aws

개발자 가이드

Amazon Quantum Ledger Database(QLDB)

Amazon Quantum Ledger Database(QLDB): 개발자 가이드

Table of Contents

Amazon QLDB란 무엇인가요?

Amazon Quantum Ledger Database(QLDB)는 완전관리형 원장 데이터베이스로, 중앙의 신뢰할 수 있는 기관이 소유하는 투명하고, 변경 불가능하며, 암호화 방식으로 검증 가능한 트랜잭션 로그를 제공합니다. Amazon QLDB를 사용하여 모든 애플리케이션 데이터 변경 내용을 추적하며 완전하고 검증 가능한 시간대별 변경 기록을 유지합니다. Amazon Web Services Services에서 사용할 수 있는 다양한 데이터베이스 옵션에 대해 자세히 알아보려면 Choosing the right database for your organization on AWS(AWS 에서 조직에 적합한 데이터베이스 선택)를 참조하세요.

원장은 일반적으로 조직의 경제 및 금융 활동 기록을 기록하는 데 사용됩니다. 많은 조직들에서 애플리케이션 데이터의 정확한 기록을 유지하기를 원하기 때문에 원장과 유사한 기능을 갖춘 애플리케이션을 빌드합니다. 예를 들어, 은행 거래의 대변 및 차변 내역을 추적하거나, 보험 청구의 데이터 계보를 확인하거나, 공급망 네트워크에서 품목의 움직임을 추적하고자 할 수 있습니다. 원장 애플리케이션은 관계형 데이터베이스에서 만든 사용자 지정 감사 테이블이나 감사 추적을 사용하여 구현되는 경우가 많습니다.

Amazon QLDB는 원장과 유사한 애플리케이션을 직접 빌드하기 위해 복잡한 개발 노력을 들일 필요가 없도록 도와주는 새로운 데이터베이스 클래스입니다. QLDB를 사용하면 데이터 변경 기록을 변경할 수 없습니다. 즉, 덮어쓰거나 변경할 수 없습니다. 또한 암호화를 사용하면 애플리케이션 데이터에 의도하지 않은 변경이 없었는지 확인할 수 있습니다. QLDB는 저널이라고 하는 변경할 수 없는 트랜잭션 로그를 사용합니다. 저널은 추가만 가능하며, 커밋된 데이터를 포함하는 순서화된 해시 체인 블록 세트로 구성됩니다.

Amazon QLDB 동영상

Amazon QLDB에 대한 개요와 이를 통해 얻을 수 있는 이점을 알아보려면 YouTube에서 이 QLDB 개요 동영상을 시청하세요.

Amazon QLDB 요금

Amazon QLDB를 사용할 경우 최소 요금 또는 필수 서비스 사용 없이 사용하는 내역에 대해서만 요금을 지불하면 됩니다. 원장 데이터베이스에서 사용한 리소스에 대해서만 비용을 지불하고 미리 프로비저닝할 필요는 없습니다.

자세한 내용은 Amazon SES 요금을 참조하세요.

QLDB 시작하기

다음 주제들을 읽고 시작하면 도움이 됩니다.

- Amazon QLDB 개요 - QLDB에 대한 상위 수준의 개요를 살펴봅니다.
- Amazon QLDB의 핵심 개념 및 용어 - QLDB의 기본 개념 및 용어를 학습합니다.
- Amazon QLDB 액세스 – AWS Management Console, API 또는 AWS Command Line Interface (AWS CLI)를 사용하여 QLDB에 액세스하는 방법을 학습니다.
- Amazon QLDB에서 IAM을 사용하는 방법 - AWS Identity and Access Management(IAM)을 사용하여 QLDB에 대한 액세스를 제어하는 방법을 알아봅니다.

QLDB 콘솔을 신속하게 시작하려면 Amazon QLDB 콘솔 시작하기 섹션을 참조하세요.

AWS 제공 드라이버를 사용하여 QLDB로 개발하는 방법에 대한 자세한 내용은 Amazon QLDB 드라이버 시작하기 섹션을 참조하세요.

Amazon QLDB 개요

다음 단원에서 Amazon QLDB 서비스 구성 요소 및 요소 간 상호 작용 방식을 간단히 설명합니다.

주제

- 저널 우선
- 변경 불가능
- 암호학적으로 검증 가능
- SQL와 유사하고 유연한 문서
- 오픈 소스 개발자 도구
- 서버리스 및 고가용성
- 엔터프라이즈급

저널 우선

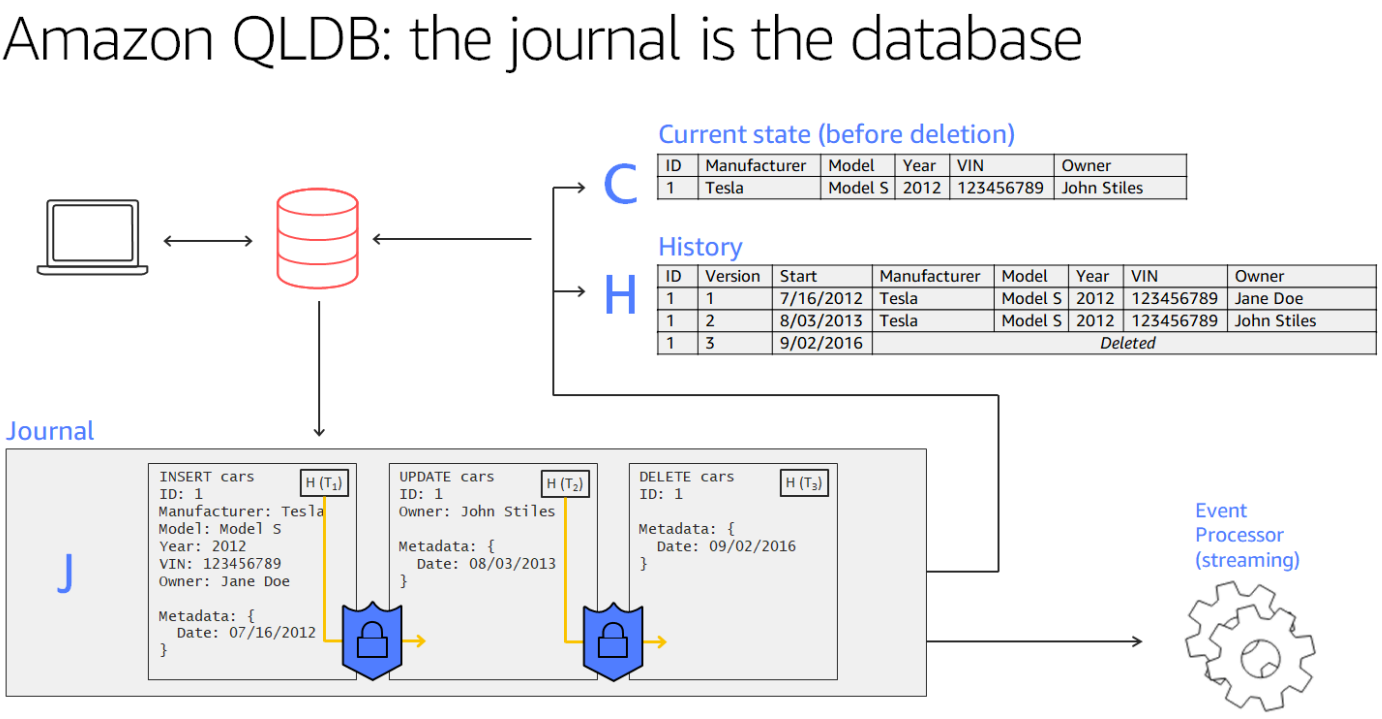
기존 데이터베이스 아키텍처에서는 일반적으로 트랜잭션의 일부로 테이블에 데이터를 씁니다. 일반적으로 내부 구현인 트랜잭션 로그는 모든 트랜잭션과 해당 트랜잭션에서 수행한 데이터베이스 수정 내용을 기록합니다. 트랜잭션 로그는 데이터베이스의 중요한 구성 요소입니다. 시스템 장애, 재해 복구

또는 데이터 복제 시 트랜잭션을 재생하려면 로그가 필요합니다. 하지만 데이터베이스 트랜잭션 로그는 변경할 수 없으며, 사용자가 직접 쉽게 액세스할 수 있도록 설계되지 않았습니다.

Amazon QLDB에서 저널은 데이터베이스의 핵심입니다. 트랜잭션 로그와 구조적으로 유사한 저널은 변경 불가하며, 애플리케이션 데이터를 관련 메타데이터와 함께 저장하는 추가 전용 데이터 구조입니다. 업데이트와 삭제를 포함한 모든 쓰기 트랜잭션은 저널 우선에 커밋됩니다.

QLDB는 저널을 사용하여 쿼리 가능한 사용자 정의 테이블로 원장 데이터를 구체화하여 원장 데이터의 현재 상태를 확인합니다. 또한 이 테이블은 문서 수정 및 메타데이터를 포함한 모든 트랜잭션 데이터에 대한 액세스 가능한 기록을 제공합니다. 또한 저널은 원장 데이터의 동시성, 시퀀싱, 암호화 검증 및 가용성을 처리합니다.

다음 다이어그램에 QLDB 저널 아키텍처가 나와 있습니다.



- 이 예제에서 애플리케이션은 원장에 연결하여 `cars`라는 이름의 테이블에 문서를 삽입, 업데이트 및 삭제하는 트랜잭션을 실행합니다.
- 데이터는 먼저 순서대로 저널에 기록됩니다.
- 그런 다음 데이터는 기본 제공 뷰가 있는 테이블로 구체화됩니다. 이러한 뷰를 통해 차량의 현재 상태와 전체 기록을 모두 쿼리할 수 있으며, 각 수정본에 버전 번호가 할당됩니다.
- 저널에서 직접 데이터를 내보내거나 스트리밍할 수도 있습니다.

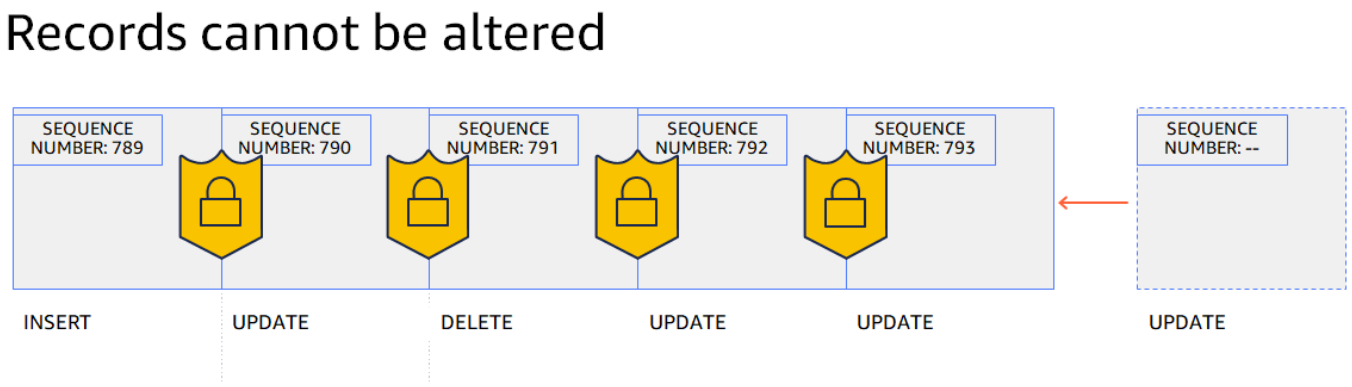
변경 불가능

QLDB 저널은 추가 전용이므로 수정하거나 덮어쓸 수 없는 모든 데이터 변경 내용을 모두 기록합니다. 커밋된 데이터를 변경할 수 있는 API나 다른 방법은 없습니다. 이 저널 구조를 사용하면 원장의 전체 기록에 액세스하고 쿼리할 수 있습니다.

> **Note**
>
> QLDB가 지원하는 불변성의 유일한 예외는 데이터 수정입니다. 이 기능을 사용하면 유럽 연합의 일반 데이터 보호 규정(GDPR) 및 캘리포니아 소비자 개인정보보호법(CCPA)과 같은 규제 법령을 준수할 수 있습니다.
>
> QLDB는 테이블 기록에서 비활성 문서 수정본을 영구적으로 삭제할 수 있는 수정 작업을 제공합니다. 이 작업을 수행하면 지정된 수정 버전의 사용자 데이터만 삭제되고 저널 시퀀스와 문서 메타데이터는 변경되지 않습니다. 이렇게 하면 원장의 전체 데이터 무결성이 유지됩니다. 자세한 내용은 문서 개정본 수정하기 섹션을 참조하세요.

QLDB는 트랜잭션의 저널에 블록 하나를 기록합니다. 각 블록에는 삽입, 업데이트 및 삭제한 문서를 나타내는 항목 객체와 해당 문서를 커밋하기 위해 실행한 명령문이 포함되어 있습니다. 이러한 블록은 데이터 무결성을 보장하기 위해 시퀀싱되고 해시 체인으로 연결됩니다.

다음 다이어그램은 이 저널 구조를 보여줍니다.



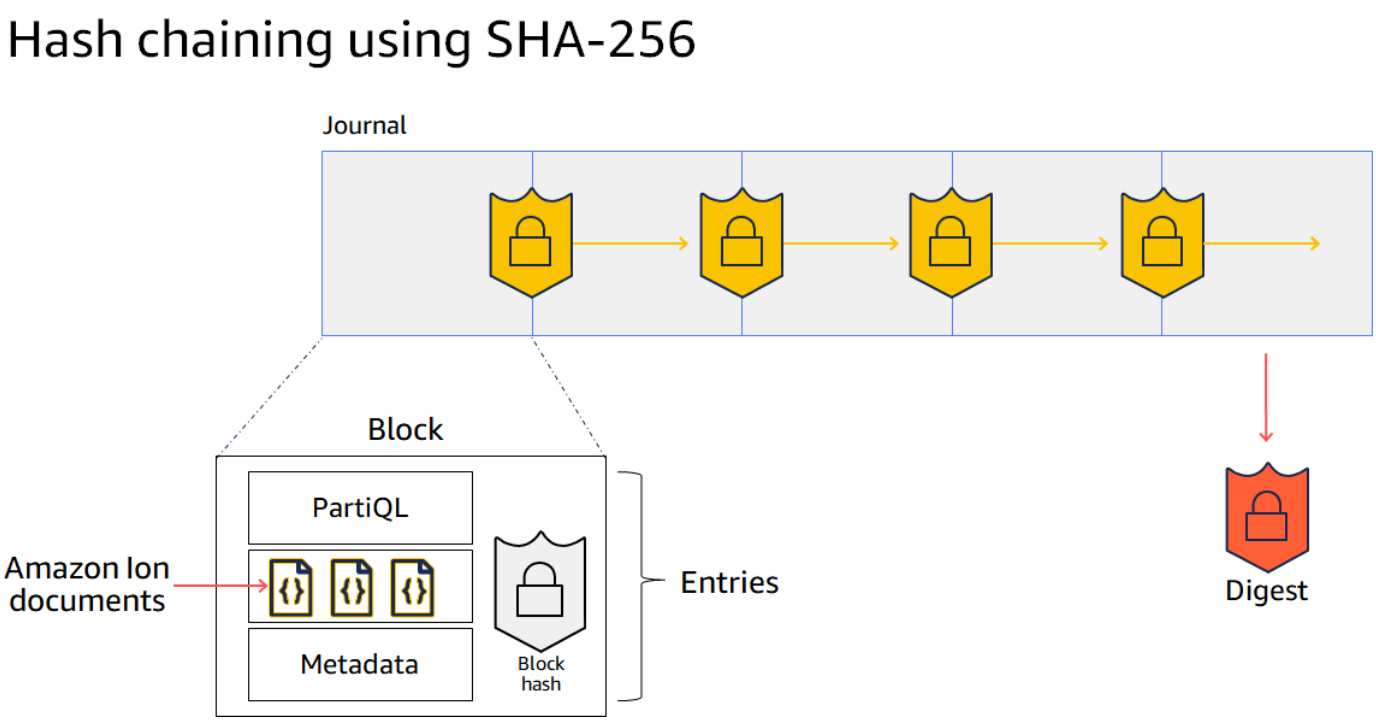
다이어그램은 트랜잭션이 검증을 위해 해시 체인으로 연결된 블록으로 저널에 커밋되는 것을 보여줍니다. 각 블록에는 주소를 지정하는 시퀀스 번호가 있습니다.

암호학적으로 검증 가능

저널 블록은 블록체인과 유사한 암호화 해싱 기법을 사용하여 시퀀싱되고 함께 연결됩니다. QLDB는 저널의 해시 체인을 사용하여 암호화 검증 방법을 통해 트랜잭션 데이터 무결성을 제공합니다. 다이제

스트(특정 시점을 기준으로 저널의 전체 해시 체인을 나타내는 해시 값)와 Merkle 감사 증명(바이너리 해시 트리 내 모든 노드의 유효성을 입증하는 메커니즘)을 사용하면 언제든지 데이터에 의도하지 않은 변경이 없었는지 확인할 수 있습니다.

다음 다이어그램은 특정 시점에서 저널의 전체 해시 체인을 다루는 다이제스트를 보여줍니다.



이 다이어그램에서 저널 블록은 SHA-256 암호화 해시 함수를 사용하여 해싱되고 후속 블록에 순차적으로 연결됩니다. 각 블록에는 데이터 문서, 메타데이터 및 트랜잭션에서 실행된 PartiQL 명령문을 포함하는 항목이 들어 있습니다.

자세한 내용은 Amazon QLDB에서의 데이터 확인 섹션을 참조하세요.

SQL와 유사하고 유연한 문서

QLDB는 PartiQL을 쿼리 언어로 사용하고 Amazon Ion을 문서 지향 데이터 모델로 사용합니다. PartiQL은 Ion과 함께 작동하도록 확장된 오픈 소스 SQL 호환 쿼리 언어입니다. PartiQL을 사용하면 익숙한 SQL 연산자를 사용하여 데이터를 삽입, 쿼리 및 관리할 수 있습니다. 플랫 문서를 쿼리할 때 구문은 SQL을 사용하여 관계형 테이블을 쿼리하는 것과 동일합니다. PartiQL의 QLDB 구현에 대한 자세한 내용은 Amazon QLDB PartiQL 참조 섹션을 참조하세요.

Amazon Ion은 JSON의 상위 집합입니다. Ion은 정형, 반정형 및 중첩 데이터를 저장하고 처리할 수 있는 유연성을 제공하는 오픈 소스 문서 기반 데이터 형식입니다. QLDB에 대해 자세히 알아보려면 Amazon QLDB의 Amazon Ion 데이터 형식 참조 섹션을 참조하세요.

기존 관계형 데이터베이스와 QLDB의 핵심 구성 요소 및 기능에 대한 상위 수준의 비교는 관계형에서 원장까지 섹션을 참조하세요.

오픈 소스 개발자 도구

애플리케이션 개발을 단순화하기 위해 QLDB는 다양한 프로그래밍 언어로 오픈 소스 드라이버를 제공합니다. 이러한 드라이버를 사용하면 원장에서 PartiQL 명령문을 실행하고 해당 명령문의 결과를 처리하여 트랜잭션 데이터 API와 상호 작용할 수 있습니다. 현재 지원되는 드라이버 언어에 대한 정보 및 자습서는 Amazon QLDB 드라이버 시작하기 섹션을 참조하세요.

Amazon Ion은 사용자를 대신해 Ion 데이터를 처리하는 클라이언트 라이브러리도 제공합니다. Ion 데이터 처리에 대한 개발자 가이드와 코드 예제는 GitHub의 Amazon Ion 설명서를 참조하세요.

서버리스 및 고가용성

QLDB는 완전 관리형, 서버리스, 고가용성 기능을 제공합니다. 애플리케이션의 수요를 지원할 수 있도록 서비스의 규모가 자동으로 조정되므로 인스턴스나 용량을 프로비저닝할 필요가 없습니다. 여러 데이터 사본이 가용 영역과 AWS 리전의 가용 영역 전체에 복제됩니다.

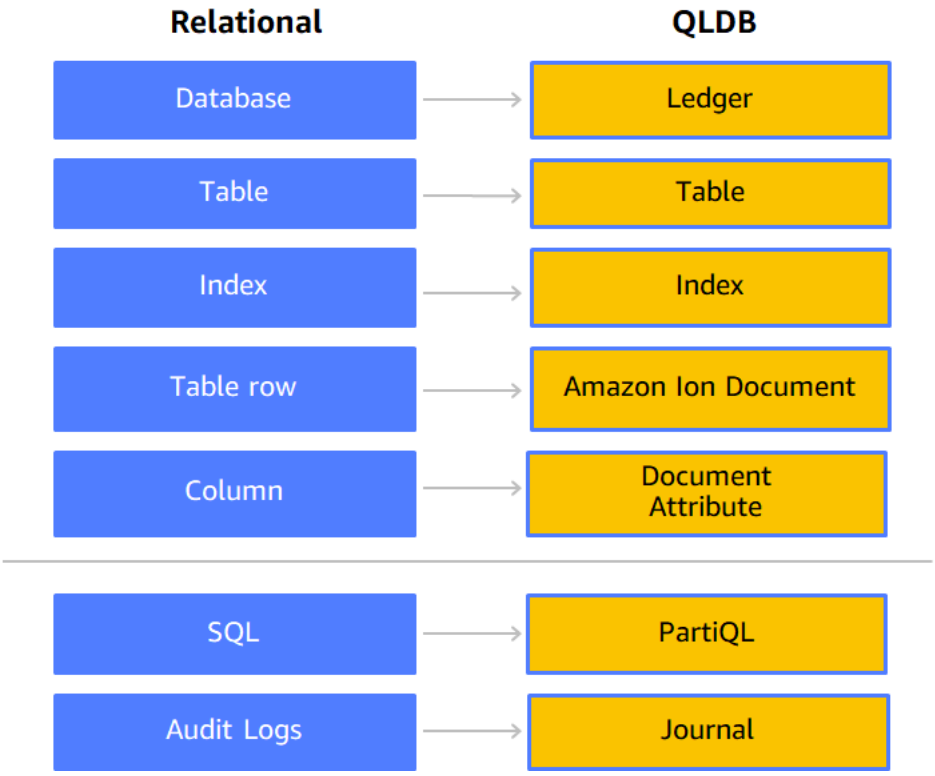
엔터프라이즈급

QLDB 트랜잭션은 원자성, 일관성, 격리 및 내구성(ACID) 속성을 완벽하게 준수합니다. QLDB는 OCC(낙관적 동시성 제어)를 사용하며 트랜잭션은 최고 수준의 격리인 완전한 직렬화 기능으로 작동합니다. 즉, 가상 읽기, 더티 읽기, 쓰기 스큐 또는 기타 유사한 동시성 문제가 발생할 위험이 없습니다. 자세한 내용은 Amazon QLDB 동시성 모델 섹션을 참조하세요.

관계형에서 원장까지

애플리케이션 개발자라면 RDBMS(관계형 데이터베이스 관리 시스템) 및 SQL(Structured Query Language)을 사용한 경험이 어느 정도 있을 것입니다. Amazon QLDB 작업을 시작해 보면 비슷한 점을 상당히 많이 발견할 수 있습니다. 고급 주제로 진행하면서 QLDB가 RDBMS를 기반으로 구축한 강력한 새 기능도 만나게 됩니다. 이 단원에서는 이들을 QLDB에서 상응하는 구성 요소 및 작업과 비교 대조하면서 공통 데이터베이스 구성 요소와 작업에 대해 설명합니다.

다음 다이어그램은 기존 RDBMS와 Amazon QLDB 간의 핵심 구성 요소 매핑 구조를 보여줍니다.



다음 테이블은 기존 RDBMS와 QLDB 간의 기본 제공 운영 기능의 주요 유사점과 차이점을 개괄적으로 보여줍니다.

작업	RDBMS	QLDB
테이블 생성	모든 열 이름과 데이터 유형을 정의하는 CREATE TABLE 명령문	스키마 없이 개방형 콘텐츠를 허용하기 위해 테이블 속성이나 데이터 유형을 정의하지 않는 CREATE TABLE 명령문
인덱스 생성	CREATE INDEX 명령문	테이블의 모든 최상위 필드에 대한 CREATE INDEX 명령문
데이터 삽입	테이블에 정의된 스키마를 준수하는 새 행 또는 튜플 내의 값을 지정하는 INSERT 명령문	테이블의 기존 문서와 상관없이 모든 유효한 Amazon Ion 형식으로 새 문서 내의 값을 지정하는 INSERT 명령문
데이터 쿼리	SELECT-FROM-WHERE 명령문	플랫 문서를 쿼리할 때 SQL과 동일한 구문을 사용하는

작업	RDBMS	QLDB
		SELECT-FROM-WHERE 명령문
데이터 업데이트	UPDATE-SET-WHERE 명령문	플랫 문서를 업데이트할 때 SQL과 동일한 구문을 사용하는 UPDATE-SET-WHERE 명령문
데이터 삭제	DELETE-FROM-WHERE 명령문	플랫 문서를 삭제할 때 SQL과 동일한 구문을 사용하는 DELETE-FROM-WHERE 명령문
중첩 및 반정형 데이터	플랫 행 또는 튜플만	Amazon Ion 데이터 형식 및 PartiQL 쿼리 언어에서 지원하는 모든 정형, 반정형 또는 중첩 데이터를 포함할 수 있는 문서
메타데이터 쿼리	기본 제공 메타데이터 없음	테이블의 기본 제공 커밋된 뷰에서 쿼리하는 SELECT 명령문
개정 기록 쿼리	기본 제공 데이터 기록 없음	기본 제공 기록 기능에서 쿼리하는 SELECT 명령문
암호학적 검증	내장된 암호화 또는 불변성 없음	저널 다이제스트를 반환하고 해당 다이제스트와 관련된 모든 문서 개정의 무결성을 검증하는 증거를 반환하는 API

QLDB의 핵심 개념 및 용어에 대한 개요는 핵심 개념 섹션을 참조하세요.

원장의 데이터를 생성, 쿼리 및 관리하는 프로세스에 대한 자세한 내용은 데이터 및 기록 작업 섹션을 참조하세요.

Amazon QLDB의 핵심 개념 및 용어

이 섹션에서는 원장 구조 및 원장의 데이터 관리 방법을 포함하여 Amazon QLDB의 핵심 개념 및 용어에 대한 개요를 제공합니다. 원장 데이터베이스인 QLDB는 다음과 같은 주요 개념에서 다른 문서 지향 데이터베이스와는 다릅니다.

주제

QLDB 데이터 객체 모델

Amazon QLDB의 기본 데이터 객체 모델은 다음과 같이 설명할 수 있습니다.

1. 원장

   첫 번째 단계는 QLDB에서 기본 AWS 리소스 타입인 원장을 만드는 것입니다. 원장 생성 방법을 알아보려면 콘솔 시작하기의 1단계: 새 원장 생성 또는 Amazon QLDB 원장의 기본 작업을 참조하세요.

   원장의 ALLOW_ALL 및 STANDARD 권한 모드 모두에 대해 이 원장 리소스에서 API 작업을 실행할 권한을 부여하는 AWS Identity and Access Management(IAM)정책을 생성합니다.

