                                          >
                                      >,
`requestheadersinserted` array <
                                struct <
                                    name: string,
                                    value: string
                                      >
                                 >,
`responsecodesent` string,
`httprequest` struct <
                    clientip: string,
                    country: string,
                    headers: array <
                                struct <
                                    name: string,
                                    value: string
                                      >
                                 >,
                    uri: string,
                    args: string,
                    httpversion: string,
                    httpmethod: string,
                    requestid: string
                      >,
`labels` array <
                struct <
                    name: string
                      >
                 >,
`captcharesponse` struct <
                        responsecode: string,
                        solvetimestamp: string,
                        failureReason: string
                          >,
```

```
  `challengeresponse` struct <
                        responsecode: string,
                        solvetimestamp: string,
                        failureReason: string
                        >,
  `ja3Fingerprint` string,
  `oversizefields` string,
  `requestbodysize` int,
  `requestbodysizeinspectedbywaf` int
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://DOC-EXAMPLE-BUCKET/prefix/'
```

2. Athena 콘솔 쿼리 편집기에서 `CREATE EXTERNAL TABLE` 문을 실행합니다. 이렇게 하면 `waf_logs` 테이블이 등록되고 Athena에서 그 안의 데이터를 쿼리할 수 있습니다.

AWS WAF 로그에 대한 쿼리 예제

다음과 같은 다수의 쿼리 예제가 이 문서에서 이전에 생성한 파티션 프로젝션 테이블을 사용합니다. 필요에 따라 예제의 테이블 이름, 열 값 및 기타 변수를 수정합니다. 쿼리 성능을 개선하고 비용을 줄이려면 필터 조건에 파티션 열을 추가합니다.

- Count the number of referrers that contain a specified term
- Count all matched IP addresses in the last 10 days that have matched excluded rules
- Group all counted managed rules by the number of times matched
- Group all counted custom rules by number of times matched

날짜 및 시간 작업

- Return the timestamp field in human-readable ISO 8601 format
- Return records from the last 24 hours
- Return records for a specified date range and IP address
- For a specified date range, count the number of IP addresses in five minute intervals
- Count the number of X-Forwarded-For IP in the last 10 days

차단된 요청 및 주소 작업

- Extract the top 100 IP addresses blocked by a specified rule type
- Count the number of times a request from a specified country has been blocked
- Count the number of times a request has been blocked, grouping by specific attributes
- Count the number of times a specific terminating rule ID has been matched
- Retrieve the top 100 IP addresses blocked during a specified date range

Example - 지정된 용어를 포함하는 참조자 수 계산

다음 쿼리는 지정된 날짜 범위에서 “amazon”이라는 용어를 포함한 참조자의 수를 셉니다.

```
WITH test_dataset AS
  (SELECT header FROM waf_logs
    CROSS JOIN UNNEST(httprequest.headers) AS t(header) WHERE "date" >= '2021/03/01'
    AND "date" < '2021/03/31')
SELECT COUNT(*) referer_count
FROM test_dataset
WHERE LOWER(header.name)='referer' AND header.value LIKE '%amazon%'
```

Example - 지난 10일 동안 제외된 규칙과 일치하는 모든 IP 주소 계산

다음 쿼리는 지난 10일 동안 IP 주소가 규칙 그룹에서 제외된 규칙과 일치하는 횟수를 셉니다.

```
WITH test_dataset AS
  (SELECT * FROM waf_logs
    CROSS JOIN UNNEST(rulegrouplist) AS t(allrulegroups))
SELECT
  COUNT(*) AS count,
  "httprequest"."clientip",
  "allrulegroups"."excludedrules",
  "allrulegroups"."ruleGroupId"
FROM test_dataset
WHERE allrulegroups.excludedrules IS NOT NULL AND from_unixtime(timestamp/1000) > now()
 - interval '10' day
```

```
GROUP BY "httprequest"."clientip", "allrulegroups"."ruleGroupId",
 "allrulegroups"."excludedrules"
ORDER BY count DESC
```

Example - 일치하는 횟수를 기준으로 계산된 모든 관리형 규칙 그룹화

2022년 10월 27일 이전에 웹 ACL 구성에서 규칙 그룹 규칙 작업을 개수로 설정한 경우 AWS WAF에서는 재정의를 웹 ACL JSON에 `excludedRules`로 저장했습니다. 이제 규칙을 개수로 재정의하기 위한 JSON 설정은 `ruleActionOverrides` 설정에 있습니다. 자세한 내용은 AWS WAF 개발자 안내서의 Action overrides in rule groups를 참조하세요. 새 로그 구조에서 개수 모드의 관리형 규칙을 추출하려면 다음 예제와 같이 `excludedRules` 필드 대신 `ruleGroupList` 섹션에서 `nonTerminatingMatchingRules`를 쿼리합니다.

```
SELECT
 count(*) AS count,
 httpsourceid,
 httprequest.clientip,
 t.rulegroupid,
 t.nonTerminatingMatchingRules
FROM "waf_logs"
CROSS JOIN UNNEST(rulegrouplist) AS t(t)
WHERE action <> 'BLOCK' AND cardinality(t.nonTerminatingMatchingRules) > 0
GROUP BY t.nonTerminatingMatchingRules, action, httpsourceid, httprequest.clientip,
 t.rulegroupid
ORDER BY "count" DESC
Limit 50
```

Example - 일치하는 횟수를 기준으로 계산된 모든 사용자 지정 규칙 그룹화

다음 쿼리는 일치하는 횟수를 기준으로 계산된 모든 사용자 지정 규칙을 그룹화합니다.

```
SELECT
  count(*) AS count,
         httpsourceid,
         httprequest.clientip,
         t.ruleid,
         t.action
FROM "waf_logs"
CROSS JOIN UNNEST(nonterminatingmatchingrules) AS t(t)
WHERE action <> 'BLOCK' AND cardinality(nonTerminatingMatchingRules) > 0
GROUP BY t.ruleid, t.action, httpsourceid, httprequest.clientip
```

```
ORDER BY "count" DESC
Limit 50
```

사용자 지정 규칙 및 관리형 규칙 그룹의 로그 위치에 대한 자세한 내용은 AWS WAF 개발자 안내서의 Monitoring and tuning을 참조하세요.

날짜 및 시간 작업

Example - 사람이 읽을 수 있는 ISO 8601 형식으로 타임스탬프 필드 반환

다음 쿼리는 `from_unixtime` 및 `to_iso8601` 함수를 사용하여 `timestamp` 필드를 사람이 읽을 수 있는 ISO 8601 형식으로 반환합니다(예: `1576280412771`이 아닌 `2019-12-13T23:40:12.000Z`). 쿼리는 HTTP 소스 이름, 소스 ID, 요청도 반환합니다.

```
SELECT to_iso8601(from_unixtime(timestamp / 1000)) as time_ISO_8601,
       httpsourcename,
       httpsourceid,
       httprequest
FROM waf_logs
LIMIT 10;
```

Example - 최근 24시간의 레코드 반환

다음 쿼리는 `WHERE` 절에 필터를 사용해 최근 24시간의 레코드의 HTTP 소스 이름, HTTP 소스 ID, HTTP 요청 필드를 반환합니다.

```
SELECT to_iso8601(from_unixtime(timestamp/1000)) AS time_ISO_8601,
       httpsourcename,
       httpsourceid,
       httprequest
FROM waf_logs
WHERE from_unixtime(timestamp/1000) > now() - interval '1' day
LIMIT 10;
```

Example - 지정된 날짜 범위 및 IP 주소에 대한 레코드 반환

다음 쿼리는 지정된 클라이언트 IP 주소에 대해 지정된 날짜 범위의 레코드를 나열합니다.

```
SELECT *
FROM waf_logs
```

```
WHERE httprequest.clientip='53.21.198.66' AND "date" >= '2021/03/01' AND "date" <
 '2021/03/31'
```

Example - 지정된 날짜 범위에 대해 5분 간격으로 IP 주소 개수 계산

다음 쿼리는 특정 날짜 범위에 대해 5분 간격으로 IP 주소 개수를 셉니다.

```
WITH test_dataset AS
  (SELECT
     format_datetime(from_unixtime((timestamp/1000) -
 ((minute(from_unixtime(timestamp / 1000))%5) * 60)),'yyyy-MM-dd HH:mm') AS
 five_minutes_ts,
     "httprequest"."clientip"
     FROM waf_logs
     WHERE "date" >= '2021/03/01' AND "date" < '2021/03/31')
SELECT five_minutes_ts,"clientip",count(*) ip_count
FROM test_dataset
GROUP BY five_minutes_ts,"clientip"
```

Example - 지난 10일 동안 X-Forwarded-For IP 수 계산

다음 쿼리는 요청 헤더를 필터링하고 지난 10일 동안의 X-Forwarded-For IP 수를 계산합니다.

```
WITH test_dataset AS
  (SELECT header
   FROM waf_logs
   CROSS JOIN UNNEST (httprequest.headers) AS t(header)
   WHERE from_unixtime("timestamp"/1000) > now() - interval '10' DAY)
SELECT header.value AS ip,
       count(*) AS COUNT
FROM test_dataset
WHERE header.name='X-Forwarded-For'
GROUP BY header.value
ORDER BY COUNT DESC
```

날짜 및 시간 함수에 대한 자세한 내용은 Trino 설명서의 날짜 및 시간 함수와 연산자를 참조하세요.

차단된 요청 및 주소 작업

Example - 지정된 규칙 유형에 의해 차단된 상위 100개 IP 주소 추출

다음 쿼리는 지정된 날짜 범위 동안 RATE_BASED 종료 규칙에 의해 차단된 상위 100개 IP 주소를 추출하고 그 횟수를 셉니다.

```
SELECT COUNT(httpRequest.clientIp) as count,
httpRequest.clientIp
FROM waf_logs
WHERE terminatingruletype='RATE_BASED' AND action='BLOCK' and "date" >= '2021/03/01'
AND "date" < '2021/03/31'
GROUP BY httpRequest.clientIp
ORDER BY count DESC
LIMIT 100
```

Example - 지정된 국가의 요청이 차단된 횟수 계산

다음 쿼리는 아일랜드(IE)에 속하며 RATE_BASED 종료 규칙에 의해 차단된 IP 주소에서 요청이 도착한 횟수를 계산합니다.

```
SELECT
  COUNT(httpRequest.country) as count,
  httpRequest.country
FROM waf_logs
WHERE
  terminatingruletype='RATE_BASED' AND
  httpRequest.country='IE'
GROUP BY httpRequest.country
ORDER BY count
LIMIT 100;
```

Example - 요청이 차단된 횟수 계산(특정 속성별로 그룹화)

다음 쿼리는 요청이 차단된 횟수를 계산하여 WebACL, RuleId, ClientIP 및 HTTP 요청 URI별로 결과를 그룹화합니다.

```
SELECT
  COUNT(*) AS count,
  webaclid,
  terminatingruleid,
  httprequest.clientip,
  httprequest.uri
FROM waf_logs
WHERE action='BLOCK'
GROUP BY webaclid, terminatingruleid, httprequest.clientip, httprequest.uri
ORDER BY count DESC
LIMIT 100;
```

Example - 특정 종료 규칙 ID가 일치하는 횟수 계산

다음 쿼리는 특정 종료 규칙 ID(`WHERE terminatingruleid='e9dd190d-7a43-4c06-bcea-409613d9506e'`)가 일치하는 횟수를 계산합니다. 이 쿼리는 WebACL, Action, ClientIP 및 HTTP 요청 URI별로 결과를 그룹화합니다.

```
SELECT
  COUNT(*) AS count,
  webaclid,
  action,
  httprequest.clientip,
  httprequest.uri
FROM waf_logs
WHERE terminatingruleid='e9dd190d-7a43-4c06-bcea-409613d9506e'
GROUP BY webaclid, action, httprequest.clientip, httprequest.uri
ORDER BY count DESC
LIMIT 100;
```

Example - 지정된 날짜 범위 동안 차단된 상위 100개 IP 주소 검색

다음 쿼리는 지정된 날짜 범위 동안 차단된 상위 100개 IP 주소를 추출합니다. 쿼리에는 IP 주소가 차단된 횟수도 나열됩니다.

```
SELECT "httprequest"."clientip", "count"(*) "ipcount", "httprequest"."country"
FROM waf_logs
WHERE "action" = 'BLOCK' and "date" >= '2021/03/01'
AND "date" < '2021/03/31'
GROUP BY "httprequest"."clientip", "httprequest"."country"
ORDER BY "ipcount" DESC limit 100
```

Amazon S3 로그 쿼리에 대한 자세한 내용은 다음 주제를 참조하세요.

- AWS 지식 센터의 Athena를 사용하여 내 Amazon S3 서버 액세스 로그를 분석하려면 어떻게 해야 합니까?
- Amazon Simple Storage Service 개발자 가이드의 Amazon Athena를 사용하여 요청에 대한 Amazon S3 액세스 로그 쿼리
- Amazon Simple Storage Service 사용 설명서의 AWS CloudTrail를 사용하여 Amazon S3 요청 식별

Amazon S3에 저장된 웹 서버 로그 쿼리

Athena를 사용하여 Amazon S3에 저장된 웹 서버 로그를 쿼리할 수 있습니다. 이 섹션의 주제는 Athena에서 테이블을 생성하여 다양한 형식의 웹 서버 로그를 쿼리하는 방법을 보여줍니다.

주제

Amazon S3에 저장된 Apache 로그 쿼리

Amazon Athena를 사용하여 Amazon S3 계정에 저장된 Apache HTTP 서버 로그 파일을 쿼리할 수 있습니다. 이 주제는 테이블 스키마를 생성하여 일반 로그 형식의 Apache 액세스 로그 파일을 쿼리하는 방법을 보여줍니다.

일반 로그 형식의 필드에는 클라이언트 IP 주소, 클라이언트 ID, 사용자 ID, 요청 수신 타임스탬프, 클라이언트 요청의 텍스트, 서버 상태 코드, 클라이언트로 반환된 개체의 크기가 포함됩니다.

다음 데이터 예제는 Apache 일반 로그 형식을 보여 줍니다.

```
198.51.100.7 - Li [10/Oct/2019:13:55:36 -0700] "GET /logo.gif HTTP/1.0" 200 232
198.51.100.14 - Jorge [24/Nov/2019:10:49:52 -0700] "GET /index.html HTTP/1.1" 200 2165
198.51.100.22 - Mateo [27/Dec/2019:11:38:12 -0700] "GET /about.html HTTP/1.1" 200 1287
198.51.100.9 - Nikki [11/Jan/2020:11:40:11 -0700] "GET /image.png HTTP/1.1" 404 230
198.51.100.2 - Ana [15/Feb/2019:10:12:22 -0700] "GET /favicon.ico HTTP/1.1" 404 30
198.51.100.13 - Saanvi [14/Mar/2019:11:40:33 -0700] "GET /intro.html HTTP/1.1" 200 1608
198.51.100.11 - Xiulan [22/Apr/2019:10:51:34 -0700] "GET /group/index.html HTTP/1.1"
 200 1344
```

Athena에서 Apache 로그에 대한 테이블 만들기

Amazon S3에 저장된 Apache 로그를 쿼리하기 전에 먼저 Athena용 테이블 스키마를 생성해야 로그 데이터를 읽을 수 있습니다. Apache 로그에 대한 Athena 테이블은 Grok SerDe를 사용해 만들 수 있습니다 . Grok SerDe를 사용하는 방법에 대한 자세한 내용은 AWS Glue 개발자 안내서의 Grok 사용자 지정 분류자 작성을 참조하세요.

Athena에서 Apache 웹 서버 로그에 대한 테이블 생성

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.

2. 다음 DDL 문을 Athena 쿼리 편집기에 붙여 넣습니다. Amazon S3의 Athena 로그를 가리키도록 LOCATION 's3://DOC-EXAMPLE-BUCKET/*apache-log-folder*/'의 값을 수정합니다.

```
CREATE EXTERNAL TABLE apache_logs (
  client_ip string,
  client_id string,
  user_id string,
  request_received_time string,
  client_request string,
  server_status string,
  returned_obj_size string
  )
ROW FORMAT SERDE
   'com.amazonaws.glue.serde.GrokSerDe'
WITH SERDEPROPERTIES (
   'input.format'='^%{IPV4:client_ip} %{DATA:client_id} %{USERNAME:user_id}
 %{GREEDYDATA:request_received_time} %{QUOTEDSTRING:client_request}
 %{DATA:server_status} %{DATA: returned_obj_size}$'
   )
STORED AS INPUTFORMAT
   'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
   'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
   's3://DOC-EXAMPLE-BUCKET/apache-log-folder/';
```

3. Athena 콘솔에서 apache_logs 테이블을 등록하는 쿼리를 실행합니다. 이 쿼리가 완료되면 로그를 Athena에서 쿼리할 수 있습니다.

Apache 로그에 대한 선택 쿼리 예제

Example - 404 오류에 대한 필터링

다음 쿼리 예제는 apache_logs 테이블에서 요청 수신 시간, 클라이언트 요청의 텍스트 및 서버 상태 코드를 선택합니다. WHERE 절은 HTTP 상태 코드 404(페이지를 찾을 수 없음)를 필터링합니다.

```
SELECT request_received_time, client_request, server_status
FROM apache_logs
WHERE server_status = '404'
```

다음 이미지는 Athena 쿼리 편집기의 쿼리 결과를 보여줍니다.

Results

	request_received_time	client_request	server_status
1	[11/Jan/2020:11:40:11 -0700]	GET /image.png HTTP/1.1	404
2	[15/Feb/2019:10:12:22 -0700]	GET /favicon.ico HTTP/1.1	404

Example - 성공적인 요청에 대한 필터링

다음 쿼리 예제는 apache_logs 테이블에서 사용자 ID, 요청 수신 시간, 클라이언트 요청의 텍스트 및 서버 상태 코드를 선택합니다. WHERE 절은 HTTP 상태 코드 200(성공)을 필터링합니다.

```
SELECT user_id, request_received_time, client_request, server_status
FROM apache_logs
WHERE server_status = '200'
```

다음 이미지는 Athena 쿼리 편집기의 쿼리 결과를 보여줍니다.

Results

	user_id	request_received_time	client_request	server_status
1	Li	[10/Oct/2019:13:55:36 -0700]	GET /logo.gif HTTP/1.0	200
2	Jorge	[24/Nov/2019:10:49:52 -0700]	GET /index.html HTTP/1.1	200
3	Mateo	[27/Dec/2019:11:38:12 -0700]	GET /about.html HTTP/1.1	200
4	Saanvi	[14/Mar/2019:11:40:33 -0700]	GET /intro.html HTTP/1.1	200
5	Xiulan	[22/Apr/2019:10:51:34 -0700]	GET /group/index.html HTTP/1.1	200

Example - 타임스탬프별 필터링

다음 예제에서는 요청 수신 시간이 지정된 타임스탬프보다 큰 레코드를 쿼리합니다.

```
SELECT * FROM apache_logs WHERE request_received_time > 10/Oct/2023:00:00:00
```

Amazon S3에 저장된 Internet Information Server(IIS) 로그 쿼리

Amazon Athena를 사용하여 Amazon S3 계정에 저장된 Microsoft Internet Information Services(IIS) 웹 서버 로그를 쿼리합니다. IIS는 다양한 로그 파일 형식을 사용하지만 이 주제에서는 Athena에서 테이블 스키마를 생성하여 W3C 확장 로그 및 IIS 로그 파일 형식의 로그를 쿼리하는 방법을 보여줍니다.

W3C 확장 로그 및 IIS 로그 파일 형식은 단일 문자 구분 기호(각각 공백 및 쉼표)를 사용하고 따옴표로 값을 둘러싸지 않으므로 LazySimpleSerDe를 사용해 이에 대한 Athena 테이블을 생성할 수 있습니다.

W3C 확장 로그 파일 형식

W3C 확장 로그 파일 데이터 형식은 공백으로 구분된 필드를 가집니다. W3C 확장 로그에 나타나는 필드는 포함할 로그 필드를 선택하는 웹 서버 관리자가 결정합니다. 다음 로그 데이터 예제에는 `date`, `time`, `c-ip`, `s-ip`, `cs-method`, `cs-uri-stem`, `sc-status`, `sc-bytes`, `cs-bytes`, `time-taken`, `cs-version` 필드가 있습니다.

```
2020-01-19 22:48:39 203.0.113.5 198.51.100.2 GET /default.html 200 540 524 157 HTTP/1.0
2020-01-19 22:49:40 203.0.113.10 198.51.100.12 GET /index.html 200 420 324 164 HTTP/1.0
2020-01-19 22:50:12 203.0.113.12 198.51.100.4 GET /image.gif 200 324 320 358 HTTP/1.0
2020-01-19 22:51:44 203.0.113.15 198.51.100.16 GET /faq.html 200 330 324 288 HTTP/1.0
```

Athena에서 W3C 확장 로그에 대한 테이블 만들기

W3C 확장 로그를 쿼리하기 전에 먼저 용 테이블 스키마를 생성해야 Athena가 로그 데이터를 읽을 수 있습니다.

Athena에서 W3C 확장 로그에 대한 테이블을 만들려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 다음과 같은 DDL 문을 Athena 콘솔에 붙여 넣습니다. 다음 사항에 유의하세요.
   a. 예제에서 쿼리할 로그의 필드에 해당하는 열을 추가하거나 제거합니다.
   b. W3C 확장 로그 파일 형식의 열 이름에는 하이픈(-)이 있습니다. 그러나 `CREATE TABLE` 문 예제는 Athena 이름 지정 규칙에 따라 이를 밑줄(_)로 바꿉니다.
   c. 공백 구분 기호를 지정하려면 `FIELDS TERMINATED BY ' '`를 사용합니다.
   d. Amazon S3의 W3C 확장 로그를 가리키도록 `LOCATION 's3://DOC-EXAMPLE-BUCKET/w3c-log-folder/'`의 값을 수정합니다.

```
CREATE EXTERNAL TABLE `iis_w3c_logs`(
```

```
  date_col string,
  time_col string,
  c_ip string,
  s_ip string,
  cs_method string,
  cs_uri_stem string,
  sc_status string,
  sc_bytes string,
  cs_bytes string,
  time_taken string,
  cs_version string
  )
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ' '
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION   's3://DOC-EXAMPLE-BUCKET/w3c-log-folder/'
```

3. Athena 콘솔에서 `iis_w3c_logs` 테이블을 등록하는 쿼리를 실행합니다. 이 쿼리가 완료되면 로그를 Athena에서 쿼리할 수 있습니다.

W3C 확장 로그 셀렉트 쿼리 예제

다음 쿼리 예제는 `iis_w3c_logs` 테이블에서 날짜, 시간, 요청 대상, 요청에 걸린 시간을 선택합니다. `WHERE` 절은 HTTP 메서드가 `GET`이고 HTTP 상태 코드가 `200`(성공)인 경우를 필터링합니다.

```
SELECT date_col, time_col, cs_uri_stem, time_taken
FROM iis_w3c_logs
WHERE cs_method = 'GET' AND sc_status = '200'
```

다음 이미지는 Athena 쿼리 편집기의 쿼리 결과를 보여줍니다.

Results

	date_col	time_col	cs_uri_stem	time_taken
1	2020-01-19	22:48:39	/default.html	157
2	2020-01-19	22:49:40	/index.html	164
3	2020-01-19	22:50:12	/image.gif	358
4	2020-01-19	22:51:44	/faq.html	288

날짜 및 시간 필드 결합

공백으로 구분된 date 및 time 필드는 로그 소스 데이터에서 분리된 항목들이지만 원할 경우 이들을 타임스탬프로 결합할 수 있습니다. SELECT 또는 CREATE TABLE AS SELECT 쿼리에서 concat() 및 date_parse() 함수를 사용하여 날짜 및 시간 열을 타임스탬프 형식으로 연결하고 변환합니다. 다음 예제에서는 CTAS 쿼리를 사용하여 derived_timestamp 열을 가진 새 테이블을 만듭니다.

```
CREATE TABLE iis_w3c_logs_w_timestamp AS
SELECT
  date_parse(concat(date_col,' ', time_col),'%Y-%m-%d %H:%i:%s') as derived_timestamp,
  c_ip,
  s_ip,
  cs_method,
  cs_uri_stem,
  sc_status,
  sc_bytes,
  cs_bytes,
  time_taken,
  cs_version
FROM iis_w3c_logs
```

다음 예제와 같이 테이블을 생성하면 새 타임스탬프 열을 직접 쿼리할 수 있습니다.

```
SELECT derived_timestamp, cs_uri_stem, time_taken
FROM iis_w3c_logs_w_timestamp
WHERE cs_method = 'GET' AND sc_status = '200'
```

다음 이미지는 쿼리의 결과를 보여줍니다.

Results

	derived_timestamp	cs_uri_stem	time_taken
1	2020-01-19 22:48:39.000	/default.html	157
2	2020-01-19 22:49:40.000	/index.html	164
3	2020-01-19 22:50:12.000	/image.gif	358
4	2020-01-19 22:51:44.000	/faq.html	288

IIS 로그 파일 형식

W3C 확장 형식과 달리 IIS 로그 파일 형식에는 고정된 필드 집합이 있으며 구분 기호로 쉼표를 포함합니다. LazySimpleSerDe는 쉼표를 구분 기호로 처리하고 쉼표 뒤의 공백을 다음 필드의 시작으로 처리합니다.

다음 예제에서는 IIS 로그 파일 형식의 샘플 데이터를 보여 줍니다.

```
203.0.113.15, -, 2020-02-24, 22:48:38, W3SVC2, SERVER5, 198.51.100.4, 254, 501, 488,
 200, 0, GET, /index.htm, -,
203.0.113.4, -, 2020-02-24, 22:48:39, W3SVC2, SERVER6, 198.51.100.6, 147, 411, 388,
 200, 0, GET, /about.html, -,
203.0.113.11, -, 2020-02-24, 22:48:40, W3SVC2, SERVER7, 198.51.100.18, 170, 531, 468,
 200, 0, GET, /image.png, -,
203.0.113.8, -, 2020-02-24, 22:48:41, W3SVC2, SERVER8, 198.51.100.14, 125, 711, 868,
 200, 0, GET, /intro.htm, -,
```

Athena에서 IIS 로그 파일에 대한 테이블 만들기

Amazon S3의 IIS 로그를 쿼리하려면 먼저 테이블 스키마를 생성해야 Athena가 로그 데이터를 읽을 수 있습니다.

Athena에서 IIS 로그 파일 형식 로그에 대한 테이블 생성

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 다음의 DDL 문을 Athena 콘솔에 붙여 넣습니다. 다음 사항에 유의하세요.
   a. 쉼표 구분 기호를 지정하려면 `FIELDS TERMINATED BY ','`를 사용합니다.
   b. LOCATION 's3://DOC-EXAMPLE-BUCKET/*iis-log-file-folder*/'의 값을 Amazon S3의 IIS 로그 형식 로그 파일을 가리키도록 수정합니다.

```
CREATE EXTERNAL TABLE `iis_format_logs`(
client_ip_address string,
user_name string,
request_date string,
request_time string,
service_and_instance string,
server_name string,
server_ip_address string,
time_taken_millisec string,
client_bytes_sent string,
server_bytes_sent string,
service_status_code string,
windows_status_code string,
request_type string,
target_of_operation string,
script_parameters string
   )
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/iis-log-file-folder/'
```

3. Athena 콘솔에서 `iis_format_logs` 테이블을 등록하는 쿼리를 실행합니다. 이 쿼리가 완료되면 로그를 Athena에서 쿼리할 수 있습니다.

IIS 로그 형식 셀렉트 쿼리 예제

다음 쿼리 예제는 `iis_format_logs` 테이블에서 요청 날짜, 요청 시간, 요청 대상, 걸린 시간(밀리초 단위)을 선택합니다. `WHERE` 절은 요청 유형이 `GET`이고 HTTP 상태 코드가 `200`(성공)인 경우를 필터링합니다. 쿼리가 성공하려면 쿼리에서 `' GET'` 및 `' 200'`에 선행하는 공백이 필요합니다.

```
SELECT request_date, request_time, target_of_operation, time_taken_millisec
FROM iis_format_logs
WHERE request_type = ' GET' AND service_status_code = ' 200'
```

다음 이미지는 샘플 데이터에 대한 쿼리 결과를 보여줍니다.

Results

	request_date	request_time	target_of_operation	time_taken_millisec
1	2020-02-24	22:48:38	/index.htm	254
2	2020-02-24	22:48:39	/about.html	147
3	2020-02-24	22:48:40	/image.png	170
4	2020-02-24	22:48:41	/intro.htm	125

NCSA 로그 파일 형식

IIS는 NCSA 로깅 형식도 사용합니다. 이 형식은 공백으로 구분된 ASCII 텍스트 형식의 고정된 필드 수를 갖습니다. 구조는 Apache 액세스 로그에 사용되는 일반적인 로그 형식과 유사합니다. NCSA 일반 로그 형식의 필드에는 클라이언트 IP 주소, 클라이언트 ID(일반적으로 사용되지 않음), 도메인\사용자 ID, 요청 수신 타임스탬프, 클라이언트 요청의 텍스트, 서버 상태 코드, 클라이언트로 반환된 개체의 크기가 포함됩니다.

다음 예제는 IIS에서 설명한 것과 같은 NCSA 일반 로그 형식의 데이터를 보여 줍니다.

```
198.51.100.7 - ExampleCorp\Li [10/Oct/2019:13:55:36 -0700] "GET /logo.gif HTTP/1.0" 200
 232
198.51.100.14 - AnyCompany\Jorge [24/Nov/2019:10:49:52 -0700] "GET /index.html
 HTTP/1.1" 200 2165
198.51.100.22 - ExampleCorp\Mateo [27/Dec/2019:11:38:12 -0700] "GET /about.html
 HTTP/1.1" 200 1287
198.51.100.9 - AnyCompany\Nikki [11/Jan/2020:11:40:11 -0700] "GET /image.png HTTP/1.1"
 404 230
198.51.100.2 - ExampleCorp\Ana [15/Feb/2019:10:12:22 -0700] "GET /favicon.ico HTTP/1.1"
 404 30
198.51.100.13 - AnyCompany\Saanvi [14/Mar/2019:11:40:33 -0700] "GET /intro.html
 HTTP/1.1" 200 1608
198.51.100.11 - ExampleCorp\Xiulan [22/Apr/2019:10:51:34 -0700] "GET /group/index.html
 HTTP/1.1" 200 1344
```

Athena에서 IIS NCSA 로그에 대한 테이블 만들기

`CREATE TABLE` 문에는 Apache 웹 서버 로그의 경우와 비슷한 Grok SerDe 및 grok 패턴을 사용할 수 있습니다. Apache 로그와 달리, grok 패턴은 `domain\user_id`에서 백슬래시의 존재를 설명하기 위해 세 번째 필드에 대해 `%{USERNAME:user_id}` 대신 `%{DATA:user_id}`를 사용합니다. Grok

SerDe를 사용하는 방법에 대한 자세한 내용은 AWS Glue 개발자 안내서의 Grok 사용자 지정 분류자 작성을 참조하세요.

Athena에서 IIS NCSA 웹 서버 로그에 대한 테이블 생성

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 다음의 DDL 문을 Athena 쿼리 편집기에 붙여 넣습니다. Amazon S3의 IIS NCSA 로그를 가리키도록 `LOCATION 's3://DOC-EXAMPLE-BUCKET/iis-ncsa-logs/'`의 값을 수정합니다.

```
CREATE EXTERNAL TABLE iis_ncsa_logs(
  client_ip string,
  client_id string,
  user_id string,
  request_received_time string,
  client_request string,
  server_status string,
  returned_obj_size string
  )
ROW FORMAT SERDE
   'com.amazonaws.glue.serde.GrokSerDe'
WITH SERDEPROPERTIES (
   'input.format'='^%{IPV4:client_ip} %{DATA:client_id} %{DATA:user_id}
 %{GREEDYDATA:request_received_time} %{QUOTEDSTRING:client_request}
 %{DATA:server_status} %{DATA: returned_obj_size}$'
   )
STORED AS INPUTFORMAT
   'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
   'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
   's3://DOC-EXAMPLE-BUCKET/iis-ncsa-logs/';
```

3. Athena 콘솔에서 `iis_ncsa_logs` 테이블을 등록하는 쿼리를 실행합니다. 이 쿼리가 완료되면 로그를 Athena에서 쿼리할 수 있습니다.

IIS NCSA 로그에 대한 선택 쿼리 예제

Example - 404 오류에 대한 필터링

다음 쿼리 예제는 `iis_ncsa_logs` 테이블에서 요청 수신 시간, 클라이언트 요청의 텍스트 및 서버 상태 코드를 선택합니다. `WHERE` 절은 HTTP 상태 코드 `404`(페이지를 찾을 수 없음)를 필터링합니다.

```
SELECT request_received_time, client_request, server_status
FROM iis_ncsa_logs
WHERE server_status = '404'
```

다음 이미지는 Athena 쿼리 편집기의 쿼리 결과를 보여줍니다.

Results

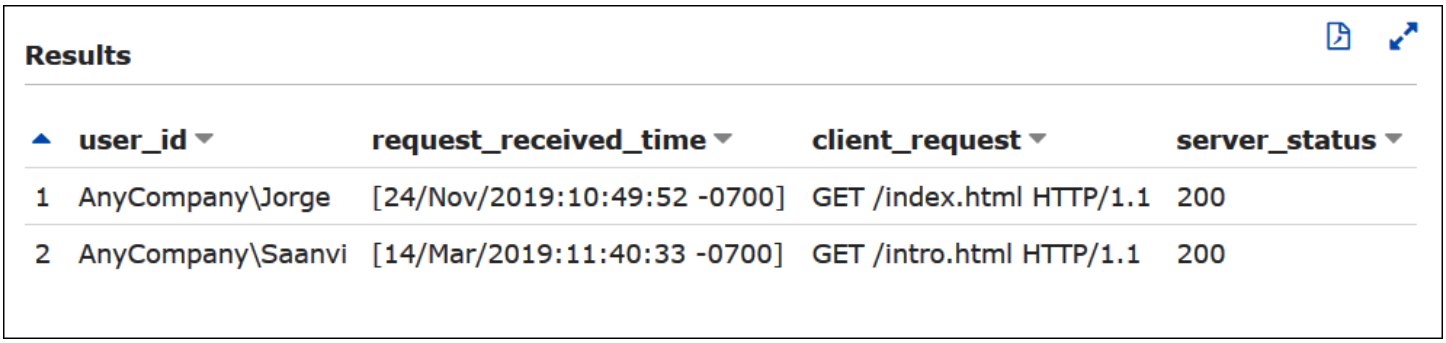
	request_received_time	client_request	server_status
1	[11/Jan/2020:11:40:11 -0700]	GET /image.png HTTP/1.1	404
2	[15/Feb/2019:10:12:22 -0700]	GET /favicon.ico HTTP/1.1	404

Example - 특정 도메인의 성공적인 요청에 대한 필터링

다음 쿼리 예제는 `iis_ncsa_logs` 테이블에서 사용자 ID, 요청 수신 시간, 클라이언트 요청의 텍스트 및 서버 상태 코드를 선택합니다. `WHERE` 절은 HTTP 상태 코드 `200`(성공)을 가진 `AnyCompany` 도메인의 사용자 요청을 필터링합니다.

```
SELECT user_id, request_received_time, client_request, server_status
FROM iis_ncsa_logs
WHERE server_status = '200' AND user_id LIKE 'AnyCompany%'
```

다음 이미지는 Athena 쿼리 편집기의 쿼리 결과를 보여줍니다.



Results

	user_id	request_received_time	client_request	server_status
1	AnyCompany\Jorge	[24/Nov/2019:10:49:52 -0700]	GET /index.html HTTP/1.1	200
2	AnyCompany\Saanvi	[14/Mar/2019:11:40:33 -0700]	GET /intro.html HTTP/1.1	200

Athena ACID 트랜잭션 사용

"ACID 트랜잭션"이라는 용어는 데이터베이스 트랜잭션에서 데이터 무결성을 보장하는 속성(원자성, 일관성, 격리 및 내구성)을 나타냅니다. ACID 트랜잭션을 사용하면 여러 사용자가 Amazon S3 객체를

어토믹(atomic) 방식으로 동시에 안정적으로 추가하고 삭제할 수 있으며 데이터 레이크에 대한 쿼리를 일관되게 읽어 기존 쿼리를 격리할 수 있습니다. Athena ACID 트랜잭션은 삽입, 삭제, 업데이트 및 시간 이동 작업에 대한 단일 테이블 지원을 Athena SQL 데이터 조작 언어(DML)에 추가합니다. 사용자와 여러 명의 동시 사용자가 Athena ACID 트랜잭션을 사용하여 Amazon S3 데이터를 행 수준에서 안정적으로 수정할 수 있습니다. Athena 트랜잭션은 잠금 의미 체계와 코디네이션을 자동으로 관리하며 사용자 지정 레코드 잠금 솔루션이 필요하지 않습니다.

Athena ACID 트랜잭션과 친숙한 SQL 구문은 비즈니스 및 규제 데이터에 대한 업데이트를 간소화합니다. 예를 들어 데이터 삭제 요청에 응답하려면 SQL `DELETE` 작업을 수행할 수 있습니다. 레코드를 수동으로 수정하려면 단일 `UPDATE` 문을 사용할 수 있습니다. 최근에 삭제된 데이터를 복구하려면 `SELECT` 문을 사용하여 시간 이동 쿼리를 실행할 수 있습니다.

Athena ACID 트랜잭션은 공유 테이블 형식을 기반으로 하기 때문에 공유 테이블 형식도 지원하는 Amazon EMR 및 Apache Spark와 같은 다른 서비스 및 엔진과 호환됩니다.

Athena 트랜잭션은 Athena 콘솔, API 작업, ODBC 및 JDBC 드라이버를 통해 사용할 수 있습니다.

주제

- Linux Foundation Delta Lake 테이블 쿼리
- Athena를 사용하여 Apache Hudi 데이터 집합 쿼리
- Apache Iceberg 테이블 사용

Linux Foundation Delta Lake 테이블 쿼리

Linux Foundation Delta Lake는 빅 데이터 분석을 위한 테이블 형식입니다. Amazon Athena를 사용하면 매니페스트 파일을 생성하거나 `MSCK REPAIR` 문을 실행할 필요 없이 Amazon S3에 저장된 Delta Lake 테이블을 직접 읽을 수 있습니다.

Delta Lake 형식은 각 데이터 파일의 열당 최소값과 최대값을 저장합니다. Athena 구현에서는 이 정보를 사용하여 조건자에 대한 파일 건너뛰기를 활성화하여 원치 않는 파일이 고려되지 않도록 합니다.

고려 사항 및 제한

Athena의 Delta Lake 지원에는 다음과 같은 고려 사항 및 제한 사항이 있습니다.

- AWS Glue 카탈로그를 포함하는 테이블만 - 기본 Delta Lake 지원은 AWS Glue에 등록된 테이블을 통해서만 지원됩니다. 다른 메타스토어에 등록된 Delta Lake 테이블이 있는 경우 해당 테이블을 그대로 유지하고 기본 메타스토어로 처리할 수 있습니다. Delta Lake 메타데이터는 메타스토어가 아닌

파일 시스템(예: Amazon S3)에 저장되므로 Athena에서는 AWS Glue에 위치 속성만 있으면 Delta Lake 테이블에서 읽을 수 있습니다.

- V3 engine only(V3 엔진 전용) - Delta Lake 쿼리는 Athena 엔진 버전 3에서만 지원됩니다. 생성된 작업 그룹이 Athena 엔진 버전 3을 사용하도록 구성되어 있는지 확인해야 합니다.
- Delta Lake 리더 버전 - Delta Lake 리더 프로토콜은 최대 버전 3까지 지원됩니다.
- 열 매핑 및 타임스탬프Ntz - Delta 테이블 열과 기본 Parquet 파일 열이 서로 다른 이름을 사용할 수 있도록 하는 Delta 열 매핑과 시간대가 없는 타임스탬프(timestampNtz)가 지원됩니다.
- No time travel support(시간 이동 지원 안 함) - Delta Lake의 시간 이동 기능을 사용하는 쿼리는 지원되지 않습니다.
- Read only(읽기 전용) - DML 문(예: `UPDATE`, `INSERT` 또는 `DELETE`)을 쓸 수 없습니다.
- Lake Formation 지원 - Lake Formation 통합은 AWS Glue와 스키마가 동기화된 Delta Lake 테이블에 사용할 수 있습니다. 자세한 내용은 AWS Lake Formation 개발자 안내서의 Using AWS Lake Formation with Amazon Athena 및 Set up permissions for a Delta Lake table을 참조하세요.
- Limited DDL support(제한된 DDL 지원) - `CREATE EXTERNAL TABLE`, `SHOW COLUMNS`, `SHOW TBLPROPERTIES`, `SHOW PARTITIONS`, `SHOW CREATE TABLE`, `DESCRIBE` DDL 문이 지원됩니다. `CREATE EXTERNAL TABLE` 문 사용에 대한 자세한 내용은 시작하기 단원을 참조하세요.
- S3 Glacier 객체 건너뛰기 미지원 – Linux Foundation Delta Lake 테이블의 객체가 Amazon S3 Glacier 스토리지 클래스에 있는 경우 `read_restored_glacier_objects` 테이블 속성을 `false`로 설정해도 효과가 없습니다.

  예를 들어 다음과 같은 명령을 실행한다고 가정하겠습니다.

```
ALTER TABLE table_name SET TBLPROPERTIES ('read_restored_glacier_objects' = 'false')
```

  Iceberg 및 Delta Lake 테이블의 경우 이 명령은 지원되지 않는 테이블 속성 키: read_restored_glacier_objects 오류를 생성합니다. Hudi 테이블의 경우 `ALTER TABLE` 명령은 오류를 생성하지 않지만 여전히 Amazon S3 Glacier 객체를 건너뛰지 않습니다. `ALTER TABLE` 명령 후에 `SELECT` 쿼리를 실행하면 계속해서 모든 개체가 반환됩니다.

지원되는 비분할 열 데이터 형식

비분할 열의 경우 `CHAR`를 제외하고 Athena에서 지원하는 모든 데이터 형식이 지원됩니다(`CHAR`는 Delta Lake 프로토콜 자체에서 지원되지 않음). 지원되는 데이터 형식은 다음과 같습니다.

```
boolean
tinyint
```

```
smallint
integer
bigint
double
float
decimal
varchar
string
binary
date
timestamp
array
map
struct
```

지원되는 파티션 열 데이터 형식

파티션 열의 경우 Athena에서는 다음과 같은 데이터 형식을 포함하는 테이블을 지원합니다.

```
boolean
integer
smallint
tinyint
bigint
decimal
float
double
date
timestamp
varchar
```

Athena의 데이터 형식에 대한 자세한 내용은 Amazon Athena의 데이터 형식 단원을 참조하세요.

시작하기

쿼리가 가능하려면 Delta Lake 테이블이 AWS Glue에 있어야 합니다. 테이블이 Amazon S3에 있지만 AWS Glue에 없는 경우 다음 구문을 사용하여 `CREATE EXTERNAL TABLE` 문을 실행합니다. 테이블이 이미 AWS Glue에 있는 경우(예: AWS Glue에서 Apache Spark 또는 다른 엔진을 사용 중인 경우) 이 단계를 건너뛰어도 됩니다.

```
CREATE EXTERNAL TABLE
  [db_name.]table_name
```

```
LOCATION 's3://DOC-EXAMPLE-BUCKET/your-folder/'
TBLPROPERTIES ('table_type' = 'DELTA')
```

열 정의, SerDe 라이브러리 및 기타 테이블 속성의 생략에 유의하세요. 기존 Hive 테이블과 달리 Delta Lake 테이블 메타데이터는 Delta Lake 트랜잭션 로그에서 추론되고 AWS Glue에 직접 동기화됩니다.

> **Note**
>
> Delta Lake 테이블의 경우 `LOCATION` 및 `table_type` 속성보다 많은 항목이 포함된 `CREATE TABLE` 문은 허용되지 않습니다.

Delta Lake 테이블 읽기

Delta Lake 테이블을 쿼리하려면 표준 SQL `SELECT` 구문을 사용하세요.

```
[ WITH with_query [, ...] ]SELECT [ ALL | DISTINCT ] select_expression [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[ HAVING condition ]
[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
[ ORDER BY expression [ ASC | DESC ] [ NULLS FIRST | NULLS LAST] [, ...] ]
[ OFFSET count [ ROW | ROWS ] ]
[ LIMIT [ count | ALL ] ]
```

`SELECT` 구문에 대한 자세한 내용은 Athena 설명서의 SELECT 단원을 참조하세요.

Delta Lake 형식은 각 데이터 파일의 열당 최소값과 최대값을 저장합니다. Athena에서는 이 정보를 사용하여 조건자에 대한 파일 건너뛰기를 활성화하여 불필요한 파일이 고려되지 않도록 합니다.

Delta Lake 메타데이터 동기화

Athena를 사용하여 Delta Lake 테이블을 생성하는 경우 Athena에서는 스키마, 파티션 열 및 테이블 속성을 비롯한 테이블 메타데이터를 AWS Glue에 동기화합니다. 시간이 경과함에 따라 이 메타데이터와 트랜잭션 로그 내 기본 테이블 메타데이터의 동기화가 손실될 수 있습니다. 테이블을 최신 상태로 유지하기 위해 다음 옵션 중 하나를 선택할 수 있습니다.

- Delta Lake 테이블에 대해 AWS Glue 크롤러를 사용합니다. 자세한 내용은 AWS 빅 데이터 블로그의 Introducing native Delta Lake table support with AWS Glue crawlers 및 AWS Glue 개발자 안내서의 Scheduling an AWS Glue crawler를 참조하세요.

- Athena에서 테이블을 삭제하고 다시 생성합니다.
- SDK, CLI 또는 AWS Glue 콘솔을 사용하여 AWS Glue에서 스키마를 수동으로 업데이트합니다.

다음 기능을 사용하려면 AWS Glue 스키마가 항상 트랜잭션 로그와 동일한 스키마를 보유해야 합니다.

- Lake Formation
- 보기
- 행 및 열 필터

워크플로에 이 기능이 필요하지 않고 이러한 호환성을 유지 관리하고 싶지 않은 경우 Athena에서 `CREATE TABLE` DDL을 사용한 후 AWS Glue에서 Amazon S3 경로를 SerDe 파라미터로 추가하면 됩니다.

Athena 및 AWS Glue 콘솔을 사용하여 Delta Lake 테이블을 생성하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. Athena 쿼리 편집기에서 다음 DDL을 사용하여 Delta Lake 테이블을 생성합니다. 이 방법을 사용하는 경우 `TBLPROPERTIES`의 값이 `'table_type' = 'delta'`가 아닌 `'spark.sql.sources.provider' = 'delta'`여야 합니다.

   Apache Spark(Athena for Apache Spark) 또는 대부분의 다른 엔진을 사용하여 테이블을 생성하는 경우 이 동일한 스키마(열 이름이 `col`이고 유형이 `array<string>`인 단일 열)가 삽입됩니다.

   ```
   CREATE EXTERNAL TABLE
      [db_name.]table_name(col array<string>)
      LOCATION 's3://DOC-EXAMPLE-BUCKET/your-folder/'
      TBLPROPERTIES ('spark.sql.sources.provider' = 'delta')
   ```

3. https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
4. 탐색 창의 데이터 카탈로그에서 테이블을 선택합니다.
5. 테이블 목록에서 테이블에 대한 링크를 선택합니다.
6. 테이블 페이지에서 작업, 테이블 편집을 선택합니다.
7. Serde 파라미터 섹션에서 값이 **`s3://DOC-EXAMPLE-BUCKET/your-folder/`**인 **`path`** 키를 추가합니다.
8. Save(저장)를 선택합니다.

추가적인 리소스

AWS Glue에서 Delta Lake 테이블을 사용하고 Athena에서 해당 테이블을 쿼리하는 방법에 대한 설명은 AWS 빅 데이터 블로그의 Handle UPSERT data operations using open-source Delta Lake and AWS Glue를 참조하세요.

Athena를 사용하여 Apache Hudi 데이터 집합 쿼리

Apache Hudi는 증분 데이터 처리를 간소화하는 오픈 소스 데이터 관리 프레임워크입니다. 레코드 수준의 삽입, 업데이트, 업서트, 삭제 작업이 훨씬 세밀하게 처리되므로 오버헤드가 줄어듭니다. `Upsert`는 레코드가 존재하지 않으면 기존 데이터 집합에 레코드를 삽입하고 레코드가 존재하면 레코드를 업데이트할 수 있는 기능입니다.

Hudi는 분석 시 성능 문제를 일으킬 수 있는 다수의 작은 파일을 발생시키지 않고도 데이터 삽입 및 업데이트 이벤트를 처리합니다. Apache Hudi는 자동으로 변경 사항을 추적하고 파일을 병합하여 최적의 크기를 유지합니다. 이렇게 하면 다수의 작은 파일을 더 적은 수의 대용량 파일에 다시 쓰고 모니터링하는 사용자 지정 솔루션을 빌드할 필요가 없습니다.

Hudi 데이터 집합은 다음과 같은 사용 사례에 적합합니다.

- 개인 정보를 제거하거나 자신의 데이터 사용 방식을 변경할 수 있는 사람들의 권리를 강제하는 일반 데이터 보호 규정(GDPR) 및 캘리포니아 소비자 개인정보보호법(CCPA)과 같은 개인 정보 보호 규정의 준수.
- 특정 데이터 삽입 및 업데이트 이벤트가 필요한 센서 및 기타 사물 인터넷(IoT) 디바이스에서의 스트리밍 데이터 작업.
- 변경 데이터 캡쳐(CDC) 시스템의 구현.

Hudi가 관리하는 데이터 세트는 오픈 스토리지 형식을 사용하여 Amazon S3에 저장됩니다. 현재 Athena는 압축된 Hudi 데이터 집합을 읽을 수 있지만 Hudi 데이터를 쓸 수는 없습니다. Athena는 Athena 엔진 버전 2에서 Hudi 버전 0.8.0까지 지원하고 Athena 엔진 버전 3에서 Hudi 버전 0.14.0까지 지원합니다. 변경될 수 있습니다. Athena는 이후 버전의 Hudi로 생성된 테이블과의 읽기 호환성을 보장할 수 없습니다. Athena 엔진 버전 관리에 대한 자세한 내용은 Athena 엔진 버전 관리 단원을 참조하세요. Hudi 기능 및 버전 관리에 대한 자세한 내용은 Apache 웹 사이트의 Hudi 설명서를 참조하세요.

Hudi 데이터 집합 테이블 유형

Hudi 데이터 집합은 다음 유형 중 하나일 수 있습니다.

- 쓸 때 복사(CoW) - 데이터가 열 기반 형식(Parquet)으로 저장되며, 각 업데이트마다 쓰기 중에 새 버전의 파일을 만듭니다.
- 읽을 때 병합(MoR) - 데이터가 열 기반 형식(Parquet)과 행 기반(Avro) 형식을 조합하여 저장됩니다. 업데이트는 행 기반 `delta` 파일에 기록되며 새 버전의 열 형식 파일을 작성할 때 필요에 따라 압축됩니다.

CoW 데이터 세트를 사용하면 레코드에 대한 업데이트가 있을 때마다 레코드가 포함된 파일이 업데이트된 값으로 다시 작성됩니다. MoR 데이터 세트를 사용하면 Hudi는 업데이트가 있을 때마다 변경된 레코드에 대한 행만 씁니다. MoR은 읽기 수행이 적고 쓰기 또는 변경이 많은 워크로드에 더 적합합니다. CoW는 자주 변경되지 않는 데이터에서 읽기 수행이 많은 워크로드에 더 적합합니다.

Hudi는 데이터 액세스를 위해 세 가지 쿼리 유형을 제공합니다.

- 스냅샷 쿼리 - 지정된 커밋 또는 압축 작업 시 테이블의 최신 스냅샷을 보는 쿼리입니다. MoR 테이블의 경우 스냅샷 쿼리는 쿼리 시의 최신 파일 슬라이스의 기본 파일과 델타 파일을 병합하여 테이블의 최신 상태를 나타냅니다.
- 증분의 쿼리 - 이 쿼리는 지정된 커밋/압축 이후 테이블에 기록된 새 데이터만 볼 수 있습니다. 이는 변경 스트림을 효과적으로 제공하여 증분 데이터 파이프라인을 사용할 수 있도록 합니다.
- 읽기 최적화 쿼리 - MoR 테이블의 경우 쿼리가 압축된 최신 데이터를 표시합니다. CoW 테이블의 경우 이 쿼리는 커밋된 최신 데이터를 보여줍니다.

다음 표에는 각 테이블 유형에 사용할 수 있는 Hudi 쿼리 유형이 나와 있습니다.

테이블 유형	가능한 Hudi 쿼리 유형
쓸 때 복사	스냅샷, 증분
읽을 때 병합	스냅샷, 증분, 읽기 최적화

현재 Athena는 스냅샷 쿼리와 읽기 최적화 쿼리를 지원하지만 증분 쿼리는 지원하지 않습니다. MOR 테이블에서 읽기 최적화 쿼리에 노출된 모든 데이터는 압축됩니다. 이는 우수한 성능을 제공하지만 최근의 델타 커밋은 포함하지 않습니다. 스냅샷 쿼리에는 가장 최신의 데이터가 포함되지만 일정한 계산 오버헤드가 발생하여 이러한 쿼리의 성능이 떨어집니다.

테이블 유형과 쿼리 유형 간의 장단점에 대한 자세한 내용은 Apache Hudi 설명서의 Table & Query Types(테이블 및 쿼리 유형)를 참조하세요.

Hudi 용어 변경: 뷰가 이제 쿼리로 변경됨

릴리스 버전 0.5.1부터 Apache Hudi는 용어의 일부를 변경했습니다. 이전에는 뷰라고 불렀던 것을 후속 릴리스에서는 쿼리라고 부릅니다. 다음 표에 이전 용어와 새 용어 간의 변경 내용이 요약되어 있습니다.

이전 용어	새 용어
CoW: 읽기 최적화 뷰 MoR: 실시간 뷰	스냅샷 쿼리
증분 뷰	증분 쿼리
MoR 읽기 최적화 뷰	읽기 최적화 쿼리

부트스트랩 작업의 테이블

Apache Hudi 버전 0.6.0부터 부트스트랩 작업 기능은 기존의 Parquet 데이터 집합에 향상된 성능을 제공합니다. 부트스트랩 작업은 데이터 집합을 다시 작성하는 대신 데이터 집합을 그대로 두고 메타데이터만 생성할 수 있습니다.

Athena를 사용하면 Amazon S3의 데이터를 기반으로 하는 다른 테이블과 마찬가지로 부트스트랩 작업에서 테이블을 쿼리할 수 있습니다. `CREATE TABLE` 문에서 `LOCATION` 절에 Hudi 테이블 경로를 지정합니다.

Amazon EMR에서 부트스트랩 작업을 사용하여 Hudi 테이블을 만드는 방법에 대한 자세한 내용은 AWS Big Data Blog(빅 데이터 블로그)의 New features from Apache Hudi available in Amazon EMR(Amazon EMR에서 사용 가능한 Apache Hudi의 새로운 기능) 항목을 참조하세요.

Hudi 메타데이터 목록

Apache Hudi에는 파일 목록, 열 통계를 사용한 데이터 건너뛰기, 블룸 필터 기반 인덱스와 같은 성능 향상을 위한 인덱싱 기능이 포함된 메타데이터 테이블이 있습니다.

Athena는 현재 이러한 기능 중 파일 목록 인덱스만 지원합니다. 파일 목록 인덱스는 파일 매핑에 대한 파티션을 유지 관리하는 인덱스에서 정보를 가져와 '파일 나열'과 같은 파일 시스템 호출을 제거합니다. 이렇게 하면 파일 시스템을 보기 위해 테이블 경로 아래에 있는 모든 파티션을 재귀적으로 나열할 필요

가 없습니다. 대규모 데이터 세트에 대한 작업을 수행하는 경우 이 인덱싱 기능을 사용하면 쓰기 및 쿼리 중에 파일 목록을 가져올 때 발생할 수 있는 지연 시간이 크게 줄어듭니다. 또한 Amazon S3 LIST 호출에서 요청 한도 제한과 같은 병목 현상을 방지합니다.

> **Note**
>
> Athena는 현재 데이터 건너뛰기 또는 블룸 필터 인덱싱을 지원하지 않습니다.

Hudi 메타데이터 테이블 활성화

메타데이터 테이블 기반 파일 목록은 기본적으로 비활성화되어 있습니다. Hudi 메타데이터 테이블 및 관련 파일 목록 기능을 활성화하려면 `hudi.metadata-listing-enabled` 테이블 속성을 `TRUE`로 설정합니다.

예

다음 `ALTER TABLE SET TBLPROPERTIES` 예제에서는 예제 `partition_cow` 테이블의 메타데이터 테이블을 활성화합니다.

```
ALTER TABLE partition_cow SET TBLPROPERTIES('hudi.metadata-listing-enabled'='TRUE')
```

고려 사항 및 제한

- Athena는 증분 쿼리를 지원하지 않습니다.
- Athena는 Hudi 데이터에 대해 CTAS 또는 INSERT INTO을(를) 지원하지 않습니다. Athena가 Hudi 데이터 집합 쓰기를 지원하길 원하시면 <athena-feedback@amazon.com>에 피드백을 보내세요.

  Hudi 데이터 쓰기에 대한 자세한 내용은 다음 리소스를 참조하세요.

  - Amazon EMR 릴리스 안내서의 Working with a Hudi dataset(Hudi 데이터 집합 작업).
  - Apache Hudi 설명서의 Writing Data(데이터 쓰기).
- Athena에서는 Hudi 테이블에 MSCK REPAIR TABLE을 사용할 수 없습니다. AWS Glue에서 생성되지 않은 Hudi 테이블 로드해야 할 경우 ALTER TABLE ADD PARTITION을 사용하세요.
- S3 Glacier 객체 건너뛰기 미지원 – Apache Hudi 테이블의 객체가 Amazon S3 Glacier 스토리지 클래스에 있는 경우 `read_restored_glacier_objects` 테이블 속성을 `false`로 설정해도 효과가 없습니다.

  예를 들어 다음과 같은 명령을 실행한다고 가정하겠습니다.

```
ALTER TABLE table_name SET TBLPROPERTIES ('read_restored_glacier_objects' = 'false')
```

Iceberg 및 Delta Lake 테이블의 경우 이 명령은 지원되지 않는 테이블 속성 키: read_restored_glacier_objects 오류를 생성합니다. Hudi 테이블의 경우 ALTER TABLE 명령은 오류를 생성하지 않지만 여전히 Amazon S3 Glacier 객체를 건너뛰지 않습니다. ALTER TABLE 명령 후에 SELECT 쿼리를 실행하면 계속해서 모든 개체가 반환됩니다.

비디오

다음 동영상은 Amazon Athena를 사용하여 Amazon S3 기반 데이터 레이크에서 읽기에 최적화된 Apache Hudi 데이터 집합을 쿼리하는 방법을 보여 줍니다.

Amazon Athena를 사용하여 Apache Hudi 데이터 집합 쿼리

Hudi 테이블 생성

이 단원에서는 Hudi 데이터의 분할된 테이블과 분할되지 않은 테이블에 대한 Athena의 CREATE TABLE 문의 예제를 제공합니다.

AWS Glue에서 생성한 Hudi 테이블이 이미 있다면 Athena에서 직접 쿼리할 수 있습니다. Athena에서 분할된 Hudi 테이블을 만들 때 ALTER TABLE ADD PARTITION을 실행하여 Hudi 데이터를 로드해야 쿼리가 가능합니다.

쓸 때 복사(CoW) 테이블 생성 예제

분할되지 않은 CoW 테이블

다음 예제에서는 Athena에서 분할되지 않은 CoW 테이블을 만듭니다.

```
CREATE EXTERNAL TABLE `non_partition_cow`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int,
  `event_type` string)
```

```
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/folder/non_partition_cow/'
```

분할된 CoW 테이블

다음 예제에서는 Athena에서 분할된 CoW 테이블을 만듭니다.

```
CREATE EXTERNAL TABLE `partition_cow`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int)
PARTITIONED BY (
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/folder/partition_cow/'
```

다음 `ALTER TABLE ADD PARTITION` 예제에서는 `partition_cow` 테이블 예제에 2개의 파티션을 추가합니다.

```
ALTER TABLE partition_cow ADD
  PARTITION (event_type = 'one') LOCATION 's3://DOC-EXAMPLE-BUCKET/folder/
partition_cow/one/'
  PARTITION (event_type = 'two') LOCATION 's3://DOC-EXAMPLE-BUCKET/folder/
partition_cow/two/'
```

읽을 때 병합(MoR) 테이블 생성 예제

Hudi는 MoR용 메타스토어에 두 개의 테이블을 만듭니다. 하나는 스냅샷 쿼리를 위한 테이블이고 하나는 읽기 최적화 쿼리를 위한 테이블입니다. 두 테이블 모두 쿼리가 가능합니다. 0.5.1 이전의 Hudi 버전에서는 읽기 최적화 쿼리용 테이블이 해당 테이블을 만들 때 지정한 이름을 갖습니다. Hudi 버전 0.5.1부터는 기본적으로 테이블 이름에 접미사 `_ro`가 붙습니다. 스냅샷 쿼리용 테이블의 이름은 지정한 이름에 `_rt`가 덧붙습니다.

분할되지 않은 읽을 때 병합(MoR) 테이블

다음 예제에서는 Athena에서 읽기 최적화 쿼리를 위한 분할되지 않은 MoR 테이블을 만듭니다. 읽기 최적화 쿼리는 `HoodieParquetInputFormat` 입력 형식을 사용합니다.

```
CREATE EXTERNAL TABLE `nonpartition_mor`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int,
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/folder/nonpartition_mor/'
```

다음 예제에서는 Athena에서 스냅샷 쿼리를 위한 분할되지 않은 MoR 테이블을 만듭니다. 스냅샷 쿼리의 경우 `HoodieParquetRealtimeInputFormat` 입력 형식을 사용합니다.

```
CREATE EXTERNAL TABLE `nonpartition_mor_rt`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
```

```
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int,
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.realtime.HoodieParquetRealtimeInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/folder/nonpartition_mor/'
```

분할된 읽을 때 병합(MoR) 테이블

다음 예제에서는 Athena에서 읽기 최적화 쿼리를 위한 분할된 MoR 테이블을 만듭니다.

```
CREATE EXTERNAL TABLE `partition_mor`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int)
PARTITIONED BY (
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.HoodieParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/folder/partition_mor/'
```

다음 `ALTER TABLE ADD PARTITION` 예제에서는 `partition_mor` 테이블 예제에 2개의 파티션을 추가합니다.

```
ALTER TABLE partition_mor ADD
```

```
  PARTITION (event_type = 'one') LOCATION 's3://DOC-EXAMPLE-BUCKET/folder/
partition_mor/one/'
  PARTITION (event_type = 'two') LOCATION 's3://DOC-EXAMPLE-BUCKET/folder/
partition_mor/two/'
```

다음 예제에서는 Athena에서 스냅샷 쿼리를 위한 분할된 MoR 테이블을 만듭니다.

```
CREATE EXTERNAL TABLE `partition_mor_rt`(
  `_hoodie_commit_time` string,
  `_hoodie_commit_seqno` string,
  `_hoodie_record_key` string,
  `_hoodie_partition_path` string,
  `_hoodie_file_name` string,
  `event_id` string,
  `event_time` string,
  `event_name` string,
  `event_guests` int)
PARTITIONED BY (
  `event_type` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hudi.hadoop.realtime.HoodieParquetRealtimeInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/folder/partition_mor/'
```

마찬가지로 다음 `ALTER TABLE ADD PARTITION` 예제에서는 `partition_mor_rt` 테이블 예제에 2개의 파티션을 추가합니다.

```
ALTER TABLE partition_mor_rt ADD
  PARTITION (event_type = 'one') LOCATION 's3://DOC-EXAMPLE-BUCKET/folder/
partition_mor/one/'
  PARTITION (event_type = 'two') LOCATION 's3://DOC-EXAMPLE-BUCKET/folder/
partition_mor/two/'
```

추가적인 리소스

- AWS Glue 사용자 지정 커넥터 및 AWS Glue 2.0 작업을 사용해 Athena로 쿼리할 수 있는 Apache Hudi 테이블을 만드는 방법에 관한 자세한 내용은 AWS Big Data Blog(빅 데이터 블로그)의 Writing

to Apache Hudi tables using AWS Glue custom connector(사용자 지정 커넥터를 사용하여 Apache Hudi 테이블에 쓰기)를 참조하세요.

- Apache Hudi, AWS Glue 및 Amazon Athena를 사용하여 데이터 레이크용 데이터 처리 프레임워크를 구축하는 방법에 대한 문서는 AWS 빅 데이터 블로그의 Simplify operational data processing in data lakes using AWS Glue and Apache Hudi를 참조하세요.

Apache Iceberg 테이블 사용

Athena는 데이터로 Apache Parquet 형식을 사용하고 메타스토어로 AWS Glue 카탈로그를 사용하는 Apache Iceberg 테이블에 대한 읽기, 시간 이동 및 쓰기 및 DDL 쿼리를 지원합니다.

Apache Iceberg는 매우 큰 분석 데이터 세트를 위한 오픈 테이블 형식입니다. Iceberg는 대규모 파일 모음을 테이블로 관리하며 레코드 수준 삽입, 업데이트, 삭제 및 시간 이동 쿼리와 같은 최신 분석 데이터 레이크 작업을 지원합니다. Iceberg 사양은 스키마 및 파티션 진화와 같은 원활한 테이블 진화를 허용하며, Amazon S3에서 사용하기에 최적화되도록 설계되었습니다. 또한 Iceberg는 동시 쓰기 시나리오에서 데이터 정확성을 보장하는 데 도움이 됩니다.

Apache Iceberg에 대한 자세한 내용은 https://iceberg.apache.org/를 참조하세요.

고려 사항 및 제한

Iceberg 테이블에 대한 Athena 지원에는 다음과 같은 고려 사항 및 제한 사항이 있습니다.

- Iceberg 버전 지원 - Athena는 Apache Iceberg 버전 1.4.2를 지원합니다.
- AWS Glue 카탈로그를 포함한 테이블만 해당 - Athena는 오픈 소스 Glue 카탈로그 구현에 정의된 사양을 기반으로 AWS Glue 카탈로그에 대해 생성된 Iceberg 테이블만 지원합니다.
- AWS Glue에 의한 테이블 잠금만 지원 - 플러그인 사용자 지정 잠금을 지원하는 오픈 소스 Glue 카탈로그 구현과 달리 Athena는 AWS Glue 낙관적 잠금만 지원합니다. Athena를 사용하여 다른 잠금 구현으로 Iceberg 테이블을 수정하면 잠재적 데이터 손실 및 트랜잭션 중단이 발생할 수 있습니다.
- 지원되는 파일 형식 - Athena의 Iceberg 파일 형식 지원은 다음 표와 같이 Athena 엔진 버전에 따라 다릅니다.

Athena 엔진 버전	PARQUET	ORC	Avro
2	예	아니요	아니요
3	예	예	예

- Iceberg v2 테이블 - Athena는 Iceberg v2 테이블에서만 생성하고 작동합니다. v1과 v2 테이블의 차이점은 Apache Iceberg 설명서의 Format version changes(포맷 버전 변경 사항)을 참조하세요.
- 시간대가 없는 시간 유형 표시 - 시간대 유형이 없는 시간 및 타임스탬프는 UTC로 표시됩니다. 시간 열의 필터 표현식에 표준 시간대가 지정되지 않은 경우 UTC가 사용됩니다.
- 타임스탬프 관련 데이터 정밀도 - Iceberg는 타임스탬프 데이터 형식에 대해 마이크로초 정밀도를 지원하지만, Athena는 읽기 및 쓰기 모두에서 타임스탬프에 대해 밀리초 정밀도만 지원합니다. Athena는 수동 압축 작업 중에 다시 작성된 시간 관련 열의 데이터에 대해 밀리초 정밀도만 유지합니다.
- 지원되지 않는 작업 - 다음 Athena 작업은 Iceberg 테이블에서 지원되지 않습니다.
  - ALTER TABLE SET LOCATION
- 보기 - 뷰 작업에 설명된 대로 `CREATE VIEW`를 사용하여 Athena 보기를 생성합니다. Iceberg 보기 사양을 사용하여 보기를 생성하려면 athena-feedback@amazon.com에 문의하세요.
- TTF 관리 명령이 AWS Lake Formation에서 지원되지 않음 - Lake Formation을 사용하여 Apache Iceberg, Apache Hudi 및 Linux Foundation Delta Lake와 같은 TransactionTable Formats(TTF)에 대한 읽기 액세스 권한을 관리할 수는 있지만, Lake Formation을 사용하여 이러한 테이블 형식에서 `VACUUM`, `MERGE`, `UPDATE` 또는 `OPTIMIZE`와 같은 작업에 대한 권한을 관리할 수는 없습니다. Athena와 Lake Formation의 통합에 대한 자세한 내용은 AWS Lake Formation 개발자 안내서의 Using AWS Lake Formation with Amazon Athena를 참조하세요.
- 중첩 필드를 기준으로 파티셔닝 - 중첩 필드에 기반한 파티셔닝은 지원되지 않습니다. 이를 시도하면 다음 메시지가 나타납니다. NOT_SUPPORTED: Partitioning by nested field is unsupported: *`column_name.nested_field_name`*.
- S3 Glacier 객체 건너뛰기 미지원 – Apache Iceberg 테이블의 객체가 Amazon S3 Glacier 스토리지 클래스에 있는 경우 `read_restored_glacier_objects` 테이블 속성을 `false`로 설정해도 효과가 없습니다.

  예를 들어 다음과 같은 명령을 실행한다고 가정하겠습니다.

  ```
  ALTER TABLE table_name SET TBLPROPERTIES ('read_restored_glacier_objects' = 'false')
  ```

  Iceberg 및 Delta Lake 테이블의 경우 이 명령은 지원되지 않는 테이블 속성 키: read_restored_glacier_objects 오류를 생성합니다. Hudi 테이블의 경우 `ALTER TABLE` 명령은 오류를 생성하지 않지만 여전히 Amazon S3 Glacier 객체를 건너뛰지 않습니다. `ALTER TABLE` 명령 후에 `SELECT` 쿼리를 실행하면 계속해서 모든 개체가 반환됩니다.

Athena가 특정 기능을 지원하도록 하려면 athena-feedback@amazon.com으로 피드백을 보내 주세요.

주제

Iceberg 테이블 생성

Athena에서 사용할 Iceberg 테이블을 생성하려면 이 페이지에서 설명하는 `CREATE TABLE` 문을 사용하거나 AWS Glue 크롤러를 사용할 수 있습니다.

CREATE TABLE 문 사용

Athena는 Iceberg v2 테이블을 생성합니다. v1과 v2 테이블의 차이점은 Apache Iceberg 설명서의 Format version changes(포맷 버전 변경 사항)을 참조하세요.

Athena `CREATE TABLE`은 데이터가 없는 Iceberg 테이블을 생성합니다. 테이블이 Iceberg 오픈 소스 Glue 카탈로그를 사용하는 경우 Apache Spark와 같은 외부 시스템에서 직접 테이블을 쿼리할 수 있습니다. 외부 테이블을 만들 필요는 없습니다.

> **Warning**
>
> `CREATE EXTERNAL TABLE`을 실행하면 오류 메시지 External keyword not supported for table type ICEBERG가 표시됩니다.

Athena에서 Iceberg 테이블을 생성하려면 다음 구문 요약과 같이 `TBLPROPERTIES` 절에서 `'table_type'` 테이블 속성을 `'ICEBERG'`로 설정하세요.

```
CREATE TABLE
```

```
  [db_name.]table_name (col_name data_type [COMMENT col_comment] [, ...] )
  [PARTITIONED BY (col_name | transform, ... )]
  LOCATION 's3://DOC-EXAMPLE-BUCKET/your-folder/'
  TBLPROPERTIES ( 'table_type' ='ICEBERG' [, property_name=property_value] )
```

Iceberg 테이블에서 쿼리할 수 있는 데이터 형식에 대한 자세한 내용은 Athena에서 Iceberg 테이블에 대해 지원되는 데이터 형식 섹션을 참조하세요.

분할

파티션이 있는 Iceberg 테이블을 생성하려면 PARTITIONED BY 구문을 사용합니다. 분할에 사용되는 열은 열 선언에서 먼저 지정해야 합니다. PARTITIONED BY 절 내에서는 열 유형을 포함하지 않아야 합니다. CREATE TABLE 구문에 파티션 변환을 정의할 수도 있습니다. 분할을 위해 여러 열을 지정하려면 다음 예와 같이 열을 쉼표(,)로 구분합니다.

```
CREATE TABLE iceberg_table (id bigint, data string, category string)
  PARTITIONED BY (category, bucket(16, id))
  LOCATION 's3://DOC-EXAMPLE-BUCKET/your-folder/'
  TBLPROPERTIES ( 'table_type' = 'ICEBERG' )
```

다음 표에서는 사용 가능한 파티션 변환 함수를 보여줍니다.

함수	설명	지원되는 유형
year(ts)	연도별 파티션	date, timestamp
month(ts)	월별 파티션	date, timestamp
day(ts)	일별 파티션	date, timestamp
hour(ts)	시간별 파티션	timestamp
bucket(*N*, col)	해시된 값 모드(Mod) *N*개 버킷별 파티션 Hive 테이블의 해시 버킷팅과 같은 개념입니다.	int, long, decimal, date, timestamp , string, binary
truncate(*L*, col)	*L*로 잘린 값별 파티션	int, long, decimal, string

Athena는 Iceberg의 숨겨진 분할을 지원합니다. 자세한 내용은 Apache Iceberg 설명서의 Iceberg의 숨겨진 분할(Iceberg's hidden partitioning)을 참조하세요.

테이블 속성

이 섹션에서는 `CREATE TABLE` 문의 `TBLPROPERTIES` 절에 키-값 페어로 지정할 수 있는 테이블 속성에 대해 설명합니다. Athena는 Iceberg 테이블을 만들거나 변경하기 위해 테이블 속성에 미리 정의된 키-값 페어의 목록만 허용합니다. 다음 테이블에서는 지정할 수 있는 테이블 속성을 보여줍니다. 압축 옵션에 대한 자세한 내용은 이 문서의 Iceberg 테이블 최적화를 참조하세요. Athena가 특정 오픈 소스 테이블 구성 속성을 지원하도록 하려면 athena-feedback@amazon.com으로 피드백을 보내세요.

format

설명	파일 데이터 형식
허용되는 속성 값	지원되는 파일 형식 및 압축 조합은 Athena 엔진 버전에 따라 다릅니다. 자세한 내용은 파일 형식별 Iceberg 테이블 압축 지원 단원을 참조하십시오.
기본값	parquet

write_compression

설명	파일 압축 코덱
허용되는 속성 값	지원되는 파일 형식 및 압축 조합은 Athena 엔진 버전에 따라 다릅니다. 자세한 내용은 파일 형식별 Iceberg 테이블 압축 지원 단원을 참조하십시오.
기본값	기본 쓰기 압축은 Athena 엔진 버전에 따라 다릅니다. 자세한 내용은 파일 형식별 Iceberg 테이블 압축 지원 단원을 참조하십시오.

optimize_rewrite_data_file_threshold

설명	데이터 최적화별 구성입니다. 최적화가 필요한 데이터 파일이 주어진 임계값보다 적으면 파일이 재작성되지 않습니다. 이를 통해 더 많은 데이터 파

	일을 누적하여 대상 크기에 더 가까운 파일을 생성하고 비용 절감을 위해 불필요한 계산을 건너뛸 수 있습니다.
허용되는 속성 값	양수. 50 미만이어야 합니다.
기본값	5

optimize_rewrite_delete_file_threshold

설명	데이터 최적화별 구성입니다. 데이터 파일과 연관된 삭제 파일이 임계값보다 적으면 데이터 파일이 재작성되지 않습니다. 이를 통해 비용 절감을 위해 각 데이터 파일에 대해 더 많은 삭제 파일을 누적할 수 있습니다.
허용되는 속성 값	양수. 50 미만이어야 합니다.
기본값	2

vacuum_min_snapshots_to_keep

설명	테이블의 기본 브랜치에 유지할 최소 스냅샷 수입니다. 이 값은 vacuum_max_snapshot_age_seconds 속성보다 우선합니다. 남은 최소 스냅샷이 vacuum_max_snapshot_age_seconds 에서 지정한 수명보다 오래된 경우 스냅샷이 보관되고 vacuum_max_snapshot_age_seconds 의 값은 무시됩니다.
허용되는 속성 값	양수.
기본값	1

vacuum_max_snapshot_age_seconds

설명	기본 브랜치에 유지할 스냅샷의 최대 수명입니다. vacuum_min_snapshots_to_keep 에서 지정한 남은 최소 스냅샷이 지정된 수명보다 오래된 경우 이 값은 무시됩니다. 이 테이블 동작 속성은 Apache

	Iceberg 구성의 `history.expire.max-snapshot-age-ms` 속성에 해당합니다.
허용되는 속성 값	양수.
기본값	432000초(5일)

vacuum_max_metadata_files_to_keep

설명	테이블의 기본 브랜치에 유지할 최대 이전 메타데이터 파일 수입니다.
허용되는 속성 값	양수.
기본값	100

CREATE TABLE 문 예제

다음 예에서는 3개의 열이 있는 Iceberg 테이블을 생성합니다.

```
CREATE TABLE iceberg_table (
  id int,
  data string,
  category string)
PARTITIONED BY (category, bucket(16,id))
LOCATION 's3://DOC-EXAMPLE-BUCKET/iceberg-folder'
TBLPROPERTIES (
  'table_type'='ICEBERG',
  'format'='parquet',
  'write_compression'='snappy',
  'optimize_rewrite_delete_file_threshold'='10'
)
```

CREATE TABLE AS SELECT(CTAS)

`CREATE TABLE AS` 문을 사용하여 Iceberg 테이블을 생성하는 방법에 대한 자세한 내용은 CREATE TABLE AS를 참조하세요. 특히 CTAS 테이블 속성 섹션에 주의하세요.

AWS Glue 크롤러 사용

AWS Glue 크롤러를 사용하여 Iceberg 테이블을 AWS Glue Data Catalog에 자동으로 등록할 수 있습니다. 다른 Iceberg 카탈로그에서 마이그레이션하려는 경우 AWS Glue 크롤러를 생성 및 예약하고 Iceberg 테이블이 위치한 Amazon S3 경로를 제공할 수 있습니다. AWS Glue 크롤러가 통과할 수 있는 Amazon S3 경로의 최대 깊이를 지정할 수 있습니다. AWS Glue 크롤러를 예약한 후 실행할 때마다 크롤러가 스키마 정보를 추출하고 스키마 변경 내용으로 AWS Glue Data Catalog를 업데이트합니다. AWS Glue 크롤러는 스냅샷 간 스키마 병합을 지원하고 AWS Glue Data Catalog의 최신 메타데이터 파일 위치를 업데이트합니다. 자세한 내용은 Data Catalog and crawlers in AWS Glue를 참조하세요.

Iceberg 테이블 관리

Athena는 Iceberg 테이블에 관한 다음 테이블 DDL 작업을 지원합니다.

ALTER TABLE RENAME

테이블의 이름을 바꿉니다.

Iceberg 테이블의 테이블 메타데이터는 Amazon S3에 저장되므로 기본 테이블 정보에 영향을 주지 않고 Iceberg 관리 테이블의 데이터베이스와 테이블 이름을 업데이트할 수 있습니다.

시놉시스

```
ALTER TABLE [db_name.]table_name RENAME TO [new_db_name.]new_table_name
```

예

```
ALTER TABLE my_db.my_table RENAME TO my_db2.my_table2
```

ALTER TABLE SET PROPERTIES

Iceberg 테이블에 속성을 추가하고 할당된 값을 설정합니다.

Iceberg 사양에 따라 테이블 속성은 AWS Glue가 아닌 Iceberg 테이블 메타데이터 파일에 저장됩니다. Athena는 사용자 정의 테이블 속성을 허용하지 않습니다. 허용되는 키-값 페어는 테이블 속성 섹션을 참조하세요. Athena가 특정 오픈 소스 테이블 구성 속성을 지원하도록 하려면 athena-feedback@amazon.com으로 피드백을 보내세요.

시놉시스

```
ALTER TABLE [db_name.]table_name SET TBLPROPERTIES ('property_name' =
 'property_value' [ , ... ])
```

예

```
ALTER TABLE iceberg_table SET TBLPROPERTIES (
  'format'='parquet',
  'write_compression'='snappy',
  'optimize_rewrite_delete_file_threshold'='10'
)
```

ALTER TABLE UNSET PROPERTIES

Iceberg 테이블에서 기존 속성을 삭제합니다.

시놉시스

```
ALTER TABLE [db_name.]table_name UNSET TBLPROPERTIES ('property_name' [ , ... ])
```

예

```
ALTER TABLE iceberg_table UNSET TBLPROPERTIES ('write_compression')
```

DESCRIBE TABLE

테이블 정보를 설명합니다.

시놉시스

```
DESCRIBE [FORMATTED] [db_name.]table_name
```

FORMATTED 옵션을 지정하면 테이블 위치 및 속성과 같은 추가 정보가 출력에 표시됩니다.

예

```
DESCRIBE iceberg_table
```

DROP TABLE

Iceberg 테이블을 삭제합니다.

> **Warning**
>
> Iceberg 테이블은 Athena에서 관리형 테이블로 간주되므로 Iceberg 테이블을 삭제하면 테이블의 모든 데이터도 제거됩니다.

시놉시스

```
DROP TABLE [IF EXISTS] [db_name.]table_name
```

예

```
DROP TABLE iceberg_table
```

SHOW CREATE TABLE

Athena에서 Iceberg 테이블을 다시 생성하는 데 사용할 수 있는 CREATE TABLE DDL 문을 표시합니다. Athena가 테이블 구조를 재현할 수 없는 경우(예: 테이블에 사용자 정의 테이블 속성이 지정됨) UNSUPPORTED 오류가 발생합니다.

시놉시스

```
SHOW CREATE TABLE [db_name.]table_name
```

예

```
SHOW CREATE TABLE iceberg_table
```

SHOW TABLE PROPERTIES

Iceberg 테이블의 하나 이상의 테이블 속성을 표시합니다. Athena에서 지원되는 테이블 속성만 표시됩니다.

시놉시스

```
SHOW TBLPROPERTIES [db_name.]table_name [('property_name')]
```

예

```
SHOW TBLPROPERTIES iceberg_table
```

Iceberg 테이블 메타데이터 쿼리

SELECT 쿼리에서 *table_name* 뒤에 다음 속성을 사용하여 Iceberg 테이블 메타데이터를 쿼리할 수 있습니다.

- $files - 테이블의 현재 데이터 파일을 표시합니다.
- $manifests - 테이블의 현재 파일 매니페스트를 표시합니다.
- $history – 테이블의 기록을 표시합니다.
- $partitions - 테이블의 현재 파티션을 표시합니다.
- $snapshots - 테이블의 스냅샷을 보여줍니다.
- $refs - 테이블의 참조를 표시합니다.

구문

다음 명령문은 Iceberg 테이블의 파일을 나열합니다.

```
SELECT * FROM "dbname"."tablename$files"
```

다음 명령문은 Iceberg 테이블의 매니페스트를 나열합니다.

```
SELECT * FROM "dbname"."tablename$manifests"
```

다음 명령문은 Iceberg 테이블의 기록을 표시합니다.

```
SELECT * FROM "dbname"."tablename$history"
```

다음 예제는 Iceberg 테이블의 파티션을 표시합니다.

```
SELECT * FROM "dbname"."tablename$partitions"
```

다음 예제는 Iceberg 테이블의 스냅샷을 나열합니다.

```
SELECT * FROM "dbname"."tablename$snapshots"
```

다음 예제는 Iceberg 테이블의 참조를 표시합니다.

```
SELECT * FROM "dbname"."tablename$refs"
```

변화하는 Iceberg 테이블 스키마

Iceberg 스키마 업데이트는 메타데이터 전용 변경 사항입니다. 스키마 업데이트를 수행할 때 데이터 파일은 변경되지 않습니다.

Iceberg 형식은 다음과 같은 스키마 진화 변경을 지원합니다.

- 추가 - 테이블 또는 중첩된 `struct`에 새 열을 추가합니다.
- 삭제 - 테이블이나 중첩된 `struct`에서 기존 열을 제거합니다.
- 이름 바꾸기 - 중첩된 `struct`에서 기존 열 또는 필드의 이름을 바꿉니다.
- 재정렬 - 열의 순서를 변경합니다.
- 형식 승격 - 열, `struct` 필드, `map` 키, `map` 값 또는 `list` 요소의 형식을 확장합니다. 현재 Iceberg 테이블에 대해 다음과 같은 경우를 지원합니다.
  - integer에서 big integer로
  - float에서 double로
  - decimal 형식의 정밀도 증가

ALTER TABLE ADD COLUMNS

기존 Iceberg 테이블에 하나 이상의 열을 추가합니다.

시놉시스

```
ALTER TABLE [db_name.]table_name ADD COLUMNS (col_name data_type [,...])
```

예제

다음 예제는 `string` 형식의 `comment` 열을 Iceberg 테이블에 추가합니다.

```
ALTER TABLE iceberg_table ADD COLUMNS (comment string)
```

다음 예제는 `struct` 형식의 `point` 열을 Iceberg 테이블에 추가합니다.

```
ALTER TABLE iceberg_table
ADD COLUMNS (point struct<x: double, y: double>)
```

다음 예제는 구조체 배열인 `points` 열을 Iceberg 테이블에 추가합니다.

```
ALTER TABLE iceberg_table
ADD COLUMNS (points array<struct<x: double, y: double>>)
```

ALTER TABLE DROP COLUMN

기존 Iceberg 테이블에서 열을 삭제합니다.

시놉시스

```
ALTER TABLE [db_name.]table_name DROP COLUMN col_name
```

예

```
ALTER TABLE iceberg_table DROP COLUMN userid
```

ALTER TABLE CHANGE COLUMN

열의 이름, 형식, 순서 또는 설명을 변경합니다.

> **Note**
>
> `ALTER TABLE REPLACE COLUMNS`는 지원되지 않습니다. `REPLACE COLUMNS`는 모든 열을 제거한 다음 새 열을 추가하므로 Iceberg에서 지원되지 않습니다. `CHANGE COLUMN`은 스키마 변화에 선호되는 구문입니다.

시놉시스

```
ALTER TABLE [db_name.]table_name
  CHANGE [COLUMN] col_old_name col_new_name column_type
  [COMMENT col_comment] [FIRST|AFTER column_name]
```

예

```
ALTER TABLE iceberg_table CHANGE comment blog_comment string AFTER id
```

SHOW COLUMNS

테이블의 열을 표시합니다.

시놉시스

```
SHOW COLUMNS (FROM|IN) [db_name.]table_name
```

예

```
SHOW COLUMNS FROM iceberg_table
```

Iceberg 테이블 데이터 쿼리 및 시간 이동 수행

Iceberg 데이터 세트를 쿼리하려면 다음과 같이 표준 SELECT 문을 사용합니다. 쿼리는 Apache Iceberg 포맷 v2 사양을 따르며 위치 및 동일 삭제 모두에 대해 병합 시 읽기 작업을 수행합니다.

```
SELECT * FROM [db_name.]table_name [WHERE predicate]
```

쿼리 시간을 최적화하기 위해 일부 술어는 데이터가 존재하는 곳으로 푸시다운됩니다.

시간 이동 및 버전 이동 쿼리

각 Apache Iceberg 테이블은 포함하는 Amazon S3 객체의 버전이 지정된 매니페스트를 유지 관리합니다. 이전 버전의 매니페스트는 시간 이동 및 버전 이동 쿼리에 사용할 수 있습니다.

Athena 시간 이동 쿼리는 지정된 날짜 및 시간을 기준으로 일관된 스냅샷의 기록 데이터에 대해 Amazon S3를 쿼리합니다. Athena의 버전 이동 쿼리는 지정된 스냅샷 ID를 기준으로 기록 데이터에 대해 Amazon S3를 쿼리합니다.

시간 이동 쿼리

시간 이동 쿼리를 실행하려면 다음 예와 같이 SELECT 문의 테이블 이름 다음에 FOR TIMESTAMP AS OF *timestamp*를 사용합니다.

```
SELECT * FROM iceberg_table FOR TIMESTAMP AS OF timestamp
```

이동을 위해 지정할 시스템 시간은 타임스탬프 또는 시간대가 있는 타임스탬프입니다. 지정하지 않으면 Athena는 이 값을 UTC 시간의 타임스탬프로 간주합니다.

다음 예제 시간 이동 쿼리는 지정된 날짜 및 시간에 대한 CloudTrail 데이터를 선택합니다.

```
SELECT * FROM iceberg_table FOR TIMESTAMP AS OF TIMESTAMP '2020-01-01 10:00:00 UTC'
```

```
SELECT * FROM iceberg_table FOR TIMESTAMP AS OF (current_timestamp - interval '1' day)
```

버전 이동 쿼리

버전 이동 쿼리를 실행하려면(즉, 지정된 버전을 기준으로 일관된 스냅샷 뷰) 다음 예와 같이 SELECT 문의 테이블 이름 다음에 FOR VERSION AS OF *version*을 사용합니다.

```
SELECT * FROM [db_name.]table_name FOR VERSION AS OF version
```

파라미터는 Iceberg 테이블 버전과 연결된 bigint 스냅샷 ID입니다.

다음 예제 버전 이동 쿼리는 지정된 버전에 대한 데이터를 선택합니다.

```
SELECT * FROM iceberg_table FOR VERSION AS OF 949530903748831860
```

> **Note**
>
> Athena 엔진 버전 2의 FOR SYSTEM_TIME AS OF 및 FOR SYSTEM_VERSION AS OF 절은 Athena 엔진 버전 3에서 FOR TIMESTAMP AS OF 및 FOR VERSION AS OF 절로 대체되었습니다.

스냅샷 ID 검색

Iceberg에서 Java SnapshotUtil 클래스를 사용하여 다음 예제와 같이 Iceberg 스냅샷 ID를 검색할 수 있습니다.

```
import org.apache.iceberg.Table;
```

```
import org.apache.iceberg.aws.glue.GlueCatalog;
import org.apache.iceberg.catalog.TableIdentifier;
import org.apache.iceberg.util.SnapshotUtil;

import java.text.SimpleDateFormat;
import java.util.Date;

Catalog catalog = new GlueCatalog();

Map<String, String> properties = new HashMap<String, String>();
properties.put("warehouse", "s3://DOC-EXAMPLE-BUCKET/my-folder");
catalog.initialize("my_catalog", properties);

Date date = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss").parse("2022/01/01 00:00:00");
long millis = date.getTime();

TableIdentifier name = TableIdentifier.of("db", "table");
Table table = catalog.loadTable(name);
long oldestSnapshotIdAfter2022 = SnapshotUtil.oldestAncestorAfter(table, millis);
```

시간 및 버전 이동 결합

다음 예와 같이 동일한 쿼리에서 시간 이동 및 버전 이동 구문을 사용하여 서로 다른 타이밍 및 버전 관리 조건을 지정할 수 있습니다.

```
SELECT table1.*, table2.* FROM
  [db_name.]table_name FOR TIMESTAMP AS OF (current_timestamp - interval '1' day) AS
 table1
  FULL JOIN
  [db_name.]table_name FOR VERSION AS OF 5487432386996890161 AS table2
  ON table1.ts = table2.ts
  WHERE (table1.id IS NULL OR table2.id IS NULL)
```

Iceberg 테이블에서 보기 생성 및 쿼리하기

Iceberg 테이블에서 Athena 보기를 생성하고 쿼리하려면 뷰 작업에 설명된 대로 `CREATE VIEW` 보기를 사용합니다.

예제

```
CREATE VIEW view1 AS SELECT * FROM iceberg_table
```

```
SELECT * FROM view1
```

Iceberg 보기 사양을 사용하여 보기를 생성하려면 athena-feedback@amazon.com에 문의하세요.

Lake Formation 세분화된 액세스 제어 작업

Athena 엔진 버전 3에서는 열 수준 및 행 수준 보안 액세스 제어를 포함하여 Iceberg 테이블을 통한 Lake Formation 세분화된 액세스 제어를 지원합니다. 이 액세스 제어는 시간 이동 쿼리 및 스키마 진화를 수행한 테이블과 함께 작동합니다. 자세한 내용은 Lake Formation 세분화된 액세스 제어 및 Athena 작업 그룹 단원을 참조하십시오.

Athena 외부에서 Iceberg 테이블을 생성한 경우 Iceberg 테이블 열 정보가 AWS Glue Data Catalog에 채워지도록 Apache Iceberg SDK 버전 0.13.0 이상을 사용하세요. Iceberg 테이블에 AWS Glue의 열 정보가 포함되어 있지 않은 경우 Athena ALTER TABLE SET PROPERTIES 문 또는 최신 Iceberg SDK를 사용하여 AWS Glue에서 테이블을 수정하고 열 정보를 업데이트할 수 있습니다.

Iceberg 테이블 데이터 업데이트

Iceberg 테이블 데이터는 INSERT, UPDATE 및 DELETE 쿼리를 사용하여 Athena에서 직접 관리할 수 있습니다. 각 데이터 관리 트랜잭션은 시간 이동을 사용하여 쿼리할 수 있는 새 스냅샷을 생성합니다. UPDATE 및 DELETE 문은 Iceberg 형식 v2 행 수준 위치 삭제 사양 및 스냅샷 격리 적용을 따릅니다.

다음 명령을 사용하여 Iceberg 테이블에서 데이터 관리 작업을 수행합니다.

INSERT INTO

Iceberg 테이블에 데이터를 삽입합니다. Athena Iceberg INSERT INTO는 스캔 데이터의 양에 따라 외부 Hive 테이블에 대한 현재 INSERT INTO 쿼리와 동일한 요금이 부과됩니다. Iceberg 테이블에 데이터를 삽입하려면 다음 구문을 사용합니다. 여기서 *query*는 VALUES (val1, val2, ...) 또는 SELECT (col1, col2, …) FROM [*db_name*.]*table_name* WHERE *predicate*일 수 있습니다. SQL 구문 및 의미 체계 세부 정보는 INSERT INTO 단원을 참조하세요.

```
INSERT INTO [db_name.]table_name [(col1, col2, …)] query
```

다음 예에서는 테이블 iceberg_table에 값을 삽입합니다.

```
INSERT INTO iceberg_table VALUES (1,'a','c1')
```

```
INSERT INTO iceberg_table (col1, col2, ...) VALUES (val1, val2, ...)
```

```
INSERT INTO iceberg_table SELECT * FROM another_table
```

DELETE

Athena Iceberg DELETE는 Iceberg 위치 삭제 파일을 테이블에 씁니다. 이는 읽기 중 병합(merge-on-read) 삭제라고 알려져 있습니다. 쓸 때 복사(copy-on-write) 삭제와 달리 읽기 중 병합 삭제는 파일 데이터를 다시 쓰지 않기 때문에 더 효율적입니다. Athena가 Iceberg 데이터를 읽을 때 Iceberg 위치 삭제 파일을 데이터 파일과 병합하여 테이블의 최신 뷰를 생성합니다. 이러한 위치 삭제 파일을 제거하려면 REWRITE DATA 압축 작업을 실행할 수 있습니다. DELETE 작업은 스캔 데이터의 양에 따라 요금이 부과됩니다. 구문은 DELETE 단원을 참조하세요.

다음 예에서는 iceberg_table에서 category 값이 c3인 행을 삭제합니다.

```
DELETE FROM iceberg_table WHERE category='c3'
```

UPDATE

Athena Iceberg UPDATE는 Iceberg 위치 삭제 파일과 새로 업데이트된 행을 데이터 파일로 동일한 트랜잭션에 씁니다. UPDATE는 INSERT INTO 및 DELETE의 조합으로 가정할 수 있습니다. UPDATE 작업은 스캔 데이터의 양에 따라 요금이 부과됩니다. 구문은 UPDATE 단원을 참조하세요.

다음 예에서는 테이블 iceberg_table에서 지정된 값을 업데이트합니다.

```
UPDATE iceberg_table SET category='c2' WHERE category='c1'
```

MERGE INTO

Iceberg 테이블에 행을 조건부로 업데이트, 삭제 또는 삽입합니다. 단일 명령문으로 업데이트, 삭제 및 삽입 작업을 결합할 수 있습니다. 구문은 MERGE INTO 단원을 참조하세요.

> **Note**
>
> MERGE INTO은 트랜잭션이며 Athena 엔진 버전 3의 Apache Iceberg 테이블에 대해서만 지원됩니다.

다음 예에서는 s 소스 테이블에 있는 t 테이블에서 모든 고객을 삭제합니다.

```
MERGE INTO accounts t USING monthly_accounts_update s
```

```
ON t.customer = s.customer
WHEN MATCHED
THEN DELETE
```

다음 예에서는 s 소스 테이블의 고객 정보로 t 대상 테이블을 업데이트합니다. s 테이블에 t 테이블의 고객 행과 일치하는 항목이 있는 경우 이 예에서는 테이블 t의 구매 수를 늘립니다. t 테이블에 s 테이블의 고객 행과 일치하는 항목이 없는 경우 이 예에서는 s 테이블의 고객 행을 t 테이블에 삽입합니다.

```
MERGE INTO accounts t USING monthly_accounts_update s
    ON (t.customer = s.customer)
    WHEN MATCHED
        THEN UPDATE SET purchases = s.purchases + t.purchases
    WHEN NOT MATCHED
        THEN INSERT (customer, purchases, address)
              VALUES(s.customer, s.purchases, s.address)
```

다음 예에서는 s 소스 테이블의 고객 정보로 t 대상 테이블을 조건부로 업데이트합니다. 이 예에서는 소스 주소가 Centreville인 일치하는 대상 행을 모두 삭제합니다. 모든 다른 일치하는 행의 경우 이 예에서는 소스 구매를 추가하고 대상 주소를 소스 주소로 설정합니다. 대상 테이블에 일치하는 항목이 없는 경우 이 예에서는 소스 테이블의 행을 삽입합니다.

```
MERGE INTO accounts t USING monthly_accounts_update s
    ON (t.customer = s.customer)
    WHEN MATCHED AND s.address = 'Centreville'
        THEN DELETE
    WHEN MATCHED
        THEN UPDATE
            SET purchases = s.purchases + t.purchases, address = s.address
    WHEN NOT MATCHED
        THEN INSERT (customer, purchases, address)
              VALUES(s.customer, s.purchases, s.address)
```

Iceberg 테이블 최적화

Iceberg 테이블에 데이터가 누적되면 파일을 여는 데 필요한 처리 시간이 늘어나기 때문에 쿼리의 효율성이 점차 떨어집니다. 테이블에 파일 삭제(delete files)가 포함된 경우 추가 컴퓨팅 비용이 발생합니다. Iceberg에서 파일 삭제는 행 수준 삭제이며 엔진은 삭제된 행을 쿼리 결과에 적용해야 합니다.

Iceberg 테이블의 쿼리 성능을 최적화하기 위해 Athena는 테이블 유지 관리 명령으로 수동 압축을 지원합니다. 압축은 테이블 내용을 변경하지 않고 테이블의 구조 레이아웃을 최적화합니다.

OPTIMIZE

`OPTIMIZE table REWRITE DATA` 압축 작업은 관련 삭제 파일의 크기와 수에 따라 데이터 파일을 보다 최적화된 레이아웃으로 재작성합니다. 구문 및 테이블 속성에 대한 자세한 내용은 OPTIMIZE 단원을 참조하세요.

예

다음 예에서는 삭제 파일을 데이터 파일로 병합하고 `category` 값이 `c1`인 대상 파일 크기에 가까운 파일을 생성합니다.

```
OPTIMIZE iceberg_table REWRITE DATA USING BIN_PACK
  WHERE category = 'c1'
```

VACUUM

`VACUUM`에서는 스냅샷 만료 및 분리된 파일 제거를 수행합니다. 이러한 작업은 메타데이터 크기를 줄이고 테이블에 지정된 보존 기간보다 오래된 현재 테이블 상태가 아닌 파일을 제거합니다. 구문에 대한 자세한 내용은 VACUUM 단원을 참조하세요.

예

다음 예에서는 테이블 속성을 사용하여 지난 3일간의 데이터를 유지하도록 `iceberg_table` 테이블을 구성한 다음 `VACUUM`을 사용하여 이전 스냅샷을 만료하고 테이블에서 분리된 파일을 제거합니다.

```
ALTER TABLE iceberg_table SET TBLPROPERTIES (
  'vacuum_max_snapshot_age_seconds'='259200'
)

VACUUM iceberg_table
```

Athena에서 Iceberg 테이블에 대해 지원되는 데이터 형식

Athena는 다음 데이터 형식이 포함된 Iceberg 테이블을 쿼리할 수 있습니다.

```
binary
boolean
date
decimal
double
```

```
float
int
list
long
map
string
struct
timestamp without time zone
```

Iceberg 테이블 유형에 대한 자세한 내용은 Apache 설명서의 schemas page for Iceberg(Iceberg에 대한 스키마 페이지)를 참조하세요.

다음 표에서는 Athena 데이터 형식과 Iceberg 테이블 데이터 형식 간의 관계를 보여줍니다.

Iceberg 형식	Athena 형식	참고
boolean	boolean	
-	tinyint	Athena의 Iceberg 테이블에는 지원되지 않습니다.
-	smallint	Athena의 Iceberg 테이블에는 지원되지 않습니다.
int	int	Athena DML 문에서 이 형식은 INTEGER입니다.
long	bigint	
double	double	
float	float	
decimal(P, S)	decimal(P, S)	P는 정밀도이며, S는 스케일입니다.
-	char	Athena의 Iceberg 테이블에는 지원되지 않습니다.
string	string	Athena DML 문에서 이 형식은 VARCHAR입니다.
binary	binary	
date	date	

Iceberg 형식	Athena 형식	참고
`time`	-	Iceberg 타임스탬프(시간대 없음)는 `CREATE TABLE`과 같은 Athena Iceberg DDL 문에 대해 지원되지만 모든 타임스탬프 형식은 Athena를 통해 쿼리할 수 있습니다.
`timestamp`	`timestamp`	
`timestamp tz`	`timestamp tz`	
`list<E>`	`array`	
`map<K,V>`	`map`	
`struct<...>`	`struct`	
`fixed(L)`	-	`fixed(L)` 형식은 현재 Athena에서 지원되지 않습니다.

Athena 데이터 형식에 대한 자세한 내용은 Amazon Athena의 데이터 형식 섹션을 참조하세요.

Iceberg 테이블의 다른 Athena 작업

데이터베이스 수준 작업

DROP DATABASE를 CASCADE 옵션과 함께 사용하면 Iceberg 테이블 데이터도 함께 제거됩니다. 다음 DDL 작업은 Iceberg 테이블에는 영향을 주지 않습니다.

- CREATE DATABASE
- ALTER DATABASE SET DBPROPERTIES
- SHOW DATABASES
- SHOW TABLES
- SHOW VIEWS

파티션 관련 작업

Iceberg 테이블이 숨겨진 분할을 사용하기 때문에 직접 물리적 파티션과 작업할 필요는 없습니다. 결과적으로 Athena의 Iceberg 테이블은 다음과 같은 파티션 관련 DDL 작업을 지원하지 않습니다.

- SHOW PARTITIONS
- ALTER TABLE ADD PARTITION
- ALTER TABLE DROP PARTITION
- ALTER TABLE RENAME PARTITION

Athena에서 Iceberg 파티션 변화를 보고 싶다면 athena-feedback@amazon.com으로 피드백을 보내주세요.

Iceberg 테이블 언로드

Iceberg 테이블은 Amazon S3 폴더에 있는 파일로 언로드될 수 있습니다. 자세한 설명은 UNLOAD을 참조하세요.

MSCK REPAIR

Iceberg 테이블은 테이블 레이아웃 정보를 추적하므로 Hive 테이블에서와 마찬가지로 MSCK REPAIR TABLE을 실행하는 것은 필요하지 않으며 또한 지원되지 않습니다.

추가적인 리소스

Apache Iceberg 테이블에서 Athena를 사용하는 방법에 대한 자세한 내용은 AWS 빅 데이터 블로그의 다음 게시물을 참조하세요.

- Accelerate data science feature engineering on transactional data lakes using Amazon Athena with Apache Iceberg
- Build an Apache Iceberg data lake using Amazon Athena, Amazon EMR, and AWS Glue
- Perform upserts in a data lake using Amazon Athena and Apache Iceberg
- Build a transactional data lake using Apache Iceberg, AWS Glue, and cross-account data shares using AWS Lake Formation and Amazon Athena
- Use Apache Iceberg in a data lake to support incremental data processing
- Build a real-time GDPR-aligned Apache Iceberg data lake

- Automate replication of relational sources into a transactional data lake with Apache Iceberg and AWS Glue
- Interact with Apache Iceberg tables using Amazon Athena and cross account fine-grained permissions using AWS Lake Formation
- Build a serverless transactional data lake with Apache Iceberg, Amazon EMR Serverless, and Amazon Athena

Amazon Athena 보안

AWS는 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객으로서 여러분은 가장 높은 보안 요구 사항을 충족하기 위해 설계된 데이터 센터 및 네트워크 아키텍처의 혜택을 받게 됩니다.

보안은 AWS과 사용자의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드의 보안 - AWS는 AWS 서비스 클라우드에서 AWS를 실행하는 인프라를 보호합니다. AWS는 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사자는 정기적으로 AWS 규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 검증합니다. Athena에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 제공 범위 내의 AWS 서비스를 참조하세요.
- 클라우드 내 보안 – 귀하의 책임은 귀하가 사용하는 AWS 서비스로 결정됩니다. 또한 데이터의 민감도, 조직의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 Amazon Athena 사용 시 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 항목에서는 보안 및 규정 준수 목표를 충족하도록 Athena를 구성하는 방법을 보여줍니다. 또한 Athena 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법에 대해 알아봅니다.

주제

- Athena의 데이터 보호
- Athena의 자격 증명 및 액세스 관리
- Athena의 로깅 및 모니터링
- Amazon Athena에 대한 규정 준수 확인
- Athena의 복원성
- Athena의 인프라 보안

- Athena의 구성 및 취약성 분석
- Athena를 사용하여 AWS Lake Formation에 등록된 데이터 쿼리

Athena의 데이터 보호

AWS 공동 책임 모델은 Amazon Athena의 데이터 보호에 적용됩니다. 이 모델에서 설명하는 것처럼 AWS는 모든 AWS 클라우드를 실행하는 글로벌 인프라를 보호할 책임이 있습니다. 사용자는 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스의 보안 구성과 관리 작업에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그에서 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터를 보호하려면 AWS 계정보안 인증 정보를 보호하고 AWS IAM Identity Center또는 AWS Identity and Access Management(IAM)를 통해 개별 사용자 계정을 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 멀티 팩터 인증 설정(MFA)을 사용하세요.
- SSL/TLS를 사용하여 AWS 리소스와 통신하세요. TLS 1.2는 필수이며 TLS 1.3를 권장합니다.
- AWS CloudTrail로 API 및 사용자 활동 로깅을 설정하세요.
- AWS 암호화 솔루션을 AWS 서비스 내의 모든 기본 보안 컨트롤과 함께 사용하세요.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 명령줄 인터페이스 또는 API를 통해 AWS에 액세스할 때 FIPS 140-2 검증된 암호화 모듈이 필요한 경우, FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 FIPS(Federal Information Processing Standard) 140-2를 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 양식 필드에 입력하지 않는 것이 좋습니다. 여기에는 Athena 또는 기타 AWS 서비스에서 콘솔, API, AWS CLI 또는 AWS SDK를 사용하여 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 보안 인증 정보를 URL에 포함시켜서는 안 됩니다.

추가 보안 단계로 aws:CalledVia 전역 조건 컨텍스트 키를 사용하여 Athena에서 보낸 요청으로만 요청을 제한할 수 있습니다. 자세한 내용은 Athena와 CalledVia 컨텍스트 키 사용 단원을 참조하십시오.

다양한 유형의 데이터 보호

Athena를 사용하여 데이터베이스 및 테이블을 작성할 때 여러 유형의 데이터가 관련됩니다. 이러한 데이터 형식에는 Amazon S3에 저장된 원본 데이터, 쿼리를 실행할 때 만드는 데이터베이스 및 테이블의 메타 데이터 또는 데이터, 쿼리 결과 데이터 및 쿼리 기록 검색을 위한 AWS Glue 크롤러가 포함됩니다. 이 섹션에서는 각 유형의 데이터에 대해 설명하고 데이터 보호에 대한 지침을 제공합니다.

- 소스 데이터 – Amazon S3 및 Athena의 데이터베이스와 테이블의 데이터를 저장하고 수정하지 않습니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 Amazon S3의 데이터 보호를 참조하세요. Amazon S3에서 소스 데이터에 대한 액세스를 제어하고 이를 암호화할 수 있습니다. Athena는 Amazon S3에서 암호화된 데이터 세트를 기반으로 테이블을 생성할 수 있습니다.
- 데이터베이스 및 테이블 메타데이터(스키마) – Athena는 스키마-온-리드(schema-on-read) 기술을 사용합니다. 즉, 테이블 정의는 Athena 쿼리 실행 시 Amazon S3의 데이터에 적용됩니다. 사용자가 정의하는 모든 스키마는 명시적으로 삭제하지 않는 한, 자동으로 저장됩니다. Athena에서 DDL 문을 사용하여 데이터 카탈로그 메타데이터를 수정할 수 있습니다. Amazon S3에 저장된 기본 데이터에 영향을 주지 않고도 테이블 정의 및 스키마를 삭제할 수도 있습니다. Athena에서 사용하는 데이터베이스 및 테이블의 메타데이터는 AWS Glue Data Catalog에 저장됩니다.

  AWS Identity and Access Management(IAM)를 사용하면 AWS Glue Data Catalog에 등록된 데이터베이스 및 테이블에 대해 세분화된 액세스 정책을 정의할 수 있습니다. AWS Glue Data Catalog의 메타데이터도 암호화할 수 있습니다. 메타데이터를 암호화하는 경우 액세스를 위해 암호화된 메타데이터에 대한 사용 권한을 사용하세요.
- 저장된 쿼리를 포함한 쿼리 결과 및 쿼리 기록 – 쿼리 결과는 전역 또는 각 작업 그룹에 대해 지정하도록 선택할 수 있는 Amazon S3의 위치에 저장됩니다. 지정하지 않으면 Athena는 각 경우에 기본 위치를 사용합니다. 쿼리 결과 및 저장된 쿼리를 저장하는 Amazon S3 버킷에 대한 액세스를 제어합니다. 또한 Amazon S3에 저장한 쿼리 결과를 암호화하도록 선택할 수 있습니다. 사용자는 Amazon S3 위치에 액세스하고 파일을 해독할 수 있는 적절한 권한을 가지고 있어야 합니다. 자세한 내용은 이 문서의 Amazon S3에 저장된 Athena 쿼리 결과 암호화 단원을 참조하세요.

  Athena는 45일 동안 쿼리 기록을 보관합니다. 콘솔에서 Athena API와 AWS CLI를 사용하여 쿼리 기록을 볼 수 있습니다. 45일 이상 된 쿼리를 저장하려면 저장하세요. 저장된 쿼리에 대한 액세스를 보호하려면 Athena에서 작업 그룹을 사용하여 저장된 쿼리에 대한 액세스 권한을, 쿼리를 볼 수 있는 권한이 있는 사용자에게만 제한합니다.

주제

- 키 관리
- 인터네트워크 트래픽 개인 정보

저장 중 암호화

동일한 리전 및 제한된 수의 리전에 있는 Amazon S3의 암호화된 데이터에 대해 Amazon Athena에서 쿼리를 실행할 수 있습니다. Amazon S3의 쿼리 결과와 AWS Glue 데이터 카탈로그의 데이터를 암호화할 수도 있습니다.

다음과 같은 Athena의 자산을 암호화할 수 있습니다.

- Athena가 Amazon S3 결과 위치로 알려진 위치에 저장하는 Amazon S3의 모든 쿼리 결과 기본 데이터 세트가 Amazon S3에서 암호화되든 그렇지 않든 Amazon S3에 저장된 쿼리 결과를 암호화할 수 있습니다. 자세한 설명은 Amazon S3에 저장된 Athena 쿼리 결과 암호화을 참조하세요.
- AWS Glue 데이터 카탈로그의 데이터. 자세한 설명은 AWS Glue 데이터 카탈로그에 저장된 암호화된 메타데이터 권한을 참조하세요.

> **Note**
>
> Athena를 사용하여 암호화된 테이블을 읽을 때 Athena는 쿼리 결과에 대한 암호화 옵션이 아니라 테이블 데이터에 지정된 암호화 옵션을 사용합니다. 쿼리 결과 및 테이블 데이터에 대해 별도의 암호화 방법 또는 키가 구성된 경우 Athena는 쿼리 결과를 암호화하거나 해독하는 데 사용되는 암호화 옵션 및 키를 사용하지 않고 테이블 데이터를 읽습니다.
> 하지만 Athena를 사용하여 암호화된 데이터가 있는 테이블에 데이터를 삽입하는 경우 Athena는 쿼리 결과에 지정된 암호화 구성을 사용하여 삽입된 데이터를 암호화합니다. 예를 들어 쿼리 결과에 `CSE_KMS` 암호화를 지정하는 경우 Athena는 쿼리 결과 암호화에 사용한 것과 동일한 AWS KMS 키 ID를 사용하여 `CSE_KMS`와 함께 삽입된 테이블 데이터를 암호화합니다.

주제

- 지원되는 Amazon S3 암호화 옵션
- Amazon S3의 암호화 데이터 권한
- AWS Glue 데이터 카탈로그에 저장된 암호화된 메타데이터 권한
- Amazon S3에 저장된 Athena 쿼리 결과 암호화
- Amazon S3의 암호화된 데이터 세트에 기반한 테이블 생성

지원되는 Amazon S3 암호화 옵션

Athena에서는 Amazon S3의 데이터 세트 및 쿼리 결과에 대해 다음과 같은 암호화 옵션을 지원합니다.

암호화 유형	설명	리전 간 지원
SSE-S3	Amazon S3 관리형 키를 사용한 서버 측 암호화(SSE).	예
SSE-KMS	AWS Key Management Service 고객 관리형 키를 사용한 서버 측 암호화(SSE). **Note** 이 암호화 유형을 사용하면 테이블을 생성할 때 Athena에서 데이터 암호화 표시를 요구하지 않습니다.	예
CSE-KMS	AWS KMS 고객 관리형 키를 사용한 클라이언트 측 암호화(CSE) Athena에서 이 옵션을 적용하려면 `CREATE TABLE` 문과, `'has_encrypted_data'='true'` 를 지정하는 `TBLPROPERTIES` 절을 사용해야 합니다. 자세한 내용은 Amazon S3의 암호화된 데이터 세트에 기반한 테이블 생성 단원을 참조하세요.	아니요

Amazon S3를 사용한 AWS KMS 암호화에 관한 자세한 내용은 AWS Key Management Service 개발자 안내서에서 AWS Key Management Service란 무엇인가? 및 Amazon Simple Storage Service(Amazon S3)의 AWS KMS 사용 방법을 참조하세요. Athena에 SSE-KMS 또는 CSE-KMS를 사용하는 방법에 대한 자세한 내용은 AWS Big Data Blog의 Launch: Amazon Athena adds support for querying encrypted data를 참조하세요.

지원되지 않는 옵션

다음 암호화 옵션은 지원되지 않습니다.

- 고객 제공 키가 있는 SSE(SSE-C).
- 클라이언트 측 관리형 키를 사용한 클라이언트 측 암호화.
- 비대칭 키.

Amazon S3 암호화 옵션을 비교하려면 Amazon Simple Storage Service 사용 설명서의 암호화를 사용한 데이터 보호를 참조하세요.

클라이언트 측 암호화를 위한 도구

클라이언트 측 암호화를 위해 다음 두 가지 도구를 사용할 수 있습니다.

- Amazon S3 암호화 클라이언트 - Amazon S3의 데이터만 암호화하며 Athena에서 지원됩니다.
- AWS Encryption SDK - SDK를 사용하여 모든 AWS에서 데이터를 암호화할 수 있으나 Athena에서 직접 지원하지는 않습니다.

이러한 도구는 호환되지 않으며 한 도구를 사용하여 암호화된 데이터는 다른 도구에서 복호화할 수 없습니다. Athena는 Amazon S3 암호화 클라이언트만 직접적으로 지원합니다. SDK를 사용하여 데이터를 암호화하면 Athena에서 쿼리를 실행할 수 있지만 데이터가 암호화된 텍스트로 반환됩니다.

Athena를 사용하여 AWS 암호화 SDK로 암호화된 데이터를 쿼리하려면 데이터를 다운로드하고 복호화한 다음 Amazon S3 암호화 클라이언트를 사용하여 데이터를 다시 암호화해야 합니다.

Amazon S3의 암호화 데이터 권한

Amazon S3에서 사용한 암호화 유형에 따라, Athena에서 사용한 정책에 "허용" 작업이라고 하는 권한을 추가해야 할 수도 있습니다.

- SSE-S3 – 암호화에 SSE-S3를 사용하면 Athena 사용자는 정책에 추가 권한이 필요하지 않습니다. 적절한 Amazon S3 위치(및 Athena 작업)에 대해 적절한 Amazon S3 권한을 갖는 것으로 충분합니다. 적절한 Athena 및 Amazon S3 권한을 허용하는 정책에 대한 자세한 내용은 Amazon Athena의 AWS 관리형 정책 및 Amazon S3에 액세스 단원을 참조하세요.
- AWS KMS – 암호화에 AWS KMS를 사용한다면, Athena 및 Amazon S3 권한 외에 특정 AWS KMS 작업을 수행할 수 있는 권한을 Athena 사용자에게 허용해야 합니다. Amazon S3에서 데이터를 암호화하는 데 사용되는 AWS KMS 고객 관리형 CMK의 주요 정책을 편집하여 이러한 작업을 허용합니다. 키 사용자를 적절한 AWS KMS 키 정책에 추가하기 위해 AWS KMS 콘솔을 사용할 수 있습니다(https://console.aws.amazon.com/kms). AWS KMS 키 정책에 사용자를 추가하는 방법에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 키 사용자가 CMK를 사용하도록 허용을 참조하세요.

> **Note**
>
> 고급 키 정책 관리자는 키 정책을 세부적으로 조정할 수 있습니다. `kms:Decrypt`는 Athena 사용자가 암호화된 데이터 세트로 작업할 때 허용되는 최소 작업입니다. 암호화된 쿼리 결과로 작업할 때 허용되는 최소 작업은 `kms:GenerateDataKey` 및 `kms:Decrypt`입니다.

AWS KMS로 암호화된 객체가 다수 포함되어 있는 Amazon S3에서 Athena를 사용해 데이터 세트를 쿼리한다면 AWS KMS가 쿼리 결과를 제한할 수 있습니다. 다수의 작은 객체가 있는 경우 이런 현상이 발생할 가능성이 높습니다. Athena가 재시도 요청을 취소해도 제한 오류가 계속 발생할 수 있습니다. 많은 수의 암호화된 객체로 작업하고 이 문제가 발생하는 경우 Amazon S3 버킷 키를 활성화하여 KMS에 대한 호출 수를 줄이는 것이 한 가지 방법입니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 Amazon S3 버킷 키를 사용하여 SSE-KMS 비용 절감을 참조하세요. 또 다른 옵션은 AWS KMS에 대한 서비스 할당량을 늘리는 것입니다. 자세한 내용은 AWS Key Management Service 개발자 안내서에서 할당량을 참조하세요.

Amazon S3와 Athena 사용 시 권한에 대한 문제 해결 정보는 권한 주제의 Athena의 문제 해결 단원을 참조하세요.

AWS Glue 데이터 카탈로그에 저장된 암호화된 메타데이터 권한

AWS Glue Data Catalog의 메타데이터를 암호화한다면, Athena 액세스에 사용한 정책에 `"kms:GenerateDataKey"`, `"kms:Decrypt"`, `"kms:Encrypt"` 작업을 추가해야 합니다. 자세한 설명은 Athena에서 AWS Glue Data Catalog의 암호화된 메타데이터에 액세스을 참조하세요.

Amazon S3에 저장된 Athena 쿼리 결과 암호화

Athena 콘솔을 사용하거나 JDBC 또는 ODBC를 사용할 경우 쿼리 결과 암호화를 설정합니다. 작업 그룹을 통해 쿼리 결과를 암호화할 수 있습니다.

콘솔에서 쿼리 결과의 암호화 구성에 대한 설정은 다음 두 가지 방법으로 구성할 수 있습니다.

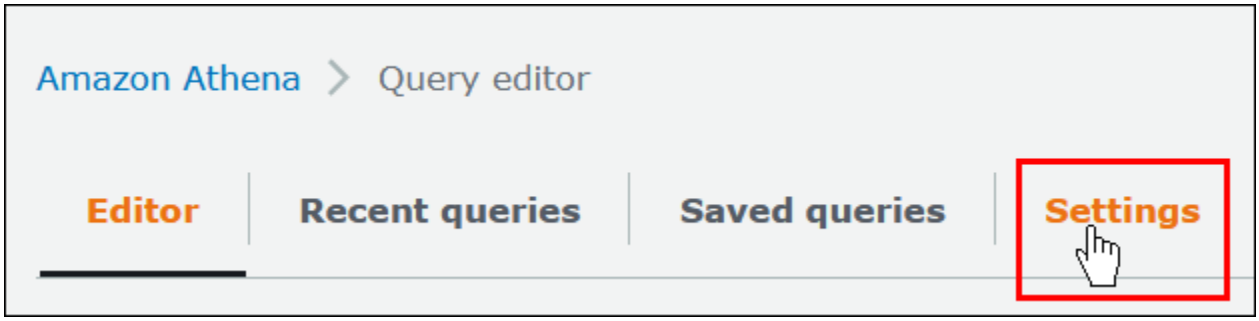
- 클라이언트 측 설정 - 콘솔에서 설정을 사용하거나 API 작업을 사용하여 쿼리 결과를 암호화하도록 지정할 경우 클라이언트 측 설정을 사용하는 것입니다. 클라이언트 측 설정에는 쿼리 결과 위치와 암호화가 포함됩니다. 이를 지정할 경우 작업 그룹 설정에서 재정의하지 않는 한 이 설정이 사용됩니다.
- 작업 그룹 설정 – 작업 그룹을 생성하거나 편집하고 클라이언트 측 설정 재정의(Override client-side settings) 필드를 선택할 경우 이 작업 그룹에서 실행되는 모든 쿼리가 작업 그룹 암호화 및 쿼리 결과

위치 설정을 사용합니다. 자세한 내용은 클라이언트 측 설정보다 우선하는 작업 그룹 설정 단원을 참조하십시오.

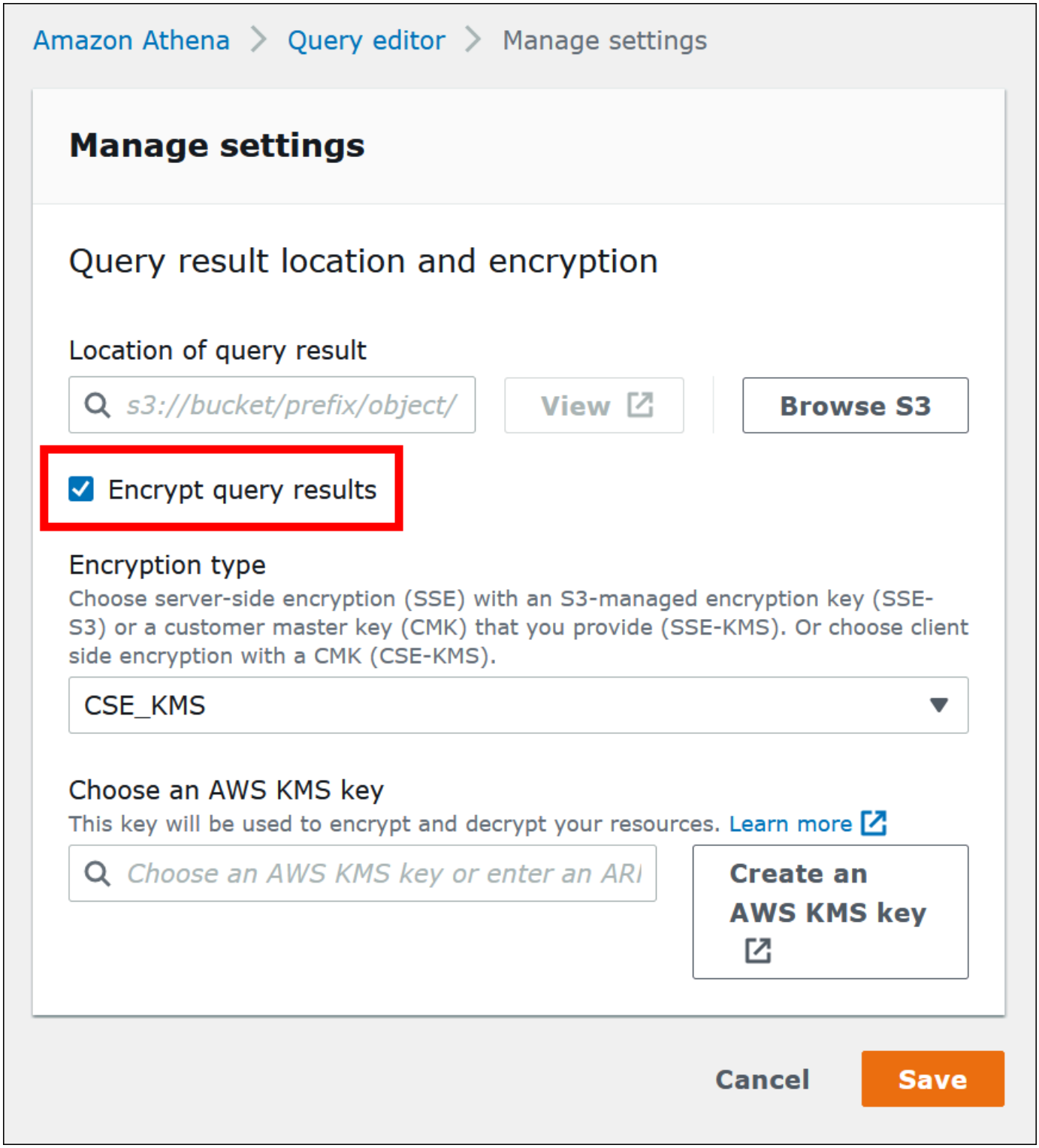
콘솔을 사용해 Amazon S3에 저장된 쿼리 결과를 암호화하려면

> **Important**
>
> 작업 그룹에 클라이언트 측 설정 재정의(Override client-side settings) 필드가 선택되어 있으면 작업 그룹의 모든 쿼리가 작업 그룹 설정을 사용합니다. Athena 콘솔의 설정(Settings) 탭에 지정된 암호화 구성 및 쿼리 결과 위치, API 작업, JDBC 및 ODBC 드라이버는 사용되지 않습니다. 자세한 내용은 클라이언트 측 설정보다 우선하는 작업 그룹 설정 단원을 참조하십시오.

1. Athena 콘솔에서 설정을 선택합니다.



2. 관리를 선택합니다.
3. 쿼리 결과 위치(Location of query result)에 Amazon S3 경로를 입력하거나 선택합니다. 이는 쿼리 결과가 저장되는 Amazon S3 위치입니다.
4. [Encrypt query results]를 선택합니다.



5. [Encryption type]에서 [CSE-KMS], [SSE-KMS] 또는 [SSE-S3]를 선택합니다. 이 세 가지 중에서 CSE-KMS는 가장 높은 수준의 암호화를 제공하고 SSE-S3는 가장 낮은 수준의 암호화를 제공합니다.
6. SSE-KMS 또는 CSE-KMS를 선택한 경우 AWS KMS 키를 지정합니다.

- 계정에 기존 AWS KMS 고객 관리형 키(CMK)에 대한 액세스 권한이 있는 경우 AWS KMS 키 선택(Choose an AWS KMS)에서 별칭을 선택하거나 AWS KMS 키 ARN을 입력합니다.
- 계정에서 기존 고객 관리형 키(CMK)에 액세스할 수 없는 경우 AWS KMS 키 생성(Create an AWS KMS key)을 선택한 다음 AWS KMS 콘솔을 엽니다. 자세한 정보는 AWS Key Management Service 개발자 안내서의 키 생성을 참조하세요.

> **Note**
>
> Athena는 데이터 읽기 및 쓰기를 위한 대칭 키만 지원합니다.

7. Athena 콘솔로 돌아가서 별칭 또는 ARN으로 생성한 키를 선택합니다.
8. Save(저장)를 선택합니다.

JDBC 또는 ODBC를 사용할 때 Athena 쿼리 결과 암호화

JDBC 또는 ODBC 드라이버를 사용하여 연결하는 경우 사용할 암호화 유형과 Amazon S3 스테이징 디렉터리 위치를 지정하는 드라이버 옵션을 구성합니다. Athena가 지원하는 암호화 프로토콜을 사용하여 쿼리 결과를 암호화하도록 JDBC 또는 ODBC 드라이버를 구성하는 방법은 ODBC 및 JDBC 드라이버로 Amazon Athena에 연결 섹션을 참조하세요.

Amazon S3의 암호화된 데이터 세트에 기반한 테이블 생성

테이블을 만들 때 Amazon S3의 데이터 세트가 암호화되어 있음을 Athena에 표시합니다. SSE-KMS 사용 시에는 필요하지 않습니다. SSE-S3 및 AWS KMS 암호화 모두 Athena가 데이터 세트를 복호화하고 테이블을 생성하는 방식을 결정할 수 있으므로 키 정보를 제공할 필요가 없습니다.

이번 주제 앞 부분에서 설명했듯이, 테이블을 만드는 사용자를 포함해 쿼리를 실행하는 사용자는 앞서 설명한 권한을 가지고 있어야 합니다.

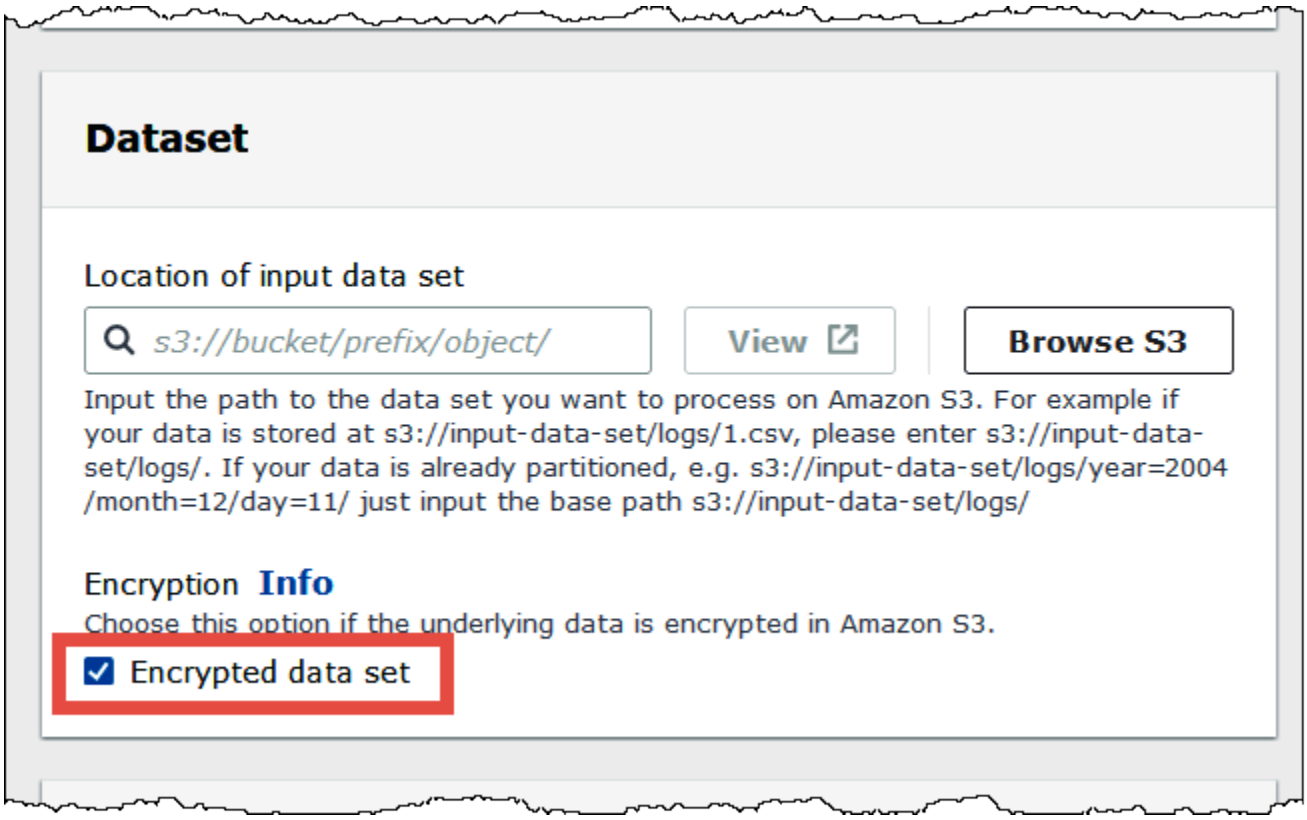
> **Important**
>
> Amazon EMR을 EMRFS와 함께 사용해 암호화된 Parquet 파일을 업로드한다면, `fs.s3n.multipart.uploads.enabled`를 `false`로 설정해 멀티파트 업로드를 비활성화해야 합니다. 이 작업을 하지 않으면, Athena는 Parquet 파일 길이를 확인하지 못하며 HIVE_CANNOT_OPEN_SPLIT 오류가 발생합니다. 자세한 내용은 Amazon EMR 관리 안내서의 Amazon S3에 대한 멀티파트 업로드 구성을 참조하세요.

Amazon S3에서 데이터 세트가 암호화되는지 표시하려면 다음 단계 중 하나를 수행합니다. SSE-KMS를 사용한다면 이 단계는 필요하지 않습니다.

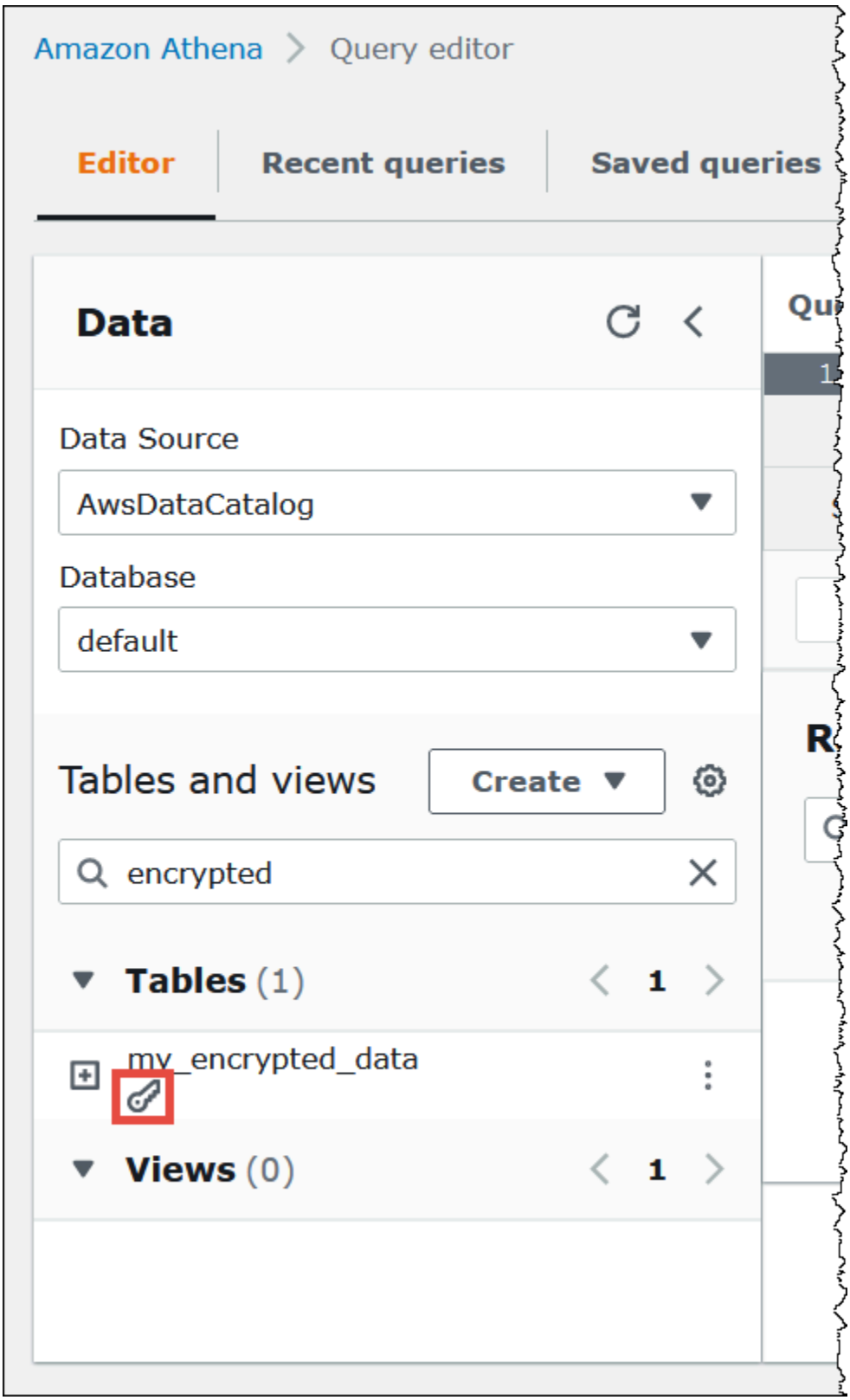
- 다음 예제처럼 CREATE TABLE 문에서 `'has_encrypted_data'='true'`를 지정하는 `TBLPROPERTIES` 절을 사용합니다.

```
CREATE EXTERNAL TABLE 'my_encrypted_data' (
   `n_nationkey` int,
   `n_name` string,
   `n_regionkey` int,
   `n_comment` string)
ROW FORMAT SERDE
   'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
   'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
LOCATION
   's3://DOC-EXAMPLE-BUCKET/folder_with_my_encrypted_data/'
TBLPROPERTIES (
   'has_encrypted_data'='true')
```

- `statement.executeQuery()`를 사용해 CREATE TABLE 문을 실행하는 경우 JDBC 드라이버를 사용해 이전 예제처럼 `TBLPROPERTIES` 값을 설정합니다.
- Athena 콘솔을 사용하여 양식을 사용하여 테이블을 생성하고 테이블 위치를 지정할 때 암호화된 데이터 세트(Encrypted data set) 옵션을 선택합니다.



Athena 콘솔 테이블 목록에서 암호화된 테이블은 키 모양 아이콘을 표시합니다.


Amazon Athena
Query editor
Editor
Recent queries
Saved queries
Data
Data Source
AwsDataCatalog
Database
default
Tables and views
Create
encrypted
Tables (1)
1
my_encrypted_data
Views (0)
1

전송 중 암호화

Amazon S3의 저장된 데이터를 암호화하는 것 외에도 Amazon Athena는 Athena와 Amazon S3 사이, 그리고 Athena와 이에 액세스하는 고객 에플리케이션 사이의 전송 중 데이터에 TLS(Transport Layer Security) 암호화를 사용합니다.

Amazon S3 버킷 IAM 정책에 `aws:SecureTransport condition`을 사용하여 HTTPS(TLS)를 통해 암호화된 연결만 허용해야 합니다.

JDBC 또는 ODBC 클라이언트로 스트리밍하는 쿼리 결과는 TLS를 사용하여 암호화됩니다. JDBC 및 ODBC 드라이버의 최신 버전 및 해당 설명서에 대한 자세한 내용은 JDBC로 Amazon Athena에 연결 및 ODBC로 Amazon Athena에 연결 단원을 참조하세요.

Athena 페더레이션된 데이터 소스 커넥터의 경우 TLS를 사용한 전송 중 암호화 지원은 개별 커넥터에 따라 다릅니다. 자세한 내용은 개별 데이터 소스 커넥터에 대한 설명서를 참조하세요.

키 관리

Amazon Athena는 AWS Key Management Service(AWS KMS)가 Amazon S3 및 Athena 쿼리 결과의 데이터 세트을 암호화하는 것을 지원합니다. AWS KMS는 고객 마스터 키(CMK)를 사용하여 Amazon S3 객체를 암호화하고 봉투 암호화(envelope encryption)를 사용합니다.

AWS KMS에서 다음 작업을 수행할 수 있습니다.

- 키 생성
- 새 CMK에 대한 자체적인 키 구성 요소를 가져오기

> **Note**
>
> Athena는 데이터 읽기 및 쓰기를 위한 대칭 키만 지원합니다.

자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS Key Management Service란 무엇인가?와, Amazon Simple Storage Service가 AWS KMS를 사용하는 방식을 참조하세요. AWS에서 계정을 위해 직접 생성하고 관리하는 키를 보려면 탐색 창에서 AWS 고객 관리형 키를 선택합니다.

SSE-KMS로 암호화된 객체를 업로드 또는 액세스하는 경우 강화된 보안을 위해 AWS 서명 버전 4를 사용합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 요청 인증에서 서명 버전 지정을 참조하세요.

Athena 워크로드가 대량의 데이터를 암호화하는 경우 Amazon S3 버킷 키를 사용하여 비용을 절감할 수 있습니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 Amazon S3 버킷 키를 사용하여 SSE-KMS 비용 절감을 참조하세요.

인터네트워크 트래픽 개인 정보

Athena 및 온프레미스 애플리케이션 간 및 Athena, Amazon S3 간 트래픽이 보호됩니다. Athena와 AWS Glue 및 AWS Key Management Service 등의 다른 서비스 간의 트래픽은 기본값으로 HTTP를 사용합니다.

- Athena 및 온프레미스 클라이언트와 애플리케이션 간의 트래픽의 경우 JDBC 또는 ODBC 클라이언트로 스트리밍되는 쿼리 결과는 TLS(전송 계층 보안)를 사용하여 암호화됩니다.

  프라이빗 네트워크와 AWS 사이에 연결 옵션 중 하나를 사용할 수 있습니다.

  - Site-to-Site VPN AWS VPN 연결. 자세한 내용은 AWS Site-to-Site VPN 사용 설명서에서 Site-to-Site VPN AWS VPN이란 무엇인가?를 참조하세요.
  - AWS Direct Connect 연결. 자세한 내용은 AWS Direct Connect 사용 설명서에서 AWS Direct Connect란 무엇입니까? 단원을 참조하세요.
- Athena와 Amazon S3 버킷 간의 트래픽의 경우 전송 계층 보안(TLS)이 Athena와 Amazon S3 사이, 그리고 Athena와 이에 액세스하는 고객 애플리케이션 사이의 전송 중 객체를 암호화합니다. Amazon S3 버킷 IAM 정책에 `aws:SecureTransport condition`을 사용할 때는 HTTPS(TLS)를 통해 암호화된 연결만 허용해야 합니다. Athena는 현재 퍼블릭 엔드포인트를 사용하여 Amazon S3 버킷의 데이터에 액세스하지만 이것이 데이터가 퍼블릭 인터넷을 통과한다는 의미는 아닙니다. Athena와 Amazon S3 간의 모든 트래픽은 AWS 네트워크를 통해 라우팅되고 TLS를 사용하여 암호화됩니다.
- 규정 준수 프로그램 - Amazon Athena는 SOC, PCI, FedRAMP 등의 여러 AWS 규정 준수 프로그램을 준수합니다. 자세한 내용은 규정 준수 프로그램 제공 범위 내 AWS 서비스를 참조하세요.

Athena의 자격 증명 및 액세스 관리

Amazon Athena는 AWS Identity and Access Management(IAM) 정책을 사용해 Athena 작업에 대한 액세스를 제한합니다. Athena의 전체 권한 목록은 서비스 권한 부여 참조에서 Amazon Athena에 사용되는 작업, 리소스 및 조건 키를 참조하세요.

IAM 정책을 사용할 때는 항상 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

Athena 쿼리를 실행하는 데 필요한 권한은 다음과 같습니다.

- 쿼리할 기본 데이터가 저장되는 Amazon S3 위치입니다. 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 Amazon S3의 자격 증명 및 액세스 관리를 참조하세요.
- 암호화된 메타데이터에 대한 추가 작업을 포함하여 데이터베이스 및 테이블과 같이 사용자가 AWS Glue Data Catalog에 저장하는 메타데이터 및 리소스. 자세한 내용은 AWS Glue 개발자 안내서의 AWS Glue에 대한 IAM 권한 설정 및 AWS Glue의 암호화 설정을 참조하세요.
- Athena API 작업. Athena의 API 작업에 관한 전체 목록은 Amazon Athena API 참조의 작업을 참조하세요.

다음 주제는 Athena의 특정 영역에 대한 권한에 관하여 추가 정보를 제공합니다.

주제

- Amazon Athena의 AWS 관리형 정책
- JDBC 및 ODBC 연결을 통한 액세스
- Amazon S3에 액세스
- Amazon S3 버킷에 대한 Athena의 계정 간 액세스
- AWS Glue Data Catalog의 데이터베이스와 테이블에 대한 세분화된 액세스
- AWS Glue 데이터 카탈로그에 대한 교차 계정 액세스
- Athena에서 AWS Glue Data Catalog의 암호화된 메타데이터에 액세스
- 작업 그룹 및 태그에 대한 액세스
- 준비된 문에 대한 액세스 허용
- Athena와 CalledVia 컨텍스트 키 사용
- 외부 Hive 메타스토어용 Athena 데이터 커넥터에 대한 액세스 허용
- 외부 Hive 메타스토어에 대한 Lambda 함수 액세스 허용
- Athena 연합 쿼리를 허용하는 IAM 권한 정책의 예제
- Amazon Athena 사용자 정의 함수(UDF)를 허용하는 IAM 권한 정책의 예
- ML과 Athena에 대한 액세스 허용
- Athena API에 대한 연합 액세스 활성화

Amazon Athena의 AWS 관리형 정책

AWS 관리형 정책은 AWS에서 생성되고 관리되는 독립 실행형 정책입니다. AWS 관리형 정책은 사용자, 그룹 및 역할에 권한 할당을 시작할 수 있도록 많은 일반 사용 사례에 대한 권한을 제공하도록 설계되었습니다.

AWS 관리형 정책은 모든 AWS 고객이 사용할 수 있기 때문에 특정 사용 사례에 대해 최소 권한을 부여하지 않을 수 있습니다. 사용 사례에 고유한 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다.

AWS 관리형 정책에서 정의한 권한은 변경할 수 없습니다. 만약 AWS가 AWS 관리형 정책에 정의된 권한을 업데이트할 경우 정책이 연결되어 있는 모든 보안 주체 엔터티(사용자, 그룹 및 역할)에도 업데이트가 적용됩니다. 새로운 AWS 서비스를 시작하거나 새로운 API 작업을 기존 서비스에 이용하는 경우 AWS가 AWS 관리형 정책을 업데이트할 가능성이 높습니다.

자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

Athena에 관리형 정책을 사용할 때의 고려 사항

관리형 정책은 사용하기 쉽고, 서비스가 향상됨에 따라 필요한 작업으로 자동 업데이트됩니다. Athena에 관리형 정책을 사용할 때 다음 사항에 유의하세요.

- AWS Identity and Access Management(IAM)를 사용하여 자신 또는 다른 사용자에 대한 Amazon Athena 서비스 작업을 허용하거나 거부하려면 자격 증명 기반 정책을 사용자 또는 그룹 등의 보안 주체에 연결합니다.
- 각 자격 증명 기반 정책은 허용되거나 거부되는 작업을 정의하는 문으로 구성됩니다. 정책을 사용자에게 연결하는 방법에 대한 자세한 내용과 단계별 지침은 IAM 사용 설명서의 관리형 정책 연결을 참조하세요. 작업 목록은 Amazon Athena API 참조를 참조하세요.
- 고객 관리형 및 인라인 자격 증명 기반 정책을 사용하면 정책 내에서 보다 상세한 Athena 작업을 지정하여 액세스를 세부 조정할 수 있습니다. `AmazonAthenaFullAccess` 정책을 시작점으로 사용하고 Amazon Athena API 참조에 나열된 특정 작업을 허용하거나 거부하는 것이 좋습니다. 인라인 정책에 대한 자세한 내용은 IAM 사용 설명서에서 관리형 정책과 인라인 정책 단원을 참조하세요.
- JDBC를 사용하여 연결하는 보안 주체도 있다면, 애플리케이션에 JDBC 드라이버 자격 증명을 입력해야 합니다. 자세한 내용은 JDBC 및 ODBC 연결을 통한 액세스 단원을 참조하십시오.
- AWS Glue 데이터 카탈로그를 암호화한 경우 Athena에 대한 자격 증명 기반 IAM 정책에 추가 작업을 지정해야 합니다. 자세한 내용은 Athena에서 AWS Glue Data Catalog의 암호화된 메타데이터에 액세스 단원을 참조하십시오.
- 작업 그룹을 만들고 사용할 경우 해당 정책에 작업 그룹 작업에 대한 관련 액세스 권한이 있는지 확인하세요. 자세한 정보는 the section called “ 작업 그룹 액세스를 위한 IAM 정책” 및 the section called “작업 그룹 정책의 예” 단원을 참조하세요.

AWS 관리형 정책: AmazonAthenaFullAccess

`AmazonAthenaFullAccess` 관리형 정책은 Athena에 대한 완전한 액세스 권한을 부여합니다.

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가합니다.

- AWS IAM Identity Center의 사용자 및 그룹:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따르세요.
- ID 제공자를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 서드 파티 자격 증명 공급자의 역할 만들기(연합)의 지침을 따르세요.
- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 IAM 사용자의 역할 생성의 지침을 따르세요.
  - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

권한 그룹화

`AmazonAthenaFullAccess` 정책은 다음 권한 집합으로 그룹화됩니다.

- **`athena`** - 보안 주체에게 Athena 리소스에 대한 액세스를 허용합니다.
- **`glue`** - 보안 주체에게 AWS Glue 데이터베이스, 테이블 및 파티션에 대한 액세스를 허용합니다. 보안 주체가 Athena에 AWS Glue Data Catalog를 사용하려면 이 권한이 필수입니다.
- **`s3`** - 보안 주체가 Amazon S3에서 쿼리 결과를 읽고 쓰거나, Amazon S3에 상주하는 공개적으로 사용 가능한 Athena 데이터 예제를 읽거나, 버킷을 나열할 수 있도록 허용합니다. 보안 주체가 Athena를 사용하여 Amazon S3 작업을 수행하려면 이 권한이 필수입니다.
- **`sns`** - 보안 주체가 Amazon SNS 주제를 나열하고 주제 속성을 가져올 수 있도록 허용합니다. 이를 통해 보안 주체는 모니터링 및 알림 목적으로 Athena에 Amazon SNS 주제를 사용할 수 있습니다.
- **`cloudwatch`** - 보안 주체에게 CloudWatch 경보의 생성, 읽기 및 삭제를 허용합니다. 자세한 내용은 CloudWatch 지표 및 이벤트를 사용하여 비용 관리 및 쿼리 모니터링 단원을 참조하십시오.
- **`lakeformation`** - 보안 주체가 Lake Formation에 등록된 데이터 레이크 위치의 데이터에 액세스하기 위해 임시 자격 증명을 요청하도록 허용합니다. 자세한 내용은 AWS Lake Formation 개발자 가이드의 기본 데이터 액세스 제어를 참조하세요.
- **`datazone`** - 보안 주체가 Amazon DataZone 프로젝트, 도메인 및 환경을 나열할 수 있습니다. Athena에서 DataZone을 사용하는 방법에 대한 자세한 내용은 Athena에서 Amazon DataZone 사용 섹션을 참조하세요.

- **`pricing`** – AWS Billing and Cost Management에 대한 액세스 권한을 제공합니다. 자세한 내용은 AWS Billing and Cost Management용 API 참조의 GetInstance를 참조하세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "BaseAthenaPermissions",
            "Effect": "Allow",
            "Action": [
                "athena:*"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BaseGluePermissions",
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:DeleteDatabase",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:CreateTable",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
                "glue:GetPartitions",
                "glue:BatchGetPartition",
                "glue:StartColumnStatisticsTaskRun",
                "glue:GetColumnStatisticsTaskRun",
                "glue:GetColumnStatisticsTaskRuns"
```

```
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BaseQueryResultsPermissions",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload",
                "s3:CreateBucket",
                "s3:PutObject",
                "s3:PutBucketPublicAccessBlock"
            ],
            "Resource": [
                "arn:aws:s3:::aws-athena-query-results-*"
            ]
        },
        {
            "Sid": "BaseAthenaExamplesPermissions",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::athena-examples*"
            ]
        },
        {
            "Sid": "BaseS3BucketPermissions",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:ListAllMyBuckets"
            ],
            "Resource": [
                "*"
```

```
            ]
        },
        {
            "Sid": "BaseSNSPermissions",
            "Effect": "Allow",
            "Action": [
                "sns:ListTopics",
                "sns:GetTopicAttributes"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BaseCloudWatchPermissions",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DeleteAlarms",
                "cloudwatch:GetMetricData"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BaseLakeFormationPermissions",
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BaseDataZonePermissions",
            "Effect": "Allow",
            "Action": [
                "datazone:ListDomains",
                "datazone:ListProjects",
                "datazone:ListAccountEnvironments"
            ],
```

```
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "BasePricingPermissions",
            "Effect": "Allow",
            "Action": [
                "pricing:GetProducts"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AWS 관리형 정책: AWSQuicksightAthenaAccess

AWSQuicksightAthenaAccess는 Amazon QuickSight와 Athena의 통합에 필요한 작업에 대한 액세스 권한을 부여합니다. AWSQuicksightAthenaAccess 정책을 IAM 보안 인증에 연결할 수 있습니다. 이 정책은 Athena와 함께 Amazon QuickSight를 사용하는 보안 주체에만 연결합니다. 이 정책에는 사용되지 않거나, 현재 퍼블릭 API에 포함되지 않거나, 반드시 JDBC 및 ODBC 드라이버와 함께 사용하는 Athena를 위한 몇 가지 작업이 포함됩니다.

권한 그룹화

AWSQuicksightAthenaAccess 정책은 다음 권한 집합으로 그룹화됩니다.

- **athena** - 보안 주체가 Athena 리소스에 대한 쿼리를 실행할 수 있도록 허용합니다.
- **glue** - 보안 주체에게 AWS Glue 데이터베이스, 테이블 및 파티션에 대한 액세스를 허용합니다. 보안 주체가 Athena에서 AWS Glue Data Catalog를 사용하려면 이 권한이 필수입니다.
- **s3** - 보안 주체가 Amazon S3 쿼리 결과를 쓰고 읽을 수 있도록 허용합니다.
- **lakeformation** - 보안 주체가 Lake Formation에 등록된 데이터 레이크 위치의 데이터에 액세스하기 위해 임시 자격 증명을 요청하도록 허용합니다. 자세한 내용은 AWS Lake Formation 개발자 가이드의 기본 데이터 액세스 제어를 참조하세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "athena:BatchGetQueryExecution",
                "athena:GetQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:ListQueryExecutions",
                "athena:StartQueryExecution",
                "athena:StopQueryExecution",
                "athena:ListWorkGroups",
                "athena:ListEngineVersions",
                "athena:GetWorkGroup",
                "athena:GetDataCatalog",
                "athena:GetDatabase",
                "athena:GetTableMetadata",
                "athena:ListDataCatalogs",
                "athena:ListDatabases",
                "athena:ListTableMetadata"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue:CreateDatabase",
                "glue:DeleteDatabase",
                "glue:GetDatabase",
                "glue:GetDatabases",
                "glue:UpdateDatabase",
                "glue:CreateTable",
                "glue:DeleteTable",
                "glue:BatchDeleteTable",
                "glue:UpdateTable",
                "glue:GetTable",
                "glue:GetTables",
                "glue:BatchCreatePartition",
                "glue:CreatePartition",
                "glue:DeletePartition",
                "glue:BatchDeletePartition",
                "glue:UpdatePartition",
                "glue:GetPartition",
```

```
                "glue:GetPartitions",
                "glue:BatchGetPartition"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListBucketMultipartUploads",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload",
                "s3:CreateBucket",
                "s3:PutObject",
                "s3:PutBucketPublicAccessBlock"
            ],
            "Resource": [
                "arn:aws:s3:::aws-athena-query-results-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lakeformation:GetDataAccess"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AWS 관리형 정책으로 Athena 업데이트

이 서비스가 해당 변경 사항을 추적하기 시작한 이후 Athena용 AWS 관리형 정책 업데이트에 대한 세부 정보를 확인합니다.

변경 사항	설명	날짜
AmazonAthenaFullAccess - 기존 정책 업데이트	Athena 사용자가 Amazon DataZone 도메인, 프로젝트 및 환경에서 작업할 수 있도록 `datazone:ListDomains` , `datazone:ListProjects` 및 `datazone:ListAccountEnvironments` 권한이 추가되었습니다. 자세한 내용은 Athena에서 Amazon DataZone 사용 단원을 참조하십시오.	2024년 1월 3일
AmazonAthenaFullAccess - 기존 정책 업데이트	비용 기반 옵티마이저 기능에 대한 통계를 검색하기 위해 Athena에 AWS Glue를 직접 호출할 권한을 제공하도록 `glue:StartColumnStatisticsTaskRun` , `glue:GetColumnStatisticsTaskRun` 및 `glue:GetColumnStatisticsTaskRuns` 권한이 추가되었습니다. 자세한 내용은 비용 기반 최적화 프로그램 사용 단원을 참조하십시오.	2024년 1월 3일
AmazonAthenaFullAccess - 기존 정책 업데이트	Athena는 AWS Billing and Cost Management에 대한 액세스를 제공하기 위해 `pricing:GetProducts` 를 추가했습니다. 자세한 내용은 AWS Billing and Cost Management용 API 참조의 GetInstance를 참조하세요.	2023년 1월 25일
AmazonAthenaFullAccess - 기존 정책 업데이트	CloudWatch 지표 값을 검색하기 위해 `cloudwatch:GetMetricData` 를 추가했습니다. 자세한	2022년 11월 14일

변경 사항	설명	날짜
	내용은 Amazon CloudWatch API 참조에서 GetMetricData를 참조하세요.	
AmazonAthenaFullAccess 및 AWSQuicksightAthenaAccess - 기존 정책에 대한 업데이트	Athena가 생성한 버킷에 대한 퍼블릭 액세스를 차단할 수 있도록 Athena에 `s3:PutBucketPublicAccessBlock` 이 추가되었습니다.	2021년 7월 7일
Athena의 변경 내용 추적 시작	Athena가 AWS 관리형 정책에 대한 변경 내용 추적을 시작했습니다.	2021년 7월 7일

JDBC 및 ODBC 연결을 통한 액세스

Athena 및 Amazon S3 버킷 같은 AWS 서비스와 리소스에 대한 액세스를 얻으려면 애플리케이션에 JDBC 또는 ODBC 드라이버 자격 증명을 제공합니다. JDBC 드라이버 또는 ODBC 드라이버를 사용한다면, IAM 권한 정책에 AWS 관리형 정책: AWSQuicksightAthenaAccess에 있는 작업이 모두 포함되었는지 확인하세요.

IAM 정책을 사용할 때마다 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

인증 방법

Athena JDBC 및 ODBC 드라이버는 다음 자격 증명 공급자를 비롯한 SAML 2.0 기반 인증을 지원합니다.

- AD FS(Active Directory Federation Services)
- Azure Active Directory(AD)
- Okta
- PingFederate

자세한 내용은 해당 드라이버의 설치 및 구성 가이드를 참조하세요. 해당 가이드는 JDBC 및 ODBC 드라이버 페이지에서 PDF 형식으로 다운로드할 수 있습니다. 추가 관련 정보는 다음을 참조하세요.

- Athena API에 대한 연합 액세스 활성화
- Athena에 대한 연합 액세스를 위해 Lake Formation과 Athena JDBC 및 ODBC 드라이버 사용
- ODBC, SAML 2.0 및 Okta 자격 증명 공급자를 사용하여 통합 인증 구성

JDBC 및 ODBC 드라이버의 최신 버전 및 해당 설명서에 대한 자세한 내용은 JDBC로 Amazon Athena에 연결 및 ODBC로 Amazon Athena에 연결 단원을 참조하세요.

Amazon S3에 액세스

ID 기반 정책, 버킷 리소스 정책, 액세스 포인트 정책 또는 위 정책들의 조합을 사용하여 Amazon S3 위치에 대한 액세스 권한을 부여할 수 있습니다. 액터가 Athena와 상호 작용하면 해당 권한이 Athena를 통해 전달되어 Athena가 액세스할 수 있는 대상이 결정됩니다. 즉, 사용자는 Athena에서 Amazon S3 버킷을 쿼리하려면 Amazon S3 버킷에 액세스할 수 있는 권한이 있어야 합니다.

IAM 정책을 사용할 때마다 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

정책에서 `aws:SourceIp`를 구성하면 Athena는 지정한 IP 주소를 사용하여 Amazon S3 버킷에 액세스합니다. `aws:SourceVpc` 또는 `aws:SourceVpce` 조건 키에 따라 Amazon S3 리소스에 대한 액세스를 제한하거나 허용할 수 없습니다.

> **Note**
>
> IAM Identity Center 인증을 사용하는 Athena 작업 그룹의 경우 신뢰할 수 있는 ID 전파 ID를 사용하도록 S3 Access Grants를 구성해야 합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 S3 Access Grants and directory identities를 참조하십시오.

Amazon S3 액세스 포인트 및 액세스 포인트 별칭

Amazon S3 버킷에 공유 데이터 세트이 있는 경우 수백 개의 사용 사례에 대해 액세스를 관리하는 하나의 버킷 정책을 유지하는 것은 어려울 수 있습니다.

Amazon S3 버킷 액세스 포인트는 이 문제를 해결하는 데 도움이 됩니다. 버킷에는 여러 액세스 포인트가 있을 수 있으며, 각 액세스 포인트는 서로 다른 방식으로 버킷에 대한 액세스 권한을 제어하는 정책이 있습니다.

생성한 각 액세스 포인트에 대해 Amazon S3는 액세스 포인트를 나타내는 별칭을 생성합니다. 별칭은 Amazon S3 버킷 이름 형식이므로 Athena에서 `CREATE TABLE` 문의 `LOCATION` 절에 이 별칭을 사용

할 수 있습니다. 이렇게 하면 버킷에 대한 Athena의 액세스는 별칭이 나타내는 액세스 포인트에 대한 정책에 의해 제어됩니다.

자세한 내용은 Amazon S3 사용 설명서의 Amazon S3에서 테이블 위치 및 액세스 포인트 사용을 참조하세요.

CalledVia 컨텍스트 키 사용

보안 강화를 위해 aws:CalledVia 전역 조건 컨텍스트 키를 사용할 수 있습니다. `aws:CalledVia` 키에는 보안 주체를 대신하여 요청을 수행한 체인의 각 서비스 목록이 정렬되어 있습니다. `aws:CalledVia` 컨텍스트 키에 Athena 서비스 보안 주체 이름 `athena.amazonaws.com`을 지정하여 Athena에서 보낸 요청으로만 요청을 제한할 수 있습니다. 자세한 내용은 Athena와 CalledVia 컨텍스트 키 사용 단원을 참조하십시오.

추가 리소스

Amazon S3 액세스 권한을 부여하는 방법에 대한 자세한 내용과 예는 다음 리소스를 참조하세요.

- 예제 시연: Amazon S3 사용 설명서의 액세스 관리.
- AWS 지식 센터의 Amazon S3 버킷에 있는 객체에 대해 계정 간 액세스 권한을 제공하려면 어떻게 해야 합니까?.
- Amazon S3 버킷에 대한 Athena의 계정 간 액세스.

Amazon S3 버킷에 대한 Athena의 계정 간 액세스

일반적인 Amazon Athena 시나리오는 버킷 소유자와 다른 계정의 사용자가 쿼리를 수행할 수 있도록 액세스 권한을 부여하는 것입니다. 이 경우 버킷 정책을 사용하여 액세스를 부여합니다.

> **Note**
>
> Athena에서 AWS Glue 데이터 카탈로그에 대해 계정 간 액세스를 적용하는 방법에 관한 자세한 내용은 AWS Glue 데이터 카탈로그에 대한 교차 계정 액세스 단원을 참조하세요.

버킷 소유자가 생성하여 버킷 `s3://DOC-EXAMPLE-BUCKET`에 적용한 다음 예제 버킷 정책은 이와 다른 계정인 계정 `123456789123`의 모든 사용자에게 액세스를 부여합니다.

```
{
   "Version": "2012-10-17",
   "Id": "MyPolicyID",
```

```
   "Statement": [
      {
          "Sid": "MyStatementSid",
          "Effect": "Allow",
          "Principal": {
             "AWS": "arn:aws:iam::123456789123:root"
          },
          "Action": [
             "s3:GetBucketLocation",
             "s3:GetObject",
             "s3:ListBucket",
             "s3:ListBucketMultipartUploads",
             "s3:ListMultipartUploadParts",
             "s3:AbortMultipartUpload"
          ],
          "Resource": [
             "arn:aws:s3:::DOC-EXAMPLE-BUCKET",
             "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
          ]
       }
    ]
}
```

계정의 특정 사용자에게 액세스를 부여하려면, `root` 대신 사용자를 지정하는 키로 `Principal` 키를 바꿉니다. 예를 들어 사용자 프로필 `Dave`의 경우 `arn:aws:iam::123456789123:user/Dave`를 사용합니다.

사용자 지정 AWS KMS 키로 암호화된 버킷에 대한 계정 간 액세스

사용자 지정 AWS Key Management Service(AWS KMS) 키로 암호화된 Amazon S3 버킷이 있는 경우, 다른 Amazon Web Services 계정의 사용자에게 해당 버킷에 대한 액세스 권한을 부여해야 할 수 있습니다.

계정 A의 AWS KMS로 암호화된 버킷에 대한 액세스 권한을 계정 B의 사용자에게 부여하려면 다음 권한이 필요합니다.

- 계정 A의 버킷 정책은 계정 B가 맡은 역할에 액세스 권한을 부여해야 합니다.
- 계정 A의 AWS KMS 키 정책은 계정 B의 사용자가 맡은 역할에 액세스 권한을 부여해야 합니다.
- 계정 B가 맡은 AWS Identity and Access Management(IAM) 역할은 계정 A의 버킷과 키 모두에 대한 액세스 권한을 부여해야 합니다.

다음 절차에서는 이러한 각 권한을 부여하는 방법에 대해 설명합니다.

계정 A의 버킷에 대한 액세스 권한을 계정 B의 사용자에게 부여하려면

- 계정 A에서 S3 버킷 정책을 검토하고 계정 B의 계정 ID에서 액세스할 수 있도록 허용하는 문이 있는지 확인합니다.

  예를 들어 다음 버킷 정책은 `s3:GetObject`에 계정 ID `111122223333`에 대한 액세스를 허용합니다.

```
{
  "Id": "ExamplePolicy1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExampleStmt1",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",
      "Principal": {
        "AWS": [
          "111122223333"
        ]
      }
    }
  ]
}
```

계정 A의 AWS KMS 키 정책에서 계정 B의 사용자에게 액세스 권한을 부여하려면

1. 계정 A의 AWS KMS 키 정책에서 계정 B가 맡은 역할에 다음 작업에 대한 권한을 부여합니다.

   - `kms:Encrypt`
   - `kms:Decrypt`
   - `kms:ReEncrypt*`
   - `kms:GenerateDataKey*`
   - `kms:DescribeKey`

다음 예제에서는 하나의 IAM 역할에만 키 액세스 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowUseOfTheKey",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/role_name"
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        }
    ]
}
```

2. 계정 A에서 AWS Management Console 정책 보기를 사용하여 키 정책을 검토합니다.
3. 키 정책에서 다음 문에 계정 B가 보안 주체로 나열되어 있는지 확인합니다.

```
"Sid": "Allow use of the key"
```

4. `"Sid": "Allow use of the key"` 문이 없으면 다음 단계를 수행하세요.

   a. 콘솔 기본 보기를 사용하여 키 정책을 보도록 전환합니다.

   b. 계정 B의 계정 ID를 키에 대한 액세스 권한이 있는 외부 계정으로 추가합니다.

계정 B가 맡은 IAM 역할에서 계정 A의 버킷 및 키에 대한 액세스 권한을 부여하려면

1. 계정 B에서 IAM 콘솔을 엽니다(https://console.aws.amazon.com/iam/).
2. 계정 B에서 사용자와 연결된 IAM 역할을 엽니다.
3. IAM 역할에 적용된 권한 정책 목록을 검토합니다.
4. 버킷에 대한 액세스 권한을 부여하는 정책이 적용되었는지 확인합니다.

다음 예제 문은 IAM 역할에 버킷 DOC-EXAMPLE-BUCKET의 s3:GetObject 및 s3:PutObject 작업에 대한 액세스 권한을 부여합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExampleStmt2",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
    }
  ]
}
```

5. 키에 대한 액세스 권한을 부여하는 정책이 적용되었는지 확인합니다.

> **Note**
>
> 계정 B가 맡은 IAM 역할에 이미 관리자 액세스 권한이 있으면 사용자의 IAM 정책에서 키에 대한 액세스 권한을 부여할 필요가 없습니다.

다음 예제 문은 IAM 역할에 arn:aws:kms:us-west-2:123456789098:key/111aa2bb-333c-4d44-5555-a111bb2c33dd 키를 사용할 수 있는 액세스 권한을 부여합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExampleStmt3",
      "Action": [
        "kms:Decrypt",
        "kms:DescribeKey",
        "kms:Encrypt",
        "kms:GenerateDataKey",
        "kms:ReEncrypt*"
```

```
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:kms:us-west-2:123456789098:key/111aa2bb-333c-4d44-5555-
a111bb2c33dd"
    }
  ]
}
```

버킷 객체에 대한 교차 계정 액세스

버킷의 소유 계정(계정 A) 이외의 계정(계정 C)에서 업로드한 객체에는 쿼리 계정(계정 B)에 읽기 액세스 권한을 부여하는 명시적인 객체 수준 ACL이 필요할 수 있습니다. 이러한 요구 사항을 피하려면 계정 C가 계정 A의 버킷에 객체를 배치하기 전에 계정 A의 역할을 맡아야 합니다. 자세한 내용은 Amazon S3 버킷에 있는 객체에 대한 교차 계정 액세스 권한을 제공하려면 어떻게 해야 합니까?를 참조하세요.

AWS Glue Data Catalog의 데이터베이스와 테이블에 대한 세분화된 액세스

Amazon Athena와 AWS Glue Data Catalog를 함께 사용하면, Athena에서 사용하는 데이터베이스 및 테이블 데이터 카탈로그 객체에 대한 리소스 수준 정책을 정의할 수 있습니다.

> **Note**
>
> 여기서 "세분화된 액세스 제어"라는 용어는 데이터베이스 및 테이블 수준 보안을 의미합니다. 열 수준, 행 수준, 셀 수준 보안에 대한 자세한 내용은 Lake Formation 포메이션의 데이터 필터링 및 셀 수준 보안 섹션을 참조하세요.

IAM ID 기반 정책에서 리소스 수준 권한을 정의합니다.

> **Important**
>
> 이 단원에서는 IAM ID 기반 정책의 리소스 수준 사용 권한에 대해 설명합니다. 이는 리소스 기반 정책과는 다릅니다. 차이점에 대한 자세한 내용은 IAM 사용 설명서의 ID 기반 정책 및 리소스 기반 정책을 참조하십시오.

이러한 작업에 대해서는 다음 주제를 참조하세요.

이 작업을 수행하려면	다음 주제를 참조하세요.
리소스에 대한 세분화된 액세스를 정의하는 IAM 정책 생성	IAM 사용 설명서의 IAM 정책 생성.
AWS Glue에서 사용하는 IAM ID 기반 정책에 대해 알아봅니다.	AWS Glue 개발자 안내서의 ID 기반 정책(IAM 정책).

이 섹션

- 제한 사항
- AWS 리전당 카탈로그 및 데이터베이스에 대한 AWS Glue 액세스
- AWS Glue의 테이블 파티션 및 버전
- 테이블과 데이터베이스에 대한 세분화된 권한 예제

제한 사항

AWS Glue Data Catalog 및 Athena에서 세분화된 액세스 제어를 사용할 때는 다음 제한 사항을 고려해야 합니다.

- IAM Identity Center 지원 Athena 작업 그룹을 사용하려면 IAM Identity Center ID를 사용하도록 Lake Formation을 구성해야 합니다. 자세한 내용은 AWS Lake Formation 개발자 안내서의 Integrating IAM Identity Center를 참조하십시오.
- 데이터베이스와 테이블에 대한 액세스만 제한할 수 있습니다. 세분화된 액세스 제어는 테이블 수준에서 적용되며 테이블 내 개별 파티션에 대한 액세스는 제한할 수 없습니다. 자세한 내용은 AWS Glue의 테이블 파티션 및 버전 단원을 참조하십시오.
- AWS Glue Data Catalog에는 `CATALOG`, `DATABASE`, `TABLE` 및 `FUNCTION` 리소스가 포함되어 있습니다.

> **Note**
>
> 이 목록에서 Athena와 AWS Glue Data Catalog 간의 공통적인 리소스는 각 계정에 대한 `TABLE`, `DATABASE`, `CATALOG`입니다. `Function`은 AWS Glue에만 적용됩니다. Athena의

삭제 작업의 경우, AWS Glue 작업에 대한 권한을 포함해야 합니다. 테이블과 데이터베이스에 대한 세분화된 권한 예제 섹션을 참조하세요.

계층 구조는 다음과 같습니다. CATALOG는 각 계정 내에서 모든 DATABASES의 상위 항목이고, 각 DATABASE는 자신의 TABLES 및 FUNCTIONS 모두의 상위 항목입니다. 예를 들어, 계정의 카탈로그 내에서 데이터베이스 db에 속한 table_test 테이블의 경우 상위 항목은 계정 내 db와 카탈로그입니다. db 데이터베이스의 경우 상위 항목은 계정 내 카탈로그이고 하위 항목은 테이블과 함수입니다. 리소스의 계층적 구조에 대한 자세한 내용은 AWS Glue 개발자 안내서의 데이터 카탈로그의 ARN 목록을 참조하세요.

- 리소스에 대한 삭제 이외의 모든 Athena 작업의 경우(예: CREATE DATABASE, CREATE TABLE, SHOW DATABASE, SHOW TABLE 또는 ALTER TABLE) 데이터 카탈로그에서 리소스(테이블 또는 데이터베이스)와 리소스의 모든 상위 항목에 대해 해당 작업을 호출할 권한이 필요합니다. 예를 들어, 테이블의 경우 상위 항목은 속하는 데이터베이스와 계정에 대한 카탈로그입니다. 데이터베이스의 경우 상위 항목은 계정의 카탈로그입니다. 테이블과 데이터베이스에 대한 세분화된 권한 예제 섹션을 참조하세요.
- Athena에서의 삭제 작업(예: DROP DATABASE 또는 DROP TABLE)의 경우에는 데이터 카탈로그에서 이 리소스의 모든 상위 항목 및 하위 항목에 대해 삭제 작업을 호출할 권한이 추가로 필요합니다. 예를 들어, 데이터베이스를 삭제하려면 데이터베이스, 카탈로그(데이터베이스의 상위 항목)과 모든 테이블, 사용자 정의 함수(테이블의 하위 항목)에 대한 권한이 필요합니다. 테이블은 하위 항목이 없습니다. DROP TABLE을 실행하려면 테이블, 테이블이 속한 데이터베이스 및 카탈로그에서 이 작업에 대한 사용 권한이 필요합니다. 테이블과 데이터베이스에 대한 세분화된 권한 예제 섹션을 참조하세요.

AWS 리전당 카탈로그 및 데이터베이스에 대한 AWS Glue 액세스

Athena에서 AWS Glue를 사용하려면 AWS 리전당 계정의 AWS Glue Data Catalog 및 데이터베이스에 대한 액세스 권한을 부여하는 정책이 필요합니다. 데이터베이스를 생성하려면 CreateDatabase 권한도 필요합니다. 다음 예제 정책에서는 AWS 리전, AWS 계정 ID 및 데이터베이스 이름을 사용자의 고유한 이름으로 바꿉니다.

```
{
   "Sid": "DatabasePermissions",
   "Effect": "Allow",
   "Action": [
      "glue:GetDatabase",
      "glue:GetDatabases",
```

```
      "glue:CreateDatabase"
    ],
    "Resource": [
      "arn:aws:glue:us-east-1:123456789012:catalog",
      "arn:aws:glue:us-east-1:123456789012:database/default"
    ]
}
```

AWS Glue의 테이블 파티션 및 버전

AWS Glue에서 테이블에는 파티션 및 버전이 있을 수 있습니다. 테이블 버전 및 파티션은 AWS Glue에서 독립적인 리소스로 간주되지 않습니다. 테이블과 테이블의 상위 리소스에 대한 액세스 권한을 부여해 테이블 버전 및 파티션에 대한 액세스 권한을 제공합니다.

세분화된 액세스 제어를 위해서 다음 액세스 권한이 적용됩니다.

- 세분화된 액세스 제어는 테이블 수준에서 적용됩니다. 데이터베이스와 테이블에 대한 액세스만 제한할 수 있습니다. 예를 들어, 분할된 테이블에 대한 액세스를 허용하는 경우 해당 액세스 권한은 테이블의 모든 파티션에 적용됩니다. 한 테이블의 개별 파티션에 대한 액세스는 제한할 수 없습니다.

  > **Important**
  >
  > AWS Glue에서 파티션에 대해 작업을 실행하려면 카탈로그, 데이터베이스 및 테이블 수준에서 파티션 작업에 대한 권한이 필요합니다. 테이블 내의 파티션에 액세스하는 것만으로는 충분하지 않습니다. 예를 들어 `myDB` 데이터베이스의 `myTable` 테이블에서 `GetPartitions`를 실행하려면 카탈로그, `myDB` 데이터베이스 및 `myTable` 리소스에 대한 `glue:GetPartitions`의 권한을 부여해야 합니다.

- 세분화된 액세스 제어는 테이블 버전에는 적용되지 않습니다. 파티션과 마찬가지로, 테이블의 이전 버전에 대한 액세스 권한은 AWS Glue에서 테이블, 테이블 상위 항목 및 테이블 버전 API에 대한 액세스를 통해 부여됩니다.

AWS Glue 작업의 권한에 대한 자세한 내용은 AWS Glue 개발자 안내서의 AWS Glue API 권한: 작업 및 리소스 참조를 참조하세요.

테이블과 데이터베이스에 대한 세분화된 권한 예제

다음 표는 Athena에서 데이터베이스와 테이블에 대한 세분화된 액세스를 허용하는 IAM ID 기반 정책의 예제를 보여줍니다. 처음에는 이 예제로 시작한 다음, 자신의 필요에 따라 특정 데이터베이스와 테이블에 대한 특정 작업을 허용하거나 거부하도록 수정하는 방법을 권장합니다.

이 예제에는 Athena와 AWS Glue를 함께 사용할 수 있도록 데이터베이스 및 카탈로그에 대한 액세스가 포함됩니다. 여러 AWS 리전의 경우 리전별로 한 줄씩 각각의 데이터베이스 및 카탈로그에 대해 유사한 정책을 포함합니다.

예제에서는 `example_db` 데이터베이스와 `test` 테이블 이름을 사용자의 고유한 데이터베이스 및 테이블 이름으로 바꿉니다.

DDL 문	리소스에 대한 액세스를 부여하는 IAM 액세스 정책 예제
ALTER DATABASE	`example_db` 데이터베이스의 속성을 수정할 수 있습니다. `{ "Effect": "Allow", "Action": [ "glue:GetDatabase", "glue:UpdateDatabase" ], "Resource": [ "arn:aws:glue: us-east-1 :123456789012 :catalog", "arn:aws:glue: us-east-1 :123456789012 :database / example_db " ] }`
데이터베이스 생성	`example_db` (이)라는 데이터베이스를 생성할 수 있습니다. `{ "Effect": "Allow", "Action": [ "glue:GetDatabase", "glue:CreateDatabase" ], "Resource": [ "arn:aws:glue: us-east-1 :123456789012 :catalog", "arn:aws:glue: us-east-1 :123456789012 :database / example_db " ] }`
CREATE TABLE	`example_db` 데이터베이스에 `test`(이)라는 테이블을 생성할 수 있습니다.

DDL 문	리소스에 대한 액세스를 부여하는 IAM 액세스 정책 예제

```
{
   "Sid": "DatabasePermissions",
   "Effect": "Allow",
   "Action": [
      "glue:GetDatabase",
      "glue:GetDatabases"
   ],
   "Resource": [
     "arn:aws:glue: us-east-1 :123456789012 :catalog",
     "arn:aws:glue: us-east-1 :123456789012 :database
/ example_db "
   ]
},
{
   "Sid": "TablePermissions",
   "Effect": "Allow",
   "Action": [
      "glue:GetTables",
      "glue:GetTable",
      "glue:GetPartitions",
      "glue:CreateTable"
   ],
   "Resource": [
     "arn:aws:glue: us-east-1 :123456789012 :catalog",
     "arn:aws:glue: us-east-1 :123456789012 :database
/ example_db ",
     "arn:aws:glue: us-east-1 :123456789012 :table/example_d
b /test"
   ]
}
```

DDL 문	리소스에 대한 액세스를 부여하는 IAM 액세스 정책 예제

DROP DATABASE	`example_db` 데이터베이스를 포함된 모든 테이블과 함께 삭제할 수 있습니다.
	
	{
	"Effect": "Allow",
	"Action": [
	"glue:GetDatabase",
	"glue:DeleteDatabase",
	"glue:GetTables",
	"glue:GetTable",
	"glue:DeleteTable"
	],
	"Resource": [
	"arn:aws:glue: *us-east-1* :*123456789012* :catalog",
	"arn:aws:glue: *us-east-1* :*123456789012* :database / *example_db* ",
	"arn:aws:glue: *us-east-1* :*123456789012* :table/*example_db* /*",
	"arn:aws:glue: *us-east-1* :*123456789012* :userDefinedFunction/ *example_db* /*"
	]
	}

DDL 문	리소스에 대한 액세스를 부여하는 IAM 액세스 정책 예제

DROP TABLE	example_db 데이터베이스에 test(이)라는 분할된 테이블을 삭제할 수 있습니다. 테이블에 파티션이 없다면 파티션 작업을 포함하지 마세요.
	
	{
	"Effect": "Allow",
	"Action": [
	"glue:GetDatabase",
	"glue:GetTable",
	"glue:DeleteTable",
	"glue:GetPartitions",
	"glue:GetPartition",
	"glue:DeletePartition"
	],
	"Resource": [
	"arn:aws:glue: *us-east-1* :*123456789012* :catalog",
	"arn:aws:glue: *us-east-1* :*123456789012* :database / *example_db* ",
	"arn:aws:glue: *us-east-1* :*123456789012* :table/*example_db* /*test*"
	]
	}

DDL 문	리소스에 대한 액세스를 부여하는 IAM 액세스 정책 예제
MSCK REPAIR TABLE	`example_db` 데이터베이스에서 `test`라는 테이블에 Hive 호환 파티션을 추가한 후 카탈로그 메타데이터를 업데이트할 수 있습니다. `{ "Effect": "Allow", "Action": ["glue:GetDatabase", "glue:CreateDatabase", "glue:GetTable", "glue:GetPartitions", "glue:GetPartition", "glue:BatchCreatePartition"], "Resource": ["arn:aws:glue: us-east-1 :123456789012 :catalog", "arn:aws:glue: us-east-1 :123456789012 :database / example_db ", "arn:aws:glue: us-east-1 :123456789 012 :table/example_db /test"] }`
SHOW DATABASES	AWS Glue Data Catalog의 모든 데이터베이스를 나열할 수 있습니다. `{ "Effect": "Allow", "Action": [ "glue:GetDatabase", "glue:GetDatabases" ], "Resource": [ "arn:aws:glue: us-east-1 :123456789012 :catalog", "arn:aws:glue: us-east-1 :123456789012 :database/*" ] }`

DDL 문	리소스에 대한 액세스를 부여하는 IAM 액세스 정책 예제

SHOW TABLES	example_db 의 모든 테이블을 나열할 수 있습니다.
	
	{
	"Effect": "Allow",
	"Action": [
	"glue:GetDatabase",
	"glue:GetTables"
	],
	"Resource": [
	"arn:aws:glue: *us-east-1* :*123456789012* :catalog",
	"arn:aws:glue: *us-east-1* :*123456789012* :database / *example_db* ",
	"arn:aws:glue: *us-east-1* :*123456789012* :table/*example_db* /*"
	]
	}

AWS Glue 데이터 카탈로그에 대한 교차 계정 액세스

Athena의 계정 간 AWS Glue 카탈로그 기능을 사용해 자신의 계정이 아닌 다른 계정의 AWS Glue 카탈로그를 등록할 수 있습니다. AWS Glue에 요구되는 IAM 권한을 구성하고 카탈로그를 Athena DataCatalog 리소스로 등록하면 Athena를 사용해 계정 간 쿼리를 실행할 수 있습니다. Athena 콘솔을 사용하여 다른 계정의 카탈로그를 등록하는 방법에 대한 자세한 내용은 다른 계정의 AWS Glue Data Catalog 등록 단원을 참조하세요.