   원장 ARN 형식:

   ```
   arn:aws:qldb:${region}:${account-id}:ledger/${ledger-name}
   ```

2. 저널 및 표

   QLDB 원장에 데이터 쓰기를 시작하려면 먼저 기본 CREATE TABLE 명령문을 사용하여 표를 생성합니다. 원장 데이터는 원장 저널에 체결된 문서의 수정본으로 구성되어 있습니다. 사용자 정의 표의 컨텍스트에서 원장에 문서 수정본을 체결합니다. QLDB에서 표은 저널의 문서 수정본 모음에 대한 구체화된 뷰를 보여줍니다.

원장의 STANDARD 권한 모드에서는 이 테이블 리소스에서 PartiQL 문을 실행할 권한을 부여하는 IAM 정책을 생성해야 합니다. 테이블 리소스에 대한 권한이 있으면 테이블의 현재 상태에 액세스하는 문을 실행할 수 있습니다. 또한 내장 `history()` 함수를 사용하여 테이블의 개정 이력을 쿼리할 수 있습니다.

표 ARN 형식:

```
arn:aws:qldb:${region}:${account-id}:ledger/${ledger-name}/table/${table-id}
```

원장 및 관련 리소스에 권한을 부여하는 자세한 방법은 Amazon QLDB에서 IAM을 사용하는 방법 섹션을 참조하세요.

3. 문서

   표은 Amazon Ion `struct` 형식의 데이터 세트인 QLDB 문서의 수정본으로 구성되어 있습니다. 문서 수정본은 고유한 문서 ID로 식별되는 문서 시퀀스의 단일 버전을 나타냅니다.

   QLDB는 체결된 문서의 전체 변경 기록을 저장합니다. 표를 사용하면 문서의 현재 상태를 쿼리하고 `history()` 함수를 사용하면 표 문서의 전체 수정 기록을 쿼리할 수 있습니다. 수정 쿼리 및 작성에 대한 자세한 설명은 데이터 및 기록 작업 섹션을 참조하세요.

4. 시스템 카탈로그

   또한 각 원장은 원장의 모든 테이블과 인덱스를 나열하기 위해 쿼리할 수 있는 시스템 정의 카탈로그 리소스를 제공합니다. 원장의 STANDARD 권한 모드에서는 다음 작업을 수행하려면 이 카탈로그 리소스에 대한 `qldb:PartiQLSelect` 권한이 필요합니다.

   - 시스템 카탈로그 표 information_schema.user_tables에 대한 `SELECT` 문을 실행합니다.
   - QLDB 콘솔의 원장 세부 정보 페이지에서 표 및 인덱스 정보를 볼 수 있습니다.
   - QLDB 콘솔의 PartiQL 편집기에서 표 및 인덱스 목록을 볼 수 있습니다.

   카탈로그 ARN 형식:

```
arn:aws:qldb:${region}:${account-id}:ledger/${ledger-name}/information_schema/
user_tables
```

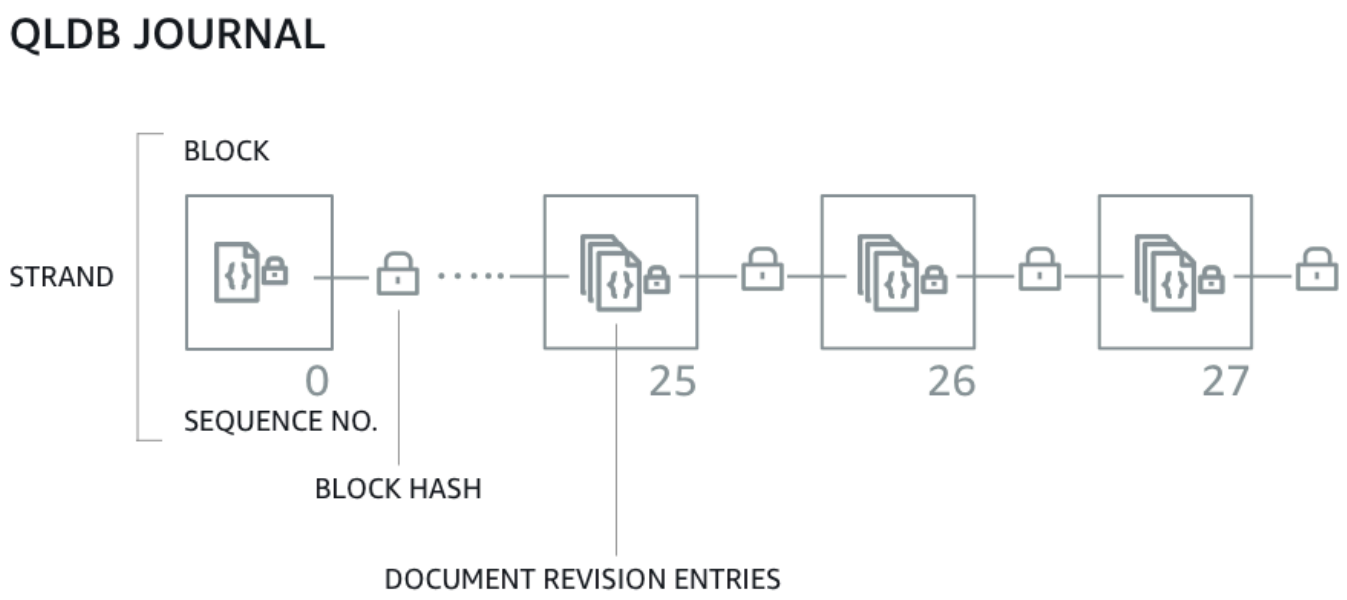
저널 우선 트랜잭션

애플리케이션이 QLDB 원장에서 데이터를 읽거나 쓸 때 데이터베이스 트랜잭션에서 데이터를 읽거나 씁니다. Amazon QLDB 할당량 및 제한에 정의된 대로 모든 트랜잭션에는 제한이 적용됩니다. QLDB는 트랜잭션 내에서 다음 단계를 수행합니다.

1. 원장에서 데이터의 현재 상태를 읽습니다.
2. 트랜잭션에 제공된 문을 수행한 다음 OCC(낙관적 동시성 제어)를 사용하여 충돌이 없는지 확인하여 직렬화가 완전히 가능한지 확인합니다.
3. OCC 충돌이 발견되지 않으면 다음과 같이 트랜잭션 결과를 반환합니다.
   - 읽기의 경우, 결과 세트를 반환하고 추가 전용 방식으로 SELECT 문을 저널에 체결합니다.
   - 쓰기의 경우 업데이트, 삭제 또는 새로 삽입된 데이터를 추가 전용 방식으로 저널에 체결합니다.

저널은 데이터의 모든 변경 사항에 대한 완전하고 변경할 수 없는 기록을 나타냅니다. QLDB는 트랜잭션의 저널에 체인 블록 하나를 기록합니다. 각 블록에는 삽입, 업데이트 및 삭제한 문서 수정본을 나타내는 항목 객체와 이를 체결한 PartiQL 문이 들어 있습니다.

다음 다이어그램은 이 저널 구조를 보여줍니다.



다이어그램은 트랜잭션이 문서 수정본 항목이 포함된 블록 형태로 저널에 커밋됨을 보여줍니다. 각 블록은 검증을 위해 해싱되고 후속 블록에 연결됩니다. 각 블록에는 스트랜드 내의 주소를 지정하는 시퀀스 번호가 있습니다.

> **Note**
>
> Amazon QLDB에서 스트랜드는 원장 저널의 파티션입니다. QLDB는 현재 단일 스트랜드가 포함된 저널만 지원합니다.

블록의 데이터 내용에 대해 자세히 알아보려면 Amazon QLDB의 저널 콘텐츠 섹션을 참조하세요.

데이터 쿼리

QLDB는 고성능 온라인 트랜잭션 처리(OLTP) 워크로드의 요구 사항을 해결하기 위한 것입니다. 원장은 저널에 체결된 트랜잭션 정보를 기반으로 데이터의 쿼리 가능한 표 보기를 제공합니다. QLDB의 표 뷰는 표 데이터의 하위 집합입니다. 뷰는 실시간으로 유지 관리되므로 애플리케이션에서 항상 쿼리할 수 있습니다.

PartiQL SELECT 문을 사용하여 다음과 같은 시스템 정의 뷰를 쿼리할 수 있습니다.

- 사용자 - 테이블에 기록한 데이터의 최신 활성 개정 (즉, 사용자 데이터의 현재 상태)입니다. 이는 QLDB의 기본 뷰입니다.
- 커밋됨 - 사용자 데이터와 시스템에서 생성한 메타데이터의 최신 활성 개정입니다. 이는 사용자 테이블에 직접 해당하는 전체 시스템 정의 테이블입니다.

쿼리 가능한 이러한 뷰 외에도 내장된 기록 함수을 사용하여 데이터의 수정 기록을 쿼리할 수 있습니다. 기록 함수는 체결된 뷰와 동일한 스키마에서 사용자 데이터와 연결된 메타데이터를 모두 반환합니다.

데이터 스토리지

QLDB에는 다음 두 가지 타입의 데이터 스토리지가 있습니다.

- 저널 스토리지 - 원장의 저널이 사용하는 디스크 공간입니다. 저널은 추가 전용으로, 데이터에 대한 모든 변경 사항에 대한 완전하고 변경 불가능하며 검증 가능한 기록을 포함합니다.
- 인덱싱된 스토리지 - 원장의 표, 인덱스, 인덱싱된 기록에 사용되는 디스크 공간입니다. 인덱싱된 스토리지는 고성능 쿼리에 최적화된 원장 데이터로 구성됩니다.

데이터가 저널에 체결되면 정의한 표로 구체화됩니다. 이러한 표은 더 빠르고 효율적인 쿼리를 위해 최적화되었습니다. 애플리케이션이 트랜잭션 데이터 API를 사용하여 데이터를 읽으면 인덱싱된 스토리지에 저장된 표와 인덱스에 액세스합니다.

QLDB API 모델

QLDB는 애플리케이션 코드가 상호 작용할 수 있는 두 가지 타입의 API를 제공합니다.

- Amazon QLDB - QLDB 리소스 관리 API(컨트롤 플레인이라고도 함). 이 API는 원장 리소스 관리 및 비트랜잭션 데이터 작업에만 사용됩니다. 이러한 작업을 사용하여 원장을 생성, 삭제, 설명, 나열 및 업데이트할 수 있습니다. 또한 데이터를 암호로 검증하고 저널 블록을 내보내거나 스트리밍할 수 있습니다.
- Amazon QLDB 세션 - QLDB 트랜잭션 데이터 API. 이 API를 사용하여 PartiQL 문을 사용하여 원장에서 데이터 트랜잭션을 실행할 수 있습니다.

> **Important**
>
> QLDB 세션 API와 직접 상호 작용하는 대신 QLDB 드라이버 또는 QLDB 쉘을 사용하여 원장에서 데이터 트랜잭션을 실행하는 것이 좋습니다.
>
> - AWS SDK로 작업하는 경우 QLDB 드라이버를 사용하세요. 드라이버는 이 QLDB 세션 데이터 API 위에 높은 수준의 추상화 계층을 제공하고 사용자를 대신하여 SendCommand 작업을 관리합니다. 지원되는 프로그래밍 언어에 대한 자세한 내용 및 목록은 드라이버 시작하기 섹션을 참조하세요.
> - AWS CLI를 사용하여 작업하는 경우 QLDB 쉘을 사용하세요. 쉘은 QLDB 드라이버를 사용하여 원장과 상호 작용하는 명령줄 인터페이스입니다. 자세한 내용은 Amazon QLDB 쉘 사용(데이터 API만 해당)을 참조하세요.

이러한 API 작업에 대한 자세한 설명은 Amazon QLDB API 참조 섹션을 참조하세요.

다음 단계

데이터에 원장을 사용하는 방법을 알아보려면 Amazon QLDB에서 데이터 및 기록 작업 섹션을 참조하고 표 생성, 데이터 삽입 및 기본 쿼리 실행 프로세스를 설명하는 예를 따르세요. 이 가이드에서는 샘플 데이터와 컨텍스트에 대한 쿼리 예를 사용하여 이러한 개념이 어떻게 작동하는지 자세히 설명합니다.

QLDB 콘솔을 사용하여 샘플 애플리케이션 자습서를 빠르게 시작하려면 Amazon QLDB 콘솔 시작하기 센션을 참조하세요.

이 섹션에 설명된 주요 용어 및 정의 목록은 Amazon QLDB 용어집 섹션을 참조하세요.

Amazon QLDB의 저널 콘텐츠

Amazon QLDB에서 저널은 데이터의 모든 변경 사항에 대한 완전하고 검증 가능한 기록을 저장하는 변경 불가능한 트랜잭션 로그입니다. 저널은 추가만 가능하며, 커밋된 데이터 및 기타 시스템 메타데이터를 포함하는 순서화된 해시 체인 블록 세트로 구성됩니다. QLDB는 트랜잭션의 저널에 체인 블록 하나를 기록합니다.

이 섹션에서는 샘플 데이터가 포함된 저널 블록의 예를 제공하고 블록의 내용을 설명합니다.

주제

블록 예제

저널 블록에는 트랜잭션에서 커밋된 문서 개정 및 이를 커밋한 PartiQL 문을 나타내는 항목과 함께 트랜잭션 메타데이터가 들어 있습니다.

다음은 샘플 데이터가 있는 블록의 예제입니다.

> **Note**
>
> 이 블록 예제는 정보 제공 목적으로만 제공됩니다. 표시된 해시는 실제 계산된 해시 값이 아닙니다.

```
{
  blockAddress:{
    strandId:"4o5UuzWSW5PIoOGm5jPA6J",
    sequenceNo:25
  },
  transactionId:"3gtB8Q8dfIMA8lQ5pzHAMo",
  blockTimestamp:2022-06-08T18:46:46.512Z,
  blockHash:{{QS5lJt8vRxT30L9OGL5oU1pxFTe+UlEwakYBCrvGQ4A=}},
  entriesHash:{{buYYc5kV4rrRtJAsrIQnfnhgkzfQ8BKjI0C2vFnYQEw=}},
```

```
 previousBlockHash:{{I1UKRIWUgkM1X6042kcoZ/eN1rn0uxhDTc08zw9kZ5I=}},
 entriesHashList:[
   {{BUCXP6oYgmug2AfPZcAZup2lKolJNTbTuV5RA1VaFpo=}},
   {{cTIRkjuULzp/4KaUEsb/S7+TG8FvpFiZHT4tEJGcANc=}},
   {{3aktJSMyJ3C5StZv4WIJLu/w3D8mGtduZvP0ldKUaUM=}},
   {{GPKIJ1+o8mMZmPj/35ZQXoca2z64MVYMCwqs/g080IM=}}
 ],
 transactionInfo:{
   statements:[
     {
       statement:"INSERT INTO VehicleRegistration VALUE ?",
       startTime:2022-06-08T18:46:46.063Z,
       statementDigest:{{KY2nL6UGUPs5lXCLVXcUaBxcEIop0Jvk4MEjcFVBfwI=}}
     },
     {
       statement:"SELECT p_id FROM Person p BY p_id WHERE p.FirstName = ? and
p.LastName = ?",
       startTime:2022-06-08T18:46:46.173Z,
       statementDigest:{{QS2nfB8XBf2ozlDx0nvtsliOYDSmNHMYC3IRH4Uh690=}}
     },
     {
       statement:"UPDATE VehicleRegistration r SET r.Owners.PrimaryOwner.PersonId = ?
WHERE r.VIN = ?",
       startTime:2022-06-08T18:46:46.278Z,
       statementDigest:{{nGtIA9Qh0/dwIplOR8J5CTeqyUVtNUQgXfltDUo2Aq4=}}
     },
     {
       statement:"DELETE FROM DriversLicense l WHERE l.LicenseNumber = ?",
       startTime:2022-06-08T18:46:46.385Z,
       statementDigest:{{ka783dcEP58Q9AVQ1m9NOJd3JAmEvXLjzl0OjN1BojQ=}}
     }
   ],
   documents:{
     HwVFkn8IMRa0xjze5xcgga:{
       tableName:"VehicleRegistration",
       tableId:"HQZ6cgIMUi204Lq1tT4oaJ",
       statements:[0,2]
     },
     IiPTRxLGJZa342zHFCFT15:{
       tableName:"DriversLicense",
       tableId:"BvtXEB1JxZg0lJlBAtbtSV",
       statements:[3]
     }
   }
```

```
  },
  revisions:[
    {
      hash:{{FR1IWcWew0yw1TnRklo2YMF/qtwb7ohsu5FD8A4DSVg=}}
    },
    {
      blockAddress:{
        strandId:"4o5UuzWSW5PIoOGm5jPA6J",
        sequenceNo:25
      },
      hash:{{6TTHbcfIVdWoFC/j90BOZi0JdHzhjSXo1tW+uHd6Dj4=}},
      data:{
        VIN:"1N4AL11D75C109151",
        LicensePlateNumber:"LEWISR261LL",
        State:"WA",
        City:"Seattle",
        PendingPenaltyTicketAmount:90.25,
        ValidFromDate:2017-08-21,
        ValidToDate:2020-05-11,
        Owners:{
          PrimaryOwner:{
            PersonId:"3Ax20JIix5J2ulu2rCMvo2"
          },
          SecondaryOwners:[]
        }
      },
      metadata:{
        id:"HwVFkn8IMRa0xjze5xcgga",
        version:0,
        txTime:2022-06-08T18:46:46.492Z,
        txId:"3gtB8Q8dfIMA8lQ5pzHAMo"
      }
    },
    {
      blockAddress:{
        strandId:"4o5UuzWSW5PIoOGm5jPA6J",
        sequenceNo:25
      },
      hash:{{ZVF/f1uSqd5DIMqzI04CCHaCGFK/J0Jf5AFzSEk0l90=}},
      metadata:{
        id:"IiPTRxLGJZa342zHFCFT15",
        version:1,
        txTime:2022-06-08T18:46:46.492Z,
        txId:"3gtB8Q8dfIMA8lQ5pzHAMo"
```

```
      }
    }
  ]
}
```

revisions 필드에서 일부 개정 객체에는 hash 값만 포함되고 기타 속성은 포함되지 않을 수 있습니다. 이는 사용자 데이터를 포함하지 않는 내부 전용 시스템 수정본입니다. 이러한 개정판의 해시는 저널의 전체 해시 체인의 일부이며, 이는 암호화 검증에 필요합니다.

블록 콘텐츠

저널 블록에는 다음과 같은 필드가 있습니다.

blockAddress

저널에서의 블록 위치입니다. 주소는 strandId 및 sequenceNo라는 두 개의 필드로 구성된 Amazon Ion 구조입니다.

예: {strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:14}

transactionId

블록을 커밋한 트랜잭션의 고유 ID입니다.

blockTimestamp

블록이 저널에 커밋된 시점의 타임스탬프입니다.

blockHash

블록을 고유하게 나타내는 256비트 해시 값입니다. 이것은 entriesHash와 previousBlockHash를 연결한 해시입니다.

entriesHash

내부 전용 시스템 항목을 포함하여 블록 내의 모든 항목을 나타내는 해시입니다. 이는 리프 노드가 entriesHashList의 모든 해시로 구성된 Merkle 트리의 루트 해시입니다.

previousBlockHash

저널에 있는 이전 체인 블록의 해시입니다.

entriesHashList

블록 내 각 항목을 나타내는 해시 목록입니다. 이 목록에는 다음 항목 해시가 포함될 수 있습니다.

- `transactionInfo`를 나타내는 Ion 해시입니다. 이 값은 전체 `transactionInfo` 구조의 Ion 해시를 가져와 계산됩니다.
- 리프 노드가 `revisions`의 모든 해시로 구성된 Merkle 트리의 루트 해시입니다.
- `redactionInfo`를 나타내는 Ion 해시입니다. 이 해시는 수정 트랜잭션으로 커밋된 블록에만 존재합니다. 이 값은 전체 `redactionInfo` 구조의 Ion 해시를 가져와 계산됩니다.
- 내부 전용 시스템 메타데이터를 나타내는 해시입니다. 이러한 해시는 모든 블록에 존재하지 않을 수 있습니다.

`transactionInfo`

블록을 커밋한 트랜잭션의 문에 대한 정보가 포함된 Amazon Ion 구조입니다. 이 구조에는 다음 필드가 있습니다.

- `statements` - PartiQL 문 목록 및 실행 시작 시점의 `startTime`입니다. 각 명령문에는 `transactionInfo` 구조의 해시를 계산하는 데 필요한 `statementDigest` 해시가 있습니다.
- `documents` - 명령문에 의해 업데이트된 문서 ID입니다. 각 문서에는 해당 문서가 속한 `tableName` 및 `tableId`와 이를 업데이트한 각 문의 인덱스가 포함됩니다.

`revisions`

블록에서 커밋된 문서 개정 목록입니다. 각 개정 구조에는 개정의 커밋된 보기에 있는 모든 필드가 포함됩니다.

여기에는 저널의 전체 해시 체인의 일부인 내부 전용 시스템 개정을 나타내는 해시도 포함될 수 있습니다.

수정된 개정

Amazon QLDB에서 `DELETE` 문은 문서를 삭제된 것으로 표시하는 새 개정본을 생성하여 논리적으로만 문서를 삭제합니다. QLDB는 테이블 기록에서 비활성 문서 개정을 영구적으로 삭제할 수 있는 데이터 수정 작업도 지원합니다.

수정 작업은 지정된 수정 버전의 사용자 데이터만 삭제하고 저널 시퀀스와 문서 메타데이터는 변경되지 않습니다. 이렇게 하면 원장의 전체 데이터 무결성이 유지됩니다. 수정 작업에 대한 자세한 내용과 예제는 문서 개정본 수정하기을 참조하십시오.

수정된 개정 예제

이전 블록 예제를 생각해 보십시오. 이 블록에서 문서 ID가 `HwVFkn8IMRa0xjze5xcgga`이고 버전 번호가 `0`인 개정을 수정한다고 가정해 보겠습니다.

수정이 완료되면 개정의 사용자 데이터(data 구조로 표시됨)가 새 dataHash 필드로 교체됩니다. 이 필드의 값은 제거된 data 구조의 Ion 해시입니다. 따라서 원장은 전반적인 데이터 무결성을 유지하고 기존 검증 API 작업을 통해 암호화 방식으로 검증 가능한 상태를 유지합니다.

다음 개정 예제는 새 dataHash 필드가 *### ####*로 강조 표시된 이 수정 결과를 보여줍니다.

> **Note**
>
> 이 개정 예제는 정보 제공 목적으로만 제공됩니다. 표시된 해시는 실제 계산된 해시 값이 아닙니다.

```
...
{
  blockAddress:{
    strandId:"4o5UuzWSW5PIoOGm5jPA6J",
    sequenceNo:25
  },
  hash:{{6TTHbcfIVdWoFC/j90BOZi0JdHzhjSXo1tW+uHd6Dj4=}},
  dataHash:{{s83jd7sfhsdfhksj7hskjdfjfpIPP/DP2hvionas2d4=}},
  metadata:{
    id:"HwVFkn8IMRa0xjze5xcgga",
    version:0,
    txTime:2022-06-08T18:46:46.492Z,
    txId:"3gtB8Q8dfIMA8lQ5pzHAMo"
  }
}
...
```

또한 QLDB는 완료된 수정 요청에 대해 저널에 새 블록을 추가합니다. 이 블록에는 다음 예제와 같이 트랜잭션에서 수정된 개정 목록이 포함된 추가 redactionInfo 항목이 포함되어 있습니다.

```
...
redactionInfo:{
  revisions:[
    {
      blockAddress:{
        strandId:"4o5UuzWSW5PIoOGm5jPA6J",
        sequenceNo:25
      },
      tableId:"HQZ6cgIMUi204Lq1tT4oaJ",
```

```
      documentId:"HwVFkn8IMRa0xjze5xcgga",
      version:0
    }
  ]
}
...
```

샘플 애플리케이션

내보낸 데이터를 사용하여 저널의 해시 체인을 검증하는 Java 코드 예제는 GitHub 리포지토리 aws-samples/amazon-qldb-dmv-sample-java를 참조하십시오. 이 샘플 애플리케이션에는 다음과 같은 클래스 파일이 포함되어 있습니다.

- ValidateQldbHashChain.java — 원장에서 저널 블록을 내보내고 내보낸 데이터를 사용하여 블록 간 해시 체인을 검증하는 자습서 코드를 포함합니다.
- JournalBlock.java — 블록 내의 각 개별 해시 구성 요소를 계산하는 방법을 보여주는 `verifyBlockHash()`라는 메서드가 들어 있습니다. 이 메서드는 `ValidateQldbHashChain.java`의 자습서 코드에서 호출됩니다.

이 전체 샘플 애플리케이션을 다운로드하고 설치하는 방법에 대한 지침은 Amazon QLDB Java 샘플 애플리케이션 설치을 참조하십시오. 자습서 코드를 실행하기 전에 Java 자습서의 1~3단계에 따라 샘플 원장을 설정하고 샘플 데이터를 로드해야 합니다.

다음 사항도 참조하십시오.

QLDB의 원장에 대한 자세한 내용은 다음 주제를 참조하십시오.

- Amazon QLDB에서 저널 데이터 내보내기 – 저널 데이터를 Amazon Simple Storage Service(S3)로 내보내는 방법을 설명합니다.
- Amazon QLDB에서 저널 데이터 스트리밍 – 저널 데이터를 Amazon Kinesis Data Streams에 스트리밍하는 방법을 알아봅니다.
- Amazon QLDB에서의 데이터 확인 – 저널 데이터의 암호화 검증에 대해 알아봅니다.

Amazon QLDB 용어집

다음은 Amazon QLDB를 사용할 때 접할 수 있는 주요 용어에 대한 정의입니다.

블록 | 다이제스트 | 문서 | 문서 ID | 문서 개정 | 항목 | 필드 | 인덱스 | 인덱싱된 스토리지 | 저널 | 저널 블록 | 저널 스토리지 | 저널 스트랜드 | 저널 팁 | 원장 | 증명 | 개정 | 세션 | 스트랜드 | 테이블 | 테이블 보기 | 보기

블록

트랜잭션에서 저널에 커밋되는 객체입니다. 단일 트랜잭션은 저널에 하나의 블록을 기록하므로 블록은 하나의 트랜잭션에만 연결될 수 있습니다. 블록에는 트랜잭션에서 커밋된 문서 개정을 나타내는 항목과 이를 커밋한 PartiQL 문이 들어 있습니다.

각 블록에는 검증을 위한 해시 값도 있습니다. 블록 해시는 해당 블록 내의 항목 해시를 이전 체인 블록의 해시와 결합하여 계산됩니다.

다이제스트

특정 시점을 기준으로 원장의 전체 문서 개정 내역을 고유하게 나타내는 256비트 해시 값입니다. 다이제스트 해시는 해당 시점에 저널에서 가장 최근에 커밋된 블록을 기준으로 저널의 전체 해시 체인에서 계산됩니다.

QLDB를 사용하면 다이제스트를 보안 출력 파일로 생성할 수 있습니다. 그런 다음 해당 출력 파일을 사용하여 해당 해시와 관련된 문서 개정의 무결성을 확인할 수 있습니다.

문서

테이블에 삽입, 업데이트 및 삭제할 수 있는 Amazon Ion `struct` 형식의 데이터 세트입니다. QLDB 문서에는 구조화, 반구조화, 중첩 및 스키마 없는 데이터가 포함될 수 있습니다.

문서 ID

QLDB가 테이블에 삽입하는 각 문서에 할당하는 UID(Universally Unique Identifier)입니다. 이 ID는 Base62로 인코딩된 영숫자 문자열로 표시되는 128비트 숫자이며 고정 길이는 22자입니다.

문서 개정

고유한 문서 ID로 식별되는 문서 시퀀스의 단일 버전을 나타내는 Ion 구조입니다. 개정에는 사용자 데이터(즉, 테이블에 작성한 데이터)와 시스템에서 생성한 메타데이터가 모두 포함됩니다. 각 개정은 테이블과 연결되며 문서 ID와 제로 기반 버전 번호의 조합으로 고유하게 식별됩니다.

항목

블록에 포함된 객체입니다. 항목은 트랜잭션에서 삽입, 업데이트 및 삭제된 문서 개정과 해당 항목을 커밋한 PartiQL 문을 나타냅니다.

각 항목에는 검증을 위한 해시 값도 있습니다. 항목 해시는 수정 해시 또는 해당 항목 내의 문 해시를 기반으로 계산됩니다.

필드

QLDB 문서의 각 속성을 구성하는 이름-값 쌍입니다. 이름은 기호 토큰이며 값은 무제한입니다.

인덱스

데이터 검색 작업의 성능을 최적화하기 위해 테이블에 만들 수 있는 데이터 구조입니다. QLDB의 인덱스에 대한 자세한 내용은 Amazon QLDB PartiQL 참조의 CREATE INDEX를 참조하십시오.

인덱싱된 스토리지

원장의 테이블, 인덱스, 인덱싱된 기록에 사용되는 디스크 공간입니다. 인덱싱된 스토리지는 고성능 쿼리에 최적화된 원장 데이터로 구성됩니다.

저널

원장에 커밋된 모든 블록의 해시 체인 집합입니다. 저널은 추가만 가능하며 원장 데이터의 모든 변경 사항에 대한 완전하고 변경 불가능한 기록을 나타냅니다.

저널 블록

블록을 참조하십시오.

저널 스토리지

원장의 저널이 사용하는 디스크 공간입니다.

저널 스트랜드

스트랜드을 참조하십시오.

저널 팁

특정 시점의 저널에서 최근에 커밋된 블록입니다.

원장

Amazon QLDB 원장 데이터베이스 리소스의 인스턴스입니다. 이는 QLDB의 기본 AWS 리소스 유형입니다. 원장은 저널 스토리지와 인덱싱된 스토리지로 구성됩니다. 원장 데이터가 저널에 커밋된 후에는 Amazon Ion 문서 개정 테이블에서 쿼리할 수 있습니다.

증명

지정된 다이제스트 및 문서 개정에 대해 QLDB가 반환하는 256비트 해시 값의 정렬된 목록입니다. 이는 지정된 수정 해시를 다이제스트 해시에 연결하기 위해 Merkle 트리 모델에서 필요한 해시로 구성됩니다. 증명을 사용하여 다이제스트와 관련된 개정의 무결성을 검증합니다. 자세한 내용은 Amazon QLDB에서의 데이터 확인을 참조하십시오.

개정

문서 개정을 참조하십시오.

세션

원장에 대한 데이터 트랜잭션 요청 및 응답에 대한 정보를 관리하는 객체입니다. 활성 세션(트랜잭션을 실행 중인 세션)은 원장과의 단일 연결을 나타냅니다. QLDB는 세션당 하나의 활성 실행 트랜잭션을 지원합니다.

스트랜드

저널의 파티션입니다. QLDB는 현재 단일 스트랜드가 포함된 저널만 지원합니다.

테이블

원장 저널에 커밋된 정렬되지 않은 문서 개정 모음을 구체화한 뷰입니다.

테이블 보기

저널에 커밋된 트랜잭션을 기반으로 하는 테이블 내 데이터의 쿼리 가능한 하위 집합입니다. PartiQL 문에서 보기는 테이블 이름의 접두사 한정자(`_ql_`로 시작)로 표시됩니다.

`SELECT` 문을 사용하여 다음과 같은 시스템 정의 보기를 쿼리할 수 있습니다.

- 사용자 - 테이블에 기록한 데이터의 최신 활성 개정 (즉, 사용자 데이터의 현재 상태)입니다. 이는 QLDB의 기본 뷰입니다.
- 커밋됨 - 사용자 데이터와 시스템에서 생성한 메타데이터의 최신 활성 개정입니다. 이는 사용자 테이블에 직접 해당하는 전체 시스템 정의 테이블입니다. 예: `_ql_committed_`*`TableName`*.

보기

테이블 보기을 참조하십시오.

Amazon QLDB 액세스

,AWS CLI() 또는 QLDB API를 사용하여 AWS Management Console아마존 AWS Command Line Interface QLDB에 액세스할 수 있습니다. 다음 섹션에서는 이러한 옵션을 사용하는 방법과 이를 사용하기 위한 필수 조건을 설명합니다.

필수 조건

QLDB에 액세스하려면 먼저 를 설정해야 합니다 (아직 AWS 계정 설정하지 않았다면).

주제

- 가입하십시오. AWS 계정
- 관리자 액세스 권한이 있는 사용자 생성
- IAM에서 QLDB 권한 관리
- 프로그래밍 방식 액세스 권한 부여(선택 사항)

가입하십시오. AWS 계정

계정이 없는 경우 다음 단계를 완료하여 계정을 만드세요. AWS 계정

가입하려면 AWS 계정

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.

   에 AWS 계정가입하면 AWS 계정 루트 사용자a가 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.

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

관리자 액세스 권한이 있는 사용자 생성

등록한 AWS 계정후에는 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 보호하고 AWS IAM Identity Center활성화하고 생성하십시오 AWS 계정 루트 사용자.

보안을 유지하세요. AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 AWS Management Console소유자로 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

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

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자 (콘솔) 에 대한 가상 MFA 디바이스 활성화를 참조하십시오.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

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

   를 ID 소스로 사용하는 방법에 대한 자습서는 사용 설명서의 기본값으로 IAM Identity Center 디렉터리사용자 액세스 구성을 참조하십시오. IAM Identity Center 디렉터리 AWS IAM Identity Center

관리 액세스 권한이 있는 사용자로 로그인

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

  IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 사용 설명서의 AWS 액세스 포털에 로그인을 참조하십시오.AWS 로그인

추가 사용자에게 액세스 권한 할당

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

지침은AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.

2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

IAM에서 QLDB 권한 관리

AWS Identity and Access Management (IAM) 을 사용하여 사용자의 QLDB 권한을 관리하는 방법에 대한 자세한 내용은 을 참조하십시오. Amazon QLDB에서 IAM을 사용하는 방법

프로그래밍 방식 액세스 권한 부여(선택 사항)

사용자가 외부 사용자와 AWS 상호 작용하려는 경우 프로그래밍 방식의 액세스가 필요합니다. AWS Management Console프로그래밍 방식의 액세스 권한을 부여하는 방법은 액세스하는 사용자 유형에 따라 다릅니다. AWS

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 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 요청에 서명할 수 있습니다. AWS	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • AWS CLI 에 대한 내용은 사용 설명서의 AWS CLI사용을 AWS IAM Identity Center위한 구성을 참조하십시오.AWS Command Line Interface • AWS SDK, 도구 및 AWS API의 경우 AWS SDK 및 도구 참조 안내서의 IAM ID 센터 인증을 참조하십시오.
IAM	임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API	IAM 사용 설명서의 AWS 리소스와 함께 임시 자격 증명 사용의 지침을 따르십시오.

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
	에 대한 프로그래밍 방식 요청에 서명할 수 있습니다. AWS	
IAM	(권장되지 않음) 장기 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 요청에 서명할 수 있습니다. AWS	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 에 대한 내용은 사용 설명서의 IAM 사용자 자격 증명을 사용한 인증을 참조하십시오. AWS CLIAWS Command Line Interface • AWS SDK 및 도구의 경우 SDK 및 도구 참조 안내서의 장기 자격 증명을 사용한 인증을 참조하십시오.AWS • AWS API의 경우 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하십시오.

Amazon QLDB에 액세스하는 방법

설정을 위한 사전 요구 사항을 완료한 AWS 계정후 다음 항목을 참조하여 QLDB에 액세스하는 방법에 대해 자세히 알아보십시오.

- 콘솔 사용
- 사용 AWS CLI (관리 API만 해당)
- Amazon QLDB 쉘 사용(데이터 API만 해당)
- API 사용

콘솔을 사용하여 Amazon QLDB에 액세스

https://console.aws.amazon.com/qldb 에서 아마존 AWS Management Console QLDB용 페이지에 액세스할 수 있습니다.

콘솔을 사용하여 QLDB에서 다음을 수행할 수 있습니다.

- 원장을 생성, 삭제, 설명하고 목록을 작성합니다
- PartiQL 편집기를 사용하여 PartiQL 문을 실행합니다.
- QLDB 리소스의 태그를 관리합니다.
- 저널 데이터를 암호로 검증합니다.
- 저널 블록을 내보내거나 스트리밍할 수 있습니다.

Amazon QLDB 원장을 생성하고 이를 샘플 애플리케이션 데이터로 설정하는 방법을 알아보려면 Amazon QLDB 콘솔 시작하기 섹션을 참조하세요.

PartiQL 편집기 빠른 참조

Amazon QLDB는 PartiQL의 하위 집합을 쿼리 언어로 지원하고 Amazon Ion은 문서 지향 데이터 형식으로 지원합니다. PartiQL의 QLDB 구현에 대한 전체 안내서와 자세한 내용은 Amazon QLDB PartiQL 참조 섹션을 참조하세요.

다음 항목에서는 QLDB에서 PartiQL을 사용하는 방법에 대한 간략한 참조 개요를 제공합니다.

주제

- QLDB의 PartiQL 빠른 팁
- 명령
- 시스템 정의 뷰
- 기본 구문 규칙
- PartiQL 편집기 키보드 바로 가기

QLDB의 PartiQL 빠른 팁

다음은 QLDB에서 PartiQL을 사용하기 위한 팁과 모범 사례를 간략하게 요약한 것입니다.

- 동시성 및 트랜잭션 제한 이해 - `SELECT` 쿼리를 포함한 모든 명령문은 30초의 트랜잭션 시간 제한을 포함하여 OCC(Optimistic Concurrency Control) 충돌 및 트랜잭션 제한의 영향을 받습니다.
- 인덱스 사용 – 카디널리티가 높은 인덱스를 사용하고 대상 쿼리를 실행하여 명령문을 최적화하고 전체 테이블 스캔을 방지합니다. 자세한 내용은 쿼리 성능 최적화 섹션을 참조하세요.
- 동등 조건자 사용 – 인덱싱된 조회에는 동등 연산자(= 또는 `IN`)가 필요합니다. 부등 연산자(<, >, `LIKE`, `BETWEEN`)는 인덱싱된 조회에 적합하지 않으며 테이블 전체를 스캔해야 합니다.

- 내부 조인만 사용 — QLDB는 내부 조인만 지원합니다. 가장 좋은 방법은 조인하려는 각 테이블에 대해 인덱싱된 필드를 조인하는 것입니다. 조인 기준과 동등 조건자 모두에 대해 높은 카디널리티 인덱스를 선택하세요.

명령

QLDB에서는 다음 PartiQL 명령을 지원합니다.

데이터 정의 언어(DDL)

Command	설명
CREATE INDEX	테이블의 최상위 문서 필드에 대한 인덱스를 생성합니다.
CREATE TABLE	테이블을 생성합니다.
DROP INDEX	테이블에서 인덱스를 삭제합니다.
DROP TABLE	기존 테이블을 비활성화합니다.
UNDROP TABLE	비활성 테이블을 다시 활성화합니다.

데이터 조작 언어(DML)

Command	설명
DELETE	문서의 최종 수정본을 새로 만들어 활성 문서를 삭제된 것으로 표시합니다.
FROM (INSERT, REMOVE, 또는 SET)	의미상 UPDATE과 같습니다.
INSERT	테이블에 하나 이상의 문서를 추가합니다.
SELECT	하나 이상의 테이블에서 데이터를 가져옵니다.
UPDATE	문서 내의 특정 요소를 업데이트, 삽입 또는 제거합니다.

DML 문 예제

INSERT

```
INSERT INTO VehicleRegistration VALUE
{
    'VIN' : 'KM8SRDHF6EU074761', --string
    'RegNum' : 1722, --integer
    'PendingPenaltyTicketAmount' : 130.75, --decimal
    'Owners' : { --nested struct
        'PrimaryOwner' : { 'PersonId': '294jJ3YUoH1IEEm8GSabOs' },
        'SecondaryOwners' : [ --list of structs
            { 'PersonId' : '1nmeDdLo3AhGswBtyM1eYh' },
            { 'PersonId': 'IN7MvYtUjkp1GMZu0F6CG9' }
        ]
    },
    'ValidToDate' : `2020-06-25T` --Ion timestamp literal with day precision
}
```

UPDATE-INSERT

```
UPDATE Vehicle AS v
INSERT INTO v VALUE 26500 AT 'Mileage'
WHERE v.VIN = '1N4AL11D75C109151'
```

UPDATE-REMOVE

```
UPDATE Person AS p
REMOVE p.Address
WHERE p.GovId = '111-22-3333'
```

SELECT – 상관 하위 쿼리

```
SELECT r.VIN, o.SecondaryOwners
FROM VehicleRegistration AS r, @r.Owners AS o
WHERE r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

SELECT – 내부 조인

```
SELECT v.Make, v.Model, r.Owners
```

```
FROM VehicleRegistration AS r INNER JOIN Vehicle AS v
ON r.VIN = v.VIN
WHERE r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

SELECT – BY 절을 사용하여 문서 ID 가져오기

```
SELECT r_id FROM VehicleRegistration AS r BY r_id
WHERE r.VIN = '1HVBBAANXWH544237'
```

시스템 정의 뷰

QLDB에서는 다음과 같은 시스템 정의 테이블 뷰를 지원합니다.

뷰	설명
table_name	사용자 데이터의 현재 상태만 포함하는 테이블의 기본 사용자 뷰입니다.
_ql_committed_ *table_name*	사용자 데이터와 시스템에서 생성한 메타데이터(예: 문서 ID)의 현재 상태를 모두 포함하는 테이블의 전체 시스템 정의 커밋된 보기입니다.
history(*table_name*)	테이블의 전체 개정 기록을 반환하는 내장 기록 함수입니다.

기본 구문 규칙

QLDB는 PartiQL에 대해 다음과 같은 기본 구문 규칙을 지원합니다.

문자	설명
'	작은따옴표는 문자열 값 또는 Amazon Ion 구조의 필드 이름을 나타냅니다.
"	큰따옴표는 따옴표로 묶인 식별자를 나타냅니다(예: 테이블 이름으로 사용되는 예약어).
`	백틱은 Ion 리터럴 값을 나타냅니다.
.	점 표기법은 상위 구조의 필드 이름에 액세스합니다.

문자	설명
[ ]	대괄호는 Ion `list`을 정의하거나 기존 목록의 0부터 시작하는 서수를 나타냅니다.
{ }	중괄호는 Ion `struct`을 정의합니다.
<< >>	이중 꺾쇠 괄호는 순서가 지정되지 않은 컬렉션인 PartiQL 백을 정의합니다. 백을 사용하여 여러 문서를 테이블에 삽입합니다.
대소문자 구분	필드 이름과 테이블 이름을 포함한 모든 QLDB 시스템 개체 이름은 대소문자를 구분합니다.

PartiQL 편집기 키보드 바로 가기

QLDB 콘솔의 PartiQL 편집기는 다음 키보드 단축키를 지원합니다.

Action	macOS	Windows
Run	Cmd+Return	Ctrl+Enter
설명	Cmd+/	Ctrl+/
Clear	Cmd+Shift+Delete	Ctrl+Shift+Delete

를 사용하여 Amazon QLDB에 액세스 (관리 AWS CLI API만 해당)

AWS Command Line Interface (AWS CLI) 를 사용하여 AWS 서비스 명령줄에서 여러 개를 제어하고 스크립트를 통해 이를 자동화할 수 있습니다. 필요에 따라 일회성 AWS CLI 작업에 를 사용할 수 있습니다. 또한 이를 사용하여 유틸리티 스크립트 내에 Amazon QLDB 작업을 포함할 수도 있습니다.

CLI 액세스를 위해서는 액세스 키 ID 및 비밀 액세스 키가 필요합니다. 가능하다면 장기 액세스 키 대신 임시 보안 인증을 사용합니다. 임시 보안 인증도 액세스 키 ID와 비밀 액세스 키로 구성되지만 보안 인증이 만료되는 시간을 나타내는 보안 토큰이 포함되어 있습니다. 자세한 내용은 IAM 사용 설명서의 AWS 리소스와 함께 임시 자격 증명 사용을 참조하십시오.

AWS CLI의 QLDB에 사용할 수 있는 모든 명령의 전체 목록을 보려면 AWS CLI 명령 참조를 참조하세요.

> **Note**
>
> 에 나열된 `qldb` 관리 API AWS CLI 작업만 지원합니다. Amazon QLDB API 참조 이 API는 원장 리소스 관리 및 비트랜잭션 데이터 작업에만 사용됩니다.
> 명령줄 인터페이스를 사용하여 `qldb-session` API로 데이터 트랜잭션을 실행하려면 QLDB 쉘을 사용하여 Amazon QLDB에 액세스(데이터 API만 해당)을 참조하세요.

주제

- AWS CLI설치 및 구성
- QLDB와 AWS CLI 함께 사용하기

AWS CLI설치 및 구성

리눅스, macOS 또는 윈도우에서 AWS CLI 실행됩니다. 이를 설치 및 구성하려면 AWS Command Line Interface 사용 설명서에서 다음 지침을 참조하세요.

1. 최신 버전의 설치 또는 업데이트 AWS CLI
2. 빠른 설정

QLDB와 AWS CLI 함께 사용하기

명령줄 형식은 Amazon QLDB 작업 이름과 해당 작업에 대한 파라미터 순으로 구성됩니다. 는 JSON 외에도 매개 변수 값에 대한 약식 구문을 AWS CLI 지원합니다.

`help`를 사용하여 QLDB에서 사용 가능한 모든 명령을 나열합니다.

```
aws qldb help
```

`help`를 사용하여 특정 명령을 설명하고 그 사용법에 대해 자세히 알아볼 수도 있습니다.

```
aws qldb create-ledger help
```

예를 들어, 원장을 생성하려면:

```
aws qldb create-ledger --name my-example-ledger --permissions-mode STANDARD
```

QLDB 쉘을 사용하여 Amazon QLDB에 액세스(데이터 API만 해당)

Amazon QLDB는 트랜잭션 데이터 API와의 상호 작용을 위한 명령행 쉘을 제공합니다. QLDB 쉘을 사용하면 원장 데이터에 대해 PartiQL 문을 실행할 수 있습니다.

이 쉘의 최신 버전은 Rust에서 작성되었으며 기본 `main` 브랜치에 있는 GitHub 리포지토리 awslabs/amazon-qldb-shell에서 오픈 소스로 제공됩니다. Python 버전(v1)도 `master` 브랜치의 동일한 리포지토리에서 계속 사용할 수 있습니다.

> **Note**
>
> Amazon QLDB 쉘은 `qldb-session` 트랜잭션 데이터 API만 지원합니다. 이 API는 QLDB 원장에서 PartiQL 문을 실행하는 데만 사용됩니다.
> 명령행 인터페이스를 사용하여 `qldb` 관리 API 작업과 상호 작용하려면 를 사용하여 Amazon QLDB에 액세스 (관리 AWS CLI API만 해당) 섹션을 참조하세요.

이 도구는 애플리케이션에 통합하거나 프로덕션 목적으로 채택하기 위한 것이 아닙니다. 이 도구의 목적은 QLDB 및 PartiQL을 빠르게 실험해 볼 수 있도록 하는 것입니다.

다음 섹션에서는 QLDB shell 작업을 시작하는 방법을 설명합니다.

주제

필수 조건

QLDB 쉘을 시작하기 전에 다음을 수행해야 합니다.

1. Amazon QLDB 액세스의 AWS 설정 지침을 따르십시오. 다음 내용이 포함됩니다:

   1. AWS에 가입합니다.
   2. 적절한 QLDB 권한을 가진 사용자를 생성합니다.
   3. 개발을 위한 프로그래밍 방식 액세스 권한을 부여합니다.
2. AWS 보안 인증 정보와 기본 AWS 리전을 설정합니다. 자세한 설명은 AWS Command Line Interface 사용자 가이드에서 구성 기본 사항을 참조하세요.

   사용 가능한 리전의 전체 목록은 AWS 일반 참조에서 Amazon QLDB 엔드포인트 및 할당량을 참조하십시오.
3. STANDARD 권한 모드의 모든 원장에 대해 적절한 표에서 PartiQL 문을 실행할 수 있는 권한을 부여하는 IAM 정책을 생성합니다. 이러한 정책을 생성하는 방법을 알아보려면 Amazon QLDB에서 표준 권한 모드로 시작하기를 참조하십시오.

쉘 설치

QLDB 쉘의 최신 버전을 설치하려면 GitHub의 README.md 파일을 참조하세요. QLDB는 GitHub 리포지토리의 릴리스 섹션에서 Linux, macOS, Windows용으로 사전 빌드된 이진수 파일을 제공합니다.

macOS의 경우 쉘은 `aws/tap` Homebrew 탭과 통합됩니다. Homebrew를 사용하여 macOS에 쉘을 설치하려면 다음 명령을 실행합니다.

```
$ xcode-select --install # Required to use Homebrew
$ brew tap aws/tap # Add AWS as a Homebrew tap
$ brew install qldbshell
```

구성

설치 후 쉘은 초기화 중에 `$XDG_CONFIG_HOME/qldbshell/config.ion`에 있는 기본 구성 파일을 로드합니다. Linux 및 macOS의 경우 이 파일은 일반적으로 `~/.config/qldbshell/config.ion`에 있습니다. 이러한 파일이 없는 경우 쉘은 기본 설정으로 실행됩니다.

설치 후 수동으로 `config.ion` 파일을 만들 수 있습니다. 이 구성 파일은 Amazon Ion 데이터 형식을 사용합니다. 다음은 최소 `config.ion` 파일의 예입니다.

```
{
  default_ledger: "my-example-ledger"
}
```

`default_ledger`가 구성 파일에 설정되어 있지 않은 경우, 쉘을 호출할 때 `--ledger` 파라미터가 필요합니다. 구성 옵션의 전체 목록은 GitHub의 README.md 파일을 참조하세요.

쉘 호출

명령행 터미널에서 특정 원장에 대해 QLDB 쉘을 호출하려면 다음 명령을 실행합니다. *my-example-ledger*를 원장 명칭을 바꿉니다.

```
$ qldb --ledger my-example-ledger
```

이 명령은 기본 AWS 리전에 연결됩니다. 지역을 명시적으로 지정하려면 다음 섹션에 설명된 대로 `--region` 또는 `--qldb-session-endpoint` 파라미터를 사용하여 명령을 실행할 수 있습니다.

`qldb` 쉘 세션을 호출한 후 다음과 같은 입력 타입을 입력할 수 있습니다.

- 쉘 명령
- 개별 트랜잭션의 단일 PartiQL 명령문
- 트랜잭션 내의 여러 PartiQL 명령문

쉘 파라미터

쉘을 호출하는 데 사용할 수 있는 플래그 및 옵션의 전체 목록을 보려면 다음과 같이 `--help` 플래그와 함께 `qldb` 명령을 실행합니다.

```
$ qldb --help
```

다음은 `qldb` 명령의 몇 가지 주요 플래그와 옵션입니다. 이러한 선택적 파라미터를 추가하여 AWS 리전, 자격 증명 프로필, 엔드포인트, 결과 형식 및 기타 구성 옵션을 재정의할 수 있습니다.

사용량

```
$ qldb [FLAGS] [OPTIONS]
```

플래그

`-h`, `--help`

도움말 정보를 인쇄합니다.

-v, **--verbose**

로깅 세부 사항을 구성합니다. 기본적으로 쉘은 오류만 기록합니다. 세부 사항 수준을 높이려면 이 인수(예:-vv)를 반복합니다. 가장 높은 수준은 trace 세부 수준에 해당하는 -vvv입니다.

-V, **--version**

버전 정보를 인쇄합니다.

OPTIONS

-l, **--ledger** ***LEDGER_NAME***

연결할 원장의 명칭입니다. default_ledger가 config.ion 파일에 설정되지 않은 경우 필요한 쉘 파라미터입니다. 이 파일에서 지역과 같은 추가 옵션을 설정할 수 있습니다.

-c, **--config** ***CONFIG_FILE***

모든 쉘 구성 옵션을 정의할 수 있는 파일입니다. 형식 지정 세부 정보 및 구성 옵션의 전체 목록은 GitHub의 README.md 파일을 참조하세요.

-f, **--format ion|table**

쿼리 결과의 출력 형식입니다. 기본값은 ion입니다.

-p, **--profile** ***###***

인증에 사용할 AWS 자격 증명 프로필의 위치입니다.

제공하지 않을 경우 쉘은 ~/.aws/credentials에 있는 기본 AWS 프로필을 사용합니다.

-r, **--region** ***REGION_CODE***

연결할 QLDB 원장의 AWS 리전 코드입니다. 예: us-east-1.

제공되지 않은 경우 쉘은 AWS 프로필에 지정된 기본 AWS 리전에 연결됩니다.

-s, **--qldb-session-endpoint** ***QLDB_SESSION_ENDPOINT***

연결할 qldb-session API 엔드포인트.

사용 가능한 QLDB 지역 및 엔드포인트의 전체 목록은 AWS 일반 참조에서 Amazon QLDB 엔드포인트 및 할당량을 참조하십시오.

명령 참조

`qldb` 세션을 호출한 후 쉘은 다음 키와 데이터베이스 명령을 지원합니다.

쉘 키

키	함수 설명
Enter	문을 실행합니다.
Escape+Enter (macOS, Linux) Shift+Enter (Windows)	새 행을 시작하여 여러 행에 걸친 문을 입력합니다. 입력 텍스트를 여러 행로 복사하여 쉘에 붙여넣을 수도 있습니다. macOS에서 Meta 키로 Escape 대신 Option을 설정하는 방법에 대한 지침은 OS X Daily 사이트를 참조하세요.
Ctrl+C	현재 명령을 취소합니다.
Ctrl+D	파일 끝(EOF)에 신호를 보내고 쉘의 현재 수준을 종료합니다. 활성 트랜잭션에 없는 경우 쉘을 종료합니다. 활성 트랜잭션에서 트랜잭션을 중단합니다.

쉘 데이터베이스 명령

명령	함수 설명
`help`	도움말 정보를 표시합니다.
`begin`	트랜잭션을 시작합니다.
`start transaction`	
`commit`	트랜잭션을 원장 일지에 체결합니다.
`abort`	트랜잭션을 중지하고 변경한 내용을 모두 거부합니다.