AWS Glue에서 계정 간 액세스에 대한 자세한 내용은 AWS Glue 개발자 안내서의 계정 간 액세스 권한 부여를 참조하세요.

시작하기 전에

이 기능은 기존의 Athena DataCatalog 리소스 API 및 계정 간 액세스를 활성화하는 기능을 사용하므로 시작하기 전에 다음 리소스를 읽어보는 것이 좋습니다.

- 데이터 원본에 연결 - Athena와 함께 AWS Glue, Hive 또는 Lambda 데이터 카탈로그 소스를 사용하는 주제를 포함하고 있습니다.
- 데이터 카탈로그 예제 정책 - 데이터 카탈로그에 대한 액세스를 제어하는 정책을 작성하는 방법을 보여 줍니다.

- Hive 메타스토어와 함께 AWS CLI 사용 - Hive 메타스토어와 함께 AWS CLI를 사용하는 방법을 보여줍니다. 다른 데이터 소스에 적용한 사용 사례도 포함되어 있습니다.

고려 사항 및 제한

현재 Athena 계정 간 AWS Glue 카탈로그에는 다음과 같은 제한 사항이 있습니다.

- Athena 엔진 버전 2 이상이 지원되는 AWS 리전에서만 기능을 사용할 수 있습니다. Athena 엔진 버전에 대한 자세한 내용은 Athena 엔진 버전 관리 단원을 참조하세요. 작업 그룹의 엔진 버전을 업그레이드하려면 Athena 엔진 버전 변경 섹션을 참조하세요.
- 다른 계정의 AWS Glue Data Catalog를 자신의 계정에 등록할 때 해당 특정 리전에 있는 다른 계정의 데이터에만 연결되는 리전별 `DataCatalog` 리소스를 생성합니다.
- 현재 계정 간 AWS Glue 카탈로그를 포함한 `CREATE VIEW` 문은 지원되지 않습니다.
- AWS 관리형 키를 사용하여 암호화된 카탈로그는 여러 계정에서 쿼리할 수 없습니다. 여러 계정에서 쿼리하려는 카탈로그의 경우 고객 관리형 키(`KMS_CMK`)를 대신 사용하세요. 고객 관리형 및 AWS 관리형 키의 차이점에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 고객 키 및 AWS 키를 참조하세요.

시작하기

다음 상황에서는 다음 예제처럼 '차용자' 계정(666666666666)이 AWS Glue 카탈로그를 참조하는 `SELECT` 쿼리를 실행하려 합니다. 이 카탈로그는 '소유자' 계정(999999999999)에 속해 있습니다.

```
SELECT * FROM ownerCatalog.tpch1000.customer
```

다음 절차에서 1a 및 1b 단계는 차용자와 소유자의 관점에서 소유자 계정의 AWS Glue 리소스에 대한 액세스 권한을 차용자 계정에 부여하는 방법을 보여줍니다. 이 예제에서는 `tpch1000` 데이터베이스와 `customer` 테이블에 액세스 권한을 부여합니다. 이러한 예제 이름을 요구 사항에 맞게 변경합니다.

1a 단계: 소유자의 AWS Glue 리소스에 액세스하는 정책을 가진 차용자 역할 생성

소유자 계정의 AWS Glue 리소스에 대한 액세스 권한이 포함된 차용자 계정 역할을 생성하려면 AWS Identity and Access Management(IAM) 콘솔 또는 IAM API를 사용할 수 있습니다. 다음 절차에서는 IAM 콘솔을 사용합니다.

차용자 역할 및 정책을 생성하여 소유자 계정의 AWS Glue 리소스에 액세스

1. 차용자 계정에서 IAM 콘솔에 로그인합니다(https://console.aws.amazon.com/iam/).

2. 서비스 탐색 창에서 액세스 관리를 확장하고 정책을 선택합니다.
3. 정책 생성을 선택합니다.
4. 정책 편집기에서 JSON을 선택합니다.
5. 정책 편집기에서 다음 정책을 입력한 다음 필요에 따라 수정합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "glue:*",
            "Resource": [
                "arn:aws:glue:us-east-1:999999999999:catalog",
                "arn:aws:glue:us-east-1:999999999999:database/tpch1000",
                "arn:aws:glue:us-east-1:999999999999:table/tpch1000/customer"
            ]
        }
    ]
}
```

6. 다음을 선택합니다.
7. 검토 및 생성 페이지의 정책 이름에 정책 이름(예: **CrossGluePolicyForBorrowerRole**)을 입력합니다.
8. 정책 생성을 선택합니다.
9. 탐색 창에서 역할을 선택합니다.
10. 역할 생성을 선택합니다.
11. 신뢰할 수 있는 엔티티 페이지에서 AWS 계정 선택 후 다음을 선택합니다.
12. 권한 추가 페이지에서 검색 상자에 생성한 정책의 이름을 입력합니다(예: **CrossGluePolicyForBorrowerRole**).
13. 정책 이름 옆에 있는 확인란을 선택하고, 다음을 선택합니다.
14. 이름, 검토 및 생성 페이지에서 역할 이름(Role name)에서 역할 이름을 입력합니다(예: **CrossGlueBorrowerRole**).
15. 역할 생성을 선택합니다.

1b 단계: 차용자에 대해 AWS Glue 액세스 권한을 부여하는 소유자 정책 생성

소유자 계정(999999999999)으로부터 차용자의 역할에 AWS Glue 액세스 권한을 부여하려면 AWS Glue 콘솔 또는 AWS Glue PutResourcePolicy API 작업을 사용합니다. 다음 절차에서는 AWS Glue 콘솔을 사용합니다.

소유자로부터 차용자 계정에 AWS Glue 액세스 권한을 부여하려면

1. 소유자 계정에서 AWS Glue 콘솔에 로그인합니다(https://console.aws.amazon.com/glue/).
2. 탐색 창에서 데이터 카탈로그를 확장하고 카탈로그 설정을 선택합니다.
3. 권한(Permissions) 상자에 다음과 같은 정책을 입력합니다. *rolename*에 1a 단계에서 차용자 계정이 생성한 역할(예: **CrossGlueBorrowerRole**)을 입력합니다. 권한 범위를 늘리고자 한다면 데이터베이스 및 테이블 리소스 유형에 와일드카드 문자 *를 사용할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::666666666666:user/username",
                    "arn:aws:iam::666666666666:role/rolename"
                ]
            },
            "Action": "glue:*",
            "Resource": [
                "arn:aws:glue:us-east-1:999999999999:catalog",
                "arn:aws:glue:us-east-1:999999999999:database/tpch1000",
                "arn:aws:glue:us-east-1:999999999999:table/tpch1000/customer"
            ]
        }
    ]
}
```

완료 후 AWS Glue API를 사용해 계정 간 호출을 몇 차례 테스트하여 권한이 예상대로 구성되었는지 확인하는 것이 좋습니다.

2단계: 차용자가 소유자 계정에 속한 AWS Glue Data Catalog 등록

다음 절차는 Athena 콘솔을 사용하여 소유자 Amazon Web Services 계정의 AWS Glue Data Catalog를 데이터 소스로 구성하는 방법을 보여줍니다. 콘솔 대신 API 작업을 사용하여 카탈로그를 등록하는 방법에 대한 자세한 내용은 API를 사용하여 소유자 계정에 속한 Athena 데이터 카탈로그 등록 단원을 참조하세요.

다른 계정에 속한 AWS Glue Data Catalog 등록

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

3. 관리를 확장하고 데이터 소스를 선택합니다.
4. 오른쪽 위에서 데이터 소스 생성(Create data source)을 선택합니다.
5. 데이터 소스 선택 페이지의 데이터 소스에서 S3 - AWS Glue Data Catalog를 선택하고 다음을 선택합니다.
6. 데이터 소스 세부 정보 입력(Enter data source details) 페이지의 AWS Glue Data Catalog 섹션에서 AWS Glue Data Catalog 선택(Choose an AWS Glue Data Catalog)에 다른 계정의 AWS Glue Data Catalog(AWS Glue Data Catalog in another account)를 선택합니다.
7. 데이터 세트 세부 정보(Dataset details)에 다음 정보를 입력합니다.
   - 데이터 소스 이름(Data source name) – 다른 계정의 데이터 카탈로그를 참조할 때 SQL 쿼리에 사용할 이름을 입력합니다.
   - 설명(Description) - (선택 사항) 다른 계정의 데이터 카탈로그에 대한 설명을 입력합니다.

- 카탈로그 ID(Catalog ID) - 데이터 카탈로그가 속한 계정의 Amazon Web Services 계정 ID(12자리)를 입력합니다. Amazon Web Services 계정 ID는 카탈로그 ID입니다.

8. (선택 사항) 태그를 확장하고 데이터 소스에 연결할 키 값 페어를 입력합니다. 태그에 대한 자세한 내용은 Athena 리소스 태깅 단원을 참조하세요.
9. 다음을 선택합니다.
10. 검토 및 생성(Review and create) 페이지에서 제공한 정보를 검토한 다음 데이터 소스 생성(Create data source)을 선택합니다. 데이터 소스 세부 정보(Data source details) 페이지는 등록한 데이터 카탈로그에 대한 데이터베이스 및 태그를 나열합니다.
11. 데이터 소스(Data Source)을 선택합니다. 등록한 데이터 카탈로그가 데이터 소스 이름(Data source name) 열에 나열됩니다.
12. 새 데이터 카탈로그에 대한 정보를 보거나 편집하려면 카탈로그를 선택한 다음 작업(Actions), 편집(Edit)을 선택합니다.
13. 새 데이터 카탈로그를 삭제하려면 카탈로그를 선택한 다음 작업(Actions), 삭제(Delete)를 선택합니다.

3단계: 차용자가 쿼리 제출

다음 예제처럼 차용자가 *catalog.database.table* 구문을 사용해 카탈로그를 참조하는 쿼리를 제출합니다.

```
SELECT * FROM ownerCatalog.tpch1000.customer
```

정규화된 구문을 사용하는 대신 차용자는 QueryExecutionContext를 통해 이를 전달하여 상황에 맞게 카탈로그를 지정할 수도 있습니다.

추가적인 Amazon S3 권한

- 차용자 계정이 Athena 쿼리를 사용하여 소유자 계정의 테이블에 새 데이터를 쓰는 경우 테이블이 소유자 계정에 있더라도 소유자는 Amazon S3의 해당 데이터에 대한 액세스 권한을 자동으로 보유하지 않습니다. 달리 구성하지 않는 한 차용자가 Amazon S3에 있는 정보의 객체 소유자이기 때문입니다. 소유자에게 데이터에 대한 액세스 권한을 부여하려면 추가 단계로 개체에 대한 권한을 적절히 설정합니다.
- MSCK REPAIR TABLE 같은 특정한 계정 간 DDL 작업은 Amazon S3 권한을 요구합니다. 예를 들어 소유자 계정 S3 버킷에 있는 데이터를 포함한 소유자 계정의 테이블에 대해 차용자 계정이 계정 간 `MSCK REPAIR` 작업을 수행할 경우 쿼리가 성공하려면 해당 버킷이 차용자가 맡은 역할에 권한을 부여해야 합니다.

버킷 권한 부여에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 ACL 버킷 권한을 설정하는 방법을 참조하세요.

동적으로 카탈로그 사용

어떤 경우에는 계정 간 AWS Glue 카탈로그를 등록하는 사전 단계 없이 신속히 테스트를 수행해야 할 수 있습니다. 앞서 이 설명서에서 설명한 필수 IAM 및 Amazon S3 권한을 올바르게 구성했다면 `DataCatalog` 리소스를 생성하지 않고도 계정 간 쿼리를 동적으로 수행할 수 있습니다.

등록 없이 카탈로그를 명시적으로 참조하려면 다음 예제의 구문을 사용합니다.

```
SELECT * FROM "glue:arn:aws:glue:us-east-1:999999999999:catalog".tpch1000.customer
```

"`glue:<arn>`" 형식을 사용합니다. 여기서 *<arn>*은(는) 사용하고자 하는 AWS Glue Data Catalog ARN입니다. 이 예제에서 Athena는 이 구문을 사용하여 마치 `DataCatalog` 객체를 별도로 생성한 것처럼 999999999999 계정의 AWS Glue 데이터 카탈로그를 동적으로 가리키도록 할 수 있습니다.

동적 카탈로그 사용에 대한 참고 사항

동적 카탈로그를 사용할 때 다음 사항에 유의해야 합니다.

- 동적 카탈로그를 사용하려면 Athena 데이터 카탈로그 API 작업에 일반적으로 사용하는 IAM 권한이 필요합니다. 주된 차이점은 데이터 카탈로그 리소스 이름이 `glue:*` 이름 지정 규칙을 따른다는 것입니다.
- 카탈로그 ARN은 쿼리가 실행되는 리전과 동일한 리전에 속해야 합니다.
- DML 쿼리 또는 뷰에서 동적 카탈로그를 사용할 경우 이스케이프된 큰따옴표(\")로 묶어야 합니다. DDL 쿼리에서 동적 카탈로그를 사용할 경우 백틱 문자(`)로 묶어야 합니다.

API를 사용하여 소유자 계정에 속한 Athena 데이터 카탈로그 등록

2단계에서 설명한 대로 Athena 콘솔을 사용하는 대신 API 작업을 사용하여 소유자 계정에 속한 데이터 카탈로그를 등록할 수 있습니다.

Athena DataCatalog 리소스의 생성자는 Athena CreateDataCatalog API 작업을 실행하는 데 필요한 권한이 있어야 합니다. 요구 사항에 따라 추가 API 작업에 액세스해야 할 수도 있습니다. 자세한 내용은 데이터 카탈로그 예제 정책 단원을 참조하세요.

다음 `CreateDataCatalog` 요청 본문은 계정 간 액세스를 위한 AWS Glue 카탈로그를 등록합니다.

```
# Example CreateDataCatalog request to register a cross-account Glue catalog:
```

```
{
    "Description": "Cross-account Glue catalog",
    "Name": "ownerCatalog",
    "Parameters": {"catalog-id" : "999999999999"  # Owner's account ID
    },
    "Type": "GLUE"
}
```

다음 샘플 코드는 Java 클라이언트를 사용하여 DataCatalog 객체를 생성합니다.

```
# Sample code to create the DataCatalog through Java client
CreateDataCatalogRequest request = new CreateDataCatalogRequest()
    .withName("ownerCatalog")
    .withType(DataCatalogType.GLUE)
    .withParameters(ImmutableMap.of("catalog-id", "999999999999"));

athenaClient.createDataCatalog(request);
```

이 단계가 끝나면 ListDataCatalogs API 작업을 호출할 때 차용자에게 ownerCatalog가 표시되어야 합니다.

추가적인 리소스

- 다른 계정의 AWS Glue Data Catalog 등록
- AWS Prescriptive Guidance Patterns 안내서의 Configure cross-account access to a shared AWS Glue Data Catalog using Amazon Athena.
- AWS 빅 데이터 블로그의 Query cross-account AWS Glue Data Catalogs using Amazon Athena
- AWS Glue 개발자 안내서의 Granting cross-account access

Athena에서 AWS Glue Data Catalog의 암호화된 메타데이터에 액세스

Amazon Athena와 함께 AWS Glue Data Catalog를 사용하는 경우 AWS Glue 콘솔이나 API를 이용해 AWS Glue Data Catalog에서 암호화를 사용할 수 있습니다. 자세한 내용은 AWS Glue 개발자 안내서의 데이터 카탈로그 암호화를 참조하세요.

AWS Glue Data Catalog를 암호화한다면, Athena에 액세스하는 데 사용하는 모든 정책에 다음 작업을 추가해야 합니다.

```
{
```

```
 "Version": "2012-10-17",
 "Statement": {
 "Effect": "Allow",
     "Action": [
           "kms:GenerateDataKey",
           "kms:Decrypt",
           "kms:Encrypt"
      ],
     "Resource": "(arn of the key used to encrypt the catalog)"
   }
}
```

IAM 정책을 사용할 때는 항상 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

작업 그룹 및 태그에 대한 액세스

작업 그룹은 Athena에서 관리하는 리소스입니다. 따라서 해당 작업 그룹 정책에서 `workgroup`을 입력으로 받는 작업을 사용할 경우 다음과 같이 작업 그룹의 ARN을 지정해야 하며 *`workgroup-name`*이 작업 그룹의 이름입니다..

```
"Resource": [arn:aws:athena:region:AWSAcctID:workgroup/workgroup-name]
```

예를 들어 Amazon Web Services 계정 `123456789012`의 `us-west-2` 리전에 있는 `test_workgroup`이라는 이름의 작업 그룹의 경우 다음 ARN을 사용하여 작업 그룹을 리소스로 지정합니다.

```
"Resource":["arn:aws:athena:us-east-2:123456789012:workgroup/test_workgroup"]
```

신뢰할 수 있는 ID 전파(TIP) 지원 작업 그룹에 액세스하려면 Athena GetWorkGroup API 작업의 응답으로 반환되는 `IdentityCenterApplicationArn`에 IAM Identity Center 사용자를 할당해야 합니다.

- 작업 그룹 정책 목록은 the section called “작업 그룹 정책의 예” 단원을 참조하세요.
- 작업 그룹에 대한 태그 기반 정책 목록은 태그 기반 IAM 액세스 제어 정책 단원을 참조하세요.
- IAM 정책을 생성하는 방법에 대한 자세한 내용은 작업 그룹 액세스를 위한 IAM 정책 단원을 참조하세요.
- Amazon Athena 작업의 전체 목록은 Amazon Athena API 참조의 API 작업 이름을 참조하세요.
- IAM 정책에 대한 자세한 내용은 IAM 사용 설명서의 시각적 편집기로 정책 생성을 참조하세요.

IAM 정책을 사용할 때는 항상 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

준비된 문에 대한 액세스 허용

이 주제에서는 Amazon Athena의 준비된 문에 대한 IAM 권한에 대해 다룹니다. IAM 정책을 사용할 때는 항상 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

준비된 문에 대한 자세한 내용은 파라미터화된 쿼리 사용 단원을 참조하세요.

준비된 문을 생성, 관리, 실행하는 데 필요한 IAM 권한은 다음과 같습니다.

```
athena:CreatePreparedStatement
athena:UpdatePreparedStatement
athena:GetPreparedStatement
athena:ListPreparedStatements
athena:DeletePreparedStatement
```

다음 표에 나타난 대로 이 권한을 사용해야 합니다.

목적	사용할 권한
PREPARE 쿼리 실행	athena:StartQueryExecution athena:CreatePreparedStatement
기존의 준비된 문을 업데이트하기 위해 PREPARE 쿼리 재실행	athena:StartQueryExecution athena:UpdatePreparedStatement
EXECUTE 쿼리 실행	athena:StartQueryExecution athena:GetPreparedStatement
DEALLOCATE PREPARE 쿼리 실행	athena:StartQueryExecution athena:DeletePreparedStatement

예

다음의 IAM 정책 예제는 지정된 계정 ID 및 작업 그룹에 준비된 문을 관리하고 실행할 수 있는 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:StartQueryExecution",
                "athena:CreatePreparedStatement",
                "athena:UpdatePreparedStatement",
                "athena:GetPreparedStatement",
                "athena:DeletePreparedStatement",
                "athena:ListPreparedStatements"
            ],
            "Resource": [
                "arn:aws:athena:*:111122223333:workgroup/<workgroup-name>"
            ]
        }
    ]
}
```

Athena와 CalledVia 컨텍스트 키 사용

보안 주체가 AWS에 요청하면 AWS는 요청을 평가하고 권한을 부여하는 요청 컨텍스트로 요청 정보를 수집합니다. JSON 정책의 `Condition` 요소를 사용하여 요청 컨텍스트의 키를 정책에서 지정한 키 값과 비교할 수 있습니다. 글로벌 조건 컨텍스트 키는 `aws:` 접두사가 있는 조건 키입니다.

aws:CalledVia 컨텍스트 키

aws:CalledVia 글로벌 컨텍스트 키를 사용하면 정책의 서비스를, IAM 보안 주체(사용자 또는 역할)를 대신하여 요청을 수행한 서비스와 비교할 수 있습니다. 보안 주체가 AWS 서비스에 요청하면 해당 서비스는 보안 주체의 자격 증명을 사용하여 다른 서비스에 대한 후속 요청을 할 수 있습니다. `aws:CalledVia` 키에는 보안 주체를 대신하여 요청을 수행한 체인의 각 서비스 목록이 정렬되어 있습니다.

`aws:CalledVia` 컨텍스트 키에 대한 서비스 주체 이름을 지정하여 컨텍스트 키를 AWS 서비스별로 지정할 수 있습니다. 예를 들어, `aws:CalledVia` 조건 키를 사용해 Athena에서 수행된 요청으로만 요청을 제한할 수 있습니다. Athena에서 어떤 정책에 `aws:CalledVia` 조건 키를 사용하려면 다음 예제처럼 Athena 서비스 보안 주체 이름 `athena.amazonaws.com`을 지정합니다.

```
...
   "Condition": {
```

```
        "ForAnyValue:StringEquals": {
            "aws:CalledVia": "athena.amazonaws.com"
        }
    }
...
```

aws:CalledVia 컨텍스트 키는 발신자가 Athena로부터 리소스를 호출할 때 발신자만 리소스(예: Lambda 함수)에 대한 액세스 권한을 갖도록 하는 데 사용할 수 있습니다.

> **Note**
>
> aws:CalledVia 컨텍스트 키는 신뢰할 수 있는 ID 전파 기능과 호환되지 않습니다.

Lambda 함수에 대한 세분화된 액세스를 위해 CalledVia 컨텍스트 키 추가(선택 사항)

Athena에서는 쿼리와 연결된 Lambda 함수를 호출하려면 발신자에게 lambda:InvokeFunction 권한이 있어야 합니다. 다음 문을 사용하면 사용자가 Athena만을 사용하여 Lambda 함수를 호출할 수 있도록 Lambda 함수에 대한 액세스 권한을 세분화할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-
east-1:111122223333:function:OneAthenaLambdaFunction",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": "athena.amazonaws.com"
                }
            }
        }
    ]
}
```

다음 예제에서는 사용자가 연합 쿼리를 실행하고 읽도록 허용하는 정책에 앞의 문을 추가하는 방법을 보여줍니다. 이러한 작업을 수행할 수 있는 보안 주체는 연합 데이터 원본과 연결된 Athena 카탈로그

를 지정하는 쿼리를 실행할 수 있습니다. 그러나 보안 주체는 Athena를 통해 함수가 호출되지 않는 한 연결된 Lambda 함수에 액세스할 수 없습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "athena:GetWorkGroup",
                "s3:PutObject",
                "s3:GetObject",
                "athena:StartQueryExecution",
                "s3:AbortMultipartUpload",
                "athena:StopQueryExecution",
                "athena:GetQueryExecution",
                "athena:GetQueryResults",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:athena:*:111122223333:workgroup/WorkGroupName",
                "arn:aws:s3:::MyQueryResultsBucket/*",
                "arn:aws:s3:::MyLambdaSpillBucket/MyLambdaSpillPrefix*"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "athena:ListWorkGroups",
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action":
                [
                "s3:ListBucket",
                "s3:GetBucketLocation"
                ],
            "Resource": "arn:aws:s3:::MyLambdaSpillBucket"
        },
        {
```

```
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": [
                "arn:aws:lambda:*:111122223333:function:OneAthenaLambdaFunction",
                "arn:aws:lambda:*:111122223333:function:AnotherAthenaLambdaFunction"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": "athena.amazonaws.com"
                }
            }
        }
    ]
}
```

CalledVia 글로벌 조건 키에 대한 자세한 내용은 IAM 사용 설명서의 AWS 글로벌 조건 컨텍스트 키를 참조하세요.

외부 Hive 메타스토어용 Athena 데이터 커넥터에 대한 액세스 허용

이 주제의 권한 정책 예제에서는 필요한 허용된 작업과 해당 작업이 허용되는 리소스를 보여줍니다. IAM 자격 증명에 유사한 권한 정책을 연결하기 전에 이러한 정책을 신중하게 검토하고 요구 사항에 따라 수정합니다.

- Example Policy to Allow an IAM Principal to Query Data Using Athena Data Connector for External Hive Metastore
- Example Policy to Allow an IAM Principal to Create an Athena Data Connector for External Hive Metastore

Example – IAM 보안 주체가 외부 Hive 메타스토어용 Athena 데이터 커넥터를 사용하여 데이터를 쿼리할 수 있도록 허용합니다.

다음 정책은 AWS 관리형 정책: AmazonAthenaFullAccess 외에 Athena 작업에 대한 전체 액세스 권한을 부여하는 IAM 보안 주체에도 연결됩니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor1",
```

```
            "Effect": "Allow",
            "Action": [
                "lambda:GetFunction",
                "lambda:GetLayerVersion",
                "lambda:InvokeFunction"
            ],
            "Resource": [
                "arn:aws:lambda:*:111122223333:function:MyAthenaLambdaFunction",
                "arn:aws:lambda:*:111122223333:function:AnotherAthenaLambdaFunction",
                "arn:aws:lambda:*:111122223333:layer:MyAthenaLambdaLayer:*"
            ]
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:PutObject",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload"
            ],
            "Resource": "arn:aws:s3:::MyLambdaSpillBucket/MyLambdaSpillLocation"
        }
    ]
}
```

권한에 대한 설명

허용된 작업	설명
`"s3:GetBucketLocation", "s3:GetObject", "s3:ListBucket", "s3:PutObject", "s3:ListMultipartUploadParts", "s3:AbortMultipartUpload"`	s3 작업을 사용하면 "arn:aws:s3::: *MyLambdaSpillBucket* /*MyLambdaSpillLocation* "으로 지정된 리소스를 읽고 쓸 수 있습니다. 여기서 *MyLambdaSpillLocation* 은 호출되는 Lambda 함수의 구성에 지정된 유출 버킷(spill bucket)을 식별합니다. *arn:aws:lambda:*: MyAWSAcct Id :layer:MyAthenaLambdaLayer :** 리소스 식별자는 배포 시 함수 아티팩트 크기를 줄

허용된 작업	설명
	이기 위해 Athena 계층을 사용하여 사용자 지정 런타임 종속성을 만드는 경우에만 필요합니다. 마지막 위치에 있는 *은 계층 버전의 와일드카드입니다.
`"lambda:GetFunction", "lambda:GetLayerVersion", "lambda:InvokeFunction"`	쿼리가 Resource 블록에 지정된 AWS Lambda 함수를 호출할 수 있습니다. `arn:aws:lambda:*:` *MyAWSAcctId* `:function:` *MyAthenaLambdaFunction* 을 예로 들 수 있습니다. 여기서 *MyAthenaLambdaFunction* 은 호출할 Lambda 함수의 이름을 지정합니다. 예제와 같이 여러 함수를 지정할 수 있습니다.

Example – IAM 보안 주체가 외부 Hive 메타스토어용 Athena 데이터 커넥터를 생성할 수 있도록 허용합니다.

다음 정책은 AWS 관리형 정책: AmazonAthenaFullAccess 외에 Athena 작업에 대한 전체 액세스 권한을 부여하는 IAM 보안 주체에도 연결됩니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "lambda:GetFunction",
                "lambda:ListFunctions",
                "lambda:GetLayerVersion",
                "lambda:InvokeFunction",
                "lambda:CreateFunction",
                "lambda:DeleteFunction",
                "lambda:PublishLayerVersion",
                "lambda:DeleteLayerVersion",
                "lambda:UpdateFunctionConfiguration",
                "lambda:PutFunctionConcurrency",
                "lambda:DeleteFunctionConcurrency"
```

```
            ],
            "Resource": "arn:aws:lambda:*:111122223333:
 function: MyAthenaLambdaFunctionsPrefix*"
        }
    ]
}
```

권한에 대한 설명

쿼리가 Resource 블록에 지정된 AWS Lambda 함수에 대한 AWS Lambda 함수를 호출할 수 있습니다. arn:aws:lambda:*:*MyAWSAcctId*:function:*MyAthenaLambdaFunction*을 예로 들 수 있습니다. 여기서 *MyAthenaLambdaFunction*은 호출되는 Lambda 함수의 이름을 지정합니다. 예제와 같이 여러 함수를 지정할 수 있습니다.

외부 Hive 메타스토어에 대한 Lambda 함수 액세스 허용

사용자 계정에서 Lambda 함수를 호출하려면 다음 권한을 가진 역할을 만들어야 합니다.

- AWSLambdaVPCAccessExecutionRole – 함수를 VPC에 연결하는 탄력적 네트워크 인터페이스를 관리하기 위한 AWS Lambda 실행 역할 권한입니다. 충분한 수의 네트워크 인터페이스 및 IP 주소를 사용할 수 있는지 확인합니다.
- AmazonAthenaFullAccess – AmazonAthenaFullAccess 관리형 정책은 Athena에 대한 모든 액세스 권한을 부여합니다.
- Lambda 함수가 S3에 쓸 수 있도록 허용하고 Athena가 S3에서 읽을 수 있도록 허용하는 Amazon S3 정책입니다.

예를 들어 다음 정책은 유출 위치 s3:\\mybucket\spill에 대한 권한을 정의합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/spill"
```

```
            ]
        }
    ]
}
```

IAM 정책을 사용할 때는 항상 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

Lambda 함수 생성

사용자 계정에서 Lambda 함수를 만들려면 함수 개발 권한 또는 `AWSLambdaFullAccess` 역할이 필요합니다. 자세한 내용은 AWS Lambda에 대한 자격 증명 기반 IAM 정책을 참조하세요.

Athena는 AWS Serverless Application Repository를 사용해 Lambda 함수를 생성하기 때문에 Lambda 함수를 생성하는 슈퍼 사용자 또는 관리자는 Athena 연합 쿼리를 허용할 IAM 정책도 보유해야 합니다.

카탈로그 등록 및 메타데이터 API 작업

카탈로그 등록 API 및 메타데이터 API 작업에 액세스하려면 AmazonAthenaFullAccess 관리형 정책을 사용합니다. 이 정책을 사용하지 않는 경우에는 다음 API 작업을 Athena 정책에 추가하세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListDataCatalogs",
                "athena:GetDataCatalog",
                "athena:CreateDataCatalog",
                "athena:UpdateDataCatalog",
                "athena:DeleteDataCatalog",
                "athena:GetDatabase",
                "athena:ListDatabases",
                "athena:GetTableMetadata",
                "athena:ListTableMetadata"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

리전 간 Lambda 호출

Athena 쿼리를 실행 중인 리전 이외의 리전에서 Lambda 함수를 호출하려면 Lambda 함수의 전체 ARN을 사용합니다. 기본적으로 Athena는 동일한 리전에 정의된 Lambda 함수를 호출합니다. Athena 쿼리를 실행하는 리전이 아닌 다른 리전의 Hive 메타스토어에 액세스하기 위해 Lambda 함수를 호출해야 하는 경우 Lambda 함수의 전체 ARN을 제공해야 합니다.

예를 들어 미국 동부(버지니아 북부) 리전에서 다음 Lambda 함수를 사용하도록 유럽(프랑크푸르트) 리전 `eu-central-1`에 카탈로그 `ehms`를 정의한다고 가정합니다.

```
arn:aws:lambda:us-east-1:111122223333:function:external-hms-service-new
```

이러한 방식으로 전체 ARN을 지정하면 Athena는 `us-east-1`의 `external-hms-service-new` Lambda 함수를 호출하여 `eu-central-1`에서 Hive 메타스토어 데이터를 가져올 수 있습니다.

> **Note**
>
> 카탈로그 `ehms`는 Athena 쿼리를 실행하는 동일한 리전에 등록되어야 합니다.

계정 간 Lambda 호출

경우에 따라 다른 계정에서 Hive 메타스토어에 액세스해야 할 수도 있습니다. 예를 들어 Hive 메타스토어를 실행하려면 Athena 쿼리에 사용하는 계정과 다른 계정에서 EMR 클러스터를 시작할 수도 있습니다. 여러 그룹 또는 팀이 VPC 내에서 여러 계정으로 Hive 메타스토어를 실행할 수도 있습니다. 또는 여러 그룹 또는 팀에서 여러 Hive 메타스토어의 메타데이터에 액세스해야 할 수도 있습니다.

Athena는 계정 간 액세스에 대한 AWS Lambda 지원을 사용하여 Hive 메타스토어에 대한 계정 간 액세스를 활성화합니다.

> **Note**
>
> Athena에 대한 계정 간 액세스는 일반적으로 Amazon S3의 메타데이터 및 데이터 모두에 대한 계정 간 액세스를 의미합니다.

다음 시나리오를 예로 들겠습니다.

- 계정 `111122223333`은 EMR 클러스터에서 실행 중인 Hive 메타스토어에 액세스하기 위해 Athena의 us-east-1에 Lambda 함수 `external-hms-service-new`를 설정합니다.

- 계정 111122223333에서 계정 444455556666이 Hive 메타스토어 데이터에 액세스하도록 허용할 수 있습니다.

Lambda 함수 external-hms-service-new에 대한 액세스 권한을 계정 444455556666에 부여하기 위해 계정 111122223333은 다음 AWS CLI add-permission 명령을 사용합니다. 명령은 가독성을 위해 형식이 지정되었습니다.

```
$ aws --profile perf-test lambda add-permission
      --function-name external-hms-service-new
      --region us-east-1
      --statement-id Id-ehms-invocation2
      --action "lambda:InvokeFunction"
      --principal arn:aws:iam::444455556666:user/perf1-test
{
    "Statement": "{\"Sid\":\"Id-ehms-invocation2\",
                  \"Effect\":\"Allow\",
                  \"Principal\":{\"AWS\":\"arn:aws:iam::444455556666:user/perf1-test
\"},
                  \"Action\":\"lambda:InvokeFunction\",
                  \"Resource\":\"arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-new\"}"
}
```

Lambda 권한을 확인하려면 다음 예제와 같이 get-policy 명령을 사용합니다. 명령은 가독성을 위해 형식이 지정되었습니다.

```
$ aws --profile perf-test lambda get-policy
      --function-name arn:aws:lambda:us-east-1:111122223333:function:external-hms-
service-new
      --region us-east-1
{
    "RevisionId": "711e93ea-9851-44c8-a09f-5f2a2829d40f",
    "Policy": "{\"Version\":\"2012-10-17\",
                \"Id\":\"default\",
                \"Statement\":[{\"Sid\":\"Id-ehms-invocation2\",
                               \"Effect\":\"Allow\",
                               \"Principal\":{\"AWS\":
\"arn:aws:iam::444455556666:user/perf1-test\"},
                               \"Action\":\"lambda:InvokeFunction\",
                               \"Resource\":\"arn:aws:lambda:us-
east-1:111122223333:function:external-hms-service-new\"}]}"
```

```
}
```

권한을 추가한 후, 카탈로그 ehms를 정의할 때 다음과 같이 us-east-1에서 Lambda 함수의 전체 ARN을 사용할 수 있습니다.

```
arn:aws:lambda:us-east-1:111122223333:function:external-hms-service-new
```

교차 리전 호출에 대한 자세한 내용은 이 주제의 앞부분에 있는 리전 간 Lambda 호출을 참조하세요.

데이터에 대한 교차 계정 액세스 권한 부여

Athena 쿼리를 실행하려면 먼저 Amazon S3의 데이터에 대한 계정 간 액세스 권한을 부여해야 합니다. 이 작업을 다음 중 한 가지 방법으로 수행할 수 있습니다.

- Amazon S3 버킷의 액세스 제어 목록 정책을 정식 사용자 ID로 업데이트합니다.
- 계정 간 액세스를 Amazon S3 버킷 정책에 추가합니다.

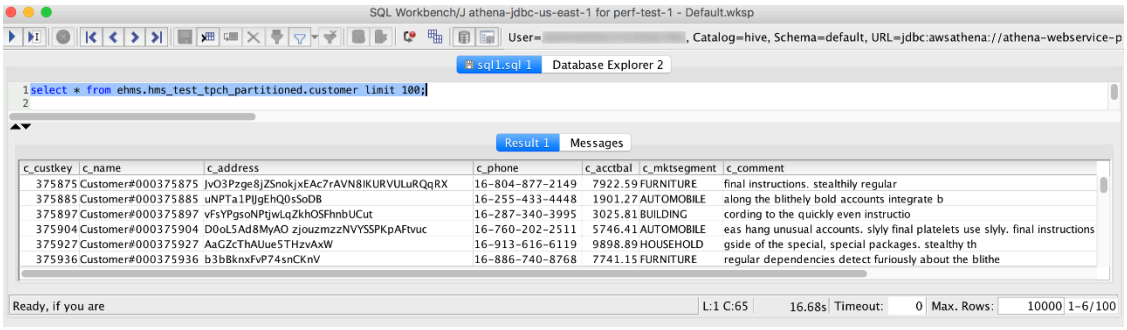
예를 들어 다음 정책을 계정 111122223333의 Amazon S3 버킷 정책에 추가하여 계정 444455556666이 지정된 Amazon S3 위치에서 데이터를 읽을 수 있도록 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1234567890123",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:user/perf1-test"
            },
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::athena-test/lambda/dataset/*"
        }
    ]
}
```

> **Note**
>
> 데이터뿐만 아니라 Amazon S3 유출 위치에도 Amazon S3에 대한 계정 간 액세스 권한을 부여해야 할 수도 있습니다. 응답 객체의 크기가 지정된 임계값을 초과하면 Lambda 함수는 여분의 데이터를 유출 위치로 분산합니다. 샘플 정책에 대한 이 주제의 시작 부분을 참조하세요.

현재 예제에서, 교차 계정 액세스가 `444455556666,`에 부여된 후 `444455556666`은 자체 `account`의 카탈로그 `ehms`를 사용하여 계정 `111122223333`에 정의된 테이블을 쿼리할 수 있습니다.

다음 예제에서 SQL Workbench 프로필 `perf-test-1`은 계정 `444455556666`에 대한 것입니다. 쿼리는 카탈로그 `ehms`를 사용하여 계정 `111122223333`의 Amazon S3 데이터 및 Hive 메타스토어에 액세스합니다.



Athena 연합 쿼리를 허용하는 IAM 권한 정책의 예제

이 주제의 권한 정책 예제에서는 필요한 허용된 작업과 해당 작업이 허용되는 리소스를 보여줍니다. IAM 자격 증명에 연결하기 전에 이러한 정책을 신중하게 검토하고 요구 사항에 따라 수정합니다.

IAM 자격 증명에 정책을 연결하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 IAM 자격 증명 권한 추가 및 제거를 참조하세요.

- Example policy to allow an IAM principal to run and return results using Athena Federated Query
- Example Policy to Allow an IAM Principal to Create a Data Source Connector

Example - IAM 보안 주체가 Athena 연합 쿼리를 사용하여 쿼리를 실행하고 결과를 반환할 수 있도록 허용합니다.

다음 자격 증명 기반 권한 정책은 사용자 또는 다른 IAM 보안 주체가 Athena 연합 쿼리를 사용하는 데 필요한 작업을 허용합니다. 이러한 작업을 수행할 수 있는 보안 주체는 연합 데이터 원본과 연결된 Athena 카탈로그를 지정하는 쿼리를 실행할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Athena",
            "Effect": "Allow",
            "Action": [
                "athena:GetDataCatalog",
                "athena:GetQueryExecution",
                "athena:GetQueryResults",
                "athena:GetWorkGroup",
                "athena:StartQueryExecution",
                "athena:StopQueryExecution"
            ],
            "Resource": [
                "arn:aws:athena:*:111122223333:workgroup/WorkgroupName",
                "arn:aws:athena:aws_region:111122223333:datacatalog/DataCatalogName"
            ]
        },
        {
            "Sid": "ListAthenaWorkGroups",
            "Effect": "Allow",
            "Action": "athena:ListWorkGroups",
            "Resource": "*"
        },
        {
            "Sid": "Lambda",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": [
                "arn:aws:lambda:*:111122223333:function:OneAthenaLambdaFunction",
                "arn:aws:lambda:*:111122223333:function:AnotherAthenaLambdaFunction"
            ]
        },
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:GetBucketLocation",
                "s3:GetObject",
                "s3:ListBucket",
                "s3:ListMultipartUploadParts",
```

```
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::MyLambdaSpillBucket",
                "arn:aws:s3:::MyLambdaSpillBucket/*",
                "arn:aws:s3:::MyQueryResultsBucket",
                "arn:aws:s3:::MyQueryResultsBucket/*"
            ]
        }
    ]
}
```

권한에 대한 설명

허용된 작업	설명
`"athena:GetQueryExecution", "athena:GetQueryResults", "athena:GetWorkGroup", "athena:StartQueryExecution", "athena:StopQueryExecution"`	연합 쿼리를 실행하는 데 필요한 Athena 권한입니다.
`"athena:GetDataCatalog", "athena:GetQueryExecution," "athena:GetQueryResults", "athena:GetWorkGroup", "athena:StartQueryExecution", "athena:StopQueryExecution"`	페더레이션된 보기 쿼리를 실행하는 데 필요한 Athena 권한입니다. 뷰에는 `GetDataCatalog` 작업이 필요합니다.
`"lambda:InvokeFunction"`	쿼리가 `Resource` 블록에 지정된 AWS Lambda 함수에 대한 AWS Lambda 함수를 호출할 수 있습니다. `arn:aws:lambda:*:` *MyAWSAcctId* `:function:` *MyAthenaLambdaFunction* 을 예로 들 수 있습니다. 여기서 *MyAthenaLambdaFunction* 은 호출되는 Lambda 함수의 이름을 지정합니다. 예제와 같이 여러 함수를 지정할 수 있습니다.
`"s3:AbortMultipartUpload", "s3:GetBucketLocation",`	`StartQueryExecution` 을 실행하는 IAM 보안 주체에 대한 쿼리 출력 버킷에 액세스

허용된 작업	설명
`"s3:GetObject", "s3:ListBucket", "s3:ListMultipartUploadParts", "s3:PutObject"`	하려면 `s3:ListBucket` 및 `s3:GetBucketLocation` 권한이 필요합니다. `s3:PutObject` , `s3:ListMultipartUploadParts` 및 `s3:AbortMultipartUpload` 는 `arn:aws:s3::: MyQueryResultsBucket /*` 리소스 식별자에 지정된 대로 쿼리 결과 버킷의 모든 하위 폴더에 쿼리 결과를 작성하도록 허용합니다. 여기서 *MyQueryResultsBucket* 은 Athena 쿼리 결과 버킷입니다. 자세한 내용은 쿼리 결과, 최근 쿼리, 출력 파일 작업 단원을 참조하십시오. `s3:GetObject` 는 `arn:aws:s3::: MyQueryResultsBucket` 으로 지정된 리소스에 대한 쿼리 결과 및 쿼리 기록을 읽을 수 있도록 허용합니다. 여기서 *MyQueryResultsBucket* 은 Athena 쿼리 결과 버킷입니다. 또한 `s3:GetObject` 는 `"arn:aws:s3::: MyLambdaSpillBucket /MyLambdaSpillPrefix *"` 로 지정된 리소스에서 읽을 수 있도록 허용합니다. 여기서 *MyLambdaSpillPrefix* 는 호출되는 Lambda 함수의 구성에서 지정됩니다.

Example - IAM 보안 주체가 데이터 원본 커넥터를 생성하도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
```

```
                "lambda:CreateFunction",
                "lambda:ListVersionsByFunction",
                "iam:CreateRole",
                "lambda:GetFunctionConfiguration",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy",
                "lambda:PutFunctionConcurrency",
                "iam:PassRole",
                "iam:DetachRolePolicy",
                "lambda:ListTags",
                "iam:ListAttachedRolePolicies",
                "iam:DeleteRolePolicy",
                "lambda:DeleteFunction",
                "lambda:GetAlias",
                "iam:ListRolePolicies",
                "iam:GetRole",
                "iam:GetPolicy",
                "lambda:InvokeFunction",
                "lambda:GetFunction",
                "lambda:ListAliases",
                "lambda:UpdateFunctionConfiguration",
                "iam:DeleteRole",
                "lambda:UpdateFunctionCode",
                "s3:GetObject",
                "lambda:AddPermission",
                "iam:UpdateRole",
                "lambda:DeleteFunctionConcurrency",
                "lambda:RemovePermission",
                "iam:GetRolePolicy",
                "lambda:GetPolicy"
            ],
            "Resource": [

"arn:aws:lambda:*:111122223333:function:MyAthenaLambdaFunctionsPrefix*",
                "arn:aws:s3:::awsserverlessrepo-changesets-1iiv3xa62ln3m/*",
                "arn:aws:iam::*:role/RoleName",
                "arn:aws:iam::111122223333:policy/*"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateUploadBucket",
```

```
                "cloudformation:DescribeStackDriftDetectionStatus",
                "cloudformation:ListExports",
                "cloudformation:ListStacks",
                "cloudformation:ListImports",
                "lambda:ListFunctions",
                "iam:ListRoles",
                "lambda:GetAccountSettings",
                "ec2:DescribeSecurityGroups",
                "cloudformation:EstimateTemplateCost",
                "ec2:DescribeVpcs",
                "lambda:ListEventSourceMappings",
                "cloudformation:DescribeAccountLimits",
                "ec2:DescribeSubnets",
                "cloudformation:CreateStackSet",
                "cloudformation:ValidateTemplate"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": "cloudformation:*",
            "Resource": [
                "arn:aws:cloudformation:*:111122223333:stack/aws-serverless-
repository-MyCFStackPrefix*/*",
                "arn:aws:cloudformation:*:111122223333:stack/
serverlessrepo-MyCFStackPrefix*/*",
                "arn:aws:cloudformation:*:*:transform/Serverless-*",
                "arn:aws:cloudformation:*:111122223333:stackset/aws-serverless-
repository-MyCFStackPrefix*:*",
                "arn:aws:cloudformation:*:111122223333:stackset/
serverlessrepo-MyCFStackPrefix*:*"
            ]
        },
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": "serverlessrepo:*",
            "Resource": "arn:aws:serverlessrepo:*:*:applications/*"
        }
    ]
}
```

권한에 대한 설명

허용된 작업	설명
`"lambda:CreateFunction", "lambda:ListVersionsByFunction", "lambda:GetFunctionConfiguration", "lambda:PutFunctionConcurrency", "lambda:ListTags", "lambda:DeleteFunction", "lambda:GetAlias", "lambda:InvokeFunction", "lambda:GetFunction", "lambda:ListAliases", "lambda:UpdateFunctionConfiguration", "lambda:UpdateFunctionCode", "lambda:AddPermission", "lambda:DeleteFunctionConcurrency", "lambda:RemovePermission", "lambda:GetPolicy" "lambda:GetAccountSettings", "lambda:ListFunctions", "lambda:ListEventSourceMappings",`	리소스로 나열된 Lambda 함수의 생성 및 관리를 허용합니다. 이 예제에서는 이름 접두사가 리소스 식별자 `arn:aws:lambda:*:` *MyAWSAcctId* `:function:` *MyAthenaLambdaFunctionsPrefix* `*`에 사용됩니다. 여기서 *MyAthenaLambdaFunctionsPrefix* 는 Lambda 함수 그룹의 이름에 사용되는 공유 접두사이므로 리소스로 개별적으로 지정할 필요가 없습니다. 하나 이상의 Lambda 함수 리소스를 지정할 수 있습니다.
`"s3:GetObject"`	리소스 식별자 `arn:aws:s3:::awsserverlessrepo-changesets-`*1iiv3xa62ln3m*`/*`에 지정된 대로 AWS Serverless Application Repository이 요구하는 버킷을 읽을 수 있습니다. 이 버킷은 사용자 계정에만 해당될 수 있습니다.
`"cloudformation:*"`	*MyCFStackPrefix* 리소스에 의해 지정된 AWS CloudFormation 스택의 생성 및 관리를 허용합니다. 이러한 스택 및 스택 세트는 AWS Serverless Application Repository이 커넥터와 UDF를 배포하는 방법입니다.
`"serverlessrepo:*"`	AWS Serverless Application Repository에서 리소스 식별자 `arn:aws:serverless`

허용된 작업	설명
	`repo:*:*:applications/*` 로 지정된 애플리케이션을 검색하고 보고 게시 및 업데이트할 수 있습니다.

Amazon Athena 사용자 정의 함수(UDF)를 허용하는 IAM 권한 정책의 예

이 주제의 권한 정책 예제에서는 필요한 허용된 작업과 해당 작업이 허용되는 리소스를 보여줍니다. IAM 자격 증명에 유사한 권한 정책을 연결하기 전에 이러한 정책을 신중하게 검토하고 요구 사항에 따라 수정합니다.

- Example Policy to Allow an IAM Principal to Run and Return Queries that Contain an Athena UDF Statement
- Example Policy to Allow an IAM Principal to Create an Athena UDF

Example - IAM 보안 주체가 Athena UDF 문을 포함한 쿼리를 실행하고 결과를 반환하도록 허용합니다.

다음 자격 증명 기반 권한 정책은 사용자 또는 다른 IAM 보안 주체가 Athena UDF 문을 사용하는 쿼리를 실행하는 데 필요한 작업을 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "athena:StartQueryExecution",
                "lambda:InvokeFunction",
                "athena:GetQueryResults",
                "s3:ListMultipartUploadParts",
                "athena:GetWorkGroup",
                "s3:PutObject",
                "s3:GetObject",
                "s3:AbortMultipartUpload",
                "athena:StopQueryExecution",
                "athena:GetQueryExecution",
                "s3:GetBucketLocation"
```

```
            ],
            "Resource": [
                "arn:aws:athena:*:MyAWSAcctId:workgroup/MyAthenaWorkGroup",
                "arn:aws:s3:::MyQueryResultsBucket/*",
                "arn:aws:lambda:*:MyAWSAcctId:function:OneAthenaLambdaFunction",
                "arn:aws:lambda:*:MyAWSAcctId:function:AnotherAthenaLambdaFunction"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "athena:ListWorkGroups",
            "Resource": "*"
        }
    ]
}
```

권한에 대한 설명

허용된 작업	설명
`"athena:StartQueryExecution", "athena:GetQueryResults", "athena:GetWorkGroup", "athena:StopQueryExecution", "athena:GetQueryExecution",`	MyAthenaWorkGroup 작업 그룹에서 쿼리를 실행하는 데 필요한 Athena 권한입니다.
`"s3:PutObject", "s3:GetObject", "s3:AbortMultipartUpload"`	s3:PutObject 및 s3:AbortMultipartUpload 는 arn:aws:s3::: *MyQueryResultsBucket* /* 리소스 식별자에 지정된 대로 쿼리 결과 버킷의 모든 하위 폴더에 쿼리 결과를 작성하도록 허용합니다. 여기서 *MyQueryResultsBucket* 은 Athena 쿼리 결과 버킷입니다. 자세한 내용은 쿼리 결과, 최근 쿼리, 출력 파일 작업 단원을 참조하세요. s3:GetObject 는 arn:aws:s3::: *MyQueryResultsBucket* 으로 지정된 리소스에 대한 쿼리 결과 및 쿼리 기록을 읽을 수 있도록 허용합니다. 여기서 *MyQueryRe*

허용된 작업	설명
	sultsBucket 은 Athena 쿼리 결과 버킷입니다. 자세한 내용은 쿼리 결과, 최근 쿼리, 출력 파일 작업 단원을 참조하세요. 또한 s3:GetObject 는 "arn:aws:s3::: *MyLambdaSpillBucket* /*MyLambdaSpillPrefix* *"로 지정된 리소스에서 읽을 수 있도록 허용합니다. 여기서 *MyLambdaSpillPrefix* 는 호출되는 Lambda 함수의 구성에서 지정됩니다.
"lambda:InvokeFunction"	쿼리가 Resource 블록에 지정된 AWS Lambda 함수를 호출할 수 있습니다. arn:aws:lambda:*: *MyAWSAcctId* :function: *MyAthenaLambdaFunction* 을 예로 들 수 있습니다. 여기서 *MyAthenaLambdaFunction* 은 호출할 Lambda 함수의 이름을 지정합니다. 예제와 같이 여러 함수를 지정할 수 있습니다.

Example - IAM 보안 주체가 Athena UDF를 생성할 수 있도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "lambda:CreateFunction",
                "lambda:ListVersionsByFunction",
                "iam:CreateRole",
                "lambda:GetFunctionConfiguration",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy",
                "lambda:PutFunctionConcurrency",
                "iam:PassRole",
```

```
                "iam:DetachRolePolicy",
                "lambda:ListTags",
                "iam:ListAttachedRolePolicies",
                "iam:DeleteRolePolicy",
                "lambda:DeleteFunction",
                "lambda:GetAlias",
                "iam:ListRolePolicies",
                "iam:GetRole",
                "iam:GetPolicy",
                "lambda:InvokeFunction",
                "lambda:GetFunction",
                "lambda:ListAliases",
                "lambda:UpdateFunctionConfiguration",
                "iam:DeleteRole",
                "lambda:UpdateFunctionCode",
                "s3:GetObject",
                "lambda:AddPermission",
                "iam:UpdateRole",
                "lambda:DeleteFunctionConcurrency",
                "lambda:RemovePermission",
                "iam:GetRolePolicy",
                "lambda:GetPolicy"
            ],
            "Resource": [

 "arn:aws:lambda:*:111122223333:function:MyAthenaLambdaFunctionsPrefix*",
                "arn:aws:s3:::awsserverlessrepo-changesets-1iiv3xa62ln3m/*",
                "arn:aws:iam::*:role/RoleName",
                "arn:aws:iam::111122223333:policy/*"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateUploadBucket",
                "cloudformation:DescribeStackDriftDetectionStatus",
                "cloudformation:ListExports",
                "cloudformation:ListStacks",
                "cloudformation:ListImports",
                "lambda:ListFunctions",
                "iam:ListRoles",
                "lambda:GetAccountSettings",
                "ec2:DescribeSecurityGroups",
```

```
                "cloudformation:EstimateTemplateCost",
                "ec2:DescribeVpcs",
                "lambda:ListEventSourceMappings",
                "cloudformation:DescribeAccountLimits",
                "ec2:DescribeSubnets",
                "cloudformation:CreateStackSet",
                "cloudformation:ValidateTemplate"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": "cloudformation:*",
            "Resource": [
                "arn:aws:cloudformation:*:111122223333:stack/aws-serverless-
repository-MyCFStackPrefix*/*",
                "arn:aws:cloudformation:*:111122223333:stack/
serverlessrepo-MyCFStackPrefix*/*",
                "arn:aws:cloudformation:*:*:transform/Serverless-*",
                "arn:aws:cloudformation:*:111122223333:stackset/aws-serverless-
repository-MyCFStackPrefix*:*",
                "arn:aws:cloudformation:*:111122223333:stackset/
serverlessrepo-MyCFStackPrefix*:*"
            ]
        },
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": "serverlessrepo:*",
            "Resource": "arn:aws:serverlessrepo:*:*:applications/*"
        }
    ]
}
```

권한에 대한 설명

허용된 작업	설명
`"lambda:CreateFunction", "lambda:ListVersionsByFunction", "lambda:GetFunctionConfiguration", "lambda:PutFunctionConcurrency",`	리소스로 나열된 Lambda 함수의 생성 및 관리를 허용합니다. 이 예제에서는 이름 접두사가 리소스 식별자 `arn:aws:lambda:*:` *MyAWSAcctId* `:function`

허용된 작업	설명
"lambda:ListTags", "lambda:DeleteFunction", "lambda:GetAlias", "lambda:InvokeFunction", "lambda:GetFunction", "lambda:ListAliases", "lambda:UpdateFunctionConfiguration", "lambda:UpdateFunctionCode", "lambda:AddPermission", "lambda:DeleteFunctionConcurrency", "lambda:RemovePermission", "lambda:GetPolicy" "lambda:GetAccountSettings", "lambda:ListFunctions", "lambda:ListEventSourceMappings",	: *MyAthenaLambdaFunctionsPrefix* *에 사용됩니다. 여기서 *MyAthenaLambdaFunctionsPrefix* 는 Lambda 함수 그룹의 이름에 사용되는 공유 접두사이므로 리소스로 개별적으로 지정할 필요가 없습니다. 하나 이상의 Lambda 함수 리소스를 지정할 수 있습니다.
"s3:GetObject"	리소스 식별자 arn:aws:s3:::awsserverlessrepo-changesets- *1iiv3xa62ln3m* /*에 지정된 대로 AWS Serverless Application Repository이 요구하는 버킷을 읽을 수 있습니다.
"cloudformation:*"	*MyCFStackPrefix* 리소스에 의해 지정된 AWS CloudFormation 스택의 생성 및 관리를 허용합니다. 이러한 스택 및 스택 세트는 AWS Serverless Application Repository이 커넥터와 UDF를 배포하는 방법입니다.
"serverlessrepo:*"	AWS Serverless Application Repository에서 리소스 식별자 arn:aws:serverlessrepo:*:*:applications/* 로 지정된 애플리케이션을 검색하고 보고 게시 및 업데이트할 수 있습니다.

ML과 Athena에 대한 액세스 허용

Athena ML 쿼리를 실행하는 IAM 보안 주체는 자신이 사용하는 Sagemaker 엔드포인트 대한 `sagemaker:invokeEndpoint` 작업을 수행할 수 있어야 합니다. 사용자 자격 증명에 연결된 ID 기반 권한 정책에 다음과 유사한 정책 문을 포함합니다. 또한 Athena 작업에 대한 전체 액세스 권한을 부여하는 AWS 관리형 정책: AmazonAthenaFullAccess, 또는 작업의 하위 집합을 허용하는 수정된 인라인 정책을 연결합니다.

예제에서 `arn:aws:sagemaker:`*`region`*`:`*`AWSAcctID`*`:`*`ModelEndpoint`*를 쿼리에 사용할 모델 엔드포인트의 ARN 또는 ARN으로 바꿉니다. 자세한 내용은 서비스 권한 부여 참조에서 SageMaker에 사용되는 작업, 리소스 및 조건 키를 참조하세요.

```
{
            "Effect": "Allow",
            "Action": [
                "sagemaker:invokeEndpoint"
            ],
            "Resource": "arn:aws:sagemaker:us-west-2:123456789012:workteam/public-
crowd/default"
}
```

IAM 정책을 사용할 때는 항상 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

Athena API에 대한 연합 액세스 활성화

이 단원에서는 조직의 사용자 또는 클라이언트 애플리케이션이 Amazon Athena API 작업을 호출하는 데 사용할 수 있는 연합 액세스에 대해 설명합니다. 이 경우 조직의 사용자는 Athena에 직접 액세스할 수 없습니다. 그 대신, Microsoft Active Directory에서 AWS 외부의 사용자 자격 증명을 관리합니다. Active Directory는 SAML 2.0(Security Assertion Markup Language 2.0)를 지원합니다.

이 시나리오에서 사용자를 인증하려면 Active Directory Federation Services(ADFS) 3.0에 액세스하고 Athena API 작업 호출을 위해 클라이언트 애플리케이션을 활성화하는 SAML.2.0 지원이 포함된 JDBC 또는 ODBC 드라이버를 사용합니다.

AWS에서 SAML 2.0 지원에 대한 자세한 내용은 IAM 사용 설명서에서 SAML 2.0 연합에 대하여를 참조하세요.

> **Note**
>
> Athena API에 대한 연합 액세스는 특정 유형의 자격 증명 공급자(IdP)인 Active Directory Federation Service(AD FS 3.0)(Windows Server의 일부)에 대해 지원됩니다. 연동 액세스는 IAM Identity Center의 신뢰할 수 있는 ID 전파 기능과 호환되지 않습니다. 액세스는 SAML 2.0을 지원하는 JDBC 또는 ODBC 드라이버 버전을 통해 설정됩니다. 자세한 내용은 JDBC로 Amazon Athena에 연결 및 ODBC로 Amazon Athena에 연결 단원을 참조하세요.

주제

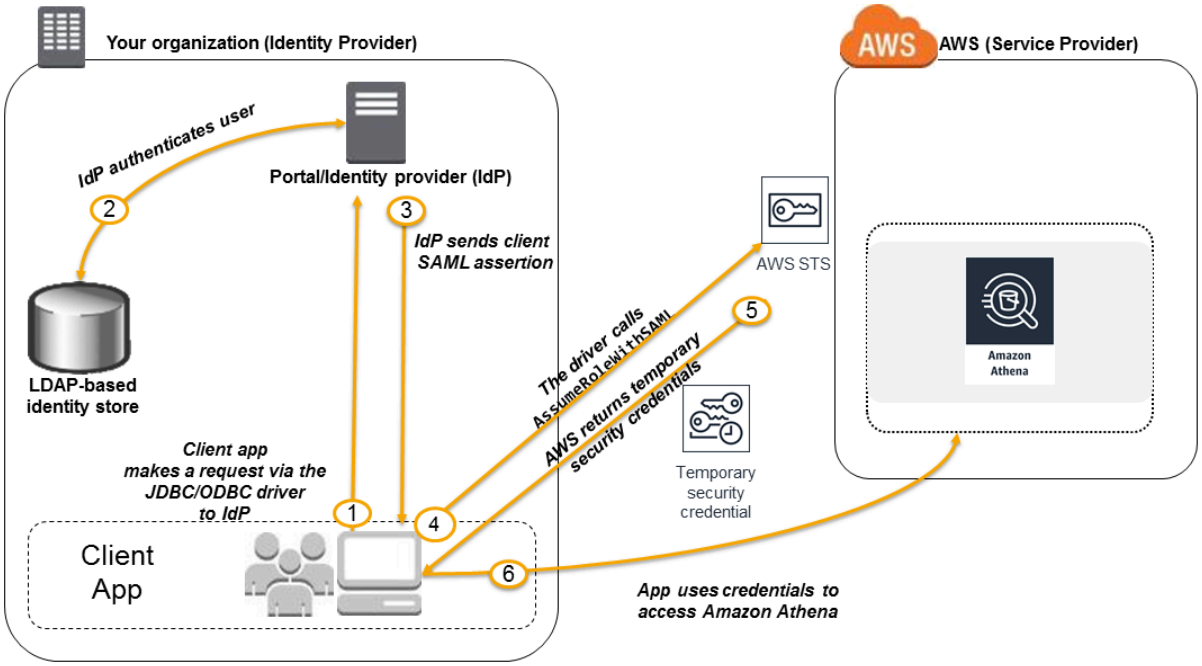
시작하기 전 준비 사항

시작하기 전에 다음 필수 조건을 완료합니다.

- 조직 내부에서 AD FS 3.0을 설치하고 IdP로 구성합니다.
- Athena 액세스에 사용되는 클라이언트에 JDBC 또는 ODBC 드라이버의 최신 버전을 설치하고 구성합니다. 드라이버는 SAML 2.0과 호환되는 연합 액세스에 대한 지원을 포함해야 합니다. 자세한 내용은 JDBC로 Amazon Athena에 연결 및 ODBC로 Amazon Athena에 연결 단원을 참조하세요.

아키텍처 다이어그램

다음 다이어그램에서 이 프로세스를 보여 줍니다.



1. 조직 내 사용자는 클라이언트 애플리케이션을 JDBC 또는 ODBC 드라이버와 함께 사용하여 조직의 IdP에게 인증을 요청합니다. IdP는 AD FS 3.0입니다.
2. IdP는 조직의 자격 증명 스토어인 Active Directory에 대해 사용자를 인증합니다.
3. IdP는 사용자에 대한 정보로 SAML 어설션을 구성하여 JDBC 또는 ODBC 드라이버를 통해 클라이언트 애플리케이션에 전송합니다.
4. JDBC 또는 ODBC 드라이버는 AWS Security Token Service AssumeRoleWithSAML API 작업을 호출하여 이 작업에 다음 파라미터를 전달합니다.
   - SAML 공급자의 ARN
   - 수임할 역할의 ARN
   - IdP의 SAML 어설션

   자세한 내용은 AWS Security Token Service API 참조의 AssumeRoleWithSAML을 참조하세요.
5. JDBC 또는 ODBC 드라이버를 통해 클라이언트 애플리케이션으로 보내는 API 응답에는 임시 보안 자격 증명이 포함됩니다.
6. 클라이언트 애플리케이션은 사용자가 Athena API 작업에 액세스할 수 있도록 임시 보안 자격 증명을 사용하여 Athena API 작업을 호출합니다.

절차: Athena API에 대한 SAML 기반 연합 액세스

이 절차에서는 조직의 IdP와 AWS 계정 간에 신뢰를 확립하여 Amazon Athena API 작업에 대한 SAML 기반 연합 액세스를 활성화합니다.

Athena API에 대한 연합 액세스를 활성화하려면:

1. 조직에서 AWS를 IdP의 서비스 공급자(SP)로 등록합니다. 이 프로세스를 신뢰 당사자 신뢰라고 합니다. 자세한 내용은 IAM 사용 설명서의 신뢰 당사자 신뢰로 SAML 2.0 IdP 구성을 참조하세요. 이 작업의 일부로 다음 단계를 수행합니다.

   a. 다음 URL에서 샘플 SAML 메타데이터 문서를 가져옵니다. https://signin.aws.amazon.com/static/saml-metadata.xml.

   b. 조직의 IdP(ADFS)에서 IdP를 AWS에 대한 자격 증명 공급자로 설명하는 동등한 메타데이터 XML 파일을 생성합니다. 메타데이터 파일에는 발행자 이름, 생성 날짜, 만료 날짜 및 AWS가 조직에서 오는 인증 응답의 유효성을 검증하는 데 사용할 수 있는 키가 포함되어야 합니다.

2. IAM 콘솔에서 SAML 자격 증명 공급자 개체를 생성합니다. 자세한 내용은 IAM 사용 설명서의 SAML 자격 증명 공급자 생성을 참조하세요. 이 단계의 일부로 다음을 포함합니다.

   a. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.

   b. 이 절차에서 1단계의 IdP(ADFS)가 생성한 SAML 메타데이터 문서를 업로드합니다.

3. IAM 콘솔에서 IdP에 대해 하나 이상의 IAM 역할을 생성합니다. 자세한 내용은 IAM 사용 설명서의 서드 파티 자격 증명 공급자의 역할 만들기(연합)을 참조하세요. 이 단계의 일부로 다음을 포함합니다.

   - 역할의 권한 정책에서 조직의 사용자가 AWS에서 수행할 수 있는 작업을 나열합니다.
   - 역할의 신뢰 정책에서 이 절차의 2단계에서 생성한 SAML 공급자 엔터티를 보안 주체로 설정합니다.

   이렇게 하면 조직과 AWS 사이에 신뢰 관계가 설정됩니다.

4. 조직의 IdP(ADFS)에서 조직 내 사용자 또는 그룹을 IAM 역할에 매핑하는 어설션을 정의합니다. IAM 역할에 대한 사용자와 그룹의 매핑을 클레임 규칙(claim rule)이라고 합니다. 조직의 다양한 사용자 및 그룹은 서로 다른 IAM 역할에 매핑될 수 있다는 점에 유의하세요.

   ADFS에서 매핑을 구성하는 방법에 대한 자세한 내용은 Windows Active Directory, ADFS 및 SAML 2.0을 사용하여 AWS에 대한 연합 활성화 블로그 게시물을 참조하세요.

5. SAML 2.0 지원이 포함된 JDBC 또는 ODBC 드라이버를 설치하고 구성합니다. 자세한 내용은 JDBC로 Amazon Athena에 연결 및 ODBC로 Amazon Athena에 연결 단원을 참조하세요.
6. 애플리케이션에서 JDBC 또는 ODBC 드라이버로 연결을 지정합니다. 애플리케이션에서 사용해야 할 연결 문자열에 대한 자세한 내용은 JDBC Driver Installation and Configuration Guide(JDBC 드라이버 설치 및 구성 가이드)의 "Using the Active Directory Federation Services (ADFS) Credentials Provider"(Active Directory Federation Services(ADFS) 자격 증명 공급자 사용) 또는 ODBC Driver Installation and Configuration Guide(ODBC 드라이버 설치 및 구성 가이드)의 비슷한 주제를 참조하세요. JDBC로 Amazon Athena에 연결 및 ODBC로 Amazon Athena에 연결 주제에서 PDF 다운로드가 가능합니다.

   다음은 드라이버에 연결 문자열을 구성하는 방법에 대한 개괄적인 요약입니다.

   1. `AwsCredentialsProviderClass configuration`에서 `com.simba.athena.iamsupport.plugin.AdfsCredentialsProvider`를 설정하여 ADFS IdP를 통해 SAML 2.0 기반 인증을 사용합니다.
   2. `idp_host`에 ADFS IdP 서버의 호스트 이름을 입력합니다.
   3. `idp_port`에 ADFS IdP가 SAML 어설션 요청에 대해 수신 대기하는 포트 번호를 입력합니다.
   4. `UID` 및 `PWD`에 AD 도메인 사용자 자격 증명을 제공합니다. Windows에서 드라이버를 사용하는 경우 `UID` 및 `PWD`를 입력하지 않으면 드라이버는 Windows 시스템에 로그인한 사용자의 사용자 자격 증명을 가져오려고 합니다.
   5. 선택적으로 `ssl_insecure`를 `true`로 설정합니다. 이 경우 드라이버는 ADFS IdP 서버에 대한 SSL 인증서의 신뢰성을 확인하지 않습니다. 드라이버에서 신뢰할 수 있도록 ADFS IdP의 SSL 인증서를 구성하지 않은 경우 `true`로 설정해야 합니다.
   6. 하나 이상의 IAM 역할에 대한 Active Directory 도메인 사용자 또는 그룹의 매핑을 활성화하려면(이 절차의 4단계에서 언급함) JDBC 또는 ODBC 연결에 대한 `preferred_role`에서 드라이버 연결을 수임할 IAM 역할(ARN)을 지정합니다. `preferred_role` 지정은 선택 사항이며 역할이 클레임 규칙에 나열된 첫 번째 역할이 아닌 경우에 유용합니다.

   이 절차를 수행하면 다음 작업이 수행됩니다.

   1. JDBC 또는 ODBC 드라이버가 AWS STS AssumeRoleWithSAML API를 호출하고 아키텍처 다이어그램의 4단계와 같이 이 API에 어설션을 전달합니다.
   2. AWS는 역할 수임 요청이 SAML 공급자 엔터티에서 참조된 IdP로부터 오는지 확인합니다.

3. 요청이 성공하면 AWS STS AssumeRoleWithSAML API 작업은 일련의 임시 보안 자격 증명을 반환하고 클라이언트 애플리케이션은 이 자격 증명을 사용하여 Athena에 대한 서명된 요청을 생성합니다.

   이제 애플리케이션은 현재 사용자에 대한 정보를 갖추었으며 프로그래밍 방식으로 Athena에 액세스할 수 있습니다.

Athena의 로깅 및 모니터링

인시던트를 탐지하고 인시던트가 발생했을 때 경고를 받고 그에 응답하려면 Amazon Athena 옵션과 함께 다음 옵션을 사용하세요.

- AWS CloudTrail을 사용하여 Athena 모니터링 - AWS CloudTrail은 Athena의 사용자, 역할 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공합니다. 이 서비스는 Athena 콘솔에서 수행한 호출과 Athena API 작업에 대한 코드 호출을 캡처합니다. 이를 통해 Athena에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다. 자세한 내용은 AWS CloudTrail을 사용하여 Amazon Athena API 호출 로깅 단원을 참조하십시오.

  Athena를 사용하여 Athena뿐만 아니라 다른 AWS 서비스에 대해서도 CloudTrail 로그 파일을 쿼리할 수 있습니다. 자세한 내용은 AWS CloudTrail 로그 쿼리 단원을 참조하십시오.
- CloudTrail 및 Amazon QuickSight로 Athena 사용량 모니터링 – Amazon QuickSight는 완전관리형의 클라우드 기반 비즈니스 인텔리전스 서비스로서 이를 통해 조직 구성원이 모든 장치에서 액세스할 수 있는 대화형 대시보드를 만들 수 있습니다. CloudTrail 및 Amazon QuickSight를 사용하여 Athena 사용량을 모니터링하는 솔루션의 예는 AWS Big Data Blog 게시물 How Realtor.com monitors Amazon Athena usage with AWS CloudTrail and Amazon QuickSight를 참조하세요.
- Athena에서 EventBridge 사용 - Amazon EventBridge는 AWS 리소스의 변경 사항을 설명하는 시스템 이벤트의 스트림을 거의 실시간으로 제공합니다. EventBridge는 이러한 운영 변경 발생 시 이를 인지하고 응답하며, 환경에 응답하기 위한 메시지를 전송하고, 함수를 활성화하며, 변경을 수행하고, 상태 정보를 기록하는 등 필요에 따라 교정 조치를 취합니다. 이벤트는 최선의 작업을 기반으로 발생됩니다. 자세한 내용은 Amazon EventBridge 사용 설명서의 Amazon EventBridge 시작하기를 참조하세요.
- 작업 그룹을 사용하여 사용자, 팀, 애플리케이션 또는 작업 부하를 분리하고 쿼리 제한을 설정하고 쿼리 비용을 통제 - 검색되는 데이터의 양에 대한 제한을 구성하고 임계값과 이러한 임계값이 위반되는 경우에 대해 Amazon SNS 경보와 같은 트리거 조치를 만들어 Amazon CloudWatch의 쿼리 관련 지표를 보고 쿼리 비용을 통제할 수 있습니다. 상위 수준의 절차는 작업 그룹 설정 단원을 참조하세요. 리소스 수준 IAM 권한을 사용하여 특정 작업 그룹에 대한 액세스를 제어할 수 있습니다. 자세한

내용은 쿼리 실행용 작업 그룹 사용 및 CloudWatch 지표 및 이벤트를 사용하여 비용 관리 및 쿼리 모니터링 단원을 참조하세요.

주제

- AWS CloudTrail을 사용하여 Amazon Athena API 호출 로깅

AWS CloudTrail을 사용하여 Amazon Athena API 호출 로깅

Athena는 Athena에서 사용자, 역할 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스인 AWS CloudTrail과 통합됩니다.

CloudTrail은 Athena에 대한 API 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 Athena 콘솔로부터의 호출과 Athena API 작업에 대한 코드 호출이 포함됩니다. 추적을 생성하면 Athena 이벤트를 포함한 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 배포할 수 있습니다. 추적을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록에서 최신 이벤트를 볼 수 있습니다.

CloudTrail에서 수집한 정보를 사용하여 Athena에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하십시오.

Athena를 사용하여 Athena 자체와 다른 AWS 서비스에서 CloudTrail 로그 파일을 쿼리할 수 있습니다. 자세한 내용은 AWS CloudTrail 로그 쿼리, Hive JSON SerDe, 및 AWS 빅 데이터 블로그 게시물 Use CTAS statements with Amazon Athena to reduce cost and improve performance를 참조하세요. 이 블로그 게시물은 CloudTrail을 사용하여 Athena 사용량에 대한 심도 있는 정보를 제공합니다.

CloudTrail의 Athena 정보

CloudTrail은 계정 생성 시 Amazon Web Services 계정에서 활성화됩니다. Athena에서 활동이 수행되면 해당 활동은 이벤트 기록(Event history)에서 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. Amazon Web Services 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 설명은 CloudTrail 이벤트 이력을 사용하여 이벤트 보기를 참조하세요.

Athena에 대한 이벤트를 포함하여 Amazon Web Services 계정에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 리전의 이벤트를 로깅하고 지정된 Amazon S3 버킷으로 로그 파일을 전송합니다. 또는 CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음을 참조하십시오.

- 추적 생성 개요
- CloudTrail 지원 서비스 및 통합
- CloudTrail에 대한 Amazon SNS 알림 구성
- 여러 리전에서 CloudTrail 로그 파일 받기 및 여러 계정에서 CloudTrail 로그 파일 받기

모든 Athena 작업은 CloudTrail에서 로깅되며 Amazon Athena API 참조에 설명되어 있습니다. 예를 들어 StartQueryExecution 및 GetQueryResults 작업을 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다.

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 신원 정보를 이용하면 다음을 쉽게 알아볼 수 있습니다.

- 루트로 요청되었는지 아니면 AWS Identity and Access Management(IAM) 사용자 자격 증명으로 했는지.
- 역할 또는 페더레이션 사용자에 대한 임시 보안 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지 여부.

자세한 내용은 CloudTrail userIdentity 요소를 참조하십시오.

Athena 로그 파일 항목의 이해

추적이란 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

> **Note**
>
> 민감한 정보가 의도치 않게 공개되는 것을 방지하기 위해 `StartQueryExecution` 및 `CreateNamedQuery` 로그의 `queryString` 항목 값은 `***OMITTED***`입니다. 이것은 설계에 따른 것입니다. 실제 쿼리 문자열에 액세스하기 위해 Athena GetQueryExecution API를 사용하고 CloudTrail 로그에서 `responseElements.queryExecutionId`의 값을 전달할 수 있습니다.

다음 예는 다음에 대한 CloudTrail 로그 항목을 보여줍니다.

- StartQueryExecution(성공)
- StartQueryExecution(실패)
- CreateNamedQuery

StartQueryExecution(성공)


```
{
 "eventVersion":"1.05",
 "userIdentity":{
    "type":"IAMUser",
    "principalId":"EXAMPLE_PRINCIPAL_ID",
    "arn":"arn:aws:iam::123456789012:user/johndoe",
    "accountId":"123456789012",
    "accessKeyId":"EXAMPLE_KEY_ID",
    "userName":"johndoe"
 },
 "eventTime":"2017-05-04T00:23:55Z",
 "eventSource":"athena.amazonaws.com",
 "eventName":"StartQueryExecution",
 "awsRegion":"us-east-1",
 "sourceIPAddress":"77.88.999.69",
 "userAgent":"aws-internal/3",
 "requestParameters":{
    "clientRequestToken":"16bc6e70-f972-4260-b18a-db1b623cb35c",
    "resultConfiguration":{
       "outputLocation":"s3://DOC-EXAMPLE-BUCKET/test/"
    },
    "queryString":"***OMITTED***"
 },
 "responseElements":{
    "queryExecutionId":"b621c254-74e0-48e3-9630-78ed857782f9"
 },
 "requestID":"f5039b01-305f-11e7-b146-c3fc56a7dc7a",
 "eventID":"c97cf8c8-6112-467a-8777-53bb38f83fd5",
 "eventType":"AwsApiCall",
 "recipientAccountId":"123456789012"
}
```


StartQueryExecution(실패)


```
{
```

```
 "eventVersion":"1.05",
 "userIdentity":{
  "type":"IAMUser",
  "principalId":"EXAMPLE_PRINCIPAL_ID",
  "arn":"arn:aws:iam::123456789012:user/johndoe",
  "accountId":"123456789012",
  "accessKeyId":"EXAMPLE_KEY_ID",
  "userName":"johndoe"
  },
 "eventTime":"2017-05-04T00:21:57Z",
 "eventSource":"athena.amazonaws.com",
 "eventName":"StartQueryExecution",
 "awsRegion":"us-east-1",
 "sourceIPAddress":"77.88.999.69",
 "userAgent":"aws-internal/3",
 "errorCode":"InvalidRequestException",
 "errorMessage":"Invalid result configuration. Should specify either output location or
 result configuration",
 "requestParameters":{
  "clientRequestToken":"ca0e965f-d6d8-4277-8257-814a57f57446",
  "queryString":"***OMITTED***"
  },
 "responseElements":null,
 "requestID":"aefbc057-305f-11e7-9f39-bbc56d5d161e",
 "eventID":"6e1fc69b-d076-477e-8dec-024ee51488c4",
 "eventType":"AwsApiCall",
 "recipientAccountId":"123456789012"
}
```


CreateNamedQuery


```
{
  "eventVersion":"1.05",
  "userIdentity":{
     "type":"IAMUser",
     "principalId":"EXAMPLE_PRINCIPAL_ID",
     "arn":"arn:aws:iam::123456789012:user/johndoe",
     "accountId":"123456789012",
     "accessKeyId":"EXAMPLE_KEY_ID",
     "userName":"johndoe"
  },
  "eventTime":"2017-05-16T22:00:58Z",
  "eventSource":"athena.amazonaws.com",
```

```
  "eventName":"CreateNamedQuery",
  "awsRegion":"us-west-2",
  "sourceIPAddress":"77.88.999.69",
  "userAgent":"aws-cli/1.11.85 Python/2.7.10 Darwin/16.6.0 botocore/1.5.48",
  "requestParameters":{
     "name":"johndoetest",
     "queryString":"***OMITTED***",
     "database":"default",
     "clientRequestToken":"fc1ad880-69ee-4df0-bb0f-1770d9a539b1"
     },
  "responseElements":{
     "namedQueryId":"cdd0fe29-4787-4263-9188-a9c8db29f2d6"
     },
  "requestID":"2487dd96-3a83-11e7-8f67-c9de5ac76512",
  "eventID":"15e3d3b5-6c3b-4c7c-bc0b-36a8dd95227b",
  "eventType":"AwsApiCall",
  "recipientAccountId":"123456789012"
},
```


Amazon Athena에 대한 규정 준수 확인

타사 감사자는 여러 AWS 규정 준수 프로그램의 일환으로 Amazon Athena의 보안 및 규정 준수를 평가합니다. 여기에는 SOC, PCI, FedRAMP 등이 포함됩니다.

특정 규정 준수 프로그램 범위에 속하는 AWS 서비스의 목록은 규정 준수 프로그램 제공 AWS 서비스 범위 내를 참조하세요. 일반 정보는 AWS 규정 준수 프로그램을 참조하세요.

AWS Artifact를 사용하여 제3자 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 AWS Artifact의 보고서 다운로드를 참조하세요.

Athena 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 결정됩니다. AWS에서는 규정 준수를 지원할 다음과 같은 리소스를 제공합니다.

- 보안 및 규정 준수 빠른 시작 가이드 – 이 배포 가이드에서는 아키텍처 고려 사항에 대해 설명하고 보안 및 규정 준수에 중점을 둔 기본 AWS 환경을 배포하기 위한 단계를 제공합니다.
- Architecting for HIPAA security and compliance on Amazon Web Services - 이 백서는 기업에서 AWS를 사용하여 HIPAA를 준수하는 애플리케이션을 생성하는 방법을 설명합니다.
- AWS 규정 준수 리소스 – 사용자의 업계와 위치에 해당할 수 있는 워크북 및 가이드 모음입니다.
- AWS Config – 이 AWS 서비스로 리소스 구성이 내부 관행, 업계 지침 및 규정을 준수하는 정도를 평가할 수 있습니다.

- AWS Security Hub - 이 AWS 서비스는 보안 산업 표준 및 모범 사례 규정 준수 여부를 확인하는 데 도움이 되도록 AWS 내 보안 상태를 종합적으로 보여줍니다.

Athena의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전에서는 물리적으로 분리되고 격리된 다수의 가용 영역을 제공하며 이러한 가용 영역은 짧은 대기 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 복수 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

AWS 글로벌 인프라뿐만 아니라 Athena도 데이터 복원성과 백업 요구 사항을 지원하는 다양한 기능을 제공합니다.

Athena는 서버리스 서비스이므로 설정하거나 관리할 인프라가 없습니다. Athena는 가용성이 높으며 여러 가용 영역에서 컴퓨팅 리소스를 사용하여 쿼리를 실행하고 특정 가용 영역에 도달할 수 없는 경우 쿼리를 자동으로 라우팅합니다. Athena는 Amazon S3를 기본 데이터 저장소로 사용하기 때문에 데이터의 가용성과 내구성이 높습니다. Amazon S3는 중요한 데이터를 저장하기 위한 견고한 인프라를 제공하며 객체의 99.999999999%에 달하는 내구도로 설계되었습니다. 데이터가 여러 시설과 각 시설의 여러 디바이스에 중복 저장됩니다.

Athena의 인프라 보안

관리형 서비스인 Amazon Athena는 AWS 글로벌 네트워크 보안으로 보호됩니다. AWS 보안 서비스와 AWS의 인프라 보호 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안에 대한 모범 사례를 사용하여 AWS 환경을 설계하려면 보안 원칙 AWS Well-Architected 프레임워크의 인프라 보호를 참조하세요.

AWS에서 게시한 API 호출을 사용하여 네트워크를 통해 Athena에 액세스합니다. 고객은 다음을 지원해야 합니다.

- 전송 계층 보안(TLS) TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군 Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 주체와 관련된 비밀 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

IAM 정책을 사용해 Athena 작업에 대한 액세스를 제한합니다. IAM 정책을 사용할 때는 항상 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

Athena 관리형 정책은 사용하기 쉽고, 서비스가 향상됨에 따라 필요한 작업으로 자동 업데이트됩니다. 고객 관리형 및 인라인 정책을 사용하면 정책 내에서 보다 세분화된 Athena 작업을 지정하여 액세스를 세부 조정할 수 있습니다. 데이터의 Amazon S3 위치에 대한 적절한 권한을 부여합니다. Amazon S3 액세스 권한을 부여하는 방법에 대한 자세한 내용과 시나리오는 Amazon Simple Storage Service 개발자 안내서의 예제 안내: 액세스 관리를 참조하세요. 허용할 Amazon S3 작업에 대한 자세한 내용과 예는 계정 간 액세스의 버킷 정책 예제를 참조하세요.

주제

- 인터페이스 VPC 엔드포인트를 사용하여 Amazon Athena에 연결

인터페이스 VPC 엔드포인트를 사용하여 Amazon Athena에 연결

Virtual Private Cloud(VPC)의 인터페이스 VPC 엔드드포인트(AWS PrivateLink)와 AWS Glue VPC 엔드포인트를 사용하여 VPC의 보안 태세를 개선할 수 있습니다. 인터페이스 VPC 엔드포인트에서는 VPC 내부에서 도달할 수 있는 대상을 제어하는 기능을 제공하여 보안을 개선합니다. 각 VPC 엔드포인트는 하나 이상의 탄력적 네트워크 인터페이스(ENI) 및 VPC 서브넷의 프라이빗 IP 주소로 표현됩니다.

인터페이스 VPC 엔드포인트는 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결 없이 VPC를 Athena에 직접 연결합니다. VPC에 있는 인스턴스는 퍼블릭 IP 주소가 없어도 Athena API와 통신할 수 있습니다.

VPC를 통해 Athena를 사용하려면 VPC 내부에 있는 인스턴스에서 연결하거나 Amazon Virtual Private Network(VPN) 또는 AWS Direct Connect를 사용하여 프라이빗 네트워크를 VPC에 연결해야 합니다. Amazon VPN에 대한 내용은 Amazon Virtual Private Cloud 사용 설명서의 VPN 연결을 참조하세요. AWS Direct Connect에 대한 자세한 내용은 AWS Direct Connect 사용 설명서의 연결 생성을 참조하세요.

Athena는 Amazon VPC와 Athena를 모두 이용할 수 있는 모든 AWS 리전에서 VPC 엔드포인트를 지원합니다.

AWS Management Console 또는 AWS Command Line Interface(AWS CLI) 명령을 사용하여 Athena에 연결할 인터페이스 VPC 엔드포인트를 생성할 수 있습니다. 자세한 내용은 인터페이스 엔드포인트 생성을 참조하세요.

인터페이스 VPC 엔드포인트를 생성한 후 엔드포인트에 대해 프라이빗 DNS 호스트 이름을 활성화하는 경우 기본 Athena 엔드포인트(https://athena.##.amazonaws.com)는 VPC 엔드포인트로 귀결됩니다.

프라이빗 DNS 호스트 이름을 활성화하지 않은 경우, Amazon VPC는 다음 형식으로 사용할 수 있는 DNS 엔드포인트를 제공합니다.

```
VPC_Endpoint_ID.athena.Region.vpce.amazonaws.com
```

자세한 내용은 Amazon VPC 사용 설명서에서 인터페이스 VPC 종단점(AWS PrivateLink)을 참조하십시오.

Athena는 VPC 내부에 있는 모든 API 작업에 대한 호출 수행을 지원합니다.

Athena에 대한 VPC 엔드포인트 정책 생성

Athena에 대한 Amazon VPC 엔드포인트 정책을 생성하여 다음과 같은 제한 사항을 지정할 수 있습니다.

- 보안 주체 - 작업을 수행할 수 있는 보안 주체.
- 작업 - 수행할 수 있는 작업.
- 리소스 - 작업을 수행할 있는 리소스.
- 신뢰할 수 있는 ID만 - `aws:PrincipalOrgId` 조건을 사용하여 AWS 조직의 일부인 보안 인증으로만 액세스를 제한합니다. 이렇게 하면 의도하지 않은 보안 주체의 액세스를 방지할 수 있습니다.
- 신뢰할 수 있는 리소스만 - `aws:ResourceOrgId` 조건을 사용하여 의도하지 않은 리소스에 대한 액세스를 방지합니다.
- 신뢰할 수 있는 ID 및 리소스만 - 의도하지 않은 보안 주체 및 리소스에 대한 액세스를 방지할 수 있는 VPC 엔드포인트에 대한 결합된 정책을 생성합니다.

자세한 내용은 Amazon VPC 사용 설명서의 엔드포인트 정책을 사용하여 VPC 엔드포인트에 대한 액세스 제어 및 AWS 백서 Building a data perimeter on AWS의 Appendix 2 – VPC endpoint policy examples를 참조하세요.

Example - VPC 엔드포인트 정책

다음 예제에서는 조직 ID로 조직 리소스에 대한 요청을 허용하고 AWS 서비스 보안 주체의 요청을 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowRequestsByOrgsIdentitiesToOrgsResources",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "my-org-id",
                    "aws:ResourceOrgID": "my-org-id"
                }
            }
        },
        {
            "Sid": "AllowRequestsByAWSServicePrincipals",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "aws:PrincipalIsAWSService": "true"
                }
            }
        }
    ]
}
```

IAM 정책을 사용할 때마다 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

공유 서브넷

공유하는 서브넷의 VPC 엔드포인트는 생성, 설명, 수정 또는 삭제할 수 없습니다. 그러나 공유하는 서브넷의 VPC 엔드포인트를 사용할 수는 있습니다. VPC 공유에 관한 자세한 내용은 Amazon VPC 사용 설명서의 다른 계정과 VPC 공유를 참조하십시오.

Athena의 구성 및 취약성 분석

Athena는 서버가 없으므로 설정이나 관리의 대상이 되는 인프라가 없습니다. AWS가 게스트 운영 체제(OS), 데이터베이스 패치, 방화벽 구성, 재해 복구 등의 기본 보안 작업을 처리합니다. 적합한 제3자가 이 절차를 검토하고 인증하였습니다. 자세한 내용은 다음 AWS 리소스를 참조하세요.

- 공동 책임 모델
- 보안, 자격 증명 및 규정 준수를 위한 모범 사례

Athena를 사용하여 AWS Lake Formation에 등록된 데이터 쿼리

AWS Lake Formation를 사용하면 Amazon S3에 저장된 데이터를 읽는 Athena 쿼리를 사용할 때 데이터베이스, 테이블 및 열 수준 액세스 정책을 정의하고 적용할 수 있습니다. Lake Formation은 Amazon S3에 저장된 데이터에 대한 권한 및 거버넌스 계층을 제공합니다. Lake Formation의 권한 계층을 사용하여 데이터베이스, 테이블, 열과 같은 데이터 카탈로그 객체를 읽을 수 있는 권한을 부여하거나 취소할 수 있습니다. Lake Formation은 권한 관리를 단순화하며, 이를 통해 데이터에 대한 세분화된 액세스 제어를 구현할 수 있습니다.

Athena를 사용하여 Lake Formation에 등록된 데이터와 Lake Formation에 등록되지 않은 데이터를 모두 쿼리할 수 있습니다.

Lake Formation 권한은 Athena를 사용하여 Lake Formation에 등록된 Amazon S3 위치에서 원본 데이터를 쿼리할 때 적용됩니다. Lake Formation 권한은 등록된 Amazon S3 데이터 위치를 가리키는 데이터베이스 및 테이블을 생성할 때도 적용됩니다. Lake Formation을 사용하여 등록된 데이터를 Athena에 사용하려면 Athena가 AWS Glue Data Catalog를 사용하도록 구성되어야 합니다.

Lake Formation 권한은 Amazon S3에 객체를 쓸 때 적용되지 않으며 Amazon S3에 저장된 데이터나 Lake Formation에 등록되지 않은 메타데이터를 쿼리할 때도 적용되지 않습니다. Lake Formation에 등록되지 않은 Amazon S3의 원본 데이터와 메타데이터의 경우 액세스 권한이 Amazon S3 및 AWS Glue 작업에 대한 IAM 권한 정책에 의해 결정됩니다. Amazon S3의 Athena 쿼리 결과 위치는 Lake Formation에 등록할 수 없으며, Amazon S3에 대한 IAM 권한 정책이 액세스 권한을 제어합니다. 또한 Lake Formation 권한은 Athena 쿼리 기록에 적용되지 않습니다. Athena 작업 그룹을 사용하여 쿼리 기록에 대한 액세스를 제어할 수 있습니다.

Lake Formation에 대한 자세한 내용은 Lake Formation FAQ 및 AWS Lake Formation 개발자 안내서를 참조하세요.

주제

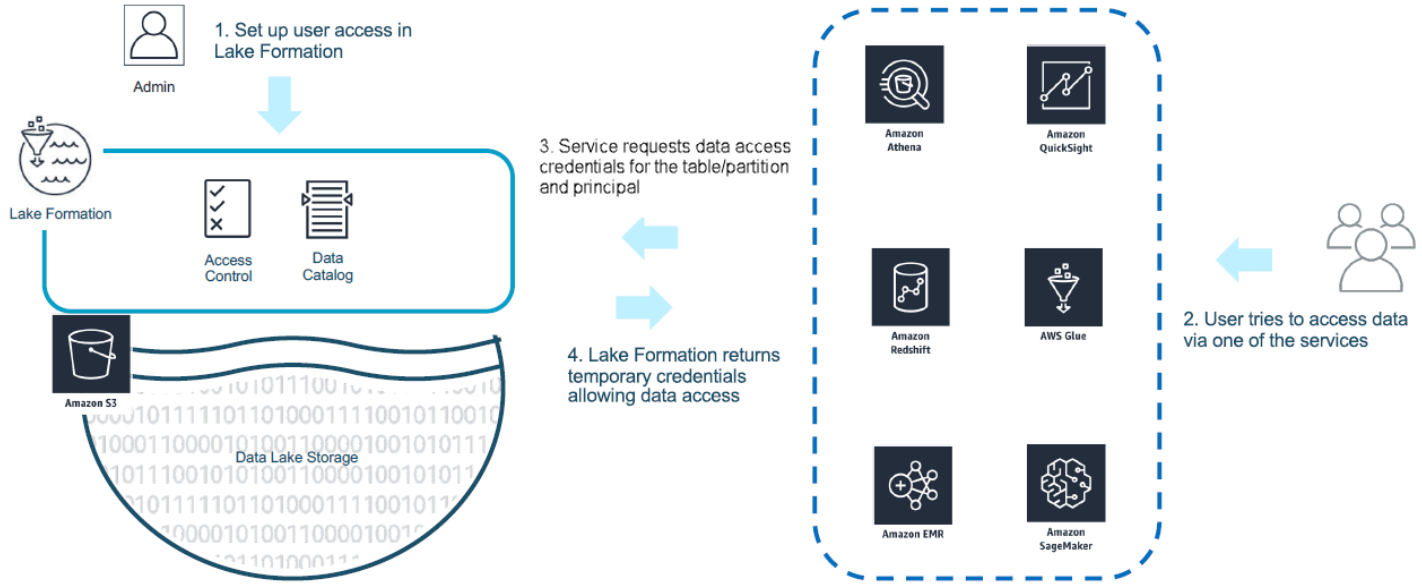
- Athena가 Lake Formation에 등록된 데이터에 액세스하는 방식
- Athena를 사용하여 Lake Formation에 등록된 데이터를 쿼리할 때의 고려 사항 및 제한 사항
- Lake Formation 및 Athena 사용자 권한 관리
- 기존 데이터베이스 및 테이블에 Lake Formation 권한 적용
- Athena에 대한 연합 액세스를 위해 Lake Formation과 Athena JDBC 및 ODBC 드라이버 사용

Athena가 Lake Formation에 등록된 데이터에 액세스하는 방식

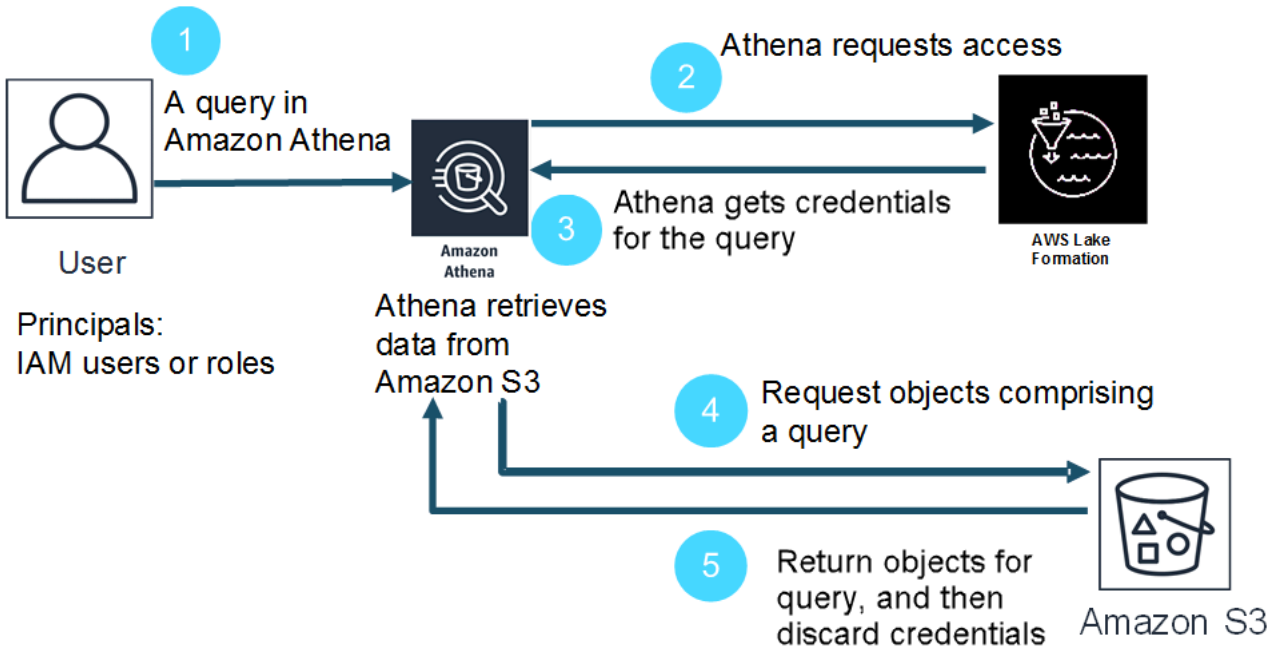
이 단원에서 설명하는 액세스 워크플로는 Lake Formation에 등록된 Amazon S3 위치 및 메타데이터 객체에 대해 Athena 쿼리를 실행할 때만 적용됩니다. 자세한 내용은 AWS Lake Formation 개발자 안내서의 데이터 레이크 등록을 참조하세요. Lake Formation 관리자는 데이터 등록 이외에도 Data Catalog의 메타데이터 및 Amazon S3의 데이터 위치에 대한 액세스 권한을 부여하거나 취소하는 Lake Formation 권한을 적용합니다. 자세한 내용은 AWS Lake Formation 개발자 안내서의 메타데이터 및 데이터에 대한 보안 및 액세스 제어를 참조하세요.

Athena 보안 주체(사용자, 그룹 또는 역할)가 Lake Formation을 사용하여 등록된 데이터에 대해 쿼리를 실행할 때마다 Lake Formation은 해당 보안 주체가 쿼리에 적합한 데이터베이스, 테이블 및 Amazon S3 위치에 대한 적절한 Lake Formation 권한을 가지고 있는지 확인합니다. 보안 주체에 액세스 권한이 있으면 Lake Formation이 일시적인 자격 증명을 Athena에 발급(vend)하고 쿼리가 실행됩니다.

다음 다이어그램은 위에서 설명한 흐름을 보여 줍니다.



다음 다이어그램은 Lake Formation에 등록된 Amazon S3 위치가 있는 테이블에서 Athena의 자격 증명 발급이 가상 SELECT 쿼리에 대해 쿼리별로 작동하는 방식을 보여 줍니다.



1. 보안 주체가 Athena에서 SELECT 쿼리를 실행합니다.
2. Athena는 쿼리를 분석하고 Lake Formation 권한을 검사하여 보안 주체에게 테이블 및 테이블 열에 대한 액세스 권한이 부여되었는지 확인합니다.
3. 보안 주체에게 액세스 권한이 있는 경우 Athena는 Lake Formation으로부터 자격 증명을 요청합니다. 보안 주체에게 액세스 권한이 없는 경우 Athena에서 액세스 거부 오류가 발생합니다.

4. Lake Formation은 Amazon S3에서 데이터를 읽을 때 사용할 자격 증명을, 허용된 열 목록과 함께 Athena에게 발급합니다.
5. Athena는 Lake Formation 임시 자격 증명을 사용하여 Amazon S3에서 데이터를 쿼리합니다. 쿼리가 완료되면 Athena가 자격 증명을 삭제합니다.

Athena를 사용하여 Lake Formation에 등록된 데이터를 쿼리할 때의 고려 사항 및 제한 사항

Athena를 사용하여 Lake Formation에 등록된 데이터를 쿼리할 때는 다음을 고려하세요. 추가 정보는 AWS Lake Formation 개발자 안내서의 AWS Lake Formation의 알려진 문제를 참조하세요.

고려 사항 및 제한

- Avro 및 Custom SerDe를 사용하여 일부 상황에서 권한이 없는 사용자에게 표시되는 열 메타데이터
- 뷰에 대한 Lake Formation 권한 작업
- Lake Formation 세분화된 액세스 제어 및 Athena 작업 그룹
- Lake Formation에 등록되지 않은 Amazon S3의 Athena 쿼리 결과 위치
- Athena 작업 그룹을 사용하여 쿼리 기록에 대한 액세스 제한
- 계정 간 데이터 카탈로그 액세스
- Lake Formation에 등록된 CSE-KMS 암호화 Amazon S3 위치
- Lake Formation에 등록된 분할된 데이터 위치는 테이블 하위 디렉터리에 있어야 함
- CTAS(Create Table As Select) 쿼리에 Amazon S3 쓰기 권한 필요
- 기본 데이터베이스에 대해 DESCRIBE 권한 필요

Avro 및 Custom SerDe를 사용하여 일부 상황에서 권한이 없는 사용자에게 표시되는 열 메타데이터

Lake Formation 열 수준 권한 부여를 통해 사용자가 Lake Formation 권한이 없는 열의 데이터에 액세스하는 것을 방지할 수 있습니다. 그러나 특정 상황에서 사용자는 데이터에 대한 권한이 없는 열을 포함하여 테이블의 모든 열을 설명하는 메타데이터에 액세스할 수 있습니다.

이 현상은 열 메타데이터가 Apache Avro 스토리지 형식을 사용하거나 테이블 스키마가 SerDe(Serializer/Deserializer) 정의와 함께 테이블 속성에 정의된 사용자 지정 SerDe를 사용하는 테이블의 테이블 속성에 저장될 때 발생합니다. Lake Formation과 함께 Athena를 사용할 때는 Lake Formation에 등록한 테이블 속성의 콘텐츠를 검토하고, 가능한 경우 테이블 속성에 저장된 정보를 제한하여 민감한 메타데이터가 사용자에게 표시되지 않도록 하는 것이 좋습니다.

뷰에 대한 Lake Formation 권한 작업

Lake Formation에 등록된 데이터의 경우 Athena 사용자는 `VIEW`가 기반으로 하는 테이블, 열 및 소스 Amazon S3 데이터 위치에 대한 Lake Formation 권한이 있는 경우에만 `VIEW`를 생성할 수 있습니다. Athena에서 `VIEW`를 생성한 후 Lake Formation 권한을 `VIEW`에 적용할 수 있습니다. `VIEW`에는 열 수준 권한을 사용할 수 없습니다. `VIEW`에 대한 Lake Formation 권한은 있지만 뷰의 기반이 되는 테이블 및 열에 대한 권한이 없는 사용자는 `VIEW`를 사용하여 데이터를 쿼리할 수 없습니다. 그러나 이러한 혼합 권한이 있는 사용자는 `DESCRIBE VIEW`, `SHOW CREATE VIEW` 및 `SHOW COLUMNS`와 같은 문을 사용하여 `VIEW` 메타데이터를 볼 수 있습니다. 그러므로 각 `VIEW`에 대한 Lake Formation 권한을 기본 테이블 권한에 맞춰야 합니다. 테이블에 정의된 셀 필터는 해당 테이블의 `VIEW`에 적용되지 않습니다. 리소스 링크 이름이 원래 계정의 리소스와 같아야 합니다. 크로스 계정 설정에서 뷰 작업을 수행할 때 추가 제한 사항이 있습니다. 계정 간에 공유되는 뷰에 대한 권한 설정에 대한 자세한 내용은 계정 간 데이터 카탈로그 액세스을 참조하세요.

Lake Formation 세분화된 액세스 제어 및 Athena 작업 그룹

동일한 Athena 작업 그룹의 사용자는 Lake Formation 세분화된 액세스 제어를 통해 작업 그룹에 액세스할 수 있도록 구성한 데이터를 볼 수 있습니다. Lake Formation에서 세분화된 액세스 제어를 사용하는 방법에 대한 자세한 내용은 AWS 빅 데이터 블로그의 AWS Lake Formation을 사용하여 세분화된 액세스 제어 관리를 참조하세요.

Lake Formation에 등록되지 않은 Amazon S3의 Athena 쿼리 결과 위치

Amazon S3의 Athena 쿼리 결과 위치는 Lake Formation에 등록할 수 없습니다. Lake Formation 권한은 이러한 위치에 대한 액세스를 제한하지 않습니다. 액세스를 제한하지 않는 한 Athena 사용자는 데이터에 대한 Lake Formation 권한이 없어도 쿼리 결과 파일과 메타데이터에 액세스할 수 있습니다. 이를 방지하려면 작업 그룹을 사용하여 쿼리 결과의 위치를 지정하고 작업 그룹 구성원에 맞게 Lake Formation 권한을 조정하는 것이 좋습니다. 그런 다음 IAM 권한 정책을 사용하여 쿼리 결과 위치에 대한 액세스를 제한할 수 있습니다. 쿼리 결과에 대한 자세한 내용은 쿼리 결과, 최근 쿼리, 출력 파일 작업 단원을 참조하세요.

Athena 작업 그룹을 사용하여 쿼리 기록에 대한 액세스 제한

Athena 쿼리 기록은 저장된 쿼리 및 전체 쿼리 문자열 목록을 드러냅니다. 작업 그룹을 사용하여 쿼리 기록에 대한 액세스를 분리하지 않으면 Lake Formation에서 데이터를 쿼리할 권한이 없는 Athena 사용자가 열 이름, 선택 기준 등을 비롯해 해당 데이터에 대해 실행되는 쿼리 문자열을 볼 수 있습니다. 작업 그룹을 사용하여 쿼리 기록을 분리하고 Athena 작업 그룹 구성원에 맞게 Lake Formation 권한을 조정하여 액세스를 제한하는 것이 좋습니다. 자세한 내용은 작업 그룹을 사용하여 쿼리 액세스 및 비용 제어 단원을 참조하세요.

계정 간 데이터 카탈로그 액세스

다른 계정의 데이터 카탈로그에 액세스하기 위해 Athena의 계정 간 AWS Glue 기능을 사용하거나 또는 Lake Formation에서 계정 간 액세스를 설정할 수 있습니다.

Athena 계정 간 데이터 카탈로그 액세스

Athena의 계정 간 AWS Glue 카탈로그 기능을 사용해 자신의 계정에 카탈로그를 등록할 수 있습니다. 이 기능은 Athena 엔진 버전 2 이상 버전에서만 사용할 수 있으며 계정 간 동일한 리전 사용으로 제한됩니다. 자세한 내용은 다른 계정의 AWS Glue Data Catalog 등록 단원을 참조하십시오.

공유할 데이터 카탈로그에 AWS Glue에서 구성된 리소스 정책이 있는 경우 AWS Resource Access Manager에 액세스할 수 있도록 업데이트해야 하고 다음 예제와 같이 계정 A의 데이터 카탈로그를 사용할 수 있는 권한을 계정 B에 부여해야 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": {
            "Service": "ram.amazonaws.com"
        },
        "Action": "glue:ShareResource",
        "Resource": [
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:table/*/*",
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:database/*",
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:catalog"
        ]
    },
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::<ACCOUNT-B>:root"
        },
        "Action": "glue:*",
        "Resource": [
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:table/*/*",
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:database/*",
          "arn:aws:glue:<REGION>:<ACCOUNT-A>:catalog"
        ]
    }
    ]
}
```

자세한 내용은 AWS Glue 데이터 카탈로그에 대한 교차 계정 액세스 단원을 참조하십시오.

Lake Formation에서 계정 간 액세스 설정

AWS Lake Formation에서는 하나의 계정을 사용하여 중앙 데이터 카탈로그를 관리할 수 있습니다. 이 기능을 사용해 계정 간 액세스를 데이터 카탈로그 메타데이터 및 기본 데이터에 구현할 수 있습니다. 예를 들어 소유자 계정은 다른 (수신자) 계정에게 테이블에 대한 SELECT 권한을 부여할 수 있습니다.

Athena 쿼리 편집기에 표시되는 공유 데이터베이스 또는 테이블의 경우 Lake Formation에서 공유 데이터베이스 또는 테이블에 연결되는 리소스 링크를 생성합니다. Lake Formation의 수신자 계정이 소유자의 테이블을 쿼리하면 CloudTrail은 수신자 계정과 소유자 계정의 로그에 데이터 액세스 이벤트를 추가합니다.

공유된 뷰의 경우 다음 사항에 유의하십시오.

- 쿼리는 소스 테이블이나 뷰가 아닌 대상 리소스 링크에서 실행되며 출력이 대상 계정에 공유됩니다.
- 뷰만 공유하는 것으로는 충분하지 않습니다. 뷰 생성과 관련된 모든 테이블은 계정 간 공유의 일부여야 합니다.
- 공유된 리소스에 생성된 리소스 링크의 이름이 소유자 계정의 리소스 이름과 일치해야 합니다. 이름이 일치하지 않으면 Failed analyzing stored view 'awsdatacatalog.*my-lf-resource-link*.*my-lf-view*': line 3:3: Schema *schema_name* does not exist와 같은 오류 메시지가 표시됩니다.

Lake Formation에서 계정 간 액세스에 대한 자세한 내용은 AWS Lake Formation 개발자 안내서의 다음 리소스를 참조하세요.

교차 계정 액세스

Lake Formation에서 리소스 링크가 작동하는 방식

계정 간 CloudTrail 로깅

Lake Formation에 등록된 CSE-KMS 암호화 Amazon S3 위치

다음과 같은 특성을 가진 Apache Iceberg와 같은 OTF(오픈 테이블 형식) 테이블은 Athena로 쿼리할 수 없습니다.

- 테이블은 Lake Formation에 등록된 Amazon S3 데이터 위치를 기반으로 합니다.
- Amazon S3의 객체가 CSE(클라이언트 측 암호화)를 사용하여 암호화됩니다.
- 암호화에서는 AWS KMS 고객 관리형 키(CSE_KMS)를 사용합니다.

CSE_KMS 키로 암호화된 OTF가 아닌 테이블을 쿼리하려면 CSE 암호화에 사용하는 AWS KMS 키의 정책에 다음 블록을 추가하세요. *<KMS_KEY_ARN>*은 데이터를 암호화하는 AWS KMS 키의 ARN입니다. *<IAM-ROLE-ARN>*은 Lake Formation에 Amazon S3 위치를 등록하는 IAM 역할의 ARN입니다.

```
{
    "Sid": "Allow use of the key",
    "Effect": "Allow",
    "Principal": {
        "AWS": "*"
    },
    "Action": "kms:Decrypt",
    "Resource": "<KMS-KEY-ARN>",
    "Condition": {
        "ArnLike": {
            "aws:PrincipalArn": "<IAM-ROLE-ARN>"
        }
    }
}
```

Lake Formation에 등록된 분할된 데이터 위치는 테이블 하위 디렉터리에 있어야 함

Lake Formation에 등록된 분할된 테이블은 Amazon S3 테이블의 하위 디렉터리인 디렉터리에 분할된 데이터가 있어야 합니다. 예를 들어, 위치가 `s3://DOC-EXAMPLE-BUCKET/mytable`이고 파티션이 `s3://DOC-EXAMPLE-BUCKET/mytable/dt=2019-07-11`, `s3://DOC-EXAMPLE-BUCKET/mytable/dt=2019-07-12` 등인 테이블은 Lake Formation에 등록하고 Athena를 사용하여 쿼리할 수 있습니다. 반면, 위치가 `s3://DOC-EXAMPLE-BUCKET/mytable`이고 파티션이 `s3://DOC-EXAMPLE-BUCKET/dt=2019-07-11`, `s3://DOC-EXAMPLE-BUCKET/dt=2019-07-12` 등인 테이블은 Lake Formation에 등록할 수 없습니다. 이러한 파티션은 `s3://DOC-EXAMPLE-BUCKET/mytable`의 하위 디렉토리가 아니기 때문에 Athena에서 읽을 수도 없습니다.

CTAS(Create Table As Select) 쿼리에 Amazon S3 쓰기 권한 필요

CTAS(Create Table As Statements)에는 테이블의 Amazon S3 위치에 대한 쓰기 액세스 권한이 필요합니다. Lake Formation에 등록된 데이터에 대해 CTAS 쿼리를 실행하려면 Athena 사용자는 데이터 위치를 읽을 수 있는 적절한 Lake Fomation 권한과 더불어 테이블의 Amazon S3 위치에 쓸 수 있는 IAM 권한이 있어야 합니다. 자세한 내용은 쿼리 결과에서 테이블 생성(CTAS) 단원을 참조하세요.

기본 데이터베이스에 대해 DESCRIBE 권한 필요

default 데이터베이스에 대해 Lake Formation DESCRIBE 권한이 필요합니다. 다음 예제에서는 AWS CLI 명령으로 AWS 계정 111122223333의 datalake_user1 사용자에게 default 데이터베이스에 대한 DESCRIBE 권한을 부여합니다.

```
aws lakeformation grant-permissions --principal
 DataLakePrincipalIdentifier=arn:aws:iam::111122223333:user/datalake_user1 --
permissions "DESCRIBE" --resource '{ "Database": {"Name":"default"}}
```

자세한 내용은 AWS Lake Formation 개발자 안내서의 Lake Formation 권한 참조를 참조하세요.

Lake Formation 및 Athena 사용자 권한 관리

Lake Formation은 Lake Formation에 등록된 Amazon S3 데이터 스토어를 쿼리할 수 있도록 자격 증명을 공급합니다. 이전에 IAM 정책을 사용하여 Amazon S3에서 데이터 위치를 읽을 수 있는 권한을 허용하거나 거부한 경우 Lake Formation 권한을 대신 사용할 수 있습니다. 그러나 다른 IAM 권한이 여전히 필요합니다.

IAM 정책을 사용할 때마다 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

다음 단원에서는 Athena를 사용하여 Lake Formation에 등록된 데이터를 쿼리하는 데 필요한 권한을 요약합니다. 자세한 정보는 AWS Lake Formation 개발자 안내서의 AWS Lake Formation의 보안을 참조하세요.

권한 요약

- Lake Formation 및 Athena에 대한 자격 증명 기반 권한
- Athena 쿼리 결과 위치에 대한 Amazon S3 권한
- 쿼리 기록에 대한 Athena 작업 그룹 멤버십
- 데이터에 대한 Lake Formation 권한
- Amazon S3 위치에 쓸 수 있는 IAM 권한
- 암호화된 데이터, 메타데이터 및 Athena 쿼리 결과에 대한 권한
- 외부 계정의 Amazon S3 버킷에 대한 리소스 기반 권한(선택 사항)

Lake Formation 및 Athena에 대한 자격 증명 기반 권한

Athena를 사용하여 Lake Formation에 등록된 데이터를 쿼리하는 사용자는 `lakeformation:GetDataAccess` 작업을 허용하는 IAM 권한 정책이 있어야 합니다. 이 작업을 허용하는 정책은 AWS 관리형 정책: AmazonAthenaFullAccess입니다. 인라인 정책을 사용하는 경우 이 작업을 허용하도록 권한 정책을 업데이트하세요.

Lake Formation에서 데이터 레이크 관리자는 데이터베이스 및 테이블과 같은 메타데이터 객체를 생성하고, 다른 사용자에게 Lake Formation 권한을 부여하며, 새로운 Amazon S3 위치를 등록할 수 있는 권한이 있습니다. 새 위치를 등록하려면 Lake Formation의 서비스 연결 역할에 대한 권한이 필요합니다. 자세한 내용은 AWS Lake Formation 개발자 안내서의 데이터 레이크 관리자 생성 및 Lake Formation의 서비스 연결 역할 권한을 참조하세요.

Lake Formation 사용자는 Athena를 사용하여 데이터 레이크 관리자가 부여한 Lake Formation 권한에 따라 데이터베이스, 테이블, 테이블 열 및 기본 Amazon S3 데이터 스토어를 쿼리할 수 있습니다. 사용자는 데이터베이스나 테이블을 만들거나 Lake Formation에 새 Amazon S3 위치를 등록할 수 없습니다. 자세한 내용은 AWS Lake Formation 개발자 안내서의 데이터 레이크 사용자 생성을 참조하세요.

Athena에서 Athena 작업 그룹에 대한 권한을 포함하여 자격 증명 기반 권한 정책은 여전히 Amazon Web Services 계정 사용자의 Athena 작업에 대한 액세스를 제어합니다. 또한 Athena 드라이버에서 사용할 수 있는 SAML 기반 인증을 통해 연합 액세스가 제공될 수 있습니다. 자세한 내용은 작업 그룹을 사용하여 쿼리 액세스 및 비용 제어, 작업 그룹 액세스를 위한 IAM 정책, Athena API에 대한 연합 액세스 활성화 단원을 참조하세요.

자세한 내용은 AWS Lake Formation 개발자 안내서의 Lake Formation 권한 부여를 참조하세요.

Athena 쿼리 결과 위치에 대한 Amazon S3 권한

Amazon S3의 Athena 쿼리 결과 위치는 Lake Formation에 등록할 수 없습니다. Lake Formation 권한은 이러한 위치에 대한 액세스를 제한하지 않습니다. 액세스를 제한하지 않는 한 Athena 사용자는 데이터에 대한 Lake Formation 권한이 없어도 쿼리 결과 파일과 메타데이터에 액세스할 수 있습니다. 이를 방지하려면 작업 그룹을 사용하여 쿼리 결과의 위치를 지정하고 작업 그룹 구성원에 맞게 Lake Formation 권한을 조정하는 것이 좋습니다. 그런 다음 IAM 권한 정책을 사용하여 쿼리 결과 위치에 대한 액세스를 제한할 수 있습니다. 쿼리 결과에 대한 자세한 내용은 쿼리 결과, 최근 쿼리, 출력 파일 작업 단원을 참조하세요.

쿼리 기록에 대한 Athena 작업 그룹 멤버십

Athena 쿼리 기록은 저장된 쿼리 및 전체 쿼리 문자열 목록을 드러냅니다. 작업 그룹을 사용하여 쿼리 기록에 대한 액세스를 분리하지 않으면 Lake Formation에서 데이터를 쿼리할 권한이 없는 Athena 사

용자가 열 이름, 선택 기준 등을 비롯해 해당 데이터에 대해 실행되는 쿼리 문자열을 볼 수 있습니다. 작업 그룹을 사용하여 쿼리 기록을 분리하고 Athena 작업 그룹 구성원에 맞게 Lake Formation 권한을 조정하여 액세스를 제한하는 것이 좋습니다. 자세한 내용은 작업 그룹을 사용하여 쿼리 액세스 및 비용 제어 단원을 참조하세요.

데이터에 대한 Lake Formation 권한

Lake Formation을 사용할 수 있는 기본 권한 외에도 Athena 사용자는 쿼리하는 리소스에 액세스할 수 있는 Lake Formation 권한이 있어야 합니다. 이러한 권한은 Lake Formation 관리자가 부여하고 관리합니다. 자세한 내용은 AWS Lake Formation 개발자 안내서의 메타데이터 및 데이터에 대한 보안 및 액세스 제어를 참조하세요.

Amazon S3 위치에 쓸 수 있는 IAM 권한

Amazon S3에 대한 Lake Formation 권한에는 Amazon S3에 쓸 수 있는 권한이 포함되어 있지 않습니다. CTAS(Create Table As Statements)에는 테이블의 Amazon S3 위치에 대한 쓰기 액세스 권한이 필요합니다. Lake Formation에 등록된 데이터에 대해 CTAS 쿼리를 실행하려면 Athena 사용자는 데이터 위치를 읽을 수 있는 적절한 Lake Fomation 권한과 더불어 테이블의 Amazon S3 위치에 쓸 수 있는 IAM 권한이 있어야 합니다. 자세한 내용은 쿼리 결과에서 테이블 생성(CTAS) 단원을 참조하세요.

암호화된 데이터, 메타데이터 및 Athena 쿼리 결과에 대한 권한

Amazon S3의 기본 소스 데이터와 Lake Formation에 등록된 데이터 카탈로그의 메타데이터를 암호화할 수 있습니다. Athena를 사용하여 Lake Formation에 등록된 데이터를 쿼리할 때 Athena가 쿼리 결과의 암호화를 처리하는 방식에는 변화가 없습니다. 자세한 내용은 Amazon S3에 저장된 Athena 쿼리 결과 암호화 단원을 참조하십시오.

- 소스 데이터 암호화 – Amazon S3 데이터 위치 소스 데이터의 암호화가 지원됩니다. Lake Formation에 등록된 암호화 Amazon S3 위치를 쿼리하는 Athena 사용자는 데이터를 암호화하고 해독할 수 있는 권한이 필요합니다. 요구 사항에 대한 자세한 내용은 지원되는 Amazon S3 암호화 옵션 및 Amazon S3의 암호화 데이터 권한 섹션을 참조하세요.
- 메타데이터 암호화 – 데이터 카탈로그의 메타데이터를 암호화할 수 있습니다. Athena를 사용하는 보안 주체의 경우, 자격 증명 기반 정책에서 메타데이터 암호화에 사용된 키에 대한 `"kms:GenerateDataKey"`, `"kms:Decrypt"` 및 `"kms:Encrypt"` 작업을 허용해야 합니다. 자세한 내용은 AWS Glue 개발자 안내서의 데이터 카탈로그 암호화와 Athena에서 AWS Glue Data Catalog의 암호화된 메타데이터에 액세스 단원을 참조하세요.

외부 계정의 Amazon S3 버킷에 대한 리소스 기반 권한(선택 사항)

다른 계정에서 Amazon S3 데이터 위치를 쿼리하려면 리소스 기반 IAM 정책(버킷 정책)이 해당 위치에 대한 액세스를 허용해야 합니다. 자세한 내용은 Amazon S3 버킷에 대한 Athena의 계정 간 액세스 단원을 참조하세요.

다른 계정의 데이터 카탈로그에 액세스하는 방법에 대한 자세한 내용은 Athena 계정 간 데이터 카탈로그 액세스 단원을 참조하세요.

기존 데이터베이스 및 테이블에 Lake Formation 권한 적용

Athena를 처음 사용하고 Lake Formation을 사용하여 쿼리 데이터에 대한 액세스를 구성하는 경우 사용자가 Amazon S3 데이터를 읽고 메타데이터를 생성할 수 있도록 IAM 정책을 구성할 필요가 없습니다. Lake Formation을 사용하여 권한을 관리할 수 있습니다.

Lake Formation에 데이터를 등록하고 IAM 권한 정책을 업데이트할 필요는 없습니다. 데이터가 Lake Formation에 등록되지 않은 경우에도 Amazon S3(해당하는 경우 AWS Glue)의 적절한 권한을 가진 Athena 사용자는 Lake Formation에 등록되지 않은 데이터를 계속 쿼리할 수 있습니다.

Lake Formation에 등록되지 않은 데이터를 쿼리하는 기존 Athena 사용자가 있는 경우 Amazon S3(해당되는 경우 AWS Glue Data Catalog)에 대한 IAM 권한을 업데이트할 수 있으며 Lake Formation 권한을 사용하여 중앙에서 사용자 액세스를 관리할 수 있습니다. Amazon S3 데이터 위치 읽기 권한의 경우, 리소스 기반 정책 및 자격 증명 기반 정책을 업데이트하여 Amazon S3 권한을 수정할 수 있습니다. 메타데이터 액세스의 경우, AWS Glue를 사용하여 세분화된 액세스 제어를 위해 리소스 수준 정책을 구성하면 Lake Formation 권한을 사용하여 대신 액세스를 관리할 수 있습니다.

자세한 내용은 AWS Lake Formation 개발자 안내서의 AWS Glue Data Catalog의 데이터베이스와 테이블에 대한 세분화된 액세스 및 AWS Lake Formation 모델로 AWS Glue 데이터 권한 업그레이드를 참조하세요.

Athena에 대한 연합 액세스를 위해 Lake Formation과 Athena JDBC 및 ODBC 드라이버 사용

Athena JDBC 및 ODBC 드라이버는 Okta 및 Microsoft Active Directory Federation Services(AD FS) 자격 증명 공급자를 사용하여 Athena에 SAML 2.0 연합을 지원합니다. Amazon Athena와 AWS Lake Formation이(가) 통합되면서 기업 자격 증명으로 Athena에 대한 SAML 기반 인증이 가능해졌습니다. Lake Formation과 AWS Identity and Access Management(IAM)을(를) 사용하면 SAML 사용자가 사용할 수 있는 데이터에 대해 세분화된 열 수준 액세스 제어를 유지할 수 있습니다. Athena JDBC 및 ODBC 드라이버가 있으면 도구 또는 프로그래밍 방식 액세스에 연합 액세스를 이용할 수 있습니다.

Athena 사용하여 Lake Formation에서 제어하는 데이터 원본에 액세스하려면 자격 증명 공급자(IdP)와 AWS Identity and Access Management(IAM) 역할을 구성하여 SAML 2.0 연합을 활성화해야 합니다. 자세한 단계는 자습서: Lake Formation 및 JDBC를 사용하여 Athena에 대한 Okta 사용자의 연합 액세스 구성 단원을 참조하세요.

필수 조건

연합 액세스를 위해 Amazon Athena 및 Lake Formation을 사용하려면 다음 요구 사항을 충족해야 합니다.

- Okta 또는 Microsoft AD FS(Active Directory Federation Services) 등의 기존 SAML 기반 자격 증명 공급자를 이용하여 회사 자격 증명을 관리합니다.
- AWS Glue Data Catalog를 메타데이터 스토어로 사용합니다.
- AWS Glue Data Catalog의 데이터베이스, 테이블 및 열에 액세스할 수 있는 권한을 Lake Formation에서 정의하고 관리합니다. 자세한 내용은 AWS Lake Formation 개발자 안내서를 참조하세요.
- Athena JDBC 드라이버 버전 2.0.14 이상 또는 Athena ODBC 드라이버 버전 1.1.3 이상을 사용합니다.

고려 사항 및 제한

Athena JDBC 또는 ODBC 드라이버와 Lake Formation을 사용하여 Athena 대한 연합 액세스를 구성할 때는 다음 사항에 유의해야 합니다.

- 현재 Athena JDBC 및 ODBC 드라이버는 Okta, Microsoft Active Directory Federation Services(AD FS) 및 Azure AD ID 제공업체를 지원합니다. Athena JDBC 드라이버에는 다른 자격 증명 공급자를 사용할 수 있게 확장 가능한 일반 SAML 클래스가 있지만 Athena에 사용할 다른 자격 증명 공급자(IdP)를 활성화하는 사용자 지정 익스텐션에 대한 지원이 제한될 수 있습니다.
- JDBC 및 ODBC 드라이버를 사용한 연동 액세스는 IAM Identity Center의 신뢰할 수 있는 ID 전파 기능과 호환되지 않습니다.
- 현재는 Athena 콘솔을 사용하여 Athena에서의 IdP 및 SAML 사용 지원을 구성할 수 없습니다. 이 지원을 구성하려면 서드 파티 자격 증명 공급자, Lake Formation 및 IAM 관리 콘솔, JDBC 또는 ODBC 드라이버 클라이언트를 사용합니다.
- 개발자는 Lake Formation 및 Athena에 사용할 자격 증명 공급자 및 SAML을 구성하기 전에 SAML 2.0 사양을 이해하고 자신의 자격 증명 공급자에서 어떻게 작동하는지 파악해야 합니다.
- SAML 공급자와 Athena JDBC 및 ODBC 드라이버는 서드 파티에서 제공하므로 그 사용과 관련된 문제에 대해 AWS를 통한 지원은 제한될 수 있습니다.

주제

- 자습서: Lake Formation 및 JDBC를 사용하여 Athena에 대한 Okta 사용자의 연합 액세스 구성

자습서: Lake Formation 및 JDBC를 사용하여 Athena에 대한 Okta 사용자의 연합 액세스 구성

이 자습서에서는 Athena의 SAML 기반 연합 사용을 활성화하기 위해 Okta, AWS Lake Formation, AWS Identity and Access Management 권한 및 Athena JDBC 드라이버를 구성하는 방법을 보여줍니다 Lake Formation은 SAML 기반 사용자에게 Athena에서 사용 가능한 데이터에 대해 세분화된 액세스 제어를 제공합니다. 이 구성을 설정하기 위해 자습서에서는 Okta 개발자 콘솔, AWS IAM 및 Lake Formation 콘솔, SQL Workbench/J tool을 사용합니다.

필수 조건

이 자습서는 다음을 이미 완료했다고 가정합니다.

- Amazon Web Services 계정을 생성했습니다. 계정을 만들려면 Amazon Web Services 홈페이지를 방문하세요.
- Amazon S3에서 Athena에 대한 쿼리 결과 위치를 설정했습니다.
- Lake Formation에 Amazon S3 데이터 버킷 위치를 등록했습니다.
- Amazon S3의 데이터를 가리키는 AWS Glue 데이터 카탈로그의 데이터베이스 및 테이블을 정의했습니다.
  - 테이블을 아직 정의하지 않았다면 AWS Glue 크롤러를 실행하거나 Athena를 사용해 액세스하려는 데이터에 대해 하나의 데이터베이스와 하나 이상의 테이블을 정의합니다.
  - 이 자습서는 뉴욕시 택시 이동 데이터 집합에 기반한 테이블을 사용합니다. 이 테이블은 Registry of Open Data on AWS에 제공되어 있습니다. 이 자습서에서는 데이터베이스 이름 `tripdb`와 테이블 이름 `nyctaxi`를 사용합니다.

자습서 단계

- 1단계: Okta 계정 생성
- 2단계: Okta에 사용자 및 그룹 추가
- 3단계: SAML 인증을 위한 Okta 애플리케이션 설정
- 4단계: AWS SAML 자격 증명 공급자 및 Lake Formation 액세스 IAM 역할 생성
- 5단계: Okta 애플리케이션에 IAM 역할 및 SAML 자격 증명 공급자 추가
- 6단계: AWS Lake Formation을(를) 통해 사용자 및 그룹 권한 부여
- 7단계: Athena JDBC 클라이언트를 통해 액세스 권한 확인

- 결론
- 관련 리소스

1단계: Okta 계정 생성

이 자습서는 Okta를 SAML 기반 자격 증명 공급자로 사용합니다. 아직 Okta 계정이 없다면 무료로 계정을 생성할 수 있습니다. SAML 인증을 위한 Okta 애플리케이션을 만들려면 Okta 계정이 필요합니다.

Okta 계정을 생성하려면

1. Okta를 사용하려면 Okta 개발자 가입 페이지를 탐색해 무료 Okta 평가판 계정을 만듭니다. 개발자 에디션 서비스는 Okta가 developer.okta.com/pricing에 지정한 한도까지 무료입니다.
2. 활성화 이메일을 받으면 계정을 활성화하세요.

   Okta 도메인 이름이 할당됩니다. 참조용으로 도메인 이름을 저장합니다. 나중에 Athena에 연결하는 JDBC 문자열에 이 도메인 이름(*<okta-idp-domain>*)을 사용합니다.

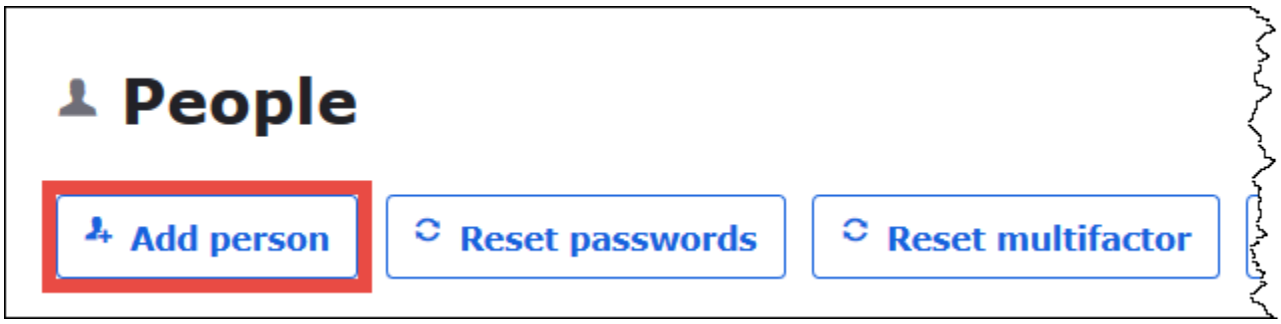
2단계: Okta에 사용자 및 그룹 추가

이 단계에서는 Okta 콘솔을 사용하여 다음 작업을 수행합니다.

- 2개의 Okta 사용자를 만듭니다.
- 2개의 Okta 그룹을 만듭니다.
- 각 Okta 그룹에 1개의 Okta 사용자를 추가합니다.

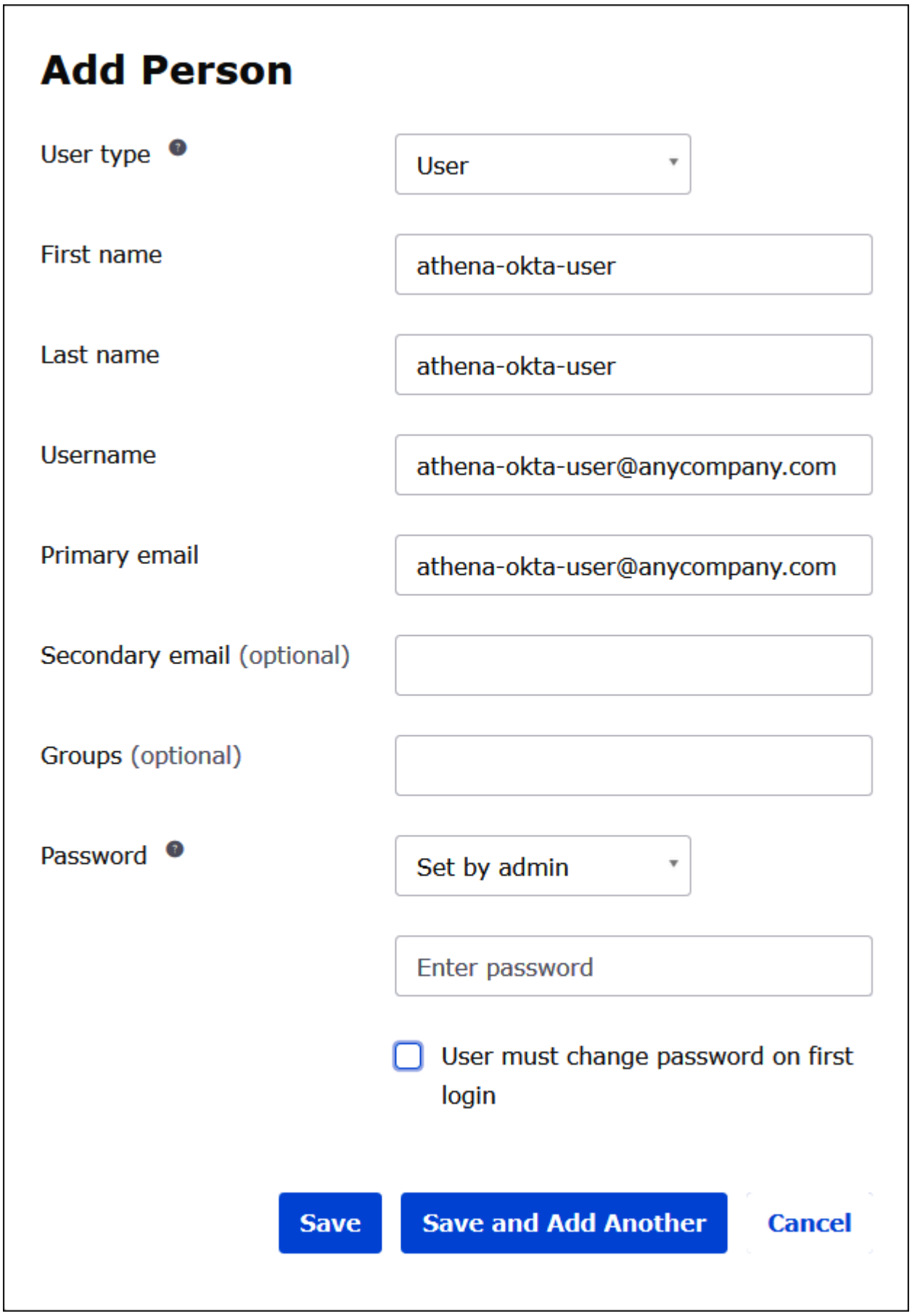
Okta에 사용자를 추가하려면

1. Okta 계정을 활성화한 후 할당된 Okta 도메인에 관리 사용자로 로그인합니다.
2. 왼쪽 탐색 창에서 디렉터리(Directory)와 사람(People)을 차례로 선택합니다.
3. 사람 추가(Add Person)를 선택해 JDBC 드라이버를 통해 Athena에 액세스할 새 사용자를 추가합니다.

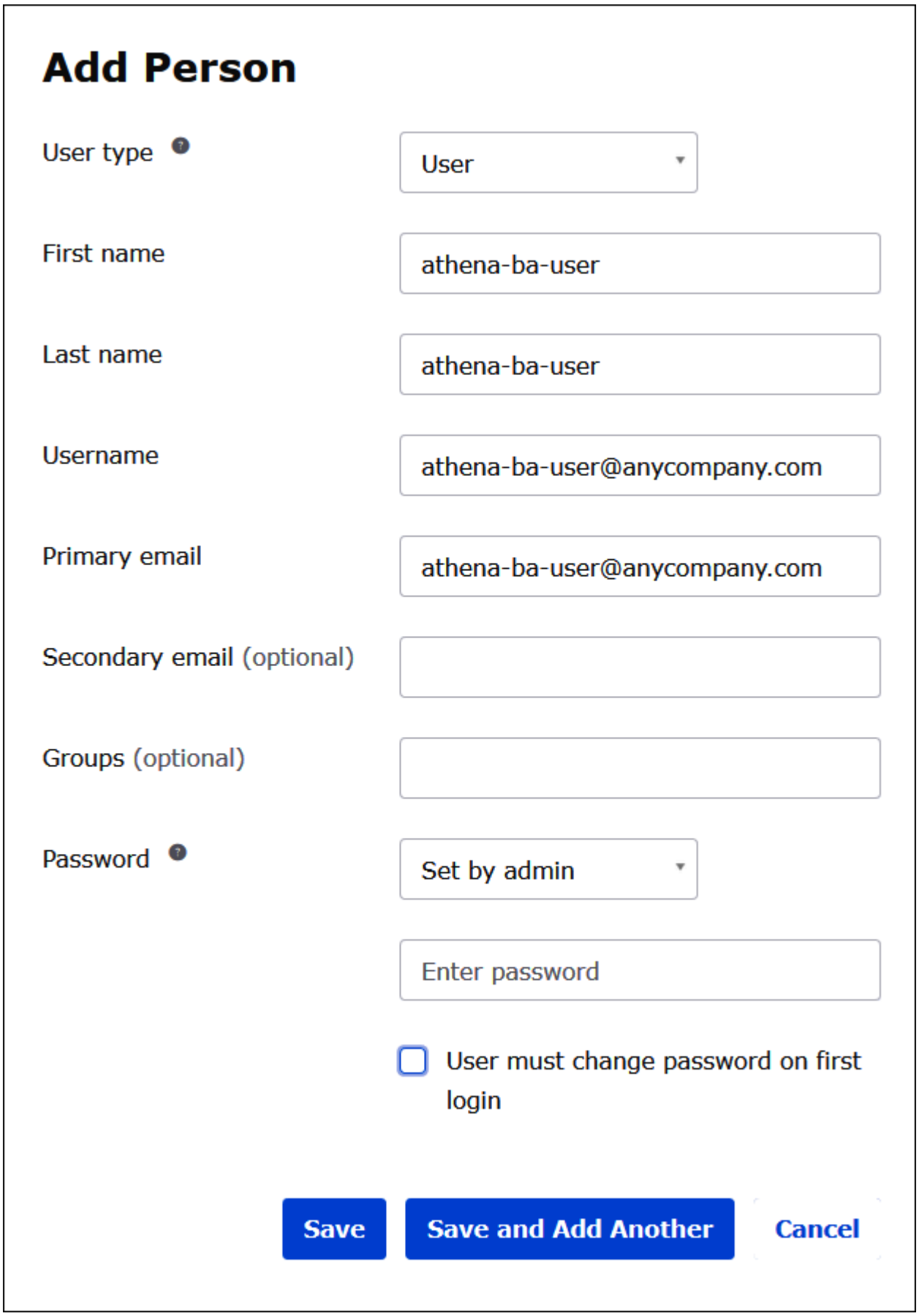


4. 사람 추가(Add Person) 대화 상자에 필수 정보를 입력합니다.

   - 이름(First name)과 성(Last name)에 값을 입력합니다. 이 자습서에서는 *athena-okta-user*를 사용합니다.
   - 사용자 이름(Username)과 기본 이메일(Primary email)을 입력합니다. 이 자습서에서는 *athena-okta-user@anycompany.com*을 사용합니다.
   - 암호(Password)에서 관리자별 설정(Set by admin)을 선택한 다음 암호를 제공합니다. 이 자습서에서는 사용자가 첫 로그인할 때 암호를 변경해야 함(User must change password on first login) 옵션을 선택 취소했습니다. 각자의 보안 요구 사항은 다를 수 있습니다.


Add Person
User type
User
First name
athena-okta-user
Last name
athena-okta-user
Username
athena-okta-user@anycompany.com
Primary email
athena-okta-user@anycompany.com
Secondary email (optional)
Groups (optional)
Password
Set by admin
Enter password
User must change password on first login
Save
Save and Add Another
Cancel

5. 다른 사용자 저장 및 추가(Save and Add Another)를 선택합니다.
6. 다른 사용자에 대한 정보를 입력합니다. 이 예제에서는 비즈니스 분석가 사용자 *athena-ba-user@anycompany.com*를 추가합니다.

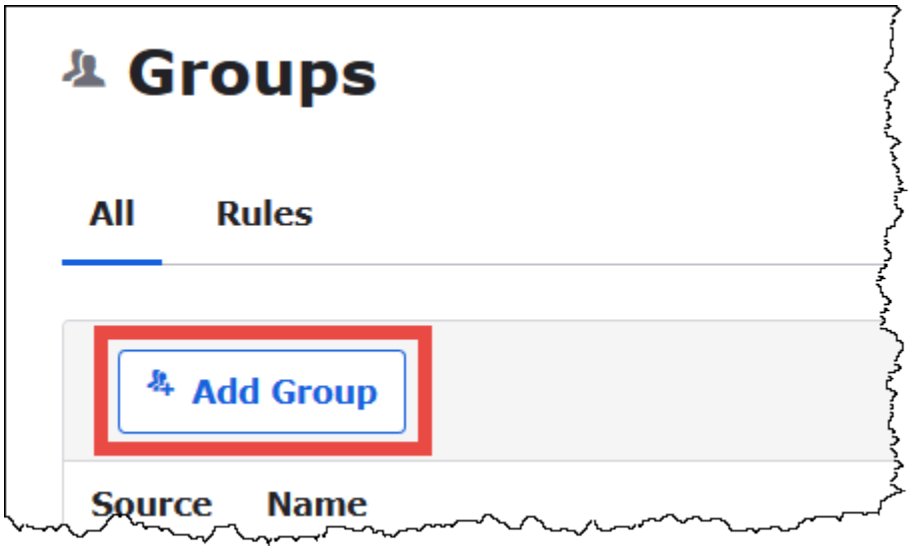

Add Person
User type
User
First name
athena-ba-user
Last name
athena-ba-user
Username
athena-ba-user@anycompany.com
Primary email
athena-ba-user@anycompany.com
Secondary email (optional)
Groups (optional)
Password
Set by admin
Enter password
User must change password on first login
Save
Save and Add Another
Cancel

7. 저장(Save)을 선택합니다.

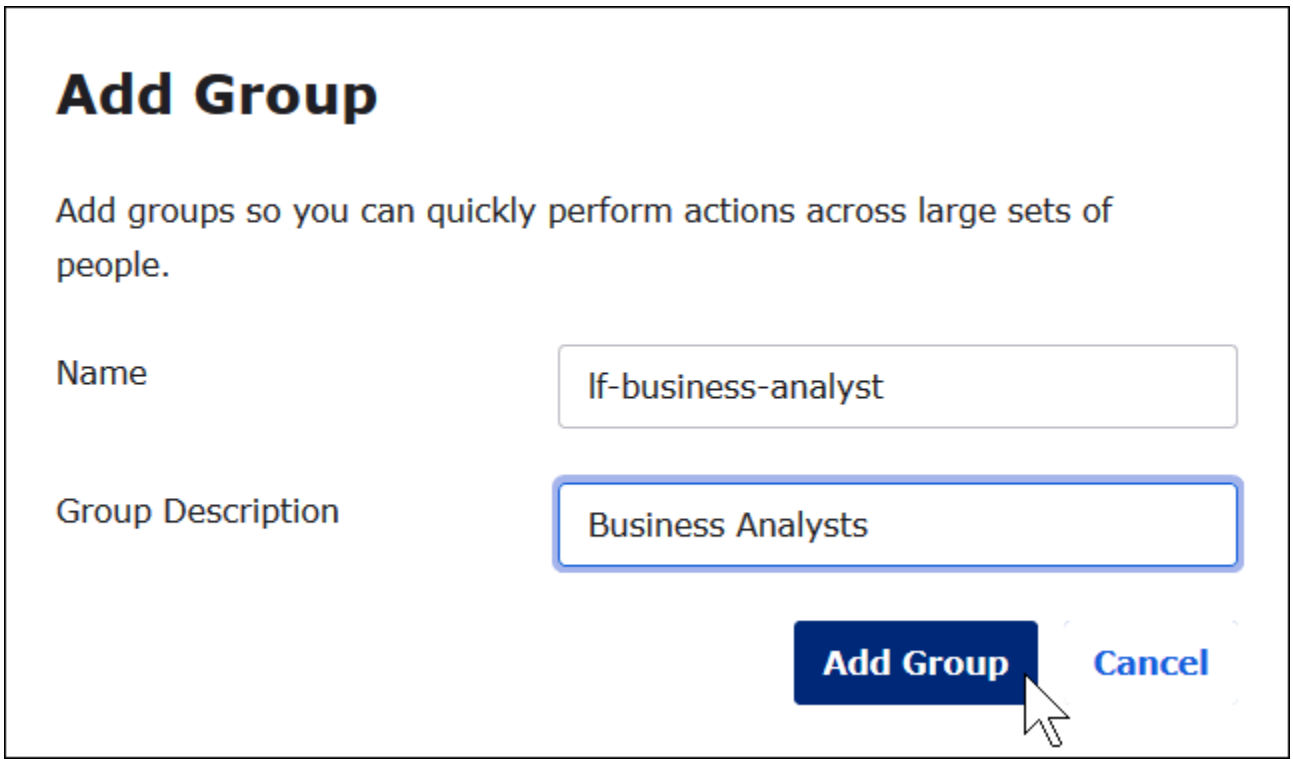
다음 절차에서는 “비즈니스 분석가” 그룹과 “개발자” 그룹을 추가하여 Athena JDBC 드라이버를 통해 두 Okta 그룹에 대한 액세스를 제공합니다.

Okta 그룹을 추가하려면

1. Okta 탐색 창에서 디렉터리(Directory)와 그룹(Groups)을 차례로 선택합니다.
2. 그룹(Groups) 페이지에서 그룹 추가(Add Group)를 선택합니다.



3. 그룹 추가(Add Group) 대화 상자에 필수 정보를 입력합니다.
   - 이름(Name)에 *lf-business-analyst*를 입력합니다.
   - 그룹 설명(Group Description)에 *#### ###*를 입력합니다.

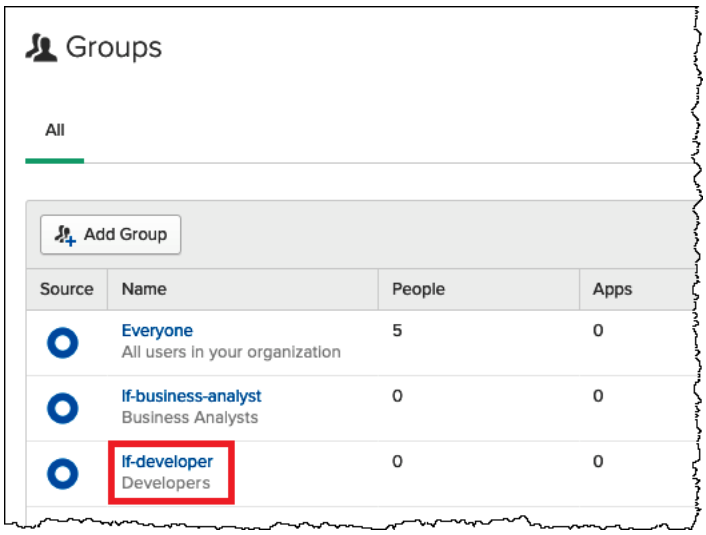


4. Add Group(그룹 추가)을 선택합니다.
5. 그룹(Groups) 페이지에서 그룹 추가(Add Group)를 다시 선택합니다. 이번에는 개발자 그룹에 대한 정보를 입력합니다.
6. 필요한 정보를 입력합니다.
   - 이름(Name)에 *lf-developer*를 입력합니다.
   - 그룹 설명(Group Description)에 ###를 입력합니다.
7. Add Group(그룹 추가)을 선택합니다.

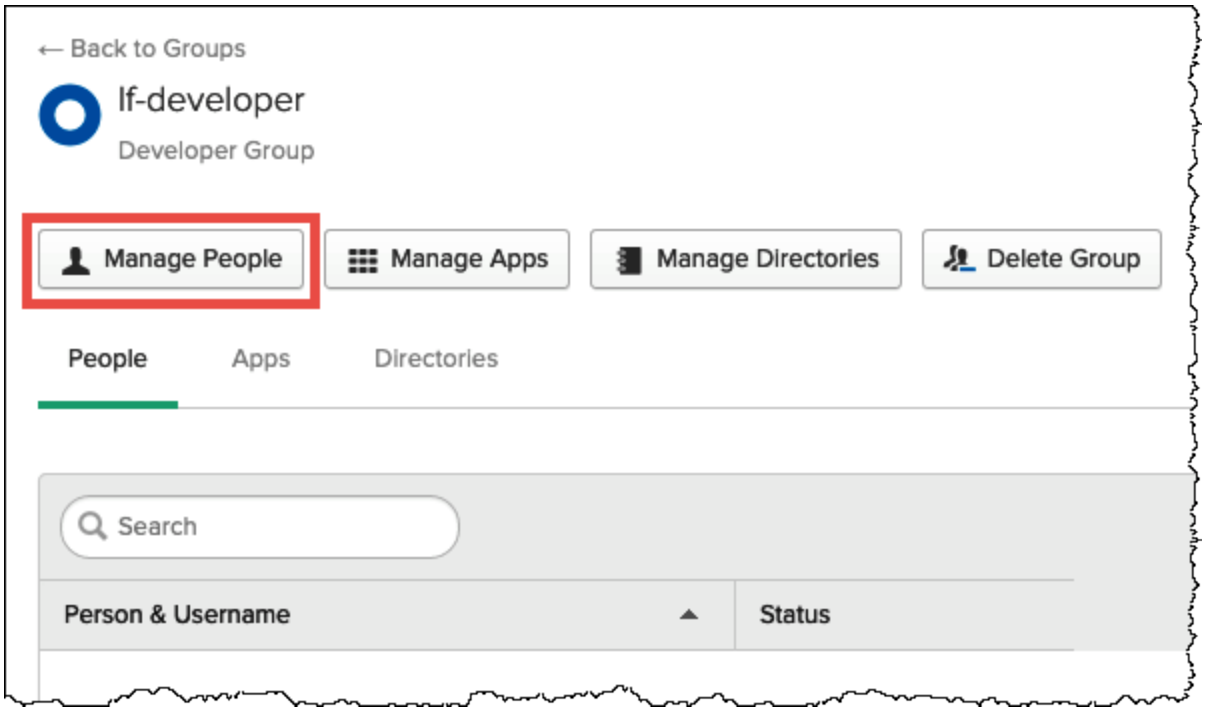
이제 두 사용자와 두 그룹이 있으므로 각 그룹에 사용자를 추가할 준비가 되었습니다.

그룹에 사용자를 추가하려면

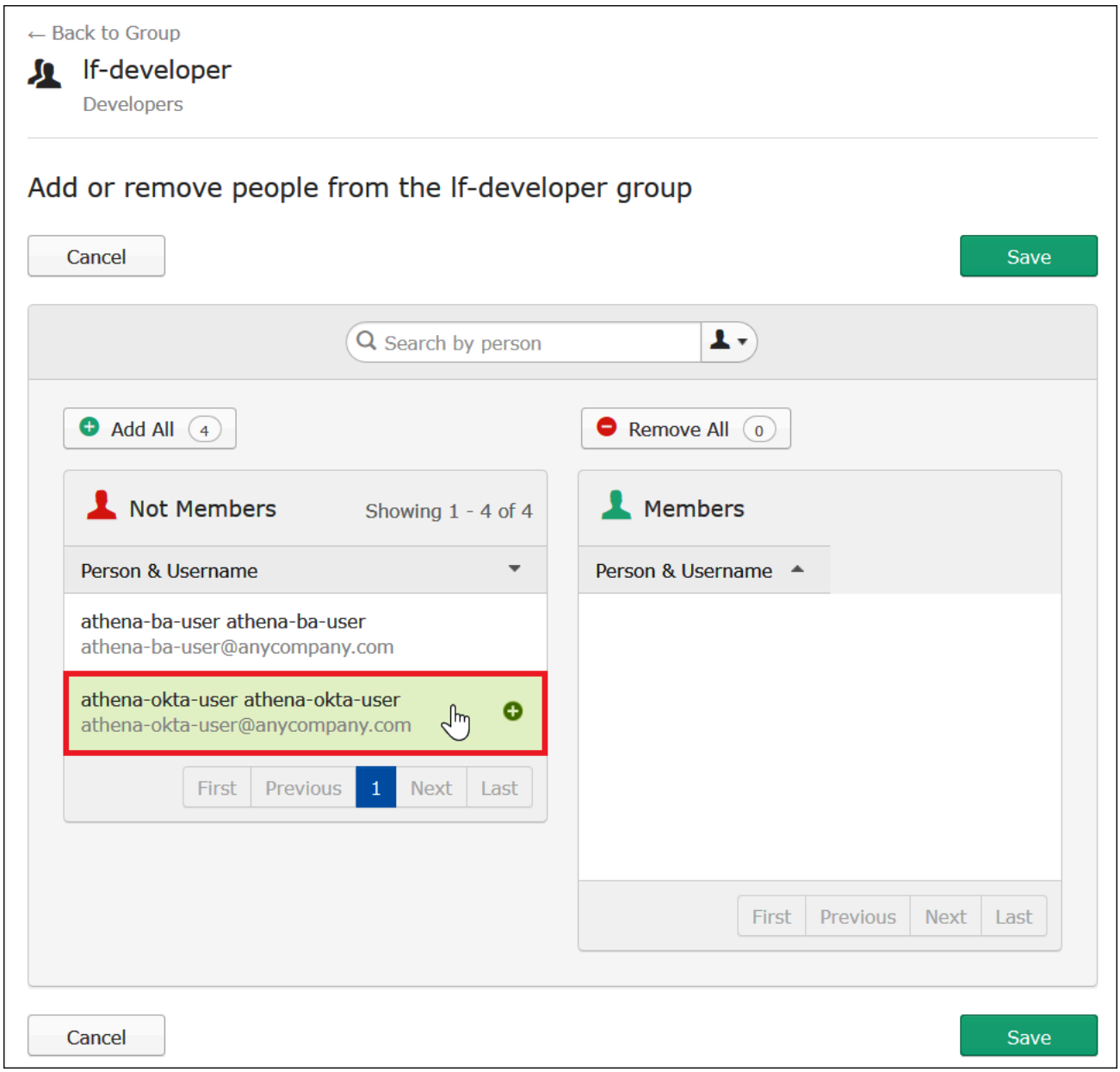
1. 그룹(Groups) 페이지에서 방금 생성한 lf-developer 그룹을 선택합니다. 개발자로 생성한 Okta 사용자 중 하나를 이 그룹에 추가합니다.



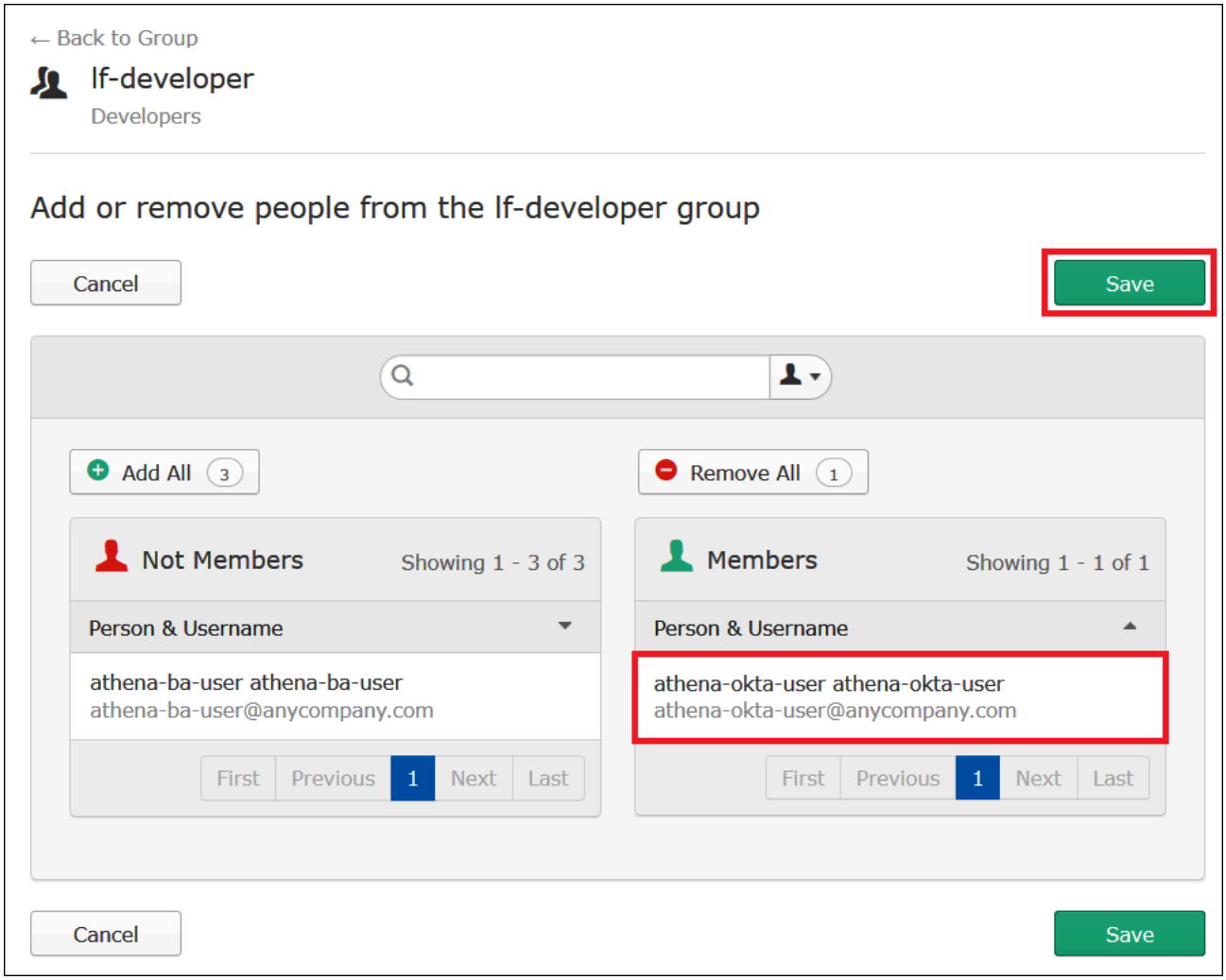
2. 사람 관리(Manage People)를 선택합니다.



3. 구성원 아님(Not Members) 목록에서 athena-okta-user를 선택합니다.

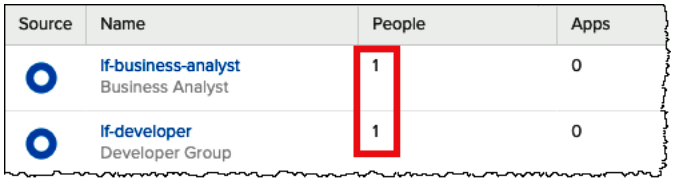


사용자에 대한 항목이 왼쪽에 있는 구성원 아님(Not Members) 목록에서 오른쪽의 구성원(Members) 목록으로 이동합니다.



4. Save(저장)를 선택합니다.
5. 그룹으로 돌아가기(Back to Group)를 선택하거나 디렉터리(Directory)를 선택한 다음 그룹(Groups)을 선택합니다.
6. lf-business-analyst 그룹을 선택합니다.
7. 사람 관리(Manage People)를 선택합니다.
8. athena-ba-user를 lf-business-analyst 그룹의 구성원 목록에 추가한 다음 저장(Save)을 선택합니다.
9. 그룹으로 돌아가기(Back to Group)를 선택하거나 디렉터리(Directory), 그룹(Groups)을 선택합니다.

   이제 그룹(Groups) 페이지에서 각 그룹에 한 명의 Okta 사용자가 있는 것을 볼 수 있습니다.



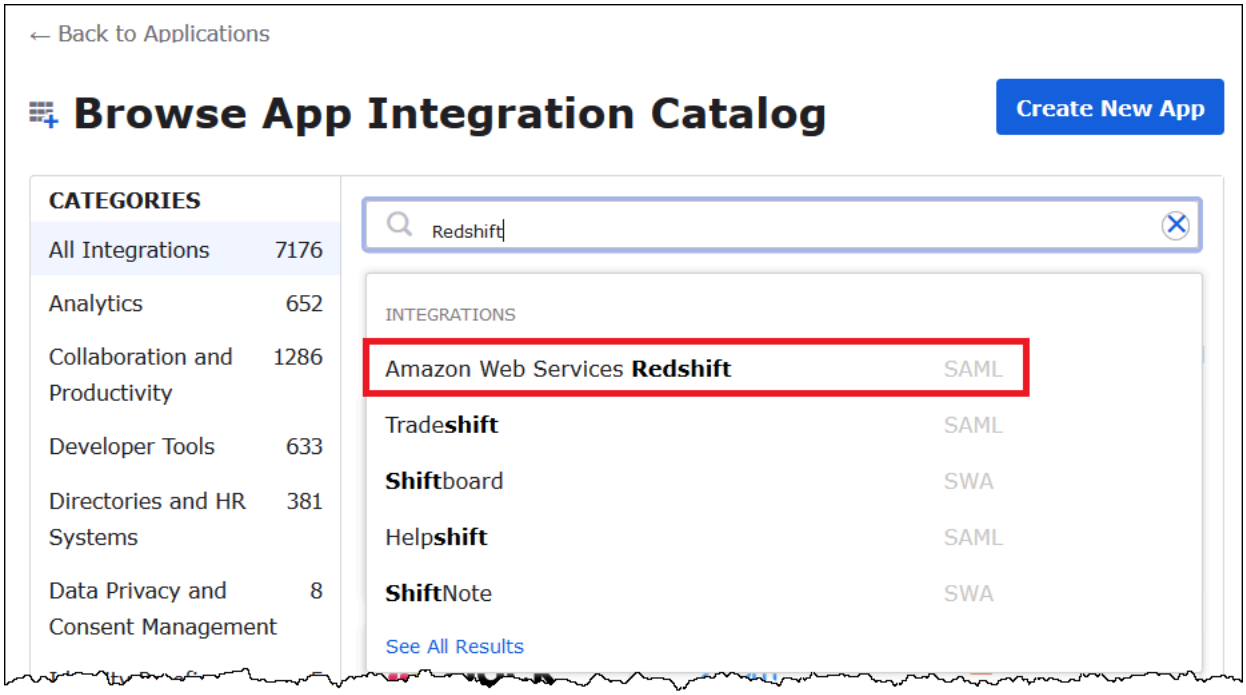
3단계: SAML 인증을 위한 Okta 애플리케이션 설정

이 단계에서는 Okta 개발자 콘솔을 사용하여 다음 작업을 수행합니다.

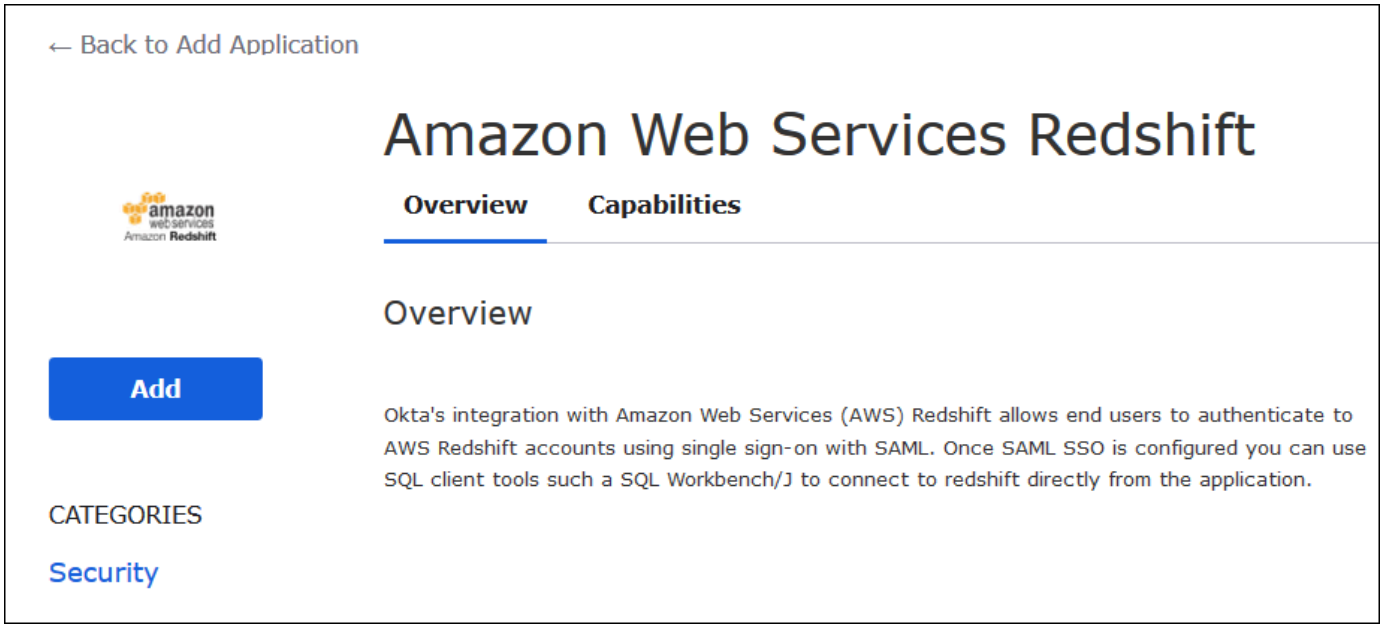
- AWS에 사용할 SAML 애플리케이션을 추가합니다.
- Okta 사용자에게 애플리케이션을 할당합니다.
- Okta 그룹에게 애플리케이션을 할당합니다.
- 나중에 AWS에 사용할 수 있도록 최종적인 자격 증명 공급자 메타데이터를 다운로드합니다.

SAML 인증을 위한 애플리케이션을 추가하려면

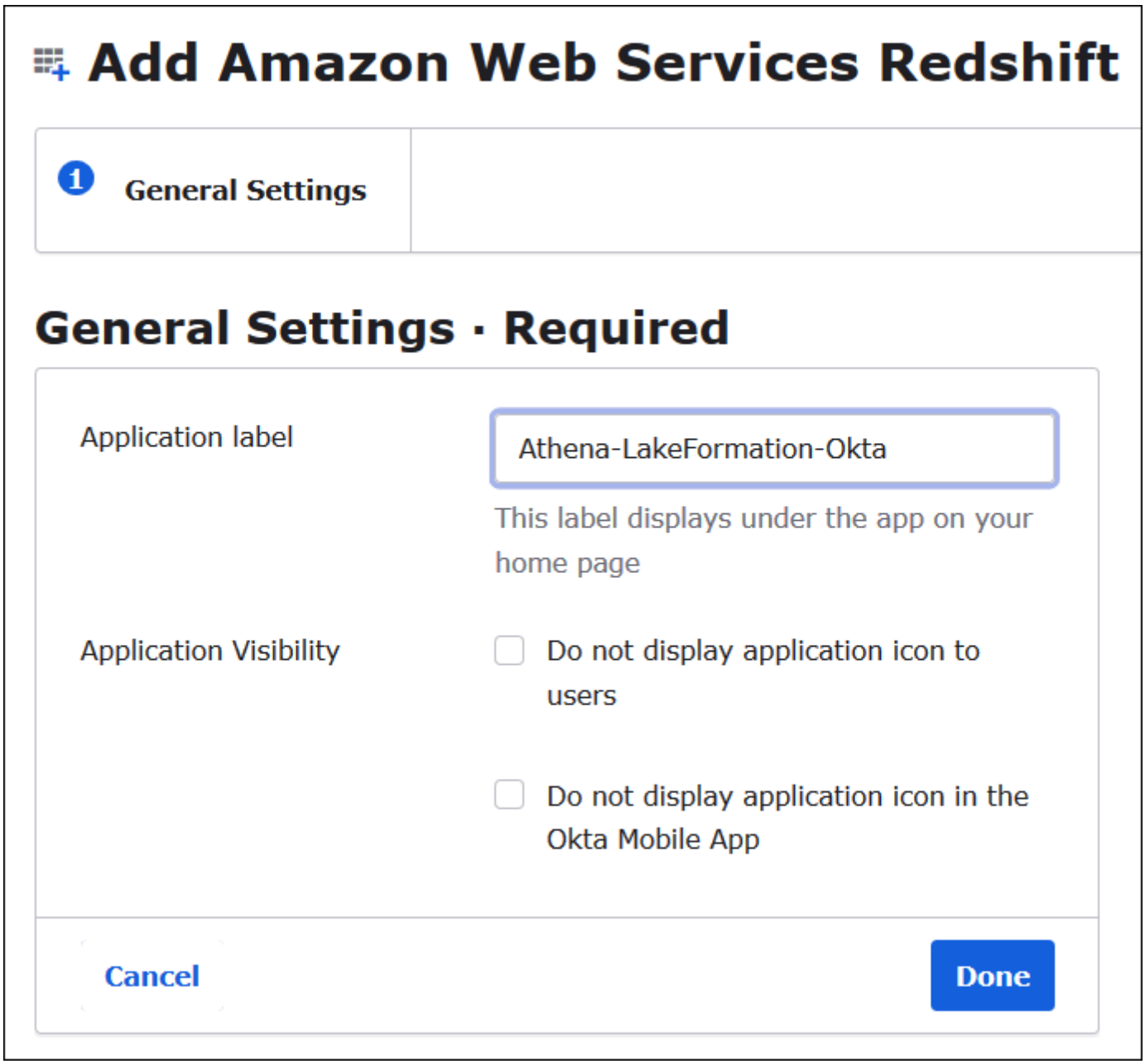
1. Okta 탐색 창에서 애플리케이션(Applications), 애플리케이션(Applications)을 선택하여 Athena에 대한 SAML 인증용 Okta 애플리케이션을 구성할 수 있습니다.
2. 앱 카탈로그 찾아보기(Browse App Catalog)를 클릭합니다.
3. 검색 상자에 **`Redshift`**을(를) 입력합니다.
4. Amazon Web Services Redshift를 선택합니다. 이 자습서의 Okta 애플리케이션은 Amazon Redshift를 위한 기존의 SAML 통합을 사용합니다.



5. Amazon Web Services Redshift 페이지에서 추가(Add)를 선택해 Amazon Redshift용 SAML 기반 애플리케이션을 생성합니다.



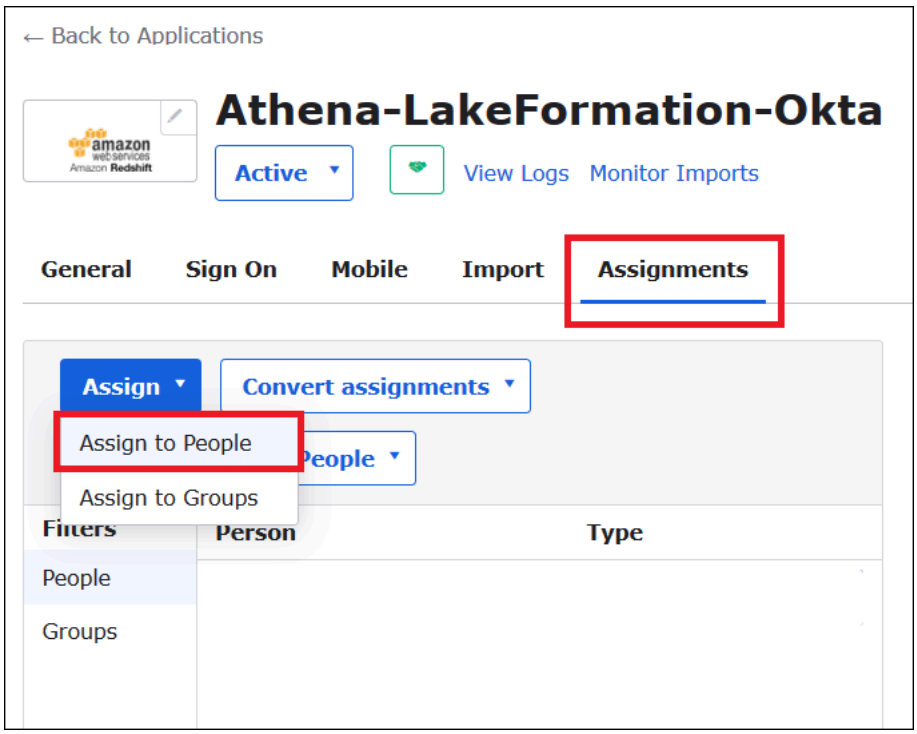
6. 애플리케이션 레이블(Application label)에 `Athena-LakeFormation-Okta`를 입력하고 완료(Done)를 선택합니다.



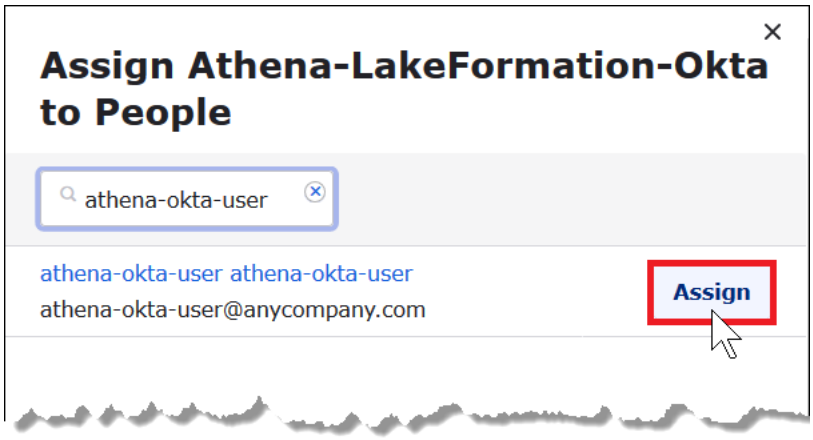
이제 Okta 애플리케이션을 생성되었고, 생성한 사용자 및 그룹에 이 애플리케이션을 할당할 수 있습니다.

애플리케이션을 사용자 및 그룹에 할당하려면

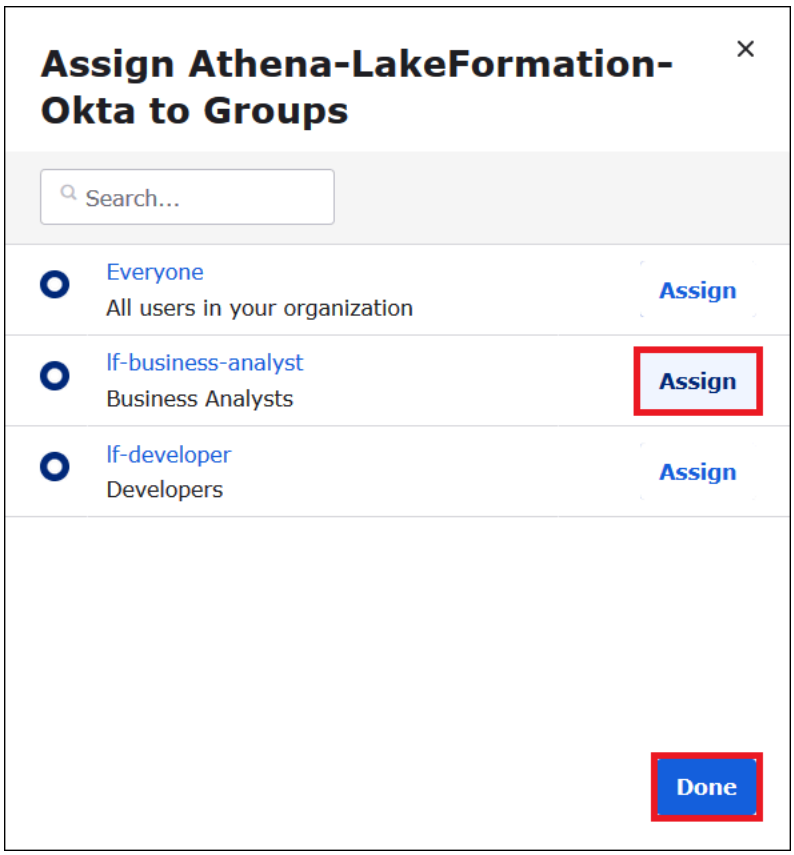
1. 애플리케이션(Applications) 페이지에서 Athena-LakeFormation-Okta 애플리케이션을 선택합니다.
2. 할당(Assignments) 탭에서 할당(Assign), 사람에게 할당(Assign to People)을 차례로 선택합니다.



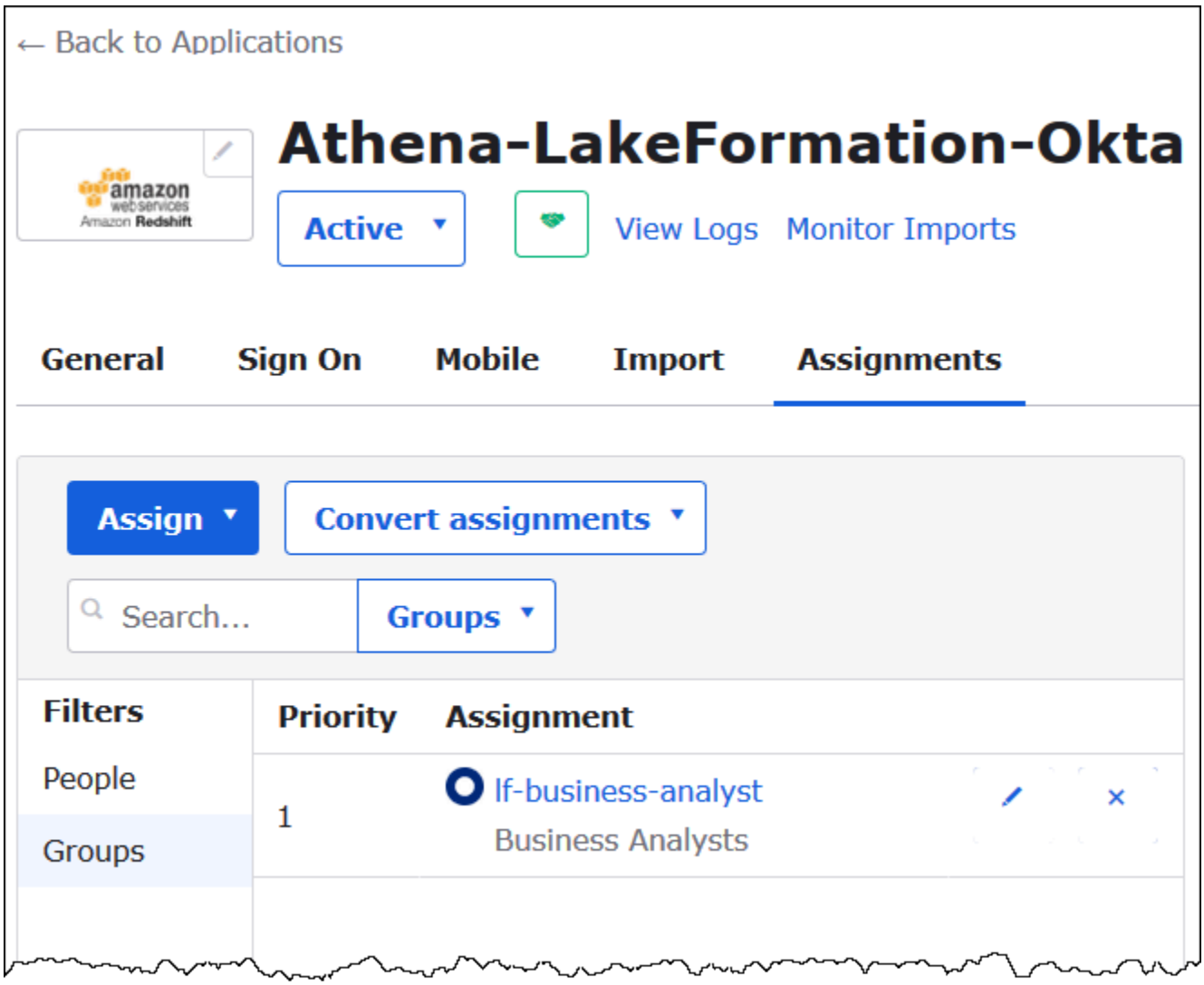
3. Athena-LakeFormation-Okta를 사람에게 할당(Assign Athena-LakeFormation-Okta to People) 대화 상자에서 이전에 생성했던 athena-okta-user 사용자를 찾습니다.
4. 할당(Assign)을 선택해 사용자를 애플리케이션에 할당합니다.



5. 저장하고 돌아가기(Save and Go Back)를 선택합니다.
6. 완료를 선택합니다.
7. Athena-LakeFormation-Okta 애플리케이션에 대한 할당(Assignments) 탭에서 할당(Assign), 그룹에게 할당(Assign to Groups)을 차례로 선택합니다.
8. lf-business-analyst에서 할당(Assign)을 선택해 Athena-LakeFormation-Okta 애플리케이션을 lf-business-analyst 그룹에 할당한 다음 완료(Done)를 선택합니다.



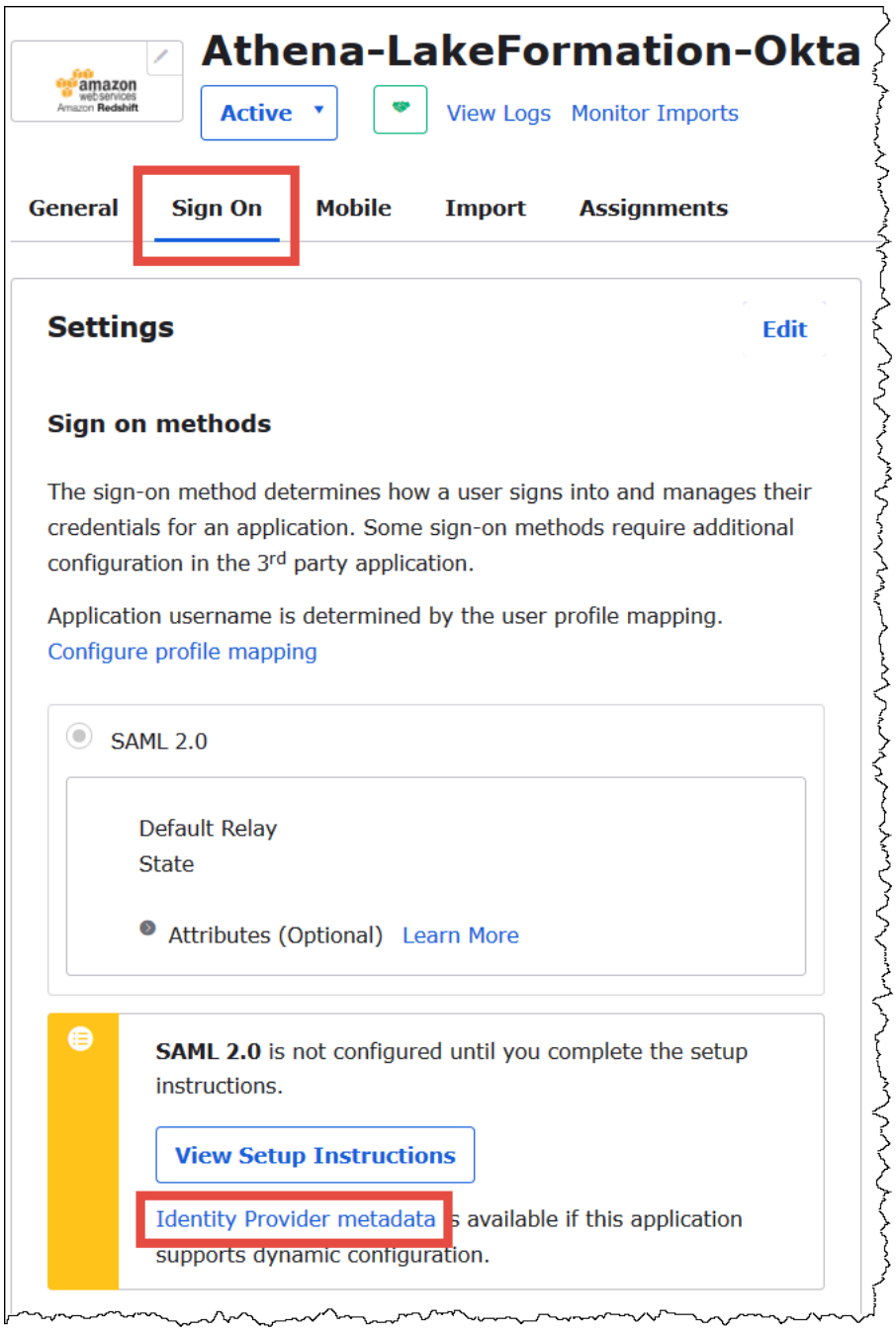
해당 그룹이 애플리케이션의 그룹 목록에 나타납니다.



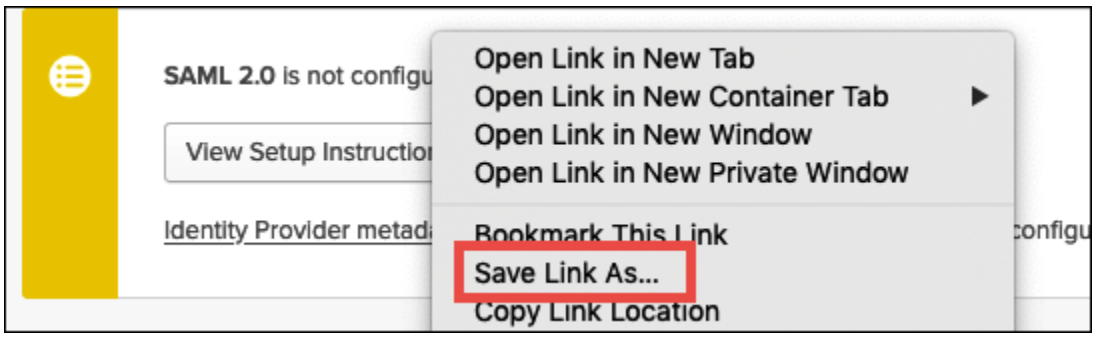
이제 AWS에 사용할 자격 증명 공급자 애플리케이션 메타데이터를 다운로드할 준비가 되었습니다..

애플리케이션 메타데이터를 다운로드하려면

1. Okta 애플리케이션 로그인(Sign On) 탭을 선택한 다음 자격 증명 공급자 메타데이터(Identity Provider metadata)를 마우스 오른쪽 버튼으로 클릭합니다.



2. 다른 이름으로 링크 저장(Save Link As)을 선택해 XML 형식의 자격 증명 공급자 메타데이터를 파일에 저장합니다. 알아볼 수 있는 이름을 지정합니다(예:`Athena-LakeFormation-idp-metadata.xml`).



4단계: AWS SAML 자격 증명 공급자 및 Lake Formation 액세스 IAM 역할 생성

이 단계에서는 AWS Identity and Access Management(IAM) 콘솔을 사용하여 다음 작업을 수행합니다.

- AWS에 대한 자격 증명 공급자를 생성합니다.
- Lake Formation 액세스를 위한 IAM 역할을 생성합니다.
- AmazonAthenaFullAccess 관리형 정책을 역할에 추가합니다.
- Lake Formation 및 AWS Glue에 대한 정책을 역할에 축가합니다.
- Athena 쿼리 결과에 대한 정책을 역할에 추가합니다.

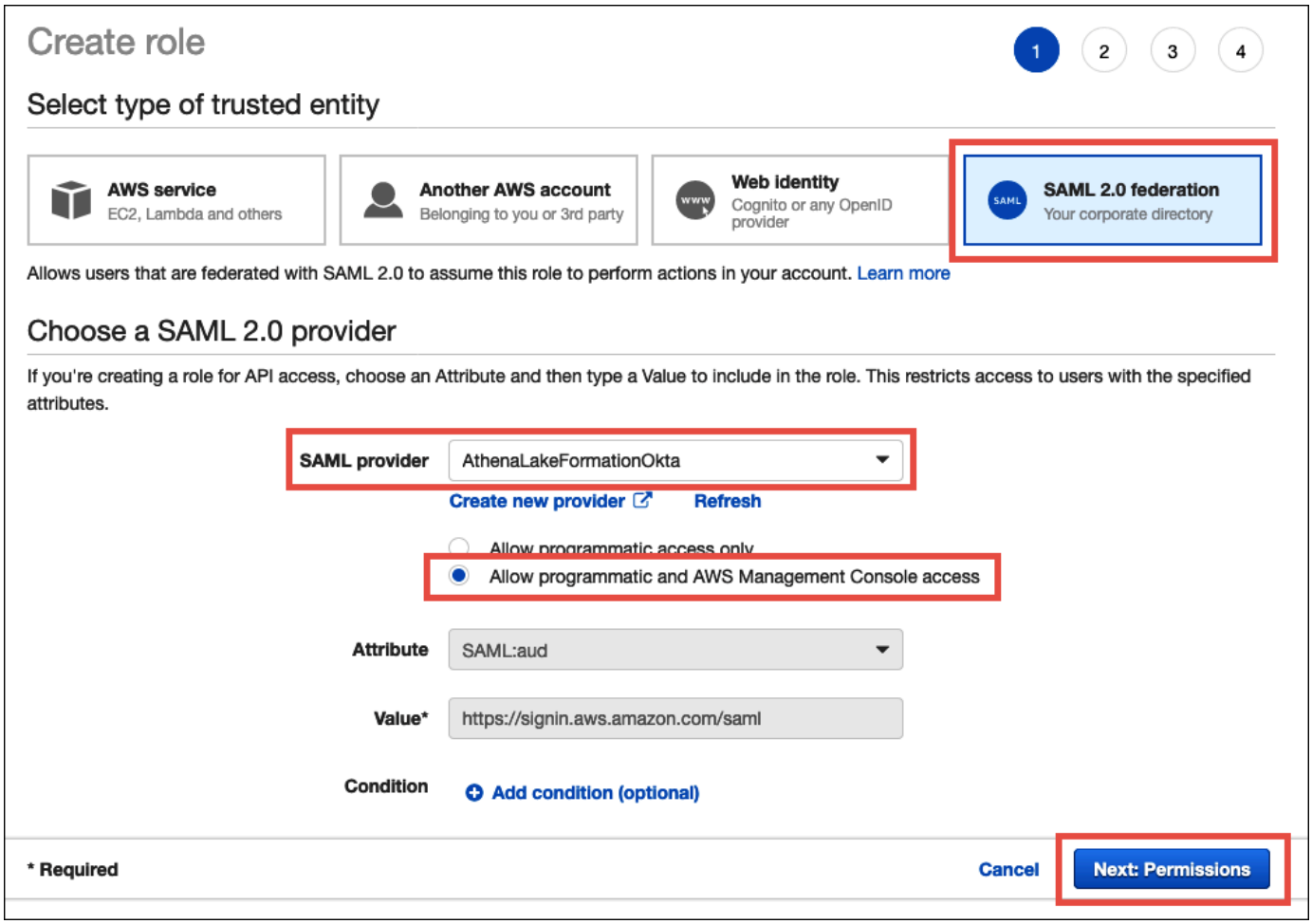
AWS SAML 자격 증명 공급자를 생성하려면

1. Amazon Web Services 계정 콘솔에 Amazon Web Services 계정 관리자로 로그인하고 IAM 콘솔(https://console.aws.amazon.com/iam/)을 탐색합니다.
2. 탐색 창에서 자격 증명 공급자(Identity providers)를 선택한 다음 공급자 추가(Add provider)를 클릭합니다.
3. 공급자 구성(Configure provider) 화면에 다음 정보를 입력합니다.
   - 공급자 유형(Provider type)에 SAML을 선택합니다.
   - 공급자 이름(Provider name)에 `AthenaLakeFormationOkta`를 입력합니다.
   - 메타데이터 문서(Metadata document)에서 파일 선택(Choose file) 옵션을 사용하여 다운로드한 자격 증명 공급자(IdP) 메타데이터 XML 파일을 업로드합니다.
4. 공급자 추가(Add Provider)를 선택합니다.

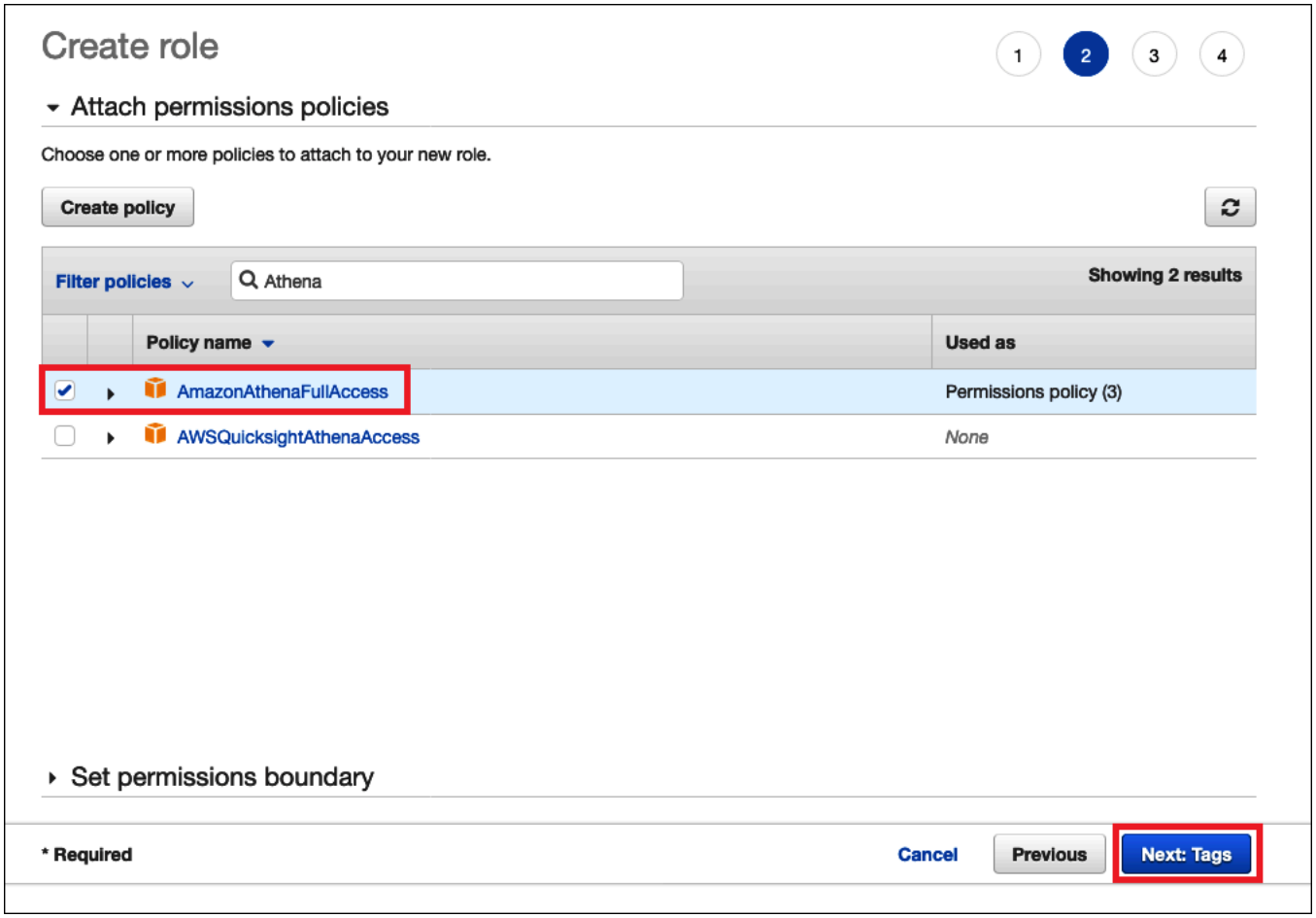
다음으로 AWS Lake Formation 액세스를 위한 IAM 역할을 생성합니다. 역할에 두 개의 인라인 정책을 추가합니다. 한 정책은 Lake Formation 및 AWS Glue API에 액세스할 수 있는 권한을 제공합니다. 다른 정책은 Athena와 Amazon S3의 Athena 쿼리 결과 위치에 대한 액세스 권한을 제공합니다.

AWS Lake Formation 액세스를 위한 IAM 역할을 생성하려면

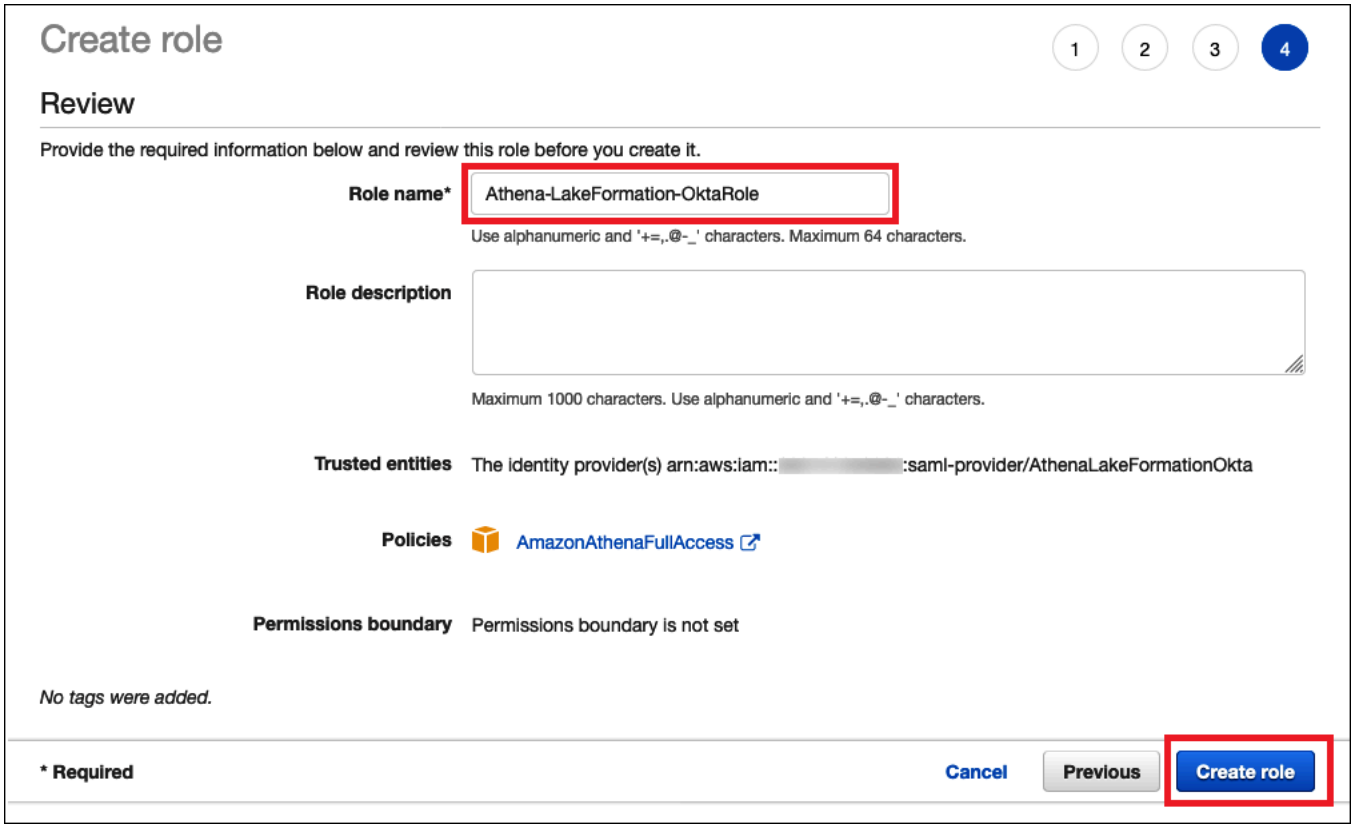
1. IAM 콘솔의 탐색 창에서 역할(Roles)을 선택하고 역할 생성(Create role)을 선택합니다.
2. 역할 생성(Create role) 페이지에서 다음 단계를 수행합니다.



   a. 신뢰할 수 있는 엔터티 유형 선택(Select type of trusted entity)에서 SAML 2.0 연합(SAML 2.0 Federation)을 선택합니다.
   b. SAML 공급자(SAML provider)에서 AthenaLakeFormationOkta를 선택합니다.
   c. SAML 공급자에서 프로그래밍 방식 및 AWS Management Console 액세스 허용 옵션을 선택합니다.
   d. 다음: 권한을 선택합니다.
3. 권한 정책 연결(Attach Permissions policies) 페이지에서 필터 정책(Filter policies)에 **Athena**를 입력합니다.
4. AmazonAthenaFullAccess 관리형 정책을 선택하고 다음: 태그(Next: Tags)를 선택합니다.



5. Add tags(태그 추가) 페이지에서 Next: Review(다음: 검토)를 선택합니다.
6. 검토(Review) 페이지에서 역할 이름(Role name)에 역할의 이름(예: *Athena-LakeFormation-OktaRole*)을 입력한 다음 역할 생성(Create role)을 선택합니다.



그런 다음 Lake Formation에 대한 액세스를 허용하는 인라인 정책, AWS Glue API, Amazon S3의 Athena 쿼리 결과를 추가합니다.

IAM 정책을 사용할 때마다 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

Lake Formation 및 AWS Glue에 대한 역할에 인라인 정책을 추가하려면

1. IAM 콘솔의 역할 목록에서 새로 생성된 `Athena-LakeFormation-OktaRole`을 선택합니다.
2. 역할에 대한 요약(Summary) 페이지의 권한(Permissions) 탭에서 인라인 정책 추가(Add inline policy)를 선택합니다.
3. 정책 생성 페이지에서 JSON을 선택합니다.
4. Lake Formation 및 AWS Glue API에 대한 액세스 권한을 제공하는 다음과 같은 인라인 정책을 추가합니다.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
```

```
        "Action": [
            "lakeformation:GetDataAccess",
            "glue:GetTable",
            "glue:GetTables",
            "glue:GetDatabase",
            "glue:GetDatabases",
            "glue:CreateDatabase",
            "glue:GetUserDefinedFunction",
            "glue:GetUserDefinedFunctions"
        ],
        "Resource": "*"
    }
}
```

5. 정책 검토를 선택합니다.
6. 이름(Name)에 정책의 이름을 입력합니다(예: **LakeFormationGlueInlinePolicy**).
7. 정책 생성(Create policy)을 선택합니다.