명령	함수 설명
`exit`	쉘을 종료합니다.
`quit`	

> **Note**
>
> 모든 QLDB 쉘 명령은 대소문자를 구분하지 않습니다.

개별 명령문 실행

README.md에 나열된 데이터베이스 명령과 쉘 메타 명령을 제외하고 쉘은 사용자가 입력하는 각 명령을 별도의 PartiQL 문으로 해석합니다. 기본적으로 쉘은 `auto-commit` 모드를 활성화합니다. 이 모드는 구성할 수 있습니다.

`auto-commit` 모드에서는 쉘이 각 명령문을 자체 트랜잭션에서 암시적으로 실행하고 오류가 발견되지 않으면 트랜잭션을 자동으로 체결합니다. 즉, 문을 실행할 때마다 `start transaction`(또는 `begin`) 및 `commit`을 수동으로 실행하지 않아도 됩니다.

트랜잭션 관리

또는 QLDB 쉘을 사용하여 트랜잭션을 수동으로 제어할 수 있습니다. 명령 및 명령문을 순차적으로 일괄 처리하여 트랜잭션 내에서 여러 명령문을 대화식으로 실행하거나 비대화식으로 실행할 수 있습니다.

대화형 트랜잭션

대화형 트랜잭션을 실행하려면 다음 단계를 수행합니다.

1. 트랜잭션을 시작하려면 `begin` 명령을 입력합니다.

   ```
   qldb> begin
   ```

   트랜잭션을 시작하면 쉘에 다음 명령 프롬프트가 표시됩니다.

   ```
   qldb *>
   ```

2. 그러면 입력한 각 명령문이 동일한 트랜잭션에서 실행됩니다.
   - 예컨대 다음과 같이 단일 명령문을 실행할 수 있습니다.

     ```
     qldb *> SELECT * FROM Vehicle WHERE VIN = '1N4AL11D75C109151'
     ```

     Enter를 누르면 쉘에 명령문의 결과가 표시됩니다.
   - 다음과 같이 여러 개의 명령문이나 명령을 세미콜론(;)구분 기호로 구분하여 입력할 수도 있습니다.

     ```
     qldb *> SELECT * FROM Vehicle WHERE VIN = '1N4AL11D75C109151'; commit
     ```
3. 트랜잭션을 종료하려면 다음 명령 중 하나를 입력합니다.
   - `commit` 명령을 입력하여 트랜잭션을 원장 저널에 체결합니다.

     ```
     qldb *> commit
     ```
   - `abort` 명령을 입력하여 트랜잭션을 중지하고 변경한 내용을 거부합니다.

     ```
     qldb *> abort
     transaction was aborted
     ```

트랜잭션 시간 초과 제한

대화형 트랜잭션은 QLDB의 시간 초과 제한을 준수합니다. 시작 후 30초 이내에 트랜잭션을 체결하지 않으면 QLDB는 트랜잭션을 자동으로 만료시키고 트랜잭션 중에 이루어진 모든 변경 사항을 거부합니다.

그러면 쉘은 명령문 결과를 표시하는 대신 만료 오류 메시지를 표시하고 일반 명령 프롬프트로 돌아갑니다. 다시 시도하려면 `begin` 명령을 다시 입력하여 새 트랜잭션을 시작해야 합니다.

```
transaction failed after 1 attempts, last error: communication failure:
 Transaction 2UMpiJ5hh7WLjVgEiMLOoO has expired
```

비대화형 트랜잭션

다음과 같이 명령과 명령문을 순차적으로 일괄 처리하여 여러 명령문으로 전체 트랜잭션을 실행할 수 있습니다.

```
qldb> begin; SELECT * FROM Vehicle WHERE VIN = '1N4AL11D75C109151'; SELECT * FROM
 Person p, DriversLicense l WHERE p.GovId = l.LicenseNumber; commit
```

각 명령과 명령문을 세미콜론(;) 구분 기호로 구분해야 합니다. 트랜잭션의 명령문이 유효하지 않은 경우 쉘은 자동으로 트랜잭션을 거부합니다. 쉘은 사용자가 입력한 후속 명령문을 처리하지 않습니다.

여러 트랜잭션을 설정할 수도 있습니다.

```
qldb> begin; statement1; commit; begin; statement2; statement3; commit
```

이전 예와 마찬가지로, 트랜잭션이 실패하면 쉘은 사용자가 입력한 후속 트랜잭션이나 명령문을 진행하지 않습니다.

트랜잭션을 종료하지 않으면 쉘이 대화형 모드로 전환되고 다음 명령문이나 문을 입력하라는 메시지가 표시됩니다.

```
qldb> begin; statement1; commit; begin
qldb *>
```

쉘 종료

현재 `qldb` 쉘 세션을 종료하려면 `exit` 또는 `quit` 명령을 입력하거나 쉘이 트랜잭션에 있지 않은 경우 키보드 단축키 Ctrl+D를 사용합니다.

```
qldb> exit
$
```

```
qldb> quit
$
```

예

QLDB에서 PartiQL 문을 작성하는 방법에 대한 자세한 설명은 Amazon QLDB PartiQL 참조 섹션을 참조하세요.

Example

다음 예에서는 일반적인 기본 명령 순서를 보여줍니다.

> **Note**
>
> QLDB 쉘은 이 예의 각 PartiQL 문을 자체 트랜잭션으로 실행합니다.
> 이 예에서는 원장 `test-ledger`이 이미 존재하고 활성 상태라고 가정합니다.

```
$ qldb --ledger test-ledger --region us-east-1

qldb> CREATE TABLE TestTable
qldb> INSERT INTO TestTable `{"Name": "John Doe"}`
qldb> SELECT * FROM TestTable
qldb> DROP TABLE TestTable
qldb> exit
```

API를 사용하여 Amazon QLDB에 액세스

AWS Management Console 및 AWS Command Line Interface (AWS CLI) 를 사용하여 Amazon QLDB와 대화식으로 작업할 수 있습니다. 하지만 QLDB를 최대한 활용하려면 QLDB 드라이버 또는 AWS SDK로 애플리케이션 코드를 작성하여 API를 사용하여 원장과 상호 작용할 수 있습니다.

드라이버를 사용하면 애플리케이션이 트랜잭션 데이터 API를 사용하여 QLDB와 상호 작용할 수 있습니다. AWS SDK는 QLDB 리소스 관리 API와의 상호 작용을 지원합니다. 이들 API에 대한 자세한 내용은 Amazon QLDB API 참조 섹션을 참조하세요.

이 드라이버는 Java, .NET, Go, Node.js 및 Python에서 QLDB에 대한 지원을 제공합니다. 이러한 언어를 신속하게 사용하려면 Amazon QLDB 드라이버 시작하기을 참조하세요.

애플리케이션에서 QLDB 드라이버 또는 AWS SDK를 사용하려면 먼저 프로그래밍 액세스 권한을 부여해야 합니다. 자세한 내용은 프로그래밍 방식 액세스 권한 부여을(를) 참조하세요.

Amazon QLDB 콘솔 시작하기

이 자습서는 첫 번째 Amazon QLDB 원장을 생성하고 테이블과 샘플 데이터로 채우는 단계를 안내합니다. 이 시나리오에서 생성하는 샘플 원장은 차량 등록에 대한 전체 기록 정보를 추적하는 자동차 부서(DMV) 애플리케이션용 데이터베이스입니다.

자산 기록은 원장 데이터베이스의 유용성을 강조하는 다양한 시나리오와 작업을 포함하기 때문에 QLDB의 일반적인 사용 사례입니다. QLDB를 사용하면 SQL과 유사한 쿼리 기능을 지원하는 문서 지향 데이터베이스에서 데이터에 대한 전체 변경 기록을 직접 액세스하고 쿼리하고 확인할 수 있습니다.

이 자습서를 진행하면서 다음 항목에서는 차량 등록을 추가하고 수정하고 해당 등록에 대한 변경 기록을 보는 방법을 설명합니다. 또한 이 안내서는 등록 문서를 암호화하여 검증하는 방법을 보여주고, 리소스를 정리하고 샘플 원장을 삭제하는 것으로 마무리합니다.

자습서의 각 단계에는 AWS Management Console 사용 지침이 있습니다.

주제

자습서 사전 조건 및 고려 사항

이 Amazon QLDB 자습서를 시작하기 전에 다음 사전 조건을 완료했는지 확인하십시오.

1. 아직 Amazon QLDB 액세스의 AWS 설정 지침을 따르지 않은 경우 이를 수행하십시오. 이러한 단계에는 AWS 가입 및 관리 사용자 생성이 포함됩니다.
2. 권한 설정의 지침에 따라 QLDB 리소스에 대한 IAM 권한을 설정합니다. 이 자습서의 모든 단계를 완료하려면 AWS Management Console을 통해 원장 리소스에 대한 전체 관리자 액세스 권한이 필요합니다.

> **Note**
>
> 이미 전체 AWS 관리자 권한이 있는 사용자로 로그인한 경우 이 단계를 건너뛸 수 있습니다.

3. (선택 사항) QLDB는 AWS Key Management Service(AWS KMS)의 키를 사용하여 저장 데이터를 암호화합니다. 다음 AWS KMS keys 유형 중 하나를 선택할 수 있습니다.
   - AWS 소유 KMS 키 - AWS가 사용자를 대신하여 소유하고 관리하는 KMS 키를 사용합니다. 이는 기본 옵션이며 추가 설정이 필요하지 않습니다.
   - 고객 관리형 KMS 키 – 귀하가 생성, 소유 및 관리하는 계정에서 대칭 암호화 KMS 키를 사용하십시오. QLDB는 비대칭 키는 지원하지 않습니다.

     이 옵션을 사용하려면 KMS 키를 만들거나 계정의 기존 키를 사용해야 합니다. 고객 관리형 키 생성에 대한 지침은 AWS Key Management Service 개발자 가이드에서 대칭 암호화 KMS 키 생성을 참조하세요.

     ID, 별칭 또는 Amazon 리소스 이름(ARN)을 사용하여 고객 관리형 KMS 키를 지정할 수 있습니다. 자세한 설명은 AWS Key Management Service 개발자 가이드에서 키 식별자(KeyId)를 참조하세요.

     > **Note**
     >
     > 교차 리전 키는 지원되지 않습니다. 지정된 KMS 키는 원장과 동일한 AWS 리전에 있어야 합니다.

권한 설정

이 단계에서는 AWS 계정의 모든 QLDB 리소스에 대해 콘솔을 통해 전체 액세스 권한을 설정합니다. 이러한 권한을 신속하게 부여하려면 AWS 관리형 정책 AmazonQLDBConsoleFullAccess를 사용하십시오.

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 따르십시오.

- 자격 증명 공급자를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 서드 파티 자격 증명 공급자의 역할 만들기(페더레이션)의 지침을 따르십시오.

- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 IAM 사용자의 역할 생성의 지침을 따르십시오.
  - (권장되지 않음) 정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르십시오.

> **Important**
>
> 이 자습서의 목적에 따라 모든 QLDB 리소스에 대한 전체 관리 액세스 권한을 자신에게 부여합니다. 그러나 프로덕션 사용 사례의 경우 최소 권한을 부여하거나 작업을 수행하는 데 필요한 권한만 부여하는 보안 모범 사례를 따르십시오. 예를 보려면 Amazon QLDB의 자격 증명 기반 정책 예을 참조하십시오.

`vehicle-registration`라는 이름의 원장을 생성하려면 1단계: 새 원장 생성로 진행하십시오.

1단계: 새 원장 생성

이 단계에서는 `vehicle-registration`라는 이름의 새 Amazon QLDB 원장을 생성합니다. 그런 다음 원장 상태가 활성인지 확인합니다. 원장에 추가한 모든 태그를 확인할 수도 있습니다.

원장을 생성할 때 기본적으로 삭제 방지가 활성화됩니다. 삭제 방지는 사용자가 원장을 삭제하는 것을 방지하는 QLDB의 기능입니다. QLDB API 또는 AWS Command Line Interface(AWS CLI)를 사용하여 원장을 생성할 때 삭제 방지를 비활성화할 수 있습니다.

새 원장을 생성하려면

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/qldb에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 시작하기를 선택합니다.
3. 첫 번째 원장 생성 카드에서 원장 생성을 선택합니다.
4. 원장 생성 페이지에서 다음을 수행합니다.

- 원장 정보 — 원장 이름은 **vehicle-registration**으로 미리 채워져 있어야 합니다.
- 권한 모드 - 원장에 할당할 권한 모드입니다. 다음 옵션 중 하나를 선택합니다.
  - 모두 허용 - 원장에 대한 API 수준 세분화로 액세스 제어를 가능하게 하는 레거시 권한 모드입니다

    이 모드를 사용하면 이 원장에 대한 SendCommand API 권한이 있는 사용자가 지정된 원장의 모든 테이블에서 모든 PartiQL 명령(따라서 ALLOW_ALL)을 실행할 수 있습니다. 이 모드는 원장에 대해 생성하는 모든 테이블 수준 또는 명령 수준 IAM 권한 정책을 무시합니다.
  - 표준 – (권장) 원장, 테이블 및 PartiQL 명령에 대해 보다 세분화된 액세스 제어를 가능하게 하는 권한 모드입니다.. 원장 데이터의 보안을 극대화하기 위해 이 권한 모드를 사용하는 것을 강력 권장합니다.

    기본적으로 이 모드는 이 원장의 모든 테이블에서 PartiQL 명령을 실행하려는 모든 요청을 거부합니다. PartiQL 명령을 허용하려면 원장에 대한 SendCommand API 권한 외에도 특정 테이블 리소스 및 PartiQL 작업에 대한 IAM 권한 정책을 생성해야 합니다. 자세한 내용은 Amazon QLDB에서 표준 권한 모드로 시작하기을 참조하세요.
- 저장 데이터 암호화 — 저장 데이터 암호화에 사용할 AWS Key Management Service(AWS KMS)의 키입니다. 다음 옵션 중 하나를 선택합니다.
  - AWS 소유 KMS 키 사용 – AWS가 사용자를 대신하여 소유하고 관리하는 KMS 키를 사용합니다. 이는 기본 옵션이며 추가 설정이 필요하지 않습니다.
  - 다른 AWS KMS 키 선택 – 사용자가 생성, 소유 및 관리하는 계정의 대칭 암호화 KMS 키를 사용합니다.

    AWS KMS 콘솔을 사용하여 새 키를 생성하려면 AWS KMS 키 생성을 선택합니다. 자세한 내용은 AWS Key Management Service 개발자 안내서의 대칭 암호화 KMS 키 생성을 참조하십시오.

    기존 KMS 키를 사용하려면 드롭다운 목록에서 하나를 선택하거나 KMS 키 ARN을 지정합니다.
- 태그 – (선택 사항) 태그를 키-값 쌍으로 연결하여 메타데이터를 원장에 추가합니다. 원장에 태그를 추가함으로써 쉽게 원장을 조직화하고 식별할 수 있습니다. 자세한 내용은 Amazon QLDB 리소스 태그 지정을 참조하십시오.

  태그 추가를 선택한 다음 키-값 쌍을 적절히 입력합니다.

5. 원하는 대로 설정되었으면 원장 생성을 선택합니다.

> **Note**
>
> QLDB 원장 상태가 활성 상태가 되면 해당 원장에 액세스할 수 있습니다. 몇 분 정도 걸릴 수 있습니다.

6. 원장 목록에서 `vehicle-registration`을 찾아 원장 상태가 활성 상태인지 확인합니다.
7. (선택 사항) `vehicle-registration` 원장 이름을 선택합니다. 차량 등록 원장 세부 정보 페이지에서 원장에 추가한 모든 태그가 태그 카드에 표시되는지 확인합니다. 이 콘솔 페이지를 사용하여 원장 태그를 편집할 수도 있습니다.

`vehicle-registration` 원장에 테이블을 생성하려면 2단계: 원장에 테이블, 인덱스 및 샘플 데이터 생성로 진행하십시오.

2단계: 원장에 테이블, 인덱스 및 샘플 데이터 생성

Amazon QLDB 원장이 활성화되면 차량, 소유자 및 등록 정보에 대한 데이터 테이블 생성을 시작할 수 있습니다. 테이블과 인덱스를 생성한 후 데이터를 로드할 수 있습니다.

이 단계에서는 `vehicle-registration` 원장에 4개의 테이블을 생성합니다.

- `VehicleRegistration`
- `Vehicle`
- `Person`
- `DriversLicense`

또한 다음과 같은 인덱스를 생성합니다.

테이블 이름	필드
`VehicleRegistration`	`VIN`
`VehicleRegistration`	`LicensePlateNumber`
`Vehicle`	`VIN`
`Person`	`GovId`

테이블 이름	필드
DriversLicense	LicensePlateNumber
DriversLicense	PersonId

QLDB 콘솔을 사용하여 인덱스가 있는 이러한 테이블을 자동으로 생성하고 샘플 데이터를 로드할 수 있습니다. 또는 콘솔의 PartiQL 편집기를 사용하여 각 PartiQL 문을 단계별로 수동으로 실행할 수 있습니다.

자동 옵션

테이블, 인덱스 및 샘플 데이터를 생성하려면

1. https://console.aws.amazon.com/qldb에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 시작하기를 선택합니다.
3. 샘플 애플리케이션 데이터 카드의 자동 옵션 아래 원장 목록에서 `vehicle-registration`을 선택합니다.
4. 샘플 데이터 로드를 선택합니다.

   작업이 성공적으로 완료되면 콘솔에 샘플 데이터가 로드되었다는 메시지가 표시됩니다.

   이 스크립트는 단일 트랜잭션에서 모든 명령문을 실행합니다. 트랜잭션의 일부가 실패하면 모든 명령문이 롤백되고 적절한 오류 메시지가 표시됩니다. 문제를 해결한 후 작업을 재시도할 수 있습니다.

   > **Note**
   >
   > - 트랜잭션 실패의 한 가지 가능한 원인은 중복 테이블을 만들려고 시도하는 것입니다. 원장에 `VehicleRegistration`, `Vehicle`, `Person`, `DriversLicense` 등의 테이블 이름이 이미 있는 경우 샘플 데이터 로드 요청이 실패합니다.
   >
   >   대신 이 샘플 데이터를 빈 원장에 로드해 보십시오.
   > - 이 스크립트는 파라미터화된 `INSERT` 문을 실행합니다. 따라서 이러한 PartiQL 문은 리터럴 데이터 대신 바인드 파라미터를 사용하여 저널 블록에 기록됩니다. 예를 들어, 저널 블록에서 다음 명령문을 볼 수 있는데, 여기서 물음표(?)는 문서 콘텐츠의 변수 자리 표시자입니다.

```
INSERT INTO Vehicle ?
```

수동 옵션

`PrimaryOwner` 필드가 비어 있는 `VehicleRegistration`에 문서를 삽입하고, `PersonId` 필드가 비어 있는 `DriversLicense`에 문서를 삽입합니다. 나중에 이 필드를 `Person` 테이블의 시스템 할당 문서 `id`로 채웁니다.

> **Tip**
>
> 가장 좋은 방법은 이 문서 `id` 메타데이터 필드를 외부 키로 사용하는 것입니다. 자세한 내용은 문서 메타데이터 쿼리을 참조하십시오.

테이블, 인덱스 및 샘플 데이터를 생성하려면

1. https://console.aws.amazon.com/qldb에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 PartiQL 편집기를 선택합니다.
3. `vehicle-registration` 원장을 선택합니다.
4. 먼저 테이블 네 개를 생성합니다. QLDB는 개방형 콘텐츠를 지원하며 스키마를 적용하지 않으므로 속성이나 데이터 유형을 지정하지 않습니다.

   쿼리 편집기 창에서 다음 문을 입력한 후 실행을 선택합니다. 명령문을 실행하려면 키보드 단축키(Windows의 경우 Ctrl+Enter, macOS의 경우 Cmd+Return)를 사용할 수도 있습니다. 키보드 단축키에 대한 자세한 내용은 PartiQL 편집기 키보드 바로 가기 섹션을 참조하세요.

   ```
   CREATE TABLE VehicleRegistration
   ```

   다음 각 작업에 대해 이 단계를 반복합니다.

   ```
   CREATE TABLE Vehicle
   ```

   ```
   CREATE TABLE Person
   ```

```
CREATE TABLE DriversLicense
```

5. 그런 다음 각 테이블의 쿼리 성능을 최적화하는 인덱스를 생성합니다.

> **Important**
>
> QLDB는 문서를 효율적으로 조회하기 위한 인덱스가 필요합니다. 인덱스가 없으면 QLDB는 문서를 읽을 때 전체 테이블 스캔을 수행해야 합니다. 이로 인해 동시성 충돌 및 트랜잭션 시간 초과를 포함하여 대규모 테이블에서 성능 문제가 발생할 수 있습니다.
> 인덱싱된 필드 또는 문서 ID(예: = 또는 IN)에서 동등 연산자를 사용하여 WHERE 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 자세한 내용은 쿼리 성능 최적화을 참조하십시오.

쿼리 편집기 창에서 다음 문을 입력한 후 실행을 선택합니다.

```
CREATE INDEX ON VehicleRegistration (VIN)
```

다음에 대해 이 단계를 반복합니다.

```
CREATE INDEX ON VehicleRegistration (LicensePlateNumber)
```

```
CREATE INDEX ON Vehicle (VIN)
```

```
CREATE INDEX ON Person (GovId)
```

```
CREATE INDEX ON DriversLicense (LicensePlateNumber)
```

```
CREATE INDEX ON DriversLicense (PersonId)
```

6. 인덱스를 생성한 후 테이블에 데이터 로드를 시작할 수 있습니다. 이 단계에서는 원장이 추적하는 차량 소유자에 대한 개인 정보가 포함된 문서를 Person 테이블에 삽입합니다.

쿼리 편집기 창에서 다음 문을 입력한 후 실행을 선택합니다.

```
INSERT INTO Person
```

```
<< {
    'FirstName' : 'Raul',
    'LastName' : 'Lewis',
    'DOB' : `1963-08-19T`,
    'GovId' : 'LEWISR261LL',
    'GovIdType' : 'Driver License',
    'Address' : '1719 University Street, Seattle, WA, 98109'
},
{
    'FirstName' : 'Brent',
    'LastName' : 'Logan',
    'DOB' : `1967-07-03T`,
    'GovId' : 'LOGANB486CG',
    'GovIdType' : 'Driver License',
    'Address' : '43 Stockert Hollow Road, Everett, WA, 98203'
},
{
    'FirstName' : 'Alexis',
    'LastName' : 'Pena',
    'DOB' : `1974-02-10T`,
    'GovId' : '744 849 301',
    'GovIdType' : 'SSN',
    'Address' : '4058 Melrose Street, Spokane Valley, WA, 99206'
},
{
    'FirstName' : 'Melvin',
    'LastName' : 'Parker',
    'DOB' : `1976-05-22T`,
    'GovId' : 'P626-168-229-765',
    'GovIdType' : 'Passport',
    'Address' : '4362 Ryder Avenue, Seattle, WA, 98101'
},
{
    'FirstName' : 'Salvatore',
    'LastName' : 'Spencer',
    'DOB' : `1997-11-15T`,
    'GovId' : 'S152-780-97-415-0',
    'GovIdType' : 'Passport',
    'Address' : '4450 Honeysuckle Lane, Seattle, WA, 98101'
} >>
```

7. 그런 다음 각 차량 소유자의 운전 면허 정보가 포함된 문서로 DriversLicense 테이블을 채우십시오.

쿼리 편집기 창에서 다음 문을 입력한 후 실행을 선택합니다.

```
INSERT INTO DriversLicense
<< {
    'LicensePlateNumber' : 'LEWISR261LL',
    'LicenseType' : 'Learner',
    'ValidFromDate' : `2016-12-20T`,
    'ValidToDate' : `2020-11-15T`,
    'PersonId' : ''
},
{
    'LicensePlateNumber' : 'LOGANB486CG',
    'LicenseType' : 'Probationary',
    'ValidFromDate' : `2016-04-06T`,
    'ValidToDate' : `2020-11-15T`,
    'PersonId' : ''
},
{
    'LicensePlateNumber' : '744 849 301',
    'LicenseType' : 'Full',
    'ValidFromDate' : `2017-12-06T`,
    'ValidToDate' : `2022-10-15T`,
    'PersonId' : ''
},
{
    'LicensePlateNumber' : 'P626-168-229-765',
    'LicenseType' : 'Learner',
    'ValidFromDate' : `2017-08-16T`,
    'ValidToDate' : `2021-11-15T`,
    'PersonId' : ''
},
{
    'LicensePlateNumber' : 'S152-780-97-415-0',
    'LicenseType' : 'Probationary',
    'ValidFromDate' : `2015-08-15T`,
    'ValidToDate' : `2021-08-21T`,
    'PersonId' : ''
} >>
```

8. 이제 `VehicleRegistration` 테이블을 차량 등록 문서로 채우십시오. 이러한 문서에는 기본 소유자와 보조 소유자를 저장하는 중첩된 `Owners` 구조가 포함되어 있습니다.

쿼리 편집기 창에서 다음 문을 입력한 후 실행을 선택합니다.

```
INSERT INTO VehicleRegistration
<< {
    'VIN' : '1N4AL11D75C109151',
    'LicensePlateNumber' : 'LEWISR261LL',
    'State' : 'WA',
    'City' : 'Seattle',
    'PendingPenaltyTicketAmount' : 90.25,
    'ValidFromDate' : `2017-08-21T`,
    'ValidToDate' : `2020-05-11T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId': '' },
        'SecondaryOwners' : []
    }
},
{
    'VIN' : 'KM8SRDHF6EU074761',
    'LicensePlateNumber' : 'CA762X',
    'State' : 'WA',
    'City' : 'Kent',
    'PendingPenaltyTicketAmount' : 130.75,
    'ValidFromDate' : `2017-09-14T`,
    'ValidToDate' : `2020-06-25T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId': '' },
        'SecondaryOwners' : []
    }
},
{
    'VIN' : '3HGGK5G53FM761765',
    'LicensePlateNumber' : 'CD820Z',
    'State' : 'WA',
    'City' : 'Everett',
    'PendingPenaltyTicketAmount' : 442.30,
    'ValidFromDate' : `2011-03-17T`,
    'ValidToDate' : `2021-03-24T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId': '' },
        'SecondaryOwners' : []
    }
},
{
```

```
    'VIN' : '1HVBBAANXWH544237',
    'LicensePlateNumber' : 'LS477D',
    'State' : 'WA',
    'City' : 'Tacoma',
    'PendingPenaltyTicketAmount' : 42.20,
    'ValidFromDate' : `2011-10-26T`,
    'ValidToDate' : `2023-09-25T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId': '' },
        'SecondaryOwners' : []
    }
},
{
    'VIN' : '1C4RJFAG0FC625797',
    'LicensePlateNumber' : 'TH393F',
    'State' : 'WA',
    'City' : 'Olympia',
    'PendingPenaltyTicketAmount' : 30.45,
    'ValidFromDate' : `2013-09-02T`,
    'ValidToDate' : `2024-03-19T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId': '' },
        'SecondaryOwners' : []
    }
} >>
```

9. 마지막으로 원장에 등록된 차량을 설명하는 문서로 `Vehicle` 테이블을 채우십시오.

   쿼리 편집기 창에서 다음 문을 입력한 후 실행을 선택합니다.

```
INSERT INTO Vehicle
<< {
    'VIN' : '1N4AL11D75C109151',
    'Type' : 'Sedan',
    'Year' : 2011,
    'Make' : 'Audi',
    'Model' : 'A5',
    'Color' : 'Silver'
},
{
    'VIN' : 'KM8SRDHF6EU074761',
    'Type' : 'Sedan',
    'Year' : 2015,
    'Make' : 'Tesla',
```

```
    'Model' : 'Model S',
    'Color' : 'Blue'
},
{
    'VIN' : '3HGGK5G53FM761765',
    'Type' : 'Motorcycle',
    'Year' : 2011,
    'Make' : 'Ducati',
    'Model' : 'Monster 1200',
    'Color' : 'Yellow'
},
{
    'VIN' : '1HVBBAANXWH544237',
    'Type' : 'Semi',
    'Year' : 2009,
    'Make' : 'Ford',
    'Model' : 'F 150',
    'Color' : 'Black'
},
{
    'VIN' : '1C4RJFAG0FC625797',
    'Type' : 'Sedan',
    'Year' : 2019,
    'Make' : 'Mercedes',
    'Model' : 'CLK 350',
    'Color' : 'White'
} >>
```

다음으로 SELECT 문을 사용하여 vehicle-registration 원장의 테이블에서 데이터를 읽을 수 있습니다. 3단계: 원장에서 테이블 쿼리로 이동합니다.

3단계: 원장에서 테이블 쿼리

Amazon QLDB 원장에 테이블을 생성하고 데이터를 로드한 후 쿼리를 실행하여 방금 삽입한 차량 등록 데이터를 검토할 수 있습니다. QLDB는 PartiQL을 쿼리 언어로 사용하고 Amazon Ion을 문서 지향 데이터 모델로 사용합니다.

PartiQL은 Ion과 함께 작동하도록 확장된 오픈 소스 SQL 호환 쿼리 언어입니다. PartiQL을 사용하면 익숙한 SQL 연산자를 사용하여 데이터를 삽입, 쿼리 및 관리할 수 있습니다. Amazon Ion은 JSON의 상위 집합입니다. Ion은 정형, 반정형 및 중첩 데이터를 저장하고 처리할 수 있는 유연성을 제공하는 오픈 소스 문서 기반 데이터 형식입니다.

이 단계에서는 SELECT 명령문을 사용하여 vehicle-registration 원장의 테이블에서 데이터를 읽습니다.

> **Warning**
>
> 인덱싱된 조회 없이 QLDB에서 쿼리를 실행하면 전체 테이블 스캔이 호출됩니다. PartiQL은 SQL과 호환되므로 이러한 쿼리를 지원합니다. 하지만 QLDB의 프로덕션 사용 사례에 대해서는 테이블 스캔을 실행하지 마세요. 테이블 스캔은 동시성 충돌 및 트랜잭션 시간 초과를 포함하여 대규모 테이블에서 성능 문제를 일으킬 수 있습니다.
> 인덱싱된 필드 또는 문서 ID(예: `WHERE indexedField = 123` 또는 `WHERE indexedField IN (456, 789)`)에서 동등 연산자를 사용하여 WHERE 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 자세한 내용은 쿼리 성능 최적화을 참조하십시오.

테이블을 쿼리하려면

1. https://console.aws.amazon.com/qldb에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 PartiQL 편집기를 선택합니다.
3. vehicle-registration 원장을 선택합니다.
4. 쿼리 편집기 창에서 다음 문을 입력하여 원장에 추가한 특정 차량 식별 번호(VIN)에 대해 Vehicle 테이블을 쿼리한 후 실행을 선택합니다.

   명령문을 실행하려면 키보드 단축키(Windows의 경우 Ctrl+Enter, macOS의 경우 Cmd+Return)를 사용할 수도 있습니다. 키보드 단축키에 대한 자세한 내용은 PartiQL 편집기 키보드 바로 가기 섹션을 참조하세요.

   ```
   SELECT * FROM Vehicle AS v
   WHERE v.VIN = '1N4AL11D75C109151'
   ```

5. 내부 조인 쿼리를 작성할 수 있습니다. 이 쿼리 예제에서는 Vehicle를 VehicleRegistration와 조인하고 지정된 VIN에 대해 등록된 차량의 속성과 함께 등록 정보를 반환합니다.

   다음 문을 입력한 다음 실행을 선택합니다.

   ```
   SELECT v.VIN, r.LicensePlateNumber, r.State, r.City, r.Owners
   FROM Vehicle AS v, VehicleRegistration AS r
   WHERE v.VIN = '1N4AL11D75C109151'
   ```

```
AND v.VIN = r.VIN
```

Person 및 DriversLicense 테이블을 조인하여 원장에 추가된 드라이버와 관련된 속성을 볼 수도 있습니다.

다음에 대해 이 단계를 반복합니다.

```
SELECT * FROM Person AS p, DriversLicense AS l
WHERE p.GovId = l.LicensePlateNumber
```

vehicle-registration 원장의 테이블에 있는 문서를 수정하는 방법에 대한 자세한 내용은 4단계: 원장의 문서 수정을 참조하십시오.

4단계: 원장의 문서 수정

이제 작업할 데이터가 있으므로 Amazon QLDB에서 vehicle-registration 원장에 있는 문서를 변경할 수 있습니다. VIN 1N4AL11D75C109151를 갖는 Audi A5를 예로 들어 보겠습니다. 이 차는 처음에 워싱턴 주 시애틀에 있는 라울 루이스라는 운전자가 소유하고 있습니다.

라울이 워싱턴 주 에버렛에 사는 브렌트 로건이라는 주민에게 차를 판다고 가정해 보겠습니다. 그러던 중 브렌트와 알렉시스 페나는 결혼하기로 결정합니다. 브렌트는 등기부 상에 알렉시스를 보조 소유자로 추가하고자 합니다. 이 단계에서는 다음과 같은 데이터 조작 언어(DML) 문을 통해 이러한 이벤트를 반영하기 위해 원장을 적절히 변경하는 방법을 설명합니다.

> **Tip**
>
> 가장 좋은 방법은 문서의 시스템 할당 id를 외래 키로 사용하는 것입니다. 고유 식별자(예: 차량의 VIN)로 사용되는 필드를 정의할 수 있지만 문서의 실제 고유 식별자는 id입니다. 이 필드는 문서의 메타데이터에 포함되며 커밋된 뷰(테이블의 시스템 정의 뷰)에서 쿼리할 수 있습니다.
>
> QLDB 뷰에 대한 자세한 내용은 핵심 개념 섹션을 참조하세요. 메타데이터에 대해 자세히 알아보려면 문서 메타데이터 쿼리 섹션을 참조하세요.

문서를 수정하려면

1. https://console.aws.amazon.com/qldb에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 PartiQL 편집기를 선택합니다.

3. vehicle-registration 원장을 선택합니다.

> **Note**
>
> 콘솔의 자동 샘플 데이터 로드 기능을 사용하여 원장을 설정하는 경우 6단계로 넘어가십시오.

4. INSERT 문을 수동으로 실행하여 샘플 데이터를 로드하는 경우 다음 단계를 계속하십시오.

   처음에 라울을 이 차량의 소유주로 등록하려면 Person 테이블에서 Raul의 시스템 할당 문서 id를 찾는 것부터 시작합니다. 이 필드는 문서의 메타데이터에 포함되어 있으며, 커밋된 보기라고 하는 테이블의 시스템 정의 뷰에서 쿼리할 수 있습니다.

   쿼리 편집기 창에서 다음 문을 입력한 후 실행을 선택합니다.

```
SELECT metadata.id FROM _ql_committed_Person AS p
WHERE p.data.FirstName = 'Raul' and p.data.LastName = 'Lewis'
```

   접두사 _ql_committed_는 Person 테이블의 커밋된 뷰를 쿼리하려는 것을 나타내는 예약된 접두사입니다. 이 뷰에서는 데이터가 data 필드에 중첩되고 메타데이터는 metadata 필드에 중첩됩니다.

5. 이제 이 id를 UPDATE 문에 사용하여 VehicleRegistration 테이블의 해당 문서를 수정하십시오. 다음 문을 입력한 다음 실행을 선택합니다.

```
UPDATE VehicleRegistration AS r
SET r.Owners.PrimaryOwner.PersonId = '294jJ3YUoH1IEEm8GSabOs' --replace with your
 id
WHERE r.VIN = '1N4AL11D75C109151'
```

   이 문을 실행하여 Owners 필드를 수정했는지 확인하십시오.

```
SELECT r.Owners FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
```

6. 차량 소유권을 에버렛에 있는 브렌트에게 이전하려면 먼저 다음 문을 사용하여 Person 테이블에서 그의 id을 찾으십시오.

```
SELECT metadata.id FROM _ql_committed_Person AS p
WHERE p.data.FirstName = 'Brent' and p.data.LastName = 'Logan'
```

그런 다음, 이 id를 사용하여 VehicleRegistration 테이블의 PrimaryOwner 및 City를 업데이트하십시오.

```
UPDATE VehicleRegistration AS r
SET r.Owners.PrimaryOwner.PersonId = '7NmE8YLPbXc0IqesJy1rpR', --replace with your
 id
    r.City = 'Everett'
WHERE r.VIN = '1N4AL11D75C109151'
```

다음 문을 실행하여 PrimaryOwner 및 City 필드를 수정했는지 확인하십시오.

```
SELECT r.Owners.PrimaryOwner, r.City
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
```

7. 알렉시스를 차량의 보조 소유자로 추가하려면 그녀의 Person id를 찾으십시오.

```
SELECT metadata.id FROM _ql_committed_Person AS p
WHERE p.data.FirstName = 'Alexis' and p.data.LastName = 'Pena'
```

그런 다음 다음 FROM-INSERT DML 문을 사용하여 이 id을 SecondaryOwners 목록에 삽입합니다.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners
    VALUE { 'PersonId' : '5Ufgdlnj06gF5CWcOIu64s' } --replace with your id
```

이 문을 실행하여 SecondaryOwners를 수정했는지 확인하십시오.

```
SELECT r.Owners.SecondaryOwners FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
```

vehicle-registration 원장의 이러한 변경 사항을 검토하려면 5단계: 문서의 개정 기록 보기을 참조하세요.

5단계: 문서의 개정 기록 보기

VIN이 1N4AL11D75C109151인 차량의 등록 데이터를 수정한 후, 등록된 모든 소유자의 기록과 기타 업데이트된 필드를 쿼리할 수 있습니다. 내장 기록 함수를 쿼리하여 삽입, 업데이트 및 삭제한 문서의 모든 개정 사항을 볼 수 있습니다.

기록 함수는 애플리케이션 데이터와 관련 메타데이터를 모두 포함하는 테이블의 커밋된 보기에서 개정 사항을 반환합니다. 메타데이터는 각 개정이 이루어진 시점, 순서, 어떤 트랜잭션이 이 개정을 수행했는지를 정확하게 보여줍니다.

이 단계에서는 `vehicle-registration` 원장의 `VehicleRegistration` 테이블에 있는 문서의 개정 기록을 쿼리합니다.

개정 기록을 보려면

1. https://console.aws.amazon.com/qldb에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 PartiQL 편집기를 선택합니다.
3. `vehicle-registration` 원장을 선택합니다.
4. 문서 기록을 조회하려면 먼저 고유한 `id`를 찾아야 합니다. 커밋된 보기를 쿼리하는 것 외에도 문서 `id`를 가져오는 또 다른 방법은 테이블의 기본 사용자 뷰에서 `BY` 키워드를 사용하는 것입니다. 자세한 내용은 BY 절을 사용하여 문서 ID 쿼리하기 섹션을 참조하세요.

   쿼리 편집기 창에서 다음 문을 입력한 후 실행을 선택합니다.

   ```
   SELECT r_id FROM VehicleRegistration AS r BY r_id
   WHERE r.VIN = '1N4AL11D75C109151'
   ```

5. 그런 다음 이 `id` 값을 사용하여 기록 함수를 쿼리할 수 있습니다. 다음 문을 입력한 다음 실행을 선택합니다. 필요에 따라 `id` 값을 고유한 문서 ID로 교체합니다.

   ```
   SELECT h.data.VIN, h.data.City, h.data.Owners
   FROM history(VehicleRegistration) AS h
   WHERE h.metadata.id = 'ADR2LQq48kB9neZDupQrMm' --replace with your id
   ```

   > **Note**
   >
   > 이 자습서의 목적을 위해 이 기록 쿼리는 문서 ID ADR2LQq48kB9neZDupQrMm의 모든 개정 내용을 반환합니다. 그러나 가장 좋은 방법은 문서 ID와 날짜 범위(시작 시간 및 종료 시간)를 모두 사용하여 기록 쿼리를 한정하는 것입니다.

> QLDB에서는 모든 SELECT 쿼리가 트랜잭션에서 처리되며 트랜잭션 시간 초과 제한이 적용됩니다. 시작 시간과 종료 시간을 포함하는 기록 쿼리는 날짜 범위 한정이라는 이점을 갖습니다. 자세한 내용은 기록 함수을 참조하십시오.

기록 함수는 커밋된 보기와 동일한 스키마의 문서를 반환합니다. 이 예제에서는 수정된 차량 등록 데이터를 프로젝션합니다. 다음과 같이 출력됩니다.

VIN	구/군/시	소유자
"1N4AL11D75C109151"	"Seattle"	{PrimaryOwner:{PersonId:""},SecondaryOwners:[]}
"1N4AL11D75C109151"	"Seattle"	{PrimaryOwner:{PersonId:"294jJ3YUoH1IEEm8GSabOs"},SecondaryOwners:[]}
"1N4AL11D75C109151"	"Everett"	{PrimaryOwner:{PersonId:"7NmE8YLPbXc0IqesJy1rpR"},SecondaryOwners:[]}
"1N4AL11D75C109151"	"Everett"	{PrimaryOwner:{PersonId:"7NmE8YLPbXc0IqesJy1rpR"},SecondaryOwners:[{PersonId:"5Ufgdlnj06gF5CWcOIu64s"}]}

> **Note**
>
> 기록 쿼리가 문서 개정 내용을 항상 순차적으로 반환하지 않을 수도 있습니다.

출력을 검토하고 변경 사항이 4단계: 원장의 문서 수정에서 수행한 내용을 반영하는지 확인합니다.

6. 그런 다음 각 개정의 문서 메타데이터를 검사할 수 있습니다. 다음 문을 입력한 다음 실행을 선택합니다. 이번에도 `id` 값을 자신의 문서 ID로 적절하게 교체하십시오.

```
SELECT VALUE h.metadata
FROM history(VehicleRegistration) AS h
WHERE h.metadata.id = 'ADR2LQq48kB9neZDupQrMm' --replace with your id
```

다음과 같이 출력됩니다.

versio	id	txTime	txId
0	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T19:20:360d-3Z	"FMoVdWuPxJg3k466Iz4i75"
1	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T21:40:199d-3Z	"KWByxe842Xw8DNHcvARPOt"
2	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T21:44:432d-3Z	"EKwDOJRwbHpFvmAyJ2Kdh9"
3	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T21:49:254d-3Z	"96EiZd7vCmJ6RAvOvTZ4YA"

이 메타데이터 필드는 각 항목이 언제 수정되었는지, 어떤 트랜잭션을 통해 수정되었는지에 대한 세부 정보를 제공합니다. 이 데이터에서 다음을 추론할 수 있습니다.

- 이 문서는 시스템이 할당한 id인 ADR2LQq48kB9neZDupQrMm로 고유하게 식별됩니다 이는 Base62로 인코딩된 문자열로 표시되는 범용 고유 식별자(UUID)입니다.
- txTime는 문서(버전 0)의 초기 개정이 2019-05-23T19:20:360d-3Z에 작성되었음을 보여줍니다.
- 이후의 각 트랜잭션은 동일한 문서 id, 증분된 버전 번호, 업데이트된 txId 및 txTime를 사용하여 새 개정을 생성합니다.

vehicle-registration 원장에서 암호화된 방식으로 문서 개정을 검증하려면 6단계: 원장에 있는 문서 검증로 이동하십시오.

6단계: 원장에 있는 문서 검증

Amazon QLDB를 사용하면 SHA-256 암호화 해싱을 사용하여 원장 저널에 있는 문서의 무결성을 효율적으로 검증할 수 있습니다. 이 예에서 알렉시스와 브렌트는 자동차 대리점에서 VIN이 1N4AL11D75C109151인 차량을 트레이드하여 새 모델로 업그레이드하기로 결정합니다. 딜러는 등록 사무소를 통해 차량 소유권을 확인하는 것으로 절차를 시작합니다.

QLDB에서 검증 및 암호화 해싱이 작동하는 방식에 대한 자세한 내용은 Amazon QLDB에서의 데이터 확인을 참조하십시오.

이 단계에서는 `vehicle-registration` 원장의 문서 개정을 확인합니다. 먼저 다이제스트를 요청합니다. 다이제스트는 출력 파일로 반환되며 원장의 전체 변경 내역에 대한 서명 역할을 합니다. 그런 다음 해당 다이제스트와 관련된 개정 증거를 요청합니다. 이 증거를 사용하면 모든 유효성 검사를 통과한 경우 개정 내용의 무결성을 확인할 수 있습니다.

다이제스트를 요청하려면

1. https://console.aws.amazon.com/qldb에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 원장을 선택합니다.
3. 원장 목록에서 `vehicle-registration`을 선택합니다.
4. 다이제스트 가져오기를 선택합니다. 다이제스트 가져오기 대화 상자에는 다음과 같은 다이제스트 세부 정보가 표시됩니다.

   - 다이제스트 – 요청한 다이제스트의 SHA-256 해시 값.
   - 다이제스트 팁 주소 – 요청한 다이제스트에 포함된 저널의 최신 블록 위치. 주소에는 다음과 같은 두 가지 필드가 있습니다.
     - `strandId` - 블록을 포함하는 저널 스트랜드의 고유 ID입니다.
     - `sequenceNo` – 스트랜드 내 블록의 위치를 지정하는 인덱스 번호.
   - 원장 – 다이제스트를 요청한 원장 이름.
   - 날짜 – 다이제스트를 요청한 시점의 타임스탬프.
5. 다이제스트 정보를 검토합니다. 그런 다음 저장을 선택합니다. 기본 파일 이름을 유지하거나 새 이름을 입력할 수 있습니다.

   이 단계는 콘텐츠가 포함된 일반 텍스트 파일을 Amazon Ion 형식으로 저장합니다. 파일의 파일 이름 확장자는 `.ion.txt`이며 이전 대화 상자에 나열된 모든 다이제스트 정보를 포함합니다. 다음은 다이제스트 파일의 내용을 보여주는 예제입니다. 필드 순서는 브라우저에 따라 다를 수 있습니다.

```
{
  "digest": "42zaJOfV8iGutVGNaIuzQWhD5Xb/5B9lScHnvxPXm9E=",
  "digestTipAddress": "{strandId:\"BlFTjlSXze9BIh1KOszcE3\",sequenceNo:73}",
  "ledger": "vehicle-registration",
  "date": "2019-04-17T16:57:26.749Z"
}
```

6. 나중에 액세스할 수 있는 위치에 이 파일을 저장하십시오. 다음 단계에서는 이 파일을 사용하여 문서 개정을 검증합니다.

원장 다이제스트를 저장한 후 해당 다이제스트를 기준으로 문서 개정을 확인하는 프로세스를 시작할 수 있습니다.

> **Note**
>
> 검증을 위한 프로덕션 사용 사례에서는 두 작업을 연속해서 수행하는 대신 이전에 저장한 다이제스트를 사용합니다. 가장 좋은 방법은 나중에 확인하려는 개정 내용이 저널에 기록되는 즉시 다이제스트를 요청하여 저장하는 것입니다.

문서 개정을 검증하려면

1. 먼저 검증하려는 문서 개정의 `id` 및 `blockAddress`를 원장에서 쿼리하십시오. 이 필드는 문서의 메타데이터에 포함되며 커밋된 뷰에서 쿼리할 수 있습니다.

   문서 `id`는 시스템에서 할당한 고유 ID 문자열입니다. `blockAddress`는 개정이 커밋된 블록 위치를 지정하는 Ion 구조입니다.

   QLDB 콘솔의 탐색 창에서 PartiQL 편집기를 선택합니다.
2. `vehicle-registration` 원장을 선택합니다.
3. 쿼리 편집기 창에서 다음 문을 입력한 후 실행을 선택합니다.

```
SELECT r.metadata.id, r.blockAddress
FROM _ql_committed_VehicleRegistration AS r
WHERE r.data.VIN = '1N4AL11D75C109151'
```

4. 쿼리가 반환하는 `id` 및 `blockAddress` 값을 복사하여 저장합니다. `id` 필드의 큰따옴표는 반드시 생략하십시오 Amazon Ion에서 문자열 데이터 유형이 큰따옴표로 구분됩니다.

5. 이제 문서 수정본을 선택했으니 확인 프로세스를 시작할 수 있습니다.

   탐색 창에서 검증을 선택합니다.

6. 문서 검증 양식의 검증하려는 문서 지정에 다음 입력 파라미터를 입력합니다.

   - 원장 — `vehicle-registration`를 선택합니다.
   - 블록 주소 – 3단계에서 쿼리를 통해 반환된 `blockAddress` 값.
   - 문서 ID – 3단계에서 쿼리를 통해 반환된 `id` 값.

7. 확인에 사용할 다이제스트 지정에서 다이제스트 선택을 선택하여 이전에 저장한 다이제스트를 선택합니다. 파일이 유효하면 콘솔의 모든 다이제스트 필드가 자동으로 채워집니다. 또는 다이제스트 파일에서 직접 다음 값을 수동으로 복사하여 붙여넣을 수 있습니다.

   - 다이제스트 – 다이제스트 파일의 `digest` 값.
   - 다이제스트 팁 주소 – 다이제스트 파일의 `digestTipAddress` 값.

8. 문서 및 다이제스트 입력 파라미터를 검토한 다음 검증을 선택합니다.

   콘솔은 다음 두 단계를 자동화합니다.

   a. 지정된 문서에 대한 증거를 QLDB에 요청합니다.

   b. QLDB에서 반환한 증거를 사용하여 제공된 다이제스트에 대해 문서 개정을 검증하는 클라이언트 측 API를 호출합니다.

   콘솔은 검증 결과 카드에 요청 결과를 표시합니다. 자세한 내용은 확인 결과을 참조하십시오.

9. 확인 로직을 테스트하려면 문서 개정 확인하기에서 6~8단계를 반복하되, 다이제스트 입력 문자열에서 한 글자만 변경합니다. 그러면 검증 요청이 실패하고 적절한 오류 메시지가 표시됩니다.

`vehicle-registration` 원장을 더 이상 사용할 필요가 없는 경우 7단계(선택 사항): 리소스 정리로 진행하십시오.

7단계(선택 사항): 리소스 정리

`vehicle-registration` 원장을 계속 사용할 수 있습니다. 그러나 더 이상 필요하지 않은 경우 삭제해야 합니다.

원장에 대해 삭제 방지가 활성화된 경우 QLDB API, AWS Command Line Interface(AWS CLI) 또는 QLDB 콘솔을 사용하여 원장을 삭제하기 전에 먼저 이를 비활성화해야 합니다.

원장을 삭제하려면

1. https://console.aws.amazon.com/qldb에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 원장을 선택합니다.
3. 이 원장에 대해 삭제 방지가 활성화된 경우 먼저 이를 비활성화해야 합니다.

   원장 목록에서 `vehicle-registration`을 선택한 다음 원장 편집을 선택합니다.
4. 원장 편집 페이지에서 삭제 방지를 해제한 다음 변경 사항 확인을 선택합니다.
5. 원장 목록에서 `vehicle-registration`을 다시 선택한 다음 삭제를 선택합니다.
6. 제공된 필드에 **`delete vehicle-registration`**을 입력하여 이를 확인합니다.

QLDB에서의 원장 작업에 대한 자세한 내용은 Amazon QLDB 시작하기: 다음 단계을 참조하십시오.

Amazon QLDB 시작하기: 다음 단계

Amazon QLDB 사용에 대한 자세한 내용은 다음 주제를 참조하십시오.

- Amazon QLDB에서 데이터 및 기록 작업
- Amazon QLDB 드라이버 시작하기
- Amazon QLDB 동시성 모델
- Amazon QLDB에서 저널 데이터 내보내기
- Amazon QLDB에서 저널 데이터 스트리밍
- Amazon QLDB에서의 데이터 확인
- Amazon QLDB PartiQL 참조

Amazon QLDB 드라이버 시작하기

이 장에는 QLDB 드라이버를 사용하여 Amazon QLDB로 개발하는 방법을 배우는 데 도움이 되는 실습용 자습서가 포함되어 있습니다. 드라이버는 QLDB API와의 상호 작용을 지원하는 AWS SDK를 기반으로 구축되었습니다.

QLDB 세션 추상화

드라이버는 트랜잭션 데이터 API(QLDB 세션) 위에 높은 수준의 추상화 계층을 제공합니다. SendCommand API 호출을 관리하여 원장 데이터에서 PartiQL 문을 실행하는 프로세스를 간소화합니다. 이러한 API 호출에는 세션 관리, 트랜잭션, 오류 발생 시 재시도 정책 등 드라이버가 대신 처리하는 여러 파라미터가 필요합니다. 또한 드라이버에는 성능 최적화 기능이 있으며 QLDB와의 상호 작용을 위한 모범 사례를 적용합니다.

> **Note**
>
> Amazon QLDB API 참조에 나열된 리소스 관리 API 작업과 상호 작용하려면 드라이버 대신 AWS SDK를 직접 사용합니다. 관리 API는 원장 리소스 관리와 내보내기, 스트리밍, 데이터 확인과 같은 비트랜잭션 데이터 작업에만 사용합니다.

Amazon Ion 지원

또한 드라이버는 Amazon Ion 라이브러리를 사용하여 트랜잭션 실행 시 Ion 데이터 처리를 지원합니다. 또한 이러한 라이브러리는 Ion 값의 해시 계산을 담당합니다. QLDB는 데이터 트랜잭션 요청의 무결성을 확인하기 위해 이러한 Ion 해시를 필요로 합니다.

드라이버 용어

이 도구는 개발자에게 친숙한 인터페이스를 제공하는 다른 데이터베이스 드라이버와 비슷하기 때문에 드라이버라고 불립니다. 이러한 드라이버는 표준 명령 및 기능 세트를 서비스의 하위 수준 API에 필요한 특정 호출로 변환하는 로직을 유사하게 캡슐화합니다.

드라이버는 GitHub의 오픈 소스이며 다음 프로그래밍 언어에서 사용할 수 있습니다.

- Java 드라이버
- .NET 드라이버
- Go 드라이버
- Node.js 드라이버

- Python 드라이버

지원되는 모든 프로그래밍 언어에 대한 일반 드라이버 정보와 추가 자습서는 다음 항목을 참조하십시오.

- 드라이버를 사용한 세션 관리
- 드라이버 권장 사항
- 드라이버 재시도 정책
- 일반적인 오류
- 샘플 애플리케이션 자습서
- Amazon Ion 작업
- PartiQL 문 통계 가져오기

Java용 Amazon QLDB 드라이버

원장의 데이터로 작업하려면 AWS에서 제공하는 드라이버를 사용하여 Java 애플리케이션에서 Amazon QLDB에 연결할 수 있습니다. 다음 주제에서는 Java용 QLDB 드라이버를 시작하는 방법에 대해 설명합니다.

주제

- 드라이버 리소스
- 사전 조건
- 기본 AWS 보안 인증 정보 및 리전 설정
- 설치
- Java용 Amazon QLDB 드라이버 - 빠른 시작 자습서
- Java용 Amazon QLDB 드라이버 - Cookbook 참조

드라이버 리소스

Java 드라이버가 지원하는 기능에 대한 자세한 정보는 다음 리소스를 참조하십시오.

- API 참조: 2.x, 1.x
- 드라이버 소스 코드(GitHub)
- 샘플 애플리케이션 소스 코드(GitHub)

- 원장 로더 프레임워크(GitHub)
- Amazon Ion 코드 예제

사전 조건

Java용 QLDB 드라이버를 시작하기 전에 다음을 수행해야 합니다.

1. Amazon QLDB 액세스의 AWS 설정 지침을 따르십시오. 다음 내용이 해당됩니다.
    1. AWS에 가입합니다.
    2. 적절한 QLDB 권한을 가진 사용자를 생성합니다.
    3. 개발을 위한 프로그래밍 방식 액세스 권한을 부여합니다.
2. 다음을 다운로드하고 설치하여 Java 개발 환경을 설정합니다.
    1. Java SE 개발 키트 8(예: Amazon Corretto 8)
    2. (선택 사항) Eclipse 또는 IntelliJ와 같은 Java 통합 개발 환경(IDE)을 선택할 수 있습니다.
3. 기본 AWS 보안 인증 정보 및 리전 설정별로 AWS SDK for Java에 대한 개발 환경을 구성합니다.

그런 다음 전체 자습서 샘플 애플리케이션을 다운로드하거나 Java 프로젝트에 드라이버만 설치하고 단축 코드 예제를 실행할 수 있습니다.

- 기존 프로젝트에 QLDB 드라이버와 AWS SDK for Java를 설치하려면 설치로 이동하세요.
- 프로젝트를 설정하고 원장에 대한 기본 데이터 트랜잭션을 보여주는 단축 코드 예제를 실행하려면 빠른 시작 자습서를 참조하십시오.
- 전체 자습서 샘플 애플리케이션에서 데이터 및 관리 API 작업에 대한 보다 심층적인 예제를 실행하려면 Java 자습서를 참조하십시오.

기본 AWS 보안 인증 정보 및 리전 설정

QLDB 드라이버와 기본 AWS SDK for Java를 사용하려면 런타임 시 애플리케이션에 AWS 보안 인증을 제공해야 합니다. 이 설명서의 코드 예제에서는 AWS SDK for Java 2.x 개발자 안내서의 보안 인증 정보 및 리전 설정에 설명된 대로 AWS 보안 인증 정보 파일을 사용한다고 가정합니다.

이 단계의 일부로 기본값 AWS 리전을 설정하여 기본 QLDB 엔드포인트를 설정해야 합니다. 코드 예제는 기본 AWS 리전의 QLDB에 연결됩니다. QLDB를 사용할 수 있는 리전의 전체 목록은 AWS 일반 참조에서 Amazon QLDB 엔드포인트 및 할당량을 참조하십시오.

다음은 ~/.aws/credentials라는 AWS 자격 증명 파일의 예입니다. 여기서 물결 문자(~)는 사용자의 홈 디렉터리입니다.

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

your_access_key_id 및 *your_secret_access_key* 값을 자신의 AWS 보안 인증 정보 값으로 대체합니다.

설치

QLDB는 다음 Java 드라이버 버전과 해당 AWS SDK 종속성을 지원합니다.

드라이버 버전	AWS SDK	상태	최근 릴리스 날짜
1.x	AWS SDK for Java 1.x	프로덕션 릴리스	2020년 3월 20일
2.x	AWS SDK for Java 2.x	프로덕션 릴리스	2021년 6월 4일

QLDB 드라이버를 설치하려면 Gradle 또는 Maven과 같은 종속성 관리 시스템을 사용할 것을 권장합니다. 예를 들어, Java 프로젝트에 다음 아티팩트를 종속성 항목으로 추가합니다.

2.x

Gradle

build.gradle 구성 파일에 이 종속성을 추가합니다.