Athena 쿼리 결과 위치에 대한 역할에 인라인 정책을 추가하려면

1. `Athena-LakeFormation-OktaRole` 역할에 대한 요약(Summary) 페이지의 권한(Permissions) 탭에서 인라인 정책 추가(Add inline policy)를 선택합니다.
2. 정책 생성 페이지에서 JSON을 선택합니다.
3. Athena 쿼리 결과 위치에 대한 역할 액세스를 허용하는 다음과 같은 인라인 정책을 추가합니다. 예제의 *<athena-query-results-bucket>* 자리 표시자를 해당 Amazon S3 버킷의 이름으로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AthenaQueryResultsPermissionsForS3",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::<athena-query-results-bucket>",
```

```
                "arn:aws:s3:::<athena-query-results-bucket>/*"
            ]
        }
    ]
}
```

4. 정책 검토를 선택합니다.
5. 이름(Name)에 정책의 이름을 입력합니다(예: **AthenaQueryResultsInlinePolicy**).
6. 정책 생성(Create policy)을 선택합니다.

그런 다음 Lake Formation 액세스 역할의 ARN과 생성한 SAML 공급자의 ARN을 복사합니다. 이 정보는 자습서의 다음 단원에서 Okta SAML 애플리케이션을 구성할 때 필요합니다.

역할 ARN 및 SAML 자격 증명 공급자 ARN을 복사하려면

1. IAM 콘솔의 요약(Summary) 페이지에서 `Athena-LakeFormation-OktaRole` 역할에 대해 역할 ARN(Role ARN) 옆에 있는 클립보드로 복사(Copy to clipboard)를 선택합니다. ARN의 형식은 다음과 같습니다.

```
arn:aws:iam::<account-id>:role/Athena-LakeFormation-OktaRole
```

2. 나중에 참조할 수 있도록 전체 ARN을 안전하게 저장합니다.
3. IAM 콘솔 탐색 창에서 자격 증명 공급자(Identity providers)를 선택합니다.
4. AthenaLakeFormationOkta 공급자를 선택합니다.
5. 요약(Summary) 페이지에서 공급자 ARN(Provider ARN) 옆의 클립보드로 복사(Copy to clipboard) 아이콘을 선택합니다. ARN은 다음과 같은 양식이어야 합니다.

```
arn:aws:iam::<account-id>:saml-provider/AthenaLakeFormationOkta
```

6. 나중에 참조할 수 있도록 전체 ARN을 안전하게 저장합니다.

5단계: Okta 애플리케이션에 IAM 역할 및 SAML 자격 증명 공급자 추가

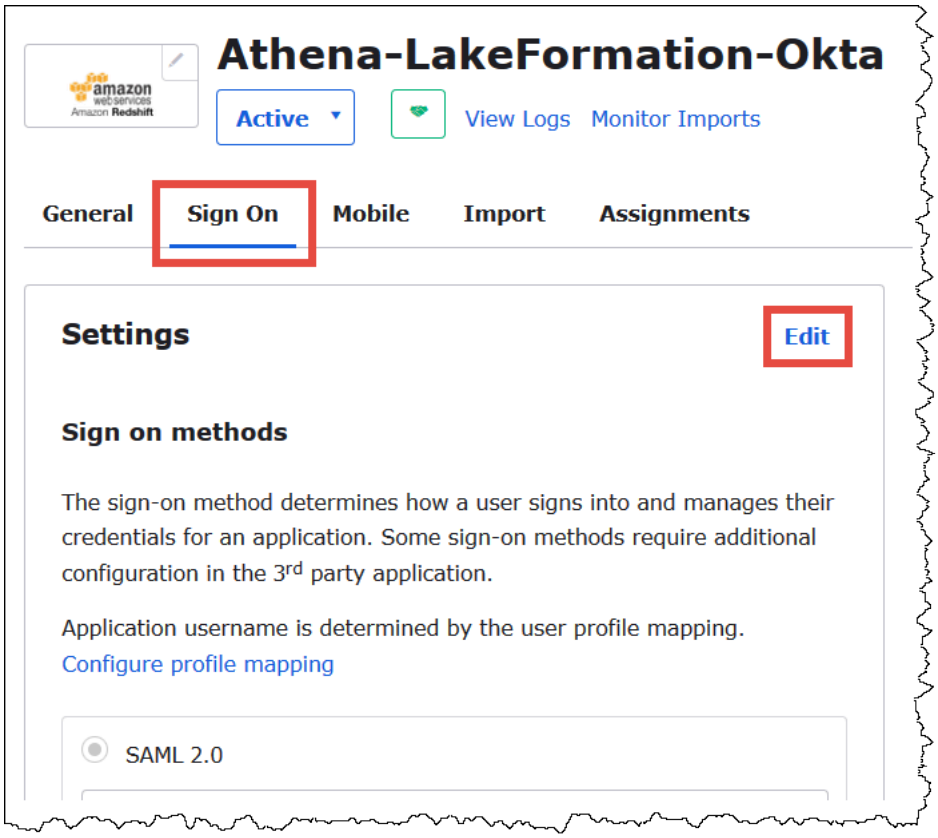
이 단계에서는 Okta 개발자 콘솔로 돌아가서 다음 작업을 수행합니다.

- Okta 애플리케이션에 사용자 및 그룹 Lake Formation URL 속성을 추가합니다.
- 자격 증명 공급자의 ARN과 IAM 역할의 ARN을 Okta 애플리케이션에 추가합니다.

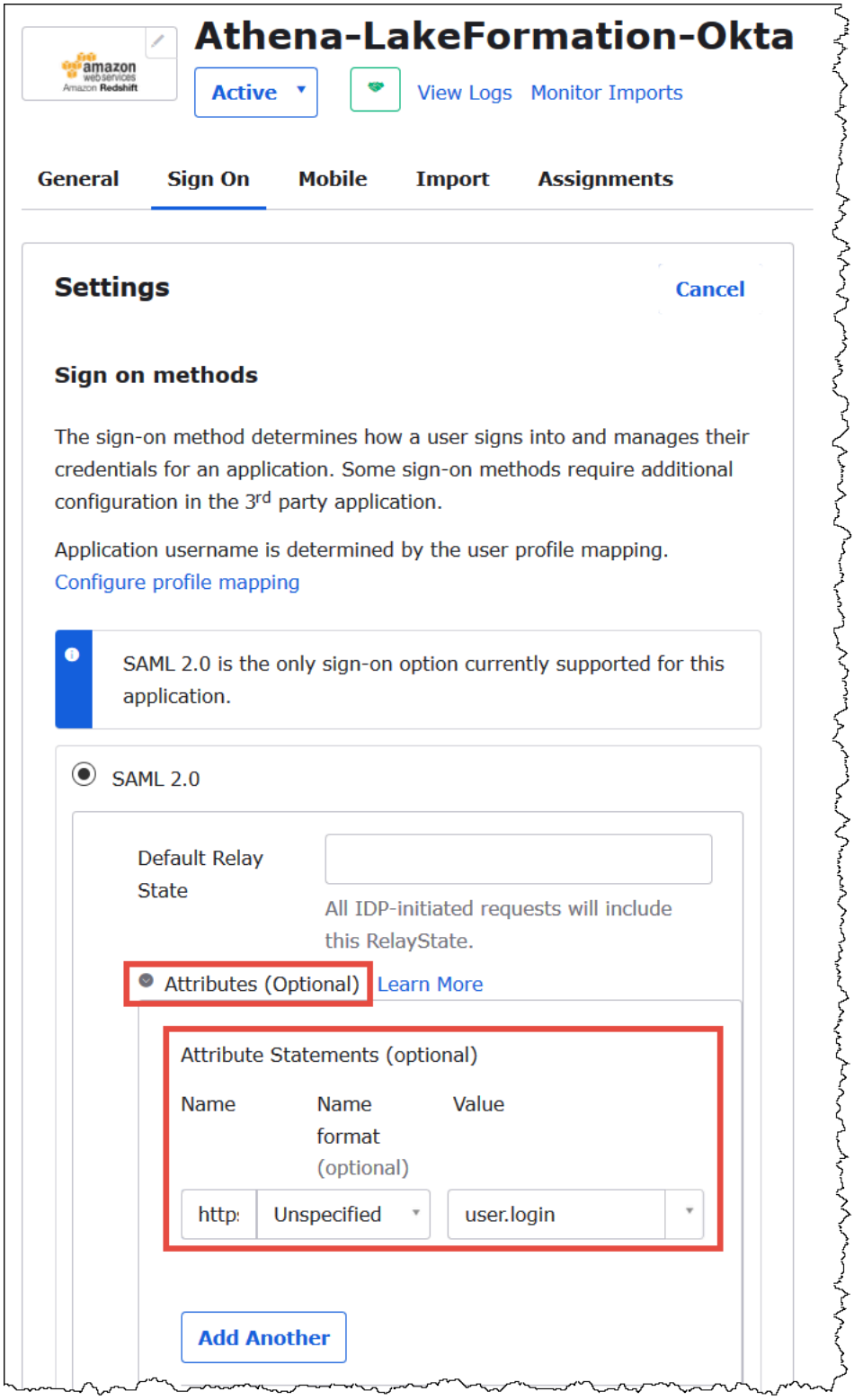
- Okta 애플리케이션 ID를 복사합니다. Okta 애플리케이션 ID는 Athena에 연결하는 JDBC 프로파일에 필요합니다.

Okta 애플리케이션에 사용자 및 그룹 Lake Formation URL 속성을 추가하려면

1. Okta 개발자 콘솔에 로그인합니다.
2. 애플리케이션(Applications) 탭을 선택한 다음 `Athena-LakeFormation-Okta` 애플리케이션을 선택합니다.
3. 애플리케이션에 대한 로그인(Sign On) 탭을 선택한 다음 편집(Edit)을 선택합니다.



4. 속성(선택 사항)(Attributes (optional))을 클릭하여 확장합니다.



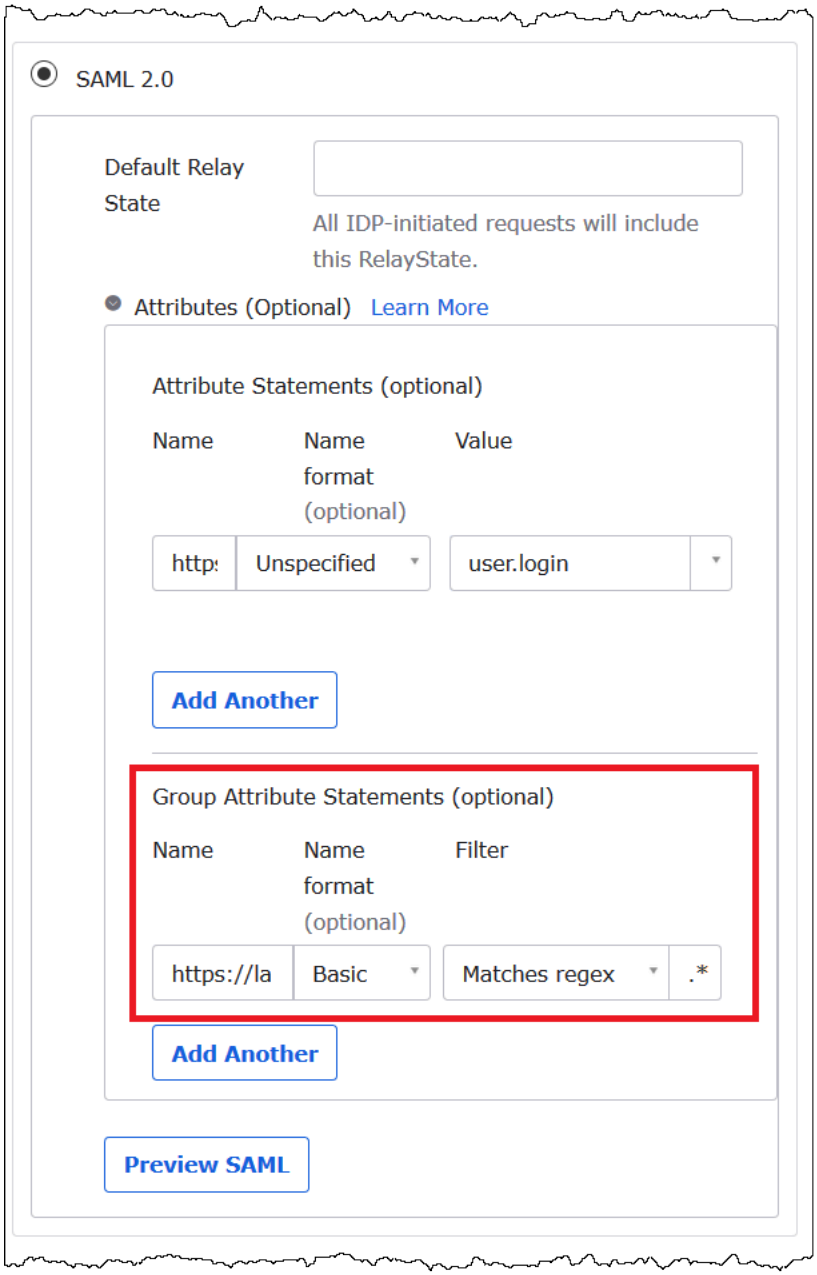
5. 속성 문(선택 사항)(Attribute Statements (optional))에 다음 속성을 추가합니다.

   - 이름에 **`https://lakeformation.amazon.com/SAML/Attributes/Username`**를 입력합니다.

- [값(Value)]에 **`user.login`**을 입력합니다.

6. 그룹 속성 문(선택 사항)(Group Attribute Statements (optional))에 다음 속성을 추가합니다.

   - 이름에 **`https://lakeformation.amazon.com/SAML/Attributes/Groups`**를 입력합니다.
   - 이름 형식(Name format)에 **`Basic`**을 입력합니다.
   - 필터(Filter)에서 정규식 일치(Matches regex)를 선택한 다음 필터 상자에 **`.*`**을 입력합니다.

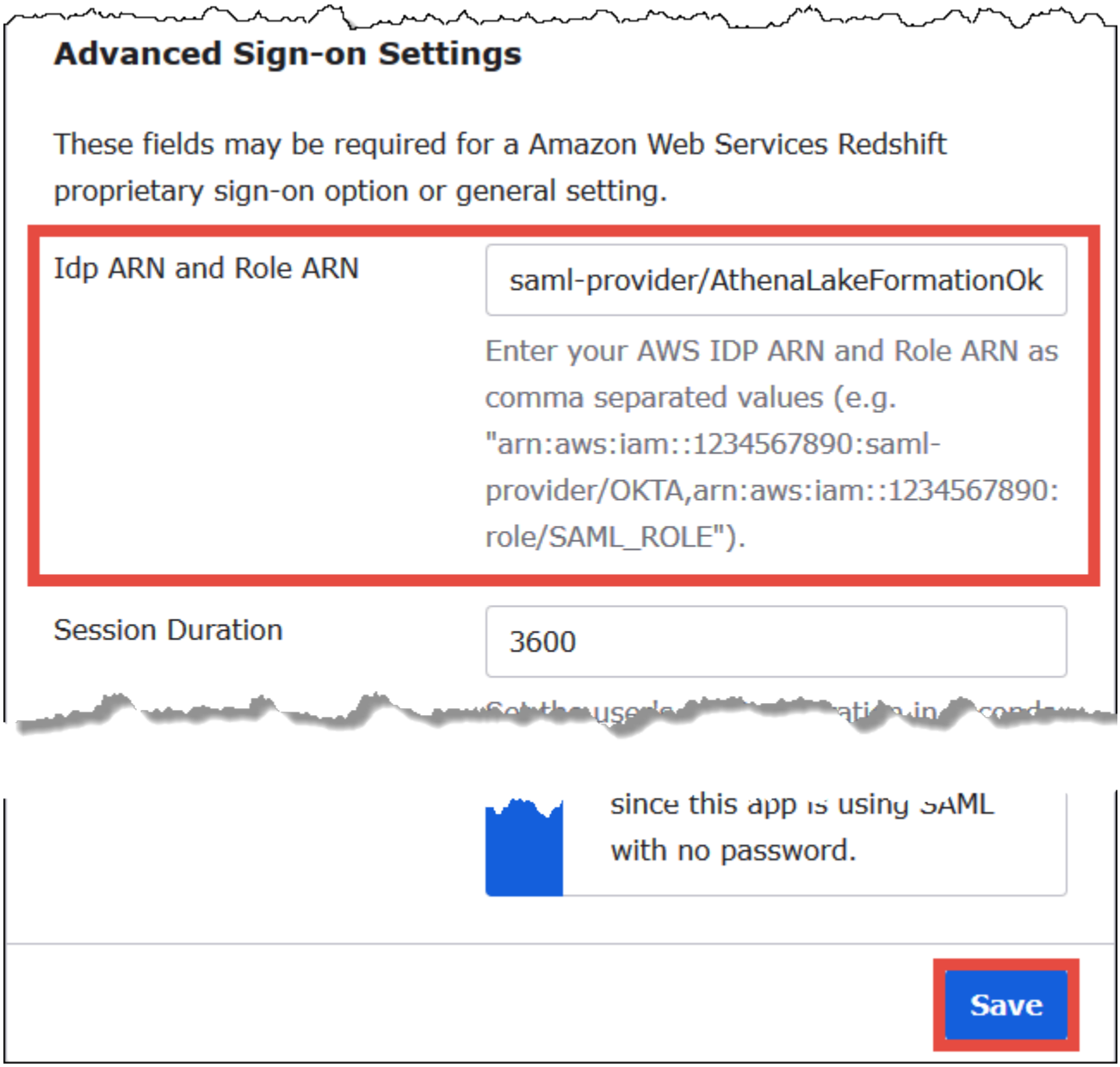


7. 고급 로그인 설정(Advanced Sign-On Settings) 섹션까지 아래로 스크롤하고, 여기서 자격 증명 공급자 및 IAM 역할 ARN을 Okta 애플리케이션에 추가합니다.

자격 증명 공급자 및 IAM 역할의 ARN을 Okta 애플리케이션에 추가하려면

1. Idp ARN 및 역할 ARN(Idp ARN and Role ARN)에 AWS 자격 증명 공급자 ARN과 역할 ARN을 *<saml-arn>*,*<role-arn>* 형식의 쉼표로 구분된 값으로 입력합니다. 결합된 문자열은 다음과 같아야 합니다.

```
arn:aws:iam::<account-id>:saml-provider/
AthenaLakeFormationOkta,arn:aws:iam::<account-id>:role/Athena-LakeFormation-
OktaRole
```

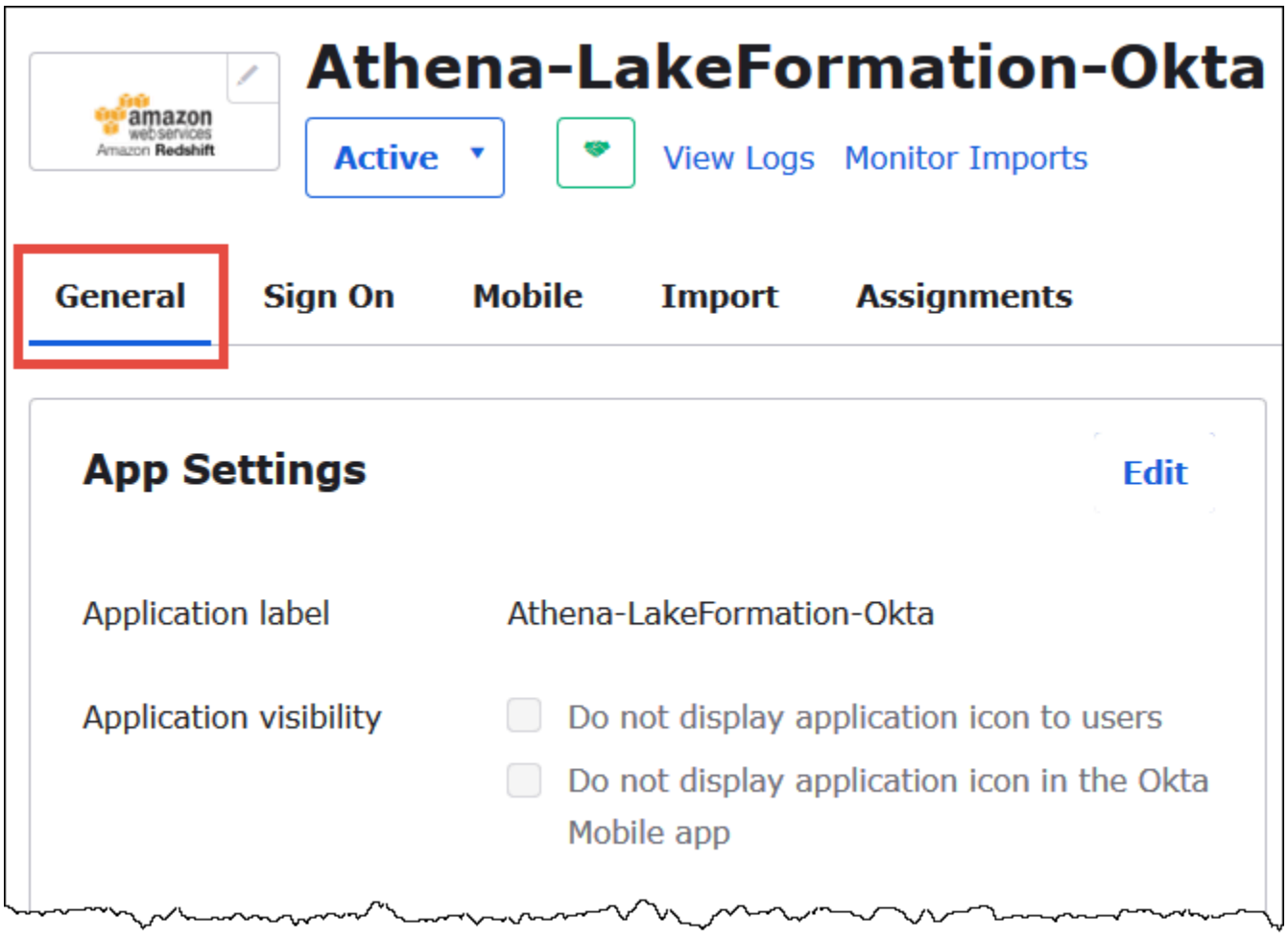


2. Save(저장)를 선택합니다.

다음으로 Okta 애플리케이션 ID를 복사합니다. 나중에 Athena에 연결하는 JDBC 문자열에 이 정보가 필요합니다.

Okta 애플리케이션 ID를 찾고 복사하려면

1. Okta 애플리케이션의 일반(General) 탭을 선택합니다.



2. 아래로 스크롤하여 앱 포함 링크(App Embed Link) 섹션으로 이동합니다.
3. 포함 링크(Embed Link)에서 URL의 Okta 애플리케이션 ID 부분을 복사하고 안전하게 저장합니다. Okta 애플리케이션 ID는 URL에서 `amazon_aws_redshift/` 뒤부터 다음 슬래시 앞까지의 부분입니다. 예를 들어, URL에 `amazon_aws_redshift/aaa/bbb`가 포함된 경우 애플리케이션 ID는 `aaa`입니다.

> **Note**
>
> 포함 링크(Embed link)를 사용하여 Athena 콘솔에 직접 로그인하여 데이터베이스를 볼 수 없습니다. SAML 사용자 및 그룹에 대한 Lake Formation 권한은 JDBC 또는 ODBC 드라이버를 사용하여 Athena 쿼리를 제출할 때만 인식됩니다. JDBC 드라이버를 사용하여 Athena에 연결하는 SQL Workbench/J 도구를 사용하여 데이터베이스를 볼 수 있습니다. SQL Workbench/J 도구는 7단계: Athena JDBC 클라이언트를 통해 액세스 권한 확인에서 다룹니다.

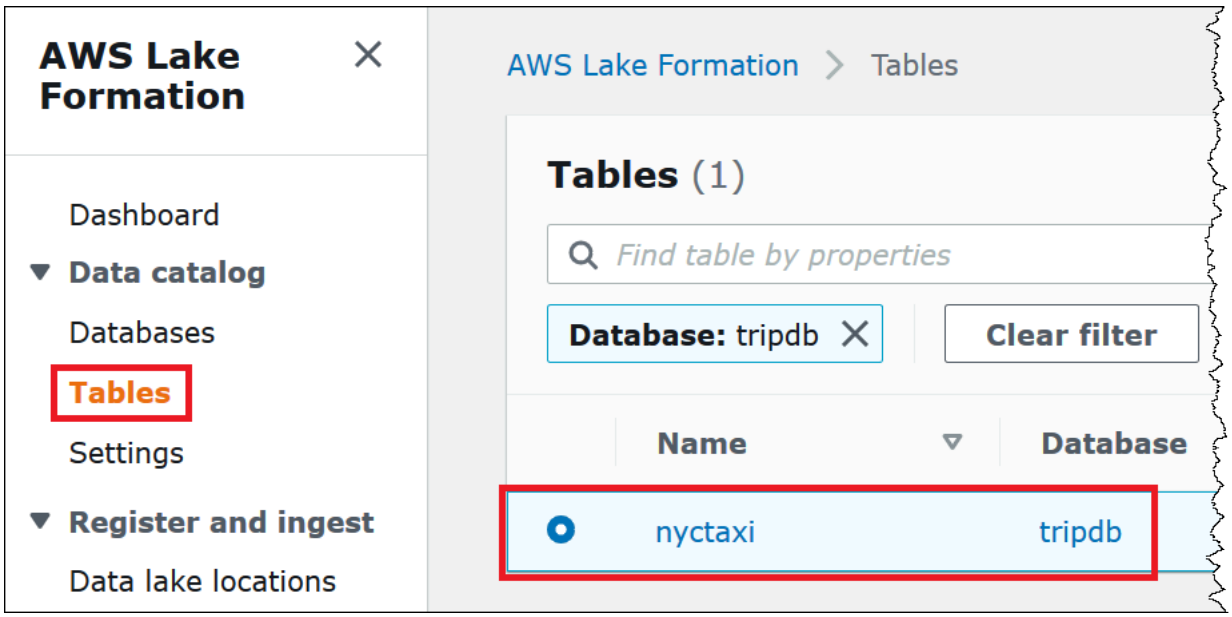
6단계: AWS Lake Formation을(를) 통해 사용자 및 그룹 권한 부여

이 단계에서는 Lake Fomation 콘솔을 사용하여 SAML 사용자 및 그룹에게 테이블에 대한 권한을 부여합니다. 다음 작업을 수행할 수 있습니다.

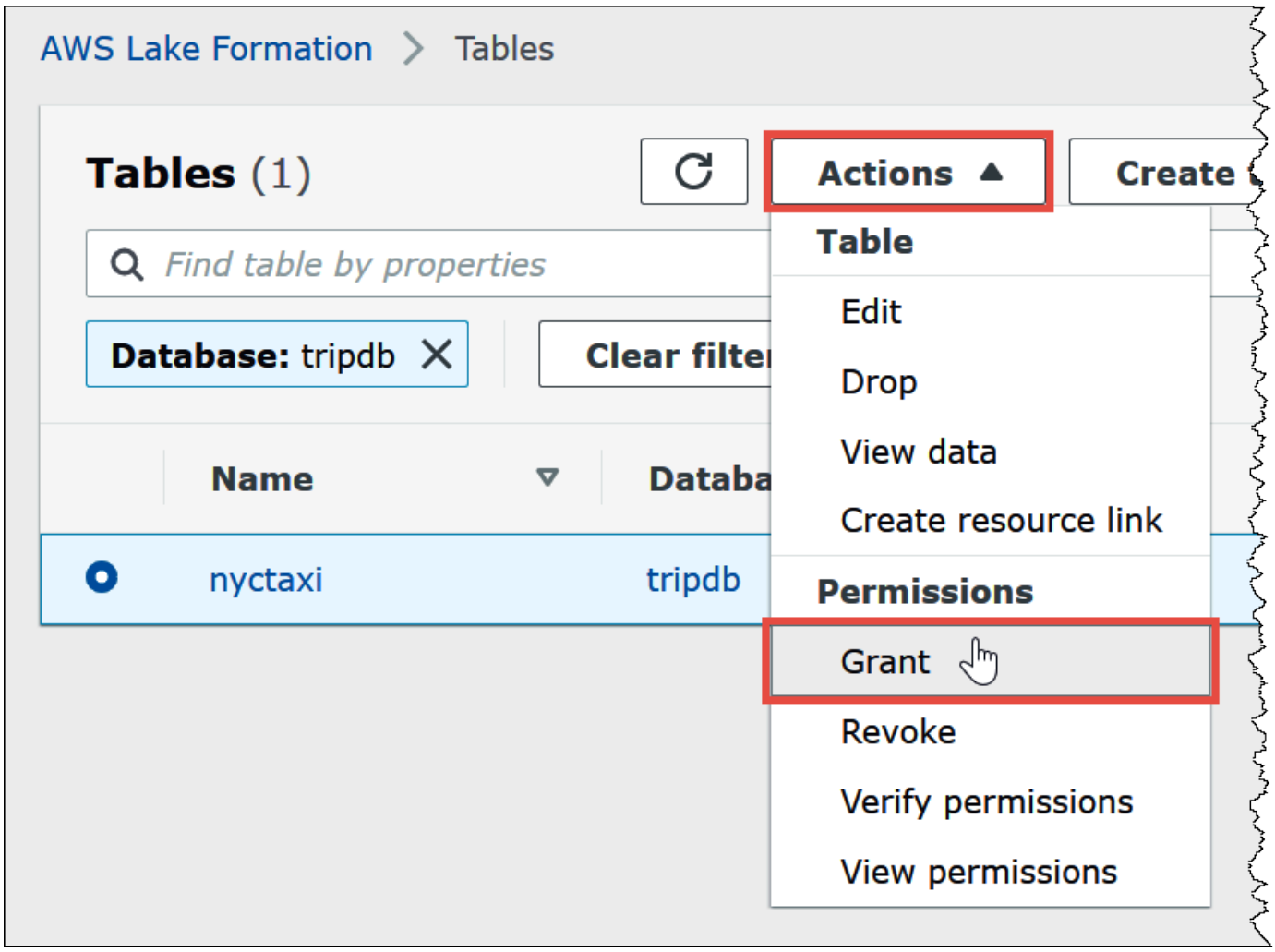
- Okta SAML 사용자의 ARN 및 테이블에 대한 연결된 사용자 권한을 지정합니다.
- Okta SAML 사용자의 ARN 및 테이블에 대한 연결된 사용자 권한을 지정합니다.
- 부여한 권한을 확인합니다.

Lake Formation에서 Okta 사용자에 대해 권한을 부여하려면

1. 데이터 레이크 관리자로 AWS Management Console에 로그인합니다.
2. Lake Formation 콘솔(https://console.aws.amazon.com/lakeformation/)을 엽니다.
3. 탐색 창에서 테이블(Tables)을 선택한 다음 권한을 부여할 테이블을 선택합니다. 이 자습서에서는 `tripdb` 데이터베이스의 `nyctaxi` 테이블을 사용합니다.



4. 작업(Actions)에서 권한 부여(Grant)를 선택합니다.



5. 권한 부여(Grant permissions) 대화 상자에 다음 정보를 입력합니다.

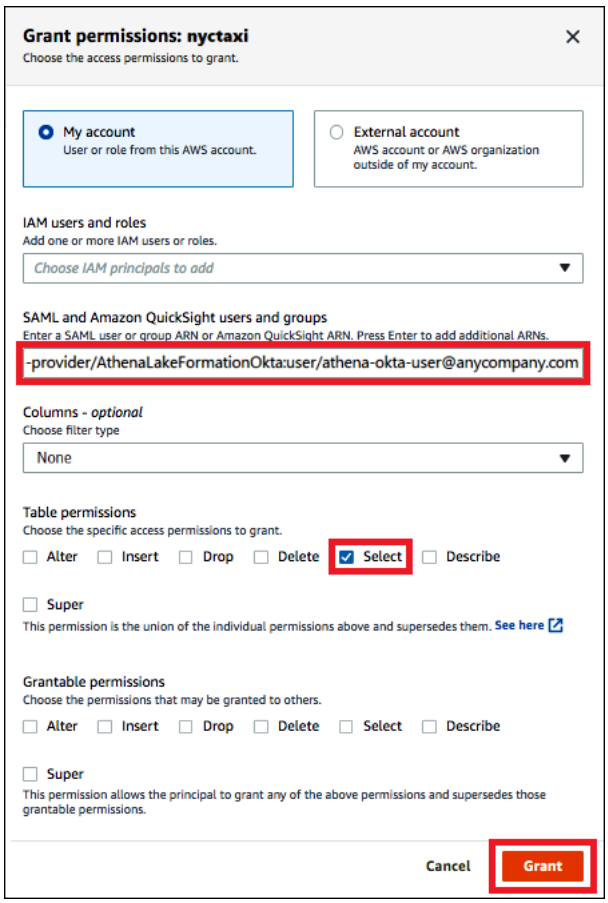
   a. SAML 및 Amazon QuickSight 사용자 및 그룹(SAML and Amazon QuickSight users and groups)에서 다음 형식으로 Okta SAML 사용자 ARN을 입력합니다.

   ```
   arn:aws:iam::<account-id>:saml-provider/AthenaLakeFormationOkta:user/<athena-okta-user>@<anycompany.com>
   ```

   b. 열(Columns)에서 필터 유형을 선택하고, 열 포함(Include columns) 또는 열 제외(Exclude columns) 옵션을 선택합니다.

   c. 필터 아래의 하나 이상의 열 선택(Choose one or more columns) 드롭다운을 사용하여 사용자에 대해 포함하거나 제외할 열을 지정합니다.

   d. 테이블 권한(Table permissions)에서 선택(Select)을 선택합니다. 이 자습서에서는 SELECT 권한만 부여하며, 각자의 요구 사항은 다를 수 있습니다.



6. 권한 부여(Grant)를 선택합니다.

이제 Okta 그룹에 대해 비슷한 단계를 수행합니다.

Lake Formation에서 Okta 사용자에 대해 권한을 부여하려면

1. Lake Formation 콘솔의 테이블(Tables) 페이지에서 nyctaxi 테이블이 계속 선택되어 있는지 확인합니다.
2. 작업(Actions)에서 권한 부여(Grant)를 선택합니다.
3. 권한 부여(Grant permissions) 대화 상자에 다음 정보를 입력합니다.

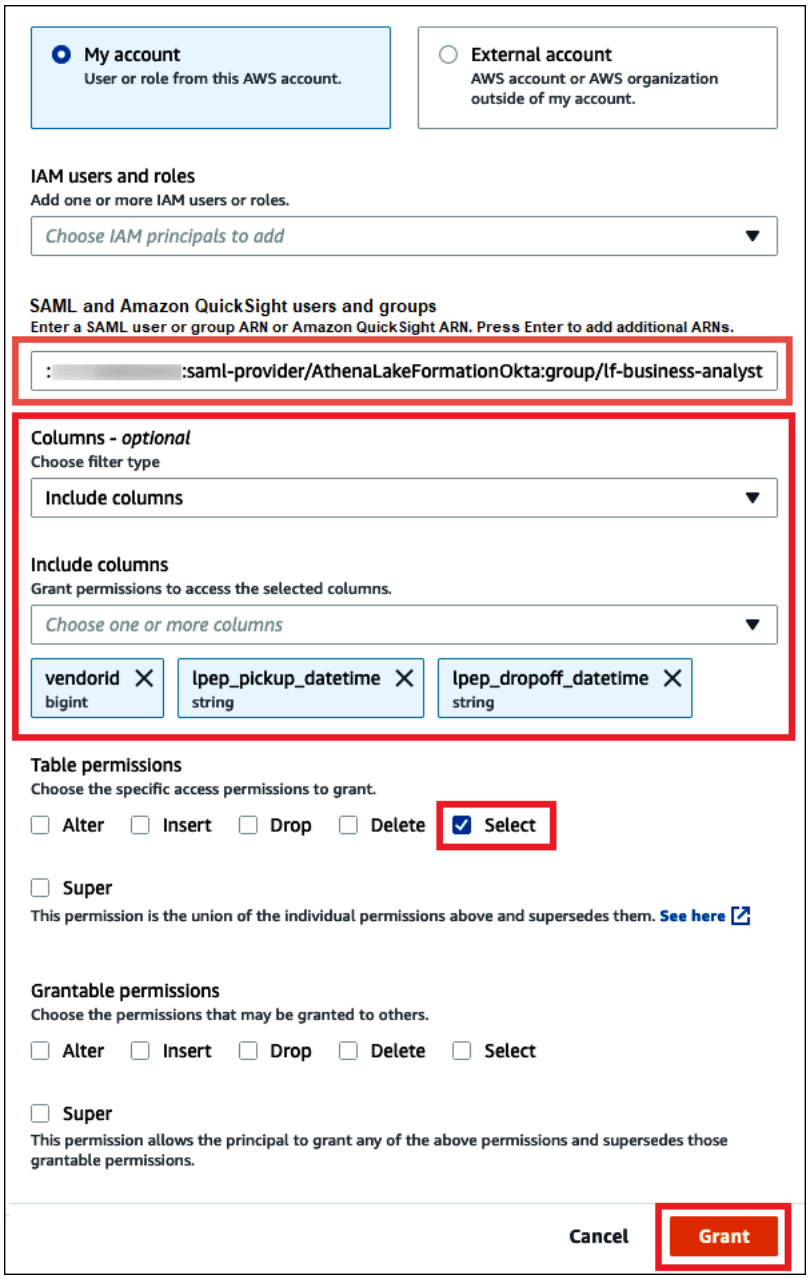
   a. SAML 및 Amazon QuickSight 사용자 및 그룹(SAML and Amazon QuickSight users and groups)에서 다음 형식으로 Okta SAML 그룹 ARN을 입력합니다.

   ```
   arn:aws:iam::<account-id>:saml-provider/AthenaLakeFormationOkta:group/lf-business-analyst
   ```

   b. 열(Columns)에서 필터 유형을 선택하고, 열 포함(Include columns)을 선택합니다.

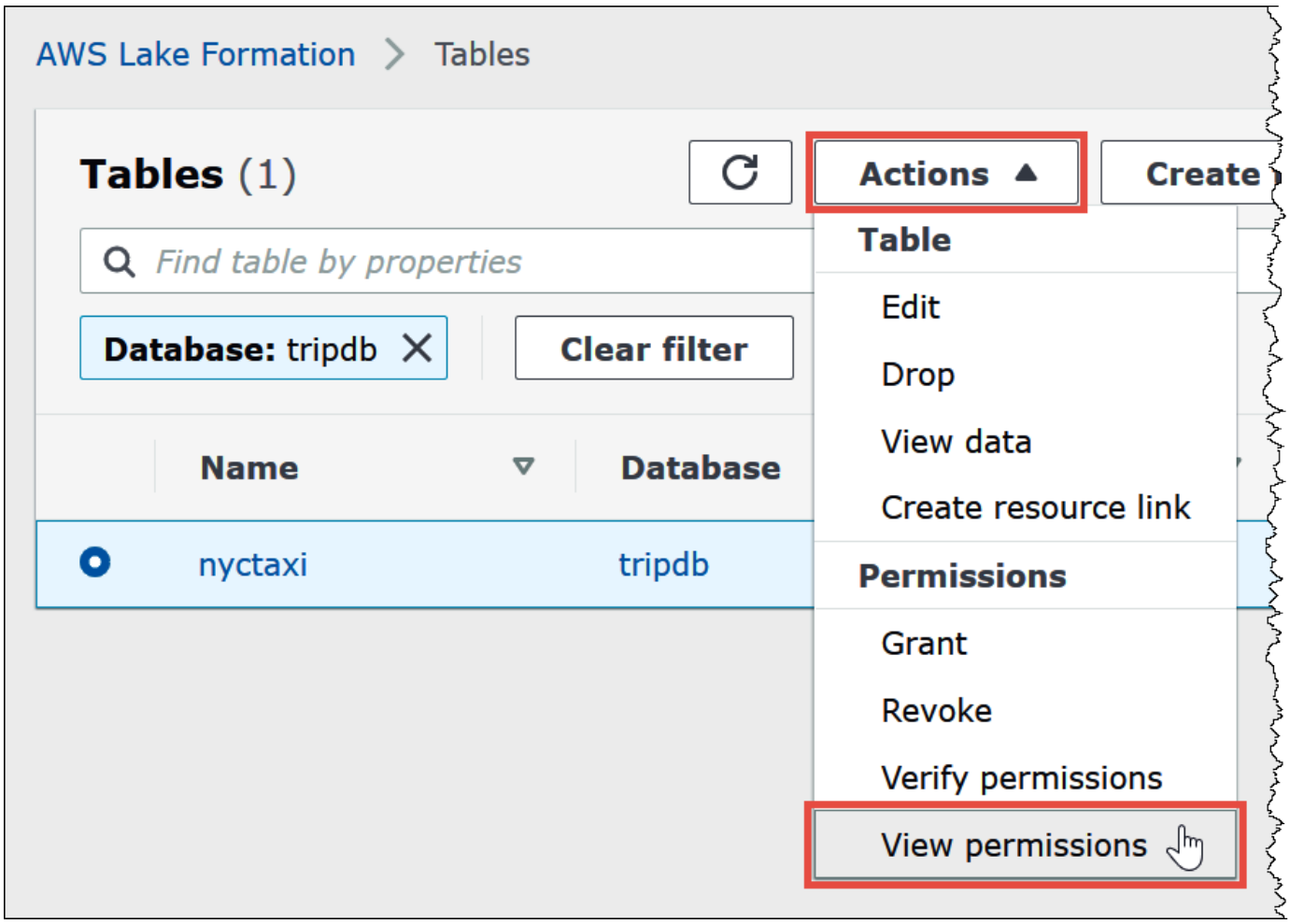
c. 하나 이상의 열 선택(Choose one or more columns)에서 테이블의 첫 3개 열을 선택합니다.

d. 테이블 권한(Table permissions)에서 부여할 특정 액세스 권한을 선택합니다. 이 자습서에서는 SELECT 권한만 부여하며, 각자의 요구 사항은 다를 수 있습니다.

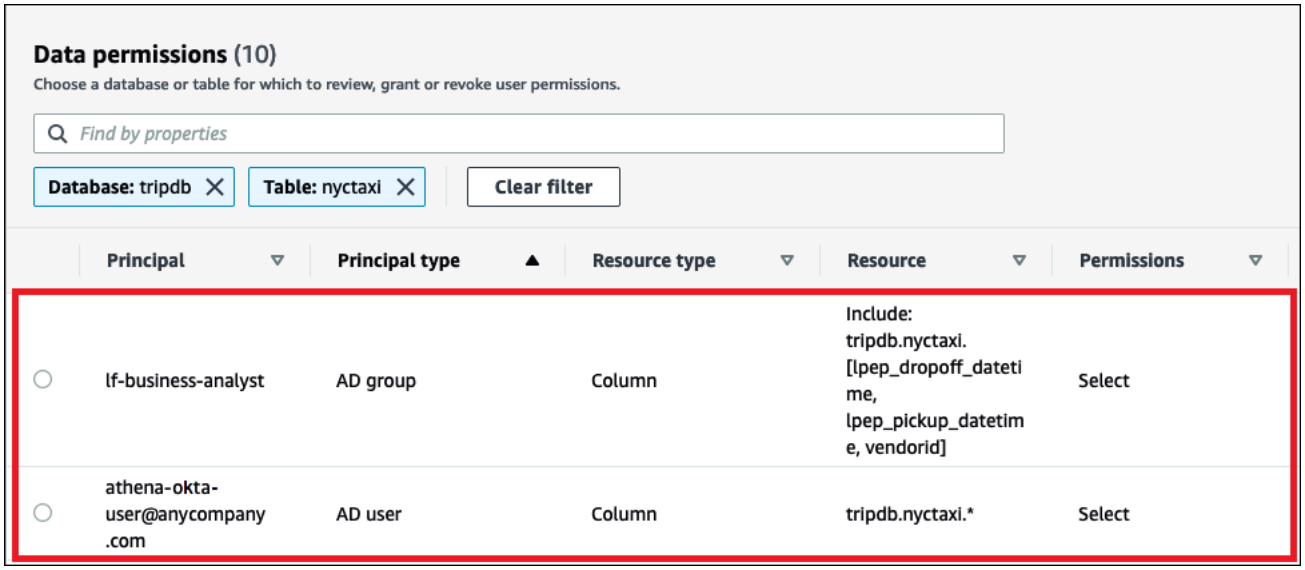


4. 권한 부여(Grant)를 선택합니다.

5. 부여한 권한을 확인하려면 작업(Actions), 권한 보기(View permissions)를 선택합니다.



`nyctaxi` 테이블에 대한 데이터 권한(Data permissions) 페이지에 athena-okta-user 및 lf-business-analyst 그룹에 대한 권한이 표시됩니다.



7단계: Athena JDBC 클라이언트를 통해 액세스 권한 확인

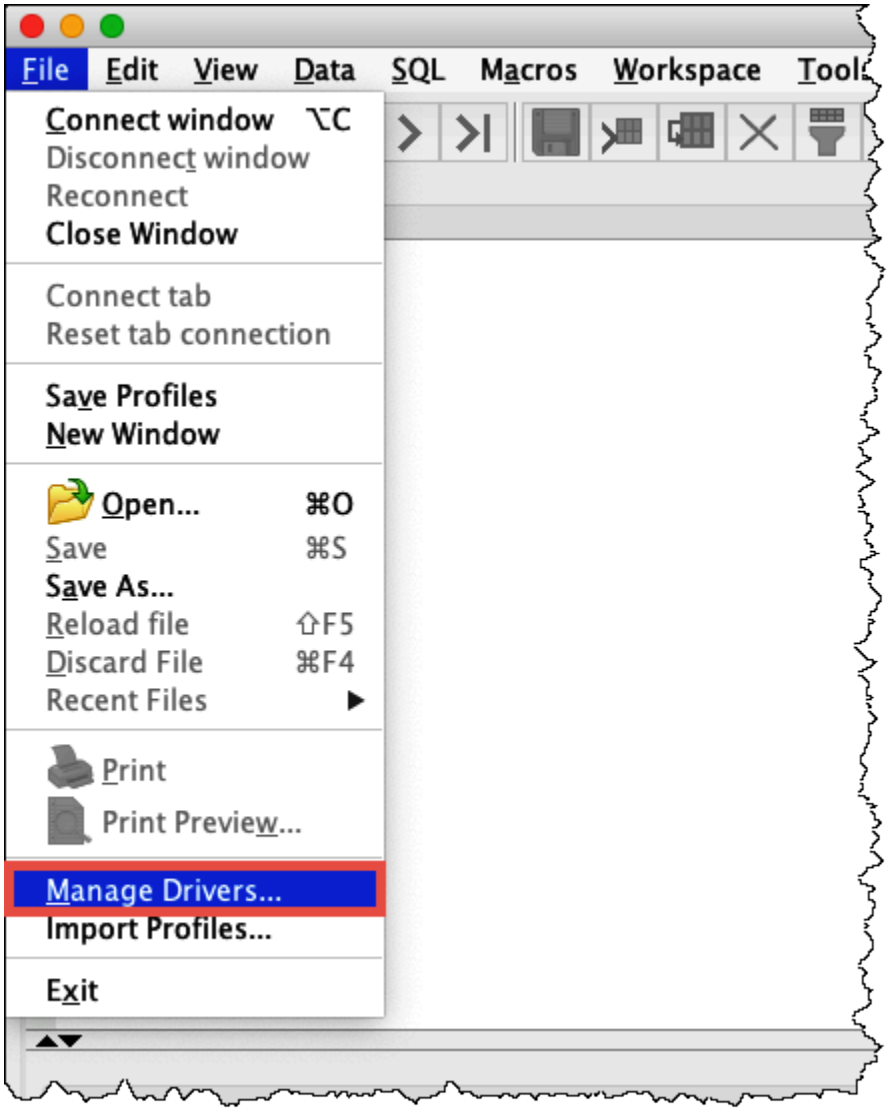
이제 Okta SAML 사용자로서 Athena 테스트 연결을 수행하기 위해 JDBC 클라이언트를 사용할 준비가 되었습니다.

이 단원에서는 다음 작업을 수행합니다.

- 테스트 클라이언트 준비 - Athena JDBC 드라이버를 다운로드하고 SQL 워크벤치를 설치한 다음 드라이버를 워크벤치에 추가합니다. 이 자습서에서는 SQL 워크벤치를 사용하여 Okta 인증을 통해 Athena 액세스하고 Lake Formation 권한을 확인합니다.
- SQL Workbench에서:
  - Athena Okta 사용자에 대한 연결을 만듭니다.
  - Athena Okta 사용자로 테스트 쿼리를 실행합니다.
  - 비즈니스 분석가 사용자에 대한 연결을 만들고 테스트합니다.
- Okta 콘솔에서 비즈니스 분석가 사용자를 개발자 그룹에 추가합니다.
- Lake Formation 콘솔에서 개발자 그룹에 대한 테이블 권한을 구성합니다.
- SQL 워크벤치에서 비즈니스 분석가 사용자로 테스트 쿼리를 실행하고 권한 변경이 결과에 어떤 영향을 미치는지 확인합니다.

테스트 클라이언트를 준비하려면

1. Lake Formation 호환 Athena JDBC 드라이버(버전 2.0.14 이상)를 다운로드하고 JDBC로 Amazon Athena에 연결에서 추출합니다.
2. 수정된 Apache 2.0 라이선스에 따라 제공되는 무료 SQL Workbench/J SQL 쿼리 도구를 다운로드하고 설치합니다.
3. SQL Workbench에서 파일(File)을 선택한 후 드라이버 관리(Manage Drivers)를 선택합니다.



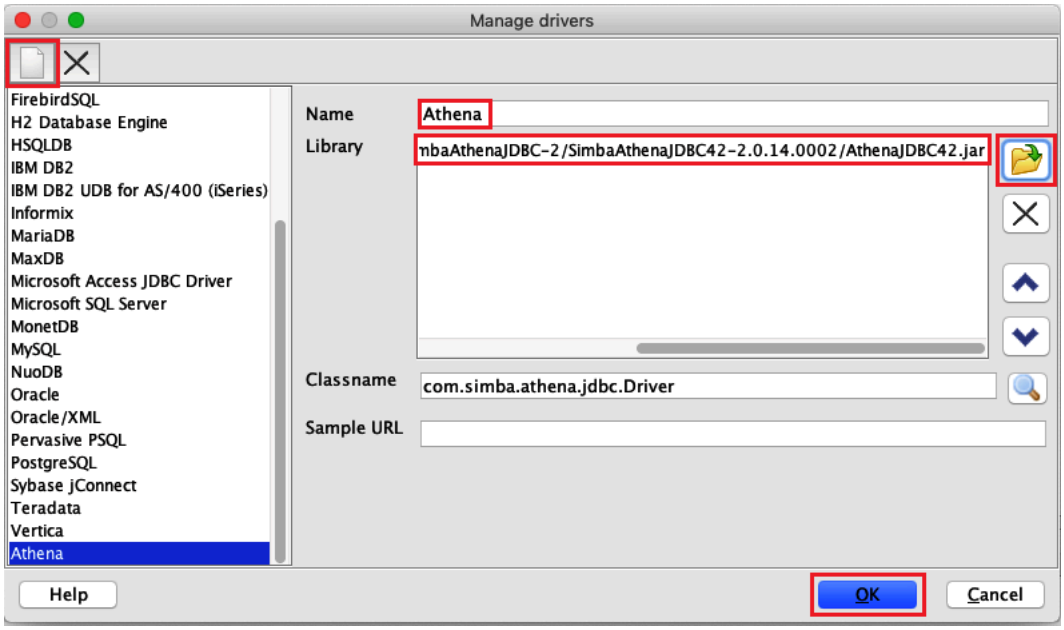
4. 드라이버 관리(Manage Drivers) 대화 상자에서 다음 단계를 수행합니다.

   a. 새 드라이버 아이콘을 선택합니다.

   b. 이름에 **`Athena`**를 입력합니다.

   c. 라이브러리(Library)에서 방금 다운로드한 Simba Athena JDBC `.jar` 파일을 찾아서 선택합니다.

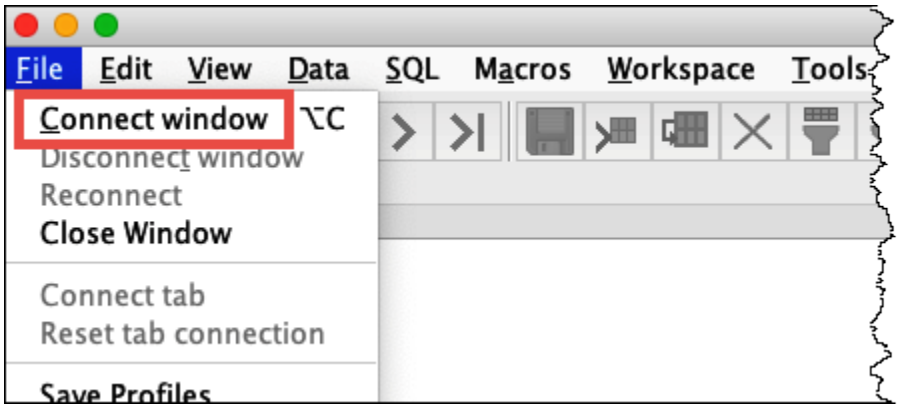
   d. 확인을 선택합니다.



이제 Athena Okta 사용자에 대한 연결을 생성하고 테스트할 준비가 되었습니다.

Okta 사용자에 대한 연결을 만들려면

1. 파일(File), 연결 창(Connect window)을 선택합니다.



2. 연결 프로파일(Connection profile) 대화 상자에서 다음 정보를 입력하여 연결을 생성합니다.

   - 이름 상자에 **`Athena_Okta_User_Connection`**을 입력합니다.
   - 드라이버(Driver)에서 Simba Athena JDBC 드라이버를 선택합니다.
   - VPC에 대해 다음 중 하나를 수행합니다.
     - 연결 URL을 사용하려면 단일 행의 연결 문자열을 입력합니다. 다음 예제에서는 가독성을 위해 줄 바꿈이 추가되었습니다.

```
jdbc:awsathena://AwsRegion=region-id;
S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/athena_results;
AwsCredentialsProviderClass=com.simba.athena.iamsupport.plugin.OktaCredentialsProvider;
user=athena-okta-user@anycompany.com;
password=password;
idp_host=okta-idp-domain;
App_ID=okta-app-id;
SSL_Insecure=true;
LakeFormationEnabled=true;
```

- AWS 프로파일 기반 URL을 사용하려면 다음 단계를 수행합니다.

  1. 다음 예제처럼 AWS 자격 증명 파일이 있는 AWS 프로파일을 구성합니다.

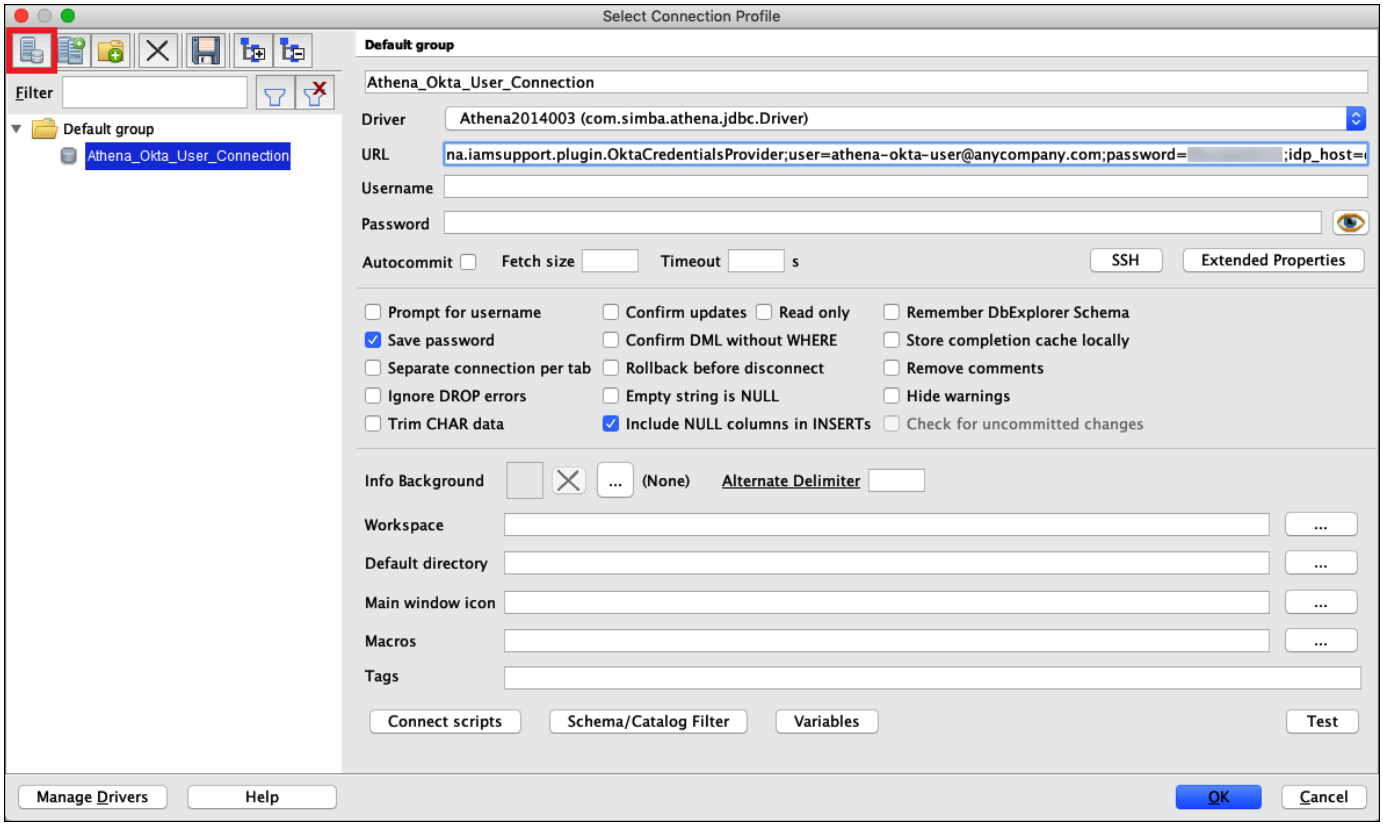
     ```
     [athena_lf_dev]
     plugin_name=com.simba.athena.iamsupport.plugin.OktaCredentialsProvider
     idp_host=okta-idp-domain
     app_id=okta-app-id
     uid=athena-okta-user@anycompany.com
     pwd=password
     ```

  2. 다음 예제와 같이 URL에 단일 행의 연결 문자열을 입력합니다. 예제에는 가독성을 위해 줄 바꿈이 추가되었습니다.

     ```
     jdbc:awsathena://AwsRegion=region-id;
     S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/athena_results;
     profile=athena_lf_dev;
     SSL_Insecure=true;
     LakeFormationEnabled=true;
     ```

  이 예제는 Athena 연결하는 데 필요한 URL의 기본 표현입니다. URL에서 지원되는 파라미터의 전체 목록은 JDBC 설명서를 참조하세요.

다음 이미지는 연결 URL을 사용하는 SQL 워크벤치 연결 프로파일을 보여 줍니다.

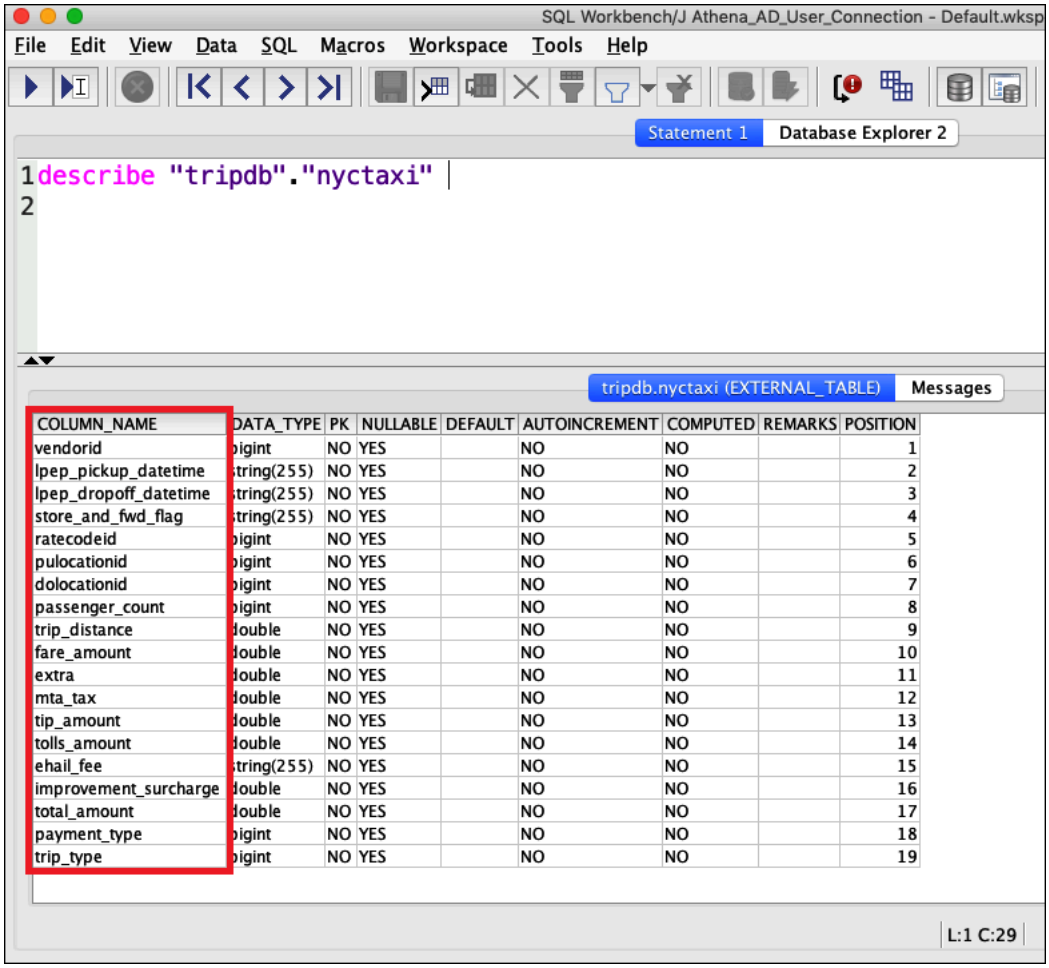


이제 Okta 사용자에 대한 연결을 설정했으므로 일부 데이터를 검색하여 테스트할 수 있습니다.

Okta 사용자에 대한 연결을 테스트하려면

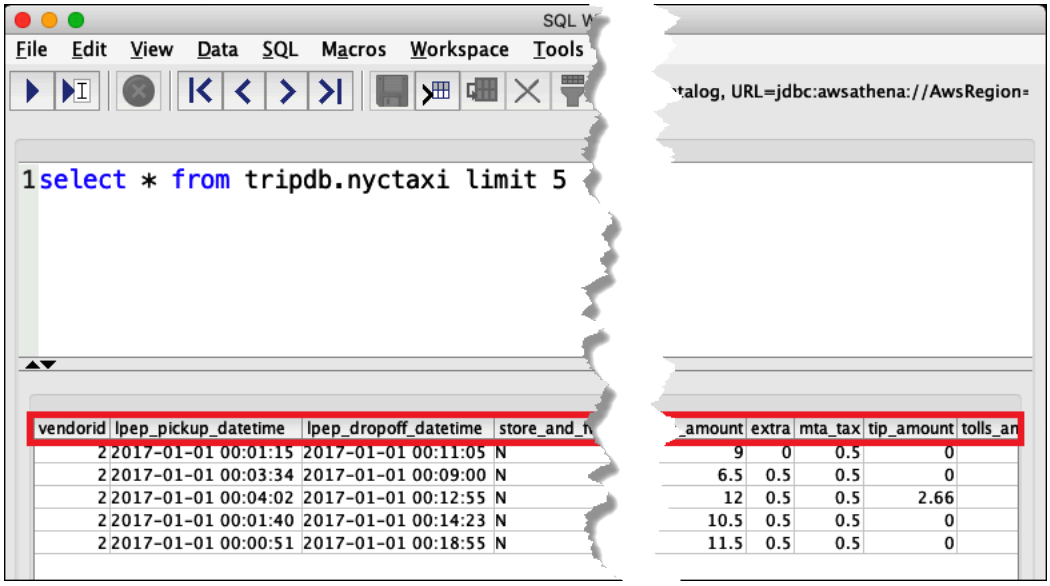
1. 테스트(Test)를 선택한 다음 연결이 성공하는지 확인합니다.
2. SQL 워크벤치의 문(Statement) 창에서 다음 SQL DESCRIBE 명령을 실행합니다. 모든 열이 표시되는지 확인합니다.

```
DESCRIBE "tripdb"."nyctaxi"
```



3. SQL 워크벤치의 문(Statement) 창에서 다음 SQL SELECT 명령을 실행합니다. 모든 열이 표시되는지 확인합니다.

```
SELECT * FROM tripdb.nyctaxi LIMIT 5
```



다음으로 lf-business-analyst 그룹의 구성원인 athena-ba-user가 앞서 Lake Formation에서 지정한 테이블의 첫 3개 열에 대해서만 액세스 권한을 갖는지 확인합니다.

athena-ba-user의 액세스 권한을 확인하려면

1. SQL 워크벤치의 연결 프로파일(Connection profile) 대화 상자에서 다른 연결 프로파일을 만듭니다.

   - 연결 프로파일 이름에 **`Athena_Okta_Group_Connection`**을 입력합니다.
   - 드라이버(Driver)에서 Simba Athena JDBC 드라이버를 선택합니다.
   - VPC에 대해 다음 중 하나를 수행합니다.
     - 연결 URL을 사용하려면 단일 행의 연결 문자열을 입력합니다. 다음 예제에서는 가독성을 위해 줄 바꿈이 추가되었습니다.

```
jdbc:awsathena://AwsRegion=region-id;
S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/athena_results;
AwsCredentialsProviderClass=com.simba.athena.iamsupport.plugin.OktaCredentialsProvider;
user=athena-ba-user@anycompany.com;
password=password;
idp_host=okta-idp-domain;
App_ID=okta-application-id;
SSL_Insecure=true;
LakeFormationEnabled=true;
```

- AWS 프로파일 기반 URL을 사용하려면 다음 단계를 수행합니다.

  1. 다음 예제처럼 자격 증명 파일이 있는 AWS 프로파일을 구성합니다.

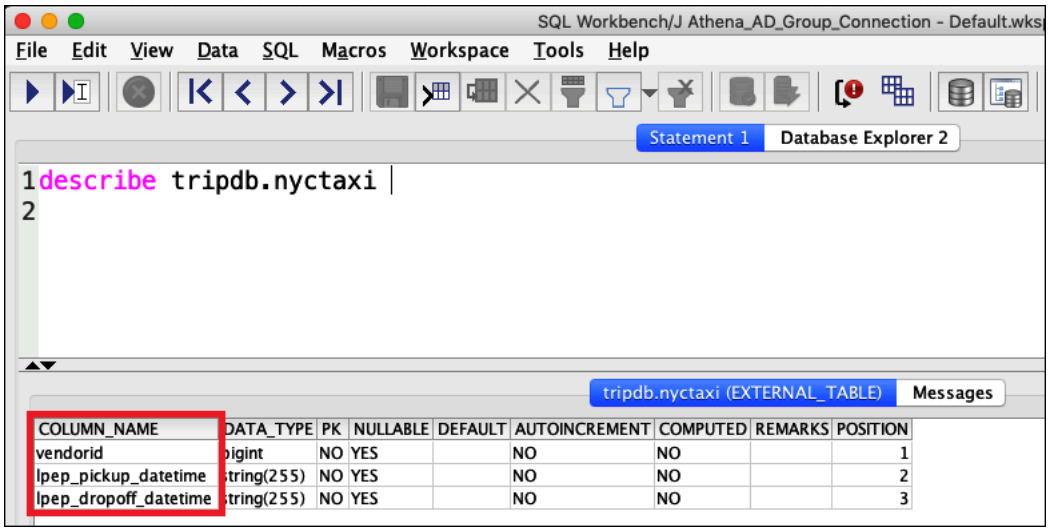
```
[athena_lf_ba]
plugin_name=com.simba.athena.iamsupport.plugin.OktaCredentialsProvider
idp_host=okta-idp-domain
app_id=okta-application-id
uid=athena-ba-user@anycompany.com
pwd=password
```

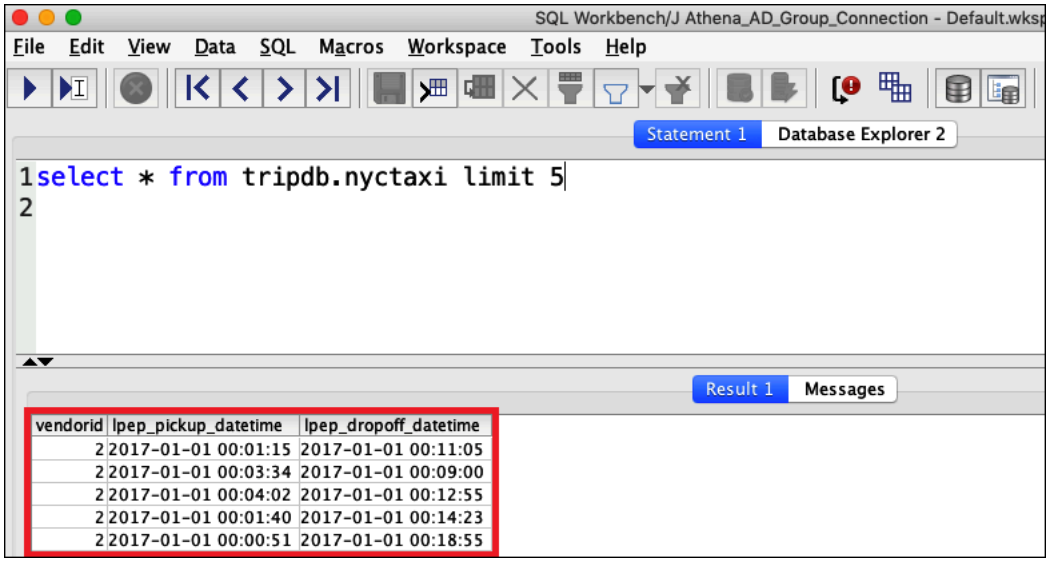
  2. 다음과 같이 URL에 단일 행의 연결 문자열을 입력합니다. 예제에는 가독성을 위해 줄 바꿈이 추가되었습니다.

```
jdbc:awsathena://AwsRegion=region-id;
S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/athena_results;
profile=athena_lf_ba;
SSL_Insecure=true;
LakeFormationEnabled=true;
```

2. 테스트(Test)를 선택해 연결이 성공적인지 확인합니다.
3. SQL 문(SQL Statement) 창에서 전에 수행했던 것과 동일한 DESCRIBE 및 SELECT SQL 명령을 실행하고 결과를 조사합니다.

   athena-ba-user는 lf-business-analyst 그룹의 구성원이기 때문에 Lake Formation 콘솔에서 지정한 첫 3개의 열만 반환됩니다.

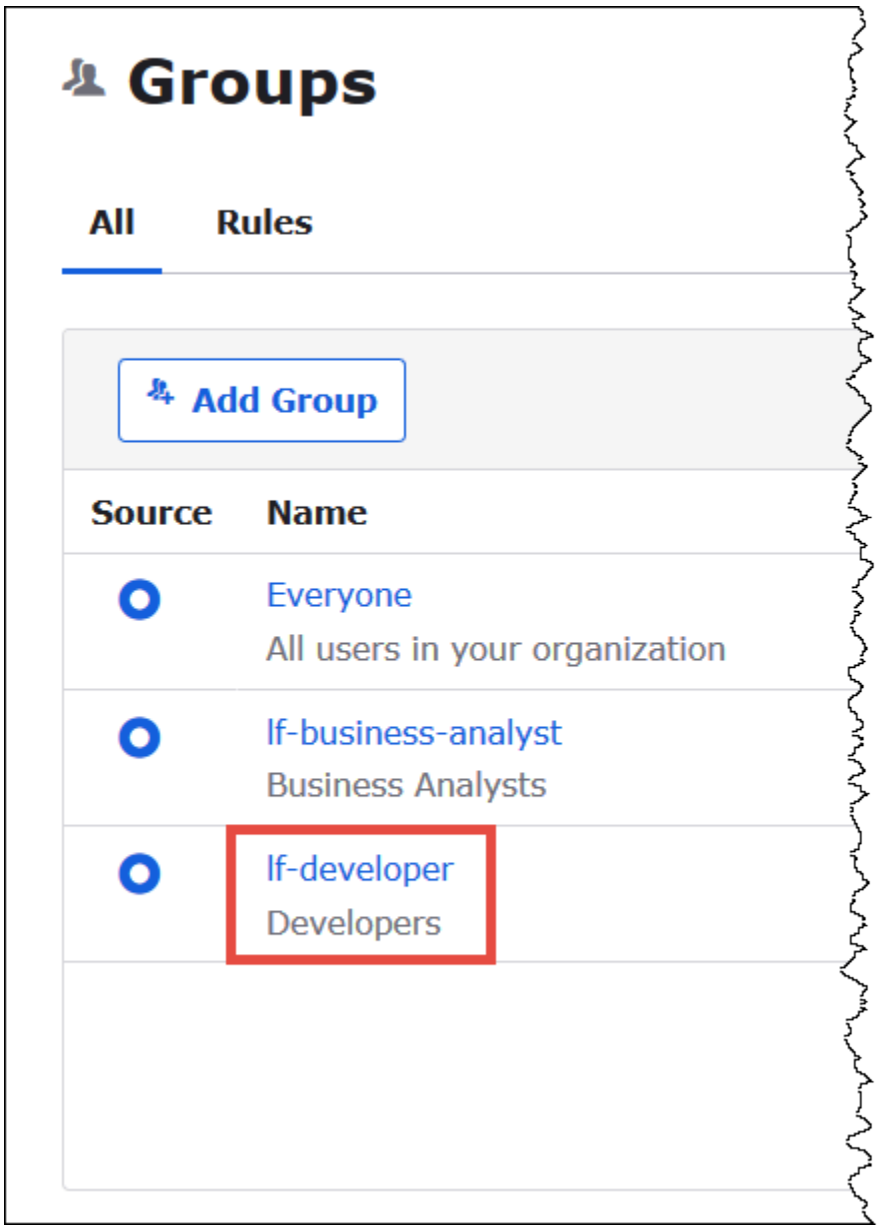




다음으로 Okta 콘솔로 돌아가 `athena-ba-user`를 `lf-developer` Okta 그룹에 추가합니다.

lf-developer 그룹에 athena-ba-user를 추가하려면

1. 할당된 Okta 도메인의 관리 사용자로 Okta 콘솔에 로그인합니다.
2. 디렉터리(Directory)를 선택한 후 그룹(Groups)을 선택합니다.
3. 그룹(Groups) 페이지에서 lf-developer 그룹을 선택합니다.



4. 사람 관리(Manage People)를 선택합니다.
5. 구성원 아님(Not Members) 목록에서 athena-ba-user를 선택해 lf-developer group에 추가합니다.
6. Save(저장)를 선택합니다.

이제 Lake Formation 콘솔로 돌아가 lf-developer 그룹에 대한 테이블 권한을 구성합니다.

lf-developer-group에 대한 테이블 권한을 구성하려면

1. Lake Formation 콘솔에 데이터 레이크 관리자로 로그인합니다.
2. 탐색 창에서 테이블을 선택합니다.

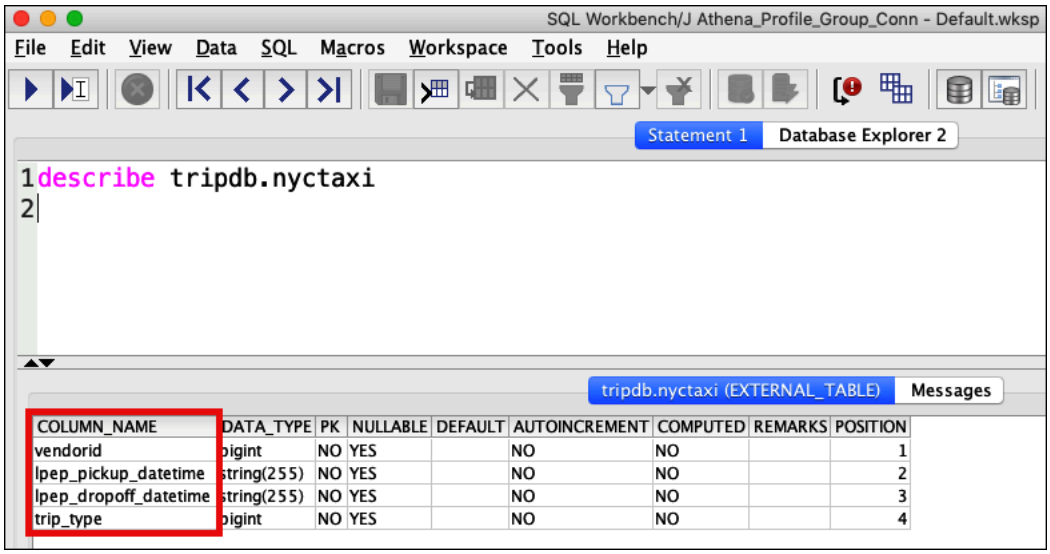
3. nyctaxi 테이블을 선택합니다.
4. 작업(Actions), 권한 부여(Grant)를 선택합니다.
5. 권한 부여(Grant Permissions) 대화 상자에 다음 정보를 입력합니다.
   - SAML 및 Amazon QuickSight의 사용자 및 그룹(SAML and Amazon QuickSight users and groups)에 다음 형식으로 Okta SAML lf-developer 그룹 ARN을 입력합니다.
   - 열(Columns)에서 필터 유형을 선택하고, 열 포함(Include columns)을 선택합니다.
   - trip_type 열을 선택합니다.
   - 테이블 권한(Table permissions)에 SELECT를 선택합니다.
6. 권한 부여(Grant)를 선택합니다.

이제 SQL 워크벤치를 사용하여 lf-developer 그룹에 대한 권한 변경을 확인할 수 있습니다. athena-ba-user는 이제 lf-developer 그룹의 구성원이므로 사용 가능한 데이터에 변경이 반영되어야 합니다.

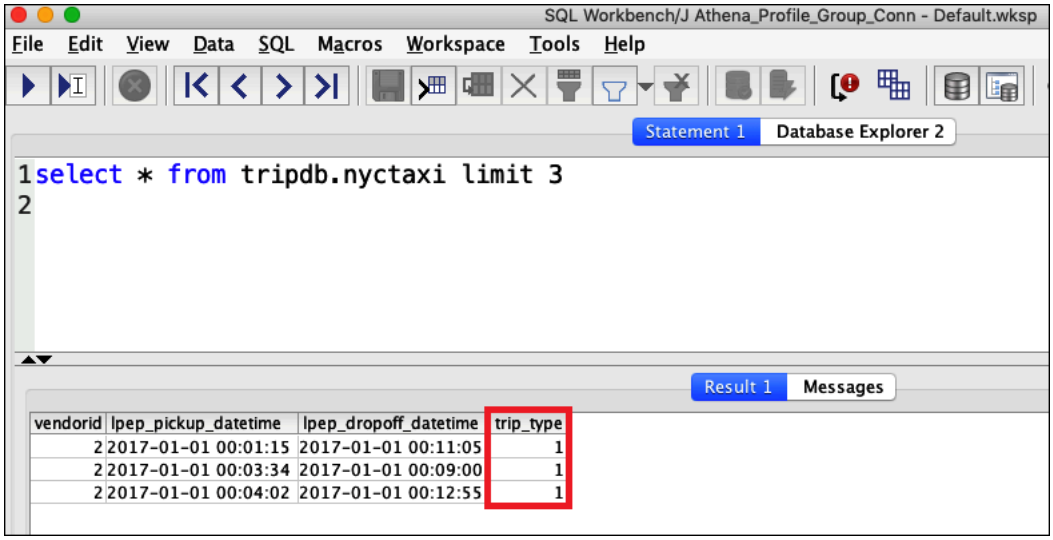
athena-ba-user에 대한 권한 변경을 확인하려면

1. SQL 워크벤치 프로그램을 닫은 다음 다시 엽니다.
2. athena-ba-user의 프로파일에 연결합니다.
3. 문(Statement) 창에서 이전에 실행한 것과 동일한 SQL 문을 실행합니다.

   이번에는 trip_type 열이 표시됩니다.



   athena-ba-user가 이제 lf-developer 및 lf-business-analyst 그룹 모두의 구성원이므로 이 그룹들에 대한 Lake Formation 권한 조합이 반환되는 열을 결정합니다.



결론

이 자습서에서는 Okta를 SAML 공급자로 사용해 Athena와 AWS Lake Formation의 통합을 구성했습니다. 데이터 레이크 AWS Glue 데이터 카탈로그에서 SAML 사용자가 사용 가능한 리소스를 제어하는 데에는 Lake Formation 및 IAM이 사용되었습니다.

관련 리소스

관련 내용은 다음 리소스를 참조하세요.

- JDBC로 Amazon Athena에 연결
- Athena API에 대한 연합 액세스 활성화
- AWS Lake Formation 개발자 안내서
- AWS Lake Formation 개발자 안내서의 데이터 카탈로그 권한 부여 및 취소.
- IAM 사용 설명서의 자격 증명 공급자 및 연동.
- IAM 사용 설명서의 IAM SAML 자격 증명 공급자 생성.
- AWS 보안 블로그의 Enabling Federation to AWS Using Windows Active Directory, ADFS, and SAML 2.0.

워크로드 관리

Athena의 작업 그룹, 용량 관리, 성능 튜닝, 압축 지원, 태그, 서비스 할당량 기능을 사용하여 워크로드를 관리할 수 있습니다.

주제

작업 그룹을 사용하여 쿼리 액세스 및 비용 제어

작업 그룹을 사용하여 사용자, 팀, 애플리케이션 또는 워크로드를 구분하고, 각 쿼리 또는 전체 작업 그룹에서 처리할 수 있는 데이터 양의 한도를 설정하고, 비용을 추적할 수 있습니다. 작업 그룹은 리소스 역할을 하기 때문에 리소스 수준 ID 기반 정책을 사용하여 특정 작업 그룹에 대한 액세스를 제어할 수 있습니다. 또한 Amazon CloudWatch에서 쿼리 관련 지표를 볼 수 있고, 스캔된 데이터의 양에 대한 한도를 구성하여 비용을 통제할 수 있으며, 임계값을 생성하여 이러한 임계값이 위반될 경우 Amazon SNS와 같은 조치를 실행할 수 있습니다.

비용을 추가로 제어하기 위해 사용자가 지정한 데이터 처리 장치 수로 용량 예약을 생성하고 예약에 하나 이상의 작업 그룹을 추가할 수 있습니다. 자세한 내용은 쿼리 처리 용량 관리 단원을 참조하십시오.

작업 그룹은 다음과 같이 IAM, CloudWatch, Amazon Simple Notification Service 및 AWS Cost and Usage Reports와 통합됩니다.

- 리소스 수준 권한이 있는 IAM ID 기반 정책은 작업 그룹에서 쿼리를 실행할 수 있는 사람을 제어합니다.
- 쿼리 지표를 활성화하는 경우 Athena는 CloudWatch에 작업 그룹 쿼리 지표를 게시합니다.
- 작업 그룹의 쿼리에 대한 데이터 사용량 제어가 사전 설정한 임계값을 초과할 경우 지정된 작업 그룹 사용자에게 경보를 전송하는 Amazon SNS 주제를 Amazon SNS에서 만들 수 있습니다.
- Billing and Cost Management 콘솔에서 비용 할당 태그로 구성된 태그를 작업 그룹에 지정하면 해당 작업 그룹에서 쿼리 실행과 관련된 비용이 해당 비용 할당 태그와 함께 Cost and Usage Reports에 나타납니다.

주제

AWS Big Data Blog(빅 데이터 블로그) 게시물 Separate queries and managing costs using Amazon Athena workgroups(Amazon Athena 작업 그룹을 사용하여 별도의 쿼리 및 비용 관리)도 참조하세요. 작업 그룹을 사용해 워크로드를 분산시키고, 사용자 액세스를 제어하고, 쿼리 사용량 및 비용을 관리하는 방법을 알려줍니다.

쿼리 실행용 작업 그룹 사용

작업 그룹을 사용하여 팀, 애플리케이션, 여러 워크로드에 대한 쿼리를 분리하는 것이 좋습니다. 예를 들어, 조직의 서로 다른 두 팀에 대해 작업 그룹을 각각 만들 수 있습니다. 워크로드를 분리할 수도 있습니다. 예를 들어, 보고서 생성과 같은 자동화된 예약 애플리케이션용으로 한 개, 분석가의 임시 사용을 위해 또 다른 한 개, 즉 두 개의 작업 그룹을 만들 수 있습니다. 작업 그룹 간에 전환할 수 있습니다.

주제

작업 그룹 사용의 이점

작업 그룹을 사용하여 다음을 수행할 수 있습니다.

사용자, 팀, 애플리케이션, 워크로드를 그룹으로 구분합니다.	각 작업 그룹은 고유한 자체 쿼리 기록과 저장된 쿼리 목록을 갖습니다. 자세한 내용은 작업 그룹의 작동 방식 단원을 참조하십시오. 작업 그룹의 모든 쿼리에 대해 작업 그룹 설정을 구성할 수 있습니다. 작업 항목에는 쿼리 결과를 저장하는 Amazon S3 위치, 예상 버킷 소유자, 암호화 및 쿼리 결과 버킷에 기록된 객체의 제어가 포함됩니다. 작업 그룹 설정을 적용할 수도 있습니다. 자세한 내용은 작업 그룹 설정 단원을 참조하십시오.

비용 제약 조건을 적용합니다.	한 작업 그룹의 쿼리에 대해 두 가지 유형의 비용 제약 조건을 설정할 수 있습니다. • 쿼리당 제한은 각 쿼리에서 스캔되는 데이터 양에 대한 임계값입니다. 데이터 양이 지정된 임계값을 초과하면 Athena가 쿼리를 취소합니다. 이 한도는 작업 그룹 내에서 실행 중인 각 쿼리에 적용됩니다. 쿼리당 한도는 한 개만 설정할 수 있으며 필요할 경우 업데이트할 수 있습니다. • 작업 그룹당 한도는 각 작업 그룹에 대해 작업 그룹의 쿼리에서 스캔한 데이터 양에 대해 설정할 수 있는 임계값입니다. 임계값을 위반하면 Amazon SNS 경보가 활성화되고 선택된 작업(예: 지정된 사용자에게 이메일 보내기)이 트리거됩니다. 각 작업 그룹에 대해 작업 그룹당 한도를 여러 개 설정할 수 있습니다. 자세한 단계는 데이터 사용량 제어 한도 설정 단원을 참조하세요.
CloudWatch에서 모든 작업 그룹 쿼리에 대해 쿼리 관련 지표를 추적할 수 있습니다.	작업 그룹에서 실행되는 각 쿼리에 대해 지표를 게시하도록 작업 그룹을 구성한 경우 Athena는 지표를 CloudWatch에 게시합니다. Athena 콘솔 내에서 작업 그룹 각각에 대해 쿼리 지표를 확인할 수 있습니다. CloudWatch에서 사용자 지정 대시보드를 생성하고, 이러한 지표에 대한 임계값과 경보를 설정할 수 있습니다.

작업 그룹의 작동 방식

Athena에서 작업 그룹은 다음과 같은 특징을 갖습니다.

- 기본적으로 각 계정에는 기본 작업 그룹이 있고, 승인된 모든 사용자가 이 작업 그룹에 액세스할 수 있도록 허용하는 기본 권한이 있습니다. 기본 작업 그룹은 삭제할 수 없습니다.
- 생성한 각 작업 그룹은 계정의 모든 쿼리가 아닌 해당 쿼리에 대해서만 저장된 쿼리 및 쿼리 기록을 표시합니다. 이렇게 하면 해당 쿼리를 계정 내의 다른 쿼리와 구분하여 직접 저장한 쿼리와 기록의 쿼리를 보다 효율적으로 찾을 수 있습니다.
- 작업 그룹을 비활성화하면 다시 활성화할 때 까지는 해당 작업 그룹에서 쿼리를 실행할 수 없습니다. 비활성화된 작업 그룹에 대해 실행한 쿼리는 해당 작업 그룹을 다시 활성화할 때까지는 실패합니다.
- 권한이 있는 경우 빈 작업 그룹과 저장된 쿼리가 포함된 작업 그룹을 삭제할 수 있습니다. 이 경우 작업 그룹을 삭제하기 전에 Athena에서 저장된 쿼리가 삭제된다고 경고합니다. 다른 사용자가 액세스

할 수 있는 작업 그룹을 삭제하기 전에 그 작업 그룹의 사용자가 쿼리를 계속 실행할 수 있는 다른 작업 그룹에 액세스할 수 있는지 확인하세요.

- 작업 그룹 전역 설정을 지정하여 작업 그룹에서 실행하는 모든 쿼리에 적용할 수 있습니다. 설정에는 Amazon S3 쿼리 결과 위치, 예상 버킷 소유자, 암호화 및 쿼리 결과 버킷에 기록된 객체의 제어가 포함됩니다.

> **Important**
>
> 작업 그룹 전역 설정을 적용하면 이 작업 그룹에서 실행하는 모든 쿼리가 작업 그룹 설정을 사용합니다. 이는 클라이언트 측 설정이 작업 그룹 설정과 다를 경우에도 적용됩니다. 자세한 설명은 클라이언트 측 설정보다 우선하는 작업 그룹 설정을 참조하세요.

작업 그룹 제한 사항

- 계정에서 리전당 최대 1,000개의 작업 그룹을 만들 수 있습니다.
- 기본 작업 그룹은 삭제할 수 없습니다.
- 각 작업 그룹에서 쿼리 탭을 10개까지 열 수 있습니다. 작업 그룹 간에 전환할 경우 쿼리 탭은 최대 3개의 작업 그룹에 대해 열린 채로 유지됩니다.

작업 그룹 설정

작업 그룹을 설정하는 작업은 작업 그룹을 생성하고 작업 그룹 사용에 대한 권한을 설정하는 절차로 이루어집니다. 먼저 조직에서 필요한 작업 그룹을 결정하고 만듭니다. 그런 다음 `workgroup` 리소스에 대한 사용자 액세스와 작업을 제어하는 IAM 작업 그룹 정책을 설정합니다. 이러한 작업 그룹에 액세스할 수 있는 사용자는 이제 해당 작업 그룹에서 쿼리를 실행할 수 있습니다.

> **Note**
>
> 작업 그룹을 처음 사용하기 시작할 때 이러한 작업 설정 작업을 수행하세요. 해당 Athena 계정에서 이미 작업 그룹을 사용할 경우, 각 계정의 사용자는 계정의 한 개 이상 작업 그룹에서 쿼리를 실행할 수 있는 권한이 필요합니다. 쿼리를 실행하기 전에 IAM 정책을 검토하여 액세스할 수 있는 작업 그룹을 확인하고, 필요할 경우 조정하고, 사용하려는 작업 그룹으로 전환해야 합니다.

작업 그룹을 생성하지 않은 경우 기본적으로 해당 계정의 모든 쿼리는 다음과 같이 기본 작업 그룹에서 실행됩니다.

Athena는 콘솔 오른쪽 상단의 작업 그룹(Workgroup) 옵션에 현재 작업 그룹을 표시합니다. 이 옵션을 사용하여 작업 그룹을 전환할 수 있습니다. 쿼리를 실행하면 현재 작업 그룹에서 실행됩니다. API 작업, 명령줄 인터페이스, JDBC 드라이버 또는 ODBC 드라이버를 사용하여 클라이언트 애플리케이션을 통해 콘솔의 작업 그룹 컨텍스트에서 쿼리를 실행할 수 있습니다. 작업 그룹에 액세스할 수 있는 경우 작업 그룹의 설정, 지표, 데이터 사용량 제어 한도를 볼 수 있습니다. 추가 권한으로 설정 및 데이터 사용 제어 제한을 편집할 수 있습니다.

작업 그룹을 설정하려면

1. 생성할 작업 그룹을 결정합니다. 예를 들어 다음을 결정할 수 있습니다.

   - 각 작업 그룹에서 쿼리를 실행할 수 있는 사용자 및 작업 구성을 소유하는 사용자. 이에 따라 생성하는 IAM 정책이 결정됩니다. 자세한 내용은 작업 그룹 액세스를 위한 IAM 정책 단원을 참조하십시오.
   - Amazon S3에서 각 작업 그룹에서 실행하는 쿼리의 쿼리 결과에 사용할 위치. 작업 그룹 쿼리 결과에 대해 위치를 지정하려면 먼저 위치가 Amazon S3 내에 존재해야 합니다. 작업 그룹을 사용하는 모든 사용자는 이 위치에 액세스할 수 있어야 합니다. 자세한 내용은 작업 그룹 설정 단원을 참조하십시오.
   - Amazon S3 쿼리 결과 버킷의 소유자가 버킷에 기록된 새 객체를 완벽하게 제어할 수 있는지 여부입니다. 예를 들어, 쿼리 결과 위치가 다른 계정이 소유한 경우 쿼리 결과에 대한 소유권 및 전체 제어 권한을 다른 계정에 부여할 수 있습니다. 자세한 내용은 AclConfiguration 섹션을 참조하세요.
   - 출력 위치 버킷의 소유자가 될 것으로 예상되는 AWS 계정의 ID를 지정합니다. 이것은 선택적으로 추가된 보안 조치입니다. 버킷 소유자의 계정 ID가 여기에서 지정한 ID와 일치하지 않으면 버킷으로의 출력 시도가 실패합니다. 자세한 내용은 Amazon S3 사용 설명서의 버킷 소유자 조건을 사용하여 버킷 소유권 확인을 참조하세요. 이 설정은 CTAS, INSERT INTO 또는 UNLOAD 문에는 적용되지 않습니다.
   - 필요한 암호화 설정과 쿼리를 암호화해야 하는 작업 그룹. 암호화된 쿼리와 암호화되지 않은 쿼리에 대해 작업 그룹을 따로 생성하는 것이 좋습니다. 즉, 작업 그룹에서 실행하는 모든 쿼리에 적용되는 암호화를 작업 그룹에 적용할 수 있습니다. 자세한 내용은 Amazon S3에 저장된 Athena 쿼리 결과 암호화 단원을 참조하십시오.
2. 필요한 대로 작업 그룹을 만들고 작업 그룹에 태그를 추가합니다. 단계는 작업 그룹 만들기를 참조하세요.

3. 사용자 그룹 또는 역할에 대해 작업 그룹에 대한 액세스를 허용하는 IAM 정책을 생성합니다. 이 정책은 작업 그룹 멤버십과 `workgroup` 리소스에 대한 작업에 대한 액세스를 설정합니다. 자세한 단계는 작업 그룹 액세스를 위한 IAM 정책 단원을 참조하세요. JSON 정책 예제는 작업 그룹 및 태그에 대한 액세스 단원을 참조하세요.
4. 작업 그룹을 설정합니다. 쿼리 결과에 대한 Amazon S3의 위치를 지정하고 필요에 따라 예상 버킷 소유자 및 암호화 설정을 지정합니다. 작업 그룹 설정을 적용할 수 있습니다. 자세한 정보는 작업 그룹 설정을 참조하세요.

   > **Important**
   >
   > 클라이언트 측 설정을 재정의할 경우, Athena는 작업 그룹의 설정을 사용합니다. 이 설정은 콘솔, 명령줄 인터페이스, API 작업에서 실행하거나 드라이버를 사용하여 실행하는 쿼리에 적용됩니다.
   >
   > 쿼리는 계속 실행되지만 특정 Amazon S3 버킷의 결과 가용성에 따라 자동화가 중단될 수 있습니다. 재정의하기 전에 사용자에 알리는 것이 좋습니다. 작업 그룹 설정을 재정의하도록 설정한 후 드라이버 또는 API에서 클라이언트 측 설정 지정을 생략할 수 있습니다.

5. 사용자에게 쿼리 실행에 사용할 작업 그룹을 알립니다. 이메일을 보내 사용할 수 있는 작업 그룹 이름, 필요한 IAM 정책 및 작업 그룹 설정을 계정 사용자에게 알려줍니다.
6. 쿼리 및 작업 그룹에 대한 비용 제어 한도(데이터 사용량 제어 한도)를 구성합니다. 임계값 위반 시 알림을 받기 위해 Amazon SNS 주제를 생성하고 구독을 구성합니다. 자세한 단계는 데이터 사용량 제어 한도 설정 및 Amazon Simple Notification Service 개발자 안내서의 Amazon SNS 시작하기를 참조하세요.
7. 쿼리를 실행할 수 있도록 작업 그룹으로 전환합니다. 쿼리를 실행하려면 해당 작업 그룹으로 전환합니다. 자세한 단계는 the section called “쿼리를 실행할 작업 그룹 지정” 단원을 참조하세요.

작업 그룹 액세스를 위한 IAM 정책

작업 그룹에 대한 액세스를 제어하려면 리소스 수준 IAM 권한이나 ID 기반 IAM 정책을 사용합니다. IAM 정책을 사용할 때는 항상 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

> **Note**
>
> 신뢰할 수 있는 ID 전파 지원 작업 그룹에 액세스하려면 Athena GetWorkGroup API 작업의 응답으로 반환되는 `IdentityCenterApplicationArn`에 IAM Identity Center 사용자를 할당해야 합니다.

다음은 Athena용 절차입니다.

IAM용 정보는 이 단원 끝부분에 나와 있는 링크를 참조하세요. JSON 작업 그룹 정책 예제에 대한 자세한 내용은 작업 그룹 정책의 예 단원을 참조하세요.

IAM 콘솔에서 시각적 편집기를 사용하려면 작업 그룹 정책을 만듭니다.

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 정책을 선택한 후 정책 생성을 선택합니다.
3. 시각적 편집기(Visual editor) 탭에서 서비스 선택(Choose a service)을 선택합니다. 그런 다음 Athena를 선택하여 정책에 추가합니다.
4. 작업 선택(Select actions)을 선택한 후 정책에 추가할 작업을 선택합니다. 시각적 편집기에 Athena에서 사용할 수 있는 작업이 표시됩니다. 자세한 내용은 서비스 권한 부여 참조에서 Amazon Athena에 사용되는 작업, 리소스 및 조건 키를 참조하세요.
5. 작업 추가(add actions)를 선택하여 특정 작업을 입력하거나 와일드카드(*)를 사용하여 여러 개의 작업을 지정합니다.

   기본적으로 생성되는 정책은 사용자가 선택하는 작업을 허용합니다. Athena의 `workgroup` 리소스에 대해 리소스 수준 권한을 지원하는 작업을 하나 이상 선택하면 편집기에 `workgroup` 리소스가 나열됩니다.
6. 리소스를 선택하여 정책에 대한 특정 작업 그룹을 지정합니다. JSON 작업 그룹 정책 예제는 작업 그룹 정책의 예 단원을 참조하세요.
7. 다음과 같이 `workgroup` 리소스를 지정합니다.

   ```
   arn:aws:athena:<region>:<user-account>:workgroup/<workgroup-name>
   ```
8. 정책 검토(Review policy)를 선택한 후 생성하려는 정책에 대한 이름(Name)과 설명(Description)(선택 사항)을 입력합니다. 정책 요약을 검토하여 의도한 권한을 부여했는지 확인합니다.
9. 정책 생성(Create policy)을 선택하고 새로운 정책을 저장합니다.

10. ID 기반 정책을 사용자, 그룹 또는 역할에 연결합니다.

자세한 내용은 서비스 권한 부여 참조와 IAM 사용 설명서에서 다음 주제를 참조하세요.

- Amazon Athena에 사용되는 작업, 리소스 및 조건 키
- 시각적 편집기를 사용하여 정책 생성
- IAM 정책 추가 및 제거
- 리소스에 대한 액세스 제어

JSON 작업 그룹 정책 예제는 작업 그룹 정책의 예 단원을 참조하세요.

Amazon Athena 작업의 전체 목록은 Amazon Athena API 참조의 API 작업 이름을 참조하세요.

작업 그룹 정책의 예

이 단원에서는 작업 그룹에 다양한 작업을 허용하는 데 사용할 수 있는 예제 정책을 설명합니다. IAM 정책을 사용할 때는 항상 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

작업 그룹은 Athena에서 관리하는 IAM 리소스입니다. 따라서 해당 작업 그룹 정책에서 `workgroup`을 입력으로 받는 작업을 사용할 경우 다음과 같이 작업 그룹의 ARN을 지정해야 합니다.

```
"Resource": [arn:aws:athena:<region>:<user-account>:workgroup/<workgroup-name>]
```

*<workgroup-name>*은 작업 그룹의 이름입니다. 예를 들어 다음과 같이 `test_workgroup`이라는 작업 그룹을 리소스로 지정할 수 있습니다.

```
"Resource": ["arn:aws:athena:us-east-1:123456789012:workgroup/test_workgroup"]
```

Amazon Athena 작업의 전체 목록은 Amazon Athena API 참조의 API 작업 이름을 참조하세요. IAM 정책에 대한 자세한 내용은 IAM 사용 설명서의 시각적 편집기로 정책 생성을 참조하세요. IAM 정책을 생성하는 방법에 대한 자세한 내용은 작업 그룹 액세스를 위한 IAM 정책 단원을 참조하세요.

- Example policy for full access to all workgroups
- Example policy for full access to a specified workgroup
- Example policy for running queries in a specified workgroup
- Example policy for running queries in the primary workgroup

- Example policy for management operations on a specified workgroup
- Example policy for listing workgroups
- Example policy for running and stopping queries in a specific workgroup
- Example policy for working with named queries in a specific workgroup
- Example policy for working with Spark notebooks

Example 모든 작업 그룹에 대해 전체 액세스를 허용하는 정책의 예

다음 정책은 계정에 있는 모든 작업 그룹 리소스에 대해 전체 액세스를 허용합니다. 해당 계정에서 다른 모든 사용자에 대해 작업 그룹을 관리해야 하는 사용자에게 이 정책을 사용하는 것이 좋습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:*"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Example 지정한 작업 그룹에 대해 전체 액세스를 허용하는 정책의 예

다음 정책은 `workgroupA`이라는 하나의 특정 작업 그룹 리소스에 대해 전체 액세스를 허용합니다. 이 정책은 특정 작업 그룹에 대해 전체 제어 권한을 가진 사용자에게 사용할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListEngineVersions",
                "athena:ListWorkGroups",
                "athena:ListDataCatalogs",
```

```
                "athena:ListDatabases",
                "athena:GetDatabase",
                "athena:ListTableMetadata",
                "athena:GetTableMetadata"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:BatchGetQueryExecution",
                "athena:GetQueryExecution",
                "athena:ListQueryExecutions",
                "athena:StartQueryExecution",
                "athena:StopQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:BatchGetNamedQuery",
                "athena:ListNamedQueries",
                "athena:DeleteNamedQuery",
                "athena:CreatePreparedStatement",
                "athena:GetPreparedStatement",
                "athena:ListPreparedStatements",
                "athena:UpdatePreparedStatement",
                "athena:DeletePreparedStatement"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:DeleteWorkGroup",
                "athena:UpdateWorkGroup",
                "athena:GetWorkGroup",
                "athena:CreateWorkGroup"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA"
            ]
        }
```

```
    ]
}
```

Example 지정된 작업 그룹에서 쿼리 실행을 위한 정책의 예

다음 정책은 지정된 workgroupA에서 사용자가 쿼리를 실행하고 보도록 허용합니다. 사용자는 작업 그룹에 대해 업데이트나 삭제 등의 관리 작업을 수행할 수 없습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Action": [
                "athena:ListEngineVersions",
                "athena:ListWorkGroups",
                "athena:ListDataCatalogs",
                "athena:ListDatabases",
                "athena:GetDatabase",
                "athena:ListTableMetadata",
                "athena:GetTableMetadata"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:GetWorkGroup",
                "athena:BatchGetQueryExecution",
                "athena:GetQueryExecution",
                "athena:ListQueryExecutions",
                "athena:StartQueryExecution",
                "athena:StopQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:BatchGetNamedQuery",
                "athena:ListNamedQueries",
                "athena:DeleteNamedQuery",
                "athena:CreatePreparedStatement",
                "athena:GetPreparedStatement",
                "athena:ListPreparedStatements",
```

```
                "athena:UpdatePreparedStatement",
                "athena:DeletePreparedStatement"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA"
            ]
        }
    ]
}
```

Example 기본 작업 그룹에서 쿼리를 실행하기 위한 정책의 예

이전 예제를 수정하여 특정 사용자가 기본 작업 그룹에서 쿼리를 실행하도록 허용할 수 있습니다.

> **Note**
>
> 지정된 작업 그룹에서 쿼리를 실행하도록 구성된 모든 사용자에게 기본 작업 그룹 리소스를 추가하는 것이 좋습니다. 이 리소스를 작업 그룹 사용자 정책에 추가하면 지정된 작업 그룹이 삭제되거나 비활성화된 경우에 유용합니다. 이렇게 하면 사용자들이 기본 작업 그룹에서 쿼리 실행을 계속할 수 있습니다.

계정의 사용자가 기본 작업 그룹에서 쿼리를 실행하도록 허용하려면, 다음 예와 같이 기본 작업 그룹의 ARN이 포함된 줄을 Example policy for running queries in a specified workgroup의 리소스 섹션에 추가합니다.

```
arn:aws:athena:us-east-1:123456789012:workgroup/primary"
```

Example 지정된 작업 그룹에서 관리 작업을 위한 정책의 예

다음 정책은 사용자가 `test_workgroup` 작업 그룹을 만들고, 삭제하고, 세부 정보를 조회하고, 업데이트할 수 있도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListEngineVersions"
            ],
```

```
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateWorkGroup",
                "athena:GetWorkGroup",
                "athena:DeleteWorkGroup",
                "athena:UpdateWorkGroup"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/test_workgroup"
            ]
        }
    ]
}
```

Example 작업 그룹 목록 조회를 위한 정책의 예

다음 정책은 모든 사용자가 모든 작업 그룹의 목록을 조회할 수 있도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListWorkGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

Example 지정된 작업 그룹에서 쿼리 실행 및 중지를 위한 정책의 예

이 정책은 사용자가 작업 그룹에서 쿼리를 실행하도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "athena:StartQueryExecution",
                "athena:StopQueryExecution"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/test_workgroup"
            ]
        }
    ]
}
```

Example 지정된 작업 그룹에서 지명한 쿼리에 대한 작업을 위한 정책의 예

다음 정책은, 지정된 작업 그룹에서 지명한 쿼리를 생성 및 삭제하고 세부 정보를 볼 수 있는 권한을 사용자에게 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:DeleteNamedQuery"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/test_workgroup"
     ]
        }
    ]
}
```

Example Athena에서 Spark 노트북으로 작업하기 위한 정책의 예

Athena에서 Spark 노트북으로 작업하려면 다음과 같은 정책을 사용하세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreatingWorkGroupWithDefaults",
            "Action": [
```

```
                "athena:CreateWorkGroup",
                "s3:CreateBucket",
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "s3:GetBucketLocation",
                "athena:ImportNotebook"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/Demo*",
                "arn:aws:s3:::123456789012-us-east-1-athena-results-bucket-*",
                "arn:aws:iam::123456789012:role/service-role/
AWSAthenaSparkExecutionRole-*",
                "arn:aws:iam::123456789012:policy/service-role/
AWSAthenaSparkRolePolicy-*"
            ]
        },
        {
            "Sid": "AllowRunningCalculations",
            "Action": [
                "athena:ListWorkGroups",
                "athena:GetWorkGroup",
                "athena:StartSession",
                "athena:CreateNotebook",
                "athena:ListNotebookMetadata",
                "athena:ListNotebookSessions",
                "athena:GetSessionStatus",
                "athena:GetSession",
                "athena:GetNotebookMetadata",
                "athena:CreatePresignedNotebookUrl"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:athena:us-east-1:123456789012:workgroup/Demo*"
        },
        {
            "Sid": "AllowListWorkGroupAndEngineVersions",
            "Action": [
                "athena:ListWorkGroups",
                "athena:ListEngineVersions"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
```

```
    ]
}
```

작업 그룹 설정

각 작업 그룹에는 다음과 같은 설정이 있습니다.

- 고유한 이름입니다. 영숫자, 대시, 밑줄을 포함하여 1-128자로 지정할 수 있습니다. 작업 그룹을 만든 후에는 작업 그룹 이름을 변경할 수 없습니다. 하지만 동일한 설정으로 다른 이름을 가진 새로운 작업 그룹을 만들 수 있습니다.
- 작업 그룹에서 실행 중인 모든 쿼리에 적용되는 설정입니다. 다음과 같은 설정이 해당됩니다.
  - 이 작업 그룹에서 실행하는 모든 쿼리의 쿼리 결과를 저장하기 위한 Amazon S3의 위치. 작업 그룹을 생성할 때 이미 존재하는 위치를 지정해야 합니다. Amazon S3 버킷 생성에 대한 자세한 내용은 버킷 생성을 참조하세요.
  - 쿼리 결과에 대한 버킷 소유자 제어 - Amazon S3 쿼리 결과 버킷의 소유자가 버킷에 기록된 새 객체를 완벽하게 제어할 수 있는지 여부입니다. 예를 들어, 쿼리 결과 위치가 다른 계정이 소유한 경우 쿼리 결과에 대한 소유권 및 전체 제어 권한을 다른 계정에 부여할 수 있습니다.
  - 예상 버킷 소유자 - 쿼리 결과 버킷의 소유자가 될 것으로 예상되는 AWS 계정의 ID입니다. 이는 추가 보안 조치입니다. 버킷 소유자의 계정 ID가 여기에서 지정한 ID와 일치하지 않으면 버킷으로의 출력 시도가 실패합니다. 자세한 내용은 Amazon S3 사용 설명서의 버킷 소유자 조건을 사용하여 버킷 소유권 확인을 참조하세요.

    > **Note**
    >
    > 예상 버킷 소유자 설정은 Athena 쿼리 결과에 대해 지정한 Amazon S3 출력 위치에만 적용됩니다. 외부 Amazon S3 버킷의 데이터 소스 위치, `CTAS` 및 `INSERT INTO` 대상 테이블 위치, `UNLOAD` 문 출력 위치, 연합 쿼리의 유출 버킷 작업 또는 다른 계정의 테이블에 대해 실행되는 `SELECT` 쿼리 등의 다른 Amazon S3 위치에는 적용되지 않습니다.

  - 암호화 설정(모든 작업 그룹 쿼리에 대해 암호화를 사용할 경우). 작업 그룹에서 일부 쿼리가 아닌 모든 쿼리만 암호화할 수 있습니다. 암호화되거나 암호화되지 않는 쿼리를 포함하는 작업 그룹을 따로 만드는 것이 좋습니다.

또한 작업 그룹은 클라이언트 측 설정을 재정의할 수 있습니다. 작업 그룹을 릴리스하기 전에 JDBC 또는 ODBC 드라이버나 Athena 콘솔의 속성(Properties) 탭에서 결과 위치 및 암호화 옵션을 파라미터로 지정할 수 있습니다. 또는 API 작업을 통해 직접 이러한 설정을 지정할 수도 있습니다. 이러한 설정을 "클라이언트 측 설정"이라고 합니다. 작업 그룹을 사용하면 작업 그룹 수준에서 이러한 설정을 구성하

여 클라이언트 수준에서 사용할 수 있는 옵션을 제어할 수 있습니다. 또한 작업 그룹 수준 설정을 적용하면 사용자가 클라이언트 측 설정을 개별적으로 구성하지 않아도 됩니다. 작업 그룹에 대해 클라이언트 측 설정 재정의 옵션을 선택하면 쿼리에서 작업 그룹 설정을 사용하고 클라이언트 측 설정을 무시합니다.

Override Client-Side Settings(클라이언트 측 설정 재정의)를 선택하면 콘솔에서 해당 설정이 변경되었다고 사용자에게 알립니다. 작업 그룹 설정을 이렇게 적용할 경우 사용자는 해당 클라이언트 측 설정을 생략할 수 있습니다. 그러면 클라이언트 측 설정이 있더라도 콘솔에서 실행되는 쿼리는 작업 그룹의 설정을 사용합니다. 또한 작업 그룹의 쿼리가 AWS CLI, API 작업 또는 JDBC 드라이버나 ODBC 드라이버를 통해 실행되는 경우 쿼리 결과 위치 및 암호화와 같은 클라이언트 측 설정이 작업 그룹 설정에 의해 재정의됩니다. 작업 그룹에 사용되는 설정을 확인하려면 작업 그룹의 세부 정보를 확인하세요.

작업 그룹의 쿼리에 대해 쿼리 한도를 설정할 수도 있습니다.

클라이언트 측 설정보다 우선하는 작업 그룹 설정

작업 그룹 생성 및 작업 그룹 편집 대화 상자에는 Override client-side settings(클라이언트 측 설정 재정의)라는 필드가 있습니다. 이 필드는 기본적으로 선택 취소되어 있습니다. 이 필드의 선택 여부에 따라 Athena는 다음을 수행합니다.

- 클라이언트 측 설정 재정의를 선택하지 않으면 작업 그룹 설정이 클라이언트 수준에서 적용되지 않습니다. 작업 그룹에 대해 클라이언트 측 설정 재정의 옵션이 선택되지 않으면 Athena는 이 작업 그룹에서 실행하는 모든 쿼리에 대해 클라이언트 설정(쿼리 결과 위치 설정, 예상되는 버킷 소유자, 암호화, 쿼리 결과 버킷에 쓴 객체 제어 포함)을 사용합니다. 각 사용자는 콘솔의 설정 메뉴에서 자체 설정을 지정할 수 있습니다. 클라이언트 측 설정이 설정되지 않은 경우 작업 그룹 전체 설정이 적용됩니다. AWS CLI, API 작업 또는 JDBC 드라이버와 ODBC 드라이버를 사용하여 클라이언트 측 설정을 재정의하지 않는 작업 그룹에서 쿼리를 실행하는 경우 쿼리는 쿼리에 지정된 설정을 사용합니다.
- 클라이언트 측 설정 재정의를 선택하면 작업 그룹 설정이 작업 그룹의 모든 클라이언트에 대해 작업 그룹 수준에서 적용됩니다. 작업 그룹에 대해 클라이언트 측 설정 재정의 옵션이 선택되면 Athena는 이 작업 그룹에서 실행하는 모든 쿼리에 대해 작업 그룹 설정(쿼리 결과 위치 설정, 예상되는 버킷 소유자, 암호화, 쿼리 결과 버킷에 쓴 객체 제어 포함)을 사용합니다. 작업 그룹 설정은 콘솔, API 작업 또는 JDBC 드라이버나 ODBC 드라이버를 사용할 때 쿼리에 지정하는 모든 클라이언트 측 설정을 재정의합니다.

  클라이언트 측 설정을 재정의하는 경우 다음에 사용자 또는 작업 그룹 사용자가 Athena 콘솔을 열 때 Athena는 작업 그룹의 쿼리가 작업 그룹의 설정을 사용함을 알리고 이 변경 사항을 승인하라는 메시지를 표시합니다.

> **Important**
>
> API 작업, AWS CLI 또는 JDBC 드라이버와 ODBC 드라이버를 사용하여 클라이언트 측 설정을 재정의하는 작업 그룹에서 쿼리를 실행하는 경우 쿼리에서 클라이언트 측 설정을 생략하거나 작업 그룹 설정에 맞게 업데이트해야 합니다. 쿼리에서 클라이언트 측 설정을 지정하지만 설정을 재정의하는 작업 그룹에서 실행하면 쿼리는 실행되지만 작업 그룹 설정이 사용됩니다. 작업 그룹 설정 보기에 대한 자세한 내용은 작업 그룹의 세부 정보 보기 섹션을 참조하세요.

작업 관리

https://console.aws.amazon.com/athena/에서 다음 작업을 수행할 수 있습니다.

문	설명
작업 그룹 만들기	새로운 작업 그룹을 만듭니다.
작업 그룹 편집	작업 그룹을 편집하고 설정을 편집합니다. 작업 그룹의 이름은 변경할 수 없지만 동일한 설정으로 다른 이름을 사용하여 새 작업 그룹을 만들 수 있습니다.
작업 그룹의 세부 정보 보기	이름, 설명, 데이터 사용 제한, 쿼리 결과 위치, 예상 쿼리 결과 버킷 소유자 및 암호화와 같은 작업 그룹의 세부 정보를 봅니다. Override client-side settings(클라이언트 측 설정 재정의)를 선택한 경우 이 작업 그룹에서 해당 설정을 강제 적용할지 여부를 지정할 수도 있습니다.
작업 그룹 삭제	작업 그룹을 삭제합니다. 작업 그룹을 삭제할 경우 쿼리 기록, 저장된 쿼리, 작업 그룹의 설정, 쿼리당 데이터 한도 제어 등도 삭제됩니다. 작업 그룹 전역의 데이터 한도 제어는 CloudWatch에서 유지되며 개별적으로 삭제할 수 있습니다. 기본 작업 그룹은 삭제할 수 없습니다.
작업 그룹 전환	액세스 권한을 가진 작업 그룹 간에 전환합니다.

문	설명
작업 그룹 간에 저장된 쿼리 복사	작업 그룹 간에 저장된 쿼리를 복사합니다. 예를 들어 미리 보기 작업 그룹에서 쿼리를 생성했는데 미리 보기가 아닌 작업 그룹에서도 사용하도록 하려 할 경우에 이 기능이 필요할 수 있습니다.
작업 그룹 활성화 및 비활성화	작업 그룹을 활성화거나 비활성화합니다. 작업 그룹이 비활성화된 경우 사용자가 쿼리를 실행하거나 새로운 이름 지정 쿼리를 만들 수 없습니다. 액세스 권한이 있는 경우 지표, 데이터 사용량 한도 제어, 작업 그룹의 설정, 쿼리 기록 및 저장된 쿼리는 계속 볼 수 있습니다.
쿼리를 실행할 작업 그룹 지정	쿼리를 실행하려면 먼저 Athena에서 사용할 작업 그룹을 지정해야 합니다. 작업 그룹에 대해 사용 권한이 있어야 합니다.
IAM Identity Center 인증을 사용하는 Athena 작업 그룹 생성	Athena에서 IAM Identity Center ID를 사용하려면 IAM Identity Center 지원 작업 그룹을 생성해야 합니다. 작업 그룹을 생성한 후 IAM Identity Center 콘솔 또는 API를 사용하여 작업 그룹에 IAM Identity Center 사용자 또는 그룹을 할당할 수 있습니다.

작업 그룹 만들기

작업 그룹을 만들려면 `CreateWorkgroup` API 작업에 대한 권한이 필요합니다. 작업 그룹 및 태그에 대한 액세스 및 작업 그룹 액세스를 위한 IAM 정책 단원을 참조하세요. 태그를 추가할 경우 `TagResource`에 대한 권한도 추가해야 합니다. 작업 그룹에 대한 태그 정책 예제 섹션을 참조하세요.

콘솔에서 작업 그룹을 생성하려면

1. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

2. Athena 콘솔 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
3. 작업 그룹 페이지에서 작업 그룹 생성을 선택합니다.
4. 작업 그룹 생성(Create workgroup) 페이지에서 다음과 같이 필드에 값을 입력합니다.

필드	설명
작업 그룹 이름	필수 사항입니다. 작업 그룹의 고유한 이름을 입력합니다. 1~128자를 사용합니다. (A-Z,a-z,0-9,_,-,.). 이 이름은 변경할 수 없습니다.
설명	선택 사항입니다. 작업 그룹에 대한 설명을 입력합니다. 최대 1024까지 입력할 수 있습니다.
Choose the type of engine(엔진 유형 선택)	Amazon S3의 데이터에 대해 임시 SQL 쿼리를 실행하거나 사전 구축된 데이터 소스 커넥터를 사용하여 Amazon S3 외부의 다양한 데이터 소스에서 페더레이션 쿼리를 실행하려면 Athena SQL을 선택합니다. Athena 쿼리 편집기, AWS CLI 또는 Athena API를 사용하여 쿼리를 실행할 수 있습니다. Python과 Apache Spark를 사용하여 Jupyter Notebook 애플리케이션을 생성, 편집, 실행하려면 Apache Spark를 선택합니다. Jupyter Notebook에는 코드, 텍스트, 마크다운, 수학, 도표 및 리치 미디어가 나열된 셀 목록이 포함되어 있습니다. 셀은 Athena의 대화형 노트북 세션에서 계산된 순서대로 실행됩니다. Spark 지원 작업 그룹 생성

필드	설명
	및 구성에 대한 자세한 내용은 Athena에서 Spark 지원 작업 그룹 생성 단원을 참조하세요. 작업 그룹을 생성한 후 분석 엔진을 업그레이드할 수 있지만(예: Athena 엔진 버전 2에서 Athena 엔진 버전 3으로) 엔진 유형을 변경할 수는 없습니다. 예를 들어 Athena 엔진 버전 3 작업 그룹을 PySpark 엔진 버전 3 작업 그룹으로 변경할 수 없습니다.
쿼리 엔진 업데이트	새 Athena 엔진 버전이 릴리스될 때 작업 그룹을 업데이트하는 방법을 선택합니다. 사용자는 Athena가 작업 그룹을 업그레이드 시기를 결정하도록 하거나 엔진 버전을 수동으로 선택할 수 있습니다. 자세한 내용은 Athena 엔진 버전 관리 단원을 참조하십시오.
인증 모드	작업 그룹에 IAM 인증 또는 페더레이션을 사용하려면 AWS Identity and Access Management(IAM)를 선택합니다. Microsoft Active Directory와 같은 SAML 2.0 ID 제공업체 사용자 및 그룹 등의 인력 ID를 지원하려면 IAM Identity Center를 선택합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 Trusted identity propagation across applications를 참조하십시오.
IAM Identity Center 액세스를 위한 서비스 역할	Athena가 사용자를 대신하여 IAM Identity Center에 액세스하려면 IAM 권한이 필요합니다. IAM 서비스 역할에 대한 자세한 내용은 IAM 사용 설명서의 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하십시오.

필드	설명
쿼리 결과의 위치 (Location of query result)	선택 사항입니다. Amazon S3 버킷 경로 또는 접두사를 입력합니다. 버킷과 접두사를 지정하려면 지정하기 전에 해당 버킷과 접두사가 존재해야 합니다. **Note** 콘솔에서 쿼리를 실행할 경우 쿼리 결과 위치 지정은 선택 사항입니다. 작업 그룹에 대해 또는 설정에서 이를 지정하지 않을 경우 Athena는 기본 쿼리 결과 위치를 사용합니다. API 또는 드라이버를 사용해 쿼리를 실행할 경우 최소한 다음 두 곳 중 하나에서 쿼리 결과 위치를 지정해야 합니다. 즉, 개인 쿼리의 경우 OutputLocation을 통해 지정하고, 작업 그룹 쿼리의 경우 WorkGroupConfiguration을 통해 지정합니다.
예상 버킷 소유자 (Expected bucket owner)	선택 사항입니다. 출력 위치 버킷의 소유자가 될 것으로 예상되는 AWS 계정의 ID를 입력합니다. 이는 추가 보안 조치입니다. 버킷 소유자의 계정 ID가 여기에서 지정한 ID와 일치하지 않으면 버킷으로의 출력 시도가 실패합니다. 자세한 내용은 Amazon S3 사용 설명서의 버킷 소유자 조건을 사용하여 버킷 소유권 확인을 참조하세요. **Note** 예상 버킷 소유자 설정은 Athena 쿼리 결과에 대해 지정한 Amazon S3 출력 위치에만 적용됩니다. 외부 Amazon S3 버킷의 데이터 소스 위치, `CTAS` 및 `INSERT INTO` 대상 테이블 위치, `UNLOAD` 문 출력 위치, 연합 쿼리의 유출 버킷 작업 또는 다른 계정의 테이블에 대해 실행되는 `SELECT` 쿼리 등의 다른 Amazon S3 위치에는 적용되지 않습니다.

필드	설명
버킷 소유자에게 쿼리 결과에 대한 전체 제어 권한 할당	이 필드는 기본적으로 선택 취소되어 있습니다. 이 옵션을 선택하고 쿼리 결과 위치 버킷에 ACL이 사용 설정된 경우, 버킷 소유자에게 쿼리 결과에 대한 전체 제어 액세스 권한을 부여합니다. 예를 들어, 쿼리 결과 위치가 다른 계정이 소유한 경우 이 옵션을 사용하여 소유권을 부여하고 쿼리 결과에 대한 모든 권한을 다른 계정에 부여할 수 있습니다. 버킷의 S3 객체 소유권 설정이 버킷 소유자 기본인 경우 버킷 소유자는 이 작업 그룹에서 작성된 모든 쿼리 결과 객체도 소유합니다. 예를 들어, 외부 계정의 작업 그룹에서 이 옵션을 활성화하고 쿼리 결과 위치를 계정의 버킷 소유자 기본의 S3 객체 소유권 설정을 가진 Amazon S3 버킷으로 설정한 경우, 외부 작업 그룹의 쿼리 결과에 대한 전체 제어 권한을 갖습니다. 쿼리 결과 버킷의 S3 객체 소유권 설정이 버킷 소유자 시행인 경우 이 옵션은 아무 영향이 없습니다. 자세한 내용은 Amazon S3 사용 설명서의 객체 소유권 설정 섹션을 참조하세요.
쿼리 결과 암호화	선택 사항입니다. Amazon S3에 저장되는 결과를 암호화합니다. 선택할 경우 작업 그룹의 모든 쿼리가 암호화됩니다. 선택할 경우 암호화 유형, 암호화 키를 선택하고 KMS 키 ARN을 입력합니다. 키가 없는 경우 AWS KMS 콘솔을 열고 키를 만드세요. 자세한 정보는 AWS Key Management Service 개발자 안내서의 키 생성을 참조하세요.

필드	설명
encryption_type 을 최소 암호화로 설정	선택 사항입니다. 작업 그룹의 모든 사용자에서 쿼리 결과에 최소 유형의 암호화를 적용하려면 이 옵션을 선택합니다. 이 옵션을 선택하면 암호화 유형의 계층 구조를 포함하는 테이블이 표시됩니다. 이 테이블에는 특정 암호화 유형을 최소 암호화로 지정할 때 작업 그룹 사용자가 사용할 수 있는 암호화 유형도 표시됩니다. 이 옵션을 사용하려면 클라이언트 측 설정 재정의를 선택하지 않아야 합니다. 자세한 내용은 작업 그룹에 대한 최소 암호화 구성 단원을 참조하십시오.
S3 Access Grants 활성화	IAM Identity Center를 인증 모드로 선택하면 이 필드가 기본적으로 선택됩니다. 이 옵션을 선택하면 Amazon S3 위치에 IAM Identity Center 사용자 또는 그룹 기반 권한이 적용됩니다.
사용자 ID 기반 S3 접두사 생성	이 옵션을 선택하면 Athena에서 쿼리 결과를 저장할 때 Amazon S3 접두사를 생성합니다. 접두사는 사용자의 IAM Identity Center 사용자 ID를 기반으로 합니다.
Publish query metrics to CloudWatch(CloudWatch에 쿼리 지표 게시)	이 필드는 기본적으로 선택됩니다. CloudWatch에 쿼리 지표를 게시합니다. CloudWatch 지표를 사용한 Athena 쿼리 모니터링 섹션을 참조하세요.
Override client-side settings(클라이언트 측 설정 재정의)	이 필드는 기본적으로 선택 취소되어 있습니다. 이 필드를 선택할 경우, 작업 그룹 설정이 작업 그룹의 모든 쿼리에 적용되고 클라이언트 측 설정을 재정의합니다. 자세한 내용은 클라이언트 측 설정보다 우선하는 작업 그룹 설정 단원을 참조하십시오.
Requester Pays S3 buckets(요청자 지불 S3 버킷)	선택 사항입니다. 작업 그룹 사용자가 요청자 지불로 구성되어 있는 Amazon S3 버킷에 저장된 데이터를 쿼리할 경우 Turn on queries on requester pays buckets in Amazon S3(Amazon S3에서 요청자 지불 버킷에 대한 쿼리 켜기)를 선택합니다. 쿼리와 관련하여 적용 가능한 데이터 액세스 및 데이터 전송 요금은 쿼리를 실행하는 사용자의 계정에 청구됩니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 요청자 지불 버킷을 참조하세요.

필드	설명
Manage per query data usage control(쿼리별 데이터 사용량 컨트롤 관리)	선택 사항입니다. 쿼리가 스캔할 수 있는 최대 데이터 양에 대한 제한을 설정합니다. 작업 그룹에 대해 쿼리 제한당 하나만 설정할 수 있습니다. 제한은 작업 그룹의 모든 쿼리에 적용되며 쿼리가 제한을 초과할 경우 취소됩니다. 자세한 내용은 데이터 사용량 제어 한도 설정 단원을 참조하십시오.
워크그룹 데이터 사용량 알림(Workgroup data usage alerts)	선택 사항입니다. 이 작업 그룹에서 실행되는 쿼리가 특정 기간 내에 지정된 양의 데이터를 스캔할 때 여러 경고 임계값을 설정합니다. 알림은 Amazon CloudWatch 경보를 사용하여 구현되며 작업 그룹의 모든 쿼리에 적용됩니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch 경보 사용을 참조하세요.
태그	선택 사항입니다. 작업 그룹에 태그를 한 개 이상 추가합니다. 태그는 Athena 작업 그룹 리소스에 할당하는 레이블입니다. 태그는 키와 값으로 구성됩니다. AWS 태깅 모범 사례를 사용하여 일관된 태그 세트를 생성하고 작업 그룹을 용도, 소유자 또는 환경별로 분류합니다. IAM 정책에서 태그를 사용하여 결제 비용을 관리할 수도 있습니다. 동일한 작업 그룹에 중복된 태그 키를 사용하지 마세요. 자세한 내용은 the section called “ 리소스에 태그 지정” 단원을 참조하십시오.

5. 작업 그룹 생성을 선택합니다. 작업 그룹(Workgroups) 페이지의 목록에 해당 작업 그룹이 나타납니다.

CreateWorkGroup API 작업을 사용하여 작업 그룹을 생성할 수도 있습니다.

> **Important**
>
> 작업 그룹을 만들었으면 작업 그룹 액세스를 위한 IAM 정책 IAM을 생성하여 작업 그룹 관련 작업의 실행을 허용합니다.

작업 그룹 편집

작업 그룹을 편집하려면 `UpdateWorkgroup` API 작업에 대한 권한이 필요합니다. 작업 그룹 및 태그에 대한 액세스 및 작업 그룹 액세스를 위한 IAM 정책 단원을 참조하세요. 태그를 추가하거나 편집할

경우 TagResource에 대한 권한도 있어야 합니다. 작업 그룹에 대한 태그 정책 예제 섹션을 참조하세요.

콘솔에서 작업 그룹을 편집하려면

1. Athena 콘솔 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
2. 작업 그룹(Workgroups) 페이지에서 편집할 작업 그룹의 버튼을 선택합니다.
3. 작업(Actions), 편집(Edit)을 선택합니다.
4. 필요 시 이 필드를 변경합니다. 필드 목록을 보려면 작업 그룹 생성을 참조하세요. 작업 그룹 이름을 제외한 모든 필드를 변경할 수 있습니다. 이름을 변경해야 할 경우, 동일한 설정으로 작업 그룹을 만든 후 새 이름을 지정하세요.
5. Save changes(변경 사항 저장)를 선택합니다. 작업 그룹(Workgroups) 페이지의 목록에 업데이트된 작업 그룹이 나타납니다.

작업 그룹의 세부 정보 보기

각 작업 그룹에 대해 해당 세부 정보를 볼 수 있습니다. 세부 정보에는 작업 그룹의 이름, 설명, 사용 설정 또는 사용 중지 여부 및 작업 그룹에서 실행되는 쿼리에 사용되는 설정(쿼리 결과 위치 및 암호화 구성 등)이 표시됩니다. 작업 그룹에 데이터 사용 한도가 있을 경우 이 설정도 표시됩니다.

작업 그룹의 세부 정보를 보려면

1. Athena 콘솔 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
2. 작업 그룹(Workgroups) 페이지에서 보려는 작업 그룹의 링크를 선택합니다. 작업 그룹에 대한 개요 세부 정보(Overview Details) 페이지가 표시됩니다.

작업 그룹 삭제

작업 그룹 삭제 권한이 있으면 작업 그룹을 삭제할 수 있습니다. 기본 작업 그룹은 삭제할 수 없습니다.

권한이 있는 경우 언제든 빈 작업 그룹을 삭제할 수 있습니다. 저장된 쿼리가 들어 있는 작업 그룹도 삭제할 수 있습니다. 이 경우 작업 그룹 삭제를 진행하기 전에 Athena가 저장된 쿼리의 삭제를 경고합니다.

작업 그룹 내에서 작업 그룹을 삭제할 경우 콘솔에서 기본 작업 그룹으로 포커스가 전환됩니다. 해당 작업 그룹에 대해 액세스 권한이 있을 경우 쿼리를 실행하고 해당 설정을 볼 수 있습니다.

작업 그룹을 삭제할 경우 작업 그룹의 설정 및 쿼리당 데이터 한도 제어가 삭제됩니다. 작업 그룹 전역의 데이터 한도 제어는 CloudWatch에서 유지되며 필요할 경우 삭제할 수 있습니다.

> **Important**
>
> 작업 그룹을 삭제하기 전에 작업 그룹의 사용자가 쿼리 실행을 계속할 수 있는 다른 작업 그룹에 속해 있는지 확인해야 합니다. 사용자의 IAM 정책에서 사용자에게 오직 이 작업 그룹에서만 쿼리를 실행하도록 허용한 경우 이 작업 그룹을 삭제하면 해당 사용자는 더 이상 쿼리를 실행할 수 있는 권한이 없어집니다. 자세한 내용은 Example policy for running queries in the primary workgroup 단원을 참조하세요.

콘솔에서 작업 그룹을 삭제하려면

1. Athena 콘솔 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
2. 작업 그룹(Workgroups) 페이지에서 삭제할 작업 그룹의 버튼을 선택합니다.
3. 작업, 삭제를 선택합니다.
4. 작업 그룹 삭제(Delete workgroup) 확인 프롬프트에서 작업 그룹의 이름을 입력한 다음 삭제(Delete)를 선택합니다.

API 작업을 사용하여 작업 그룹을 삭제하려면 `DeleteWorkGroup` 작업을 사용합니다.

작업 그룹 전환

작업 그룹에 대해 권한이 있을 경우 한 작업 그룹에서 다른 작업 그룹으로 전환할 수 있습니다.

각 작업 그룹에서 쿼리 탭을 10개까지 열 수 있습니다. 작업 그룹 간에 전환할 경우 쿼리 탭은 최대 3개의 작업 그룹에 대해 열린 채로 유지됩니다.

작업 그룹 전환

1. Athena 콘솔에서 오른쪽 상단의 작업 그룹(Workgroup) 옵션을 사용하여 작업 그룹을 선택합니다.
2. 작업 그룹 ***workgroup-name*** 설정(Workgroup workgroup-name settings) 대화 상자가 나타나면 승인(Acknowledge)을 선택합니다.

작업 그룹(Workgroup) 옵션은 전환한 작업 그룹의 이름을 표시합니다. 이제 이 작업 그룹에서 쿼리를 실행할 수 있습니다.

작업 그룹 간에 저장된 쿼리 복사

현재 Athena 콘솔에는 저장된 쿼리를 한 작업 그룹에서 다른 작업 그룹으로 직접 복사할 수 있는 옵션이 없지만 다음 절차를 사용하여 수동으로 동일한 작업을 수행할 수 있습니다.

작업 그룹 간에 저장된 쿼리를 복사하려면

1. Athena 콘솔에서, 쿼리를 복사해 올 작업 그룹에서 저장된 쿼리(Saved queries) 탭을 클릭합니다.
2. 복사하려는 저장된 쿼리의 링크를 선택합니다. Athena는 쿼리 편집기에서 쿼리를 엽니다.
3. 쿼리 편집기에서 쿼리 텍스트를 선택한 후 **Ctrl+C**를 눌러 복사합니다.
4. 대상 작업 그룹으로 전환하거나, 작업 그룹을 생성한 다음 대상 작업 그룹으로 전환합니다.
5. 쿼리 편집기에서 새 탭을 연 다음 **Ctrl+V**를 눌러 텍스트를 새 탭에 붙여 넣습니다.
6. 쿼리 편집기에서 다른 이름으로 저장(Save as)을 클릭하여 대상 작업 그룹에 쿼리를 저장합니다.
7. 이름 선택(Choose a name) 대화 상자에서 쿼리의 이름과 설명(선택 사항)을 입력합니다.
8. Save(저장)를 선택합니다.

작업 그룹 활성화 및 비활성화

권한이 있는 경우 콘솔에서 또는 API 작업이나 JDBC 및 ODBC 드라이버를 사용하여 작업 그룹을 활성화하거나 비활성화할 수 있습니다.

작업 그룹을 활성화거나 비활성화하려면

1. Athena 콘솔 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
2. 작업 그룹(Workgroups) 페이지에서 작업 그룹의 링크를 선택합니다.
3. 오른쪽 상단에서 작업 그룹 활성화(Enable workgroup) 또는 작업 그룹 비활성화(Disable workgroup)를 선택합니다.
4. 확인 프롬프트에서 활성화(Enable) 또는 비활성화(Disable)를 선택합니다. 작업 그룹을 비활성화한 경우 해당 작업 그룹의 사용자는 그 작업 그룹에서 쿼리를 실행하거나 새로운 이름 지정 쿼리를 만들 수 없습니다. 작업 그룹을 활성화한 경우, 사용자는 해당 작업 그룹을 사용하여 쿼리를 실행할 수 있습니다.

쿼리를 실행할 작업 그룹 지정

사용할 작업 그룹을 지정하려면 작업 그룹에 대한 권한이 있어야 합니다.

사용할 작업 그룹 지정

1. 사용하려는 작업 그룹에서 쿼리를 실행할 수 있는 권한이 있는지 확인합니다. 자세한 내용은 the section called “ 작업 그룹 액세스를 위한 IAM 정책” 단원을 참조하십시오.
2. 작업 그룹을 지정하려면 다음 옵션 중 하나를 사용합니다.

   - Athena 콘솔을 사용하는 경우 작업 그룹을 전환하여 작업 그룹을 설정합니다.
   - Athena API 작업을 사용할 경우 API 작업에 작업 그룹 이름을 지정합니다. 예를 들어 다음과 같이 StartQueryExecution에 작업 그룹 이름을 설정할 수 있습니다.

     ```
     StartQueryExecutionRequest startQueryExecutionRequest = new
      StartQueryExecutionRequest()
                  .withQueryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
                  .withQueryExecutionContext(queryExecutionContext)
                  .withWorkGroup(WorkgroupName)
     ```

   - JDBC 또는 ODBC 드라이버를 사용할 경우 Workgroup 구성 파라미터를 사용하여 연결 문자열에 작업 그룹 이름을 설정합니다. 드라이버는 작업 그룹 이름을 Athena에 전달합니다. 다음 예제에서와 같이 연결 문자열에 작업 그룹 파라미터를 지정합니다.

     ```
     jdbc:awsathena://AwsRegion=<AWSREGION>;UID=<ACCESSKEY>;
     PWD=<SECRETKEY>;S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/<athena-
     output>-<AWSREGION>/;
     Workgroup=<WORKGROUPNAME>;
     ```

작업 그룹에 대한 최소 암호화 구성

Athena SQL 작업 그룹의 관리자는 Amazon S3에서 작업 그룹의 모든 쿼리 결과에 대해 최소 수준의 암호화를 적용할 수 있습니다. 이 기능을 사용하면 쿼리 결과가 암호화되지 않은 상태로 Amazon S3 버킷에 저장되지 않도록 합니다.

최소 암호화가 활성화된 작업 그룹의 사용자가 쿼리를 제출하면 사용자가 구성한 최소 수준 또는 사용 가능한 경우 더 높은 수준으로만 암호화를 설정할 수 있습니다. Athena에서는 사용자가 쿼리를 실행할 때 지정된 수준 또는 작업 그룹에 설정된 수준에서 쿼리 결과를 암호화합니다.

사용 가능한 수준은 다음과 같습니다.

- 기본 - Amazon S3 관리형 키를 사용한 Amazon S3 서버 측 암호화(SSE-S3).
- 중급 - KMS 관리형 키를 사용한 서버 측 암호화(SSE-KMS).

- 고급 - KMS 관리형 키를 사용한 클라이언트 측 암호화(CSE_KMS).

고려 사항 및 제한

- Apache Spark 지원 작업 그룹에서는 최소 암호화 기능을 사용할 수 없습니다.
- 최소 암호화 기능은 작업 그룹에서 클라이언트 측 설정 재정의 옵션을 활성화하지 않은 경우에만 작동합니다.
- 작업 그룹에 클라이언트 측 설정 재정의 옵션이 활성화된 경우 작업그룹 암호화 설정이 우선하며 최소 암호화 설정은 영향을 주지 않습니다.
- 이 기능을 활성화하는 데 비용은 들지 않습니다.

작업 그룹에 대한 최소 암호화 활성화

작업 그룹을 생성하거나 업데이트할 때 Athena SQL 작업 그룹의 쿼리 결과에 대해 최소 암호화 수준을 활성화할 수 있습니다. 이를 위해 Athena 콘솔, Athena API 또는 AWS CLI를 사용할 수 있습니다.

Athena 콘솔을 사용하여 최소 암호화 활성화

Athena 콘솔을 사용하여 작업 그룹의 생성 또는 편집을 시작하려면 작업 그룹 생성 또는 작업 그룹 편집을 참조하세요. 작업 그룹을 구성하는 경우 다음 단계를 사용하여 최소 암호화를 활성화합니다.

작업 그룹 쿼리 결과에 대해 최소 암호화 수준을 구성하려면

1. 추가 구성 섹션에서 설정을 확장합니다.
2. 클라이언트 측 설정 재정의 옵션을 지우거나 선택되지 않았는지 확인합니다.
3. 추가 구성 섹션에서 쿼리 결과 구성을 확장합니다.
4. 쿼리 결과 암호화 옵션을 선택합니다.
5. 암호화 유형에서 작업 그룹 쿼리 결과에 대해 Athena에서 사용할 암호화 방법을 선택합니다(SSE_S3, SSE_KMS 또는 CSE_KMS). 이러한 암호화 유형은 기본, 중급, 고급 보안 수준에 해당합니다.
6. 선택한 암호화 방법을 모든 사용자에 대한 최소 암호화 수준으로 적용하려면 최소 암호화로 ***encryption_method*** 설정을 선택합니다.

   이 옵션을 선택하면 선택한 암호화 유형이 최소 암호화로 구성될 때 사용자에게 허용되는 암호화 계층 구조 및 암호화 수준이 테이블에 표시됩니다.
7. 작업 그룹을 생성하거나 작업 그룹 구성을 업데이트한 후 작업 그룹 생성 또는 변경 사항 저장을 선택합니다.

Athena API 또는 AWS CLI를 사용하여 최소 암호화를 활성화하려면

CreateWorkGroup 또는 UpdateWorkGroup API를 사용하여 Athena SQL 작업 그룹을 생성하거나 업데이트하는 경우 EnforceWorkGroupConfiguration을 `false`로, EnableMinimumEncryptionConfiguration을 `true`로 설정하고 EncryptionOption을 사용하여 암호화 유형을 지정합니다.

AWS CLI에서 `create-work-group` 또는 `update-work-group` 명령과 함께 `--configuration` 또는 `--configuration-updates` 파라미터를 사용하고 API의 옵션에 해당하는 옵션을 지정합니다.

IAM Identity Center 지원 Athena 작업 그룹 사용

AWS IAM Identity Center의 신뢰할 수 있는 ID 전파 기능을 사용하면 AWS 분석 서비스 전반에서 인력 ID를 사용할 수 있습니다. 또한 서비스별 ID 제공업체 구성 또는 IAM 역할 설정을 수행할 필요가 없습니다.

IAM Identity Center를 사용하면 인력 ID(인력 사용자)의 로그인 보안을 관리할 수 있습니다. IAM Identity Center를 사용하면 인력 사용자를 생성하거나 연결하고 인력 사용자의 모든 AWS 계정 및 애플리케이션에 대한 액세스를 중앙에서 관리하는 기능을 한 곳에서 이용할 수 있습니다. 다중 계정 권한을 사용하여 이러한 사용자에게 AWS 계정 대한 액세스 권한을 할당할 수 있습니다. 애플리케이션 할당을 사용하여 IAM Identity Center 지원 애플리케이션, 클라우드 애플리케이션 및 고객 Security Assertion Markup Language(SAML 2.0) 애플리케이션에 대한 액세스 권한을 사용자에게 할당할 수 있습니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 Trusted identity propagation across applications를 참조하십시오.

현재 신뢰할 수 있는 ID 전파에 대한 Athena SQL 지원을 통해 Amazon EMR Studio와 EMR Studio의 Athena SQL 인터페이스에 동일한 ID를 사용할 수 있습니다. EMR Studio에서 Athena SQL과 함께 IAM Identity Center ID를 사용하려면 Athena에서 IAM Identity Center 지원 작업 그룹을 생성해야 합니다. 그런 다음 IAM Identity Center 콘솔 또는 API를 사용하여 IAM Identity Center 지원 Athena 작업 그룹에 IAM Identity Center 사용자 또는 그룹을 할당할 수 있습니다. 신뢰할 수 있는 ID 전파를 사용하는 Athena 작업 그룹의 쿼리는 IAM Identity Center가 활성화된 EMR Studio의 Athena SQL 인터페이스에서 실행해야 합니다.

고려 사항 및 제한

Amazon Athena에서 신뢰할 수 있는 ID 전파를 사용하는 경우 다음 사항을 고려하십시오.

- 작업 그룹이 생성된 후에는 작업 그룹의 인증 방법을 변경할 수 없습니다.
  - IAM Identity Center 지원 작업 그룹을 지원하도록 기존 Athena SQL 작업 그룹을 수정할 수 없습니다.

  - 리소스 수준 IAM 권한이나 ID 기반 IAM 정책을 지원하도록 IAM Identity Center 지원 작업 그룹을 수정할 수 없습니다.
- 신뢰할 수 있는 ID 전파 지원 작업 그룹에 액세스하려면 Athena GetWorkGroup API 작업의 응답으로 반환되는 `IdentityCenterApplicationArn`에 IAM Identity Center 사용자를 할당해야 합니다.
- 신뢰할 수 있는 ID 전파 ID를 사용하도록 Amazon S3 Access Grants를 구성해야 합니다. 자세한 내용은 Amazon S3 사용 설명서의 S3 Access Grants and corporate directory identities를 참조하십시오.
- IAM Identity Center 지원 Athena 작업 그룹을 사용하려면 IAM Identity Center ID를 사용하도록 Lake Formation을 구성해야 합니다. 구성 정보는 AWS Lake Formation 개발자 안내서의 Integrating IAM Identity Center를 참조하십시오.
- 기본적으로 신뢰할 수 있는 ID 전파를 사용하는 작업 그룹에서는 쿼리가 30분 후 시간 초과됩니다. 쿼리 제한 시간 증가를 요청할 수 있지만 신뢰할 수 있는 ID 전파 작업 그룹에서 쿼리를 실행할 수 있는 최대 시간은 1시간입니다.
- 신뢰할 수 있는 ID 전파 작업 그룹의 사용자 또는 그룹 권한 변경 사항이 적용되는 데 최대 1시간이 걸릴 수 있습니다.
- 신뢰할 수 있는 ID 전파를 사용하는 Athena 작업 그룹의 쿼리는 Athena 콘솔에서 직접 실행할 수 없습니다. 이러한 쿼리는 IAM Identity Center가 활성화된 EMR Studio의 Athena 인터페이스에서 실행해야 합니다. EMR Studio에서 Athena를 사용하는 방법에 대한 자세한 내용은 Amazon EMR 관리 안내서의 Use the Amazon Athena SQL editor in EMR Studio를 참조하십시오.
- 신뢰할 수 있는 ID 전파는 다음 Athena 기능과 호환되지 않습니다.
  - `aws:CalledVia` 컨텍스트 키
  - Athena for Spark 작업 그룹
  - Athena API에 대한 연동 액세스
  - Lake Formation과 Athena JDBC 및 ODBC 드라이버를 사용하여 Athena에 대한 연동 액세스
- 다음 AWS 리전에서만 Athena와 함께 신뢰할 수 있는 ID 전파를 사용할 수 있습니다.
  - `us-east-2` - 미국 동부(오하이오)
  - `us-east-1` - 미국 동부(버지니아 북부)
  - `us-west-1` - 미국 서부(캘리포니아 북부)
  - `us-west-2` - 미국 서부(오레곤)
  - `af-south-1` – 아프리카(케이프타운)
  - `ap-east-1` – 아시아 태평양(홍콩)

- ap-southeast-3 – 아시아 태평양(자카르타)
- ap-south-1 – 아시아 태평양(뭄바이)
- ap-northeast-3 – 아시아 태평양(오사카)
- ap-northeast-2 – 아시아 태평양(서울)
- ap-southeast-1 – 아시아 태평양(싱가포르)
- ap-southeast-2 – 아시아 태평양(시드니)
- ap-northeast-1 – 아시아 태평양(도쿄)
- ca-central-1 - 캐나다(중부)
- eu-central-1 – 유럽(프랑크푸르트)
- eu-west-1 – 유럽(아일랜드)
- eu-west-2 – 유럽(런던)
- eu-south-1 – 유럽(밀라노)
- eu-west-3 – 유럽(파리)
- eu-north-1 – 유럽(스톡홀름)
- me-south-1 – 중동(바레인)
- sa-east-1 – 남아메리카(상파울루)

필요한 권한

Athena 콘솔에서 IAM Identity Center 지원 작업 그룹을 생성하는 관리자의 IAM 사용자는 다음 정책을 연결해야 합니다.

- AmazonAthenaFullAccess 관리형 정책. 세부 정보는 AWS 관리형 정책: AmazonAthenaFullAccess을 참조하세요.
- 다음은 IAM 및 IAM Identity Center 작업을 허용하는 인라인 정책입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "iam:createRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "iam:ListRoles",
```

```
                "iam:PassRole",
                "identitystore:ListUsers",
                "identitystore:ListGroups",
                "identitystore:CreateUser",
                "identitystore:CreateGroup",
                "sso:ListInstances",
                "sso:CreateInstance",
                "sso:DeleteInstance",
                "sso:DescribeUser",
                "sso:DescribeGroup",
                "sso:ListTrustedTokenIssuers",
                "sso:DescribeTrustedTokenIssuer",
                "sso:ListApplicationAssignments",
                "sso:DescribeRegisteredRegions",
                "sso:GetManagedApplicationInstance",
                "sso:GetSharedSsoConfiguration",
                "sso:PutApplicationAssignmentConfiguration",
                "sso:CreateApplication",
                "sso:DeleteApplication",
                "sso:PutApplicationGrant",
                "sso:PutApplicationAuthenticationMethod",
                "sso:PutApplicationAccessScope",
                "sso:ListDirectoryAssociations",
                "sso:CreateApplicationAssignment",
                "sso:DeleteApplicationAssignment",
                "organizations:ListDelegatedAdministrators",
                "organizations:DescribeAccount",
                "organizations:DescribeOrganization",
                "organizations:CreateOrganization",
                "sso-directory:SearchUsers",
                "sso-directory:SearchGroups",
                "sso-directory:CreateUser"
            ],
            "Effect": "Allow",
            "Resource": [
                "*"
            ]
        }
    ]
}
```

IAM Identity Center 지원 Athena 작업 그룹 생성

다음 절차에서는 IAM Identity Center 지원 Athena 작업 그룹 생성과 관련된 단계 및 옵션을 보여줍니다. Athena 작업그룹에 사용할 수 있는 다른 구성 옵션에 대한 설명은 작업 그룹 만들기 섹션을 참조하십시오.

Athena 콘솔에서 SSO 지원 작업 그룹 생성

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. Athena 콘솔 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
3. 작업 그룹 페이지에서 작업 그룹 생성을 선택합니다.
4. 작업 그룹 생성 페이지의 작업 그룹 이름에 작업 그룹의 이름을 입력합니다.
5. 분석 엔진으로 Athena SQL 기본값을 사용합니다.
6. 인증으로 IAM Identity Center를 선택합니다.
7. IAM Identity Center 액세스를 위한 서비스 역할로 기존 서비스 역할을 선택하거나 새 서비스 역할을 생성합니다.

   Athena가 사용자를 대신하여 IAM Identity Center에 액세스할 수 있는 권한이 필요합니다. 이를 위해서는 Athena에 서비스 역할이 필요합니다. 서비스 역할은 AWS 서비스가 사용자를 대신하여 다른 AWS 서비스에 액세스할 수 있도록 권한을 부여하는 IAM 역할입니다. 자세한 내용은 IAM 사용 설명서의 AWS서비스에 대한 권한을 위임할 역할 생성을 참조하세요.
8. 쿼리 결과 구성을 확장한 다음 쿼리 결과의 위치에 Amazon S3 경로를 입력하거나 선택합니다.
9. (선택 사항) 쿼리 결과 암호화를 선택합니다.
10. (선택 사항) 사용자 ID 기반 S3 접두사 생성을 선택합니다.

    IAM Identity Center 지원 작업 그룹을 생성할 때는 기본적으로 S3 Access Grants 활성화 옵션이 선택됩니다. Amazon S3 Access Grants를 사용하여 Amazon S3의 Athena 쿼리 결과 위치(접두사)에 대한 액세스를 제어할 수 있습니다. Amazon S3 Access Grants에 대한 자세한 내용은 Managing access with Amazon S3 Access Grants를 참조하십시오.

    IAM Identity Center 인증을 사용하는 Athena 작업 그룹에서 Amazon S3 Access Grants에 의해 관리되는 ID 기반 쿼리 결과 위치 생성을 활성화할 수 있습니다. 이러한 사용자 ID 기반 Amazon S3 접두사를 사용하면 Athena 작업 그룹의 사용자는 쿼리 결과를 동일한 작업 그룹의 다른 사용자와 격리된 상태로 유지할 수 있습니다.

    사용자 접두사 옵션을 활성화하면 Athena는 사용자 ID를 작업 그룹의 쿼리 결과 출력 위치에 Amazon S3 경로 접두사로 추가합니다(예: `s3://DOC-EXAMPLE-BUCKET/${user_id}`). 이 기

능을 사용하려면 `user_id` 접두사가 있는 위치에 대한 사용자 권한만 허용하도록 Access Grants를 구성해야 합니다. Athena 쿼리 결과에 대한 액세스를 제한하는 Amazon S3 Access Grants의 위치 역할 정책 샘플은 샘플 역할 정책을 참조하세요.

> **Note**
>
> 사용자 ID S3 접두사 옵션을 선택하면 다음 단계에서 설명하는 것처럼 작업 그룹에 대한 클라이언트 측 설정 재정의 옵션이 자동으로 활성화됩니다. 클라이언트 측 설정 재정의 옵션은 사용자 ID 접두사 기능의 요구 사항입니다.

11. 설정을 확장한 다음 클라이언트 측 설정 재정의가 선택되어 있는지 확인합니다.

    클라이언트 측 설정 재정의를 선택하면 작업 그룹 설정이 작업 그룹의 모든 클라이언트에 대해 작업 그룹 수준에서 적용됩니다. 자세한 내용은 클라이언트 측 설정보다 우선하는 작업 그룹 설정 단원을 참조하십시오.

12. (선택 사항) 작업 그룹 만들기에 설명된 대로 필요한 기타 구성 설정을 지정합니다.
13. 작업 그룹 생성을 선택합니다.
14. Athena 콘솔의 작업 그룹 섹션을 사용하여 IAM Identity Center 디렉터리의 사용자 또는 그룹을 IAM Identity Center 활성화 Athena 작업 그룹에 할당할 수 있습니다.

샘플 역할 정책

다음 샘플은 Athena 쿼리 결과에 대한 액세스를 제한하는 Amazon S3 Access Grant 위치에 연결할 역할에 대한 정책을 보여줍니다.

```
{
    "Statement": [{
        "Action": ["s3:*"],
        "Condition": {
            "ArnNotEquals": {
                "s3:AccessGrantsInstanceArn": "arn:aws:s3:${region}:${account}:access-
grants/default"
            },
            "StringNotEquals": {
                "aws:ResourceAccount": "${account}"
            }
        },
        "Effect": "Deny",
        "Resource": "*",
```

```
        "Sid": "ExplicitDenyS3"
    }, {
        "Action": ["kms:*"],
        "Effect": "Deny",
        "NotResource": "arn:aws:kms:${region}:${account}:key/${keyid}",
        "Sid": "ExplictDenyKMS"
    }, {
        "Action": ["s3:ListMultipartUploadParts", "s3:GetObject"],
        "Condition": {
            "ArnEquals": {
                "s3:AccessGrantsInstanceArn": "arn:aws:s3:${region}:${account}:access-
grants/default"
            },
            "StringEquals": {
                "aws:ResourceAccount": "${account}"
            }
        },
        "Effect": "Allow",
        "Resource": "arn:aws:s3:::ATHENA-QUERY-RESULT-LOCATION/${identitystore:UserId}/
*",
        "Sid": "ObjectLevelReadPermissions"
    }, {
        "Action": ["s3:PutObject", "s3:AbortMultipartUpload"],
        "Condition": {
            "ArnEquals": {
                "s3:AccessGrantsInstanceArn": "arn:aws:s3:${region}:${account}:access-
grants/default"
            },
            "StringEquals": {
                "aws:ResourceAccount": "${account}"
            }
        },
        "Effect": "Allow",
        "Resource": "arn:aws:s3:::ATHENA-QUERY-RESULT-LOCATION/${identitystore:UserId}/
*",
        "Sid": "ObjectLevelWritePermissions"
    }, {
        "Action": "s3:ListBucket",
        "Condition": {
            "ArnEquals": {
                "s3:AccessGrantsInstanceArn": "arn:aws:s3:${region}:${account}:access-
grants/default"
            },
            "StringEquals": {
```

```
                "aws:ResourceAccount": "${account}"
            },
            "StringLikeIfExists": {
                "s3:prefix": ["${identitystore:UserId}", "${identitystore:UserId}/*"]
            }
        },
        "Effect": "Allow",
        "Resource": "arn:aws:s3:::ATHENA-QUERY-RESULT-LOCATION",
        "Sid": "BucketLevelReadPermissions"
    }, {
        "Action": ["kms:GenerateDataKey", "kms:Decrypt"],
        "Effect": "Allow",
        "Resource": "arn:aws:kms:${region}:${account}:key/${keyid}",
        "Sid": "KMSPermissions"
    }],
    "Version": "2012-10-17"
}
```

Athena 작업 그룹 API

다음은 Athena 작업 그룹에 사용되는 몇 가지 REST API 작업입니다. `ListWorkGroups`를 제외한 다음 모든 작업에서 작업 그룹을 지정해야 합니다. `StartQueryExecution`과 같은 다른 작업에서는 작업 그룹 파라미터가 선택 사항이며 여기서 작업을 설명하지 않습니다. 전체 작업 목록은 Amazon Athena API 참조를 참조하세요.

- CreateWorkGroup
- DeleteWorkGroup
- GetWorkGroup
- ListWorkGroups
- UpdateWorkGroup

작업 그룹 문제 해결

다음 팁을 사용하여 작업 그룹의 문제를 해결하세요.

- 계정의 개별 사용자에 대한 권한을 확인합니다. 사용자가 쿼리 결과의 위치 및 쿼리를 실행하려는 작업 그룹에 대해 액세스 권한이 있어야 합니다. 사용자가 작업 그룹을 전환할 경우 두 작업 그룹 모두에 대해 권한이 있어야 합니다. 자세한 설명은 작업 그룹 액세스를 위한 IAM 정책을 참조하세요.

- Athena 콘솔의 컨텍스트에 주의를 기울여서 어느 작업 그룹에서 쿼리를 실행하려 하고 있는지 확인합니다. 드라이버를 사용할 경우 필요한 드라이버에 작업 그룹을 설정해야 합니다. 자세한 설명은 the section called “쿼리를 실행할 작업 그룹 지정”을 참조하세요.
- API 또는 드라이버를 사용하여 쿼리를 실행할 경우 개별 쿼리에 대해서는 OutputLocation(클라이언트 측)을 사용하여 쿼리 결과 위치를 지정해야 합니다. 작업 그룹의 경우 WorkGroupConfiguration을 사용합니다. 두 방법 중 하나를 사용하여 위치를 지정하지 않은 경우 쿼리 런타임 중에 Athena에서 오류가 발생합니다.
- 클라이언트 측 설정을 작업 그룹 설정으로 재정의한 경우 쿼리 결과 위치에 대한 오류가 발생할 수 있습니다. 예를 들어 작업 그룹의 사용자가, 쿼리 결과를 저장하기 위한 Amazon S3의 작업 그룹 위치에 대해 권한이 없을 수 있습니다. 이 경우 필요한 권한을 추가하세요.
- 작업 그룹은 API 작업의 동작에 변경을 적용합니다. 다음과 같은 기존 API 작업을 호출하려면 계정의 사용자들이 작업을 수행하는 작업 그룹에 대한 리소스 기반 권한을 IAM에서 소유해야 합니다. 기존의 작업 그룹 및 작업 그룹 작업에 대한 권한이 없을 경우 다음 API 작업은 `AccessDeniedException`을 발생시킵니다. CreateNamedQuery, DeleteNamedQuery, GetNamedQuery, ListNamedQueries, StartQueryExecution, StopQueryExecution, ListQueryExecutions, GetQueryExecution, GetQueryResults, GetQueryResultsStream(이 API 작업은 드라이버와 함께 사용하는 경우에만 사용할 수 있으며 공개적으로 사용하는 경우에는 노출되지 않음). 자세한 내용은 서비스 권한 부여 참조에서 Amazon Athena에 사용되는 작업, 리소스 및 조건 키를 참조하세요.

  BatchGetQueryExecution 및 BatchGetNamedQuery API 작업에 대한 호출은 사용자가 액세스 권한이 있는 작업 그룹에서 실행한 쿼리에 대해서만 정보를 반환합니다. 사용자가 작업 그룹에 대해 액세스 권한이 없을 경우 이러한 API 작업은 처리되지 않은 ID 목록에 승인되지 않은 쿼리 ID를 포함시켜 반환합니다. 자세한 내용은 the section called “ Athena 작업 그룹 API” 단원을 참조하세요.
- 쿼리가 실행되는 작업 그룹이 쿼리 결과 위치 강제 적용(enforced query results location)으로 구성된 경우 CTAS 쿼리에 대해 `external_location`을 지정하면 안 됩니다. 이러한 경우 Athena에서 오류가 발생하고 `external_location`을 지정한 쿼리가 실패합니다. 예를 들어 쿼리 결과 위치에 대해 클라이언트 측 설정의 재정의할 경우 이 쿼리가 실패하고 작업 그룹에서 해당 위치 `CREATE TABLE <DB>.<TABLE1> WITH (format='Parquet', external_location='s3://DOC-EXAMPLE-BUCKET/test/') AS SELECT * FROM <DB>.<TABLE2> LIMIT 10;`를 사용합니다.

다음과 같은 오류가 나타날 수 있습니다. 다음 표에는 작업 그룹과 관련된 몇 가지 오류 목록과 권장 해결 방법이 나와 있습니다.

작업 그룹 오류

Error	발생 조건...
취소됨 쿼리 상태. 스캔한 바이트 제한을 초과했습니다.	쿼리가 쿼리당 데이터 한도에 도달하여 취소되었습니다. 데이터를 읽는 양을 줄이도록 쿼리를 다시 작성하거나 계정 관리자에게 문의하세요.
arn:aws:iam::123456789012:user/abc 사용자는 *arn:aws:athena:us-east-1:123456789012:workgroup/workgroupname* 리소스에 대해 athena:StartQueryExecution 작업을 수행하도록 승인되지 않았습니다.	사용자가 작업 그룹에서 쿼리를 실행했으나 작업 그룹에 대해 액세스 권한이 없습니다. 작업 그룹에 대해 액세스 권한을 갖도록 정책을 업데이트하세요.
INVALID_INPUT. <name> 작업 그룹이 비활성화되었습니다.	사용자가 작업 그룹에서 쿼리를 실행했으나 작업 그룹이 비활성화되었습니다. 관리자가 작업 그룹을 비활성화했을 수 있습니다. 또한 해당 작업 그룹에 대해 액세스 권한이 없을 수 있습니다. 두 경우 모두 액세스 권한을 가진 관리자에게 연락하여 작업 그룹을 수정해 달라고 요청하세요.
INVALID_INPUT. <name> 작업 그룹을 찾을 수 없습니다.	사용자가 작업 그룹에서 쿼리를 실행했으나 작업 그룹이 존재하지 않습니다. 작업 그룹이 삭제되었을 수 있습니다. 다른 작업 그룹으로 전환하여 쿼리를 실행하세요.
InvalidRequestException: StartQueryExecution 작업을 호출하는 동안 출력 위치를 제공하지 않았습니다. 출력 위치는 Workgroup 결과 구성 설정 또는 API 입력으로 필요합니다.	사용자가 쿼리 결과 위치를 지정하지 않고 API를 사용하여 쿼리를 실행했습니다. 개별 쿼리에 대해 OutputLocation(클라이언트 측)을 사용하여 또는 작업 그룹에서 WorkGroupConfiguration을 사용하여 쿼리 결과 위치를 설정해야 합니다.
CTAS(Create Table As Select) 쿼리는 모든 쿼리에 대해 중앙 집중식 출력 위치를 적용하는 'external_location' 속성과 함께 Athena 작업 그룹에 제출되었기 때문에 실패했습니다. 'external	쿼리를 실행한 작업 그룹이 쿼리 결과 위치 강제 적용으로 구성된 경우 CTAS 쿼리에 대해 `external_location` 을 지정하세요. 이 경우

Error	발생 조건...
_location' 속성을 제거하고 쿼리를 다시 제출하세요.	`external_location` 을 제거하고 쿼리를 다시 실행하세요.
`prepared_statement_name` 준비된 문을 생성할 수 없습니다. 이 작업 그룹에서 준비된 문 수가 1000개 제한을 초과했습니다.	이 작업 그룹에 1,000개 제한을 초과하는 준비된 문이 포함되어 있습니다. 이 문제를 해결하려면 DEALLOCATE PREPARE를 사용해 작업 그룹에서 준비된 문을 하나 이상 제거합니다. 또는 새 작업 그룹을 생성합니다.