```
dependencies {
    compile group: 'software.amazon.qldb', name: 'amazon-qldb-driver-java', version:
 '2.3.1'
}
```

Maven

pom.xml 구성 파일에 이 종속성을 추가합니다.

```
<dependencies>
  <dependency>
    <groupId>software.amazon.qldb</groupId>
    <artifactId>amazon-qldb-driver-java</artifactId>
    <version>2.3.1</version>
  </dependency>
</dependencies>
```

이 아티팩트에는 AWS SDK for Java 2.x 코어 모듈, Amazon Ion 라이브러리 및 기타 필수 종속성이 자동으로 포함됩니다.

1.x

Gradle

`build.gradle` 구성 파일에 이 종속성을 추가합니다.

```
dependencies {
    compile group: 'software.amazon.qldb', name: 'amazon-qldb-driver-java', version:
 '1.1.0'
}
```

Maven

`pom.xml` 구성 파일에 이 종속성을 추가합니다.

```
<dependencies>
  <dependency>
    <groupId>software.amazon.qldb</groupId>
    <artifactId>amazon-qldb-driver-java</artifactId>
    <version>1.1.0</version>
  </dependency>
</dependencies>
```

이 아티팩트에는 AWS SDK for Java 코어 모듈, Amazon Ion 라이브러리 및 기타 필수 종속성이 자동으로 포함됩니다.

> **Important**
>
> Amazon Ion 네임스페이스 — 애플리케이션에서 Amazon Ion 클래스를 가져올 때는 네임스페이스 `com.amazon.ion` 아래에 있는 패키지를 사용해야 합니다. AWS SDK for Java은

네임스페이스 `software.amazon.ion` 아래의 다른 Ion 패키지에 따라 다르지만 이 패키지는 QLDB 드라이버와 호환되지 않는 레거시 패키지입니다.

원장에서 기본 데이터 트랜잭션을 실행하는 방법에 대한 단축 코드 예제는 Cookbook 참조를 참조하십시오.

기타 옵션 라이브러리

선택 사항으로 프로젝트에 다음과 같은 유용한 라이브러리를 추가할 수도 있습니다. 이러한 아티팩트는 Java 자습서 샘플 애플리케이션의 필수 종속성입니다.

1. aws-java-sdk-qldb – AWS SDK for Java의 QLDB 모듈입니다. 현재 지원되는 QLDB 최소 버전은 `1.11.785`입니다.

   애플리케이션에서 이 모듈을 사용하여 Amazon QLDB API 참조에 나열된 관리 API 작업과 직접 상호 작용할 수 있습니다.

2. jackson-dataformat-ion – FasterXML의 Ion용 Jackson 데이터 형식 모듈입니다. 샘플 애플리케이션에는 `2.10.0` 버전 이상이 필요합니다.

Gradle

`build.gradle` 구성 파일에 이 종속성을 추가합니다.

```
dependencies {
    compile group: 'com.amazonaws', name: 'aws-java-sdk-qldb', version: '1.11.785'
    compile group: 'com.fasterxml.jackson.dataformat', name: 'jackson-dataformat-
ion', version: '2.10.0'
}
```

Maven

`pom.xml` 구성 파일에 이 종속성을 추가합니다.

```
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-qldb</artifactId>
    <version>1.11.785</version>
```

```
  </dependency>
  <dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-ion</artifactId>
    <version>2.10.0</version>
  </dependency>
</dependencies>
```

Java용 Amazon QLDB 드라이버 - 빠른 시작 자습서

이 자습서에서는 Java용 Amazon QLDB 드라이버의 최신 버전을 사용하여 간단한 애플리케이션을 설정하는 방법을 알아봅니다. 이 안내서에는 드라이버 설치 단계 및 기본적인 CRUD(생성, 읽기, 업데이트 및 삭제) 작업에 대한 단축 코드 예제가 포함되어 있습니다. 전체 샘플 애플리케이션에서 이러한 작업을 보여 주는 자세한 예를 보려면 Java 자습서 섹션을 참조하세요.

주제

필수 조건

시작하기 전에 다음을 수행해야 합니다.

1. Java 드라이버를 위한 사전 조건을 아직 완료하지 않은 경우, 완료하세요. 여기에는 AWS 가입, 개발을 위한 프로그래밍 액세스 권한 부여, Java 통합 개발 환경(IDE)설치가 포함됩니다.
2. `quick-start`라는 명칭의 원장을 생성합니다.

   원장 생성 방법을 알아보려면 콘솔 시작하기의 Amazon QLDB 원장의 기본 작업 또는 1단계: 새 원장 생성 섹션을 참조하세요.

1단계: 프로젝트 설정

먼저 Java 프로젝트를 설정합니다. 이 자습서에서는 Maven 종속성 관리 시스템을 사용하는 것이 좋습니다.

> **Note**
>
> 이러한 설정 단계를 자동화하는 기능이 있는 IDE를 사용하는 경우 2단계: 드라이버 초기화로 넘어가도 됩니다.

1. 애플리케이션을 위한 폴더를 생성합니다.

```
$ mkdir myproject
$ cd myproject
```

2. 다음 명령을 입력하여 Maven 템플릿에서 프로젝트를 초기화합니다. *project-package*, *project-name*, *maven-template*을 원하는 값으로 적절하게 바꿉니다.

```
$ mvn archetype:generate
  -DgroupId=project-package \
  -DartifactId=project-name \
  -DarchetypeArtifactId=maven-template \
  -DinteractiveMode=false
```

*maven-template*의 경우 다음 기본 Maven 템플릿을 사용할 수 있습니다. `maven-archetype-quickstart`

3. Java용 QLDB 드라이버를 프로젝트 종속 항목으로 추가하려면 새로 만든 `pom.xml` 파일로 이동하여 다음 아티팩트를 추가합니다.

```
<dependency>
    <groupId>software.amazon.qldb</groupId>
    <artifactId>amazon-qldb-driver-java</artifactId>
    <version>2.3.1</version>
</dependency>
```

이 아티팩트에는 AWS SDK for Java 2.x 코어 모듈, Amazon Ion 라이브러리 및 기타 필수 종속성이 자동으로 포함됩니다. 이제 `pom.xml` 파일은 다음과 같아야 합니다.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>software.amazon.qldb</groupId>
  <artifactId>qldb-quickstart</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>qldb-quickstart</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>software.amazon.qldb</groupId>
        <artifactId>amazon-qldb-driver-java</artifactId>
        <version>2.3.1</version>
    </dependency>
  </dependencies>
</project>
```

4. App.java 파일을 엽니다.

   그런 다음 다음 단계의 코드 예를 점진적으로 추가하여 몇 가지 기본 CRUD 작업을 시도해 보세요. 또는 단계별 자습서를 건너뛰고 전체 애플리케이션을 실행할 수도 있습니다.

2단계: 드라이버 초기화

quick-start라는 명칭의 원장에 연결되는 드라이버의 인스턴스를 초기화합니다. 다음 코드를 App.java 파일에 추가합니다.

```
import java.util.*;
import com.amazon.ion.*;
import com.amazon.ion.system.*;
import software.amazon.awssdk.services.qldbsession.QldbSessionClient;
import software.amazon.qldb.*;
```

```
public final class App {
    public static IonSystem ionSys = IonSystemBuilder.standard().build();
    public static QldbDriver qldbDriver;

    public static void main(final String... args) {
        System.out.println("Initializing the driver");
        qldbDriver = QldbDriver.builder()
            .ledger("quick-start")
            .transactionRetryPolicy(RetryPolicy
                .builder()
                .maxRetries(3)
                .build())
            .sessionClientBuilder(QldbSessionClient.builder())
            .build();
    }
}
```

3단계: 테이블 및 인덱스 생성

다음 코드 예에서는 `CREATE TABLE` 및 `CREATE INDEX` 문을 실행하는 방법을 보여줍니다.

`main` 메서드에서 `People`라는 명칭의 표와 해당 표의 `lastName` 필드를 위한 인덱스를 만드는 다음 코드를 추가합니다. 인덱스는 쿼리 성능을 최적화하고 OCC(낙관적 동시성 제어) 충돌 예외를 제한하는 데 필요합니다.

```
// Create a table and an index in the same transaction
qldbDriver.execute(txn -> {
    System.out.println("Creating a table and an index");
    txn.execute("CREATE TABLE People");
    txn.execute("CREATE INDEX ON People(lastName)");
});
```

4단계: 문서 삽입

다음 코드 예에서는 `INSERT` 문을 실행하는 방법을 보여줍니다. QLDB는 PartiQL 쿼리 언어(SQL 호환) 및 Amazon Ion 데이터 형식(JSON의 상위 집합)을 지원합니다.

`People` 테이블에 문서를 삽입하는 다음 코드를 추가합니다.

```
// Insert a document
```

```
qldbDriver.execute(txn -> {
    System.out.println("Inserting a document");
    IonStruct person = ionSys.newEmptyStruct();
    person.put("firstName").newString("John");
    person.put("lastName").newString("Doe");
    person.put("age").newInt(32);
    txn.execute("INSERT INTO People ?", person);
});
```

이 예에서는 물음표(?)를 변수 자리 표시자로 사용하여 문서 정보를 해당 문에 전달합니다. 자리 표시자를 사용할 때는 `IonValue` 타입의 값을 전달해야 합니다.

> **Tip**
>
> 단일 INSERT 문을 사용하여 여러 문서를 삽입하려면 다음과 같이 IonList 타입의 파라미터(명시적으로 `IonValue`로 캐스팅됨)를 해당 문에 전달할 수 있습니다.
>
> ```
> // people is an IonList explicitly cast as an IonValue
> txn.execute("INSERT INTO People ?", (IonValue) people);
> ```
>
> `IonList`를 전달할 때는 변수 자리 표시자(?)를 이중 꺾쇠 괄호(`<<...>>`)로 묶지 마세요. 수동 PartiQL 문에서 이중 꺾쇠 괄호는 백으로 알려진 정렬되지 않은 모음을 의미합니다.

5단계: 문서 쿼리

다음 코드 예에서는 `SELECT` 문을 실행하는 방법을 보여줍니다.

`People` 테이블에서 문서를 쿼리하는 다음 코드를 추가합니다.

```
// Query the document
qldbDriver.execute(txn -> {
    System.out.println("Querying the table");
    Result result = txn.execute("SELECT * FROM People WHERE lastName = ?",
            ionSys.newString("Doe"));
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("firstName")); // prints John
    System.out.println(person.get("lastName")); // prints Doe
    System.out.println(person.get("age")); // prints 32
});
```

6단계: 문서 업데이트

다음 코드 예에서는 UPDATE 문을 실행하는 방법을 보여줍니다.

1. age를 42로 업데이트하여 People 표의 문서를 업데이트하는 다음 코드를 추가합니다.

```
// Update the document
qldbDriver.execute(txn -> {
    System.out.println("Updating the document");
    final List<IonValue> parameters = new ArrayList<>();
    parameters.add(ionSys.newInt(42));
    parameters.add(ionSys.newString("Doe"));
    txn.execute("UPDATE People SET age = ? WHERE lastName = ?", parameters);
});
```

2. 문서를 다시 쿼리하여 업데이트된 값을 확인합니다.

```
// Query the updated document
qldbDriver.execute(txn -> {
    System.out.println("Querying the table for the updated document");
    Result result = txn.execute("SELECT * FROM People WHERE lastName = ?",
            ionSys.newString("Doe"));
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("firstName")); // prints John
    System.out.println(person.get("lastName")); // prints Doe
    System.out.println(person.get("age")); // prints 32
});
```

3. Maven 또는 IDE를 사용하여 App.java 파일을 컴파일하고 실행합니다.

전체 애플리케이션 실행

다음 코드 예는 App.java 애플리케이션의 전체 버전입니다. 이전 단계를 개별적으로 수행하는 대신 이 코드 예를 처음부터 끝까지 복사하여 실행할 수도 있습니다. 이 애플리케이션은 quick-start이라는 명칭의 원장에 대한 몇 가지 기본 CRUD 작업을 보여줍니다.

> **Note**
>
> 이 코드를 실행하기 전에 quick-start 원장에 People이라는 명칭의 활성 테이블이 아직 없는지 확인하세요.

첫 번째 행에서 *project-package*를 1단계: 프로젝트 설정의 Maven 명령에 사용한 groupId 값으로 바꾸세요.

```
package project-package;

import java.util.*;
import com.amazon.ion.*;
import com.amazon.ion.system.*;
import software.amazon.awssdk.services.qldbsession.QldbSessionClient;
import software.amazon.qldb.*;

public class App {
    public static IonSystem ionSys = IonSystemBuilder.standard().build();
    public static QldbDriver qldbDriver;

    public static void main(final String... args) {
        System.out.println("Initializing the driver");
        qldbDriver = QldbDriver.builder()
            .ledger("quick-start")
            .transactionRetryPolicy(RetryPolicy
                .builder()
                .maxRetries(3)
                .build())
            .sessionClientBuilder(QldbSessionClient.builder())
            .build();

        // Create a table and an index in the same transaction
        qldbDriver.execute(txn -> {
            System.out.println("Creating a table and an index");
            txn.execute("CREATE TABLE People");
            txn.execute("CREATE INDEX ON People(lastName)");
        });

        // Insert a document
        qldbDriver.execute(txn -> {
            System.out.println("Inserting a document");
            IonStruct person = ionSys.newEmptyStruct();
            person.put("firstName").newString("John");
            person.put("lastName").newString("Doe");
            person.put("age").newInt(32);
            txn.execute("INSERT INTO People ?", person);
        });
```

```
        // Query the document
        qldbDriver.execute(txn -> {
            System.out.println("Querying the table");
            Result result = txn.execute("SELECT * FROM People WHERE lastName = ?",
                    ionSys.newString("Doe"));
            IonStruct person = (IonStruct) result.iterator().next();
            System.out.println(person.get("firstName")); // prints John
            System.out.println(person.get("lastName")); // prints Doe
            System.out.println(person.get("age")); // prints 32
        });

        // Update the document
        qldbDriver.execute(txn -> {
            System.out.println("Updating the document");
            final List<IonValue> parameters = new ArrayList<>();
            parameters.add(ionSys.newInt(42));
            parameters.add(ionSys.newString("Doe"));
            txn.execute("UPDATE People SET age = ? WHERE lastName = ?", parameters);
        });

        // Query the updated document
        qldbDriver.execute(txn -> {
            System.out.println("Querying the table for the updated document");
            Result result = txn.execute("SELECT * FROM People WHERE lastName = ?",
                    ionSys.newString("Doe"));
            IonStruct person = (IonStruct) result.iterator().next();
            System.out.println(person.get("firstName")); // prints John
            System.out.println(person.get("lastName")); // prints Doe
            System.out.println(person.get("age")); // prints 42
        });
    }
}
```

Maven 또는 IDE를 사용하여 `App.java` 파일을 컴파일하고 실행합니다.

Java용 Amazon QLDB 드라이버 - Cookbook 참조

이 참조 가이드는 Java용 Amazon QLDB 드라이버의 일반적인 사용 사례를 보여줍니다. 이 Java 코드 예제는 드라이버를 사용하여 기본 CRUD(생성, 읽기, 업데이트, 삭제) 작업을 수행하는 방법을 보여줍니다. 또한 Amazon Ion 데이터를 처리하기 위한 코드 예제도 포함되어 있습니다. 또한 이 가이드에서는 트랜잭션에 멱등성을 부여하고 고유성 제약하는 모범 사례를 중점적으로 설명합니다.

> **Note**
>
> 해당하는 경우, 일부 사용 사례에서는 Java용 QLDB 드라이버의 지원되는 각 주요 버전마다 다른 코드 예제가 있습니다.

목차

드라이버 가져오기

다음 코드 예제는 드라이버, QLDB 세션 클라이언트, Amazon Ion 패키지 및 기타 관련 종속 항목을 가져옵니다.

2.x

```
import com.amazon.ion.IonStruct;
```

```
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;

import software.amazon.awssdk.services.qldbsession.QldbSessionClient;
import software.amazon.qldb.QldbDriver;
import software.amazon.qldb.Result;
```

1.x

```
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;

import com.amazonaws.services.qldbsession.AmazonQLDBSessionClientBuilder;
import software.amazon.qldb.PooledQldbDriver;
import software.amazon.qldb.Result;
```

드라이버 인스턴스화

다음 코드 예제는 지정된 원장 이름에 연결되는 드라이버 인스턴스를 만들고 사용자 지정 재시도 제한이 있는 지정된 재시도 로직을 사용합니다.

> **Note**
>
> 또한 이 예제는 Amazon Ion 시스템 객체(`IonSystem`)를 인스턴스화합니다. 이 참조에서 일부 데이터 작업을 실행할 때 Ion 데이터를 처리하려면 이 객체가 필요합니다. 자세한 내용은 Amazon Ion 작업 섹션을 참조하세요.

2.x

```
QldbDriver qldbDriver = QldbDriver.builder()
    .ledger("vehicle-registration")
    .transactionRetryPolicy(RetryPolicy
        .builder()
        .maxRetries(3)
        .build())
    .sessionClientBuilder(QldbSessionClient.builder())
    .build();
```

```
IonSystem SYSTEM = IonSystemBuilder.standard().build();
```

1.x

```
PooledQldbDriver qldbDriver = PooledQldbDriver.builder()
    .withLedger("vehicle-registration")
    .withRetryLimit(3)
    .withSessionClientBuilder(AmazonQLDBSessionClientBuilder.standard())
    .build();

IonSystem SYSTEM = IonSystemBuilder.standard().build();
```

CRUD 작업

QLDB는 트랜잭션의 일부로 CRUD(생성, 읽기, 업데이트, 삭제) 작업을 실행합니다.

> **Warning**
>
> 가장 좋은 방법은 쓰기 트랜잭션이 완전한 멱등성을 부여하는 것입니다.

트랜잭션에 멱등성 부여하기

재시도 시 예상치 못한 부작용이 발생하지 않도록 쓰기 트랜잭션에 멱등성을 부여하는 것이 좋습니다. 여러 번 실행하여 매번 동일한 결과를 생성할 수 있는 트랜잭션은 멱등성을 가집니다.

이름이 `Person`인 테이블에 문서를 삽입하는 트랜잭션을 예로 들어 보겠습니다. 트랜잭션은 먼저 문서가 테이블에 이미 존재하는지 여부를 확인해야 합니다. 이렇게 확인하지 않으면 테이블에 문서가 중복될 수 있습니다.

QLDB가 서버 측에서 트랜잭션을 성공적으로 커밋했지만 응답을 기다리는 동안 클라이언트 제한 시간이 초과되었다고 가정해 보겠습니다. 트랜잭션이 멱등성을 가지지 않는 경우 재시도 시 동일한 문서가 두 번 이상 삽입될 수 있습니다.

인덱스를 사용하여 전체 테이블 스캔 방지

인덱싱된 필드 또는 문서 ID(예: `WHERE indexedField = 123` 또는 `WHERE indexedField IN (456, 789)`)에서 동등 연산자를 사용하여 `WHERE` 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 이 인덱싱된 조회가 없으면 QLDB는 테이블 스캔을 수행해야 하며, 이로 인해 트랜잭션 제한 시간이 초과되거나 OCC(낙관적 동시성 제어) 충돌이 발생할 수 있습니다.

OCC에 대한 자세한 내용은 Amazon QLDB 동시성 모델 단원을 참조하세요.

암시적으로 생성된 트랜잭션

QldbDriver.execute 메서드는 Executor 인스턴스를 수신하는 Lambda 함수를 받아들이며, 이 함수를 사용하여 명령문을 실행할 수 있습니다. `Executor` 인스턴스는 암시적으로 생성된 트랜잭션을 래핑합니다.

`Executor.execute` 메서드를 사용하여 Lambda 함수 내에서 명령문을 실행할 수 있습니다. 드라이버는 Lambda 함수가 반환될 때 트랜잭션을 암시적으로 커밋합니다.

다음 섹션에서는 기본 CRUD 작업을 실행하고, 사용자 지정 재시도 로직을 지정하고, 고유성 제약 조건을 구현하는 방법을 보여줍니다.

> **Note**
>
> 해당하는 경우 이 섹션에서는 내장된 Ion 라이브러리와 Jackson Ion 매퍼 라이브러리를 모두 사용하여 Amazon Ion 데이터를 처리하는 코드 예제를 제공합니다. 자세한 내용은 Amazon Ion 작업 섹션을 참조하세요.

목차

테이블 생성

```
qldbDriver.execute(txn -> {
```

```
    txn.execute("CREATE TABLE Person");
});
```

인덱스 생성

```
qldbDriver.execute(txn -> {
    txn.execute("CREATE INDEX ON Person(GovId)");
});
```

문서 읽기

```
// Assumes that Person table has documents as follows:
// { GovId: "TOYENC486FH", FirstName: "Brent" }

qldbDriver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM Person WHERE GovId = 'TOYENC486FH'");
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("GovId")); // prints TOYENC486FH
    System.out.println(person.get("FirstName")); // prints Brent
});
```

쿼리 파라미터 사용

다음 코드 예제는 Ion 유형 쿼리 파라미터를 사용합니다.

```
qldbDriver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM Person WHERE GovId = ?",
            SYSTEM.newString("TOYENC486FH"));
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("GovId")); // prints TOYENC486FH
    System.out.println(person.get("FirstName")); // prints Brent
});
```

다음 코드 예제는 여러 쿼리 파라미터를 사용합니다.

```
qldbDriver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM Person WHERE GovId = ? AND FirstName
 = ?",
            SYSTEM.newString("TOYENC486FH"),
            SYSTEM.newString("Brent"));
    IonStruct person = (IonStruct) result.iterator().next();
```

```
    System.out.println(person.get("GovId")); // prints TOYENC486FH
    System.out.println(person.get("FirstName")); // prints Brent
});
```

다음 코드 예제는 쿼리 파라미터 목록을 사용합니다.

```
qldbDriver.execute(txn -> {
    final List<IonValue> parameters = new ArrayList<>();
    parameters.add(SYSTEM.newString("TOYENC486FH"));
    parameters.add(SYSTEM.newString("ROEE1"));
    parameters.add(SYSTEM.newString("YH844"));
    Result result = txn.execute("SELECT * FROM Person WHERE GovId IN (?,?,?)",
 parameters);
    IonStruct person = (IonStruct) result.iterator().next();
    System.out.println(person.get("GovId")); // prints TOYENC486FH
    System.out.println(person.get("FirstName")); // prints Brent
});
```

Jackson 매퍼 사용

```
// Assumes that Person table has documents as follows:
// {GovId: "TOYENC486FH", FirstName: "Brent" }

qldbDriver.execute(txn -> {
    try {
        Result result = txn.execute("SELECT * FROM Person WHERE GovId =
 'TOYENC486FH'");
        Person person = MAPPER.readValue(result.iterator().next(), Person.class);
        System.out.println(person.getFirstName()); // prints Brent
        System.out.println(person.getGovId()); // prints TOYENC486FH
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

쿼리 파라미터 사용

다음 코드 예제는 Ion 유형 쿼리 파라미터를 사용합니다.

```
qldbDriver.execute(txn -> {
    try {
        Result result = txn.execute("SELECT * FROM Person WHERE GovId = ?",
```

```
                MAPPER.writeValueAsIonValue("TOYENC486FH"));
        Person person = MAPPER.readValue(result.iterator().next(), Person.class);
        System.out.println(person.getFirstName()); // prints Brent
        System.out.println(person.getGovId()); // prints TOYENC486FH
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

다음 코드 예제는 여러 쿼리 파라미터를 사용합니다.

```
qldbDriver.execute(txn -> {
    try {
        Result result = txn.execute("SELECT * FROM Person WHERE GovId = ? AND FirstName
 = ?",
                MAPPER.writeValueAsIonValue("TOYENC486FH"),
                MAPPER.writeValueAsIonValue("Brent"));
        Person person = MAPPER.readValue(result.iterator().next(), Person.class);
        System.out.println(person.getFirstName()); // prints Brent
        System.out.println(person.getGovId()); // prints TOYENC486FH
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

다음 코드 예제는 쿼리 파라미터 목록을 사용합니다.

```
qldbDriver.execute(txn -> {
    try {
        final List<IonValue> parameters = new ArrayList<>();
        parameters.add(MAPPER.writeValueAsIonValue("TOYENC486FH"));
        parameters.add(MAPPER.writeValueAsIonValue("ROEE1"));
        parameters.add(MAPPER.writeValueAsIonValue("YH844"));
        Result result = txn.execute("SELECT * FROM Person WHERE GovId IN (?,?,?)",
 parameters);
        Person person = MAPPER.readValue(result.iterator().next(), Person.class);
        System.out.println(person.getFirstName()); // prints Brent
        System.out.println(person.getGovId()); // prints TOYENC486FH
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

> **Note**
>
> 인덱싱된 조회 없이 쿼리를 실행하면 전체 테이블 스캔이 호출됩니다. 이 예제에서는 성능을 최적화하기 위해 GovId 필드에 인덱스를 사용하는 것이 좋습니다. GovId에 인덱스를 사용하지 않으면 쿼리에 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

문서 삽입하기

다음 코드 예제는 Ion 데이터 유형을 삽입합니다.

```
qldbDriver.execute(txn -> {
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    Result result = txn.execute("SELECT * FROM Person WHERE GovId = ?",
            SYSTEM.newString("TOYENC486FH"));
    // Check if there is a result
    if (!result.iterator().hasNext()) {
        IonStruct person = SYSTEM.newEmptyStruct();
        person.put("GovId").newString("TOYENC486FH");
        person.put("FirstName").newString("Brent");
        // Insert the document
        txn.execute("INSERT INTO Person ?", person);
    }
});
```

Jackson 매퍼 사용

다음 코드 예제는 Ion 데이터 유형을 삽입합니다.

```
qldbDriver.execute(txn -> {
    try {
        // Check if a document with GovId:TOYENC486FH exists
        // This is critical to make this transaction idempotent
        Result result = txn.execute("SELECT * FROM Person WHERE GovId = ?",
                MAPPER.writeValueAsIonValue("TOYENC486FH"));
        // Check if there is a result
        if (!result.iterator().hasNext()) {
            // Insert the document
            txn.execute("INSERT INTO Person ?",
```

```
                    MAPPER.writeValueAsIonValue(new Person("Brent", "TOYENC486FH")));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
});
```

이 트랜잭션은 문서를 Person 테이블에 삽입합니다. 삽입하기 전에 먼저 문서가 테이블에 이미 있는지 확인합니다. 이 검사를 통해 트랜잭션은 본질적으로 멱등성을 가지게 됩니다. 이 트랜잭션을 여러 번 실행하더라도 의도하지 않은 부작용이 발생하지는 않습니다.

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 GovId 필드에 인덱스를 사용하는 것이 좋습니다. GovId에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

하나의 명령문에 여러 문서 삽입

단일 INSERT 문을 사용하여 여러 문서를 삽입하려면 다음과 같이 IonList 타입의 파라미터(명시적으로 IonValue로 캐스팅됨)를 해당 문에 전달할 수 있습니다.

```
// people is an IonList explicitly cast as an IonValue
txn.execute("INSERT INTO People ?", (IonValue) people);
```

IonList를 전달할 때는 변수 자리 표시자(?)를 이중 꺾쇠 괄호(<<...>>)로 묶지 마세요. 수동 PartiQL 문에서 이중 꺾쇠 괄호는 백으로 알려진 정렬되지 않은 모음을 의미합니다.

명시적 캐스팅이 필요한 이유는 무엇인가요?

TransactionExecutor.execute 메서드가 오버로드되었습니다. 이는 다양한 수의 IonValue 인수(varargs) 또는 단일 List<IonValue> 인수를 받아들입니다. ion-java에서는 IonList는 List<IonValue>로 구현됩니다.

Java는 오버로드된 메서드를 호출할 때 기본적으로 가장 구체적인 메서드를 구현합니다. 이 경우 IonList 파라미터를 전달하면 List<IonValue>를 받는 메서드가 기본값으로 사용됩니다. 호출 시 이 메서드 구현은 목록의 IonValue 요소를 고유한 값으로 전달합니다. 따라서 varargs 메서드를 대신 호출하려면 IonList 파라미터를 IonValue으로 명시적으로 캐스팅해야 합니다.

문서 업데이트

```
qldbDriver.execute(txn -> {
    final List<IonValue> parameters = new ArrayList<>();
    parameters.add(SYSTEM.newString("John"));
    parameters.add(SYSTEM.newString("TOYENC486FH"));
    txn.execute("UPDATE Person SET FirstName = ? WHERE GovId = ?", parameters);
});
```

Jackson 매퍼 사용

```
qldbDriver.execute(txn -> {
    try {
        final List<IonValue> parameters = new ArrayList<>();
        parameters.add(MAPPER.writeValueAsIonValue("John"));
        parameters.add(MAPPER.writeValueAsIonValue("TOYENC486FH"));
        txn.execute("UPDATE Person SET FirstName = ? WHERE GovId = ?", parameters);
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 `GovId` 필드에 인덱스를 사용하는 것이 좋습니다. `GovId`에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

문서 삭제

```
qldbDriver.execute(txn -> {
    txn.execute("DELETE FROM Person WHERE GovId = ?",
            SYSTEM.newString("TOYENC486FH"));
});
```

Jackson 매퍼 사용

```
qldbDriver.execute(txn -> {
    try {
```

```
        txn.execute("DELETE FROM Person WHERE GovId = ?",
                MAPPER.writeValueAsIonValue("TOYENC486FH"));
    } catch (IOException e) {
        e.printStackTrace();
    }
});
```

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 GovId 필드에 인덱스를 사용하는 것이 좋습니다. GovId에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

하나의 트랜잭션에서 여러 명령문 실행

```
// This code snippet is intentionally trivial. In reality you wouldn't do this because
 you'd
// set your UPDATE to filter on vin and insured, and check if you updated something or
 not.
public static boolean InsureCar(QldbDriver qldbDriver, final String vin) {
    final IonSystem ionSystem = IonSystemBuilder.standard().build();
    final IonString ionVin = ionSystem.newString(vin);

    return qldbDriver.execute(txn -> {
        Result result = txn.execute(
                "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
                ionVin);
        if (!result.isEmpty()) {
            txn.execute("UPDATE Vehicles SET insured = TRUE WHERE vin = ?", ionVin);
            return true;
        }
        return false;
    });
}
```

재시도 로직

드라이버의 execute 메서드에는 재시도 가능한 예외(예: 시간 초과 또는 OCC 충돌)가 발생할 경우 트랜잭션을 재시도하는 재시도 메커니즘이 내장되어 있습니다.

2.x

최대 재시도 횟수와 백오프 전략을 구성할 수 있습니다.

기본 재시도 제한은 4이고 기본 백오프 전략은 DefaultQldbTransactionBackoffStrategy입니다. RetryPolicy 인스턴스를 사용하여 드라이버 인스턴스별 및 트랜잭션별 재시도 구성을 설정할 수 있습니다.

다음 코드 예제는 드라이버 인스턴스에 대한 사용자 지정 재시도 제한 및 사용자 지정 백오프 전략을 사용하여 재시도 로직을 지정합니다.

```
public void retry() {
    QldbDriver qldbDriver = QldbDriver.builder()
            .ledger("vehicle-registration")
            .transactionRetryPolicy(RetryPolicy.builder()
                    .maxRetries(2)
                    .backoffStrategy(new CustomBackOffStrategy()).build())
            .sessionClientBuilder(QldbSessionClient.builder())
            .build();
}

private class CustomBackOffStrategy implements BackoffStrategy {

    @Override
    public Duration calculateDelay(RetryPolicyContext retryPolicyContext) {
        return Duration.ofMillis(1000 * retryPolicyContext.retriesAttempted());
    }
}
```

다음 코드 예제는 특정 함수에 대한 사용자 지정 재시도 제한 및 사용자 지정 백오프 전략을 사용하여 재시도 로직을 지정합니다. `execute`에 대한 이 구성은 드라이버 인스턴스에 설정된 재시도 로직을 재정의합니다.

```
public void retry() {
    Result result = qldbDriver.execute(txn -> { txn.execute("SELECT * FROM Person
 WHERE GovId = ?",
            SYSTEM.newString("TOYENC486FH")); },
            RetryPolicy.builder()
                    .maxRetries(2)
                    .backoffStrategy(new CustomBackOffStrategy())
                    .build());
}
```

```
private class CustomBackOffStrategy implements BackoffStrategy {

    // Configuring a custom backoff which increases delay by 1s for each attempt.
    @Override
    public Duration calculateDelay(RetryPolicyContext retryPolicyContext) {
        return Duration.ofMillis(1000 * retryPolicyContext.retriesAttempted());
    }
}
```

1.x

최대 재시도 횟수는 구성 가능합니다. PooledQldbDriver를 초기화할 때 retryLimit 속성을 설정하여 재시도 제한을 구성할 수 있습니다.

기본 재시도 한도는 4입니다.

고유성 제약 조건 구현

QLDB는 고유 인덱스를 지원하지 않지만 애플리케이션에서 이 동작을 구현할 수 있습니다.

Person 테이블의 GovId 필드에 고유성 제약 조건을 구현하려고 한다고 가정해 보겠습니다. 이렇게 하면 다음 작업을 수행하는 트랜잭션을 작성합니다.

1. 테이블에 지정된 GovId가 있는 기존 문서가 없는지 확인합니다.
2. 어설션이 통과하면 문서를 삽입합니다.

경쟁 트랜잭션이 어설션을 동시에 통과하면 트랜잭션 중 하나만 성공적으로 커밋됩니다. 다른 트랜잭션은 OCC 충돌 예외가 발생하여 실패합니다.

다음 코드 예제는 이 고유성 제약 조건 구현 방법을 보여줍니다.

```
qldbDriver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM Person WHERE GovId = ?",
            SYSTEM.newString("TOYENC486FH"));
    // Check if there is a result
    if (!result.iterator().hasNext()) {
        IonStruct person = SYSTEM.newEmptyStruct();
        person.put("GovId").newString("TOYENC486FH");
        person.put("FirstName").newString("Brent");
        // Insert the document
```

```
        txn.execute("INSERT INTO Person ?", person);
    }
});
```

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 `GovId` 필드에 인덱스를 사용하는 것이 좋습니다. `GovId`에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

Amazon Ion 작업

QLDB에서 Amazon Ion 데이터를 처리하는 방법은 여러 가지가 있습니다. Ion 라이브러리의 기본 제공 메서드를 사용하여 필요에 따라 유연하게 문서를 만들고 수정할 수 있습니다. 또는 FasterXML의 Ion용 Jackson 데이터 형식 모듈을 사용하여 Ion 문서를 POJO(Plain Old Java Object) 모델에 매핑할 수 있습니다.

다음 섹션에서는 두 기술을 모두 사용하여 Ion 데이터를 처리하는 코드 예제를 제공합니다.

목차

Ion 패키지 가져오기

아티팩트 ion-java를 Java 프로젝트에 종속 항목으로 추가합니다.

Gradle

```
dependencies {
    compile group: 'com.amazon.ion', name: 'ion-java', version: '1.6.1'
}
```

Maven

```
<dependencies>
```

```
  <dependency>
    <groupId>com.amazon.ion</groupId>
    <artifactId>ion-java</artifactId>
    <version>1.6.1</version>
  </dependency>
</dependencies>
```

다음 Ion 패키지를 가져옵니다.

```
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSystem;
import com.amazon.ion.system.IonSystemBuilder;
```

Jackson 매퍼 사용

Java 프로젝트에 아티팩트 jackson-dataformat-ion을 종속 항목으로 추가합니다. QLDB에는 버전 `2.10.0` 이상이 필요합니다.

Gradle

```
dependencies {
    compile group: 'com.fasterxml.jackson.dataformat', name: 'jackson-dataformat-
ion', version: '2.10.0'
}
```

Maven

```
<dependencies>
  <dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-ion</artifactId>
    <version>2.10.0</version>
  </dependency>
</dependencies>
```

다음 Ion 패키지를 가져옵니다.

```
import com.amazon.ion.IonReader;
import com.amazon.ion.IonStruct;
```

```
import com.amazon.ion.system.IonReaderBuilder;
import com.amazon.ion.system.IonSystemBuilder;

import com.fasterxml.jackson.dataformat.ion.IonObjectMapper;
import com.fasterxml.jackson.dataformat.ion.ionvalue.IonValueMapper;
```

Ion 초기화

```
IonSystem SYSTEM = IonSystemBuilder.standard().build();
```

Jackson 매퍼 사용

```
IonObjectMapper MAPPER = new IonValueMapper(IonSystemBuilder.standard().build());
```

Ion 객체 생성

다음 코드 예제는 `IonStruct` 인터페이스와 내장 메서드를 사용하여 Ion 객체를 만듭니다.

```
IonStruct ionStruct = SYSTEM.newEmptyStruct();

ionStruct.put("GovId").newString("TOYENC486FH");
ionStruct.put("FirstName").newString("Brent");

System.out.println(ionStruct.toPrettyString()); // prints a nicely formatted copy of
 ionStruct
```

Jackson 매퍼 사용

다음과 같이 `Person`라는 명칭의 JSON 매핑된 모델 클래스가 있다고 가정하겠습니다.

```
import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

public static class Person {
    private final String firstName;
    private final String govId;

    @JsonCreator
    public Person(@JsonProperty("FirstName") final String firstName,
                  @JsonProperty("GovId") final String govId) {
        this.firstName = firstName;
```

```
        this.govId = govId;
    }

    @JsonProperty("FirstName")
    public String getFirstName() {
        return firstName;
    }

    @JsonProperty("GovId")
    public String getGovId() {
        return govId;
    }
}
```

다음 코드 예제는 Person의 인스턴스에서 IonStruct 객체를 만듭니다.

```
IonStruct ionStruct = (IonStruct) MAPPER.writeValueAsIonValue(new Person("Brent",
 "TOYENC486FH"));
```

Ion 객체 읽기

다음 코드 예제는 ionStruct 인스턴스의 각 필드를 인쇄합니다.

```
// ionStruct is an instance of IonStruct
System.out.println(ionStruct.get("GovId")); // prints TOYENC486FH
System.out.println(ionStruct.get("FirstName")); // prints Brent
```

Jackson 매퍼 사용

다음 코드 예제는 IonStruct 객체를 읽고 이를 Person의 인스턴스에 매핑합니다.

```
// ionStruct is an instance of IonStruct
IonReader reader = IonReaderBuilder.standard().build(ionStruct);
Person person = MAPPER.readValue(reader, Person.class);
System.out.println(person.getFirstName()); // prints Brent
System.out.println(person.getGovId()); // prints TOYENC486FH
```

Ion 작업에 대한 자세한 정보는 GitHub의 Amazon Ion 설명서를 참조하십시오. QLDB에서 Ion을 사용하는 방법에 대한 추가 코드 예제는 Amazon QLDB에서 Amazon Ion 데이터 타입을 사용한 작업 섹션을 참조하세요.

.NET용 Amazon QLDB 드라이버

원장의 데이터로 작업하려면 AWS에서 제공하는 드라이버를 사용하여 Microsoft .NET 애플리케이션에서 Amazon QLDB에 연결할 수 있습니다. 드라이버는 .NET Standard 2.0을 대상으로 합니다. 구체적으로, .NET Core (LTS) 2.1+ 및 .NET Framework 4.5.2+를 지원합니다. 호환성에 대한 자세한 내용은 Microsoft Docs 사이트의 .NET Standard을 참조하세요.

Amazon Ion 유형과 네이티브 C# 유형 간에 수동으로 변환할 필요가 없도록 하려면 Ion 객체 매퍼를 사용하는 것이 좋습니다.

다음 주제에서는 .NET용 QLDB 드라이버를 시작하는 방법에 대해 설명합니다.

주제

드라이버 리소스

.NET 드라이버에서 지원하는 기능에 대한 자세한 정보는 다음 리소스를 참조하십시오.

- API 참조
- 드라이버 소스 코드(GitHub)
- 샘플 애플리케이션 소스 코드(GitHub)
- Amazon Ion Cookbook
- Ion 객체 매퍼(GitHub)

필수 조건

.NET용 QLDB 드라이버를 시작하기 전에 다음을 수행해야 합니다.

1. Amazon QLDB 액세스의 AWS 설정 지침을 따르십시오. 다음 내용이 해당됩니다.
    1. AWS에 가입합니다.

   2. 적절한 QLDB 권한을 가진 사용자를 생성합니다.
   3. 개발을 위한 프로그래밍 방식 액세스 권한을 부여합니다.
2. Microsoft .NET 다운로드 사이트에서 .NET Core SDK 버전 2.1 이상을 다운로드하여 설치합니다.
3. (선택 사항) Visual Studio, Mac용 Visual Studio 또는 Visual Studio Code와 같은 통합 개발 환경(IDE)을 원하는 대로 설치합니다. 이러한 프로그램은 Microsoft Visual Studio 사이트에서 다운로드할 수 있습니다.
4. AWS SDK for .NET를 위한 개발 환경 구성:

   1. AWS 보안 인증을 설정합니다. 공유 보안 인증 파일을 생성할 것을 권장합니다.

      지침은 AWS SDK for .NET 개발자 안내서의 보안 인증 파일을 사용하여 AWS 보안 인증 구성을 참조하세요.
   2. 기본 AWS 리전을 설정하십시오. 방법을 알아보려면 AWS 리전 선택을 참조하십시오.

      사용 가능한 리전의 전체 목록은 AWS 일반 참조에서 Amazon QLDB 엔드포인트 및 할당량을 참조하십시오.

그런 다음 기본 샘플 애플리케이션을 설정하고 단축 코드 예제를 실행하거나 기존 .NET 프로젝트에 드라이버를 설치할 수 있습니다.

- 기존 프로젝트에 QLDB 드라이버와 AWS SDK for .NET를 설치하려면 설치로 이동하세요.
- 프로젝트를 설정하고 원장에 대한 기본 데이터 트랜잭션을 보여주는 단축 코드 예제를 실행하려면 빠른 시작 자습서를 참조하십시오.

설치

NuGet 패키지 관리자를 사용하여 .NET용 QLDB 드라이버를 설치합니다. 원하는 대로 Visual Studio 또는 IDE를 사용하여 프로젝트 종속 항목을 추가하는 것이 좋습니다. 드라이버 패키지 이름은 Amazon.QLDB.Driver입니다.

예를 들어 Visual Studio의 경우, 도구 메뉴에서 NuGet 패키지 관리자 콘솔을 엽니다. 그런 다음 PM> 프롬프트에 다음 명령을 입력합니다.

```
PM> Install-Package Amazon.QLDB.Driver
```

드라이버를 설치하면 AWS SDK for .NET 및 Amazon Ion 패키지를 포함한 해당 종속 항목도 설치됩니다.

Ion 객체 매퍼 설치

.NET용 QLDB 드라이버 버전 1.3.0에는 Amazon Ion을 사용할 필요 없이 네이티브 C# 데이터 유형을 수락하고 반환할 수 있는 지원이 도입되었습니다. 이 기능을 사용하려면 프로젝트에 다음 패키지를 추가하세요.

- Amazon.QLDB.Driver.Serialization - Ion 값을 C# POCO(Plain Old CLR Object)에 매핑하거나 그 반대로도 매핑할 수 있는 라이브러리입니다. 이 Ion 객체 매퍼를 사용하면 애플리케이션이 Ion을 사용할 필요 없이 네이티브 C# 데이터 유형과 직접 상호 작용할 수 있습니다. 이 라이브러리를 사용하는 방법에 대한 간단한 가이드는 GitHub 리포지토리 `awslabs/amazon-qldb-driver-dotnet`의 SERIALIZATION.md 파일을 참조하세요.

이 패키지를 설치하려면 다음 명령을 입력합니다.

```
PM> Install-Package Amazon.QLDB.Driver.Serialization
```

원장에서 기본 데이터 트랜잭션을 실행하는 방법에 대한 단축 코드 예제는 Cookbook 참조를 참조하십시오.

.NET용 Amazon QLDB 드라이버 — 빠른 시작 자습서

이 자습서에서는 .NET용 Amazon QLDB 드라이버를 사용하여 간단한 애플리케이션을 설정하는 방법을 알아봅니다. 이 안내서에는 드라이버 설치 단계 및 기본적인 CRUD(생성, 읽기, 업데이트 및 삭제) 작업에 대한 단축 코드 예제가 포함되어 있습니다.

주제

필수 조건

시작하기 전에 다음을 수행해야 합니다.

1. 아직 완료하지 않은 경우 .NET 드라이버에 대해 필수 조건를 완료합니다. 여기에는 AWS 등록, 개발을 위한 프로그래밍 방식 액세스 권한 부여, .NET Core SDK 설치가 포함됩니다.
2. `quick-start`라는 명칭의 원장을 생성합니다.

   원장 생성 방법을 알아보려면 콘솔 시작하기의 Amazon QLDB 원장의 기본 작업 또는 1단계: 새 원장 생성 섹션을 참조하세요.

1단계: 프로젝트 설정

먼저 .NET 프로젝트를 설정합니다.

1. 템플릿 애플리케이션을 생성하고 실행하려면 bash, PowerShell 또는 명령 프롬프트와 같은 터미널에 `dotnet` 명령을 입력합니다.

   ```
   $ dotnet new console --output Amazon.QLDB.QuickStartGuide
   $ dotnet run --project Amazon.QLDB.QuickStartGuide
   ```

   이 템플릿은 `Amazon.QLDB.QuickStartGuide`이라는 폴더를 생성합니다. 해당 폴더에 같은 이름의 프로젝트와 `Program.cs`이라는 이름의 파일이 생성됩니다. 프로그램에는 `Hello World!` 출력을 표시하는 코드가 포함되어 있습니다.

2. NuGet 패키지 관리자를 사용하여 .NET용 QLDB 드라이버를 설치합니다. 프로젝트에 종속성을 추가하려면 Visual Studio 또는 선택한 IDE를 사용하는 것이 좋습니다. 드라이버 패키지 이름은 Amazon.QLDB.Driver입니다.

   - 예를 들어 Visual Studio의 경우, 도구 메뉴에서 NuGet 패키지 관리자 콘솔을 엽니다. 그런 다음 `PM>` 프롬프트에 다음 명령을 입력합니다.

     ```
     PM> Install-Package Amazon.QLDB.Driver
     ```

   - 또는 터미널에서 다음 명령을 입력할 수 있습니다.

     ```
     $ cd Amazon.QLDB.QuickStartGuide
     $ dotnet add package Amazon.QLDB.Driver
     ```

드라이버를 설치하면 AWS SDK for .NET및 Amazon Ion 라이브러리를 비롯한 해당 종속 항목도 설치됩니다.

3. 드라이버의 직렬화 라이브러리를 설치합니다.