CloudWatch 지표 및 이벤트를 사용하여 비용 관리 및 쿼리 모니터링

작업 그룹에서는 쿼리당 또는 작업 그룹당 데이터 사용량 제어 한도를 설정하고, 이러한 한도가 초과될 때 경보를 설정할 수 있으며, CloudWatch에 쿼리 지표를 게시할 수 있습니다.

각 작업 그룹에서 다음을 수행할 수 있습니다.

- 쿼리당 그리고 작업 그룹당 데이터 사용량 제어를 구성하고, 쿼리가 임계값을 위반할 경우 수행할 작업을 설정합니다.
- 쿼리 지표를 보고 분석하며 CloudWatch에 게시합니다. 콘솔에서 작업그룹을 만들 경우 CloudWatch에 지표를 게시하는 설정이 자동으로 선택됩니다. API 작업을 사용할 경우 지표 게시를 활성화해야 합니다. 지표가 게시되면 작업 그룹 패널의 지표 탭 아래에 표시됩니다. 지표는 기본 작업 그룹에 대해 기본적으로 비활성화됩니다.

비디오

다음 동영상은 CloudWatch에서 사용자 지정 대시보드를 생성하고 지표에 대한 경보 및 트리거를 설정하는 방법을 보여 줍니다. Athena 콘솔에서 사전 입력된 대시보드를 직접 사용해 이러한 쿼리 지표를 이용할 수 있습니다.

Amazon CloudWatch를 사용하여 Amazon Athena 쿼리 모니터링

주제

- CloudWatch 쿼리 지표 활성화
- CloudWatch 지표를 사용한 Athena 쿼리 모니터링
- Amazon EventBridge 이벤트를 사용하여 Athena 이벤트 모니터링

- Athena 사용량 지표 모니터링
- 데이터 사용량 제어 한도 설정

CloudWatch 쿼리 지표 활성화

콘솔에서 작업그룹을 만들 경우 CloudWatch에 쿼리 지표를 게시하는 설정이 기본적으로 선택됩니다.

Athena 콘솔에서 작업 그룹에 대해 쿼리 지표를 활성화 또는 비활성화하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

3. 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
4. 수정할 작업 그룹의 링크를 선택합니다.
5. 작업 그룹에 대한 세부 정보 페이지에서 편집(Edit)을 선택합니다.
6. 설정(Settings) 섹션에서 AWS CloudWatch에 쿼리 지표 게시(Publish query metrics to CloudWatch)를 선택하거나 선택 취소합니다.

API 작업, 명령줄 인터페이스 또는 JDBC 드라이버가 있는 클라이언트 애플리케이션을 사용하여 작업 그룹을 생성하고 쿼리 지표 게시를 활성화하는 경우 WorkGroupConfiguration에서 `PublishCloudWatchMetricsEnabled`를 `true`로 설정합니다. 다음 예제는 지표 구성만 보여 주고 다른 구성은 생략합니다.

```
"WorkGroupConfiguration": {
      "PublishCloudWatchMetricsEnabled": "true"
```

```
    ....
    }
```

CloudWatch 지표를 사용한 Athena 쿼리 모니터링

CloudWatch에 쿼리 지표 게시 옵션을 선택하면 Athena는 쿼리 관련 지표를 Amazon CloudWatch에 게시합니다. 사용자 지정 대시보드를 만들거나, CloudWatch의 지표에 대해 경보 및 트리거를 설정하거나, Athena 콘솔에서 사전 입력된 대시보드를 직접 사용할 수 있습니다.

작업 그룹의 쿼리에 대해 쿼리 지표를 활성화하면 Athena 콘솔에서 각 작업 그룹에 대한 작업 그룹 패널의 지표 탭에 지표가 표시됩니다.

Athena는 다음의 지표를 CloudWatch 콘솔에 게시합니다.

- `DPUAllocated` - 쿼리를 실행하기 위해 용량 예약에 프로비저닝된 총 데이터 처리 단위(DPU) 수입니다.
- `DPUConsumed` - 예약에서 특정 시간에 `RUNNING` 상태의 쿼리가 소비하는 DPU 수입니다. 작업 그룹이 용량 예약과 연결되어 있고 예약과 연결된 모든 작업 그룹을 포함하는 경우에만 생성되는 지표입니다.
- `DPUCount` - 쿼리가 소비하는 최대 DPU 수로, 쿼리가 완료될 때 정확히 한 번 게시됩니다.
- `EngineExecutionTime` - 쿼리를 실행하는 데 걸린 시간(밀리초)입니다.
- `ProcessedBytes` - DML 쿼리당 Athena가 스캔한 바이트 수입니다.
- `QueryPlanningTime` - Athena가 쿼리 처리 흐름을 계획하는 데 걸린 시간(밀리초)입니다.
- `QueryQueueTime` - 쿼리가 리소스를 기다리면서 쿼리 대기열에 있던 시간(밀리초)입니다.
- `ServicePreProcessingTime` - Athena가 쿼리 엔진에 쿼리를 제출하기 전에 쿼리를 사전 처리하는 데 걸린 시간(밀리초)입니다.
- `ServiceProcessingTime` - 쿼리 엔진이 쿼리 실행을 완료한 후 Athena가 쿼리 결과를 처리하는 데 걸린 시간(밀리초)입니다.
- `TotalExecutionTime` - Athena가 DDL 또는 DML 쿼리를 실행하는 데 걸린 시간(밀리초)입니다.

자세한 설명은 이 문서 뒷부분의 Athena의 CloudWatch 지표 및 차원 목록 섹션을 참조하세요.

이러한 지표에는 다음과 같은 차원이 있습니다.

- `CapacityReservation` - 해당하는 경우 쿼리를 실행하는 데 사용된 용량 예약의 이름입니다.
- `QueryState` - `SUCCEEDED`, `FAILED` 또는 `CANCELED`

- `QueryType` - `DML`, `DDL` 또는 `UTILITY`
- `WorkGroup` - 작업 그룹의 이름입니다.

Athena는 다음의 지표를 `AmazonAthenaForApacheSpark` 네임스페이스 아래 CloudWatch 콘솔에 게시합니다.

- `DPUCount` - 세션 중에 계산을 실행하는 데 사용된 DPU 수입니다.

이 지표에는 다음과 같은 차원이 있습니다.

- `SessionId` - 계산이 제출되는 세션의 ID입니다.
- `WorkGroup` - 작업 그룹의 이름입니다.

자세한 내용은 이 주제의 후반부에서 Athena의 CloudWatch 지표 및 차원 목록 단원을 참조하세요. Athena 사용량 지표에 대한 자세한 내용은 Athena 사용량 지표 모니터링 섹션을 참조하세요.

콘솔에서 작업 그룹에 대한 쿼리 지표를 보려면

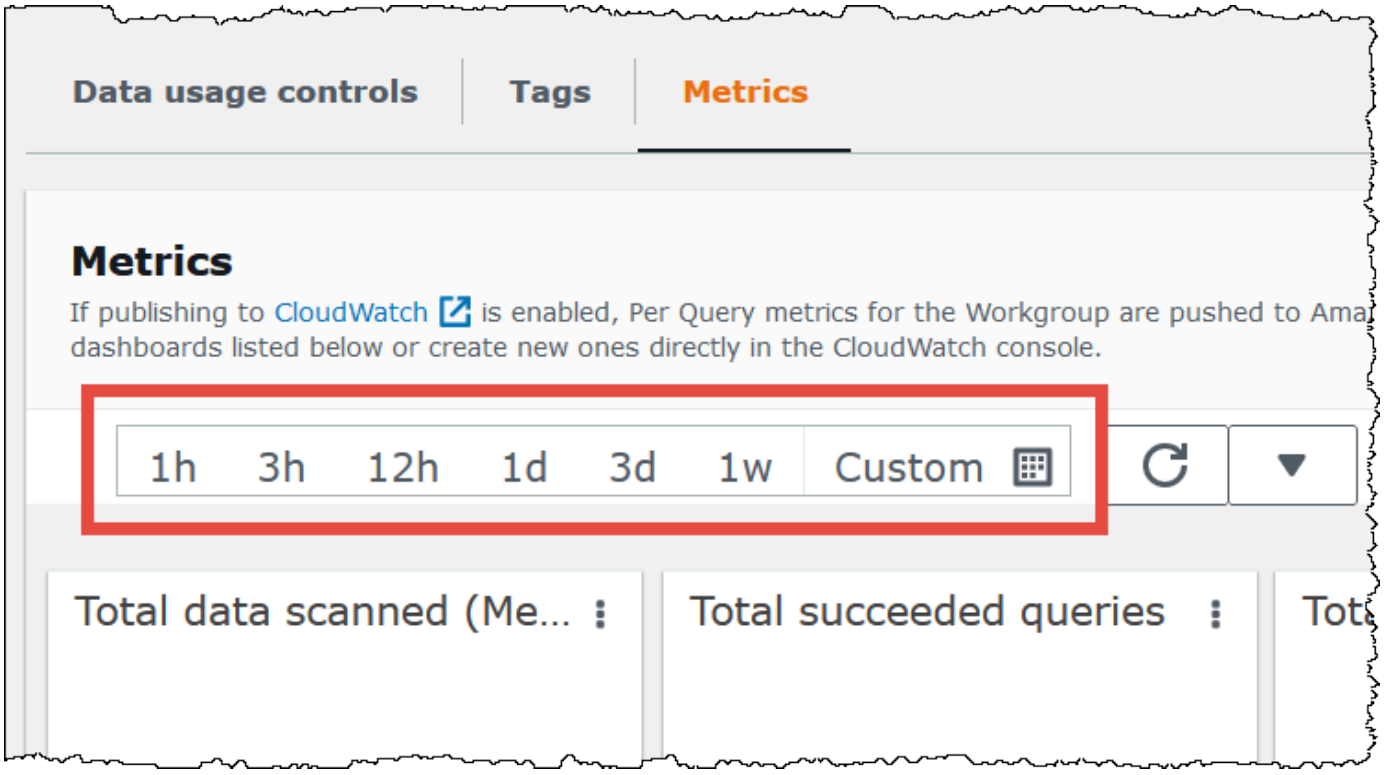
1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

3. 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
4. 목록에서 원하는 작업 그룹을 선택한 다음 지표(Metrics) 탭을 선택합니다.

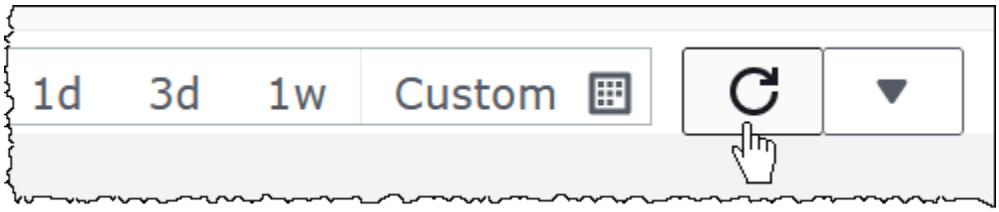
   지표 대시보드가 표시됩니다.

> **Note**
>
> 작업 그룹에 대한 지표를 최근에 활성화했거나 최근 쿼리 활동이 없었다면 대시보드의 그래프가 비어 있을 수 있습니다. 다음 단계에서 지정하는 간격마다 CloudWatch에서 쿼리 활동을 검색합니다.

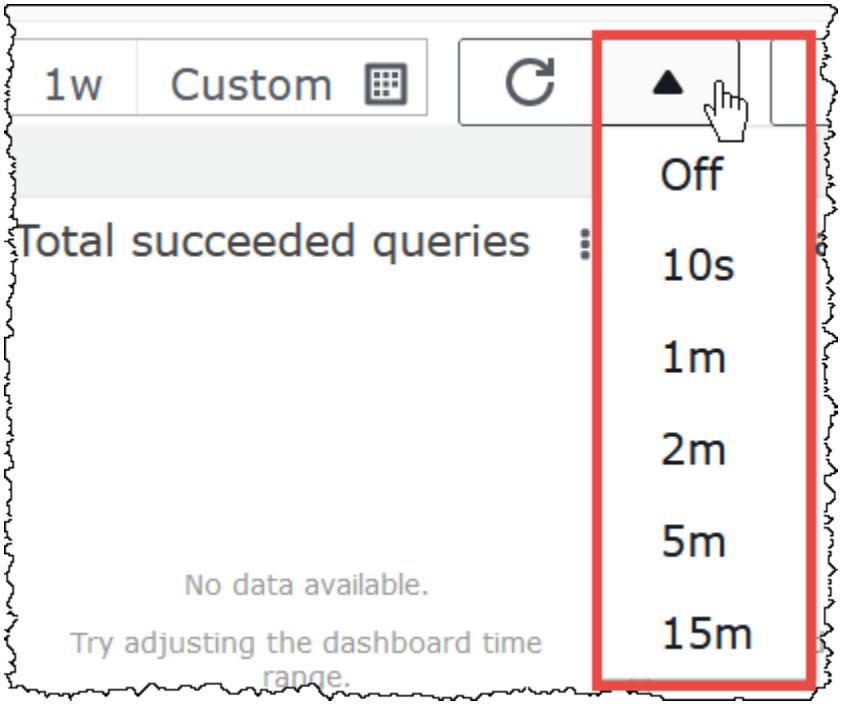
5. 지표(Metrics) 섹션에서 Athena가 CloudWatch에서 쿼리 활동을 가져오는 데 사용할 지표 간격을 선택하거나, 사용자 지정 간격을 지정합니다.



6. 표시된 지표를 새로 고치려면 새로 고침 아이콘을 선택합니다.



7. 새로 고침 아이콘 옆에 있는 화살표를 클릭하여 지표 표시를 업데이트할 빈도를 선택합니다.



Amazon CloudWatch 콘솔에서 지표를 보려면

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창에서 지표, 모든 지표를 선택합니다.
3. AWS/Athena 네임스페이스를 선택합니다.

CLI를 사용하여 지표를 보려면

- 다음 중 하나를 수행하십시오.
  - Athena에 대한 지표를 나열하려면 명령 프롬프트를 열고 다음 명령을 사용합니다.

```
aws cloudwatch list-metrics --namespace "AWS/Athena"
```

  - 사용 가능한 모든 지표의 목록을 보려면 다음 명령을 사용합니다.

```
aws cloudwatch list-metrics"
```

Athena의 CloudWatch 지표 및 차원 목록

Athena에서 CloudWatch 지표를 활성화했다면 다음 지표가 작업 그룹별로 CloudWatch로 전송됩니다. 다음 지표에서 `AWS/Athena` 네임스페이스를 사용합니다.

지표 이름	설명
DPUAllocated	쿼리를 실행하기 위해 용량 예약에 프로비저닝된 총 데이터 처리 단위(DPU) 수입니다.
DPUConsumed	예약에서 특정 시간에 `RUNNING` 상태의 쿼리가 소비하는 DPU 수입니다. 이 지표는 작업 그룹이 용량 예약과 연결되어 있고 예약과 연결된 모든 작업 그룹을 포함하는 경우에만 생성됩니다. 작업 그룹을 한 예약에서 다른 예약으로 이동하면 해당 작업 그룹이 첫 번째 예약에 속했던 시점의 데이터가 지표에 포함됩니다. 용량 예약에 대한 자세한 내용은 쿼리 처리 용량 관리 섹션을 참조하세요.
DPUCount	쿼리가 소비하는 최대 DPU 수로, 쿼리가 완료될 때 정확히 한 번 게시됩니다. 이 지표는 용량 예약에 연결된 작업 그룹에 대해서만 생성됩니다.
EngineExecutionTime	쿼리를 실행하는 데 걸린 시간(밀리초)입니다.
ProcessedBytes	DML 쿼리당 Athena가 스캔한 바이트 수입니다. 취소된 쿼리의 경우(한도에 다다른 경우, 쿼리가 사용자에 의해 또는 자동으로 취소됨), 취소 시간 이전에 스캔한 데이터의 양을 포함합니다. DDL 쿼리에 대해서는 이 지표가 보고되지 않습니다.
QueryPlanningTime	Athena가 쿼리 처리 흐름을 계획하는 데 걸린 시간(밀리초)입니다. 여기에는 데이터 소스로부터 테이블 파티션을 검색하는 데 소요된 시간이 포함됩니다. 쿼리 엔진은 쿼리 계획을 수행하기 때문에 쿼리 계획 시간은 EngineExecutionTime의 하위 집합입니다.
QueryQueueTime	쿼리가 리소스를 기다리면서 쿼리 대기열에 있던 시간(밀리초) 입니다. 일시적인 오류가 발생하면 쿼리가 자동으로 대기열에 다시 추가될 수 있습니다.
ServicePreProcessingTime	Athena가 쿼리 엔진에 쿼리를 제출하기 전에 쿼리를 사전 처리하는 데 걸린 시간(밀리초)입니다.

지표 이름	설명
ServiceProcessingTime	쿼리 엔진이 쿼리 실행을 완료한 후 Athena가 쿼리 결과를 처리하는 데 걸린 시간(밀리초)입니다.
TotalExecutionTime	Athena가 DDL 또는 DML 쿼리를 실행하는 데 걸린 시간(밀리초)입니다. TotalExecutionTime에는 QueryQueueTime, QueryPlanningTime, EngineExecutionTime 및 ServiceProcessingTime이 포함됩니다..

Athena의 이 지표에는 다음과 같은 차원이 있습니다.

측정기준	설명
CapacityReservation	해당하는 경우 쿼리를 실행하는 데 사용된 용량 예약의 이름입니다. 용량 예약이 사용되지 않으면 이 차원은 데이터를 반환하지 않습니다.
QueryState	쿼리 상태입니다. 유효한 통계: SUCCEEDED, FAILED 또는 CANCELED.
QueryType	쿼리 유형입니다. 유효한 통계: `DDL`, `DML` 또는 `UTILITY`. 실행된 쿼리 문의 유형입니다. `DDL`은 데이터 정의 언어(DDL) 쿼리 문을 나타냅니다. `DML`은 데이터 조작 언어(DML) 쿼리 문을 나타냅니다(예: `CREATE TABLE AS SELECT`). `UTILITY`는 DDL 및 DML 이외의 쿼리 문(예:`SHOW CREATE TABLE` 또는 `DESCRIBE TABLE`)을 나타냅니다.
WorkGroup	작업 그룹의 이름입니다.

Amazon EventBridge 이벤트를 사용하여 Athena 이벤트 모니터링

Amazon EventBridge와 함께 Amazon Athena를 사용하여 쿼리 상태에 대한 실시간 알림을 받을 수 있습니다. 사용자가 제출한 쿼리의 상태가 전환되면 Athena는 해당 쿼리 상태 전환에 대한 정보를 포함하는 이벤트를 EventBridge에 게시합니다. 원하는 이벤트에 대한 간단한 규칙을 작성하고 규칙과 일치

하는 이벤트 발생 시 자동으로 작업을 수행하도록 할 수 있습니다. 예를 들어 쿼리가 종료 상태에 도달할 때 AWS Lambda 함수를 호출하는 규칙을 만들 수 있습니다. 이벤트는 최선의 작업을 기반으로 발생됩니다.

Athena에 대한 이벤트 규칙을 생성하기 전에 다음을 수행해야 합니다.

- Eventbridge의 이벤트, 규칙, 대상을 숙지해야 합니다. 자세한 내용은 Amazon EventBridge란 무엇입니까?를 참조하십시오. 규칙을 설정하는 방법에 대한 자세한 내용은 Amazon EventBridge 시작하기를 참조하세요.
- 이벤트 규칙에 사용할 대상을 만듭니다.

> **Note**
>
> Athena는 현재 한 가지 유형의 이벤트인 Athena 쿼리 상태 변경을 제공하지만 다른 이벤트 유형 및 세부 정보를 추가할 수도 있습니다. 프로그래밍 방식으로 이벤트 JSON 데이터를 역직렬화하는 경우, 추가 속성이 추가되었을 때 알 수 없는 속성을 처리할 수 있도록 애플리케이션이 준비되어야 합니다.

Athena 이벤트 형식

다음은 Amazon Athena 이벤트의 기본 패턴입니다.

```
{
    "source":[
        "aws.athena"
    ],
    "detail-type":[
        "Athena Query State Change"
    ],
    "detail":{
        "currentState":[
            "SUCCEEDED"
        ]
    }
}
```

Athena 쿼리 상태 변경 이벤트

다음 예제에서는 `currentState` 값이 `SUCCEEDED`인 Athena 쿼리 상태 변경 이벤트를 보여 줍니다.

```
{
    "version":"0",
    "id":"abcdef00-1234-5678-9abc-def012345678",
    "detail-type":"Athena Query State Change",
    "source":"aws.athena",
    "account":"123456789012",
    "time":"2019-10-06T09:30:10Z",
    "region":"us-east-1",
    "resources":[

    ],
    "detail":{
        "versionId":"0",
        "currentState":"SUCCEEDED",
        "previousState":"RUNNING",
        "statementType":"DDL",
        "queryExecutionId":"01234567-0123-0123-0123-012345678901",
        "workgroupName":"primary",
        "sequenceNumber":"3"
    }
}
```

다음 예제에서는 currentState 값이 FAILED인 Athena 쿼리 상태 변경 이벤트를 보여 줍니다. athenaError 블록은 currentState가 FAILED인 경우에만 나타납니다. errorCategory 및 errorType에 대한 값 관련 내용은 Athena 오류 카탈로그 단원을 참조하세요.

```
{
    "version":"0",
    "id":"abcdef00-1234-5678-9abc-def012345678",
    "detail-type":"Athena Query State Change",
    "source":"aws.athena",
    "account":"123456789012",
    "time":"2019-10-06T09:30:10Z",
    "region":"us-east-1",
    "resources":[
    ],
    "detail":{
        "athenaError": {
            "errorCategory": 2.0, //Value depends on nature of exception
            "errorType": 1306.0, //Type depends on nature of exception
            "errorMessage": "Amazon S3 bucket not found", //Message depends on nature
 of exception
```

```
            "retryable":false //Retryable value depends on nature of exception
        },
        "versionId":"0",
        "currentState": "FAILED",
        "previousState": "RUNNING",
        "statementType":"DML",
        "queryExecutionId":"01234567-0123-0123-0123-012345678901",
        "workgroupName":"primary",
        "sequenceNumber":"3"
    }
}
```

출력 속성

JSON 출력에는 다음 속성이 포함됩니다.

속성	설명
athenaError	currentState 가 FAILED인 경우에만 나타납니다. 오류 범주, 오류 유형, 오류 메시지, 오류를 일으킨 작업을 재시도할 수 있는지 여부 등 발생한 오류에 대한 정보가 포함되어 있습니다. 이러한 각 필드의 값은 오류의 특성에 따라 달라집니다. errorCategory 및 errorType 에 대한 값 관련 내용은 Athena 오류 카탈로그 단원을 참조하세요.
versionId	세부 정보 객체 스키마의 버전 번호입니다.
currentState	이벤트 발생 시 쿼리가 다른 상태로 전환됩니다.
previousState	이벤트 발생 시 쿼리가 다른 상태에서 전환됩니다.
statementType	실행된 쿼리 문의 유형입니다.
queryExecutionId	실행된 쿼리에 대한 고유 식별자입니다.
workgroupName	쿼리가 실행된 작업 그룹의 이름입니다.
sequenceNumber	실행된 단일 쿼리와 관련된 수신 이벤트의 중복 제거 및 순서 지정을 수행할 수 있는 단순 증가 수치입니다. 동일한 상태 전환에 대해 중복 이벤트가 게시되면 sequenceNumber 값은 동일합니다. 드물지만 다시 대기열에 올리는 쿼리와 같이 쿼리에 상태 전환이 두 번 이상 발생하는 경우

속성	설명
	sequenceNumber 를 사용하여 동일한 currentState 및 previousState 값을 가진 이벤트의 순서를 지정할 수 있습니다.

예

다음 예제에서는 구독한 Amazon SNS 주제에 이벤트를 게시합니다. Athena가 쿼리되면 이메일을 받게 됩니다. 이 예제에서는 Amazon SNS 주제가 존재하고 사용자가 이를 구독했다고 가정합니다.

Amazon SNS 주제에 Athena 이벤트를 게시하려면

1. Amazon SNS 주제의 대상을 만듭니다. 다음 예제와 같이 Amazon SNS 주제에 게시할 수 있는 events.amazonaws.com 권한을 EventBridge 이벤트 서비스 보안 주체에 부여합니다.

```
{
    "Effect":"Allow",
    "Principal":{
        "Service":"events.amazonaws.com"
    },
    "Action":"sns:Publish",
    "Resource":"arn:aws:sns:us-east-1:111111111111:your-sns-topic"
}
```

2. 다음 예제와 같이 AWS CLI events put-rule 명령을 사용하여 Athena 이벤트에 대한 규칙을 만듭니다.

```
aws events put-rule --name {ruleName} --event-pattern '{"source": ["aws.athena"]}'
```

3. 다음 예제와 같이 AWS CLI events put-targets 명령을 사용하여 Amazon SNS 주제 대상을 규칙에 연결합니다.

```
aws events put-targets --rule {ruleName} --targets Id=1,Arn=arn:aws:sns:us-
east-1:111111111111:your-sns-topic
```

4. Athena를 쿼리하고 호출되는 대상을 관찰합니다. Amazon SNS 주제에서 해당 이메일을 수신해야 합니다.

Amazon Athena에서 AWS 사용자 알림 사용

AWS 사용자 알림을 사용하여 Amazon Athena 이벤트에 대한 알림을 받을 전송 채널을 설정할 수 있습니다. 이벤트가 지정한 규칙과 일치하면 알림을 받습니다. 이메일, AWS Chatbot 채팅 알림 또는 AWS Console Mobile Application 푸시 알림을 비롯한 여러 채널을 통해 이벤트에 대한 알림을 받을 수 있습니다. 콘솔 알림 센터에서도 알림을 볼 수 있습니다. 사용자 알림은 집계를 지원하므로 특정 이벤트 중에 받는 알림 수를 줄일 수 있습니다.

자세한 내용은 AWS 사용자 알림 사용 설명서를 참조하십시오.

Athena 사용량 지표 모니터링

CloudWatch 사용량 지표를 사용하여 CloudWatch 그래프 및 대시보드에서 현재 서비스 사용량을 표시하여 계정에서 리소스가 어떻게 사용되고 있는지 확인할 수 있습니다.

Athena의 경우 사용량 가용성 지표는 Athena의 AWS 서비스 할당량에 해당합니다. 사용량이 서비스 할당량에 가까워지면 경고하는 경보를 구성할 수 있습니다. Athena 서비스 할당량에 대한 자세한 내용은 Service Quotas 섹션을 참조하세요. AWS 사용량 지표에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 AWS 사용량 지표를 참조하세요.

Athena는 `AWS/Usage` 네임스페이스에 다음 지표를 게시합니다.

지표 이름	설명
`ResourceCount`	쿼리 유형(DML 또는 DDL)으로 구분된 계정당 AWS 리전에 따라 대기 중인 모든 쿼리와 실행 쿼리의 합계입니다. Maximum은 이 지표에 가장 유용한 통계입니다. 이 지표는 1분마다 주기적으로 게시됩니다. 쿼리를 실행하지 않는 경우 이 지표는 아무 것도 보고하지 않습니다(0도 보고하지 않음). 이 지표는 지표가 계산될 때 활성 쿼리가 실행 중인 경우에만 게시됩니다.

다음 차원은 Athena에 의해 게시되는 사용량 지표를 구체화하는 데 사용됩니다.

측정기준	설명
`Service`	리소스가 포함된 AWS 서비스의 이름 Athena의 경우 이 차원의 값은 `Athena`입니다.

측정기준	설명
`Resource`	실행 중인 리소스의 유형입니다. Athena 쿼리 사용량에 대한 리소스 값은 `ActiveQueryCount` 입니다.
`Type`	보고되는 엔터티의 유형입니다. 현재 Athena 사용량 지표에 대한 유일한 유효 값은 `Resource`입니다.
`Class`	추적 중인 리소스의 클래스입니다. Athena의 경우, `Class`는 `DML` 또는 `DDL`이 될 수 있습니다.

CloudWatch 콘솔에서 Athena 리소스 사용량 지표 보기

CloudWatch 콘솔을 사용하여 Athena 사용량 지표의 그래프를 표시하고 사용량이 서비스 할당량에 가까워지면 경고하는 경보를 구성할 수 있습니다.

Athena 리소스 사용량 지표를 보려면

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창에서 지표, 모든 지표를 선택합니다.
3. 사용량(Usage)을 선택한 다음 AWS 리소스별(By Resource)을 선택합니다.

   서비스 할당량 사용량 지표 목록이 나타납니다.
4. Athena 및 ActiveQueryCount 옆에 있는 확인란을 선택합니다.
5. 그래프로 표시된 지표 탭을 선택합니다.

   위의 그래프는 AWS 리소스의 현재 사용량을 표시합니다.

그래프에 서비스 할당량을 추가하고 서비스 할당량에 가까워지면 알려주는 경보를 설정하는 데 대한 자세한 내용은 Amazon CloudWatch 사용 설명서에서 서비스 할당량 시각화 및 경보 설정을 참조하세요. 작업 그룹당 사용량 한도 설정에 대한 자세한 내용은 데이터 사용량 제어 한도 설정 섹션을 참조하세요.

데이터 사용량 제어 한도 설정

Athena에서는 쿼리당 한도와 작업 그룹당 한도의 두 가지 비용 관리를 설정할 수 있습니다. 각 작업 그룹에 대해 쿼리당 한도 한 개와 작업 그룹당 한도 여러 개를 설정할 수 있습니다.

- 쿼리당 제어 한도는 쿼리당 스캔된 데이터의 총량을 지정합니다. 작업 그룹에서 실행하는 쿼리가 이 한도를 초과할 경우 쿼리가 취소됩니다. 한 작업 그룹에서 쿼리당 제어 한도를 한 개만 생성할 수 있으며 이 한도는 해당 작업 그룹에서 실행되는 각 쿼리에 적용됩니다. 이 한도를 변경해야 할 경우 수정할 수 있습니다. 자세한 방법은 쿼리당 데이터 사용량 제어를 생성하려면을 참조하세요.
- 작업 그룹 전역 데이터 사용량 제어 한도는 이 작업 그룹에서 실행하는 모든 쿼리에 대해 지정된 시간 동안 스캔된 데이터의 총량을 지정합니다. 작업 그룹당 한도를 여러 개 만들 수 있습니다. 작업 그룹 전역 쿼리 한도를 사용하여 해당 작업 그룹에서 실행 중인 쿼리별로 스캔된 데이터의 시간별 또는 일간 집계에 대해 임계값을 여러 개 설정할 수 있습니다.

  스캔된 데이터의 집계량이 임계값을 초과할 경우 Amazon SNS 주제에 알림을 푸시할 수 있습니다. 이렇게 하려면 한도를 위반할 때 관리자에게 알리도록 Athena 콘솔에서 Amazon SNS 경보 및 작업을 구성합니다. 자세한 방법은 작업 그룹당 데이터 사용량 제어를 생성하려면을 참조하세요. CloudWatch 콘솔에서 Athena가 게시하는 모든 지표에 대해 경보 및 조치를 생성할 수도 있습니다. 예를 들어 실패한 쿼리의 수에 대한 경고를 설정할 수 있습니다. 이 경고는 쿼리 수가 특정 임계값에 이를 경우 관리자에게 이메일을 보내게 할 수 있습니다. 한도를 초과할 경우 조치는 지정된 사용자에게 Amazon SNS 경보 알림을 보냅니다.

  수행할 수 있는 기타 조치:

  - Lambda 함수를 간접적으로 호출합니다. 자세한 내용은 Amazon Simple Notification Service 개발자 안내서에서 Amazon SNS 알림을 사용하여 Lambda 함수 호출 단원을 참조하세요.
  - 작업 그룹을 비활성화하여 쿼리가 더 이상 실행되지 않도록 합니다. 단계는 작업 그룹 활성화 및 비활성화를 참조하세요.

쿼리당 한도 및 작업 그룹당 한도는 서로 독립적입니다. 지정된 조치는 한도가 초과될 때마다 수행됩니다. 둘 이상의 사용자가 동일한 작업 그룹에서 동시에 쿼리를 실행할 경우, 각 쿼리는 지정된 한도를 초과하지 않지만, 스캔한 데이터의 총합은 작업 그룹당 데이터 사용 한도를 초과할 수 있습니다. 이 경우 Amazon SNS 경보가 사용자에게 전송됩니다.

쿼리당 데이터 사용량 제어를 생성하려면

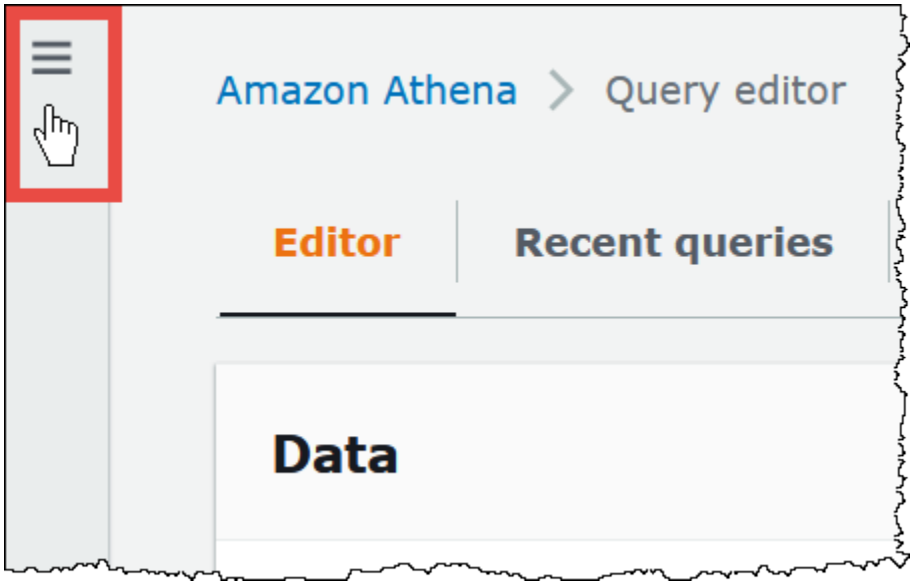
쿼리당 제어 한도는 쿼리당 스캔된 데이터의 총량을 지정합니다. 작업 그룹에서 실행하는 쿼리가 이 한도를 초과할 경우 쿼리가 취소됩니다. 취소된 쿼리는 Amazon Athena 요금에 따라 요금이 청구됩니다.

> **Note**
>
> 취소되거나 실패한 쿼리의 경우 Athena가 Amazon S3에 일부 결과를 이미 썼을 수 있습니다. 이 경우 Athena는 결과가 저장된 Amazon S3 접두사에서 일부 결과를 삭제하지 않습니다. 일부 결과를 포함한 Amazon S3 접두사는 제거해야 합니다. Athena는 Amazon S3 멀티파트

업로드를 사용하여 Amazon S3에 데이터를 씁니다. 쿼리가 실패할 경우 멀티파트 업로드를 종료하도록 버킷 수명 주기 정책을 설정할 것을 권장합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버킷 수명 주기 정책을 사용한 미완료 멀티파트 업로드 중단을 참조하세요.

한 작업 그룹에서 쿼리당 제어 한도를 한 개만 생성할 수 있으며 이 한도는 해당 작업 그룹에서 실행되는 각 쿼리에 적용됩니다. 이 한도를 변경해야 할 경우 수정할 수 있습니다.

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.



3. 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
4. 목록에서 작업 그룹을 선택합니다.
5. 데이터 사용량 컨트롤(Data usage controls) 탭의 쿼리별 데이터 사용량 컨트롤(Per query data usage control) 섹션에서 관리(Manage)를 선택합니다.
6. 쿼리당 데이터 사용량 컨트롤 관리(Manage per query data usage control) 페이지에서 다음 값을 지정합니다.

   - 데이터 제한(Data limit)에 10MB(최소) ~ 7EB(최대) 범위의 값을 지정합니다.

> **Note**
>
> 이 한도는 콘솔에서 작업 그룹 내의 데이터 사용량 제어에 대해 부여하는 한도입니다. 이 한도가 Athena의 쿼리 한도를 나타내는 것은 아닙니다.

- 드롭다운 목록에서 단위 값을 선택합니다(예: 킬로바이트 KB(Kilobytes KB) 또는 엑사바이트 EB(Exabytes EB)).

기본 작업은 쿼리가 제한을 초과할 경우 쿼리를 취소하는 것입니다. 이 설정은 변경할 수 없습니다.

7. Save(저장)를 선택합니다.

작업 그룹당 데이터 사용량 알림 생성

작업 그룹에서 실행되는 쿼리가 특정 기간 내에 지정된 양의 데이터를 스캔할 때 여러 경고 임계값을 설정할 수 있습니다. 알림은 Amazon CloudWatch 경보를 사용하여 구현되며 작업 그룹의 모든 쿼리에 적용됩니다. 임계값에 도달하면 Amazon SNS에서 지정한 사용자에게 이메일을 보내도록 할 수 있습니다. 임계값에 도달하면 쿼리가 자동으로 취소되지 않습니다.

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.
3. 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
4. 목록에서 작업 그룹을 선택합니다.
5. 편집(Edit)을 선택하여 작업 그룹의 설정을 편집합니다.
6. 아래로 스크롤하여 작업 그룹 데이터 사용량 알림 - 선택 사항을 확장합니다.
7. 알림 추가(Add a alert)를 선택합니다.
8. 데이터 사용량 임계값 구성(Data usage threshold configuration)에 대해 다음과 같이 값을 지정합니다.

   - 데이터 임계값(Data threshold)에 숫자를 지정한 다음 드롭다운 목록에서 단위 값을 선택합니다.
   - 기간(Time period) 드롭다운 목록에서 시간을 선택합니다.
   - SNS 주제 선택(SNS topic selection) 드롭다운 목록에서 Amazon SNS 주제를 선택합니다. 또는 SNS 주제 생성(Create SNS topic)을 선택하고 Amazon SNS 콘솔로 바로 이동해 Amazon SNS 주제를 생성한 후 Athena 계정의 사용자 중 하나에 대해 이 주제의 구독을 설정합니다. 자세한

내용은 Amazon Simple Notification Service 개발자 안내서의 Amazon SNS 시작하기를 참조하십시오.

9. 새 알림을 생성하는 경우 알림 추가(Add alert)를 선택하고 기존 알림을 편집한 경우 저장(Save)을 선택합니다.

쿼리 처리 용량 관리

용량 예약을 사용하여 Athena에서 실행하는 쿼리에 대한 전용 처리 용량을 확보할 수 있습니다. 용량 예약을 통해 가장 중요한 대화형 워크로드의 우선순위를 지정하고 해당 워크로드를 제어 및 확장할 수 있는 워크로드 관리 기능을 이용할 수 있습니다. 예를 들어 언제든지 용량을 추가하여 동시에 실행할 수 있는 쿼리 수를 늘리고 용량을 사용할 수 있는 워크로드를 제어하며 워크로드 사이에서 용량을 공유할 수 있습니다. 용량은 Athena에서 완전하게 관리하며 필요한 시간 동안 보관됩니다. 설정이 쉽고 SQL 문을 변경할 필요도 없습니다.

쿼리의 처리 용량을 확보하려면 용량 예약을 생성하고 필요한 데이터 처리 장치(DPU) 수를 지정하며 예약에 하나 이상의 작업 그룹을 할당합니다.

용량 예약을 사용할 때 작업 그룹은 중요한 역할을 합니다. 작업 그룹을 사용하면 쿼리를 논리적으로 그룹화할 수 있습니다. 용량 예약을 사용하면 작업 그룹에 용량을 선택적으로 할당하여 각 작업 그룹에 대한 쿼리의 작동 방식과 청구 방식을 제어할 수 있습니다. 작업 그룹에 대한 자세한 내용은 작업 그룹을 사용하여 쿼리 액세스 및 비용 제어 섹션을 참조하세요.

예약에 작업 그룹을 할당하면 할당된 작업 그룹에 제출하는 쿼리에 우선순위를 부여할 수 있습니다. 예를 들어 시간이 중요한 금융 보고 쿼리에 사용되는 작업 그룹에 용량을 할당하여 해당 쿼리를 다른 작업 그룹의 덜 중요한 쿼리와 분리할 수 있습니다. 이렇게 하면 다른 워크로드를 독립적으로 실행하는 동시에 중요한 워크로드에서 일관된 쿼리 실행이 가능합니다.

용량 예약과 작업 그룹을 함께 사용하여 다양한 요구 사항을 충족할 수 있습니다. 다음은 몇 가지 예제 시나리오입니다.

- 격리 - 중요한 워크로드를 격리하기 위해 하나의 예약에 단일 작업 그룹을 할당합니다. 지정된 작업 그룹의 쿼리만 선택한 예약의 처리 용량을 사용합니다.
- 공유 - 여러 워크로드에서 한 예약의 용량을 공유할 수 있습니다. 예를 들어 특정 워크로드 세트의 월별 비용을 예측하려는 경우 단일 예약에 여러 작업 그룹을 할당할 수 있습니다. 할당된 작업 그룹에서 예약 용량을 공유합니다.
- 혼합 모델 - 동일한 계정에서 동시에 용량 예약 및 쿼리별 청구를 사용할 수 있습니다. 예를 들어 프로덕션 애플리케이션을 지원하는 쿼리를 안정적으로 실행하려면 해당 쿼리에 대한 작업 그룹을 용

량 예약에 할당합니다. 프로덕션 작업 그룹으로 이동하기 전에 쿼리를 개발할 때 예약과 연결되지 않아 쿼리별 청구를 사용하는 별도의 작업 그룹을 사용합니다.

DPU 이해

용량은 데이터 처리 장치(DPU)로 측정됩니다. DPU는 Athena에서 사용자를 대신해 데이터에 액세스하고 데이터를 처리하는 데 사용하는 컴퓨팅 및 메모리 리소스를 나타냅니다. 1개의 DPU는 4개의 vCPU와 16GB의 메모리를 제공합니다. 지정하는 DPU 수는 동시에 실행할 수 있는 쿼리 수에 영향을 줍니다. 예를 들어 256개의 DPU를 포함하는 예약은 128개의 DPU를 포함하는 예약보다 약 두 배 더 많은 동시 쿼리 수를 지원할 수 있습니다.

계정 및 리전당 최대 1,000개의 DPU를 포함하는 최대 100개의 용량 예약을 생성할 수 있습니다. 요청할 수 있는 최소 DPU 수는 24개입니다. 사용 사례에 1,000개가 넘는 DPU가 필요한 경우 athena-feedback@amazon.com으로 문의하세요.

용량 요구 사항 추정에 대한 자세한 내용은 용량 요구 사항 결정 섹션을 참조하세요. 요금 정보는 Amazon Athena 요금을 참조하세요.

고려 사항 및 제한

- 이 기능을 사용하려면 Athena 엔진 버전 3이 필요합니다.
- 한 번에 최대 하나의 예약에 단일 작업 그룹을 할당할 수 있으며, 예약에 최대 20개의 작업 그룹을 추가할 수 있습니다.
- Spark 지원 작업 그룹은 용량 예약에 추가할 수 없습니다.
- 예약에 할당된 작업 그룹을 삭제하려면 먼저 예약에서 작업 그룹을 제거합니다.
- 프로비저닝할 수 있는 최소 DPU 수는 24개입니다.
- 계정 및 리전당 최대 1,000개의 DPU를 포함하는 최대 100개의 용량 예약을 생성할 수 있습니다.
- 용량에 대한 요청은 보장되지 않으며, 완료하는 데 최대 30분이 소요될 수 있습니다.
- 최소 청구 기간은 예약당 1시간입니다. 1시간 후에는 분당 용량이 청구됩니다. 요금 정보는 Amazon Athena 요금을 참조하세요.
- 예약 용량은 다른 용량 예약(AWS 계정 또는 AWS 리전)으로 이전할 수 없습니다.
- 용량 예약에서 DDL 쿼리는 DPU를 소비합니다.
- 프로비저닝된 용량에서 실행되는 쿼리는 DDL 및 DML에 대한 활성 쿼리 한도에 포함되지 않습니다.
- 모든 DPU가 사용 중인 경우 제출된 쿼리는 대기열에 보관됩니다. 이러한 쿼리는 거부되지 않지만, 온디맨드 용량으로 전환되지 않습니다.

- DPUConsumed CloudWatch 지표는 예약 단위가 아닌 작업 그룹 단위입니다. 따라서 작업 그룹을 한 예약에서 다른 예약으로 이동하면 해당 작업 그룹이 첫 번째 예약에 속했던 시점의 데이터가 DPUConsumed 지표에 포함됩니다. Athena에서 CloudWatch 지표를 사용하는 방법에 대한 자세한 내용은 CloudWatch 지표를 사용한 Athena 쿼리 모니터링 섹션을 참조하세요.
- 현재 이 기능은 다음 AWS 리전에서 사용할 수 있습니다.
  - 미국 동부(오하이오)
  - 미국 동부(버지니아 북부)
  - 미국 서부(오레곤)
  - 아시아 태평양(싱가포르)
  - 아시아 태평양(시드니)
  - 아시아 태평양(도쿄)
  - 유럽(아일랜드)
  - 유럽(스톡홀름)

주제

용량 요구 사항 결정

용량 예약을 생성하기 전에 올바른 DPU 수를 할당할 수 있도록 필요한 용량을 추정할 수 있습니다. 또한 예약을 사용한 후에 예약에서 용량의 부족 또는 초과 상태를 확인하는 것이 좋습니다. 이 주제에서는 이러한 추정을 위해 사용할 수 있는 기법과 함께 사용량 및 비용을 평가하는 몇 가지 AWS 도구도 설명합니다.

주제

필요한 용량 추정

용량 요구 사항을 추정할 때 특정 쿼리에 필요한 용량과 일반적으로 필요한 용량이라는 두 가지 측면을 고려하는 것이 좋습니다.

쿼리당 용량 요구 사항 추정

쿼리에 필요할 수 있는 DPU 수를 결정하려는 경우 다음 지침을 사용할 수 있습니다.

- DDL 쿼리는 4개의 DPU를 소비합니다.
- DML 쿼리는 4~124개의 DPU를 소비합니다.

Athena는 쿼리가 제출될 때 DML 쿼리에 필요한 DPU 수를 결정합니다. 이 수는 데이터 크기, 스토리지 형식, 쿼리 구성 및 기타 요인에 따라 달라집니다. 일반적으로 Athena는 가장 적으면서 가장 효율적인 DPU 수를 선택하려고 합니다. Athena에서 쿼리를 성공적으로 완료하는 데 더 많은 컴퓨팅 성능이 필요하다고 판단하면 쿼리에 할당된 DPU 수가 늘어납니다.

워크로드별 용량 요구 사항 추정

여러 쿼리를 동시에 실행하는 데 필요한 용량을 결정하려면 다음 테이블의 일반 지침을 고려합니다.

동시 쿼리 수	필요한 DPU 수
10	40개 이상
20	96개 이상
30개 이상	240개 이상

필요한 실제 DPU 수는 목표와 분석 패턴에 따라 달라집니다. 예를 들어 쿼리를 대기열에 넣지 않고 즉시 시작하려는 경우 최대 동시 쿼리 수요를 결정한 후 적절히 DPU 수를 프로비저닝합니다.

최대 수요보다 적게 DPU를 프로비저닝할 수 있지만, 최대 수요가 발생하면 대기열이 나타날 수 있습니다. 대기열이 나타나면 Athena는 쿼리를 대기열에 넣고 사용 가능한 용량이 확보되면 쿼리를 실행합니다.

고정된 예산 안에서 쿼리를 실행하는 것이 목표인 경우 AWS 요금 계산기를 사용하여 예산에 맞는 DPU 수를 결정할 수 있습니다.

마지막으로, 데이터 크기, 스토리지 형식, 쿼리 작성 방식이 쿼리에 필요한 DPU 수에 영향을 준다는 점을 기억하세요. 쿼리 성능을 높이려면 데이터를 압축 또는 파티셔닝하거나 열 형식으로 변환할 수 있습니다. 자세한 내용은 Athena의 성능 튜닝 단원을 참조하십시오.

더 많은 용량이 필요하다는 신호

할당한 용량이 부족함을 알리는 두 가지 징후로, 용량 부족 오류 메시지가 나타나거나 대기열에 쿼리가 들어갑니다.

용량 부족 오류 메시지와 함께 쿼리에 실패하는 경우 용량 예약의 DPU 수가 쿼리에 비해 너무 적습니다. 예를 들어 24개의 DPU로 예약한 상태에서 24개가 넘는 DPU가 필요한 쿼리를 실행하면 쿼리에 실패합니다. Athena의 EventBridge 이벤트를 사용하여 이 쿼리 오류를 모니터링할 수 있습니다. DPU를 더 추가하고 쿼리를 다시 실행하세요.

대기열에 있는 쿼리가 많으면 다른 쿼리에서 용량을 완전히 사용하고 있다는 의미입니다. 대기열을 줄이려면 다음 중 하나를 수행합니다.

- 예약에 DPU를 추가하여 쿼리 동시 처리 기능을 개선합니다.
- 예약에서 작업 그룹을 제거하여 다른 쿼리에 사용할 용량을 확보합니다.

대기열에 있는 쿼리가 너무 많은지 확인하려면 용량 예약에서 작업 그룹에 대한 Athena 쿼리 대기열 시간 CloudWatch 지표를 사용합니다. 값이 원하는 임계값을 초과하는 경우 용량 예약에 DPU를 추가할 수 있습니다.

유휴 용량 확인

유휴 용량을 확인하려면 예약의 DPU 수를 줄이거나 워크로드를 늘린 후에 결과를 관찰할 수 있습니다.

유휴 용량을 확인하려면

1. 다음 중 하나를 수행하십시오.
   - 예약의 DPU 수 감소(사용 가능한 리소스 감소)
   - 예약에 작업 그룹 추가(워크로드 증가)
2. CloudWatch를 사용하여 쿼리 대기열 시간을 측정합니다.
3. 대기열 시간이 원하는 수준을 초과하면 다음 중 하나를 수행합니다.
   - 작업 그룹 제거
   - 용량 예약에 DPU 추가

4. 변경할 때마다 성능과 쿼리 대기열 시간을 확인합니다.
5. 워크로드 및/또는 DPU 수를 계속 조정하여 원하는 균형을 맞춥니다.

원하는 기간이 아닐 때 용량을 유지하지 않으려면 예약을 취소하고 나중에 다른 예약을 생성할 수 있습니다. 그러나 최근에 다른 예약에서 용량을 취소했어도 새 요량에 대한 요청은 보장되지 않으며, 새 예약을 생성하는 데 시간이 걸립니다.

용량 요구 사항 및 비용을 평가하기 위한 도구

AWS에서 다음 서비스 및 기능을 사용하여 Athena 사용량 및 비용을 측정할 수 있습니다.

CloudWatch 지표

쿼리 관련 지표를 작업 그룹 수준에서 Amazon CloudWatch에 게시하도록 Athena를 구성할 수 있습니다. 작업 그룹에 대한 지표를 활성화하면 작업 그룹 쿼리에 대한 지표가 Athena 콘솔의 작업 그룹 세부 정보 페이지에 표시됩니다.

CloudWatch에 게시된 Athena 지표와 해당 수치에 대한 자세한 내용은 CloudWatch 지표를 사용한 Athena 쿼리 모니터링 섹션을 참조하세요.

CloudWatch 사용량 지표

CloudWatch 사용량 지표를 사용하여 CloudWatch 그래프 및 대시보드에서 현재 서비스 사용량을 표시하여 계정에서 리소스가 어떻게 사용되고 있는지 확인할 수 있습니다. Athena의 경우 사용량 가용성 지표는 Athena에 대한 AWS 서비스 할당량에 해당합니다. 사용량이 서비스 할당량에 가까워지면 경고하는 경보를 구성할 수 있습니다.

자세한 내용은 Athena 사용량 지표 모니터링 단원을 참조하십시오.

Amazon EventBridge 이벤트

Amazon EventBridge와 함께 Amazon Athena를 사용하여 쿼리 상태에 대한 실시간 알림을 받을 수 있습니다. 사용자가 제출한 쿼리의 상태가 변경되면 Athena는 해당 쿼리 상태 전환에 대한 정보를 포함하는 이벤트를 EventBridge에 게시합니다. 원하는 이벤트에 대한 간단한 규칙을 작성하고 규칙과 일치하는 이벤트 발생 시 자동으로 작업을 수행하도록 할 수 있습니다.

자세한 정보는 다음 리소스를 참조하세요.

- Amazon EventBridge 이벤트를 사용하여 Athena 이벤트 모니터링
- Amazon EventBridge란 무엇인가요?
- Amazon EventBridge 이벤트

Tags

Athena에서 용량 예약은 태그를 지원합니다. 태그는 키와 값으로 구성됩니다. Athena에서 비용을 추적하기 위해 AWS에서 생성된 비용 할당 태그를 사용할 수 있습니다. AWS에서는 비용 할당 태그를 사용하여 Cost and Usage Report에서 리소스 비용을 구성합니다. 이 방법을 통해 AWS 비용을 더 쉽게 분류하고 추적할 수 있습니다. Athena에 대한 비용 할당 태그를 활성화하려면 AWS Billing and Cost Management 콘솔을 사용합니다.

자세한 정보는 다음 리소스를 참조하세요.

- Athena 리소스 태깅
- AWS에서 생성하는 비용 할당 태그 활성화
- AWS 비용 할당 태그 사용

용량 예약 생성

먼저 필요한 수의 DPU가 포함된 용량 예약을 생성하고 해당 용량을 쿼리에 사용할 하나 이상의 작업 그룹을 할당합니다. 나중에 필요에 따라 용량을 조정하여 보다 일관된 성능을 제공하거나 비용을 효과적으로 관리할 수 있습니다. 용량 요구 사항 추정에 대한 자세한 내용은 용량 요구 사항 결정 섹션을 참조하세요.

> **Important**
>
> 용량에 대한 요청은 보장되지 않으며, 완료하는 데 최대 30분이 소요될 수 있습니다.

용량 예약을 생성하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.
3. 관리, 용량 예약을 선택합니다.
4. 용량 예약 생성을 선택합니다.
5. 용량 예약 생성 페이지에서 용량 예약 이름에 이름을 입력합니다. 이름은 1~128자로 고유해야 하며 a~z, A~Z, 0~9, _(밑줄), .(마침표), -(하이픈)만 사용해야 합니다. 예약을 생성한 후에는 이름을 변경할 수 없습니다.
6. DPU의 경우 원하는 데이터 처리 장치(DPU)의 수를 선택하거나 입력합니다(증분 단위: 4). 자세한 내용은 DPU 이해 단원을 참조하십시오.

7. (선택 사항) 태그 옵션을 확장하고 새 태그 추가를 선택하여 용량 예약 리소스에 연결할 사용자 지정 키 및 값 페어를 하나 이상 추가합니다. 자세한 내용은 Athena 리소스 태깅 단원을 참조하십시오.
8. 검토를 선택합니다.
9. 용량 예약 생성 확인 프롬프트에서 DPU 수, AWS 리전 및 기타 정보를 확인합니다. 수락하는 경우 제출을 선택합니다.

   세부 정보 페이지에서 용량 예약 상태는 대기 중으로 표시됩니다. 예약 용량에서 쿼리를 실행할 수 있으면 상태가 활성으로 표시됩니다.

이제 예약에 하나 이상의 작업 그룹을 추가할 수 있습니다. 단계는 예약에 작업 그룹 추가를 참조하세요.

예약 관리

용량 예약 페이지에서 용량 예약을 보고 관리할 수 있습니다. DPU 추가 또는 감소, 작업 그룹 할당 수정, 예약 태그 지정 또는 예약 취소와 같은 관리 작업을 수행할 수 있습니다.

용량 예약을 보고 관리하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.
3. 관리, 용량 예약을 선택합니다.
4. 용량 예약 페이지에서 다음 작업을 수행할 수 있습니다.

   - 용량 예약을 생성하려면 용량 예약 생성을 선택합니다.
   - 검색 상자를 사용하여 DPU의 이름 또는 수를 기준으로 예약을 필터링합니다.
   - 상태 드롭다운 메뉴를 선택하여 용량 예약 상태(예: 활성 또는 취소됨)에 따라 필터링합니다. 예약 상태에 대한 자세한 내용은 예약 상태 파악 섹션을 참조하세요.
   - 용량 예약에 대한 세부 정보를 보려면 예약에 대한 링크를 선택합니다. 예약 세부 정보 페이지에는 용량 편집, 작업 그룹 추가, 작업 그룹 제거, 예약 취소 옵션이 포함되어 있습니다.
   - 예약을 편집하려면(예: DPU 추가 또는 제거) 예약에 대한 버튼을 선택하고 편집을 선택합니다.
   - 예약을 취소하려면 예약에 대한 버튼을 선택하고 취소를 선택합니다.

예약 상태 파악

다음 테이블에서는 용량 예약의 가능한 상태 값을 설명합니다.

상태 표시기	설명
보류중	Athena에서 용량 요청을 처리 중입니다. 쿼리를 실행할 용량이 준비되지 않았습니다.
활성	쿼리를 실행할 용량이 사용 가능합니다.
실패	용량에 대한 요청이 완료되지 않았습니다. 용량 요청 이행은 보장되지 않습니다. 실패한 예약 수는 계정의 DPU 한도 계산에 포함됩니다. 사용량을 확보하려면 예약을 취소해야 합니다.
업데이트 대기 중	Athena에서 예약 변경을 처리 중입니다. 예를 들어 예약을 편집하여 DPU를 추가하거나 제거한 후에 이 상태가 나타납니다.
취소 중	Athena에서 예약 취소 요청을 처리 중입니다. 예약을 사용하던 작업 그룹에서 여전히 실행 중인 쿼리는 완료할 수 있지만 작업 그룹의 다른 쿼리는 온디맨드 용량(프로비저닝되지 않음)을 사용합니다.
취소됨	용량 예약 취소가 완료됩니다. 취소된 예약은 콘솔에 45일 동안 남아 있습니다. 45일이 지나면 Athena에서 예약을 삭제합니다. 45일 동안 예약의 용도를 변경하거나 재사용할 수 없지만, 기록 참조를 위해 해당 태그를 참조하고 세부 정보를 볼 수 있습니다. 취소된 용량은 향후 재예약이 보장되지 않습니다. 또는 다른 예약(AWS 계정 또는 AWS 리전)으로 이전될 수 없습니다.

활성 DPU 및 대상 DPU 이해

Athena 콘솔의 용량 예약 목록에서 예약에는 활성 DPU 및 대상 DPU라는 두 개의 DPU 값이 표시됩니다.

- 활성 DPU - 예약에서 쿼리를 실행하는 데 사용할 수 있는 DPU 수입니다. 예를 들어 100개의 DPU를 요청하고 해당 요청이 이행되면 활성 DPU는 100으로 표시됩니다.
- 대상 DPU - 예약에서 이동 처리 중인 DPU의 수입니다. 예약이 생성되거나 DPU 수의 증가 또는 감소가 대기 중인 경우 대상 DPU에는 활성 DPU와 다른 값이 표시됩니다.

예를 들어 DPU가 24개인 예약을 생성하는 요청을 제출하면 예약 상태는 대기 중, 활성 DPU는 0, 대상 DPU는 24로 표시됩니다.

DPU가 100개인 예약에서 예약을 편집하여 DPU를 20개로 늘리도록 요청하는 경우 상태는 업데이트 대기 중, 활성 DPU는 100, 대상 DPU는 120으로 표시됩니다.

DPU가 100개인 예약에서 예약을 편집하여 DPU를 20개로 줄이도록 요청하는 경우 상태는 업데이트 대기 중, 활성 DPU는 100, 대상 DPU는 80으로 표시됩니다.

이러한 전환 중에 Athena는 사용자 요청에 따라 DPU 수를 확보하거나 줄이기 위해 적극적으로 노력합니다. 활성 DPU가 대상 DPU와 같아지면 대상 수에 도달한 것이며 대기 중인 변경 사항은 없습니다.

이러한 값을 프로그래밍 방식으로 검색하기 위해 GetCapacityReserve API 작업을 호출할 수 있습니다. API는 활성 DPU와 대상 DPU를 `AllocatedDpus` 및 `TargetDpus`로 참조합니다.

주제

용량 예약 편집

용량 예약을 생성한 후 DPU 수를 조정하고 사용자 지정 태그를 추가하거나 제거할 수 있습니다.

용량 예약을 편집하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.
3. 관리, 용량 예약을 선택합니다.
4. 용량 예약 목록에서 다음 중 하나를 수행합니다.
   - 예약 옆에 있는 버튼을 선택하고 편집을 선택합니다.
   - 예약 링크를 선택하고 편집을 선택합니다.
5. DPU 경우 원하는 데이터 처리 장치의 수를 선택하거나 입력합니다(증분 단위: 4). 보유할 수 있는 최소 DPU 수는 24개입니다. 자세한 내용은 DPU 이해 단원을 참조하십시오.

> **Note**
>
> 언제든지 기존 용량 예약에 DPU를 추가할 수 있습니다. 하지만 DPU 수를 줄이려면 예약을 생성하거나 예약에 DPU를 추가한 후 1시간이 지나야 합니다.

6. (선택 사항) 태그에서 제거를 선택하여 태그를 제거하거나 새 태그 추가를 선택하여 새 태그를 추가합니다.
7. 제출을 선택합니다. 예약의 세부 정보 페이지에 업데이트된 구성이 표시됩니다.

예약에 작업 그룹 추가

용량 예약을 생성한 후 예약에 최대 20개의 작업 그룹을 추가할 수 있습니다. 예약에 작업 그룹을 추가하면 예약 용량으로 실행해야 하는 쿼리를 Athena에 알립니다. 예약에 연결되지 않은 작업 그룹의 쿼리는 스캔한 테라바이트(TB)당 기본 요금 모델을 사용하여 계속 실행됩니다.

예약에 작업 그룹이 두 개 이상 있는 경우 해당 작업 그룹의 쿼리는 예약 용량을 사용할 수 있습니다. 언제든지 작업 그룹을 추가하고 제거할 수 있습니다. 작업 그룹을 추가하거나 제거할 때 실행 중인 쿼리는 중단되지 않습니다.

예약이 대기 중 상태인 경우 추가한 작업 그룹의 쿼리는 예약이 활성화될 때까지 스캔한 테라바이트(TB)당 기본 요금 모델을 사용하여 계속 실행됩니다.

용량 예약에 하나 이상의 작업 그룹을 추가하려면

1. 용량 예약의 세부 정보 페이지에서 작업 그룹 추가를 선택합니다.
2. 작업 그룹 추가 페이지에서 추가할 작업 그룹을 선택하고 작업 그룹 추가를 선택합니다. 둘 이상의 예약에 작업 그룹을 할당할 수 없습니다.

   용량 예약의 세부 정보 페이지에는 추가한 작업 그룹이 나열됩니다. 해당 작업 그룹에서 실행되는 쿼리는 예약이 활성화될 때 예약한 용량을 사용합니다.

예약에서 작업 그룹 제거

더 이상 작업 그룹 전용 용량이 필요하지 않거나 작업 그룹을 고유한 예약으로 이동하려는 경우 언제든지 작업 그룹을 제거할 수 있습니다. 예약에서 작업 그룹을 제거하는 작업은 간단합니다. 예약에서 작업 그룹을 제거한 후 제거된 작업 그룹의 쿼리는 기본적으로 온디맨드 용량(프로비저닝되지 않음)을 사용하며 스캔한 테라바이트(TB)를 기준으로 요금이 청구됩니다.

예약에서 하나 이상의 작업 그룹을 제거하려면

1. 용량 예약의 세부 정보 페이지에서 제거하려는 작업 그룹을 선택합니다.
2. 작업 그룹 제거를 선택합니다. 작업 그룹을 제거하시겠습니까? 프롬프트에서는 작업 그룹이 예약에서 제거되기 전에 현재 활성 상태인 모든 쿼리가 완료된다는 메시지를 표시합니다.
3. 제거를 선택합니다. 용량 예약의 세부 정보 페이지에 제거된 작업 그룹이 더 이상 없다고 표시됩니다.

용량 예약 취소

더 이상 용량 예약을 사용하지 않으려는 경우 취소할 수 있습니다. 예약을 사용하던 작업 그룹에서 여전히 실행 중인 쿼리는 완료할 수 있지만 작업 그룹의 다른 쿼리는 더 이상 예약을 사용하지 않습니다.

> **Note**
>
> 취소된 용량은 향후 재예약이 보장되지 않습니다. 또는 다른 예약(AWS 계정 또는 AWS 리전)으로 이전될 수 없습니다.

용량 예약을 취소하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.
3. 관리, 용량 예약을 선택합니다.
4. 용량 예약 목록에서 다음 중 하나를 수행합니다.
   - 예약 옆에 있는 버튼을 선택하고 취소를 선택합니다.
   - 예약 링크를 선택하고 용량 예약 취소를 선택합니다.
5. 용량 예약을 취소하시겠습니까? 프롬프트에 취소를 입력하고 용량 예약 취소를 선택합니다.

   예약 상태가 취소 중으로 변경되고 진행 상황 배너에서 취소가 진행 중임을 알립니다.

   취소가 완료되면 용량 예약은 남아 있지만 상태는 취소됨으로 표시됩니다. 취소 후 45일이 지나면 예약이 삭제됩니다. 45일 동안 취소된 예약의 용도를 변경하거나 재사용할 수 없지만, 기록 참조를 위해 해당 태그를 참조하고 볼 수 있습니다.

용량 예약 삭제

취소된 용량 예약에 대한 모든 참조를 삭제하려면 예약을 삭제하면 됩니다. 예약을 삭제하려면 먼저 취소해야 합니다. 삭제된 예약은 계정에서 즉시 제거되고 ARN을 포함하여 더 이상 참조할 수 없습니다.

용량 예약을 삭제하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.
3. 관리, 용량 예약을 선택합니다.
4. 용량 예약 목록에서 다음 중 하나를 수행합니다.
   - 취소된 예약 옆에 있는 버튼을 선택하고 작업, 삭제를 선택합니다.
   - 예약 링크를 선택하고 삭제를 선택합니다.
5. 용량 예약을 삭제하시겠습니까? 프롬프트에서 삭제를 선택합니다.

   용량 예약이 삭제되었음을 알리는 배너가 나타납니다. 삭제된 예약은 더 이상 용량 예약 목록에 표시되지 않습니다.

용량 예약에 대한 IAM 정책

용량 예약에 대한 액세스를 제어하려면 리소스 수준 IAM 권한이나 자격 증명 기반 IAM 정책을 사용합니다. IAM 정책을 사용할 때마다 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

다음은 Athena용 절차입니다.

IAM용 정보는 이 단원 끝부분에 나와 있는 링크를 참조하세요. JSON 용량 예약 정책 예제에 대한 자세한 내용은 용량 예약 정책 예제 섹션을 참조하세요.

용량 예약 정책을 생성하기 위해 IAM 콘솔에서 시각적 편집기를 사용하려면

1. AWS Management Console에 로그인하고 https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 정책을 선택한 후 정책 생성을 선택합니다.
3. 시각적 편집기(Visual editor) 탭에서 서비스 선택(Choose a service)을 선택합니다. 그런 다음 Athena를 선택하여 정책에 추가합니다.

4. 작업 선택(Select actions)을 선택한 후 정책에 추가할 작업을 선택합니다. 시각적 편집기에 Athena에서 사용할 수 있는 작업이 표시됩니다. 자세한 내용은 서비스 권한 부여 참조에서 Amazon Athena에 사용되는 작업, 리소스 및 조건 키를 참조하세요.
5. 작업 추가를 선택하여 특정 작업을 입력하거나 와일드카드 문자(*)를 사용하여 여러 개의 작업을 지정합니다.

   기본적으로 생성되는 정책은 사용자가 선택하는 작업을 허용합니다. Athena의 `capacity-reservation` 리소스에 대해 리소스 수준 권한을 지원하는 작업을 하나 이상 선택하면 편집기에 `capacity-reservation` 리소스가 나열됩니다.
6. 리소스를 선택하여 정책에 대한 특정 용량 예약 리소스를 지정합니다. JSON 용량 예약 정책의 예제는 용량 예약 정책 예제 섹션을 참조하세요.
7. 다음과 같이 `capacity-reservation` 리소스를 지정합니다.

   ```
   arn:aws:athena:<region>:<user-account>:capacity-reservation/<capacity-reservation-name>
   ```
8. 정책 검토(Review policy)를 선택한 후 생성하려는 정책에 대한 이름(Name)과 설명(Description)(선택 사항)을 입력합니다. 정책 요약을 검토하여 의도한 권한을 부여했는지 확인합니다.
9. 정책 생성(Create policy)을 선택하고 새로운 정책을 저장합니다.
10. ID 기반 정책을 사용자, 그룹 또는 역할에 연결합니다.

자세한 내용은 서비스 권한 부여 참조와 IAM 사용 설명서에서 다음 주제를 참조하세요.

- Amazon Athena에 사용되는 작업, 리소스 및 조건 키
- 시각적 편집기를 사용하여 정책 생성
- IAM 정책 추가 및 제거
- 리소스에 대한 액세스 제어

JSON 용량 예약 정책의 예제는 용량 예약 정책 예제 섹션을 참조하세요.

Amazon Athena 작업의 전체 목록은 Amazon Athena API 참조의 API 작업 이름을 참조하세요.

용량 예약 정책 예제

이 섹션에서는 용량 예약에서 다양한 작업을 활성화하는 데 사용할 수 있는 정책 예제를 설명합니다. IAM 정책을 사용할 때마다 IAM 모범 사례를 따라야 합니다. 자세한 내용은 IAM 사용 설명서의 IAM 보안 모범 사례를 참조하세요.

용량 예약은 Athena가 관리하는 IAM 리소스입니다. 따라서 용량 예약 정책에서 `capacity-reservation`을 입력으로 사용하는 작업을 사용할 경우 다음과 같이 용량 예약의 ARN을 지정해야 합니다.

```
"Resource": [arn:aws:athena:<region>:<user-account>:capacity-reservation/<capacity-reservation-name>]
```

여기에서 *<capacity-reservation-name>*은 용량 예약 이름입니다. 예를 들어 다음과 같이 `test_capacity_reservation`이라는 용량 예약에서 이를 리소스로 지정합니다.

```
"Resource": ["arn:aws:athena:us-east-1:123456789012:capacity-reservation/
test_capacity_reservation"]
```

Amazon Athena 작업의 전체 목록은 Amazon Athena API 참조의 API 작업 이름을 참조하세요. IAM 정책에 대한 자세한 내용은 IAM 사용 설명서의 시각적 편집기로 정책 생성을 참조하세요.

- Example policy to list capacity reservations
- Example policy for management operations

Example 용량 예약을 나열하는 정책 예제

다음 정책에서는 모든 사용자가 모든 용량 예약을 나열하도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListCapacityReservations"
            ],
            "Resource": "*"
        }
    ]
}
```

Example 관리 작업을 위한 정책 예제

다음 정책에서는 사용자가 test_capacity_reservation 용량 예약을 생성 및 취소하고 세부 정보를 확인하며 업데이트하도록 허용합니다. 또한 이 정책에서는 사용자가 workgroupA 및 workgroupB를 test_capacity_reservation에 할당하도록 허용합니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect": "Allow",
         "Action": [
             "athena:CreateCapacityReservation",
             "athena:GetCapacityReservation",
             "athena:CancelCapacityReservation",
             "athena:UpdateCapacityReservation",
             "athena:GetCapacityAssignmentConfiguration",
             "athena:PutCapacityAssignmentConfiguration"
         ],
         "Resource": [
             "arn:aws:athena:us-east-1:123456789012:capacity-
reservation/test_capacity_reservation",
             "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA",
             "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupB"
         ]
      }
   ]
}
```

Athena 용량 예약 API

다음 목록에는 Athena 용량 예약 API 작업에 대한 참조 링크가 포함되어 있습니다. 데이터 구조 및 기타 Athena API 작업에 대해서는 Amazon Athena API 참조를 참조하세요.

- CancelCapacityReservation
- CreateCapacityReservation
- GetCapacityAssignmentConfiguration
- GetCapacityReservation
- ListCapacityReservations
- PutCapacityAssignmentConfiguration

- UpdateCapacityReservation

Athena의 성능 튜닝

이 주제에서는 Athena 쿼리의 성능을 개선하기 위한 일반적인 정보와 구체적인 제안, 그리고 한도 및 리소스 사용량과 관련된 오류를 해결하는 방법을 제공합니다.

Service quotas

Athena는 쿼리 실행 시간, 계정의 동시 쿼리 수, API 요청 속도와 같은 지표에 대한 할당량을 적용합니다. 이러한 할당량에 대한 자세한 내용은 Service Quotas 섹션을 참조하세요. 할당량을 초과하면 쿼리를 제출할 때나 쿼리 실행 중에 쿼리에 실패합니다.

이 페이지에 나온 많은 성능 최적화 팁은 쿼리 실행 시간을 줄이는 데 도움이 될 수 있습니다. 최적화는 용량을 확보하여 동시 할당량 내에서 더 많은 쿼리를 실행하고 쿼리의 실행 시간이 길어질 때 쿼리가 취소되지 않도록 방지할 수 있습니다.

동시 쿼리 수 및 API 요청 수에 대한 할당량은 AWS 계정 및 AWS 리전당 계산됩니다. 워크로드가 동일한 할당량을 두고 경쟁하지 않도록 하려면 AWS 계정당 워크로드를 실행하거나 별도의 프로비저닝된 용량 예약을 사용하는 것이 좋습니다.

동일한 계정에서 두 개의 워크로드를 실행하는 경우 워크로드 중 하나에서 쿼리가 급증할 수 있습니다. 이 경우 나머지 워크로드가 제한되거나 쿼리 실행이 차단될 수 있습니다. 이를 방지하려면 워크로드를 별도의 계정으로 이동하여 각 워크로드에 고유한 동시 할당량을 제공할 수 있습니다. 워크로드 중 하나 또는 둘 모두에 대해 프로비저닝된 용량 예약을 생성해도 됩니다.

다른 서비스의 할당량

Athena에서 쿼리를 실행할 때 할당량을 적용하는 다른 서비스를 호출할 수 있습니다. 쿼리 실행 중에 Athena는 AWS Glue Data Catalog, Amazon S3, 기타 AWS 서비스(예: IAM 및 AWS KMS)에 대한 API 호출을 수행할 수 있습니다. 페더레이션된 쿼리를 사용하는 경우 Athena는 AWS Lambda도 직접적으로 호출합니다. 이러한 모든 서비스에는 초과할 수 있는 자체 한도 및 할당량이 있습니다. 쿼리 실행 시 이러한 서비스에서 오류가 발생하면 쿼리에 실패하고 소스 서비스에서도 오류가 발생합니다. 복구 가능한 오류는 재시도되지만 문제가 제때 해결되지 않으면 쿼리에 실패할 수 있습니다. 오류 메시지를 자세히 읽고 Athena에서 보낸 것인지, 다른 서비스에서 보낸 것인지 확인합니다. 이 문서에서는 관련 오류 중 일부를 다룹니다.

Amazon S3 서비스 할당량으로 인한 오류 해결에 대한 자세한 내용은 이 문서에서 나중에 나오는 너무 많은 수의 파일 방지 섹션을 참조하세요. Amazon S3 성능 최적화에 대한 자세한 내용은 Amazon S3 사용 설명서의 모범 사례 설계 패턴: Amazon S3 성능 최적화를 참조하세요.

리소스 제한

Athena는 분산 쿼리 엔진에서 쿼리를 실행합니다. 쿼리를 제출하면 Athena 엔진 쿼리 플래너에서 쿼리를 실행하는 데 필요한 컴퓨팅 용량을 추정하고 적절히 컴퓨팅 노드 클러스터를 준비합니다. DDL 쿼리와 같은 일부 쿼리는 하나의 노드에서만 실행됩니다. 대규모 데이터 세트에 대한 복잡한 쿼리는 훨씬 더 큰 클러스터에서 실행됩니다. 노드는 동일한 메모리, CPU 및 디스크 구성을 사용하여 균일합니다. Athena는 보다 까다로운 쿼리를 처리하기 위해 스케일 업이 아닌, 스케일 아웃으로 확장합니다.

쿼리 요구량이 쿼리를 실행하는 클러스터에서 사용할 수 있는 리소스를 초과하는 경우가 있습니다. 이 경우 쿼리에 실패하고 Query exhausted resources at this scale factor 오류가 발생합니다.

일반적으로 가장 많이 소진되는 리소스는 메모리이지만, 디스크 공간이 소진되는 경우도 드물게 있습니다. 메모리 오류는 엔진에서 조인 또는 창 함수를 수행할 때 흔히 발생하지만 개별 개수와 집계에서 발생할 수도 있습니다.

일단 'out of resource' 오류로 쿼리에 실패했더라도 다시 실행했을 때 성공할 수도 있습니다. 쿼리 실행은 결정적이지 않습니다. 데이터를 로드하는 데 걸리는 시간, 중간 데이터 세트가 노드에 분산되는 방식 등의 요인으로 인해 리소스 사용량이 달라질 수 있습니다. 예를 들어 두 테이블을 조인하고 조인 조건의 값 분포에서 편차가 큰 쿼리를 가정합니다. 이러한 쿼리는 대부분의 경우 성공할 수 있지만, 가장 일반적인 값이 동일한 노드에서 처리되면 실패합니다.

쿼리가 사용 가능한 리소스를 초과하지 않도록 하려면 이 문서에 언급된 성능 튜닝 팁을 사용합니다. 특히 사용 가능한 리소스를 소진하는 쿼리를 최적화하는 방법에 대한 팁은 조인 최적화, 창 함수 최적화 및 근사치를 사용하여 쿼리 최적화 섹션을 참조하세요.

쿼리 최적화 기법

이 섹션에 설명된 쿼리 최적화 기술을 사용하면 쿼리를 더 빠르게 실행하거나 Athena의 리소스 한도를 초과하는 쿼리 문제를 해결할 수 있습니다.

조인 최적화

분산 쿼리 엔진에서 조인을 실행하는 다양한 전략이 있습니다. 가장 일반적인 두 가지는 분산 해시 조인과 복잡한 조인 조건을 사용하는 쿼리입니다.

분산 해시 조인

가장 일반적인 유형의 조인은 관계 비교 연산자를 조인 조건으로 사용합니다. Athena는 이 유형의 조인을 분산 해시 조인으로 실행합니다.

분산 해시 조인에서 엔진은 조인 측 중 하나에서 조회 테이블(해시 테이블)을 빌드합니다. 이 조인 측을 빌드 측이라고 합니다. 빌드 측 레코드는 여러 노드에 분산되어 있습니다. 각 노드는 해당 하위 세트에

대한 조회 테이블을 빌드합니다. 그러면 조인의 다른 측(프로브 측)이 노드를 통해 스트리밍됩니다. 프로브 측의 레코드는 빌드 측과 동일한 방식으로 노드에 분산됩니다. 이를 통해 각 노드는 자체 조회 테이블에서 일치하는 레코드를 조회하여 조인을 수행할 수 있습니다.

조인의 빌드 측에서 생성된 조회 테이블이 메모리에 맞지 않으면 쿼리에 실패할 수 있습니다. 빌드 측의 전체 크기가 사용 가능한 메모리보다 작더라도 레코드 분산 편차가 너무 크면 쿼리에 실패할 수 있습니다. 극단적인 경우 모든 레코드의 조인 조건 값이 같고 단일 노드의 메모리에 맞아야 할 수도 있습니다. 값 세트가 동일한 노드로 전송되고 값의 합계가 사용 가능한 메모리를 초과하면 편차가 적은 쿼리라도 실패할 수 있습니다. 노드는 레코드를 디스크로 유출할 수 있지만 유출 시 쿼리 실행 속도가 느려지고 이 방법으로는 쿼리 실패를 막지 못할 수도 있습니다.

Athena는 더 큰 관계를 프로브 측으로 사용하고 더 작은 관계를 빌드 측으로 사용하도록 조인 순서를 변경하려고 합니다. 그러나 Athena는 테이블의 데이터를 관리하지 않기 때문에 정보가 제한적이며 종종 첫 번째 테이블이 더 크고 두 번째 테이블은 더 작다고 가정해야 합니다.

관계 기반 조인 조건을 사용하여 조인을 작성하는 경우 `JOIN` 키워드 왼쪽의 테이블이 프로브 측이고 오른쪽 테이블이 빌드 측이라고 가정합니다. 오른쪽 테이블(빌드 측)이 더 작은 테이블인지 확인합니다. 조인의 빌드 측 크기를 메모리에 맞도록 작게 설정할 수 없다면 빌드 테이블의 하위 세트를 조인하는 쿼리를 여러 번 실행하는 방법을 고려합니다.

기타 조인 유형

복잡한 조인 조건의 쿼리(예: `LIKE`, `>` 또는 다른 연산자를 사용하는 쿼리)는 종종 계산이 까다롭습니다. 최악의 경우에는 조인의 한쪽에 있는 모든 레코드를 조인 반대쪽에 있는 모든 레코드와 비교해야 합니다. 실행 시간은 레코드 수의 제곱에 비례하여 증가하므로 이러한 쿼리는 최대 실행 시간을 초과할 위험이 있습니다.

Athena에서 쿼리를 실행하는 방법을 미리 확인하려면 `EXPLAIN` 문을 사용할 수 있습니다. 자세한 내용은 Athena에서 EXPLAIN 및 EXPLAIN ANALYZE 사용 및 Athena EXPLAIN 문 결과의 이해 단원을 참조하세요.

창 함수 최적화

창 함수는 리소스를 많이 사용하는 작업이므로 쿼리 실행 속도가 느려지거나 쿼리에 실패하고 Query exhausted resources at this scale factor 메시지가 표시될 수 있습니다. 창 함수는 결과를 계산하기 위해 작업하는 모든 레코드를 메모리에 보관합니다. 창 크기가 너무 크면 창 함수의 메모리가 부족해질 수 있습니다.

사용 가능한 메모리 한도 내에서 쿼리를 실행하려면 창 함수가 작동하는 창 크기를 줄입니다. 이렇게 하기 위해 `PARTITIONED BY` 절을 추가하거나 기존 파티셔닝 절의 범위를 좁힐 수 있습니다.

대신 창이 아닌 함수 사용

창 함수가 있는 쿼리를 창 함수 없이 다시 작성할 수도 있습니다. 예를 들어 상위 N개 레코드를 찾기 위해 `row_number`를 사용하는 대신, `ORDER BY` 및 `LIMIT`를 사용할 수 있습니다. 레코드 중복을 제거하기 위해 `row_number` 또는 `rank`를 사용하는 대신, max_by, min_by, arbitrary와 같은 집계 함수를 사용할 수 있습니다.

예를 들어 센서의 업데이트가 포함된 데이터 세트가 있다고 가정합니다. 센서는 주기적으로 배터리 상태를 보고하고 위치와 같은 일부 메타데이터를 포함합니다. 각 센서의 마지막 배터리 상태와 위치를 확인하려는 경우 다음 쿼리를 사용할 수 있습니다.

```
SELECT sensor_id,
       arbitrary(location) AS location,
       max_by(battery_status, updated_at) AS battery_status
FROM sensor_readings
GROUP BY sensor_id
```

위치와 같은 메타데이터는 모든 레코드에서 동일하므로 `arbitrary` 함수를 사용하여 그룹에서 임의의 값을 선택할 수 있습니다.

마지막 배터리 상태를 가져오기 위해 `max_by` 함수를 사용할 수 있습니다. `max_by` 함수는 다른 열의 최댓값을 찾은 레코드에서 열의 값을 선택합니다. 이 경우 그룹 내 마지막 업데이트 시간과 함께 레코드에서 배터리 상태를 반환합니다. 이 쿼리는 창 함수를 사용하는 동등한 쿼리보다 실행 속도가 빠르고 메모리 사용량도 적습니다.

집계 최적화

Athena는 집계를 수행할 때 `GROUP BY` 절의 열을 사용하여 여러 워커 노드에서 레코드를 분산시킵니다. 레코드를 그룹과 일치시키는 작업을 최대한 효율적으로 수행하기 위해 노드에서는 레코드를 메모리에 보관하지만 필요한 경우 디스크로 유출합니다.

`GROUP BY` 절에서 중복 열을 포함하지 않는 것도 좋은 방법입니다. 열 수가 적을수록 필요한 메모리도 적어지기 때문에 더 적은 열을 사용하여 그룹을 설명하는 쿼리가 보다 효율적입니다. 또한 숫자 열은 문자열보다 메모리를 덜 사용합니다. 예를 들어 숫자 카테고리 ID와 카테고리 이름이 모두 있는 데이터 세트를 집계하는 경우 `GROUP BY` 절에서 카테고리 ID 열만 사용합니다.

열이 집계 표현식 또는 `GROUP BY` 절의 일부여야 한다는 점을 처리하기 위해 쿼리에서 `GROUP BY` 절에 열을 포함하기도 합니다. 이 규칙을 따르지 않으면 다음과 같은 오류 메시지가 나타날 수 있습니다.

EXPRESSION_NOT_AGGREGATE: line 1:8: 'category' must be an aggregate expression or appear in GROUP BY clause

GROUP BY 절에 중복 열을 추가할 필요가 없도록 다음 예제와 같이 arbitrary 함수를 사용할 수 있습니다.

```
SELECT country_id,
       arbitrary(country_name) AS country_name,
       COUNT(*) AS city_count
FROM world_cities
GROUP BY country_id
```

ARBITRARY 함수는 그룹에서 임의의 값을 반환합니다. 이 함수는 그룹 내 모든 레코드에서 열 값이 같지만 값으로 그룹을 식별하지 못하는 경우에 유용합니다.

상위 N개 쿼리 최적화

ORDER BY 절은 정렬된 순서로 쿼리 결과를 반환합니다. Athena는 분산 정렬을 사용하여 여러 노드에서 정렬 작업을 병렬로 실행합니다.

결과를 정렬할 필요가 없는 경우 ORDER BY 절을 추가하지 않습니다. 또한 필요한 쿼리가 아닌 경우 내부 쿼리에 ORDER BY를 추가하지 않습니다. 대부분의 경우 쿼리 플래너에서 중복 정렬을 제거할 수 있지만, 보장되지는 않습니다. 이 규칙의 예외는 내부 쿼리가 상위 N개 작업(예: 최근 N개 또는 가장 일반적인 N개 값 찾기)을 수행하는 경우입니다.

Athena에서 ORDER BY가 LIMIT와 함께 나타나는 경우 상위 N개 쿼리를 실행 중이고 적절히 전용 작업을 사용하는 것입니다.

> **Note**
>
> Athena에서 상위 N개를 사용하는 row_number와 같은 창 함수를 종종 탐지할 수 있어도 ORDER BY 및 LIMIT를 사용하는 더 간단한 버전을 사용하는 것이 좋습니다. 자세한 내용은 창 함수 최적화 단원을 참조하십시오.

필수 열만 포함

열이 필요하지 않은 경우 쿼리에 포함하지 않습니다. 쿼리에서 처리해야 하는 데이터가 적을수록 실행 속도가 빨라집니다. 그러면 필요한 메모리 양과 노드 사이에서 전송해야 하는 데이터 양이 모두 줄어듭니다. 열 기반 파일 형식을 사용하는 경우 열 수를 줄이면 Amazon S3에서 읽는 데이터 양도 줄어듭니다.

Athena에서 결과의 열 수에 대한 구체적인 한도는 없지만 쿼리 실행 방식에 따라 가능한 총 열 크기가 제한됩니다. 총 열 크기에는 이름 및 유형이 포함됩니다.

예를 들어 관계 설명자의 크기 제한을 초과하는 관계로 인해 다음 오류가 발생합니다.

GENERIC_INTERNAL_ERROR: io.airlift.bytecode.CompilationException

이 문제를 해결하려면 쿼리의 열 개수를 줄이거나 하위 쿼리를 만들고 더 적은 양의 데이터를 검색하는 JOIN을 사용합니다. 가장 바깥쪽 쿼리에서 SELECT *를 수행하는 쿼리가 있는 경우 *를 필요한 열만 포함하는 목록으로 변경해야 합니다.

근사치를 사용하여 쿼리 최적화

Athena에서는 개별 값, 가장 빈번한 값, 백분위수(근사 중앙값 포함) 계산 및 히스토그램 생성을 위한 근사 집계 함수를 지원합니다. 정확한 값이 필요하지 않은 경우 이 함수를 사용합니다.

COUNT(DISTINCT col) 연산과 달리 approx_distinct는 메모리를 훨씬 적게 사용하고 실행 속도도 더 빠릅니다. 마찬가지로 histogram 대신 numeric_histogram을 사용하면 근사치 계산 방법을 사용하므로 메모리가 적게 소비됩니다.

LIKE 최적화

LIKE를 사용하여 일치하는 문자열을 찾을 수 있지만 문자열이 길면 컴퓨팅 용량을 많이 소비합니다. regexp_like 함수는 대부분의 경우에 더 빠른 대안이며 유연성이 더 뛰어납니다.

찾고 있는 하위 문자열을 고정하여 검색을 최적화할 수도 있습니다. 예를 들어 접두사를 찾으려면 '%*substr*% 대신 '*substr*%'를 사용하는 것이 훨씬 좋습니다. 또는 regexp_like를 사용하는 경우 '^*substr*'을 사용합니다.

UNION 대신 UNION ALL 사용

UNION ALL 및 UNION은 두 쿼리의 결과를 하나의 결과로 결합하는 두 가지 방법입니다. UNION ALL은 첫 번째 쿼리의 레코드를 두 번째 쿼리와 연결합니다. UNION은 동일한 작업을 수행하되 중복된 항목도 제거합니다. UNION은 모든 레코드를 처리하고 중복을 찾아야 하므로 메모리와 컴퓨팅 용량을 많이 소비하지만, UNION ALL은 비교적 속도가 빠른 연산입니다. 레코드의 중복을 제거해야 하는 경우가 아니라면 최상의 성능을 위해 UNION ALL을 사용합니다.

결과 세트가 큰 경우 UNLOAD 사용

쿼리 결과가 커질 것으로 예상되는 경우(예: 수만 행 이상) UNLOAD를 사용하여 결과를 내보냅니다. 대부분의 경우 이 방법이 일반 쿼리를 실행하는 것보다 빠르며 UNLOAD를 사용하는 경우 출력을 더 잘 제어할 수 있습니다.

쿼리 실행이 완료되면 Athena는 결과를 압축되지 않은 단일 CSV 파일로 Amazon S3에 저장합니다. 단, 결과가 압축되지 않고 연산을 병렬화할 수 없으므로 UNLOAD보다 시간이 오래 걸립니다. 반면

UNLOAD는 워커 노드에서 직접 결과를 쓰고 컴퓨팅 클러스터의 병렬 처리를 최대한 활용합니다. 또한 결과를 압축된 형식과 JSON, Parquet 등의 다른 파일 형식으로 기록하도록 UNLOAD를 구성할 수 있습니다.

자세한 내용은 UNLOAD 단원을 참조하십시오.

CTAS 또는 Glue ETL을 사용하여 자주 사용되는 집계 구체화

쿼리 '구체화'는 사전 계산된 복잡한 쿼리 결과(예: 집계 및 조인)를 후속 쿼리에서 재사용할 수 있도록 저장하여 쿼리 성능을 가속화하는 방법입니다.

많은 쿼리에 동일한 조인 및 집계가 포함된 경우 공통 하위 쿼리를 새 테이블로 구체화하고 해당 테이블에 대해 쿼리를 실행할 수 있습니다. 쿼리 결과에서 테이블 생성(CTAS) 또는 Glue ETL과 같은 전용 ETL 도구를 사용하여 새 테이블을 생성할 수 있습니다.

예를 들어 주문 데이터 세트의 다양한 측면을 보여주는 위젯을 포함하는 대시보드를 가정합니다. 각 위젯에는 자체 쿼리가 있지만 모든 쿼리가 동일한 조인 및 필터를 공유합니다. 주문 테이블은 라인 품목 테이블과 결합되며 지난 3개월만 표시하는 필터가 있습니다. 이러한 쿼리의 일반적인 기능을 식별하면 위젯에서 사용할 수 있는 새 테이블을 생성할 수 있습니다. 그러면 중복이 줄어들고 성능이 향상됩니다. 단점은 새 테이블을 최신 상태로 유지해야 한다는 점입니다.

쿼리 결과 재사용

동일한 쿼리는 짧은 시간 안에 여러 번 실행하는 것이 일반적입니다. 예를 들어 여러 사람이 같은 데이터 대시보드를 열 때 이런 상황이 발생할 수 있습니다. 이 경우 쿼리를 실행할 때 이전에 계산된 결과를 재사용하도록 Athena에 지시할 수 있습니다. 재사용할 결과의 최대 수명을 지정합니다. 이전에 동일한 쿼리가 해당 기간 안에 실행된 경우 Athena는 쿼리를 다시 실행하는 대신 해당 결과를 반환합니다. 자세한 내용은 Amazon Athena 사용 설명서의 쿼리 결과 재사용 섹션 및 AWS 빅 데이터 블로그의 Reduce cost and improve query performance with Amazon Athena Query Result Reuse를 참조하세요.

데이터 최적화 기법

성능은 쿼리뿐만 아니라 데이터 세트의 구성 방식, 데이터 세트가 사용하는 파일 형식 및 압축에 따라 달라집니다.

데이터 파티셔닝

파티셔닝은 테이블을 여러 부분으로 나누고 날짜, 국가 또는 지역과 같은 속성을 기반으로 관련 데이터를 함께 보관합니다. 파티션 키는 가상 열 역할을 합니다. 테이블 생성 시 파티션 키를 정의하고 쿼리 필터링에 해당 파티션 키를 사용합니다. 파티션 키 열을 필터링하면 일치하는 파티션의 데이터만 읽습

니다. 예를 들어 데이터 세트가 날짜별로 파티셔닝되어 있고 쿼리에 지난 주에만 일치하는 필터가 있는 경우 지난 주의 데이터만 읽습니다. 파티셔닝에 대한 자세한 내용은 Athena에서 데이터 분할 섹션을 참조하세요.

쿼리를 지원하는 파티션 키를 선택합니다.

파티셔닝은 쿼리 성능에 큰 영향을 미치므로 데이터 세트와 테이블을 설계할 때 파티셔닝을 신중하게 고려해야 합니다. 파티션 키가 너무 많으면 너무 작은 파일과 너무 많은 파일로 데이터 세트가 조각화될 수 있습니다. 반대로 파티션 키가 너무 적거나 전혀 파티셔닝하지 않으면 쿼리에서 필요 이상으로 많은 데이터를 스캔합니다.

드문 쿼리에 대한 최적화 방지

가장 일반적인 쿼리에 대해 최적화하고 드문 쿼리에 대해서는 최적화하지 않는 것이 좋습니다. 예를 들어 쿼리가 일 단위로 표시되는 경우 일부 쿼리가 해당 수준으로 필터링되더라도 시간 단위로 파티셔닝하지 않습니다. 데이터에 세밀한 수준의 타임스탬프 열이 있는 경우 시간 단위로 필터링하는 드문 쿼리에서 이 타임스탬프 열을 사용할 수 있습니다. 드물지만 필요 이상으로 데이터를 조금 더 많이 스캔하더라도 드문 상황에 대비해 전체 성능을 낮추는 작업은 일반적으로 좋은 절충안이 아닙니다.

쿼리에서 스캔해야 하는 데이터를 줄여 성능을 개선하려면 열 기반 파일 형식을 사용하고 레코드를 정렬된 상태로 유지합니다. 시간 단위로 파티셔닝하는 대신 타임스탬프를 기준으로 레코드를 정렬합니다. 기간이 짧은 쿼리의 경우 타임스탬프를 기준으로 정렬하는 방법이 시간 단위로 파티셔닝하는 것만큼 효율적입니다. 또한 타임스탬프를 기준으로 정렬해도 일반적으로 일 단위로 계산되는 기간에서는 쿼리 성능이 저하되지 않습니다. 자세한 내용은 열 기반 파일 형식 사용 단원을 참조하십시오.

수만 개의 파티션이 있는 테이블에 대한 쿼리의 경우 모든 파티션 키에 조건자가 있으면 성능이 개선됩니다. 가장 일반적인 쿼리에 대해 파티셔닝 체계를 설계해야 하는 또 다른 이유가 바로 여기에 있습니다. 자세한 내용은 관계 조건자로 파티션 쿼리 단원을 참조하십시오.

파티션 프로젝션 사용

파티션 프로젝션은 파티션 정보를 AWS Glue Data Catalog에 저장하지 않고 AWS Glue에서 테이블 속성의 규칙으로 저장하는 Athena 기능입니다. Athena에서 파티션 프로젝션으로 구성된 테이블에 대한 쿼리를 계획할 때 테이블의 파티션 프로젝션 규칙을 읽습니다. Athena에서는 AWS Glue Data Catalog에서 파티션을 조회하는 대신, 쿼리와 규칙을 기반으로 메모리에서 읽을 파티션을 계산합니다.

파티션 프로젝션은 파티션 관리를 단순화하는 것 외에도, 파티션 수가 많은 데이터 세트의 성능을 개선할 수 있습니다. 쿼리에 파티션 키의 특정 값 대신 범위가 포함된 경우 카탈로그에서 일치하는 파티션을 조회할 때 파티션이 많을수록 시간이 더 오래 걸립니다. 파티션 프로젝션을 사용하면 카탈로그로 이동하지 않고도 메모리에서 필터를 계산할 수 있으며 속도도 훨씬 더 빠를 수 있습니다.

경우에 따라 파티션 프로젝션으로 인해 성능이 저하되기도 합니다. 한 가지 예로 '스파스' 테이블이 이에 해당합니다. 스파스 테이블에는 파티션 프로젝션 구성에서 설명하는 파티션 키 값의 모든 순열에 대한 데이터가 포함되지 않습니다. 스파스 테이블을 사용하면 쿼리에서 계산된 파티션 세트와 파티션 프로젝션 구성은 데이터가 없어도 모두 Amazon S3에 나열됩니다.

파티션 프로젝션을 사용하는 경우 모든 파티션 키에 조건자를 포함해야 합니다. 불필요한 Amazon S3 나열을 피하려면 가능한 값의 범위를 좁힙니다. 값의 범위가 백만 개인 파티션 키와 해당 파티션 키에 필터가 없는 쿼리를 가정합니다. 쿼리를 실행하기 위해 Athena는 최소 백만 건의 Amazon S3 나열 작업을 수행해야 합니다. 파티션 프로젝션을 사용하든, 카탈로그에 파티션 정보를 저장하든 관계없이 특정 값을 쿼리할 때 쿼리 속도가 가장 빠릅니다. 자세한 내용은 관계 조건자로 파티션 쿼리 단원을 참조하십시오.

파티션 프로젝션에 대해 테이블을 구성하는 경우 지정하는 범위가 적절해야 합니다. 쿼리에서 파티션 키에 조건자가 포함되지 않는 경우 해당 키 범위의 모든 값이 사용됩니다. 데이터 세트가 특정 날짜에 생성된 경우 해당 날짜를 날짜 범위의 시작점으로 사용합니다. 종료 날짜 범위로는 `NOW`를 사용합니다. 값의 수가 많은 숫자 범위는 피하고 대신 injected 형식을 사용하는 것이 좋습니다.

파티션 프로젝션에 대한 자세한 내용은 Amazon Athena를 사용한 파티션 프로젝션 단원을 참조하세요.

파티션 인덱스 사용

파티션 인덱스는 파티션 수가 많은 테이블의 파티션 조회 성능을 개선하는 AWS Glue Data Catalog의 기능입니다.

카탈로그에서 파티션 목록은 관계형 데이터베이스의 테이블과 같습니다. 테이블에는 파티션 키에 해당하는 열과 파티션 위치에 해당하는 추가 열이 있습니다. 파티셔닝 테이블을 쿼리하는 경우 이 테이블을 스캔하면 파티션 위치를 조회합니다.

관계형 데이터베이스와 마찬가지로 인덱스를 추가하여 쿼리 성능을 높일 수 있습니다. 여러 인덱스를 추가하여 다양한 쿼리 패턴을 지원할 수 있습니다. AWS Glue Data Catalog 파티션 인덱스는 관계 연산자와 비교 연산자(예: >, >=, <)를 `AND` 연산자로 결합하여 모두 지원합니다. 자세한 내용은 AWS Glue 개발자 안내서의 Working with partition indexes in AWS Glue 및 AWS 빅 데이터 블로그의 Improve Amazon Athena query performance using AWS Glue Data Catalog partition indexes를 참조하세요.

파티션 키 유형으로 항상 STRING 사용

파티션 키에서 쿼리하는 경우 파티션 필터링을 AWS Glue로 푸시다운하려면 Athena에 `STRING` 유형의 파티션 키가 필요합니다. 파티션 수가 적지 않은 경우 다른 유형을 사용하면 성능이 저하될 수 있습니다. 파티션 키 값이 날짜나 숫자와 비슷한 경우 쿼리에서 적절한 유형으로 변환합니다.

오래되고 비어 있는 파티션 제거

Amazon S3의 파티션에서 데이터를 제거하는 경우(예: Amazon S3 수명 주기 사용) AWS Glue Data Catalog에서도 파티션 항목을 제거해야 합니다. 쿼리 계획 중에 쿼리와 일치하는 모든 파티션이 Amazon S3에 나열됩니다. 빈 파티션이 많은 경우 이러한 파티션을 나열하며 발생하는 오버헤드로 인해 성능이 저하될 수 있습니다.

또한 수천 개의 파티션이 있는 경우 더 이상 관련이 없는 오래된 데이터에 대한 파티션 메타데이터를 제거하는 것이 좋습니다. 예를 들어 쿼리에서 1년이 넘은 데이터를 확인하지 않는 경우 오래된 파티션에 대한 파티션 메타데이터를 주기적으로 제거할 수 있습니다. 파티션 수가 수만 개로 늘어나는 경우 사용하지 않는 파티션을 제거하면 모든 파티션 키에서 조건자를 포함하지 않는 쿼리의 속도를 높일 수 있습니다. 쿼리에서 모든 파티션 키에 조건자를 포함하는 방법에 대한 자세한 내용은 관계 조건자로 파티션 쿼리 섹션을 참조하세요

관계 조건자로 파티션 쿼리

모든 파티션 키에서 관계 조건자를 포함하는 쿼리는 파티션 메타데이터를 직접 로드할 수 있으므로 실행 속도가 더 빠릅니다. 하나 이상의 파티션 키에 조건자가 없거나 조건자에서 값 범위를 선택하는 쿼리는 피합니다. 이러한 쿼리의 경우 일치하는 값을 찾으려면 모든 파티션 목록을 필터링해야 합니다. 대부분의 테이블에서 이러한 오버헤드가 최소한의 수준이지만, 파티션이 수만 개가 넘는 테이블의 경우 이러한 오버헤드는 심각해질 수 있습니다.

쿼리를 다시 작성하여 파티션을 관계 조건자로 필터링할 수 없는 경우 파티션 프로젝션을 시도할 수 있습니다. 자세한 내용은 파티션 프로젝션 사용 단원을 참조하십시오.

파티션 유지 관리에 MSCK REPAIR TABLE 사용 방지

`MSCK REPAIR TABLE`은 실행 시간이 오래 걸릴 수 있으며 새 파티션만 추가하고 오래된 파티션은 제거하지 않으므로 파티션을 효율적으로 관리할 수 있는 방법이 아닙니다(고려 사항 및 제한 참조)

파티션은 AWS Glue Data Catalog API, ALTER TABLE ADD PARTITION 또는 AWS Glue 크롤러를 사용하여 수동으로 관리하는 것이 보다 효율적입니다. 다른 방법으로, 파티션 프로젝션을 사용할 수도 있습니다. 그러면 파티션을 모두 관리할 필요가 없습니다. 자세한 내용은 Amazon Athena를 사용한 파티션 프로젝션 단원을 참조하십시오.

쿼리가 파티셔닝 체계와 호환되는지 확인

`EXPLAIN` 문을 사용하여 쿼리에서 스캔할 파티션을 미리 확인할 수 있습니다. 쿼리 앞에 `EXPLAIN` 키워드를 접두사로 추가하고 각 테이블의 `Fragment 2 [SOURCE]` 출력 하단 근처에서 소스 조각(예:`EXPLAIN`)를 조회합니다. 오른쪽이 파티션 키로 정의된 할당을 조회합니다. 아래 줄에는 쿼리가 실행될 때 스캔할 해당 파티션 키의 모든 값 목록이 포함되어 있습니다.

예를 들어 dt 파티션 키가 있는 테이블에 쿼리가 있고 이 쿼리에 EXPLAIN 접두사를 추가한다고 가정합니다. 쿼리의 값이 날짜이고 필터로 3일 범위를 선택하는 경우 EXPLAIN 출력은 다음과 비슷합니다.

```
dt := dt:string:PARTITION_KEY
    :: [[2023-06-11], [2023-06-12], [2023-06-13]]
```

EXPLAIN 출력에서는 플래너가 이 파티션 키에 대해 쿼리와 일치하는 세 개의 값을 찾았음을 보여줍니다. 또한 해당 값도 표시합니다. EXPLAIN 사용에 대한 자세한 내용은 Athena에서 EXPLAIN 및 EXPLAIN ANALYZE 사용 및 Athena EXPLAIN 문 결과의 이해 섹션을 참조하세요.

열 기반 파일 형식 사용

Parquet 및 ORC와 같은 열 기반 파일 형식은 분산 분석 워크로드를 위해 설계되었습니다. 이 파일 형식에서는 데이터를 행 대신 열을 기준으로 구성합니다. 데이터를 열 형식으로 구성하면 다음과 같은 이점이 있습니다.

- 쿼리에 필요한 열만 로드됨
- 로드해야 하는 전체 데이터 양이 감소함
- 열 값이 함께 저장되므로 데이터를 효율적으로 압축할 수 있음
- 엔진에서 불필요한 데이터를 로드하지 않고 건너뛸 수 있도록 하는 메타데이터가 파일에 포함될 수 있음

파일 메타데이터를 사용하는 방법의 예로, 파일 메타데이터에서 데이터 페이지에 최솟값 및 최댓값에 대한 정보가 포함될 수 있습니다. 쿼리된 값이 메타데이터에 명시된 범위를 벗어나면 페이지를 건너뛸 수 있습니다.

이 메타데이터를 사용하여 성능을 향개선하는 한 가지 방법은 파일 내의 데이터를 정렬하는 것입니다. 예를 들어 created_at 항목이 짧은 기간에 포함된 레코드를 찾는 쿼리를 가정합니다. 데이터가 created_at 열을 기준으로 정렬된 경우 Athena는 파일 메타데이터의 최솟값과 최댓값을 사용하여 데이터 파일의 불필요한 부분을 건너뛸 수 있습니다.

열 기반 파일 형식을 사용하는 경우 파일 크기가 너무 작지 않아야 합니다. 너무 많은 수의 파일 방지에서 설명한 것처럼 작은 파일이 많은 데이터 세트는 성능 문제를 일으킬 수 있습니다. 열 기반 파일 형식에서 특히 그렇습니다. 크기가 작은 파일의 경우 열 기반 파일 형식에서 오버헤드의 단점이 장점보다 더 큽니다.

Parquet 및 ORC는 내부적으로 행 그룹(Parquet) 및 스트라이프(ORC)로 구성됩니다. 행 그룹의 기본 크기는 128MB이고 스트라이프의 경우 64MB입니다. 열이 많은 경우 더 나은 성능을 위해 행 그룹과 스

트라이프 크기를 늘릴 수 있습니다. 행 그룹 또는 스트라이프 크기를 기본값보다 작게 줄이는 것은 권장되지 않습니다.

다른 데이터 형식을 Parquet 또는 ORC로 변환하기 위해 AWS Glue ETL 또는 Athena를 사용할 수 있습니다. ETL에 대해 Athena 사용에 대한 자세한 내용은 ETL 및 데이터 분석에 CTAS 및 INSERT INTO 사용 섹션을 참조하세요.

데이터 압축

Athena에서는 다양한 압축 형식을 지원합니다. 압축을 해제하기 전에 스캔한 바이트 수만큼 비용이 청구되므로 압축된 데이터를 쿼리하는 것이 더 빠르고 비용도 저렴합니다.

gzip 형식은 뛰어난 압축률을 제공하며 다른 여러 도구 및 서비스에서 폭넓게 지원됩니다. zstd(Zstandard) 형식은 성능과 압축률 간 균형이 잘 잡힌 최신 압축 형식입니다.

JSON 및 CSV 데이터와 같은 텍스트 파일을 압축할 때는 파일 수와 파일 크기 사이의 균형을 맞춥니다. 대부분의 압축 형식을 사용하려면 처음부터 리더에서 파일을 읽어야 합니다. 즉, 압축된 텍스트 파일은 일반적으로 병렬로 처리할 수 없습니다. 압축되지 않은 큰 파일은 쿼리 처리 중에 병렬 처리 성능을 높이기 위해 종종 작업자 사이에서 분할되기도 하지만, 대부분의 압축 형식에서는 지원되지 않습니다.

너무 많은 수의 파일 방지에서 설명한 것처럼 파일이 너무 많거나 너무 적지 않은 것이 좋습니다. 파일 수는 쿼리를 처리할 수 있는 작업자 수의 한도이므로 이 규칙은 특히 압축된 파일에 적용됩니다.

Athena에서 압축 사용에 대한 자세한 내용은 Athena 압축 지원 섹션을 참조하세요.

카디널리티가 높은 키에서 조회할 때 버킷팅 사용

버킷팅은 열 중 하나의 값을 기준으로 레코드를 별도의 파일로 분산하는 기법입니다. 이렇게 하면 값이 같은 모든 레코드가 같은 파일에 보관됩니다. 버킷팅은 카디널리티가 높은 키가 있고 많은 쿼리에서 해당 키의 특정 값을 조회하는 경우에 유용합니다.

예를 들어 특정 사용자에 대한 레코드 세트를 쿼리한다고 가정합니다. 데이터가 사용자 ID로 버킷팅되는 경우 Athena는 특정 ID에 대한 레코드를 포함하는 파일과 그렇지 않은 파일을 미리 파악합니다. 이를 통해 Athena가 ID를 포함할 수 있는 파일만 읽을 수 있으므로 읽는 데이터의 양이 크게 줄어듭니다. 또한 특정 ID를 찾기 위해 데이터를 검색하는 데 필요한 컴퓨팅 시간도 줄어듭니다.

버킷팅의 단점

쿼리에서 데이터가 버킷팅되는 열의 여러 값을 자주 검색하는 경우에는 버킷팅의 효율성이 떨어집니다. 쿼리되는 값이 많을수록 전체 또는 대부분의 파일을 읽어야 할 가능성이 커집니다. 예를 들어 버킷

이 3개이고 쿼리에서 세 개의 다른 값을 조회하는 경우 모든 파일을 읽어야 할 수 있습니다. 버킷팅은 쿼리에서 단일 값을 조회할 때 가장 효과적입니다.

자세한 내용은 Athena에서 파티셔닝 및 버킷팅 단원을 참조하십시오.

너무 많은 수의 파일 방지

많은 수의 작은 파일로 구성된 데이터 세트는 전체 쿼리 성능을 저하시킵니다. Athena는 쿼리를 계획할 때 모든 파티션 위치를 나열하므로 시간이 걸립니다. 각 파일을 처리하고 요청하는 데에도 계산 오버헤드가 발생합니다. 따라서 Amazon S3에서 큰 파일 하나를 로드하는 것이 많은 수의 작은 파일에서 동일한 레코드를 로드하는 것보다 빠릅니다.

극단적인 경우에는 Amazon S3 서비스 한도에 도달할 수 있습니다. Amazon S3는 단일 인덱스 파티션에 대해 초당 최대 5,500개의 요청을 지원합니다. 처음에는 버킷을 단일 인덱스 파티션으로 취급하지만 요청 로드가 증가하면 여러 인덱스 파티션으로 분할할 수 있습니다.

Amazon S3는 요청 패턴을 확인하고 키 접두사를 기반으로 분할합니다. 데이터 세트가 수천 개의 파일로 구성된 경우 Athena에서 수신되는 요청이 요청 할당량을 초과할 수 있습니다. 파일 수가 적더라도 동일한 데이터 세트에 대해 여러 개의 동시 쿼리를 수행하면 할당량을 초과할 수 있습니다. 동일한 파일에 액세스하는 다른 애플리케이션이 총 요청 수에 영향을 미칠 수도 있습니다.

요청 속도(`limit`)를 초과하면 Amazon S3는 다음 오류를 반환합니다. 이 오류는 Athena에서 쿼리 상태 정보에 포함됩니다.

SlowDown: Please reduce your request rate

문제를 해결하려면 먼저 오류가 단일 쿼리에서 발생했는지, 아니면 동일한 파일을 읽는 여러 쿼리에서 발생했는지 확인합니다. 후자인 경우 쿼리가 동시에 실행되지 않도록 쿼리 실행을 조정합니다. 이를 위해 애플리케이션에 대기열 메커니즘을 추가하거나 재시도를 추가합니다.

단일 쿼리를 실행할 때 오류가 발생하는 경우 데이터 파일을 결합하거나 읽는 파일 수를 줄이도록 쿼리를 수정합니다. 작은 파일을 결합하기에 가장 좋은 시점은 파일을 쓰기 전입니다. 이를 위해 다음 기법을 고려합니다.

- 더 큰 파일을 쓰도록 파일을 쓰는 프로세스를 변경합니다. 예를 들어 레코드를 쓰기 전에 더 오래 레코드를 버퍼링할 수 있습니다.
- Amazon S3의 한 위치에 파일을 배치하고 Glue ETL과 같은 도구를 사용하여 파일을 더 큰 파일로 결합합니다. 그런 다음 큰 파일을 테이블이 가리키는 위치로 이동합니다. 자세한 내용은 AWS Glue 개발자 안내서의 Reading input files in larger groups 및 AWS re:Post 지식 센터의 더 큰 파일을 출력하도록 AWS Glue ETL 작업을 구성하려면 어떻게 해야 합니까?를 참조하세요.

- 파티션 키 수를 줄입니다. 파티션 키가 너무 많으면 각 파티션에 레코드 수가 적어 작은 파일 수가 너무 많아질 수 있습니다. 생성할 파티션 결정에 대한 자세한 내용은 쿼리를 지원하는 파티션 키를 선택합니다. 섹션을 참조하세요.

파티션 이외의 추가 스토리지 계층 구조 방지

쿼리 계획 오버헤드를 방지하려면 각 파티션 위치에 파일을 플랫 구조로 저장합니다. 추가 디렉터리 계층 구조는 사용하지 않습니다.

Athena는 쿼리를 계획할 때 쿼리와 일치하는 모든 파티션의 모든 파일을 나열합니다. Amazon S3에 디렉터리 자체는 없지만, / 슬래시를 디렉터리 구분 기호로 해석하는 것이 규칙입니다. Athena는 파티션 위치를 나열할 때 찾은 모든 디렉터리를 반복적으로 나열합니다. 파티션 내 파일을 계층 구조로 구성하면 여러 차례 나열이 수행됩니다.

모든 파일이 파티션 위치에 바로 있으면 대부분의 경우 한 번의 나열 작업만 수행하면 됩니다. 그러나 Amazon S3는 나열 작업당 1,000개의 객체만 반환하므로 파티션에 1,000개가 넘는 파일이 있으면 순차 나열 작업이 여러 차례 필요합니다. 파티션에 1,000개가 넘는 파일이 있으면 더 심각한 다른 성능 문제가 발생할 수도 있습니다. 자세한 내용은 너무 많은 수의 파일 방지 단원을 참조하십시오.

필요한 경우에만 SymlinkTextInputFormat 사용

SymlinkTextInputFormat 기법을 사용하면 테이블의 파일이 파티션으로 깔끔하게 구성되지 않은 상황을 해결할 수 있습니다. 예를 들어 모든 파일에서 접두사가 같거나 스키마가 다른 파일이 같은 위치에 있는 경우 symlink가 유용할 수 있습니다.

하지만 symlink를 사용하면 쿼리 실행이 더 간접적으로 이루어집니다. 이렇게 간접적으로 이루어진 쿼리 실행은 전체 성능에 영향을 미칩니다. symlink 파일을 읽고 symlink 파일이 정의한 위치를 나열해야 합니다. 이로 인해 일반적인 Hive 테이블에서는 필요하지 않은 여러 번의 왕복 작업이 Amazon S3에 추가됩니다. 결론적으로, 파일 재구성과 같은 더 나은 옵션을 사용할 수 없는 경우에만 `SymlinkTextInputFormat`을 사용해야 합니다.

추가적인 리소스

Athena 성능 튜닝에 대한 추가 정보는 다음 리소스를 고려하세요.

- AWS Big Data Blog(빅 데이터 블로그) 게시물 Top 10 performance tuning tips for Amazon Athena(Amazon Athena의 성능 튜닝을 위한 10가지 팁) 읽기
- 페더레이션된 쿼리의 성능을 개선하기 위해 조건자 푸시다운을 사용하는 방법에 대한 문서는 AWS 빅 데이터 블로그의 Improve federated queries with predicate pushdown in Amazon Athena를 참조하세요.

- Athena 쿼리 엔진의 성능 최적화에 대한 문서는 AWS 빅 데이터 블로그의 Run queries 3x faster with up to 70% cost savings on the latest Amazon Athena engine을 참조하세요.
- 다른 AWS Big Data Blog(빅 데이터 블로그)의 Athena 게시물 읽기
- Amazon Athena 태그를 사용하여 AWS re:Post에 대해 질문하기
- AWS 지식 센터의 Athena 주제 참조
- AWS Support에 문의(AWS Management Console에서 지원, 지원 센터 클릭)

Amazon S3 제한 방지

제한은 서비스, 애플리케이션 또는 시스템 사용 속도를 제한하는 프로세스입니다. AWS에서는 제한을 사용하여 Amazon S3 서비스의 과도한 사용을 방지하고 모든 사용자에 대한 Amazon S3의 가용성과 응답성을 높일 수 있습니다. 그러나 제한 기능은 Amazon S3와의 데이터 전송 속도를 제한하므로 상호 작용이 제한되지 않도록 하는 것이 중요합니다.

서비스 수준에서 제한 감소

서비스 수준에서 Amazon S3 제한을 방지하려면 사용량을 모니터링하고 서비스 할당량을 조정하거나 파티셔닝과 같은 특정 기술을 사용할 수 있습니다. 다음은 제한으로 이어질 수 있는 몇 가지 조건입니다.

- 계정의 API 요청 한도 초과 - Amazon S3에는 계정 유형 및 사용량에 따른 기본 API 요청 한도가 있습니다. 단일 객체에서 초당 최대 요청 수를 초과하는 경우 Amazon S3 서비스의 과부하를 방지하기 위해 요청이 제한될 수 있습니다.
- 불충분한 데이터 파티셔닝 -데이터를 적절하게 파티셔닝하지 않고 대량의 데이터를 전송하는 경우 Amazon S3에서 요청을 제한할 수 있습니다. 파티셔닝에 대한 자세한 내용은 이 문서의 파티셔닝 사용 섹션을 참조하세요.
- 많은 수의 작은 객체 - 가능하면 많은 수의 작은 파일을 만들지 않습니다. Amazon S3에는 파티셔닝된 접두사 1개에 대해 초당 5,500개의 GET 요청 한도가 있으며, Athena 쿼리는 이 한도를 공유합니다. 하나의 쿼리에서 수백만 개의 작은 객체를 스캔하는 경우 쿼리는 Amazon S3에 의해 제한될 수 있습니다.

과도한 스캔을 피하려면 AWS Glue ETL을 사용하여 파일을 주기적으로 압축하거나 테이블을 분할하고 파티션 키 필터를 추가하세요. 자세한 정보는 다음 리소스를 참조하세요.

- 더 큰 파일을 출력하도록 AWS Glue ETL 작업을 구성하려면 어떻게 해야 합니까? (AWS 지식 센터)
- Reading input files in larger groups(AWS Glue 개발자 안내서)

테이블 최적화

제한 문제가 발생하는 경우 데이터를 구조화하는 것이 중요합니다. Amazon S3는 대량의 데이터를 처리할 수 있지만 데이터가 구조화되는 방식 때문에 때때로 제한이 발생합니다.

다음 섹션에서는 제한 문제를 방지하기 위해 Amazon S3에서 데이터를 구조화하는 방법에 대한 몇 가지 제안을 제공합니다.

파티셔닝 사용

파티셔닝을 사용하면 언제든지 액세스해야 하는 데이터의 양을 제한하여 제한을 줄일 수 있습니다. 특정 열에서 데이터를 파티셔닝하면 요청을 여러 객체에서 고르게 분산하고 단일 객체에 대한 요청 수를 줄일 수 있습니다. 스캔해야 하는 데이터의 양을 줄이면 쿼리 성능이 향상되고 비용이 절감됩니다.

테이블을 생성할 때 가상 열 역할을 하는 파티션을 정의할 수 있습니다. `CREATE TABLE` 문에서 파티션이 있는 테이블을 생성하려면 `PARTITIONED BY (column_name data_type)` 절을 사용하여 데이터를 파티셔닝하는 키를 정의합니다.

쿼리에서 스캔하는 파티션을 제한하기 위해 쿼리의 `WHERE` 절에서 파티션을 조건자로 지정할 수 있습니다. 따라서 필터로 자주 사용되는 열이 파티셔닝에 적합합니다. 시간 간격에 따라 데이터를 파티셔닝하는 것이 일반적이며, 이때 여러 수준의 파티셔닝 체계가 형성될 수 있습니다.

파티셔닝에도 비용이 듭니다. 테이블에서 파티션 수를 늘리면 파티션 메타데이터를 검색하고 처리하는 데 필요한 시간도 늘어납니다. 따라서 과도하게 파티셔닝하면 보다 신중한 파티셔닝으로 얻을 수 있는 혜택이 사라질 수 있습니다. 데이터가 한 파티션 값으로 심하게 편중되고 대부분의 쿼리에서 이 값을 사용하는 경우 추가 오버헤드가 발생할 수 있습니다.

Athena에서 파티셔닝에 대한 자세한 내용은 파티셔닝이란 무엇인가요? 섹션을 참조하세요.

데이터 버킷팅

데이터를 파티셔닝하는 또 다른 방법은 데이터를 단일 파티션 내에 버킷팅하는 것입니다. 버킷팅에서는 함께 그룹화하려는 행이 포함된 하나 이상의 열을 지정합니다. 그런 다음 해당 행을 여러 버킷에 넣습니다. 이 방식으로 읽어야 하는 버킷만 쿼리하여 스캔해야 하는 데이터 행 수를 줄일 수 있습니다.

버킷팅에 사용할 열을 선택하는 경우 카디널리티가 높고(즉, 개별 값이 많음) 균일하게 분산되었으며 데이터를 필터링하는 데 자주 사용되는 열을 선택합니다. 버킷팅에 사용하기에 좋은 열의 예로는 ID 열과 같은 기본 키가 있습니다.

Athena에서 버킷팅 사용에 대한 자세한 내용은 버킷팅이란 무엇인가요? 섹션을 참조하세요.

AWS Glue 파티션 인덱스 사용

AWS Glue 파티션 인덱스를 사용하여 하나 이상의 파티션 값을 기반으로 테이블의 데이터를 구성할 수 있습니다. AWS Glue 파티션 인덱스를 사용하면 데이터 전송 횟수, 데이터 처리량, 쿼리 처리 시간을 줄일 수 있습니다.

AWS Glue 파티션 인덱스는 파티션 키 및 해당 값을 포함하여 테이블의 파티션에 대한 정보가 들어 있는 메타데이터 파일입니다. 파티션 인덱스는 Amazon S3 버킷에 저장되며 테이블에 새 파티션이 추가되면 AWS Glue에 의해 자동으로 업데이트됩니다.

AWS Glue 파티션 인덱스가 존재하면 테이블의 모든 파티션을 로드하는 대신 쿼리에서 파티션의 하위 세트를 가져오려고 합니다. 쿼리는 쿼리와 관련된 데이터의 하위 세트에서만 실행됩니다.

AWS Glue에서 테이블을 생성할 때 테이블에 정의된 파티션 키 조합에서 파티션 인덱스를 생성할 수 있습니다. 테이블에서 파티션 인덱스를 하나 이상 생성한 후에는 파티션 필터링을 활성화하는 속성을 테이블에 추가해야 합니다. 그런 다음 Athena에서 테이블을 쿼리할 수 있습니다.

AWS Glue에서 파티션 인덱스 생성에 대한 자세한 내용은 AWS Glue 개발자 안내서의 Working with partition indexes in AWS Glue를 참조하세요. 파티션 필터링을 활성화하기 위해 테이블 속성을 추가하는 방법에 대한 자세한 내용은 AWS Glue 파티션 인덱싱 및 필터링 섹션을 참조하세요.

데이터 압축 및 파일 분할 사용

파일이 최적 크기이거나 파일을 논리적 그룹으로 분할할 수 있는 경우 데이터 압축을 통해 쿼리 속도를 크게 높일 수 있습니다. 일반적으로 압축률이 높을수록 데이터를 압축하고 압축 해제하는 데 더 많은 CPU 사이클이 필요합니다. Athena의 경우 기본적으로 데이터를 압축하는 Apache Parquet 또는 Apache ORC를 사용하는 것이 좋습니다. Athena의 데이터 압축에 대한 자세한 내용은 Athena 압축 지원 섹션을 참조하세요.

파일을 분할하면 Athena에서 단일 파일을 읽는 작업을 여러 리더에 분산시켜 병렬 처리 성능을 높일 수 있습니다. 단일 파일을 분할할 수 없는 경우 하나의 리더만 파일을 읽고 다른 리더가 유휴 상태로 둘 수 있습니다. Apache Parquet 및 Apache ORC도 분할 가능한 파일을 지원합니다.

최적화된 열 기반 데이터 스토어 사용

데이터를 열 형식으로 변환하면 Athena 쿼리 성능이 크게 향상됩니다. 열 기반 파일을 생성할 때 고려할 한 가지 최적화 기법은 파티션 키를 기준으로 데이터를 정렬하는 것입니다.

Apache Parquet 및 Apache ORC는 흔히 사용되는 오픈 소스 열 기반 데이터 스토어입니다. 기존 Amazon S3 데이터 소스를 이러한 형식 중 하나로 변환하는 방법에 대한 자세한 내용은 열 기반 형식으로 변환 섹션을 참조하세요.

하나의 큰 Parquet 블록 크기 또는 ORC 스트라이프 크기 사용

Parquet 및 ORC에는 최적화를 위해 조정할 수 있는 데이터 스토리지 파라미터가 있습니다. Parquet에서는 블록 크기를 최적화할 수 있습니다. ORC에서는 스트라이프 크기를 최적화할 수 있습니다. 블록이나 스트라이프가 클수록 각 블록이나 스트라이프에 저장할 수 있는 행 수가 많아집니다. 기본적으로 Parquet 블록 크기는 128MB이고 ORC 스트라이프 크기는 64MB입니다.

ORC 스트라이프가 8MB(기본값 `hive.orc.max_buffer_size`) 미만인 경우 Athena는 전체 ORC 스트라이프를 읽습니다. 스트라이프 크기가 작은 경우 Athena는 초당 입력 및 출력 작업과 열 선택성 사이에서 이와 같이 절충합니다.

열 수가 매우 많은 테이블의 경우 블록 또는 스트라이프 크기가 작으면 필요 이상으로 많은 데이터가 스캔될 수 있습니다. 이 경우에는 블록 크기가 클수록 효율적입니다.

복잡한 유형에 대해 ORC 사용

현재 Parquet에 저장된 복잡한 데이터 형식(예: `array`, `map` 또는 `struct`)의 열을 쿼리하는 경우 Athena는 지정된 열만 선택적으로 읽는 대신, 전체 데이터 행을 읽습니다. 이것은 Athena에서 알려진 문제입니다. 이 문제를 해결하려면 ORC를 사용합니다.

압축 알고리즘 선택

사용자가 구성할 수 있는 또 다른 파라미터는 데이터 블록의 압축 알고리즘입니다. Athena에서 Parquet 및 ORC용으로 지원되는 압축 알고리즘에 대한 자세한 내용은 Athena 압축 지원을 참조하세요.

Athena에서 열 기반 스토리지 형식 최적화에 대한 자세한 내용은 AWS 빅 데이터 블로그 게시물 Top 10 Performance Tuning Tips for Amazon Athena의 'Optimize columnar data store generation' 섹션을 참조하세요.

Iceberg 테이블 사용

Apache Iceberg는 Amazon S3에서 최적화된 사용을 위해 설계된 초대형 분석 데이터 세트를 위한 오픈 테이블 형식입니다. Iceberg 테이블을 사용하면 Amazon S3에서 제한을 줄이는 데 도움이 될 수 있습니다.

Iceberg 테이블은 다음과 같은 이점을 제공합니다.

- Iceberg 테이블은 하나 이상의 열로 분할할 수 있습니다. 이렇게 하면 데이터 액세스가 최적화되고 쿼리로 스캔해야 하는 데이터의 양이 줄어듭니다.

- Iceberg 객체 스토리지 모드는 Amazon S3에서 사용할 수 있도록 Iceberg 테이블을 최적화하므로 대량의 데이터와 많은 쿼리 워크로드를 처리할 수 있습니다.
- 객체 스토리지 모드의 Iceberg 테이블은 확장 가능하고 내결함성이 뛰어나며 내구성이 뛰어나므로 제한을 줄이는 데 도움이 될 수 있습니다.
- ACID 트랜잭션 지원이란, 여러 사용자가 원자 단위로 Amazon S3 객체를 추가하고 삭제할 수 있음을 의미합니다.

Apache Iceberg에 대한 자세한 내용은 Apache Iceberg를 참조하세요. Athena에서 Apache Iceberg 테이블 사용에 대한 자세한 내용은 Iceberg 테이블 사용을 참조하세요.

쿼리 최적화

Athena에서 SQL 쿼리를 최적화하려는 경우 이 섹션의 제안 사항을 사용합니다.

ORDER BY 절과 함께 LIMIT 사용

`ORDER BY` 절은 정렬된 순서로 데이터를 반환합니다. 이를 위해 Athena에서 모든 데이터 행을 단일 워커 노드로 보낸 후 행을 정렬해야 합니다. 이러한 유형의 쿼리는 오래 실행되거나 실패할 수도 있습니다.

쿼리의 효율성을 높이려면 상위 또는 하위 *N*개 값을 확인하고 `LIMIT` 절도 사용합니다. 그러면 정렬과 제한을 단일 작업자가 아닌 개별 워커 노드로 푸시하여 정렬 비용을 크게 줄일 수 있습니다.

JOIN 절 최적화

두 테이블을 조인하는 경우 Athena는 오른쪽에 있는 테이블을 워커 노드에 분산시킨 후 왼쪽의 테이블을 스트리밍하여 조인을 수행합니다.

따라서 조인 왼쪽에 큰 테이블을 지정하고 오른쪽에 작은 테이블을 지정합니다. 이렇게 하면 Athena는 메모리를 덜 사용하고 지연 시간을 줄이면서 쿼리를 실행합니다.

다음 사항에도 주의하세요.

- 여러 `JOIN` 명령을 사용하는 경우 가장 큰 테이블부터 가장 작은 테이블의 순서로 지정합니다.
- 쿼리에 필요하지 않는 한 크로스 조인을 사용하지 않습니다.

GROUP BY 절 최적화

`GROUP BY` 연산자는 `GROUP BY` 열을 기반으로 워커 노드에 행을 분산시킵니다. 이러한 열은 메모리에서 참조되며 행을 모을 때 값이 비교됩니다. `GROUP BY` 열이 일치하면 값이 함께 집계됩니다. 이 프로

세스의 작동 방식을 고려하면 카디널리티가 가장 높은 열부터 가장 낮은 열의 순서로 정렬하는 것이 좋습니다.

문자열 대신 숫자 사용

숫자는 문자열에 비해 메모리 사용량이 적고 처리 속도가 빠르므로 가능하면 문자열 대신 숫자를 사용합니다.

열 수 제한

데이터를 저장하는 데 필요한 총 메모리 양을 줄이려면 `SELECT` 문에 지정된 열 수를 제한합니다.

LIKE 대신 정규식 사용

큰 문자열에서 `LIKE '%string%'`과 같은 절을 포함하는 쿼리는 컴퓨팅 집약적일 수 있습니다. 문자열 열에서 여러 값을 필터링하는 경우 대신 regexp_like() 함수와 정규식을 사용합니다. 값의 긴 목록을 비교할 때 특히 유용합니다.

LIMIT 절 사용

쿼리를 실행할 때 모든 열을 선택하는 대신 `LIMIT` 절을 사용하여 필요한 열만 반환합니다. 이렇게 하면 쿼리 실행 파이프라인을 통해 처리되는 데이터 세트의 크기가 줄어듭니다. `LIMIT` 절은 문자열 기반 열 수가 많은 테이블을 쿼리할 때 더 유용합니다. `LIMIT` 절은 쿼리에서 여러 조인 또는 집계를 수행할 때 유용합니다.

추가적인 리소스

Amazon Simple Storage Service 사용 설명서의 모범 사례 설계 패턴: Amazon S3 성능 최적화.

Athena의 성능 튜닝

Athena 압축 지원

주제

- 압축 형식 지정
- 압축 안 함 지정
- 참고 및 리소스
- 파일 형식별 Hive 테이블 압축 지원
- 파일 형식별 Iceberg 테이블 압축 지원
- Athena에서 ZSTD 압축 수준 사용

Athena는 여러 압축 형식을 사용하는 테이블에서 데이터를 읽는 것을 비롯하여, 데이터 읽기 및 쓰기를 위한 다양한 압축 형식을 지원합니다. 예를 들어 Athena는 일부 Parquet 파일이 Snappy로 압축되고 다른 Parquet 파일이 GZIP으로 압축된 경우 Parquet 파일 형식을 사용하는 테이블의 데이터를 성공적으로 읽을 수 있습니다. ORC, 텍스트 파일, JSON 스토리지 형식에도 동일한 원칙이 적용됩니다.

Athena는 다음 압축 형식을 지원합니다.

- BZIP2 – Burrows-Wheeler 알고리즘을 사용하는 형식입니다.
- DEFLATE - LZSS 및 Huffman 코딩을 기반으로 한 압축 알고리즘입니다. Deflate는 Avro 파일 형식에만 해당됩니다.
- GZIP - Deflate를 기반으로 한 압축 알고리즘입니다. Athena 엔진 버전 2 및 3의 Hive 테이블과 Athena 엔진 버전 2의 Iceberg 테이블의 경우 GZIP은 Parquet 및 텍스트 파일 스토리지 형식의 파일에 대한 기본 쓰기 압축 형식입니다. `tar.gz` 형식의 파일은 지원되지 않습니다.
- LZ4 - Lempel-Ziv 77(LZ7) 패밀리의 이 멤버도 최대 데이터 압축이 아닌 압축 및 압축 해제 속도에 중점을 둡니다. LZ4에는 다음과 같은 프레이밍 형식이 있습니다.
  - LZ4 Raw/Unframed – LZ4 블록 압축 형식의 프레이밍되지 않은 표준 구현입니다. 자세한 내용은 GitHub에서 LZ4 블록 형식 설명을 참조하세요.
  - LZ4 framed - LZ4의 일반적인 프레이밍 구현입니다. 자세한 내용은 GitHub에서 LZ4 프레임 형식 설명을 참조하세요.
  - LZ4 hadoop-compatible - LZ4의 Apache Hadoop 구현입니다. 이 구현은 LZ4 압축을 BlockCompressorStream.java 클래스로 래핑합니다.
- LZO - 최대 데이터 압축이 아닌 높은 압축 및 압축 해제 속도에 중점을 둔 Lempel–Ziv–Oberhumer 알고리즘을 사용하는 형식입니다. LZO에는 두 가지 구현이 있습니다.
  - Standard LZO - 자세한 내용은 Oberhumer 웹 사이트에서 LZO 요약을 참조하세요.
  - LZO hadoop-compatible - 이 구현은 LZO 알고리즘을 BlockCompressorStream.java 클래스로 래핑합니다.
- SNAPPY – Lempel-Ziv 77 (LZ7) 패밀리의 일부인 압축 알고리즘입니다. Snappy는 데이터를 최대한 압축하는 것이 아니라, 높은 압축 및 압축 해제 속도에 중점을 둡니다.
- ZLIB - Deflate를 기반으로 한 ZLIB는 ORC 데이터 스토리지 형식의 파일에 대한 기본 쓰기 압축 형식입니다. 자세한 내용은 GitHub에서 zlib 페이지를 참조하세요.
- ZSTD - Zstandard 실시간 데이터 압축 알고리즘은 높은 압축비를 제공하는 빠른 압축 알고리즘입니다. Zstandard(ZSTD) 라이브러리는 BSD 라이선스를 사용하는 오픈 소스 소프트웨어로 제공됩니다. ZSTD는 Iceberg 테이블의 기본 압축입니다. ZSTD 압축 데이터를 작성할 때 Athena는 기본적으로 ZSTD 압축 수준 3을 사용합니다. Athena의 ZSTD 압축 수준 사용에 대한 자세한 내용은 Athena에서 ZSTD 압축 수준 사용 섹션을 참조하세요.

압축 형식 지정

CREATE TABLE 또는 CTAS 문을 작성할 때, Athena가 해당 테이블에 쓸 경우 사용할 압축 유형을 지정하는 압축 속성을 지정할 수 있습니다.

- CTAS의 경우 CTAS 테이블 속성 섹션을 참조하세요. 예를 보려면 CTAS 쿼리 예제을 참조하세요.
- CREATE TABLE의 경우 ALTER TABLE SET TBLPROPERTIES에서 압축 테이블 속성 목록을 참조하세요.

압축 안 함 지정

CREATE TABLE 문은 압축되지 않은 파일을 지원합니다. 압축되지 않은 파일을 작성하려면 다음 구문을 사용합니다.

- 테이블 만들기(텍스트 파일 또는 JSON) —TBLPROPERTIES를 지정합니다.`write.compression = NONE`.
- 테이블 만들기(Parquet) —`parquet.compression = UNCOMPRESSED`에서 TBLPROPERTIES를 지정합니다.
- 테이블 생성 (ORC) — `orc.compress = NONE`에서 TBLPROPERTIES를 지정합니다.

참고 및 리소스

- 현재 `.GZ` 또는 `.BZIP2`와 같은 대문자 파일 확장명은 Athena에서 인식되지 않습니다. 파일 확장명이 대문자인 데이터 세트을 사용하지 않거나 데이터 파일 확장명의 이름을 소문자로 바꿉니다.
- CSV, TSV 및 JSON 형식 데이터의 경우 Athena는 파일 확장명에서 압축 형식을 결정합니다. 파일 확장자가 없는 경우 Athena는 데이터를 압축되지 않은 일반 텍스트로 처리합니다. 데이터가 압축된 경우 파일 이름에 gz와 같은 압축 확장명이 포함되어 있는지 확인하세요.
- ZIP 파일 형식은 지원되지 않습니다.
- Athena에서 querying Amazon Data Firehose 로그를 쿼리하는 경우, 지원되는 형식에는 GZIP 압축 또는 SNAPPY 압축을 사용하는 ORC 파일이 포함됩니다.
- 압축 사용에 대한 자세한 내용은 AWS 빅 데이터 블로그 게시물 Top 10 performance tuning tips for Amazon Athena(Amazon Athena의 성능 튜닝을 위한 10가지 팁)의 섹션 3(“Compress and split files(파일 압축 및 분할)”)을 참조하세요.

파일 형식별 Hive 테이블 압축 지원

Athena에서 Hive 압축 지원은 엔진 버전에 따라 다릅니다.

Athena 엔진 버전 3의 Hive 압축 지원

다음 표에는 Apache Hive의 스토리지 파일 형식에 대한 Athena 엔진 버전 3의 압축 형식 지원이 요약되어 있습니다. 텍스트 파일 형식에는 TSV, CSV, JSON 및 텍스트용 사용자 정의 SerDes가 포함됩니다. 셀의 "예" 또는 "아니요"는 별도로 명시된 경우를 제외하고 읽기 작업과 쓰기 작업에 동일하게 적용됩니다. 이 테이블에서는 CREATE TABLE, CTAS 및 INSERT INTO를 쓰기 작업으로 간주합니다. Athena의 ZSTD 압축 수준 사용에 대한 자세한 내용은 Athena에서 ZSTD 압축 수준 사용 섹션을 참조하세요.

	Avro	Ion	ORC	PARQUET	텍스트 파일
bzip2	예	예	아니요	아니요	예
DEFLATE	예	아니요	아니요	아니요	아니요
GZIP	아니요	예	아니요	예	예
LZ4	아니요	예	예	예	예
LZO	아니요	쓰기 - 아니요 읽기 - 예	아니요	예	쓰기 - 아니요 읽기 - 예
Snappy	예	예	예	예	예
ZLIB	아니요	아니요	예	아니요	아니요
ZSTD	예	예	예	예	예
NONE	예	예	예	예	예

Athena 엔진 버전 2의 Hive 압축 지원

다음 표에는 Apache Hive용 Athena 엔진 버전 2의 압축 형식 지원이 요약되어 있습니다. 텍스트 파일 형식에는 TSV, CSV, JSON 및 텍스트용 사용자 정의 SerDes가 포함됩니다. 셀의 “예” 또는 “아니요”는 별도로 명시된 경우를 제외하고 읽기 작업과 쓰기 작업에 동일하게 적용됩니다. 이 테이블에서는 CREATE TABLE, CTAS 및 INSERT INTO를 쓰기 작업으로 간주합니다.

	Avro	Ion	ORC	PARQUET	텍스트 파일
bzip2	예	예	아니요	아니요	예
DEFLATE	예	아니요	아니요	아니요	아니요
GZIP	아니요	예	아니요	예	예
LZ4	아니요	아니요	예	쓰기 - 예 읽기 - 아니요	쓰기 - 아니요 읽기 - 예
LZO	아니요	쓰기 - 아니요 읽기 - 예	아니요	예	쓰기 - 아니요 읽기 - 예
Snappy	예	예	예	예	예
ZLIB	아니요	아니요	예	아니요	아니요
ZSTD	아니요	예	예	예	예
NONE	예	예	예	예	예

파일 형식별 Iceberg 테이블 압축 지원

Athena에서 Apache Iceberg 압축 지원은 엔진 버전에 따라 다릅니다.

Athena 엔진 버전 3의 Iceberg 압축 지원

다음 표에는 Apache Iceberg의 스토리지 파일 형식에 대한 Athena 엔진 버전 3의 압축 형식 지원이 요약되어 있습니다. 셀의 “예” 또는 “아니요”는 별도로 명시된 경우를 제외하고 읽기 작업과 쓰기 작업에 동일하게 적용됩니다. 이 테이블에서는 CREATE TABLE, CTAS 및 INSERT INTO를 쓰기 작업으로 간주합니다. Athena 엔진 버전 3에서 Iceberg에 대한 기본 스토리지 형식은 Parquet입니다. Athena 엔진 버전 3에서 Iceberg에 대한 기본 압축 형식은 ZSTD입니다. Athena의 ZSTD 압축 수준 사용에 대한 자세한 내용은 Athena에서 ZSTD 압축 수준 사용 섹션을 참조하세요.

	Avro	ORC	Parquet(기본값)
bzip2	아니요	아니요	아니요
GZIP	예	아니요	예
LZ4	아니요	예	아니요
Snappy	예	예	예
ZLIB	아니요	예	아니요
ZSTD	예	예	예(기본값)
NONE	예(None 또는 Deflate 지정)	예	예(None 또는 Uncompressed 지정)

Athena 엔진 버전 2의 Iceberg 압축 지원

다음 표에는 Apache Iceberg용 Athena 엔진 버전 2의 압축 형식 지원이 요약되어 있습니다. 셀의 “예” 또는 “아니요”는 별도로 명시된 경우를 제외하고 읽기 작업과 쓰기 작업에 동일하게 적용됩니다. 이 테이블에서는 CREATE TABLE, CTAS 및 INSERT INTO를 쓰기 작업으로 간주합니다. Athena 엔진 버전 2에서 Iceberg에 대한 기본 스토리지 형식은 Parquet입니다. Athena 엔진 버전 2에서 Iceberg에 대한 기본 압축 형식은 GZIP입니다.

	Avro (지원되지 않음)	ORC (지원되지 않음)	Parquet(기본값)
bzip2	아니요	아니요	아니요

	Avro (지원되지 않음)	ORC (지원되지 않음)	Parquet(기본값)
GZIP	아니요	아니요	예(기본값)
LZ4	아니요	아니요	아니요
Snappy	아니요	아니요	예
ZLIB	아니요	아니요	아니요
ZSTD	아니요	아니요	예
NONE	아니요	아니요	예

Athena에서 ZSTD 압축 수준 사용

Zstandard 실시간 데이터 압축 알고리즘은 높은 압축비를 제공하는 빠른 압축 알고리즘입니다. Zstandard(ZSTD) 라이브러리는 오픈 소스 소프트웨어이며 BSD 라이선스를 사용합니다. Athena는 ZSTD 압축 ORC, Parquet 및 텍스트 파일 데이터 읽기 및 쓰기를 지원합니다.

ZSTD 압축 수준을 사용하여 요구 사항에 따라 압축률과 속도를 조정할 수 있습니다. ZSTD 라이브러리는 1~22의 압축 수준을 지원합니다. Athena는 기본적으로 ZSTD 압축 수준 3을 사용합니다.

압축 수준은 압축 속도와 달성된 압축량 간의 세분화된 균형을 제공합니다. 압축 수준이 낮을수록 속도는 빨라지지만 파일 크기는 커집니다. 예를 들어 속도가 가장 중요한 경우 수준 1을 사용하고 크기가 가장 중요한 경우 수준 22를 사용할 수 있습니다. 수준 3이 많은 사용 사례에 적합하며 기본값입니다. 수준 20부터는 더 많은 메모리가 필요하므로 주의하여 사용하세요. ZSTD 라이브러리는 또한 압축 속도와 비율의 범위를 확장하는 네거티브 압축 수준을 제공합니다. 자세한 내용은 Zstandard Compression RFC(Zstandard 압축 RFC)를 참조하세요.

많은 압축 수준은 미세 조정을 위한 상당한 기회를 제공합니다. 그러나 압축 수준을 결정할 때 데이터를 측정하고 장단점을 고려해야 합니다. 압축 속도와 압축된 데이터 크기 간의 적절한 균형을 위해 기본 수준 3 또는 수준 6~9를 사용하는 것이 좋습니다. 크기가 가장 중요하고 압축 속도가 중요하지 않은 경우에는 수준 20 이상을 예약하세요.

고려 사항 및 제한

Athena에서 ZSTD 압축 수준을 사용할 때 다음 사항을 고려하세요.

- ZSTD compression_level 속성은 Athena 엔진 버전 3에서만 지원됩니다.
- ZSTD compression_level 속성은 ALTER TABLE, CREATE TABLE, CREATE TABLE AS(CTAS) 및 UNLOAD 문에 지원됩니다.
- compression_level 속성은 선택 사항입니다.
- compression_level 속성은 ZSTD 압축에만 지원됩니다.
- 가능한 압축 수준은 1~22입니다.
- 기본 압축 수준은 3입니다.

Athena에서 Apache Hive ZSTD 압축 지원에 대한 자세한 내용은 파일 형식별 Hive 테이블 압축 지원 섹션을 참조하세요. Athena에서 Apache Iceberg ZSTD 압축 지원에 대한 자세한 내용은 파일 형식별 Iceberg 테이블 압축 지원 섹션을 참조하세요.

ZSTD 압축 수준 지정

ALTER TABLE, CREATE TABLE, CREATE TABLE AS 및 UNLOAD 문에 ZSTD 압축 수준을 지정하려면 compression_level 속성을 사용합니다. ZSTD 압축 자체를 지정하려면 문의 구문에서 사용하는 개별 압축 속성을 사용해야 합니다.

ALTER TABLE SET TBLPROPERTIES

ALTER TABLE SET TBLPROPERTIES 문 SET TBLPROPERTIES 절에서 'write.compression' = ' ZSTD' 또는 'parquet.compression' = 'ZSTD'를 사용하여 ZSTD 압축을 지정합니다. 그런 다음 compression_level 속성을 사용하여 1에서 22 사이의 값을 지정합니다(예: 'compression_level' = 5). 압축 수준 속성을 지정하지 않으면 압축 수준은 기본적으로 3으로 설정됩니다.

예

다음 예제에서는 ZSTD 압축 및 ZSTD 압축 수준 4와 함께 Parquet 파일 형식을 사용하도록 테이블 existing_table을 수정합니다. 압축 수준 값은 정수가 아닌 문자열로 입력해야 합니다.

```
ALTER TABLE existing_table
SET TBLPROPERTIES ('parquet.compression' = 'ZSTD', 'compression_level' = 4)
```

CREATE TABLE

CREATE TABLE 문 TBLPROPERTIES 절에서 'write.compression' = 'ZSTD' 또는 'parquet.compression' = 'ZSTD'를 지정한 다음 compression_level =

*compression_level*을 사용하고 1에서 22 사이의 값을 지정합니다. compression_level 속성이 지정되지 않은 경우 기본 압축 수준은 3입니다.

예

다음 예제에서는 ZSTD 압축과 ZSTD 압축 수준 4를 사용하여 Parquet 파일 형식으로 테이블을 생성합니다.

```
CREATE EXTERNAL TABLE new_table (
  `col0` string COMMENT '',
  `col1` string COMMENT ''
)
STORED AS PARQUET
LOCATION 's3://DOC-EXAMPLE-BUCKET/'
TBLPROPERTIES ('write.compression' = 'ZSTD', 'compression_level' = 4)
```

CREATE TABLE AS (CTAS)

CREATE TABLE AS 문 WITH 절에서 write_compression = 'ZSTD' 또는 parquet_compression = 'ZSTD'를 지정한 다음 compression_level = *compression_level*을 사용하고 1에서 22 사이의 값을 지정합니다. compression_level 속성이 지정되지 않은 경우 기본 압축 수준은 3입니다.

예

다음 CTAS 예제에서는 ZSTD 압축과 압축 수준 4를 사용하여 Parquet를 파일 형식으로 지정합니다.

```
CREATE TABLE new_table
WITH ( format = 'PARQUET', write_compression = 'ZSTD', compression_level = 4)
AS SELECT * FROM old_table
```

UNLOAD

UNLOAD 문 WITH 절에서 compression = 'ZSTD'를 지정한 다음 compression_level = *compression_level*을 사용하고 1에서 22 사이의 값을 지정합니다. compression_level 속성이 지정되지 않은 경우 기본 압축 수준은 3입니다.

예

다음 예제에서는 Parquet 파일 형식, ZSTD 압축 및 ZSTD 압축 수준 4를 사용하여 쿼리 결과를 지정된 위치로 언로드합니다.

```
UNLOAD (SELECT * FROM old_table)
TO 's3://DOC-EXAMPLE-BUCKET/'
WITH (format = 'PARQUET', compression = 'ZSTD', compression_level = 4)
```

Athena 리소스 태깅

각 태그는 사용자가 정의하는 키와 값으로 구성됩니다. Athena 리소스에 태그를 지정할 때 사용자 지정 메타데이터를 할당합니다. 태그를 사용하여 AWS 리소스를 용도, 소유자, 환경 등으로 다양하게 분류할 수 있습니다. Athena에서 작업 그룹, 데이터 카탈로그 및 용량 예약과 같은 리소스에 태그를 지정할 수 있습니다. 예를 들어, 계정의 작업 그룹에 대해 작업 그룹 소유자를 추적하거나 해당 용도별로 작업 그룹을 식별할 수 있는 태그를 설정할 수 있습니다. Billing and Cost Management 콘솔에서 비용 할당 태그로 태그를 활성화하면 쿼리 실행과 관련된 비용이 해당 비용 할당 태그와 함께 Cost and Usage Report에 나타납니다. AWS 태그 지정 모범 사례를 사용하여 각 조직의 요구 사항에 맞게 일관된 태그 세트를 생성하는 것이 좋습니다.

Athena 콘솔이나 API 작업을 사용하여 태그 작업을 수행할 수 있습니다.

주제

- 태그 기본 사항
- 태그 제한
- 콘솔의 작업 그룹에서 태그 작업
- 태그 작업 사용
- 태그 기반 IAM 액세스 제어 정책

태그 기본 사항

태그는 Athena 리소스에 할당하는 레이블입니다. 각 태그는 사용자가 정의하는 키와 선택적 값으로 구성됩니다.

태그를 사용하면 다양한 방식으로 AWS 리소스를 분류할 수 있습니다. 예를 들어 계정의 작업 그룹에 대해 각 작업 그룹 소유자나 용도를 추적할 수 있는 태그를 정의할 수 있습니다.

새로운 Athena 작업 그룹 또는 데이터 카탈로그를 생성할 때 태그를 추가할 수 있으며, 해당 작업 그룹 또는 데이터 카탈로그에서 태그를 추가, 편집 또는 제거할 수 있습니다. 태그는 콘솔에서 편집할 수 있습니다. API 작업을 사용하여 태그를 편집하려면 기존 태그를 제거하고 새 태그를 추가합니다. 리소스를 삭제하면 리소스 태그도 삭제됩니다.

Athena는 리소스에 태그를 자동으로 할당하지 않습니다. 태그 키와 값을 편집할 수 있으며 언제든지 리소스에서 태그를 제거할 수 있습니다. 태그의 값을 빈 문자열로 설정할 수 있지만 태그의 값을 Null로 설정할 수는 없습니다. 중복된 태그 키를 동일한 리소스에 추가하지 마세요. 이렇게 할 경우 Athena에서 오류 메시지가 표시됩니다. 기존 태그 키를 사용하여 리소스에 태그를 지정하는 데 TagResource 작업을 사용하는 경우 새 태그 값이 기존 값을 덮어씁니다.

IAM에서는 Amazon Web Services 계정에서 태그를 생성, 편집, 제거하거나 태그 목록을 조회할 수 있는 권한을 가진 사용자를 제어할 수 있습니다. 자세한 내용은 태그 기반 IAM 액세스 제어 정책 단원을 참조하세요.

Amazon Athena 태그 작업의 전체 목록은 Amazon Athena API 참조의 API 작업 이름을 참조하세요.

청구에 태그를 사용할 수 있습니다. 자세한 내용은 AWS Billing and Cost Management 사용 설명서의 결제 목적으로 태그 사용을 참조하세요.

자세한 내용은 태그 제한 단원을 참조하세요.

태그 제한

태그에는 다음 제한 사항이 있습니다.

- Athena에서는 작업 그룹 및 데이터 카탈로그에 태그를 지정할 수 있습니다. 쿼리에는 태그를 지정할 수 없습니다.
- 리소스당 최대 태그 수는 50개입니다. 한도를 유지하려면 사용하지 않는 태그를 검토하여 삭제하세요.
- 각 리소스에 대해 각 태그 키는 고유하며 하나의 값만 가질 수 있습니다. 동일한 리소스에 중복된 태그 키를 동시에 추가하지 마세요. 이렇게 할 경우 Athena에서 오류 메시지가 표시됩니다. 개별 `TagResource` 작업의 기존 태그 키를 사용하여 리소스에 태그를 지정할 경우 새 태그 값이 기존 값을 덮어씁니다.
- 태그 키 길이는 UTF-8 형식의 1-128자의 유니코드 문자입니다.
- 태그 값 길이는 UTF-8 형식의 0-256자의 유니코드 문자입니다.

  태그 추가, 편집, 제거, 목록 조회 등과 같은 태그 지정 작업을 하려면 작업 그룹 리소스의 ARN을 지정해야 합니다.
- Athena에서는 UTF-8 형식의 문자, 숫자, 공백 및 + - = . _ : / @ 문자를 사용할 수 있습니다.
- 태그 키와 값은 대/소문자를 구분합니다.
- 태그 키의 `"aws:"` 접두사는 AWS용으로 예약되어 있습니다. 이 접두사가 지정된 태그 키는 편집하거나 삭제할 수 없습니다. 이 접두사가 지정된 태그는 리소스당 태그 수 제한에 포함되지 않습니다.

- 할당한 태그는 해당 Amazon Web Services 계정에서만 사용할 수 있습니다.

콘솔의 작업 그룹에서 태그 작업

Athena 콘솔을 사용하여 해당 계정의 각 작업 그룹에서 사용 중인 태그를 볼 수 있습니다. 작업 그룹별로만 태그를 볼 수 있습니다. 또한 Athena 콘솔을 사용하여 한 번에 하나의 작업 그룹에서 태그를 적용, 편집, 제거할 수 있습니다.

작성한 태그를 사용하여 작업 그룹을 검색할 수 있습니다.

주제

개별 작업 그룹에 대한 태그 조회

Athena 콘솔에서 개별 작업 그룹의 태그 표시

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

3. 탐색 메뉴에서 작업 그룹(Workgroups)을 선택한 다음 원하는 작업 그룹을 선택합니다.
4. 다음 중 하나를 수행하십시오.
   - 태그 탭을 선택합니다. 태그 목록이 긴 경우 검색 상자를 사용합니다.

- 편집(Edit)을 선택하고 태그(Tags) 섹션까지 아래로 스크롤합니다.

개별 작업 그룹에 대한 태그 추가 및 삭제

작업 그룹 탭에서 개별 작업 그룹에 대한 태그를 직접 관리할 수 있습니다.

> **Note**
>
> 사용자가 콘솔에서 작업 그룹을 생성할 때 태그를 추가하도록 하거나 CreateWorkGroup 작업을 사용할 때 태그를 전달하도록 하려면 TagResource 및 CreateWorkGroup 작업에 대한 IAM 권한을 해당 사용자에게 부여해야 합니다.

작업 그룹을 새로 생성할 때 태그 추가

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 탐색 메뉴에서 작업 그룹(Workgroups)을 클릭합니다.
3. 작업 그룹 생성(Create workgroup)을 선택하고 필요에 따라 값을 입력합니다. 자세한 단계는 작업 그룹 만들기 단원을 참조하세요.
4. 태그(Tags) 섹션에서 키와 값을 지정하여 태그를 하나 이상 추가합니다. 동일한 작업 그룹에 중복된 태그 키를 동시에 추가하지 마세요. 이렇게 할 경우 Athena에서 오류 메시지가 표시됩니다. 자세한 내용은 태그 제한 단원을 참조하세요.
5. 완료했으면 작업 그룹 생성(Create workgroup)을 선택합니다.

기존 작업 그룹에 대해 태그를 추가하거나 편집하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
3. 수정할 작업 그룹을 선택합니다.
4. 다음 중 하나를 수행하십시오.
   - 태그(Tags) 탭을 선택한 후 태그 관리(Manage tags)를 선택합니다.
   - 편집(Edit)을 선택하고 태그(Tags) 섹션까지 아래로 스크롤합니다.
5. 각 태그에 대한 키 및 값을 지정합니다. 자세한 내용은 태그 제한 단원을 참조하십시오.
6. Save(저장)를 선택합니다.

개별 작업 그룹에서 태그를 삭제하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
3. 수정할 작업 그룹을 선택합니다.
4. 다음 중 하나를 수행하십시오.
   - 태그(Tags) 탭을 선택한 후 태그 관리(Manage tags)를 선택합니다.
   - 편집(Edit)을 선택하고 태그(Tags) 섹션까지 아래로 스크롤합니다.
5. 태그 목록에서 삭제하려는 태그에 대해 제거(Remove)를 선택한 후 저장(Save)을 선택합니다.

태그 작업 사용

다음 태그 작업을 사용하여 리소스에서 태그를 추가, 제거 또는 나열할 수 있습니다.

API	CLI	작업 설명
`TagResource`	`tag-resource`	지정된 ARN이 있는 리소스에 태그를 한 개 이상 추가하거나 덮어씁니다.
`UntagResource`	`untag-resource`	지정된 ARN이 있는 리소스에서 태그를 한 개 이상 삭제합니다.
`ListTagsForResource`	`list-tags-for-resource`	지정된 ARN이 있는 리소스의 태그를 한 개 이상 나열합니다.

리소스를 생성할 때 태그 추가

작업 그룹 또는 데이터 카탈로그를 생성할 때 태그를 추가하려면 `CreateWorkGroup` 또는 `CreateDataCatalog` API 작업이나 AWS CLI `create-work-group` 또는 `create-data-catalog` 명령과 함께 `tags` 파라미터를 사용합니다.

API 작업을 사용하여 태그 관리

이 섹션의 예제에서는 태그 API 작업을 사용하여 작업 그룹 및 데이터 카탈로그의 태그를 관리하는 방법을 보여줍니다. 예제는 Java 프로그래밍 언어로 작성되어 있습니다.

Example TagResource

다음 예제에서는 작업 그룹 workgroupA에 두 개의 태그를 추가합니다.

```
List<Tag> tags = new ArrayList<>();
tags.add(new Tag().withKey("tagKey1").withValue("tagValue1"));
tags.add(new Tag().withKey("tagKey2").withValue("tagValue2"));

TagResourceRequest request = new TagResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA")
    .withTags(tags);

client.tagResource(request);
```

다음 예제에서는 데이터 카탈로그 datacatalogA에 두 개의 태그를 추가합니다.

```
List<Tag> tags = new ArrayList<>();
tags.add(new Tag().withKey("tagKey1").withValue("tagValue1"));
tags.add(new Tag().withKey("tagKey2").withValue("tagValue2"));

TagResourceRequest request = new TagResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA")
    .withTags(tags);

client.tagResource(request);
```

> **Note**
>
> 중복된 태그 키를 동일한 리소스에 추가하지 마세요. 이렇게 할 경우 Athena에서 오류 메시지가 표시됩니다. 개별 TagResource 작업의 기존 태그 키를 사용하여 리소스에 태그를 지정할 경우 새 태그 값이 기존 값을 덮어씁니다.

Example UntagResource

다음 예제에서는 작업 그룹 workgroupA에서 tagKey2를 제거합니다.

```
List<String> tagKeys = new ArrayList<>();
tagKeys.add("tagKey2");
```

```
UntagResourceRequest request = new UntagResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA")
    .withTagKeys(tagKeys);

client.untagResource(request);
```

다음 예제에서는 데이터 카탈로그 datacatalogA에서 tagKey2를 제거합니다.

```
List<String> tagKeys = new ArrayList<>();
tagKeys.add("tagKey2");

UntagResourceRequest request = new UntagResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA")
    .withTagKeys(tagKeys);

client.untagResource(request);
```

Example ListTagsForResource

다음 예제에서는 작업 그룹 workgroupA에 대한 태그를 나열합니다.

```
ListTagsForResourceRequest request = new ListTagsForResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA");

ListTagsForResourceResult result = client.listTagsForResource(request);

List<Tag> resultTags = result.getTags();
```

다음 예제에서는 데이터 카탈로그 datacatalogA에 대한 태그를 나열합니다.

```
ListTagsForResourceRequest request = new ListTagsForResourceRequest()
    .withResourceARN("arn:aws:athena:us-east-1:123456789012:datacatalog/datacatalogA");

ListTagsForResourceResult result = client.listTagsForResource(request);

List<Tag> resultTags = result.getTags();
```

AWS CLI를 사용하여 태그 관리

다음 섹션에서는 AWS CLI를 사용하여 데이터 카탈로그에서 태그를 생성하고 관리하는 방법을 보여줍니다.

리소스에 태그 추가: Tag-resource

`tag-resource` 명령은 지정된 리소스에 태그를 한 개 이상 추가합니다.

구문

```
aws athena tag-resource --resource-arn
arn:aws:athena:region:account_id:datacatalog/catalog_name --tags
Key=string,Value=string Key=string,Value=string
```

`--resource-arn` 파라미터는 태그가 추가되는 리소스를 지정합니다. `--tags` 파라미터는 리소스에 태그로 추가할 공백으로 구분된 키-값 페어의 목록을 지정합니다.

Example

다음 예제에서는 데이터 카탈로그 `mydatacatalog`에 태그를 추가합니다.

```
aws athena tag-resource --resource-arn arn:aws:athena:us-
east-1:111122223333:datacatalog/mydatacatalog --tags Key=Color,Value=Orange
 Key=Time,Value=Now
```

결과를 표시하려면 `list-tags-for-resource` 명령을 사용합니다.

`create-data-catalog` 명령 사용 시 태그 추가에 대한 자세한 내용은 [카탈로그 등록: Create-data-catalog] 단원을 참조하세요.

리소스에 대한 태그 나열: List-tags-for-resource

`list-tags-for-resource` 명령은 지정된 리소스에 대한 태그를 나열합니다.

구문

```
aws athena list-tags-for-resource --resource-arn
arn:aws:athena:region:account_id:datacatalog/catalog_name
```

`--resource-arn` 파라미터는 태그를 나열하기 위한 리소스를 지정합니다.

다음 예제에서는 데이터 카탈로그 `mydatacatalog`에 대한 태그를 나열합니다.

```
aws athena list-tags-for-resource --resource-arn arn:aws:athena:us-
east-1:111122223333:datacatalog/mydatacatalog
```

다음 샘플 결과는 JSON 형식입니다.

```
{
    "Tags": [
        {
            "Key": "Time",
            "Value": "Now"
        },
        {
            "Key": "Color",
            "Value": "Orange"
        }
    ]
}
```

리소스에서 태그 제거: Untag-resource

`untag-resource` 명령은 지정된 리소스에서 지정된 태그 키 및 관련된 값을 제거합니다.

구문

```
aws athena untag-resource --resource-arn
arn:aws:athena:region:account_id:datacatalog/catalog_name --tag-keys
key_name [key_name ...]
```

`--resource-arn` 파라미터는 태그를 제거하기 위한 리소스를 지정합니다. `--tag-keys` 파라미터는 공백으로 구분된 키 이름 목록을 사용합니다. `untag-resource` 명령은 지정된 각 키 이름에서 키 및 해당 값을 모두 제거합니다.

다음 예제에서는 `Color` 및 `Time` 키와 해당 값을 `mydatacatalog` 카탈로그 리소스에서 제거합니다.

```
aws athena untag-resource --resource-arn arn:aws:athena:us-
east-1:111122223333:datacatalog/mydatacatalog --tag-keys Color Time
```

태그 기반 IAM 액세스 제어 정책

태그를 사용하면 `Condition` 블록이 포함된 IAM 정책을 작성하여 태그를 기반으로 리소스에 대한 액세스를 제어할 수 있습니다.

작업 그룹에 대한 태그 정책 예제

Example 1. 기본 태깅 정책

다음 IAM 정책은 workgroupA라는 작업 그룹에 대해 쿼리를 실행하고 태그 작업을 할 수 있도록 허용합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
            "Effect": "Allow",
            "Action": [
                "athena:ListWorkGroups",
                "athena:ListEngineVersions",
                "athena:ListDataCatalogs",
                "athena:ListDatabases",
                "athena:GetDatabase",
                "athena:ListTableMetadata",
                "athena:GetTableMetadata"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:GetWorkGroup",
                "athena:TagResource",
                "athena:UntagResource",
                "athena:ListTagsForResource",
                "athena:StartQueryExecution",
                "athena:GetQueryExecution",
                "athena:BatchGetQueryExecution",
                "athena:ListQueryExecutions",
                "athena:StopQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:BatchGetNamedQuery",
                "athena:ListNamedQueries",
                "athena:DeleteNamedQuery",
                "athena:CreatePreparedStatement",
                "athena:GetPreparedStatement",
```

```
                "athena:ListPreparedStatements",
                "athena:UpdatePreparedStatement",
                "athena:DeletePreparedStatement"
            ],
            "Resource": "arn:aws:athena:us-east-1:123456789012:workgroup/workgroupA"
        }
    ]
}
```

Example 2: 태그 키와 태그 값 페어를 기반으로 작업 그룹에서 작업을 거부하는 정책 블록

작업 그룹과 같은 리소스에 연결된 태그를 리소스 태그라고 합니다. 리소스 태그를 사용하면 키-값 페어(예: `stack`, `production`)로 태그가 지정된 작업 그룹에 대해 명시된 작업을 거부하는 다음과 같은 정책 블록을 작성할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "athena:GetWorkGroup",
                "athena:UpdateWorkGroup",
                "athena:DeleteWorkGroup",
                "athena:TagResource",
                "athena:UntagResource",
                "athena:ListTagsForResource",
                "athena:StartQueryExecution",
                "athena:GetQueryExecution",
                "athena:BatchGetQueryExecution",
                "athena:ListQueryExecutions",
                "athena:StopQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:BatchGetNamedQuery",
                "athena:ListNamedQueries",
                "athena:DeleteNamedQuery",
                "athena:CreatePreparedStatement",
                "athena:GetPreparedStatement",
                "athena:ListPreparedStatements",
                "athena:UpdatePreparedStatement",
```

```
                "athena:DeletePreparedStatement"
            ],
            "Resource": "arn:aws:athena:us-east-1:123456789012:workgroup/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stack": "production"
                }
            }
        }
    ]
}
```

Example 3. 지정된 태그에 대한 태그 변경 작업 요청을 제한하는 정책 블록

태그를 변경하는 작업에 파라미터로 전달되는 태그(예: 태그가 있는 TagResource, UntagResource, 또는 CreateWorkGroup)를 요청 태그라고 합니다. 다음 예제 정책 블록에서는 전달된 태그 중 하나에 costcenter 키와 1, 2 또는 3 값이 있는 경우에만 CreateWorkGroup 작업을 허용합니다.

> **Note**
>
> IAM 역할이 CreateWorkGroup 작업의 일환으로 태그를 전달할 수 있도록 하려면 TagResource 및 CreateWorkGroup 작업에 대한 권한을 해당 역할에 부여해야 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateWorkGroup",
                "athena:TagResource"
            ],
            "Resource": "arn:aws:athena:us-east-1:123456789012:workgroup/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/costcenter": [
                        "1",
                        "2",
                        "3"
                    ]
```

```
                }
            }
        }
    ]
}
```

데이터 카탈로그에 대한 태그 정책 예제

Example 1. 기본 태깅 정책

다음 IAM 정책을 사용하면 datacatalogA라는 데이터 카탈로그에 대한 태그와 상호 작용할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena:ListWorkGroups",
                "athena:ListEngineVersions",
                "athena:ListDataCatalogs",
                "athena:ListDatabases",
                "athena:GetDatabase",
                "athena:ListTableMetadata",
                "athena:GetTableMetadata"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:GetWorkGroup",
                "athena:TagResource",
                "athena:UntagResource",
                "athena:ListTagsForResource",
                "athena:StartQueryExecution",
                "athena:GetQueryExecution",
                "athena:BatchGetQueryExecution",
                "athena:ListQueryExecutions",
                "athena:StopQueryExecution",
                "athena:GetQueryResults",
                "athena:GetQueryResultsStream",
```

```
                "athena:CreateNamedQuery",
                "athena:GetNamedQuery",
                "athena:BatchGetNamedQuery",
                "athena:ListNamedQueries",
                "athena:DeleteNamedQuery"
            ],
            "Resource": [
                "arn:aws:athena:us-east-1:123456789012:workgroup/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:CreateDataCatalog",
                "athena:GetDataCatalog",
                "athena:UpdateDataCatalog",
                "athena:DeleteDataCatalog",
                "athena:ListDatabases",
                "athena:GetDatabase",
                "athena:ListTableMetadata",
                "athena:GetTableMetadata",
                "athena:TagResource",
                "athena:UntagResource",
                "athena:ListTagsForResource"
            ],
            "Resource": "arn:aws:athena:us-
east-1:123456789012:datacatalog/datacatalogA"
        }
    ]
}
```

Example 2: 태그 키와 태그 값 페어를 기반으로 데이터 카탈로그에서 작업을 거부하는 정책 블록

리소스 태그를 사용하여 특정 태그 키-값 페이로 태그가 지정된 데이터 카탈로그에서 특정 작업을 거부하는 정책 블록을 작성할 수 있습니다. 다음 예제 정책은 태그 키-값 페어(`stack`, `production`)가 있는 데이터 카탈로그에 대한 작업을 거부합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
```

```
                "athena:CreateDataCatalog",
                "athena:GetDataCatalog",
                "athena:UpdateDataCatalog",
                "athena:DeleteDataCatalog",
                "athena:GetDatabase",
                "athena:ListDatabases",
                "athena:GetTableMetadata",
                "athena:ListTableMetadata",
                "athena:StartQueryExecution",
                "athena:TagResource",
                "athena:UntagResource",
                "athena:ListTagsForResource"
            ],
            "Resource": "arn:aws:athena:us-east-1:123456789012:datacatalog/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stack": "production"
                }
            }
        }
    ]
}
```

Example 3. 지정된 태그에 대한 태그 변경 작업 요청을 제한하는 정책 블록

태그를 변경하는 작업에 파라미터로 전달되는 태그(예: 태그가 있는 `TagResource`, `UntagResource`, 또는 `CreateDataCatalog`)를 요청 태그라고 합니다. 다음 예제 정책 블록에서는 전달된 태그 중 하나에 `costcenter` 키와 1, 2 또는 3 값이 있는 경우에만 `CreateDataCatalog` 작업을 허용합니다.

> **Note**
>
> IAM 역할이 `CreateDataCatalog` 작업의 일환으로 태그를 전달할 수 있도록 하려면 `TagResource` 및 `CreateDataCatalog` 작업에 대한 권한을 해당 역할에 부여해야 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
```

```
                "athena:CreateDataCatalog",
                "athena:TagResource"
            ],
            "Resource": "arn:aws:athena:us-east-1:123456789012:datacatalog/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/costcenter": [
                        "1",
                        "2",
                        "3"
                    ]
                }
            }
        }
    ]
}
```

Service Quotas

> **Note**
>
> Service Quotas 콘솔은 Amazon Athena 할당량에 대한 정보를 제공합니다. Service Quotas 콘솔을 사용하여 조정 가능한 할당량에 대한 할당량 증가를 요청할 수도 있습니다. AWS Glue 관련 스키마 제한은 AWS Glue 엔드포인트 및 할당량 페이지를 참조하세요. AWS 서비스 할당량에 대한 일반적인 정보는 AWS 일반 참조의 AWS 서비스 할당량을 참조하세요.

쿼리

계정에는 Amazon Athena에 대해 다음과 같은 쿼리 관련 할당량이 있습니다. 자세한 내용은 AWS 일반 참조의 Amazon Athena 엔드포인트 및 할당량을 참조하세요.

- 활성 DDL 쿼리(Active DDL queries) - 활성 DDL 쿼리 수입니다. DDL 쿼리에는 `CREATE TABLE` 및 `ALTER TABLE ADD PARTITION` 쿼리가 포함됩니다.
- DDL 쿼리 제한 시간(DDL query timeout) - DDL 쿼리가 취소되기 전에 실행할 수 있는 최대 시간(분)입니다.
- 활성 DML 쿼리(Active DML queries) - 활성 DML 쿼리 수입니다. DML 쿼리에는 `SELECT`, `CREATE TABLE AS`(CTAS) 및 `INSERT INTO` 쿼리가 포함됩니다. 구체적인 할당량은 AWS 리전에 따라 다릅니다.

- DDL 쿼리 제한 시간(DML query timeout) - DML 쿼리가 취소되기 전에 실행할 수 있는 최대 시간(분)입니다. 이 제한 시간을 최대 240분까지 늘리도록 요청할 수 있습니다.

할당량 증가를 요청하려면 Athena Service Quotas 콘솔을 사용합니다.

Athena는 전체 서비스 부하와 수신 요청 수에 따라 리소스를 할당하여 쿼리를 처리합니다. 쿼리는 실행되기 전에 일시적으로 대기열에 머무를 수 있습니다. 비동기 프로세스는 대기열에서 쿼리를 선택하고 리소스가 가용 상태가 되는 즉시 물리적 리소스에서 쿼리를 실행합니다. 단, 계정 구성이 이를 허용해야 합니다.

DML 또는 DDL 쿼리 할당량은 실행 중인 쿼리와 대기 중인 쿼리를 모두 포함합니다. 예를 들어 DML 쿼리 할당량이 25이고 실행 중인 쿼리와 대기 중인 쿼리의 합계가 26인 경우 쿼리 26은 TooManyRequestsException 오류를 발생시킵니다.

> **Note**
>
> Athena에서 실행하는 쿼리의 동시성을 직접 제어하려는 경우 용량 예약을 사용할 수 있습니다. 자세한 내용은 쿼리 처리 용량 관리 단원을 참조하십시오.

쿼리 문자열 길이

허용되는 최대 쿼리 문자열 길이는 262144바이트이며, 문자열은 UTF-8로 인코딩됩니다. 이 할당량은 조정 가능한 할당량이 아닙니다. 그러나 긴 쿼리를 여러 개의 작은 쿼리로 분할하여 이러한 제한을 해결할 수 있습니다. 자세한 내용은 AWS 지식 센터에서 Athena의 최대 쿼리 문자열 길이를 늘리려면 어떻게 해야 합니까?를 참조하세요.

작업 그룹

Athena 작업 그룹으로 작업할 때는 다음 사항을 기억하세요.

- Athena 서비스 할당량은 계정의 모든 작업 그룹 간에 공유됩니다.
- 한 계정에서 리전별로 생성할 수 있는 최대 작업 그룹 수는 1,000개입니다.
- 작업 그룹에서 준비된 문의 최대 개수는 1000입니다.
- 작업 그룹당 최대 태그 수는 50개입니다. 자세한 내용은 태그 제한 단원을 참조하세요.

데이터베이스, 테이블, 파티션

- Athena와 함께 AWS Glue Data Catalog를 사용할 경우 테이블, 데이터베이스, 파티션에 대한 서비스 할당량 관련 사항은 AWS Glue 엔드포인트 및 할당량을 참조하세요(예: 계정당 최대 데이터베이스 또는 테이블 수).
  - Athena는 1천만 개의 파티션이 있는 AWS Glue 테이블에 대한 쿼리를 지원하지만, 단일 스캔으로 1백만 개 이상의 파티션을 읽을 수는 없습니다.
- AWS Glue Data Catalog를 사용하지 않는 경우 테이블당 파티션 수는 20,000개입니다. 할당량 증가를 요청할 수 있습니다.

Amazon S3 버킷

Amazon S3 버킷으로 작업할 때는 다음 사항을 기억하세요.

- Amazon S3에는 계정당 버킷 100개의 기본 서비스 할당량이 있습니다.
- Athena에는 결과를 기록하기 위해 별도의 버킷이 필요합니다.
- AWS 계정당 Amazon S3 버킷 최대 1,000개의 할당량 증가를 요청할 수 있습니다.

계정당 API 호출 할당량

Athena API에는 계정당(쿼리당이 아님) API에 호출할 수 있는 수에 다음과 같은 기본 할당량이 있습니다.

API 이름	초당 기본 호출 수	버스트 용량
`BatchGetNamedQuery` , `ListNamedQueries` , `ListQueryExecutions`	5	최대 10개
`CreateNamedQuery` , `DeleteNamedQuery` , `GetNamedQuery`	5	최대 20
`BatchGetQueryExecution`	20	최대 40
`StartQueryExecution` , `StopQueryExecution`	20	최대 80

API 이름	초당 기본 호출 수	버스트 용량
`GetQueryExecution` , `GetQueryResults`	100	최대 200

예를 들어 `StartQueryExecution`의 경우 초당 최대 20회 호출할 수 있습니다. 또한 이 API가 4초 동안 호출되지 않는 경우 계정이 최대 80회 호출의 버스트 용량을 누적합니다. 이 경우 애플리케이션은 버스트 모드에서 이 API를 최대 80회 호출합니다.

이러한 API를 사용하고 초당 호출 수의 기본 할당량 또는 계정의 버스트 용량을 초과하는 경우, Athena API는 ""ClientError: An error occurred (ThrottlingException) when calling the <API_name> operation: Rate exceeded"와 유사한 오류를 생성합니다. 초당 호출 수 또는 이 계정의 API에 대한 버스트 용량을 줄입니다.

계정별 API 호출에 대한 Athena 할당량은 Athena Service Quotas 콘솔에서 변경할 수 없습니다. Athena API 호출에 대한 할당량 증가를 요청하려면 AWS Support 서비스 한도 증가 페이지로 이동하여 양식을 작성해 제출하세요.

Athena 엔진 버전 관리

Athena는 수시로 향상된 성능, 기능 및 코드 수정을 제공하기 위해 새로운 엔진 버전을 릴리스합니다. 새 엔진 버전을 이용할 수 있게 되면 Athena 콘솔 및 AWS Health Dashboard를 통해 알림을 제공합니다. AWS Health Dashboard는 사용자의 AWS 서비스 또는 계정에 영향을 미칠 수 있는 이벤트에 대해 알립니다. AWS Health Dashboard에 대한 자세한 내용은 AWS Health Dashboard 시작하기를 참조하세요.

엔진 버전 관리는 작업 그룹별로 구성됩니다. 작업 그룹을 사용하여 쿼리에서 사용하는 쿼리 엔진을 제어하고 Athena에서 작업 그룹을 자동으로 업그레이드할지 여부를 제어할 수 있습니다. 사용 중인 쿼리 엔진은 쿼리 편집기, 작업 그룹 세부 정보 페이지, Athena API에서 표시됩니다.

- 기본적으로 작업 그룹은 자동 업그레이드되도록 구성됩니다. 작업 그룹이 자동 업그레이드 설정된 경우 Athena에서 비호환성을 발견하지 않는 한 작업 그룹을 업그레이드합니다.
- 지정된 버전을 사용하도록 작업 그룹을 구성하면 Athena는 해당 작업 그룹의 버전을 변경하지 않습니다.

두 경우 모두 Athena는 버전을 더 이상 사용할 수 없을 때 작업 그룹을 업그레이드합니다. Athena는 AWS Health Dashboard에서 엔진 버전이 더 이상 제공되지 않는 시기를 알립니다. AWS Health

Dashboard는 사용자의 AWS 서비스 또는 계정에 영향을 미칠 수 있는 이벤트에 대해 알립니다. AWS Health Dashboard에 대한 자세한 내용은 AWS Health Dashboard 시작하기를 참조하세요.

새 엔진 버전을 사용하기 시작할 때 비호환성으로 인해 쿼리의 일부 하위 집합이 중단될 수 있습니다. 새로운 Athena 버전이 출시되면 호환성에 영향을 미치는 변경 사항이 발표됩니다. 작업 그룹을 사용해 새 엔진을 사용하는 테스트 작업 그룹을 만들거나 기존 작업 그룹을 테스트로 업그레이드하여 업그레이드 전에 쿼리를 테스트해야 합니다. 자세한 내용은 엔진 버전 업그레이드 전 쿼리 테스트 단원을 참조하십시오.

주제

- Athena 엔진 버전 변경
- Athena 엔진 버전 참조

Athena 엔진 버전 변경

Athena는 수시로 향상된 성능, 기능 및 코드 수정을 제공하기 위해 새로운 엔진 버전을 릴리스합니다. 새 엔진 버전을 이용할 수 있게 되면 Athena 콘솔에서 알림을 제공합니다. 사용자는 Athena가 업그레이드 시기를 결정하도록 선택하거나 작업 그룹별로 Athena 엔진 버전을 수동으로 지정할 수 있습니다.

주제

- 작업 그룹의 쿼리 엔진 버전 찾기
- Athena 콘솔에서 엔진 버전 변경
- AWS CLI를 사용하여 엔진 버전 변경하기
- 작업 그룹을 만들 때 엔진 버전 지정
- 엔진 버전 업그레이드 전 쿼리 테스트
- 실패한 쿼리 문제 해결

작업 그룹의 쿼리 엔진 버전 찾기

작업 그룹페이지를 사용하여 작업 그룹의 현재 엔진 버전을 찾을 수 있습니다.

작업 그룹의 현재 엔진 버전을 찾으려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

3. Athena 콘솔 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
4. 작업 그룹(Workgroups) 페이지에서 원하는 작업 그룹을 찾습니다. 작업 그룹의 쿼리 엔진 버전(Query engine version) 열에는 쿼리 엔진 버전이 표시됩니다.

Athena 콘솔에서 엔진 버전 변경

사용자는 새 엔진 버전을 이용할 수 있을 때 Athena가 작업 그룹을 업그레이드할 시기를 결정하도록 하거나 작업 그룹이 사용할 Athena 엔진 버전을 수동으로 지정할 수 있습니다. 현재 하나의 버전만 사용할 수 있는 경우 다른 버전을 수동으로 지정할 수 없습니다.

> **Note**
>
> 작업 그룹의 엔진 버전을 변경하려면 작업 그룹에 대해 `athena:ListEngineVersions` 작업을 수행할 권한이 있어야 합니다. IAM 정책 예제는 작업 그룹 정책의 예 단원을 참조하세요.

Athena가 작업 그룹을 업그레이드할 시기를 결정하도록 하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.
3. 콘솔 탐색 창에서 작업 그룹(Workgoups)을 선택합니다.
4. 작업 그룹 목록에서 구성할 작업 그룹에 대한 링크를 선택합니다.
5. 편집을 선택합니다.

6. 쿼리 엔진 버전(Query engine version) 섹션에서 쿼리 엔진 업데이트(Update query engine)에 대해 자동(Automatic)을 선택하여 Athena가 작업 그룹을 업그레이드할 시기를 선택할 수 있도록 합니다. 이것이 기본 설정입니다.
7. Save changes(변경 사항 저장)를 선택합니다.

   작업 그룹 목록에서 작업 그룹의 Query engine update status(쿼리 엔진 업데이트 상태)가 Automatic(자동)으로 표시됩니다.

엔진 버전을 수동으로 선택하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.
3. 콘솔 탐색 창에서 작업 그룹(Workgoups)을 선택합니다.
4. 작업 그룹 목록에서 구성할 작업 그룹에 대한 링크를 선택합니다.
5. 편집을 선택합니다.
6. 쿼리 엔진 버전(Query engine version) 섹션에서 쿼리 엔진 업데이트(Update query engine)에 대해 수동(Manual)을 선택하여 수동으로 엔진 버전을 선택합니다.
7. 쿼리 엔진 버전(Query engine version) 옵션을 사용하여 작업 그룹에서 사용할 엔진 버전을 선택합니다. 다른 엔진 버전을 사용할 수 없는 경우 다른 엔진 버전을 지정할 수 없습니다.
8. Save changes(변경 사항 저장)를 선택합니다.

   작업 그룹 목록에서 작업 그룹의 Query engine update status(쿼리 엔진 업데이트 상태)가 Manual(수동)로 표시됩니다.

AWS CLI를 사용하여 엔진 버전 변경하기

AWS CLI를 사용하여 엔진 버전을 변경하려면 다음 예의 구문을 사용하세요.

```
aws athena update-work-group --work-group workgroup-name --configuration-updates
 EngineVersion={SelectedEngineVersion='Athena engine version 3'}
```

작업 그룹을 만들 때 엔진 버전 지정

작업 그룹을 만들 때 작업 그룹이 사용할 엔진 버전을 지정하거나 작업 그룹을 업그레이드할 시기를 Athena가 결정하도록 할 수 있습니다. 새 엔진 버전을 사용할 수 있는 경우 다른 작업 그룹을 업그레이드하기 전에 새 엔진을 테스트할 작업 그룹을 생성하는 것이 가장 좋습니다. 작업 그룹의 엔진 버전을

지정하려면 작업 그룹에 대해 `athena:ListEngineVersions` 권한이 있어야 합니다. IAM 정책 예제는 작업 그룹 정책의 예 단원을 참조하세요.

작업그룹을 만들 때 엔진 버전을 지정하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.
3. 콘솔 탐색 창에서 작업 그룹(Workgoups)을 선택합니다.
4. 작업 그룹 페이지에서 작업 그룹 생성을 선택합니다.
5. 작업 그룹 생성(Create workgroup) 페이지의 쿼리 엔진 버전(Query engine version) 섹션에서 다음 중 하나를 수행합니다.
   - 자동(Automatic)을 선택하면 Athena가 작업 그룹을 업그레이드할 시기를 선택할 수 있습니다. 이것이 기본 설정입니다.
   - 사용 가능한 경우 다른 엔진 버전을 수동으로 선택하려면 수동(Manual)을 선택합니다.
6. 필요에 따라 기타 필드에 정보를 입력합니다. 기타 필드에 대한 자세한 내용은 작업 그룹 만들기 단원을 참조하세요.
7. 작업 그룹 생성을 선택합니다.

엔진 버전 업그레이드 전 쿼리 테스트

작업 그룹이 새 엔진 버전으로 업그레이드되면 비호환성으로 인해 일부 쿼리가 중단될 수 있습니다. 엔진 버전 업그레이드가 원활하게 진행되는지 확인하기 위해 미리 쿼리를 테스트할 수 있습니다.

엔진 버전 업그레이드 전에 쿼리를 테스트하려면

1. 사용 중인 작업 그룹의 엔진 버전을 확인합니다. 작업 그룹(Workgroups) 페이지의 작업 그룹에 대한 쿼리 엔진 버전(Query engine version) 열에 사용 중인 엔진 버전이 표시됩니다. 자세한 내용은 작업 그룹의 쿼리 엔진 버전 찾기 단원을 참조하십시오.
2. 새 엔진 버전을 사용하는 테스트 작업 그룹을 만듭니다. 자세한 내용은 작업 그룹을 만들 때 엔진 버전 지정 단원을 참조하세요.
3. 새 작업 그룹을 사용하여 테스트하려는 쿼리를 실행합니다.
4. 쿼리가 실패하면 Athena 엔진 버전 참조를 이용하여 쿼리에 영향을 줄 수 있는 주요 변경 사항을 확인합니다. 일부 변경 사항의 경우 쿼리 구문의 업데이트가 필요할 수 있습니다.

5. 쿼리가 계속 실패하면 AWS Support에 지원을 문의하세요. AWS Management Console에서 지원(Support), 지원 센터(Support Center)를 선택하거나 Amazon Athena 태그를 사용하여 AWS re:Post에서 질문하세요.

실패한 쿼리 문제 해결

엔진 버전 업그레이드 후 쿼리가 실패하는 경우 Athena 엔진 버전 참조를 이용하여 쿼리의 구문에 영향을 줄 수 있는 변경 사항을 포함해 주요 변경 사항을 확인합니다.

쿼리가 계속 실패하면 AWS Support에 지원을 문의하세요. AWS Management Console에서 지원(Support), 지원 센터(Support Center)를 선택하거나 Amazon Athena 태그를 사용하여 AWS re:Post에서 질문하세요.

Athena 엔진 버전 참조

이 단원에서는 Athena 쿼리 엔진의 변경 사항을 설명합니다.

주제

- Athena 엔진 버전 3
- Athena 엔진 버전 2

Athena 엔진 버전 3

엔진 버전 3의 경우 Athena에서는 Athena 엔진 내에서 통합되고 조정되는 커뮤니티 개선 사항에 더 빠르게 액세스할 수 있도록 Trino 및 Presto 프로젝트로 동시성을 개선하는 지속적인 통합 접근 방식을 오픈 소스 소프트웨어 관리에 도입합니다.

이 Athena 엔진 버전 3 릴리스에서는 Athena 엔진 버전 2의 모든 기능을 지원합니다. 이 문서에서는 Athena 엔진 버전 2와 Athena 엔진 버전 3의 주요 차이점을 중심으로 설명합니다. 자세한 내용은 AWS 빅 데이터 블로그의 Upgrade to Athena engine version 3 to increase query performance and access more analytics features 문서를 참조하세요.

- 시작하기
- 개선 사항 및 새로운 기능
  - 추가한 기능
  - 추가된 함수
  - 성능 개선

시작하기

시작하려면 Athena 엔진 버전 3을 사용하는 새 Athena 작업 그룹을 생성하거나 버전 3을 사용하도록 기존 작업 그룹을 구성하세요. Athena 작업 그룹에서는 쿼리를 제출하는 기능의 중단 없이 엔진 버전 2에서 엔진 버전 3으로 업그레이드할 수 있습니다.

자세한 내용은 Athena 엔진 버전 변경하기 섹션을 참조하세요.

개선 사항 및 새로운 기능

나열된 기능과 업데이트에는 Athena 자체 및 오픈 소스 Trino에서 통합된 기능의 개선 사항이 포함되어 있습니다. SQL 쿼리 연산자 및 함수의 전체 목록은 Trino 설명서를 참조하세요.

추가한 기능

Apache Spark 버킷팅 알고리즘 지원

Athena에서는 Spark 해시 알고리즘으로 생성한 버킷을 읽을 수 있습니다. 데이터를 원래 Spark 해시 알고리즘으로 작성했다고 지정하려면 `CREATE TABLE` 명령문의 `TBLPROPERTIES` 절에 `('bucketing_format'='spark')`를 추가하세요. 이 속성을 지정하지 않으면 Hive 해시 알고리즘이 사용됩니다.

```
CREATE EXTERNAL TABLE `spark_bucket_table`(
  `id` int,
  `name` string
  )
CLUSTERED BY (`name`)
INTO 8 BUCKETS
STORED AS PARQUET
```

```
LOCATION
  's3://DOC-EXAMPLE-BUCKET/to/bucketed/table/'
TBLPROPERTIES ('bucketing_format'='spark')
```

추가된 함수

이 섹션의 함수는 Athena 엔진 버전 3의 새로운 함수입니다.

집계 함수

listagg(x, separator) – 구분 기호 문자열로 구분된 연결된 입력 값을 반환합니다.

```
SELECT listagg(value, ',') WITHIN GROUP (ORDER BY value) csv_value
FROM (VALUES 'a', 'c', 'b') t(value);
```

배열 함수

contains_sequence(x, seq) – 배열 x에 모든 배열 시퀀스가 순차적 하위 집합(동일한 연속 순서의 모든 값)으로 포함되면 true를 반환합니다.

```
SELECT contains_sequence(ARRAY [1,2,3,4,5,6], ARRAY[1,2]);
```

이진 함수

murmur3(binary) – 이진수의 128비트 MurmurHash3 해시를 계산합니다.

```
SELECT murmur3(from_base64('aaaaaa'));
```

변환 함수

format_number(number) – 단위 기호를 사용하는 형식이 지정된 문자열을 반환합니다.

```
SELECT format_number(123456); -- '123K'
```

```
SELECT format_number(1000000); -- '1M'
```

날짜 및 시간 함수

timezone_hour(timestamp) – 타임스탬프 중에서 시간대 오프셋의 시간을 반환합니다.

```
SELECT EXTRACT(TIMEZONE_HOUR FROM TIMESTAMP '2020-05-10 12:34:56 +08:35');
```

timezone_minute(timestamp) – 타임스탬프 중에서 시간대 오프셋의 분을 반환합니다.

```
SELECT EXTRACT(TIMEZONE_MINUTE FROM TIMESTAMP '2020-05-10 12:34:56 +08:35');
```

지리 공간 함수

to_encoded_polyline(Geometry) – 라인스트링 또는 다중점을 폴리라인에 인코딩합니다.

```
SELECT to_encoded_polyline(ST_GeometryFromText(
   'LINESTRING (-120.2 38.5, -120.95 40.7, -126.453 43.252)'));
```

from_encoded_polyline(varchar) – 폴리라인을 라인스트링에 디코딩합니다.

```
SELECT ST_AsText(from_encoded_polyline('_p~iF~ps|U_ulLnnqC_mqNvxq`@'));
```

to_geojson_geometry(SphericalGeography) – GeoJSON 형식으로 지정된 구형 지리를 반환합니다.

```
SELECT to_geojson_geometry(to_spherical_geography(ST_GeometryFromText(
   'LINESTRING (0 0, 1 2, 3 4)')));
```

from_geojson_geometry(varchar) – GeoJSON 표현에서 지오메트리가 아닌 키/값을 제거한 구형 지리 유형 객체를 반환합니다. `Feature`와 `FeatureCollection`은 지원되지 않습니다.

```
SELECT
 from_geojson_geometry(to_geojson_geometry(to_spherical_geography(ST_GeometryFromText(
   'LINESTRING (0 0, 1 2, 3 4)'))));
```

geometry_nearest_points(Geometry, Geometry) – 각 지오메트리에서 서로 가장 가까운 점을 반환합니다. 지오메트리가 하나라도 비어 있으면 null을 반환합니다. 그렇지 않으면 지오메트리에 두 점의 최소 거리가 있는 두 `Point` 객체의 행을 반환합니다. 첫 번째 점은 첫 번째 Geometry 인수, 두 번째 점은 두 번째 Geometry 인수에서 나옵니다. 최소 거리가 동일한 쌍이 여러 개 있는 경우 한 쌍을 임의로 선택합니다.

```
SELECT geometry_nearest_points(ST_GeometryFromText(
   'LINESTRING (50 100, 50 200)'), ST_GeometryFromText(
   'LINESTRING (10 10, 20 20)'));
```

다이제스트 함수 설정

make_set_digest(x) – x의 모든 입력 값을 세트 setdigest에 작성합니다.

```
SELECT make_set_digest(value) FROM (VALUES 1, 2, 3) T(value);
```

문자열 함수

soundex(char) – char의 음성 표현이 포함된 문자열을 반환합니다.

```
SELECT name
FROM nation
WHERE SOUNDEX(name) = SOUNDEX('CHYNA'); -- CHINA
```

concat_ws(string0, string1, ..., stringN) – `string0`을 구분 기호로 사용하는 `string1, string2, ..., stringN`의 연결을 반환합니다. `string0`이 null이면 반환 값도 null입니다. 구분 기호 뒤의 인수에서 제공되는 null 값은 건너뜁니다.

```
SELECT concat_ws(',', 'def', 'pqr', 'mno');
```

윈도 함수

GROUPS – 그룹 기반 창 프레임에 대한 지원을 추가합니다.

```
SELECT array_agg(a) OVER(
   ORDER BY a ASC NULLS FIRST GROUPS BETWEEN 1 PRECEDING AND 2 FOLLOWING)
FROM (VALUES 3, 3, 3, 2, 2, 1, null, null) T(a);
```

성능 개선

Athena 엔진 버전 3의 성능 개선 사항에는 다음이 포함됩니다.

- 더 빠른 AWS Glue 테이블 메타데이터 검색 – 여러 테이블을 포함하는 쿼리에서 쿼리 계획 시간이 단축됩니다.
- RIGHT JOIN에 대한 동적 필터링 – 이제 다음 예와 같이 등가 조인 조건이 있는 오른쪽 조인에 대해 동적 필터링이 활성화됩니다.

```
SELECT *
FROM lineitem RIGHT JOIN tpch.tiny.supplier
```

```
ON lineitem.suppkey = supplier.suppkey
WHERE supplier.name = 'abc';
```

- 대량으로 준비된 명령문 – 명령문이 대량으로 준비되도록 기본 HTTP 요청/응답 헤더 크기를 2MB로 늘렸습니다.
- approx_percentile() - 이제 `approx_percentile` 함수는 분산에서 대략적인 분위수 값을 검색하기 위해 `qdigest` 대신 `tdigest`를 사용합니다. 그 결과 성능은 향상되고 메모리 사용량이 줄어듭니다. 이 변경으로 인해 함수는 Athena 엔진 버전 2에서 반환했던 것과 다른 결과를 반환합니다. 자세한 내용은 approx_percentile 함수는 다른 결과를 반환합니다. 단원을 참조하십시오.

신뢰성 향상 기능

Athena 엔진 버전 3의 일반 엔진 메모리 사용량 및 추적이 개선되었습니다. 대규모 쿼리가 노드 충돌로 인한 실패에 덜 취약합니다.

쿼리 구문 향상 기능

INTERSECT ALL – `INTERSECT ALL`에 대한 지원을 추가했습니다.

```
SELECT * FROM (VALUES 1, 2, 3, 4) INTERSECT ALL SELECT * FROM (VALUES 3, 4);
```

EXCEPT ALL – `EXCEPT ALL`에 대한 지원을 추가했습니다.

```
SELECT * FROM (VALUES 1, 2, 3, 4) EXCEPT ALL SELECT * FROM (VALUES 3, 4);
```

RANGE PRECEDING – 창 함수의 `RANGE PRECEDING`에 대한 지원을 추가했습니다.

```
SELECT sum(x) over (order by x range 1 preceding)
FROM (values (1), (1), (2), (2)) t(x);
```

MATCH_RECOGNIZE – 다음 예와 같이 행 패턴 일치에 대한 지원을 추가했습니다.

```
SELECT m.id AS row_id, m.match, m.val, m.label
FROM (VALUES(1, 90),(2, 80),(3, 70),(4, 70)) t(id, value)
MATCH_RECOGNIZE (
        ORDER BY id
        MEASURES match_number() AS match,
        RUNNING LAST(value) AS val,
        classifier() AS label
```

```
        ALL ROWS PER MATCH
        AFTER MATCH SKIP PAST LAST ROW
        PATTERN (() | A) DEFINE A AS true
) AS m;
```

데이터 형식 및 데이터 유형 향상 기능

Athena 엔진 버전 3에서는 다음과 같은 데이터 형식과 데이터 유형이 향상되었습니다.

- LZ4 및 ZSTD – LZ4 및 ZSTD 압축 Parquet 데이터 읽기에 대한 지원을 추가했습니다. ZSTD 압축 ORC 데이터 쓰기에 대한 지원을 추가했습니다.
- 심볼릭 링크 기반 테이블 – Avro 파일의 심볼릭 링크 기반 테이블 생성에 대한 지원을 추가했습니다. 예를 들면 다음과 같습니다.

  ```
  CREATE TABLE test_avro_symlink
  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
  ...
  INPUTFORMAT 'org.apache.hadoop.hive.ql.io.SymlinkTextInputFormat'
  ```

- SphericalGeography – SphericalGeography 유형에서는 지리적 좌표 (측지 좌표, 위도/경도 또는 경도/위도라고도 함)에 표시되는 공간 형상에 대한 기본 지원을 제공합니다. 지리적 좌표는 각도 단위(도)로 표현된 구면 좌표입니다.

  `to_spherical_geography` 함수에서는 다음 예와 같이 기하(평면) 좌표 중에서 지리(구면) 좌표를 반환합니다.

  ```
  SELECT to_spherical_geography(ST_GeometryFromText(
     'LINESTRING (-40.2 28.9, -40.2 31.9, -37.2 31.9)'));
  ```

호환성에 영향을 미치는 변경

Athena 엔진 버전 2에서 Athena 엔진 버전 3으로 마이그레이션하는 경우 특정 변경 사항이 테이블 스키마, 구문 또는 데이터 유형 사용량에 영향을 미칠 수 있습니다. 이 섹션에서는 관련 오류 메시지를 나열하고 권장 해결 방법을 제공합니다.

쿼리 구문 변경 사항

IGNORE NULLS는 value가 아닌 창 함수와 함께 사용할 수 없음

오류 메시지: Cannot specify null treatment clause for `bool_or` function.

원인: 이제 value 함수 `first_value`, `last_value`, `nth_value`, `lead`, `lag`에서만 `IGNORE NULLS`를 사용할 수 있습니다. ANSI SQL 사양을 준수하도록 변경되었습니다.

권장 솔루션: value가 아닌 창 함수에서 쿼리 문자열에 있는 `IGNORE NULLS`를 제거합니다.

CONCAT 함수에는 두 개 이상의 인수가 있어야 함

오류 메시지: INVALID_FUNCTION_ARGUMENT: There must be two or more concatenation arguments(INVALID_FUNCTION_ARGUMENT: 두 개 이상의 연결 인수가 있어야 합니다)

원인: 이전에는 `CONCAT` 문자열 함수에서 단일 인수를 허용했습니다. Athena 엔진 버전 3에서는 `CONCAT` 함수에 최소 두 개의 인수가 필요합니다.

권장 솔루션: `CONCAT(str)` 항목을 `CONCAT(str, '')`로 변경합니다.

Athena 엔진 버전 3에서는 함수 인수가 127개를 초과할 수 없습니다. 자세한 내용은 함수 호출에 사용하는 인수가 너무 많음 단원을 참조하십시오.

approx_percentile 함수는 다른 결과를 반환합니다.

`approx_percentile` 함수는 Athena 엔진 버전 3에서 Athena 엔진 버전 2와는 다른 결과를 반환합니다.

오류 메시지: 없습니다.

원인: `approx_percentile` 함수는 버전 변경의 영향을 받습니다.

> **Important**
>
> `approx_percentile` 함수의 출력은 근사치이고 근사치는 버전마다 변경될 수 있으므로 중요한 애플리케이션에서는 `approx_percentile` 함수에 의존해서는 안 됩니다.

권장 솔루션: Athena 엔진 버전 2에서 `approx_percentile`의 동작에 대한 근사치를 구하려는 경우 Athena 엔진 버전 3에서 다른 함수 세트를 사용할 수 있습니다. 예를 들어 Athena 엔진 버전 2에서 다음 쿼리를 가정합니다.

```
SELECT approx_percentile(somecol, 2E-1)
```

Athena 엔진 버전 3에서 동일한 출력의 근사치를 구하려면 다음 예제와 같이 `qdigest_agg` 및 `value_at_quantile` 함수를 사용할 수 있습니다. 단, 이 해결 방법을 사용하더라도 동일한 동작이 보장되지는 않습니다.

```
SELECT value_at_quantile(qdigest_agg(somecol, 1), 2E-1)
```

지리 공간 함수에서 varbinary 입력이 지원되지 않음

오류 메시지: FUNCTION_NOT_FOUND for st_XXX(st_XXX에 대한 FUNCTION_NOT_FOUND)

원인: 몇 가지 지리 공간 함수에서 더는 이상 기존 VARBINARY 입력 유형 또는 텍스트 관련 함수 서명을 지원하지 않습니다.

권장 솔루션: 지리 공간 함수를 사용하여 입력 유형을 지원되는 유형으로 변환합니다. 지원되는 입력 유형은 오류 메시지에 표시됩니다.

GROUP BY 절에서 중첩된 열은 큰따옴표로 묶어야 합니다.

오류 메시지: "*column_name*"."*nested_column*" must be an aggregate expression or appear in GROUP BY clause

원인: Athena 엔진 버전 3에서는 GROUP BY 절의 중첩된 열 이름을 큰따옴표로 묶어야 합니다. 예를 들어 다음 쿼리에서는 GROUP BY 절에서 user.name이 큰따옴표로 묶여 있지 않기 때문에 오류가 발생합니다.

```
SELECT "user"."name" FROM dataset
GROUP BY user.name
```

권장 솔루션: 다음 예제와 같이 GROUP BY 절의 중첩된 열 이름을 큰따옴표를 묶습니다.

```
SELECT "user"."name" FROM dataset
GROUP BY "user"."name"
```

Iceberg 테이블에서 OPTIMIZE를 사용할 때 예기치 않은 FilterNode 오류 발생

오류 메시지: 계획에서 예상치 못한 FilterNode가 발견되었으며, 커넥터가 제공된 WHERE 표현식을 처리하지 못한 것 같습니다.

원인: Iceberg 테이블에서 실행된 OPTIMIZE 문은 필터 표현식에 파티션되지 않은 열을 포함하는 WHERE 절을 사용했습니다.

권장 해결 방법: 이 OPTIMIZE 문은 파티션별 필터링만 지원합니다. 파티션 테이블에서 OPTIMIZE를 실행할 때는 파티션 열만 WHERE 절에 포함하세요. 파티션되지 않은 테이블에서 OPTIMIZE를 실행하는 경우 WHERE 절을 지정하지 마세요.

Log() 함수 인수 순서

Athena 엔진 버전 2에서는 `log()` 함수의 인수 순서가 `log(value, base)`였습니다. Athena 엔진 버전 3에서는 SQL 표준에 따라 `log(base, value)`로 변경되었습니다.

Minute() 함수에서 연도-달 간격이 지원되지 않음

오류 메시지: Unexpected parameters (interval year to month) for function minute(함수 minute에 대한 예기치 않은 파라미터(연도-달 간격)입니다). 예상: minute(시간대 포함 타임스탬프), minute(시간대 포함 시간), minute(타임스탬프), minute(시간), minute(일-초 간격).

원인: Athena 엔진 버전 3에서는 ANSI SQL 사양에 따라 형식 검사가 `EXTRACT`에 더 정밀하게 적용되었습니다.

권장 솔루션: 유형이 권한 함수 서명과 일치하도록 쿼리를 업데이트합니다.

ORDER BY 표현식이 SELECT 목록에 나타나야 함

오류 메시지: For SELECT DISTINCT, ORDER BY expressions must appear in SELECT list(SELECT DISTINCT의 경우 ORDER BY 표현식이 SELECT 목록에 나타나야 합니다).

원인: `SELECT` 절에서 잘못된 테이블 별칭 지정이 사용되었습니다.

권장 솔루션: `ORDER BY` 표현식의 모든 열에 `SELECT DISTINCT` 절의 적절한 참조가 있는지 다시 확인합니다.

하위 쿼리에서 반환된 여러 열을 비교할 때 쿼리 실패

오류 메시지 예제: Value expression and result of subquery must be of the same type: row(varchar, varchar) vs row(row(varchar, varchar))

원인: Athena 엔진 버전 3에서 구문 업데이트로 인해 다음 예제와 같이 쿼리가 하위 쿼리에서 반환된 여러 값을 비교하려고 할 때 하위 쿼리의 `SELECT` 문이 해당 열 목록을 괄호로 묶으면 이 오류가 발생합니다.

```
SELECT *
FROM table1
WHERE (t1_col1, t1_col2)
IN (SELECT (t2_col1, t2_col2) FROM table2)
```

해결 방법: Athena 엔진 버전 3에서는 다음과 같이 업데이트된 쿼리 예제와 같이 하위 쿼리의 `SELECT` 문에서 열 목록을 묶은 괄호를 제거합니다.

```
SELECT *
FROM table1
WHERE (t1_col1, t1_col2)
IN (SELECT t2_col1, t2_col2 FROM table2)
```

SKIP은 DML 쿼리의 예약어임

SKIP이라는 단어는 이제 SELECT와 같은 DML 쿼리의 예약어입니다. DML 쿼리에서 SKIP을 식별자로 사용하려면 큰따옴표로 묶습니다.

Athena의 예약어에 대한 자세한 내용은 예약어 섹션을 참조하세요.

SYSTEM_TIME 및 SYSTEM_VERSION 절은 시간 이동에 더 이상 사용되지 않습니다.

오류 메시지: mismatched input 'SYSTEM_TIME'(입력 'SYSTEM_TIME'이 일치하지 않습니다). 예상: 'TIMESTAMP', 'VERSION'

원인: Athena 엔진 버전 2의 Iceberg 테이블에서 타임스탬프 및 버전 시간 이동에 FOR SYSTEM_TIME AS OF 및 FOR SYSTEM_VERSION AS OF 절을 사용했습니다. Athena 엔진 버전 3에서는 FOR TIMESTAMP AS OF 및 FOR VERSION AS OF 절을 사용합니다.

권장 솔루션: 다음 예와 같이 시간 이동 작업에 TIMESTAMP AS OF 및 VERSION AS OF 절을 사용하도록 SQL 쿼리를 업데이트합니다.

타임스탬프별 시간 이동:

```
SELECT * FROM TABLE FOR TIMESTAMP AS OF (current_timestamp - interval '1' day)
```

버전별 시간 이동:

```
SELECT * FROM TABLE FOR VERSION AS OF 949530903748831860
```

배열 생성자에 대한 인수가 너무 많음

오류 메시지: TOO_MANY_ARGUMENTS: Too many arguments for array constructor.

원인: 이제 배열 생성자의 최대 요소 수가 254개로 설정됩니다.

권장 솔루션: 다음 예제와 같이 요소 수가 각각 254개 이하인 여러 배열로 나누고 CONCAT 함수를 사용하여 배열을 연결합니다.

```
CONCAT(
ARRAY[x1,x2,x3...x254],
ARRAY[y1,y2,y3...y254],
...
)
```

길이가 0으로 구분된 식별자는 허용되지 않음

오류 메시지: Zero-length delimited identifier not allowed(길이가 0으로 구분된 식별자는 허용되지 않습니다).

원인: 쿼리에서 빈 문자열을 열 별칭으로 사용했습니다.

권장 솔루션: 쿼리를 업데이트하여 열에 비어 있지 않은 별칭을 사용합니다.

데이터 처리 변경 사항

버킷 검증

오류 메시지: HIVE_INVALID_BUCKET_FILES: Hive table is corrupt.

원인: 테이블이 손상되었을 수 있습니다. 버킷 테이블의 쿼리 정확성을 보장하기 위해 Athena 엔진 버전 3에서는 버킷 테이블에서 추가 검증을 활성화하여 쿼리의 정확성을 보장하고 런타임 시 예상치 못한 오류를 방지합니다.

권장 솔루션: Athena 엔진 버전 3을 사용하여 테이블을 다시 생성합니다.

struct를 JSON으로 캐스팅하면 이제 필드 이름이 반환됨

Athena 엔진 버전 3의 `SELECT` 쿼리에서 `struct`를 JSON으로 캐스팅하면 이제 캐스팅이 필드 이름과 값을 모두 반환합니다(예: 그냥 값 대신 `"useragent":null`(예: `null`)).

Iceberg 테이블 열 수준 보안 적용 변경

오류 메시지: Access Denied: Cannot select from columns(액세스 거부됨: 열에서 선택할 수 없습니다)

원인: Iceberg 테이블이 Athena 외부에서 생성되고 Apache Iceberg SDK 0.13.0 이전 버전을 사용합니다. 이전 SDK 버전에서는 AWS Glue의 열이 채워지지 않으므로 Lake Formation에서는 액세스 권한이 부여된 열을 확인할 수 없습니다.

권장 솔루션: Athena ALTER TABLE SET PROPERTIES 문을 사용하여 업데이트를 수행하거나 최신 Iceberg SDK를 사용하여 테이블을 수정하고 AWS Glue에서 열 정보를 업데이트합니다.

List 데이터 유형의 Null이 이제 UDF로 전파됨

오류 메시지: Null Pointer Exception(Null 포인터 예외)

원인: 이 문제는 UDF 커넥터를 사용하고 사용자 정의 Lambda 함수를 구현한 경우 영향을 미칠 수 있습니다.

Athena 엔진 버전 2에서는 사용자 정의 함수에 전달된 List 데이터 유형의 null을 필터링했습니다. Athena 엔진 버전 3에서는 이제 null이 보존되고 UDF로 전달됩니다. 이로 인해 UDF에서 확인하지 않고 null 요소 역참조를 시도하면 null 포인터 예외가 발생할 수 있습니다.

예를 들면 DynamoDB와 같은 원래 데이터 소스에 `[null, 1, null, 2, 3, 4]` 데이터가 있는 경우 사용자 정의 Lambda 함수에 다음과 같이 전달됩니다.

Athena 엔진 버전 2: `[1,2,3,4]`

Athena 엔진 버전 3: `[null, 1, null, 2, 3, 4]`

권장 솔루션: 사용자 정의 Lambda 함수가 목록 데이터 유형의 null 요소를 처리하는지 확인합니다.

문자형 배열의 하위 문자열에 더는 패딩된 공백이 포함되지 않음

오류 메시지: 오류는 발생하지 않지만, 반환된 문자열에 더는 패딩된 공백이 포함되지 않습니다. 예를 들면 `substr(char[20],1,100)`에서 이제는 길이가 100이 아니라 20인 문자열을 반환합니다.

권장 솔루션: 조치가 필요하지 않습니다.

지원되지 않는 10진수 열 유형 강제 변환

오류 메시지: HIVE_CURSOR_ERROR: Failed to read Parquet file: s3://DOC-EXAMPLE-BUCKET/*path*/*file_name*.parquet 또는 Unsupported column type (varchar) for Parquet column ([*column_name*]

원인: Athena 엔진 버전 2에서는 데이터 형식을 `varchar`에서 10진수로 강제 변환하려고 할 때 성공하기도 하지만 실패하는 경우가 더 빈번합니다. Athena 엔진 버전 3에서는 값을 읽으려고 하기 전에 유형이 호환되는지 확인하는 유형 검증 기능을 제공하므로 이러한 강제 변환 시도는 이제 항상 실패합니다.

권장 솔루션: Athena 엔진 버전 2와 Athena 엔진 버전 3 모두에서 Parquet 파일의 10진수 열에 대해 `varchar` 대신 숫자 데이터 형식을 사용하도록 AWS Glue에서 스키마를 수정합니다. 데이터를 다시 크롤링하고 새 열 데이터 형식이 10진수 유형인지 확인하거나 Athena에서 테이블을 수동으로 다시 생성하고 `decimal(`*precision*`, `*scale*`)` 구문을 사용하여 열에 대해 decimal 데이터 형식을 지정합니다.

부동 또는 double NaN 값은 더 이상 bigint로 변환할 수 없음

오류 메시지: INVALID_CAST_ARGUMENT: Cannot cast real/double NaN to bigint

원인: Athena 엔진 버전 3에서는 `NaN`을 더 이상 `bigint`와 같은 0으로 변환할 수 없습니다.

권장 솔루션: `bigint`로 변환할 때 `float` 또는 `double` 열에 `NaN` 값이 없는지 확인합니다.

uuid() 함수 반환 유형 변경

다음 문제는 테이블 및 보기에 모두 영향을 줍니다.

오류 메시지: Unsupported Hive type: uuid

원인: Athena 엔진 버전 2에서 `uuid()` 함수는 문자열을 반환했지만 Athena 엔진 버전 3에서는 무작위로 생성된 유사 UUID(유형 4)를 반환합니다. Athena에서 UUID 열 데이터 형식은 지원되지 않으므로 Athena 엔진 버전 3에서 UUID 열을 생성하기 위해 CTAS 쿼리에서 더 이상 `uuid()` 함수를 직접 사용할 수 없습니다.

예를 들어 다음 `CREATE TABLE` 문은 Athena 엔진 버전 2에서는 성공적으로 완료되지만 Athena 엔진 버전 3에서는 NOT_SUPPORTED: Unsupported Hive type: uuid를 반환합니다.

```
CREATE TABLE uuid_table AS
   SELECT uuid() AS myuuid
```

마찬가지로 다음 `CREATE VIEW` 문은 Athena 엔진 버전 2에서는 성공적으로 완료되지만 Athena 엔진 버전 3에서는 Invalid column type for column myuuid: Unsupported Hive type: uuid를 반환합니다.

```
CREATE VIEW uuid_view AS
   SELECT uuid() AS myuuid
```

Athena 엔진 버전 2에서 생성된 보기를 Athena 엔진 버전 3에서 쿼리하면 다음과 같은 오류가 발생합니다.

VIEW_IS_STALE: line 1:15: View 'awsdatacatalog.mydatabase.uuid_view' is stale or in invalid state: column [myuuid] of type uuid projected from query view at position 0 cannot be coerced to column [myuuid] of type varchar stored in view definition

권장 솔루션: 테이블 또는 보기를 생성하는 경우 다음 예제와 같이 `cast()` 함수를 사용하여 `uuid()`의 출력을 `varchar`로 변환합니다.

```
CREATE TABLE uuid_table AS
   SELECT CAST(uuid() AS VARCHAR) AS myuuid
```

```
CREATE VIEW uuid_view AS
   SELECT CAST(uuid() AS VARCHAR) AS myuuid
```

CHAR 및 VARCHAR 강제 변환 문제

Athena 엔진 버전 3에서 `varchar` 및 `char` 관련 강제 변환 문제가 발생하는 경우 이 섹션의 해결 방법을 사용합니다. 이 해결 방법을 사용할 수 없는 경우 AWS Support에 문의하세요.

CHAR 및 VARCHAR 입력이 혼합된 상태에서 CONCAT 함수 실패

문제: Athena 엔진 버전 2에서는 다음 쿼리에 성공합니다.

```
SELECT concat(CAST('abc' AS VARCHAR(20)), '12', CAST('a' AS CHAR(1)))
```

하지만 Athena 엔진 버전 3에서는 다음과 같이 동일한 쿼리에 실패합니다.

오류 메시지: FUNCTION_NOT_FOUND: line 1:8: Unexpected parameters (varchar(20), varchar(2), char(1)) for function concat. Expected: concat(char(x), char(y)), concat(array(E), E) E, concat(E, array(E)) E, concat(array(E)) E, concat(varchar), concat(varbinary)

권장 솔루션: `concat` 함수를 사용하는 경우 `char` 또는 `varchar`를 혼합하지 않고 둘 중 하나로만 변환합니다.

CHAR 및 VARCHAR 입력에서 SQL || 연결 실패

Athena 엔진 버전 3에서는 이중 수직 막대 `||` 연결 연산자를 사용하려면 가 입력으로 `varchar`를 사용해야 합니다. `varchar` 및 `char` 유형의 조합을 입력으로 사용할 수는 없습니다.

오류 메시지: TYPE_NOT_FOUND: line 1:26: Unknown type: char(65537)

원인: 다음 예제와 같이 `char` 및 `varchar`를 연결하기 위해 `||`를 사용하는 쿼리에서 오류가 발생할 수 있습니다.

```
SELECT CAST('a' AS CHAR) || CAST('b' AS VARCHAR)
```

권장 솔루션: 다음 예제와 같이 `varchar`를 `varchar`에 연결합니다.

```
SELECT CAST('a' AS VARCHAR) || CAST('b' AS VARCHAR)
```

CHAR 및 VARCHAR UNION 쿼리 실패

오류 메시지: NOT_SUPPORTED: Unsupported Hive type: char(65536). 지원되는 CHAR 유형: CHAR(<=255)

원인: 다음 예제와 같이 `char` 및 `varchar`를 결합하려고 시도하는 쿼리가 원인입니다.

```
CREATE TABLE t1 (c1) AS SELECT CAST('a' as CHAR) as c1 UNION ALL SELECT CAST('b' AS
 VARCHAR) AS c1
```

권장 솔루션: 쿼리 예제와 같이`char` 대신 `'a'`를 `varchar`로 변환합니다.

CHAR 또는 VARCHAR 강제 변환 후 불필요한 빈 공백

Athena 엔진 버전 3에서 배열 또는 단일 열을 구성할 때 `char(X)` 및 `varchar` 데이터를 단일 유형으로 변환하는 경우 `char(65535)`가 대상 유형이고 각 필드는 불필요한 후행 공백을 여러 개 포함합니다.

원인: Athena 엔진 버전 3에서 `varchar` 및 `char(X)`를 `char(65535)`로 강제 변환한 후 데이터 오른쪽을 공백으로 채웁니다.

권장 솔루션: 각 필드를 명시적으로 `varchar`로 변환합니다.

타임스탬프 변경 사항

varchar 동작 변경에 시간대 포함 타임스탬프 캐스팅

Athena 엔진 버전 2에서는 `varchar`에 시간대 포함 `Timestamp`를 캐스팅하면 일부 시간대 리터럴이 변경되었습니다(예: `US/Eastern`이 `America/New_York`으로 변경됨). Athena 엔진 버전 3에서는 이 동작이 발생하지 않습니다.

날짜 타임스탬프 오버플로 오류 발생

오류 메시지: Millis overflow: XXX(밀리초 오버플로: XXX)

원인: Athena 엔진 버전 2에서는 ISO 8601 날짜의 오버플로를 확인하지 않았기 때문에 일부 날짜에서 음수 타임스탬프가 생성되었습니다. Athena 엔진 버전 3에서는 이 오버플로를 확인하고 예외를 발생시킵니다.

권장 솔루션: 타임스탬프가 범위 내에 있는지 확인합니다.

TIME이 지원되지 않는 정치적 시간대

오류 메시지: INVALID LITERAL

원인: 쿼리(예: `SELECT TIME '13:21:32.424 America/Los_Angeles'`).

권장 솔루션: `TIME`에 정치적 시간대를 함께 사용하지 않습니다.

타임스탬프 열의 정밀도 불일치로 직렬화 오류 발생

오류 메시지: SERIALIZATION_ERROR: Could not serialize column '*COLUMNZ*' of type 'timestamp(3)' at position *X*:*Y*(SERIALIZATION_ERROR: 위치 X:Y의 유형 'timestamp(3)'의 열 'COLUMNZ'는 직렬화할 수 없습니다).

*COLUMNZ*는 문제의 원인이 되는 열의 출력 이름입니다. 숫자 *X*: *Y*는 출력에서 열의 위치를 나타냅니다.

원인: Athena 엔진 버전 3에서는 데이터의 타임스탬프의 정밀도가 테이블 사양의 열 데이터 유형에 지정된 정밀도와 동일한지 확인합니다. 현재 이 정밀도는 항상 3입니다. 데이터의 정밀도가 이보다 크면 쿼리가 실패하고 오류가 표시됩니다.

권장 솔루션: 데이터를 검사하여 타임스탬프의 정밀도가 밀리초 단위인지 확인합니다.

Iceberg 테이블에 대한 UNLOAD 및 CTAS 쿼리의 타임스탬프 정밀도가 잘못됨

오류 메시지: Incorrect timestamp precision for timestamp(6); the configured precision is MILLISECONDS

원인: Athena 엔진 버전 3에서는 데이터의 타임스탬프의 정밀도가 테이블 사양의 열 데이터 형식에 지정된 정밀도와 동일한지 확인합니다. 현재 이 정밀도는 항상 3입니다. 데이터의 정밀도가 이보다 큰 경우(예: 밀리초 대신 마이크로초) 쿼리에 실패하고 오류가 기록될 수 있습니다.

해결 방법: 이 문제를 해결하려면 먼저 Iceberg 테이블을 생성하는 다음 CTAS 예제에서처럼 타임스탬프 정밀도를 6으로 변환(CAST)합니다. Timestamp precision (3) not supported for Iceberg 오류가 발생하지 않도록 하려면 정밀도를 3 대신 6으로 지정해야 합니다.

```
CREATE TABLE my_iceberg_ctas
WITH (table_type = 'ICEBERG', location = 's3://DOC-EXAMPLE-BUCKET/table_ctas/',
format = 'PARQUET')
AS SELECT id, CAST(dt AS timestamp(6)) AS "dt"
FROM my_iceberg
```

Athena는 타임스탬프 6을 지원하지 않으므로 값을 다시 타임스탬프로 변환합니다(예: 보기에서와 같이). 다음 예제에서는 `my_iceberg_ctas` 테이블에서 보기를 생성합니다.

```
CREATE OR REPLACE VIEW my_iceberg_ctas_view AS
SELECT cast(dt AS timestamp) AS dt
FROM my_iceberg_ctas
```

ORC 파일의 Timestamp로 Long 유형을 읽거나 반대의 경우 이제는 잘못된 형식의 ORC 파일 오류 발생

오류 메시지: Error opening Hive split ‘FILE (SPLIT POSITION)’ Malformed ORC file.(Hive 분할 ‘FILE(SPLIT POSITION)’ 형식이 잘못된 ORC 파일을 여는 동안 오류가 발생했습니다). Cannot read SQL type timestamp from ORC stream .long_type of type LONG(LONG 유형의 ORC 스트림 .long_type에서 SQL 유형 타임스탬프를 읽을 수 없습니다.)

원인: 이제 Athena 엔진 버전 3에서는 Long 데이터 형식에서 `Timestamp`로 또는 `Timestamp`에서 `Long`으로 암시적 강제 변환을 거부합니다. 이전에는`Long` 값이 Epoch 밀리초였던 것처럼 암시적으로 타임스탬프로 변환되었습니다.

권장 솔루션: `from_unixtime` 함수를 사용하여 열을 명시적으로 캐스팅하거나 `from_unixtime` 함수를 사용하여 향후 쿼리를 위한 추가 열을 생성합니다.

시간 및 연도-달 간격은 지원되지 않음

오류 메시지: TYPE MISMATCH

원인: Athena 엔진 버전 3에서는 시간 및 연도-달 간격을 지원하지 않습니다(예: `SELECT TIME '01:00' + INTERVAL '3' MONTH`).

int96 Parquet 형식의 타임스탬프 오버플로

오류 메시지: Invalid timeOfDayNanos(잘못된 timeOfDayNanos)

원인: `int96` Parquet 형식의 타임스탬프 오버플로입니다.

권장 솔루션: 문제가 있는 특정 파일을 식별합니다. 그런 다음에 잘 알려진 최신 Parquet 라이브러리로 데이터 파일을 다시 생성하거나 Athena CTAS를 사용합니다. 문제가 지속되면 Athena 지원팀에 연락하여 데이터 파일이 어떻게 생성되는지 알려주세요.

문자열에서 타임스탬프로 변환할 때 날짜와 시간 값 사이에 공백이 필요함

오류 메시지: INVALID_CAST_ARGUMENT: Value cannot be cast to timestamp.

원인: Athena 엔진 버전 3에서는 cast에 대한 입력 문자열에서 날짜와 시간 값 사이의 유효한 구분 기호로 더 이상 하이픈을 허용하지 않습니다. 예를 들어 다음 쿼리는 Athena 엔진 버전 2에서는 작동하지만 Athena 엔진 버전 3에서는 작동하지 않습니다.

```
SELECT CAST('2021-06-06-23:38:46' AS timestamp) AS this_time
```

권장 솔루션: Athena 엔진 버전 3에서는 다음 예제와 같이 날짜 및 시간 사이의 하이픈을 공백으로 바꿉니다.

```
SELECT CAST('2021-06-06 23:38:46' AS timestamp) AS this_time
```

to_iso8601() timestamp 반환 값 변경

오류 메시지: 없음

원인: Athena 엔진 버전 2에서는 to_iso8601 함수에 전달된 값에 시간대가 없어도 해당 함수는 시간대가 포함된 타임스탬프를 반환합니다. Athena 엔진 버전 3에서 to_iso8601 함수는 전달된 인수에 시간대가 포함된 경우에만 시간대가 포함된 타임스탬프를 반환합니다.

예를 들어 다음 쿼리는 현재 날짜를 to_iso8601 함수에 두 번 전달합니다. 처음에는 시간대가 있는 타임스탬프로, 그 다음에는 타임스탬프로 전달합니다.

```
SELECT TO_ISO8601(CAST(CURRENT_DATE AS TIMESTAMP WITH TIME ZONE)),
 TO_ISO8601(CAST(CURRENT_DATE AS TIMESTAMP))
```

다음 출력은 각 엔진에서 쿼리 결과를 보여줍니다.

Athena 엔진 버전 2:

#	_col0	_col1
1	2023-02-24T00:00:00.000Z	2023-02-24T00:00:00.000Z

Athena 엔진 버전 3:

#	_col0	_col1
1	2023-02-24T00:00:00.000Z	2023-02-24T00:00:00.000

권장 솔루션: 이전 동작을 복제하려면 다음 예제와 같이 to_iso8601에 전달하기 전에 with_timezone 함수에 타임스탬프 값을 전달할 수 있습니다.

```
SELECT to_iso8601(with_timezone(TIMESTAMP '2023-01-01 00:00:00.000', 'UTC'))
```

Result

#	_col0
1	2023-01-01T00:00:00.000Z

at_timezone() 첫 번째 파라미터에서 날짜를 지정해야 함

문제: Athena 엔진 버전 3에서 at_timezone 함수는 time_with_timezone 값을 첫 번째 파라미터로 사용할 수 없습니다.

원인: 날짜 정보가 없으면 전달된 값이 일광 절약 시간인지, 표준 시간인지 확인할 수 없습니다. 예를 들어 전달된 값이 태평양 일광 절약 시간(PDT)인지, 태평양 표준시(PST)인지 확인할 방법이 없으므로 at_timezone('12:00:00 UTC', 'America/Los_Angeles')은 모호합니다.

제한 사항

Athena 엔진 버전 3에는 다음과 같은 제한 사항이 있습니다.

- 쿼리 성능 – 많은 쿼리가 Athena 엔진 버전 3에서 더 빠르게 실행되지만, 일부 쿼리 계획은 Athena 엔진 버전 2와 다를 수 있습니다. 따라서 일부 쿼리는 대기 시간 또는 비용이 다를 수 있습니다.
- Trino 및 Presto 커넥터 – Trino 또는 Presto 커넥터는 지원되지 않습니다. Amazon Athena 연합 쿼리를 사용하여 데이터 소스를 연결합니다. 자세한 내용은 Amazon Athena 연합 쿼리 사용 단원을 참조하십시오.
- 내결함성 실행 – Trino 내결함성 실행(Trino Tardigrade)은 지원되지 않습니다.
- 함수 파라미터 한도 - 함수는 127개를 초과하는 파라미터를 사용할 수 없습니다. 자세한 내용은 함수 호출에 사용하는 인수가 너무 많음 단원을 참조하십시오.

리소스 제한으로 인해 쿼리가 실패하지 않도록 Athena 엔진 버전 2에서는 다음의 제한이 도입되었습니다. 이러한 제한은 사용자가 구성할 수 없습니다.

- 결과 요소 수 - min(col, n), max(col, n), min_by(col1, col2, n), max_by(col1, col2, n) 함수의 경우 결과 요소 수 n이 10,000개 이하로 제한됩니다.

- 그룹화 세트 - 그룹화 집합의 최대 슬라이스 수는 2048개입니다.
- 최대 텍스트 파일 줄 길이 - 텍스트 파일의 기본 최대 줄 길이는 200MB입니다.
- 시퀀스 함수 최대 결과 크기 - 시퀀스 함수의 최대 결과 크기는 50,000개의 항목입니다. 예를 들어, `SELECT sequence(0,45000,1)`은 성공하지만 `SELECT sequence(0,55000,1)`은 오류 메시지와 함께 실패합니다. 시퀀스 함수의 결과는 50,000개 항목 이하여야 합니다. 이 제한은 타임스탬프를 포함해 시퀀스 함수의 모든 입력 형식에 적용됩니다.

Athena 엔진 버전 2

Athena 엔진 버전 2에는 다음 변경 사항이 도입되었습니다.

- 개선 사항 및 새로운 기능
  - 그룹화, 조인, 하위 쿼리 개선 사항
  - 데이터 형식 기능 향상
  - 추가된 함수
  - 성능 개선
  - JSON 관련 개선 사항
- 호환성에 영향을 미치는 변경
  - 버그 수정
  - 지리 공간 함수 변경 사항
  - ANSI SQL 규정 준수
  - 대체된 함수
  - Limits

개선 사항 및 새로운 기능

- EXPLAIN 및 EXPLAIN ANALYZE – Athena에서 `EXPLAIN` 문을 사용하여 SQL 쿼리의 실행 계획을 볼 수 있습니다. `EXPLAIN ANALYZE`를 사용하여 SQL 쿼리에 대한 분산 실행 계획과 각 작업의 비용을 확인합니다. 자세한 내용은 Athena에서 EXPLAIN 및 EXPLAIN ANALYZE 사용 단원을 참조하십시오.
- 연합 쿼리 - Athena 엔진 버전 2에서는 연합 쿼리가 지원됩니다. 자세한 내용은 Amazon Athena 연합 쿼리 사용 단원을 참조하세요.
- 지리 공간 함수 - 25개 이상의 지리 공간 함수가 추가되었습니다. 자세한 내용은 Athena 엔진 버전 2의 새 지리 공간 함수 단원을 참조하세요.

- 중첩 스키마 - 비용을 절감하는 중첩 스키마 읽기 지원이 추가되었습니다.
- 준비된 문 - 다른 쿼리 파라미터를 사용하여 동일한 쿼리를 반복적으로 실행하기 위해 준비된 문을 사용합니다. 준비된 문에는 런타임 시 값을 전달하는 자리 표시자 파라미터가 포함되어 있습니다. 준비된 문은 SQL 명령어 삽입 공격을 방지하는 데 도움이 됩니다. 자세한 내용은 파라미터화된 쿼리 사용 단원을 참조하십시오.
- 스키마 변화(schema evolution) 지원 - Parquet 형식의 데이터에 대한 스키마 변화 지원이 추가되었습니다.
  - 파티션 스키마가 테이블 스키마와 다른 파티션에서 배열, 맵 또는 행 유형 열 읽기에 대한 지원이 추가되었습니다. 파티션이 생성된 후 테이블 스키마가 업데이트될 때 이 작업이 발생할 수 있습니다. 변경된 열 유형은 호환이 가능해야 합니다. 행 유형의 경우 후행 필드를 추가하거나 삭제할 수 있지만 해당 필드(서수 기준)는 같은 이름을 가져야 합니다.
  - 이제 ORC 파일은 누락된 필드가 있는 구조체 열을 가질 수 있습니다. 이로써 ORC 파일을 다시 작성하지 않고도 테이블 스키마를 변경할 수 있습니다.
  - ORC 구조체 열은 이제 서수가 아닌 이름으로 매핑됩니다. 이 때문에 ORC 파일의 누락되거나 남는 구조체 필드가 올바르게 처리됩니다.
- SQL OFFSET - 이제 `SELECT` 문에서 SQL `OFFSET` 절이 지원됩니다. 자세한 내용은 SELECT 단원을 참조하십시오.
- UNLOAD 문 - `UNLOAD` 문을 사용하여 `SELECT` 쿼리의 출력을 PARQUET, ORC, AVRO, JSON 형식으로 쓸 수 있습니다. 자세한 내용은 UNLOAD 단원을 참조하십시오.

그룹화, 조인, 하위 쿼리 개선 사항

- 복합 그룹화(Complex grouping) - 복합 그룹화 연산이 지원됩니다.
- 상호 연관 하위 쿼리(Correlated subquery) - `IN` 조건자에서 상호 연관 하위 쿼리 및 강제 변환이 필요한 상호 연관 하위 쿼리가 지원됩니다.
- 크로스 조인(CROSS JOIN) - `LATERAL` 파생 테이블에 대한 `CROSS JOIN`이 지원됩니다.
- 그룹화 집합(GROUPING SETS) - `GROUPING SETS`을 사용하는 쿼리의 집계에서 `ORDER BY` 절이 지원됩니다.
- Lambda 표현식 - Lambda 표현식에서의 행 필드 역참조가 지원됩니다.
- semijoin의 Null 값 - semijoin 왼쪽의 null 값이 지원됩니다(즉, `IN` 조건자와 하위 쿼리).
- 공간 조인(Spatial joins) - 브로드캐스트 공간 조인 및 공간 왼쪽 조인(spatial left join)이 지원됩니다.
- 디스크로 분산(Spill to disk) - 메모리 집약적 `INNER JOIN` 및 `LEFT JOIN` 연산에 대해 Athena가 중간 작업 결과를 디스크로 오프로드합니다. 이를 통해 많은 양의 메모리가 필요한 쿼리를 실행할 수 있습니다.

데이터 형식 기능 향상

- INTEGER의 INT - `INTEGER` 데이터 형식에 대한 별칭으로서 `INT`가 지원됩니다.
- INTERVAL 형식 - `INTERVAL` 형식으로 캐스팅이 지원됩니다.
- IPADDRESS - DML 쿼리에서 IP 주소를 나타내는 새로운 `IPADDRESS` 유형이 추가되었습니다. `VARBINARY` 형식과 `IPADDRESS` 형식 간의 캐스팅이 지원됩니다. DDL 쿼리에서는 `IPADDRESS` 유형이 인식되지 않습니다.
- IS DISTINCT FROM - `JSON` 및 `IPADDRESS` 형식에 대해 `IS DISTINCT FROM`이 지원됩니다.
- Null 동치 검사 - 이제 `ARRAY`, `MAP`, `ROW` 데이터 구조에서 Null 값에 대한 동치 검사가 지원됩니다. 예를 들어, 표현식 `ARRAY ['1', '3', null] = ARRAY ['1', '2', null]`은 `false`를 반환합니다. 이전에 null 요소는 비교가 지원되지 않음 오류 메시지를 반환했습니다.
- 행 형식 강제 변환 - 이제 필드 이름에 관계없이 행 형식 간의 강제 변환이 허용됩니다. 이전에는 원본 형식의 필드 이름이 대상 형식과 일치하거나 대상 형식에 익명 필드 이름이 있는 경우에만 행 형식이 다른 행 형식으로 강제 변환되었습니다.
- 시간 빨셈 - 모든 `TIME` 및 `TIMESTAMP` 형식에 대한 빨셈이 구현되었습니다.
- 유니코드 - 문자열 리터럴에서 이스케이프 처리된 유니코드 시퀀스가 지원됩니다.
- VARBINARY 연결 - `VARBINARY` 값의 연결이 지원됩니다.

  윈도 값 함수 - 윈도 값 함수가 이제 `IGNORE NULLS`와 `RESPECT NULLS`를 지원합니다.

함수의 추가 입력 형식

이제 다음 함수에 추가적인 입력 형식이 허용됩니다. 각 함수에 대한 자세한 내용은 Presto 설명서의 해당 링크를 방문하세요.

- approx_distinct() – approx_distinct() 함수가 이제 다음 형식을 지원합니다. `INTEGER`, `SMALLINT`, `TINYINT`, `DECIMAL`, `REAL`, `DATE`, `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, `TIME`, `TIME WITH TIME ZONE`, `IPADDRESS`, `CHAR`.
- Avg(), sum() – avg() 및 sum() 집계 함수가 이제 `INTERVAL` 데이터 형식을 지원합니다.
- Lpad(), rpad() – lpad 및 rpad 함수가 이제 `VARBINARY` 입력에서 작동합니다.
- Min(), max() – 이제 min() 및 max() 집계 함수가 쿼리 분석 시간에 알 수 없는 입력 형식을 허용하므로 `NULL` 리터럴과 함께 함수를 사용할 수 있습니다.
- regexp_replace() – 대체 시마다 Lambda 함수를 실행할 수 있는 regexp_replace()의 변형이 추가되었습니다.

- Sequence() – 암시적인 1일 단위 증분(implicit one-day step increment)을 갖는 변형을 포함해, sequence() 함수에 대한 DATE 변형이 추가되었습니다.
- ST_Area() – ST_Area() 지리 공간 함수가 이제 모든 geometry 형식을 지원합니다.
- Substr() – substr 함수가 이제 VARBINARY 입력에서 작동합니다.
- zip_with() – 이제 zip_with()에 길이가 일치하지 않는 배열을 사용할 수 있습니다. 누락된 위치는 null로 채워집니다. 이전에는 길이가 다른 배열이 전달되었을 때 오류가 발생했습니다. 이 변경으로 인해 원래 null이었던 값과 배열을 같은 길이로 채우기 위해 추가된 값을 구별하기가 어려울 수 있습니다.

추가된 함수

다음 목록에는 Athena 엔진 버전 2의 새로운 시작인 함수가 포함되어 있습니다. 이 목록에는 지리 공간 함수가 포함되어 있지 않습니다. 지리 공간 함수의 목록은 Athena 엔진 버전 2의 새 지리 공간 함수을(를) 참조하세요.

각 함수에 대한 자세한 내용은 Presto 설명서의 해당 링크를 방문하세요.

집계 함수

reduce_agg()

배열 함수와 연산자

array_sort() - 이 함수의 변형(비교 연산자로 Lambda 함수를 취함)이 추가되었습니다.

ngrams()

이진 함수와 연산자

from_big_endian_32()

from_ieee754_32()

from_ieee754_64()

hmac_md5()

hmac_sha1()

hmac_sha256()

hmac_sha512()

spooky_hash_v2_32()

spooky_hash_v2_64()

to_big_endian_32()

to_ieee754_32()

to_ieee754_64()

날짜 및 시간 함수와 연산자

millisecond()

parse_duration()

to_milliseconds()

맵 함수와 연산자

multimap_from_entries()

수학 함수와 연산자

inverse_normal_cdf()

wilson_interval_lower()

wilson_interval_upper()

분위수 다이제스트 함수

분위수 다이제스트 함수 및 `qdigest` 분위수 다이제스트 형식이 추가되었습니다.

문자열 함수와 연산자

hamming_distance()

split_to_multimap()

성능 개선

Athena 엔진 버전 2에서는 다음과 같은 기능의 성능이 향상되었습니다.

쿼리 성능

- 브로드캐스트 조인 성능 - 작업자 노드에 동적 파티션 정리를 적용하여 브로드캐스트 조인 성능을 개선했습니다.

- 버킷팅된 테이블 - 쓰기 중인 데이터가 이미 적절히 분할된 경우(예: 버킷팅된 조인의 출력인 경우) 버킷팅된 테이블에 대한 쓰기 성능이 향상되었습니다.
- DISTINCT - `DISTINCT`를 사용하는 일부 쿼리의 성능이 향상되었습니다.

  동적 필터링 및 파티션 정리 – 개선으로 성능이 향상되고 쿼리에서 스캔되는 데이터 양이 감소되었습니다.
- 필터 및 프로젝션 연산 - 이제 가능한 경우 필터 및 프로젝션 연산이 항상 열을 기준으로 처리됩니다. 사전 인코딩이 효과적인 경우 엔진이 자동으로 이를 활용합니다.
- 수집 교환(Gathering exchanges) - 수집 교환을 통해 쿼리의 성능이 개선되었습니다.
- 전역 집계(Global aggregations) - 필터링된 전역 집계를 수행하는 일부 쿼리의 성능이 개선되었습니다.
- GROUPING SETS, CUBE, ROLLUP - `GROUPING SETS`, `CUBE` 또는 `ROLLUP`이 관련된 쿼리의 성능이 개선되었습니다. 이를 활용하여 하나의 쿼리에 여러 열 집합을 집계할 수 있습니다.
- 매우 선별적인 필터 - 매우 선별적인 필터로 쿼리의 성능이 개선되었습니다.
- JOIN 및 AGGREGATE 작업 - `JOIN` 및 `AGGREGATE` 작업의 성능이 향상되었습니다.
- LIKE – `information_schema` 테이블의 열에 `LIKE` 조건자를 사용하는 쿼리의 성능이 향상되었습니다.
- ORDER BY 및 LIMIT - 불필요한 데이터 교환을 방지하기 위해 `ORDER BY` 및 `LIMIT`가 관련된 쿼리의 계획, 성능, 메모리 사용량을 개선했습니다.
- ORDER BY – 이제 `ORDER BY` 연산이 기본으로 배포되어 더 큰 `ORDER BY` 절을 사용할 수 있습니다.
- ROW 형식 변환 - `ROW` 형식 간의 변환 시 성능이 향상되었습니다.
- 구조 형식 - 구조 형식을 처리하고 스캔, 조인, 집계 또는 테이블 쓰기를 포함하는 쿼리의 성능이 개선되었습니다.
- 테이블 스캔 - 경우에 따라 중복 테이블 스캔을 방지하기 위한 최적화 규칙이 도입되었습니다.
- UNION - `UNION` 쿼리의 성능이 개선되었습니다.

쿼리 계획 성능

- 계획 성과 - 다수의 열을 가진 여러 테이블을 조인하는 쿼리의 계획 성능이 개선되었습니다.
- 조건자 평가 - 계획의 조건자 푸시다운 중 조건자 평가 성능이 향상되었습니다.
- 캐스팅에 조건자 푸시다운 지원 - 열의 유형이 일치하도록 값 목록의 값에 캐스팅이 필요한 경우 *<column>* `IN` *<values list>* 조건자에 대해 조건자 푸시다운을 지원합니다.

- 조건자 추론 및 푸시다운 - *<symbol>* IN *<subquery>* 조건자를 사용하는 쿼리에서 조건자 추론 및 푸시다운이 확대되었습니다.
- 시간 초과 - 드문 경우지만 쿼리 계획 시간 초과가 발생할 수 있는 버그가 수정되었습니다.

조인 성능

- 맵 열을 포함한 조인 - 맵 열을 포함한 조인 및 집계의 성능이 개선되었습니다.
- 비동치(non-equality) 조건만 포함한 조인 - 해시 조인 대신 중첩된 루프 조인을 사용하여 비동치 조건만 포함한 조인의 성능이 개선되었습니다.
- 외부 조인 - 이제 외부 조인이 관련된 쿼리에 대해 조인 배포 유형이 자동으로 선택됩니다.
- 함수 조인의 범위 - 조건이 함수의 범위인 조인의 성능이 개선되었습니다(예:`a JOIN b ON b.x < f(a.x) AND b.x > g(a.x)`).
- 디스크 유출 - `JOIN` 작업의 성능을 높이고 메모리 오류를 줄이기 위해 디스크 유출 관련 버그 및 메모리 문제가 수정되었습니다.

하위 쿼리 성능

- 상호 연관 EXISTS 하위 쿼리 - 상호 관련된 `EXISTS` 하위 쿼리의 성능이 개선되었습니다.
- 등식이 포함된 상호 연관 하위 쿼리 - 등식 조건자가 포함된 상호 연관 하위 쿼리에 대한 지원이 개선되었습니다.
- 부등식이 포함된 상호 연관 하위 쿼리 - 부등식이 포함된 상호 연관 하위 쿼리의 성능이 개선되었습니다.
- 하위 쿼리에 대한 Count(*) 집계 - 알려진 상수의 카디널리티를 갖는 하위 쿼리에 대한 `count(*)` 집계 성능이 개선되었습니다.
- 외부 쿼리 필터 전파 - 외부 쿼리의 필터를 하위 쿼리로 전파할 수 있는 경우 상호 연관 하위 쿼리의 성능이 개선되었습니다.

함수 성능

- 집계 윈도 함수 - 집계 윈도 함수의 성능이 개선되었습니다.
- element_at() – 맵의 크기에 대한 비율이 아닌 상수 시간(constant time)이 되는 맵에 대해 `element_at()`의 성능이 개선되었습니다.
- Grouping() – `grouping()`을 포함하는 쿼리의 성능이 개선되었습니다.
- JSON 캐스팅 - `JSON`에서 `ARRAY` 또는 `MAP` 형식으로의 캐스팅 성능이 개선되었습니다.

- 맵 반환 함수 - 맵을 반환하는 함수의 성능이 개선되었습니다.
- 맵대맵(Map-to-map) 캐스팅 - 맵대맵 캐스트의 성능이 개선되었습니다.
- Min() and max() – `min()` 및 `max()` 함수가 불필요한 객체 생성을 방지하도록 최적화되어 가비지 수집 오버헤드가 감소했습니다.
- row_number() – 생성된 행 번호에서 `row_number()` 뒤에 필터를 사용하는 쿼리의 성능과 메모리 사용량이 개선되었습니다.
- 윈도 함수 - 동일한 `PARTITION BY` 및 `ORDER BY` 절이 있는 윈도 함수를 포함한 쿼리의 성능이 개선되었습니다.
- 윈도 함수 - 비슷한 사양을 가진 특정 윈도 함수의 성능이 개선되었습니다(예: `LAG`).

지리 공간 성능

- 지오메트리 직렬화 - 지오메트리 값의 직렬화 성능이 개선되었습니다.
- 지리 공간 함수 - `ST_Intersects()`, `ST_Contains()`, `ST_Touches()`, `ST_Within()`, `ST_Overlaps()`, `ST_Disjoint()`, `transform_values()`, `ST_XMin()`, `ST_XMax()`, `ST_YMin()`, `ST_YMax()`, `ST_Crosses()`, `array_intersect()`의 성능이 개선되었습니다.
- ST_Distance() – `ST_Distance()` 함수가 관련된 조인 쿼리의 성능이 개선되었습니다.
- ST_Intersection() – 좌표 축에 정렬된 직사각형에 대한 `ST_Intersection()` 함수가 최적화되었습니다(예: `ST_Envelope()` 및 `bing_tile_polygon()` 함수에 의해 생성된 다각형).

JSON 관련 개선 사항

맵 함수

- 모든 경우에서 `O(n)`부터 `O(1)`까지 맵 아래 첨자의 성능이 개선되었습니다. 이전에는 특정 함수와 리더가 생성한 맵만 이러한 개선 사항을 활용했습니다.
- `map_from_entries()` 및 `map_entries()` 함수를 추가했습니다.

캐스팅

- `REAL`, `TINYINT` 또는 `SMALLINT`에서 `JSON`으로 캐스팅하는 기능이 추가되었습니다.
- 이제 `JSON`이 `ROW`에 모든 필드를 포함하지 않더라도 `JSON`을 `ROW`로 캐스팅할 수 있습니다.
- `CAST(json_parse(...) AS ...)`의 성능이 개선되었습니다.
- `JSON`에서 `ARRAY` 또는 `MAP` 형식으로 캐스팅하는 성능이 개선되었습니다.

새 JSON 함수

- is_json_scalar()

호환성에 영향을 미치는 변경

호환성에 영향을 미치는 변경 사항에는 버그 수정, 지리 공간 함수에 대한 변경, 대체된 함수, 제한 도입이 있습니다. ANSI SQL 규정 준수의 개선으로 인해 비표준 동작에 의존했던 쿼리는 호환되지 않을 수 있습니다.

버그 수정

다음 변경은 쿼리가 성공적으로 실행되었지만 결과가 정확하지 않았던 동작 문제를 수정합니다.

- fixed_len_byte_array Parquet 열이 이제 DECIMAL로 허용됨 - Parquet 스키마에 `DECIMAL`로 주석이 지정된 경우 `fixed_len_byte_array` 형식의 Parquet 열에 대한 쿼리가 성공하고 올바른 값을 반환합니다. `DECIMAL` 주석이 없는 `fixed_len_byte_array` 열에 대한 쿼리는 오류와 함께 실패합니다. 이전에는 DECIMAL 주석이 없는 `fixed_len_byte_array` 열에 대한 쿼리가 성공했지만 이해할 수 없는 값을 반환했습니다.
- json_parse()가 더 이상 후행 문자를 무시하지 않음 – 이전에는 `[1,2]abc` 같은 입력이 `[1,2]`로 성공적으로 파싱되었습니다. 이제 후행 문자를 사용하면 오류 메시지('[1, 2]abc'을 JSON으로 변환할 수 없음)가 생성됩니다.
- Round() 소수점 정밀도 수정 – x가 소수이거나, x가 스케일이 0인 소수이고 d가 음의 정수인 경우 이제 `round(x, d)`가 x를 올바르게 반올림합니다. 이전에는 이러한 경우 반올림이 발생하지 않았습니다.
- round(x, d) 및 truncate(x, d) – 이제 `round(x, d)` 및 `truncate(x, d)` 함수의 부호수에서 d 파라미터가 `INTEGER` 형식입니다. 이전에는 d가 `BIGINT` 형식일 수 있었습니다..
- 중복 키를 포함한 Map() – 이제 `map()`은 중복 키에 대해 아무 경고 없이 손상된 맵을 생성하는 대신 오류를 표시합니다. 현재 중복 키를 사용하여 맵 값을 구성한 쿼리는 실패하며 오류를 표시합니다.
- null 항목이 있을 시 map_from_entries()에서 오류 발생 – 이제 입력 배열에 null 항목이 포함되어 있으면 `map_from_entries()`가 오류를 발생시킵니다. `NULL`을 값으로 전달하여 맵을 구성하는 쿼리는 이제 실패합니다.
- 테이블 - 지원되지 않는 파티션 유형을 가진 테이블은 더 이상 만들 수 없습니다.
- 통계 함수의 수치 안정성 개선 - 통계 함수 `corr()`, `covar_samp()` , `regr_intercept()`, `regr_slope()`의 수치 안정성이 개선되었습니다.

- 이제 Parquet에 정의된 TIMESTAMP 정밀도가 반영됨 - 이제 Parquet 스키마에서 `TIMESTAMP` 값의 정밀도와 `TIMESTAMP` 열에 대해 정의된 정밀도가 일치해야 합니다. 정밀도가 일치하지 않을 경우 타임스탬프가 잘못됩니다.
- 시간대 정보 - 이제 표준 시간대 정보가 Java 1.8 SDK의 java.time 패키지를 사용하여 계산됩니다.
- INTERVAL_DAY_TO_SECOND 및 INTERVAL_YEAR_TO_MONTH 데이터 형식의 합 - 더 이상 `SUM(NULL)`을 직접 사용할 수 없습니다. `SUM(NULL)`을 사용하려면 `NULL`을 `BIGINT`, `DECIMAL`, `REAL`, `DOUBLE`, `INTERVAL_DAY_TO_SECOND` 또는 `INTERVAL_YEAR_TO_MONTH` 같은 데이터 형식으로 캐스팅해야 합니다.

지리 공간 함수 변경 사항

지리 공간 함수의 변경 사항은 다음과 같습니다.

- 함수 이름 변경 - 일부 함수 이름이 변경되었습니다. 자세한 내용은 Athena 엔진 버전 2의 지리 공간 함수 이름 변경 단원을 참조하세요.
- VARBINARY 입력 - 더 이상 지리 공간 함수에 대한 입력에 `VARBINARY` 형식이 직접적으로 지원되지 않습니다. 예를 들어 지오메트리의 면적을 직접 계산하려면 지오메트리가 `VARCHAR` 또는 `GEOMETRY` 형식의 입력이어야 합니다. 해결 방법은 다음 예제와 같이 변환 함수를 사용하는 것입니다.
  - `ST_area()`를 사용해 Well-Known Binary(WKB) 형식의 `VARBINARY` 입력에 대해 면적을 계산하려면 먼저 입력을 `ST_GeomFromBinary()`로 전달합니다. 예를 들어 다음과 같습니다.

    ```
    ST_area(ST_GeomFromBinary(<wkb_varbinary_value>))
    ```

  - `ST_area()`를 사용해 기존 바이너리 형식의 `VARBINARY` 입력에 대해 면적을 계산하려면 먼저 동일한 입력을 `ST_GeomFromLegacyBinary()`로 전달합니다. 예를 들어 다음과 같습니다.

    ```
    ST_area(ST_GeomFromLegacyBinary(<legacy_varbinary_value>))
    ```

- ST_ExteriorRing() 및 ST_Polygon() – 이제 `ST_ExteriorRing()` 및 `ST_Polygon()`이 입력으로 다각형만 받아들입니다. 이전에는 이러한 함수가 다른 지오메트리를 잘못 받아들였습니다.
- ST_Distance() – SQL/MM 사양의 요구에 따라 이제 입력 중 하나가 빈 지오메트리인 경우 `ST_Distance()` 함수가 `NULL`을 반환합니다. 이전에는 `NaN`이 반환되었습니다.

ANSI SQL 규정 준수

다음 구문 및 동작 문제가 ANSI SQL 표준을 따르도록 수정되었습니다.

- Cast() 연산 - REAL 또는 DOUBLE에서 DECIMAL로의 Cast() 연산이 이제 SQL 표준을 준수합니다. 예를 들어 이전에는 `cast (double '10000000000000000000000000000000' as decimal(38))`이 `100000000000000005366162204393472`를 반환했지만 이제 `10000000000000000000000000000000`을 반환합니다.
- JOIN ... USING – 이제 `JOIN ... USING`이 표준 SQL 시맨틱을 준수합니다. 이전에는 `JOIN ... USING`이 열의 테이블 이름을 한정하도록 요구했으며, 두 테이블의 열이 출력에 표시되었습니다. 테이블 한정(qualification)은 이제 유효하지 않으며 열이 출력에서 한 번만 표시됩니다.
- ROW 유형 리터럴 제거 - `ROW<int, int>(1, 2)` ROW 유형 리터럴이 더 이상 지원되지 않습니다. 대신 `ROW(1 int, 2 int)` 구문을 사용합니다.
- 그룹화된 집계 시맨틱 - 그룹화된 집계에 등식 시맨틱이 아닌 `IS NOT DISTINCT FROM` 시맨틱을 사용합니다. 이제 `NaN` 부동 소수점 값을 그룹화할 때 그룹화된 집계가 올바른 결과를 반환하며 성능이 개선되었습니다. null을 포함한 맵, 목록 및 행 형식에 대한 그룹화가 지원됩니다.
- 따옴표가 있는 형식은 더 이상 허용되지 않음 - ANSI SQL 표준에 따라 데이터 형식을 더 이상 따옴표로 묶을 수 없습니다. 예를 들어 `SELECT "date" '2020-02-02'`는 더 이상 유효한 쿼리가 아닙니다. 대신 `SELECT date '2020-02-02'` 구문을 사용합니다.
- 익명 행 필드 액세스 - 더 이상 `[.field0, .field1, ...]` 구문을 사용하여 익명 행 필드에 액세스 할 수 없습니다.
- 복합 그룹화(Complex grouping) 연산 - 복합 그룹화 연산 `GROUPING SETS`, `CUBE`, `ROLLUP`이 입력 열로 구성된 표현식에 대해 그룹화를 지원하지 않습니다. 열 이름만 허용됩니다.

대체된 함수

다음 함수는 더 이상 지원되지 않으며 동일한 결과를 산출하는 구문으로 대체되었습니다.

- information_schema.__internal_partitions__ – Athena 엔진 버전 2에서는 `__internal_partitions__`를 더 이상 사용할 수 없습니다. 동등한 구문으로 `SELECT * FROM "<table_name>$partitions"` 또는 `SHOW PARTITIONS`를 사용하세요. 자세한 내용은 특정 테이블에 대한 파티션 나열 단원을 참조하세요.
- 대체된 지리 공간 함수 - 이름이 변경된 지리 공간 함수 목록은 Athena 엔진 버전 2의 지리 공간 함수 이름 변경을 참조하세요.

Limits

리소스 제한으로 인해 쿼리가 실패하지 않도록 Athena 엔진 버전 2에서는 다음의 제한이 도입되었습니다. 이러한 제한은 사용자가 구성할 수 없습니다.

- 결과 요소 수 - `min(col, n)`, `max(col, n)`, `min_by(col1, col2, n)`, `max_by(col1, col2, n)` 함수의 경우 결과 요소 수 n이 10,000개 이하로 제한됩니다.
- 그룹화 세트 - 그룹화 집합의 최대 슬라이스 수는 2048개입니다.
- 최대 텍스트 파일 줄 길이 - 텍스트 파일의 기본 최대 줄 길이는 200MB입니다.
- 시퀀스 함수 최대 결과 크기 - 시퀀스 함수의 최대 결과 크기는 50,000개의 항목입니다. 예를 들어, `SELECT sequence(0,45000,1)`은 성공하지만 `SELECT sequence(0,55000,1)`은 오류 메시지와 함께 실패합니다. 시퀀스 함수의 결과는 50,000개 항목 이하여야 합니다. 이 제한은 타임스탬프를 포함해 시퀀스 함수의 모든 입력 형식에 적용됩니다.

Athena에 대한 SQL 참조

Amazon Athena는 데이터 정의 언어(DDL) 및 데이터 조작 언어(DML) 설명, 함수, 연산자 및 데이터 형식의 하위 집합을 지원합니다. 일부 예외를 제외하고 Athena DDL은 HiveQL DDL을 기반으로 하고 Athena DML은 Trino를 기반으로 합니다. Athena 엔진 버전에 대한 자세한 내용은 Athena 엔진 버전 관리 단원을 참조하세요.

주제

- Amazon Athena의 데이터 형식
- DML 쿼리, 함수 및 연산자
- DDL 문
- Amazon Athena의 SQL 쿼리에 대한 고려 사항 및 제한 사항

Amazon Athena의 데이터 형식

`CREATE TABLE`을 실행할 때 각 열에 포함할 수 있는 열 이름과 데이터 형식을 지정합니다. 생성한 테이블은 AWS Glue Data Catalog에 저장됩니다.

다른 쿼리 엔진과의 상호 운용성을 높이기 위해 Athena는 `CREATE TABLE`과 같은 DDL 문에 Apache Hive 데이터 유형 이름을 사용합니다. `SELECT`, `CTAS`, `INSERT INTO` 등과 같은 DML 쿼리의 경우 Athena는 Trino 데이터 유형 이름을 사용합니다. 다음 표는 Athena에서 지원되는 데이터 유형을 보여줍니다. DDL 및 DML 유형이 이름, 가용성 또는 구문 측면에서 다른 경우 별도의 열에 표시됩니다.

DDL	DML	설명
BOOLEAN		값은 `true` 및 `false`입니다.

DDL	DML	설명
TINYINT		2의 보수 형식의 부호 있는 8비트 정수로, 최솟값은 -2^7, 최댓값은 2^7-1입니다.
SMALLINT		2의 보수 형식의 부호 있는 16비트 정수로, 최솟값은 -2^{15}, 최댓값은 $2^{15}-1$입니다.
INT, INTEGER		2의 보수 형식의 부호 있는 32비트 값으로, 최솟값은 -2^{31}, 최댓값은 $2^{31}-1$입니다.
BIGINT		2의 보수 형식의 부호 있는 64비트 정수로, 최솟값은 -2^{63}, 최댓값은 $2^{63}-1$입니다.
FLOAT	REAL	32비트 부호 포함 단정밀도 부동 소수점 숫자입니다. 범위는 1.40129846432481707e-45에서 3.40282346638528860e+38로, 양수 또는 음수입니다. 부동 소수점 산술에 대한 IEEE 표준(IEEE 754)을 따릅니다.
DOUBLE		64비트 부호 포함 배정밀도 부동 소수점 숫자입니다. 범위는 4.94065645841246544e-324d에서 1.79769313486231570e+308d로, 양수 또는 음수입니다. 부동 소수점 산술에 대한 IEEE 표준(IEEE 754)을 따릅니다.
DECIMAL(*precision* , *scale*)		*precision* 은 총 자릿수이고, *scale*(선택 사항)은 소수부의 자릿수이며, 기본값은 0입니다. 예를 들어 `decimal(11,5)` , `decimal(15)` 형식 정의를 사용합니다. *precision* 최대값은 38이고, *scale* 최대값은 38입니다.
CHAR, CHAR(*length*)		길이가 1~255자로 지정된 고정 길이 문자 데이터입니다(예: char(10)). *length*가 지정되면 읽을 때 문자열이 지정된 길이만큼 잘립니다. 기본 데이터 문자열이 더 긴 경우 기본 데이터 문자열은 변경되지 않습니다. 자세한 내용은 CHAR Hive 데이터 형식을 참조하세요.
STRING	VARCHAR	가변 길이 문자 데이터.

DDL	DML	설명
VARCHAR(*length*)		최대 읽기 길이를 가진 가변 길이 문자 데이터. 읽을 때 문자열이 지정된 길이만큼 잘립니다. 기본 데이터 문자열이 더 긴 경우 기본 데이터 문자열은 변경되지 않습니다.
BINARY	VARBINARY	가변 길이 이진 데이터.
TIME		밀리초 단위의 정밀도의 하루 중 시간.
사용할 수 없음	TIME(*precision*)	특정 정밀도의 하루 중 시간. `TIME(3)`은 `TIME`과 동일합니다.
사용할 수 없음	TIME WITH TIME ZONE	시간대의 하루 중 시간. 시간대는 UTC 기준 오프셋으로 지정되어야 합니다.
날짜		년, 월, 일 단위로 구성된 달력 날짜.
TIMESTAMP	TIMESTAMP, TIMESTAMP WITHOUT TIME ZONE	밀리초 단위의 정밀도의 달력 날짜와 하루 중 시간.
사용할 수 없음	TIMESTAMP(*precision*), TIMESTAMP(*precision*) WITHOUT TIME ZONE	특정 정밀도의 달력 날짜 및 하루 중 시간. `TIMESTAMP(3)` 은 `TIMESTAMP` 과 동일합니다.
사용할 수 없음	TIMESTAMP(시간대 사용)	시간대의 달력 날짜 및 하루 중 시간. 시간대는 UTC로부터의 오프셋, IANA 시간대 이름 또는 UTC, UT, Z 또는 GMT를 사용하여 지정될 수 있습니다.
사용할 수 없음	TIMESTAMP(*precision*) WITH TIME ZONE	시간대에서 특정 정밀도의 달력 날짜 및 하루 중 시간입니다.

DDL	DML	설명
사용할 수 없음	INTERVAL YEAR TO MONTH	하나 이상의 달 간격
사용할 수 없음	INTERVAL DAY TO SECOND	하나 이상의 초, 분, 시간 또는 일 간격
ARRAY<*element_type* >	ARRAY[*element_type*]	값으로 구성된 배열. 모든 값은 동일한 유형이어야 합니다.
MAP<*key_type*, *value_type* >	MAP(*key_type*, *value_type*)	키로 값을 조회할 수 있는 맵입니다. 모든 키의 값이 같아야 하고, 모든 값의 값이 같아야 합니다.
STRUCT<*field_name_1* :*field_type_1* , *field_name_2* :*field_type_2* , ...>	ROW(*field_name_1 field_type_1* , *field_name_2 field_type_2* , ...)	이름이 지정된 필드와 값이 포함된 데이터 구조입니다.
사용할 수 없음	JSON	JSON 값 유형은 JSON 객체, JSON 배열, JSON 번호, JSON 문자열, `true`, `false` 또는 `null`가 될 수 있습니다.
사용할 수 없음	UUID	UUID(범용 고유 식별자).
사용할 수 없음	IPADDRESS	IPv4 주소 또는 IPv6 주소.
사용할 수 없음	HyperLogLog	이러한 데이터 유형은 대략적인 함수 내부를 지원합니다. 각 유형에 대한 자세한 내용은 Trino 설명서의 해당 항목을 참조하세요.
	P4HyperLogLog	
	SetDigest	
	QDigest	
	TDigest	

데이터 유형 예제

다음 표는 DML 데이터 유형에 대한 예제 리터럴을 보여줍니다.

데이터 유형	예제
BOOLEAN	`true` `false`
TINYINT	`TINYINT '123'`
SMALLINT	`SMALLINT '123'`
INT, INTEGER	`123456790`
BIGINT	`BIGINT '1234567890'` `2147483648`
REAL	`'123456.78'`
DOUBLE	`1.234`
DECIMAL(*precision* , *scale*)	`DECIMAL '123.456'`
CHAR, CHAR(*length*)	`CHAR 'hello world'`, `CHAR 'hello ''world''!'`
VARCHAR, VARCHAR(*length*)	`VARCHAR 'hello world'`, `VARCHAR 'hello ''world''!'`
VARBINARY	`X'00 01 02'`
TIME, TIME(*precision*)	`TIME '10:11:12'` , `TIME '10:11:12.345'`
TIME WITH TIME ZONE	`TIME '10:11:12.345 -06:00'`
날짜	`DATE '2024-03-25'`
TIMESTAMP, TIMESTAMP WITHOUT TIME ZONE,	`TIMESTAMP '2024-03-25 11:12:13'` , `TIMESTAMP '2024-03-25 11:12:13.456'`

데이터 유형	예제
TIMESTAMP(*precision*), TIMESTAMP(*precision*) WITHOUT TIME ZONE	
TIMESTAMP WITH TIME ZONE, TIMESTAMP (*precision*) WITH TIME ZONE	`TIMESTAMP '2024-03-25 11:12:13.456 Europe/Berlin'`
INTERVAL YEAR TO MONTH	`INTERVAL '3' MONTH`
INTERVAL DAY TO SECOND	`INTERVAL '2' DAY`
ARRAY[*element_type*]	`ARRAY['one', 'two', 'three']`
MAP(*key_type*, *value_type*)	`MAP(ARRAY['one', 'two', 'three'], ARRAY[1, 2, 3])` 맵은 키 배열과 값 배열로 생성됩니다.

ROW(*field_name_1 field_type_1* , *field_name_2 field_type_2* , ...)	`ROW('one', 'two', 'three')`
	
	이 방법으로 생성된 행에는 열 이름이 없습니다. 열 이름을 추가하려면 다음 예제와 같이 CAST를 사용할 수 있습니다.
	
	`CAST(ROW(1, 2, 3) AS ROW(one INT, two INT, three INT))`

JSON	`JSON '{"one":1, "two": 2, "three": 3}'`
UUID	`UUID '12345678-90ab-cdef-1234-567890abcdef'`

IPADDRESS	`IPADDRESS '10.0.0.1'`
	
	`IPADDRESS '2001:db8::1'`

데이터 유형에 대한 고려 사항

크기 제한

크기 제한이 지정되지 않은 데이터 유형의 경우 한 행의 모든 데이터에 대한 실제 제한은 32MB라는 점에 유의하세요. 자세한 설명은 Amazon Athena의 SQL 쿼리에 대한 고려 사항 및 제한 사항에서 Row or column size limitation 섹션을 참조하십시오.

CHAR 및 VARCHAR

CHAR(*n*) 값에는 항상 *n*개의 문자가 포함됩니다. 예를 들어, 'abc'를 CHAR(7)로 변환하면 4개의 후행 공백이 추가됩니다.

CHAR 값 비교에는 선행 공백과 후행 공백이 포함됩니다.

CHAR 또는 VARCHAR에 대해 길이가 지정되면 읽을 때 문자열이 지정된 길이만큼 잘립니다. 기본 데이터 문자열이 더 긴 경우 기본 데이터 문자열은 변경되지 않습니다.

CHAR 또는 VARCHAR에서 작은따옴표를 이스케이프하려면 추가로 작은 따옴표를 사용합니다.

문자열이 아닌 데이터 유형을 DML 쿼리의 문자열로 변환하려면 VARCHAR 데이터 유형으로 캐스팅합니다.

substr 함수를 사용하여 CHAR 데이터 유형에서 지정된 길이의 하위 문자열을 반환하려면 먼저 CHAR 값을 VARCHAR로 캐스팅해야 합니다. 다음 예제에서 col1은 CHAR 데이터 유형을 사용합니다.

```
substr(CAST(col1 AS VARCHAR), 1, 4)
```

DECIMAL

특정 10진수 값이 있는 행을 선택하는 경우와 같이 DECIMAL 쿼리에서 10진수 값을 리터럴로 지정하려면 다음 예제와 같이 SELECT 유형을 지정하고 쿼리에서 10진수 값을 작은따옴표로 묶인 리터럴로 나열합니다.

```
SELECT * FROM my_table
WHERE decimal_value = DECIMAL '0.12'
```

```
SELECT DECIMAL '44.6' + DECIMAL '77.2'
```

타임스탬프 데이터 작업

이 섹션에서는 Athena에서 타임스탬프 데이터를 작업할 때 고려해야 할 몇 가지 사항을 설명합니다.

> **Note**
>
> Athena 엔진 버전 2와 Athena 엔진 버전 3 사이에서 타임스탬프 처리가 조금 변경되었습니다. Athena 엔진 버전 3에서 발생할 수 있는 타임스탬프 관련 오류와 권장 솔루션은 Athena 엔진 버전 3 참조의 타임스탬프 변경 사항 섹션을 참조하세요.

Amazon S3 객체에 타임스탬프 데이터를 쓰기 위한 형식

Amazon S3 객체에 타임스탬프 데이터를 써야 하는 형식은 사용하는 SerDe 라이브러리와 열 데이터 형식 모두에 따라 달라집니다.

- 테이블 열 형식이 `DATE`인 경우 Athena는 데이터의 해당 열 또는 속성을 ISO 형식 `YYYY-MM-DD`의 문자열이거나 Parquet 또는 ORC와 같은 기본 제공 날짜 유형으로 예상합니다.
- 테이블 열 형식이 `TIME`인 경우 Athena는 데이터의 해당 열 또는 속성을 ISO 형식 `HH:MM:SS`의 문자열이거나 Parquet 또는 ORC와 같은 기본 제공 시간 유형으로 예상합니다.
- 테이블 열 형식이 `TIMESTAMP`인 경우 Athena는 데이터의 해당 열 또는 속성을 `YYYY-MM-DD HH:MM:SS.SSS` 형식의 문자열(날짜 및 시간 사이의 공백에 주의)이거나 Parquet, ORC 또는 Ion과 같은 기본 제공 시간 유형으로 예상합니다.

> **Note**
>
> OpenCSVSerDe 타임스탬프는 예외이며, 밀리초 분해능의 UNIX 에포크로 인코딩해야 합니다.

시간을 기준으로 파티셔닝된 데이터가 레코드의 타임스탬프 필드와 일치하는지 확인

데이터 생성자는 파티션 값이 파티션 내 데이터에 맞게 조정되었는지 확인해야 합니다. 예를 들어 데이터에 `timestamp` 속성이 있고 Firehose를 사용하여 Amazon S3로 데이터를 로드하는 경우 Firehose의 기본 파티셔닝은 12시간 형식의 시간에 기반하므로 동적 파티셔닝을 사용해야 합니다.

파티션 키의 데이터 형식으로 문자열 사용

성능상의 이유로 파티션 키의 데이터 유형으로 `STRING`을 사용하는 것이 좋습니다. `DATE` 유형을 사용할 때 Athena는 `YYYY-MM-DD` 형식의 파티션 값을 날짜로 인식하지만 이 경우 성능이 저하될 수 있습니다. 이러한 이유로 대신 파티션 키에 `STRING` 데이터 형식을 사용하는 것이 좋습니다.

시간을 기준으로 파티셔닝된 타임스탬프 필드에 대한 쿼리를 작성하는 방법

시간을 기준으로 파티셔닝된 타임스탬프 필드에 대한 쿼리를 작성하는 방법은 쿼리하려는 테이블 유형에 따라 달라집니다.

Hive 테이블

Athena에서 가장 흔히 사용되는 Hive 테이블의 경우 쿼리 엔진은 열과 파티션 키 사이의 관계를 알지 못합니다. 따라서 항상 쿼리에 열과 파티션 키 모두에 대한 조건자를 추가해야 합니다.

예를 들어 `event_time` 열과 `event_date` 파티션 키가 있고 23:00 및 03:00 사이에 이벤트를 쿼리하려고 한다고 가정합니다. 이 경우 다음 예제와 같이 열과 파티션 키 모두에 대한 조건자를 쿼리에 포함해야 합니다.

```
WHERE event_time BETWEEN start_time AND end_time
  AND event_date BETWEEN start_time_date AND end_time_date
```

Iceberg 테이블

Iceberg 테이블을 사용하면 쿼리를 단순화하는 계산된 파티션 값을 사용할 수 있습니다. 예를 들어 Iceberg 테이블이 다음과 같은 `PARTITIONED BY` 절을 사용하여 생성되었다고 가정합니다.

```
PARTITIONED BY (event_date month(event_time))
```

이 경우 쿼리 엔진은 `event_time` 조건자 값을 기반으로 파티션을 자동으로 정리합니다. 따라서 다음 예제와 같이 쿼리에서는 `event_time`에 대한 조건자만 지정하면 됩니다.

```
WHERE event_time BETWEEN start_time AND end_time
```

자세한 내용은 Iceberg 테이블 생성 단원을 참조하십시오.

DML 쿼리, 함수 및 연산자

Athena DML 쿼리 엔진에서는 일반적으로 Trino 및 Presto 구문을 지원하며 자체 개선 사항을 추가합니다. Trino 또는 Presto의 일부 기능은 Athena에서 지원되지 않습니다. 자세한 내용은 이 단원의 특정 문에 대한 주제 및 고려 사항 및 제한을 참조하세요. 함수에 대한 자세한 내용은 Amazon Athena의 함수 섹션을 참조하세요. Athena 엔진 버전에 대한 자세한 내용은 Athena 엔진 버전 관리 단원을 참조하세요.

DDL 문에 대한 자세한 내용은 DDL 문 단원을 참조하세요. 지원되지 않는 DDL 문의 목록은 지원되지 않는 DDL 단원을 참조하세요.

SELECT

0개 이상의 테이블에서 데이터 행을 검색합니다.

> **Note**
>
> 이 주제에서는 참조할 수 있는 요약 정보를 제공합니다. SELECT 및 SQL 언어에 대한 포괄적인 정보는 이 설명서에서 다루지 않습니다. Athena와 관련된 SQL 사용에 대한 자세한 내용은 Amazon Athena의 SQL 쿼리에 대한 고려 사항 및 제한 사항 및 Amazon Athena를 사용하여 SQL 쿼리 실행 단원을 참조하세요. 데이터베이스 생성, 테이블 생성 및 Athena의 테이블에서 SELECT 쿼리 실행에 대한 예제는 시작하기 항목을 참조하세요.

시놉시스

```
[ WITH with_query [, ...] ]
SELECT [ ALL | DISTINCT ] select_expression [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[ HAVING condition ]
[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
[ ORDER BY expression [ ASC | DESC ] [ NULLS FIRST | NULLS LAST] [, ...] ]
[ OFFSET count [ ROW | ROWS ] ]
[ LIMIT [ count | ALL ] ]
```

> **Note**
>
> SQL SELECT 문의 예약어는 큰따옴표로 묶어야 합니다. 자세한 내용은 SQL SELECT 문의 예약어 목록 단원을 참조하십시오.

파라미터

[WITH with_query [, ...]]

WITH를 사용하여 중첩 쿼리를 평면화하거나 하위 쿼리를 단순화할 수 있습니다.

Athena 엔진 버전 3부터 WITH 절을 사용하여 재귀 쿼리를 생성하는 작업이 지원됩니다. 최대 재귀 깊이는 10입니다.

WITH 절은 쿼리에서 SELECT 목록에 선행하고 SELECT 쿼리 내에서 사용할 하나 이상의 하위 쿼리를 정의합니다.

각각의 하위 쿼리는 FROM 절에서 참조할 수 있는 뷰 정의와 비슷한 임시 테이블을 정의합니다. 이러한 테이블은 쿼리를 실행할 때만 사용됩니다.

with_query 구문은 다음과 같습니다.

```
subquery_table_name [ ( column_name [, ...] ) ] AS (subquery)
```

위치:

- subquery_table_name은 WITH 절 하위 쿼리의 결과를 정의하는 임시 테이블의 고유한 이름입니다. 각각의 subquery에는 FROM 절에서 참조될 수 있는 테이블 이름이 있어야 합니다.
- column_name [, ...]은 출력 열 이름의 선택적 목록입니다. 열 이름의 수는 subquery로 정의되는 열 개수보다 적거나 같아야 합니다.
- subquery는 임의의 쿼리 설명문입니다.

[ALL | DISTINCT] select_expression

select_expression는 선택하려는 행을 결정합니다. select_expression은 다음 형식 중 하나를 사용할 수 있습니다.

```
expression [ [ AS ] column_alias ] [, ...]
```

```
row_expression.* [ AS ( column_alias [, ...] ) ]
```

```
relation.*
```

```
*
```

- expression [[AS] column_alias] 구문은 출력 열을 지정합니다. 선택적 [AS] column_alias 구문은 출력에서 열에 사용할 사용자 지정 제목 이름을 지정합니다.
- row_expression.* [AS (column_alias [, ...])]의 경우에, row_expression은 임의의 데이터 유형 ROW의 표현식입니다. 행의 필드는 결과에 포함할 출력 열을 정의합니다.
- relation.*의 경우, relation의 열이 결과에 포함됩니다. 이 구문에서는 열 별칭을 사용할 수 없습니다.

- 별표 *는 모든 열이 결과 집합에 포함되도록 지정합니다.
- 결과 집합에서 열의 순서는 select 표현식이 지정한 순서와 동일합니다. select 표현식이 여러 열을 반환하는 경우 열 순서는 소스 관계 또는 행 유형 표현식에 사용된 순서를 따릅니다.
- 열 별칭을 지정하면 별칭이 기존 열 또는 행 필드 이름보다 우선합니다. select 표현식에 열 이름이 없는 경우 인덱스가 0인 익명 열 이름(_col0, _col1, _col2, ...)이 출력에 표시됩니다.
- 기본값은 ALL입니다. ALL을 사용하면 마치 생략된 것처럼 처리됩니다. 모든 열에 대한 모든 행이 선택되고 중복이 유지됩니다.
- 열에 중복된 값이 포함되어 있는 경우 고유 값만 반환하려면 DISTINCT를 사용합니다.

FROM from_item [, ...]

아래에 설명된 대로 from_item이 뷰, 조인 생성 또는 하위 쿼리가 될 수 있는 쿼리 입력을 나타냅니다.

from_item은 다음 중 하나일 수 있습니다.

- table_name [[AS] alias [(column_alias [, ...])]]

  table_name이 행을 선택할 대상 테이블의 이름인 경우 alias는 SELECT 설명의 출력을 제공할 이름이고 column_alias는 지정된 alias에 대한 열을 정의합니다.

-또는-

- join_type from_item [ON join_condition | USING (join_column [, ...])]

  여기에서 join_type은 다음 중 하나입니다.

  - [INNER] JOIN
  - LEFT [OUTER] JOIN
  - RIGHT [OUTER] JOIN
  - FULL [OUTER] JOIN
  - CROSS JOIN
  - ON join_condition | USING (join_column [, ...]) join_condition을 사용하면 여러 테이블의 조인 키에 열 이름을 지정할 수 있으며 join_column을 사용하려면 join_column가 두 테이블에 모두 존재해야 합니다.

[WHERE condition]

사용자가 지정한 condition에 따라 결과를 필터링합니다. 여기서 condition에는 일반적으로 다음과 같은 구문이 있습니다.

```
column_name operator value [[[AND | OR] column_name operator value] ...]
```

*###*는 비교 연산자(=, >, <, >=, <=, <>, !=) 중 하나일 수 있습니다.

다음 하위 쿼리 표현식은 WHERE절에서 사용할 수도 있습니다.

- [NOT] BETWEEN *integer_A* AND *integer_B* – 다음 예시처럼 두 정수 사이의 범위를 지정합니다. 열 데이터 형식이 varchar인 경우 먼저 열을 정수로 변환해야 합니다.

```
SELECT DISTINCT processid FROM "webdata"."impressions"
WHERE cast(processid as int) BETWEEN 1500 and 1800
ORDER BY processid
```

- [NOT] LIKE *value* - 지정된 패턴을 검색합니다. 다음 예시와 같이 백분율 기호(%)를 와일드카드 문자로 사용할 수 있습니다.

```
SELECT * FROM "webdata"."impressions"
WHERE referrer LIKE '%.org'
```

- [NOT] IN (*value*[, *value*[, ...]) - 다음 예시와 같이 열에 사용 가능한 값 목록을 지정합니다.

```
SELECT * FROM "webdata"."impressions"
WHERE referrer IN ('example.com','example.net','example.org')
```

[GROUP BY [ALL | DISTINCT] grouping_expressions [, ...]]

SELECT 설명의 출력을 일치하는 값의 행으로 나눕니다.

ALL 및 DISTINCT는 중복되는 그룹화 집합이 각각 다른 출력 행을 생성할지 결정합니다. 생략된 경우, ALL로 간주됩니다.

grouping_expressions를 사용하면 복합 그룹화 작업을 수행할 수 있습니다. 복합 그룹화(complex grouping) 연산을 수행하면 하나의 쿼리에 여러 집합을 집계해야 하는 분석을 수행할 수 있습니다.

grouping_expressions 요소는 다음과 입력 열에서 수행되는 모든 함수(예: SUM, AVG, 또는 COUNT)일 수 있습니다.

GROUP BY 식은 SELECT 설명 출력에 나타나지 않는 입력 열 이름별로 출력을 그룹화할 수 있습니다.

모든 출력 식은 집계 함수이거나 GROUP BY 절에 있는 열이어야 합니다.

단일 쿼리를 사용하여 여러 열 집합을 집계해야 하는 분석을 수행할 수 있습니다.

Athena는 GROUPING SETS, CUBE, ROLLUP을 사용한 복합적 집계를 지원합니다. GROUP BY GROUPING SETS은 그룹화할 여러 열 목록을 지정합니다. GROUP BY CUBE은 주어진 열 집합에 대해 가능한 모든 그룹화 집합을 생성합니다. GROUP BY ROLLUP은 주어진 열 집합에 대해 가능한 모든 소계를 생성합니다. 복합 그룹화 연산은 입력 열로 구성된 표현식에 대해 그룹화를 지원하지 않습니다. 열 이름만 허용됩니다.

흔히 UNION ALL을 사용해도 이러한 GROUP BY 작업과 동일한 결과를 얻을 수 있지만 GROUP BY를 사용하는 쿼리는 데이터를 한 번만 읽는 이점이 있는 데 비해 UNION ALL은 기본 데이터를 세 번 읽으며, 데이터 원본이 변경될 경우 일관성 없는 결과를 생성할 수 있습니다.

[HAVING condition]

집계 함수 및 GROUP BY 절과 함께 사용됩니다. condition을 만족하지 않는 그룹을 제거하여 어떤 그룹을 선택할지 제어합니다. 이 필터링은 그룹과 집계 처리가 완료된 후 적용됩니다.

[{ UNION | INTERSECT | EXCEPT } [ALL | DISTINCT] union_query]]

UNION, INTERSECT, EXCEPT는 둘 이상의 SELECT 문 결과를 하나의 쿼리로 결합합니다. ALL 또는 DISTINCT는 최종 결과 집합에 포함된 열의 고유성을 제어합니다.

UNION은 첫 번째 쿼리의 결과 행을 두 번째 쿼리의 결과 행과 결합합니다. 중복된 항목을 제거하기 위해 UNION은 메모리를 소비하는 해시 테이블을 빌드합니다. 쿼리에서 중복 항목을 제거할 필요가 없는 경우 성능 향상을 위해 UNION ALL을 사용하는 것이 좋습니다. 다중 UNION 절은 괄호를 사용하여 처리 순서를 명시적으로 정의하지 않는 한 왼쪽에서 오른쪽으로 처리됩니다.

INTERSECT는 첫 번째 쿼리와 두 번째 쿼리의 결과에 모두 존재하는 행만 반환합니다.

EXCEPT는 두 번째 쿼리에서 찾은 행을 제외한 첫 번째 쿼리의 결과 행을 반환합니다.

ALL은 동일한 행이라 하더라도 모든 행을 포함시킵니다.

DISTINCT는 결합된 결과 집합에 고유한 행만 포함시킵니다.

[ORDER BY expression [ASC | DESC] [NULLS FIRST | NULLS LAST] [, ...]]

하나 이상의 출력 expression으로 결과 집합을 정렬합니다.

절에 여러 식이 포함되어 있으면 결과 집합은 첫 번째 expression에 따라 정렬됩니다. 그리고 첫 번째 식에서 일치하는 값이 있는 행에 두 번째 expression이 적용되고, 세 번째도 동일하게 적용됩니다.

각 expression은 SELECT의 출력 열을 지정하거나 위치별로 출력 열의 서수를 1부터 지정할 수 있습니다.

ORDER BY는 GROUP BY 또는 HAVING 절 이후 마지막 단계로 평가됩니다. ASC 및 DESC는 결과를 오름차순이나 내림차순으로 정렬하도록 결정합니다.

오름차순 또는 내림차순 정렬 순서와 상관없이 기본 null 순서는 NULLS LAST입니다.

[OFFSET count [ROW | ROWS]]

OFFSET 절을 사용하여 결과 집합에서 여러 개의 선행 행을 삭제합니다. ORDER BY 절이 있는 경우 OFFSET 절은 정렬된 결과 집합에 대해 평가되며, 건너뛴 행이 삭제된 후에도 집합이 정렬된 상태로 유지됩니다. 쿼리에 ORDER BY 절이 없는 경우 삭제되는 행은 임의로 선택됩니다. OFFSET에 의해 지정된 수가 결과 집합의 크기와 같거나 더 크면 최종 결과는 비어 있습니다.

LIMIT [count | ALL]

결과 집합의 행 수를 count로 제한합니다. LIMIT ALL은 LIMIT 절 생략과 동일합니다. 쿼리에 ORDER BY 절이 없는 경우 결과는 임의입니다.

TABLESAMPLE [BERNOULLI | SYSTEM] (백분율)

샘플링 방법을 기반으로 테이블에서 행을 선택하는 선택적 연산자입니다.

BERNOULLI는 percentage의 확률로 테이블 샘플에 포함할 각 행을 선택합니다. 테이블의 모든 물리적 블록이 스캔되고 샘플 percentage와 실행 시간에 계산된 임의 값 사이의 비교를 기반으로 특정 행을 건너뜁니다.

SYSTEM을 사용하면 테이블이 데이터의 논리적 세그먼트로 나뉘고 이 세분 수준에서 테이블이 샘플링됩니다.

특정 세그먼트의 모든 행이 선택되거나 샘플 percentage와(과) 실행 시간에 계산된 임의 값 사이의 비교 결과에 따라 세그먼트를 건너뜁니다. SYSTEM 샘플링은 커넥터에 따라 다릅니다. 이 메서드는 독립적인 샘플링 확률을 보장하지 않습니다.

[UNNEST (array_or_map) [WITH ORDINALITY]]

배열 또는 맵을 관계로 확장합니다. 배열은 하나의 열로 확장됩니다. 맵은 두 개의 열(키, 값)로 확장됩니다.

복수의 인수로 UNNEST를 사용할 수 있습니다. 이 경우 가장 큰 카디널리티 인수와 동일한 개수의 행이 있는 여러 열로 확장됩니다.

다른 열은 null로 채워집니다.

WITH ORDINALITY 절은 끝에 순서 열을 추가합니다.

UNNEST는 일반적으로 JOIN과 함께 사용되며 JOIN 왼쪽의 관계에서 열을 참조할 수 있습니다.

Amazon S3의 소스 데이터에 대한 파일 위치 가져오기

테이블 행의 데이터에 대한 Amazon S3 파일 위치를 보려면 다음 예시처럼 SELECT 쿼리에 "$path"를 사용할 수 있습니다.

```
SELECT "$path" FROM "my_database"."my_table" WHERE year=2019;
```

이 쿼리는 다음과 같은 결과를 반환합니다.

```
s3://DOC-EXAMPLE-BUCKET/datasets_mytable/year=2019/data_file1.json
```

테이블의 데이터에 대한 S3 파일 이름 경로를 정렬된 고유 목록으로 반환하려면 다음 예시와 같이 SELECT DISTINCT 및 ORDER BY를 사용합니다.

```
SELECT DISTINCT "$path" AS data_source_file
FROM sampledb.elb_logs
ORDER By data_source_file ASC
```

경로가 없는 파일 이름만 반환하려면 다음 예시와 같이 regexp_extract 함수에 "$path"를 파라미터로 전달합니다.

```
SELECT DISTINCT regexp_extract("$path", '[^/]+$') AS data_source_file
FROM sampledb.elb_logs
ORDER By data_source_file ASC
```

특정 파일의 데이터를 반환하려면 다음 예시와 같이 WHERE 절에 파일을 지정합니다.

```
SELECT *,"$path" FROM my_database.my_table WHERE "$path" = 's3://DOC-EXAMPLE-BUCKET/
my_table/my_partition/file-01.csv'
```

자세한 내용과 예시는 지식 센터 문서 Athena 테이블의 행에 대한 Amazon S3 원본 파일을 확인하려면 어떻게 해야 합니까?를 참조하세요.

> **Note**
>
> Athena에서는 Hive 또는 Iceberg의 숨겨진 메타데이터 열 `$bucket`, `$file_modified_time`, `$file_size` 및 `$partition`을 보기에서 지원하지 않습니다.

작은따옴표의 이스케이프 처리

작은따옴표를 이스케이프 처리하려면 다음 예제와 같이 작은따옴표 앞에 다른 작은따옴표를 추가합니다. 이를 큰따옴표와 혼동하지 마십시오.

```
Select 'O''Reilly'
```

결과

```
O'Reilly
```

추가적인 리소스

Athena에서 SELECT 문 사용에 관한 자세한 내용은 다음 리소스를 참조하세요.

이에 대한 자세한 내용은	다음을 참조:
Athena에서 쿼리 실행	Amazon Athena를 사용하여 SQL 쿼리 실행
SELECT를 사용해 테이블 생성	쿼리 결과에서 테이블 생성(CTAS)
SELECT 쿼리의 데이터를 다른 테이블에 삽입	INSERT INTO
SELECT 문에서 내장 함수 사용	Amazon Athena의 함수
SELECT 문에서 사용자가 정의한 함수 사용	사용자 정의 함수를 사용한 쿼리
데이터 카탈로그 메타데이터 쿼리	AWS Glue Data Catalog 쿼리

INSERT INTO

SELECT 원본 테이블에서 실행되는 쿼리 설명을 기반으로 하거나 해당 설명의 일부로 제공된 VALUES 세트를 기반으로 하는 대상 테이블에 새 행을 삽입합니다. 원본 테이블이 CSV 또는 JSON과 같은 한

가지 형식으로 된 기본 데이터를 기반으로 하고 대상 테이블이 Parquet 또는 ORC와 같은 다른 형식을 기반으로 할 때 INSERT INTO 쿼리를 사용하여 선택된 데이터를 대상 테이블의 형식에 맞춰 변환할 수 있습니다.

고려 사항 및 제한

Athena에서 INSERT 쿼리를 사용할 때는 다음 사항을 고려하세요.

- Amazon S3에서 암호화된 기본 데이터가 포함된 테이블에서 INSERT 쿼리를 실행할 때 INSERT 쿼리가 작성하는 출력 파일은 기본적으로 암호화되지 않습니다. 암호화된 데이터가 포함된 테이블에 삽입하는 경우 INSERT 쿼리 결과를 암호화할 것을 권장합니다.

  콘솔을 사용한 쿼리 결과 암호화에 대한 자세한 내용은 Amazon S3에 저장된 Athena 쿼리 결과 암호화 단원을 참조하세요. AWS CLI 또는 Athena API에서 암호화를 사용하도록 설정하려면 StartQueryExecution 작업의 EncryptionConfiguration 속성을 사용해 필요에 따라 Amazon S3 암호화 옵션을 지정합니다.
- INSERT INTO 문의 경우 예상 버킷 소유자 설정이 Amazon S3의 대상 테이블 위치에 적용되지 않습니다. 예상 버킷 소유자 설정은 Athena 쿼리 결과에 대해 지정한 Amazon S3 출력 위치에만 적용됩니다. 자세한 내용은 Athena 콘솔을 사용하여 쿼리 결과 위치 지정 단원을 참조하십시오.
- ACID 준수 INSERT INTO 문은 Iceberg 테이블 데이터 업데이트의 INSERT INTO 단원을 참조하세요.

지원되는 형식 및 SerDes

다음 형식 및 SerDes를 사용하여 데이터에서 생성된 테이블에서 INSERT 쿼리를 실행할 수 있습니다.

데이터 형식	SerDe
Avro	org.apache.hadoop.hive.serde2.avro.AvroSerDe
Ion	com.amazon.ionhiveserde.IonHiveSerDe
JSON	org.apache.hive.hcatalog.data.JsonSerDe
ORC	org.apache.hadoop.hive.ql.io.orc.OrcSerde
PARQUET	org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe

데이터 형식	SerDe
텍스트 파일	org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe Note CSV, TSV 및 사용자 지정 구분 기호로 구분된 파일이 지원됩니다.

버킷팅된 테이블은 지원되지 않음

`INSERT INTO`은(는) 버킷팅된 테이블에서 지원되지 않습니다. 자세한 내용은 Athena에서 파티셔닝 및 버킷팅 단원을 참조하십시오.

연합 쿼리가 지원되지 않음

연합 쿼리에는 `INSERT INTO`가 지원되지 않습니다. 이를 실행하려고 하면 이 작업은 현재 외부 카탈로그에서 지원되지 않습니다(This operation is currently not supported for external catalogs)라는 오류 메시지가 표시될 수 있습니다. 연합 쿼리에 대한 자세한 내용은 Amazon Athena 연합 쿼리 사용 섹션을 참조하세요.

분할

`INSERT INTO` 또는 `CREATE TABLE AS SELECT` 쿼리에 분할을 사용할 때 이 단원의 요점들을 고려하세요.

Limits

이 `INSERT INTO` 문은 대상 테이블에 최대 100개의 파티션 쓰기를 지원합니다. 100개 이상의 파티션이 있는 테이블에서 `SELECT` 절을 실행하면 `SELECT` 쿼리가 100개 이하의 파티션으로 제한되지 않는 한 쿼리가 실패합니다.

이 제한을 해결하는 방법에 대한 자세한 내용은 CTAS 및 INSERT INTO를 사용하여 100개 파티션 한도 해결 단원을 참조하세요.

열 정렬

`INSERT INTO` 또는 `CREATE TABLE AS SELECT` 문은 분할된 열이 `SELECT` 문에서 프로젝션된 열 목록의 마지막 열이 될 것으로 예상합니다.

소스 테이블이 분할되지 않았거나 대상 테이블과 달리 다른 열에 분할된 경우 INSERT INTO *destination_table* SELECT * FROM *source_table*와 같은 쿼리는 소스 테이블의 마지막 열 값을 대상 테이블의 파티션 열 값으로 간주합니다. 분할되지 않은 테이블에서 분할된 테이블을 만들려고 할 때는 이 점에 유의해야 합니다.

리소스

INSERT INTO와 파티셔닝 사용에 관한 자세한 내용은 다음 리소스를 참조하세요.

- 분할된 데이터를 분할된 테이블에 삽입하려면 CTAS 및 INSERT INTO를 사용하여 100개 파티션 한도 해결 단원을 참조하세요.
- 분할되지 않은 데이터를 분할된 테이블에 삽입하려면 ETL 및 데이터 분석에 CTAS 및 INSERT INTO 사용 단원을 참조하세요.

Amazon S3에 작성되는 파일

Athena는 Amazon S3의 소스 데이터 위치에 파일을 INSERT 명령의 결과로서 작성합니다. 각 INSERT 작업은 기존 파일에 추가하는 게 아니라 새 파일을 생성합니다. 파일 위치는 테이블의 구조와 SELECT 쿼리(있는 경우)에 따라 달라집니다. Athena는 각 INSERT 쿼리에 대해 데이터 매니페스트 파일을 생성합니다. 매니페스트는 쿼리가 작성한 파일을 추적합니다. 이는 Amazon S3의 Athena 쿼리 결과에 저장됩니다. 자세한 내용은 쿼리 출력 파일 식별 단원을 참조하십시오.

트랜잭션이 많은 업데이트 방지

INSERT INTO를 사용하여 Amazon S3의 테이블에 행을 추가할 때 Athena는 기존 파일을 다시 쓰거나 수정하지 않습니다. 대신 행을 하나 이상의 새 파일로 씁니다. 작은 파일이 많은 테이블은 쿼리 성능을 저하시키고, PutObject 및 GetObject와 같은 쓰기 및 읽기 작업을 수행하면 Amazon S3에서 비용이 증가하므로 INSERT INTO 사용 시에는 다음 사항을 고려하세요.

- 대량의 행 배치에서는 INSERT INTO 작업 실행 빈도를 줄이세요.
- 데이터 모으기 양이 많은 경우 Amazon Data Firehose와 같은 서비스를 사용해 보세요.
- INSERT INTO를 함께 사용하지 마세요. 대신 행을 큰 파일로 모아 Athena가 쿼리할 수 있도록 Amazon S3에 직접 업로드하세요.

분리된 파일 찾기

CTAS 또는 INSERT INTO 문이 실패하는 경우 분리된 데이터가 해당 데이터 위치에 남을 수 있고 후속 쿼리에서 이 데이터를 읽을 수 있습니다. 검사 또는 삭제할 분리된 파일을 찾으려면 Athena에서 제공

하는 데이터 매니페스트 파일을 사용하여 작성할 파일 목록을 추적할 수 있습니다. 자세한 내용은 쿼리 출력 파일 식별 및 DataManifestLocation를 참조하세요.

INSERT INTO...SELECT

쿼리가 하나의 테이블 source_table에서 실행되도록 지정하면 두 번째 테이블 destination_table에 삽입할 행이 결정됩니다. SELECT 쿼리가 source_table의 열을 지정하는 경우 해당 열은 destination_table의 열과 정확하게 일치해야 합니다.

SELECT 쿼리에 대한 자세한 내용은 SELECT 단원을 참조하세요.

시놉시스

```
INSERT INTO destination_table
SELECT select_query
FROM source_table_or_view
```

예제

vancouver_pageviews 테이블의 모든 행을 선택한 다음 canada_pageviews 테이블에 삽입합니다.

```
INSERT INTO canada_pageviews
SELECT *
FROM vancouver_pageviews;
```

2019-07-01에서 2019-07-31 사이의 값이 포함된 date 열이 있는 vancouver_pageviews 테이블의 행을 선택한 다음 canada_july_pageviews에 삽입합니다.

```
INSERT INTO canada_july_pageviews
SELECT *
FROM vancouver_pageviews
WHERE date
    BETWEEN date '2019-07-01'
        AND '2019-07-31';
```

country 열에 usa의 값이 포함된 행에서 cities_world 테이블의 city 및 state 열의 값을 선택한 다음 cities_usa 테이블의 city 및 state 열에 삽입합니다.

```
INSERT INTO cities_usa (city,state)
```

```
SELECT city,state
FROM cities_world
    WHERE country='usa'
```

INSERT INTO...VALUES

열 및 값을 지정하여 기존 테이블에 행을 삽입합니다. 지정된 열 및 관련 데이터 형식은 대상 테이블의 열 및 데이터 형식과 정확하게 일치해야 합니다.

> **Important**
>
> Athena는 각 INSERT 작업에 대해 파일을 생성하기 때문에 VALUES를 사용하여 행을 삽입하는 것은 권장하지 않습니다. 이로 인해 작은 파일이 많이 생성되어 테이블의 쿼리 성능이 저하될 수 있습니다. INSERT 쿼리가 생성하는 파일을 식별하려면 데이터 매니페스트 파일을 검사합니다. 자세한 내용은 쿼리 결과, 최근 쿼리, 출력 파일 작업 단원을 참조하십시오.

시놉시스

```
INSERT INTO destination_table [(col1,col2,...)]
VALUES (col1value,col2value,...)[,
       (col1value,col2value,...)][,
       ...]
```

예제

다음 예시에서 도시 테이블에는 id, city, state, state_motto의 세 가지 열이 있습니다. id 열은 INT 형식이고 기타 모든 열은 VARCHAR 형식입니다.

cities 테이블에 단일 행 하나를 삽입하고 모든 열 값을 지정합니다.

```
INSERT INTO cities
VALUES (1,'Lansing','MI','Si quaeris peninsulam amoenam circumspice')
```

cities 테이블에 행 두 개를 삽입합니다.

```
INSERT INTO cities
VALUES (1,'Lansing','MI','Si quaeris peninsulam amoenam circumspice'),
       (3,'Boise','ID','Esto perpetua')
```

DELETE

Apache Iceberg 테이블의 행을 삭제합니다. DELETE는 트랜잭션이며 Apache Iceberg 테이블에서만 지원됩니다.

시놉시스

Iceberg 테이블에서 행을 삭제하려면 다음 구문을 사용합니다.

```
DELETE FROM [db_name.]table_name [WHERE predicate]
```

자세한 내용과 예제는 Iceberg 테이블 데이터 업데이트의 DELETE 단원을 참조하세요.

UPDATE

Apache Iceberg 테이블의 행을 업데이트합니다. UPDATE는 트랜잭션이며 Apache Iceberg 테이블에서만 지원됩니다.

시놉시스

Iceberg 테이블의 행을 업데이트하려면 다음 구문을 사용합니다.

```
UPDATE [db_name.]table_name SET xx=yy[,...] [WHERE predicate]
```

자세한 내용과 예제는 Iceberg 테이블 데이터 업데이트의 UPDATE 단원을 참조하세요.

MERGE INTO

Apache Iceberg 테이블에 행을 조건부로 업데이트, 삭제 또는 삽입합니다. 단일 명령문으로 업데이트, 삭제 및 삽입 작업을 결합할 수 있습니다.

> **Note**
>
> MERGE INTO은 트랜잭션이며 Athena 엔진 버전 3의 Apache Iceberg 테이블에 대해서만 지원됩니다.

시놉시스

Iceberg 테이블에서 행을 조건부로 업데이트, 삭제 또는 삽입하려면 다음 구문을 사용합니다.

```
MERGE INTO target_table [ [ AS ]  target_alias ]
USING { source_table | query } [ [ AS ] source_alias ]
ON search_condition
when_clause [...]
```

*when_clause*는 다음 중 하나입니다.

```
WHEN MATCHED [ AND condition ]
    THEN DELETE
```

```
WHEN MATCHED [ AND condition ]
    THEN UPDATE SET ( column = expression [, ...] )
```

```
WHEN NOT MATCHED [ AND condition ]
    THEN INSERT (column_name[, column_name ...]) VALUES (expression, ...)
```

MERGE는 서로 다른 MATCHED 조건을 가진 임의 개수의 WHEN 절을 지원합니다. 조건 절은 MATCHED 상태와 일치 조건에 따라 선택된 첫 번째 WHEN 절에서 DELETE, UPDATE 또는 INSERT 작업을 실행합니다.

각 소스 행에 대해 WHEN 절이 순서대로 처리됩니다. 일치하는 첫 번째 WHEN 절만 실행됩니다. 후속 절은 무시됩니다. 하나의 대상 테이블 행이 둘 이상의 소스 행과 일치할 경우 사용자 오류가 발생합니다.

소스 행이 WHEN 절과 일치하지 않고 WHEN NOT MATCHED 절이 없는 경우 해당 소스 행은 무시됩니다.

UPDATE 작업이 있는 WHEN 절에서 열 값 표현식은 대상 또는 소스의 모든 필드를 참조할 수 있습니다. NOT MATCHED의 경우 INSERT 표현식은 소스의 모든 필드를 참조할 수 있습니다.

예

다음 예제에서는 첫 번째 테이블에 행이 없는 경우 두 번째 테이블의 행을 첫 번째 테이블에 병합합니다. VALUES 절에 나열된 열에 소스 테이블 별칭이 접두사로 사용되어야 합니다. INSERT 절에 나열된 대상 열에는 이 접두사를 사용해서는 안 됩니다.

```
MERGE INTO iceberg_table_sample as ice1
USING iceberg2_table_sample as ice2
ON ice1.col1 = ice2.col1
WHEN NOT MATCHED
THEN INSERT (col1)
      VALUES (ice2.col1)
```

더 많은 MERGE INTO 예제는 Iceberg 테이블 데이터 업데이트 섹션을 참조하세요.

OPTIMIZE

연결된 삭제 파일의 크기와 수에 따라 데이터 파일을 보다 최적화된 레이아웃으로 재작성하여 Apache Iceberg 테이블의 행을 최적화합니다.

> **Note**
>
> OPTIMIZE는 트랜잭션이며 Apache Iceberg 테이블에 대해서만 지원됩니다.

구문

다음 구문 요약은 Iceberg 테이블의 데이터 레이아웃을 최적화하는 방법을 보여줍니다.

```
OPTIMIZE [db_name.]table_name REWRITE DATA USING BIN_PACK
  [WHERE predicate]
```

> **Note**
>
> WHERE 절 ###에는 파티션 열만 사용할 수 있습니다. 파티션되지 않은 열을 지정하면 쿼리가 실패합니다.

압축 작업은 재작성 프로세스 중에 스캔된 데이터의 양에 따라 요금이 부과됩니다. REWRITE DATA 작업은 술어를 사용하여 일치하는 행이 포함된 파일을 선택합니다. 파일의 행이 술어와 일치하는 경우 최적화를 위해 파일이 선택됩니다. 따라서 압축 작업의 영향을 받는 파일 수를 제어하려면 WHERE 절을 지정합니다.

압축 속성 구성

압축을 위해 선택할 파일의 크기와 압축 후 결과 파일 크기를 제어하려면 테이블 속성 파라미터를 사용할 수 있습니다. ALTER TABLE SET PROPERTIES 명령을 사용하여 다음 테이블 속성을 구성할 수 있습니다.

추가적인 리소스

Iceberg 테이블 최적화

VACUUM

VACUUM 문은 더 이상 필요하지 않은 데이터 파일을 제거하여 Apache Iceberg 테이블에서 테이블 유지 관리를 수행합니다.

> **Note**
>
> VACUUM은 트랜잭션이며 Athena 엔진 버전 3의 Apache Iceberg 테이블에 대해서만 지원됩니다.

더 이상 관련 없는 데이터 파일을 제거하고 메타데이터 크기와 스토리지 사용을 줄이려면 Iceberg 테이블에서VACUUM 문을 실행하는 것이 좋습니다. 단, VACUUM 문은 Amazon S3에 API를 직접적으로 호출하므로 Amazon S3에 대한 관련 요청에 요금이 부과됩니다.

> **Warning**
>
> 스냅샷 만료 작업을 실행하면 더 이상 만료된 스냅샷으로 이동할 수 없습니다.

시놉시스

Iceberg 테이블에 더 이상 필요하지 않은 데이터 파일을 제거하려면 다음 구문을 사용합니다.

```
VACUUM [database_name.]target_table
```

이름이 밑줄로 시작하는 테이블(예: _mytable)에서 VACUUM을 실행하려면 다음 예제와 같이 테이블 이름을 백틱으로 묶습니다. 테이블 이름 앞에 데이터베이스 이름을 붙이는 경우 데이터베이스 이름을 백틱으로 묶지 마세요. 큰따옴표는 백틱 대신 사용할 수 없다는 점에 유의하세요.

이 동작은 특히 VACUUM에 발생합니다. CREATE 및 INSERT INTO 문에서는 밑줄로 시작하는 테이블 이름에 백틱을 사용하지 않아도 됩니다.

```
VACUUM `_mytable`
VACUUM my_database.`_mytable`
```

또한 VACUUM은 Iceberg 데이터가 Amazon S3 버킷이 아닌 Amazon S3 폴더에 있을 것으로 예상합니다. 예를 들어 Iceberg 데이터가 s3://DOC-EXAMPLE-BUCKET/myicebergfolder/가 아닌 s3://DOC-EXAMPLE-BUCKET/에 있는 경우 VACUUM 문이 실패하고 GENERIC_INTERNAL_ERROR: Path missing in file system location: s3://DOC-EXAMPLE-BUCKET라는 오류 메시지가 표시됩니다.

작업 수행됨

VACUUM은 다음 작업을 수행합니다.

- vacuum_max_snapshot_age_seconds 테이블 속성에 지정된 시간보다 오래된 스냅샷을 제거합니다. 기본적으로 이 속성은 432000초(5일)로 설정됩니다.
- 보존 기간 내에 있지 않고 vacuum_min_snapshots_to_keep 테이블 속성에 지정된 수를 초과하는 스냅샷을 제거합니다. 기본 값은 1입니다.

  CREATE TABLE 문에서 이러한 테이블 속성을 지정할 수 있습니다. 테이블이 생성된 이후에 ALTER TABLE SET PROPERTIES 문을 사용하여 테이블을 업데이트할 수 있습니다.
- 스냅샷 제거로 인해 연결할 수 없는 메타데이터 및 데이터 파일을 제거합니다. vacuum_max_metadata_files_to_keep 테이블 속성을 설정하여 보존할 이전 메타데이터 파일 수를 구성할 수 있습니다. 기본 값은 100입니다.
- vacuum_max_snapshot_age_seconds 테이블 속성에 지정된 시간보다 오래된 분리된 파일을 제거합니다. 분리된 파일은 테이블의 데이터 디렉터리에서 테이블 상태에 포함되지 않는 파일입니다.

Athena의 Apache Iceberg 테이블 생성 및 관리에 대한 자세한 내용은 Iceberg 테이블 생성 및 Iceberg 테이블 관리 단원을 참조하세요.

Athena에서 EXPLAIN 및 EXPLAIN ANALYZE 사용

EXPLAIN 문은 지정된 SQL 문의 논리적 또는 분산 실행 계획을 보여주거나 SQL 문의 유효성을 검사합니다. 결과를 텍스트 형식 또는 데이터 형식으로 출력하여 그래프로 렌더링할 수 있습니다.

> **Note**
>
> EXPLAIN 구문을 사용하지 않고도 쿼리에 대한 논리적 계획 및 분산된 계획의 그래픽 표현을 Athena 콘솔에서 볼 수 있습니다. 자세한 내용은 SQL 쿼리에 대한 실행 계획 보기 단원을 참조하십시오.

EXPLAIN ANALYZE 문은 지정된 SQL 문의 분산 실행 계획과 SQL 쿼리의 각 작업 계산 비용을 모두 표시합니다. 결과를 텍스트 또는 JSON 형식으로 출력할 수 있습니다.

고려 사항 및 제한

Athena의EXPLAIN 및 EXPLAIN ANALYZE 문에는 다음과 같은 제한이 있습니다.

- EXPLAIN 쿼리는 어떠한 데이터도 스캔하지 않으므로 Athena는 이에 대해 과금하지 않습니다. 그러나 EXPLAIN 쿼리는 AWS Glue를 호출하여 테이블 메타데이터를 검색하기 때문에 호출이 Glue의 프리 티어 한도를 초과할 경우 Glue에서 요금이 발생할 수 있습니다.
- EXPLAIN ANALYZE 쿼리가 실행되면 스캔 데이터를 수행하고 Athena가 스캔한 데이터 양에 대해 요금을 청구하기 때문입니다.
- Lake Formation에 정의된 행 또는 셀 필터링 정보와 쿼리 통계 정보는 EXPLAIN 및 EXPLAIN ANALYZE의 출력에 표시되지 않습니다.

EXPLAIN 구문

```
EXPLAIN [ ( option [, ...]) ] statement
```

*##*은 다음 중 하나일 수 있습니다.

```
FORMAT { TEXT | GRAPHVIZ | JSON }
TYPE { LOGICAL | DISTRIBUTED | VALIDATE | IO }
```

FORMAT 옵션이 지정되지 않은 경우 출력은 기본적으로 TEXT 형식입니다. IO 유형은 쿼리가 읽는 테이블 및 스키마에 대한 정보를 제공합니다. IO는 Athena 엔진 버전 2에서만 지원되며 JSON 형식으로만 반환될 수 있습니다.

EXPLAIN ANALYZE 구문

EXPLAIN에 포함된 출력 외에도 EXPLAIN ANALYZE 출력에는 CPU 사용량, 입력 행 수 및 출력 행 수와 같은 지정된 쿼리에 대한 런타임 통계도 포함됩니다.

```
EXPLAIN ANALYZE [ ( option [, ...]) ] statement
```

*##*은 다음 중 하나일 수 있습니다.

```
FORMAT { TEXT | JSON }
```

FORMAT 옵션이 지정되지 않은 경우 출력은 기본적으로 TEXT 형식입니다. EXPLAIN ANALYZE에 대한 모든 쿼리는 DISTRIBUTED이며, TYPE 옵션은 EXPLAIN ANALYZE에 사용할 수 없기 때문입니다.

*statement*는 다음 중 하나일 수 있습니다.

```
SELECT
CREATE TABLE AS SELECT
INSERT
UNLOAD
```

EXPLAIN 예제

EXPLAIN에 대한 다음 예는 더 간단한 예에서 더 복잡한 예로 진행됩니다.

EXPLAIN 예 1: EXPLAIN 문을 사용하여 텍스트 형식의 쿼리 계획 표시

다음 예에서 EXPLAIN은 Elastic Load Balancing 로그 기반 SELECT 쿼리에 대한 실행 계획을 보여줍니다. 형식은 기본적으로 텍스트 출력입니다.

```
EXPLAIN
SELECT
   request_timestamp,
   elb_name,
   request_ip
FROM sampledb.elb_logs;
```

결과

```
- Output[request_timestamp, elb_name, request_ip] => [[request_timestamp, elb_name,
 request_ip]]
    - RemoteExchange[GATHER] => [[request_timestamp, elb_name, request_ip]]
        - TableScan[awsdatacatalog:HiveTableHandle{schemaName=sampledb,
 tableName=elb_logs,
analyzePartitionValues=Optional.empty}] => [[request_timestamp, elb_name, request_ip]]
                LAYOUT: sampledb.elb_logs
                request_ip := request_ip:string:2:REGULAR
                request_timestamp := request_timestamp:string:0:REGULAR
                elb_name := elb_name:string:1:REGULAR
```

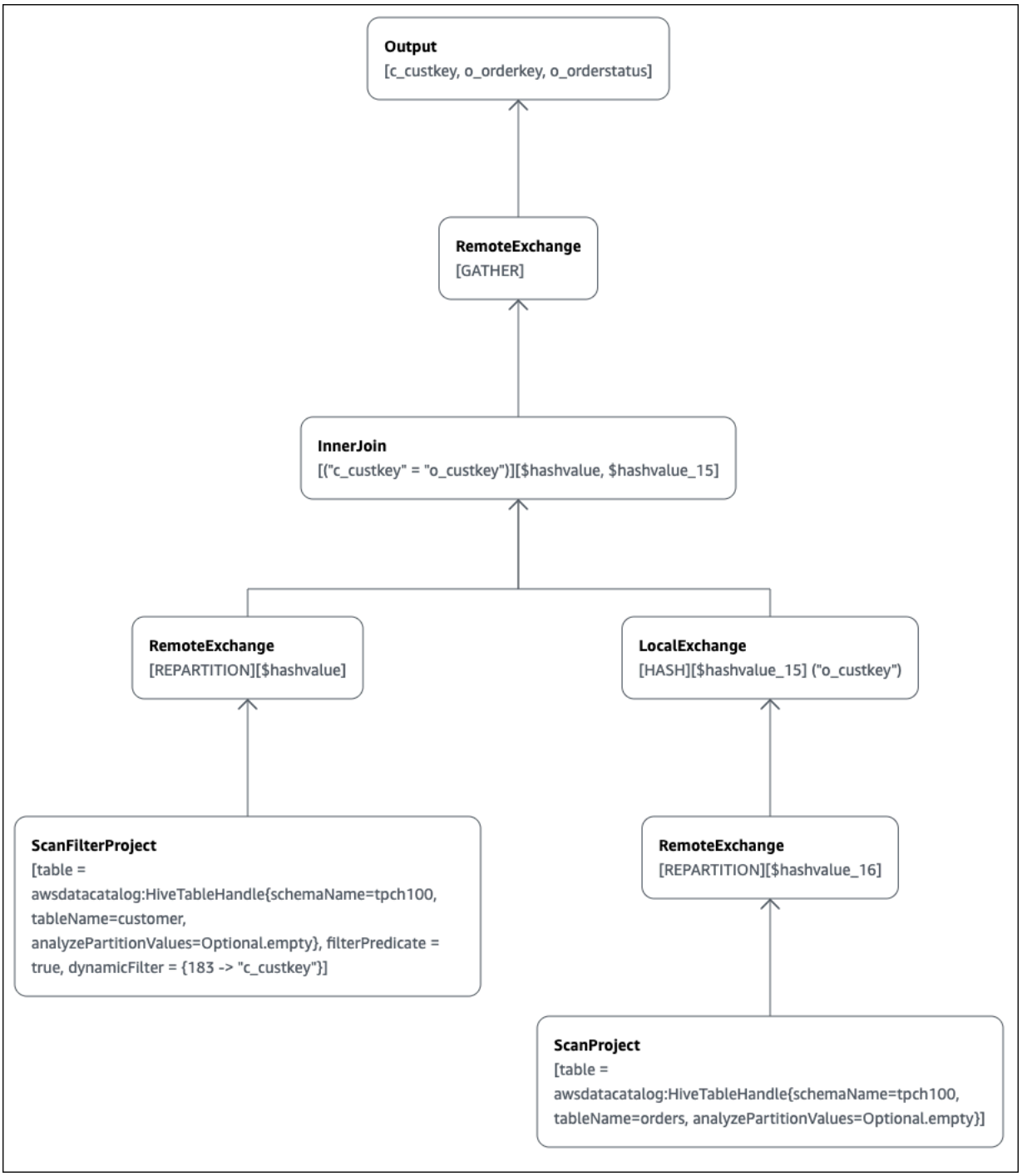
EXPLAIN 예 2: 쿼리 계획을 그래프로 작성

Athena 콘솔을 사용하여 쿼리 계획을 그래프로 작성할 수 있습니다. Athena 쿼리 편집기에 다음과 같이 SELECT 문을 입력한 다음 EXPLAIN을 선택합니다.

```
SELECT
      c.c_custkey,
```

```
    o.o_orderkey,
    o.o_orderstatus
FROM tpch100.customer c
JOIN tpch100.orders o
    ON c.c_custkey = o.o_custkey
```

Athena 쿼리 편집기의 Explain 페이지가 열리고 쿼리에 대한 분산 계획과 논리적 계획이 표시됩니다. 다음 그래프는 예제의 논리적 계획을 보여줍니다.



> **Important**
>
> 현재 일부 파티션 필터는 Athena에서 쿼리에 적용하더라도 중첩 연산자 트리 그래프에 표시되지 않을 수도 있습니다. 이러한 필터의 효과를 확인하려면 쿼리에서 EXPLAIN 또는 EXPLAIN ANALYZE를 실행하고 결과를 확인합니다.

Athena 콘솔에서 쿼리 계획 그래프 작성 기능 사용에 대한 자세한 내용은 SQL 쿼리에 대한 실행 계획 보기 단원을 참조하세요.

EXPLAIN 예 3: EXPLAIN 문을 사용하여 파티션 정리 확인

분할된 테이블을 쿼리할 때 분할된 키에 필터링 조건자를 사용하면 쿼리 엔진은 조건자를 분할된 키에 적용해 읽는 데이터의 양을 줄입니다.

다음 예제에서는 EXPLAIN 쿼리를 사용하여 분할된 테이블에 대한 SELECT 쿼리의 파티션 정리를 확인합니다. 먼저 CREATE TABLE 문은 tpch100.orders_partitioned 테이블을 생성합니다. 테이블은 o_orderdate 열에서 분할됩니다.

```
CREATE TABLE `tpch100.orders_partitioned`(
  `o_orderkey` int,
  `o_custkey` int,
  `o_orderstatus` string,
  `o_totalprice` double,
  `o_orderpriority` string,
  `o_clerk` string,
  `o_shippriority` int,
  `o_comment` string)
PARTITIONED BY (
  `o_orderdate` string)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://DOC-EXAMPLE-BUCKET/<your_directory_path>/'
```

SHOW PARTITIONS 명령으로 표시된 것처럼 tpch100.orders_partitioned 테이블에는 o_orderdate에 여러 개의 파티션이 있습니다.

```
SHOW PARTITIONS tpch100.orders_partitioned;

o_orderdate=1994
o_orderdate=2015
o_orderdate=1998
o_orderdate=1995
o_orderdate=1993
o_orderdate=1997
o_orderdate=1992
o_orderdate=1996
```

다음 EXPLAIN 쿼리는 지정된 SELECT 문의 파티션 정리를 확인합니다.

```
EXPLAIN
SELECT
   o_orderkey,
   o_custkey,
   o_orderdate
FROM tpch100.orders_partitioned
WHERE o_orderdate = '1995'
```

결과

```
Query Plan
- Output[o_orderkey, o_custkey, o_orderdate] => [[o_orderkey, o_custkey, o_orderdate]]
    - RemoteExchange[GATHER] => [[o_orderkey, o_custkey, o_orderdate]]
        - TableScan[awsdatacatalog:HiveTableHandle{schemaName=tpch100,
 tableName=orders_partitioned,
analyzePartitionValues=Optional.empty}] => [[o_orderkey, o_custkey, o_orderdate]]
                LAYOUT: tpch100.orders_partitioned
                o_orderdate := o_orderdate:string:-1:PARTITION_KEY
                    :: [[1995]]
                o_custkey := o_custkey:int:1:REGULAR
                o_orderkey := o_orderkey:int:0:REGULAR
```

결과에서 굵은 텍스트는 o_orderdate = '1995' 조건자가 PARTITION_KEY에 적용되었음을 나타냅니다.

EXPLAIN 예 4: EXPLAIN 쿼리를 사용하여 조인 순서 및 조인 유형 확인

다음 EXPLAIN 쿼리는 SELECT 문의 조인 순서 및 조인 유형을 확인합니다. 이와 같은 쿼리를 사용하면 쿼리 메모리 사용량을 검사하여 EXCEEDED_LOCAL_MEMORY_LIMIT 오류가 발생할 확률을 줄일 수 있습니다.

```
EXPLAIN (TYPE DISTRIBUTED)
   SELECT
      c.c_custkey,
      o.o_orderkey,
      o.o_orderstatus
   FROM tpch100.customer c
   JOIN tpch100.orders o
       ON c.c_custkey = o.o_custkey
   WHERE c.c_custkey = 123
```

결과

```
Query Plan
Fragment 0 [SINGLE]
    Output layout: [c_custkey, o_orderkey, o_orderstatus]
    Output partitioning: SINGLE []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    - Output[c_custkey, o_orderkey, o_orderstatus] => [[c_custkey, o_orderkey,
 o_orderstatus]]
        - RemoteSource[1] => [[c_custkey, o_orderstatus, o_orderkey]]

Fragment 1 [SOURCE]
    Output layout: [c_custkey, o_orderstatus, o_orderkey]
    Output partitioning: SINGLE []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    - CrossJoin => [[c_custkey, o_orderstatus, o_orderkey]]
            Distribution: REPLICATED
        - ScanFilter[table = awsdatacatalog:HiveTableHandle{schemaName=tpch100,
tableName=customer, analyzePartitionValues=Optional.empty}, grouped = false,
filterPredicate = ("c_custkey" = 123)] => [[c_custkey]]
                LAYOUT: tpch100.customer
                c_custkey := c_custkey:int:0:REGULAR
        - LocalExchange[SINGLE] () => [[o_orderstatus, o_orderkey]]
            - RemoteSource[2] => [[o_orderstatus, o_orderkey]]

Fragment 2 [SOURCE]
    Output layout: [o_orderstatus, o_orderkey]
```

```
    Output partitioning: BROADCAST []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    - ScanFilterProject[table = awsdatacatalog:HiveTableHandle{schemaName=tpch100,
tableName=orders, analyzePartitionValues=Optional.empty}, grouped = false,
filterPredicate = ("o_custkey" = 123)] => [[o_orderstatus, o_orderkey]]
            LAYOUT: tpch100.orders
            o_orderstatus := o_orderstatus:string:2:REGULAR
            o_custkey := o_custkey:int:1:REGULAR
            o_orderkey := o_orderkey:int:0:REGULAR
```

예제 쿼리는 성능 향상을 위해 크로스 조인으로 최적화되었습니다. 결과는 tpch100.orders가 BROADCAST 배포 유형으로 배포될 예정임을 보여줍니다. 이는 tpch100.orders 테이블이 조인 작업을 수행하는 모든 노드에 배포될 것임을 의미합니다. BROADCAST 배포 유형은 tpch100.orders 테이블의 모든 필터링된 결과가 조인 작업을 수행하는 각 노드의 메모리에 부합할 것을 요구합니다.

그러나 tpch100.customer 테이블은 tpch100.orders보다 작습니다. tpch100.customer는 더 적은 메모리를 필요로 하므로 쿼리를 tpch100.orders 대신에 BROADCAST tpch100.customer에 다시 작성할 수 있습니다. 이렇게 하면 쿼리가 EXCEEDED_LOCAL_MEMORY_LIMIT 오류를 수신할 가능성이 낮아집니다. 이 전략은 다음 사항을 가정합니다.

- tpch100.customer.c_custkey가 tpch100.customer 테이블에서 고유합니다.
- tpch100.customer와 tpch100.orders 사이에 일대다 매핑 관계가 있습니다.

다음 예제에서는 다시 작성된 쿼리를 보여줍니다.

```
SELECT
    c.c_custkey,
    o.o_orderkey,
    o.o_orderstatus
FROM tpch100.orders o
JOIN tpch100.customer c -- the filtered results of tpch100.customer are distributed to
 all nodes.
    ON c.c_custkey = o.o_custkey
WHERE c.c_custkey = 123
```

EXPLAIN 예 5: EXPLAIN 쿼리를 사용하여 영향이 없는 조건자 제거

EXPLAIN 쿼리를 사용하여 필터링 조건자의 영향을 확인할 수 있습니다. 다음 예제와 같이 결과를 사용하여 영향이 없는 조건자를 제거할 수 있습니다.

```
EXPLAIN
```

```
   SELECT
      c.c_name
   FROM tpch100.customer c
   WHERE c.c_custkey = CAST(RANDOM() * 1000 AS INT)
   AND c.c_custkey BETWEEN 1000 AND 2000
   AND c.c_custkey = 1500
```

결과

```
Query Plan
- Output[c_name] => [[c_name]]
    - RemoteExchange[GATHER] => [[c_name]]
        - ScanFilterProject[table =
awsdatacatalog:HiveTableHandle{schemaName=tpch100,
tableName=customer, analyzePartitionValues=Optional.empty},
filterPredicate = (("c_custkey" = 1500) AND ("c_custkey" =
CAST(("random"() * 1E3) AS int)))] => [[c_name]]
                LAYOUT: tpch100.customer
                c_custkey := c_custkey:int:0:REGULAR
                c_name := c_name:string:1:REGULAR
```

결과의 `filterPredicate`는 옵티마이저가 원래 세 개인 조건자를 두 개의 조건자로 병합하고 애플리케이션의 순서를 변경했음을 보여줍니다.

```
filterPredicate = (("c_custkey" = 1500) AND ("c_custkey" = CAST(("random"() * 1E3) AS
 int)))
```

결과에 따르면 `AND c.c_custkey BETWEEN 1000 AND 2000` 조건자는 아무런 영향이 없으므로 쿼리 결과의 변화 없이 이 조건자를 제거할 수 있습니다.

`EXPLAIN` 쿼리의 결과에 사용된 용어에 관한 자세한 내용은 Athena EXPLAIN 문 결과의 이해 단원을 참조하세요.

EXPLAIN ANALYZE 예제

다음 예에서는 `EXPLAIN ANALYZE` 쿼리 및 출력 예를 보여줍니다.

EXPLAIN ANALYZE 예 1: EXPLAIN ANALYZE를 사용하여 텍스트 형식으로 쿼리 계획 및 계산 비용 표시

다음 예에서 `EXPLAIN ANALYZE`는 CloudFront 로그의 `SELECT` 쿼리에 대한 실행 계획 및 계산 비용을 보여줍니다. 형식은 기본적으로 텍스트 출력입니다.

```
EXPLAIN ANALYZE SELECT FROM cloudfront_logs LIMIT 10
```

결과

```
 Fragment 1
     CPU: 24.60ms, Input: 10 rows (1.48kB); per task: std.dev.: 0.00, Output: 10 rows
 (1.48kB)
     Output layout: [date, time, location, bytes, requestip, method, host, uri, status,
 referrer,\
       os, browser, browserversion]
Limit[10] => [[date, time, location, bytes, requestip, method, host, uri, status,
 referrer, os,\
  browser, browserversion]]
             CPU: 1.00ms (0.03%), Output: 10 rows (1.48kB)
             Input avg.: 10.00 rows, Input std.dev.: 0.00%
LocalExchange[SINGLE] () => [[date, time, location, bytes, requestip, method, host,
 uri, status, referrer, os,\
 browser, browserversion]]
                 CPU: 0.00ns (0.00%), Output: 10 rows (1.48kB)
                 Input avg.: 0.63 rows, Input std.dev.: 387.30%
RemoteSource[2] => [[date, time, location, bytes, requestip, method, host, uri, status,
 referrer, os,\
  browser, browserversion]]
                     CPU: 1.00ms (0.03%), Output: 10 rows (1.48kB)
                     Input avg.: 0.63 rows, Input std.dev.: 387.30%

 Fragment 2
     CPU: 3.83s, Input: 998 rows (147.21kB); per task: std.dev.: 0.00, Output: 20 rows
 (2.95kB)
     Output layout: [date, time, location, bytes, requestip, method, host, uri, status,
 referrer, os,\
       browser, browserversion]
LimitPartial[10] => [[date, time, location, bytes, requestip, method, host, uri,
 status, referrer, os,\
  browser, browserversion]]
             CPU: 5.00ms (0.13%), Output: 20 rows (2.95kB)
             Input avg.: 166.33 rows, Input std.dev.: 141.42%
TableScan[awsdatacatalog:HiveTableHandle{schemaName=default, tableName=cloudfront_logs,
\
  analyzePartitionValues=Optional.empty},
grouped = false] => [[date, time, location, bytes, requestip, method, host, uri, st
                 CPU: 3.82s (99.82%), Output: 998 rows (147.21kB)
                 Input avg.: 166.33 rows, Input std.dev.: 141.42%
```

```
                LAYOUT: default.cloudfront_logs
                date := date:date:0:REGULAR
                referrer := referrer:string:9:REGULAR
                os := os:string:10:REGULAR
                method := method:string:5:REGULAR
                bytes := bytes:int:3:REGULAR
                browser := browser:string:11:REGULAR
                host := host:string:6:REGULAR
                requestip := requestip:string:4:REGULAR
                location := location:string:2:REGULAR
                time := time:string:1:REGULAR
                uri := uri:string:7:REGULAR
                browserversion := browserversion:string:12:REGULAR
                status := status:int:8:REGULAR
```

EXPLAIN ANALYZE 예 2: EXPLAIN ANALYZE를 사용하여 JSON 형식으로 쿼리 계획 표시

다음 예에서는 CloudFront 로그 기반 SELECT 쿼리에 대한 실행 계획 및 계산 비용을 보여줍니다. 이 예에서는 JSON을 출력 형식으로 지정합니다.

```
EXPLAIN ANALYZE (FORMAT JSON) SELECT * FROM cloudfront_logs LIMIT 10
```

결과

```
{
    "fragments": [{
        "id": "1",

        "stageStats": {
            "totalCpuTime": "3.31ms",
            "inputRows": "10 rows",
            "inputDataSize": "1514B",
            "stdDevInputRows": "0.00",
            "outputRows": "10 rows",
            "outputDataSize": "1514B"
        },
        "outputLayout": "date, time, location, bytes, requestip, method, host,\
           uri, status, referrer, os, browser, browserversion",

        "logicalPlan": {
            "1": [{
                "name": "Limit",
                "identifier": "[10]",
```

```
                "outputs": ["date", "time", "location", "bytes", "requestip", "method",
"host",\
                  "uri", "status", "referrer", "os", "browser", "browserversion"],
                "details": "",
                "distributedNodeStats": {
                    "nodeCpuTime": "0.00ns",
                    "nodeOutputRows": 10,
                    "nodeOutputDataSize": "1514B",
                    "operatorInputRowsStats": [{
                        "nodeInputRows": 10.0,
                        "nodeInputRowsStdDev": 0.0
                    }]
                },
                "children": [{
                    "name": "LocalExchange",
                    "identifier": "[SINGLE] ()",
                    "outputs": ["date", "time", "location", "bytes", "requestip",
"method", "host",\
                      "uri", "status", "referrer", "os", "browser", "browserversion"],
                    "details": "",
                    "distributedNodeStats": {
                        "nodeCpuTime": "0.00ns",
                        "nodeOutputRows": 10,
                        "nodeOutputDataSize": "1514B",
                        "operatorInputRowsStats": [{
                            "nodeInputRows": 0.625,
                            "nodeInputRowsStdDev": 387.2983346207417
                        }]
                    },
                    "children": [{
                        "name": "RemoteSource",
                        "identifier": "[2]",
                        "outputs": ["date", "time", "location", "bytes", "requestip",
"method", "host",\
                          "uri", "status", "referrer", "os", "browser",
"browserversion"],
                        "details": "",
                        "distributedNodeStats": {
                            "nodeCpuTime": "0.00ns",
                            "nodeOutputRows": 10,
                            "nodeOutputDataSize": "1514B",
                            "operatorInputRowsStats": [{
                                "nodeInputRows": 0.625,
                                "nodeInputRowsStdDev": 387.2983346207417
```

```
                        }]
                    },
                    "children": []
                }]
            }]
        }]
    }
  }, {
      "id": "2",

      "stageStats": {
          "totalCpuTime": "1.62s",
          "inputRows": "500 rows",
          "inputDataSize": "75564B",
          "stdDevInputRows": "0.00",
          "outputRows": "10 rows",
          "outputDataSize": "1514B"
      },
      "outputLayout": "date, time, location, bytes, requestip, method, host, uri,
status,\
         referrer, os, browser, browserversion",

      "logicalPlan": {
          "1": [{
              "name": "LimitPartial",
              "identifier": "[10]",
              "outputs": ["date", "time", "location", "bytes", "requestip", "method",
"host", "uri",\
                "status", "referrer", "os", "browser", "browserversion"],
              "details": "",
              "distributedNodeStats": {
                  "nodeCpuTime": "0.00ns",
                  "nodeOutputRows": 10,
                  "nodeOutputDataSize": "1514B",
                  "operatorInputRowsStats": [{
                      "nodeInputRows": 83.33333333333333,
                      "nodeInputRowsStdDev": 223.60679774997897
                  }]
              },
              "children": [{
                  "name": "TableScan",
                  "identifier": "[awsdatacatalog:HiveTableHandle{schemaName=default,\
                     tableName=cloudfront_logs,
analyzePartitionValues=Optional.empty},\
```

```
                        grouped = false]",
                    "outputs": ["date", "time", "location", "bytes", "requestip",
 "method", "host", "uri",\
                        "status", "referrer", "os", "browser", "browserversion"],
                    "details": "LAYOUT: default.cloudfront_logs\ndate :=
 date:date:0:REGULAR\nreferrer :=\
                        referrer: string:9:REGULAR\nos := os:string:10:REGULAR
\nmethod := method:string:5:\
                        REGULAR\nbytes := bytes:int:3:REGULAR\nbrowser :=
 browser:string:11:REGULAR\nhost :=\
                        host:string:6:REGULAR\nrequestip := requestip:string:4:REGULAR
\nlocation :=\
                        location:string:2:REGULAR\ntime := time:string:1: REGULAR
\nuri := uri:string:7:\
                        REGULAR\nbrowserversion := browserversion:string:12:REGULAR
\nstatus :=\
                        status:int:8:REGULAR\n",
                    "distributedNodeStats": {
                        "nodeCpuTime": "1.62s",
                        "nodeOutputRows": 500,
                        "nodeOutputDataSize": "75564B",
                        "operatorInputRowsStats": [{
                            "nodeInputRows": 83.33333333333333,
                            "nodeInputRowsStdDev": 223.60679774997897
                        }]
                    },
                    "children": []
                }]
            }]
        }
    }]
}
```

추가적인 리소스

자세한 내용은 다음 리소스를 참조하세요.

- Athena EXPLAIN 문 결과의 이해
- SQL 쿼리에 대한 실행 계획 보기
- 완료된 쿼리에 대한 통계 및 실행 세부 정보 보기
- Trino `EXPLAIN` 설명서
- Trino `EXPLAIN ANALYZE` 설명서

- AWS 빅 데이터 블로그의 Optimize Federated Query Performance using EXPLAIN and EXPLAIN ANALYZE in Amazon Athena

Athena EXPLAIN 문 결과의 이해

이 주제에서는 Athena EXPLAIN 문 결과에 사용되는 연산 용어에 대한 간략한 안내를 제공합니다.

EXPLAIN 문 출력 유형

EXPLAIN 문 출력은 다음 두 유형 중 하나입니다.

- 논리적 계획(Logical plan) - SQL 엔진이 문을 실행하는 데 사용하는 논리적 계획을 표시합니다. 이 옵션의 구문은 EXPLAIN 또는 EXPLAIN (TYPE LOGICAL)입니다.
- 배포된 계획(Distributed plan) - 배포된 환경의 실행 계획을 표시합니다. 출력은 처리 단계인 조각들을 보여줍니다. 각 계획 조각은 하나 이상의 노드에서 처리됩니다. 조각을 처리하는 노드 간에는 데이터를 교환할 수 있습니다. 이 옵션의 구문은 EXPLAIN (TYPE DISTRIBUTED)입니다.

  배포된 계획의 출력에서 조각(처리 단계)은 Fragment *number* [*fragment_type*]으로 표시됩니다. 여기서 *number*는 0부터 시작하는 정수이고 *fragment_type*은 조각이 노드에 의해 실행되는 방식을 지정합니다. 데이터 교환의 레이아웃에 대한 통찰력을 제공하는 조각 유형은 다음 표에 설명되어 있습니다.

  배포된 계획 조각 유형

조각 유형	설명
SINGLE	조각이 하나의 노드에서 실행됩니다.
HASH	조각이 고정된 수의 노드에서 실행됩니다. 입력 데이터가 해시 함수를 사용하여 배포됩니다.
ROUND_ROBIN	조각이 고정된 수의 노드에서 실행됩니다. 입력 데이터가 라운드 로빈 방식으로 배포됩니다.
BROADCAST	조각이 고정된 수의 노드에서 실행됩니다. 입력 데이터가 모든 노드에 브로드캐스트됩니다.
SOURCE	조각이 입력 분할에 액세스되는 노드에서 실행됩니다.

Exchange

교환 관련 용어는 작업자 노드 간에 데이터가 교환되는 방식을 설명합니다. 전송은 로컬 또는 원격일 수 있습니다.

LocalExchange [*exchange_type*]

> 쿼리의 여러 단계에 대해 작업자 노드 내에서 로컬로 데이터를 전송합니다. *exchange_type*의 값은 이 단원의 뒤에서 설명한 대로 논리적 교환 또는 배포된 교환 유형 중 하나일 수 있습니다.

RemoteExchange [*exchange_type*]

> 쿼리의 여러 단계에 대해 작업자 노드 간에 데이터를 전송합니다. *exchange_type*의 값은 이 단원의 뒤에서 설명한 대로 논리적 교환 또는 배포된 교환 유형 중 하나일 수 있습니다.

논리적 교환 유형

다음 교환 유형은 논리적 계획의 교환 단계 중에 수행되는 작업에 대해 설명합니다.

- **GATHER** - 하나의 작업자 노드가 다른 모든 작업자 노드로부터 출력을 수집합니다. 예를 들어, select 쿼리의 마지막 단계에서 모든 노드의 결과를 수집하고 결과를 Amazon S3에 기록합니다.
- **REPARTITION** - 다음 연산자에 적용하는 데 필요한 분할 스키마를 기반으로 특정 작업자에게 행 데이터를 보냅니다.
- **REPLICATE** - 행 데이터를 모든 작업자에게 복사합니다.

배포된 교환 유형

다음 교환 유형은 배포된 계획의 노드 간에 데이터가 교환될 때 데이터의 레이아웃을 나타냅니다.

- **HASH** - 교환이 해시 함수를 사용하여 여러 대상에 데이터를 배포합니다.
- **SINGLE** - 교환이 하나의 대상에 데이터를 배포합니다.

스캔

다음 용어는 쿼리 중에 데이터를 스캔하는 방식을 설명합니다.

TableScan

> Amazon S3 또는 Apache Hive 커넥터에서 온 테이블의 소스 데이터를 스캔하고 필터 조건자로부터 발생된 파티션 정리를 적용합니다.

ScanFilter

Amazon S3 또는 Apache Hive 커넥터에서 온 테이블의 소스 데이터를 스캔하고 필터 조건자 및 파티션 정리 전반에 적용되지 않은 추가 필터 조건자로부터 발생된 파티션 정리를 적용합니다.

ScanFilterProject

먼저 Amazon S3 또는 Apache Hive 커넥터에서 온 테이블의 소스 데이터를 스캔하고 필터 조건자 및 파티션 정리 전반에 적용되지 않은 추가 필터 조건자로부터 발생된 파티션 정리를 적용합니다. 그런 다음 출력 데이터의 메모리 레이아웃을 새로운 프로젝션으로 수정하여 후속 단계의 성능을 향상시킵니다.

조인

두 테이블 간에 데이터를 조인합니다. 조인은 조인 유형 및 배포 유형별로 범주화할 수 있습니다.

조인 유형

조인 유형은 조인 작업이 발생하는 방식을 정의합니다.

CrossJoin - 조인된 두 테이블의 데카르트 곱을 산출합니다.

InnerJoin - 두 테이블에서 일치하는 값을 가진 레코드를 선택합니다.

LeftJoin - 왼쪽 테이블의 모든 레코드와 오른쪽 테이블의 일치하는 레코드를 선택합니다. 일치하는 항목이 없으면 오른쪽의 결과는 NULL입니다.

RightJoin - 오른쪽 테이블의 모든 레코드와 왼쪽 테이블의 일치하는 레코드를 선택합니다. 일치하는 항목이 없으면 왼쪽의 결과는 NULL입니다.

FullJoin - 왼쪽 또는 오른쪽 테이블 레코드에서 일치항목이 있는 모든 레코드를 선택합니다. 조인된 테이블에는 두 테이블의 모든 레코드가 포함되며 양쪽에서 누락된 일치 항목에 대해서는 NULL을 채웁니다.

> **Note**
>
> 성능상의 이유로 쿼리 엔진은 조인 쿼리를 다른 조인 유형으로 다시 작성하여 같은 결과를 낼 수 있습니다. 예를 들어, 하나의 테이블에 조건자가 있는 내부 조인 쿼리는 `CrossJoin`으로 다시 작성할 수 있습니다. 이렇게 하면 조건자가 테이블의 스캔 단계로 푸시다운되어 더 적은 데이터가 검색됩니다.

조인 배포 유형

배포 유형은 조인 작업이 수행될 때 작업자 노드 간에 데이터가 교환되는 방식을 정의합니다.

분할식(Partitioned) - 왼쪽 테이블과 오른쪽 테이블이 모든 작업자 노드에서 모두 해시 분할됩니다. 분할식 배포는 각 노드에서 더 적은 메모리를 소비합니다. 분할된 배포는 복제된 조인보다 훨씬 느릴 수 있습니다. 두 개의 큰 테이블을 조인할 때 분할식 조인이 적합합니다.

복제식(Replicated)- 조인 작업을 수행하기 위해 한 테이블은 모든 작업자 노드에서 해시 분할되고 다른 테이블은 모든 작업자 노드에 복제됩니다. 복제식 배포는 분할식 조인보다 훨씬 빠를 수 있지만 각 작업자 노드에서 더 많은 메모리를 소비합니다. 복제된 테이블이 너무 크면 작업자 노드에서 메모리 부족 오류가 발생할 수 있습니다. 조인된 테이블 중 하나가 작은 경우 복제식 조인이 적합합니다.

PREPARE

`statement_name` 이름으로 나중에 실행할 SQL 문을 작성합니다. 문에는 물음표로 표시되는 파라미터가 포함될 수 있습니다. 파라미터 값을 제공하고 준비된 문을 실행하려면 EXECUTE을 사용합니다.

시놉시스

```
PREPARE statement_name FROM statement
```

다음 표는 이러한 파라미터를 설명합니다.

파라미터	설명
`statement_name`	준비되는 문의 이름입니다. 이 이름은 작업 그룹 내에서 고유해야 합니다.
`statement`	SELECT, CTAS 또는 INSERT INTO 쿼리.

> **Note**
>
> 작업 그룹에서 준비된 문의 최대 개수는 1000입니다.

예제

다음 예제에서는 파라미터 없이 select 쿼리를 준비합니다.

```
PREPARE my_select1 FROM
SELECT * FROM nation
```

다음 예제에서는 파라미터를 포함하여 select 쿼리를 준비합니다. `productid` 및 `quantity`에 대한 값은 `EXECUTE` 문의 `USING` 절로 제공됩니다.

```
PREPARE my_select2 FROM
SELECT order FROM orders WHERE productid = ? and quantity < ?
```

다음 예제에서는 insert 쿼리를 준비합니다.

```
PREPARE my_insert FROM
INSERT INTO cities_usa (city, state)
SELECT city, state
FROM cities_world
WHERE country = ?
```

추가적인 리소스

준비된 문을 사용한 쿼리

EXECUTE

DEALLOCATE PREPARE

INSERT INTO

EXECUTE

준비된 문을 `statement_name` 이름으로 실행합니다. 준비된 문에서 물음표에 대한 파라미터 값은 쉼표로 구분된 목록의 `USING` 절로 정의됩니다. 준비된 문을 생성하려면 PREPARE을 사용합니다.

시놉시스

```
EXECUTE statement_name [ USING parameter1[, parameter2, ... ] ]
```

예제

다음 예제에서는 파라미터 없이 쿼리를 준비하고 실행합니다.

```
PREPARE my_select1 FROM
SELECT name FROM nation
EXECUTE my_select1
```

다음 예제에서는 단일 파라미터를 사용해 쿼리를 준비하고 실행합니다.

```
PREPARE my_select2 FROM
SELECT * FROM "my_database"."my_table" WHERE year = ?
EXECUTE my_select2 USING 2012
```

이는 다음과 동일합니다.

```
SELECT * FROM "my_database"."my_table" WHERE year = 2012
```

다음 예제에서는 두 개의 파라미터를 사용해 쿼리를 준비하고 실행합니다.

```
PREPARE my_select3 FROM
SELECT order FROM orders WHERE productid = ? and quantity < ?
EXECUTE my_select3 USING 346078, 12
```

추가적인 리소스

준비된 문을 사용한 쿼리

PREPARE

INSERT INTO

DEALLOCATE PREPARE

현재 작업 그룹의 준비된 문에서 지정된 이름을 가진 준비된 문을 제거합니다.

시놉시스

```
DEALLOCATE PREPARE statement_name
```

예제

다음 예제는 현재 작업 그룹에서 준비된 문 `my_select1`을 제거합니다.

```
DEALLOCATE PREPARE my_select1
```

추가적인 리소스

준비된 문을 사용한 쿼리

PREPARE

UNLOAD

SELECT 문의 쿼리 결과를 지정된 데이터 형식으로 작성합니다. UNLOAD에 지원되는 형식에는 Apache Parquet, ORC, Apache Avro, JSON이 있습니다. CSV는 Athena SELECT 명령에서 지원하는 유일한 출력 형식이지만 다양한 출력 형식을 지원하는 UNLOAD 명령을 사용하여 SELECT 쿼리를 묶고 UNLOAD에서 지원하는 형식 중 하나로 출력을 다시 작성할 수 있습니다.

CTAS 문을 사용하여 CSV 외의 형식으로 데이터를 출력할 수 있지만 이러한 문은 Athena에서 테이블을 만들어야 합니다. UNLOAD 문은 SELECT 쿼리의 결과를 CSV 외의 형식으로 출력할 때 유용하지만 관련된 테이블을 요구하지 않습니다. 예를 들어 어떤 다운스트림 애플리케이션은 SELECT 쿼리 결과가 JSON 형식이 되도록 요구할 수 있고, 추가적인 분석을 위해 SELECT 쿼리의 결과를 사용하려는 경우 Parquet 또는 ORC가 CSV에 대해 성능상 이점을 제공할 수 있습니다.

고려 사항 및 제한

Athena에서 UNLOAD 문을 사용할 때는 다음 사항을 유의해야 합니다.

- 파일의 전역 정렬 없음 – UNLOAD 결과는 여러 개의 파일에 병렬로 작성됩니다. UNLOAD 문의 SELECT 쿼리에 정렬 순서가 지정된 경우 각 파일의 내용은 순서대로 정렬되지만 파일들은 서로 간에 정렬되지 않습니다.
- 분리된 데이터가 삭제되지 않음 - 실패 시 Athena는 분리된 데이터를 삭제하지 않습니다. 이 동작은 CTAS 및 INSERT INTO 문의 동작과 같습니다.
- 최대 파티션 - UNLOAD에 사용할 수 있는 최대 파티션 수는 100개입니다.
- 메타데이터 및 매니페스트 파일 - Athena는 각각의 UNLOAD 쿼리에 대해 메타데이터 파일 및 데이터 매니페스트 파일을 생성합니다. 매니페스트는 쿼리가 작성한 파일을 추적합니다. 두 파일은 모두 Amazon S3의 Athena 쿼리 결과 위치에 저장됩니다. 자세한 내용은 쿼리 출력 파일 식별 단원을 참조하세요.
- 암호화 – UNLOAD 출력 파일은 Amazon S3에 사용되는 암호화 구성에 따라 암호화됩니다. UNLOAD 결과를 암호화하도록 암호화 구성을 설정하려면 EncryptionConfiguration API를 사용합니다.

- 준비된 문 – UNLOAD는 준비된 문과 함께 사용할 수 있습니다. Athena의 준비된 문에 대한 자세한 내용은 파라미터화된 쿼리 사용 단원을 참조하세요.
- 서비스 할당량 - UNLOAD는 DML 쿼리 할당량을 사용합니다. 할당량에 관한 자세한 내용은 Service Quotas 단원을 참조하세요.
- 예상 버킷 소유자 – 예상 버킷 소유자 설정은 UNLOAD 쿼리에 지정된 대상 Amazon S3 위치에 적용되지 않습니다. 예상 버킷 소유자 설정은 Athena 쿼리 결과에 대해 지정한 Amazon S3 출력 위치에만 적용됩니다. 자세한 내용은 Athena 콘솔을 사용하여 쿼리 결과 위치 지정 단원을 참조하십시오.

구문

UNLOAD 문은 다음 구문을 사용합니다.

```
UNLOAD (SELECT col_name[, ...] FROM old_table)
TO 's3://DOC-EXAMPLE-BUCKET/my_folder/'
WITH ( property_name = 'expression' [, ...] )
```

파티션에 쓰는 경우를 제외하고, TO 대상에는 데이터가 없는 Amazon S3의 위치를 지정해야 합니다. UNLOAD 쿼리는 지정된 위치에 데이터를 쓰기 전에 버킷 위치가 비어 있는지 확인합니다. UNLOAD는 지정된 위치에 이미 데이터가 있는 경우 해당 위치에 데이터를 쓰지 않습니다. UNLOAD는 기존 데이터를 덮어쓰지 않습니다. 버킷 위치를 UNLOAD의 대상으로 재사용하려면 버킷 위치에서 데이터를 삭제한 다음 쿼리를 다시 실행합니다.

UNLOAD가 파티션에 쓸 때는 이 동작이 다르다는 점에 유의하세요. SELECT 문, TO 위치, 파티션이 동일한 같은 UNLOAD 쿼리를 여러 번 실행하는 경우 각 UNLOAD 쿼리는 지정된 위치와 파티션에 있는 Amazon S3로 데이터를 언로드합니다.

파라미터

*property_name*에 가능한 값은 다음과 같습니다.

format = '***file_format***'

필수 사항입니다. 출력의 파일 형식을 지정합니다. *file_format*에 가능한 값은 ORC, PARQUET, AVRO, JSON, 또는 TEXTFILE입니다.

compression = '***compression_format***'

선택 사항. 이 옵션은 ORC 및 Parquet 형식에만 특정합니다. ORC은 기본값이 zlib이고, Parquet는 기본값이 gzip입니다. 지원되는 압축 형식에 대한 자세한 내용은 Athena 압축 지원을 참조하세요.

> **Note**
>
> 이 옵션은 AVRO 형식에 적용되지 않습니다. Athena는 JSON 및 TEXTFILE 형식에 gzip를 사용합니다.

compression_level = ***compression_level***

선택 사항입니다. ZSTD 압축에 사용할 압축 수준입니다. 이 속성은 ZSTD 압축에만 적용됩니다. 자세한 내용은 Athena에서 ZSTD 압축 수준 사용 단원을 참조하십시오.

field_delimiter = '***delimiter***'

선택 사항입니다. CSV, TSV 및 기타 텍스트 형식에 대해 단일 문자의 필드 구분 기호를 지정합니다. 다음은 쉼표를 구분 기호로 지정한 예입니다.

```
WITH (field_delimiter = ',')
```

현재 여러 문자로 된 필드 구분 기호는 지원되지 않습니다. 필드 구분 기호를 지정하지 않으면 8진법 문자 \001(^A)이 사용됩니다.

partitioned_by = ARRAY[***col_name***[,...]]

선택 사항입니다. 출력을 분할하는 기준이 되는 열의 배열 목록입니다.

> **Note**
>
> SELECT 문에서 분할된 열의 이름이 문의 열 목록 마지막에 있는지 확인합니다.

예제

다음 예제는 SELECT 쿼리의 출력을 JSON 형식으로 Amazon S3 위치 s3://DOC-EXAMPLE-BUCKET/unload_test_1/에 작성합니다.

```
UNLOAD (SELECT * FROM old_table)
TO 's3://DOC-EXAMPLE-BUCKET/unload_test_1/'
WITH (format = 'JSON')
```

다음 예제는 SELECT 쿼리의 출력을, Snappy 압축을 사용해 Parquet 형식으로 작성합니다.

```
UNLOAD (SELECT * FROM old_table)
TO 's3://DOC-EXAMPLE-BUCKET/'
WITH (format = 'PARQUET',compression = 'SNAPPY')
```

다음 예제는 마지막 열을 기준으로 출력을 분할하여 네 개의 열을 텍스트 형식으로 작성합니다.

```
UNLOAD (SELECT name1, address1, comment1, key1 FROM table1)
TO 's3://DOC-EXAMPLE-BUCKET/ partitioned/'
WITH (format = 'TEXTFILE', partitioned_by = ARRAY['key1'])
```

다음 예제에서는 Parquet 파일 형식, ZSTD 압축 및 ZSTD 압축 수준 4를 사용하여 쿼리 결과를 지정된 위치로 언로드합니다.

```
UNLOAD (SELECT * FROM old_table)
TO 's3://DOC-EXAMPLE-BUCKET/'
WITH (format = 'PARQUET', compression = 'ZSTD', compression_level = 4)
```

추가적인 리소스

- AWS 빅 데이터 블로그의 Simplify your ETL and ML pipelines using the Amazon Athena UNLOAD feature.

Amazon Athena의 함수

Athena 엔진 버전 간 함수 변경에 대해서는 Athena 엔진 버전 참조 섹션을 참조하세요. `AT TIME ZONE` 연산자에 사용할 수 있는 표준 시간대 목록은 지원되는 시간대를 참조하세요.

Athena 엔진 버전 3

Athena 엔진 버전 3의 함수는 Trino 기준입니다. Trino 함수, 연산자, 표현식에 대한 자세한 내용은 Trino 설명서에서 함수 및 연산자와 다음과 같은 하위 섹션을 참조하세요.

- 집계
- 배열
- 이진
- 비트
- Color(색상)

- 비교
- 조건
- 변환
- 날짜 및 시간
- 10진수
- 지리 공간
- HyperLogLog
- IP 주소
- JSON
- Lambda
- 논리적
- 기계 학습
- 지도
- 수학 연산
- 분위수 다이제스트
- 정규식
- 세션
- 세트 다이제스트
- 문자열
- 표
- Teradata
- T-다이제스트
- URL
- UUID
- Window

Athena 엔진 버전 2

Athena 엔진 버전 2의 함수는 Presto 0.217을 기반으로 합니다. Athena 엔진 버전 2의 지리 공간 함수에 대해서는 Athena 엔진 버전 2의 지리 공간 함수를 참조하세요.

> **Note**
>
> Presto 0.217 함수에 대한 버전별 설명서는 더 이상 사용할 수 없습니다. 현재 Presto 함수, 연산자 및 표현식에 대한 자세한 내용을 알아보려면 Presto Functions and Operators(Presto 함수 및 연산자)를 참조하거나 이 섹션의 하위 범주 링크를 방문하세요.

- 논리 연산자
- 비교 함수와 연산자
- 조건식
- 변환 함수
- 수학 함수와 연산자
- 비트 함수
- 소수 함수와 연산자
- 문자열 함수와 연산자
- 이진 함수
- 날짜 및 시간 함수와 연산자
- 정규식 함수
- JSON 함수와 연산자
- URL 함수
- 집계 함수
- 윈도 함수
- 색상 함수
- 배열 함수와 연산자
- 맵 함수와 연산자
- 람다 식과 함수
- Teradata 함수

지원되는 시간대

`SELECT timestamp` 문에서 `AT TIME ZONE` 연산자를 사용하여 다음 예제와 같이 반환된 타임스탬프에 시간대를 지정할 수 있습니다.

```
SELECT timestamp '2012-10-31 01:00 UTC' AT TIME ZONE 'America/Los_Angeles' AS la_time;
```

결과

```
la_time

2012-10-30 18:00:00.000 America/Los_Angeles
```

다음 목록에는 Athena에서 AT TIME ZONE 연산자에 사용할 수 있는 표준 시간대가 포함되어 있습니다. 시간대 관련 추가 함수 및 예제는 시간대 함수 및 예제 단원을 참조하세요.

```
Africa/Abidjan
Africa/Accra
Africa/Addis_Ababa
Africa/Algiers
Africa/Asmara
Africa/Asmera
Africa/Bamako
Africa/Bangui
Africa/Banjul
Africa/Bissau
Africa/Blantyre
Africa/Brazzaville
Africa/Bujumbura
Africa/Cairo
Africa/Casablanca
Africa/Ceuta
Africa/Conakry
Africa/Dakar
Africa/Dar_es_Salaam
Africa/Djibouti
Africa/Douala
Africa/El_Aaiun
Africa/Freetown
Africa/Gaborone
Africa/Harare
Africa/Johannesburg
Africa/Juba
Africa/Kampala
Africa/Khartoum
Africa/Kigali
Africa/Kinshasa
```

```
Africa/Lagos
Africa/Libreville
Africa/Lome
Africa/Luanda
Africa/Lubumbashi
Africa/Lusaka
Africa/Malabo
Africa/Maputo
Africa/Maseru
Africa/Mbabane
Africa/Mogadishu
Africa/Monrovia
Africa/Nairobi
Africa/Ndjamena
Africa/Niamey
Africa/Nouakchott
Africa/Ouagadougou
Africa/Porto-Novo
Africa/Sao_Tome
Africa/Timbuktu
Africa/Tripoli
Africa/Tunis
Africa/Windhoek
America/Adak
America/Anchorage
America/Anguilla
America/Antigua
America/Araguaina
America/Argentina/Buenos_Aires
America/Argentina/Catamarca
America/Argentina/ComodRivadavia
America/Argentina/Cordoba
America/Argentina/Jujuy
America/Argentina/La_Rioja
America/Argentina/Mendoza
America/Argentina/Rio_Gallegos
America/Argentina/Salta
America/Argentina/San_Juan
America/Argentina/San_Luis
America/Argentina/Tucuman
America/Argentina/Ushuaia
America/Aruba
America/Asuncion
America/Atikokan
```

```
America/Atka
America/Bahia
America/Bahia_Banderas
America/Barbados
America/Belem
America/Belize
America/Blanc-Sablon
America/Boa_Vista
America/Bogota
America/Boise
America/Buenos_Aires
America/Cambridge_Bay
America/Campo_Grande
America/Cancun
America/Caracas
America/Catamarca
America/Cayenne
America/Cayman
America/Chicago
America/Chihuahua
America/Coral_Harbour
America/Cordoba
America/Costa_Rica
America/Creston
America/Cuiaba
America/Curacao
America/Danmarkshavn
America/Dawson
America/Dawson_Creek
America/Denver
America/Detroit
America/Dominica
America/Edmonton
America/Eirunepe
America/El_Salvador
America/Ensenada
America/Fort_Nelson
America/Fort_Wayne
America/Fortaleza
America/Glace_Bay
America/Godthab
America/Goose_Bay
America/Grand_Turk
America/Grenada
```

```
America/Guadeloupe
America/Guatemala
America/Guayaquil
America/Guyana
America/Halifax
America/Havana
America/Hermosillo
America/Indiana/Indianapolis
America/Indiana/Knox
America/Indiana/Marengo
America/Indiana/Petersburg
America/Indiana/Tell_City
America/Indiana/Vevay
America/Indiana/Vincennes
America/Indiana/Winamac
America/Indianapolis
America/Inuvik
America/Iqaluit
America/Jamaica
America/Jujuy
America/Juneau
America/Kentucky/Louisville
America/Kentucky/Monticello
America/Knox_IN
America/Kralendijk
America/La_Paz
America/Lima
America/Los_Angeles
America/Louisville
America/Lower_Princes
America/Maceio
America/Managua
America/Manaus
America/Marigot
America/Martinique
America/Matamoros
America/Mazatlan
America/Mendoza
America/Menominee
America/Merida
America/Metlakatla
America/Mexico_City
America/Miquelon
America/Moncton
```

```
America/Monterrey
America/Montevideo
America/Montreal
America/Montserrat
America/Nassau
America/New_York
America/Nipigon
America/Nome
America/Noronha
America/North_Dakota/Beulah
America/North_Dakota/Center
America/North_Dakota/New_Salem
America/Ojinaga
America/Panama
America/Pangnirtung
America/Paramaribo
America/Phoenix
America/Port-au-Prince
America/Port_of_Spain
America/Porto_Acre
America/Porto_Velho
America/Puerto_Rico
America/Punta_Arenas
America/Rainy_River
America/Rankin_Inlet
America/Recife
America/Regina
America/Resolute
America/Rio_Branco
America/Rosario
America/Santa_Isabel
America/Santarem
America/Santiago
America/Santo_Domingo
America/Sao_Paulo
America/Scoresbysund
America/Shiprock
America/Sitka
America/St_Barthelemy
America/St_Johns
America/St_Kitts
America/St_Lucia
America/St_Thomas
America/St_Vincent
```

```
America/Swift_Current
America/Tegucigalpa
America/Thule
America/Thunder_Bay
America/Tijuana
America/Toronto
America/Tortola
America/Vancouver
America/Virgin
America/Whitehorse
America/Winnipeg
America/Yakutat
America/Yellowknife
Antarctica/Casey
Antarctica/Davis
Antarctica/DumontDUrville
Antarctica/Macquarie
Antarctica/Mawson
Antarctica/McMurdo
Antarctica/Palmer
Antarctica/Rothera
Antarctica/South_Pole
Antarctica/Syowa
Antarctica/Troll
Antarctica/Vostok
Arctic/Longyearbyen
Asia/Aden
Asia/Almaty
Asia/Amman
Asia/Anadyr
Asia/Aqtau
Asia/Aqtobe
Asia/Ashgabat
Asia/Ashkhabad
Asia/Atyrau
Asia/Baghdad
Asia/Bahrain
Asia/Baku
Asia/Bangkok
Asia/Barnaul
Asia/Beirut
Asia/Bishkek
Asia/Brunei
Asia/Calcutta
```

```
Asia/Chita
Asia/Choibalsan
Asia/Chongqing
Asia/Chungking
Asia/Colombo
Asia/Dacca
Asia/Damascus
Asia/Dhaka
Asia/Dili
Asia/Dubai
Asia/Dushanbe
Asia/Gaza
Asia/Harbin
Asia/Hebron
Asia/Ho_Chi_Minh
Asia/Hong_Kong
Asia/Hovd
Asia/Irkutsk
Asia/Istanbul
Asia/Jakarta
Asia/Jayapura
Asia/Jerusalem
Asia/Kabul
Asia/Kamchatka
Asia/Karachi
Asia/Kashgar
Asia/Kathmandu
Asia/Katmandu
Asia/Khandyga
Asia/Kolkata
Asia/Krasnoyarsk
Asia/Kuala_Lumpur
Asia/Kuching
Asia/Kuwait
Asia/Macao
Asia/Macau
Asia/Magadan
Asia/Makassar
Asia/Manila
Asia/Muscat
Asia/Nicosia
Asia/Novokuznetsk
Asia/Novosibirsk
Asia/Omsk
```

```
Asia/Oral
Asia/Phnom_Penh
Asia/Pontianak
Asia/Pyongyang
Asia/Qatar
Asia/Qyzylorda
Asia/Rangoon
Asia/Riyadh
Asia/Saigon
Asia/Sakhalin
Asia/Samarkand
Asia/Seoul
Asia/Shanghai
Asia/Singapore
Asia/Srednekolymsk
Asia/Taipei
Asia/Tashkent
Asia/Tbilisi
Asia/Tehran
Asia/Tel_Aviv
Asia/Thimbu
Asia/Thimphu
Asia/Tokyo
Asia/Tomsk
Asia/Ujung_Pandang
Asia/Ulaanbaatar
Asia/Ulan_Bator
Asia/Urumqi
Asia/Ust-Nera
Asia/Vientiane
Asia/Vladivostok
Asia/Yakutsk
Asia/Yangon
Asia/Yekaterinburg
Asia/Yerevan
Atlantic/Azores
Atlantic/Bermuda
Atlantic/Canary
Atlantic/Cape_Verde
Atlantic/Faeroe
Atlantic/Faroe
Atlantic/Jan_Mayen
Atlantic/Madeira
Atlantic/Reykjavik
```

```
Atlantic/South_Georgia
Atlantic/St_Helena
Atlantic/Stanley
Australia/ACT
Australia/Adelaide
Australia/Brisbane
Australia/Broken_Hill
Australia/Canberra
Australia/Currie
Australia/Darwin
Australia/Eucla
Australia/Hobart
Australia/LHI
Australia/Lindeman
Australia/Lord_Howe
Australia/Melbourne
Australia/NSW
Australia/North
Australia/Perth
Australia/Queensland
Australia/South
Australia/Sydney
Australia/Tasmania
Australia/Victoria
Australia/West
Australia/Yancowinna
Brazil/Acre
Brazil/DeNoronha
Brazil/East
Brazil/West
CET
CST6CDT
Canada/Atlantic
Canada/Central
Canada/Eastern
Canada/Mountain
Canada/Newfoundland
Canada/Pacific
Canada/Saskatchewan
Canada/Yukon
Chile/Continental
Chile/EasterIsland
Cuba
EET
```

```
EST5EDT
Egypt
Eire
Europe/Amsterdam
Europe/Andorra
Europe/Astrakhan
Europe/Athens
Europe/Belfast
Europe/Belgrade
Europe/Berlin
Europe/Bratislava
Europe/Brussels
Europe/Bucharest
Europe/Budapest
Europe/Busingen
Europe/Chisinau
Europe/Copenhagen
Europe/Dublin
Europe/Gibraltar
Europe/Guernsey
Europe/Helsinki
Europe/Isle_of_Man
Europe/Istanbul
Europe/Jersey
Europe/Kaliningrad
Europe/Kiev
Europe/Kirov
Europe/Lisbon
Europe/Ljubljana
Europe/London
Europe/Luxembourg
Europe/Madrid
Europe/Malta
Europe/Mariehamn
Europe/Minsk
Europe/Monaco
Europe/Moscow
Europe/Nicosia
Europe/Oslo
Europe/Paris
Europe/Podgorica
Europe/Prague
Europe/Riga
Europe/Rome
```

```
Europe/Samara
Europe/San_Marino
Europe/Sarajevo
Europe/Simferopol
Europe/Skopje
Europe/Sofia
Europe/Stockholm
Europe/Tallinn
Europe/Tirane
Europe/Tiraspol
Europe/Ulyanovsk
Europe/Uzhgorod
Europe/Vaduz
Europe/Vatican
Europe/Vienna
Europe/Vilnius
Europe/Volgograd
Europe/Warsaw
Europe/Zagreb
Europe/Zaporozhye
Europe/Zurich
GB
GB-Eire
Hongkong
Iceland
Indian/Antananarivo
Indian/Chagos
Indian/Christmas
Indian/Cocos
Indian/Comoro
Indian/Kerguelen
Indian/Mahe
Indian/Maldives
Indian/Mauritius
Indian/Mayotte
Indian/Reunion
Iran
Israel
Jamaica
Japan
Kwajalein
Libya
MET
MST7MDT
```

```
Mexico/BajaNorte
Mexico/BajaSur
Mexico/General
NZ
NZ-CHAT
Navajo
PRC
PST8PDT
Pacific/Apia
Pacific/Auckland
Pacific/Bougainville
Pacific/Chatham
Pacific/Chuuk
Pacific/Easter
Pacific/Efate
Pacific/Enderbury
Pacific/Fakaofo
Pacific/Fiji
Pacific/Funafuti
Pacific/Galapagos
Pacific/Gambier
Pacific/Guadalcanal
Pacific/Guam
Pacific/Honolulu
Pacific/Johnston
Pacific/Kiritimati
Pacific/Kosrae
Pacific/Kwajalein
Pacific/Majuro
Pacific/Marquesas
Pacific/Midway
Pacific/Nauru
Pacific/Niue
Pacific/Norfolk
Pacific/Noumea
Pacific/Pago_Pago
Pacific/Palau
Pacific/Pitcairn
Pacific/Pohnpei
Pacific/Ponape
Pacific/Port_Moresby
Pacific/Rarotonga
Pacific/Saipan
Pacific/Samoa
```

```
Pacific/Tahiti
Pacific/Tarawa
Pacific/Tongatapu
Pacific/Truk
Pacific/Wake
Pacific/Wallis
Pacific/Yap
Poland
Portugal
ROK
Singapore
Turkey
US/Alaska
US/Aleutian
US/Arizona
US/Central
US/East-Indiana
US/Eastern
US/Hawaii
US/Indiana-Starke
US/Michigan
US/Mountain
US/Pacific
US/Pacific-New
US/Samoa
W-SU
WET
```

시간대 함수 및 예제

다음은 시간대와 관련된 몇 가지 추가 함수 및 예제입니다.

- at_timezone(*#####*, *##*) – *##*에 해당하는 현지 시간의 *#####* 값을 반환합니다.

  예

  ```
  SELECT at_timezone(timestamp '2021-08-22 00:00 UTC', 'Canada/Newfoundland')
  ```

  결과

  ```
  2021-08-21 21:30:00.000 Canada/Newfoundland
  ```

- timezone_hour(*#####*) – 타임스탬프에서 시간대의 시간 오프셋을 `bigint`로 반환합니다.

예

```
SELECT timezone_hour(timestamp '2021-08-22 04:00 UTC' AT TIME ZONE 'Canada/
Newfoundland')
```

결과

```
-2
```

- timezone_minute(*#####*) – *#####*에서 시간대의 분 오프셋을 bigint로 반환합니다.

  예

  ```
  SELECT timezone_minute(timestamp '2021-08-22 04:00 UTC' AT TIME ZONE 'Canada/
  Newfoundland')
  ```

  결과

  ```
  -30
  ```

- with_timezone(*#####*, *##*) – 지정한 *#####* 및 *##* 값에서 시간대와 함께 타임스탬프를 반환합니다.

  예

  ```
  SELECT with_timezone(timestamp '2021-08-22 04:00', 'Canada/Newfoundland')
  ```

  결과

  ```
  2021-08-22 04:00:00.000 Canada/Newfoundland
  ```

DDL 문

Athena에서 바로 다음 DDL 문을 사용합니다.

Athena 쿼리 엔진은 부분적으로 HiveQL DDL을 기반으로 합니다.

Athena는 모든 DDL 설명을 지원하지는 않으며, HiveQL DDL과 Athena DDL 간에는 일부 차이점이 있습니다. 자세한 내용은 이 단원의 참조 주제 및 지원되지 않는 DDL을(를) 참조하세요.

주제

지원되지 않는 DDL

다음 DDL 문은 Athena에서 지원되지 않습니다.

- ALTER INDEX
- ALTER TABLE *table_name* ARCHIVE PARTITION
- ALTER TABLE *table_name* CLUSTERED BY
- ALTER TABLE *table_name* EXCHANGE PARTITION
- ALTER TABLE *table_name* NOT CLUSTERED
- ALTER TABLE *table_name* NOT SKEWED
- ALTER TABLE *table_name* NOT SORTED
- ALTER TABLE *table_name* NOT STORED AS DIRECTORIES
- ALTER TABLE *table_name* partitionSpec CHANGE COLUMNS
- ALTER TABLE *table_name* partitionSpec COMPACT
- ALTER TABLE *table_name* partitionSpec CONCATENATE
- ALTER TABLE *table_name* partitionSpec SET FILEFORMAT
- ALTER TABLE *table_name* SET SERDEPROPERTIES
- ALTER TABLE *table_name* SET SKEWED LOCATION
- ALTER TABLE *table_name* SKEWED BY
- ALTER TABLE *table_name* TOUCH
- ALTER TABLE *table_name* UNARCHIVE PARTITION
- COMMIT
- CREATE INDEX
- 역할 생성
- CREATE TABLE *table_name* LIKE *existing_table_name*
- CREATE TEMPORARY MACRO
- DELETE FROM
- DESCRIBE DATABASE
- DFS
- DROP INDEX

- DROP ROLE
- DROP TEMPORARY MACRO
- EXPORT TABLE
- GRANT ROLE
- IMPORT TABLE
- LOCK DATABASE
- LOCK TABLE
- REVOKE ROLE
- ROLLBACK
- SHOW COMPACTIONS
- SHOW CURRENT ROLES
- SHOW GRANT
- SHOW INDEXES
- SHOW LOCKS
- SHOW PRINCIPALS
- SHOW ROLE GRANT
- SHOW ROLES
- SHOW STATS
- SHOW TRANSACTIONS
- START TRANSACTION
- UNLOCK DATABASE
- UNLOCK TABLE

ALTER DATABASE SET DBPROPERTIES

데이터베이스에 대해 하나 이상의 속성을 생성합니다. DATABASE와 SCHEMA는 동일하게 사용됩니다. 두 용어의 의미는 동일합니다.

시놉시스

```
ALTER {DATABASE|SCHEMA} database_name
```

```
  SET DBPROPERTIES ('property_name'='property_value' [, ...] )
```

파라미터

SET DBPROPERTIES ('property_name'='property_value' [, ...]

property_name이라는 데이터베이스에 속성을 지정하고, 속성의 값을 각각 property_value로 설정합니다. property_name이 이미 존재하면 이전 값이 property_value로 재설정됩니다.

예제

```
ALTER DATABASE jd_datasets
  SET DBPROPERTIES ('creator'='John Doe', 'department'='applied mathematics');
```

```
ALTER SCHEMA jd_datasets
  SET DBPROPERTIES ('creator'='Jane Doe');
```

ALTER TABLE ADD COLUMNS

기존 테이블에 하나 이상의 열을 추가합니다. 선택적 PARTITION 구문을 사용할 때 파티션 메타데이터를 업데이트합니다.

시놉시스

```
ALTER TABLE table_name
  [PARTITION
   (partition_col1_name = partition_col1_value
   [,partition_col2_name = partition_col2_value][,...])]
  ADD COLUMNS (col_name data_type)
```

파라미터

PARTITION (partition_col_name = partition_col_value [,...])

지정한 열 이름/값 조합으로 파티션을 생성합니다. 열의 데이터 형식이 문자열인 경우에만 인용 부호로 partition_col_value를 묶습니다.

ADD COLUMNS (col_name data_type [,col_name data_type,...])

기존 열 뒤에(즉, 파티션 열 앞에) 열을 추가합니다.

예제

```
ALTER TABLE events ADD COLUMNS (eventowner string)
```

```
ALTER TABLE events PARTITION (awsregion='us-west-2') ADD COLUMNS (event string)
```

```
ALTER TABLE events PARTITION (awsregion='us-west-2') ADD COLUMNS (eventdescription
 string)
```

참고

- `ALTER TABLE ADD COLUMNS`를 실행한 후 Athena 쿼리 편집기 탐색 창에서 새 테이블 열을 보려면 편집기에서 테이블 목록을 수동으로 새로 고친 다음 테이블을 다시 확장합니다.
- `ALTER TABLE ADD COLUMNS`는 `date` 데이터 형식을 가진 열에 대해 작동하지 않습니다. 이 문제를 해결하려면 `timestamp` 데이터 형식을 대신 사용하세요.

ALTER TABLE ADD PARTITION

테이블에 대해 하나 이상의 파티션 열을 만듭니다. 각 파티션은 하나 이상의 개별 열 이름/값 조합으로 구성됩니다. 지정된 각각의 조합에 별개의 데이터 디렉터리가 생성되어 상황에 따라 쿼리 성능을 개선할 수 있습니다. 분할된 열은 테이블 데이터 자체 내에 존재하지 않으므로 테이블 자체의 열과 이름이 같은 열 이름을 사용하면 오류가 발생합니다. 자세한 내용은 Athena에서 데이터 분할 단원을 참조하십시오.

Athena에서 테이블과 그 파티션은 동일한 데이터 형식을 사용해야 하지만 스키마는 다를 수 있습니다. 자세한 내용은 파티션이 있는 테이블의 업데이트 단원을 참조하세요.

IAM 정책에 필요한 리소스 수준 권한(`glue:CreatePartition` 등)에 대한 자세한 내용은 AWS Glue API 권한: 작업 및 리소스 참조 및 AWS Glue Data Catalog의 데이터베이스와 테이블에 대한 세분화된 액세스 섹션을 참조하세요. Athena 사용 시의 권한에 대한 문제 해결 정보는 권한 주제의 Athena의 문제 해결 섹션을 참조하세요.

시놉시스

```
ALTER TABLE table_name ADD [IF NOT EXISTS]
  PARTITION
  (partition_col1_name = partition_col1_value
```

```
  [,partition_col2_name = partition_col2_value]
  [,...])
  [LOCATION 'location1']
  [PARTITION
  (partition_colA_name = partition_colA_value
  [,partition_colB_name = partition_colB_value
  [,...])]
  [LOCATION 'location2']
  [,...]
```

파라미터

파티션을 추가할 때 파티션에 대해 하나 이상의 열 이름/값 페어와 해당 파티션의 데이터 파일이 있는 Amazon S3 경로를 지정합니다.

[IF NOT EXISTS]

동일한 정의의 파티션이 이미 있으면 오류가 억제되도록 합니다.

PARTITION (partition_col_name = partition_col_value [,...])

지정한 열 이름/값 조합으로 파티션을 생성합니다. 열의 데이터 형식이 문자열인 경우에만 `partition_col_value`를 문자열 문자로 묶습니다.

[LOCATION 'location']

위의 명령문에서 정의한 파티션을 저장할 디렉터리를 지정합니다. 데이터가 Hive 스타일 파티셔닝(pk1=v1/pk2=v2/pk3=v3)을 사용하는 경우 `LOCATION` 절은 선택 사항입니다. Hive 스타일 파티셔닝을 사용하면 전체 Amazon S3 URI가 테이블 위치, 파티션 키 이름 및 파티션 키 값을 기반으로 자동으로 구성됩니다. 자세한 내용은 Athena에서 데이터 분할 단원을 참조하십시오.

고려 사항

Amazon Athena는 단일 `ALTER TABLE ADD PARTITION` DDL 문에 추가할 수 있는 파티션 수에 제한을 두지 않습니다. 하지만 상당한 수의 파티션을 추가해야 하는 경우 잠재적인 성능 문제를 방지하기 위해 작업을 더 작은 배치로 나누는 것을 고려해야 합니다. 다음 예제에서는 연속 명령을 사용하여 파티션을 개별적으로 추가하고 중복된 파티션을 추가하지 않도록 `IF NOT EXISTS`를 사용합니다.

```
ALTER TABLE table_name ADD IF NOT EXISTS PARTITION (ds='2023-01-01')
ALTER TABLE table_name ADD IF NOT EXISTS PARTITION (ds='2023-01-02')
ALTER TABLE table_name ADD IF NOT EXISTS PARTITION (ds='2023-01-03')
```

Athena에서 파티션을 사용할 때 다음 사항에 유의하세요.

- Athena는 1천만 개의 파티션이 있는 AWS Glue 테이블에 대한 쿼리를 지원하지만, 단일 스캔으로 1백만 개 이상의 파티션을 읽을 수는 없습니다.
- 쿼리를 최적화하고 스캔되는 파티션 수를 줄이려면 파티션 정리 또는 파티션 인덱스 사용과 같은 전략을 고려해 보세요.
- AWS Glue Data Catalog를 사용하지 않는 경우 테이블당 최대 파티션 수는 20,000개입니다. 할당량 증가를 요청할 수 있습니다.

Athena에서의 파티션 사용에 대한 추가 고려 사항은 Athena에서 데이터 분할 섹션을 참조하세요.

예제

다음 예제는 Hive 스타일로 파티셔닝된 데이터의 테이블에 단일 파티션을 추가합니다.

```
ALTER TABLE orders ADD
  PARTITION (dt = '2016-05-14', country = 'IN');
```

다음 예제는 Hive 스타일로 파티셔닝된 데이터의 테이블에 다중 파티션을 추가합니다.

```
ALTER TABLE orders ADD
  PARTITION (dt = '2016-05-31', country = 'IN')
  PARTITION (dt = '2016-06-01', country = 'IN');
```

테이블이 Hive 스타일로 파티셔닝된 데이터를 지원하지 않는 경우 `LOCATION` 절은 필수이며, 파티션의 데이터를 포함하는 접두사에 대해 전체 Amazon S3 URI여야 합니다.

```
ALTER TABLE orders ADD
  PARTITION (dt = '2016-05-31', country = 'IN') LOCATION 's3://DOC-EXAMPLE-BUCKET/path/
to/INDIA_31_May_2016/'
  PARTITION (dt = '2016-06-01', country = 'IN') LOCATION 's3://DOC-EXAMPLE-BUCKET/path/
to/INDIA_01_June_2016/';
```

파티션이 이미 존재하는 경우 오류를 무시하려면 다음 예제와 같이 `IF NOT EXISTS` 절을 사용합니다.

```
ALTER TABLE orders ADD IF NOT EXISTS
  PARTITION (dt = '2016-05-14', country = 'IN');
```

0바이트 **_$folder$** 파일

ALTER TABLE ADD PARTITION 문을 실행할 때 이미 존재하는 파티션과 잘못된 Amazon S3 위치를 지정하는 경우 Amazon S3에 *partition_value*_$folder$ 형식의 0바이트 자리 표시자 파일이 생성됩니다. 이러한 파일은 수동으로 제거해야 합니다.

이 오류가 발생하지 않도록 하려면 다음 예제와 같이 ALTER TABLE ADD PARTITION 문에서 ADD IF NOT EXISTS 구문을 사용합니다.

```
ALTER TABLE table_name ADD IF NOT EXISTS PARTITION […]
```

ALTER TABLE DROP PARTITION

명명된 테이블에 지정된 하나 이상의 파티션을 삭제합니다.

시놉시스

```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION (partition_spec) [, PARTITION
 (partition_spec)]
```

파라미터

[IF EXISTS]

지정된 파티션이 없는 경우 오류 메시지를 억제합니다.

PARTITION (partition_spec)

각각의 partition_spec은(는) partition_col_name = partition_col_value [,...] 형식으로 열 이름/값 조합을 지정합니다.

예제

```
ALTER TABLE orders
DROP PARTITION (dt = '2014-05-14', country = 'IN');
```

```
ALTER TABLE orders
DROP PARTITION (dt = '2014-05-14', country = 'IN'), PARTITION (dt = '2014-05-15',
 country = 'IN');
```

참고

ALTER TABLE DROP PARTITION 문은 한 번에 모든 파티션을 삭제하기 위한 단일 구문을 제공하지 않으며 삭제할 파티션 범위를 지정하는 필터링 기준을 지원하지 않습니다.

차선책으로, AWS Glue API GetPartitions 및 BatchDeletePartition 작업을 스크립팅에 사용할 수 있습니다. GetPartitions 작업은 SQL WHERE 표현식과 같은 복잡한 필터 표현식을 지원합니다. GetPartitions를 사용하여 삭제할 파티션의 필터링된 목록을 만들려면, BatchDeletePartition 작업을 사용하여 25개 단위의 배치로 파티션을 삭제할 수 있습니다.

Important

알려진 문제로 인해 ALTER TABLE DROP PARTITION 문에 잘못된 파티션을 지정하면 AWS Glue에서 테이블의 모든 파티션이 삭제됩니다. 예를 들어 다음 명령문은 지정된 파티션이 없더라도 *my_table* 테이블의 모든 파티션을 삭제합니다. 이 문제를 해결하려면 ALTER TABLE DROP PARTITION 문을 실행하기 전에 파티션 정보를 올바르게 입력해야 합니다.

```
ALTER TABLE my_table DROP IF EXISTS PARTITION(zzz='');
```

ALTER TABLE RENAME PARTITION

파티션 값의 이름을 바꿉니다.

Note

ALTER TABLE RENAME PARTITION은 파티션 열의 이름을 바꾸지 않습니다. AWS Glue 콘솔을 사용하여 파티션 열 이름을 변경할 수 있습니다. 자세한 내용은 이 문서의 후반부에서 AWS Glue의 파티션 열 이름 바꾸기 단원을 참조하세요.

시놉시스

이름이 table_name인 테이블의 경우 partition_spec으로 지정된 파티션 값을 new_partition_spec에서 지정한 값으로 이름을 바꿉니다.

```
ALTER TABLE table_name PARTITION (partition_spec) RENAME TO PARTITION
 (new_partition_spec)
```

파라미터

PARTITION (partition_spec)

각각의 `partition_spec`은(는) `partition_col_name = partition_col_value [,...]` 형식으로 열 이름/값 조합을 지정합니다.

예제

```
ALTER TABLE orders
PARTITION (dt = '2014-05-14', country = 'IN') RENAME TO PARTITION (dt = '2014-05-15',
 country = 'IN');
```

AWS Glue의 파티션 열 이름 바꾸기

AWS Glue 콘솔에서 파티션 열 이름을 바꾸려면 다음 절차를 사용하세요.

AWS Glue 콘솔에서 테이블 파티션 열의 이름 바꾸기

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창에서 테이블을 선택합니다.
3. 테이블 페이지에서 테이블 필터링 검색 상자를 사용하여 변경하려는 테이블을 찾을 수 있습니다.
4. 이름 열에서 변경하려는 테이블의 링크를 선택합니다.
5. 스키마 섹션의 테이블 세부 정보 페이지에서 다음 절차 중 하나를 수행하세요.
   - 이름을 JSON 형식으로 바꾸려면 스키마를 JSON으로 편집을 선택합니다.
   - 이름을 직접 변경하려면 스키마 편집을 선택합니다. 이 절차에서는 스키마 편집을 선택합니다.
6. 이름을 바꾸려는 분할된 열의 확인란을 선택한 다음 편집을 클릭합니다.
7. 스키마 항목 편집 대화 상자의 이름에 파티션 열의 새 이름을 입력합니다.
8. 새 테이블 버전으로 저장을 선택합니다. 이 작업을 수행하면 파티션 열 이름이 업데이트되고 데이터의 물리적 사본을 별도로 만들지 않고도 스키마 변경 기록이 보존됩니다.
9. 테이블 버전을 비교하려면 테이블의 세부 정보 페이지에서 작업을 선택한 다음 버전 비교를 선택합니다.

추가적인 리소스

파티셔닝에 대한 자세한 내용은 Athena에서 데이터 분할 섹션을 참조하세요.

ALTER TABLE REPLACE COLUMNS

LazySimpleSerDe로 생성된 테이블에서 모든 열을 제거하고 지정된 열 집합으로 대체합니다. 선택적 PARTITION 구문을 사용할 때 파티션 메타데이터를 업데이트합니다. ALTER TABLE REPLACE COLUMNS를 사용하여 유지하려는 열만 지정해 열을 삭제할 수도 있습니다.

시놉시스

```
ALTER TABLE table_name
  [PARTITION
   (partition_col1_name = partition_col1_value
   [,partition_col2_name = partition_col2_value][,...])]
  REPLACE COLUMNS (col_name data_type [, col_name data_type, ...])
```

파라미터

PARTITION (partition_col_name = partition_col_value [,...])

지정한 열 이름/값 조합으로 파티션을 지정합니다. 열의 데이터 형식이 문자열인 경우에만 인용 부호로 partition_col_value를 묶습니다.

REPLACE COLUMNS (col_name data_type [,col_name data_type,...])

기존 열을 지정된 열 이름 및 데이터 유형으로 바꿉니다.

참고

- ALTER TABLE REPLACE COLUMNS 실행 후 Athena 쿼리 편집기 탐색 창에서 테이블 열의 변경을 확인하려면 편집기의 테이블 목록을 수동으로 새로 고친 다음 테이블을 다시 확장해야 할 수 있습니다.
- ALTER TABLE REPLACE COLUMNS는 date 데이터 형식을 가진 열에 대해 작동하지 않습니다. 이 문제를 해결하려면 테이블에서 timestamp 데이터 형식을 대신 사용하세요.
- 단 하나의 열만 바꾸는 경우에도 구문은 ALTER TABLE *table-name* REPLACE COLUMNS와 같이 columns가 복수형이어야 합니다. 바꾸려는 열뿐만 아니라 유지하려는 열도 지정해야 합니다. 그렇지 않으면 지정하지 않은 열이 삭제됩니다. 이 구문과 동작은 Apache Hive DDL에서 파생됩니다. Apache 설명서에서 열 추가/바꾸기를 참조하세요.

예

다음 예에서 LazySimpleSerDe를 사용하여 만든 테이블 names_cities에는 col1, col2 및 col3이라는 세 개의 열이 있습니다. 모든 열은 string 형식입니다. 테이블의 열을 표시하기 위해 다음 명령에서는 SHOW COLUMNS 문을 사용합니다.

```
SHOW COLUMNS IN names_cities
```

쿼리의 결과:

```
col1
col2
col3
```

다음 ALTER TABLE REPLACE COLUMNS 명령은 열 이름을 first_name, last_name 및 city로 바꿉니다. 기본 소스 데이터는 영향을 받지 않습니다.

```
ALTER TABLE names_cities
REPLACE COLUMNS (first_name string, last_name string, city string)
```

결과를 테스트하기 위해 SHOW COLUMNS를 다시 실행합니다.

```
SHOW COLUMNS IN names_cities
```

쿼리의 결과:

```
first_name
last_name
city
```

새 열 이름을 표시하는 또 다른 방법은 Athena 쿼리 편집기에서 표를 미리 보거나 자체 SELECT 쿼리를 실행하는 것입니다.

ALTER TABLE SET LOCATION

table_name이라는 테이블의 위치를 변경하고, 선택적으로 partition_spec인 파티션을 변경합니다.

시놉시스

```
ALTER TABLE table_name [ PARTITION (partition_spec) ] SET LOCATION 'new location'
```

파라미터

PARTITION (partition_spec)

`partition_spec` 파라미터로 위치를 변경하고자 하는 파티션을 지정합니다. `partition_spec`은 `partition_col_name = partition_col_value` 형식으로 열 이름/값 조합을 지정합니다.

SET LOCATION '새 위치'

새 위치를 지정합니다. 이 때 위치는 Amazon S3 위치여야 합니다. 구문에 대한 자세한 내용은 Amazon S3에서의 테이블 위치를 참조하세요.

예제

```
ALTER TABLE customers PARTITION (zip='98040', state='WA') SET LOCATION 's3://DOC-
EXAMPLE-BUCKET/custdata/';
```

ALTER TABLE SET TBLPROPERTIES

테이블에 사용자 지정 또는 미리 정의된 메타데이터 속성을 추가하고 할당된 값을 설정합니다. 테이블의 속성을 보려면 SHOW TBLPROPERTIES 명령을 사용합니다.

Apache Hive 관리형 테이블이 지원되지 않으므로 `'EXTERNAL'='FALSE'` 설정은 효과가 없습니다.

시놉시스

```
ALTER TABLE table_name SET TBLPROPERTIES ('property_name' = 'property_value' [ , ... ])
```

파라미터

SET TBLPROPERTIES ('property_name' = 'property_value' [, ...])

추가할 메타데이터 속성을 `property_name`으로 지정하고 각각의 값을 `property value`로 지정합니다. `property_name`이 이미 존재하는 경우, 해당 값은 새로 지정된 `property_value`로 설정됩니다.

다음의 미리 정의된 테이블 속성은 특별한 용도로 사용됩니다.

미리 정의된 속성	설명
`classification`	AWS Glue의 데이터 형식을 나타냅니다. 가능한 값은 `csv`, `parquet`, `orc`, `avro` 또는 `json`입니다. CloudTrail 콘솔에서 Athena용으로 생성된 테이블은 `classification` 속성 값으로 `cloudtrail` 을 추가합니다. 자세한 내용은 CREATE TABLE의 TBLPROPERTIES 단원을 참조하세요.
`has_encrypted_data`	`LOCATION`에 의해 지정된 데이터 집합이 암호화되었는지 여부를 나타냅니다. 자세한 내용은 CREATE TABLE 및 Amazon S3의 암호화된 데이터 세트에 기반한 테이블 생성의 TBLPROPERTIES 단원을 참조하세요.
`orc.compress`	ORC 형식의 데이터에 대해 압축 형식을 지정합니다. 자세한 내용은 ORC SerDe 단원을 참조하세요.
`parquet.compression`	Parquet 형식의 데이터에 대해 압축 형식을 지정합니다. 자세한 내용은 Parquet SerDe 단원을 참조하십시오.
`write.compression`	텍스트 파일 또는 JSON 형식의 데이터에 대한 압축 형식을 지정합니다. Parquet 및 ORC 형식의 경우 각각 `parquet.compression` 및 `orc.compress` 속성을 사용합니다.
`compression_level`	사용할 압축 수준을 지정합니다. 이 속성은 ZSTD 압축에만 적용됩니다. 가능한 값은 1~22입니다. 기본값은 3입니다. 자세한 내용은 Athena에서 ZSTD 압축 수준 사용 단원을 참조하십시오.
`projection.*`	파티션 프로젝션에 이러한 사용자 지정 속성을 사용하면 Athena가 테이블에서 쿼리를 실행할 때 예상되는 파티션 패턴을 알 수 있습니다. 자세한 내용은 Amazon Athena를 사용한 파티션 프로젝션 단원을 참조하세요.
`skip.header.line.count`	테이블을 정의할 때 데이터의 헤더를 무시합니다. 자세한 내용은 헤더 무시 단원을 참조하세요.
`storage.location.template`	프로젝션된 파티션에 대한 사용자 지정 Amazon S3 경로 템플릿을 지정합니다. 자세한 내용은 파티션 프로젝션 설정 단원을 참조하십시오.

예제

다음 예제에서는 테이블 속성에 주석 메모를 추가합니다.

```
ALTER TABLE orders
SET TBLPROPERTIES ('notes'="Please don't drop this table.");
```

다음 예제에서는 ZSTD 압축 및 ZSTD 압축 수준 4와 함께 Parquet 파일 형식을 사용하도록 테이블 `existing_table`을 수정합니다.

```
ALTER TABLE existing_table
SET TBLPROPERTIES ('parquet.compression' = 'ZSTD', 'compression_level' = 4)
```

CREATE DATABASE

데이터베이스를 생성합니다. `DATABASE`와 `SCHEMA`는 동일한 의미로 통용됩니다. 두 용어의 의미는 동일합니다.

> **Note**
>
> 데이터베이스 생성, 테이블 생성 및 Athena의 테이블에서 SELECT 쿼리 실행에 대한 예제는 시작하기 항목을 참조하세요.

시놉시스

```
CREATE {DATABASE|SCHEMA} [IF NOT EXISTS] database_name
  [COMMENT 'database_comment']
  [LOCATION 'S3_loc']
  [WITH DBPROPERTIES ('property_name' = 'property_value') [, ...]]
```

파라미터

[IF NOT EXISTS]

`database_name`이라는 데이터베이스가 이미 있으면 오류가 억제되도록 합니다.

[COMMENT database_comment]

`comment`라는 내장 메타데이터 속성의 메타데이터 값과 `database_comment`에 입력하는 값을 설정합니다. AWS Glue에서 `COMMENT` 콘텐츠는 데이터베이스 속성의 `Description` 필드에 작성됩니다.

[LOCATION S3_loc]

데이터베이스 파일과 메타 스토어가 존재할 위치를 S3_loc로 지정합니다. 이 위치는 Amazon S3 위치여야 합니다.

[WITH DBPROPERTIES ('property_name' = 'property_value') [, ...]]

데이터베이스 정의에 사용자 지정 메타데이터 속성을 지정할 수 있습니다.

예제

```
CREATE DATABASE clickstreams;
```

```
CREATE DATABASE IF NOT EXISTS clickstreams
  COMMENT 'Site Foo clickstream data aggregates'
  LOCATION 's3://DOC-EXAMPLE-BUCKET/clickstreams/'
  WITH DBPROPERTIES ('creator'='Jane D.', 'Dept.'='Marketing analytics');
```

데이터베이스 속성 보기

CREATE DATABASE를 사용해 AWSDataCatalog에 생성한 데이터베이스의 데이터베이스 속성을 보려면 다음 예시처럼 AWS CLI 명령 aws glue get-database를 사용합니다.

```
aws glue get-database --name <your-database-name>
```

JSON 출력에서 결과가 다음과 같이 보입니다.

```
{
    "Database": {
        "Name": "<your-database-name>",
        "Description": "<your-database-comment>",
        "LocationUri": "s3://DOC-EXAMPLE-BUCKET",
        "Parameters": {
            "<your-database-property-name>": "<your-database-property-value>"
        },
        "CreateTime": 1603383451.0,
        "CreateTableDefaultPermissions": [
            {
                "Principal": {
                    "DataLakePrincipalIdentifier": "IAM_ALLOWED_PRINCIPALS"
                },
                "Permissions": [
```

```
                    "ALL"
                ]
            }
        ]
    }
}
```

AWS CLI에 대한 자세한 내용은 AWS Command Line Interface 사용 설명서를 참조하세요.

CREATE TABLE

지정한 파라미터와 이름으로 테이블을 생성합니다.

> **Note**
>
> 이 페이지에는 요약 참조 정보가 포함되어 있습니다. Athena에서 테이블을 생성하는 방법과 예제 CREATE TABLE 문에 대한 자세한 내용은 Athena에서 테이블 생성 단원을 참조하세요. 데이터베이스 생성, 테이블 생성 및 Athena의 테이블에서 SELECT 쿼리 실행에 대한 예제는 시작하기 항목을 참조하세요.

시놉시스

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
 [db_name.]table_name [(col_name data_type [COMMENT col_comment] [, ...] )]
 [COMMENT table_comment]
 [PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
 [CLUSTERED BY (col_name, col_name, ...) INTO num_buckets BUCKETS]
 [ROW FORMAT row_format]
 [STORED AS file_format]
 [WITH SERDEPROPERTIES (...)]
 [LOCATION 's3://DOC-EXAMPLE-BUCKET/[folder]/']
 [TBLPROPERTIES ( ['has_encrypted_data'='true | false',]
 ['classification'='aws_glue_classification',] property_name=property_value [, ...] ) ]
```

파라미터

EXTERNAL

테이블이 사용자가 지정한 LOCATION의 Amazon S3에 있는 기본 데이터 파일을 기반으로 하도록 지정합니다. Iceberg 테이블을 생성할 때를 제외하고 항상 EXTERNAL 키워드를 사용하세요.

Iceberg가 아닌 테이블에 대해 EXTERNAL 키워드 없이 CREATE TABLE을 사용하는 경우 Athena에서 오류가 발생합니다. 외부 테이블을 만들 때 참조되는 데이터는 기본 형식 또는 ROW FORMAT, STORED AS, WITH SERDEPROPERTIES 절로 지정한 형식을 준수해야 합니다.

[IF NOT EXISTS]

이 파라미터는 동일한 이름의 테이블이 이미 존재하는지 확인합니다. 존재하는 경우 파라미터는 TRUE를 반환하고 Amazon Athena는 CREATE TABLE 작업을 취소합니다. Athena가 데이터 카탈로그를 직접적으로 호출하기 전에 취소가 발생하므로 AWS CloudTrail 이벤트를 내보내지 않습니다.

[db_name.]table_name

테이블의 이름이 생성되도록 지정합니다. 옵션 db_name 파라미터는 테이블이 있는 데이터베이스를 지정합니다. 생략하면 현재 데이터베이스로 가정됩니다. 테이블 이름에 숫자가 포함되어 있는 경우 table_name을 인용 부호로 묶습니다(예: "table123"). table_name이 밑줄로 시작하는 경우 백틱을 사용합니다(예: `_mytable`). 밑줄 이외의 특수 문자는 지원되지 않습니다.

Athena 테이블 이름은 대소문자를 구분하지 않으나 Apache Spark로 작업하는 경우 Spark에는 소문자 테이블 이름이 필요합니다.

[(col_name data_type [COMMENT col_comment] [, ...])]

열의 데이터 형식과 함께, 작성될 각 열의 이름을 지정합니다. 열 이름은 밑줄 (_) 이외의 특수 문자를 허용하지 않습니다. col_name이 밑줄로 시작하는 경우 열 이름을 악센트 부호(`)로 묶습니다(예: `_mycolumn`).

data_type 값은 다음 중 하나일 수 있습니다.

- boolean – 값은 true 및 false입니다.
- tinyint - 2의 보수 형식의 부호 있는 8비트 정수이며, 최솟값은 -2^7이고 최댓값은 2^7-1입니다.
- smallint - 2의 보수 형식의 부호 있는 16비트 정수이며, 최솟값은 -2^15이고 최댓값은 2^15-1입니다.
- int - CREATE TABLE과 같은 DDL(데이터 정의 언어) 쿼리에서는 int 키워드를 사용하여 정수를 나타냅니다. 다른 쿼리에서 키워드 integer를 사용합니다. 여기에서 integer는 2의 보수 형식의 부호 있는 32비트 값으로 표현되며, 최솟값은 -2^31, 최댓값은 2^31-1입니다. JDBC 드라이버의 경우 비즈니스 분석 애플리케이션과 호환될 수 있도록 integer이(가) 반환됩니다.
- bigint - 2의 보수 형식의 부호 있는 64비트 정수이며, 최솟값은 -2^63이고 최댓값은 2^63-1입니다.

- `double` – 64비트 부호 포함 배정밀도 부동 소수점 숫자입니다. 범위는 4.94065645841246544e-324d ~ 1.79769313486231570e+308d, 양수 또는 음수입니다. `double`은 부동 소수점 산술에 대한 IEEE 표준(IEEE 754)을 따릅니다.
- `float` – 32비트 부호 포함 단정밀도 부동 소수점 숫자입니다. 범위는 1.40129846432481707e-45 ~ 3.40282346638528860e+38, 양수 또는 음수입니다. `float`는 부동 소수점 산술에 대한 IEEE 표준(IEEE 754)을 따릅니다. Presto의 `real`에 해당합니다. Athena에서는 `CREATE TABLE` 등의 DDL 문에서 `float`을(를) 사용하고, `SELECT CAST` 등의 SQL 함수에서 `real`을(를) 사용합니다. 이 AWS Glue 크롤러는 `float`로 값을 반환하며, Athena는 내부에서 `real` 및 `float` 형식을 번역합니다(2018년 6월 5일 릴리스 정보 참조).
- `decimal [ (precision, scale) ]`, 여기에서 *`precision`*은 총 자릿수이고, *`scale`*(선택 사항)은 소수부의 자릿수이며, 기본값은 0입니다. 예를 들어 `decimal(11,5)`, `decimal(15)` 형식 정의를 사용합니다. *`precision`* 최대값은 38이고, *`scale`* 최대값은 38입니다.

  쿼리 DDL 표현식에서 특정 10진수 값이 있는 행을 선택하는 경우와 같이 10진수 값을 리터럴로 지정하려면 `decimal` 형식 정의를 지정하고, 쿼리에서 10진수 값을 (작은따옴표로 묶인) 리터럴로 나열합니다(예: `decimal_value = decimal '0.12'`).
- `char` – 길이가 1~255자로 지정된 고정 길이 문자 데이터입니다(예: `char(10)`). 자세한 내용은 CHAR Hive 데이터 형식을 참조하세요.
- `varchar` – 길이가 1~65535자로 지정된 가변 길이 문자 데이터입니다(예: `varchar(10)`). 자세한 내용은 VARCHAR Hive 데이터 형식을 참조하세요.
- `string` – 작은따옴표 또는 큰따옴표로 묶인 문자열 리터럴입니다.

  > **Note**
  >
  > 비 문자열 데이터 유형은 Athena에서 `string`으로 캐스팅할 수 없습니다. 대신 `varchar`로 캐스팅합니다.

- `binary` - (Parquet의 데이터용)
- `date` – ISO 형식의 날짜(예: *`YYYY`*`-`*`MM`*`-`*`DD`*)입니다. 예를 들면 `date '2008-09-15'`입니다. 단, OpenCSVSerDe는 예외적으로 1970년 1월 1일 이후 경과된 일 수를 사용합니다. 자세한 내용은 CSV 처리를 위한 OpenCSVSerDe 단원을 참조하세요.
- `timestamp` – `java.sql.Timestamp` 호환 형식의 날짜 및 시간 인스턴트로, 최대 해상도는 밀리초입니다(예: *`yyyy`*`-`*`MM`*`-`*`dd HH`*`:`*`mm`*`:`*`ss`*`[.f...]`). 예를 들면 `timestamp '2008-09-15 03:04:05.324'`입니다. 단, OpenCSVSerDe는 예외적으로 UNIX 숫자 형식의 `TIMESTAMP` 데이터를 사용합니다(예: `1579059880000`). 자세한 내용은 CSV 처리를 위한 OpenCSVSerDe 단원을 참조하세요.

- `array` < data_type >
- `map` < primitive_type, data_type >
- `struct` < col_name : data_type [comment col_comment] [, ...] >

[COMMENT table_comment]

`comment` 테이블 속성을 만들고, 지정한 `table_comment`로 채웁니다.

[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]

`col_name`, `data_type` 및 `col_comment`가 지정되어 있으며 하나 이상의 파티션 열로 분할된 테이블을 생성합니다. 테이블에는 고유한 열 이름과 값 조합으로 구성된 하나 이상의 파티션이 있을 수 있습니다. 지정된 각각의 조합에 별개의 데이터 디렉터리가 생성되어 상황에 따라 쿼리 성능을 개선할 수 있습니다. 분할된 열은 테이블 데이터 자체 내에는 존재하지 않습니다. 테이블 열과 동일한 `col_name`에 대한 값을 사용하는 경우 오류가 발생합니다. 자세한 내용은 데이터 파티셔닝을 참조하세요.

> **Note**
>
> 파티션이 있는 테이블을 만든 후 MSCK REPAIR TABLE 절로 구성된 후속 쿼리를 실행하여 파티션 메타데이터(예: `MSCK REPAIR TABLE cloudfront_logs;`)를 새로 고칩니다. Hive와 호환되지 않는 파티션의 경우 데이터를 쿼리할 수 있도록 파티션을 로드하려면 ALTER TABLE ADD PARTITION을 사용합니다.

[CLUSTERED BY (col_name, col_name, ...) INTO num_buckets BUCKETS]

파티셔닝 여부와 상관없이 지정된 `col_name` 열을 버킷(buckets)이라는 데이터 하위 집합으로 나눕니다. 이 `num_buckets` 파라미터는 생성할 버킷 수를 지정합니다. 버킷팅은 대용량 데이터 세트에 대한 일부 쿼리의 성능을 향상시킬 수 있습니다.

[ROW FORMAT row_format]

해당되는 경우 테이블의 행 형식과 기본 소스 데이터를 지정합니다. `row_format`의 경우 `DELIMITED` 절을 이용해 하나 이상의 구분 기호를 지정하거나 아래 설명된 대로 `SERDE` 절을 사용합니다. `ROW FORMAT`을 생략하거나 `ROW FORMAT DELIMITED`를 지정하면 기본 SerDe가 사용됩니다.

- [DELIMITED FIELDS TERMINATED BY char [ESCAPED BY char]]
- [DELIMITED COLLECTION ITEMS TERMINATED BY char]

- [MAP KEYS TERMINATED BY char]
- [LINES TERMINATED BY char]
- [NULL DEFINED AS char]

  SATES AS 파일 형식이 `TEXTFILE`인 경우 Hive 0.13에서만 사용할 수 있습니다.

--또는--

- SERDE 'serde_name' [WITH SERDEPROPERTIES ("property_name" = "property_value", "property_name" = "property_value" [, ...])]

  `serde_name`은 사용할 SerDe를 나타냅니다. `WITH SERDEPROPERTIES` 절을 사용하면 SerDe에서 허용하는 하나 이상의 사용자 지정 속성을 제공할 수 있습니다.

[STORED AS file_format]

테이블 데이터의 파일 형식을 지정합니다. 지정하지 않을 경우, `TEXTFILE`이 기본값입니다. `file_format` 옵션은 다음과 같습니다.

- SEQUENCEFILE
- TEXTFILE
- RCFILE
- ORC
- PARQUET
- AVRO
- ION
- INPUTFORMAT input_format_classname OUTPUTFORMAT output_format_classname

[LOCATION 's3://DOC-EXAMPLE-BUCKET/[folder]/']

테이블을 생성할 Amazon S3의 기본 데이터 위치를 지정합니다. 위치 경로는 버킷 이름 또는 버킷 이름과 하나 이상의 폴더여야 합니다. 파티션을 사용하는 경우 분할된 데이터의 루트를 지정합니다. 테이블 위치에 대한 자세한 내용은 Amazon S3에서 테이블 위치 단원을 참조하세요. 데이터 형식 및 권한에 대한 자세한 내용은 Athena의 테이블 및 Amazon S3의 데이터에 대한 요구 사항 단원을 참조하세요.

폴더 또는 버킷에 후행 슬래시를 사용합니다. 파일 이름이나 glob 문자를 사용하지 마세요.

다음 사용:

s3://DOC-EXAMPLE-BUCKET/

s3://DOC-EXAMPLE-BUCKET/*folder*/

s3://DOC-EXAMPLE-BUCKET/*folder*/*anotherfolder*/

사용 금지:

s3://DOC-EXAMPLE-BUCKET

s3://DOC-EXAMPLE-BUCKET/*

s3://DOC-EXAMPLE-BUCKET/*mydatafile.dat*

[TBLPROPERTIES (['has_encrypted_data'='true | false',] ['classification'='classification_value',] property_name=property_value [, ...])]

사전 정의된 테이블 속성 외에 테이블 정의에 사용할 사용자 지정 메타데이터 키-값 페어를 지정합니다(예: "comment").

has_encrypted_data - Athena에는 has_encrypted_data라는 속성이 내장되어 있습니다. 이 속성을 true로 설정하면 LOCATION에서 지정한 기본 데이터 세트가 암호화된다는 뜻입니다. 생략할 경우 작업 그룹의 설정이 클라이언트 측 설정을 재정의하지 않으면, false로 간주됩니다. 기본 데이터가 암호화되는데 false로 설정하거나 생략되면, 쿼리 결과가 오류가 됩니다. 자세한 내용은 저장 중 암호화 단원을 참조하십시오.

classification - CloudTrail 콘솔에서 Athena용으로 생성된 테이블은 classification 속성 값으로 cloudtrail을 추가합니다. AWS Glue에서 ETL 작업을 실행하려면 csv, parquet, orc, avro, 또는 json 등 AWS Glue의 데이터 형식을 나타내기 위해 classification 속성이 포함된 테이블을 생성해야 합니다. 예를 들면 'classification'='csv'입니다. 이 속성을 지정하지 않으면 ETL 작업에 실패합니다. 이어서 AWS Glue 콘솔, API 또는 CLI를 사용해 이를 지정할 수 있습니다. 자세한 내용은 AWS Glue 개발자 안내서의 Athena와 함께 ETL에 AWS Glue 작업 사용 및 AWS Glue에 작업 작성을 참조하세요.

compression_level – compression_level 속성은 사용할 압축 수준을 지정합니다. 이 속성은 ZSTD 압축에만 적용됩니다. 가능한 값은 1~22입니다. 기본값은 3입니다. 자세한 내용은 Athena에서 ZSTD 압축 수준 사용 단원을 참조하십시오.

다른 테이블 속성에 대한 자세한 내용은 ALTER TABLE SET TBLPROPERTIES 섹션을 참조하세요.

예제

CREATE TABLE 문을 사용하는 다음 예제는 Amazon S3에 저장된 탭으로 구분된 행성 데이터에 기반한 테이블을 생성합니다.

```
CREATE EXTERNAL TABLE planet_data (
  planet_name string,
  order_from_sun int,
  au_to_sun float,
  mass float,
  gravity_earth float,
  orbit_years float,
  day_length float
  )
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
LOCATION 's3://DOC-EXAMPLE-BUCKET/tsv/'
```

다음 사항에 주의하세요.

- ROW FORMAT DELIMITED 절은 데이터가 특정 문자로 구분됨을 나타냅니다.
- FIELDS TERMINATED BY '\t' 절은 TSV 데이터의 필드가 탭 문자('\t')로 구분되도록 지정합니다.
- STORED AS TEXTFILE 절은 데이터가 Amazon S3에 일반 텍스트 파일로 저장된 데이터임을 나타냅니다.

데이터를 쿼리하려면 다음과 같은 간단한 SELECT 문을 사용할 수 있습니다.

```
SELECT * FROM planet_data
```

예제를 사용하여 Athena에서 자체 TSV 테이블을 생성하려면 테이블과 열 이름을 자체 테이블 및 열의 이름과 데이터 유형으로 바꾸고 TSV 파일이 저장된 Amazon S3 경로를 가리키도록 LOCATION 절을 업데이트합니다.

테이블 생성에 대한 자세한 내용은 Athena에서 테이블 생성 단원을 참조하세요.

CREATE TABLE AS

SELECT 쿼리의 결과로 채워지는 새 테이블을 생성합니다. 빈 테이블을 생성하려면 CREATE TABLE을 사용합니다. CREATE TABLE AS에서는 CREATE TABLE DDL 문을 SELECT DML 문과 결합하므로 기술적으로 DDL과 DML을 모두 포함합니다. CREATE TABLE AS가 다른 DDL 문과 함께 여기에 그룹화되어 있기는 하지만 Athena의 CTAS 쿼리는 Service Quotas을 위해 DML로 처리됩니다. Athena에서 Service Quotas에 대한 자세한 내용은 Service Quotas 섹션을 참조하세요.

> **Note**
>
> CTAS 문의 경우 예상 버킷 소유자 설정이 Amazon S3의 대상 테이블 위치에 적용되지 않습니다. 예상 버킷 소유자 설정은 Athena 쿼리 결과에 대해 지정한 Amazon S3 출력 위치에만 적용됩니다. 자세한 내용은 Athena 콘솔을 사용하여 쿼리 결과 위치 지정 단원을 참조하십시오.

이 참조 주제 외에 CREATE TABLE AS에 대한 추가 정보는 쿼리 결과에서 테이블 생성(CTAS) 섹션을 참조하세요.

주제

- 시놉시스
- CTAS 테이블 속성
- 예제

시놉시스

```
CREATE TABLE table_name
[ WITH ( property_name = expression [, ...] ) ]
AS query
[ WITH [ NO ] DATA ]
```

위치:

WITH (property_name = expression [, ...])

선택적 CTAS 테이블 속성입니다. 일부는 데이터 스토리지 형식과 관련이 있습니다. CTAS 테이블 속성 섹션을 참조하세요.

쿼리

새 테이블을 생성하는 데 사용되는 SELECT 쿼리입니다.

> **Important**
>
> 파티션을 사용하여 쿼리를 생성하려는 경우 SELECT 문에서 열 목록 마지막에 분할된 열의 이름을 지정합니다.

[WITH [NO] DATA]

WITH NO DATA가 사용되는 경우 원래 테이블과 스키마가 동일한 새로운 빈 테이블이 생성됩니다.

> **Note**
>
> 쿼리 결과 출력에 열 머리글을 포함하려면 CTAS 쿼리 대신 간단한 SELECT 쿼리를 사용할 수 있습니다. 쿼리 결과 위치에서 결과를 검색하거나 Athena 콘솔을 사용하여 결과를 직접 다운로드할 수 있습니다. 자세한 내용은 쿼리 결과, 최근 쿼리, 출력 파일 작업 단원을 참조하십시오.

CTAS 테이블 속성

Athena의 각 CTAS 테이블에는 WITH (property_name = expression [, ...])을 사용해 지정하는 선택적 CTAS 테이블 속성 목록이 있습니다. 이런 파라미터 사용에 대한 자세한 정보는 CTAS 쿼리 예제 단원을 참조하세요.

WITH (property_name = expression [, ...],)

table_type = ['HIVE', 'ICEBERG']

선택 사항입니다. 기본값은 HIVE입니다. 결과 테이블의 테이블 유형 지정

예제

```
WITH (table_type ='ICEBERG')
```

external_location = [location]

> **Note**
>
> Iceberg 테이블은 외부 테이블이 아니므로 이 속성은 Iceberg 테이블에 적용되지 않습니다. CTAS 문에서 Iceberg 테이블의 루트 위치를 정의하려면 이 단원의 뒷부분에 설명된 `location` 속성을 사용하세요.

선택 사항입니다. Athena가 Amazon S3에서 CTAS 쿼리를 저장하는 위치입니다.

예제

```
WITH (external_location ='s3://DOC-EXAMPLE-BUCKET/tables/parquet_table/')
```

Athena는 쿼리 결과에 동일한 경로를 두 번 사용하지 않습니다. 위치를 수동으로 지정한 경우, 지정한 Amazon S3 위치에 데이터가 없는지 확인하세요. Athena는 데이터 삭제를 시도하지 않습니다. 동일한 위치를 다시 사용하려는 경우에는 데이터를 수동으로 삭제하세요. 그렇지 않으면 CTAS 쿼리에 실패합니다.

쿼리 결과 위치를 강제하는 작업 그룹에서 `external_location`을 지정한 CTAS 쿼리를 실행할 경우 쿼리가 오류 메시지를 표시하며 실패합니다. 작업 그룹에 지정된 쿼리 결과 위치를 보려면 작업 그룹 세부 정보를 조회하세요.

작업 그룹이 쿼리 결과 위치에 대한 클라이언트 측 설정을 재정의하는 경우 Athena는 다음 위치에 테이블을 생성합니다.

```
s3://DOC-EXAMPLE-BUCKET/tables/query-id/
```

`external_location` 속성을 사용하여 위치를 지정하지 않으며 작업 그룹이 클라이언트 측 설정을 재정의하지 않는 경우, Athena는 쿼리 결과 위치에 대한 클라이언트 측 설정을 사용하여 다음 위치에 테이블을 생성합니다.

```
s3://DOC-EXAMPLE-BUCKET/Unsaved-or-query-name/year/month/date/tables/query-id/
```

is_external = [boolean]

선택 사항입니다. 테이블이 외부 테이블인지 나타냅니다. 기본값은 true입니다. Iceberg 테이블의 경우 이 값을 false로 설정해야 합니다.

예제

```
WITH (is_external = false)
```

location = [location]

Iceberg 테이블에 필요합니다. 쿼리 결과에서 생성될 Iceberg 테이블의 루트 위치를 지정합니다.

예제

```
WITH (location ='s3://DOC-EXAMPLE-BUCKET/tables/iceberg_table/')
```

field_delimiter = [delimiter]

선택 사항으로, 텍스트 기반 데이터 스토리지 형식과 관련이 있습니다. CSV, TSV 및 텍스트 파일의 단일 문자 필드 구분 기호입니다. 예를 들면 `WITH (field_delimiter = ',')`입니다. 현재 CTAS 쿼리에는 여러 문자로 된 필드 구분 기호가 지원되지 않습니다. 필드 구분 기호를 지정하지 않으면 기본적으로 `\001`이 사용됩니다.

format = [storage_format]

CTAS 쿼리 결과에 대한 스토리지 형식입니다(예: `ORC`, `PARQUET`, `AVRO`, `JSON`, `ION` 또는 `TEXTFILE`). Iceberg 테이블의 경우 허용되는 형식은 `ORC`, `PARQUET` 및 `AVRO`입니다. 생략하면 기본값은 `PARQUET`입니다. 이 파라미터의 이름인 `format`은 소문자로 표시되어야 합니다. 그렇지 않으면 CTAS 쿼리에 실패합니다.

예제

```
WITH (format = 'PARQUET')
```

bucketed_by = ARRAY[column_name[,…], bucket_count = [int]]

> **Note**
>
> 이 속성은 Iceberg 테이블에 적용되지 않습니다. Iceberg 테이블의 경우 파티셔닝을 버킷 변환과 함께 사용합니다.

데이터를 버킷팅할 버킷의 배열 목록입니다. 생략하면 Athena는 이 쿼리에 데이터를 버킷팅하지 않습니다.

bucket_count = [int]

> **Note**
>
> 이 속성은 Iceberg 테이블에 적용되지 않습니다. Iceberg 테이블의 경우 파티셔닝을 버킷 변환과 함께 사용합니다.

데이터를 버킷팅할 버킷의 이름입니다. 생략하면 Athena는 데이터를 버킷팅하지 않습니다. 예제

```
CREATE TABLE bucketed_table WITH (
  bucketed_by = ARRAY[column_name],
  bucket_count = 30, format = 'PARQUET',
  external_location ='s3://DOC-EXAMPLE-BUCKET/tables/parquet_table/'
) AS
SELECT
  *
FROM
  table_name
```

partitioned_by = ARRAY[col_name[,…]]

> **Note**
>
> 이 속성은 Iceberg 테이블에 적용되지 않습니다. Iceberg 테이블에 대해 파티션 변환을 사용하려면 이 단원의 뒷부분에 설명된 `partitioning` 속성을 사용하세요.

선택 사항입니다. CTAS 테이블이 분할되는 기준이 되는 열의 배열 목록입니다. 분할된 열의 이름이 SELECT 문의 열 목록 마지막에 나열되어 있는지 확인하세요.

partitioning = ARRAY[partition_transform, ...]

선택 사항입니다. 생성할 Iceberg 테이블의 파티셔닝을 지정합니다. Iceberg는 다양한 파티션 변환과 파티션 진화를 지원합니다. 파티션 변환은 다음 표에 요약되어 있습니다.

변환	설명
year(ts)	매년 하나의 파티션을 생성합니다. 파티션 값은 ts ~ 1970년 1월 1일 사이 연도의 정수 차이입니다.
month(ts)	각 연도에 매월 하나의 파티션을 생성합니다. 파티션 값은 ts ~ 1970년 1월 1일 사이 월의 정수 차이입니다.
day(ts)	각 연도에 매일 하나의 파티션을 생성합니다. 파티션 값은 ts ~ 1970년 1월 1일 사이 날짜의 정수 차이입니다.
hour(ts)	매일 시간당 하나의 파티션을 생성합니다. 파티션 값은 분과 초가 0으로 설정된 타임스탬프입니다.
bucket(x, nbuckets)	데이터를 지정된 수의 버킷으로 해시합니다. 파티션 값은 0과 nbuckets - 1(포함) 사이의 x 정수 해시입니다.
truncate(s, nchars)	파티션 값을 s의 첫 nchars자로 만듭니다.

예제

```
WITH (partitioning = ARRAY['month(order_date)',
                           'bucket(account_number, 10)',
                           'country']))
```

optimize_rewrite_min_data_file_size_bytes = [long]

선택 사항입니다. 데이터 최적화별 구성입니다. 최적화를 위해 지정된 값보다 작은 파일이 포함됩니다. 기본값은 write_target_data_file_size_bytes 값의 0.75배입니다. 이 속성은 Iceberg 테이블에만 적용됩니다. 자세한 내용은 Iceberg 테이블 최적화 단원을 참조하십시오.

예제

```
WITH (optimize_rewrite_min_data_file_size_bytes = 402653184)
```

optimize_rewrite_max_data_file_size_bytes = [long]

선택 사항입니다. 데이터 최적화별 구성입니다. 최적화를 위해 지정된 값보다 큰 파일이 포함됩니다. 기본값은 `write_target_data_file_size_bytes` 값의 1.8배입니다. 이 속성은 Iceberg 테이블에만 적용됩니다. 자세한 내용은 Iceberg 테이블 최적화 단원을 참조하십시오.

예제

```
WITH (optimize_rewrite_max_data_file_size_bytes = 966367641)
```

optimize_rewrite_data_file_threshold = [int]

선택 사항입니다. 데이터 최적화별 구성입니다. 최적화가 필요한 데이터 파일이 주어진 임계값보다 적으면 파일이 재작성되지 않습니다. 이를 통해 더 많은 데이터 파일을 누적하여 대상 크기에 더 가까운 파일을 생성하고 비용 절감을 위해 불필요한 계산을 건너뛸 수 있습니다. 기본값은 5입니다. 이 속성은 Iceberg 테이블에만 적용됩니다. 자세한 내용은 Iceberg 테이블 최적화 단원을 참조하십시오.

예제

```
WITH (optimize_rewrite_data_file_threshold = 5)
```

optimize_rewrite_delete_file_threshold = [int]

선택 사항입니다. 데이터 최적화별 구성입니다. 데이터 파일과 연관된 삭제 파일이 임계값보다 적으면 데이터 파일이 재작성되지 않습니다. 이를 통해 비용 절감을 위해 각 데이터 파일에 대해 더 많은 삭제 파일을 누적할 수 있습니다. 기본값은 2입니다. 이 속성은 Iceberg 테이블에만 적용됩니다. 자세한 내용은 Iceberg 테이블 최적화 단원을 참조하십시오.

예제

```
WITH (optimize_rewrite_delete_file_threshold = 2)
```

vacuum_min_snapshots_to_keep = [int]

선택 사항입니다. Vacuum 관련 구성입니다. 보존할 최신 스냅샷의 최소 수입니다. 기본 값은 1입니다. 이 속성은 Iceberg 테이블에만 적용됩니다. 자세한 내용은 VACUUM 단원을 참조하십시오.

> **Note**
>
> vacuum_min_snapshots_to_keep 속성을 사용하려면 Athena 엔진 버전 3이 필요합니다.

예제

```
WITH (vacuum_min_snapshots_to_keep = 1)
```

vacuum_max_snapshot_age_seconds = [long]

선택 사항입니다. Vacuum 관련 구성입니다. 보존할 스냅샷의 수명을 나타내는 기간(초)입니다. 기본값은 432,000(5일)입니다. 이 속성은 Iceberg 테이블에만 적용됩니다. 자세한 내용은 VACUUM 단원을 참조하십시오.

> **Note**
>
> vacuum_max_snapshot_age_seconds 속성을 사용하려면 Athena 엔진 버전 3이 필요합니다.

예제

```
WITH (vacuum_max_snapshot_age_seconds = 432000)
```

write_compression = [compression_format]

압축을 지정할 수 있는 모든 스토리지 형식에 사용할 압축 유형입니다. compression_format 값은 데이터가 테이블에 기록될 때 사용할 압축을 지정합니다. TEXTFILE, JSON, PARQUET 및 ORC파일 형식에 대한 압축을 지정할 수 있습니다.

예를 들어 format 속성이 PARQUET을 스토리지 형식으로 지정할 경우, write_compression의 값은 Parquet에 대한 압축 형식을 지정합니다. 이 경우 write_compression의 값을 지정하는 것은 parquet_compression의 값을 지정하는 것과 같습니다.

마찬가지로, format 속성이 ORC를 스토리지 형식으로 지정할 경우, write_compression의 값은 ORC에 대한 압축 형식을 지정합니다. 이 경우 write_compression의 값을 지정하는 것은 orc_compression의 값을 지정하는 것과 같습니다.

동일한 CTAS 쿼리에서 여러 압축 형식 테이블 속성을 지정할 수 없습니다. 예를 들어 `write_compression` 및 `parquet_compression` 모두를 동일한 쿼리에서 지정할 수는 없습니다. `write_compression` 및 `orc_compression`에도 동일하게 적용됩니다. 각 파일 형식에 지원되는 압축 유형에 대한 자세한 내용은 Athena 압축 지원 섹션을 참조하세요.

`orc_compression = [compression_format]`

`ORC` 데이터를 테이블에 쓸 때 `ORC` 파일 형식에 사용할 압축 유형입니다. 예를 들면 `WITH (orc_compression = 'ZLIB')`입니다. `ORC` 파일의 청크(ORC Postscript 제외)가 지정한 압축을 사용하여 압축됩니다. 생략하면 `ORC`에 ZLIB 압축이 기본적으로 사용됩니다.

> **Note**
>
> 일관성을 위해 `orc_compression` 대신 `write_compression` 속성을 사용하는 것이 좋습니다. `format` 속성을 사용하여 스토리지 형식을 `ORC`로 지정한 후 `write_compression` 속성을 사용하여 `ORC`가 사용할 압축 형식을 지정합니다.

`parquet_compression = [compression_format]`

Parquet 데이터를 테이블에 쓸 때 Parquet 파일 형식에 사용할 압축 유형입니다. 예를 들면 `WITH (parquet_compression = 'SNAPPY')`입니다. 이 압축은 Parquet 파일 내의 열 청크에 적용됩니다. 생략하면 Parquet에 GZIP 압축이 기본적으로 사용됩니다.

> **Note**
>
> 일관성을 위해 `parquet_compression` 대신 `write_compression` 속성을 사용하는 것이 좋습니다. `format` 속성을 사용하여 스토리지 형식을 `PARQUET`로 지정한 후 `write_compression` 속성을 사용하여 `PARQUET`가 사용할 압축 형식을 지정합니다.

`compression_level = [compression_level]`

사용할 압축 수준입니다. 이 속성은 ZSTD 압축에만 적용됩니다. 가능한 값은 1~22입니다. 기본값은 3입니다. 자세한 내용은 Athena에서 ZSTD 압축 수준 사용 단원을 참조하십시오.

예제

CTAS 쿼리의 예제는 다음 리소스를 참조하세요.

- CTAS 쿼리 예제
- ETL 및 데이터 분석에 CTAS 및 INSERT INTO 사용
- Amazon Athena와 CTAS 문을 활용한 비용 절감 및 성능 개선
- CTAS 및 INSERT INTO를 사용하여 100개 파티션 한도 해결

CREATE VIEW

지정된 SELECT 쿼리에서 새 뷰를 생성합니다. 이 뷰는 향후 쿼리에서 참조할 수 있는 논리적 테이블입니다. 뷰에는 어떤 데이터도 포함되어 있지 않으며 데이터를 작성하지 않습니다. 대신에 뷰에서 지정된 쿼리는 다른 쿼리로 이 뷰를 참조할 때마다 실행됩니다.

> **Note**
>
> 이 주제에서는 참조할 수 있는 요약 정보를 제공합니다. Athena에서의 뷰 사용에 대한 자세한 내용은 뷰 작업 단원을 참조하세요. 뷰 제한에 대한 자세한 내용은 뷰 제한 사항 섹션을 참조하세요.

시놉시스

```
CREATE [ OR REPLACE ] VIEW view_name AS query
```

선택적 OR REPLACE 절을 사용하여 기존 뷰를 바꿔 업데이트할 수 있습니다. 자세한 내용은 뷰 생성 단원을 참조하십시오.

예제

테이블 orders에서 뷰 test를 만들려면 다음과 유사한 쿼리를 사용합니다.

```
CREATE VIEW test AS
SELECT
orderkey,
orderstatus,
totalprice / 2 AS half
FROM orders;
```

테이블 orders에서 뷰 orders_by_date를 만들려면 다음 쿼리를 사용합니다.

```
CREATE VIEW orders_by_date AS
SELECT orderdate, sum(totalprice) AS price
FROM orders
GROUP BY orderdate;
```

기존 뷰를 업데이트하려면 다음과 유사한 예제를 사용합니다.

```
CREATE OR REPLACE VIEW test AS
SELECT orderkey, orderstatus, totalprice / 4 AS quarter
FROM orders;
```

SHOW COLUMNS, SHOW CREATE VIEW, DESCRIBE VIEW 및 DROP VIEW도 참조하세요.

DESCRIBE

지정된 테이블의 파티션 열을 포함하여 하나 이상의 열을 표시합니다. 이 명령은 복잡한 열의 속성을 검사하는 데 유용합니다.

시놉시스

```
DESCRIBE [EXTENDED | FORMATTED] [db_name.]table_name [PARTITION partition_spec]
 [col_name ( [.field_name] | [.'$elem$'] | [.'$key$'] | [.'$value$'] )]
```

> **Important**
>
> 이 문에 대한 구문은 `DESCRIBE table_name`입니다(`DESCRIBE TABLE table_name`가 아님). 후자의 구문을 사용하면 오류 메시지 FAILED: SemanticException [Error 10001]: Table not found table가 표시됩니다.

파라미터

[EXTENDED | FORMATTED]

출력 형식을 결정합니다. 이러한 파라미터를 생략하면 파티션 열을 포함하여 열 이름 및 해당 데이터 형식을 테이블 형식으로 표시합니다. `FORMATTED`를 지정하면 열 이름 및 데이터 형식을 테이블 형식으로 표시할 뿐만 아니라 자세한 테이블 및 스토리지 정보도 표시합니다. `EXTENDED`는 열 및 데이터 형식 정보를 테이블 형식으로 표시하고 테이블의 자세한 메타데이터를 Thrift 직렬화 형식으로 표시합니다. 이 형식은 읽기 더 어렵고 주로 디버깅에 유용합니다.

[PARTITION partition_spec]

포함된 경우 partition_spec에서 지정한 파티션에 대한 메타데이터를 나열합니다. 여기서 partition_spec은 (partition_column = partition_col_value, partition_column = partition_col_value, ...) 형식입니다.

[col_name ([.field_name] | [.'$elem$'] | [.'key'] | [.'$value$'])*]

검사할 열과 속성을 지정합니다. 구문의 요소에 .field_name을, 배열 요소에 '$elem$'을, 맵 키에 'key'를, 맵 값에 '$value$'를 지정할 수 있습니다. 이를 재귀적으로 지정하여 복잡한 열을 더 자세히 탐색할 수 있습니다.

예제

```
DESCRIBE orders
```

```
DESCRIBE FORMATTED mydatabase.mytable PARTITION (part_col = 100) columnA;
```

다음 쿼리 및 출력은 Amazon EMR 샘플 데이터를 기반으로 한 impressions 테이블의 열 및 데이터 형식 정보를 표시합니다.

```
DESCRIBE impressions
```

```
requestbegintime          string                                      from
 deserializer
adid                      string                                      from
 deserializer
impressionid              string                                      from
 deserializer
referrer                  string                                      from
 deserializer
useragent                 string                                      from
 deserializer
usercookie                string                                      from
 deserializer
ip                        string                                      from
 deserializer
number                    string                                      from
 deserializer
processid                 string                                      from
 deserializer
```

```
browsercokie                string                                              from
 deserializer
requestendtime              string                                              from
 deserializer
timers                      struct<modellookup:string,requesttime:string>  from
 deserializer
threadid                    string                                              from
 deserializer
hostname                    string                                              from
 deserializer
sessionid                   string                                              from
 deserializer
dt                          string

# Partition Information
# col_name                  data_type                 comment

dt                          string
```

다음 예제 쿼리 및 출력은 FORMATTED 옵션이 사용된 경우 동일한 테이블에 대한 결과를 표시합니다.

```
DESCRIBE FORMATTED impressions
```

```
requestbegintime            string                                              from
 deserializer
adid                        string                                              from
 deserializer
impressionid                string                                              from
 deserializer
referrer                    string                                              from
 deserializer
useragent                   string                                              from
 deserializer
usercookie                  string                                              from
 deserializer
ip                          string                                              from
 deserializer
number                      string                                              from
 deserializer
processid                   string                                              from
 deserializer
browsercokie                string                                              from
 deserializer
```

```
requestendtime            string                                          from
 deserializer
timers                    struct<modellookup:string,requesttime:string>  from
 deserializer
threadid                  string                                          from
 deserializer
hostname                  string                                          from
 deserializer
sessionid                 string                                          from
 deserializer
dt                        string

# Partition Information
# col_name                data_type               comment

dt                        string

# Detailed Table Information
Database:                 sampledb
Owner:                    hadoop
CreateTime:               Thu Apr 23 02:55:21 UTC 2020
LastAccessTime:           UNKNOWN
Protect Mode:             None
Retention:                0
Location:                 s3://us-east-1.elasticmapreduce/samples/hive-ads/tables/
impressions
Table Type:               EXTERNAL_TABLE
Table Parameters:
        EXTERNAL                  TRUE
        transient_lastDdlTime     1587610521

# Storage Information
SerDe Library:                        org.openx.data.jsonserde.JsonSerDe
InputFormat:                          org.apache.hadoop.mapred.TextInputFormat
OutputFormat:
 org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat
Compressed:                           No
Num Buckets:                          -1
Bucket Columns:                       []
Sort Columns:                         []
Storage Desc Params:
        paths                                 requestbegintime, adid, impressionid,
 referrer, useragent, usercookie, ip
```

```
        serialization.format                    1
```

다음 예제 쿼리 및 출력은 EXTENDED 옵션이 사용된 경우 동일한 테이블에 대한 결과를 표시합니다. 자세한 테이블 정보는 한 줄로 출력되지만 여기서는 가독성을 위해 형식이 지정됩니다.

```
DESCRIBE EXTENDED impressions
```

```
requestbegintime          string                                              from
 deserializer
adid                      string                                              from
 deserializer
impressionid              string                                              from
 deserializer
referrer                  string                                              from
 deserializer
useragent                 string                                              from
 deserializer
usercookie                string                                              from
 deserializer
ip                        string                                              from
 deserializer
number                    string                                              from
 deserializer
processid                 string                                              from
 deserializer
browsercokie              string                                              from
 deserializer
requestendtime            string                                              from
 deserializer
timers                    struct<modellookup:string,requesttime:string>  from
 deserializer
threadid                  string                                              from
 deserializer
hostname                  string                                              from
 deserializer
sessionid                 string                                              from
 deserializer
dt                        string

# Partition Information
# col_name                data_type                comment

dt                        string
```

```
Detailed Table Information       Table(tableName:impressions, dbName:sampledb,
 owner:hadoop, createTime:1587610521,
lastAccessTime:0, retention:0, sd:StorageDescriptor(cols:
[FieldSchema(name:requestbegintime, type:string, comment:null),
FieldSchema(name:adid, type:string, comment:null), FieldSchema(name:impressionid,
 type:string, comment:null),
FieldSchema(name:referrer, type:string, comment:null), FieldSchema(name:useragent,
 type:string, comment:null),
FieldSchema(name:usercookie, type:string, comment:null), FieldSchema(name:ip,
 type:string, comment:null),
FieldSchema(name:number, type:string, comment:null), FieldSchema(name:processid,
 type:string, comment:null),
FieldSchema(name:browsercokie, type:string, comment:null),
 FieldSchema(name:requestendtime, type:string, comment:null),
FieldSchema(name:timers, type:struct<modellookup:string,requesttime:string>,
 comment:null), FieldSchema(name:threadid,
type:string, comment:null), FieldSchema(name:hostname, type:string, comment:null),
 FieldSchema(name:sessionid,
type:string, comment:null)], location:s3://us-east-1.elasticmapreduce/samples/hive-ads/
tables/impressions,
inputFormat:org.apache.hadoop.mapred.TextInputFormat,
outputFormat:org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat, compressed:false,
 numBuckets:-1,
serdeInfo:SerDeInfo(name:null, serializationLib:org.openx.data.jsonserde.JsonSerDe,
 parameters:{serialization.format=1,
paths=requestbegintime, adid, impressionid, referrer, useragent, usercookie, ip}),
 bucketCols:[], sortCols:[], parameters:{},
skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[],
 skewedColValueLocationMaps:{}),
storedAsSubDirectories:false), partitionKeys:[FieldSchema(name:dt, type:string,
 comment:null)],
parameters:{EXTERNAL=TRUE, transient_lastDdlTime=1587610521}, viewOriginalText:null,
 viewExpandedText:null,
tableType:EXTERNAL_TABLE)
```

DESCRIBE VIEW

명명된 뷰의 열 목록을 보여 줍니다. 이를 통해 복잡한 뷰의 속성을 검사할 수 있습니다.

시놉시스

```
DESCRIBE [db_name.]view_name
```

예

```
DESCRIBE orders;
```

SHOW COLUMNS, SHOW CREATE VIEW, SHOW VIEWS 및 DROP VIEW도 참조하세요.

DROP DATABASE

카탈로그에서 명명된 데이터베이스를 제거합니다. 데이터베이스에 테이블이 포함되어 있으면 DROP DATABASE를 실행하기 전에 테이블을 삭제하거나 CASCADE 절을 사용해야 합니다. DATABASE와 SCHEMA는 동일한 의미로 통용됩니다. 두 용어의 의미는 동일합니다.

시놉시스

```
DROP {DATABASE | SCHEMA} [IF EXISTS] database_name [RESTRICT | CASCADE]
```

파라미터

[IF EXISTS]

database_name이 없으면 오류가 억제되도록 합니다.

[RESTRICT|CASCADE]

DROP 작업 중에 database_name 내부의 테이블을 어떻게 취급할지 결정합니다. RESTRICT를 지정하면 테이블을 포함한 데이터베이스는 삭제되지 않습니다. 이는 기본 설정 동작입니다. CASCADE를 지정하면 데이터베이스와 그에 속한 테이블이 모두 삭제됩니다.

예제

```
DROP DATABASE clickstreams;
```

```
DROP SCHEMA IF EXISTS clickstreams CASCADE;
```

> **Note**
>
> 이름에 특수 문자(예: my-database)가 있는 데이터베이스를 삭제하려고 시도하면 오류 메시지가 표시될 수 있습니다. 이 문제를 해결하려면 데이터베이스 이름을 백틱(`) 문자로 묶어봅니

다. Athena의 이름 지정 데이터베이스에 대한 자세한 내용은 테이블, 데이터베이스 및 열의 이름 섹션을 참조하세요.

DROP TABLE

table_name이라는 테이블에 대한 메타데이터 테이블 정의를 제거합니다. 외부 테이블을 삭제해도 기본 데이터는 그대로 유지됩니다.

시놉시스

```
DROP TABLE [IF EXISTS] table_name
```

파라미터

[IF EXISTS]

table_name이 없으면 오류가 억제되도록 합니다.

예제

```
DROP TABLE fulfilled_orders
```

```
DROP TABLE IF EXISTS fulfilled_orders
```

Athena 콘솔 쿼리 편집기를 사용하여 밑줄(_) 이외의 특수 문자가 있는 테이블을 삭제하려면 다음 예제와 같이 백틱을 사용합니다.

```
DROP TABLE `my-athena-database-01.my-athena-table`
```

JDBC 커넥터를 사용하여 특수 문자가 있는 테이블을 삭제할 때는 백틱 문자가 필요하지 않습니다.

```
DROP TABLE my-athena-database-01.my-athena-table
```

DROP VIEW

기존 뷰를 삭제합니다. 뷰가 없는 경우 선택적 IF EXISTS 절이 오류가 억제되도록 합니다.

자세한 내용은 뷰 작업 단원을 참조하십시오.

시놉시스

```
DROP VIEW [ IF EXISTS ] view_name
```

예제

```
DROP VIEW orders_by_date
```

```
DROP VIEW IF EXISTS orders_by_date
```

CREATE VIEW, SHOW COLUMNS, SHOW CREATE VIEW, SHOW VIEWS 및 DESCRIBE VIEW도 참조하세요.

MSCK REPAIR TABLE

Hive 호환 파티션을 추가한 후 `MSCK REPAIR TABLE` 명령을 사용하여 카탈로그의 메타데이터를 업데이트합니다.

`MSCK REPAIR TABLE` 명령은 테이블을 생성한 후 파일 시스템에 추가된 Hive 호환 파티션을 위해 Amazon S3 등의 파일 시스템을 스캔합니다. `MSCK REPAIR TABLE`은 테이블 메타데이터의 파티션과 S3의 파티션을 비교합니다. 테이블을 생성할 때 지정한 S3 위치에 새 파티션이 있으면 메타데이터와 Athena 테이블에 해당 파티션이 추가됩니다.

물리적 파티션을 추가하면 카탈로그의 메타데이터가 파일 시스템의 데이터 레이아웃과 일치하지 않게 되므로 새 파티션에 대한 정보를 카탈로그에 추가해야 합니다. 메타데이터를 업데이트하려면 Athena에서 새 파티션의 데이터를 쿼리할 수 있도록 `MSCK REPAIR TABLE`을 실행합니다.

> **Note**
>
> `MSCK REPAIR TABLE`은 메타데이터에 파티션을 추가하기만 합니다. 파티션을 제거하지는 않습니다. Amazon S3에서 파티션을 수동으로 삭제한 후 메타데이터에서 파티션을 제거하려면 `ALTER TABLE table-name DROP PARTITION` 명령을 실행합니다. 자세한 내용은 ALTER TABLE DROP PARTITION 단원을 참조하세요.

고려 사항 및 제한

`MSCK REPAIR TABLE`을 사용할 때는 다음 사항에 유의하세요.

- 모든 파티션을 추가하려면 다소 시간이 걸릴 수 있습니다. 이 작업의 시간이 초과되면 카탈로그에 몇 개의 파티션만 추가되는 불완전한 상태가 됩니다. 모든 파티션이 추가될 때까지 동일한 테이블에서 MSCK REPAIR TABLE을 실행해야 합니다. 자세한 내용은 Athena에서 데이터 분할 단원을 참조하십시오.
- Hive와 호환되지 않는 파티션의 경우 데이터를 쿼리할 수 있도록 파티션을 로드하려면 ALTER TABLE ADD PARTITION을 사용합니다.
- Athena에 사용될 파티션 위치는 s3 프로토콜(예: s3://DOC-EXAMPLE-BUCKET/*folder*/)을 사용해야 합니다. Athena에서, 다른 프로토콜(예: s3a://*bucket*/*folder*/)을 사용하는 위치는 포함 테이블에서 MSCK REPAIR TABLE 쿼리를 실행할 때 쿼리 실패를 초래하게 됩니다.
- MSCK REPAIR TABLE은 일치하는 파티션 스키마를 찾기 위해 폴더와 하위 폴더를 모두 스캔하기 때문에 별도의 폴더 계층 구조에 있는 별도의 테이블에 데이터를 보관해야 합니다. 예를 들어 테이블 1의 데이터를 s3://DOC-EXAMPLE-BUCKET1에 두고 테이블 2에 대한 데이터을 s3://DOC-EXAMPLE-BUCKET1/table-2-data에 두었다고 가정합시다. 두 테이블이 모두 문자열로 분할된 경우 MSCK REPAIR TABLE은 테이블 2의 파티션을 테이블 1에 추가합니다. 이를 방지하려면 대신에 s3://DOC-EXAMPLE-BUCKET1 및 s3://DOC-EXAMPLE-BUCKET2와 같은 별도의 폴더 구조를 사용하세요. 이 동작은 Amazon EMR과 Apache Hive에서도 동일합니다.
- 알려진 문제로 인해 파티션 값에 콜론(MSCK REPAIR TABLE)이 포함된 경우(예: 파티션 값이 타임스탬프인 경우) :이 자동으로 실패합니다. 임시 해결책으로 ALTER TABLE ADD PARTITION을 사용합니다.
- MSCK REPAIR TABLE에서는 밑줄(_)로 시작하는 파티션 열 이름을 추가하지 않습니다. 이 제한을 해결하려면 ALTER TABLE ADD PARTITION을 사용합니다.

시놉시스

```
MSCK REPAIR TABLE table_name
```

예제

```
MSCK REPAIR TABLE orders;
```

문제 해결

MSCK REPAIR TABLE을 실행한 후 Athena가 AWS Glue Data Catalog의 테이블에 파티션을 추가하지 않으면 다음을 확인하세요.

- AWS Glue 액세스 – AWS Identity and Access Management(IAM) 역할에 `glue:BatchCreatePartition` 작업을 허용하는 정책이 있는지 확인합니다. 자세한 내용은 이 문서의 후반부에서 IAM 정책에서 glue:BatchCreatePartition 허용 단원을 참조하세요.
- Amazon S3 액세스 - 역할에 Amazon S3에 액세스할 수 있는 충분한 권한(`s3:DescribeJob` 작업 포함)이 있는 정책이 있는지 확인합니다. 허용할 Amazon S3 작업에 대한 예는 Amazon S3 버킷에 대한 Athena의 계정 간 액세스의 버킷 정책 예시를 참조하세요.
- Amazon S3 객체 키 대소문자 표기 방식 – Amazon S3 경로가 카멜 표기법이 아닌 소문자인지 확인합니다(예: `userId` 대신 `userid`). 또는 `ALTER TABLE ADD PARTITION`을 사용하여 객체 키 이름을 지정합니다. 자세한 내용은 이 문서의 후반부에서 Amazon S3 경로 변경 또는 재정의 단원을 참조하세요.
- 쿼리 시간 제한 – `MSCK REPAIR TABLE`은 처음으로 테이블을 만들거나 데이터와 파티션 메타 데이터 간의 패리티에 대한 불확실성이 있을 때 가장 적합합니다. `MSCK REPAIR TABLE`을 사용해 새 파티션을 자주 추가하는데(예: 날마다) 쿼리 시간 제한이 발생하는 경우 ALTER TABLE ADD PARTITION 사용을 고려하는 것이 좋습니다.
- 파일 시스템에서 파티션 누락 - Amazon S3에서 파티션을 수동으로 삭제한 다음 `MSCK REPAIR TABLE`을 실행하면 Partitions missing from filesystem 오류 메시지가 나타날 수 있습니다. 이는 `MSCK REPAIR TABLE`이 기한 경과된 파티션을 테이블 메타데이터를 제거하지 않았기 때문입니다. 테이블 메타데이터에서 삭제된 파티션을 제거하려면 ALTER TABLE DROP PARTITION을 대신 실행합니다. SHOW PARTITIONS는 비슷하게 파티션을 나열하지만 파일 시스템의 파티션을 제외하고 메타데이터의 파티션만 나열합니다.
- "NullPointerException name is null" 오류

  `TableType` 속성을 지정하지 않고 AWS Glue CreateTable API 작업 또는 AWS CloudFormation `AWS::Glue::Table` 템플릿을 사용하여 Athena에서 사용할 테이블을 만든 다음 `SHOW CREATE TABLE` 또는 `MSCK REPAIR TABLE` 같은 DDL 쿼리를 실행하면, 실패: NullPointerException Name이 null임(FAILED: NullPointerException Name is null)이라는 오류 메시지가 표시될 수 있습니다.

  이 오류를 해결하려면 AWS Glue `CreateTable` API 호출 또는 AWS CloudFormation 템플릿의 일부로 TableInput `TableType` 속성의 값을 지정하세요. `TableType`의 가능한 값은 `EXTERNAL_TABLE` 또는 `VIRTUAL_VIEW`입니다.

  이 요구 사항은 AWS Glue `CreateTable` API 작업 또는 `AWS::Glue::Table` 템플릿을 사용하여 테이블을 만들 때만 적용됩니다. DDL 문이나 AWS Glue 크롤러를 사용하여 Athena용 테이블을 생성할 경우 `TableType` 속성이 자동으로 정의됩니다.

다음 섹션에서 여러 세부 정보를 추가로 제공합니다.

IAM 정책에서 glue:BatchCreatePartition 허용

`MSCK REPAIR TABLE` 실행을 위해 사용 중인 역할에 연결된 IAM 정책을 검토합니다. Athena와 함께 AWS Glue Data Catalog를 사용하는 경우 IAM 정책에서 `glue:BatchCreatePartition` 작업을 허용해야 합니다. `glue:BatchCreatePartition` 작업을 허용하는 IAM 정책의 예는 AWS 관리형 정책: AmazonAthenaFullAccess 단원을 참조하세요.

Amazon S3 경로 변경 또는 재정의

Amazon S3 경로에 있는 하나 이상의 객체 키가 소문자가 아닌 카멜 표기법으로 표시된 경우 `MSCK REPAIR TABLE`에서는 AWS Glue Data Catalog에 파티션을 추가하지 않을 수도 있습니다. 예를 들어 Amazon S3 경로에 객체 키 이름 `userId`가 포함되어 있는 경우 다음 파티션은 AWS Glue Data Catalog에 추가되지 않을 수 있습니다.

```
s3://DOC-EXAMPLE-BUCKET/path/userId=1/

s3://DOC-EXAMPLE-BUCKET/path/userId=2/

s3://DOC-EXAMPLE-BUCKET/path/userId=3/
```

이 문제를 해결하려면 다음 중 한 가지를 사용하십시오.

- Amazon S3 객체 키를 생성하는 경우 카멜 표기법 대신 소문자를 사용합니다.

  ```
  s3://DOC-EXAMPLE-BUCKET/path/userid=1/

  s3://DOC-EXAMPLE-BUCKET/path/userid=2/

  s3://DOC-EXAMPLE-BUCKET/path/userid=3/
  ```

- 다음 예제와 같이 ALTER TABLE ADD PARTITION을 사용하여 위치를 재정의합니다.

  ```
  ALTER TABLE table_name ADD [IF NOT EXISTS]
  PARTITION (userId=1)
  LOCATION 's3://DOC-EXAMPLE-BUCKET/path/userId=1/'
  PARTITION (userId=2)
  LOCATION 's3://DOC-EXAMPLE-BUCKET/path/userId=2/'
  PARTITION (userId=3)
  LOCATION 's3://DOC-EXAMPLE-BUCKET/path/userId=3/'
  ```

Amazon S3 객체 키 이름은 대문자를 사용할 수 있지만 Amazon S3 버킷 이름 자체는 항상 소문자여야 합니다. 자세한 내용은 Amazon S3 사용 설명서의 객체 키 명명 지침 및 버킷 이름 지정 규칙을 참조하세요.

SHOW COLUMNS

지정된 단일 테이블 또는 보기의 열 이름만 표시합니다. 자세한 정보를 확보하려면 대신 AWS Glue Data Catalog를 쿼리합니다. 자세한 내용과 예제는 AWS Glue Data Catalog 쿼리 주제의 다음 섹션을 참조하세요.

- 열 메타데이터(예: 데이터 형식)를 보려면 지정된 테이블 또는 뷰에 대한 열 나열 또는 검색 섹션을 참조하세요.
- `AwsDataCatalog`에서 특정 데이터베이스의 모든 테이블에 대한 모든 열을 보려면 지정된 테이블 또는 뷰에 대한 열 나열 또는 검색 섹션을 참조하세요.
- `AwsDataCatalog`에서 모든 데이터베이스의 모든 테이블에 대한 모든 열을 보려면 모든 테이블에 대한 모든 열 나열 섹션을 참조하세요.
- 데이터베이스의 특정 테이블에서 공통적으로 포함하는 열을 보려면 특정 테이블에서 공통적으로 포함하는 열 나열 섹션을 참조하세요.

시놉시스

```
SHOW COLUMNS {FROM|IN} database_name.table_name
```

```
SHOW COLUMNS {FROM|IN} table_name [{FROM|IN} database_name]
```

`FROM` 및 `IN` 키워드는 서로 바꿔 사용할 수 있습니다. *`table_name`* 또는 *`database_name`*에 하이픈 같은 특수 문자가 있는 경우 이름을 억음 부호(backquote)로 둘러쌉니다(예: `` `my-database`.`my-table` ``). *`table_name`* 또는 *`database_name`*을 작은따옴표나 큰따옴표로 둘러싸면 안 됩니다. 현재 `LIKE`의 사용 및 패턴 일치 표현식은 지원되지 않습니다.

예제

다음의 동일한 예제는 `customers` 데이터베이스에 있는 `orders` 테이블의 열을 보여줍니다. 처음 두 예제는 `customers`가 현재 데이터베이스라고 가정합니다.

```
SHOW COLUMNS FROM orders
```

```
SHOW COLUMNS IN orders
```

```
SHOW COLUMNS FROM customers.orders
```

```
SHOW COLUMNS IN customers.orders
```

```
SHOW COLUMNS FROM orders FROM customers
```

```
SHOW COLUMNS IN orders IN customers
```

SHOW CREATE TABLE

`table_name`이라는 기존 테이블을 분석하여 이를 작성한 쿼리를 생성합니다.

시놉시스

```
SHOW CREATE TABLE [db_name.]table_name
```

파라미터

TABLE [db_name.]table_name

db_name 파라미터는 선택 항목입니다. 생략하면 컨텍스트의 기본값은 현재 데이터베이스입니다.

> **Note**
> 테이블 이름은 필수 항목입니다.

예제

```
SHOW CREATE TABLE orderclickstoday;
```

```
SHOW CREATE TABLE `salesdata.orderclickstoday`;
```

문제 해결

TableType 속성을 지정하지 않고 AWS Glue CreateTable API 작업 또는 AWS CloudFormation AWS::Glue::Table 템플릿을 사용하여 Athena에서 사용할 테이블을 만든 다음 SHOW CREATE TABLE 또는 MSCK REPAIR TABLE 같은 DDL 쿼리를 실행하면, 실패: NullPointerException Name이 null임(FAILED: NullPointerException Name is null)이라는 오류 메시지가 표시될 수 있습니다.

이 오류를 해결하려면 AWS Glue CreateTable API 호출 또는 AWS CloudFormation 템플릿의 일부로 TableInput TableType 속성의 값을 지정하세요. TableType의 가능한 값은 EXTERNAL_TABLE 또는 VIRTUAL_VIEW입니다.

이 요구 사항은 AWS Glue CreateTable API 작업 또는 AWS::Glue::Table 템플릿을 사용하여 테이블을 만들 때만 적용됩니다. DDL 문이나 AWS Glue 크롤러를 사용하여 Athena용 테이블을 생성할 경우 TableType 속성이 자동으로 정의됩니다.

SHOW CREATE VIEW

지정된 뷰를 생성하는 SQL 문을 보여 줍니다.

시놉시스

```
SHOW CREATE VIEW view_name
```

예제

```
SHOW CREATE VIEW orders_by_date
```

CREATE VIEW 및 DROP VIEW도 참조하십시오.

SHOW DATABASES

메타스토어에 정의된 모든 데이터베이스를 나열합니다. DATABASES 또는 SCHEMAS를 사용할 수 있습니다. 두 용어의 의미는 동일합니다.

SHOW DATABASES는 프로그래밍 방식으로는 ListDatabases Athena API 작업과 동일합니다. AWS SDK for Python (Boto3)에서 동일한 메서드는 list_databases입니다.

시놉시스

```
SHOW {DATABASES | SCHEMAS} [LIKE 'regular_expression']
```

파라미터

[LIKE '*regular_expression*']

지정한 *regular_expression*과 일치하는 기준으로 데이터베이스 목록을 필터링합니다. 와일드카드 문자 일치의 경우 문자 0에서 무제한으로 일치시키는 조합 `.*`를 사용할 수 있습니다.

예제

```
SHOW SCHEMAS;
```

```
SHOW DATABASES LIKE '.*analytics';
```

SHOW PARTITIONS

정렬되지 않은 순서로 Athena 테이블의 모든 파티션을 나열합니다.

시놉시스

```
SHOW PARTITIONS table_name
```

- 테이블에 파티션을 표시하고 특정 순서로 나열하려면 특정 테이블에 대한 파티션 나열 페이지의 AWS Glue Data Catalog 쿼리 섹션을 참조하세요.
- 파티션의 내용을 보려면 데이터 쿼리 페이지의 Athena에서 데이터 분할 섹션을 참조하세요.
- `SHOW PARTITIONS`는 Athena에 의해 프로젝션되었지만 AWS Glue 카탈로그에 등록되지 않은 파티션은 나열하지 않습니다. 파티션 프로젝션에 대한 자세한 내용은 Amazon Athena를 사용한 파티션 프로젝션 단원을 참조하세요.
- `SHOW PARTITIONS`는 실제 파일 시스템의 파티션이 아니라 메타데이터의 파티션을 나열합니다. Amazon S3에서 파티션을 수동으로 삭제한 후 메타데이터를 업데이트하려면 ALTER TABLE DROP PARTITION을 실행합니다.

예제

다음 쿼리 예제는 미국 교통국의 항공편 테이블 데이터를 보여주는 `flight_delays_csv` 테이블의 파티션을 보여줍니다. `flight_delays_csv` 테이블 예제에 대한 자세한 내용은 CSV, TSV, 사용자 지정 구분 기호로 구분된 파일에 대한 LazySimpleSerDe 단원을 참조하세요. 테이블은 연도별로 분할되어 있습니다.

```
SHOW PARTITIONS flight_delays_csv
```

결과

```
year=2007
year=2015
year=1999
year=1993
year=1991
year=2003
year=1996
year=2014
year=2004
year=2011
...
```

다음 쿼리 예제는 샘플 웹 검색 데이터가 들어 있는 `impressions` 테이블의 파티션을 보여줍니다. `impressions` 테이블 예제에 대한 자세한 내용은 Athena에서 데이터 분할 단원을 참조하세요. 테이블은 dt(datetime) 열을 기준으로 분할되어 있습니다.

```
SHOW PARTITIONS impressions
```

결과

```
dt=2009-04-12-16-00
dt=2009-04-13-18-15
dt=2009-04-14-00-20
dt=2009-04-12-13-00
dt=2009-04-13-02-15
dt=2009-04-14-12-05
dt=2009-04-14-06-15
dt=2009-04-12-21-15
dt=2009-04-13-22-15
...
```

정렬된 순서로 파티션 목록 표시

결과 목록에서 파티션을 정렬하려면 `SHOW PARTITIONS` 대신에 다음 `SELECT` 구문을 사용합니다.

```
SELECT * FROM database_name."table_name$partitions" ORDER BY column_name
```

다음 쿼리는 flight_delays_csv 예제의 파티션 목록을 정렬된 순서로 보여줍니다.

```
SELECT * FROM "flight_delays_csv$partitions" ORDER BY year
```

결과

```
year
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
...
```

자세한 내용은 특정 테이블에 대한 파티션 나열 페이지의 AWS Glue Data Catalog 쿼리 단원을 참조하세요.

SHOW TABLES

데이터베이스의 모든 기본 테이블과 뷰를 나열합니다.

시놉시스

```
SHOW TABLES [IN database_name] ['regular_expression']
```

파라미터

[IN database_name]

테이블을 나열할 database_name을 지정합니다. 생략하면 현재 컨텍스트의 데이터베이스로 가정됩니다.

> **Note**
>
> database_name에서 하이픈과 같이 지원되지 않는 문자를 사용할 경우 SHOW TABLES가 실패할 수도 있습니다. 해결 방법으로 데이터베이스 이름을 백틱으로 묶어봅니다.

['regular_expression']

지정한 regular_expression과 일치하는 기준으로 테이블 목록을 필터링합니다. AWSDataCatalog 테이블에서 문자를 나타내는 데 * 또는 .* 와일드카드 표현식을 사용할 수 있습니다. Apache Hive 데이터베이스의 경우 .* 와일드카드 표현식을 사용합니다. 문자 사이의 선택을 나타내려면 | 문자를 사용합니다.

예제

Example - **sampledb** 데이터베이스의 모든 테이블을 표시합니다

```
SHOW TABLES IN sampledb
```

Results

```
alb_logs
cloudfront_logs
elb_logs
flights_2016
flights_parquet
view_2016_flights_dfw
```

Example - “flights”라는 단어를 포함하는 **sampledb**의 모든 테이블 이름을 표시합니다

```
SHOW TABLES IN sampledb '*flights*'
```

Results

```
flights_2016
flights_parquet
view_2016_flights_dfw
```

Example - “logs”라는 단어로 끝나는 **sampledb**의 모든 테이블 이름을 표시합니다

```
SHOW TABLES IN sampledb '*logs'
```

Results

```
alb_logs
cloudfront_logs
elb_logs
```

SHOW TBLPROPERTIES

명명된 테이블의 테이블 속성을 나열합니다.

시놉시스

```
SHOW TBLPROPERTIES table_name [('property_name')]
```

파라미터

[('property_name')]

포함된 경우 property_name이라는 속성의 값만 나열됩니다.

예제

```
SHOW TBLPROPERTIES orders;
```

```
SHOW TBLPROPERTIES orders('comment');
```

SHOW VIEWS

지정된 데이터베이스 또는 현재 데이터베이스(데이터베이스 이름이 생략된 경우)의 뷰를 나열합니다. 선택적 LIKE 절을 정규식과 함께 사용하여 뷰 이름 목록을 제한합니다.

Athena는 각 값이 뷰 이름인 STRING 입력 값 목록을 반환합니다.

시놉시스

```
SHOW VIEWS [IN database_name] [LIKE 'regular_expression']
```

파라미터

[IN database_name]

뷰를 나열할 `database_name`을 지정합니다. 생략하면 현재 컨텍스트의 데이터베이스로 가정됩니다.

[LIKE 'regular_expression']

지정한 `regular_expression`과 일치하는 기준으로 뷰 목록을 필터링합니다. 임의의 문자를 나타내는 `*` 와일드카드 또는 문자 사이의 선택을 나타내는 `|`만 사용할 수 있습니다.

예제

```
SHOW VIEWS;
```

```
SHOW VIEWS IN marketing_analytics LIKE 'orders*'
```

SHOW COLUMNS, SHOW CREATE VIEW, DESCRIBE VIEW 및 DROP VIEW도 참조하세요.

Amazon Athena의 SQL 쿼리에 대한 고려 사항 및 제한 사항

Athena에서 쿼리를 실행할 때 다음 고려 사항 및 제한에 유의하세요.

- 저장 프로시저 – 저장 프로시저는 지원되지 않습니다.
- 최대 파티션 수 - `CREATE TABLE AS SELECT`(CTAS) 문으로 생성할 수 있는 최대 파티션 수는 100개입니다. 자세한 내용은 CREATE TABLE AS를 참조하세요. 해결 방법은 CTAS 및 INSERT INTO를 사용하여 100개 파티션 한도 해결 단원을 참조하세요.
- 지원되지 않는 문 - 다음 문은 지원되지 않습니다.
  - `CREATE TABLE LIKE`는 지원되지 않습니다.
  - `DESCRIBE INPUT` 및 `DESCRIBE OUTPUT`은(는) 지원되지 않습니다.
  - `MERGE` 문은 트랜잭션 테이블 형식에 대해서만 지원됩니다. 자세한 내용은 MERGE INTO 단원을 참조하십시오.
  - `UPDATE` 설명은 지원되지 않습니다.
- Trino 및 Presto 커넥터 – Trino 또는 Presto 커넥터는 지원되지 않습니다. Amazon Athena 연합 쿼리를 사용하여 데이터 소스를 연결합니다. 자세한 내용은 Amazon Athena 연합 쿼리 사용 단원을 참조하십시오.

- 파티션이 많은 테이블의 시간 초과 - 수천 개의 파티션이 있는 테이블을 쿼리하면 Athena가 시간 초과될 수 있습니다. 이 문제는 테이블에 `string` 유형이 아닌 파티션이 많을 때 발생할 수 있습니다. `string` 유형을 사용하면 Athena가 메타스토어 수준에서 파티션을 제거합니다. 그러나 다른 데이터 형식을 사용하면 Athena가 서버 측 파티션을 정리합니다. 파티션이 많을수록 이 프로세스가 오래 걸리며 쿼리 시간이 초과될 가능성이 높아집니다. 이 문제를 해결하려면 Athena가 메타스토어 수준에서 파티션을 정리하도록 파티션 유형을 `string`으로 설정하세요. 이렇게 하면 오버헤드가 줄어들고 쿼리가 시간 초과되는 것을 방지할 수 있습니다.
- S3 Glacier 지원 - 복원된 Amazon S3 Glacier 객체를 쿼리하는 방법에 대한 자세한 내용은 복원된 Amazon S3 Glacier 객체 쿼리 섹션을 참조하세요.
- 숨김으로 처리된 파일 - Athena는 밑줄(_) 또는 점(`.`)으로 시작하는 소스 파일을 숨김으로 처리합니다. 이 제한을 해결하려면 파일 이름을 바꿉니다.
- 행 또는 열 크기 제한 - 하나의 행 또는 해당 열의 크기는 32MB를 초과할 수 없습니다. 예를 들어 CSV 또는 JSON 파일의 행에 300MB의 한 열이 포함된 경우 이 한도를 초과할 수 있습니다. 이 제한을 초과하면 텍스트 파일의 줄이 너무 깁니다라는 오류 메시지가 나타날 수도 있습니다. 이 제한을 해결하려면 모든 행의 열에 있는 데이터 합계가 32MB 미만이어야 합니다.
- LIMIT 절 최댓값 - `LIMIT` 절에 대해 지정할 수 있는 최대 행 수는

  9223372036854775807. `ORDER BY`를 사용하는 경우 LIMIT 절에 대해 지원되는 최대 행 수는 2147483647입니다. 이 제한을 초과하면 오류 메시지 NOT_SUPPORTED: ORDER BY LIMIT > 2147483647 is not supported가 표시됩니다.
- information_schema – AWS Glue 메타데이터의 양이 소량에서 중간 정도라면 `information_schema` 쿼리가 가장 성능이 뛰어납니다. 메타데이터의 양이 많으면 오류가 발생할 수 있습니다. `information_schema` 데이터베이스에 AWS Glue 메타데이터를 쿼리하는 방법에 대한 자세한 내용은 AWS Glue Data Catalog 쿼리 단원을 참조하세요.
- 배열 초기화 – Java 관련 제한으로 인해 Athena에서 인수가 254개를 초과하는 배열은 초기화할 수 없습니다.
- 숨겨진 메타데이터 열 - Hive 또는 Iceberg의 숨겨진 메타데이터 열 `$bucket`, `$file_modified_time`, `$file_size` 및 `$partition`은 보기에서 지원되지 않습니다. Athena에서 `$path` 메타데이터 열 사용에 대한 자세한 내용은 Amazon S3의 소스 데이터에 대한 파일 위치 가져오기 섹션을 참조하세요.

최대 쿼리 문자열 길이, 쿼리 시간 제한에 대한 할당량 및 활성 DML 쿼리 수에 대한 할당량에 관한 자세한 내용은 Service Quotas 단원을 참조하세요.

Athena의 문제 해결

Athena 팀은 고객이 제기한 문제로부터 다음과 같은 문제 해결 정보를 수집했습니다. 종합적이지는 않지만 일반적인 성능, 시간 초과, 메모리 부족 문제에 대한 조언이 포함되어 있습니다.

주제

CREATE TABLE AS SELECT(CTAS)

Duplicated data occurs with concurrent CTAS statements

Athena는 CTAS에 대한 동시 검증을 관리하지 않습니다. 동시에 같은 위치에 대해 실행되는 중복된 CTAS 문이 있는지 확인하세요. CTAS 또는 INSERT INTO 문이 실패하더라도 분리된(orphaned) 데이터가 해당 문에 지정된 데이터 위치에 남아있을 수 있습니다.

HIVE_TOO_MANY_OPEN_PARTITIONS

CTAS 문을 사용하여 파티션이 100개 이상인 테이블을 만들면 "HIVE_TOO_MANY_OPEN_PARTITIONS: 파티션/버킷에 대해 열린 작성자 100개를 초과했습니다" 오류가 발생할 수 있습니다. 이러한 제한은 CTAS 문(최대 100개의 파티션 생성) 및 일련의 INSERT INTO 문(각각 최대 100개의 파티션 삽입)을 사용하여 해결할 수 있습니다. 자세한 내용은 CTAS 및 INSERT INTO를 사용하여 100개 파티션 한도 해결 단원을 참조하십시오.

데이터 파일 문제

Athena cannot read hidden files

Athena는 밑줄(_) 또는 점(.)으로 시작하는 소스 파일을 숨김으로 처리합니다. 이 제한을 해결하려면 파일 이름을 바꿉니다.

Athena reads files that I excluded from the AWS Glue crawler

Athena는 사용자가 AWS Glue 크롤러에 지정하는 제외 패턴을 인식하지 못합니다. 예를 들어 .csv 및 .json 파일이 모두 포함된 Amazon S3 버킷이 있는데 .json 파일을 크롤러에서 제외한다면 Athena는 두 파일 그룹을 모두 쿼리합니다. 이 문제를 방지하려면 제외할 파일을 다른 위치에 배치하면 됩니다.

HIVE_BAD_DATA: Error parsing field value

다음과 같은 상황에서 이 오류가 발생할 수 있습니다.

- 테이블에 정의된 데이터 형식이 원본 데이터와 일치하지 않거나 한 필드에 서로 다른 형식의 데이터가 포함된 경우. 권장되는 해결 방법은 AWS 지식 센터의 "HIVE_BAD_DATA: 필드 X: 입력 문자열: "12312845691"에 대해 필드 값을 구문 분석하는 동안 오류 발생" 오류와 함께 Amazon Athena 쿼리 실패를 참조하세요.
- Null 값이 정수 필드에 존재하는 경우. 한 가지 해결 방법은 null 값을 포함한 열을 string으로 생성한 다음 CAST를 사용하여 쿼리의 필드를 변환하면서 null에 대해 기본값 0을 부여하는 것입니다. 자세한 내용은 AWS 지식 센터의 Athena에서 CSV 데이터를 쿼리하는 경우 "HIVE_BAD_DATA: Error parsing field value '' for field X: For input string: """ 오류가 발생합니다를 참조하세요.

HIVE_CANNOT_OPEN_SPLIT: Error opening Hive split s3://DOC-EXAMPLE-BUCKET

이 오류는 많은 수의 객체가 있는 Amazon S3 버킷 접두사를 쿼리할 때 발생할 수 있습니다. 자세한 내용은 AWS 지식 센터의 Athena에서 "HIVE_CANNOT_OPEN_SPLIT: Error opening Hive split s3://DOC-EXAMPLE-BUCKET/: Slow down" 오류를 해결하려면 어떻게 해야 하나요?를 참조하세요.

HIVE_CURSOR_ERROR: com.amazonaws.services.s3.model.AmazonS3Exception: The specified key does not exist

이 오류는 일반적으로 쿼리 실행 중에 파일이 제거되었을 때 발생합니다. 쿼리를 다시 실행하거나, 워크플로를 조사하여 쿼리 실행 중에 다른 작업 또는 프로세스가 파일을 수정하고 있는지에 확인합니다.

HIVE_CURSOR_ERROR: Unexpected end of input stream

이 메시지는 파일이 손상되었거나 비어 있음을 나타냅니다. 파일의 무결성을 확인하고 쿼리를 다시 실행합니다.

HIVE_FILESYSTEM_ERROR: 파일에 대한 잘못된 파일 크기 ***1234567***

이 메시지는 쿼리 계획과 쿼리 실행 간에 파일이 변경되었을 때 발생할 수 있습니다. 일반적으로 Amazon S3 파일이 현재 위치에서 교체될 때 발생합니다. 예를 들어, PUT은 객체가 이미 존재하는 키에서 수행됩니다. Athena는 쿼리가 실행 중일 때 파일의 내용을 삭제하거나 바꾸는 것을 지원하지 않습니다. 이 오류를 방지하려면 쿼리가 실행되지 않을 때 파일을 덮어쓰거나 삭제하는 작업을 예약하거나 새 파일이나 파티션에만 데이터를 씁니다.

HIVE_UNKNOWN_ERROR: Unable to create input format

이 오류는 다음과 같은 문제로 인해 발생할 수 있습니다.

- AWS Glue Crawler가 데이터 형식을 분류할 수 없는 경우
- 특정 AWS Glue 테이블 정의 속성이 비어 있는 경우
- Athena가 Amazon S3에 있는 파일의 데이터 형식을 지원하지 않는 경우

자세한 내용은 AWS 지식 센터의 Athena에서 “입력 형식을 생성할 수 없음” 오류를 해결하려면 어떻게 해야 합니까?를 참조하거나 지식 센터 동영상을 시청하세요.

쿼리 결과를 저장하기 위해 제공된 S3 위치가 잘못되었습니다.

쿼리 결과에 대해 유효한 S3 위치를 지정했는지 확인합니다. 자세한 내용은 쿼리 결과 위치 지정 주제에서 쿼리 결과, 최근 쿼리, 출력 파일 작업 단원을 참조하세요.

Linux Foundation Delta Lake 테이블

Delta Lake 테이블 스키마가 동기화되지 않음

AWS Glue에 오래된 스키마가 있는 Delta Lake 테이블을 쿼리하면 다음과 같은 오류 메시지가 나타날 수 있습니다.

```
INVALID_GLUE_SCHEMA: Delta Lake table schema in Glue does not match the most recent
 schema of the
Delta Lake transaction log. Please ensure that you have the correct schema defined in
 Glue.
```

스키마를 Athena에 추가한 후 AWS Glue에서 수정하면 스키마가 오래될 수 있습니다. 스키마를 업데이트하려면 다음 단계 중 하나를 수행하세요.

- AWS Glue에서 AWS Glue 크롤러를 실행합니다.
- Athena에서 테이블을 삭제하고 다시 생성합니다.
- Athena에서 ALTER TABLE ADD COLUMNS 문을 사용하거나 AWS Glue에서 테이블 스키마를 편집하여 누락된 열을 수동으로 추가합니다.

페더레이션 쿼리

ListTableMetadata 호출 중 제한 시간 초과

데이터 소스에 테이블이 많거나 데이터 소스가 느리거나 네트워크 속도가 느린 경우 ListTableMetadata API 호출 제한 시간이 초과될 수 있습니다. 이 문제를 해결하려면 다음 단계를 시도합니다.

- 테이블 수 확인 - 테이블이 1,000개가 넘으면 테이블 수를 줄여봅니다. 가장 빠른 `ListTableMetadata` 응답을 얻으려면 카탈로그당 테이블 수를 1,000개 미만으로 줄이는 것이 좋습니다.
- Lambda 구성 확인 - Lambda 함수 동작을 모니터링하는 것은 중요합니다. 페더레이션된 카탈로그를 사용하는 경우 Lambda 함수의 실행 로그를 검토해야 합니다. 결과에 따라 메모리와 제한 시간 값을 적절히 조정합니다. 제한 시간과 관련된 잠재적 문제를 식별하려면 Lambda 구성을 다시 확인합니다. 자세한 내용은 AWS Lambda 개발자 안내서의 Lambda 함수 시간 제한 구성(콘솔)을 참조하세요.

- 페더레이션된 데이터 소스 로그 확인 - 페더레이션된 데이터 소스의 로그와 오류 메시지를 검사하여 문제나 오류가 있는지 확인합니다. 로그는 제한 시간 초과의 원인에 대한 소중한 인사이트를 제공할 수 있습니다.
- **StartQueryExecution**을 사용하여 메타데이터 가져오기 – 테이블이 1,000개가 넘는 경우 페더레이션 커넥터를 사용하여 메타데이터를 검색하는 데 예상보다 오래 걸릴 수 있습니다. StartQueryExecution의 비동기적 특성으로 인해 Athena는 가장 최적의 방식으로 쿼리를 실행하므로 `ListTableMetadata` 대신 `StartQueryExecution`을 사용하는 것이 좋습니다. 다음 AWS CLI 예제에서는 `ListTableMetadata` 대신 `StartQueryExecution`을 사용하여 데이터 카탈로그에 있는 테이블의 모든 메타데이터를 가져오는 방법을 보여줍니다.

  먼저 다음 예제와 같이 모든 테이블을 가져오는 쿼리를 실행합니다.

```
aws athena start-query-execution --region us-east-1 \
--query-string "SELECT table_name FROM information_schema.tables LIMIT 50" \
--work-group "your-work-group-name"
```

  그 다음으로, 다음 예제와 같이 개별 테이블의 메타데이터를 검색합니다.

```
aws athena start-query-execution --region us-east-1 \
--query-string "SELECT * FROM information_schema.columns \
WHERE table_name = 'your-table-name' AND \
table_catalog = 'your-catalog-name'" \
--work-group "your-work-group-name"
```

  결과를 얻는 데 걸리는 시간은 카탈로그의 테이블 수에 따라 다릅니다.

페더레이션된 쿼리 문제 해결에 대한 자세한 내용은 GitHub의 awslabs/aws-athena-query-federation 섹션에서 Common_Problems를 참조하거나 개별 Athena 데이터 소스 커넥터에 대한 설명서를 참조하세요.

JSON 관련 오류

NULL or incorrect data errors when trying to read JSON data

JSON 데이터를 읽으려고 할 때 NULL 또는 잘못된 데이터 오류는 여러 가지 원인으로 인해 발생할 수 있습니다. OpenX SerDe를 사용할 때 오류를 일으키는 행을 파악하려면 `ignore.malformed.json`을 `true`로 설정합니다. 잘못된 형식의 레코드는 NULL로 반환됩니다. 자

세한 내용은 AWS 지식 센터의 Amazon Athena에서 JSON 데이터를 읽으려고 할 때 오류가 발생하는 이유는 무엇입니까?를 참조하거나 지식 센터 동영상을 시청하세요.

HIVE_BAD_DATA: field 0의 필드 값을 구문 분석하는 중 오류 발생: java.lang.String을 org.openx.data.jsonserde.json.JSONObject로 캐스팅할 수 없음

Athena 쿼리에서 열을 구문 분석하지 못할 때 OpenX JSON SerDe에서 이 오류가 발생합니다. 이러한 상황은 `map` 또는 `struct`로 열을 정의했는 데 기초 데이터가 실제로 `string`, `int` 또는 기본 형식일 경우에 발생할 수 있습니다.

HIVE_CURSOR_ERROR: Row is not a valid JSON object - JSONException: Duplicate key

이 오류는 Athena를 사용하여 서로 다른 경우에 동일한 이름의 여러 태그를 사용하는 AWS Config 리소스를 쿼리할 때 발생합니다. 해결 방법은 `WITH SERDEPROPERTIES 'case.insensitive'='false'`를 사용하여 `CREATE TABLE`을 실행하고 이름을 매핑하는 것입니다. `case.insensitive` 및 매핑에 대한 자세한 내용은 JSON SerDe 라이브러리 단원을 참조하세요. 자세한 내용은 AWS 지식 센터의 Athena에서 AWS Config의 파일을 읽을 때 "HIVE_CURSOR_ERROR: Row is not a valid JSON Object - JSONException: Duplicate key"를 어떻게 해결해야 합니까?를 참조하세요.

가독성 좋게 꾸민 JSON의 HIVE_CURSOR_ERROR 메시지

Hive JSON SerDe 및 OpenX JSON SerDe 라이브러리에서는 각 JSON 문서가 레코드의 필드를 구분하는 줄 종료 문자가 없는 한 줄의 텍스트에 있을 것으로 예상합니다. JSON 텍스트가 가독성 좋게 꾸민 형식이면 테이블을 만든 후 쿼리하려고 할 때 HIVE_CURSOR_ERROR: 행이 유효한 JSON 객체가 아님(HIVE_CURSOR_ERROR: Row is not a valid JSON Object) 또는 HIVE_CURSOR_ERROR: JsonParseException: 예기치 않은 입력 종료: OBJECT의 닫기 마커 필요(HIVE_CURSOR_ERROR: JsonParseException: Unexpected end-of-input: expected close marker for OBJECT) 같은 오류 메시지가 나타날 수 있습니다. 자세한 내용은 GitHub의 OpenX SerDe 문서에서 JSON 데이터 파일을 참조하세요.

여러 JSON 레코드가 SELECT COUNT를 1로 반환함

OpenX JSON SerDe를 사용하는 경우 레코드가 줄 바꿈 문자로 구분되어 있는지 확인합니다. 자세한 내용은 AWS 지식 센터의 입력 JSON 파일에 다수의 레코드가 있는 경우에도 Amazon Athena의 SELECT COUNT 쿼리가 레코드를 1개만 반환합니다를 참조하세요.

Cannot query a table created by a AWS Glue crawler that uses a custom JSON classifier

Athena 엔진은 사용자 지정 JSON 분류자를 지원하지 않습니다. 이 문제를 해결하려면 사용자 지정 분류자를 사용하지 않고 새 테이블을 생성해야 합니다. JSON을 변환하려면 CTAS를 사용하거나 뷰를 생성합니다. 예를 들어 배열로 작업하는 경우 UNNEST 옵션을 사용하여 JSON을 평면화할 수 있습니다. 또 다른 옵션은 사용자 지정 분류자를 지원하는 AWS Glue ETL 작업을 사용해 Amazon S3에서 데이터를 parquet로 변환한 다음 Athena에서 쿼리하는 것입니다.

MSCK REPAIR TABLE

MSCK REPAIR TABLE 관련 문제에 대한 자세한 내용은 MSCK REPAIR TABLE 페이지의 고려 사항 및 제한 및 문제 해결 단원을 참조하세요.

출력 문제

Unable to verify/create output bucket

지정된 쿼리 결과 위치가 없거나 적절한 사용 권한이 없는 경우 이 오류가 발생할 수 있습니다. 자세한 내용은 AWS 지식 센터의 Amazon Athena에서 "Unable to verify/create output bucket" 오류를 해결하려면 어떻게 해야 합니까?를 참조하세요.

TIMESTAMP result is empty

Athena에는 Java TIMESTAMP 형식이 필요합니다. 자세한 내용은 AWS 지식 센터의 Amazon Athena에서 테이블을 쿼리할 때 TIMESTAMP 결과가 비어 있음을 참조하세요.

Athena 쿼리 출력을 CSV 이외의 형식으로 저장

기본적으로 Athena는 CSV 형식으로만 파일을 출력합니다. SELECT 쿼리의 결과를 다른 형식으로 출력하려면 UNLOAD 문을 사용하면 됩니다. 자세한 내용은 UNLOAD 단원을 참조하십시오. 또한 `format` 테이블 속성을 사용하여 출력 형식을 구성하는 CTAS 쿼리를 사용할 수도 있습니다. UNLOAD와 달리, CTAS 기법을 사용하려면 테이블을 만들어야 합니다. 자세한 내용은 AWS 지식 센터의 Athena 쿼리 출력을 압축 형식과 같은 CSV 이외의 형식으로 저장하려면 어떻게 해야 합니까?를 참조하세요.

The S3 location provided to save your query results is invalid

출력 버켓 위치가 쿼리를 실행하는 리전과 동일한 리전에 있지 않은 경우 이 오류 메시지가 나타날 수 있습니다. 이를 방지하려면 쿼리를 실행하는 리전에 쿼리 결과 위치를 지정합니다. 단계는 쿼리 결과 위치 지정를 참조하세요.

Parquet 문제

org.apache.parquet.io.GroupColumnIO cannot be cast to org.apache.parquet.io.PrimitiveColumnIO

이 오류는 parquet 스키마 불일치로 인해 발생합니다. 기본 형식이 아닌 열(예: `array`)이 AWS Glue에서 기본 형식(예: `string`)으로 선언된 것입니다. 이 문제를 해결하려면 파일의 데이터 스키마를 조사하여 AWS Glue에 선언된 스키마와 비교합니다.

Parquet 통계 문제

Parquet 데이터를 읽을 때 다음과 같은 오류 메시지가 나타날 수 있습니다.

```
HIVE_CANNOT_OPEN_SPLIT: Index x out of bounds for length y
HIVE_CURSOR_ERROR: Failed to read x bytes
HIVE_CURSOR_ERROR: FailureException at Malformed input: offset=x
HIVE_CURSOR_ERROR: FailureException at java.io.IOException:
can not read class org.apache.parquet.format.PageHeader: Socket is closed by peer.
```

이 문제를 해결하려면 다음 예제와 같이 CREATE TABLE 또는 ALTER TABLE SET TBLPROPERTIES 문을 사용하여 Parquet SerDe `parquet.ignore.statistics` 속성을 `true`로 설정합니다.

CREATE TABLE 예제

```
...
ROW FORMAT SERDE
'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
WITH SERDEPROPERTIES ('parquet.ignore.statistics'='true')
STORED AS PARQUET
...
```

ALTER TABLE 예제

```
ALTER TABLE ... SET TBLPROPERTIES ('parquet.ignore.statistics'='true')
```

Parquet Hive SerDe에 대한 자세한 내용은 Parquet SerDe 섹션을 참조하세요.

파티셔닝 문제

MSCK REPAIR TABLE does not remove stale partitions

Amazon S3에서 파티션을 수동으로 삭제한 다음 MSCK REPAIR TABLE을 실행하면 파일 시스템에서 파티션이 누락됨(Partitions missing from filesystem) 오류 메시지가 나타날 수 있습니다. MSCK REPAIR TABLE은 기한 경과된 파티션을 테이블 메타데이터에서 제거하지 않기 때문입니다. 기한 경과 파티션을 수동으로 제거하려면 ALTER TABLE DROP PARTITION을 사용합니다. 자세한 내용은 MSCK REPAIR TABLE 주제의 "문제 해결" 단원을 참조하세요.

MSCK REPAIR TABLE failure

많은 양의 파티션(예: 100,000개 이상)이 특정 테이블과 연결된 경우 `MSCK REPAIR TABLE`은 메모리 제한 때문에 실패할 수 있습니다. 이 제한을 해결하려면 ALTER TABLE ADD PARTITION을 대신 사용합니다.

MSCK REPAIR TABLE은 파티션을 삭제하지만 AWS Glue에 파티션을 추가하지 않습니다.

이 문제는 Amazon S3 경로가 소문자가 아닌 카멜 표기법이거나 IAM 정책이 `glue:BatchCreatePartition` 작업을 허용하지 않을 때 발생할 수 있습니다. 자세한 내용은 AWS 지식 센터의 MSCK REPAIR TABLE이 Athena에서 파티션을 감지하지만 AWS Glue Data Catalog에 파티션을 추가하지 않습니다를 참조하세요.

Partition projection ranges with the date format of dd-MM-yyyy-HH-mm-ss or yyyy-MM-dd do not work

올바르게 작동하려면 날짜 형식을 `yyyy-MM-dd HH:00:00`으로 설정해야 합니다. 자세한 내용은 Stack Overflow 게시물 Athena 파티션 프로젝션이 예상대로 작동하지 않음을 참조하세요.

PARTITION BY는 BIGINT 형식을 지원하지 않습니다.

데이터 형식을 `string`으로 변환하고 다시 시도하세요.

No meaningful partitions available

이 오류 메시지는 일반적으로 파티션 설정이 손상되었음을 의미합니다. 이 문제를 해결하려면 테이블을 삭제하고 새 파티션을 가진 테이블을 만듭니다.

Partition projection does not work in conjunction with range partitions

시간 범위 단위 projection.*<columnName>*.interval.unit이 파티션의 구분 기호와 일치하는지 확인합니다. 예를 들어 파티션이 일별로 구분되면 시간 범위 단위가 작동하지 않습니다.

범위를 하이픈으로 지정한 경우 파티션 프로젝션 오류 발생

`range` 테이블 속성을 쉼표 대신 하이픈으로 지정하면 다음과 같은 오류가 발생합니다. INVALID_TABLE_PROPERTY: For input string: "*number*-*number*". 범위 값을 하이픈이 아닌 쉼표로 구분해야 합니다. 자세한 내용은 정수 형식 단원을 참조하십시오.

HIVE_UNKNOWN_ERROR: Unable to create input format

하나 이상의 glue 파티션은 서로 다른 형식으로 선언됩니다. 각 glue 파티션은 독립적으로 고유한 입력 형식이 있기 때문입니다. 파티션이 AWS Glue에서 어떻게 정의되어 있는지 확인하세요.

HIVE_PARTITION_SCHEMA_MISMATCH

파티션의 스키마가 테이블의 스키마와 다르면 HIVE_PARTITION_SCHEMA_MISMATCH 오류 메시지와 함께 쿼리가 실패할 수 있습니다. 자세한 내용은 "HIVE_PARTITION_SCHEMA_MISMATCH"를 피하기 위한 파티션 스키마 동기화 단원을 참조하세요.

SemanticException table is not partitioned but partition spec exists

`CREATE TABLE` 문에 파티션이 정의되지 않으면 이 오류가 발생할 수 있습니다. 자세한 내용은 AWS 지식 센터의 Athena에서 "FAILED: SemanticException table is not partitioned but partition spec exists"라는 오류를 어떻게 해결해야 합니까?를 참조하세요.

0바이트 **_$folder$** 파일

`ALTER TABLE ADD PARTITION` 문을 실행할 때 이미 존재하는 파티션과 잘못된 Amazon S3 위치를 지정하는 경우 Amazon S3에 *partition_value*`_$folder$` 형식의 0바이트 자리 표시자 파일이 생성됩니다. 이러한 파일은 수동으로 제거해야 합니다.

이러한 일이 발생하지 않도록 하려면 아래와 같이 `ALTER TABLE ADD PARTITION` 문에서 `ADD IF NOT EXISTS` 구문을 사용합니다.

```
ALTER TABLE table_name ADD IF NOT EXISTS PARTITIION […]
```

Zero records returned from partitioned data

이 문제는 여러 가지 이유로 발생할 수 있습니다. 가능한 원인과 해결 방법은 AWS 지식 센터의 파티션을 정의하여 Amazon Athena에 테이블을 만들었는데 테이블을 쿼리할 때 0개의 레코드가 반환됩니다를 참조하세요.

HIVE_TOO_MANY_OPEN_PARTITIONS 섹션도 참조하십시오.

권한

Amazon S3 쿼리할 때 Access Denied 오류가 표시됨

이 오류는 버킷의 데이터를 읽을 권한이 없거나 결과 버킷에 대한 쓰기 권한이 없거나 Amazon S3 경로에 `us-east-1.amazonaws.com` 같은 리전 엔드포인트가 포함된 경우에 발생합니다. 자세한 내용은 AWS 지식 센터의 Athena 쿼리를 실행하면 "Access Denied" 오류가 나타납니다를 참조하세요.

Amazon S3의 암호화된 데이터에 대해 DDL 쿼리를 실행할 때 Access Denied with Status Code: 403이 표시됨

다음 조건에 해당되면 Access Denied (Service: Amazon S3; Status Code: 403; Error Code: AccessDenied; Request ID: *<request_id>*)라는 오류 메시지가 나타날 수 있습니다.

1. `ALTER TABLE ADD PARTITION` 또는 `MSCK REPAIR TABLE` 같은 DDL 쿼리를 실행합니다.
2. 기본 암호화가 `SSE-S3`를 사용하도록 구성된 버킷이 있습니다.
3. 이 버킷에는 `PutObject` 요청에서 PUT 헤더 `"s3:x-amz-server-side-encryption": "true"` 및 `"s3:x-amz-server-side-encryption": "AES256"`을 지정하도록 하는 다음과 같은 버킷 정책도 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::<resource-name>/*",
            "Condition": {
                "Null": {
                    "s3:x-amz-server-side-encryption": "true"
                }
```

```
            }
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::<resource-name>/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "AES256"
                }
            }
        }
    ]
}
```

이와 같은 경우 권장되는 해결 방법은 버킷의 기본 암호화가 이미 존재한다는 점을 감안하여 위와 같은 버킷 정책을 제거하는 것입니다.

다른 계정의 Amazon S3 버킷을 쿼리할 때 Access Denied with Status Code: 403이 표시됨

이 오류는 다른 AWS 서비스가 작성한 로그를 쿼리할 때 두 번째 계정이 버킷 소유자이지만 버킷의 객체를 소유하지 않은 경우에 발생할 수 있습니다. 자세한 내용은 AWS 지식 센터의 다른 계정의 버킷을 쿼리할 때 Amazon Athena에서 Amazon S3 예외 "Access Denied with Status Code: 403"이 표시됨을 참조하거나 지식 센터 동영상을 시청하세요.

IAM 역할 자격 증명을 사용하여 Athena JDBC 드라이버에 연결

역할의 임시 자격 증명을 검색하여 Athena에 대한 JDBC 연결을 인증할 수 있습니다. 임시 자격 증명의 최대 수명은 12시간입니다. 자세한 내용은 AWS 지식 센터의 JDBC 드라이버를 사용해 Athena에 연결할 때 IAM 역할 자격 증명을 사용하거나 다른 IAM 역할로 전환하는 방법은 어떻게 되나요?를 참조하세요.

쿼리 구문 문제

FAILED: NullPointerException name is null

`TableType` 속성을 지정하지 않고 AWS Glue CreateTable API 작업 또는 AWS CloudFormation `AWS::Glue::Table` 템플릿을 사용하여 Athena에서 사용할 테이블을 만든 다음 `SHOW CREATE`

TABLE 또는 MSCK REPAIR TABLE 같은 DDL 쿼리를 실행하면, 실패: NullPointerException Name이 null임(FAILED: NullPointerException Name is null)이라는 오류 메시지가 표시될 수 있습니다.

이 오류를 해결하려면 AWS Glue CreateTable API 호출 또는 AWS CloudFormation 템플릿의 일부로 TableInput TableType 속성의 값을 지정하세요. TableType의 가능한 값은 EXTERNAL_TABLE 또는 VIRTUAL_VIEW입니다.

이 요구 사항은 AWS Glue CreateTable API 작업 또는 AWS::Glue::Table 템플릿을 사용하여 테이블을 만들 때만 적용됩니다. DDL 문이나 AWS Glue 크롤러를 사용하여 Athena용 테이블을 생성할 경우 TableType 속성이 자동으로 정의됩니다.

함수가 등록되지 않음

이 오류는 Athena에서 지원하지 않는 함수를 사용하려고 할 때 발생합니다. Athena에서 지원하는 함수 목록은 Amazon Athena의 함수를 참조하세요. 또는 쿼리 편집기에서 SHOW FUNCTIONS 문을 실행하세요. 자체적인 사용자 정의 함수(UDF)를 작성할 수도 있습니다. 자세한 내용은 AWS 지식 센터의 Athena에서 '함수가 등록되지 않음'이라는 구문 오류를 어떻게 해결합니까?를 참조하세요.

GENERIC_INTERNAL_ERROR 예외

GENERIC_INTERNAL_ERROR 예외는 다음과 같은 다양한 원인으로 인해 발생할 있을 수 있습니다.

- GENERIC_INTERNAL_ERROR: Null – 다음 조건 중 하나에서 이 예외가 나타날 수 있습니다.
  - 테이블 정의에 있는 열의 데이터 형식과 데이터 세트의 실제 데이터 형식 간에 스키마가 일치하지 않습니다.
  - 구문이 부정확한 CREATE TABLE AS SELECT(CTAS) 쿼리를 실행 중입니다.
- GENERIC_INTERNAL_ERROR: Parent builder is null – array 데이터 형식의 열이 있는 테이블을 쿼리하고 OpenCSVSerDe 라이브러리를 사용하는 경우 이 예외가 발생할 수 있습니다. OpenCSVerde 형식은 array 데이터 형식을 지원하지 않습니다.
- GENERIC_INTERNAL_ERROR: 값이 MAX_INT를 초과함(GENERIC_INTERNAL_ERROR: Value exceeds MAX_INT) – 소스 데이터 열이 INT 데이터 형식으로 정의되어 있고 2,147,483,647보다 큰 숫자 값이 열에 있는 경우 이 예외가 표시될 수 있습니다.
- GENERIC_INTERNAL_ERROR: 값이 MAX_BYTE를 초과함(GENERIC_INTERNAL_ERROR: Value exceeds MAX_BYTE) - 소스 데이터 열에 데이터 형식 BYTE에 허용되는 크기를 초과하는 숫자 값이 있는 경우 이 예외가 표시될 수 있습니다. BYTE 데이터 형식은 TINYINT와 동일합니다. TINYINT는 2의 보수 형식의 부호 있는 8비트 정수로, 최솟값은 -128이고 최댓값은 127입니다.
- GENERIC_INTERNAL_ERROR: 파티션 값 수가 필터 수와 일치하지 않음(GENERIC_INTERNAL_ERROR: Number of partition values does not match number of filters) -

Amazon Simple Storage Service(Amazon S3) 데이터에 일관성 없는 파티션이 있는 경우 이 예외가 발생할 수 있습니다. 다음 조건에서는 파티션이 일관되지 않을 수 있습니다.

- Amazon S3 S3의 파티션이 변경되었습니다(예: 새 파티션이 추가됨).
- 테이블의 파티션 열 수가 파티션 메타데이터의 열 수와 일치하지 않습니다.

이러한 각 오류에 대한 자세한 내용은 AWS 지식 센터에서 Amazon Athena 테이블을 쿼리할 때 'GENERIC_INTERNAL_ERROR' 오류를 해결하려면 어떻게 해야 합니까?를 참조하세요.

일치하는 그룹 수가 열 수와 일치하지 않습니다.

이 오류는 CREATE TABLE 문에 Regex SerDe를 사용할 때 정규식 일치 그룹의 수가 테이블에 지정한 열 수와 일치하지 않는 경우에 발생합니다. 자세한 내용은 AWS 지식 센터의 Amazon Athena에서 "Number of matching groups doesn't match the number of columns"라는 RegexSerDe 오류를 해결하려면 어떻게 합니까?를 참조하세요.

queryString failed to satisfy constraint: Member must have length less than or equal to 262144

Athena의 최대 쿼리 문자열 길이(262,144바이트)는 조정 가능한 할당량이 아닙니다. AWS Support에서 할당량을 늘릴 수는 없지만 사용자가 긴 쿼리를 더 작은 쿼리로 분할하여 문제를 해결할 수 있습니다. 자세한 내용은 AWS 지식 센터에서 Athena의 최대 쿼리 문자열 길이를 늘리려면 어떻게 해야 합니까?를 참조하세요.

SYNTAX_ERROR: Column cannot be resolved

이 오류는 바이트 순서 표시(BOM)가 포함된 UTF-8 인코딩 CSV 파일로부터 AWS Glue 크롤러가 생성한 테이블을 쿼리할 때 발생할 수 있습니다. AWS Glue는 BOM을 인식하지 못하여 이를 물음표로 변경하기 때문에 Amazon Athena에서 이를 인식할 수 없습니다. 해결책은 Athena 또는 AWS Glue에서 물음표를 제거하는 것입니다.

함수 호출에 사용하는 인수가 너무 많음

Athena 엔진 버전 3에서 함수의 인수는 127개를 초과할 수 없습니다. 이 제한은 설계에 따른 수치입니다. 함수에서 파라미터가 127개를 초과하는 경우 다음과 같은 오류 메시지가 나타납니다.

TOO_MANY_ARGUMENTS: line *nnn*:*nn*: Too many arguments for function call *function_name*().

이 문제를 해결하려면 함수 호출당 파라미터를 더 적게 사용합니다.

쿼리 시간 초과 문제

Athena 쿼리에서 시간 초과 오류가 발생하는 경우 CloudTrail 로그를 확인하세요. AWS Glue 또는 Lake Formation API의 제한으로 인해 쿼리 시간이 초과될 수 있습니다. 이러한 오류가 발생하는 경우 해당 오류 메시지는 제한 문제가 아니라 쿼리 시간 초과 문제를 나타낼 수 있습니다. 문제를 해결하려면 AWS Support에 문의하기 전에 CloudTrail 로그를 확인할 수 있습니다. 자세한 내용은 AWS CloudTrail 로그 쿼리 및 AWS CloudTrail을 사용하여 Amazon Athena API 호출 로깅 단원을 참조하세요.

`ListTableMetadata` API를 직접적으로 호출할 때 페더레이션 쿼리의 쿼리 시간 초과 문제에 대한 자세한 내용은 ListTableMetadata 호출 중 제한 시간 초과 섹션을 참조하세요.

제한(Throttling) 문제

쿼리가 Amazon S3, AWS KMS, AWS Glue 또는 AWS Lambda와 같은 종속 서비스의 한도를 초과하면 다음 메시지를 예상할 수 있습니다. 이러한 문제를 해결하려면 동일한 계정에서 발생하는 동시 호출 수를 줄여야 합니다.

Service	오류 메시지
AWS Glue	AWSGlueException: 속도를 초과했습니다.
AWS KMS	You have exceeded the rate at which you may call KMS. Reduce the frequency of your calls.
AWS Lambda	Rate exceeded TooManyRequestsException
Amazon S3	AmazonS3Exception: 요청 속도를 줄여야 합니다.

Athena를 사용할 때 Amazon S3 제한을 방지하는 방법에 대한 자세한 내용은 Amazon S3 제한 방지 섹션을 참조하세요.

보기

Views created in Apache Hive shell do not work in Athena

근본적으로 다른 구현이기 때문에 Apache Hive 셸에서 생성된 뷰는 Athena와 호환되지 않습니다. 이 문제를 해결하려면 Athena에서 뷰를 다시 생성합니다.

View is stale; it must be re-created

뷰의 기초가 되는 테이블이 변경되거나 삭제된 경우 이 오류가 나타날 수 있습니다. 해결 방법은 뷰를 다시 생성하는 것입니다. 자세한 내용은 AWS 지식 센터의 Athena에서 "View is stale; it must be re-created"라는 오류를 어떻게 해결해야 합니까?를 참조하세요.

작업 그룹

작업 그룹 문제 해결에 대한 자세한 내용은 작업 그룹 문제 해결 단원을 참조하세요.

추가적인 리소스

다음 페이지에서는 Amazon Athena의 문제 해결을 위한 추가 정보를 제공합니다.

- Athena 오류 카탈로그
- Service Quotas
- Amazon Athena의 SQL 쿼리에 대한 고려 사항 및 제한 사항
- 지원되지 않는 DDL
- 테이블, 데이터베이스 및 열의 이름
- Amazon Athena의 데이터 형식
- 지원되는 SerDes 및 데이터 형식
- Athena 압축 지원
- 예약어
- 작업 그룹 문제 해결

다음 AWS 리소스도 도움이 될 수 있습니다.

- AWS 지식 센터의 Athena 주제
- AWS re:Post에 대한 Amazon Athena 질문
- AWS Big Data Blog(빅 데이터 블로그)의 Athena 게시물

문제 해결에는 종종 전문가 또는 도우미 커뮤니티의 반복적인 질의와 발견이 필요한 경우가 많습니다. 이 페이지에 있는 제안 사항을 시도한 후에도 문제가 계속 발생하면 AWS Support에 문의하거나(AWS Management Console에서 지원(Support), 지원 센터(Support Center) 클릭) Amazon Athena 태그를 사용하여 AWS re:Post에 대해 질문하세요.

Athena 오류 카탈로그

Athena는 실패한 쿼리를 이해하고 쿼리 실패가 발생한 후 조치를 취할 수 있도록 표준화된 오류 정보를 제공합니다. 이 `AthenaError` 기능은 `ErrorCategory` 필드와 `ErrorType` 필드를 포함합니다. `ErrorCategory`는 실패한 쿼리의 원인이 시스템 오류, 사용자 오류 또는 기타 오류로 인한 것인지를 지정합니다. `ErrorType`에서는 오류 원인에 대한 더 세부적인 정보를 제공합니다. 두 필드를 결합하면 주변 상황과 발생한 특정 오류의 원인을 더 잘 이해할 수 있습니다.

오류 카탈로그

다음 표에는 Athena 오류 범주 값과 그 의미가 나열되어 있습니다.

오류 카탈로그	소스
1	SYSTEM
2	USER
3	기타

오류 유형 참조

다음 표에는 Athena 오류 유형 값과 그 의미가 나열되어 있습니다.

오류 유형	설명
0	이 스케일 팩터에서 쿼리가 리소스를 소진함
1	이 스케일 팩터에서 쿼리가 리소스를 소진함
2	이 스케일 팩터에서 쿼리가 리소스를 소진함
3	이 스케일 팩터에서 쿼리가 리소스를 소진함

오류 유형	설명
4	이 스케일 팩터에서 쿼리가 리소스를 소진함
5	이 스케일 팩터에서 쿼리가 리소스를 소진함
6	이 스케일 팩터에서 쿼리가 리소스를 소진함
7	이 스케일 팩터에서 쿼리가 리소스를 소진함
8	이 스케일 팩터에서 쿼리가 리소스를 소진함
100	내부 서비스 오류
200	쿼리 엔진에서 내부 오류 발생
201	쿼리 엔진에서 내부 오류 발생
202	쿼리 엔진에서 내부 오류 발생
203	드라이버 오류
204	메타스토어에 오류 발생
205	쿼리 엔진에서 내부 오류 발생
206	쿼리 시간 초과
207	쿼리 엔진에서 내부 오류 발생
208	쿼리 엔진에서 내부 오류 발생
209	쿼리 취소 실패
210	쿼리 시간 초과
211	쿼리 엔진에서 내부 오류 발생
212	쿼리 엔진에서 내부 오류 발생
213	쿼리 엔진에서 내부 오류 발생

오류 유형	설명
214	쿼리 엔진에서 내부 오류 발생
215	쿼리 엔진에서 내부 오류 발생
216	쿼리 엔진에서 내부 오류 발생
217	쿼리 엔진에서 내부 오류 발생
218	쿼리 엔진에서 내부 오류 발생
219	쿼리 엔진에서 내부 오류 발생
220	쿼리 엔진에서 내부 오류 발생
221	쿼리 엔진에서 내부 오류 발생
222	쿼리 엔진에서 내부 오류 발생
223	쿼리 엔진에서 내부 오류 발생
224	쿼리 엔진에서 내부 오류 발생
225	쿼리 엔진에서 내부 오류 발생
226	쿼리 엔진에서 내부 오류 발생
227	쿼리 엔진에서 내부 오류 발생
228	쿼리 엔진에서 내부 오류 발생
229	쿼리 엔진에서 내부 오류 발생
230	쿼리 엔진에서 내부 오류 발생
231	쿼리 엔진에서 내부 오류 발생
232	쿼리 엔진에서 내부 오류 발생
233	Iceberg 오류

오류 유형	설명
234	Lake Formation 오류
235	쿼리 엔진에서 내부 오류 발생
236	쿼리 엔진에서 내부 오류 발생
237	직렬화 오류
238	Amazon S3 메타데이터 업로드 실패
239	일반 지속성 오류
240	쿼리 제출 실패
300	내부 서비스 오류
301	내부 서비스 오류
302	내부 서비스 오류
303	내부 서비스 오류
400	내부 서비스 오류
401	Amazon S3 쿼리 결과 쓰기 실패
402	Amazon S3 쿼리 결과 쓰기 실패
1000	사용자 오류
1001	데이터 오류
1002	데이터 오류
1003	DDL 작업 실패
1004	스키마 오류
1005	직렬화 오류

오류 유형	설명
1006	구문 오류
1007	데이터 오류
1008	쿼리가 거부됨
1009	쿼리 실패
1010	내부 서비스 오류
1011	사용자가 쿼리를 취소함
1012	쿼리 엔진에서 내부 오류 발생
1013	쿼리 엔진에서 내부 오류 발생
1014	사용자가 쿼리를 취소함
1100	잘못된 인수가 제공됨
1101	잘못된 속성이 제공됨
1102	쿼리 엔진에서 내부 오류 발생
1103	잘못된 속성이 제공됨
1104	쿼리 엔진에서 내부 오류 발생
1105	쿼리 엔진에서 내부 오류 발생
1106	잘못된 함수 인수가 제공됨
1107	잘못된 뷰
1108	함수 등록 실패
1109	Amazon S3 경로를 찾을 수 없음
1110	제공된 테이블 또는 뷰가 존재하지 않음

오류 유형	설명
1200	쿼리가 지원되지 않음
1201	제공된 디코더는 지원되지 않음
1202	지원되지 않는 쿼리 유형
1300	일반적인 찾을 수 없음 오류
1301	일반 엔터티를 찾을 수 없음
1302	파일을 찾을 수 없음
1303	제공된 함수 또는 함수 구현을 찾을 수 없음
1304	쿼리 엔진에서 내부 오류 발생
1305	쿼리 엔진에서 내부 오류 발생
1306	Amazon S3 버킷을 찾을 수 없음
1307	엔진을 찾을 수 없음
1308	쿼리 엔진에서 내부 오류 발생
1400	스로틀링 오류
1401	AWS Glue 스로틀링으로 인한 쿼리 실패
1402	AWS Glue의 테이블 버전이 너무 많아서 쿼리에 실패함
1403	Amazon S3 스로틀링으로 인한 쿼리 실패
1404	Amazon Athena 스로틀링으로 인해 쿼리에 실패함
1405	Amazon Athena 스로틀링으로 인해 쿼리에 실패함
1406	Amazon Athena 스로틀링으로 인해 쿼리에 실패함
1500	권한 오류

오류 유형	설명
1501	Amazon S3 권한 오류
1602	예약 용량 한도를 초과했습니다. 용량이 부족하여 이 쿼리를 실행할 수 없습니다.
1700	Lake Formation 내부 예외로 인한 쿼리 실패
1701	AWS Glue 내부 예외로 인한 쿼리 실패
9999	내부 서비스 오류

코드 샘플

이 주제의 예제에서는 Athena 애플리케이션 작성의 출발점으로 SDK for Java 2.x를 사용합니다.

> **Note**
>
> 다른 언어별 AWS SDK를 사용하여 Athena를 프로그래밍하는 방법에 대한 자세한 내용은 다음 리소스를 참조하십시오.
>
> - AWS Command Line Interface (`athena`)
> - AWS SDK for .NET (`Amazon.Athena.Model`)
> - AWS SDK for C++ (`Aws::Athena::AthenaClient`)
> - AWS SDK for Go (`athena`)
> - AWS SDK for JavaScript v3(`AthenaClient`)
> - AWS SDK for PHP 3.x(`Aws\Athena`)
> - AWS SDK for Python (Boto3) (`Athena.Client`)
> - AWS SDK for Ruby v3(`Aws::Athena::Client`)

이 섹션의 Java 코드 예제 실행에 대한 자세한 내용은 GitHub의 AWS 코드 예제 리포지토리에 있는 Amazon Athena Java Readme를 참조하십시오. Athena용 Java 프로그래밍 참조는 AWS SDK for Java 2.x의 AthenaClient를 참조하십시오.

- Java 코드 예
  - 상수
  - Athena에 액세스할 클라이언트 생성
  - 쿼리 실행 작업
    - 쿼리 실행 시작
    - 쿼리 실행 중지
    - 쿼리 실행 나열
  - 명명된 쿼리 작업
    - 명명된 쿼리 생성
    - 명명된 쿼리 삭제
    - 쿼리 실행 나열

> **Note**
>
> 이 샘플은 문자열에 상수(예: ATHENA_SAMPLE_QUERY)를 사용합니다. 이는 ExampleConstants.java 클래스 선언에 정의되어 있습니다. 이 상수를 사용자의 고유한 문자열 또는 정의된 상수로 바꿉니다.

상수

ExampleConstants.java 클래스는 Athena의 시작하기 자습서에서 생성한 테이블을 쿼리하는 방법을 보여줍니다.

```
package aws.example.athena;

public class ExampleConstants {

    public static final int CLIENT_EXECUTION_TIMEOUT = 100000;
    public static final String ATHENA_OUTPUT_BUCKET = "s3://bucketscott2"; // change
 the Amazon S3 bucket name to match
                                                                            // your
 environment
    // Demonstrates how to query a table with a comma-separated value (CSV) table.
    // For information, see
    // https://docs.aws.amazon.com/athena/latest/ug/work-with-data.html
```

```
    public static final String ATHENA_SAMPLE_QUERY = "SELECT * FROM scott2;"; // change
 the Query statement to match
                                                                              // your
 environment
    public static final long SLEEP_AMOUNT_IN_MS = 1000;
    public static final String ATHENA_DEFAULT_DATABASE = "mydatabase"; // change the
 database to match your database

}
```

Athena에 액세스할 클라이언트 생성

AthenaClientFactory.java 클래스는 Amazon Athena 클라이언트를 생성 및 구성하는 방법을 보여 줍니다.

```
package aws.example.athena;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.AthenaClientBuilder;

public class AthenaClientFactory {
    private final AthenaClientBuilder builder = AthenaClient.builder()
            .region(Region.US_WEST_2)
            .credentialsProvider(ProfileCredentialsProvider.create());

    public AthenaClient createClient() {
        return builder.build();
    }
}
```

쿼리 실행 시작

StartQueryExample은 Athena에 쿼리를 제출하고 결과를 사용할 수 있을 때까지 기다린 후 결과를 처리하는 방법을 보여 줍니다.

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.model.QueryExecutionContext;
```

```
import software.amazon.awssdk.services.athena.model.ResultConfiguration;
import software.amazon.awssdk.services.athena.model.StartQueryExecutionRequest;
import software.amazon.awssdk.services.athena.model.StartQueryExecutionResponse;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.GetQueryExecutionRequest;
import software.amazon.awssdk.services.athena.model.GetQueryExecutionResponse;
import software.amazon.awssdk.services.athena.model.QueryExecutionState;
import software.amazon.awssdk.services.athena.model.GetQueryResultsRequest;
import software.amazon.awssdk.services.athena.model.GetQueryResultsResponse;
import software.amazon.awssdk.services.athena.model.ColumnInfo;
import software.amazon.awssdk.services.athena.model.Row;
import software.amazon.awssdk.services.athena.model.Datum;
import software.amazon.awssdk.services.athena.paginators.GetQueryResultsIterable;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class StartQueryExample {

    public static void main(String[] args) throws InterruptedException {
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        String queryExecutionId = submitAthenaQuery(athenaClient);
        waitForQueryToComplete(athenaClient, queryExecutionId);
        processResultRows(athenaClient, queryExecutionId);
        athenaClient.close();
    }

    // Submits a sample query to Amazon Athena and returns the execution ID of the
    // query.
    public static String submitAthenaQuery(AthenaClient athenaClient) {
        try {
            // The QueryExecutionContext allows us to set the database.
            QueryExecutionContext queryExecutionContext =
 QueryExecutionContext.builder()
                    .database(ExampleConstants.ATHENA_DEFAULT_DATABASE)
```

```
                .build();

        // The result configuration specifies where the results of the query should
go.
        ResultConfiguration resultConfiguration = ResultConfiguration.builder()
                .outputLocation(ExampleConstants.ATHENA_OUTPUT_BUCKET)
                .build();

        StartQueryExecutionRequest startQueryExecutionRequest =
StartQueryExecutionRequest.builder()
                .queryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
                .queryExecutionContext(queryExecutionContext)
                .resultConfiguration(resultConfiguration)
                .build();

        StartQueryExecutionResponse startQueryExecutionResponse = athenaClient
                .startQueryExecution(startQueryExecutionRequest);
        return startQueryExecutionResponse.queryExecutionId();

    } catch (AthenaException e) {
        e.printStackTrace();
        System.exit(1);
    }
    return "";
  }

  // Wait for an Amazon Athena query to complete, fail or to be cancelled.
  public static void waitForQueryToComplete(AthenaClient athenaClient, String
queryExecutionId)
        throws InterruptedException {
    GetQueryExecutionRequest getQueryExecutionRequest =
GetQueryExecutionRequest.builder()
            .queryExecutionId(queryExecutionId)
            .build();

    GetQueryExecutionResponse getQueryExecutionResponse;
    boolean isQueryStillRunning = true;
    while (isQueryStillRunning) {
        getQueryExecutionResponse =
athenaClient.getQueryExecution(getQueryExecutionRequest);
        String queryState =
getQueryExecutionResponse.queryExecution().status().state().toString();
        if (queryState.equals(QueryExecutionState.FAILED.toString())) {
            throw new RuntimeException(
```

```
                        "The Amazon Athena query failed to run with error message: " +
getQueryExecutionResponse
                                .queryExecution().status().stateChangeReason());
            } else if (queryState.equals(QueryExecutionState.CANCELLED.toString())) {
                throw new RuntimeException("The Amazon Athena query was cancelled.");
            } else if (queryState.equals(QueryExecutionState.SUCCEEDED.toString())) {
                isQueryStillRunning = false;
            } else {
                // Sleep an amount of time before retrying again.
                Thread.sleep(ExampleConstants.SLEEP_AMOUNT_IN_MS);
            }
            System.out.println("The current status is: " + queryState);
        }
    }

    // This code retrieves the results of a query
    public static void processResultRows(AthenaClient athenaClient, String
queryExecutionId) {
        try {
            // Max Results can be set but if its not set,
            // it will choose the maximum page size.
            GetQueryResultsRequest getQueryResultsRequest =
GetQueryResultsRequest.builder()
                    .queryExecutionId(queryExecutionId)
                    .build();

            GetQueryResultsIterable getQueryResultsResults = athenaClient
                    .getQueryResultsPaginator(getQueryResultsRequest);
            for (GetQueryResultsResponse result : getQueryResultsResults) {
                List<ColumnInfo> columnInfoList =
result.resultSet().resultSetMetadata().columnInfo();
                List<Row> results = result.resultSet().rows();
                processRow(results, columnInfoList);
            }

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    private static void processRow(List<Row> row, List<ColumnInfo> columnInfoList) {
        for (Row myRow : row) {
            List<Datum> allData = myRow.data();
```

```
            for (Datum data : allData) {
                System.out.println("The value of the column is " +
 data.varCharValue());
            }
        }
    }
}
```

쿼리 실행 중지

StopQueryExecutionExample은 쿼리 예제를 실행하고 쿼리를 즉시 중지하고 쿼리가 취소되었는지 쿼리 상태를 확인합니다.

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.model.StopQueryExecutionRequest;
import software.amazon.awssdk.services.athena.model.GetQueryExecutionRequest;
import software.amazon.awssdk.services.athena.model.GetQueryExecutionResponse;
import software.amazon.awssdk.services.athena.model.QueryExecutionState;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.QueryExecutionContext;
import software.amazon.awssdk.services.athena.model.ResultConfiguration;
import software.amazon.awssdk.services.athena.model.StartQueryExecutionRequest;
import software.amazon.awssdk.services.athena.model.StartQueryExecutionResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class StopQueryExecutionExample {
    public static void main(String[] args) {
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        String sampleQueryExecutionId = submitAthenaQuery(athenaClient);
        stopAthenaQuery(athenaClient, sampleQueryExecutionId);
```

```
        athenaClient.close();
    }

    public static void stopAthenaQuery(AthenaClient athenaClient, String
sampleQueryExecutionId) {
        try {
            StopQueryExecutionRequest stopQueryExecutionRequest =
StopQueryExecutionRequest.builder()
                    .queryExecutionId(sampleQueryExecutionId)
                    .build();

            athenaClient.stopQueryExecution(stopQueryExecutionRequest);
            GetQueryExecutionRequest getQueryExecutionRequest =
GetQueryExecutionRequest.builder()
                    .queryExecutionId(sampleQueryExecutionId)
                    .build();

            GetQueryExecutionResponse getQueryExecutionResponse = athenaClient
                    .getQueryExecution(getQueryExecutionRequest);
            if (getQueryExecutionResponse.queryExecution()
                    .status()
                    .state()
                    .equals(QueryExecutionState.CANCELLED)) {

                System.out.println("The Amazon Athena query has been cancelled!");
            }

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    // Submits an example query and returns a query execution Id value
    public static String submitAthenaQuery(AthenaClient athenaClient) {
        try {
            QueryExecutionContext queryExecutionContext =
QueryExecutionContext.builder()
                    .database(ExampleConstants.ATHENA_DEFAULT_DATABASE)
                    .build();

            ResultConfiguration resultConfiguration = ResultConfiguration.builder()
                    .outputLocation(ExampleConstants.ATHENA_OUTPUT_BUCKET)
                    .build();
```

```
            StartQueryExecutionRequest startQueryExecutionRequest =
 StartQueryExecutionRequest.builder()
                    .queryExecutionContext(queryExecutionContext)
                    .queryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
                    .resultConfiguration(resultConfiguration).build();

            StartQueryExecutionResponse startQueryExecutionResponse = athenaClient
                    .startQueryExecution(startQueryExecutionRequest);
            return startQueryExecutionResponse.queryExecutionId();

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
        return null;

    }
}
```

쿼리 실행 나열

ListQueryExecutionsExample은 쿼리 실행 ID의 목록을 가져오는 방법을 보여 줍니다.

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.ListQueryExecutionsRequest;
import software.amazon.awssdk.services.athena.model.ListQueryExecutionsResponse;
import software.amazon.awssdk.services.athena.paginators.ListQueryExecutionsIterable;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class ListQueryExecutionsExample {
```

```
    public static void main(String[] args) {
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        listQueryIds(athenaClient);
        athenaClient.close();
    }

    public static void listQueryIds(AthenaClient athenaClient) {
        try {
            ListQueryExecutionsRequest listQueryExecutionsRequest =
 ListQueryExecutionsRequest.builder().build();
            ListQueryExecutionsIterable listQueryExecutionResponses = athenaClient
                    .listQueryExecutionsPaginator(listQueryExecutionsRequest);
            for (ListQueryExecutionsResponse listQueryExecutionResponse :
 listQueryExecutionResponses) {
                List<String> queryExecutionIds =
 listQueryExecutionResponse.queryExecutionIds();
                System.out.println("\n" + queryExecutionIds);
            }

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

명명된 쿼리 생성

`CreateNamedQueryExample`은 명명된 쿼리를 생성하는 방법을 보여 줍니다.

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.CreateNamedQueryRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
```

```
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class CreateNamedQueryExample {
    public static void main(String[] args) {
        final String USAGE = """

                Usage:
                    <name>

                Where:
                    name - the name of the Amazon Athena query.\s
                """;

        if (args.length != 1) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String name = args[0];
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        createNamedQuery(athenaClient, name);
        athenaClient.close();
    }

    public static void createNamedQuery(AthenaClient athenaClient, String name) {
        try {
            // Create the named query request.
            CreateNamedQueryRequest createNamedQueryRequest =
 CreateNamedQueryRequest.builder()
                    .database(ExampleConstants.ATHENA_DEFAULT_DATABASE)
                    .queryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
                    .description("Sample Description")
                    .name(name)
                    .build();

            athenaClient.createNamedQuery(createNamedQueryRequest);
            System.out.println("Done");
```

```
        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

명명된 쿼리 삭제

DeleteNamedQueryExample은 명명된 쿼리 ID를 사용하여 명명된 쿼리를 삭제하는 방법을 보여 줍니다.

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.model.DeleteNamedQueryRequest;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.CreateNamedQueryRequest;
import software.amazon.awssdk.services.athena.model.CreateNamedQueryResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class DeleteNamedQueryExample {
    public static void main(String[] args) {
        final String USAGE = """

                Usage:
                    <name>

                Where:
                    name - the name of the Amazon Athena query.\s
                """;

        if (args.length != 1) {
            System.out.println(USAGE);
```

```
            System.exit(1);
        }

        String name = args[0];
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        String sampleNamedQueryId = getNamedQueryId(athenaClient, name);
        deleteQueryName(athenaClient, sampleNamedQueryId);
        athenaClient.close();
    }

    public static void deleteQueryName(AthenaClient athenaClient, String
sampleNamedQueryId) {
        try {
            DeleteNamedQueryRequest deleteNamedQueryRequest =
DeleteNamedQueryRequest.builder()
                    .namedQueryId(sampleNamedQueryId)
                    .build();

            athenaClient.deleteNamedQuery(deleteNamedQueryRequest);

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static String getNamedQueryId(AthenaClient athenaClient, String name) {
        try {
            CreateNamedQueryRequest createNamedQueryRequest =
CreateNamedQueryRequest.builder()
                    .database(ExampleConstants.ATHENA_DEFAULT_DATABASE)
                    .queryString(ExampleConstants.ATHENA_SAMPLE_QUERY)
                    .name(name)
                    .description("Sample description")
                    .build();

            CreateNamedQueryResponse createNamedQueryResponse =
athenaClient.createNamedQuery(createNamedQueryRequest);
            return createNamedQueryResponse.namedQueryId();

        } catch (AthenaException e) {
```

```
            e.printStackTrace();
            System.exit(1);
        }
        return null;
    }
}
```

명명된 쿼리 나열

ListNamedQueryExample은 명명된 쿼리 ID의 목록을 가져오는 방법을 보여 줍니다.

```
package aws.example.athena;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.athena.AthenaClient;
import software.amazon.awssdk.services.athena.model.AthenaException;
import software.amazon.awssdk.services.athena.model.ListNamedQueriesRequest;
import software.amazon.awssdk.services.athena.model.ListNamedQueriesResponse;
import software.amazon.awssdk.services.athena.paginators.ListNamedQueriesIterable;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */
public class ListNamedQueryExample {
    public static void main(String[] args) {
        AthenaClient athenaClient = AthenaClient.builder()
                .region(Region.US_WEST_2)
                .build();

        listNamedQueries(athenaClient);
        athenaClient.close();
    }

    public static void listNamedQueries(AthenaClient athenaClient) {
        try {
            ListNamedQueriesRequest listNamedQueriesRequest =
 ListNamedQueriesRequest.builder()
```

```
                    .build();

            ListNamedQueriesIterable listNamedQueriesResponses = athenaClient
                    .listNamedQueriesPaginator(listNamedQueriesRequest);
            for (ListNamedQueriesResponse listNamedQueriesResponse :
 listNamedQueriesResponses) {
                List<String> namedQueryIds = listNamedQueriesResponse.namedQueryIds();
                System.out.println(namedQueryIds);
            }

        } catch (AthenaException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Amazon Athena에서 Apache Spark 사용

Amazon Athena를 사용하면 리소스를 계획, 구성 또는 관리할 필요 없이 Apache Spark를 사용하여 데이터 분석 및 탐색을 대화식으로 쉽게 실행할 수 있습니다. Athena에서 Apache Spark 애플리케이션을 실행하는 것은 추가 구성 없이 결과를 직접 처리하고 수신하기 위해 Spark 코드를 제출하는 것을 의미합니다. Amazon Athena 콘솔의 간소화된 노트북 환경을 활용하면 Python 또는 Athena 노트북 API를 통해 Apache Spark 애플리케이션을 개발할 수 있습니다. Amazon Athena의 Apache Spark는 서버리스이며, 자동 온디맨드로 확장하여 즉시 컴퓨팅을 제공함으로써 변화하는 데이터 볼륨 및 처리 요구 사항을 충족할 수 있습니다.

Amazon Athena는 다음과 같은 기능을 제공합니다.

- 콘솔 사용 현황 - Amazon Athena 콘솔에서 Spark 애플리케이션을 제출합니다.
- 스크립팅 - Python에서 Apache Spark 애플리케이션을 대화식으로 빠르게 빌드하고 디버그합니다.
- 동적 조정 - Amazon Athena에서 작업을 실행하는 데 필요한 컴퓨팅 및 메모리 리소스를 자동으로 결정하고 지정된 최대값까지 해당 리소스를 지속적으로 확장합니다. 이러한 동적 조정은 속도에 영향을 주지 않으면서 비용을 절감합니다.
- 노트북 환경 - Athena 노트북 편집기를 사용하면 익숙한 인터페이스로 계산을 생성, 편집, 실행할 수 있습니다. Athena 노트북은 Jupyter 노트북과 호환되며 계산을 위해 순서대로 실행되는 셀 목록을 포함합니다. 셀 콘텐츠에는 코드, 텍스트, 마크다운, 수학, 도표 및 리치 미디어가 포함될 수 있습니다.

자세한 내용은 AWS 빅 데이터 블로그의 Explore your data lake using Amazon Athena for Apache Spark를 참조하세요.

고려 사항 및 제한

- 현재 Apache Spark용 Amazon Athena는 다음 AWS 리전에서 사용할 수 있습니다.
  - 아시아 태평양(뭄바이)
  - 아시아 태평양(싱가포르)
  - 아시아 태평양(시드니)
  - 아시아 태평양(도쿄)
  - 유럽(프랑크푸르트)
  - 유럽(아일랜드)

  - 미국 동부(버지니아 북부)
  - 미국 동부(오하이오)
  - 미국 서부(오레곤)
- AWS Lake Formation는 지원되지 않습니다.
- 파티션 프로젝션을 사용하는 테이블은 지원되지 않습니다.
- Apache Spark 지원 작업 그룹은 Athena 노트북 편집기를 사용할 수 있지만 Athena 쿼리 편집기는 사용할 수 없습니다. Athena SQL 작업 그룹만 Athena 쿼리 편집기를 사용할 수 있습니다.
- 교차 리전 보기 쿼리는 지원되지 않습니다. Athena SQL에서 생성된 보기는 Athena for Spark에서 쿼리할 수 없습니다. 두 엔진의 보기는 다르게 구현되므로 엔진 간에 교차하여 사용할 수 없습니다.
- MLlib(Apache Spark 기계 학습 라이브러리) 및 `pyspark.ml` 패키지는 지원되지 않습니다. 지원되는 Python 라이브러리 목록은 사전 설치된 Python 라이브러리 목록 단원을 참조하세요.
- 현재 `pip install`은 Athena for Spark 세션에서 지원되지 않습니다.
- 노트북당 하나의 활성 세션만 허용됩니다.
- 여러 사용자가 콘솔을 사용하여 작업 그룹의 기존 세션을 열 때 동일한 노트북에 액세스합니다. 혼동을 피하려면 직접 생성한 세션만 여세요.
- Amazon Athena와 함께 사용할 수 있는 Apache Spark 애플리케이션용 호스팅 도메인(예: `analytics-gateway.us-east-1.amazonaws.com`)은 인터넷 PSL(Public Suffix List)에 등록됩니다. 도메인에 민감한 쿠키를 설정해야 하는 경우 사이트 간 요청 위조(CSRF) 시도로부터 도메인을 보호할 수 있도록 `__Host-` 접두사가 붙은 쿠키를 사용하는 것이 좋습니다. 자세한 내용은 Mozilla.org 개발자 설명서의 Set-Cookie 페이지를 참조하세요.
- Athena의 Spark 노트북, 세션 및 작업 그룹 문제 해결에 대한 자세한 내용은 Athena for Spark 문제 해결 단원을 참조하세요.

Amazon Athena에서 Apache Spark 시작하기

Amazon Athena에서 Apache Spark를 시작하려면 먼저 Spark 지원 작업 그룹을 생성해야 합니다. 작업 그룹으로 전환한 후 노트북을 생성하거나 기존 노트북을 열 수 있습니다. Athena에서 노트북을 열면 해당 노트북에 대한 새 세션이 자동으로 시작되고 Athena Notebook 편집기에서 직접 노트북으로 작업할 수 있습니다.

> **Note**
>
> 노트북을 생성하기 전에 Spark 지원 작업 그룹을 만들어야 합니다.

Athena에서 Spark 지원 작업 그룹 생성

Athena에서 작업 그룹을 사용하여 사용자, 팀, 애플리케이션 또는 작업 그룹을 그룹화하고 비용을 추적할 수 있습니다. Amazon Athena에서 Apache Spark를 사용하려면 Spark 엔진을 사용하는 Amazon Athena 작업 그룹을 생성합니다.

> **Note**
>
> Apache Spark 지원 작업 그룹은 Athena 노트북 편집기를 사용할 수 있지만 Athena 쿼리 편집기는 사용할 수 없습니다. Athena SQL 작업 그룹만 Athena 쿼리 편집기를 사용할 수 있습니다.

Athena에서 Spark 지원 작업 그룹을 생성하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

3. 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
4. 작업 그룹 페이지에서 작업 그룹 생성을 선택합니다.
5. Workgroup name(작업 그룹 이름)에 Apache Spark 작업 그룹의 이름을 입력합니다.
6. (선택 사항) Description(설명)에 작업 그룹에 대한 설명을 입력합니다.
7. Analytics engine(분석 엔진)에서 Apache Spark를 선택합니다.

> **Note**
>
> 작업 그룹을 생성한 이후에 작업 그룹의 분석 엔진 유형을 변경할 수 없습니다. 예를 들어 Athena 엔진 버전 3 작업 그룹을 PySpark 엔진 버전 3 작업 그룹으로 변경할 수 없습니다.

8. 이 자습서에서는 Turn on example notebook(예제 노트북 켜기)을 선택합니다. 이 선택적 기능은 이름이 `example-notebook-random_string`인 예제 노트북을 작업 그룹에 추가하고, 노트북에서 계정의 특정 데이터베이스 및 테이블을 생성, 표시, 삭제하는 데 사용되는 AWS Glue 관련 권한과 샘플 데이터 세트에 대한 Amazon S3 읽기 권한을 추가합니다. 추가된 권한을 보려면 View additional permissions details(추가 권한 세부 정보 보기)를 선택합니다.

> **Note**
>
> 예제 노트북을 실행하면 추가 비용이 발생할 수 있습니다.

9. Additional configurations(추가 구성)에서 다음을 수행합니다.

   - Use defaults(기본값 사용) 설정을 사용합니다. 이 옵션은 기본값이며 Spark 지원 작업 그룹을 시작하는 데 도움이 됩니다. 이 옵션을 사용하면 Athena에서 Amazon S3에 IAM 역할 및 계산 결과 위치를 자동으로 생성합니다. IAM 역할의 이름과 생성할 S3 버킷 위치는 Additional configurations(추가 구성) 머리글 아래 상자에 표시됩니다.
   - Use defaults(기본값 사용) 설정을 비활성화한 다음 작업 그룹 구성 사용자 지정 단원의 단계를 계속 진행하여 작업 그룹을 수동으로 구성합니다.

10. (선택 사항) Tags(태그) - 이 옵션을 사용하여 작업 그룹에 태그를 추가합니다. 자세한 내용은 Athena 리소스 태깅 단원을 참조하십시오.
11. 작업 그룹 생성을 선택합니다. 작업 그룹이 생성되었다는 메시지가 표시되고 작업 그룹이 작업 그룹 목록에 표시됩니다.

작업 그룹 구성 사용자 지정

노트북에 대한 IAM 역할 및 계산 결과 위치를 지정하려면 이 단원의 단계를 따르세요. Additional configurations(추가 구성) 옵션에서 Use defaults(기본값 사용)를 선택한 경우 이 단원을 건너뛰고 노트북 탐색기 열기 및 작업 그룹 전환 로 바로 이동합니다.

다음 절차에서는 이전 단원에서 To create a Spark enabled workgroup in Athena(Athena에서 Spark 지원 작업 그룹을 생성하려면) 절차의 1~9단계를 완료했다고 가정합니다.

작업 그룹 구성을 사용자 지정하려면

1. 사용자의 고유 IAM 역할을 생성 또는 사용하거나 노트북 암호화를 구성하려면 IAM role configuration(IAM 역할 구성)을 확장합니다.

   - Service Role(서비스 역할)에서 다음 중 하나를 선택합니다.
     - 서비스 역할 생성 - Athena에서 서비스 역할을 자동으로 생성하려면 이 옵션을 선택합니다. 역할에 부여되는 권한을 보려면 View permission details(권한 세부 정보 보기)를 선택합니다.
     - 기존 서비스 역할 선택 - 드롭다운 목록에서 기존 IAM 역할을 선택합니다. 선택한 역할에는 첫 번째 옵션의 권한이 포함되어야 합니다. 노트북 지원 작업 그룹의 권한에 대한 자세한 내용은 Spark 지원 작업 그룹 문제 해결 단원을 참조하세요.
   - Notebook and calculation code encryption key management(노트북 및 계산 코드 암호화 키 관리)에서 다음 옵션 중 하나를 선택합니다.
     - Amazon Athena 소유 - AWS KMS 키를 Amazon Athena에서 소유하고 관리합니다. 이 키 사용에 대한 추가 요금은 부과되지 않습니다.
     - 계정에 저장되고 사용자가 소유 및 관리하는 대칭 키 - 이 옵션에 대해 다음 중 하나를 수행합니다.
       - 기존 키를 사용하려면 검색 상자를 사용하여 AWS KMS를 선택하거나 키 ARN을 입력합니다.
       - AWS KMS 콘솔에서 키를 생성하려면 AWS KMS 키 생성을 선택합니다. 실행 역할은 생성한 키를 사용할 수 있는 권한이 있어야 합니다.

   > **Important**
   >
   > 작업 그룹의 AWS KMS key를 변경해도 업데이트 이전에 관리된 노트북에서는 여전히 이전 KMS 키를 참조합니다. 업데이트 이후에 관리되는 노트북에서는 새 KMS 키를 사용합니다. 새 KMS 키를 참조하도록 이전 노트북을 업데이트하려면 각 이전 노트북을 내보낸 다음 가져옵니다. 이전 노트북 참조를 새 KMS 키로 업데이트하기 전에 이전 KMS 키를 삭제하면 이전 노트북을 더 이상 해독할 수 없으므로 복구할 수 없습니다.
   > 이 동작은 KMS 키의 표시 이름인 별칭에 대한 업데이트에도 적용됩니다. 새 KMS 키를 가리키도록 KMS 키 별칭을 업데이트할 경우 별칭 업데이트 이전에 관리되는 노트북에서는 여전히 이전 KMS 키를 참조하고 별칭 업데이트 이후에 관리되는 노트북에서는 새 KMS 키를 사용합니다. KMS 키 또는 별칭을 업데이트하기 이전에 다음 사항을 고려하세요.

2. 사용자의 고유 계산 결과 설정을 지정하려면 Calculation result settings(계산 결과 설정)를 확장한 후 다음 옵션 중에서 선택합니다.

   - 새 S3 버킷 생성 - 이 옵션은 계산 결과를 위한 Amazon S3 버킷을 계정에 생성합니다. 버킷 이름은 *account_id*-*region*-athena-results-bucket-*alphanumeric_id* 형식을 따르며, ACL 비활성화, 퍼블릭 액세스 차단, 버전 관리 비활성화, 버킷 소유자 적용 설정을 사용합니다.
   - 기존 S3 위치 선택 - 이 옵션에 대해 다음을 수행합니다.
     - 검색 상자에 기존 위치의 S3 경로를 입력하거나 Browse S3(S3 찾아보기)를 선택하여 목록에서 버킷을 선택합니다.

       > **Note**
       >
       > Amazon S3에서 기존 위치를 선택할 때 해당 위치에 슬래시(/)를 추가하지 마세요. 그러면 계산 세부 정보 페이지의 계산 결과 위치 링크가 잘못된 디렉터리를 가리키게 됩니다. 이 경우 작업 그룹의 결과 위치를 편집하여 후행 슬래시를 제거합니다.

     - (선택 사항) 선택한 기존 버킷에 대한 자세한 내용을 보려면 View(보기)를 선택하여 Amazon S3 콘솔의 Buckets(버킷) 페이지를 엽니다.
     - (선택 사항) Expected bucket owner(예상 버킷 소유자)에 쿼리 결과 출력 위치 버킷의 소유자가 될 것으로 예상되는 AWS 계정 ID를 입력합니다. 가능하면 이 옵션을 추가 보안 조치로 선택하는 것이 좋습니다. 버킷 소유자의 계정 ID가 지정한 ID와 일치하지 않으면 버킷으로의 출력 시도가 실패합니다. 자세한 내용은 Amazon S3 사용 설명서의 버킷 소유자 조건을 사용하여 버킷 소유권 확인을 참조하세요.
     - (선택 사항) 계산 결과 위치를 다른 계정에서 소유한 경우 해당 계정에 쿼리 결과에 대한 전체 제어 권한을 부여하려면 Assign bucket owner full control over query results(버킷 소유자에게 쿼리 결과에 대한 전체 제어 권한 할당)를 선택합니다.

3. (선택 사항) Encrypt calculation results(계산 결과 암호화)를 선택한 후 다음 중 하나를 선택합니다.

   - SSE_S3 - S3 관리형 서버 측 암호화 키입니다.
   - SSE_KMS - 사용자가 제공하는 키입니다. AWS KMS 키 선택에서 다음 중 하나를 선택할 수 있습니다.
     - AWS 소유 키 사용 - AWS에서 소유하고 관리하는 키를 사용합니다.
     - 다른 AWS KMS 키 선택(고급) - 키를 선택하거나 생성합니다.
       - 기존 키를 사용하려면 검색 상자를 사용하여 AWS KMS를 선택하거나 키 ARN을 입력합니다.

- KMS 콘솔에서 키를 생성하려면 AWS KMS 키 생성을 선택합니다. KMS 콘솔에서 키 생성을 완료한 후 Athena 콘솔의 작업 그룹 생성 페이지로 돌아가서 AWS KMS 키 선택 또는 ARN 입력 검색 상자를 사용하여 방금 생성한 키를 선택합니다.

4. (선택 사항) Other settings(기타 설정) - 이 옵션을 확장하여 작업 그룹에 대한 Publish CloudWatch metrics(CloudWatch 지표 게시) 옵션을 활성화하거나 비활성화합니다. 이 필드는 기본적으로 선택됩니다. 자세한 내용은 CloudWatch 지표를 사용하여 Apache Spark 계산 모니터링 단원을 참조하십시오.
5. (선택 사항) Tags(태그) - 이 옵션을 사용하여 작업 그룹에 태그를 추가합니다. 자세한 내용은 Athena 리소스 태깅 단원을 참조하십시오.
6. 작업 그룹 생성을 선택합니다. 작업 그룹이 생성되었다는 메시지가 표시되고 작업 그룹이 작업 그룹 목록에 표시됩니다.

노트북 탐색기 열기 및 작업 그룹 전환

방금 생성한 Spark 지원 작업 그룹을 사용하려면 먼저 해당 작업 그룹으로 전환해야 합니다. Spark 지원 작업 그룹을 전환하려면 노트북 탐색기 또는 노트북 편집기에서 Workgroup(작업 그룹) 옵션을 사용할 수 있습니다.

> **Note**
>
> 시작하기 전에 브라우저가 서드 파티 쿠키를 차단하지 않는지 확인합니다. 브라우저가 기본값 또는 사용자가 지정한 설정으로 서드 파티 쿠키를 차단하는 경우 노트북이 실행되지 않습니다. 쿠키 관리에 관한 자세한 내용은 다음을 참조하세요.
>
> - Chrome
> - Firefox
> - Safari

노트북 탐색기를 열고 작업 그룹을 전환하려면

1. 탐색 창에서 Notebook explorer(노트북 탐색기)를 선택합니다.
2. 콘솔 오른쪽 상단의 Workgroup(작업 그룹) 옵션을 사용하여 생성한 Spark 지원 작업 그룹을 선택합니다. 예제 노트북이 노트북 목록에 표시됩니다.

   다음과 같은 방법으로 노트북 탐색기를 사용할 수 있습니다.

- 연결된 노트북 이름을 선택하여 새 세션에서 노트북을 엽니다.
- 노트북의 이름을 바꾸거나, 삭제하거나, 내보내려면 Actions(작업) 메뉴를 사용합니다.
- 노트북 파일을 가져오려면 Import file(파일 가져오기)을 선택합니다.
- 노트북을 생성하려면 Create notebook(노트북 생성)을 선택합니다.

예제 노트북 실행

샘플 노트북에서는 공개적으로 사용 가능한 뉴욕시 택시 여행 데이터 세트에서 데이터를 쿼리합니다. 노트북에는 Spark DataFrames, Spark SQL, AWS Glue Data Catalog를 사용하는 방법을 보여주는 예제가 나와 있습니다.

예제 노트북을 실행하려면

1. 노트북 탐색기에서 연결된 예제 노트북의 이름을 선택합니다.

   그러면 기본 파라미터를 사용하여 노트북 세션이 시작되고 노트북 편집기에서 노트북이 열립니다. 새 Apache Spark 세션이 기본 파라미터(최대 20개 DPU)를 사용하여 시작되었다는 메시지가 표시됩니다.
2. 셀을 순서대로 실행하고 결과를 확인하려면 노트북의 각 셀에 대해 Run(실행) 버튼을 한 번씩 선택합니다.

   - 아래로 스크롤하여 결과를 확인하고 새 셀을 보기로 불러옵니다.
   - 셀에 계산이 있는 경우 진행률 표시줄에 완료율, 경과된 시간 및 남은 시간이 표시됩니다.
   - 예제 노트북에서 사용자 계정에 샘플 데이터베이스 및 테이블을 생성합니다. 마지막 셀은 정리 단계로 이들 항목을 제거합니다.

> **Note**
>
> 예제 노트북에서 폴더, 테이블 또는 데이터베이스 이름을 변경하는 경우 해당 변경 내용이 사용 중인 IAM 역할에 반영되는지 확인합니다. 그렇지 않으면 권한이 부족하여 노트북이 실행되지 않을 수 있습니다.

세션 세부 정보 편집

노트북 세션을 시작한 후 테이블 형식, 암호화, 세션 유휴 제한 시간, 사용할 최대 동시 데이터 처리 장치(DPU) 수와 같은 세션 세부 정보를 편집할 수 있습니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다.

세션 세부 정보를 편집하려면

1. 노트북 편집기의 오른쪽 상단 Session(세션) 메뉴에서 Edit session(세션 편집)을 선택합니다.
2. 세션 세부 정보 편집 대화 상자의 Spark 속성 섹션에서 다음 옵션에 대한 값을 선택하거나 입력합니다.

   - 추가 테이블 형식 – Linux Foundation Delta Lake, Apache Hudi, Apache Iceberg 또는 사용자 지정을 선택합니다.
     - Delta, Hudi 또는 Iceberg 테이블 옵션에서 해당 테이블 형식에 필요한 테이블 속성은 테이블에서 편집 및 JSON으로 편집 옵션에 자동으로 제공됩니다. 이러한 테이블 형식 사용에 대한 자세한 내용은 Amazon Athena for Apache Spark에서 비 Hive 테이블 형식 사용 섹션을 참조하세요.
     - 사용자 정의 또는 기타 테이블 유형에 대한 테이블 속성을 추가하거나 제거하려면 테이블에서 편집 및 JSON으로 편집 옵션을 사용합니다.
     - 테이블에서 편집 옵션의 경우 속성 추가를 선택하여 속성을 추가하거나 제거를 선택하여 속성을 제거합니다. 속성 이름과 값을 입력하려면 키 및 값 상자를 사용합니다.
     - JSON으로 편집 옵션의 경우 JSON 텍스트 편집기를 사용하여 구성을 직접 편집합니다.
       - JSON 텍스트를 클립보드에 복사하려면 복사를 선택합니다.
       - JSON 편집기에서 모든 텍스트를 제거하려면 지우기를 선택합니다.
       - 줄 바꿈을 구성하거나 JSON 편집기의 색상 테마를 선택하려면 설정(톱니) 아이콘을 선택합니다.
   - Spark 암호화 켜기 - 디스크에 기록되어 Spark 네트워크 노드를 통해 전송되는 데이터를 암호화하려면 이 옵션을 선택합니다. 자세한 내용은 Apache Spark 암호화 활성화 단원을 참조하십시오.
3. 세션 파라미터 섹션에서 다음 옵션에 대한 값을 선택하거나 입력합니다.

   - Session idle timeout(세션 유휴 제한 시간) - 1~480분 사이의 값을 선택하거나 입력합니다. 기본값은 20입니다.

- Coordinator size(코디네이터 크기) - 코디네이터는 노트북 세션에서 처리 작업을 조율하고 다른 실행기를 관리하는 특수 실행기입니다. 현재는 1 DPU가 기본값이며 사용 가능한 유일한 값입니다.
- Executor size(실행기 크기) - 실행기는 노트북 세션이 Athena에서 요청할 수 있는 가장 작은 컴퓨팅 단위입니다. 현재는 1 DPU가 기본값이며 사용 가능한 유일한 값입니다.
- Max concurrent value(최대 동시 값) - 동시에 실행할 수 있는 최대 DPU 수입니다. 기본값은 20이고 최소값은 3이며 최대값은 60입니다. 이 값을 늘려도 추가 리소스가 자동으로 할당되지 않지만 Athena에서 컴퓨팅 로드에 필요하고 리소스가 사용 가능할 때 지정된 최대값까지 할당하려고 시도합니다.

4. Save(저장)를 선택합니다.
5. Confirm edit(편집 확인) 프롬프트에서 Confirm(확인)을 선택합니다.

   Athena에서 노트북을 저장하고 지정된 파라미터로 새 세션을 시작합니다. 노트북 편집기의 배너에 수정된 파라미터로 새 세션이 시작되었다는 알림이 표시됩니다.

> **Note**
>
> 노트북의 세션 설정이 Athena에 저장됩니다. 세션 파라미터를 편집하고 세션을 종료한 후 다음에 노트북 세션을 시작하면 구성된 세션 파라미터가 Athena에서 사용됩니다.

세션 및 계산 세부 정보 보기

노트북을 실행한 후 세션 및 계산 세부 정보를 볼 수 있습니다.

세션 및 계산 세부 정보를 보려면

1. 오른쪽 상단의 Session(세션) 메뉴에서 View details(세부 정보 보기)를 선택합니다.
   - Current session(현재 세션) 탭에 세션 ID, 생성 시간, 상태, 작업 그룹 등 현재 세션에 대한 정보가 표시됩니다.
   - History(기록) 탭에 이전 세션의 세션 ID가 나열됩니다. 이전 세션의 세부 정보를 보려면 History(기록) 탭을 선택한 다음 목록에서 세션 ID를 선택합니다.
   - Calculations(계산) 섹션에는 세션에서 실행된 계산 목록이 표시됩니다.
2. 계산에 대한 세부 정보를 보려면 계산 ID를 선택합니다.
3. Calculation details(계산 세부 정보) 페이지에서 다음 작업을 수행할 수 있습니다.

- 계산에 대한 코드를 보려면 Code(코드) 섹션을 확인하세요.
- 계산 결과를 보려면 Results(결과) 탭을 선택합니다.
- 텍스트 형식으로 표시된 결과를 다운로드하려면 Download results(결과 다운로드)를 선택합니다.
- Amazon S3에서 계산 결과에 대한 정보를 보려면 View in S3(S3에서 보기)를 선택합니다.

세션 종료

노트북 세션을 종료하려면

1. 노트북 편집기의 오른쪽 상단 Session(세션) 메뉴에서 Terminate(종료)를 선택합니다.
2. Confirm session termination(세션 종료 확인) 프롬프트에서 Confirm(확인)을 선택합니다. 노트북이 저장되고 노트북 편집기로 돌아갑니다.

> **Note**
>
> 노트북 편집기에서 노트북 탭을 닫더라도 활성 노트북의 세션이 종료되지 않습니다. 세션이 종료되었는지 확인하려면 Session(세션), Terminate (종료) 옵션을 사용합니다.

사용자 노트북 생성

Spark 지원 Athena 작업 그룹을 생성한 후 사용자 노트북을 만들 수 있습니다.

노트북을 생성하려면

1. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.
2. Athena 콘솔 탐색 창에서 Notebook explorer(노트북 탐색기) 또는 Notebook editor(노트북 편집기)를 선택합니다.
3. 다음 중 하나를 수행하십시오.
   - Notebook explorer(노트북 탐색기)에서 Create notebook(노트북 생성)을 선택합니다.
   - Notebook editor(노트북 편집기)에서 Create notebook(노트북 생성)을 선택하거나 더하기 아이콘(+)을 선택하여 노트북을 추가합니다.
4. Create notebook(노트북 생성) 대화 상자에서 Notebook name(노트북 이름)에 이름을 입력합니다.

5. (선택 사항) Spark 속성을 확장하고 다음 옵션에 대한 값을 선택하거나 입력합니다.

   - 추가 테이블 형식 – Linux Foundation Delta Lake, Apache Hudi, Apache Iceberg 또는 사용자 지정을 선택합니다.
     - Delta, Hudi 또는 Iceberg 테이블 옵션에서 해당 테이블 형식에 필요한 테이블 속성은 테이블에서 편집 및 JSON으로 편집 옵션에 자동으로 제공됩니다. 이러한 테이블 형식 사용에 대한 자세한 내용은 Amazon Athena for Apache Spark에서 비 Hive 테이블 형식 사용 섹션을 참조하세요.
     - 사용자 정의 또는 기타 테이블 유형에 대한 테이블 속성을 추가하거나 제거하려면 테이블에서 편집 및 JSON으로 편집 옵션을 사용합니다.
     - 테이블에서 편집 옵션의 경우 속성 추가를 선택하여 속성을 추가하거나 제거를 선택하여 속성을 제거합니다. 속성 이름과 값을 입력하려면 키 및 값 상자를 사용합니다.
     - JSON으로 편집 옵션의 경우 JSON 텍스트 편집기를 사용하여 구성을 직접 편집합니다.
       - JSON 텍스트를 클립보드에 복사하려면 복사를 선택합니다.
       - JSON 편집기에서 모든 텍스트를 제거하려면 지우기를 선택합니다.
       - 줄 바꿈을 구성하거나 JSON 편집기의 색상 테마를 선택하려면 설정(톱니) 아이콘을 선택합니다.
   - Spark 암호화 켜기 - 디스크에 기록되어 Spark 네트워크 노드를 통해 전송되는 데이터를 암호화하려면 이 옵션을 선택합니다. 자세한 내용은 Apache Spark 암호화 활성화 단원을 참조하십시오.
6. (선택 사항) Session parameters(세션 파라미터)를 확장한 후 다음 옵션의 값을 선택하거나 입력합니다.

   - Session idle timeout(세션 유휴 제한 시간) - 1~480분 사이의 값을 선택하거나 입력합니다. 기본값은 20입니다.
   - Coordinator size(코디네이터 크기) - 코디네이터는 노트북 세션에서 처리 작업을 조율하고 다른 실행기를 관리하는 특수 실행기입니다. 현재는 1 DPU가 기본값이며 사용 가능한 유일한 값입니다. DPU(데이터 처리 단위)는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다.
   - Executor size(실행기 크기) - 실행기는 노트북 세션이 Athena에서 요청할 수 있는 가장 작은 컴퓨팅 단위입니다. 현재는 1 DPU가 기본값이며 사용 가능한 유일한 값입니다.
   - Max concurrent value(최대 동시 값) - 동시에 실행할 수 있는 최대 DPU 수입니다. 기본값은 20이고 최대값은 60입니다. 이 값을 늘려도 추가 리소스가 자동으로 할당되지 않지만 Athena에서 컴퓨팅 로드에 필요하고 리소스가 사용 가능할 때 지정된 최대값까지 할당하려고 시도합니다.
7. 생성(Create)을 선택합니다. 노트북이 노트북 편집기의 새 세션에서 열립니다.

이전에 생성된 노트북 열기

이전에 생성된 노트북을 열려면

1. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.
2. Athena 콘솔 탐색 창에서 Notebook editor(노트북 편집기) 또는 Notebook explorer(노트북 탐색기)를 선택합니다.
3. 다음 중 하나를 수행하십시오.

   - Notebook editor(노트북 편집기)의 Recent notebooks(최근 노트북) 또는 Saved notebooks(저장된 노트북) 목록에서 노트북을 선택합니다. 노트북이 새 세션에서 열립니다.
   - Notebook explorer(노트북 탐색기)의 목록에서 노트북의 이름을 선택합니다. 노트북이 새 세션에서 열립니다.

노트북 파일 관리에 대한 자세한 내용은 노트북 파일 관리 단원을 참조하세요.

노트북으로 작업

Athena 노트북 탐색기에서 노트북을 관리하고 Athena 노트북 편집기를 사용하여 세션에서 노트북을 편집하고 실행할 수 있습니다. 요구 사항에 따라 노트북 세션에 대한 DPU 사용을 구성할 수 있습니다.

노트북을 중지하면 연결된 세션이 종료됩니다. 모든 파일이 저장되지만 선언된 변수, 함수 및 클래스에서 진행 중인 변경 사항은 손실됩니다. 노트북을 다시 시작하면 노트북 파일이 다시 로드되고 코드를 다시 실행할 수 있습니다.

세션 및 계산

각 노트북은 단일 Python 커널과 연결되고 Python 코드를 실행합니다. 노트북에는 명령을 포함하는 셀이 하나 이상 존재할 수 있습니다. 노트북에서 셀을 실행하려면 먼저 노트북에 대한 세션을 생성합니다. 세션에서는 노트북의 변수 및 상태를 추적합니다.

노트북에서 셀을 실행한다는 것은 현재 세션에서 계산을 실행하는 것을 의미합니다. 계산은 노트북의 상태를 진행하며 Amazon S3에서 읽기, 다른 데이터 스토어에 쓰기와 같은 작업을 수행할 수 있습니다. 세션이 실행 중인 동안 계산을 통해 노트북에 대해 유지되는 상태를 사용 및 수정합니다.

상태가 더 이상 필요 없는 경우 세션을 종료할 수 있습니다. 세션을 종료하면 노트북은 그대로 유지되지만 변수 및 기타 상태 정보는 삭제됩니다. 여러 프로젝트를 동시에 진행해야 하는 경우 프로젝트별로 하나의 세션을 생성할 수 있으며, 세션은 서로 독립적입니다.

세션에는 DPU로 측정되는 전용 컴퓨팅 용량이 있습니다. 세션을 생성할 때 세션에 여러 DPU를 할당할 수 있습니다. 작업 요구 사항에 따라 세션마다 용량이 다를 수 있습니다.

Athena 노트북 편집기 사용

Athena 노트북 편집기는 코드를 작성하고 실행하기 위한 대화형 환경입니다. 다음 단원에서는 환경의 기능에 대해 설명합니다.

명령 모드 vs. 편집 모드

노트북 편집기에는 셀에 텍스트를 입력하는 편집 모드와 복사, 붙여넣기 또는 실행과 같은 명령을 편집기 자체에 실행하는 명령 모드라는 모달 사용자 인터페이스가 있습니다.

편집 모드와 명령 모드를 사용하려면 다음 작업을 수행할 수 있습니다.

- 편집 모드로 전환하려면 **ENTER** 키를 누르거나 셀을 선택합니다. 셀이 편집 모드에 있는 경우 셀의 왼쪽 여백이 녹색으로 표시됩니다.
- 명령 모드로 전환하려면 **ESC** 키를 누르거나 셀 외부를 클릭합니다. 일반적으로 명령은 모든 셀이 아닌 현재 선택한 셀에만 적용됩니다. 편집기가 명령 모드에 있는 경우 셀의 왼쪽 여백이 파란색으로 표시됩니다.
- 명령 모드에서는 키보드 바로 가기와 편집기 위의 메뉴를 사용할 수 있지만 개별 셀에 텍스트를 입력할 수는 없습니다.
- 셀을 선택하려면 해당 셀을 선택합니다.
- 모든 셀을 선택하려면 **Ctrl+A**(Windows) 또는 **Cmd+A**(Mac) 키를 누릅니다.

노트북 편집기 메뉴

노트북 편집기 상단의 메뉴에 있는 아이콘은 다음 옵션을 제공합니다.

- 저장 - 노트북의 현재 상태를 저장합니다.
- 아래에 셀 삽입 - 현재 선택한 셀 아래에 새 (빈) 셀을 추가합니다.
- 선택한 셀 잘라내기 - 선택한 셀을 현재 위치에서 제거하고 셀을 메모리에 복사합니다.
- 선택한 셀 복사 - 선택한 셀을 메모리에 복사합니다.
- 아래에 셀 붙여넣기 - 복사한 셀을 현재 셀 아래에 붙여넣습니다.
- 선택한 셀을 위로 이동 - 현재 셀을 위 셀 위로 이동합니다.

- 선택한 셀을 아래로 이동 - 현재 셀을 아래 셀 아래로 이동합니다.
- 실행 - 현재 (선택한) 셀을 실행합니다. 출력은 현재 셀 바로 아래에 표시됩니다.
- 모두 실행 - 노트북의 모든 셀을 실행합니다. 각 셀에 대한 출력은 셀 바로 아래에 표시됩니다.
- 중지(커널 중단) - 커널을 중단하여 현재 노트북을 중지합니다.
- 서식 옵션 - 다음 중에서 셀 서식을 선택합니다.
  - 코드 - Python 코드에 사용합니다(기본값).
  - 마크다운 - GitHub 스타일 마크다운 형식으로 텍스트를 입력하는 데 사용합니다. 마크다운을 렌더링하려면 셀을 실행합니다.
  - 원시 NBConvert - 수정되지 않은 형식으로 텍스트를 입력하는 데 사용합니다. Raw NBConvert(원시 NBConvert)로 표시된 셀은 Jupyter nbconvert 명령줄 도구를 사용하여 HTML과 같은 다른 형식으로 변환할 수 있습니다.
- 머리글 - 셀의 머리글 수준을 변경하는 데 사용합니다.
- 명령 팔레트 - Jupyter Notebook 명령과 키보드 바로 가기를 포함합니다. 키보드 바로 가기에 대한 자세한 내용은 이 문서의 뒷부분에 나오는 단원을 참조하세요.
- 세션 - 이 메뉴의 옵션을 사용하여 세션에 대한 세부 정보를 보거나, 세션 파라미터를 편집하거나, 세션을 종료할 수 있습니다.

명령 모드 키보드 바로 가기

다음은 몇 가지 일반적인 노트북 편집기 명령 모드 키보드 바로 가기입니다. 이 바로 가기는 **ESC** 키를 눌러 명령 모드로 전환한 이후에 사용할 수 있습니다. 편집기에서 사용할 수 있는 전체 명령 목록을 보려면 **ESC + H**를 누르세요.

키	작업
1 - 6	셀 유형을 마크다운으로 변경하고 머리글 수준을 입력한 숫자로 설정합니다.
a	현재 셀 위에 셀 생성
b	현재 셀 아래에 셀 생성
c	현재 셀을 메모리에 복사
d d	현재 셀 삭제
h	키보드 바로 가기 도움말 화면 표시

키	작업
j	한 셀 아래로 이동
k	한 셀 위로 이동
m	현재 셀 형식을 마크다운으로 변경
r	현재 셀 형식을 원시로 변경
s	노트북 저장
v	현재 셀 아래에 메모리 내용 붙여넣기
x	선택한 셀 잘라내기
y	셀 형식을 코드로 변경
z	실행 취소
Ctrl+Enter	현재 셀을 실행하고 명령 모드로 전환
Shift+Enter 또는 **Alt+Enter**	현재 셀을 실행하고 출력 아래에 새 셀을 생성한 다음 편집 모드에서 새 셀 입력
Space	한 페이지 아래로 이동
Shift+Space	한 페이지 위로 이동
Shift + L	셀에서 행 번호 표시 여부 전환

명령 모드 바로 가기 편집

노트북 편집기에는 명령 모드 키보드 바로 가기를 사용자 지정하는 옵션이 있습니다.

명령 모드 바로 가기를 편집하려면

1. 노트북 편집기 메뉴에서 Command palette(명령 팔레트)를 선택합니다.

2. 명령 팔레트에서 Edit command mode keyboard shortcuts(명령 모드 키보드 바로 가기 편집) 명령을 선택합니다.
3. Edit command mode shortcuts(명령 모드 바로 가기 편집) 인터페이스를 사용하여 원하는 명령을 키보드에 매핑하거나 다시 매핑합니다.

   명령 모드 바로 가기 편집 지침을 보려면 Edit command mode shortcuts(명령 모드 바로 가기 편집) 화면의 아래쪽으로 스크롤하세요.

Athena for Apache Spark에서 매직 명령을 사용하는 방법에 대한 자세한 내용은 매직 명령 사용 섹션을 참조하세요.

매직 명령 사용

매직 명령 또는 매직은 노트북 셀에서 실행할 수 있는 특수 명령입니다. 예를 들어 `%env`는 노트북 세션의 환경 변수를 표시합니다. Athena IPython 6.0.3에서 매직 함수를 지원합니다.

이 섹션에서는 Athena for Apache Spark에서 몇 가지 주요 매직 명령을 보여줍니다.

- Athena의 매직 명령 목록을 보려면 노트북 셀에서 **`%lsmagic`** 명령을 실행합니다.
- Athena 노트북에서 매직을 사용하여 그래프를 생성하는 방법에 대한 자세한 내용은 데이터 그래프 생성을 위한 매직 섹션을 참조하세요.
- 추가 매직 명령에 대한 자세한 내용은 IPython 설명서의 Built-in magic commands를 참조하세요.

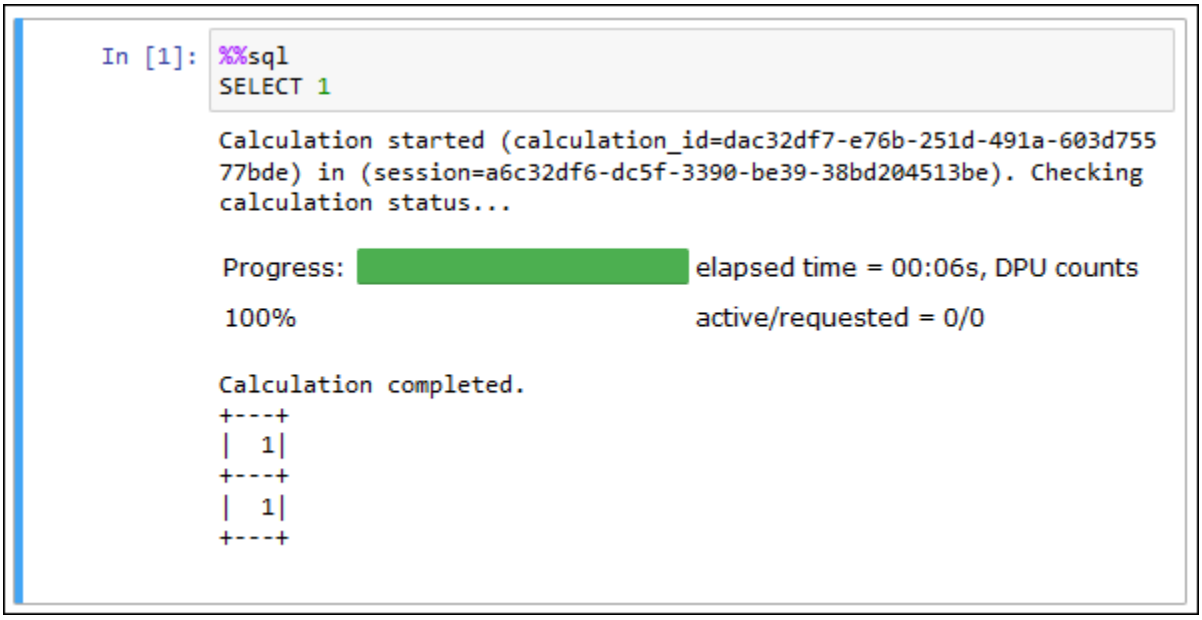
> **Note**
>
> 현재 `%pip` 명령을 실행하면 실패합니다. 이는 알려진 문제입니다.

셀 매직

여러 줄로 기록되는 매직은 앞에 이중 퍼센트 기호(%%)가 붙고 셀 매직 함수 또는 셀 매직이라고 합니다.

%%sql

이 셀 매직을 사용하면 Spark SQL 문으로 장식하지 않고도 SQL 문을 직접 실행할 수 있습니다. 또한 이 명령은 반환된 데이터 프레임에서 암시적으로 `.show()`를 직접 호출하여 출력을 표시합니다.



이 %%sql 명령은 열 출력을 20자 너비로 자동으로 잘라냅니다. 현재 이 설정은 구성할 수 없습니다. 이 제한 사항을 해결하려면 다음 전체 구문을 사용하고 show 메서드의 파라미터를 적절히 수정합니다.

```
spark.sql("""YOUR_SQL""").show(n=number, truncate=number, vertical=bool)
```

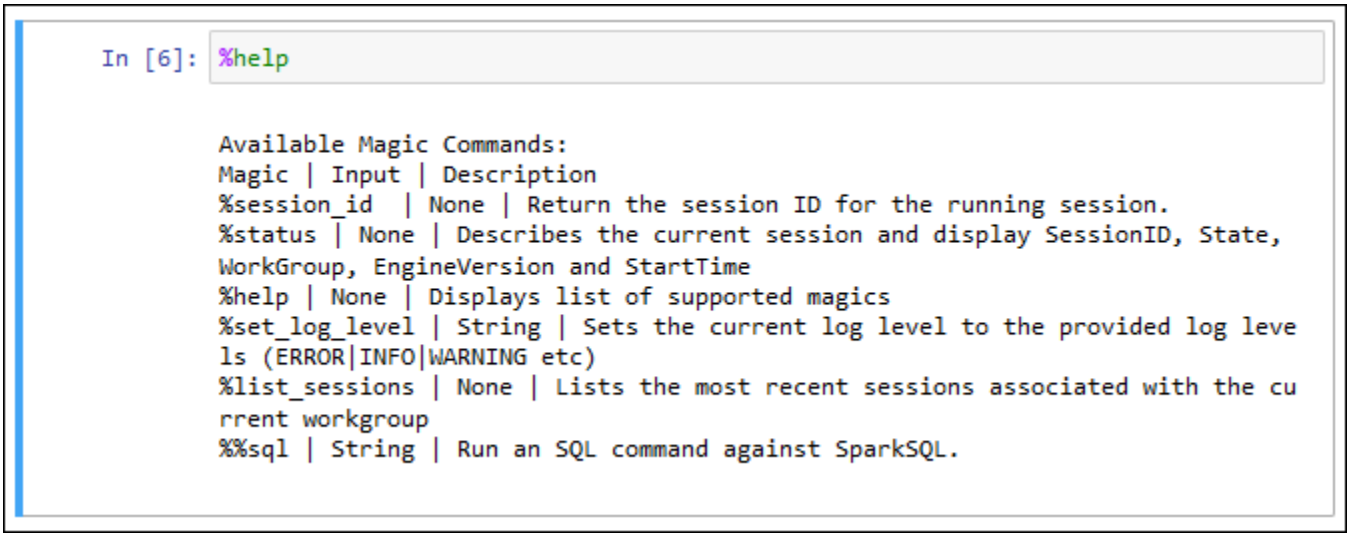
- n int, 선택 사항. 표시할 행 수입니다.
- 잘라내기 - bool 또는 int, 선택 사항 - true인 경우 20자보다 긴 문자열을 자릅니다. 1보다 큰 숫자로 설정하면 긴 문자열을 지정된 길이만큼 자르고 셀을 오른쪽 정렬합니다.
- vertical - bool, 선택 사항. true인 경우 출력 행을 세로로 인쇄합니다(열 값당 한 줄).

행 매직

한 줄로 된 매직은 앞에 퍼센트 기호(%)가 붙고 행 매직 함수 또는 행 매직이라고 합니다.

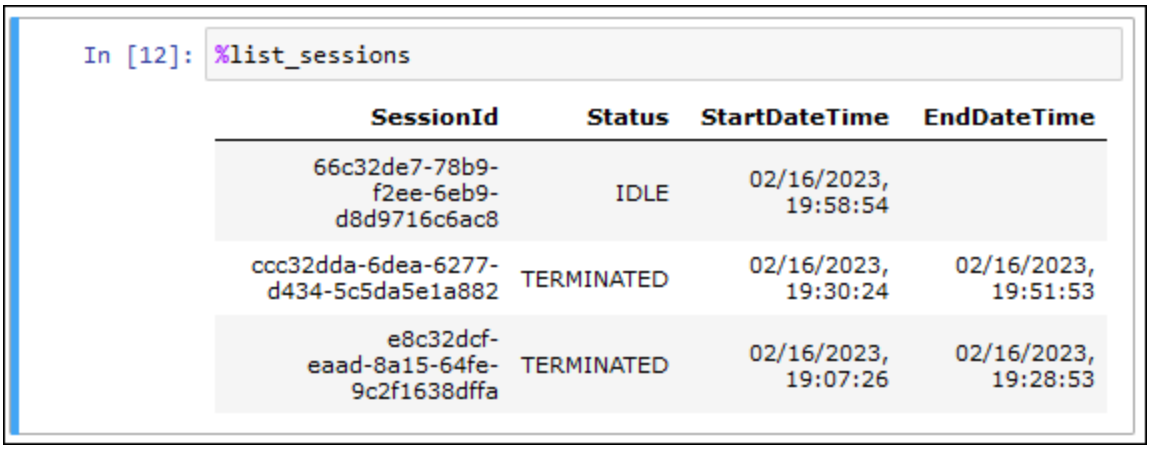
%help

사용 가능한 매직 명령에 대한 설명을 표시합니다.



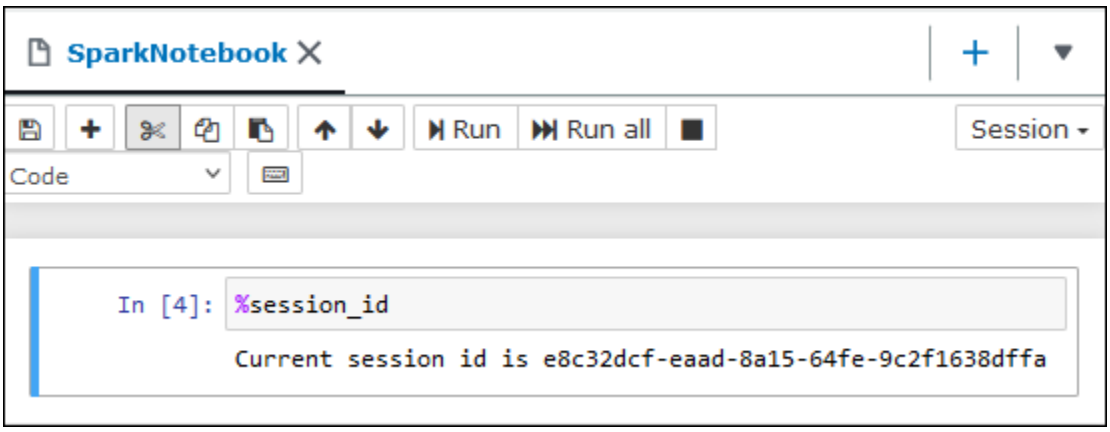
%list_sessions

노트북과 관련된 세션을 나열합니다. 각 세션의 정보에는 세션 ID, 세션 상태, 세션 시작 및 종료 날짜와 시간이 포함됩니다.



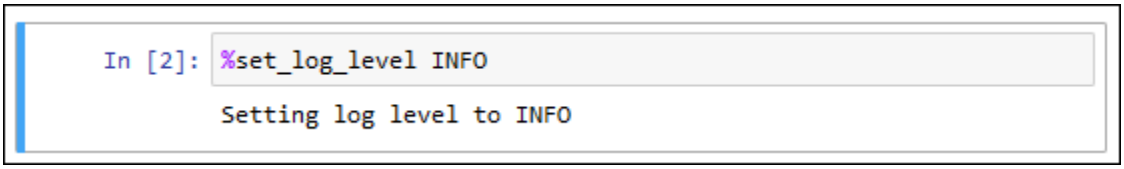
%session_id

현재 세션 ID를 검색합니다.



%set_log_level

지정된 로그 수준을 사용하도록 로거를 설정하거나 재설정합니다. 가능한 값은 DEBUG, ERROR, FATAL, INFO 및 WARN 또는 WARNING입니다. 값은 대문자여야 하며 작은따옴표 또는 큰따옴표로 묶지 않아야 합니다.



%status

현재 세션을 설명합니다. 출력에는 세션 ID, 세션 상태, 작업 그룹 이름, PySpark 엔진 버전 및 세션 시작 시간이 포함됩니다. 이 매직 명령에서는 세션 세부 정보를 검색하려면 활성 세션이 필요합니다.

다음은 상태에 대해 가능한 값입니다.

작성 중 - 리소스 획득을 포함하여 세션을 시작하는 중입니다.

작성됨 - 세션이 시작되었습니다.

유휴 - 세션에서 계산을 수락할 수 있습니다.

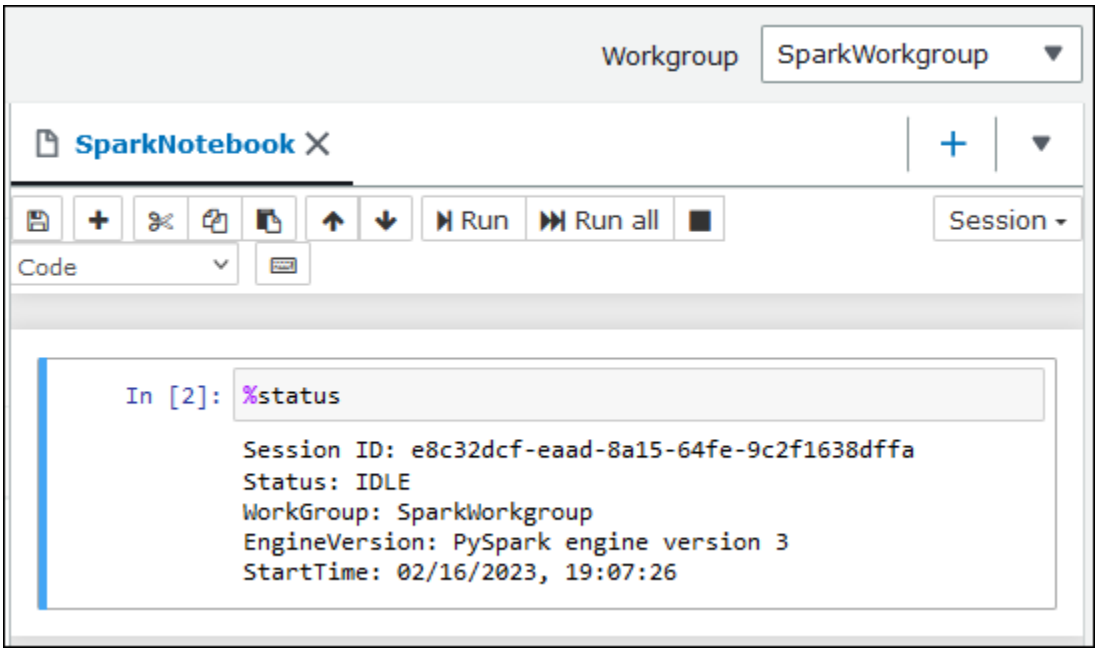
사용 중 - 세션이 다른 작업을 처리 중이며 계산을 수락할 수 없습니다.

종료 중 - 서비스가 종료되는 중입니다.

종료됨 - 세션과 해당 리소스가 더 이상 실행 중이 아닙니다.

성능 저하됨 - 세션에 정상 상태의 코디네이터가 없습니다.

실패 - 장애로 인해 세션과 해당 리소스가 더 이상 실행 중이 아닙니다.



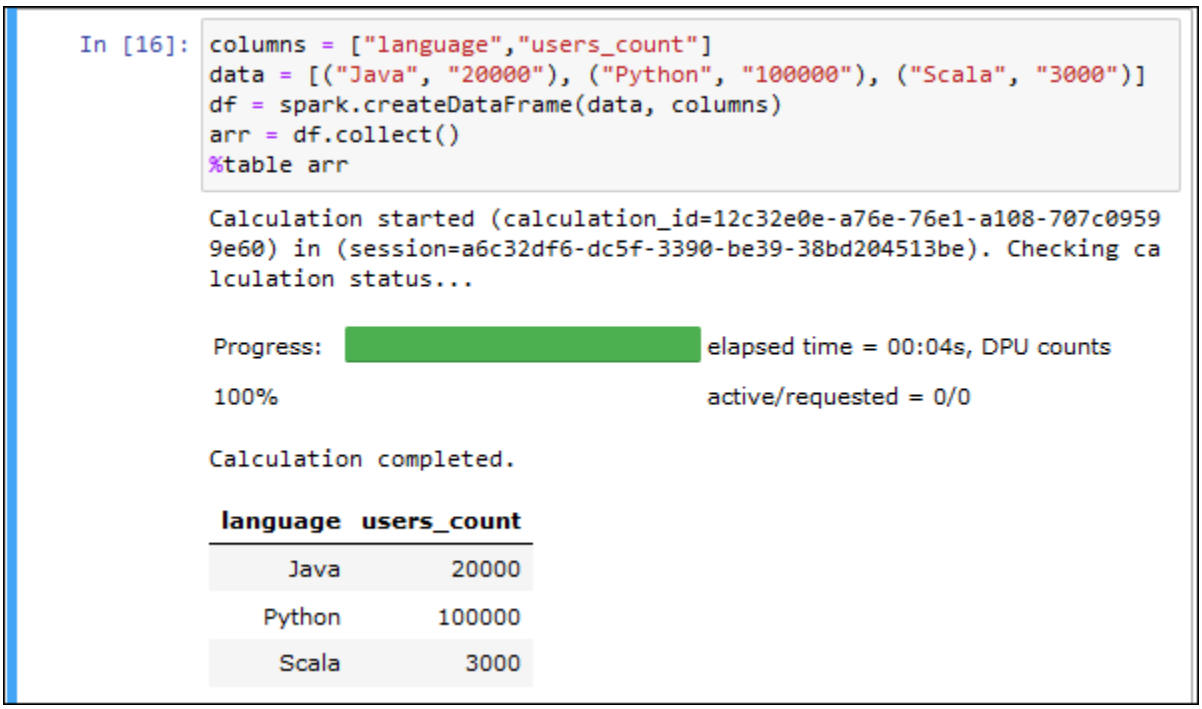
데이터 그래프 생성을 위한 매직

이 섹션의 행 매직은 그래프 생성 라이브러리와 함께 사용하거나 특정 유형의 데이터에 대한 데이터를 렌더링하는 데 특화되어 있습니다.

%table

`%table` 매직 명령을 사용하여 데이터 프레임 데이터를 테이블 형식으로 표시할 수 있습니다.

다음 예제에서는 2개 열과 3개 행의 데이터로 구성된 데이터 프레임을 생성하고 해당 데이터를 테이블 형식으로 표시합니다.



%matplot

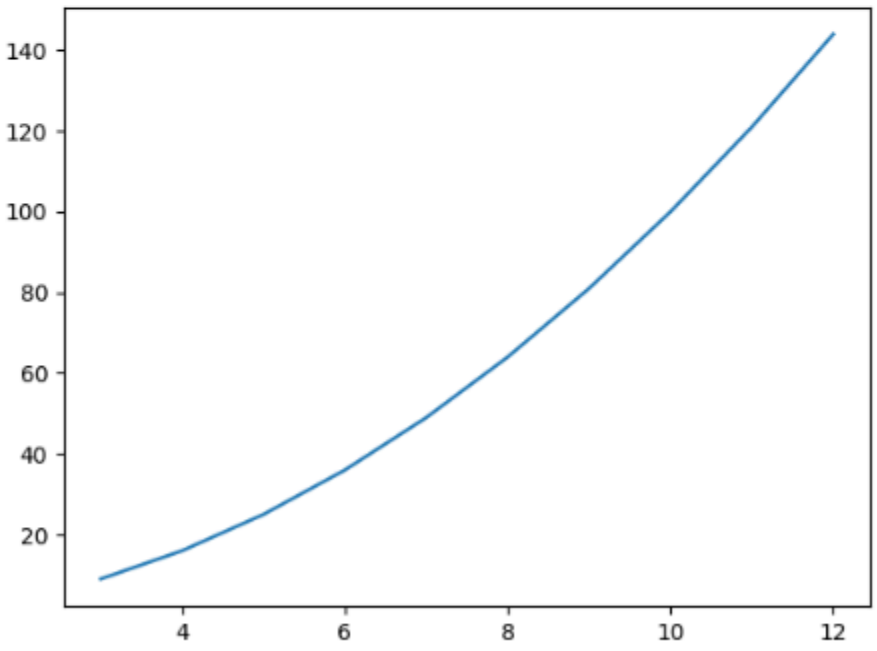
Matplotlib는 Python으로 정적, 애니메이션 및 대화형 시각화를 생성하기 위한 포괄적인 라이브러리입니다. matplotlib 라이브러리를 노트북 셀로 가져온 후 `%matplot` 매직 명령을 사용하여 그래프를 생성할 수 있습니다.

다음 예제에서는 matplotlib 라이브러리를 가져와서 x 및 y 좌표 세트를 생성한 후 `%matplot` 매직 명령을 사용하여 점의 그래프를 생성합니다.

```
import matplotlib.pyplot as plt
x=[3,4,5,6,7,8,9,10,11,12]
y= [9,16,25,36,49,64,81,100,121,144]
plt.plot(x,y)
%matplot plt
```

In [12]:
```
import matplotlib.pyplot as plt
x=[3,4,5,6,7,8,9,10,11,12]
y= [9,16,25,36,49,64,81,100,121,144]
plt.plot(x,y)
%matplot plt
```

```
Calculation started (calculation_id=5ac32e04-81b6-
9ee7-ce55-539ee2ce383e) in (session=a6c32df6-dc5f-
3390-be39-38bd204513be). Checking calculation stat
us...

Progress:                                elapsed time =

100%                                     00:02s

Calculation completed.
```




```
[<matplotlib.lines.Line2D object at 0x7f6e29e580>]
```

matplotlib 및 seaborn 라이브러리 함께 사용

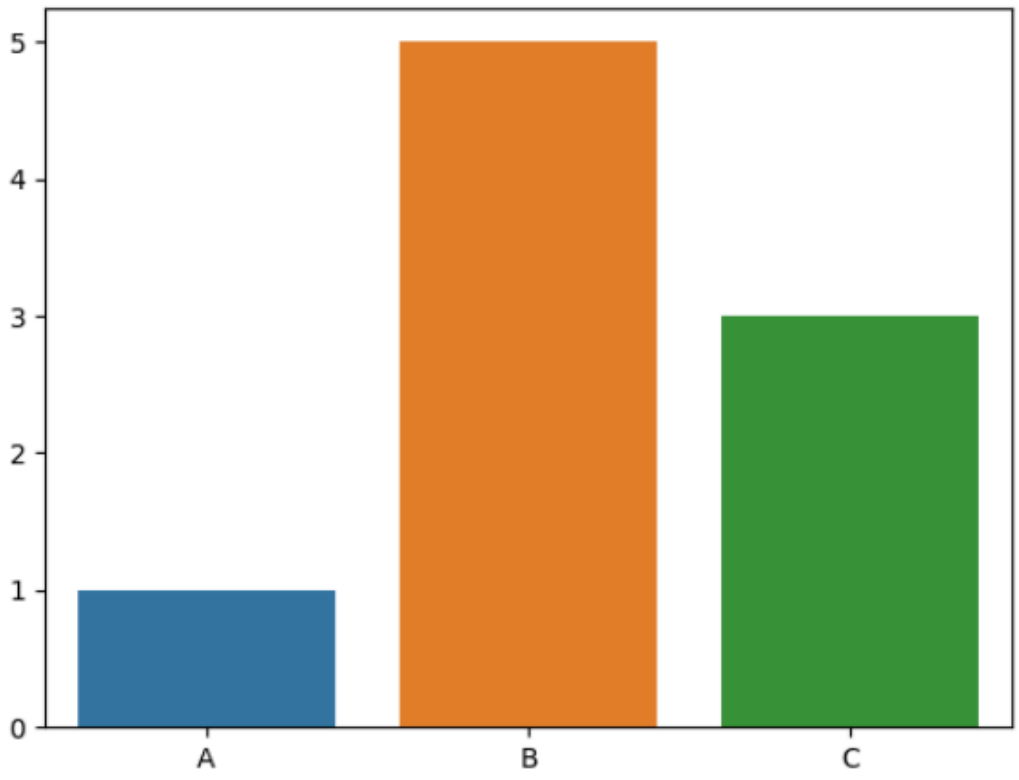
Seaborn은 Python으로 통계 그래프를 생성하기 위한 라이브러리입니다. matplotlib를 기반으로 구축되며 pandas(Python 데이터 분석) 데이터 구조와 긴밀하게 통합됩니다. `%matplot` 매직 명령을 사용하여 seaborn 데이터를 렌더링할 수도 있습니다.

다음 예제에서는 matplotlib 라이브러리와 seaborn 라이브러리를 모두 사용하여 간단한 막대 그래프를 생성합니다.

```
import matplotlib.pyplot as plt
import seaborn as sns

x = ['A', 'B', 'C']
y = [1, 5, 3]

sns.barplot(x, y)
%matplot plt
```

In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns


x = ['A', 'B', 'C']
y = [1, 5, 3]

sns.barplot(x, y)
%matplot plt
```

```
Calculation started (calculation_id=08c32e1b-233b-4a72-6571-1ae7a28a7b78)
in (session=64c32e1a-f45e-1d52-b54b-85202e2a9233). Checking calculation st
atus...

Progress: 100% elapsed time = 00:04s

Calculation completed.
```

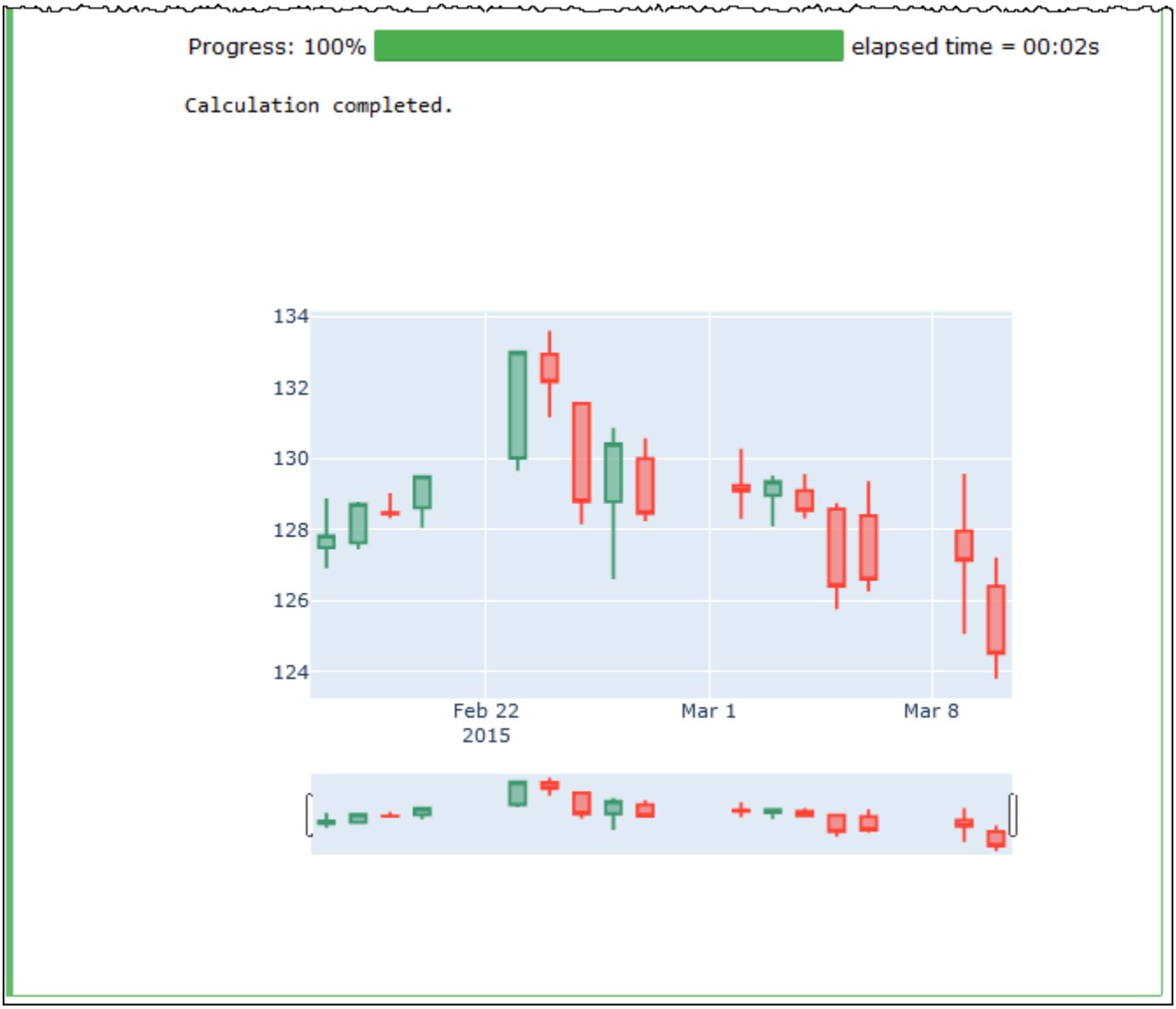


%plotly

Plotly는 대화형 그래프를 생성하는 데 사용할 수 있는 Python용 오픈 소스 그래프 생성 라이브러리입니다. `%ploty` 매직 명령을 사용하여 ploty 데이터를 렌더링합니다.

다음 예제에서는 주가 데이터에서 StringIO, plotly 및 pandas 라이브러리를 사용하여 2015년 2월과 3월의 주식 활동에 대한 그래프를 생성합니다.

```
from io import StringIO
csvString = """
Date,AAPL.Open,AAPL.High,AAPL.Low,AAPL.Close,AAPL.Volume,AAPL.Adjusted,dn,mavg,up,direction
2015-02-17,127.489998,128.880005,126.919998,127.830002,63152400,122.905254,106.7410523,117.9276
2015-02-18,127.629997,128.779999,127.449997,128.720001,44891700,123.760965,107.842423,118.94033
2015-02-19,128.479996,129.029999,128.330002,128.449997,37362400,123.501363,108.8942449,119.8891
2015-02-20,128.619995,129.5,128.050003,129.5,48948400,124.510914,109.7854494,120.7635001,131.74
2015-02-23,130.020004,133,129.660004,133,70974100,127.876074,110.3725162,121.7201668,133.067817
2015-02-24,132.940002,133.600006,131.169998,132.169998,69228100,127.078049,111.0948689,122.6648
2015-02-25,131.559998,131.600006,128.149994,128.789993,74711700,123.828261,113.2119183,123.6296
2015-02-26,128.789993,130.869995,126.610001,130.419998,91287500,125.395469,114.1652991,124.2823
2015-02-27,130,130.570007,128.240005,128.460007,62014800,123.510987,114.9668484,124.8426669,134
2015-03-02,129.25,130.279999,128.300003,129.089996,48096700,124.116706,115.8770904,125.4036668,
2015-03-03,128.960007,129.520004,128.089996,129.360001,37816300,124.376308,116.9535132,125.9551
2015-03-04,129.100006,129.559998,128.320007,128.539993,31666300,123.587892,118.0874253,126.4730
2015-03-05,128.580002,128.75,125.760002,126.410004,56517100,121.539962,119.1048311,126.848667,1
2015-03-06,128.399994,129.369995,126.260002,126.599998,72842100,121.722637,120.190797,127.22883
2015-03-09,127.959999,129.570007,125.059998,127.139999,88528500,122.241834,121.6289771,127.6311
2015-03-10,126.410004,127.220001,123.800003,124.510002,68856600,119.71316,123.1164763,127.92350
"""
csvStringIO = StringIO(csvString)

from io import StringIO
import plotly.graph_objects as go
import pandas as pd
from datetime import datetime
df = pd.read_csv(csvStringIO)
fig = go.Figure(data=[go.Candlestick(x=df['Date'],
open=df['AAPL.Open'],
high=df['AAPL.High'],
low=df['AAPL.Low'],
close=df['AAPL.Close'])])
%plotly fig
```



노트북 파일 관리

노트북 탐색기를 사용하여 노트북 생성 및 열기 이외에 노트북 이름 바꾸기, 삭제, 내보내기, 가져오기를 수행하거나 노트북에 대한 세션 기록을 볼 수도 있습니다.

노트북의 이름을 바꾸려면

1. 이름을 바꾸려는 노트북의 활성 세션을 종료합니다. 노트북의 이름을 바꾸려면 먼저 노트북의 활성 세션을 종료해야 합니다.
2. Notebook explorer(노트북 탐색기)를 엽니다.
3. Notebooks(노트북) 목록에서 이름을 바꾸려는 노트북에 대한 옵션 버튼을 선택합니다.
4. Actions(작업) 메뉴에서 Rename(이름 바꾸기)을 선택합니다.

5. Rename notebook(노트북 이름 바꾸기) 프롬프트에서 새 이름을 입력한 다음 Save(저장)를 선택합니다. 새 노트북 이름이 노트북 목록에 표시됩니다.

노트북을 삭제하려면

1. 삭제하려는 노트북의 활성 세션을 종료합니다. 노트북을 삭제하려면 먼저 노트북의 활성 세션을 종료해야 합니다.
2. Notebook explorer(노트북 탐색기)를 엽니다.
3. Notebooks(노트북) 목록에서 삭제하려는 노트북에 대한 옵션 버튼을 선택합니다.
4. 작업 메뉴에서 삭제를 선택합니다.
5. Delete notebook?(노트북 삭제) 프롬프트에서 노트북의 이름을 입력한 다음 Delete(삭제)를 선택하여 삭제를 확인합니다. 노트북 이름이 노트북 목록에서 제거됩니다.

노트북을 내보내려면

1. Notebook explorer(노트북 탐색기)를 엽니다.
2. Notebooks(노트북) 목록에서 내보내려는 노트북에 대한 옵션 버튼을 선택합니다.
3. Actions(작업) 메뉴에서 Export file(파일 내보내기)을 선택합니다.

노트북을 가져오려면

1. Notebook explorer(노트북 탐색기)를 엽니다.
2. Import file(파일 가져오기)을 선택합니다.
3. 로컬 컴퓨터에서 가져오려는 파일의 위치를 찾은 다음 Open(열기)을 선택합니다. 가져온 노트북이 노트북 목록에 표시됩니다.

노트북의 세션 기록을 보려면

1. Notebook explorer(노트북 탐색기)를 엽니다.
2. Notebooks(노트북) 목록에서 세션 기록을 보려는 노트북에 대한 옵션 버튼을 선택합니다.
3. Actions(작업) 메뉴에서 Session history(세션 기록)를 선택합니다.
4. History(기록) 탭에서 Session ID(세션 ID)를 선택하여 세션 및 해당 계산에 대한 정보를 볼 수 있습니다.

Amazon Athena for Apache Spark에서 비 Hive 테이블 형식 사용

Athena for Spark에서 세션 및 노트북을 사용하는 경우 Apache Hive 테이블 외에도 Linux Foundation Delta Lake, Apache Hudi, Apache Iceberg 테이블을 사용할 수 있습니다.

고려 사항 및 제한

Athena for Spark에서 Apache Hive 이외의 테이블 형식을 사용하는 경우 다음 사항을 고려합니다.

- Apache Hive 이외에도 노트북당 하나의 테이블 형식만 지원됩니다. Athena for Spark에서 여러 테이블 형식을 사용하려면 각 테이블 형식에 대해 별도의 노트북을 생성합니다. Athena for Spark에서 노트북을 생성하는 방법에 대한 자세한 내용은 사용자 노트북 생성 섹션을 참조하세요.
- Delta Lake, Hudi 및 Iceberg 테이블 형식은 AWS Glue를 사용하여 Athena for Spark에서 메타스토어로 테스트되었습니다. 다른 메타스토어를 사용할 수도 있지만 현재 이러한 사용은 지원되지 않습니다.
- 추가 테이블 형식을 사용하려면 Athena 콘솔 및 이 설명서에 나와 있는 대로 기본 `spark_catalog` 속성을 재정의합니다. 이러한 비 Hive 카탈로그는 고유한 테이블 형식 외에도 Hive 테이블을 읽을 수 있습니다.

테이블 버전

다음 테이블에는 Amazon Athena for Apache Spark에서 지원되는 비 Hive 테이블 버전이 나와 있습니다.

테이블 형식	지원되는 버전
Apache Iceberg	1.2.1
Apache Hudi	0.13
Linux Foundation Delta Lake	2.0.2

Athena for Spark에서 이러한 테이블 형식 `.jar` 파일과 해당 종속성이 Spark 드라이버 및 실행기의 클래스 경로에 로드됩니다.

주제

- Apache Iceberg

- Apache Hudi
- Linux Foundation Delta Lake

Apache Iceberg

Apache Iceberg는 Amazon Simple Storage Service(S3)의 대형 데이터 세트를 위한 오픈 테이블 형식입니다. 이 테이블 형식은 대형 테이블, 원자성 커밋, 동시 쓰기, SQL 호환 테이블 진화 등에서 빠른 쿼리 성능을 제공합니다.

Athena for Spark에서 Apache Iceberg 테이블을 사용하려면 다음 Spark 속성을 구성합니다. Apache Iceberg를 테이블 형식으로 선택하면 Athena for Spark 콘솔에서 이러한 속성이 기본적으로 구성됩니다. 관련 단계는 세션 세부 정보 편집 또는 사용자 노트북 생성 섹션을 참조하세요.

```
"spark.sql.catalog.spark_catalog": "org.apache.iceberg.spark.SparkSessionCatalog",
"spark.sql.catalog.spark_catalog.catalog-impl":
 "org.apache.iceberg.aws.glue.GlueCatalog",
"spark.sql.catalog.spark_catalog.io-impl": "org.apache.iceberg.aws.s3.S3FileIO",
"spark.sql.extensions":
 "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions"
```

다음 절차는 Athena for Stork 노트북에서 Apache Iceberg 테이블을 사용하는 방법을 보여줍니다. 노트북의 새 셀에서 각 단계를 실행합니다.

Athena for Spark에서 Apache Iceberg 테이블을 사용하려면

1. 노트북에서 사용할 상수를 정의합니다.

   ```
   DB_NAME = "NEW_DB_NAME"
   TABLE_NAME = "NEW_TABLE_NAME"
   TABLE_S3_LOCATION = "s3://DOC-EXAMPLE-BUCKET"
   ```

2. Apache Spark DataFrame을 생성합니다.

   ```
   columns = ["language","users_count"]
   data = [("Golang", 3000)]
   df = spark.createDataFrame(data, columns)
   ```

3. 데이터베이스를 생성합니다.

   ```
   spark.sql("CREATE DATABASE {} LOCATION '{}'".format(DB_NAME, TABLE_S3_LOCATION))
   ```

4. 빈 Apache Iceberg 테이블을 생성합니다.

```
spark.sql("""
CREATE TABLE {}.{} (
language string,
users_count int
) USING ICEBERG
""".format(DB_NAME, TABLE_NAME))
```

5. 테이블에 데이터 행을 삽입합니다.

```
spark.sql("""INSERT INTO {}.{} VALUES ('Golang',
 3000)""".format(DB_NAME, TABLE_NAME))
```

6. 새 테이블을 쿼리할 수 있는지 확인합니다.

```
spark.sql("SELECT * FROM {}.{}".format(DB_NAME, TABLE_NAME)).show()
```

Spark DataFrames 및 Iceberg 테이블을 사용하는 방법에 대한 자세한 내용 및 예제는 Apache Iceberg 설명서의 Spark Queries를 참조하세요.

Apache Hudi

Apache Hudi는 증분 데이터 처리를 간소화하는 오픈 소스 데이터 관리 프레임워크입니다. 레코드 수준의 삽입, 업데이트, 업서트 및 삭제 작업이 더 높은 정밀도로 처리되며, 이때 오버헤드가 줄어듭니다.

Athena for Spark에서 Apache Hudi 테이블을 사용하려면 다음 Spark 속성을 구성합니다. Apache Hudi를 테이블 형식으로 선택하면 Athena for Spark 콘솔에서 이러한 속성이 기본적으로 구성됩니다. 관련 단계는 세션 세부 정보 편집 또는 사용자 노트북 생성 섹션을 참조하세요.

```
"spark.sql.catalog.spark_catalog": "org.apache.spark.sql.hudi.catalog.HoodieCatalog",
"spark.serializer": "org.apache.spark.serializer.KryoSerializer",
"spark.sql.extensions": "org.apache.spark.sql.hudi.HoodieSparkSessionExtension"
```

다음 절차는 Athena for Stork 노트북에서 Apache Hudi 테이블을 사용하는 방법을 보여줍니다. 노트북의 새 셀에서 각 단계를 실행합니다.

Athena for Spark에서 Apache Hudi 테이블을 사용하려면

1. 노트북에서 사용할 상수를 정의합니다.

```
DB_NAME = "NEW_DB_NAME"
TABLE_NAME = "NEW_TABLE_NAME"
TABLE_S3_LOCATION = "s3://DOC-EXAMPLE-BUCKET"
```

2. Apache Spark DataFrame을 생성합니다.

```
columns = ["language","users_count"]
data = [("Golang", 3000)]
df = spark.createDataFrame(data, columns)
```

3. 데이터베이스를 생성합니다.

```
spark.sql("CREATE DATABASE {} LOCATION '{}'".format(DB_NAME, TABLE_S3_LOCATION))
```

4. 빈 Apache Hudi 테이블을 생성합니다.

```
spark.sql("""
CREATE TABLE {}.{} (
language string,
users_count int
) USING HUDI
TBLPROPERTIES (
primaryKey = 'language',
type = 'mor'
);
""".format(DB_NAME, TABLE_NAME))
```

5. 테이블에 데이터 행을 삽입합니다.

```
spark.sql("""INSERT INTO {}.{} VALUES ('Golang',
 3000)""".format(DB_NAME,TABLE_NAME))
```

6. 새 테이블을 쿼리할 수 있는지 확인합니다.

```
spark.sql("SELECT * FROM {}.{}".format(DB_NAME, TABLE_NAME)).show()
```

Linux Foundation Delta Lake

Linux Foundation Delta Lake는 빅 데이터 분석에 사용할 수 있는 테이블 형식입니다. Athena for Spark를 사용하여 Amazon S3에 저장된 Delta Lake 테이블을 직접 읽을 수 있습니다.

Athena for Spark에서 Delta Lake 테이블을 사용하려면 다음 Spark 속성을 구성합니다. Delta Lake를 테이블 형식으로 선택하면 Athena for Spark 콘솔에서 이러한 속성이 기본적으로 구성됩니다. 관련 단계는 세션 세부 정보 편집 또는 사용자 노트북 생성 섹션을 참조하세요.

```
"spark.sql.catalog.spark_catalog" : "org.apache.spark.sql.delta.catalog.DeltaCatalog",
"spark.sql.extensions" : "io.delta.sql.DeltaSparkSessionExtension"
```

다음 절차는 Athena for Stork 노트북에서 Delta Lake 테이블을 사용하는 방법을 보여줍니다. 노트북의 새 셀에서 각 단계를 실행합니다.

Athena for Spark에서 Delta Lake 테이블을 사용하려면

1. 노트북에서 사용할 상수를 정의합니다.

```
DB_NAME = "NEW_DB_NAME"
TABLE_NAME = "NEW_TABLE_NAME"
TABLE_S3_LOCATION = "s3://DOC-EXAMPLE-BUCKET"
```

2. Apache Spark DataFrame을 생성합니다.

```
columns = ["language","users_count"]
data = [("Golang", 3000)]
df = spark.createDataFrame(data, columns)
```

3. 데이터베이스를 생성합니다.

```
spark.sql("CREATE DATABASE {} LOCATION '{}'".format(DB_NAME, TABLE_S3_LOCATION))
```

4. 빈 Delta Lake 테이블을 생성합니다.

```
spark.sql("""
CREATE TABLE {}.{} (
  language string,
  users_count int
) USING DELTA
""".format(DB_NAME, TABLE_NAME))
```

5. 테이블에 데이터 행을 삽입합니다.

```
spark.sql("""INSERT INTO {}.{} VALUES ('Golang',
 3000)""".format(DB_NAME, TABLE_NAME))
```

6. 새 테이블을 쿼리할 수 있는지 확인합니다.

```
spark.sql("SELECT * FROM {}.{}".format(DB_NAME, TABLE_NAME)).show()
```

Apache Spark용 Amazon Athena의 Python 라이브러리 지원

이 페이지에서는 Apache Spark용 Amazon Athena에서 사용되는 런타임, 라이브러리 및 패키지에 사용되는 용어와 그에 따른 수명 주기 관리에 대해 설명합니다.

정의

- Amazon Athena for Apache Spark(Apache Spark용 Amazon Athena)는 오픈 소스 Apache Spark의 사용자 지정 버전입니다. 현재 버전을 보려면 노트북 셀에서 `print(f'{spark.version}')` 명령을 실행합니다.
- Athena runtime(Athena 런타임)은 코드가 실행되는 환경입니다. 환경에는 Python 인터프리터와 PySpark 라이브러리가 포함됩니다.
- external library or package(외부 라이브러리 또는 패키지)는 Athena 런타임에는 포함되지 않지만 Athena for Spark 작업에 포함될 수 있는 Java, Scala JAR 또는 Python 라이브러리입니다. 외부 패키지는 Amazon 또는 사용자가 만들 수 있습니다.
- convenience package(편의 패키지)는 Athena에서 선택된 외부 패키지 모음으로, Spark 애플리케이션에 포함하도록 선택할 수 있습니다.
- bundle(번들)은 Athena 런타임과 편의 패키지를 결합합니다.
- user library(사용자 라이브러리)는 Athena for Spark 작업에 명시적으로 추가되는 외부 라이브러리 또는 패키지입니다.
  - 사용자 라이브러리는 편의 패키지에 포함되지 않는 외부 패키지입니다. 일부 `.py` 파일을 작성하고 압축한 다음 `.zip` 파일을 애플리케이션에 추가할 때처럼 사용자 라이브러리를 로드하고 설치해야 합니다.
- Athena for Spark application(Athena for Spark 애플리케이션)은 Athena for Spark에 제출되는 작업 또는 쿼리입니다.

수명 주기 관리

런타임 버전 관리 및 지원 중단

Athena 런타임의 주요 구성 요소는 Python 인터프리터입니다. Python은 진화하는 언어이므로 새 버전이 정기적으로 릴리스되고 이전 버전에 대한 지원이 제거됩니다. 더 이상 사용되지 않는 Python 인터프리터 버전으로 프로그램을 실행하는 대신 가능하면 최신 Athena 런타임을 사용하는 것이 좋습니다.

Athena 런타임 지원 중단 일정은 다음과 같습니다.

1. Athena는 새 런타임을 제공한 이후에 6개월 동안 이전 런타임을 계속 지원합니다. 이 기간 동안 이전 런타임에 대한 보안 패치 및 업데이트를 적용합니다.
2. 6개월 후에는 이전 런타임에 대한 지원을 종료합니다. 이 경우 이전 런타임에 대한 보안 패치 및 기타 업데이트를 더 이상 적용하지 않습니다. 이전 런타임을 사용하는 Spark 애플리케이션은 더 이상 기술 지원을 받을 수 없습니다.
3. 12개월 후에는 이전 런타임을 사용하는 작업 그룹에서 더 이상 Spark 애플리케이션을 업데이트하거나 편집할 수 없습니다. 이 기간이 끝나기 전에 Spark 애플리케이션을 업데이트하는 것이 좋습니다. 이 기간이 끝난 후에도 기존 노트북을 계속 실행할 수 있지만, 노트북에서 이전 런타임을 계속 사용할 경우 해당 효과에 대한 경고가 기록됩니다.
4. 18개월 후에는 이전 런타임을 사용하여 작업 그룹에서 더 이상 작업을 실행할 수 없습니다.

편의 패키지 버전 관리 및 지원 중단

편의 패키지의 내용은 시간이 지남에 따라 변경됩니다. Athena 때때로 이러한 편의 패키지를 추가, 제거 또는 업그레이드합니다.

Athena는 편의 패키지에 대해 다음 지침을 사용합니다.

- 편의 패키지에는 1, 2, 3과 같은 간단한 버전 관리 체계가 있습니다.
- 각 편의 패키지 버전에는 특정 버전의 외부 패키지가 포함되어 있습니다. Athena에서 편의 패키지를 생성한 후에는 편의 패키지의 외부 패키지 세트와 해당 버전이 변경되지 않습니다.
- Athena는 새 외부 패키지를 포함하거나, 외부 패키지를 제거하거나, 하나 이상의 외부 패키지 버전을 업그레이드할 때 새 편의 패키지 버전을 생성합니다.

Athena는 패키지에서 사용되는 Athena 런타임 지원을 중단할 때 편의 패키지 지원을 중단합니다. Athena는 지원하는 번들 수를 제한하기 위해 패키지 지원을 더 빨리 중단할 수 있습니다.

편의 패키지 지원 중단 일정은 Athena 런타임 지원 중단 일정을 따릅니다.

사전 설치된 Python 라이브러리 목록

사전 설치된 Python 라이브러리에는 다음이 포함됩니다.

```
boto3==1.24.31
botocore==1.27.31
certifi==2022.6.15
charset-normalizer==2.1.0
cycler==0.11.0
cython==0.29.30
docutils==0.19
fonttools==4.34.4
idna==3.3
jmespath==1.0.1
joblib==1.1.0
kiwisolver==1.4.4
matplotlib==3.5.2
mpmath==1.2.1
numpy==1.23.1
packaging==21.3
pandas==1.4.3
patsy==0.5.2
pillow==9.2.0
plotly==5.9.0
pmdarima==1.8.5
pyathena==2.9.6
pyparsing==3.0.9
python-dateutil==2.8.2
pytz==2022.1
requests==2.28.1
s3transfer==0.6.0
scikit-learn==1.1.1
scipy==1.8.1
seaborn==0.11.2
six==1.16.0
statsmodels==0.13.2
sympy==1.10.1
tenacity==8.0.1
threadpoolctl==3.1.0
urllib3==1.26.10
pyarrow==9.0.0
```

참고

- MLlib(Apache Spark 기계 학습 라이브러리) 및 `pyspark.ml` 패키지는 지원되지 않습니다.
- 현재 `pip install`은 Athena for Spark 세션에서 지원되지 않습니다.

Python 라이브러리를 Amazon Athena for Apache Spark로 가져오는 방법에 대한 자세한 내용은 Apache Spark용 Amazon Athena로 파일 및 Python 라이브러리 가져오기 섹션을 참조하세요.

Apache Spark용 Amazon Athena로 파일 및 Python 라이브러리 가져오기

이 문서에서는 파일 및 Python 라이브러리를 Apache Spark용 Amazon Athena로 가져오는 방법에 대한 예제를 제공합니다.

고려 사항 및 제한

- Python 버전 – 현재 Athena for Spark는 Python 버전 3.9.16을 사용합니다. Python 패키지는 Python 마이너 버전에 민감합니다.
- Athena for Spark 아키텍처 - Athena for Spark는 ARM64 아키텍처 기반 Amazon Linux 2를 사용합니다. 일부 Python 라이브러리는 이 아키텍처에 대한 바이너리를 배포하지 않습니다.
- 바이너리 공유 객체(SO) – SparkContext addPyFile 메서드는 바이너리 공유 객체를 탐지하지 않으므로 Athena for Spark에서 공유 객체에 종속된 Python 패키지를 추가할 때 사용할 수 없습니다.
- Resilient Distributed Dataset(RDD) – RDD는 지원되지 않습니다.
- Dataframe.foreach – PySpark DataFrame.foreach 메서드는 지원되지 않습니다.

예제

예제에서는 다음과 같은 규칙을 사용합니다.

- 자리 표시자 Amazon S3 위치 `s3://DOC-EXAMPLE-BUCKET`. 사용자의 S3 버킷으로 대체합니다.
- Unix 셸에서 실행되는 모든 코드 블록은 *`directory_name`* $으로 표시됩니다. 예를 들어, `ls` 디렉터리의 `/tmp` 명령과 해당 출력은 다음과 같이 표시됩니다.

```
/tmp $ ls
```

출력

```
file1 file2
```

- 로컬 임시 디렉터리에 파일을 쓴 후 노트북에 파일 추가
- Amazon S3에서 파일 가져오기
- Python 파일 추가 및 UDF 등록
- Python .zip 파일 가져오기
- 두 버전의 Python 라이브러리를 별도의 모듈로 가져오기
- PyPI에서 Python .zip 파일 가져오기
- PyPI에서 종속성이 있는 Python .zip 파일 가져오기

계산에 사용할 텍스트 파일 가져오기

이 단원의 예제에서는 Athena for Spark의 노트북에서 계산에 사용할 텍스트 파일을 가져오는 방법을 보여줍니다.

로컬 임시 디렉터리에 파일을 쓴 후 노트북에 파일 추가

다음 예제에서는 로컬 임시 디렉터리에 파일을 쓴 후 노트북에 추가하고 테스트하는 방법을 보여줍니다.

```
import os
from pyspark import SparkFiles
tempdir = '/tmp/'
path = os.path.join(tempdir, "test.txt")
with open(path, "w") as testFile:
    _ = testFile.write("5")
sc.addFile(path)

def func(iterator):
    with open(SparkFiles.get("test.txt")) as testFile:
        fileVal = int(testFile.readline())
        return [x * fileVal for x in iterator]

#Test the file
from pyspark.sql.functions import udf
from pyspark.sql.functions import col

udf_with_import = udf(func)
```

```
df = spark.createDataFrame([(1, "a"), (2, "b")])
df.withColumn("col", udf_with_import(col('_2'))).show()
```

출력

```
Calculation completed.
+---+---+-------+
| _1| _2|    col|
+---+---+-------+
|  1|  a|[aaaaa]|
|  2|  b|[bbbbb]|
+---+---+-------+
```

Amazon S3에서 파일 가져오기

다음 예제에서는 Amazon S3에서 노트북으로 파일을 가져와서 테스트하는 방법을 보여줍니다.

Amazon S3에서 노트북으로 파일을 가져오려면

1. 값 5가 포함된 한 줄로 구성된 `test.txt`라는 파일을 생성합니다.
2. Amazon S3에서 버킷에 파일을 추가합니다. 이 예제에서는 `s3://DOC-EXAMPLE-BUCKET` 위치를 사용합니다.
3. 다음 코드를 사용하여 파일을 노트북으로 가져온 다음 테스트합니다.

```
from pyspark import SparkFiles
sc.addFile('s3://DOC-EXAMPLE-BUCKET/test.txt')

def func(iterator):
   with open(SparkFiles.get("test.txt")) as testFile:
       fileVal = int(testFile.readline())
       return [x * fileVal for x in iterator]

#Test the file
from pyspark.sql.functions import udf
from pyspark.sql.functions import col

udf_with_import = udf(func)
df = spark.createDataFrame([(1, "a"), (2, "b")])
df.withColumn("col", udf_with_import(col('_2'))).show()
```

출력

```
Calculation completed.
+---+---+-------+
| _1| _2|    col|
+---+---+-------+
|  1|  a|[aaaaa]|
|  2|  b|[bbbbb]|
+---+---+-------+
```

Python 파일 추가

이 단원의 예제에서는 Python 파일 및 라이브러리를 Athena의 Spark 노트북에 추가하는 방법을 보여줍니다.

Python 파일 추가 및 UDF 등록

다음 예제에서는 Amazon S3의 Python 파일을 노트북에 추가하고 UDF를 등록하는 방법을 보여줍니다.

Python 파일을 노트북에 추가하고 UDF를 등록하려면

1. 사용자의 Amazon S3 위치를 사용하여 다음 콘텐츠가 포함된 `s3://DOC-EXAMPLE-BUCKET/file1.py` 파일을 생성합니다.

```
def xyz(input):
    return 'xyz  - udf ' + str(input);
```

2. 동일한 S3 위치를 사용하여 다음 콘텐츠가 포함된 `s3://DOC-EXAMPLE-BUCKET/file2.py` 파일을 생성합니다.

```
from file1 import xyz
def uvw(input):
    return 'uvw -> ' + xyz(input);
```

3. Athena for Spark 노트북에서 다음 명령을 실행합니다.

```
sc.addPyFile('s3://DOC-EXAMPLE-BUCKET/file1.py')
sc.addPyFile('s3://DOC-EXAMPLE-BUCKET/file2.py')

def func(iterator):
    from file2 import uvw
```

```
    return [uvw(x) for x in iterator]

from pyspark.sql.functions import udf
from pyspark.sql.functions import col

udf_with_import = udf(func)

df = spark.createDataFrame([(1, "a"), (2, "b")])

df.withColumn("col", udf_with_import(col('_2'))).show(10)
```

출력

```
Calculation started (calculation_id=1ec09e01-3dec-a096-00ea-57289cdb8ce7) in
 (session=c8c09e00-6f20-41e5-98bd-4024913d6cee). Checking calculation status...
Calculation completed.
+---+---+--------------------+
| _1| _2|                 col|
+---+---+--------------------+
| 1 |  a|[uvw -> xyz - ud... |
| 2 |  b|[uvw -> xyz - ud... |
+---+---+--------------------+
```

Python .zip 파일 가져오기

Python `addPyFile` 및 `import` 메서드를 사용하여 Python .zip 파일을 노트북으로 가져올 수 있습니다.

> **Note**
>
> Athena Spark로 가져오는 `.zip` 파일에는 Python 패키지만 포함될 수 있습니다. 예를 들어 C 기반 파일이 있는 패키지의 포함은 지원되지 않습니다.

Python **`.zip`** 파일을 노트북으로 가져오려면

1. 로컬 컴퓨터의 데스크톱 디렉터리(예: `\tmp`)에 `moduletest` 디렉터리를 생성합니다.
2. `moduletest` 디렉터리에 다음 콘텐츠로 `hello.py`라는 파일을 생성합니다.

```
def hi(input):
```

```
    return 'hi ' + str(input);
```

3. 동일한 디렉터리에서 이름이 `__init__.py`인 빈 파일을 추가합니다.

   이제 디렉터리 콘텐츠가 다음과 같이 나열됩니다.

```
/tmp $ ls moduletest
__init__.py       hello.py
```

4. `zip` 명령을 사용하여 두 모듈 파일을 `moduletest.zip` 파일에 배치합니다.

```
moduletest $ zip -r9 ../moduletest.zip *
```

5. Amazon S3의 버킷에 `.zip` 파일을 업로드합니다.
6. 다음 코드를 사용하여 Python `.zip` 파일을 노트북으로 가져옵니다.

```
sc.addPyFile('s3://DOC-EXAMPLE-BUCKET/moduletest.zip')

from moduletest.hello import hi

from pyspark.sql.functions import udf
from pyspark.sql.functions import col

hi_udf = udf(hi)

df = spark.createDataFrame([(1, "a"), (2, "b")])

df.withColumn("col", hi_udf(col('_2'))).show()
```

   출력

```
Calculation started (calculation_id=6ec09e8c-6fe0-4547-5f1b-6b01adb2242c) in
 (session=dcc09e8c-3f80-9cdc-bfc5-7effa1686b76). Checking calculation status...
Calculation completed.
+---+---+----+
| _1| _2| col|
+---+---+----+
|  1|  a|hi a|
|  2|  b|hi b|
+---+---+----+
```

두 버전의 Python 라이브러리를 별도의 모듈로 가져오기

다음 코드 예제에서는 Amazon S3의 한 위치에서 두 가지 버전의 Python 라이브러리를 두 개별 모듈로 추가하고 가져오는 방법을 보여줍니다. 이 코드는 S3에서 각 라이브러리 파일을 추가하고 가져온 다음 라이브러리 버전을 인쇄하여 가져오기를 확인합니다.

```
sc.addPyFile('s3://DOC-EXAMPLE-BUCKET/python-third-party-libs-test/
simplejson_v3_15.zip')
sc.addPyFile('s3://DOC-EXAMPLE-BUCKET/python-third-party-libs-test/
simplejson_v3_17_6.zip')

import simplejson_v3_15
print(simplejson_v3_15.__version__)
```

출력

```
3.15.0
```

```
import simplejson_v3_17_6
print(simplejson_v3_17_6.__version__)
```

출력

```
3.17.6
```

PyPI에서 Python .zip 파일 가져오기

이 예제에서는 `pip` 명령을 사용하여 Python 패키지 인덱스(PyPI)에서 bpabel/piglatin 프로젝트의 Python .zip 파일을 다운로드합니다.

PyPI에서 Python .zip 파일을 가져오려면

1. 로컬 데스크톱에서 다음 명령을 사용하여 `testpiglatin` 디렉터리를 만들고 가상 환경을 생성합니다.

   ```
   /tmp $ mkdir testpiglatin
   /tmp $ cd testpiglatin
   testpiglatin $ virtualenv .
   ```

   출력

```
created virtual environment CPython3.9.6.final.0-64 in 410ms
creator CPython3Posix(dest=/private/tmp/testpiglatin, clear=False,
 no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle,
 via=copy, app_data_dir=/Users/user1/Library/Application Support/virtualenv)
added seed packages: pip==22.0.4, setuptools==62.1.0, wheel==0.37.1
activators
 BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,PythonAct
```

2. 프로젝트를 보관할 unpacked 하위 디렉터리를 생성합니다.

```
testpiglatin $ mkdir unpacked
```

3. pip 명령을 사용하여 unpacked 디렉터리에 프로젝트를 설치합니다.

```
testpiglatin $ bin/pip install -t $PWD/unpacked piglatin
```

출력

```
Collecting piglatin
Using cached piglatin-1.0.6-py2.py3-none-any.whl (3.1 kB)
Installing collected packages: piglatin
Successfully installed piglatin-1.0.6
```

4. 디렉터리의 내용을 확인합니다.

```
testpiglatin $ ls
```

출력

```
bin lib pyvenv.cfg unpacked
```

5. unpacked 디렉터리로 변경하고 내용을 표시합니다.

```
testpiglatin $ cd unpacked
unpacked $ ls
```

출력

```
piglatin piglatin-1.0.6.dist-info
```

6. zip 명령을 사용하여 piglatin 프로젝트의 내용을 library.zip 파일에 포함합니다.

```
unpacked $ zip -r9 ../library.zip *
```

출력

```
adding: piglatin/ (stored 0%)
adding: piglatin/__init__.py (deflated 56%)
adding: piglatin/__pycache__/ (stored 0%)
adding: piglatin/__pycache__/__init__.cpython-39.pyc (deflated 31%)
adding: piglatin-1.0.6.dist-info/ (stored 0%)
adding: piglatin-1.0.6.dist-info/RECORD (deflated 39%)
adding: piglatin-1.0.6.dist-info/LICENSE (deflated 41%)
adding: piglatin-1.0.6.dist-info/WHEEL (deflated 15%)
adding: piglatin-1.0.6.dist-info/REQUESTED (stored 0%)
adding: piglatin-1.0.6.dist-info/INSTALLER (stored 0%)
adding: piglatin-1.0.6.dist-info/METADATA (deflated 48%)
```

7. (선택 사항) 다음 명령을 사용하여 로컬에서 가져오기를 테스트합니다.

   a. Python 경로를 library.zip 파일 위치로 설정하고 Python을 시작합니다.

```
/home $ PYTHONPATH=/tmp/testpiglatin/library.zip
/home $ python3
```

   출력

```
Python 3.9.6 (default, Jun 29 2021, 06:20:32)
[Clang 12.0.0 (clang-1200.0.32.29)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
```

   b. 라이브러리를 가져오고 테스트 명령을 실행합니다.

```
>>> import piglatin
>>> piglatin.translate('hello')
```

   출력

```
'ello-hay'
```

8. 다음과 같은 명령을 사용하여 Amazon S3에서 `.zip` 파일을 추가하고 Athena에 있는 노트북으로 파일을 가져와서 테스트합니다.

```
sc.addPyFile('s3://DOC-EXAMPLE-BUCKET/library.zip')

import piglatin
piglatin.translate('hello')

from pyspark.sql.functions import udf
from pyspark.sql.functions import col

hi_udf = udf(piglatin.translate)

df = spark.createDataFrame([(1, "hello"), (2, "world")])

df.withColumn("col", hi_udf(col('_2'))).show()
```

출력

```
Calculation started (calculation_id=e2c0a06e-f45d-d96d-9b8c-ff6a58b2a525) in
 (session=82c0a06d-d60e-8c66-5d12-23bcd55a6457). Checking calculation status...
Calculation completed.
+---+-----+--------+
| _1|   _2|     col|
+---+-----+--------+
|  1|hello|ello-hay|
|  2|world|orld-way|
+---+-----+--------+
```

PyPI에서 종속성이 있는 Python .zip 파일 가져오기

이 예제에서는 PyPI에서 마크다운의 텍스트를 Gemini 텍스트 형식으로 변환하는 md2gemini 패키지를 가져옵니다. 이 패키지는 다음에 대한 종속성을 가집니다.

```
cjkwrap
mistune
wcwidth
```

종속성이 있는 Python .zip 파일을 가져오려면

1. 로컬 컴퓨터에서 다음 명령을 사용하여 testmd2gemini 디렉터리를 만들고 가상 환경을 생성합니다.

```
/tmp $ mkdir testmd2gemini
/tmp $ cd testmd2gemini
testmd2gemini$ virtualenv .
```

2. 프로젝트를 보관할 unpacked 하위 디렉터리를 생성합니다.

```
testmd2gemini $ mkdir unpacked
```

3. pip 명령을 사용하여 unpacked 디렉터리에 프로젝트를 설치합니다.

```
/testmd2gemini $ bin/pip install -t $PWD/unpacked md2gemini
```

출력

```
Collecting md2gemini
  Downloading md2gemini-1.9.0-py3-none-any.whl (31 kB)
Collecting wcwidth
  Downloading wcwidth-0.2.5-py2.py3-none-any.whl (30 kB)
Collecting mistune<3,>=2.0.0
  Downloading mistune-2.0.2-py2.py3-none-any.whl (24 kB)
Collecting cjkwrap
  Downloading CJKwrap-2.2-py2.py3-none-any.whl (4.3 kB)
Installing collected packages: wcwidth, mistune, cjkwrap, md2gemini
Successfully installed cjkwrap-2.2 md2gemini-1.9.0 mistune-2.0.2 wcwidth-0.2.5
...
```

4. unpacked 디렉토리로 변경하고 내용을 확인합니다.

```
testmd2gemini $ cd unpacked
unpacked $ ls -lah
```

출력

```
total 16
drwxr-xr-x  13 user1  wheel   416B Jun  7 18:43 .
drwxr-xr-x   8 user1  wheel   256B Jun  7 18:44 ..
```

```
drwxr-xr-x   9 user1  staff   288B Jun  7 18:43 CJKwrap-2.2.dist-info
drwxr-xr-x   3 user1  staff    96B Jun  7 18:43 __pycache__
drwxr-xr-x   3 user1  staff    96B Jun  7 18:43 bin
-rw-r--r--   1 user1  staff   5.0K Jun  7 18:43 cjkwrap.py
drwxr-xr-x   7 user1  staff   224B Jun  7 18:43 md2gemini
drwxr-xr-x  10 user1  staff   320B Jun  7 18:43 md2gemini-1.9.0.dist-info
drwxr-xr-x  12 user1  staff   384B Jun  7 18:43 mistune
drwxr-xr-x   8 user1  staff   256B Jun  7 18:43 mistune-2.0.2.dist-info
drwxr-xr-x  16 user1  staff   512B Jun  7 18:43 tests
drwxr-xr-x  10 user1  staff   320B Jun  7 18:43 wcwidth
drwxr-xr-x   9 user1  staff   288B Jun  7 18:43 wcwidth-0.2.5.dist-info
```

5. zip 명령을 사용하여 md2gemini 프로젝트의 내용을 md2gemini.zip 파일에 포함합니다.

```
unpacked $ zip -r9 ../md2gemini *
```

출력

```
  adding: CJKwrap-2.2.dist-info/ (stored 0%)
  adding: CJKwrap-2.2.dist-info/RECORD (deflated 37%)
  ....
  adding: wcwidth-0.2.5.dist-info/INSTALLER (stored 0%)
  adding: wcwidth-0.2.5.dist-info/METADATA (deflated 62%)
```

6. (선택 사항) 다음 명령을 사용하여 라이브러리가 로컬 컴퓨터에서 작동하는지 테스트합니다.

   a. Python 경로를 md2gemini.zip 파일 위치로 설정하고 Python을 시작합니다.

```
/home $ PYTHONPATH=/tmp/testmd2gemini/md2gemini.zip
/home python3
```

   b. 라이브러리를 가져오고 테스트를 실행합니다.

```
>>> from md2gemini import md2gemini
>>> print(md2gemini('[abc](https://abc.def)'))
```

출력

```
https://abc.def abc
```

7. 다음 명령을 사용하여 Amazon S3에서 `.zip` 파일을 추가하고 Athena에 있는 노트북으로 파일을 가져와서 비 UDF 테스트를 수행합니다.

```
# (non udf test)
sc.addPyFile('s3://DOC-EXAMPLE-BUCKET/md2gemini.zip')
from md2gemini import md2gemini
print(md2gemini('[abc](https://abc.def)'))
```

출력

```
Calculation started (calculation_id=0ac0a082-6c3f-5a8f-eb6e-f8e9a5f9bc44) in
 (session=36c0a082-5338-3755-9f41-0cc954c55b35). Checking calculation status...
Calculation completed.
=> https://abc.def (https://abc.def/) abc
```

8. 다음 명령을 사용하여 UDF 테스트를 수행합니다.

```
# (udf test)

from pyspark.sql.functions import udf
from pyspark.sql.functions import col
from md2gemini import md2gemini


hi_udf = udf(md2gemini)
df = spark.createDataFrame([(1, "[first website](https://abc.def)"), (2, "[second
 website](https://aws.com)")])
df.withColumn("col", hi_udf(col('_2'))).show()
```

출력

```
Calculation started (calculation_id=60c0a082-f04d-41c1-a10d-d5d365ef5157) in
 (session=36c0a082-5338-3755-9f41-0cc954c55b35). Checking calculation status...
Calculation completed.
+---+--------------------+--------------------+
| _1|                  _2|                 col|
+---+--------------------+--------------------+
|  1|[first website](h...|=> https://abc.de...|
|  2|[second website](...|=> https://aws.co...|
+---+--------------------+--------------------+
```

JAR 파일 및 사용자 지정 Spark 구성 추가

Amazon Athena for Apache Spark에서 세션을 생성하거나 편집할 때 Spark 속성을 사용하여 세션에 대한 `.jar` 파일, 패키지 또는 다른 사용자 지정 구성을 지정할 수 있습니다. Spark 속성을 지정하려면 Athena 콘솔, AWS CLI 또는 Athena API를 사용할 수 있습니다.

Athena 콘솔을 사용하여 Spark 속성 지정

Athena 콘솔에서 노트북을 생성하거나 현재 세션을 편집할 때 Spark 속성을 지정할 수 있습니다.

노트북 생성 또는 세션 세부 정보 편집 대화 상자에서 속성을 추가하려면

1. Spark 속성을 확장합니다.
2. 속성을 추가하려면 테이블에서 편집 또는 JSON으로 편집 옵션을 사용합니다.
   - 테이블에서 편집 옵션의 경우 속성 추가를 선택하여 속성을 추가하거나 제거를 선택하여 속성을 제거합니다. 키 및 값 상자를 사용하여 속성 이름과 값을 입력합니다.
     - 사용자 지정 `.jar` 파일을 추가하려면 `spark.jars` 속성을 사용합니다.
     - 패키지 파일을 지정하려면 `spark.jars.packages` 속성을 사용합니다.
   - 구성을 직접 입력하고 편집하려면 JSON으로 편집 옵션을 선택합니다. JSON 텍스트 편집기에서 다음 작업을 수행할 수 있습니다.
     - 복사를 선택하여 JSON 텍스트를 클립보드에 복사합니다.
     - 지우기를 선택하여 JSON 편집기에서 모든 텍스트를 제거합니다.
     - 설정(톱니) 아이콘을 선택하여 줄 바꿈을 구성하거나 JSON 편집기의 색상 테마를 선택합니다.

참고

- Athena for Spark에서 속성을 설정할 수 있습니다. 이 방법은 SparkConf 객체에서 Spark 속성을 직접 설정하는 것과 같습니다.
- 모든 Spark 속성을 `spark.` 접두사로 시작합니다. 다른 접두사의 속성은 무시됩니다.
- Athena에서 사용자 지정 구성에 대해 일부 Spark 속성은 사용할 수 없습니다. 구성이 제한된 `StartSession` 요청을 제출하면 세션이 시작되지 않습니다.
  - `spark.athena.` 접두사는 예약되어 있으므로 사용할 수 없습니다.

AWS CLI 또는 Athena API를 사용하여 사용자 지정 구성 제공

AWS CLI 또는 Athena API를 사용하여 세션 구성을 제공하려면 StartSession API 작업 또는 start-session CLI 명령을 사용합니다. `StartSession` 요청에서 EngineConfiguration 객체의 `SparkProperties` 필드를 사용하여 구성 정보를 JSON 형식으로 전달합니다. 그러면 지정된 구성으로 세션이 시작됩니다. 요청 구문은 Amazon Athena API 참조의 StartSession을 참조하세요.

세션 시작 오류 문제 해결

세션 시작 중에 사용자 지정 구성 오류가 발생하면 Athena for Spark 콘솔에 오류 메시지 배너가 표시됩니다. 세션 시작 오류를 해결하기 위해 세션 상태 변경 또는 로깅 정보를 확인할 수 있습니다.

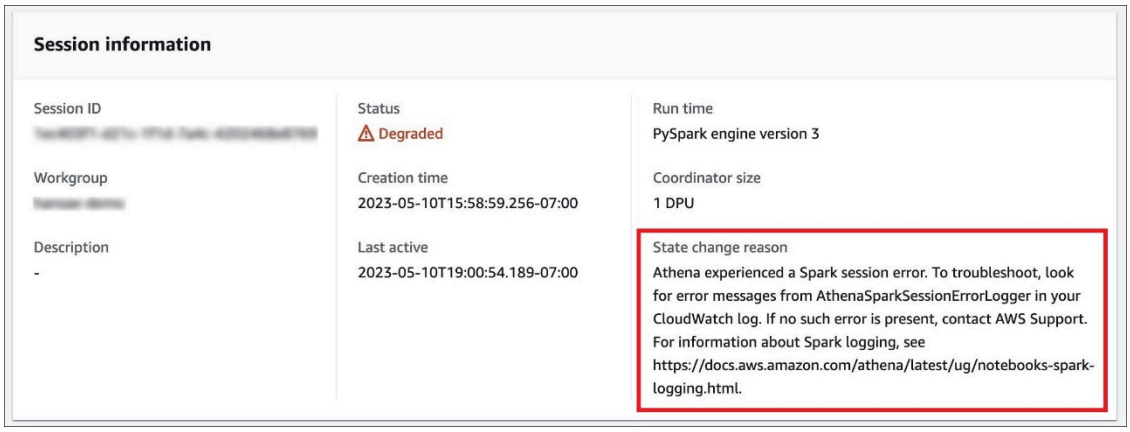
세션 상태 변경 정보 보기

Athena 노트북 편집기 또는 Athena API에서 세션 상태 변경에 대한 세부 정보를 얻을 수 있습니다.

Athena 콘솔에서 세션 상태 정보를 보려면

1. Athena 노트북 편집기의 오른쪽 상단에 있는 세션 메뉴에서 세부 정보 보기를 선택합니다.
2. 현재 세션 탭을 확인합니다. 세션 정보 섹션에는 세션 ID, 작업 그룹, 상태 및 상태 변경 이유와 같은 정보가 표시됩니다.

   다음 화면 캡처 예제에서는 Athena의 Spark 세션 오류에 대한 세션 정보 대화 상자의 상태 변경 이유 섹션에 있는 정보를 보여줍니다.



Athena API를 사용하여 세션 상태 정보를 보려면

- Athena API에서는 SessionStatus 객체의 `StateChangeReason` 필드에서 세션 상태 변경 정보를 찾을 수 있습니다.

> **Note**
>
> 세션을 수동으로 중지한 후 또는 유휴 제한 시간(기본값은 20분) 이후 세션이 중지된 경우 StateChangeReason 값이 Session was terminated per request로 변경되었습니다.

로깅을 사용하여 세션 시작 오류 문제 해결

세션 시작 중에 발생하는 사용자 지정 구성 오류는 Amazon CloudWatch에서 기록합니다. CloudWatch Logs는 `AthenaSparkSessionErrorLogger`에서 오류 메시지를 검색하여 실패한 세션 시작 문제를 해결합니다.

Spark 로깅에 대한 자세한 내용은 Athena에서 Spark 애플리케이션 이벤트 로깅을 참조하세요.

Athena for Spark에서 세션 문제 해결에 대한 자세한 내용은 세션 문제 해결 섹션을 참조하세요.

지원되는 데이터 및 스토리지 형식

다음 표에는 Apache Spark용 Athena에서 기본적으로 지원되는 형식이 나와 있습니다.

Data format(데이터 형식)	읽기(Read)	쓰기(Write)	Write compression(쓰기 압축)
parquet	예	예	없음, 비압축, snappy, gzip
orc	예	예	없음, snappy, zlib, lzo
json	예	예	bzip2, gzip, deflate
csv	예	예	bzip2, gzip, deflate
텍스트	예	예	없음, bzip2, gzip, deflate
이진 파일	yes	해당 사항 없음	N/A

CloudWatch 지표를 사용하여 Apache Spark 계산 모니터링

Spark 지원 작업 그룹에 대한 Publish CloudWatch metrics 옵션을 선택하면 Athena에서 계산 관련 지표를 Amazon CloudWatch에 게시합니다. CloudWatch 콘솔에서 사용자 지정 대시보드를 생성하고, 지표에 대한 경보 및 트리거를 설정할 수 있습니다.

Athena는 다음의 지표를 `AmazonAthenaForApacheSpark` 네임스페이스 아래 CloudWatch 콘솔에 게시합니다.

- `DPUCount` - 세션 중에 계산을 실행하는 데 사용된 DPU 수입니다.

이 지표에는 다음과 같은 차원이 있습니다.

- `SessionId` - 계산이 제출되는 세션의 ID입니다.
- `WorkGroup` - 작업 그룹의 이름입니다.

Amazon CloudWatch 콘솔에서 Spark 지원 작업 그룹에 대한 지표를 보려면

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창에서 지표, 모든 지표를 선택합니다.
3. AmazonAthenaForApacheSpark 네임스페이스를 선택합니다.

CLI를 사용하여 지표를 보려면

- 다음 중 하나를 수행하십시오.
  - Athena Spark 지원 작업 그룹에 대한 지표를 나열하려면 명령 프롬프트를 열고 다음 명령을 사용합니다.

```
aws cloudwatch list-metrics --namespace "AmazonAthenaForApacheSpark"
```

  - 사용 가능한 모든 지표의 목록을 보려면 다음 명령을 사용합니다.

```
aws cloudwatch list-metrics
```

Athena에서 Apache Spark 계산을 위한 CloudWatch 지표 및 차원 목록

Spark 지원 Athena 작업 그룹에서 CloudWatch 지표를 활성화했다면 다음 지표가 작업 그룹별로 CloudWatch로 전송됩니다. 지표에는 `AmazonAthenaForApacheSpark` 네임스페이스가 사용됩니다.

지표 이름	설명
DPUCount	세션 중에 계산을 실행하는 데 사용된 DPU(데이터 처리 단위) 수입니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다.

이 지표에는 다음과 같은 차원이 있습니다.

측정기준	설명
SessionId	계산이 제출되는 세션의 ID입니다.
WorkGroup	작업 그룹의 이름입니다.

Athena for Spark에서 Amazon S3 요청자 지불 버킷 활성화

Amazon S3 버킷이 요청자 지불로 구성된 경우 쿼리를 실행하는 사용자의 계정에 쿼리와 관련된 데이터 액세스 및 데이터 전송 요금이 부과됩니다. 자세한 내용은 Amazon S3 사용 설명서의 스토리지 전송 및 사용량에 대한 요청자 지불액 버킷 사용을 참조하세요.

Athena for Spark에서는 요청자 지불 버킷은 작업 그룹이 아닌 세션을 기반으로 활성화됩니다. 상위 수준에서 요청자 지불 버킷을 활성화하려면 다음 단계가 포함됩니다.

1. Amazon S3 콘솔에서 버킷에 대한 속성에서 요청자 지불을 활성화하고 액세스를 지정하는 버킷 정책을 추가합니다.
2. IAM 콘솔에서 버킷에 대한 액세스를 허용하는 IAM 정책을 생성하고 요청자 지불 버킷에 액세스하는 데 사용할 IAM 역할에 정책을 연결합니다.
3. Athena for Spark에서 세션 속성을 추가하여 요청자 지불 기능을 활성화합니다.

1. Amazon S3 버킷에서 요청자 지불 활성화 및 버킷 정책 추가

Amazon S3 버킷에 대한 요청자 지불을 활성화하려면

1. https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.
2. 버킷 목록에서 요청자 지불을 활성화할 버킷에 대한 링크를 선택합니다.
3. 버킷 페이지에서 속성 탭을 선택합니다.
4. 요청자 지불 섹션으로 스크롤을 내리고 편집을 선택합니다.
5. 요청자 지불 편집 페이지에서 활성화를 선택하고 변경 사항 저장을 선택합니다.
6. 권한 탭을 선택합니다.
7. 버킷 정책 섹션에서 편집을 선택합니다.
8. 버킷 정책 편집 페이지에서 소스 버킷에 원하는 버킷 정책을 적용합니다. 다음 정책 예제는 모든 AWS 보안 주체("AWS": "*")에 대한 액세스를 제공하지만 액세스 권한을 더 세분화할 수 있습니다. 예를 들어 다른 계정에서 특정 IAM 역할만 지정하려고 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Statement1",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::account_number-us-east-1-my-s3-requester-pays-
bucket",
                "arn:aws:s3:::account_number-us-east-1-my-s3-requester-pays-bucket/
*"
            ]
        }
    ]
}
```

2. IAM 정책을 생성하고 IAM 역할에 연결합니다.

그런 다음 버킷에 액세스를 허용하는 IAM 정책을 생성합니다. 그리고 요청자 지불 버킷에 액세스하는 데 사용할 역할에 정책을 연결합니다.

요청자 지불 버킷에 대한 IAM 정책을 생성하고 정책을 역할에 연결하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. IAM 콘솔의 탐색 창에서 정책을 선택합니다.
3. 정책 생성을 선택합니다.
4. JSON을 선택합니다.
5. 정책 편집기에서 다음과 같은 정책을 추가합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "s3:*"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::account_number-us-east-1-my-s3-requester-pays-
bucket",
                "arn:aws:s3:::account_number-us-east-1-my-s3-requester-pays-bucket/
*"
            ]
        }
    ]
}
```

6. 다음을 선택합니다.
7. 검토 및 생성 페이지에 정책의 이름과 설명(선택 사항)을 입력하고 정책 생성을 선택합니다.
8. 탐색 창에서 역할을 선택합니다.
9. 역할 페이지에서 사용하려는 역할을 찾고 역할 이름 링크를 선택합니다.
10. 권한 정책 섹션에서 권한 추가, 정책 연결을 차례로 선택합니다.
11. 기타 권한 정책 섹션에서 생성한 정책의 확인란을 선택하고 권한 추가를 선택합니다.

3. Athena for Spark 세션 속성 추가

요청자 지불에 대한 Amazon S3 버킷과 관련 권한을 구성한 후 Athena for Spark 세션에서 이 기능을 활성화할 수 있습니다.

Athena for Spark 세션에서 요청자 지불 버킷을 활성화하려면

1. 노트북 편집기의 오른쪽 상단 Session(세션) 메뉴에서 Edit session(세션 편집)을 선택합니다.
2. Spark 속성을 확장합니다.
3. JSON으로 편집을 선택합니다.
4. JSON 텍스트 편집기에 다음을 입력합니다.

```
{
  "spark.hadoop.fs.s3.useRequesterPaysHeader":"true"
}
```

5. Save(저장)를 선택합니다.

Apache Spark 암호화 활성화

Athena에서 Apache Spark 암호화를 활성화할 수 있습니다. 이렇게 하면 Spark 노드 간에 전송 중 데이터는 물론, Spark에서 로컬로 저장된 저장 데이터도 암호화됩니다. 이 데이터의 보안을 강화하기 위해 Athena는 다음과 같은 암호화 구성을 사용합니다.

```
spark.io.encryption.keySizeBits="256"
spark.io.encryption.keygen.algorithm="HmacSHA384"
```

Spark 암호화를 활성화하려면 Athena 콘솔, AWS CLI 또는 Athena API를 사용할 수 있습니다.

Athena 콘솔을 사용하여 Spark 암호화 활성화

Spark 암호화가 활성화된 새 노트북을 생성하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.
3. 다음 중 하나를 수행하십시오.
   - Notebook explorer(노트북 탐색기)에서 Create notebook(노트북 생성)을 선택합니다.

- Notebook editor(노트북 편집기)에서 Create notebook(노트북 생성)을 선택하거나 더하기 아이콘(+)을 선택하여 노트북을 추가합니다.

4. 노트북 이름에는 노트북의 이름을 입력합니다.
5. Spark 속성 옵션을 확장합니다.
6. Spark 암호화 켜기를 선택합니다.
7. 생성(Create)을 선택합니다.

생성한 노트북 세션은 암호화됩니다. 평소처럼 새 노트북을 사용합니다. 나중에 노트북을 사용하는 새 세션을 시작하면 새 세션도 암호화됩니다.

또한 Athena 콘솔을 사용하여 기존 노트북의 Spark 암호화를 활성화할 수도 있습니다.

기존 노트북의 암호화를 활성화하려면

1. 이전에 생성한 노트북의 새 세션을 엽니다.
2. 노트북 편집기의 오른쪽 상단 Session(세션) 메뉴에서 Edit session(세션 편집)을 선택합니다.
3. 세션 세부 정보 편집 대화 상자에서 Spark 속성을 확장합니다.
4. Spark 암호화 켜기를 선택합니다.
5. Save(저장)를 선택합니다.

콘솔은 암호화가 활성화된 새 세션을 시작합니다. 이 노트북에 대해 생성한 이후 세션에서도 암호화가 활성화됩니다.

AWS CLI를 사용하여 Spark 암호화 활성화

AWS CLI를 사용하여 적절한 Spark 속성을 지정해 세션을 시작할 때 암호화를 활성화할 수 있습니다.

AWS CLI를 사용하여 Spark 암호화를 활성화하려면

1. 다음과 같은 명령을 사용하여 Spark 암호화 속성을 지정하는 엔진 구성 JSON 객체를 생성합니다.

```
ENGINE_CONFIGURATION_JSON=$(
  cat <<EOF
{
    "CoordinatorDpuSize": 1,
    "MaxConcurrentDpus": 20,
    "DefaultExecutorDpuSize": 1,
```

```
    "SparkProperties": {
      "spark.authenticate": "true",
      "spark.io.encryption.enabled": "true",
      "spark.network.crypto.enabled": "true"
    }
}
EOF
)
```

2. AWS CLI에서 다음 예제와 같이 `athena start-session` 명령을 사용하여 생성한 JSON 객체를 `--engine-configuration` 인수로 전달합니다.

```
aws athena start-session \
   --region "region" \
   --work-group "your-work-group" \
   --engine-configuration "$ENGINE_CONFIGURATION_JSON"
```

Athena API를 사용하여 Spark 암호화 활성화

Athena API를 사용하여 Spark 암호화를 활성화하려면 StartSession 작업 및 해당 EngineConfiguration `SparkProperties` 파라미터를 사용하여 `StartSession` 요청의 암호화 구성을 지정합니다.

Athena for Spark에서 크로스 계정 AWS Glue 액세스 구성

이 주제에서는 크로스 계정 AWS Glue 액세스를 위해 소비자 계정 *666666666666* 및 소유자 계정 *999999999999*을 구성하는 방법을 보여줍니다. 계정이 구성되면 소비자 계정은 소유자의 AWS Glue 데이터베이스 및 테이블에서 Athena for Spark의 쿼리를 실행할 수 있습니다.

1. AWS Glue에서 소비자 역할에 대한 액세스를 제공합니다.

AWS Glue에서는 소유자가 소유자의 AWS Glue 데이터 카탈로그에 대한 소비자 역할 액세스를 제공하는 정책을 생성합니다.

소유자의 데이터 카탈로그에 대한 소비자 역할 액세스를 허용하는 AWS Glue 정책을 추가하려면

1. 카탈로그 소유자 계정을 사용하여 AWS Management Console에 로그인합니다.
2. https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
3. 탐색 창에서 데이터 카탈로그를 확장하고 카탈로그 설정을 선택합니다.

4. 데이터 카탈로그 설정 페이지의 권한 섹션에서 다음과 같은 정책을 추가합니다. 이 정책은 소유자 계정 *999999999999*에 있는 데이터 카탈로그에 대한 소비자 계정 *666666666666* 액세스 역할을 제공합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Cataloguers",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::666666666666:role/Admin",
          "arn:aws:iam::666666666666:role/AWSAthenaSparkExecutionRole"
        ]
      },
      "Action": "glue:*",
      "Resource": [
        "arn:aws:glue:us-west-2:999999999999:catalog",
        "arn:aws:glue:us-west-2:999999999999:database/*",
        "arn:aws:glue:us-west-2:999999999999:table/*"
      ]
    }
  ]
}
```

2. 액세스를 위해 소비자 계정 구성

소비자 계정에서 소유자의 AWS Glue Data Catalog, 데이터베이스 및 테이블에 대한 액세스를 허용하는 정책을 생성하고 정책을 역할에 연결합니다. 다음 예제에서는 소비자 계정 *666666666666*을 사용합니다.

소유자 AWS Glue Data Catalog에 액세스하기 위한 AWS Glue 정책을 생성하려면

1. 소비자 계정을 사용하여 AWS Management Console에 로그인합니다.
2. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
3. 서비스 탐색 창에서 액세스 관리를 확장하고 정책을 선택합니다.
4. 정책 생성을 선택합니다.
5. 권한 지정 페이지에서 JSON을 선택합니다.

6. 정책 편집기에서 소유자 계정의 데이터 카탈로그에 대한 AWS Glue 작업을 허용하는 다음과 같은 JSON 문을 입력합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "glue:*",
            "Resource": [
                "arn:aws:glue:us-east-1:999999999999:catalog",
                "arn:aws:glue:us-east-1:999999999999:database/*",
                "arn:aws:glue:us-east-1:999999999999:table/*"
            ]
        }
    ]
}
```

7. 다음을 선택합니다.
8. 검토 및 생성 페이지의 정책 이름에 정책 이름을 입력합니다.
9. 정책 생성을 선택합니다.

그런 다음 소비자 계정에서 IAM 콘솔을 사용하여 방금 생성한 정책을 IAM 역할 또는 소비자 계정이 소유자의 데이터 카탈로그에 액세스하는 데 사용할 역할에 연결합니다.

소비자 계정의 역할에 AWS Glue 정책을 연결하려면

1. 소비자 계정 IAM 콘솔의 탐색 창에서 역할을 선택합니다.
2. 역할 페이지에서 정책을 연결하려는 역할을 찾습니다.
3. 권한 추가를 선택하고 정책 연결을 선택합니다.
4. 방금 생성한 정책을 찾습니다.
5. 정책의 확인란을 선택하고 권한 추가를 선택합니다.
6. 단계를 반복하여 사용하려는 다른 역할에 정책을 추가합니다.

3. 세션을 구성하고 쿼리를 생성합니다.

Athena Spark의 요청자 계정에서 지정된 역할을 사용하여 노트북 생성 또는 현재 세션 편집을 통해 액세스를 테스트할 세션을 생성합니다. 세션 속성을 구성할 때 다음 중 하나를 지정합니다.

- Glue 카탈로그 구분자 – 이 방법에서는 쿼리에 소유자 계정 ID를 포함합니다. 세션을 사용하여 다른 소유자의 데이터 카탈로그를 쿼리하려는 경우 이 방법을 사용합니다.
- Glue 카탈로그 ID - 이 방법에서는 데이터베이스를 직접 쿼리합니다. 세션을 사용하여 단일 소유자의 데이터 카탈로그만 쿼리하려는 경우 이 방법이 더 편리합니다.

AWS Glue 카탈로그 구분자 방법 사용

세션 속성을 편집할 때 다음을 추가합니다.

```
{
    "spark.hadoop.aws.glue.catalog.separator": "/"
}
```

셀에서 쿼리를 실행할 때 다음 예제와 같은 구문을 사용합니다. 이 FROM 절에서는 데이터베이스 이름 앞에 카탈로그 ID와 구분자가 필요합니다.

```
df = spark.sql('SELECT requestip, uri, method, status FROM `999999999999/
mydatabase`.cloudfront_logs LIMIT 5')
df.show()
```

AWS Glue 카탈로그 ID 방법 사용

세션 속성을 편집할 때 다음 속성을 입력합니다. *999999999999*를 소유자 계정 ID로 바꿉니다.

```
{
    "spark.hadoop.hive.metastore.glue.catalogid": "999999999999"
}
```

셀에서 쿼리를 실행할 때는 다음과 같은 구문을 사용합니다. 이 FROM 절에서는 데이터베이스 이름 앞에 카탈로그 ID와 구분자가 필요하지 않습니다.

```
df = spark.sql('SELECT * FROM mydatabase.cloudfront_logs LIMIT 10')
df.show()
```

추가적인 리소스

AWS Glue 데이터 카탈로그에 대한 교차 계정 액세스

AWS Lake Formation 개발자 안내서의 Managing cross-account permissions using both AWS Glue and Lake Formation.

AWS Prescriptive Guidance Patterns의 Configure cross-account access to a shared AWS Glue Data Catalog using Amazon Athena.

Amazon Athena for Apache Spark의 서비스 할당량

서비스 할당량(한도라고도 함)은 AWS 계정에서 사용할 수 있는 서비스 리소스 또는 작업의 최대 수를 말합니다. Amazon Athena for Spark에서 사용할 수 있는 다른 AWS 서비스의 서비스 할당량에 대한 자세한 내용은 Amazon Web Services 일반 참조의 AWS service quotas를 참조하세요.

> **Note**
>
> 새 AWS 계정의 초기 할당량은 낮지만 시간이 지남에 따라 증가할 수 있습니다. Amazon Athena for Apache Spark는 각 AWS 리전 내의 계정 사용량을 모니터링하고 사용량에 따라 할당량을 자동으로 늘립니다. 요구 사항이 명시된 한도를 초과하는 경우 고객 지원에 문의하세요.

Amazon Athena for Apache Spark에 대한 서비스 할당량은 다음 테이블에 나열되어 있습니다.

명칭	기본값	조정 가능	설명
Apache Spark DPU 동시성	160	아니요	현재 AWS 리전의 단일 계정에서 Apache Spark 계산을 위해 동시에 사용할 수 있는 최대 데이터 처리 장치(DPU) 수입니다. DPU는 4 vCPU의 컴퓨팅 파워와 16GB 메모리로 구성된 프로세싱 파워의 상대적 측정값입니다.
Apache Spark 세션 DPU 동시성	60	아니요	세션 내에서 Apache Spark 계산을 위해 동시에 사용할 수 있는 최대 DPU 수입니다.

Athena 노트북 API

다음 목록에는 Athena 노트북 API 작업에 대한 참조 링크가 포함되어 있습니다. 데이터 구조 및 기타 Athena API 작업에 대해서는 Amazon Athena API 참조를 참조하세요.

- CreateNotebook
- CreatePresignedNotebookUrl
- DeleteNotebook
- ExportNotebook
- GetCalculationExecution
- GetCalculationExecutionCode
- GetCalculationExecutionStatus
- GetNotebookMetadata
- GetSession
- GetSessionStatus
- ImportNotebook
- ListApplicationDPUSizes
- ListCalculationExecutions
- ListExecutors
- ListNotebookMetadata
- ListNotebookSessions
- ListSessions
- StartCalculationExecution
- StartSession
- StopCalculationExecution
- TerminateSession
- UpdateNotebook
- UpdateNotebookMetadata

Athena for Spark의 알려진 문제

이 페이지에서는 Apache Spark용 Athena의 일부 알려진 문제에 대해 설명합니다.

테이블을 생성할 때 잘못된 인수 예외 발생

Spark에서는 빈 위치 속성으로 데이터베이스를 생성할 수 없지만 Spark 외부에서 생성된 AWS Glue의 데이터베이스에는 빈 `LOCATION` 속성이 존재할 수 있습니다.

테이블을 생성하고 빈 LOCATION 필드가 있는 AWS Glue 데이터베이스를 지정하면 IllegalArgumentException: Cannot create a path from an empty string.(IllegalArgumentException: 빈 문자열에서 경로를 생성할 수 없습니다) 예외가 발생할 수 있습니다.

예를 들어 AWS Glue의 기본 데이터베이스에 빈 LOCATION 필드가 있는 경우에 다음 명령을 실행하면 예외가 발생합니다.

```
spark.sql("create table testTable (firstName STRING)")
```

권장 솔루션 A - AWS Glue를 사용하여 사용 중인 데이터베이스에 위치를 추가합니다.

AWS Glue 데이터베이스에 위치를 추가하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택합니다.
3. 데이터베이스 목록에서 편집하려는 데이터베이스를 선택합니다.
4. 데이터베이스에 대한 세부 정보 페이지에서 Edit(편집)을 선택합니다.
5. Update a database(데이터베이스 업데이트) 페이지에서 Location(위치)에 Amazon S3 위치를 입력합니다.
6. Update Database(데이터베이스 업데이트)를 선택합니다.

권장 솔루션 B - Amazon S3에 유효한 기존 위치가 있는 다른 AWS Glue 데이터베이스를 사용합니다. 예를 들어 dbWithLocation 데이터베이스가 있는 경우 spark.sql("use dbWithLocation") 명령을 사용하여 해당 데이터베이스로 전환합니다.

권장 솔루션 C - Spark SQL을 사용하여 테이블을 생성할 경우 다음 예제와 같이 location의 값을 지정합니다.

```
spark.sql("create table testTable (firstName STRING)
       location 's3://DOC-EXAMPLE-BUCKET/'").
```

권장 솔루션 D - 테이블을 생성할 때 위치를 지정했지만 문제가 계속 발생하는 경우, 제공한 Amazon S3 경로에 후행 슬래시가 있는지 확인합니다. 예를 들어 다음 명령을 실행하면 잘못된 인수 예외가 발생합니다.

```
spark.sql("create table testTable (firstName STRING)
```

```
        location 's3://DOC-EXAMPLE-BUCKET'")
```

이 문제를 해결하려면 위치에 후행 슬래시를 추가합니다(예:'s3:// DOC-EXAMPLE-BUCKET/').

작업 그룹 위치에 생성된 데이터베이스

spark.sql('create database db')과 같은 명령을 사용하여 데이터베이스를 생성하고 데이터베이스의 위치를 지정하지 않는 경우 Athena에서 작업 그룹 위치에 하위 디렉터리를 생성하고 해당 위치를 새로 만든 데이터베이스에 사용합니다.

AWS Glue 기본 데이터베이스에서 Hive 관리형 테이블 관련 문제

AWS Glue에서 기본 데이터베이스의 Location 속성이 비어 있지 않고 Amazon S3의 유효한 위치를 지정하며 Athena for Spark를 사용하여 AWS Glue 기본 데이터베이스에서 Hive 관리형 테이블을 생성하는 경우, AWS Glue 데이터베이스에서 지정한 위치 대신 Athena Spark 작업 그룹에 지정된 Amazon S3 위치로 데이터가 기록됩니다.

이 문제는 Apache Hive에서 기본 데이터베이스를 처리하는 방식 때문에 발생합니다. Apache Hive는 Hive 웨어하우스 루트 위치에 테이블 데이터를 생성하는데, 이 위치가 실제 기본 데이터베이스 위치와 다를 수 있습니다.

Athena for Spark를 사용하여 AWS Glue의 기본 데이터베이스 아래에 Hive 관리형 테이블을 생성하는 경우 AWS Glue 테이블 메타데이터에서 서로 다른 두 위치를 가리킬 수 있습니다. 이로 인해 INSERT 또는 DROP TABLE 작업을 시도할 때 예상치 못한 동작이 발생할 수 있습니다.

문제를 재현하는 단계는 다음과 같습니다.

1. Athena for Spark에서는 다음 방법 중 하나를 사용하여 Hive 관리형 테이블을 생성하거나 저장합니다.
   - CREATE TABLE $tableName과 같은 SQL 문
   - df.write.mode("overwrite").saveAsTable($tableName)과 같은 PySpark 명령은 Dataframe API에서 path 옵션을 지정하지 않습니다.

   이때 AWS Glue 콘솔에 Amazon S3의 잘못된 테이블 위치가 표시될 수 있습니다.
2. Athena for Spark에서는 DROP TABLE $table_name 문을 사용하여 사용자가 생성한 테이블을 삭제합니다.
3. DROP TABLE 문을 실행한 후에도 Amazon S3의 기본 파일이 여전히 존재합니다.

이 문제를 해결하려면 다음 중 한 가지를 사용하십시오.

솔루션 A - Hive 관리형 테이블을 생성하는 경우 다른 AWS Glue 데이터베이스를 사용합니다.

솔루션 B - AWS Glue에서 기본 데이터베이스의 빈 위치를 지정합니다. 그런 다음 기본 데이터베이스에 관리형 테이블을 생성합니다.

Athena for Spark 및 Athena SQL 사이에서 CSV 및 JSON 파일 형식 비호환성

오픈 소스 Spark의 알려진 문제로 인해 Athena for Spark에서 CSV 또는 JSON 데이터로 테이블을 생성하는 경우 Athena SQL에서 테이블을 읽지 못하거나 그 반대의 경우가 발생할 수 있습니다.

예를 들어 다음 방법 중 하나를 사용하여 Athena for Spark에서 테이블을 생성할 수 있습니다.

- 다음과 같은 `USING csv` 구문을 사용합니다.

```
spark.sql('''CREATE EXTERNAL TABLE $tableName (
$colName1 $colType1,
$colName2 $colType2,
$colName3 $colType3)
USING csv
PARTITIONED BY ($colName1)
LOCATION $s3_location''')
```

- 다음과 같은 DataFrame API 구문을 사용합니다.

```
df.write.format('csv').saveAsTable($table_name)
```

오픈 소스 Spark의 알려진 문제로 인해 결과 테이블에 대한 Athena SQL의 쿼리에 실패할 수 있습니다.

제안 솔루션 - Athena for Spark에서 Apache Hive 구문을 사용하여 테이블을 생성하세요. 자세한 내용은 Apache Spark 설명서의 CREATE HIVEFORMAT TABLE을 참조하세요.

Athena for Spark 문제 해결

다음 정보를 사용하여 Athena에서 노트북 및 세션을 사용할 때 발생할 수 있는 문제를 해결합니다.

주제

- Spark 지원 작업 그룹 문제 해결

- Spark EXPLAIN 문을 사용하여 Spark SQL 문제 해결
- Athena에서 Spark 애플리케이션 이벤트 로깅
- CloudTrail을 사용하여 Athena 노트북 API 호출 문제 해결
- 68k 코드 블록 크기 문제 해결ㅊ
- 세션 문제 해결
- 테이블 문제 해결
- 지원 받기

Spark 지원 작업 그룹 문제 해결

다음 정보를 사용하여 Athena에서 Spark 지원 작업 그룹 문제를 해결합니다.

기존 IAM 역할을 사용할 때 세션이 응답하지 않음

Spark 지원 작업 그룹에 대해 새 `AWSAthenaSparkExecutionRole`을 생성하지 않고 기존 IAM 역할을 업데이트하거나 선택한 경우 세션에서 응답을 중지할 수 있습니다. 이 경우 Spark 지원 작업 그룹 실행 역할에 다음과 같은 신뢰 및 권한 정책을 추가해야 할 수 있습니다.

다음 신뢰 정책 예시를 추가합니다. 정책에는 실행 역할에 대한 혼동된 대리자 검사가 포함됩니다. *111122223333*, *aws-region* 및 *workgroup-name* 값을 사용 중인 AWS 계정 ID, AWS 리전 및 작업 그룹으로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "athena.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:athena:aws-
region:111122223333:workgroup/workgroup-name"
                }
```

```
            }
        }
    ]
}
```

노트북 지원 작업 그룹에 대해 다음 기본 정책과 같은 권한 정책을 추가합니다. 자리 표시자 Amazon S3 위치 및 AWS 계정 ID를 사용 중인 해당 항목과 일치하도록 수정합니다. `DOC-EXAMPLE-BUCKET`, *`aws-region`*, *`111122223333`* 및 *`workgroup-name`* 값을 사용 중인 Amazon S3 버킷, AWS 리전, AWS 계정 ID 및 작업 그룹으로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:ListBucket",
                "s3:DeleteObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",
                "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "athena:GetWorkGroup",
                "athena:CreatePresignedNotebookUrl",
                "athena:TerminateSession",
                "athena:GetSession",
                "athena:GetSessionStatus",
                "athena:ListSessions",
                "athena:StartCalculationExecution",
                "athena:GetCalculationExecutionCode",
                "athena:StopCalculationExecution",
                "athena:ListCalculationExecutions",
                "athena:GetCalculationExecution",
                "athena:GetCalculationExecutionStatus",
                "athena:ListExecutors",
                "athena:ExportNotebook",
```

```
                "athena:UpdateNotebook"
            ],
            "Resource": "arn:aws:athena:aws-region:111122223333:workgroup/workgroup-
name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:CreateLogGroup",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:aws-region:111122223333:log-group:/aws-athena:*",
                "arn:aws:logs:aws-region:111122223333:log-group:/aws-athena*:log-
stream:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "logs:DescribeLogGroups",
            "Resource": "arn:aws:logs:aws-region:111122223333:log-group:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "AmazonAthenaForApacheSpark"
                }
            }
        }
    ]
}
```

Spark EXPLAIN 문을 사용하여 Spark SQL 문제 해결

Spark EXPLAIN 문을 Spark SQL과 함께 사용하여 Spark 코드 문제를 해결할 수 있습니다. 다음 코드 및 출력 예는 이 사용법을 보여줍니다.

Example – Spark SELECT 문

```
spark.sql("select * from select_taxi_table").explain(True)
```

출력

```
Calculation started (calculation_id=20c1ebd0-1ccf-ef14-db35-7c1844876a7e) in
(session=24c1ebcb-57a8-861e-1023-736f5ae55386).
Checking calculation status...

Calculation completed.
== Parsed Logical Plan ==
'Project [*]
+- 'UnresolvedRelation [select_taxi_table], [], false

== Analyzed Logical Plan ==
VendorID: bigint, passenger_count: bigint, count: bigint
Project [VendorID#202L, passenger_count#203L, count#204L]
+- SubqueryAlias spark_catalog.spark_demo_database.select_taxi_table
   +- Relation spark_demo_database.select_taxi_table[VendorID#202L,
       passenger_count#203L,count#204L] csv

== Optimized Logical Plan ==
Relation spark_demo_database.select_taxi_table[VendorID#202L,
passenger_count#203L,count#204L] csv

== Physical Plan ==
FileScan csv spark_demo_database.select_taxi_table[VendorID#202L,
passenger_count#203L,count#204L]
Batched: false, DataFilters: [], Format: CSV,
Location: InMemoryFileIndex(1 paths)
[s3://DOC-EXAMPLE-BUCKET/select_taxi],
PartitionFilters: [], PushedFilters: [],
ReadSchema: struct<VendorID:bigint,passenger_count:bigint,count:bigint>
```

Example – Spark 데이터 프레임

다음 코드 예는 Spark 데이터 프레임과 함께 EXPLAIN을 사용하는 방법을 보여줍니다.

```
taxi1_df=taxi_df.groupBy("VendorID", "passenger_count").count()
taxi1_df.explain("extended")
```

출력

```
Calculation started (calculation_id=d2c1ebd1-f9f0-db25-8477-3effc001b309) in
(session=24c1ebcb-57a8-861e-1023-736f5ae55386).
Checking calculation status...

Calculation completed.
== Parsed Logical Plan ==
'Aggregate ['VendorID, 'passenger_count],
['VendorID, 'passenger_count, count(1) AS count#321L]
+- Relation [VendorID#49L,tpep_pickup_datetime#50,tpep_dropoff_datetime#51,
passenger_count#52L,trip_distance#53,RatecodeID#54L,store_and_fwd_flag#55,
PULocationID#56L,DOLocationID#57L,payment_type#58L,fare_amount#59,
extra#60,mta_tax#61,tip_amount#62,tolls_amount#63,improvement_surcharge#64,
total_amount#65,congestion_surcharge#66,airport_fee#67] parquet

== Analyzed Logical Plan ==
VendorID: bigint, passenger_count: bigint, count: bigint
Aggregate [VendorID#49L, passenger_count#52L],
[VendorID#49L, passenger_count#52L, count(1) AS count#321L]
+- Relation [VendorID#49L,tpep_pickup_datetime#50,tpep_dropoff_datetime#51,
passenger_count#52L,trip_distance#53,RatecodeID#54L,store_and_fwd_flag#55,
PULocationID#56L,DOLocationID#57L,payment_type#58L,fare_amount#59,extra#60,
mta_tax#61,tip_amount#62,tolls_amount#63,improvement_surcharge#64,
total_amount#65,congestion_surcharge#66,airport_fee#67] parquet

== Optimized Logical Plan ==
Aggregate [VendorID#49L, passenger_count#52L],
[VendorID#49L, passenger_count#52L, count(1) AS count#321L]
+- Project [VendorID#49L, passenger_count#52L]
   +- Relation [VendorID#49L,tpep_pickup_datetime#50,tpep_dropoff_datetime#51,
passenger_count#52L,trip_distance#53,RatecodeID#54L,store_and_fwd_flag#55,
PULocationID#56L,DOLocationID#57L,payment_type#58L,fare_amount#59,extra#60,
mta_tax#61,tip_amount#62,tolls_amount#63,improvement_surcharge#64,
total_amount#65,congestion_surcharge#66,airport_fee#67] parquet

== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- HashAggregate(keys=[VendorID#49L, passenger_count#52L], functions=[count(1)],
output=[VendorID#49L, passenger_count#52L, count#321L])
   +- Exchange hashpartitioning(VendorID#49L, passenger_count#52L, 1000),
      ENSURE_REQUIREMENTS, [id=#531]
      +- HashAggregate(keys=[VendorID#49L, passenger_count#52L],
         functions=[partial_count(1)], output=[VendorID#49L,
```

```
        passenger_count#52L, count#326L])
        +- FileScan parquet [VendorID#49L,passenger_count#52L] Batched: true,
           DataFilters: [], Format: Parquet,
           Location: InMemoryFileIndex(1 paths)[s3://DOC-EXAMPLE-BUCKET/
           notebooks/yellow_tripdata_2016-01.parquet], PartitionFilters: [],
           PushedFilters: [],
           ReadSchema: struct<VendorID:bigint,passenger_count:bigint>
```

Athena에서 Spark 애플리케이션 이벤트 로깅

Athena 노트북 편집기에서는 표준 Jupyter, Spark 및 Python 로깅을 허용합니다. `df.show()`를 사용하여 PySpark DataFrame 콘텐츠를 표시하거나 `print("Output")`를 사용하여 셀 출력에 값을 표시할 수 있습니다. 계산에 대한 `stdout`, `stderr` 및 `results` 출력은 Amazon S3의 쿼리 결과 버킷 위치에 기록됩니다.

Amazon CloudWatch에 Spark 애플리케이션 이벤트 로깅

Athena 세션에서는 사용 중인 계정의 Amazon CloudWatch에 로그를 기록할 수도 있습니다.

로그 스트림 및 로그 그룹 이해

CloudWatch에서는 로그 활동을 로그 스트림과 로그 그룹으로 구성합니다.

로그 스트림 - CloudWatch 로그 스트림은 동일한 소스를 공유하는 일련의 로그 이벤트입니다. CloudWatch Logs에서 각 별도의 로그 소스가 별도의 로그 스트림을 구성합니다.

로그 그룹 - CloudWatch Logs에서 로그 그룹은 동일한 보존 기간, 모니터링 및 액세스 제어 설정을 공유하는 로그 스트림의 그룹입니다.

하나의 로그 그룹이 가질 수 있는 로그 스트림의 수는 제한이 없습니다.

Athena에서 노트북 세션을 처음으로 시작하면 다음 예제와 같이 Spark 지원 작업 그룹의 이름을 사용하는 로그 그룹이 Athena의 CloudWatch에서 생성됩니다.

```
/aws-athena/workgroup-name
```

이 로그 그룹은 하나 이상의 로그 이벤트를 생성하는 세션의 실행기별로 하나의 로그 스트림을 수신합니다. 실행기는 노트북 세션이 Athena에서 요청할 수 있는 가장 작은 컴퓨팅 단위입니다. CloudWatch에서 로그 스트림의 이름은 세션 ID와 실행기 ID로 시작합니다.

CloudWatch 로그 그룹 및 로그 스트림에 대한 자세한 내용은 Amazon CloudWatch Logs 사용 설명서의 로그 그룹 및 로그 스트림 작업을 참조하세요.

Athena for Spark에서 표준 로거 객체 사용

Athena for Spark 세션에서는 다음 두 글로벌 표준 로거 객체를 사용하여 Amazon CloudWatch에 로그를 기록할 수 있습니다.

- athena_user_logger - CloudWatch에만 로그를 전송합니다. 다음 예제와 같이 Spark 애플리케이션의 정보를 CloudWatch에 직접 기록하려는 경우 이 객체를 사용합니다.

  ```
  athena_user_logger.info("CloudWatch log line.")
  ```

  이 예에서는 다음과 같이 CloudWatch에 로그 이벤트를 기록합니다.

  ```
  AthenaForApacheSpark: 2022-01-01 12:00:00,000 INFO builtins: CloudWatch log line.
  ```

- athena_shared_logger - 지원을 위해 CloudWatch와 AWS 모두에 동일한 로그를 전송합니다. 다음 예제와 같이 이 객체를 사용하여 문제 해결을 위해 AWS 서비스 팀과 로그를 공유할 수 있습니다.

  ```
  athena_shared_logger.info("Customer debug line.")
  var = [...some variable holding customer data...]
  athena_shared_logger.info(var)
  ```

  이 예에서는 debug 행과 var 변수 값을 CloudWatch Logs에 기록하고 각 행의 사본을 AWS Support로 보냅니다.

  > **Note**
  >
  > 개인 정보 보호를 위해 계산 코드와 결과는 AWS와 공유되지 않습니다. athena_shared_logger를 호출하여 AWS Support에 표시하려는 정보만 기록해야 합니다.

제공된 로거는 Apache Log4j를 통해 이벤트를 기록하고 이 인터페이스의 로깅 수준을 상속합니다. 가능한 로그 수준 값은 DEBUG, ERROR, FATAL, INFO, WARN 또는 WARNING입니다. 로거에서 해당하는 명명된 함수를 사용하여 이러한 값을 생성할 수 있습니다.

> **Note**
>
> `athena_user_logger` 또는 `athena_shared_logger` 이름을 다시 바인딩하지 마세요. 그러면 나머지 세션 동안 로깅 객체가 CloudWatch에 쓸 수 없습니다.

예: CloudWatch에 노트북 이벤트 로깅

다음 절차는 Amazon CloudWatch Logs에 Athena 노트북 이벤트를 로깅하는 방법을 보여줍니다.

Amazon CloudWatch Logs에 Athena 노트북 이벤트를 로깅하려면

1. Amazon Athena에서 Apache Spark 시작하기에 따라 Athena에서 고유한 이름을 가진 Spark 지원 작업 그룹을 생성합니다. 이 자습서에서는 `athena-spark-example` 작업 그룹 이름을 사용합니다.
2. 사용자 노트북 생성의 단계에 따라 노트북을 생성하고 새 세션을 시작합니다.
3. Athena 노트북 편집기의 새 노트북 셀에 다음 명령을 입력합니다.

   ```
   athena_user_logger.info("Hello world.")
   ```
4. 셀을 실행합니다.
5. 다음 중 하나를 수행하여 현재 세션 ID를 검색합니다.

   - 셀 출력(예: `... session=72c24e73-2c24-8b22-14bd-443bdcd72de4`)을 확인합니다.
   - 새 셀에서 매직 명령 `%session_id`를 실행합니다.
6. 세션 ID를 저장합니다.
7. 노트북 세션을 실행하는 데 사용 중인 것과 동일한 AWS 계정를 사용하여 https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
8. CloudWatch 콘솔 탐색 창에서 Log groups(로그 그룹)를 선택합니다.
9. 로그 그룹 목록에서 다음 예와 같이 Spark 지원 Athena 작업 그룹의 이름이 있는 로그 그룹을 선택합니다.

   ```
   /aws-athena/athena-spark-example
   ```

   Log streams(로그 스트림) 섹션에는 작업 그룹에 대한 하나 이상의 로그 스트림 링크 목록이 포함되어 있습니다. 각 로그 스트림 이름에서는 세션 ID, 실행기 ID 및 고유 UUID가 슬래시 문자로 구분되어 있습니다.

예를 들어 세션 ID가 5ac22d11-9fd8-ded7-6542-0412133d3177이고 실행기 ID가 f8c22d11-9fd8-ab13-8aba-c4100bfba7e2인 경우 로그 스트림의 이름은 다음 예와 비슷합니다.

```
5ac22d11-9fd8-ded7-6542-0412133d3177/f8c22d11-9fd8-ab13-8aba-c4100bfba7e2/f012d7cb-cefd-40b1-90b9-67358f003d0b
```

10. 세션에 대한 로그 스트림 링크를 선택합니다.
11. Log events(로그 이벤트) 페이지에서 Message(메시지) 열을 확인합니다.

    실행한 셀의 로그 이벤트는 다음과 유사합니다.

```
AthenaForApacheSpark: 2022-01-01 12:00:00,000 INFO builtins: Hello world.
```

12. Athena 노트북 편집기로 돌아갑니다.
13. 새 셀에 다음 코드를 입력합니다. 이 코드는 변수를 CloudWatch에 기록합니다.

```
x = 6
athena_user_logger.warn(x)
```

14. 셀을 실행합니다.
15. 동일한 로그 스트림에 대한 CloudWatch 콘솔 Log events(로그 이벤트) 페이지로 돌아갑니다.
16. 이제 로그 스트림에는 다음과 같은 메시지와 로그 이벤트 항목이 포함되어 있습니다.

```
AthenaForApacheSpark: 2022-01-01 12:00:00,000 WARN builtins: 6
```

CloudTrail을 사용하여 Athena 노트북 API 호출 문제 해결

노트북 API 호출 문제를 해결하려면 Athena CloudTrail 로그를 검사하여 이상 동작을 조사하거나 사용자가 시작한 작업을 검색할 수 있습니다. Athena에서 CloudTrail 사용 방법에 대한 자세한 내용은 AWS CloudTrail을 사용하여 Amazon Athena API 호출 로깅 단원을 참조하세요.

다음 예는 Athena 노트북 API에 대한 CloudTrail 로그 항목을 보여줍니다.

- StartSession
- TerminateSession
- ImportNotebook

- UpdateNotebook
- StartCalculationExecution

StartSession

다음 예는 노트북 StartSession 이벤트에 대한 CloudTrail 로그를 보여줍니다.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID:alias",
        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/alias",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-14T16:41:51Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-14T17:05:36Z",
    "eventSource": "athena.amazonaws.com",
    "eventName": "StartSession",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.10",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36",
    "requestParameters": {
        "workGroup": "notebook-workgroup",
        "engineConfiguration": {
            "coordinatorDpuSize": 1,
            "maxConcurrentDpus": 20,
```

```
            "defaultExecutorDpuSize": 1,
            "additionalConfigs": {
                "NotebookId": "b8f5854b-1042-4b90-9d82-51d3c2fd5c04",
                "NotebookIframeParentUrl": "https://us-east-1.console.aws.amazon.com"
            }
        },
        "notebookVersion": "KeplerJupyter-1.x",
        "sessionIdleTimeoutInMinutes": 20,
        "clientRequestToken": "d646ff46-32d2-42f0-94d1-d060ec3e5d78"
    },
    "responseElements": {
        "sessionId": "a2c1ebba-ad01-865f-ed2d-a142b7451f7e",
        "state": "CREATED"
    },
    "requestID": "d646ff46-32d2-42f0-94d1-d060ec3e5d78",
    "eventID": "b58ce998-eb89-43e9-8d67-d3d8e30561c9",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "athena.us-east-1.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```


TerminateSession

다음 예는 노트북 TerminateSession 이벤트에 대한 CloudTrail 로그를 보여줍니다.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID:alias",
        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/alias",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
            "sessionIssuer": {
```

```
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-14T16:41:51Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-14T17:21:03Z",
    "eventSource": "athena.amazonaws.com",
    "eventName": "TerminateSession",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.11",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36",
    "requestParameters": {
        "sessionId": "a2c1ebba-ad01-865f-ed2d-a142b7451f7e"
    },
    "responseElements": {
        "state": "TERMINATING"
    },
    "requestID": "438ea37e-b704-4cb3-9a76-391997cf42ee",
    "eventID": "49026c5a-bf58-4cdb-86ca-978e711ad238",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "athena.us-east-1.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```

ImportNotebook

다음 예는 노트북 ImportNotebook 이벤트에 대한 CloudTrail 로그를 보여줍니다. 보안을 위해 일부 콘텐츠는 숨겨져 있습니다.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID:alias",
        "arn": "arn:aws:sts::123456789012:assumed-role/Admin/alias",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/Admin",
                "accountId": "123456789012",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-14T16:41:51Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-14T17:08:54Z",
    "eventSource": "athena.amazonaws.com",
    "eventName": "ImportNotebook",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36",
    "requestParameters": {
        "workGroup": "notebook-workgroup",
        "name": "example-notebook-name",
        "payload": "HIDDEN_FOR_SECURITY_REASONS",
        "type": "IPYNB",
        "contentMD5": "HIDDEN_FOR_SECURITY_REASONS"
    },
    "responseElements": {
```

```
        "notebookId": "05f6225d-bdcc-4935-bc25-a8e19434652d"
    },
    "requestID": "813e777f-6dac-41f4-82a7-e99b7b33f319",
    "eventID": "4abec837-143b-4458-9c1f-fa9fb88ab69b",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "athena.us-east-1.amazonaws.com"
    },
    "sessionCredentialFromConsole": "true"
}
```


UpdateNotebook

다음 예는 노트북 UpdateNotebook 이벤트에 대한 CloudTrail 로그를 보여줍니다. 보안을 위해 일부 콘텐츠는 숨겨져 있습니다.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "EXAMPLE_PRINCIPAL_ID:AthenaExecutor-9cc1ebb2-aac5-
b1ca-8247-5d827bd8232f",
        "arn": "arn:aws:sts::123456789012:assumed-role/AWSAthenaSparkExecutionRole-
om0yj71w5l/AthenaExecutor-9cc1ebb2-aac5-b1ca-8247-5d827bd8232f",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/service-role/
AWSAthenaSparkExecutionRole-om0yj71w5l",
                "accountId": "123456789012",
                "userName": "AWSAthenaSparkExecutionRole-om0yj71w5l"
            },
            "webIdFederationData": {},
            "attributes": {
```

```
                "creationDate": "2022-10-14T16:48:06Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-14T16:52:22Z",
    "eventSource": "athena.amazonaws.com",
    "eventName": "UpdateNotebook",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.13",
    "userAgent": "Boto3/1.24.84 Python/3.8.14 Linux/4.14.225-175.364.amzn2.aarch64
 Botocore/1.27.84",
    "requestParameters": {
        "notebookId": "c87553ff-e740-44b5-884f-a70e575e08b9",
        "payload": "HIDDEN_FOR_SECURITY_REASONS",
        "type": "IPYNB",
        "contentMD5": "HIDDEN_FOR_SECURITY_REASONS",
        "sessionId": "9cc1ebb2-aac5-b1ca-8247-5d827bd8232f"
    },
    "responseElements": null,
    "requestID": "baaba1d2-f73d-4df1-a82b-71501e7374f1",
    "eventID": "745cdd6f-645d-4250-8831-d0ffd2fe3847",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "athena.us-east-1.amazonaws.com"
    }
}
```


StartCalculationExecution

다음 예는 노트북 StartCalculationExecution 이벤트에 대한 CloudTrail 로그를 보여줍니다. 보안을 위해 일부 콘텐츠는 숨겨져 있습니다.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
```

```
        "principalId": "EXAMPLE_PRINCIPAL_ID:AthenaExecutor-9cc1ebb2-aac5-
b1ca-8247-5d827bd8232f",
        "arn": "arn:aws:sts::123456789012:assumed-role/AWSAthenaSparkExecutionRole-
om0yj71w5l/AthenaExecutor-9cc1ebb2-aac5-b1ca-8247-5d827bd8232f",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "EXAMPLE_PRINCIPAL_ID",
                "arn": "arn:aws:iam::123456789012:role/service-role/
AWSAthenaSparkExecutionRole-om0yj71w5l",
                "accountId": "123456789012",
                "userName": "AWSAthenaSparkExecutionRole-om0yj71w5l"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-10-14T16:48:06Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-10-14T16:52:37Z",
    "eventSource": "athena.amazonaws.com",
    "eventName": "StartCalculationExecution",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.14",
    "userAgent": "Boto3/1.24.84 Python/3.8.14 Linux/4.14.225-175.364.amzn2.aarch64
 Botocore/1.27.84",
    "requestParameters": {
        "sessionId": "9cc1ebb2-aac5-b1ca-8247-5d827bd8232f",
        "description": "Calculation started via Jupyter notebook",
        "codeBlock": "HIDDEN_FOR_SECURITY_REASONS",
        "clientRequestToken": "0111cd63-4fd0-4ad8-a738-fd350115fc21"
    },
    "responseElements": {
        "calculationExecutionId": "82c1ebb4-bd08-e4c3-5631-a662fb2ff2c5",
        "state": "CREATING"
    },
    "requestID": "1a107461-3f1b-481e-b8a2-7fbd524e2373",
    "eventID": "b74dbd00-e839-4bd1-a1da-b75fbc70ab9a",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
```

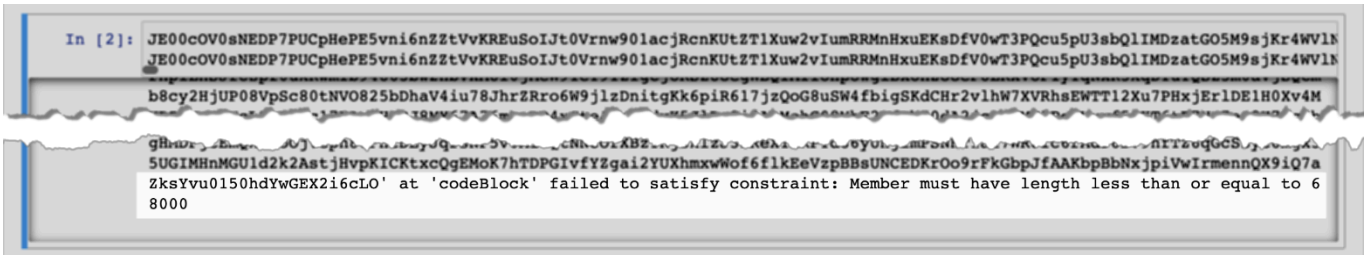
```
    "recipientAccountId": "123456789012",
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "athena.us-east-1.amazonaws.com"
    }
}
```

68k 코드 블록 크기 문제 해결ㅊ

Athena for Spark에서 알려진 계산 코드 블록 크기 한도는 6만 8,000자입니다. 이 한도를 초과하는 코드 블록을 사용하여 계산을 실행하면 다음과 같은 오류 메시지가 나타날 수 있습니다.

'...' at 'codeBlock' failed to satisfy constraint: Member must have length less than or equal to 68000

다음 이미지는 Athena 콘솔 노트북 편집기에서 발생한 이 오류를 보여줍니다.



다음 예제와 같이 코드 블록이 큰 계산을 실행하기 위해 AWS CLI를 사용할 때도 같은 오류가 발생할 수 있습니다.

```
aws athena start-calculation-execution \
    --session-id "{SESSION_ID}" \
    --description "{SESSION_DESCRIPTION}" \
    --code-block "{LARGE_CODE_BLOCK}"
```

이 명령은 다음과 같은 오류 메시지를 표시합니다.

{LARGE_CODE_BLOCK} at 'codeBlock' failed to satisfy constraint: Member must have length less than or equal to 68000

차선책

이 문제를 해결하려면 쿼리 또는 계산 코드가 있는 파일을 Amazon S3에 업로드합니다. 그런 다음 boto3를 사용하여 파일을 읽고 SQL 또는 코드를 실행합니다.

다음 예제에서는 SQL 쿼리 또는 Python 코드가 있는 파일을 Amazon S3에 이미 업로드했다고 가정합니다.

SQL 예제

다음 코드 예제는 Amazon S3 버킷에서 `large_sql_query.sql` 파일을 읽고 파일에 포함된 대형 쿼리를 실행합니다.

```
s3 = boto3.resource('s3')
def read_s3_content(bucket_name, key):
    response = s3.Object(bucket_name, key).get()
    return response['Body'].read()

# SQL
sql = read_s3_content('bucket_name', 'large_sql_query.sql')
df = spark.sql(sql)
```

PySpark 예제

다음 코드 예제는 Amazon S3에서 `large_py_spark.py` 파일을 읽고 파일에 있는 대형 코드 블록을 실행합니다.

```
s3 = boto3.resource('s3')

def read_s3_content(bucket_name, key):
    response = s3.Object(bucket_name, key).get()
    return response['Body'].read()

# PySpark
py_spark_code = read_s3_content('bucket_name', 'large_py_spark.py')
exec(py_spark_code)
```

세션 문제 해결

이 항목의 정보를 사용하여 세션 문제를 해결합니다.

비정상 상태 세션

Session in unhealthy state.(세션이 비정상 상태입니다) Please create a new session(새 세션을 생성하세요)이라는 오류 메시지가 표시되는 경우 기존 세션을 종료하고 새 세션을 생성합니다.

노트북 서버에 연결할 수 없습니다.

노트북을 열 때 다음 오류 메시지가 표시될 수 있습니다.

```
A connection to the notebook server could not be established.
The notebook will continue trying to reconnect.
Check your network connection or notebook server configuration.
```

원인

Athena는 노트북을 열 때 세션을 생성하고 미리 서명된 노트북 URL을 사용하여 노트북에 연결합니다. 노트북 연결에서는 WSS(WebSocket Secure) 프로토콜을 사용합니다.

이 오류는 다음과 같은 이유로 발생할 수 있습니다.

- 로컬 방화벽(예: 회사 전체 방화벽)이 WSS 트래픽을 차단하고 있습니다.
- 로컬 컴퓨터의 프록시 또는 바이러스 백신 소프트웨어가 WSS 연결을 차단하고 있습니다.

Solution

`us-east-1` 리전에 다음과 같은 WSS 연결이 있다고 가정합니다.

```
wss://94c2bcdf-66f9-4d17-9da6-7e7338060183.analytics-gateway.us-east-1.amazonaws.com/
api/kernels/33c78c82-b8d2-4631-bd22-1565dc6ec152/channels?session_id=
7f96a3a048ab4917b6376895ea8d7535
```

오류를 해결하려면 다음 전략 중 하나를 사용합니다.

- 와일드카드 패턴 구문을 사용하여 AWS 리전 및 AWS 계정에서 포트 443의 목록 WSS 트래픽을 허용합니다.

  ```
  wss://*amazonaws.com
  ```

- 와일드카드 패턴 구문을 사용하여 한 AWS 리전의 포트 443 및 지정된 AWS 리전의 AWS 계정에서 목록 WSS 트래픽을 허용합니다. 다음 예에는 `us-east-1`가 사용됩니다.

  ```
  wss://*analytics-gateway.us-east-1.amazonaws.com
  ```

테이블 문제 해결

테이블을 생성할 때 경로 오류를 생성할 수 없습니다.

오류 메시지: IllegalArgumentException: 빈 문자열에서 경로를 생성할 수 없습니다.

원인: 이 오류는 Athena에서 Apache Spark를 사용하여 AWS Glue 데이터베이스에서 테이블을 생성하고, 데이터베이스에 빈 `LOCATION` 속성이 있는 경우에 발생할 수 있습니다.

권장 솔루션: 자세한 내용 및 해결 방법은 테이블을 생성할 때 잘못된 인수 예외 발생 단원을 참조하세요.

AWS Glue 테이블을 쿼리할 때 AccessDeniedException 예외 발생

오류 메시지: pyspark.sql.utils.AnalysisException: Unable to verify existence of default database: com.amazonaws.services.glue.model.AccessDeniedException: User: arn:aws:sts::*aws-account-id*:assumed-role/AWSAthenaSparkExecutionRole-*unique-identifier*/AthenaExecutor-*unique-identifier* is not authorized to perform: glue:GetDatabase on resource: arn:aws:glue:*aws-region*:*aws-account-id*:catalog because no identity-based policy allows the glue:GetDatabase action (Service: AWSGlue; Status Code: 400; Error Code: AccessDeniedException; Request ID: *request-id*; Proxy: null)

원인: Spark 지원 작업 그룹의 실행 역할에 AWS Glue 리소스에 액세스할 수 있는 권한이 없습니다.

권장 솔루션: 이 문제를 해결하려면 실행 역할에 AWS Glue 리소스에 대한 액세스 권한을 부여한 다음 실행 역할에 액세스 권한을 부여하도록 Amazon S3 버킷 정책을 편집합니다.

다음 절차에서는 이 단계를 자세히 설명합니다.

실행 역할에 AWS Glue 리소스에 대한 액세스 권한을 부여하려면

1. https://console.aws.amazon.com/athena/에서 Athena 콘솔을 엽니다.
2. 콘솔 탐색 창이 표시되지 않으면 왼쪽의 확장 메뉴를 선택합니다.

3. Athena 콘솔 탐색 창에서 작업 그룹(Workgroups)을 선택합니다.
4. 작업 그룹(Workgroups) 페이지에서 보려는 작업 그룹의 링크를 선택합니다.
5. 작업 그룹에 대한 Overview Details(개요 세부 정보) 페이지에서 Role ARN(역할 ARN) 링크를 선택합니다. 링크를 클릭하면 IAM 콘솔에서 Simple 실행 역할이 열립니다.
6. Permissions policies(권한 정책) 섹션에서 연결된 역할 정책 이름을 선택합니다.
7. Edit policy(정책 편집)를 선택한 다음 JSON을 선택합니다.
8. 역할에 AWS Glue 액세스 권한을 추가합니다. 일반적으로 `glue:GetDatabase` 및 `glue:GetTable` 작업에 대한 권한을 추가합니다. IAM 역할 구성에 대한 자세한 내용은 IAM 사용 설명서의 IAM 자격 증명 권한 추가 및 제거를 참조하세요.
9. Review policy(정책 검토)를 선택한 다음 Save changes(변경 사항 저장)를 선택합니다.
10. 실행 역할에 액세스 권한을 부여하도록 Amazon S3 버킷 정책을 편집합니다. 버킷과 버킷 내 객체 모두에 대한 액세스 권한을 역할에 부여해야 합니다. 작업 단계는 Amazon Simple Storage Service 사용 설명서의 Amazon S3 콘솔을 사용하여 버킷 정책 추가 단원을 참조하세요.

지원 받기

AWS에서 지원을 받으려면 AWS Management Console에서 Support(지원), Support Center(지원 센터)를 선택합니다. 원활한 이용을 위해 다음 정보를 준비해 두시기 바랍니다.

- Athena 쿼리 ID
- 세션 ID
- 계산 ID

릴리스 정보

릴리스 날짜별로 Amazon Athena 기능, 개선 사항 및 버그 수정에 대해 설명합니다.

주제

2024년 Athena 릴리스 정보

2024년 4월 26일

2024년 4월 26일 게시

Athena에서 JDBC 드라이버 버전 3.2.0을 릴리스합니다. 드라이버의 해당 버전에 대한 자세한 내용은 Amazon Athena JDBC 3.x 릴리스 정보 단원을 참조하세요. JDBC 3.x 드라이버를 다운로드하려면 JDBC 3.x 드라이버 다운로드 섹션을 참조하세요.

2024년 4월 24일

2024년 4월 24일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

- Parquet-Athena는 이제 Parquet에서 목록 또는 맵 그룹에 포함되지 않은 주석이 없는 반복 프리미티브 필드에 대해 이전 버전과 호환되는 읽기를 지원합니다. 이 변경으로 인해 자동으로 잘못된 결과가 반환되지 않고 스키마 불일치에 대한 오류 메시지가 개선됩니다.

  자세한 내용은 GitHub.com의 Parquet에서 주석이 없는 반복 프리미티브 필드에 대한 이전 버전과 호환되는 읽기 지원을 참조하세요.

- Iceberg OPTIMIZE-파티션되지 않은 키 필터를 WHERE 절에서 사용하면 데이터가 손실되는 OPTIMIZE 쿼리 관련 문제를 해결했습니다. 자세한 내용은 OPTIMIZE 단원을 참조하십시오.

2024년 4월 16일

2024년 4월 16일 게시

새로운 Amazon Athena 페더레이션 쿼리 패스스루 기능을 사용하여 기본 데이터 소스에서 직접 전체 쿼리를 실행할 수 있습니다. 페더레이션 패스스루 쿼리를 사용하면 원본 데이터 소스의 고유한 함수, 쿼리 언어 및 성능 기능을 활용할 수 있습니다. 예를 들어 PartiQL 언어를 사용하여 DynamoDB에서 Athena 쿼리를 실행할 수 있습니다. 페더레이션 패스스루 쿼리는 Athena에서 사용할 수 없는 데이터 소스의 함수를 집계, 조인 또는 호출하는 SELECT 쿼리를 실행하려는 경우에도 유용합니다. 패스스루 쿼리를 사용하면 Athena에서 처리하는 데이터의 양을 줄이고 쿼리 시간을 단축할 수 있습니다.

자세한 내용은 페더레이션 패스스루 쿼리 실행 단원을 참조하십시오. 현재 사용하는 커넥터를 최신 버전으로 업그레이드하려면 데이터 소스 커넥터 업데이트 섹션을 참조하세요.

2024년 4월 10일

2024년 4월 10일 게시

Athena에서 다음 기능과 개선 사항을 발표했습니다.

ODBC 1.2.3.1000 드라이버

Athena용 ODBC 1.2.3.1000 드라이버 릴리스.

해결된 문제:

- 프록시 서버 연결 문제 - 루트 인증서 없이 프록시 서버가 사용된 경우 커넥터가 연결을 설정하지 못했습니다.

자세한 내용을 알아보고 ODBC 1.x 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 Athena ODBC 1.x 드라이버 섹션을 참조하세요.

JDBC 2.1.5 드라이버

Athena용 JBDC 2.1.5 드라이버 릴리스.

업데이트 및 개선 사항:

- AWS Java SDK를 버전 1.12.687로 업데이트했습니다.
- 버전 2.16.0을 사용하도록 Jackson 라이브러리를 업데이트했습니다.
- 버전 1.3.14를 사용하도록 Logback 라이브러리를 업데이트했습니다.

자세한 내용을 알아보고 JDBC 2.x 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 Athena JDBC 2.x 드라이버 섹션을 참조하세요.

2024년 4월 8일

2024년 4월 8일 게시

Athena에서 ODBC 드라이버 버전 2.0.3.0을 발표합니다. 자세한 내용은 2.0.3.0 릴리스 정보를 참조하세요. 새 ODBC v2 드라이버를 다운로드하려면 ODBC 2.x 드라이버 다운로드 섹션을 참조하세요. 연결 정보는 Amazon Athena ODBC 2.x 섹션을 참조하세요.

2024년 3월 15일

2024년 3월 18일 게시

Amazon Athena에서 캐나다 서부(캘거리) 리전의 Athena SQL의 가용성을 발표합니다.

각 AWS 리전에서 사용할 수 있는 전체 AWS 서비스 목록은 AWS 리전별 서비스를 참조하세요.

2024년 2월 15일

2024년 2월 15일 게시

Athena에서 JDBC 드라이버 버전 3.1.0을 릴리스합니다.

Amazon Athena JDBC 드라이버 버전 3.1.0에는 Microsoft AD FS(Active Directory Federation Services) Windows 통합 인증 및 양식 기반 인증에 대한 지원이 추가됩니다. 3.1.0 릴리스에는 기타 사소한 개선 사항 및 버그 수정도 포함되어 있습니다.

JDBC v3 드라이버를 다운로드하려면 JDBC 3.x 드라이버 다운로드 섹션을 참조하세요.

2024년 1월 31일

2024년 1월 31일 게시

Athena에서 다음 기능과 개선 사항을 발표했습니다.

- Hudi 업그레이드 - 이제 Athena SQL을 사용하여 Hudi 0.14.0 테이블을 쿼리할 수 있습니다. Athena SQL을 사용하여 Hudi 테이블을 쿼리하는 방법에 대한 자세한 내용은 Athena를 사용하여 Apache Hudi 데이터 집합 쿼리 섹션을 참조하세요.

2023년 Athena 릴리스 정보

2023년 12월 14일

2023년 12월 14일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

Athena에서 JDBC 드라이버 버전 2.1.3을 출시했습니다. 이 드라이버는 다음 문제를 해결합니다.

- Spring Boot 및 Gradle 애플리케이션 로깅과의 충돌을 피하도록 로깅이 개선되었습니다.
- `executeBatch()` JDBC 메서드를 사용하여 레코드를 삽입하면 드라이버가 하나의 레코드만 잘못 삽입했습니다. Athena에서 쿼리의 일괄 실행을 지원하지 않으므로 이제 `executeBatch()`를 사용하면 드라이버가 오류를 보고합니다. 이 제한 사항을 해결하려면 루프에서 단일 쿼리를 제출하면 됩니다.

새 JDBC 드라이버, 릴리스 정보 및 문서를 다운로드하려면 Athena JDBC 2.x 드라이버 섹션을 참조하세요.

2023년 12월 9일

2023년 12월 9일 게시

Athena용 ODBC 1.2.1.1000 드라이버를 출시했습니다.

기능 및 개선 사항:

- RStudio 지원 업데이트 - 이제 ODBC 드라이버가 macOS에서 RStudio를 지원합니다.
- 단일 카탈로그 및 스키마 지원 - 이제 커넥터가 단일 카탈로그와 스키마를 반환할 수 있습니다. 자세한 내용은 다운로드 가능한 설치 및 구성 가이드를 참조하십시오.

해결된 문제:

- 준비된 문 - 열 단위 스키마를 사용하는 파라미터 배열이 포함된 준비된 문을 실행하면 커넥터가 잘못된 쿼리 결과를 반환했습니다.

- 열 크기 - `$file_modified_time` 시스템 열을 선택하면 커넥터가 잘못된 열 크기를 반환했습니다.
- SQLPreate - `SELECT` 쿼리에서 `SQLPrepare`와 관련된 파라미터를 바인딩하면 커넥터가 오류를 반환했습니다.

자세한 내용을 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 Athena ODBC 1.x 드라이버을 참조하세요.

2023년 12월 7일

2023년 12월 7일 게시

Athena에서 ODBC 드라이버 버전 2.0.2.1을 발표했습니다. 자세한 내용은 2.0.2.1 릴리스 정보를 참조하세요. 새 ODBC v2 드라이버를 다운로드하려면 ODBC 2.x 드라이버 다운로드 섹션을 참조하세요. 연결 정보는 Amazon Athena ODBC 2.x 섹션을 참조하세요.

2023년 12월 5일

2023년 12월 5일 게시

이제 AWS IAM Identity Center 인증 모드를 사용하는 Athena SQL 작업 그룹을 생성할 수 있습니다. 이러한 작업 그룹은 IAM Identity Center의 신뢰할 수 있는 ID 전파 기능을 지원합니다. 신뢰할 수 있는 ID 전파를 통해 Amazon Athena 및 Amazon EMR Studio와 같은 AWS 분석 서비스에서 ID를 사용할 수 있습니다.

자세한 내용은 IAM Identity Center 지원 Athena 작업 그룹 사용 단원을 참조하십시오.

2023년 11월 28일

2023년 11월 28일 게시

이제 빠른 쿼리 결과를 위해 Amazon S3 Express One Zone 스토리지 클래스의 데이터를 쿼리할 수 있습니다. S3 Express One Zone은 가장 자주 액세스하는 데이터와 지연 시간에 민감한 애플리케이션에 대해 일관되게 10밀리초 미만의 데이터 액세스를 제공하도록 특별히 설계된 고성능 단일 가용 영역 스토리지 클래스입니다. 시작하려면 Athena에서 원활하게 쿼리를 수행할 수 있도록 S3 Express One Zone 스토리지로 데이터를 이동하고 AWS Glue Data Catalog를 사용하여 데이터를 카탈로그화하십시오.

자세한 내용은 S3 Express One Zone 데이터 쿼리 단원을 참조하십시오.

2023년 11월 27일

2023년 11월 27일 게시

Athena에서 다음 기능과 개선 사항을 발표했습니다.

- Glue Data Catalog 뷰 - Glue Data Catalog 뷰는 Amazon Athena, Amazon Redshift 등의 AWS 서비스 전반에서 공통된 단일 뷰를 제공합니다. Glue Data Catalog 뷰에서 액세스 권한은 뷰를 쿼리하는 사용자 대신 뷰를 만든 사용자가 정의합니다. 이러한 뷰는 액세스 제어를 강화하고, 완전한 레코드를 보장하는 데 도움이 되며, 보안을 강화하고, 기본 테이블에 대한 액세스를 차단할 수 있습니다.

  자세한 내용은 AWS Glue Data Catalog 뷰 사용 단원을 참조하십시오.
- CloudTrail Lake 지원 – 이제 Amazon Athena를 사용하여 AWS CloudTrail Lake의 데이터를 분석할 수 있습니다. AWS CloudTrail Lake는 감사, 보안 및 운영 조사를 위해 활동 로그를 집계, 불변하게 저장 및 분석하는 데 사용할 수 있는 CloudTrail용 관리형 데이터 레이크입니다. Athena에서 CloudTrail Lake 활동 로그를 쿼리하기 위해 데이터를 이동하거나 별도의 데이터 처리 파이프라인을 구축할 필요가 없습니다. ETL 작업이 필요하지 않습니다.

  시작하려면 CloudTrail Lake에서 데이터 페더레이션을 활성화합니다. CloudTrail Lake 이벤트 데이터 스토어 메타데이터를 AWS Glue Data Catalog와 공유하면 CloudTrail은 필요한 AWS Glue Data Catalog 리소스를 생성하고 AWS Lake Formation에 데이터를 등록합니다. Lake Formation에서는 Athena를 사용하여 이벤트 데이터 스토어를 쿼리할 수 있는 사용자와 역할을 지정할 수 있습니다.

  자세한 내용은 AWS CloudTrail 사용 설명서의 Enable Lake query federation을 참조하십시오.

2023년 11월 17일

2023년 11월 17일 게시

Athena에서 다음 기능과 개선 사항을 발표했습니다.

특성

- 비용 기반 최적화 프로그램 - Athena는 AWS Glue의 통계를 사용하여 비용 기반 최적화의 일반 공급을 발표했습니다. Athena SQL에서 쿼리를 최적화하기 위해 AWS Glue의 테이블에 대해 Athena가 테이블 또는 열 수준의 통계를 수집하도록 요청할 수 있습니다. 쿼리의 모든 테이블에 통계가 있는 경우 Athena는 통계를 사용하여 대체 실행 계획을 검토하고 가장 빠른 실행 계획을 선택합니다.

  자세한 내용은 비용 기반 최적화 프로그램 사용 단원을 참조하십시오.

- Amazon EMR Studio 통합 - 이제 Athena 콘솔을 직접 사용하지 않고도 Amazon EMR Studio에서 Athena를 사용할 수 있습니다. Amazon EMR에 Athena 통합을 통해 다음 작업을 수행할 수 있습니다.
  - Athena SQL 쿼리 수행
  - 쿼리 결과 보기
  - 쿼리 기록 보기
  - 저장된 쿼리 보기
  - 파라미터화된 쿼리 수행
  - 데이터 카탈로그의 데이터베이스, 테이블 및 뷰 보기

  자세한 내용은 Amazon EMR Studio 주제에서 Athena와 AWS 서비스 통합 단원을 참조하세요.
- 중첩된 액세스 제어 - Athena에서 중첩된 데이터에 대한 Lake Formation 액세스 제어 지원을 발표했습니다. Lake Formation에서는 `struct` 데이터 유형이 있는 중첩 열에 데이터 필터를 정의하고 적용할 수 있습니다. 데이터 필터링을 사용하여 중첩된 열의 하위 구조에 대한 사용자 액세스를 제한할 수 있습니다. 중첩된 데이터에 대한 데이터 필터를 생성하는 방법에 대한 자세한 내용은 AWS Lake Formation 개발자 안내서의 Creating a data filter를 참조하세요.
- 프로비저닝된 용량 사용량 지표 - Athena에서 용량 예약에 대한 새로운 CloudWatch 지표를 발표했습니다. 새 지표를 사용하여 프로비저닝한 DPU 수와 쿼리에서 사용 중인 DPU 수를 추적할 수 있습니다. 쿼리가 끝나면 쿼리에서 소비한 DPU 수를 볼 수도 있습니다.

  자세한 내용은 CloudWatch 지표를 사용한 Athena 쿼리 모니터링 단원을 참조하십시오.

개선 사항

- 오류 메시지 변경 - 이제 `Insufficient Lake Formation permissions` 오류 메시지가 `Table not found` 또는 `Schema not found`로 표시됩니다. 이 변경은 악의적인 행위자가 오류 메시지에서 테이블 또는 데이터베이스 리소스의 존재를 유추하지 못하도록 하기 위한 것입니다.

2023년 11월 16일

2023월 11월 16일 게시

Athena에서 호환되는 SQL 개발 및 비즈니스 인텔리전스 애플리케이션의 데이터에 연결하고 쿼리하며 시각화하는 경험을 개선하는 새로운 JDBC 드라이버를 출시했습니다. 새 드라이버는 간단하게 업그레이드할 수 있습니다. 드라이버가 Amazon S3에서 쿼리 결과를 직접 읽을 수 있으므로 쿼리 결과를 더 빨리 사용할 수 있습니다.

자세한 내용은 Athena JDBC 3.x 드라이버 단원을 참조하십시오.

2023년 10월 31일

2023년 10월 31일 게시

Amazon Athena는 프로비저닝된 용량에 대해 1시간 예약을 발표했습니다. 오늘부터 프로비저닝된 용량을 예약하고 1시간 후에 릴리스할 수 있습니다. 이 변경을 통해 시간 경과에 따라 수요가 변하는 워크로드의 비용을 더욱 간단하게 최적화할 수 있습니다.

프로비저닝된 용량은 가장 중요한 대화형 워크로드의 우선순위를 지정하고 제어 및 확장할 수 있는 워크로드 관리 기능을 제공하는 Athena의 기능입니다. 언제든지 용량을 추가하여 동시에 실행할 수 있는 쿼리 수를 늘리고 용량을 사용할 수 있는 워크로드를 제어하며 워크로드 사이에서 용량을 공유할 수 있습니다.

자세한 내용은 쿼리 처리 용량 관리 단원을 참조하십시오. 요금에 대한 자세한 내용은 Amazon Athena 요금 페이지를 참조하세요.

2023년 10월 25일

2023년 10월 26일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

jackson-core 패키지 - 숫자 값이 1000자보다 큰 JSON 텍스트는 이제 실패합니다. 이 수정 사항은 보안 문제 sonatype-2022-6438을 해결합니다.

2023년 10월 17일

2023년 10월 17일 게시

Athena에서 ODBC 드라이버 버전 2.0.2.0을 발표했습니다. 자세한 내용은 2.0.2.0 릴리스 정보를 참조하세요. 새 ODBC v2 드라이버를 다운로드하려면 ODBC 2.x 드라이버 다운로드 섹션을 참조하세요. 연결 정보는 Amazon Athena ODBC 2.x 섹션을 참조하세요.

2023년 9월 26일

2023년 9월 26일 게시

Athena에서 다음 기능과 개선 사항을 발표했습니다.

- Delta Lake 테이블에 대한 Lake Formation 읽기 지원. Athena에서 Delta Lake 테이블을 사용하는 방법에 대한 자세한 내용은 Linux Foundation Delta Lake 테이블 쿼리 섹션을 참조하세요.

2023년 8월 23일

2023년 8월 23일 게시

Amazon Athena에서 이스라엘(텔아비브) 리전의 Athena SQL의 가용성을 발표했습니다.

각 AWS 리전에서 사용할 수 있는 전체 AWS 서비스 목록은 AWS 리전별 서비스를 참조하세요.

2023년 8월 10일

2023년 8월 10일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

ODBC 드라이버 버전 2.0.1.1

Athena에서 ODBC 드라이버 버전 2.0.1.1을 발표했습니다. 자세한 내용은 2.0.1.1 릴리스 정보를 참조하세요. 새 ODBC v2 드라이버를 다운로드하려면 ODBC 2.x 드라이버 다운로드 섹션을 참조하세요. 연결 정보는 Amazon Athena ODBC 2.x 섹션을 참조하세요.

JDBC 드라이버 버전 2.1.1

Athena에서 JDBC 드라이버 버전 2.1.1을 출시했습니다. 이 드라이버는 다음 문제를 해결합니다.

- 정규식이 포함된 명령문을 사용하여 테이블을 생성할 때 발생한 오류.
- `ApplicationName` 연결 파라미터를 잘못 적용하여 발생한 문제.

새 JDBC 드라이버, 릴리스 정보 및 문서를 다운로드하려면 JDBC로 Amazon Athena에 연결 섹션을 참조하세요.

2023년 7월 31일

2023년 7월 31일 게시

Amazon Athena는 추가로 AWS 리전에서 Athena SQL의 가용성을 발표했습니다.

이 릴리스에서는 Athena SQL의 가용성을 아시아 태평양(하이데라바드), 아시아 태평양(멜버른), 유럽(스페인) 및 유럽(취리히)으로 확대합니다.

각 AWS 리전에서 사용할 수 있는 전체 AWS 서비스 목록은 AWS 리전별 서비스를 참조하세요.

2023년 7월 27일

2023년 7월 27일 게시

Athena에서 Google BigQuery 커넥터 버전 2023.30.1을 출시했습니다. 이 커넥터 버전은 쿼리 실행 시간을 줄이고 BigQuery 프라이빗 엔드포인트에서 쿼리 지원을 추가했습니다.

Google BigQuery 커넥터에 대한 자세한 내용은 Amazon Athena Google BigQuery 커넥터 섹션을 참조하세요. 기존 데이터 소스 커넥터 업데이트에 대한 자세한 내용은 데이터 소스 커넥터 업데이트 섹션을 참조하세요.

2023년 7월 24일

2023년 7월 24일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

- 집합을 사용한 쿼리 - 집합을 사용하는 특정 쿼리의 성능이 개선되었습니다.
- 유형 비교를 사용한 조인 - 서로 다른 두 유형 간 비교를 포함하는 `JOIN` 문의 잠재적 쿼리 실패를 수정했습니다.
- 중첩된 열의 하위 쿼리 - 중첩된 열에서 하위 쿼리가 상호 연관될 때 발생하는 쿼리 실패와 관련된 문제를 수정했습니다.
- Iceberg 보기 - Apache Iceberg 보기에서 타임스탬프 열 정밀도와 관련된 호환성 문제를 수정했습니다. 이제 타임스탬프 열이 있는 Iceberg 보기는 열이 생성된 위치(Athena 엔진 버전 2 또는 Athena 엔진 버전 3)에 관계없이 읽을 수 있습니다.

2023년 7월 20일

2023년 7월 20일 게시

Athena에서 JDBC 드라이버 버전 2.1.0을 출시했습니다. 드라이버는 새로운 개선 사항을 포함하며 문제도 해결되었습니다.

개선 사항

다음과 같은 Jackson JSON 구문 분석기 라이브러리가 업그레이드되었습니다.

- jackson-annotations 2.15.2(이전의 2.14.0)
- jackson-core 2.15.2(이전의 2.14.0)
- jackson-databind 2.15.2(이전의 2.14.0)

해결된 문제

- sql2o 라이브러리를 사용할 때 배열 파라미터 전달과 관련된 문제를 수정했습니다.

자세한 내용을 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결을 참조하세요.

2023년 7월 13일

2023년 9월 19일 게시

Athena에서 다음 기능과 개선 사항을 발표했습니다.

- EXPLAIN ANALYZE - `EXPLAIN ANALYZE` 출력에 대기열, 분석, 계획 및 실행 시간에 대한 지원을 추가했습니다.
- EXPLAIN - 이제 쿼리에 집계가 포함된 경우 `EXPLAIN` 출력에서 통계를 표시합니다.
- Parquet Hive SerDe - Parquet 데이터를 읽을 때 처리 통계를 무시할 수 있는 `parquet.ignore.statistics` 속성을 추가했습니다. 자세한 설명은 Parquet 통계 무시을 참조하세요.

`EXPLAIN` 및 `EXPLAIN ANALYZE`에 대한 자세한 내용은 Athena에서 EXPLAIN 및 EXPLAIN ANALYZE 사용섹션을 참조하세요. Parquet Hive SerDe에 대한 자세한 내용은 Parquet SerDe 섹션을 참조하세요.

2023년 7월 3일

2023년 7월 25일 게시

2023년 7월 3일을 기점으로 Athena는 CloudTrail 로그에서 쿼리 문자열을 수정합니다. 이제 쿼리 문자열의 값은 `***OMITTED***`입니다. 민감한 정보를 포함할 수 있는 테이블 이름이나 필터 값이 의도치 않게 공개되지 않도록 하기 위해 이와 같이 변경되었습니다. 이전에 CloudTrail 로그를 사용하여 전체 쿼리 문자열에 액세스했다면 지금은 `Athena::GetQueryExecution` API를 사용하여 CloudTrail 로

그에서 `responseElements.queryExecutionId`의 값을 전달하는 것이 좋습니다. 자세한 내용은 Amazon Athena API 참조의 GetQueryExecution 작업을 참조하세요.

2023년 6월 30일

2023년 6월 30일 게시

이제 Athena 쿼리 편집기에서는 더 빠른 쿼리 작성 경험을 위해 코드 미리 입력하기 제안을 지원합니다. 이제 다음 기능을 사용하여 정확성과 효율성이 향상된 SQL 쿼리를 작성할 수 있습니다.

- 입력할 때 키워드, 로컬 변수, 스니펫 및 카탈로그 항목에 대한 제안이 실시간으로 표시됩니다.
- 데이터베이스 이름이나 테이블 이름을 입력하고 점을 입력하면 편집기에 선택할 수 있는 테이블 또는 열 목록이 편리하게 표시됩니다.
- 스니펫 제안에 마우스를 대면 스니펫의 구문 및 사용법에 대한 간략한 개요가 시놉시스에 표시됩니다.
- 코드 가독성을 높이기 위해 키워드와 강조 표시 규칙도 Trino 및 Hive의 최신 구문에 맞게 업데이트되었습니다.

이 기능은 기본적으로 활성화되어 있습니다. 코드 편집기 기본 설정에서 이 기능을 활성화하거나 비활성화할 수 있습니다.

Athena 쿼리 편집기에서 코드 미리 입력하기 제안을 사용해보려면 Athena 콘솔(https://console.aws.amazon.com/athena/)을 방문하세요.

2023년 6월 29일

2023년 6월 29일 게시

- Athena에서 ODBC 드라이버 버전 2.0.1.0을 발표했습니다. 자세한 내용은 2.0.1.0 릴리스 정보를 참조하세요. 새 ODBC v2 드라이버를 다운로드하려면 ODBC 2.x 드라이버 다운로드 섹션을 참조하세요. 연결 정보는 Amazon Athena ODBC 2.x 섹션을 참조하세요.
- Athena와 이 기능은 이제 중동(UAE) 리전에서 사용할 수 있습니다. 각 AWS 리전에서 사용할 수 있는 전체 AWS 서비스 목록은 AWS 리전별 서비스를 참조하세요.

2023년 6월 28일

2023년 6월 28일 게시

이제 Amazon Athena를 사용하여 S3 Glacier Flexible Retrieval(이전의 Glacier) 및 S3 Glacier Deep Archive Amazon S3 스토리지 클래스로부터 복원된 객체를 쿼리할 수 있습니다. 이 기능을 테이블별로 구성합니다. 이 기능은 Athena 엔진 버전 3의 Apache Hive 테이블에서만 지원됩니다.

자세한 내용은 복원된 Amazon S3 Glacier 객체 쿼리 단원을 참조하십시오.

2023년 6월 12일

2023년 6월 12일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

- Parquet Reader 타임스탬프 - Parquet Reader의 타임스탬프를 `bigint`(밀리초)로 읽는 기능을 추가했습니다. 이 업데이트는 Athena 엔진 버전 2에서 지원을 통해 패리티를 제공합니다.
- EXPLAIN ANALYZE - `EXPLAIN ANALYZE` 출력 및 쿼리 통계에 실제 입력 읽기 시간을 추가했습니다. `EXPLAIN ANALYZE`에 대한 자세한 내용은 Athena에서 EXPLAIN 및 EXPLAIN ANALYZE 사용 섹션을 참조하십시오.
- INSERT - `INSERT`로 작성된 테이블에서 쿼리 성능을 개선했습니다. `INSERT`에 대한 자세한 내용은 INSERT INTO 섹션을 참조하십시오.
- Delta Lake 테이블 - Delta Lake 테이블에서 동시 수정이 발생할 경우 완전히 삭제되지 않던 `DROP TABLE` 관련 문제를 수정했습니다.

2023년 6월 8일

2023년 6월 8일 게시

Amazon Athena for Apache Spark에서 다음과 같은 새로운 기능을 발표했습니다.

- 사용자 지정 Java 라이브러리 및 구성 지원 - 이제 Athena의 Apache Spark 세션에 대해 자체 Java 패키지와 사용자 지정 구성을 사용할 수 있습니다. Spark 속성을 통해 Athena 콘솔, AWS CLI 또는 Athena API를 사용하여 `.jar` 파일, 패키지 또는 기타 사용자 지정 구성을 지정할 수 있습니다. 자세한 내용은 JAR 파일 및 사용자 지정 Spark 구성 추가 단원을 참조하십시오.
- Apache Hudi, Apache Iceberg, Delta Lake 테이블에 대한 지원 – 이제 Athena for Spark는 Apache Iceberg, Apache Hudi, Linux Foundation Delta Lake 오픈 소스 데이터 레이크 스토리지 테이블 형식을 지원합니다. 자세한 내용은 Amazon Athena for Apache Spark에서 비 Hive 테이블 형식 사용 및 Athena for Spark에서 Apache Iceberg, Apache Hudi, Linux Foundation Delta Lake 테이블 사용에 관한 개별 주제를 참조하세요.

- Apache Spark에 대한 암호화 지원 - 이제 Athena for Spark에서 Spark 노드 간 전송 중 데이터 및 Spark에서 디스크에 저장한 로컬 저장 데이터에 대한 암호화를 활성화할 수 있습니다. Spark 암호화를 활성화하려면 Athena 콘솔, AWS CLI 또는 Athena API를 사용할 수 있습니다. 자세한 내용은 Apache Spark 암호화 활성화 단원을 참조하십시오.

Amazon Athena for Apache Spark에 대한 자세한 내용은 Amazon Athena에서 Apache Spark 사용 섹션을 참조하세요.

2023년 6월 2일

2023년 6월 2일 게시

이제 Athena에서 용량 예약을 삭제하고 AWS CloudFormation 템플릿을 사용하여 Athena 용량 예약을 지정할 수 있습니다.

- 용량 예약 삭제 - 이제 Athena에서 취소된 용량 예약을 삭제할 수 있습니다. 예약을 삭제하려면 먼저 취소해야 합니다. 용량 예약을 삭제하면 계정에서 예약이 즉시 제거됩니다. 삭제된 예약은 ARN을 포함하여 더 이상 참조할 수 없습니다. 예약을 삭제하려면 Athena 콘솔 또는 Athena API를 사용할 수 있습니다. 자세한 내용은 Amazon Athena 사용 설명서의 용량 예약 삭제 섹션 및 Amazon Athena API 참조의 DeleteCapacityReservation을 참조하세요.
- 용량 예약에 대해 AWS CloudFormation 템플릿 사용 – 이제 AWS CloudFormation 템플릿을 사용해 `AWS::Athena::CapacityReservation` 리소스를 사용하는 Athena 용량 예약을 지정할 수 있습니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 AWS::Athena::CapacityReservation을 참조하세요.

용량 예약을 사용하여 Athena에서 용량을 프로비저닝하는 방법에 대한 자세한 내용은 쿼리 처리 용량 관리 섹션을 참조하세요.

2023년 5월 25일

2023년 5월 25일 게시

Athena에서 페더레이션된 쿼리 성능을 개선하는 데이터 소스 커넥터 업데이트를 출시했습니다. 새로운 푸시다운 최적화 및 동적 필터링을 통해 Athena에서가 아니라 소스 데이터베이스에서 더 많은 작업을 수행할 수 있습니다. 이러한 최적화는 쿼리 런타임과 스캔되는 데이터 양을 줄여줍니다. 이 개선 사항을 적용하려면 Athena 엔진 버전 3이 필요합니다.

다음 커넥터가 업데이트되었습니다.

- Azure 데이터 레이크 스토리지
- Azure Synapse
- Cloudera Hive
- Cloudera Impala
- Db2
- DynamoDB
- Google BigQuery
- Hortonworks
- MySQL
- Oracle
- PostgreSQL
- Redshift
- SAP HANA
- Snowflake
- SQL Server
- Teradata

데이터 소스 커넥터 업그레이드에 대한 자세한 내용은 데이터 소스 커넥터 업데이트 섹션을 참조하세요.

2023년 5월 18일

2023년 5월 18일 게시

이제 Amazon Athena에 대한 IPv6 인바운드 연결에 AWS PrivateLink를 사용할 수 있습니다.

Amazon Athena는 AWS PrivateLink를 포함하도록 Internet Protocol 버전 6(IPv6) 엔드포인트를 통한 인바운드 연결 지원을 확대했습니다. 오늘부터 이전에 사용 가능했던 퍼블릭 IPv6 엔드포인트 외에도 Amazon Virtual Private Cloud(VPC)의 AWS PrivateLink를 사용하여 Athena에 안전하게 비공개로 연결할 수 있습니다.

인터넷의 급속한 성장으로 인해 Internet Protocol 버전 4(IPv4) 주소의 가용성이 고갈되고 있습니다. IPv6은 사용 가능한 주소 수를 몇 배로 늘려주므로 더 이상 VPC에서 중복되는 주소 공간을 관리하지

않아도 됩니다. 이번 릴리스에서는 이제 IPv6 주소 지정의 이점과 AWS PrivateLink의 보안 및 성능 이점을 결합할 수 있습니다.

AWS 서비스에 프로그래밍 방식으로 연결하려면 AWS CLI 또는 AWS SDK를 사용하여 엔드포인트를 지정할 수 있습니다. 서비스 엔드포인트 및 Athena 서비스 엔드포인트에 대한 자세한 내용은Amazon Web Services 일반 참조의 AWS service endpoints 및 Amazon Athena endpoints and quotas를 참조하세요.

2023년 5월 15일

2023년 5월 15일 게시

Athena에서는 DynamoDB, CloudWatch Logs, CloudWatch Metrics 및 AWS CMDB에 대한 Apache Spark DataSourceV2(DSV2) 커넥터 출시를 발표했습니다. 새 DSV2 커넥터를 사용하면 Spark를 사용하여 이러한 데이터 소스를 쿼리할 수 있습니다. DSV2 커넥터는 해당하는 Athena의 페더레이션된 커넥터와 동일한 파라미터를 사용합니다. DSV2 커넥터는 Spark 작업자에서 직접 실행되므로 커넥터를 사용하기 위해 Lambda 함수를 배포하지 않아도 됩니다.

자세한 내용은 Apache Spark용 Athena 데이터 소스 커넥터 단원을 참조하십시오.

2023년 5월 10일

2023년 5월 10일 게시

Athena용 ODBC 1.1.20 드라이버를 출시했습니다.

기능 및 개선 사항:

- Lake Formation 엔드포인트 재정의 지원.
- AD FS 인증 플러그인에는 Relying Party 값(`LoginToRP`)을 설정하기 위한 새 파라미터가 있습니다.
- AWS 라이브러리 업데이트.

버그 수정:

- `SQLPrepare()` 메서드 제출에 실패한 경우 준비된 명령문 할당 취소 실패.
- C 유형을 SQL 유형으로 변환할 때 준비된 명령문 파라미터를 바인딩하는 중 오류 발생.
- `EXPLAIN` 및 `EXPLAIN ANALYZE` 쿼리에서 `SQLPrepare()` 및 `SQLExecute()`를 사용할 때 데이터를 반환하지 못하는 문제.

자세한 내용을 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 ODBC로 Amazon Athena에 연결을 참조하세요.

2023년 5월 8일

2023년 5월 8일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

- Hudi 통합 업데이트 - Athena는 Apache Hudi와의 통합을 업데이트했습니다. 이제 Athena를 사용하여 Hudi 0.12.2 테이블을 쿼리할 수 있으며 이제 Hudi 테이블에 대한 Hudi 메타데이터 목록이 지원됩니다. 자세한 내용은 Athena를 사용하여 Apache Hudi 데이터 집합 쿼리 및 Hudi 메타데이터 목록 섹션을 참조하세요.
- 타임스탬프 변환 수정 - 타임스탬프 변환 처리를 정밀도가 낮은 데이터 형식으로 수정했습니다. 이전에 Athena 엔진 버전 3에서는 변환 중에 값을 잘라내는 대신 대상 유형으로 값을 잘못 반올림했습니다.

  다음 예제는 수정 이전의 잘못된 처리를 보여줍니다.

  예제 1: 타임스탬프를 마이크로초에서 밀리초로 변환

  샘플 데이터

```
A, 2020-06-10 15:55:23.383
B, 2020-06-10 15:55:23.382
C, 2020-06-10 15:55:23.383345
D, 2020-06-10 15:55:23.383945
E, 2020-06-10 15:55:23.383345734
F, 2020-06-10 15:55:23.383945278
```

  다음 쿼리는 특정 값과 일치하는 타임스탬프를 검색하려고 시도합니다.

```
SELECT *
FROM table
WHERE timestamps.col = timestamp'2020-06-10 15:55:23.383'
```

  쿼리는 다음 결과를 반환합니다.

```
A, 2020-06-10 15:55:23.383
C, 2020-06-10 15:55:23.383
```

```
E, 2020-06-10 15:55:23.383
```

수정 전에 Athena는 `2020-06-10 15:55:23.383945` 또는 `2020-06-10 15:55:23.383945278`을 포함하지 않습니다. `2020-06-10 15:55:23.384`로 반올림되었기 때문입니다.

예제 2: 타임스탬프에서 날짜로 변환

다음 쿼리는 잘못된 결과를 반환했습니다.

```
SELECT date(timestamp '2020-12-31 23:59:59.999')
```

Result

```
2021-01-01
```

수정하기 전에 Athena에서 값을 반올림하여 다음날로 바꾸었습니다. 이제 이러한 값은 반올림되지 않고 잘립니다.

2023년 4월 28일

2023년 4월 28일 게시

이제 Amazon Athena에서 용량 예약을 사용하여 완전관리형 컴퓨팅 용량에서 SQL 쿼리를 실행할 수 있습니다.

프로비저닝된 용량은 가장 중요한 대화형 워크로드의 우선순위를 지정하고 제어 및 확장할 수 있는 워크로드 관리 기능을 제공합니다. 언제든지 용량을 추가하여 동시에 실행할 수 있는 쿼리 수를 늘리고 용량을 사용할 수 있는 워크로드를 제어하며 워크로드 사이에서 용량을 공유할 수 있습니다.

자세한 내용은 쿼리 처리 용량 관리 단원을 참조하십시오. 요금에 대한 자세한 내용은 Amazon Athena 요금 페이지를 참조하세요.

2023년 4월 17일

2023년 4월 17일 게시

Athena에서 JDBC 드라이버 버전 2.0.36을 출시했습니다. 드라이버는 새로운 기능을 포함하며 문제도 해결되었습니다.

새로운 기능

- 이제 AD FS 인증과 함께 사용자 지정 가능한 신뢰 당사자 식별자를 사용할 수 있습니다.
- 이제 커넥터를 사용하는 애플리케이션 이름을 사용자 에이전트 문자열에 추가할 수 있습니다.

해결된 문제

- 존재하지 않는 스키마를 검색하는 데 `getSchema()`를 사용할 때 발생하는 오류가 수정되었습니다.

자세한 내용을 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결을 참조하세요.

2023년 4월 14일

2023년 6월 20일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

- 문자열을 타임스탬프로 변환할 때 요일과 시간 또는 시간대 사이에 공백이 필요합니다. 자세한 내용은 문자열에서 타임스탬프로 변환할 때 날짜와 시간 값 사이에 공백이 필요함 단원을 참조하십시오.
- 타임스탬프 정밀도가 처리되는 방식에서 주요 변경 사항을 제거했습니다. Athena 엔진 버전 2와 Athena 엔진 버전 3 사이에서 일관성을 유지하기 위해 이제 타임스탬프 정밀도가 기본적으로 마이크로초가 아닌 밀리초로 설정됩니다.
- 이제 Athena에서 쿼리를 실행할 때 쿼리 출력 버킷에 대한 액세스를 일관되게 적용합니다. StartQueryExecution 작업을 실행하는 모든 IAM 보안 주체가 출력 쿼리 버킷에서 S3:GetBucketLocation 권한을 갖고 있는지 확인하세요.

2023년 4월 4일

2023년 4월 4일 게시

이제 Amazon Athena를 사용하여 페더레이션된 데이터 소스에서 보기를 생성하고 쿼리할 수 있습니다. 하나의 페더레이션된 보기를 사용하여 여러 외부 테이블 또는 데이터 하위 세트를 쿼리할 수 있습니다. 이렇게 하면 필요한 SQL을 단순화하고 SQL을 사용하여 데이터를 쿼리해야 하는 최종 사용자의 데이터 소스를 유연하게 난독화할 수 있습니다.

자세한 내용은 뷰 작업 및 페더레이션된 쿼리 실행 단원을 참조하세요.

2023년 3월 30일

2023년 3월 30일 게시

Amazon Athena는 추가로 AWS 리전에서 Amazon Athena for Apache Spark의 가용성을 발표했습니다.

이 릴리스에서는 Amazon Athena for Apache Spark의 가용성을 아시아 태평양(뭄바이), 아시아 태평양(싱가포르), 아시아 태평양(시드니) 및 유럽(프랑크푸르트)으로 확장합니다.

Amazon Athena for Apache Spark에 대한 자세한 내용은 Amazon Athena에서 Apache Spark 사용 섹션을 참조하세요.

2023년 3월 28일

2023년 3월 28일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

- `GetQueryExecution` 및 `BatchGetQueryExecution` Athena API 작업에 대한 응답에서 새 `subStatementType` 필드에는 실행된 쿼리 유형(예: `SELECT`, `INSERT`, `UNLOAD`, `CREATE_TABLE` 또는 `CREATE_TABLE_AS_SELECT`)이 표시됩니다.
- Apache Hive 쓰기 작업에서 매니페스트 파일이 올바르게 암호화되지 않는 버그가 수정되었습니다.
- 이제 Athena 엔진 버전 3이 `approx_percentile` 함수의 `NaN` 및 `Infinity` 값을 올바르게 처리합니다. 이 `approx_percentile` 함수는 데이터 세트의 대략적인 백분위수를 지정된 백분율로 반환합니다.

  Athena 엔진 버전 2에서는 `NaN`을 `Infinity`보다 큰 값으로 잘못 처리했습니다. 이제 Athena 엔진 버전 3은 다른 분석 및 통계 함수에서 이러한 값을 처리하는 방식에 따라 `NaN` 및 `Infinity`를 처리합니다. 다음에서 새로운 동작을 더 자세히 설명합니다.
  - `NaN`이 데이터 세트에 있으면 Athena에서 `NaN`을 반환합니다.
  - `NaN`은 없지만 `Infinity`이 있는 경우 Athena는 `Infinity`를 매우 큰 수로 처리합니다.
  - `Infinity` 값이 여러 개 있는 경우 Athena는 두 값을 동일한 매우 큰 수로 처리합니다. 필요한 경우 Athena에서 `Infinity`를 출력합니다.
  - 단일 데이터 세트에 `-Infinity` 및 `-Double.MAX_VALUE`가 모두 있고 백분위수 결과가 `-Double.MAX_VALUE`이면 Athena에서 `-Infinity`를 반환합니다.
  - 단일 데이터 세트에 `Infinity` 및 `Double.MAX_VALUE`가 모두 있고 백분위수 결과가 `Double.MAX_VALUE`이면 Athena에서 `Infinity`를 반환합니다.

- 계산에서 `Infinity` 및 `NaN`을 제외하려면 다음 예제와 같이 `is_finite()` 함수를 사용합니다.

```
approx_percentile(x, 0.5) FILTER (WHERE is_finite(x))
```

2023년 3월 27일

2023년 3월 27일 게시

이제 Amazon Athena에서 Athena SQL 작업 그룹에 대해 최소 암호화 수준을 지정할 수 있습니다. 이 기능을 사용하면 Athena SQL 작업 그룹의 모든 쿼리 결과를 사용자가 지정한 암호화 수준 이상으로 암호화합니다. 여러 수준의 암호화 강도 중에서 선택하여 데이터를 보호할 수 있습니다. Athena 콘솔, AWS CLI, API 또는 SDK를 사용하여 원하는 최소 암호화 수준을 구성할 수 있습니다.

Apache Spark 지원 작업 그룹에서는 최소 암호화 기능을 사용할 수 없습니다. 자세한 내용은 작업 그룹에 대한 최소 암호화 구성 단원을 참조하십시오.

2023년 3월 17일

2023년 3월 17일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

- Amazon Athena DynamoDB 커넥터에서 쿼리에 실패하고 KeyConditionExpressions must only contain one condition per key 오류 메시지를 표시하던 문제를 수정했습니다.

  이 문제는 Athena 엔진 버전 3에서 Athena 엔진 버전 2보다 더 많은 종류의 조건자를 푸시다운할 수 있는 기회를 인식하기 때문에 발생합니다. Athena 엔진 버전 3에서는 `some_column LIKE 'someprefix%'`와 같은 절이 지정된 열에 하한 및 상한을 적용하는 필터 조건자로 푸시다운됩니다. Athena 엔진 버전 2에서는 이러한 조건을 푸시다운하지 않았습니다. Athena 엔진 버전 3에서 `some_column`이 정렬 키 열인 경우 엔진은 필터 조건자를 DynamoDB 커넥터로 푸시다운합니다. 그러면 필터 조건자가 DynamoDB 서비스로 푸시다운됩니다. DynamoDB는 정렬 키에 대해 둘 이상의 필터 조건을 지원하지 않으므로 DynamoDB에서 오류를 반환합니다.

  이 문제를 해결하려면 Amazon Athena DynamoDB 커넥터를 버전 2023.11.1로 업데이트합니다. 커넥터 업데이트에 대한 지침은 데이터 소스 커넥터 업데이트 섹션을 참조하세요.

2023년 3월 8일

2023년 3월 8일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

- 타임스탬프 조건자 값이 밀리초가 아닌 마이크로초로 전송되는 페더레이션된 쿼리 관련 문제를 수정했습니다.

2023년 2월 15일

2023년 2월 15일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

- 이제 클라이언트 측 암호화를 사용하여 Iceberg 쓰기 작업을 위해 Amazon S3의 데이터를 암호화할 수 있습니다.
- Amazon S3에서 Iceberg 쓰기 작업을 위해 서버 측 암호화에 영향을 주던 문제를 수정했습니다.

2023년 1월 31일

2023년 1월 31일 게시

이제 Amazon Athena를 사용하여 Google Cloud Storage에서 데이터를 쿼리할 수 있습니다. Amazon S3와 마찬가지로 Google Cloud Storage는 버킷에 데이터를 저장하는 관리형 서비스입니다. Google Cloud Storage용 Athena 커넥터를 사용하여 외부 데이터에서 대화형 통합 쿼리를 실행합니다.

자세한 내용은 Amazon Athena Google Cloud Storage 커넥터 단원을 참조하십시오.

2023년 1월 20일

2023년 1월 20일 게시

이제 Athena 압축 지원에 대한 확장된 설명서를 볼 수 있습니다. Hive 테이블 압축, Iceberg 테이블 최적화 및 ZSTD 압축 수준에 대한 개별 주제가 추가되었습니다.

자세한 내용은 Athena 압축 지원 단원을 참조하십시오.

2023년 1월 3일

2023년 1월 3일 게시

Athena에서 다음 업데이트를 발표합니다.

- Hive 메타스토어에 대한 추가 명령 - Athena를 사용하여 자체 관리형 Apache Hive 메타스토어에 메타데이터 카탈로그로 연결하고 Amazon S3에 저장된 데이터를 쿼리할 수 있습니다. 이 릴리스에서는 `CREATE TABLE AS`(CTAS), `INSERT INTO` 및 12개의 추가 DDL(데이터 정의 언어) 명령을 사용하여 Apache Hive 메타스토어와 상호 작용할 수 있습니다. 이 확장된 SQL 기능 세트를 사용하여 Athena에서 직접 Hive 메타스토어 스키마를 관리할 수 있습니다.

  자세한 내용은 외부 Hive 메타스토어용 Athena 데이터 커넥터 사용 단원을 참조하십시오.
- JDBC 드라이버 버전 2.0.35 - Athena에서 JDBC 드라이버 버전 2.0.35를 릴리스합니다. JDBC 2.0.35 드라이버에는 다음과 같은 업데이트가 포함되어 있습니다.
  - 이제 이 드라이버에서는 Jackson JSON 구문 분석기에 다음 라이브러리를 사용합니다.
    - jackson-annotations 2.14.0(이전 2.13.2)
    - jackson-core 2.14.0(이전 2.13.2)
    - jackson-databind 2.14.0(이전 2.13.2.2)
  - JDBC 버전 4.1에 대한 지원이 중단되었습니다.

  자세한 내용을 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결 단원을 참조하세요.

2022년 Athena 릴리스 정보

2022년 12월 14일

2022년 12월 14일 게시

이제 Kafka용 Amazon Athena 커넥터를 사용하여 스트리밍 데이터에 대한 SQL 쿼리를 실행할 수 있습니다. 예를 들어 Amazon Managed Streaming for Apache Kafka(Amazon MSK)에서 실시간 스트리밍 데이터에 대한 분석 쿼리를 실행하고 이를 Amazon S3의 데이터 레이크에 있는 기록 데이터와 조인할 수 있습니다.

Kafka용 Amazon Athena 커넥터는 여러 스트리밍 엔진에서 쿼리를 지원합니다. Athena를 사용하여 Amazon MSK 프로비저닝 및 서버리스 클러스터, 자체 관리형 Kafka 배포 및 Confluent Cloud의 스트리밍 데이터에서 SQL 쿼리를 실행할 수 있습니다.

자세한 내용은 Amazon Athena MSK 커넥터 단원을 참조하십시오.

2022년 12월 2일

2022년 12월 2일 게시

Athena에서 JDBC 드라이버 버전 2.0.34를 릴리스합니다. JDBC 2.0.34 드라이버에는 다음과 같은 새로운 기능과 해결된 문제가 포함되어 있습니다.

- 쿼리 결과 재사용 지원 - 이제 쿼리를 실행할 때마다 Athena에서 결과를 다시 계산하는 대신 지정한 시간 제한까지 이전에 실행한 쿼리 결과를 재사용할 수 있습니다. 자세한 내용은 JDBC 다운로드 페이지에서 제공되는 설치 및 구성 안내서와 쿼리 결과 재사용 단원을 참조하세요.
- Ec2InstanceMetadata 지원 - JDBC 드라이버에서 이제 IAM 인스턴스 프로필을 사용하여 Ec2InstanceMetadata 인증 방법을 지원합니다.
- 문자 기반 예외 수정 - 특정 언어 문자가 포함된 쿼리에서 발생하는 예외를 수정했습니다.
- 취약점 수정 - 커넥터와 함께 패키징된 AWS 종속성 관련 취약점을 수정했습니다.

자세한 내용을 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결을 참조하세요.

2022년 11월 30일

2022년 11월 30일 게시

이제 Athena에서 Apache Spark 애플리케이션 및 Jupyter 호환 노트북을 대화식으로 생성하고 실행할 수 있습니다. 리소스를 계획, 구성 또는 관리할 필요 없이 Spark를 사용하여 Athena에서 데이터 분석을 실행합니다. 처리를 위해 Spark 코드를 제출하고 결과를 직접 수신합니다. Amazon Athena 콘솔의 간소화된 노트북 환경을 활용하면 Python 또는 Athena 노트북 API를 통해 Apache Spark 애플리케이션을 개발할 수 있습니다.

Amazon Athena의 Apache Spark는 서버리스이며, 자동 온디맨드로 확장하여 즉시 컴퓨팅을 제공함으로써 변화하는 데이터 볼륨 및 처리 요구 사항을 충족할 수 있습니다.

자세한 내용은 Amazon Athena에서 Apache Spark 사용 단원을 참조하십시오.

2022년 11월 18일

2022년 11월 18일 게시

이제 IBM Db2용 Amazon Athena 커넥터를 사용하여 Athena에서 Db2를 쿼리할 수 있습니다. 예를 들어 Db2의 데이터 웨어하우스와 Amazon S3의 데이터 레이크에 대해 분석 쿼리를 실행할 수 있습니다.

Amazon Athena Db2 커넥터는 Lambda 환경 변수를 통해 여러 가지 구성 옵션을 제공합니다. 구성 옵션, 파라미터, 연결 문자열, 배포 및 제한 사항에 대한 자세한 내용은 Amazon Athena IBM Db2 커넥터 단원을 참조하세요.

2022년 11월 17일

2022월 11월 17일 게시

Athena 엔진 버전 3의 Apache Iceberg 지원은 이제 다음과 같은 향상된 ACID 트랜잭션 기능을 제공합니다.

- ORC 및 Avro 지원 - Apache Avro 및 Apache ORC 행 및 열 기반 파일 형식을 사용하여 Iceberg 테이블을 생성합니다. 이러한 형식에 대한 지원이 Parquet에 대한 기존 지원에 추가됩니다.
- MERGE INTO - `MERGE INTO` 명령을 사용하여 대규모 데이터를 효율적으로 병합합니다. `MERGE INTO`는 `INSERT`, `UPDATE` 및 `DELETE` 연산을 하나의 트랜잭션으로 결합합니다. 그러면 데이터 파이프라인의 처리 오버헤드가 줄고 기록할 SQL이 감소됩니다. 자세한 내용은 Iceberg 테이블 데이터 업데이트 및 MERGE INTO 단원을 참조하세요.
- CTAS 및 VIEW 지원 - `CREATE TABLE AS SELECT`(CTAS) 및 `CREATE VIEW` 문을 Iceberg 테이블과 함께 사용하세요. 자세한 내용은 CREATE TABLE AS 및 CREATE VIEW 단원을 참조하세요.
- VACUUUM 지원 - `VACUUM` 문을 사용하여 더 이상 필요하지 않은 스냅샷 및 데이터를 삭제하여 데이터 레이크를 최적화할 수 있습니다. 이 기능을 사용하여 읽기 성능을 개선하고 GDPR과 같은 규제 요구 사항을 충족할 수 있습니다. 자세한 내용은 Iceberg 테이블 최적화 및 VACUUM 단원을 참조하세요.

이러한 새 기능은 Athena 엔진 버전 3이 필요하며 Athena가 지원되는 모든 리전에서 사용할 수 있습니다. Athena 콘솔, 드라이버 또는 API와 함께 사용할 수 있습니다.

Athena에서 Iceberg를 사용하는 방법에 대한 자세한 내용은 Apache Iceberg 테이블 사용 단원을 참조하세요.

2022년 11월 14일

2022년 11월 14일 게시

이제 Amazon Athena에서 IPv6를 통해 Athena 함수를 호출하는 데 사용할 수 있는 인바운드 연결용 IPv6 엔드포인트를 지원합니다. 이 기능을 사용하여 IPv6 규정 준수 요구 사항을 충족할 수 있습니다. 또한 IPv4와 IPv6 간의 주소 변환을 처리하기 위한 추가 네트워킹 장비가 필요하지 않습니다.

이 기능을 사용하려면 IPv4와 IPv6를 모두 지원하는 새로운 Athena 듀얼 스택 엔드포인트를 사용하도록 애플리케이션을 구성해야 합니다. 듀얼 스택 엔드포인트에서는 `athena.`*`region`*`.api.aws` 형식을 사용합니다. 예를 들어 미국 동부(버지니아 북부) 리전의 듀얼 스택 엔드포인트는 `athena.us-east-1.api.aws`입니다.

듀얼 스택 Athena 엔드포인트에 요청하는 경우, 엔드포인트가 네트워크 및 클라이언트에서 사용하는 프로토콜에 따라 IPv6 또는 IPv4 주소로 확인됩니다. AWS 서비스에 프로그래밍 방식으로 연결하려면 AWS CLI 또는 AWS SDK를 사용하여 엔드포인트를 지정할 수 있습니다.

서비스 엔드포인트에 대한 자세한 내용은 AWS 서비스 엔드포인트를 참조하세요. Athena의 서비스 엔드포인트에 대한 자세한 내용은 AWS 설명서의 Amazon Athena 엔드포인트 및 할당량을 참조하세요.

추가 비용 없이 인바운드 연결에 새로운 Athena 듀얼 스택 엔드포인트를 사용할 수 있습니다. 듀얼 스택 엔드포인트는 일반적으로 모든 AWS 리전에서 사용할 수 있습니다.

2022년 11월 11일

2022년 11월 11일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

- 확장된 Lake Formation 세분화된 액세스 제어 - 이제 지원되는 파일 또는 테이블 형식으로 저장된 데이터에 대해 Athena 쿼리에서 AWS Lake Formation 세분화된 액세스 제어 정책을 사용할 수 있습니다. Lake Formation의 세분화된 액세스 제어를 활용하면 데이터 필터를 사용하여 쿼리 결과의 데이터에 대한 액세스를 제한하여 열 수준, 행 수준 및 셀 수준 보안을 달성할 수 있습니다. Athena에서 지원되는 테이블 형식에는 Apache Iceberg, Apache Hudi, Apache Hive가 있습니다. Athena가 지원하는 모든 리전에서 확장된 세분화된 액세스 제어를 사용할 수 있습니다. 확장된 테이블 및 파일 형식을 지원하려면 Athena 엔진 버전 3이 필요합니다. 여기서는 새로운 기능과 향상된 쿼리 성능을 제공하지만 Lake Formation에서 세분화된 액세스 제어 정책을 설정하는 방법은 변경되지 않습니다.

  Athena에서 이 확장되고 세분화된 액세스 제어를 사용할 경우 다음 사항을 고려하세요.

  - EXPLAIN - Lake Formation에 정의된 행 또는 셀 필터링 정보와 쿼리 통계 정보는 `EXPLAIN` 및 `EXPLAIN ANALYZE`의 출력에 표시되지 않습니다. Athena의 `EXPLAIN`에 대한 자세한 내용은 Athena에서 EXPLAIN 및 EXPLAIN ANALYZE 사용 단원을 참조하세요.
  - 외부 Hive 메타스토어 - Apache Hive 숨김 열은 세분화된 액세스 제어 필터링에 사용될 수 없으며 Apache Hive 숨김 시스템 테이블은 세분화된 액세스 제어에서 지원되지 않습니다. 자세한 내용은 고려 사항 및 제한 주제에서 외부 Hive 메타스토어용 Athena 데이터 커넥터 사용 섹션을 참조하세요.
  - Query 통계 – 쿼리에서 행 수준 필터가 Lake Formation에 정의된 경우 단계 수준의 입력 및 출력 행 수와 데이터 크기 정보가 Athena 쿼리 통계에 표시되지 않습니다. Athena 쿼리 통계를 보는 방법에 대한 자세한 내용은 완료된 쿼리에 대한 통계 및 실행 세부 정보 보기 및 GetQueryRuntimeStatistics를 참조하세요.

- 작업 그룹 - 동일한 Athena 작업 그룹의 사용자는 Lake Formation 세분화된 액세스 제어를 통해 작업 그룹에 액세스할 수 있도록 구성한 데이터를 볼 수 있습니다. Athena를 사용하여 Lake Formation에 등록된 데이터를 쿼리하는 방법에 대한 자세한 내용은 Athena를 사용하여 AWS Lake Formation에 등록된 데이터 쿼리 단원을 참조하세요.

  Lake Formation에서 세분화된 액세스 제어를 사용하는 방법에 대한 자세한 내용은 AWS 빅 데이터 블로그의 AWS Lake Formation을 사용하여 세분화된 액세스 제어 관리를 참조하세요.
- Athena 페더레이션 쿼리 - Athena 페더레이션 쿼리에서는 이제 `struct` 객체 내 필드 이름의 원래 대/소문자를 유지합니다. 이전에는 `struct` 필드 이름이 자동으로 소문자로 지정되었습니다.

2022년 11월 8일

2022년 11월 8일 게시

이제 쿼리 결과 재사용 캐싱 기능을 사용하여 Athena에서 반복 쿼리를 가속화할 수 있습니다. 반복 쿼리는 최근에 제출한 것과 동일한 결과를 생성하는 SQL 쿼리입니다. 동일한 여러 쿼리를 실행해야 하는 경우 결과 재사용 캐싱을 통해 결과 생성에 필요한 시간을 줄일 수 있습니다. 또한 결과 재사용 캐싱은 스캔되는 바이트 수를 줄여 비용을 절감합니다.

자세한 내용은 쿼리 결과 재사용 단원을 참조하십시오.

2022년 10월 13일

2022년 10월 13일 게시

Athena에서 Athena 엔진 버전 3을 발표합니다.

Athena에서는 SQL 쿼리 엔진을 업그레이드하여 Trino 오픈 소스 프로젝트의 최신 기능을 포함했습니다. Athena 엔진 버전 2의 모든 기능 지원 이외에 Athena 엔진 버전 3에는 50개 이상의 새로운 SQL 함수, 30개의 새로운 기능 및 90가지가 넘는 쿼리 성능 개선 사항이 포함되어 있습니다. 오늘 출시와 함께 Athena에서는 Athena 엔진 내에서 통합되고 조정되는 커뮤니티 개선 사항에 더 빠르게 액세스할 수 있도록 Trino 및 Presto 프로젝트로 통화를 개선하는 지속적인 통합 접근 방식도 오픈 소스 소프트웨어 관리에 도입합니다.

자세한 내용은 Athena 엔진 버전 3 단원을 참조하십시오.

2022년 10월 10일

2022년 10월 10일 게시

Athena에서 JDBC 드라이버 버전 2.0.33을 릴리스합니다. JDBC 2.0.33 드라이버에는 다음과 같은 변경 사항이 포함되어 있습니다.

- 새 드라이버 버전, JDBC 버전 및 플러그인 이름 속성이 보안 인증 정보 공급자 클래스의 사용자 에이전트 문자열에 추가되었습니다.
- 오류 메시지가 수정되고 필요한 정보가 추가되었습니다.
- 연결이 종료되거나 Athena에서 준비한 명령문 실행이 실패하는 경우 이제는 준비된 명령문의 할당이 취소됩니다.

자세한 내용을 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결을 참조하세요.

2022년 9월 23일

2022년 9월 26일 게시

Amazon Athena Neptune 커넥터에서 이제는 열 및 테이블 이름의 대/소문자를 구분하지 않는 일치를 지원합니다.

- AWS Glue의 테이블에서 열 이름이 모두 소문자로 표시되더라도 대/소문자를 사용하는 Neptune 테이블의 열 이름을 Neptune 데이터 소스 커넥터에서 확인할 수 있습니다. 이 동작을 활성화하려면 Neptune 커넥터 Lambda 함수의 `enable_caseinsensitivematch` 환경 변수를 `true`로 설정하세요.
- AWS Glue에서는 소문자 테이블 이름만 지원하므로 Neptune의 AWS Glue 테이블을 생성할 때 AWS Glue 테이블 파라미터 `"glabel" = table_name`를 지정하세요.

Neptune 커넥터에 대한 자세한 내용은 Amazon Athena Neptune 커넥터 섹션을 참조하세요.

2022년 9월 13일

2022년 9월 13일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

- 외부 Hive 메타데이터 – Athena는 이제 `WHERE` 절에 외부 Hive 메타스토어(EHMS)에 존재하지 않는 파티션이 포함된 경우 예외를 발생시키는 대신 `NULL`을 반환합니다. 새 동작은 AWS Glue Data Catalog의 동작과 일치합니다.

- 파라미터화된 쿼리 – 파라미터화된 쿼리의 값을 이제 DOUBLE 데이터 형식으로 캐스팅할 수 있습니다.
- Apache Iceberg – 이제 Amazon S3 버킷에서 객체 잠금이 활성화된 경우 Iceberg 테이블에 대한 쓰기 작업이 성공합니다.

2022년 8월 31일

2022년 8월 31일 게시

Amazon Athena가 아시아 태평양(자카르타) 리전에서 Athena 출시와 그 기능을 발표했습니다.

이 릴리스에서는 아시아 태평양(홍콩), 아시아 태평양(자카르타), 아시아 태평양(뭄바이), 아시아 태평양(오사카), 아시아 태평양(서울), 아시아 태평양(싱가포르), 아시아 태평양(시드니), 아시아 태평양(도쿄) 등 아시아 태평양 리전에서의 Athena 서비스 제공을 확대했습니다. 해당 및 기타에서 이용 가능한 AWS 서비스의 전체 목록은 AWS 리전 서비스 목록을 참조하세요.

2022년 8월 23일

2022년 8월 23일 게시

Athena Query Federation SDK의 릴리스 v2022.32.1에는 다음과 같은 변경 사항이 포함됩니다.

- Amazon Athena Oracle 데이터 소스 커넥터에는 Amazon RDS 인스턴스에 대한 SSL 기반 연결에 관한 지원이 추가되었습니다. 지원은 전송 계층 보안(TLS) 프로토콜 및 클라이언트에 의한 서버 인증으로 제한됩니다. 상호 인증은 Amazon RDS에서 지원되지 않으므로 업데이트에는 상호 인증에 대한 지원이 포함되지 않습니다.

  자세한 내용은 Amazon Athena Oracle 커넥터 단원을 참조하십시오.

2022년 8월 3일

2022년 8월 3일 게시

Athena가 JDBC 드라이버 버전 2.0.32를 릴리스했습니다. JDBC 2.0.32 드라이버에는 다음과 같은 변경 사항이 포함되어 있습니다.

- Athena SDK로 전송되는 User-Agent 문자열은 드라이버 버전, JDBC 사양 버전 및 인증 플러그인 이름을 포함하도록 확장되었습니다.

- `CheckNonProxyHost` 파라미터 값을 제공하지 않을 때 발생하던 `NullPointerException`을 수정했습니다.
- BrowserSaml 인증 플러그인에서 `login_url` 구문 분석 관련 문제를 수정했습니다.
- `UseProxyforIdp` 파라미터를 `true`로 설정할 때 발생하던 프록시 호스트 문제를 수정했습니다.

자세한 내용을 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결을 참조하세요.

2022년 8월 1일

2022년 8월 1일 게시

Athena는 Athena Query Federation SDK 및 Athena 사전 구축된 데이터 원본 커넥터에 대한 개선 사항을 발표했습니다. 개선 사항은 다음과 같습니다.

- 구조체 구문 분석 – 복잡한 특정 구조체가 모든 데이터를 표시하지 못했던 Athena Query Federation SDK의 `GlueFieldLexer` 구문 분석 오류를 수정했습니다. 이 문제는 Athena Query Federation SDK를 기반으로 구축된 커넥터에 영향을 미쳤습니다.
- AWS Glue 테이블 – AWS Glue 테이블의 `set` 및 `decimal` 열 유형에 대한 지원을 추가했습니다.
- DynamoDB 커넥터 – DynamoDB 속성 이름의 대소문자를 무시하는 기능을 추가했습니다. 자세한 내용은 Amazon Athena DynamoDB 커넥터 페이지의 파라미터 단원에서 `disable_projection_and_casing`을 참조하세요.

자세한 내용은 GitHub에서 Athena Query Federation의 v2022.30.2 릴리스를 참조하세요.

2022년 7월 21일

2022년 7월 21일 게시

이제 Athena 콘솔에서 성능 지표 및 대화형 시각적 쿼리 분석 도구를 사용하여 쿼리를 분석하고 디버그할 수 있습니다. 쿼리 성능 데이터 및 실행 세부 정보를 통해 쿼리의 병목 현상을 파악하고 쿼리의 각 단계에 대한 연산자 및 통계를 검사하며 단계 간 데이터 흐름의 양을 추적하고 쿼리 조건자의 영향을 확인할 수 있습니다. 이제 다음 작업을 수행할 수 있습니다.

- 클릭 한 번으로 쿼리에 대한 분산 및 논리적 실행 계획에 액세스합니다.
- 단계가 실행되기 전에 각 단계의 작업을 살펴봅니다.

- 대기열, 계획, 실행 단계에서 소요된 시간에 대한 지표를 이용해 완료된 쿼리의 성능을 시각화합니다.
- 쿼리로 처리되고 출력되는 소스 데이터의 행 수 및 양에 대한 정보를 확인합니다.
- 컨텍스트에 맞게 표시되고 대화형 그래프로 서식이 지정된 쿼리의 실행 세부 정보를 자세하게 확인합니다.
- 정확한 단계 수준의 실행 세부 정보를 사용하여 쿼리를 통한 데이터 흐름을 파악합니다.
- 새로운 API를 사용하여 프로그래밍 방식으로 쿼리 성능 데이터를 분석하여 쿼리 런타임 통계를 가져옵니다. 오늘 함께 공개되었습니다.

쿼리에서 이러한 기능을 사용하는 방법을 알아보려면 AWS YouTube 채널에서 새로운 쿼리 분석 도구를 사용한 Amazon Athena 쿼리 최적화 비디오 자습서를 시청하세요.

설명서는 SQL 쿼리에 대한 실행 계획 보기 및 완료된 쿼리에 대한 통계 및 실행 세부 정보 보기 단원을 참조하세요.

2022년 7월 11일

2022년 7월 11일 게시

이제 SQL 문을 미리 준비하지 않고도 Athena 콘솔 또는 API에서 직접 파라미터화된 쿼리를 실행할 수 있습니다.

Athena 콘솔에서 물음표 형태의 파라미터가 있는 쿼리를 실행하면, 이제 파라미터 값을 직접 입력하라는 메시지가 사용자 인터페이스에 표시됩니다. 이렇게 하면 쿼리를 실행할 때마다 쿼리 편집기에서 리터럴 값을 수정할 필요가 없습니다.

향상된 쿼리 실행 API를 사용하는 경우, 이제 한 번의 호출로 실행 파라미터와 해당 값을 제공할 수 있습니다.

자세한 내용은 이 사용 설명서의 파라미터화된 쿼리 사용 및 AWS 빅 데이터 블로그 게시물 Amazon Athena 파라미터화된 쿼리를 사용하여 데이터를 서비스로 제공 단원을 참조하세요.

2022년 7월 8일

2022년 7월 8일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

- 쿼리 실패를 초래하는 SageMaker 엔드포인트(UDF)에 대한 DATE 열 변환 처리 문제를 수정했습니다.

2022년 6월 6일

2022년 6월 6일 게시

Athena가 JDBC 드라이버 버전 2.0.31을 릴리스했습니다. JDBC 2.0.31 드라이버에는 다음과 같은 변경 사항이 포함되어 있습니다.

- log4j 종속성 문제 – log4j 종속성으로 인해 발생하는 드라이버 클래스를 찾을 수 없음 오류 메시지를 해결했습니다.

자세한 내용을 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결을 참조하세요.

2022년 5월 25일

2022년 5월 25일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

- Iceberg 지원
  - 리전 간 쿼리에 대한 지원이 도입되었습니다. 이제 사용 중인 AWS 리전과 다른 AWS 리전의 Iceberg 테이블을 쿼리할 수 있습니다. 중국 리전-리전 간 쿼리 작성은 중국 리전에서 지원되지 않습니다.
  - 서버 측 암호화 구성에 대한 지원이 도입되었습니다. 이제 SSE-S3/SSE-KMS를 사용하여 Amazon S3에서 Iceberg 쓰기 작업의 데이터를 암호화할 수 있습니다.

  Athena에서 Apache Iceberg를 사용하는 방법에 대한 자세한 내용은 Apache Iceberg 테이블 사용 단원을 참조하세요.
- JDBC 2.0.30 드라이버 릴리스

  Athena용 JDBC 2.0.30 드라이버에는 다음과 같은 개선 사항이 포함되어 있습니다.
  - 파라미터화되고 준비된 문에 영향을 주는 데이터 경쟁 문제를 해결합니다.
  - Gradle 빌드 환경에서 발생한 애플리케이션 시작 문제를 수정합니다.

JDBC 2.0.30 드라이버, 릴리스 정보 및 문서를 다운로드하려면 JDBC로 Amazon Athena에 연결 단원을 참조하세요.

2022년 5월 6일

2022년 5월 6일 게시

Athena용 JDBC 2.0.29 및 ODBC 1.1.17 드라이버를 릴리스했습니다.

이러한 드라이버에는 다음과 같은 변경 사항이 포함되어 있습니다.

- SAML 플러그인 브라우저 시작 프로세스를 업데이트했습니다.

이러한 변경 사항에 대해 자세히 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결 및 ODBC로 Amazon Athena에 연결 섹션을 참조하세요.

2022년 4월 22일

2022년 4월 22일 게시

Athena에서 다음 수정 및 개선 사항을 발표했습니다.

- 다음 조건이 충족될 때 파티션 캐시가 있는 파티션 인덱스 및 필터링 기능에서 발생하는 문제를 해결했습니다.
  - 테이블의 AWS Glue 테이블 속성에서 `partition_filtering.enabled` 키가 `true`로 설정되었습니다.
  - 동일한 테이블이 다른 파티션 필터 값을 사용하여 여러 번 사용되었습니다.

2022년 4월 21일

2022년 4월 21일 게시

이제 Amazon Athena를 사용하여 Google BigQuery, Azure Synapse, Snowflake를 비롯한 새 데이터 소스에 대한 연합 쿼리를 실행할 수 있습니다. 새 데이터 소스 커넥터는 다음을 포함합니다.

- Azure Data Lake Storage(ADLS) Gen2
- Azure Synapse

- Cloudera Hive
- Cloudera Impala
- Google BigQuery
- Hortonworks
- Microsoft SQL Server
- Oracle
- SAP HANA(Express Edition)
- Snowflake
- Teradata

Athena가 지원하는 데이터 소스의 전체 목록은 사용 가능한 데이터 소스 커넥터 단원을 참조하세요.

사용 가능한 소스를 쉽게 검색하고 데이터에 연결할 수 있도록 Athena 콘솔의 업데이트된 데이터 소스 화면에서 사용 가능한 커넥터를 검색, 정렬 및 필터링할 수 있습니다.

연합 소스 쿼리에 대한 자세한 내용은 Amazon Athena 연합 쿼리 사용 및 페더레이션된 쿼리 실행 섹션을 참조하세요.

2022년 4월 13일

2022년 4월 13일 게시

Athena가 JDBC 드라이버 버전 2.0.28을 릴리스했습니다. JDBC 2.0.28 드라이버에는 다음과 같은 변경 사항이 포함되어 있습니다.

- JWT 지원 - 이제 드라이버는 인증을 위해 JSON 웹 토큰(JWT)을 지원합니다. JDBC 드라이버와 함께 JWT를 사용하는 방법에 대한 자세한 내용은 JDBC 드라이버 페이지에서 다운로드 가능한 설치 및 구성 가이드를 참조하세요.
- Log4j 라이브러리 업데이트 - 이제 JDBC 드라이버는 다음과 같은 Log4j 라이브러리를 사용합니다.
  - Log4j-api 2.17.1(이전 2.17.0)
  - Log4j-core 2.17.1(이전 2.17.0)
  - Log4j-jcl 2.17.2
- 기타 개선 사항 - 새 드라이버에는 다음과 같은 개선 사항 및 버그 수정도 포함되어 있습니다.
  - 이제 JDBC를 통해 Athena 준비된 문 기능을 사용할 수 있습니다. 준비된 문에 대한 자세한 내용은 파라미터화된 쿼리 사용를 참조하세요.

- 이제 Athena JDBC SAML 연동은 중국 리전에서 이용 기능합니다.
- 사소한 추가 개선 사항.

자세한 내용을 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결을 참조하세요.

2022년 3월 30일

2022년 3월 30일 게시

<shared id="ATE"/>에서 다음 기능과 개선 사항을 발표했습니다.

- 교차 리전 쿼리— 이제 Athena를 사용하여 Amazon S3 버킷의 데이터를 쿼리할 수 있습니다.AWS 리전아시아 태평양(홍콩), 중동(바레인), 아프리카(케이프타운) 및 유럽(밀라노) 가 포함됩니다. 중국 리전-리전 간 쿼리 작성은 중국 리전에서 지원되지 않습니다.
  - Athena를 사용할 수 있는 AWS 리전 목록은 Amazon Athena 엔드포인트 및 할당량을 참조하십시오.
  - 기본적으로 비활성화되어 있는 AWS 리전 활성화에 대한 자세한 내용은 리전 활성화를 참조하십시오.
  - 리전 간 쿼리에 대한 자세한 내용은 리전 간 쿼리를 참조하세요.

2022년 3월 18일

2022년 3월 18일 게시

<shared id="ATE"/>에서 다음 기능과 개선 사항을 발표했습니다.

- 동적 필터링– 해당 테이블의 각 레코드에 필터를 효율적으로 적용하여 정수 열고 관련하여 동적 필터링이 개선되었습니다.
- Iceberg— 2GB보다 큰 Iceberg Parquet 파일을 작성할 때 오류가 발생하는 문제를 수정했습니다.
- 압축되지 않은 출력–CREATE TABLE문은 이제 압축되지 않은 파일 쓰기를 지원합니다. 압축되지 않은 파일을 작성하려면 다음 구문을 사용합니다.
  - 테이블 만들기(텍스트 파일 또는 JSON) —TBLPROPERTIES를 지정합니다.`write.compression = NONE`.
  - 테이블 만들기(Parquet) —`parquet.compression = UNCOMPRESSED`에서 TBLPROPERTIES를 지정합니다.

  - 테이블 생성(ORC) — `orc.compress = NONE`에서 `TBLPROPERTIES`를 지정합니다.
- 압축 — 한 형식으로 압축된 파일을 만들었지만 기본값이 아닌 압축 방법을 사용할 때 다른 압축 형식 파일 확장자를 사용하는 텍스트 파일 테이블에 대한 삽입 문제가 수정되었습니다.
- Avro— Avro 파일에서 고정 형식의 소수 자릿수를 읽을 때 발생하는 문제가 해결되었습니다.

2022년 3월 2일

2022년 3월 2일 게시

Athena에서 다음 기능과 개선 사항을 발표했습니다.

- 이제 쿼리 결과 버킷에 ACL이 사용될 때 Amazon S3 버킷 소유자에게 쿼리 결과에 대한 전체 제어 액세스 권한을 부여할 수 있습니다. 자세한 내용은 쿼리 결과 위치 지정 단원을 참조하십시오.
- 이제 기존 명명된 쿼리를 업데이트할 수 있습니다. 자세한 내용은 저장된 쿼리 사용 단원을 참조하십시오.

2022년 2월 23일

2022년 2월 23일 게시

Athena에서 다음 기능과 개선 사항을 발표했습니다.

- 메모리 처리가 개선되어 성능을 향상시키고 메모리 오류를 줄입니다.
- Athena가 이제 스트라이프 바닥글에 저장된 표준 시간대 정보가 포함된 ORC 타임스탬프 열을 읽고 바닥글에 표준 시간대 (UTC) 가 있는 ORC 파일을 씁니다. 이 변경 사항은 읽을 ORC 파일이 아닌 UTC 표준 시간대 환경에서 만들어진 경우에만 ORC 타임스탬프 읽기 동작에 영향을 줍니다.
- 잘못된 심볼릭 링크 테이블 크기 추정치를 수정하여 차선의 쿼리 계획이 생성되었습니다.
- 이제 Hive 메타스토어 데이터 소스에서 Athena 콘솔에서 측면 분해 뷰를 쿼리할 수 있습니다.
- Amazon S3 읽기 오류 메시지가 더 자세한 Amazon S3 오류 코드를 포함하도록 개선되었습니다.
- ORC 형식의 출력 파일이 Apache Hive 3.1과 호환되지 않는 문제를 수정했습니다.
- 특정 DML 및 DDL 쿼리에서 따옴표가 있는 테이블 이름이 실패하는 문제를 수정했습니다.

2022년 2월 15일

2022년 2월 15일 게시

Amazon Athena에서 모든 AWS 리전의 활성 DML 쿼리 할당량이 늘어났습니다. 활성 쿼리는 실행 중인 쿼리와 대기 중인 쿼리를 모두 포함합니다. 이 변경으로 이제 이전보다 더 많은 DML 쿼리를 활성 상태로 만들 수 있습니다.

Athena 서비스 할당량에 대한 자세한 내용은 Service Quotas 단원을 참조하세요. Athena를 사용하는 리전의 쿼리 할당량은 AWS 일반 참조의 Amazon Athena endpoints and quotas를 참조하세요.

할당량 사용량을 모니터링하기 위해 CloudWatch 사용량 지표를 사용할 수 있습니다. Athena는 `AWS/Usage` 네임스페이스에 다음 `ActiveQueryCount` 지표를 게시합니다. 자세한 내용은 Athena 사용량 지표 모니터링 단원을 참조하십시오.

사용량을 검토한 후 Service Quotas 콘솔을 사용하여 할당량 증가를 요청할 수 있습니다. 이전에 계정에 대한 할당량 증가를 요청한 경우에도 요청된 할당량이 새 기본 활성 DML 쿼리 할당량을 초과하면 계속 적용됩니다. 그렇지 않으면 모든 계정이 새 기본값을 사용합니다.

2022년 2월 14일

2022년 2월 14일 게시

이 릴리스는 Athena GetQueryExecution API 작업의 AthenaError 응답 객체에 `ErrorType` 하위 필드를 추가합니다.

기존 `ErrorCategory` 필드는 실패한 쿼리(시스템, 사용자 또는 기타)의 일반 소스를 나타내며, `ErrorType` 필드는 발생한 오류에 대한 더 세부적인 정보를 제공합니다. 두 필드의 정보를 결합하여 쿼리 실패의 원인을 파악할 수 있습니다.

자세한 내용은 Athena 오류 카탈로그 단원을 참조하십시오.

2022년 2월 9일

2022년 2월 9일 게시

이전 Athena 콘솔은 더 이상 사용할 수 없습니다. Athena의 새로운 콘솔은 이전 콘솔의 모든 기능을 지원하지만 사용하기 쉽고 현대적인 인터페이스를 제공하며 쿼리 개발, 데이터 분석 및 사용 관리 경험을 향상시키는 새로운 기능을 포함합니다. 새로운 Athena 콘솔을 사용하려면 https://console.aws.amazon.com/athena/를 방문하세요.

2022년 2월 8일

2022년 2월 8일 게시

예상 버킷 소유자 – 추가된 보안 조치로 이제 Athena에서 쿼리 결과 출력 위치 버킷의 소유자가 될 것으로 예상되는 AWS 계정 ID를 선택적으로 지정할 수 있습니다. 쿼리 결과 버킷 소유자의 계정 ID가 지정한 계정 ID와 일치하지 않는 경우 버킷에 대한 출력 시도는 Amazon S3 권한 오류와 함께 실패합니다. 클라이언트 또는 작업 그룹 수준에서 이 설정을 지정할 수 있습니다.

자세한 내용은 쿼리 결과 위치 지정 단원을 참조하십시오.

2022년 1월 28일

2022년 1월 28일 게시

Athena에서 다음과 같은 엔진 기능 향상을 발표합니다.

- Apache Hudi - Hudi MoR(Merge on Read) 테이블에 대한 스냅샷 쿼리는 이제 `INT64` 데이터 형식이 있는 타임스탬프 열을 읽을 수 있습니다.
- UNION 쿼리 - 동일한 테이블을 여러 번 스캔하는 특정 `UNION` 쿼리에 대한 성능이 향상되고 데이터 스캔이 감소했습니다.
- 분리 쿼리 - 필터의 각 파티션 열에 대해 분리 값만 있는 쿼리의 성능이 향상되었습니다.
- 파티션 프로젝션 향상
  - 이제 `injected` 유형의 열에 대한 필터 조건에 다중 분리 값이 허용됩니다. 자세한 내용은 삽입 형식 단원을 참조하십시오.
  - 필터에 분리 값만 있는 `CHAR` 또는 `VARCHAR`와 같은 문자열 기반 유형의 열에 대한 성능이 향상되었습니다.

2022년 1월 13일

2022년 1월 13일 게시

Athena용 JDBC 2.0.27 및 ODBC 1.1.15 드라이버를 릴리스했습니다.

JDBC 2.0.27 드라이버에는 다음과 같은 변경 사항이 포함되어 있습니다.

- 외부 카탈로그를 검색하도록 드라이버가 업데이트되었습니다.
- 확장된 드라이버 버전 번호는 이제 Athena API 호출의 일부로 사용자 `user-agent` 문자열에 포함됩니다.

ODBC 1.1.15 드라이버에는 다음과 같은 변경 사항이 포함되어 있습니다.

- SQLParamData()에 대한 두 번째 호출 문제를 수정합니다.

이러한 변경 사항에 대해 자세히 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결 및 ODBC로 Amazon Athena에 연결 섹션을 참조하세요.

2021년 Athena 릴리스 정보

2021년 11월 26일

2021년 11월 26일 게시

Athena는 Athena의 SQL 데이터 조작 언어(DML)에 쓰기, 삭제, 업데이트 및 시간 이동 작업을 추가하는 Athena ACID 트랜잭션의 공개 미리 보기를 발표했습니다. Athena ACID 트랙젝션을 통해 여러 명의 동시 사용자가 Amazon S3 데이터를 행 수준에서 안정적으로 수정할 수 있습니다. Apache Iceberg 테이블 형식을 기반으로 구축된 Athena ACID 트랜잭션은 Iceberg 테이블 형식도 지원하는 Amazon EMR 및 Apache Spark와 같은 다른 서비스 및 엔진과 호환됩니다.

Athena ACID 트랜잭션과 친숙한 SQL 구문은 비즈니스 및 규제 데이터에 대한 업데이트를 간소화합니다. 예를 들어 데이터 삭제 요청에 응답하려면 SQL DELETE 작업을 수행할 수 있습니다. 레코드를 수동으로 수정하려면 단일 UPDATE 문을 사용할 수 있습니다. 최근에 삭제된 데이터를 복구하려면 SELECT 문을 사용하여 시간 이동 쿼리를 실행할 수 있습니다. Athena 트랜잭션은 Athena 콘솔, API 작업, ODBC 및 JDBC 드라이버를 통해 사용할 수 있습니다.

자세한 내용은 Athena ACID 트랜잭션 사용 단원을 참조하십시오.

2021년 11월 24일

2021년 11월 24일 게시

Athena는 ZStandard 압축 COR, Parquet 및 텍스트 파일 데이터 읽기 및 쓰기를 지원한다고 발표합니다. ZStandard 압축 데이터를 작성할 때 Athena는 ZStandard 압축 레벨 3을 사용합니다.

Athena의 데이터 압축에 대한 자세한 내용은 Athena 압축 지원 섹션을 참조하세요.

2021년 11월 22일

2021월 11월 22일 게시

이제 Amazon Athena 콘솔에서 AWS Step Functions 워크플로를 관리하여, 확장 가능한 데이터 처리 파이프라인을 구축하고, 사용자 지정 비즈니스 로직을 기반으로 쿼리를 실행하고, 관리 및 경고 작업을 자동화하는 등의 작업을 쉽게 수행할 수 있습니다.

Step Functions은 이제 Athena 업그레이드된 콘솔과 통합되며, 이를 사용하여 Athena를 호출하는 상태 머신의 대화형 워크플로 다이어그램을 볼 수 있습니다. 시작하려면 왼쪽 탐색 패널에서 워크플로(Workflows)를 선택합니다. Athena 쿼리가 있는 기존 상태 머신이 있는 경우 상태 머신을 선택하여 워크플로의 대화형 다이어그램을 봅니다. Step Functions를 처음 사용하는 경우 Athena 콘솔에서 샘플 프로젝트를 시작하고 사용 사례에 맞게 사용자 지정하여 시작할 수 있습니다.

자세한 내용은 Amazon Athena 및 AWS Step Functions를 사용하여 ETL 파이프라인을 구축하고 오케스트레이션하기 또는 Step Functions 설명서를 참조하세요.

2021년 11월 18일

2021월 11월 18일 게시

Athena에서 새로운 기능과 개선 사항을 발표했습니다.

- 다음 예와 같이 DISTINCT, ORDER BY 또는 둘 다 포함하는 집계 쿼리에 대한 디스크 유출을 지원합니다.

```
SELECT array_agg(orderstatus ORDER BY orderstatus)
FROM orders
GROUP BY orderpriority, custkey
```

- DISTINCT를 사용하는 쿼리에 대한 메모리 처리 문제를 해결했습니다. DISTINCT 쿼리를 사용할 때 이 스케일 팩터에서 소진된 리소스 쿼리 오류 메시지를 피하려면 DISTINCT에 대한 카디널리티가 낮은 열을 선택하거나 쿼리의 데이터 크기를 줄입니다.
- 특정 열을 지정하지 않는 SELECT COUNT(*) 쿼리에서 행 버퍼링 없이 카운트만 유지하여 성능 및 메모리 사용량을 개선했습니다.
- 다음과 같은 문자열 함수를 도입했습니다.
  - translate(source, from, to) - to 문자열에서 해당 문자를 대체하는 from 문자열에서 발견되는 문자를 포함하는 source 문자열을 반환합니다. from 문자열에 중복이 포함되어 있으면 첫 번째 문자열만 사용됩니다. source 문자가 from 문자열에 없으면 source 문자가 변환 없이 복사됩니다. from 문자열에 일치하는 문자의 인덱스가 to 문자열의 길이보다 길면 문자가 결과 문자열에서 생략됩니다.
  - concat_ws(string0, array(varchar)) - string0을 구분자로 사용하여 배열의 요소 연결을 반환합니다. string0이 null이면 반환 값도 null입니다. 배열의 모든 null 값을 건너뜁니다.

- `struct`에서 누락된 하위 필드에 액세스하려고 할 때 쿼리가 실패하는 버그가 수정되었습니다. 이제 쿼리가 누락된 하위 필드에 대해 null을 반환합니다.
- 10진수 데이터 형식에 대한 일관성 없는 해싱 문제가 수정되었습니다.
- 파티션에 열이 너무 많을 때 리소스가 소모되던 문제가 수정되었습니다.

2021년 11월 17일

2021월 11월 17일 게시

Amazon Athena는 이제 파티션 인덱싱을 지원하여 AWS Glue Data Catalog의 분할된 테이블에 대한 쿼리를 가속화합니다.

분할된 테이블을 쿼리할 때 Athena는 사용 가능한 테이블 파티션을 검색하고 쿼리와 관련된 하위 집합으로 필터링합니다. 새 데이터와 파티션이 추가되면 파티션을 처리하는 데 더 많은 시간이 필요하며 쿼리 런타임이 늘어날 수 있습니다. 파티션 처리를 최적화하고 고도로 분할된 테이블에 대한 쿼리 성능을 향상시키기 위해 Athena는 이제 AWS Glue 파티션 인덱스를 지원합니다.

자세한 내용은 AWS Glue 파티션 인덱싱 및 필터링 단원을 참조하십시오.

2021년 11월 16일

2021월 11월 16일 게시

새롭고 향상된 Amazon Athena 콘솔은 이제 Athena를 사용할 수 있는 AWS 상용 및 GovCloud 리전에서 일반적으로 사용할 수 있습니다. Athena의 새로운 콘솔은 이전 콘솔의 모든 기능을 지원하지만 사용하기 쉽고 현대적인 인터페이스를 제공하며 쿼리 개발, 데이터 분석 및 사용 관리 경험을 향상시키는 새로운 기능을 포함합니다. 이제 다음 작업을 수행할 수 있습니다.

- 재설계된 조회 탭 모음에서 여러 조회 탭을 재정렬, 탐색하거나 닫을 수 있습니다.
- 향상된 SQL 및 텍스트 형식 지정을 사용하여 쿼리를 더 쉽게 읽고 편집할 수 있습니다.
- 전체 결과 집합을 다운로드하는 것 외에도 쿼리 결과를 클립보드에 복사할 수 있습니다.
- 쿼리 기록, 저장된 쿼리 및 작업 그룹을 정렬하고 표시하거나 숨길 열을 선택할 수 있습니다.
- 간소화된 인터페이스를 사용하여 클릭 수를 줄이면서 데이터 소스와 작업 그룹을 구성할 수 있습니다.
- 쿼리 결과, 쿼리 기록, 줄 바꿈 등을 표시하기 위한 기본 설정을 지정할 수 있습니다.
- 새롭고 향상된 키보드 바로 가기와 포함된 제품 설명서로 생산성을 높일 수 있습니다.

오늘 발표로 새롭게 디자인된 콘솔이 이제 기본이 되었습니다. 콘솔의 왼쪽 하단에서 피드백(Feedback)을 선택하여 귀하의 경험에 대해 의견을 보내주세요.

원하는 경우 왼쪽의 탐색 패널에서 AWS 계정에 로그인하고, Amazon Athena를 선택한 다음 새로운 Athena 경험(New Athena experience)을 선택 취소하여 기존 콘솔을 사용할 수 있습니다.

2021년 11월 12일

2021월 11월 12일 게시

이제 Amazon Athena를 사용하여 자신의 계정 이외의 AWS 계정에 위치한 데이터 소스에 대한 연동 쿼리를 실행할 수 있습니다. 지금까지 이 데이터를 쿼리하려면 데이터 소스와 사용자가 데이터를 쿼리한 것과 같은 AWS 계정을 사용하는 커넥터가 필요했습니다.

데이터 관리자는 데이터 커넥터를 데이터 분석가의 계정과 공유하여 교차 계정 연합 쿼리를 사용 설정할 수 있습니다. 데이터 분석가는 데이터 관리자가 공유한 데이터 커넥터를 계정에 추가할 수 있습니다. 원래 계정의 커넥터에 대한 구성 변경 사항은 공유 커넥터에 자동으로 적용됩니다.

지원하는 교차 계정 연합 쿼리에 대한 자세한 내용은 계정 간 연합 쿼리 사용 설정 섹션을 참조하세요. 연합 소스 쿼리에 대한 자세한 내용은 Amazon Athena 연합 쿼리 사용 및 페더레이션된 쿼리 실행 섹션을 참조하세요.

2021년 11월 2일

2021월 11월 2일 게시

이제 `EXPLAIN ANALYZE` 문을 사용하여 SQL 쿼리에 대한 분산 실행 계획 및 각 작업의 비용을 확인할 수 있습니다.

자세한 내용은 Athena에서 EXPLAIN 및 EXPLAIN ANALYZE 사용 단원을 참조하십시오.

2021년 10월 29일

2021년 10월 29일 발표

Athena는 JDBC 2.0.25 및 ODBC 1.1.13 드라이버를 출시하고 기능과 개선 사항을 발표합니다.

JDBC 및 ODBC 드라이버

Athena용 JDBC 2.0.25 및 ODBC 1.1.13 드라이버를 릴리스했습니다. 두 드라이버 모두 브라우저 SAML 멀티 팩터 인증을 지원하며, SAML 2.0 공급자와 함께 작동하도록 구성할 수 있습니다.

JDBC 2.0.25 드라이버에는 다음과 같은 변경 사항이 포함되어 있습니다.

- 브라우저 SAML 인증을 지원합니다. 드라이버에는 SAML 2.0 공급자와 작동하도록 구성할 수 있는 브라우저 SAML 플러그 인이 포함되어 있습니다.
- AWS Glue API 호출을 지원합니다. `GlueEndpointOverride` 파라미터를 사용하여 AWS Glue 엔드포인트를 재정의할 수 있습니다.
- `com.simba.athena.amazonaws` 클래스 경로를 `com.amazonaws`로 변경했습니다.

ODBC 1.1.13 드라이버에는 다음과 같은 변경 사항이 포함되어 있습니다.

- 브라우저 SAML 인증을 지원합니다. 드라이버에는 SAML 2.0 공급자와 작동하도록 구성할 수 있는 브라우저 SAML 플러그 인이 포함되어 있습니다. ODBC 드라이버와 함께 브라우저 SAML 플러그 인을 사용하는 방법에 대한 예는 ODBC, SAML 2.0 및 Okta 자격 증명 공급자를 사용하여 통합 인증 구성 섹션을 참조하세요.
- 이제 인증을 위해 ADFS, Azure AD 또는 브라우저 Azure AD를 사용할 때 역할 세션 기간을 구성할 수 있습니다.

이러한 변경 사항과 기타 변경 사항에 대해 자세히 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결 및 ODBC로 Amazon Athena에 연결 섹션을 참조하세요.

기능 및 개선 사항

Athena에서 다음 기능과 개선 사항을 발표했습니다.

- 경우에 따라 중복 테이블 스캔을 방지하기 위한 최적화 규칙이 도입되었습니다.

2021년 10월 4일

2021-10-04 발표

Athena에서 다음 기능과 개선 사항을 발표했습니다.

- SQL OFFSET - 이제 `SELECT` 문에서 SQL `OFFSET` 절이 지원됩니다. 자세한 내용은 SELECT 단원을 참조하십시오.
- CloudWatch 사용량 지표 - 이제 Athena가 `ActiveQueryCount` 지표를 `AWS/Usage` 네임스페이스에 게시합니다. 자세한 내용은 Athena 사용량 지표 모니터링 단원을 참조하십시오.
- 쿼리 계획 - 드물게, 쿼리 계획 시간 초과가 발생할 수 있는 버그가 수정되었습니다.

2021년 9월 16일

2021-09-16 발표

Athena에서 다음과 같은 새로운 기능과 개선 사항을 발표했습니다.

특성

- CTAS에서 `write_compression` 테이블 속성을 사용하여 텍스트 파일 및 JSON 압축을 지정하는 기능에 대한 지원이 추가되었습니다. Parquet 및 ORC 형식에 대해 CTAS에 `write_compression` 속성을 지정할 수도 있습니다. 자세한 내용은 CTAS 테이블 속성 단원을 참조하십시오.
- 이제 BZIP2 압축 형식으로 텍스트 파일 및 JSON 파일을 쓸 수 있도록 지원됩니다. Athena의 압축 형식에 대한 자세한 내용은 Athena 압축 지원 섹션을 참조하세요.

개선 사항

- ID 정보가 UDF Lambda 함수로 전송되지 않는 버그가 수정되었습니다.
- 분리 필터 조건과 관련된 술어 푸시다운 문제를 수정했습니다.
- 십진수 유형에 대한 해싱 문제가 수정되었습니다.
- 불필요한 통계 수집 문제를 수정했습니다.
- 일관되지 않은 오류 메시지를 제거했습니다.
- 작업자 노드에 동적 파티션 정리를 적용하여 브로드캐스트 조인 성능을 개선했습니다.
- 연합 쿼리의 경우:
  - 페더레이션 쿼리의 `CONSTRAINT_VIOLATION` 오류 발생을 줄이기 위해 구성을 변경했습니다.

2021년 9월 15일

2021-09-15 발표

이제 새롭게 디자인된 Amazon Athena 콘솔(평가판)을 사용할 수 있습니다. 새로운 Athena JDBC 드라이버가 출시되었습니다.

Athena 콘솔 평가판

이제 Athena를 사용할 수 있는 모든 AWS 리전에서 새로 디자인된 Amazon Athena 콘솔(평가판)을 사용할 수 있습니다. 새로운 콘솔은 사용하기 쉽고 현대적인 인터페이스에서 기존 콘솔의 모든 기능을 지원합니다.

새 콘솔로 전환하려면 AWS 계정에 로그인하고 Amazon Athena를 선택합니다. AWS 콘솔 탐색 모음에서 [새 콘솔로 전환(Switch to the new console)]을 선택합니다. 기본 콘솔로 돌아가려면 왼쪽의 탐색 창에서 [새로운 Athena 경험(New Athena experience)]을 선택 취소합니다.

지금 바로 새로운 콘솔을 사용해 보세요. 왼쪽 하단에서 [피드백(Feedback)]을 선택하여 귀하의 경험에 대해 의견을 보내주세요.

Athena JDBC 드라이버 2.0.24

Athena에서 Athena용 JDBC 드라이버 버전 2.0.24의 제공을 발표했습니다. 이 릴리스에서는 모든 자격 증명 공급자에 대한 프록시 지원을 업데이트합니다. 이제 드라이버에서 `NonProxyHosts` 연결 속성에 의해 지원되지 않는 모든 호스트에 대해 프록시 인증을 지원합니다.

편의상 이 릴리스에는 JDBC 드라이버의 다운로드(AWS SDK 포함 및 미포함)가 포함되어 있습니다. 이 JDBC 드라이버 버전을 사용하면 AWS-SDK와 Athena JDBC 드라이버가 프로젝트에 내장됩니다.

자세한 내용을 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결 섹션을 참조하세요.

2021년 8월 31일

2021-08-31 발표

Athena가 다음의 기능 향상 및 버그 수정을 발표했습니다.

- Athena 페더레이션 향상 - Athena에서는 Athena Query Federation SDK의 일부로서 맵 유형에 대한 지원을 추가하고 복잡한 유형에 대한 지원을 개선했습니다. 이 버전에는 몇 가지 메모리 향상 및 성능 최적화 기능도 포함되어 있습니다.
- 새 오류 범주 - 오류 메시지의 `USER` 및 `SYSTEM` 오류 범주가 도입되었습니다. 이러한 범주는 사용자가 직접 수정할 수 있는 오류(`USER`)와 Athena Support의 도움이 필요한 오류(`SYSTEM`)를 구분하는 데 도움이 됩니다.
- 연합 쿼리 오류 메시징 - 연합 쿼리 관련 오류의 업데이트된 `USER_ERROR` 분류입니다.
- JOIN - `JOIN` 작업의 성능을 높이고 메모리 오류를 줄이기 위해 디스크 유출 관련 버그 및 메모리 문제를 수정했습니다.

2021년 8월 12일

2021-08-12 발표

Athena용 ODBC 1.1.12 드라이버를 릴리스했습니다. 이 버전은 `SQLPrepare()`, `SQLGetInfo()` 및 `EndpointOverride`와 관련된 문제를 해결합니다.

새 드라이버, 릴리스 정보 및 문서를 다운로드하려면 ODBC로 Amazon Athena에 연결 섹션을 참조하세요.

2021년 8월 6일

2021-08-06 발표

Amazon Athena가 아시아 태평양(오사카) 리전에서 Athena 출시와 그 기능을 발표했습니다.

이 릴리스에서는 아시아 태평양(홍콩), 아시아 태평양(뭄바이), 아시아 태평양(오사카), 아시아 태평양(서울), 아시아 태평양(싱가포르), 아시아 태평양(시드니), 아시아 태평양(도쿄) 등 아시아 태평양 리전에서의 Athena 서비스 제공을 확대했습니다. 해당 및 기타에서 이용 가능한 AWS 서비스의 전체 목록은 AWS 리전 서비스 목록을 참조하세요.

2021년 8월 5일

2021-08-05 발표

`UNLOAD` 문을 사용해 `SELECT` 쿼리의 출력을 PARQUET, ORC, AVRO, JSON 형식으로 쓸 수 있습니다.

자세한 내용은 UNLOAD 단원을 참조하세요.

2021년 7월 30일

2021-07-30 발표

Athena가 다음의 기능 향상 및 버그 수정을 발표했습니다.

- 동적 필터링 및 파티션 정리 – 개선으로 다음 예제와 같이 성능이 향상되고 특정 쿼리에서 스캔되는 데이터 양이 감소되었습니다.

  이 예제는 `Table_B`가 총 20MB 미만의 파일 크기를 갖는 분할되지 않은 테이블이라고 가정합니다. 이와 같은 쿼리는 `Table_A`에서 더 적은 데이터를 읽고 쿼리가 더 빨리 완료됩니다.

```
SELECT *
FROM Table_A
```

```
JOIN Table_B ON Table_A.date = Table_B.date
WHERE Table_B.column_A = "value"
```

- LIMIT을 사용한 ORDER BY, LIMIT을 사용한 DISTINCT - `ORDER BY` 또는 `DISTINCT`에 이어 `LIMIT` 절을 사용하는 쿼리의 성능이 개선되었습니다.
- S3 Glacier Deep Archive 파일 - Athena가 S3 Glacier Deep Archive 파일과 S3 Glacier 외의 파일이 혼합된 테이블을 쿼리할 때 이제 Athena가 S3 Glacier Deep Archive 파일을 건너뜁니다. 이전에는 쿼리 위치에서 이러한 파일을 수동으로 이동해야 했습니다. 그렇지 않으면 쿼리가 실패했습니다. Athena를 사용하여 S3 Glacier Deep Archive 스토리지의 객체를 쿼리하려면 이를 복원해야 합니다. 자세한 내용은 Amazon S3 사용 설명서의 아카이브된 객체 복원을 참조하세요.
- CTAS `bucketed_by` 테이블 속성에 의해 생성된 빈 파일이 올바르게 암호화되지 않는 버그를 수정했습니다.

2021년 7월 21일

2021-07-21 발표

2021년 7월의 Microsoft Power BI Desktop 릴리스부터 Amazon Athena용 데이터 소스 커넥터를 사용하여 보고서와 대시보드를 만들 수 있습니다. 이 Amazon Athena용 커넥터는 Power BI에서 표준 커넥터로 사용 가능하고, DirectQuery를 지원하며, 대규모 데이터 세트에 대한 분석과 Power BI Gateway를 통한 콘텐츠 새로 고침이 가능합니다.

커넥터는 기존 ODBC 데이터 소스 이름(DSN)을 사용하여 Athena에 연결하고 쿼리를 실행하기 때문에 Athena ODBC 드라이버가 필요합니다. 최신 ODBC 드라이버를 다운로드하려면 ODBC로 Amazon Athena에 연결 단원을 참조하세요.

자세한 내용은 Amazon Athena Power BI 커넥터 사용 단원을 참조하세요.

2021년 7월 16일

2021-07-16 발표

Amazon Athena가 업데이트를 통해 Apache Hudi를 통합했습니다. Hudi는 Amazon S3 데이터 레이크에서 증분 데이터 처리를 간소화하는 데 사용되는 오픈 소스 데이터 관리 프레임워크입니다. 업데이트된 통합으로 사용자는 Athena를 사용하여 Amazon EMR, Apache Spark, Apache Hive 또는 기타 호환 서비스를 통해 관리되는 Hudi 0.8.0 테이블을 쿼리할 수 있습니다. 또한 Athena는 이제 두 가지 추가 기능을 지원합니다. 즉, MOR(읽을 때 병합) 테이블에 대한 스냅샷 쿼리와 부트스트랩 테이블에 대한 읽기 지원입니다.

Apache Hudi는 레코드 수준의 데이터 처리를 제공함으로써 변경 데이터 캡쳐(CDC) 파이프라인 개발을 간소화하고, GDPR에 기반한 업데이트 및 삭제를 준수하고, 데이터 삽입 및 이벤트 업데이트가 필요한 센서나 장치로부터 받은 스트리밍 데이터를 더욱 잘 관리할 수 있습니다. 0.8.0 릴리스를 사용하면 데이터를 복사하지 않고도 큰 Parquet 테이블을 Hudi로 쉽게 마이그레이션할 수 있어 Athena를 통한 쿼리와 분석이 가능합니다. Athena에서 새롭게 지원하는 스냅샷 쿼리를 사용하면 스트리밍 테이블 업데이트를 거의 실시간으로 볼 수 있습니다.

Athena에서 Hudi를 사용하는 방법에 대해 알아보려면 Athena를 사용하여 Apache Hudi 데이터 집합 쿼리 단원을 참조하세요.

2021년 7월 8일

2021-07-08 발표

Athena용 ODBC 1.1.11 드라이버를 릴리스했습니다. 이제 ODBC 드라이버가 JSON 웹 토큰(JWT)을 사용하여 연결을 인증할 수 있습니다. Linux에서는 Workgroup 속성의 기본값이 기본(Primary)으로 설정되어 있습니다.

자세한 내용을 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 ODBC로 Amazon Athena에 연결 단원을 참조하세요.

2021년 7월 1일

2021-07-01 발표

2021년 7월 1일에 미리보기 Workgroup의 특별 처리가 종료되었습니다. `AmazonAthenaPreviewFunctionality` Workgroup은 그 이름을 유지하지만 더 이상 특별한 상태를 갖지 않습니다. 사용자는 `AmazonAthenaPreviewFunctionality` Workgroup을 계속 사용하여 쿼리를 보고, 수정하고, 구성하고, 실행할 수 있습니다. 그러나 이전에 미리 보기에 있던 기능을 사용하는 쿼리는 이제 표준 Athena 결제 약관을 적용 받습니다. 결제에 대한 자세한 내용은 Amazon Athena 요금을 참조하세요.

2021년 6월 23일

2021-06-23 발표

Athena용 JDBC 2.0.23 및 ODBC 1.1.10 드라이버를 릴리스했습니다. 두 드라이버는 향상된 읽기 성능을 제공하며 EXPLAIN 문 및파라미터화된 쿼리를 지원합니다.

`EXPLAIN` 문은 SQL 쿼리의 논리적 실행 계획 또는 분산 실행 계획을 보여 줍니다. 파라미터화된 쿼리를 사용하면 런타임에 제공되는 다양한 값을 사용해 동일한 쿼리를 여러 번 사용할 수 있습니다.

또한 JDBC 릴리스는 Active Directory Federation Services 2019 지원과 AWS STS에 대한 사용자 지정 엔드포인트 재정의 옵션의 지원이 추가되었습니다. ODBC 릴리스는 IAM 프로파일 자격 증명 관련 문제를 해결합니다.

자세한 내용을 알아보고 새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결 및 ODBC로 Amazon Athena에 연결 단원을 참조하세요.

2021년 5월 12일

2021-05-12 발표

이제 Amazon Athena를 사용하여 자신의 계정이 아닌 다른 계정에서 AWS Glue 카탈로그를 등록할 수 있습니다. AWS Glue에 요구되는 IAM 권한을 구성하면 Athena를 사용해 계정 간 쿼리를 실행할 수 있습니다.

자세한 내용은 다른 계정의 AWS Glue Data Catalog 등록 및 AWS Glue 데이터 카탈로그에 대한 교차 계정 액세스 단원을 참조하세요.

2021년 5월 10일

2021-05-10 발표

Athena용 ODBC 드라이버 버전 1.1.9.1001을 릴리스했습니다. 이 버전은 Azure Active Directory(AD) 사용 시 `BrowserAzureAD` 인증 유형과 관련된 문제를 수정합니다.

새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 ODBC로 Amazon Athena에 연결 단원을 참조하세요.

2021년 5월 5일

2021-05-05 발표

이제 연합 쿼리에 Amazon Athena Vertica 커넥터를 사용하여 Athena에서 Vertica 데이터 소스를 쿼리할 수 있습니다. 예를 들어 Vertica의 데이터 웨어하우스와 Amazon S3의 데이터 레이크에 대해 분석 쿼리를 실행할 수 있습니다.

Athena Vertica 커넥터를 배포하려면 AWS Serverless Application Repository의 AthenaVerticaConnector 페이지를 방문하세요.

Amazon Athena Vertica 커넥터는 Lambda 환경 변수를 통해 여러 가지 구성 옵션을 제공합니다. 구성 옵션, 파라미터, 연결 문자열, 배포 및 제한 사항에 대한 자세한 내용은 Amazon Athena Vertica 커넥터 단원을 참조하세요.

Vertica 커넥터 사용에 대한 자세한 내용은 AWS Big Data Blog의 Querying a Vertica data source in Amazon Athena using the Athena Federated Query SDK를 참조하세요.

2021년 4월 30일

2021-04-30 발표

Athena용 JDBC 2.0.21 및 ODBC 1.1.9 드라이버를 릴리스했습니다. 두 릴리스 모두 Azure Active Directory(AD)를 사용한 SAML 인증 및 PingFederate를 사용한 SAML 인증을 지원합니다. JDBC 릴리스는 파라미터화된 쿼리도 지원합니다. Athena의 파라미터화된 쿼리에 대한 자세한 내용은 파라미터화된 쿼리 사용 단원을 참조하세요.

새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결 및 ODBC로 Amazon Athena에 연결 단원을 참조하세요.

2021년 4월 29일

2021-04-29 발표

Amazon Athena가 중국 (베이징) 및 중국(닝샤) 리전에서 Athena 엔진 버전 2 출시를 발표했습니다.

Athena 엔진 버전 2에 대한 자세한 내용은 Athena 엔진 버전 2 단원을 참조하세요.

2021년 4월 26일

2021-04-26 발표

Athena 엔진 버전 2의 윈도 값 함수가 이제 `IGNORE NULLS` 및 `RESPECT NULLS`를 지원합니다.

자세한 내용은 Presto 설명서의 Value Functions를 참조하세요.

2021년 4월 21일

2021-04-21 발표

Amazon Athena가 유럽(밀라노) 및 아프리카(케이프타운) 리전에서 Athena 엔진 버전 2 출시를 발표했습니다.

Athena 엔진 버전 2에 대한 자세한 내용은 Athena 엔진 버전 2 단원을 참조하세요.

2021년 4월 5일

2021-04-05 발표

EXPLAIN 문

이제 Athena에서 `EXPLAIN` 문을 사용하여 SQL 쿼리의 실행 계획을 볼 수 있습니다.

자세한 내용은 Athena에서 EXPLAIN 및 EXPLAIN ANALYZE 사용 및 Athena EXPLAIN 문 결과의 이해 단원을 참조하세요.

SQL 쿼리에서 SageMaker 기계 학습 모델 호출

이제 Amazon SageMaker를 통한 기계 학습 모델 추론을 Amazon Athena에서 누구나 사용할 수 있습니다. SQL 쿼리에 기계 학습 모델을 사용하면 SQL 쿼리에서 함수를 호출해 이상 감지, 고객 집단 분석, 시계열 예측 같은 복잡한 작업을 간소화할 수 있습니다.

자세한 내용은 Machine Learning(ML) with Amazon Athena 사용 단원을 참조하세요.

사용자 정의 함수(UDF)

이제 Athena에서 사용자 정의 함수(UDF)를 누구나 사용할 수 있습니다. UDF를 사용하여 하나의 SQL 쿼리에서 레코드 또는 레코드 그룹을 처리하는 사용자 지정 함수를 활용할 수 있습니다.

자세한 내용은 사용자 정의 함수를 사용한 쿼리 단원을 참조하세요.

2021년 3월 30일

2021-03-30 발표

Amazon Athena가 아시아 태평양(홍콩) 및 중동(바레인) 리전에서 Athena 엔진 버전 2 출시를 발표했습니다.

Athena 엔진 버전 2에 대한 자세한 내용은 Athena 엔진 버전 2 단원을 참조하세요.

2021년 3월 25일

2021-03-25 발표

Amazon Athena가 유럽(스톡홀름) 리전에서 Athena 엔진 버전 2 출시를 발표했습니다.

Athena 엔진 버전 2에 대한 자세한 내용은 Athena 엔진 버전 2 단원을 참조하세요.

2021년 3월 5일

2021-03-05 발표

Amazon Athena가 캐나다(중부), 유럽(프랑크푸르트) 및 남아메리카(상파울루) 리전에서 Athena 엔진 버전 2 출시를 발표했습니다.

Athena 엔진 버전 2에 대한 자세한 내용은 Athena 엔진 버전 2 단원을 참조하세요.

2021년 2월 25일

2021-02-25 발표

Amazon Athena가 아시아 태평양(서울), 아시아 태평양(싱가포르), 아시아 태평양(시드니), 유럽(런던), 유럽(파리) 리전에서 Athena 엔진 버전 2의 일반 공개를 발표했습니다.

Athena 엔진 버전 2에 대한 자세한 내용은 Athena 엔진 버전 2 단원을 참조하세요.

2020년 Athena 릴리스 정보

2020년 12월 16일

2020-12-16 발표

Amazon Athena가 추가 지역에서 Athena 엔진 버전 2, Athena 연합 쿼리, AWS PrivateLink 출시를 발표했습니다.

Athena 엔진 버전 2와 Athena 연합 쿼리

Amazon Athena가 아시아 태평양(뭄바이), 아시아 태평양(도쿄), 유럽(아일랜드), 미국 서부(캘리포니아 북부) 리전에서 Athena 엔진 버전 2 및 Athena 연합 쿼리의 일반 공개를 발표했습니다. Athena 엔진 버전 2 및 연합 쿼리는 미국 동부(버지니아 북부), 미국 동부(오하이오), 미국 서부(오레곤) 리전에서 이미 제공되고 있습니다.

자세한 내용은 Athena 엔진 버전 2 및 Amazon Athena 연합 쿼리 사용 단원을 참조하세요.

AWS PrivateLink

이제 유럽(스톡홀름) 리전에서 Athena용 AWS PrivateLink가 지원됩니다. Athena용 AWS PrivateLink에 대한 자세한 내용은 인터페이스 VPC 엔드포인트를 사용하여 Amazon Athena에 연결 단원을 참조하세요.

2020년 11월 24일

2020-11-24 발표

Athena용 JDBC 2.0.16 및 ODBC 1.1.6 드라이버를 릴리스했습니다. 이 릴리스는 계정 수준에서 Okta Verify 멀티 팩터 인증(MFA)을 지원합니다. Okta MFA를 사용하여 SMS 인증 및 Google Authenticator 인증을 요인으로 구성할 수도 있습니다.

새 드라이버, 릴리스 정보 및 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결 및 ODBC로 Amazon Athena에 연결 단원을 참조하세요.

2020년 11월 11일

2020-11-11 발표

Amazon Athena가 미국 동부(버지니아 북부), 미국 동부(오하이오) 및 미국 서부(오레곤) 리전에서 Athena 엔진 버전 2 및 연합 쿼리의 일반 공개를 발표했습니다.

Athena 엔진 버전 2

Amazon Athena가 미국 동부(버지니아 북부), 미국 동부(오하이오) 및 미국 서부(오레곤) 리전에서 새로운 쿼리 엔전 버전인 Athena 엔진 버전 2의 일반 공개를 발표했습니다.

Athena 엔진 버전 2에는 Parquet 형식 데이터에 대한 스키마 변화 지원, 추가 지리 공간 함수, 비용 절감을 위한 중첩 스키마 읽기 지원, JOIN 및 AGGREGATE 작업의 성능 향상과 같은 성능 향상 및 새로운 기능이 포함되었습니다.

- 개선 사항, 주요 변경 사항, 버그 수정에 대한 자세한 내용은Athena 엔진 버전 2를 참조하세요.
- 업그레이드 방법에 대한 자세한 내용은 Athena 엔진 버전 변경 단원을 참조하세요.
- 쿼리 테스트에 대한 자세한 내용은 엔진 버전 업그레이드 전 쿼리 테스트 단원을 참조하세요.

연합 SQL 쿼리

이제 미국 동부(버지니아 북부), 미국 동부(오하이오) 및 미국 서부(오레곤) 리전에서 `AmazonAthenaPreviewFunctionality` Workgroup의 사용 없이 Athena의 연합 쿼리를 사용할 수 있습니다.

연합 SQL 쿼리를 사용하여 관계형, 비관계형, 객체 및 사용자 지정 데이터 소스에서 SQL 쿼리를 실행할 수 있습니다. 연합 쿼리를 통해 온프레미스에서 실행되거나 클라우드에서 호스팅되는 여러 원본의 데이터를 스캔하는 단일 SQL 쿼리를 제출할 수 있습니다.

여러 애플리케이션에 걸쳐 분산된 데이터에 대한 분석을 실행하는 작업은 다음과 같은 이유로 복잡하고 시간이 많이 소요될 수 있습니다.

- 분석에 필요한 데이터는 관계형, 키-값, 문서, 인 메모리, 검색, 그래프, 객체, 시계열 및 원장 데이터 스토어에 분산되어 있는 경우가 많습니다.
- 이러한 원본에 분산되어 있는 데이터를 분석하기 위해 분석가들은 데이터를 쿼리할 수 있도록 추출 및 변환하고 데이터 웨어하우스로 로드하는 복잡한 파이프라인을 구축합니다.
- 여러 원본의 데이터에 액세스하려면 새로운 프로그래밍 언어와 데이터 액세스 구문을 학습해야 합니다.

Athena의 연합 SQL 쿼리를 사용하면 데이터가 있는 위치에서 인 플레이스 쿼리를 실행할 수 있기 때문에 이러한 복잡성을 없앨 수 있습니다. 분석가는 익숙한 SQL 구문을 사용하여 여러 데이터 소스의 `JOIN` 데이터를 빠르게 분석하고, 후속 사용을 위해 Amazon S3에 결과를 저장할 수 있습니다.

데이터 소스 커넥터

연합 쿼리를 처리하기 위해 Athena는 AWS Lambda에서 실행되는 Athena 데이터 소스 커넥터를 사용합니다. 다음 오픈 소스, 사전 내장 커넥터는 Athena에서 작성 및 테스트되었습니다. 이를 사용하여 Athena에서 해당 데이터 소스에 대한 SQL 쿼리를 실행할 수 있습니다.

- CloudWatch
- CloudWatch 지표
- documentdb
- DynamoDB
- OpenSearch
- HBase
- Neptune
- Redis
- Timestream
- TPC 벤치마크 DS(TPC-DS)

사용자 지정 데이터 소스 커넥터

개발자는 Athena Query Federation SDK를 사용하여 원하는 데이터 소스에 대한 커넥터를 구축함으로써 해당 데이터 소스에 대해 Athena로 SQL 쿼리를 실행할 수 있습니다. Athena Query Federation

Connector는 AWS에서 제공하는 커넥터보다 훨씬 뛰어난 연합 쿼리 기능을 제공합니다. 커넥터가 AWS Lambda에서 실행되므로 인프라를 관리하거나 최대 수요에 맞게 확장 계획을 세우지 않아도 됩니다.

다음 단계

- 연합 쿼리 기능에 대해 자세히 알아보려면 Amazon Athena 연합 쿼리 사용 단원을 참조하세요.
- 기존 커넥터 사용을 시작하려면 커넥터 배포 및 데이터 소스에 연결을 참조하세요.
- Athena Query Federation SDK를 사용하여 자체 데이터 소스 커넥터를 구축하는 방법을 알아보려면 GitHub의 Example Athena Connector를 참조하세요.

2020년 10월 22일

2020-10-22 발표

이제 AWS Step Functions로 Athena를 호출할 수 있습니다. AWS Step Functions는 Amazon States Language를 사용하여 특정한 AWS 서비스를 직접 제어할 수 있습니다. Athena와 함께 Step Functions를 사용하면 쿼리 실행을 시작 및 중지하거나, 쿼리 결과를 가져오거나, 임시 또는 예약된 데이터 쿼리를 실행하거나, Amazon S3의 데이터 레이크에서 결과를 검색할 수 있습니다.

자세한 내용은 AWS Step Functions 개발자 안내서의 Step Functions로 Athena 호출을 참조하세요.

2020년 7월 29일

2020-07-29 발표

JDBC 드라이버 버전 2.0.13을 릴리스했습니다. 이 릴리스에서는 Athena에 등록된 여러 데이터 카탈로그의 사용, 인증을 위한 Okta 서비스, VPC 엔드포인트 연결을 지원합니다.

새 드라이버 버전을 다운로드하고 사용하려면 JDBC로 Amazon Athena에 연결 단원을 참조하세요.

2020년 7월 9일

2020-07-09 발표

Amazon Athena에 압축된 Hudi 데이터 세트에 대한 쿼리 지원이 추가되었고, Athena에 등록한 데이터 카탈로그의 생성, 업데이트 또는 삭제를 위한 AWS CloudFormation `AWS::Athena::DataCatalog` 리소스가 추가되었습니다.

Apache Hudi 데이터 세트 쿼리

Apache Hudi는 증분 데이터 처리를 간소화하는 오픈 소스 데이터 관리 프레임워크입니다. Amazon Athena가 이제 Amazon S3 기반 데이터 레이크에서 Apache Hudi 데이터 세트의 읽기 최적화 뷰에 대한 쿼리를 지원합니다.

자세한 내용은 Athena를 사용하여 Apache Hudi 데이터 집합 쿼리 단원을 참조하세요.

AWS CloudFormation 데이터 카탈로그 리소스

Amazon Athena의 연합 쿼리 기능을 사용하여 데이터 소스를 쿼리하려면 먼저 Athena에서 데이터 카탈로그를 등록해야 합니다. 이제 AWS CloudFormation `AWS::Athena::DataCatalog` 리소스를 사용하여 Athena에 등록한 데이터 카탈로그를 생성, 업데이트 또는 삭제할 수 있습니다.

자세한 내용은 AWS CloudFormation 사용 설명서의 AWS::Athena::DataCatalog를 참조하세요.

2020년 6월 1일

2020년 6월 1일 발표

Amazon Athena와 함께 Apache Hive 메타스토어를 메타카탈로그로 사용

이제 Athena를 통해 AWS Glue Data Catalog 외에 하나 이상의 Apache Hive 메타스토어에 Athena를 연결할 수 있습니다.

자체 호스팅된 Hive 메타스토어에 연결하려면 Athena 하이브 메타스토어 커넥터가 필요합니다. Athena는 개발자가 사용할 수 있는 참조 구현 커넥터를 제공합니다. 이 커넥터는 계정에서 AWS Lambda 함수로 실행됩니다.

자세한 내용은 외부 Hive 메타스토어용 Athena 데이터 커넥터 사용 단원을 참조하세요.

2020년 5월 21일

2020-05-21 발표

Amazon Athena가 파티션 프로젝션에 대한 지원을 추가했습니다. 파티션 프로젝션을 사용하여 고도로 분할된 테이블의 쿼리 처리 속도를 높이고 파티션 관리를 자동화합니다. 자세한 내용은 Amazon Athena를 사용한 파티션 프로젝션 단원을 참조하세요.

2020년 4월 1일

2020년 4월 1일 발표

미국 동부(버지니아 북부) 리전에 더하여 이제 아시아 태평양(뭄바이), 유럽(아일랜드), 미국 서부(오레곤) 리전에서 Amazon Athena 연합 쿼리, 사용자 정의 함수(UDF), 기계 학습 추론, 외부 Hive 메타스토어 기능을 사용할 수 있습니다.

2020년 3월 11일

2020년 3월 11일 발표

이제 Amazon Athena가 쿼리 상태 전환을 위한 Amazon EventBridge 이벤트를 게시합니다. 쿼리 상태가 전환되면(예: 실행 중에서 성공 또는 취소됨 등의 종료 상태로 전환) Athena가 EventBridge에 쿼리 상태 변경 이벤트를 게시합니다. 이 이벤트에는 쿼리 상태 전환에 대한 정보가 포함됩니다. 자세한 내용은 Amazon EventBridge 이벤트를 사용하여 Athena 이벤트 모니터링 단원을 참조하십시오.

2020년 3월 6일

2020년 3월 6일 발표

이제 AWS CloudFormation `AWS::Athena::WorkGroup` 리소스를 사용하여 Amazon Athena Workgroup을 생성하고 업데이트할 수 있습니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 AWS::Athena::WorkGroup을 참조하세요.

2019년 Athena 릴리스 정보

2019년 11월 26일

2019년 12월 17일 발표

Amazon Athena는 관계형, 비관계형, 객체 및 사용자 지정 데이터 소스에서 SQL 쿼리 실행, SQL 쿼리에서 기계 학습 모델 호출, 사용자 정의 함수(UDF)(미리 보기), Amazon Athena와 함께 Apache Hive 메타스토어를 메타데이터 카탈로그로 사용(미리 보기) 및 네 가지 추가 쿼리 관련 지표를 추가로 지원합니다.

연합 SQL 쿼리

연합 SQL 쿼리를 사용하여 관계형, 비관계형, 객체 및 사용자 지정 데이터 소스에서 SQL 쿼리를 실행할 수 있습니다.

이제 Athena의 연합 쿼리를 사용하여 관계형, 비관계형, 개체 및 사용자 정의 데이터 소스에 저장된 데이터를 스캔할 수 있습니다. 연합 쿼리를 통해 온프레미스에서 실행되거나 클라우드에서 호스팅되는 여러 원본의 데이터를 스캔하는 단일 SQL 쿼리를 제출할 수 있습니다.

여러 애플리케이션에 걸쳐 분산된 데이터에 대한 분석을 실행하는 작업은 다음과 같은 이유로 복잡하고 시간이 많이 소요될 수 있습니다.

- 분석에 필요한 데이터는 관계형, 키-값, 문서, 인 메모리, 검색, 그래프, 객체, 시계열 및 원장 데이터 스토어에 분산되어 있는 경우가 많습니다.
- 이러한 원본에 분산되어 있는 데이터를 분석하기 위해 분석가들은 데이터를 쿼리할 수 있도록 추출 및 변환하고 데이터 웨어하우스로 로드하는 복잡한 파이프라인을 구축합니다.
- 여러 원본의 데이터에 액세스하려면 새로운 프로그래밍 언어와 데이터 액세스 구문을 학습해야 합니다.

Athena의 연합 SQL 쿼리를 사용하면 데이터가 있는 위치에서 인 플레이스 쿼리를 실행할 수 있기 때문에 이러한 복잡성을 없앨 수 있습니다. 분석가는 익숙한 SQL 구문을 사용하여 여러 데이터 소스의 `JOIN` 데이터를 빠르게 분석하고, 후속 사용을 위해 Amazon S3에 결과를 저장할 수 있습니다.

데이터 소스 커넥터

Athena는 AWS Lambda에서 실행되는 Athena 데이터 소스 커넥터를 사용하여 연합 쿼리를 처리합니다. 이러한 오픈 소스 데이터 소스 커넥터를 사용하여 Athena에서 Amazon DynamoDB, Apache HBase, Amazon Document DB, Amazon CloudWatch, Amazon CloudWatch 지표 및 JDBC 호환 관계형 데이터베이스(Apache 2.0 라이선스에 따른 MySQL, PostgreSQL 등)에 대한 연합 SQL 쿼리를 실행할 수 있습니다.

사용자 지정 데이터 소스 커넥터

개발자는 Athena Query Federation SDK를 사용하여 원하는 데이터 소스에 대한 커넥터를 구축함으로써 해당 데이터 소스에 대해 Athena로 SQL 쿼리를 실행할 수 있습니다. Athena Query Federation Connector는 AWS에서 제공하는 커넥터보다 훨씬 뛰어난 연합 쿼리 기능을 제공합니다. 커넥터가 AWS Lambda에서 실행되므로 인프라를 관리하거나 최대 수요에 맞게 확장 계획을 세우지 않아도 됩니다.

미리 보기 가용성

미국 동부(버지니아 북부) 리전에서 Athena 연합 쿼리를 미리 보기에서 사용할 수 있습니다.

다음 단계

- 미리 보기를 시작하려면 Athena 미리 보기 기능 FAQ의 지침을 따릅니다.
- 연합 쿼리 기능에 대한 자세한 내용은 Amazon Athena 연합 쿼리 사용(미리 보기)을 참조하세요.
- 기존 커넥터 사용을 시작하려면 커넥터 배포 및 데이터 소스에 연결을 참조하세요.

- Athena Query Federation SDK를 사용하여 자체 데이터 소스 커넥터를 구축하는 방법을 알아보려면 GitHub의 Example Athena Connector를 참조하세요.

SQL 쿼리에서 기계 학습 모델 호출

이제 추론을 위한 기계 학습 모델을 Athena 쿼리에서 바로 호출할 수 있습니다. SQL 쿼리에서 기계 학습 모델을 사용할 수 있는 덕분에 이상 탐지, 고객 집단 분석, 매출 예측 같은 복잡한 작업을 SQL 쿼리에서 함수를 호출하는 것처럼 간단히 수행할 수 있습니다.

ML 모델

Amazon SageMaker에 제공되는 수십 개의 기본 제공 기계 학습 알고리즘을 사용하거나, 모델을 직접 교육하거나, AWS Marketplace에서 모델 패키지를 찾아 구독하고 Amazon SageMaker 호스팅 서비스에 배포할 수 있습니다. 추가 설정은 필요하지 않습니다. Athena 콘솔, Athena API, Athena의 미리 보기 JDBC 드라이버를 통해 SQL 쿼리에서 이러한 ML 모델을 호출할 수 있습니다.

미리 보기 가용성

이제 미국 동부(버지니아 북부) 리전에서 Athena의 ML 기능을 미리 보기에서 사용할 수 있습니다.

다음 단계

- 미리 보기를 시작하려면 Athena 미리 보기 기능 FAQ의 지침을 따릅니다.
- 기계 학습 기능에 대한 자세한 내용은 Amazon Athena와 함께 기계 학습(ML) 사용(미리 보기)을 참조하세요.

사용자 정의 함수(UDF)(미리 보기)

이제 사용자 지정 스칼라 함수를 작성하고 Athena 쿼리에서 호출할 수 있습니다. Athena Query Federation SDK를 사용하여 Java에서 UDF를 작성할 수 있습니다. Athena에 제출되는 SQL 쿼리에서 UDF를 사용하면 AWS Lambda에서 UDF가 호출되어 실행됩니다. UDF는 SQL 쿼리의 `SELECT` 절과 `FILTER` 절 모두에서 사용할 수 있습니다. 동일한 쿼리에서 여러 UDF를 호출할 수 있습니다.

미리 보기 가용성

이제 미국 동부(버지니아 북부) 리전에서 Athena UDF 기능을 미리 보기 모드에서 사용할 수 있습니다.

다음 단계

- 미리 보기를 시작하려면 Athena 미리 보기 기능 FAQ의 지침을 따릅니다.

- 자세한 내용은 사용자 정의 함수로 쿼리(미리 보기)를 참조하세요.
- UDF 구현 예제는 GitHub의 Amazon Athena UDF Connector를 참조하세요.
- Athena Query Federation SDK를 사용하여 고유한 함수를 작성하는 방법은 Lambda를 사용하여 UDF 생성 및 배포를 참조하세요.

Amazon Athena를 통해 Apache Hive 메타스토어를 메타카탈로그로 사용(미리 보기)

이제 Athena를 통해 AWS Glue Data Catalog 외에 하나 이상의 Apache Hive 메타스토어에 Athena를 연결할 수 있습니다.

메타스토어 커넥터

자체 호스팅된 Hive 메타스토어에 연결하려면 Athena Hive 메타스토어 커넥터가 필요합니다. Athena는 개발자가 사용할 수 있는 참조 구현 커넥터를 제공합니다. 이 커넥터는 계정에서 AWS Lambda 함수로 실행됩니다. 자세한 내용은 외부 Hive 메타스토어에 Athena 데이터 커넥터 사용(미리 보기)을 참조하세요.

미리 보기 가용성

미국 동부(버지니아 부북) 리전에서 Hive 메타스토어 기능을 미리 보기 모드에서 사용할 수 있습니다.

다음 단계

- 미리 보기를 시작하려면 Athena 미리 보기 기능 FAQ의 지침을 따릅니다.
- 이 기능에 대한 자세한 내용은 외부 Hive 메타스토어에 Athena 데이터 커넥터 사용(미리 보기)을 참조하세요.

새 쿼리 관련 지표

이제 Athena가 Amazon Athena 성능을 파악하는 데 도움을 줄 수 있는 추가적인 쿼리 지표를 게시합니다. Athena는 쿼리 관련 지표를 Amazon CloudWatch에 게시합니다. 이 릴리스에서는 Athena가 다음과 같은 추가 쿼리 지표를 게시합니다.

- 쿼리 계획 시간 – 쿼리를 계획하는 데 소요된 시간입니다. 여기에는 데이터 소스로부터 테이블 파티션을 검색하는 데 소요된 시간이 포함됩니다.
- 쿼리 대기열 시간 – 쿼리가 대기열에서 리소스를 기다린 시간입니다.
- 서비스 처리 시간 – 쿼리 엔진이 처리를 완료한 후 결과를 작성하는 데 소요된 시간입니다.
- 총 실행 시간 – Athena에서 쿼리를 실행하는 데 소요된 시간입니다.

이러한 새로운 쿼리 지표를 사용하려면 사용자 지정 대시보드를 생성하거나, CloudWatch에서 지표에 경보와 트리거를 설정하거나, Athena 콘솔에서 미리 구성된 대시보드를 직접 사용하면 됩니다.

다음 단계

자세한 내용은 CloudWatch 지표를 사용하여 Athena 쿼리 모니터링을 참조하세요.

2019년 11월 12일

2019년 12월 17일 발표

이제 중동(바레인) 리전에서 Amazon Athena를 사용할 수 있습니다.

2019년 11월 8일

2019년 12월 17일 발표

이제 미국 서부(캘리포니아 북부) 리전과 유럽(파리) 리전에서 Amazon Athena를 사용할 수 있습니다.

2019년 10월 8일

2019년 12월 17일 발표

이제 Amazon Athena는 Virtual Private Cloud(VPC)의 인터페이스 VPC 엔드포인트를 통해 Athena에 직접 연결할 수 있습니다. 이 기능을 사용하면 VPC에서 인터넷 게이트웨이가 없어도 Athena에 쿼리를 안전하게 제출할 수 있습니다.

Athena에 연결할 인터페이스 VPC 엔드포인트를 생성하려면 AWS Management Console 또는 AWS Command Line Interface(AWS CLI)를 사용하면 됩니다. 인터페이스 엔드포인트 생성에 대한 자세한 내용은 인터페이스 엔드포인트 생성을 참조하세요.

인터페이스 VPC 엔드포인트를 사용하면 VPC API와 Athena API 간의 통신이 AWS 네트워크 내에서 안전하게 이루어집니다. 이 기능을 사용하는 데 따른 추가 Athena 비용은 없습니다. 인터페이스 VPC 엔드포인트 비용이 부과됩니다.

이 기능에 대한 자세한 내용은 인터페이스 VPC 엔드포인트를 사용하여 Amazon Athena에 연결을 참조하세요.

2019년 9월 19일

2019년 12월 17일 발표

Amazon Athena가 INSERT INTO 문을 사용하여 기존 테이블에 새 데이터를 삽입하는 기능을 추가로 지원합니다. 소스 테이블에서 실행되는 SELECT 쿼리 문 또는 쿼리 문의 일부로 제공되는 값을 기반으로 대상 테이블에 새 행을 삽입할 수 있습니다. 지원되는 데이터 형식에는 Avro, JSON, ORC, Parquet 및 Text 파일이 포함됩니다.

또한 INSERT INTO 문은 ETL 프로세스를 간소화할 수도 있습니다. 예를 들어, 단일 쿼리에서 INSERT INTO를 사용하면 소스 테이블에서 JSON 형식의 데이터를 선택하여 대상 테이블에 Parquet 형식으로 작성할 수 있습니다.

INSERT INTO 문은 Athena에서 SELECT 쿼리에 요금을 부과하는 방식과 유사하게 SELECT 단계에서 스캔한 바이트 수를 기준으로 요금이 청구됩니다. 자세한 내용은 Amazon Athena 요금을 참조하세요.

지원되는 형식, SerDes, 예제를 비롯해 INSERT INTO 사용에 관한 자세한 내용은 Athena 사용 설명서의 INSERT INTO를 참조하세요.

2019년 9월 12일

2019년 12월 17일 발표

이제 아시아 태평양(홍콩) 리전에서 Amazon Athena를 사용할 수 있습니다.

2019년 8월 16일

2019년 12월 17일 발표

Amazon Athena가 Amazon S3 요청자 지불 버킷의 데이터 쿼리를 추가로 지원합니다.

Amazon S3 버킷이 요청자 지불로 구성된 경우 버킷 소유자가 아닌 요청자가 Amazon S3 요청 및 데이터 전송 비용을 지불합니다. 이제 Workgroup 관리자는 Athena에서 Workgroup 구성원이 S3 요청자 지불 버킷을 쿼리할 수 있도록 Workgroup 설정을 구성할 수 있습니다.

Workgroup에 대한 요청자 지불 설정을 구성하는 방법에 관한 자세한 내용은 Amazon Athena 사용 설명서의 Workgroup 생성을 참조하세요. 요청자 지불 버킷에 대해 자세히 알아보려면 Amazon Simple Storage Service 개발자 안내서의 요청자 지불 버킷을 참조하세요.

2019년 8월 9일

2019년 12월 17일 발표

이제 Amazon Athena가 Amazon S3에 저장된 데이터에 대해 AWS Glue Data Catalog에 정의된 신규 또는 기존 데이터베이스, 테이블, 열의 액세스를 세분화해서 제어할 수 있는 AWS Lake Formation 정책의 시행을 지원합니다.

이 기능은 미국 동부(오하이오), 미국 동부(버지니아 북부), 미국 서부(오레곤), 아시아 태평양(도쿄), 유럽(아일랜드) AWS 리전에서 사용할 수 있습니다. 이 기능을 사용하는 데 따른 추가 비용은 없습니다.

이 기능 사용에 대한 자세한 내용은 Athena를 사용하여 AWS Lake Formation에 등록된 데이터 쿼리 단원을 참조하세요. AWS Lake Formation에 대한 자세한 정보는 AWS Lake Formation 섹션을 참조하십시오.

2019년 6월 26일

이제 유럽(스톡홀름) 리전에서 Amazon Athena를 사용할 수 있습니다. 지원되는 리전의 목록은 AWS 리전 및 엔드포인트를 참조하세요.

2019년 5월 24일

2019-05-24 발표

이제 AWS GovCloud(미국 동부) 및 AWS GovCloud(미국 서부) 리전에서 Amazon Athena를 사용할 수 있습니다. 지원되는 리전의 목록은 AWS 리전 및 엔드포인트를 참조하세요.

2019년 3월 5일

2019-03-05 발표

이제 캐나다(중부) 리전에서 Amazon Athena를 사용할 수 있습니다. 지원되는 리전의 목록은 AWS 리전 및 엔드포인트를 참조하세요. Athena Workgroup을 지원하는 ODBC 드라이버 새 버전을 릴리스했습니다. 자세한 정보는 ODBC 드라이버 출시 정보를 참조하십시오.

ODBC 드라이버 버전 1.0.5와 해당 설명서를 다운로드하려면 ODBC로 Amazon Athena에 연결 단원을 참조하세요. 이 버전에 대한 자세한 정보는 ODBC 드라이버 출시 정보를 참조하세요.

ODBC 드라이버를 통해 Workgroup을 사용하려면 다음 예제에 나와 있는 것처럼 연결 문자열에 새로운 연결 속성인 `Workgroup`을 설정합니다.

```
Driver=Simba Athena ODBC
 Driver;AwsRegion=[Region];S3OutputLocation=[S3Path];AuthenticationType=IAM
 Credentials;UID=[YourAccessKey];PWD=[YourSecretKey];Workgroup=[WorkgroupName]
```

자세한 내용을 알아보려면 ODBC 드라이버 설치 및 구성 가이드 버전 1.0.5에서 '작업 그룹'을 검색하세요. Workgroup에 태그를 사용할 경우 ODBC 드라이버 연결 문자열에 대해 변경된 사항은 없습니다. 태그를 사용하려면 ODBC 드라이버 최신 버전(현재 이 버전)으로 업그레이드하세요.

이 드라이버 버전을 사용하면 Athena API Workgroup 작업을 사용하여 Workgroup을 생성 및 관리할 수 있고, Athena API 태그 작업을 사용하여 Workgroup에서 태그를 추가하거나 태그 목록을 조회하거나 태그를 제거할 수 있습니다. 시작하려면 먼저 IAM에서 Workgroup 작업과 태그 작업에 대해 리소스 수준 권한을 가지고 있는지 확인합니다.

자세한 내용은 다음을 참조하세요.

- 쿼리 실행용 작업 그룹 사용 및 작업 그룹 정책의 예.
- Athena 리소스 태깅, , 및 태그 기반 IAM 액세스 제어 정책

JDBC 드라이버 또는 AWS SDK를 사용할 경우 최신 버전의 드라이버와 SDK를 최신 버전으로 업그레이드합니다. 최신 버전은 모두 Athena의 Workgroup과 태그에 대한 지원을 이미 포함하고 있습니다. 자세한 내용은 JDBC로 Amazon Athena에 연결 단원을 참조하세요.

2019년 2월 22일

2019-02-22 발표

Amazon Athena에서 Workgroup에 대해 태그 지원이 추가되었습니다. 각 태그는 사용자가 정의하는 키와 값으로 구성됩니다. Workgroup에 태그를 지정할 때 사용자 지정 메타데이터를 할당하게 됩니다. AWS 태그 지정 모범 사례를 참조하여 분류에 도움이 되도록 작업 그룹에 태그를 추가할 수 있습니다. 태그를 사용하여 Workgroup에 대한 액세스를 제한하고 비용을 추적할 수 있습니다. 예를 들어, 비용 센터별로 Workgroup을 생성할 수 있습니다. 그런 다음 이러한 Workgroup에 태그를 추가하여 각 비용 센터에 대해 Athena 비용을 추적할 수 있습니다. 자세한 내용은 AWS Billing and Cost Management 사용 설명서의 결제 목적으로 태그 사용을 참조하세요.

Athena 콘솔이나 API 작업을 사용하여 태그 작업을 수행할 수 있습니다. 자세한 내용은 Athena 리소스 태깅 단원을 참조하십시오.

Athena 콘솔에서 각 Workgroup에 태그를 한 개 이상 추가할 수 있고 태그별로 검색할 수 있습니다. Workgroup은 Athena에서 IAM이 제어하는 리소스입니다. IAM에서 내가 만든 Workgroup에 대해 태그를 추가하거나 제거하거나 목록을 조회할 수 있는 사용자를 제한할 수 있습니다. `CreateWorkGroup` API 작업과, 태그 파라미터(선택 사항)를 사용하여 작업에 태그를 한 개 이상 추가할 수도 있습니다. 태그를 추가하거나 제거하거나 태그 목록을 조회하려면 `TagResource`, `UntagResource`, `ListTagsForResource`를 사용합니다. 자세한 내용은 태그 작업 사용 단원을 참조하십시오.

Workgroup을 만들 때 사용자가 태그를 추가할 수 있도록 허용하려면 각 사용자에게 `TagResource` 및 `CreateWorkGroup` API 작업에 대한 IAM 권한을 부여해야 합니다. 자세한 정보와 지침은 태그 기반 IAM 액세스 제어 정책 단원을 참조하세요.

Workgroup에 태그를 사용할 경우 JDBC 드라이버에 대해 변경된 사항은 없습니다. 새 Workgroup을 생성한 후 JDBC 드라이버 또는 AWS SDK를 사용하는 경우, 최신 버전의 JDBC 드라이버 및 SDK로 업그레이드하세요. 자세한 설명은 JDBC로 Amazon Athena에 연결을 참조하세요.

2019년 2월 18일

2019-02-18 발표

Workgroup에서 쿼리를 실행하여 쿼리 비용을 제어할 수 있는 기능이 추가되었습니다. 자세한 설명은 작업 그룹을 사용하여 쿼리 액세스 및 비용 제어을 참조하세요. Athena에 사용되는 JSON OpenX SerDe가 개선되었고, Athena가 `GLACIER` 스토리지 클래스로 이전된 객체를 무시하지 못하는 문제를 수정했으며, Network Load Balancer 로그 쿼리를 위한 예제를 추가했습니다.

다음과 같은 변경이 이루어졌습니다.

- Workgroup에 대한 지원이 추가되었습니다. Workgroup을 사용하여 사용자, 팀, 애플리케이션 또는 워크로드를 구분할 수 있고, 각 쿼리 또는 전체 Workgroup에서 처리할 수 있는 데이터 양의 한도를 설정할 수 있습니다. Workgroup은 IAM 리소스 역할을 하기 때문에 리소스 수준 권한을 사용하여 특정 Workgroup에 대한 액세스를 제어할 수 있습니다. 또한 Amazon CloudWatch에서 쿼리 관련 지표를 볼 수 있고, 스캔된 데이터의 양에 대한 한도를 구성하여 쿼리 비용을 제어할 수 있으며, 임계값을 생성하여 이러한 임계값이 위반될 경우 Amazon SNS 경보와 같은 조치를 실행할 수 있습니다. 자세한 내용은 쿼리 실행용 작업 그룹 사용 및 CloudWatch 지표 및 이벤트를 사용하여 비용 관리 및 쿼리 모니터링 단원을 참조하세요.

  Workgroup은 IAM 리소스입니다. IAM의 Workgroup 관련 작업, 리소스, 조건의 전체 목록은 서비스 인증 참조의 Amazon Athena에 사용되는 작업, 리소스 및 조건 키 단원을 참조하세요. 새로운 Workgroup을 만들기 전에 Workgroup IAM 정책 및 AWS 관리형 정책: AmazonAthenaFullAccess을(를) 사용하고 있는지 확인하세요.

  콘솔에서 또는 Workgroup API 작업이나 JDBC 드라이버를 사용하여 Workgroup 사용을 시작할 수 있습니다. 상위 수준의 절차는 작업 그룹 설정 단원을 참조하세요. Workgroup을 지원하는 JDBC 드라이버를 다운로드하려면 JDBC로 Amazon Athena에 연결 단원을 참조하세요.

  JDBC 드라이버를 통해 Workgroup을 사용할 경우 다음 예제와 같이 `Workgroup` 구성 파라미터를 사용하여 연결 문자열에 Workgroup 이름을 설정해야 합니다.

  ```
  jdbc:awsathena://AwsRegion=<AWSREGION>;UID=<ACCESSKEY>;
  PWD=<SECRETKEY>;S3OutputLocation=s3://DOC-EXAMPLE-BUCKET/<athena-
  output>-<AWSREGION>/;
  Workgroup=<WORKGROUPNAME>;
  ```

SQL 문을 실행하는 방식이나 드라이버에 대해 JDBC API 호출을 수행하는 방식은 변경되지 않았습니다. 드라이버는 Workgroup 이름을 Athena에 전달합니다.

작업 그룹에 소개된 차이에 대한 정보는 Athena 작업 그룹 API 및 작업 그룹 문제 해결 단원을 참조하세요.

- Athena에 사용되는 JSON OpenX SerDe가 개선되었습니다. 이러한 개선에는 다음이 포함되며 그 밖에도 다양한 개선이 이루어졌습니다.
  - `ConvertDotsInJsonKeysToUnderscores` 속성을 지원합니다. `TRUE`로 설정하면 SerDe가 키 이름의 점을 밑줄로 바꿀 수 있습니다. 예를 들어 JSON 데이터 세트에 이름이 `"a.b"`인 키가 있을 경우 이 속성을 사용하여 Athena에서 열 이름이 `"a_b"`가 되도록 정의할 수 있습니다. 기본값은 `FALSE`입니다. 기본적으로 Athena는 열 이름에 점을 허용하지 않습니다.
  - `case.insensitive` 속성을 지원합니다. 기본적으로 Athena에서는 JSON 데이터 세트의 모든 키가 소문자를 사용해야 합니다. `WITH SERDE PROPERTIES ("case.insensitive"= FALSE;)`를 사용하면 데이터에서 대소문자를 구분하는 키 이름을 사용할 수 있습니다. 기본값은 `TRUE`입니다. `TRUE`로 설정하면 SerDe가 모든 대문자 열을 소문자로 변환합니다.

  자세한 내용은 OpenX JSON SerDe 단원을 참조하세요.
- Amazon S3 수명 주기 정책에 따라 Glacier에 아카이브된 Amazon S3 객체를 Athena가 처리할 때 `"access denied"` 오류 메시지를 반환하는 문제가 수정되었습니다. 이 문제가 해결됨에 따라 Athena는 `GLACIER` 스토리지 클래스로 이전된 객체를 무시합니다. Athena는 `GLACIER` 스토리지 클래스의 데이터에 대한 쿼리를 지원하지 않습니다.

  자세한 내용은 Amazon Simple Storage Service 사용 설명서의 the section called “Athena의 테이블 및 Amazon S3의 데이터에 대한 요구 사항” 및 GLACIER 스토리지 클래스로 이전(객체 보관)을 참조하세요.
- TLS(전송 계층 보안) 요청에 대한 정보를 받는 Network Load Balancer 액세스 로그 쿼리를 위한 예제를 추가했습니다. 자세한 내용은 the section called “Network Load Balancer” 단원을 참조하십시오.

2018년 Athena 릴리스 정보

2018년 11월 20일

2018-11-20 발표

AD FS와 SAML 2.0(Security Assertion Markup Language 2.0)을 사용하여 Athena API에 대한 연합 액세스를 지원하는 새로운 버전의 JDBC 및 ODBC 드라이버를 릴리스했습니다. 자세한 정보는 JDBC 드라이버 출시 정보 및 ODBC 드라이버 출시 정보를 참조하십시오.

이 릴리스에서는 Athena에 대한 연합 액세스가 Active Directory Federation Service(AD FS 3.0)에 지원됩니다. 액세스는 SAML 2.0을 지원하는 JDBC 또는 ODBC 드라이버 버전을 통해 설정됩니다. Athena API에 대한 연합 액세스를 구성하는 방법에 대한 자세한 정보는 the section called “Athena API에 대한 연합 액세스 활성화” 단원을 참조하세요.

JDBC 드라이버 버전 2.0.6과 해당 설명서를 다운로드하려면 JDBC로 Amazon Athena에 연결 단원을 참조하세요. 이 버전에 대한 자세한 정보는 JDBC 드라이버 출시 정보를 참조하세요.

ODBC 드라이버 버전 1.0.4와 해당 설명서를 다운로드하려면 ODBC로 Amazon Athena에 연결 단원을 참조하세요. 이 버전에 대한 자세한 정보는 ODBC 드라이버 출시 정보를 참조하세요.

AWS에서 SAML 2.0 지원에 대한 자세한 내용은 IAM 사용 설명서에서 SAML 2.0 기반 연합에 대하여를 참조하세요.

2018년 10월 15일

2018-10-15 발표

AWS Glue Data Catalog로 업그레이드했다면, 다음에 대한 지원을 제공하는 새로운 기능 2가지를 이용할 수 있습니다.

- Data Catalog 메타데이터 암호화 Data Catalog의 데이터베이스와 테이블 메타데이터를 암호화하고 싶다면, Athena에 특정 정책을 추가해야 합니다. 더 자세한 정보는 AWS Glue Data Catalog에 저장된 암호화된 메타데이터 액세스를 참조하세요.
- AWS Glue Data Catalog의 리소스 액세스를 위한 세분화된 권한 이제 Athena에서 사용하는 Data Catalog의 특정 데이터베이스 및 테이블에 대한 액세스를 제한하거나 허용하는 ID 기반(IAM) 정책을 정의할 수 있습니다. 자세한 내용은 AWS Glue Data Catalog의 데이터베이스와 테이블에 대한 세분화된 액세스 단원을 참조하십시오.

> **Note**
>
> 데이터는 Amazon S3 버킷에 상주하며 이러한 데이터에 대한 액세스는 Amazon S3에 액세스에 의해 제어됩니다. 데이터베이스와 테이블의 데이터에 액세스하려면 데이터를 저장하는 Amazon S3 버킷에 대한 액세스 제어 정책을 계속 사용합니다.

2018년 10월 10일

2018-10-10 발표

Athena가 SELECT 쿼리 문의 결과로부터 테이블을 생성하는 CREATE TABLE AS SELECT를 지원합니다. 자세한 정보는 쿼리 결과에서 테이블 생성(CTAS)을 참조하세요.

CTAS 쿼리를 생성하기 전에 Athena 설명서에서 쿼리의 동작에 대해 알아 두어야 합니다. 이 설명서에는 Amazon S3에서 쿼리를 저장하는 위치, CTAS 쿼리 결과 저장에 지원되는 형식 목록, 생성할 수 있는 파티션 수와 지원되는 압축 형식에 대한 정보가 들어 있습니다. 자세한 내용은 CTAS 쿼리에 대한 고려 사항 및 제한 사항 단원을 참조하십시오.

CTAS 쿼리를 사용해 다음 작업을 수행할 수 있습니다.

- 쿼리 결과에서 한 번에 테이블을 생성합니다.
- 예제를 활용하여 Athena 콘솔에서 CTAS 쿼리를 생성합니다. 구문에 대한 내용은 CREATE TABLE AS를 참조하세요.
- 쿼리 결과를 PARQUET, ORC, AVRO, JSON 및 TEXTFILE 등과 같은 다른 스토리지 형식으로 변환합니다. 자세한 내용은 CTAS 쿼리에 대한 고려 사항 및 제한 사항 및 열 기반 스토리지 형식 단원을 참조하세요.

2018년 9월 6일

2018-09-06 발표

ODBC 드라이버 새 버전(버전 1.0.3)을 출시했습니다. 새로운 ODBC 드라이버 버전은 기본적으로 결과를 통해 페이징하는 대신 결과를 스트리밍하며, 따라서 비즈니스 인텔리전스 도구는 대량 데이터 세트를 더 빠르게 검색할 수 있습니다. 이 버전에는 개선 사항, 버그 수정 및 "프록시 서버에 SSL 사용"에 대한 업데이트된 설명서도 포함되어 있습니다. 자세한 내용은 드라이버의 출시 정보를 참조하세요.

ODBC 드라이버 버전 1.0.3과 그 설명서를 다운로드하는 방법은 ODBC로 Amazon Athena에 연결 단원을 참조하세요.

결과 스트리밍 기능을 이용하려면 이 새 버전의 ODBC 드라이버를 사용해야 합니다. JDBC 드라이버에서도 사용할 수 있습니다. 결과 스트리밍 관련 정보를 확인하려면 ODBC 드라이버 설치 및 구성 가이드에서 UseResultsetStreaming을 검색해 보십시오.

ODBC 드라이버 버전 1.0.3은 이전 버전 드라이버를 즉시 대체할 수 있습니다. 최신 드라이버로 마이그레이션할 것을 권장합니다.

Important

ODBC 드라이버 버전 1.0.3을 사용하려면 다음 요구 사항을 따릅니다.

- 포트 444를 아웃바운드 트래픽에 개방된 상태로 둡니다.
- Athena의 정책 목록에 `athena:GetQueryResultsStream` 정책 작업을 추가합니다. 이 정책 작업은 API가 직접적으로 노출하지 않으며, 스트리밍 결과 지원 기능에 따라 ODBC 및 JDBC 드라이버와 함께 사용해야 합니다. 정책 예제는 AWS 관리형 정책: AWSQuicksightAthenaAccess을 참조하세요.

2018년 8월 23일

2018-08-23 발표

이상의 DDL 관련 기능 지원을 추가하고 다음과 같은 다양한 버그를 수정했습니다.

- Parquet의 데이터에 대한 `BINARY` 및 `DATE` 데이터 형식과 Avro의 데이터에 대한 `DATE` 및 `TIMESTAMP` 데이터 형식 지원이 추가되었습니다.
- DDL 쿼리의 `INT` 및 `DOUBLE` 지원을 추가했습니다. `INTEGER`는 `INT`의 별칭이며, `DOUBLE PRECISION`은 `DOUBLE`의 별칭입니다.
- `DROP TABLE` 및 `DROP DATABASE` 쿼리의 성능을 개선했습니다.
- 데이터 버킷에 아무것도 없을 때 Amazon S3에서 `_$folder$` 객체가 생성되지 않도록 수정했습니다.
- 파티션 값을 입력하지 않으면 `ALTER TABLE ADD PARTITION`에서 오류가 발생하는 문제를 수정했습니다.
- 문에서 정규화된 이름이 지정되면 파티션 확인 시 `DROP TABLE`이(가) 데이터베이스 이름을 무시하는 문제를 수정했습니다.

Athena에서 지원하는 데이터 형식에 대한 자세한 정보는 Amazon Athena의 데이터 형식 단원을 참조하세요.

Athena, JDBC 드라이버, Java 데이터 형식 간에 지원되는 데이터 형식 매핑에 대한 자세한 내용은 JDBC 드라이버 설치 및 구성 안내서의 “데이터 형식”을 참조하세요.

2018년 8월 16일

2018-08-16 발표

JDBC 드라이버 버전 2.0.5를 출시했습니다. 새로운 JDBC 드라이버 버전은 기본적으로 결과를 통해 페이징하는 대신 결과를 스트리밍하며, 따라서 비즈니스 인텔리전스 도구는 대량 데이터 세트를 더 빠르게 검색할 수 있습니다. JDBC 드라이버 이전 버전과 비교하면 다음과 같은 성능 개선 사항이 적용되었습니다.

- 10,000개 미만의 열을 가져올 때의 성능이 약 2배 상승했습니다.
- 10,000개 이상의 열을 가져올 때의 성능이 약 5~6배 상승했습니다.

결과 스트리밍 기능을 이용하려면 반드시 JDBC 드라이버를 사용해야 합니다. ODBC 드라이버는 사용할 수 없습니다. Athena API와 함께 사용할 수는 없습니다. 결과 스트리밍 관련 정보를 확인하려면 JDBC 드라이버 설치 및 구성 가이드에서 UseResultsetStreaming을 검색해 보십시오.

JDBC 드라이버 버전 2.0.5와 그 설명서를 다운로드하는 방법은 JDBC로 Amazon Athena에 연결 단원을 참조하세요.

JDBC 드라이버 버전 2.0.5는 이전 버전 드라이버(2.0.2)를 즉시 대체할 수 있습니다. JDBC 드라이버 버전 2.0.5 사용을 보장하려면 `athena:GetQueryResultsStream` 정책 작업을 Athena의 정책 목록에 추가해야 합니다. 이 정책 작업은 API가 직접적으로 노출하지 않으며, 스트리밍 결과 지원 기능에 따라 JDBC 드라이버와 함께 사용해야 합니다. 정책 예제는 AWS 관리형 정책: AWSQuicksightAthenaAccess을 참조하세요. 드라이버 버전 2.0.2에서 2.0.5로 마이그레이션하는 방법에 대한 자세한 정보는 JDBC 드라이버 마이그레이션 가이드를 참조하십시오.

1.x 드라이버를 2.x 드라이버로 마이그레이션하려면 기존 구성을 새 구성으로 마이그레이션해야 합니다. 드라이버 현재 버전으로 마이그레이션할 것을 적극 권장합니다. 자세한 내용은 JDBC 드라이버 마이그레이션 가이드를 참조하세요.

2018년 8월 7일

2018-08-07 발표

이제 Amazon Virtual Private Cloud 흐름 로그를 GZIP 형식으로 Amazon S3에 직접 저장하여 Athena에서 쿼리할 수도 있습니다. 자세한 정보는 Amazon VPC 흐름 로그 쿼리 및 Amazon VPC 플로우 로그는 이제 S3에 제공될 수 있습니다를 참조하세요.

2018년 6월 5일

2018-06-05 발표

주제

- 뷰 지원
- 개선 사항 및 오류 메시지 업데이트
- 버그 수정

뷰 지원

뷰 지원이 추가되었습니다. 이제 Athena에서 CREATE VIEW, DESCRIBE VIEW, DROP VIEW, SHOW CREATE VIEW, SHOW VIEWS를 사용할 수 있습니다. 뷰를 정의하는 쿼리는 쿼리에서 해당 뷰를 참조할 때마다 실행됩니다. 자세한 내용은 뷰 작업 단원을 참조하세요.

개선 사항 및 오류 메시지 업데이트

- CloudTrail SerDe 문제를 해결하고 JSON 문자열 구문 분석을 지원하기 위해 GSON 2.8.0 라이브러리가 CloudTrail SerDe에 포함되었습니다.
- 열 재정렬을 허용하여 Parquet 및 경우에 따라 ORC에 대한 Athena의 파티션 스키마 검증을 개선했습니다. 이를 통해 Athena는 시간이 지남에 따른 스키마 변화와 AWS Glue 크롤러가 추가한 테이블을 더 효과적으로 처리할 수 있습니다. 자세한 내용은 스키마 업데이트 처리 단원을 참조하십시오.
- `SHOW VIEWS`에 대한 구문 분석 지원이 추가되었습니다.
- 가장 일반적인 오류 메시지에 대해 다음 사항을 개선했습니다.
  - Athena 쿼리에서 SerDe가 열 구문 분석에 실패할 때 발생하는 내부 오류 메시지를, 오류를 설명하는 오류 메시지로 대체했습니다. 이전에는 구문 분석 오류 발생 시 Athena가 내부 오류를 표시했습니다. 새 오류 메시지는 "HIVE_BAD_DATA: Error parsing field value for field 0: java.lang.String cannot be cast to org.openx.data.jsonserde.json.JSONObject"입니다.
  - 세부 정보를 추가하여 불충분한 권한에 대한 오류 메시지를 개선했습니다.

버그 수정

다음 버그를 수정했습니다.

- `REAL`을 `FLOAT` 데이터 형식으로 내부 변환하는 문제를 수정했습니다. 이를 통해 `FLOAT` 데이터 형식을 반환하는 AWS Glue 크롤러와의 통합이 개선되었습니다.

- Athena가 AVRO DECIMAL(논리적 형식)을 DECIMAL 형식으로 변환하지 못하는 문제를 수정했습니다.
- Athena가 TIMESTAMP 데이터 형식의 값을 참조한 WHERE 절이 있는 Parquet 데이터에 대한 쿼리 결과를 반환하지 못한 문제를 수정했습니다.

2018년 5월 17일

2018-05-17 발표

Athena의 쿼리 동시 할당량이 5개에서 20개로 증가되었습니다. 즉, 한 번에 최대 DDL 쿼리 20개와 SELECT 쿼리 20개를 제출하고 실행할 수 있습니다. 동시 할당량은 DDL 및 SELECT 쿼리에 각각 따로 적용됩니다.

Athena의 동시 할당량은 서비스에 동시에 제출할 수 있는 쿼리 수로 정의됩니다. 한 번에 동일한 유형(DDL 또는 SELECT)의 쿼리를 최대 20개 제출할 수 있습니다. 동시 쿼리 할당량을 초과하는 쿼리를 제출하면 Athena API에 오류 메시지가 표시됩니다.

Athena에 쿼리를 제출하면 전체 서비스 부하와 수신 요청 양에 따라 리소스를 할당하여 쿼리를 처리합니다. 쿼리를 최대한 빨리 처리할 수 있도록 당사는 서비스를 지속적으로 모니터링하고 조정합니다.

자세한 설명은 Service Quotas을 참조하세요. 이 할당량은 조정 가능한 할당량입니다. Service Quotas 콘솔을 사용하여 동시 쿼리의 할당량 증가를 요청할 수 있습니다.

2018년 4월 19일

2018-04-19 발표

ResultSet 데이터를 배열(Array) 데이터 형식으로 반환하는 기능과 개선 사항 및 버그 수정을 포함하는 JDBC 드라이버(버전 2.0.2)의 최신 버전이 출시되었습니다. 자세한 내용은 드라이버의 출시 정보를 참조하세요.

최신 JDBC 드라이버 버전 2.0.2와 그 설명서를 다운로드하는 방법은 JDBC로 Amazon Athena에 연결 단원을 참조하세요.

JDBC 드라이버 최신 버전은 2.0.2입니다. 1.x 드라이버를 2.x 드라이버로 마이그레이션하려면 기존 구성을 새 구성으로 마이그레이션해야 합니다. 최신 드라이버로 마이그레이션할 것을 적극 권장합니다.

드라이버 최신 버전에 적용된 변경 사항, 버전 차이점, 예제를 보려면 JDBC 드라이버 마이그레이션 가이드를 참조하세요.

2018년 4월 6일

2018-04-06 발표

Athena 콘솔에서 쿼리를 입력하는 데 자동 완성을 사용합니다.

2018년 3월 15일

2018-03-15 발표

CloudTrail 콘솔에서 직접 CloudTrail 로그 파일용 Athena 테이블을 자동으로 생성할 수 있는 기능이 추가되었습니다. 자세한 설명은 CloudTrail 콘솔을 사용하여 CloudTrail 로그용 Athena 테이블 생성 을 참조하세요.

2018년 2월 2일

2018-02-12 발표

`GROUP BY` 절을 사용하는 메모리 집약적인 쿼리에 대해 디스크에 중간 데이터를 안전하게 로드하는 기능이 추가되었습니다. 그러면 이러한 쿼리의 안정성이 개선되어 "쿼리 리소스 모두 사용" 오류를 방지합니다.

2018년 1월 19일

2018-01-19 발표

Athena는 오픈 소스 분산 쿼리 엔진인 Presto를 사용하여 쿼리를 실행합니다.

Athena에는 관리할 버전이 없습니다. Athena의 기본 엔진을 Presto 버전 0.172에 기반한 버전으로 투명하게 업그레이드했습니다. 여러분은 아무 작업도 수행할 필요가 없습니다.

이 업그레이드로 이제 Athena에서 Presto 0.172 Lambda 표현식을 비롯한 Presto 0.172 함수 및 연산자를 사용할 수 있습니다.

커뮤니티가 제공한 수정을 포함해 이 릴리스의 주요 업데이트는 다음과 같습니다.

- 헤더 무시를 지원합니다. Athena가 헤더를 무시할 수 있도록 테이블을 정의할 때 `skip.header.line.count` 속성을 사용할 수 있습니다. 이는 LazySimpleSerDe 및 OpenCSV SerDe를 사용하는 쿼리에 대해 지원되며, Grok 또는 Regex SerDes의 경우 지원되지 않습니다.

- STRING 함수에 CHAR(n) 데이터 유형을 지원합니다. CHAR(n) 범위는 [1.255], VARCHAR(n) 범위는 [1,65535]입니다.
- 상관관계가 있는 하위 쿼리를 지원합니다.
- Presto Lambda 식과 함수를 지원합니다.
- DECIMAL 유형과 연산자의 성능을 개선했습니다.
- SELECT sum(col_name) FILTER인 경우 id > 0 등의 필터링된 집계를 지원합니다.
- DECIMAL, TINYINT, SMALLINT, REAL 데이터 형식의 조건자를 푸시 다운합니다.
- 정량화된 비교 조건자 ALL, ANY, SOME을 지원합니다.
- 추가된 함수: arrays_overlap(), array_except(), levenshtein_distance(), codepoint(), skewness(), kurtosis(), typeof().
- 시간대 인수를 취하는 from_unixtime() 함수의 변형을 추가했습니다.
- bitwise_and_agg() 및 bitwise_or_agg() 집계 함수를 추가했습니다.
- xxhash64() 및 to_big_endian_64() 함수를 추가했습니다.
- json_extract() 및 json_extract_scalar() 함수에 대한 JSON 경로 아래 첨자와 백슬래시를 사용하는 백슬래시 또는 큰따옴표 이스케이프가 추가되었습니다. 이전에 백슬래시는 일반 문자로 처리되었으므로, 이제 백슬래시를 사용하는 모든 호출의 의미 체계가 변경됩니다.

함수와 연산자에 대한 자세한 내용은 이 설명서의 DML 쿼리, 함수 및 연산자 단원과 Presto 설명서의 Functions and operators(함수 및 연산자)를 참조하세요.

Athena는 Presto의 일부 기능을 지원하지 않습니다. 자세한 정보는 제한을 참조하세요.

2017년 Athena 릴리스 정보

2017년 11월 13일

2017-11-13 발표

Athena를 ODBC 드라이버에 연결하는 지원이 추가되었습니다. 자세한 설명은 ODBC로 Amazon Athena에 연결을 참조하세요.

2017년 11월 1일

2017-11-01 발표

지리 공간 데이터 쿼리, 아시아 태평양(서울), 아시아 태평양(뭄바이), EU(런던) 리전에 대한 지원을 추가했습니다. 자세한 내용은 지리 공간 데이터 쿼리 및 AWS 리전 리전 및 엔드포인트를 참조하세요.

2017년 10월 19일

2017-10-19 발표

EU(프랑크푸르트)에 대한 지원을 추가했습니다. 지원되는 리전 목록은 AWS 리전 및 엔드포인트를 참조하세요.

2017년 10월 3일

2017-10-03 발표

AWS CloudFormation으로 명명된 Athena 쿼리를 생성합니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 AWS::Athena::NamedQuery를 참조하세요.

2017년 9월 25일

2017-09-25 발표

아시아 태평양(시드니)에 대한 지원이 추가되었습니다. 지원되는 리전 목록은 AWS 리전 및 엔드포인트를 참조하세요.

2017년 8월 14일

2017-08-14 발표

AWS Glue Data Catalog 통합 기능과, Athena 관리형 데이터 카탈로그에서 AWS Glue Data Catalog로 업데이트하기 위한 마이그레이션 마법사가 추가되었습니다. 자세한 내용은 AWS Glue와의 통합 단원을 참조하세요.

2017년 8월 4일

2017-08-04 발표

로그 같은 비정형 텍스트 파일의 레코드에 대해 보다 쉬운 패턴 일치를 제공하는 Grok SerDe 지원이 추가되었습니다. 자세한 내용은 Grok SerDe 단원을 참조하세요. 콘솔을 사용하여 쿼리 기록을 스크롤하는 키보드 바로 가기가 추가되었습니다(Windows 사용 CTRL + ⇧/⇩, Mac 사용 CMD + ⇧/⇩).

2017년 6월 22일

2017-06-22 발표

아시아 태평양(도쿄) 및 아시아 태평양(싱가포르) 리전에 대한 지원이 추가되었습니다. 지원되는 리전 목록은 AWS 리전 및 엔드포인트를 참조하세요.

2017년 6월 8일

2017-06-08 발표

유럽(아일랜드)에 대한 지원이 추가되었습니다. 자세한 내용은 AWS 리전 및 엔드포인트를 참조하세요.

2017년 5월 19일

2017-05-19 발표

Athena에 Amazon Athena API 및 AWS CLI 지원을 추가했습니다. JDBC 드라이버를 버전 1.1.0으로 업데이트했습니다. 다양한 문제가 해결되었습니다.

- Amazon Athena는 Athena를 위한 애플리케이션 프로그래밍을 지원합니다. 자세한 내용은 Amazon Athena API 참조를 확인하세요. 최신 AWS SDK에는 Athena API에 대한 지원이 포함되어 있습니다. 설명서 링크 및 다운로드는 Amazon Web Services용 도구의 SDK 단원을 참조하세요.
- AWS CLI에는 Athena를 위한 명령이 포함되어 있습니다. 자세한 내용은 Amazon Athena API 참조를 확인하세요.
- 새로운 Athena API는 물론 최신 기능과 버그 수정을 지원하는 새로운 JDBC 드라이버 1.1.0을 사용할 수 있습니다. https://downloads.athena.us-east-1.amazonaws.com/drivers/AthenaJDBC41-1.1.0.jar에서 드라이버를 다운로드합니다. 최신 Athena JDBC 드라이버로 업그레이드하는 것이 좋으나 이전 드라이버 버전을 계속 사용할 수 있습니다. 이전 드라이버 버전은 Athena API를 지원하지 않습니다. 자세한 내용은 JDBC로 Amazon Athena에 연결 단원을 참조하세요.
- 이전 버전의 Athena에서 정책 설명과 관련된 작업은 더 이상 사용되지 않습니다. JDBC 드라이버 버전 1.1.0으로 업그레이드하고 고객 관리형 또는 인라인 IAM 정책이 JDBC 사용자에게 연결되어 있는 경우 IAM 정책을 업데이트해야 합니다. 반면, 이전 버전의 JDBC 드라이버는 Athena API를 지원하지 않으므로 이전 버전 JDBC 사용자에게 연결된 정책에서는 더 이상 사용되지 않는 작업만 지정할 수 있습니다. 따라서 고객 관리형 또는 인라인 IAM 정책은 업데이트할 필요가 없습니다.
- 이러한 정책 관련 작업은 Athena API가 릴리스되기 전에 Athena에서 사용되었습니다. 이처럼 정책에서 더 이상 사용되지 않는 작업에는 반드시 1.1.0 이전 버전의 JDBC 드라이버를 사용합니다.

JDBC 드라이버를 업그레이드하는 경우 더 이상 사용되지 않는 작업을 허용하거나 거부하는 정책 설명을 적절한 API 작업으로 바꿉니다. 그렇지 않으면 오류가 발생합니다.

더 이상 사용되지 않는 정책 관련 작업	해당하는 Athena API 작업
`athena:RunQuery`	`athena:StartQueryExecution`
`athena:CancelQueryExecution`	`athena:StopQueryExecution`
`athena:GetQueryExecutions`	`athena:ListQueryExecutions`

개선 사항

- 쿼리 문자열 제한을 256KB로 늘렸습니다.

버그 수정

- 콘솔에서 결과를 스크롤할 때 쿼리 결과가 잘못된 형식으로 표시되는 문제가 수정되었습니다.
- Amazon S3 데이터 파일의 `\u0000` 문자열이 오류를 일으키는 문제가 해결되었습니다.
- JDBC 드라이버를 통해 이루어진 쿼리를 취소하라는 요청이 실패하는 문제가 해결되었습니다.
- AWS CloudTrail SerDe가 미국 동부(오하이오)의 Amazon S3 데이터 처리에 실패하는 문제가 해결되었습니다.
- 분할된 테이블에서 `DROP TABLE`이 실패하는 문제가 해결되었습니다.

2017년 4월 4일

2017-04-04 발표

Amazon S3 데이터 암호화에 대한 지원이 추가되었으며 암호화 지원, 개선 및 버그 수정을 포함한 JDBC 드라이버 업데이트(버전 1.0.1)가 출시되었습니다.

특성

- 다음과 같은 암호화 기능이 추가되었습니다.

  - Amazon S3에서 암호화된 데이터 쿼리를 지원합니다.
  - Athena 쿼리 결과의 암호화를 지원합니다.
- 새 버전의 드라이버는 새로운 암호화 기능을 지원하고 개선 사항을 추가하며 문제를 수정합니다.
- `ALTER TABLE`을 사용하여 열을 추가, 교체 및 변경할 수 있는 기능이 추가되었습니다. 자세한 정보는 Hive 설명서의 Alter Column을 참조하세요.
- LZO 압축 데이터 쿼리에 대한 지원이 추가되었습니다.

자세한 내용은 저장 중 암호화 단원을 참조하십시오.

개선 사항

- 페이지 크기가 개선되어 JDBC 쿼리 성능이 향상되고 100개가 아닌 1,000개의 행이 반환됩니다.
- JDBC 드라이버 인터페이스를 사용하여 쿼리를 취소하는 기능이 추가되었습니다.
- JDBC 연결 URL에 JDBC 옵션을 지정하는 기능이 추가되었습니다. 최신 JDBC 드라이버에 대해 JDBC로 Amazon Athena에 연결 내용을 참조하세요.
- 드라이버에 PROXY 설정이 추가되었으며 이제 AWS SDK for Java에서 ClientConfiguration을 사용하여 이 설정을 수행할 수 있습니다.

버그 수정

다음 버그를 수정했습니다.

- JDBC 드라이버 인터페이스를 사용하여 여러 쿼리가 실행된 경우 병목 오류가 발생합니다.
- 10진수 데이터 형식을 프로젝션하면 JDBC 드라이버가 중지됩니다.
- 테이블에 정의된 데이터 형식과 상관없이 JDBC 드라이버가 모든 데이터 형식을 문자열로 반환합니다. 예를 들어 `INT` 데이터 형식으로 정의된 열을 선택하고 `resultSet.GetObject()`를 사용하면 `INT` 대신 `STRING` 데이터 형식이 반환됩니다.
- JDBC 드라이버는 쿼리가 실행되는 시점이 아니라 연결이 이루어진 시점에 자격 증명을 확인합니다.
- URL을 따라 스키마가 지정된 경우 JDBC 드라이버를 통해 실행된 쿼리는 실패합니다.

2017년 3월 24일

2017-03-24 발표

AWS CloudTrail SerDe를 추가하고 성능을 개선했으며 파티션 문제를 수정했습니다.

특성

- AWS CloudTrail SerDe는 CloudTrail 로그 읽기용 Hive JSON SerDe로 대체되었습니다. CloudTrail 로그 쿼리에 대한 자세한 내용은 AWS CloudTrail 로그 쿼리 단원을 참조하세요.

개선 사항

- 많은 수의 파티션을 검사할 때 성능이 향상되었습니다.
- `MSCK Repair Table` 작업의 성능이 개선되었습니다.
- 기본 리전 이외의 리전에 저장된 Amazon S3 데이터를 쿼리하는 기능이 추가되었습니다. 표준 Athena 요금 외에 Amazon S3의 표준 리전 간 데이터 전송 요금이 적용됩니다.

버그 수정

- 파티션이 로드되지 않을 때 "테이블을 찾을 수 없음 오류"가 발생하는 버그가 수정되었습니다.
- `ALTER TABLE ADD PARTITION IF NOT EXISTS` 쿼리에 예외가 발생하는 버그를 수정했습니다.
- `DROP PARTITIONS`의 버그를 수정했습니다.

2017년 20월 2일

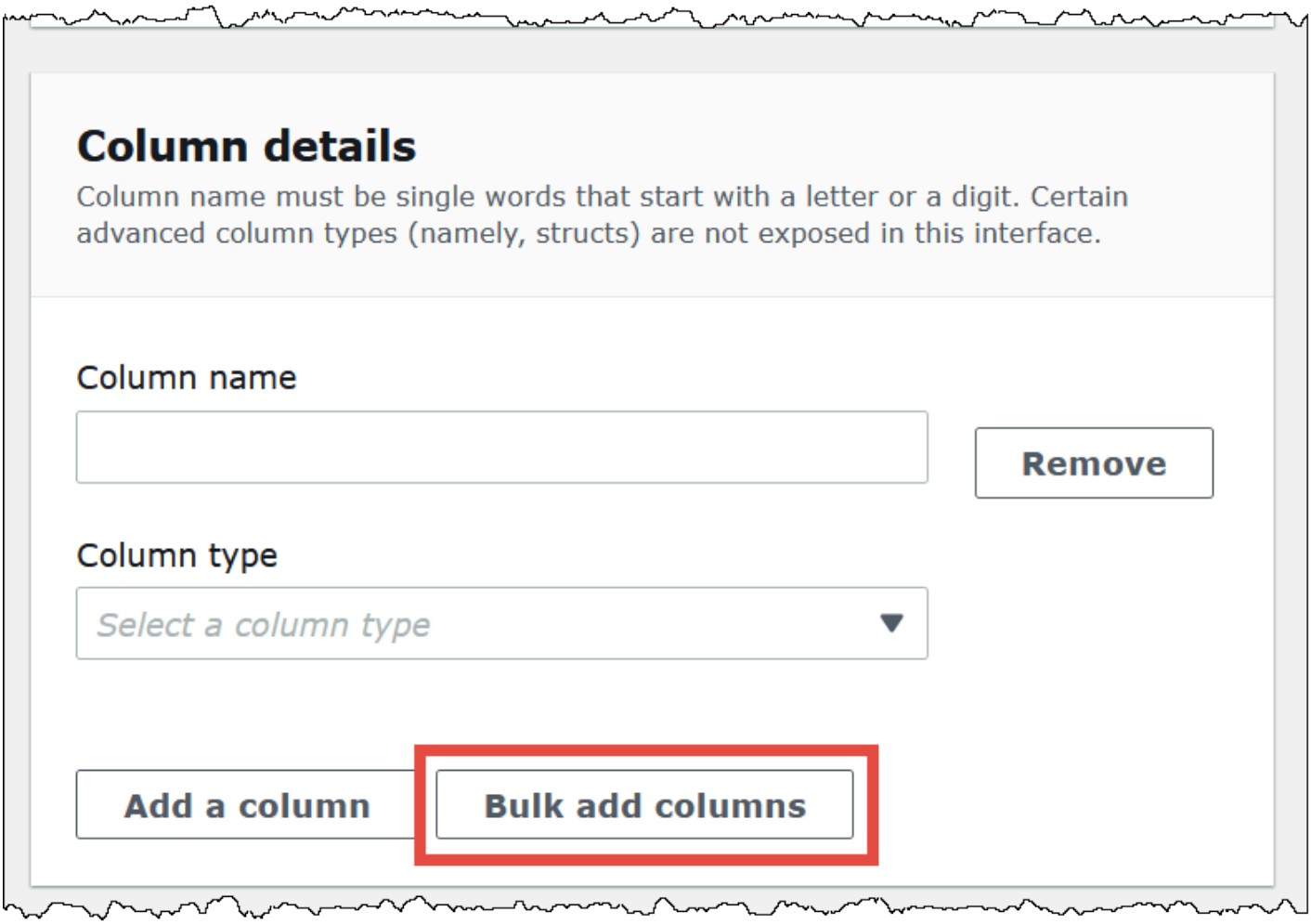
2017-02-20 발표

AvroSerDe 및 OpenCSVSerDe, 미국 동부(오하이오) 리전, 콘솔 마법사에서 열을 일괄 편집하는 기능에 대한 지원이 추가되었습니다. 대용량 Parquet 테이블의 성능이 개선되었습니다.

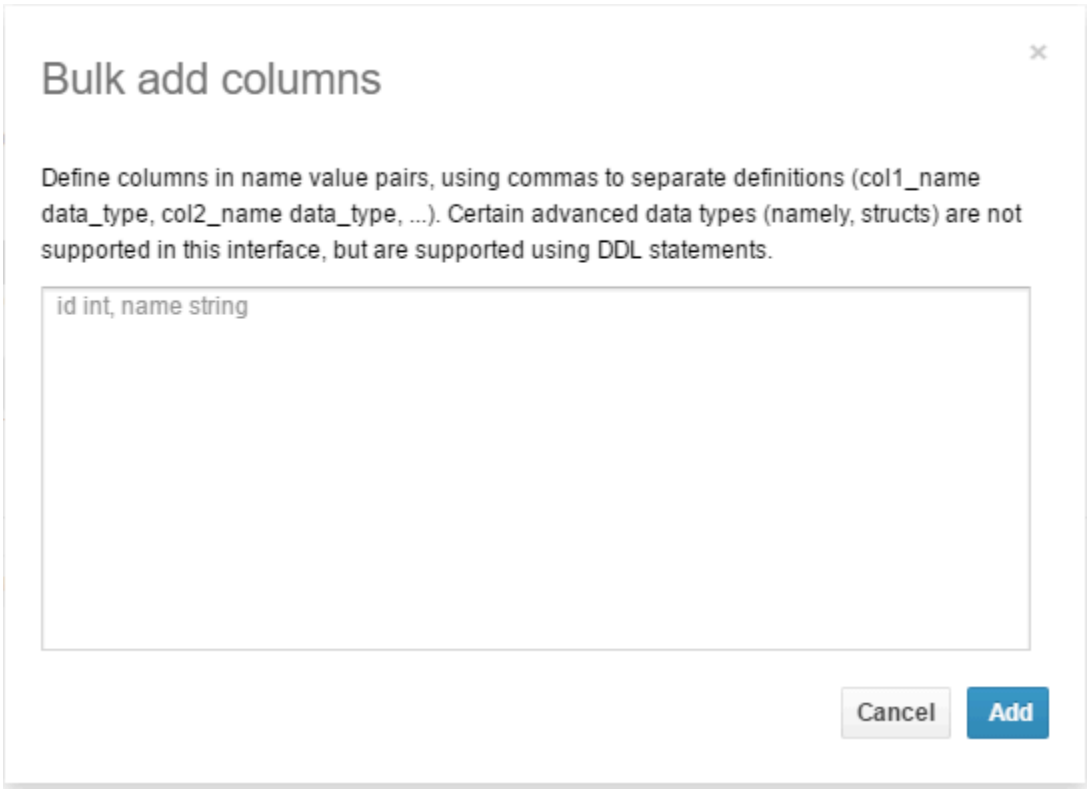
특성

- 새로운 SerDes에 추가된 지원:
  - Avro SerDe
  - CSV 처리를 위한 OpenCSVSerDe
- 미국 동부(오하이오) 리전(us-east-2)이 출시되었습니다. 이제 이 리전에서 쿼리를 실행할 수 있습니다.
- 이제 S3 버킷 데이터에서 테이블 생성(Create Table From S3 bucket data) 양식을 사용하여 테이블 스키마를 대량으로 정의할 수 있습니다. 쿼리 편집기에서 생성(Create), S3 버킷 데이터(S3 bucket

data)를 선택한 다음, 열 세부 정보(Column details) 섹션에서 열 일괄 추가(Bulk add columns)를 선택합니다.



텍스트 상자에 이름-값 페어를 입력하고 [Add]를 선택합니다.



개선 사항

- 대용량 Parquet 테이블의 성능이 개선되었습니다.

문서 기록

최종 설명서 업데이트: 2024년 5월 28일.

사용자의 의견을 수렴하기 위해 설명서를 자주 업데이트합니다. 다음 표에서는 Amazon Athena 설명서에 대한 중요 추가 사항을 설명합니다. 모든 업데이트가 표시되지는 않습니다.

변경 사항	설명	릴리스 날짜
`AmazonAthenaFullAccess` 관리형 정책이 업데이트되었습니다.	AmazonAthenaFullAccess 관리형 정책에 `datazone:ListDomains`, `datazone:ListProjects` 및 `datazone:ListAccountEnvironments` 권한이 추가되었습니다. 추가된 작업을 통해 Athena 사용자는 Amazon DataZone 도메인, 프로젝트 및 환경에서 작업할 수 있습니다. 자세한 내용은 Athena에서 Amazon DataZone 사용 단원을 참조하십시오.	2024년 1월 3일
`AmazonAthenaFullAccess` 관리형 정책이 업데이트되었습니다.	AmazonAthenaFullAccess 관리형 정책에 `glue:StartColumnStatisticsTaskRun`, `glue:GetColumnStatisticsTaskRun` 및 `glue:GetColumnStatisticsTaskRuns` 권한이 추가되었습니다. 추가된 작업을 통해 Athena는 AWS Glue를 직접 호출하여 비용 기반 옵티마이저 기능에 대한 통계를 검색할 수 있습니다. 자세한 내용은 비용 기반 최적화 프로그램 사용 단원을 참조하십시오.	2024년 1월 3일
IAM Identity Center 지원 Athena 작업 그룹에 대한 설명서가 추가되었습니다.	IAM Identity Center 인증 모드를 사용하는 Athena SQL 작업 그룹을 생성할 수 있습니다. 이러한 작업 그룹은 Amazon Athena, Amazon EMR Studio 등의 AWS 서비스 전반에서 동일한 ID 사용을 지원합니다. 자세한 내용은 IAM Identity Center 지원 Athena 작업 그룹 사용 단원을 참조하십시오.	2023년 12월 5일
S3 Express One Zone 데이터 쿼리에 대한 설명서가 추가되었습니다.	Athena를 사용하여 Amazon S3 Express One Zone 스토리지 클래스의 데이터를 쿼리할 수 있습니다. 자세한 내용은 S3 Express One Zone 데이터 쿼리 단원을 참조하십시오.	2023년 11월 28일

변경 사항	설명	릴리스 날짜
Glue Data Catalog 뷰에 대한 설명서가 추가되었습니다.	Glue Data Catalog 뷰를 사용하여 Amazon Athena, Amazon Redshift 등의 AWS 서비스 전반에서 공통된 단일 뷰를 제공할 수 있습니다. 자세한 내용은 AWS Glue Data Catalog 뷰 사용 단원을 참조하십시오.	2023년 11월 27일
비용 기반 최적화 프로그램 기능에 대한 설명서가 추가되었습니다.	AWS Glue의 통계를 사용하여 Athena SQL에서 쿼리를 최적화할 수 있습니다. 자세한 내용은 비용 기반 최적화 프로그램 사용 단원을 참조하십시오.	2023년 11월 17일
Athena JDBC 3.x 드라이버에 대한 설명서가 추가되었습니다.	Athena JDBC 3.x 드라이버를 사용하여 Amazon S3에서 직접 쿼리 결과를 읽을 수 있습니다. JDBC 3.x 드라이버는 JDBC 2.x 드라이버가 지원하는 거의 모든 인증 방법을 지원합니다. 자세한 내용은 Athena JDBC 3.x 드라이버 단원을 참조하십시오.	2023년 11월 16일
Athena에서 DataZone 사용에 대한 설명서가 추가되었습니다.	DataZone을 사용하여 Athena, AWS Glue, Lake Formation 등의 AWS 분석 서비스 전반에서 경험을 단순화할 수 있습니다. 자세한 내용은 Athena에서 Amazon DataZone 사용 단원을 참조하십시오.	2023년 10월 4일
용량 예약에 대한 설명서가 추가되었습니다.	이제 Amazon Athena에서 용량 예약을 사용하여 완전관리형 컴퓨팅 용량에서 SQL 쿼리를 실행할 수 있습니다. 자세한 내용은 쿼리 처리 용량 관리 단원을 참조하십시오.	2023년 4월 28일
페더레이션된 보기 쿼리에 관한 설명서가 추가되었습니다.	이제 Athena를 사용하여 페더레이션된 데이터 소스에서 보기를 생성하고 쿼리할 수 있습니다. 자세한 내용은 페더레이션된 보기 쿼리 단원을 참조하십시오.	2023년 4월 4일
Amazon S3에서 제한 방지에 관한 설명서가 추가되었습니다.	자세한 내용은 Amazon S3 제한 방지 단원을 참조하십시오.	2023년 3월 24일

변경 사항	설명	릴리스 날짜
`AmazonAthenaFullAccess` 관리형 정책이 업데이트되었습니다.	AmazonAthenaFullAccess 관리형 정책에 `pricing:GetProducts` 가 추가되었습니다. 추가된 작업은 AWS Billing and Cost Management에 대한 액세스를 제공합니다. 자세한 내용은 AWS Billing and Cost Management용 API 참조의 GetInstance를 참조하세요.	2023년 1월 25일
Athena 압축 지원에 대한 설명서가 확장되었습니다.	Hive 테이블 압축, Iceberg 테이블 최적화 및 ZSTD 압축 수준에 대한 개별 주제가 추가되었습니다. 자세한 내용은 Athena 압축 지원 단원을 참조하십시오.	2023년 1월 20일
Apache Spark용 Amazon Athena에 대한 설명서가 추가되었습니다.	이제 Amazon Athena에서 Apache Spark 애플리케이션 및 Jupyter 호환 노트북을 대화식으로 생성하고 실행할 수 있습니다. 자세한 내용은 Amazon Athena에서 Apache Spark 사용 단원을 참조하십시오.	2022년 11월 30일
Athena IBM Db2 커넥터에 대한 설명서가 추가되었습니다.	IBM Db2용 Amazon Athena 커넥터를 사용하여 Athena에서 Db2를 쿼리할 수 있습니다. 자세한 내용은 Amazon Athena IBM Db2 커넥터 단원을 참조하세요.	2022년 11월 18일
쿼리 결과 재사용에 대한 설명서가 추가되었습니다.	Athena에서 쿼리를 다시 실행할 때 이제 마지막 저장된 쿼리 결과를 재사용하도록 선택할 수 있습니다. 그러면 스캔되는 바이트 수의 측면에서 성능이 향상되고 비용이 절감됩니다. 자세한 내용은 쿼리 결과 재사용 단원을 참조하십시오.	2022년 11월 8일
CloudTrail 로그에 대한 설명서가 업데이트되었습니다.	CloudTrail 로그 쿼리를 위한 `CREATE TABLE` DDL은 CloudTrail SerDe 대신 JSON SerDe를 사용하도록 업데이트되었습니다. 자세한 내용은 AWS CloudTrail 로그 쿼리 단원을 참조하십시오.	2022년 11월 3일
Athena 엔진 버전 3에 대한 설명서를 추가했습니다.	Athena 엔진 버전 3에 대한 자세한 내용은 Athena 엔진 버전 3 섹션을 참조하세요.	2022년 10월 13일

변경 사항	설명	릴리스 날짜
Okta 플러그인을 사용하여 ODBC용 SSO를 구성하는 방법에 대한 자습서를 추가했습니다.	Okta 자격 증명 공급자를 사용하여 Single Sign-On(SSO) 기능에 대한 Okta 플러그인과 Amazon Athena ODBC 드라이버를 구성합니다. 자세한 내용은 Okta 플러그인 및 Okta ID 공급자를 사용하여 ODBC에 대한 SSO 구성 단원을 참조하십시오.	2022년 8월 23일
Athena 콘솔에서 쿼리 계획 및 통계를 보는 방법에 대한 설명서가 추가되었습니다.	Athena 쿼리 편집기를 사용하여 쿼리가 실행될 방식의 그래픽 표현과 완료된 쿼리가 실행된 방식에 대한 그래프, 세부 정보, 통계를 볼 수 있습니다. 자세한 내용을 알아보려면 SQL 쿼리에 대한 실행 계획 보기 및 완료된 쿼리에 대한 통계 및 실행 세부 정보 보기 섹션을 참조하세요.	2022년 7월 21일
외부 Hive 메타스토어의 Apache Hive 뷰를 쿼리하기 위한 설명서가 추가되었습니다.	Athena를 사용하여 외부 Hive 메타스토어에 생성된 Apache 뷰를 쿼리할 수 있습니다. 일부 Hive 함수는 지원되지 않거나 특별한 처리가 필요합니다. 자세한 내용은 Hive 뷰 작업 단원을 참조하십시오.	2022년 4월 22일
저장된 쿼리에 대한 문서가 추가되었습니다.	Athena의 저장된 쿼리 기능을 사용하여 쿼리를 저장하고, 호출하고, 편집하고, 이름을 바꿀 수 있습니다. 자세한 내용은 이 가이드의 저장된 쿼리 사용 섹션 및 Amazon Athena API 참조의 UpdateNamedQuery를 참조하세요.	2022년 2월 28일
Apache Iceberg 지원에 대한 미리 보기 문서가 추가되었습니다.	Athena는 데이터에 대한 Apache Parquet 형식 및 메타스토어의 AWS Glue 카탈로그를 사용하는 Apache Iceberg 테이블에 대한 읽기, 시간 이동 및 쓰기 쿼리를 지원합니다. 자세한 내용은 Apache Iceberg 테이블 사용 단원을 참조하십시오.	2021년 11월 26일
교차 계정 연합 쿼리에 대한 설명서가 추가되었습니다.	교차 계정 연합 쿼리 기능을 사용하여 다른 계정의 데이터 소스를 쿼리할 수 있습니다. 이 기능을 사용하기 위한 권한 설정에 대한 자세한 내용은 계정 간 연합 쿼리 사용 설정 단원을 참조하세요.	2021년 11월 12일

변경 사항	설명	릴리스 날짜
Athena UNLOAD 문에 대한 설명서가 추가되었습니다.	UNLOAD 문을 사용하면 SELECT 문의 쿼리 결과를 Apache Parquet, ORC, Apache Avro, JSON 형식에 쓸 수 있습니다. 자세한 내용은 UNLOAD 단원을 참조하세요.	2021년 8월 5일
Athena EXPLAIN 문의 기능에 대한 설명서가 추가되었습니다.	자세한 내용은 Athena에서 EXPLAIN 및 EXPLAIN ANALYZE 사용 및 Athena EXPLAIN 문 결과의 이해 단원을 참조하세요.	2021년 4월 5일
Athena의 문제 해결 및 성능 조정에 대한 페이지가 추가되었습니다.	자세한 내용은 Athena의 문제 해결 및 Athena의 성능 튜닝 단원을 참조하세요.	2020년 12월 30일
Athena 엔진 버전 관리 및 Athena 엔진 버전 2에 대한 설명서가 추가되었습니다.	자세한 내용은 Athena 엔진 버전 관리 단원을 참조하세요.	2020년 11월 11일
정식 출시 릴리스에 대한 연합 쿼리 설명서가 업데이트되었습니다.	자세한 내용은 Amazon Athena 연합 쿼리 사용 및 Athena와 CalledVia 컨텍스트 키 사용 단원을 참조하세요.	2020년 11월 11일
Athena에 대한 연동 액세스를 위해 Lake Formation과 함께 JDBC 드라이버를 사용하는 방법에 대한 설명서가 추가되었습니다.	자세한 내용은 Athena에 대한 연합 액세스를 위해 Lake Formation과 Athena JDBC 및 ODBC 드라이버 사용 및 자습서: Lake Formation 및 JDBC를 사용하여 Athena에 대한 Okta 사용자의 연합 액세스 구성 단원을 참조하세요.	2020년 9월 25일

변경 사항	설명	릴리스 날짜
Amazon Athena OpenSearch 데이터 커넥터에 대한 설명서가 추가되었습니다.	자세한 내용은 Amazon Athena OpenSearch 커넥터 단원을 참조하십시오.	2020년 7월 21일
Hudi 데이터 세트 쿼리에 대한 설명서가 추가되었습니다.	자세한 내용은 Athena를 사용하여 Apache Hudi 데이터 집합 쿼리 단원을 참조하세요.	2020년 7월 9일
Amazon S3 저장된 Apache 웹 서버 로그 및 IIS 웹 서버 로그를 쿼리하는 방법에 대한 설명서가 추가되었습니다.	자세한 내용은 Amazon S3에 저장된 Apache 로그 쿼리 및 Amazon S3에 저장된 Internet Information Server(IIS) 로그 쿼리 단원을 참조하세요.	2020년 7월 8일
Athena Data Connector for External Hive Metastore의 정식 릴리스에 대한 설명서가 추가되었습니다.	자세한 내용은 외부 Hive 메타스토어용 Athena 데이터 커넥터 사용 단원을 참조하세요.	2020년 6월 1일
데이터 카탈로그 리소스에 태그 지정에 대한 설명서를 추가했습니다.	자세한 내용은 Athena 리소스 태깅 단원을 참조하세요.	2020년 6월 1일
파티션 프로젝션에 대한 설명서를 추가했습니다.	자세한 내용은 Amazon Athena를 사용한 파티션 프로젝션 단원을 참조하세요.	2020년 5월 21일

변경 사항	설명	릴리스 날짜
Athena에 대한 Java 코드 예제가 업데이트되었습니다.	자세한 내용은 코드 샘플 단원을 참조하세요.	2020년 5월 11일
Amazon GuardDuty 결과 쿼리에 대한 주제가 추가되었습니다.	자세한 내용은 Amazon GuardDuty 결과 쿼리 단원을 참조하세요.	2020년 3월 19일
Athena 쿼리 상태 전환을 모니터링하기 위한 CloudWatch Events 사용에 대한 주제가 추가되었습니다.	자세한 내용은 Amazon EventBridge 이벤트를 사용하여 Athena 이벤트 모니터링 단원을 참조하십시오.	2020년 3월 11일
Athena에서 AWS Global Accelerator 흐름 로그 쿼리에 대한 주제가 추가되었습니다.	자세한 내용은 AWS Global Accelerator 흐름 로그 쿼리 단원을 참조하세요.	2020년 2월 6일

변경 사항	설명	릴리스 날짜
• INSERT INTO와 함께 CTAS를 사용하여 분할되지 않은 소스의 데이터를 분할된 대상에 추가하는 방법에 대한 설명서가 추가되었습니다. • Athena용 ODBC 드라이버 1.1.0 미리 보기 버전에 대한 다운로드 링크가 추가되었습니다. • `SHOW DATABASES LIKE` 정규식에 대한 설명이 수정되었습니다. • CTA 주제에서 `partitioned_by` 구문이 수정되었습니다. • 기타 사소한 사항이 수정되었습니다.	설명서 업데이트에는 다음 주제가 포함되지만 이에 제한되지는 않습니다. • ETL 및 데이터 분석에 CTAS 및 INSERT INTO 사용 • ODBC로 Amazon Athena에 연결(이제 1.1.0 미리 보기 기능이 1.1.2 ODBC 드라이버에 포함됩니다.) • SHOW DATABASES • CREATE TABLE AS	2020년 2월 4일
INSERT INTO와 함께 CTAS를 사용하여 분할된 소스의 데이터를 분할된 대상에 추가하는 방법에 대한 설명서가 추가되었습니다.	자세한 내용은 CTAS 및 INSERT INTO를 사용하여 100개 파티션 한도 해결 단원을 참조하세요.	2020년 1월 22일

변경 사항	설명	릴리스 날짜
쿼리 결과 위치 정보가 업데이트되었습니다.	Athena에서 더 이상 'default(기본)' 쿼리 결과 위치를 생성하지 않습니다. 자세한 내용은 쿼리 결과 위치 지정 단원을 참조하세요.	2020년 1월 20일
AWS Glue Data Catalog 쿼리에 대한 주제가 추가되었습니다. Athena에서 서비스 할당량(이전의 “서비스 제한”)에 대한 정보가 업데이트되었습니다.	자세한 정보는 다음 주제를 참조하세요. • AWS Glue Data Catalog 쿼리 • Service Quotas	2020년 1월 17일
`TIMESTAMP` 유형이 UNIX 숫자 형식으로 지정되어야 한다는 점을 설명하기 위해 OpenCSVSerDe에 대한 주제가 수정되었습니다.	자세한 내용은 CSV 처리를 위한 OpenCSVSerDe 단원을 참조하세요.	2020년 1월 15일
Athena가 비대칭 키를 지원하지 않는다는 점을 설명하기 위해 암호화에 대한 보안 주제가 업데이트되었습니다.	Athena는 데이터 읽기 및 쓰기를 위한 대칭 키만 지원합니다. 자세한 내용은 지원되는 Amazon S3 암호화 옵션 단원을 참조하세요.	2020년 1월 8일
사용자 지정 AWS KMS 키로 암호화된 Amazon S3 버킷에 대한 교차 계정 액세스 정보가 추가되었습니다.	자세한 내용은 사용자 지정 AWS KMS 키로 암호화된 버킷에 대한 계정 간 액세스 단원을 참조하십시오.	2019년 12월 13일

변경 사항	설명	릴리스 날짜
연합 쿼리, 외부 Hive 메타스토어, 기계 학습 및 사용자 지정 함수에 대한 설명서가 추가되었습니다. 새로운 CloudWatch 지표가 추가되었습니다.	자세한 정보는 다음 주제를 참조하세요. • Amazon Athena 연합 쿼리 사용 • 사용 가능한 데이터 소스 커넥터 • 외부 Hive 메타스토어용 Athena 데이터 커넥터 사용 • Machine Learning(ML) with Amazon Athena 사용 • 사용자 정의 함수를 사용한 쿼리 • Athena의 CloudWatch 지표 및 차원 목록	2019년 11월 26일
새 INSERT INTO 명령 단원이 추가되었고 데이터 매니페스트 파일 지원용 쿼리 결과 위치 정보가 업데이트되었습니다.	자세한 내용은 INSERT INTO 및 쿼리 결과, 최근 쿼리, 출력 파일 작업 단원을 참조하세요.	2019년 9월 18일
인터페이스 VPC 종단점(PrivateLink) 지원 단원이 추가되었습니다. JDBC 드라이버가 업데이트되었습니다. 보강된 VPC 흐름 로그 정보가 업데이트되었습니다.	자세한 내용은 인터페이스 VPC 엔드포인트를 사용하여 Amazon Athena에 연결, Amazon VPC 흐름 로그 쿼리, JDBC로 Amazon Athena에 연결 단원을 참조하세요.	2019년 9월 11일
AWS Lake Formation와(과) 통합 단원이 추가되었습니다.	자세한 내용은 Athena를 사용하여 AWS Lake Formation에 등록된 데이터 쿼리 단원을 참조하세요.	2019년 6월 26일

변경 사항	설명	릴리스 날짜
기타 AWS 서비스와의 일관성에 대한 보안 단원이 업데이트되었습니다.	자세한 내용은 Amazon Athena 보안 단원을 참조하세요.	2019년 6월 26일
AWS WAF 로그 쿼리 작업 단원이 추가되었습니다.	자세한 내용은 AWS WAF 로그 쿼리 단원을 참조하세요.	2019년 5월 31일
Athena 작업 그룹을 지원하는 ODBC 드라이버 새 버전을 출시했습니다.	ODBC 드라이버 버전 1.0.5와 해당 설명서를 다운로드하려면 ODBC로 Amazon Athena에 연결 단원을 참조하세요. Workgroup에 태그를 사용할 경우 ODBC 드라이버 연결 문자열에 대해 변경된 사항은 없습니다. 태그를 사용하려면 ODBC 드라이버 최신 버전(현재 이 버전)으로 업그레이드하세요. 이 드라이버 버전을 사용하면 Athena API Workgroup 작업을 사용하여 Workgroup을 생성 및 관리할 수 있고, Athena API 태그 작업을 사용하여 Workgroup에서 태그를 추가하거나 태그 목록을 조회하거나 태그를 제거할 수 있습니다. 시작하려면 먼저 IAM에서 작업 그룹 작업과 태그 작업에 대해 리소스 수준 권한을 가지고 있는지 확인합니다.	2019년 3월 5일
Amazon Athena에서 작업 그룹에 대해 태그 지원이 추가되었습니다.	각 태그는 사용자가 정의하는 키와 값으로 구성됩니다. Workgroup에 태그를 지정할 때 사용자 지정 메타데이터를 할당하게 됩니다. 예를 들어, 비용 센터별로 Workgroup을 생성할 수 있습니다. 그런 다음 이러한 Workgroup에 태그를 추가하여 각 비용 센터에 대해 Athena 비용을 추적할 수 있습니다. 자세한 내용은 AWS Billing and Cost Management 사용 설명서의 결제 목적으로 태그 사용을 참조하세요.	2019년 2월 22일

변경 사항	설명	릴리스 날짜
Athena에 사용되는 JSON OpenX SerDe가 개선되었습니다.	이러한 개선에는 다음이 포함되며 그 밖에도 다양한 개선이 이루어졌습니다. • `ConvertDotsInJsonKeysToUnderscores` 속성을 지원합니다. `TRUE`로 설정하면 SerDe가 키 이름의 점을 밑줄로 바꿀 수 있습니다. 예를 들어 JSON 데이터 세트에 이름이 `"a.b"`인 키가 있을 경우 이 속성을 사용하여 Athena에서 열 이름이 `"a_b"`가 되도록 정의할 수 있습니다. 기본값은 `FALSE`입니다. 기본적으로 Athena는 열 이름에 점을 허용하지 않습니다. • `case.insensitive` 속성을 지원합니다. 기본적으로 Athena에서는 JSON 데이터 세트의 모든 키가 소문자를 사용해야 합니다. `WITH SERDE PROPERTIES ("case.insensitive"= FALSE;)` 를 사용하면 데이터에서 대소문자를 구분하는 키 이름을 사용할 수 있습니다. 기본값은 `TRUE`입니다. `TRUE`로 설정하면 SerDe가 모든 대문자 열을 소문자로 변환합니다. 자세한 내용은 OpenX JSON SerDe 단원을 참조하세요.	2019년 2월 18일
Workgroup에 대한 지원이 추가되었습니다.	Workgroup을 사용하여 사용자, 팀, 애플리케이션 또는 워크로드를 구분할 수 있고, 각 쿼리 또는 전체 Workgroup에서 처리할 수 있는 데이터 양의 한도를 설정할 수 있습니다. Workgroup은 IAM 리소스 역할을 하기 때문에 리소스 수준 권한을 사용하여 특정 Workgroup에 대한 액세스를 제어할 수 있습니다. 또한 Amazon CloudWatch에서 쿼리 관련 지표를 볼 수 있고, 스캔된 데이터의 양에 대한 한도를 구성하여 쿼리 비용을 제어할 수 있으며, 임계값을 생성하여 이러한 임계값이 위반될 경우 Amazon SNS 경보와 같은 조치를 실행할 수 있습니다. 자세한 내용은 쿼리 실행용 작업 그룹 사용 및 CloudWatch 지표 및 이벤트를 사용하여 비용 관리 및 쿼리 모니터링 단원을 참조하세요.	2019년 2월 18일

변경 사항	설명	릴리스 날짜
Network Load Balancer의 로그 분석 지원이 추가되었습니다.	Network Load Balancer의 로그 분석에 대한 Athena 쿼리 예제를 추가했습니다. 이러한 로그는 Network Load Balancer로 전송된 TLS(전송 계층 보안) 요청에 대한 세부 정보를 수신합니다. 이러한 액세스 로그를 사용하여 트래픽 패턴을 분석하고 문제를 해결할 수 있습니다. 자세한 설명은 the section called “Network Load Balancer”을 참조하세요.	2019년 1월 24일
AD FS와 SAML 2.0(Security Assertion Markup Language 2.0)을 사용하여 Athena API에 대한 연동 액세스를 지원하는 새로운 버전의 JDBC 및 ODBC 드라이버를 릴리스했습니다.	드라이버의 이 릴리스에서는 Athena에 대한 연동 액세스가 Active Directory Federation Service(AD FS 3.0)에 지원됩니다. 액세스는 SAML 2.0을 지원하는 JDBC 또는 ODBC 드라이버 버전을 통해 설정됩니다. Athena API에 대한 연동 액세스를 구성하는 방법에 대한 자세한 정보는 the section called “Athena API에 대한 연합 액세스 활성화” 단원을 참조하세요.	2018년 11월 10일
Athena에서 데이터베이스 및 테이블에 대해 세분화된 액세스 제어에 대한 지원이 추가되었습니다. 또한 Data Catalog의 데이터베이스와 테이블 메타데이터를 암호화할 수 있는 정책이 Athena에서 추가되었습니다.	Athena에서 사용하는 데이터베이스와 테이블 같은, AWS Glue Data Catalog의 리소스에 대한 세분화된 액세스를 제공하는 자격 증명 기반(IAM) 정책을 생성하는 기능이 추가되었습니다. 또한 Athena에 특정 정책을 추가해 Data Catalog의 데이터베이스와 테이블 메타데이터를 암호화할 수 있습니다. 자세한 내용은 AWS Glue Data Catalog의 데이터베이스와 테이블에 대한 세분화된 액세스 단원을 참조하세요.	2018년 10월 15일

변경 사항	설명	릴리스 날짜
CREATE TABLE AS SELECT 문에 대한 지원이 추가되었습니다. 설명서의 다른 개선 사항 적용	CREATE TABLE AS SELECT 문에 대한 지원이 추가되었습니다. 쿼리 결과에서 테이블 생성(CTAS), CTAS 쿼리에 대한 고려 사항 및 제한 사항 및 CTAS 쿼리 예제 단원을 참조하세요.	2018년 10월 10일
결과를 스트리밍하는 대신 페이지로 가져오는 기능을 기본으로 지원하는 ODBC 드라이버 버전 1.0.3이 출시되었습니다. 설명서의 다른 개선 사항 적용	ODBC 드라이버 버전 1.0.3은 스트리밍 결과를 지원하고 이 버전에는 개선 사항, 버그 수정 및 "프록시 서버에 SSL 사용"에 대한 업데이트된 설명서도 포함되어 있습니다. ODBC 드라이버 버전 1.0.3과 그 설명서를 다운로드하는 방법은 ODBC로 Amazon Athena에 연결 단원을 참조하세요.	2018년 9월 6일
결과를 스트리밍하는 대신 페이지로 가져오는 기능을 기본으로 지원하는 JDBC 드라이버 버전 2.0.5가 출시되었습니다. 설명서의 다른 개선 사항 적용	결과를 스트리밍하는 대신 페이지로 가져오는 기능을 기본으로 지원하는 JDBC 드라이버 2.0.5가 출시되었습니다. 자세한 설명은 JDBC로 Amazon Athena에 연결을 참조하세요.	2018년 8월 16일

변경 사항	설명	릴리스 날짜
Amazon S3에 GZIP 형식으로 바로 저장할 수 있는 Amazon Virtual Private Cloud 흐름 로그 쿼리 방법을 설명서에 추가했습니다. 업데이트된 ALB 로그 쿼리 예시	Amazon S3에 GZIP 형식으로 바로 저장할 수 있는 Amazon Virtual Private Cloud 흐름 로그 쿼리 방법을 설명서에 추가했습니다. 자세한 설명은 Amazon VPC 흐름 로그 쿼리을 참조하세요. 업데이트된 ALB 로그 쿼리 예시 자세한 설명은 Application Load Balancer 로그 쿼리을 참조하세요.	2018년 8월 7일
뷰 지원이 추가되었습니다. 다양한 데이터 스토리지 형식의 스키마 조작에 대한 지침을 추가했습니다.	뷰 지원이 추가되었습니다. 자세한 설명은 뷰 작업을 참조하세요. 다양한 데이터 스토리지 형식의 스키마 업데이트를 처리하는 지침이 포함된 이 안내서가 업데이트되었습니다. 자세한 설명은 스키마 업데이트 처리을 참조하세요.	2018년 6월 5일
기본 쿼리 동시 한도가 5개에서 20개로 늘어났습니다.	한 번에 최대 DDL 쿼리 20개와 SELECT 쿼리 20개를 제출하고 실행할 수 있습니다. 자세한 설명은 Service Quotas을 참조하세요.	2018년 5월 17일
쿼리 편집기에 쿼리 탭 및 자동 완성 구성 기능을 추가했습니다.	쿼리 편집기에 쿼리 탭 및 자동 완성 구성 기능을 추가했습니다. 자세한 설명은 시작하기을 참조하세요.	2018년 5월 8일
JDBC 드라이버 버전 2.0.2를 출시했습니다.	JDBC 드라이버 새 버전(버전 2.0.2)을 출시했습니다. 자세한 설명은 JDBC로 Amazon Athena에 연결을 참조하세요.	2018년 4월 19일
Athena 콘솔에서 쿼리를 입력하기 위한 자동 완성 기능을 추가했습니다.	Athena 콘솔에서 쿼리를 입력하기 위한 자동 완성 기능을 추가했습니다.	2018년 4월 6일

변경 사항	설명	릴리스 날짜
CloudTrail 콘솔에서 직접 CloudTrail 로그 파일용 Athena 테이블을 생성할 수 있는 기능이 추가되었습니다.	CloudTrail 콘솔에서 직접 CloudTrail 로그 파일용 Athena 테이블을 자동 생성할 수 있는 기능이 추가되었습니다. 자세한 설명은 CloudTrail 콘솔을 사용하여 CloudTrail 로그용 Athena 테이블 생성 을 참조하세요.	2018년 3월 15일
GROUP BY 절을 사용하는 쿼리에 대해 디스크에 중간 데이터를 안전하게 로드하는 기능이 추가되었습니다.	GROUP BY 절을 사용하는 메모리 집약적인 쿼리에 대해 디스크에 중간 데이터를 안전하게 로드하는 기능이 추가되었습니다. 그러면 이러한 쿼리의 안정성이 개선되어 "쿼리 리소스 모두 사용" 오류를 방지합니다. 자세한 내용은 2018년 2월 2일의 출시 정보를 참조하세요.	2018년 2월 2일
Presto 버전 0.172에 대한 지원이 추가되었습니다.	Amazon Athena의 기본 엔진을 Presto 버전 0.172에 기반한 버전으로 업그레이드했습니다. 자세한 내용은 2018년 1월 19일의 출시 정보를 참조하세요.	2018년 1월 19일
ODBC 드라이브에 대한 지원이 추가되었습니다.	Athena를 ODBC 드라이버에 연결하는 지원이 추가되었습니다. 자세한 내용은 ODBC로 Amazon Athena에 연결을 참조하세요.	2017년 11월 13일
아시아 태평양(서울), 아시아 태평양(뭄바이), 유럽(런던) 리전에 대한 지원이 추가되었습니다. 지리 공간 데이터 쿼리에 대한 지원이 추가되었습니다.	지리 공간 데이터 쿼리, 아시아 태평양(서울), 아시아 태평양(뭄바이), 유럽(런던) 리전에 대한 지원이 추가되었습니다. 자세한 내용은 지리 공간 데이터 쿼리 및 AWS 리전 및 엔드포인트를 참조하세요.	2017년 11월 1일
유럽(프랑크푸르트)에 대한 지원이 추가되었습니다.	유럽(프랑크푸르트)에 대한 지원이 추가되었습니다. 지원되는 리전 목록은 AWS 리전 및 엔드포인트를 참조하세요.	2017년 10월 19일

변경 사항	설명	릴리스 날짜
AWS CloudFormation을 통해 명명된 Athena 쿼리에 대한 지원이 추가되었습니다.	AWS CloudFormation을 통해 명명된 Athena 쿼리를 생성할 수 있는 기능이 추가되었습니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 AWS::Athena::NamedQuery를 참조하세요.	2017년 10월 3일
아시아 태평양(시드니)에 대한 지원이 추가되었습니다.	아시아 태평양(시드니)에 대한 지원이 추가되었습니다. 지원되는 리전 목록은 AWS 리전 및 엔드포인트를 참조하세요.	2017년 9월 25일
AWS 서비스 로그와 맵, 배열, 중첩 데이터, JSON 포함 데이터와 같은 다양한 형식의 데이터를 쿼리할 수 있는 섹션이 이 설명서에 추가되었습니다.	AWS 서비스 로그 쿼리와, Athena에서 다양한 형식의 데이터를 쿼리하는 방법에 대한 예제가 추가되었습니다. 자세한 설명은 Amazon Athena를 사용하여 SQL 쿼리 실행을 참조하세요.	2017년 9월 5일
AWS Glue Data Catalog에 대한 지원이 추가되었습니다.	AWS Glue Data Catalog 통합 기능과 Athena 관리형 데이터 카탈로그에서 AWS Glue Data Catalog로 업데이트하기 위한 마이그레이션 마법사가 추가되었습니다. 자세한 내용은 AWS Glue와 통합 및 AWS Glue를 참조하세요.	2017년 8월 14일
Grok SerDe에 대한 지원이 추가되었습니다.	로그 같은 비정형 텍스트 파일의 레코드에 대해 보다 쉬운 패턴 일치를 제공하는 Grok SerDe 지원이 추가되었습니다. 자세한 내용은 Grok SerDe를 참조하세요. 콘솔을 사용 시 쿼리 기록을 스크롤하는 키보드 단축키가 추가되었습니다.	2017년 8월 4일
아시아 태평양(도쿄)에 대한 지원이 추가되었습니다.	아시아 태평양(도쿄) 및 아시아 태평양(싱가포르) 리전에 대한 지원이 추가되었습니다. 지원되는 리전 목록은 AWS 리전 및 엔드포인트를 참조하세요.	2017년 6월 22일

변경 사항	설명	릴리스 날짜
유럽(아일랜드)에 대한 지원이 추가되었습니다.	유럽(아일랜드)에 대한 지원이 추가되었습니다. 자세한 내용은 AWS 리전 및 엔드포인트를 참조하세요.	2017년 6월 8일
Amazon Athena API 및 AWS CLI 지원이 추가되었습니다.	Amazon Athena API 및 Athena용 AWS CLI 지원이 추가되었습니다. JDBC 드라이버가 버전 1.1.0으로 업데이트되었습니다.	2017년 5월 19일
Amazon S3 데이터 암호화에 대한 지원이 추가되었습니다.	Amazon S3 데이터 암호화에 대한 지원이 추가되었으며 암호화 지원, 개선 및 버그 수정을 포함한 JDBC 드라이버 업데이트(버전 1.0.1)가 릴리스되었습니다. 자세한 내용은 저장 중 암호화 단원을 참조하세요.	2017년 4월 4일
AWS CloudTrail SerDe가 추가되었습니다.	AWS CloudTrail SerDe가 추가되고 성능이 개선되었으며 파티션 문제가 해결되었습니다. • AWS CloudTrail SerDe는 CloudTrail 로그 읽기용 Hive JSON SerDe로 대체되었습니다. CloudTrail 로그 쿼리에 대한 자세한 내용은 AWS CloudTrail 로그 쿼리 단원을 참조하세요. • 많은 수의 파티션을 검사할 때 성능이 향상되었습니다. • `MSCK Repair Table` 작업의 성능이 개선되었습니다. • 기본 리전 이외의 리전에 저장된 Amazon S3 데이터를 쿼리하는 기능이 추가되었습니다. 표준 Athena 요금 외에 Amazon S3의 표준 리전 간 데이터 전송 요금이 적용됩니다.	2017년 3월 24일
미국 동부(오하이오)에 대한 지원이 추가되었습니다.	Avro SerDe 및 CSV 처리를 위한 OpenCSVSerDe, 미국 동부(오하이오), 콘솔 마법사의 일괄 편집 열에 대한 지원이 추가되었습니다. 대용량 Parquet 테이블의 성능이 개선되었습니다.	2017년 20월 2일
	Amazon Athena 사용 설명서의 최초 릴리스입니다.	2016년 11월

AWS 용어집

최신 AWS 용어는 AWS 용어집참조의 AWS 용어집을 참조하세요.