```
PM> Install-Package Amazon.QLDB.Driver.Serialization
```

4. `Program.cs` 파일을 엽니다.

그런 다음 다음 단계의 코드 예를 점진적으로 추가하여 몇 가지 기본 CRUD 작업을 시도해 보세요. 또는 단계별 자습서를 건너뛰고 전체 애플리케이션을 실행할 수도 있습니다.

> **Note**
>
> - 동기식 API와 비동기식 API 선택 - 드라이버는 동기식 및 비동기식 API를 제공합니다. 여러 요청을 차단하지 않고 처리하는 수요가 많은 애플리케이션의 경우, 성능 향상을 위해 비동기식 API를 사용하는 것이 좋습니다. 드라이버는 동기식으로 작성된 기존 코드 베이스의 편의성을 높이기 위해 동기식 API를 제공합니다.
>
> 이 자습서에는 동기식 및 비동기식 코드 예제가 모두 포함되어 있습니다. API에 대한 자세한 내용은 API 설명서의 IQldbDriver 및 IAsyncQldbDriver 인터페이스를 참조하십시오.
> - Amazon Ion 데이터 처리 — 이 자습서에서는 기본적으로 Ion 객체 매퍼를 사용하여 Amazon Ion 데이터를 처리하는 코드 예제를 제공합니다. QLDB는 .NET 드라이버 버전 1.3.0에서 Ion 객체 매퍼를 도입했습니다. 해당하는 경우, 이 자습서에서는 표준 Ion 라이브러리를 대안으로 사용하는 코드 예제도 제공합니다. 자세한 내용은 Amazon Ion 작업 섹션을 참조하세요.

2단계: 드라이버 초기화

`quick-start`라는 명칭의 원장에 연결되는 드라이버의 인스턴스를 초기화합니다. 다음 코드를 `Program.cs` 파일에 추가합니다.

Async

```
using Amazon.QLDB.Driver;
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;
```

```
namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        public class Person
        {
            public string FirstName { get; set; }

            public string LastName { get; set; }

            public int Age { get; set; }

            public override string ToString()
            {
                return FirstName + ", " + LastName + ", " + Age.ToString();
            }
        }

        static async Task Main(string[] args)
        {
            Console.WriteLine("Create the async QLDB driver");
            IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
                .WithLedger("quick-start")
                .WithSerializer(new ObjectSerializer())
                .Build();
        }
    }
}
```

Sync

```
using Amazon.QLDB.Driver;
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        public class Person
        {
            public string FirstName { get; set; }
```

```
            public string LastName { get; set; }

            public int Age { get; set; }

            public override string ToString()
            {
                return FirstName + ", " + LastName + ", " + Age.ToString();
            }
        }

        static void Main(string[] args)
        {
            Console.WriteLine("Create the sync QLDB driver");
            IQldbDriver driver = QldbDriver.Builder()
                .WithLedger("quick-start")
                .WithSerializer(new ObjectSerializer())
                .Build();
        }
    }
}
```

Ion 라이브러리 사용

Async

```
using System;
using System.Threading.Tasks;
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        static IValueFactory valueFactory = new ValueFactory();

        static async Task Main(string[] args)
        {
            Console.WriteLine("Create the async QLDB driver");
            IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
```

```
                .WithLedger("quick-start")
                .Build();
        }
    }
}
```

Sync

```
using System;
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;
using Amazon.QLDB.Driver;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        static IValueFactory valueFactory = new ValueFactory();

        static void Main(string[] args)
        {
            Console.WriteLine("Create the sync QLDB Driver");
            IQldbDriver driver = QldbDriver.Builder()
                .WithLedger("quick-start")
                .Build();
        }
    }
}
```

3단계: 테이블 및 인덱스 생성

이 자습서의 나머지 부분부터 6단계까지는 이전 코드 예제에 다음 코드 예제를 추가해야 합니다.

이 단계에서 다음 코드는 `CREATE TABLE` 및 `CREATE INDEX` 문을 실행하는 방법을 보여줍니다. `Person`라는 이름의 테이블과 해당 테이블의 `firstName` 필드에 대한 인덱스를 생성합니다. 인덱스는 쿼리 성능을 최적화하고 OCC(낙관적 동시성 제어) 충돌 예외를 제한하는 데 필요합니다.

Async

```
Console.WriteLine("Creating the table and index");
```

```
// Creates the table and the index in the same transaction.
// Note: Any code within the lambda can potentially execute multiple times due to
 retries.
// For more information, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-retry-policy
await driver.Execute(async txn =>
{
    await txn.Execute("CREATE TABLE Person");
    await txn.Execute("CREATE INDEX ON Person(firstName)");
});
```

Sync

```
Console.WriteLine("Creating the tables and index");

// Creates the table and the index in the same transaction.
// Note: Any code within the lambda can potentially execute multiple times due to
 retries.
// For more information, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-retry-policy
driver.Execute(txn =>
{
    txn.Execute("CREATE TABLE Person");
    txn.Execute("CREATE INDEX ON Person(firstName)");
});
```

4단계: 문서 삽입

다음 코드 예에서는 INSERT 문을 실행하는 방법을 보여줍니다. QLDB는 PartiQL 쿼리 언어(SQL 호환) 및 Amazon Ion 데이터 형식(JSON의 상위 집합)을 지원합니다.

Person 테이블에 문서를 삽입하는 다음 코드를 추가합니다.

Async

```
Console.WriteLine("Inserting a document");

Person myPerson = new Person {
    FirstName = "John",
    LastName = "Doe",
    Age = 32
};
```

```
await driver.Execute(async txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("INSERT INTO Person ?", myPerson);
    await txn.Execute(myQuery);
});
```

Sync

```
Console.WriteLine("Inserting a document");

Person myPerson = new Person {
    FirstName = "John",
    LastName = "Doe",
    Age = 32
};

driver.Execute(txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("INSERT INTO Person ?", myPerson);
    txn.Execute(myQuery);
});
```

Ion 라이브러리 사용

Async

```
Console.WriteLine("Inserting a document");

// This is one way of creating Ion values. We can also use an IonLoader.
// For more details, see: https://docs.aws.amazon.com/qldb/latest/developerguide/
driver-cookbook-dotnet.html#cookbook-dotnet.ion
IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("firstName", valueFactory.NewString("John"));
ionPerson.SetField("lastName", valueFactory.NewString("Doe"));
ionPerson.SetField("age", valueFactory.NewInt(32));

await driver.Execute(async txn =>
{
    await txn.Execute("INSERT INTO Person ?", ionPerson);
});
```

Sync

```
Console.WriteLine("Inserting a document");

// This is one way of creating Ion values, we can also use an IonLoader.
// For more details, see: https://docs.aws.amazon.com/qldb/latest/developerguide/
driver-cookbook-dotnet.html#cookbook-dotnet.ion
IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("firstName", valueFactory.NewString("John"));
ionPerson.SetField("lastName", valueFactory.NewString("Doe"));
ionPerson.SetField("age", valueFactory.NewInt(32));

driver.Execute(txn =>
{
    txn.Execute("INSERT INTO Person ?", ionPerson);
});
```

Tip

단일 INSERT 문을 사용하여 여러 문서를 삽입하려면 다음과 같이 Ion 목록 타입 파라미터를 해당 에 전달할 수 있습니다.

```
// people is an Ion list
txn.Execute("INSERT INTO Person ?", people);
```

Ion 목록을 전달할 때는 변수 자리 표시자(?)를 이중 꺾쇠 괄호(<<...>>)로 묶지 마십시오. 수동 PartiQL 문에서 이중 꺾쇠 괄호는 백으로 알려진 정렬되지 않은 모음을 의미합니다.

5단계: 문서 쿼리

다음 코드 예에서는 SELECT 문을 실행하는 방법을 보여줍니다.

Person 테이블에서 문서를 쿼리하는 다음 코드를 추가합니다.

Async

```
Console.WriteLine("Querying the table");

// The result from driver.Execute() is buffered into memory because once the
```

```
// transaction is committed, streaming the result is no longer possible.
IAsyncResult<Person> selectResult = await driver.Execute(async txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person WHERE FirstName
 = ?", "John");
    return await txn.Execute(myQuery);
});

await foreach (Person person in selectResult)
{
    Console.WriteLine(person);
    // John, Doe, 32
}
```

Sync

```
Console.WriteLine("Querying the table");

// The result from driver.Execute() is buffered into memory because once the
// transaction is committed, streaming the result is no longer possible.
IResult<Person> selectResult = driver.Execute(txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person WHERE FirstName
 = ?", "John");
    return txn.Execute(myQuery);
});

foreach (Person person in selectResult)
{
    Console.WriteLine(person);
    // John, Doe, 32
}
```

Ion 라이브러리 사용

Async

```
Console.WriteLine("Querying the table");

IIonValue ionFirstName = valueFactory.NewString("John");

// The result from driver.Execute() is buffered into memory because once the
```

```
// transaction is committed, streaming the result is no longer possible.
IAsyncResult selectResult = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE firstName = ?",
 ionFirstName);
});

await foreach (IIonValue row in selectResult)
{
    Console.WriteLine(row.GetField("firstName").StringValue);
    Console.WriteLine(row.GetField("lastName").StringValue);
    Console.WriteLine(row.GetField("age").IntValue);
}
```

Sync

```
Console.WriteLine("Querying the table");

IIonValue ionFirstName = valueFactory.NewString("John");

// The result from driver.Execute() is buffered into memory because once the
// transaction is committed, streaming the result is no longer possible.
IResult selectResult = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE firstName = ?", ionFirstName);
});

foreach (IIonValue row in selectResult)
{
    Console.WriteLine(row.GetField("firstName").StringValue);
    Console.WriteLine(row.GetField("lastName").StringValue);
    Console.WriteLine(row.GetField("age").IntValue);
}
```

이 예에서는 물음표(?)를 변수 자리 표시자로 사용하여 문서 정보를 해당 문에 전달합니다. 자리 표시자를 사용할 때는 `IonValue` 타입의 값을 전달해야 합니다.

6단계: 문서 업데이트

다음 코드 예에서는 `UPDATE` 문을 실행하는 방법을 보여줍니다.

1. `age`를 42로 업데이트하여 `Person` 테이블의 문서를 업데이트하는 다음 코드를 추가합니다.

Async

```
Console.WriteLine("Updating the document");

await driver.Execute(async txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("UPDATE Person SET Age = ? WHERE
 FirstName = ?", 42, "John");
    await txn.Execute(myQuery);
});
```

Sync

```
Console.WriteLine("Updating the document");

driver.Execute(txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("UPDATE Person SET Age = ? WHERE
 FirstName = ?", 42, "John");
    txn.Execute(myQuery);
});
```

2. 문서를 다시 쿼리하여 업데이트된 값을 확인합니다.

Async

```
Console.WriteLine("Querying the table for the updated document");

IAsyncResult<Person> updateResult = await driver.Execute(async txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person WHERE
 FirstName = ?", "John");
    return await txn.Execute(myQuery);
});

await foreach (Person person in updateResult)
{
    Console.WriteLine(person);
    // John, Doe, 42
}
```

Sync

```
Console.WriteLine("Querying the table for the updated document");

IResult<Person> updateResult = driver.Execute(txn =>
{
    IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person WHERE
 FirstName = ?", "John");
    return txn.Execute(myQuery);
});

foreach (Person person in updateResult)
{
    Console.WriteLine(person);
    // John, Doe, 42
}
```

3. 애플리케이션을 실행하려면 Amazon.QLDB.QuickStartGuide 프로젝트 디렉터리의 상위 디렉터리에서 다음 명령을 입력합니다.

```
$ dotnet run --project Amazon.QLDB.QuickStartGuide
```

Ion 라이브러리 사용

1. age를 42로 업데이트하여 Person 테이블의 문서를 업데이트하는 다음 코드를 추가합니다.

Async

```
Console.WriteLine("Updating the document");

IIonValue ionIntAge = valueFactory.NewInt(42);
IIonValue ionFirstName2 = valueFactory.NewString("John");

await driver.Execute(async txn =>
{
    await txn.Execute("UPDATE Person SET age = ? WHERE firstName = ?",
 ionIntAge, ionFirstName2);
});
```

Sync

```
Console.WriteLine("Updating a document");

IIonValue ionIntAge = valueFactory.NewInt(42);
IIonValue ionFirstName2 = valueFactory.NewString("John");

driver.Execute(txn =>
{
    txn.Execute("UPDATE Person SET age = ? WHERE firstName = ?", ionIntAge,
 ionFirstName2);
});
```

2. 문서를 다시 쿼리하여 업데이트된 값을 확인합니다.

Async

```
Console.WriteLine("Querying the table for the updated document");

IIonValue ionFirstName3 = valueFactory.NewString("John");

IAsyncResult updateResult = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE firstName = ?",
 ionFirstName3 );
});

await foreach (IIonValue row in updateResult)
{
    Console.WriteLine(row.GetField("firstName").StringValue);
    Console.WriteLine(row.GetField("lastName").StringValue);
    Console.WriteLine(row.GetField("age").IntValue);
}
```

Sync

```
Console.WriteLine("Querying the table for the updated document");

IIonValue ionFirstName3 = valueFactory.NewString("John");

IResult updateResult = driver.Execute(txn =>
{
```

```
    return txn.Execute("SELECT * FROM Person WHERE firstName = ?",
 ionFirstName3);
});

foreach (IIonValue row in updateResult)
{
    Console.WriteLine(row.GetField("firstName").StringValue);
    Console.WriteLine(row.GetField("lastName").StringValue);
    Console.WriteLine(row.GetField("age").IntValue);
}
```

3. 애플리케이션을 실행하려면 `Amazon.QLDB.QuickStartGuide` 프로젝트 디렉터리의 상위 디렉터리에서 다음 명령을 입력합니다.

```
$ dotnet run --project Amazon.QLDB.QuickStartGuide
```

전체 애플리케이션 실행

다음 코드 예는 `Program.cs` 애플리케이션의 전체 버전입니다. 이전 단계를 개별적으로 수행하는 대신 이 코드 예를 처음부터 끝까지 복사하여 실행할 수도 있습니다. 이 애플리케이션은 `quick-start`이라는 명칭의 원장에 대한 몇 가지 기본 CRUD 작업을 보여줍니다.

> **Note**
>
> 이 코드를 실행하기 전에 `quick-start` 원장에 `Person`이라는 명칭의 활성 테이블이 아직 없는지 확인하세요.

Async

```
using Amazon.QLDB.Driver;
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        public class Person
        {
            public string FirstName { get; set; }
```

```
            public string LastName { get; set; }

            public int Age { get; set; }

            public override string ToString()
            {
                return FirstName + ", " + LastName + ", " + Age.ToString();
            }
        }

        static async Task Main(string[] args)
        {
            Console.WriteLine("Create the async QLDB driver");
            IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
                .WithLedger("quick-start")
                .WithSerializer(new ObjectSerializer())
                .Build();

            Console.WriteLine("Creating the table and index");

            // Creates the table and the index in the same transaction.
            // Note: Any code within the lambda can potentially execute multiple
 times due to retries.
            // For more information, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-retry-policy
            await driver.Execute(async txn =>
            {
                await txn.Execute("CREATE TABLE Person");
                await txn.Execute("CREATE INDEX ON Person(firstName)");
            });

            Console.WriteLine("Inserting a document");

            Person myPerson = new Person {
                FirstName = "John",
                LastName = "Doe",
                Age = 32
            };

            await driver.Execute(async txn =>
            {
                IQuery<Person> myQuery = txn.Query<Person>("INSERT INTO Person ?",
 myPerson);
```

```
                await txn.Execute(myQuery);
            });

            Console.WriteLine("Querying the table");

            // The result from driver.Execute() is buffered into memory because once
the
            // transaction is committed, streaming the result is no longer possible.
            IAsyncResult<Person> selectResult = await driver.Execute(async txn =>
            {
                IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person
WHERE FirstName = ?", "John");
                return await txn.Execute(myQuery);
            });

            await foreach (Person person in selectResult)
            {
                Console.WriteLine(person);
                // John, Doe, 32
            }

            Console.WriteLine("Updating the document");

            await driver.Execute(async txn =>
            {
                IQuery<Person> myQuery = txn.Query<Person>("UPDATE Person SET Age
= ? WHERE FirstName = ?", 42, "John");
                await txn.Execute(myQuery);
            });

            Console.WriteLine("Querying the table for the updated document");

            IAsyncResult<Person> updateResult = await driver.Execute(async txn =>
            {
                IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person
WHERE FirstName = ?", "John");
                return await txn.Execute(myQuery);
            });

            await foreach (Person person in updateResult)
            {
                Console.WriteLine(person);
                // John, Doe, 42
            }
```

```
        }
    }
}
```

Sync

```
using Amazon.QLDB.Driver;
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        public class Person
        {
            public string FirstName { get; set; }

            public string LastName { get; set; }

            public int Age { get; set; }

            public override string ToString()
            {
                return FirstName + ", " + LastName + ", " + Age.ToString();
            }
        }

        static void Main(string[] args)
        {
            Console.WriteLine("Create the sync QLDB driver");
            IQldbDriver driver = QldbDriver.Builder()
                .WithLedger("quick-start")
                .WithSerializer(new ObjectSerializer())
                .Build();

            Console.WriteLine("Creating the table and index");

            // Creates the table and the index in the same transaction.
            // Note: Any code within the lambda can potentially execute multiple
 times due to retries.
            // For more information, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-retry-policy
```

```
            driver.Execute(txn =>
            {
                txn.Execute("CREATE TABLE Person");
                txn.Execute("CREATE INDEX ON Person(firstName)");
            });

            Console.WriteLine("Inserting a document");

            Person myPerson = new Person {
                FirstName = "John",
                LastName = "Doe",
                Age = 32
            };

            driver.Execute(txn =>
            {
                IQuery<Person> myQuery = txn.Query<Person>("INSERT INTO Person ?",
myPerson);
                txn.Execute(myQuery);
            });

            Console.WriteLine("Querying the table");

            // The result from driver.Execute() is buffered into memory because once
the
            // transaction is committed, streaming the result is no longer possible.
            IResult<Person> selectResult = driver.Execute(txn =>
            {
                IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person
WHERE FirstName = ?", "John");
                return txn.Execute(myQuery);
            });

            foreach (Person person in selectResult)
            {
                Console.WriteLine(person);
                // John, Doe, 32
            }

            Console.WriteLine("Updating the document");

            driver.Execute(txn =>
            {
```

```
                IQuery<Person> myQuery = txn.Query<Person>("UPDATE Person SET Age
 = ? WHERE FirstName = ?", 42, "John");
                txn.Execute(myQuery);
            });

            Console.WriteLine("Querying the table for the updated document");

            IResult<Person> updateResult = driver.Execute(txn =>
            {
                IQuery<Person> myQuery = txn.Query<Person>("SELECT * FROM Person
 WHERE FirstName = ?", "John");
                return txn.Execute(myQuery);
            });

            foreach (Person person in updateResult)
            {
                Console.WriteLine(person);
                // John, Doe, 42
            }

        }
    }
}
```

Ion 라이브러리 사용

Async

```
using System;
using System.Threading.Tasks;
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        static IValueFactory valueFactory = new ValueFactory();

        static async Task Main(string[] args)
```

```
        {
            Console.WriteLine("Create the async QLDB driver");
            IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
                .WithLedger("quick-start")
                .Build();

            Console.WriteLine("Creating the table and index");

            // Creates the table and the index in the same transaction.
            // Note: Any code within the lambda can potentially execute multiple
 times due to retries.
            // For more information, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-retry-policy
            await driver.Execute(async txn =>
            {
                await txn.Execute("CREATE TABLE Person");
                await txn.Execute("CREATE INDEX ON Person(firstName)");
            });

            Console.WriteLine("Inserting a document");

            // This is one way of creating Ion values. We can also use an IonLoader.
            // For more details, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-cookbook-dotnet.html#cookbook-dotnet.ion
            IIonValue ionPerson = valueFactory.NewEmptyStruct();
            ionPerson.SetField("firstName", valueFactory.NewString("John"));
            ionPerson.SetField("lastName", valueFactory.NewString("Doe"));
            ionPerson.SetField("age", valueFactory.NewInt(32));

            await driver.Execute(async txn =>
            {
                await txn.Execute("INSERT INTO Person ?", ionPerson);
            });

            Console.WriteLine("Querying the table");

            IIonValue ionFirstName = valueFactory.NewString("John");

            // The result from driver.Execute() is buffered into memory because once
 the
            // transaction is committed, streaming the result is no longer possible.
            IAsyncResult selectResult = await driver.Execute(async txn =>
            {
```

```
                return await txn.Execute("SELECT * FROM Person WHERE firstName = ?",
 ionFirstName);
            });

            await foreach (IIonValue row in selectResult)
            {
                Console.WriteLine(row.GetField("firstName").StringValue);
                Console.WriteLine(row.GetField("lastName").StringValue);
                Console.WriteLine(row.GetField("age").IntValue);
            }

            Console.WriteLine("Updating the document");

            IIonValue ionIntAge = valueFactory.NewInt(42);
            IIonValue ionFirstName2 = valueFactory.NewString("John");

            await driver.Execute(async txn =>
            {
                await txn.Execute("UPDATE Person SET age = ? WHERE firstName = ?",
 ionIntAge, ionFirstName2);
            });

            Console.WriteLine("Querying the table for the updated document");

            IIonValue ionFirstName3 = valueFactory.NewString("John");

            IAsyncResult updateResult = await driver.Execute(async txn =>
            {
                return await txn.Execute("SELECT * FROM Person WHERE firstName = ?",
 ionFirstName3);
            });

            await foreach (IIonValue row in updateResult)
            {
                Console.WriteLine(row.GetField("firstName").StringValue);
                Console.WriteLine(row.GetField("lastName").StringValue);
                Console.WriteLine(row.GetField("age").IntValue);
            }
        }
    }
}
```

Sync

```
using System;
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;
using Amazon.QLDB.Driver;

namespace Amazon.QLDB.QuickStartGuide
{
    class Program
    {
        static IValueFactory valueFactory = new ValueFactory();

        static void Main(string[] args)
        {
            Console.WriteLine("Create the sync QLDB Driver");
            IQldbDriver driver = QldbDriver.Builder()
                .WithLedger("quick-start")
                .Build();

            Console.WriteLine("Creating the tables and index");

            // Creates the table and the index in the same transaction.
            // Note: Any code within the lambda can potentially execute multiple
 times due to retries.
            // For more information, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-retry-policy
            driver.Execute(txn =>
            {
                txn.Execute("CREATE TABLE Person");
                txn.Execute("CREATE INDEX ON Person(firstName)");
            });

            Console.WriteLine("Inserting a document");

            // This is one way of creating Ion values. We can also use an IonLoader.
            // For more details, see: https://docs.aws.amazon.com/qldb/latest/
developerguide/driver-cookbook-dotnet.html#cookbook-dotnet.ion
            IIonValue ionPerson = valueFactory.NewEmptyStruct();
            ionPerson.SetField("firstName", valueFactory.NewString("John"));
            ionPerson.SetField("lastName", valueFactory.NewString("Doe"));
            ionPerson.SetField("age", valueFactory.NewInt(32));

            driver.Execute(txn =>
```

```
            {
                txn.Execute("INSERT INTO Person ?", ionPerson);
            });

            Console.WriteLine("Querying the table");

            IIonValue ionFirstName = valueFactory.NewString("John");

            // The result from driver.Execute() is buffered into memory because once
the
            // transaction is committed, streaming the result is no longer possible.
            IResult selectResult = driver.Execute(txn =>
            {
                return txn.Execute("SELECT * FROM Person WHERE firstName = ?",
ionFirstName);
            });

            foreach (IIonValue row in selectResult)
            {
                Console.WriteLine(row.GetField("firstName").StringValue);
                Console.WriteLine(row.GetField("lastName").StringValue);
                Console.WriteLine(row.GetField("age").IntValue);
            }

            Console.WriteLine("Updating a document");

            IIonValue ionIntAge = valueFactory.NewInt(42);
            IIonValue ionFirstName2 = valueFactory.NewString("John");

            driver.Execute(txn =>
            {
                txn.Execute("UPDATE Person SET age = ? WHERE firstName = ?",
ionIntAge, ionFirstName2);
            });

            Console.WriteLine("Querying the table for the updated document");

            IIonValue ionFirstName3 = valueFactory.NewString("John");

            IResult updateResult = driver.Execute(txn =>
            {
                return txn.Execute("SELECT * FROM Person WHERE firstName = ?",
ionFirstName3);
            });
```

```
            foreach (IIonValue row in updateResult)
            {
                Console.WriteLine(row.GetField("firstName").StringValue);
                Console.WriteLine(row.GetField("lastName").StringValue);
                Console.WriteLine(row.GetField("age").IntValue);
            }
        }
    }
}
```

전체 애플리케이션을 실행하려면 `Amazon.QLDB.QuickStartGuide` 프로젝트 디렉터리의 상위 디렉터리에서 다음 명령을 입력합니다.

```
$ dotnet run --project Amazon.QLDB.QuickStartGuide
```

.NET용 Amazon QLDB 드라이버 - Cookbook 참조

이 참조 가이드는 .NET용 Amazon QLDB 드라이버의 일반적인 사용 사례를 보여줍니다. 이 C# 코드 예제는 드라이버를 사용하여 기본 CRUD(생성, 읽기, 업데이트, 삭제) 작업을 실행하는 방법을 보여줍니다. 또한 Amazon Ion 데이터를 처리하기 위한 코드 예제도 포함되어 있습니다. 또한 이 가이드에서는 트랜잭션에 멱등성을 부여하고 고유성 제약하는 모범 사례를 중점적으로 설명합니다.

> **Note**
>
> 이 주제는 기본적으로 Ion 객체 매퍼를 사용하여 Amazon Ion 데이터를 처리하는 코드 예제를 제공합니다. QLDB는 .NET 드라이버 버전 1.3.0에서 Ion 객체 매퍼를 도입했습니다. 해당하는 경우, 이 주제에서는 표준 Ion 라이브러리를 대안으로 사용하는 코드 예제도 제공합니다. 자세한 내용은 Amazon Ion 작업 섹션을 참조하세요.

목차

드라이버 가져오기

다음 코드 예제에서는 드라이브를 가져옵니다.

```
using Amazon.QLDB.Driver;
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;
```

Ion 라이브러리 사용

```
using Amazon.QLDB.Driver;
using Amazon.IonDotnet.Builders;
```

드라이버 인스턴스화

다음 코드 예제는 기본 설정을 사용하여 지정된 원장 이름에 연결하는 드라이버 인스턴스를 만듭니다.

Async

```
IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
```

```
    .WithLedger("vehicle-registration")
    // Add Serialization library
    .WithSerializer(new ObjectSerializer())
    .Build();
```

Sync

```
IQldbDriver driver = QldbDriver.Builder()
    .WithLedger("vehicle-registration")
    // Add Serialization library
    .WithSerializer(new ObjectSerializer())
    .Build();
```

Ion 라이브러리 사용

Async

```
IAsyncQldbDriver driver = AsyncQldbDriver.Builder().WithLedger("vehicle-
registration").Build();
```

Sync

```
IQldbDriver driver = QldbDriver.Builder().WithLedger("vehicle-
registration").Build();
```

CRUD 작업

QLDB는 트랜잭션의 일부로 CRUD(생성, 읽기, 업데이트, 삭제) 작업을 실행합니다.

> **Warning**
>
> 가장 좋은 방법은 쓰기 트랜잭션이 완전한 멱등성을 부여하는 것입니다.

트랜잭션에 멱등성 부여하기

재시도 시 예상치 못한 부작용이 발생하지 않도록 쓰기 트랜잭션에 멱등성을 부여하는 것이 좋습니다. 여러 번 실행하여 매번 동일한 결과를 생성할 수 있는 트랜잭션은 멱등성을 가집니다.

이름이 Person인 테이블에 문서를 삽입하는 트랜잭션을 예로 들어 보겠습니다. 트랜잭션은 먼저 문서가 테이블에 이미 존재하는지 여부를 확인해야 합니다. 이렇게 확인하지 않으면 테이블에 문서가 중복될 수 있습니다.

QLDB가 서버 측에서 트랜잭션을 성공적으로 커밋했지만 응답을 기다리는 동안 클라이언트 제한 시간이 초과되었다고 가정해 보겠습니다. 트랜잭션이 멱등성을 가지지 않는 경우 재시도 시 동일한 문서가 두 번 이상 삽입될 수 있습니다.

인덱스를 사용하여 전체 테이블 스캔 방지

인덱싱된 필드 또는 문서 ID(예: `WHERE indexedField = 123` 또는 `WHERE indexedField IN (456, 789)`)에서 동등 연산자를 사용하여 `WHERE` 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 이 인덱싱된 조회가 없으면 QLDB는 테이블 스캔을 수행해야 하며, 이로 인해 트랜잭션 제한 시간이 초과되거나 OCC(낙관적 동시성 제어) 충돌이 발생할 수 있습니다.

OCC에 대한 자세한 내용은 Amazon QLDB 동시성 모델 단원을 참조하세요.

암시적으로 생성된 트랜잭션

Amazon.QLDB.Driver.IQldbDriver.Execute 메서드는 Amazon.QLDB.Driver.TransactionExecutor의 인스턴스를 수신하는 Lambda 함수를 허용하며, 이 인스턴스를 사용하여 명령문을 실행할 수 있습니다. `TransactionExecutor`의 인스턴스는 암시적으로 생성된 트랜잭션을 래핑합니다.

트랜잭션 실행자의 `Execute` 메서드를 사용하여 Lambda 함수 내에서 명령문을 실행할 수 있습니다. 드라이버는 Lambda 함수가 반환될 때 트랜잭션을 암시적으로 커밋합니다.

다음 섹션에서는 기본 CRUD 작업을 실행하고, 사용자 지정 재시도 로직을 지정하고, 고유성 제약 조건을 구현하는 방법을 보여줍니다.

목차

- 재시도 로직
- 고유성 제약 조건 구현

테이블 생성

Async

```
IAsyncResult<Table> createResult = await driver.Execute(async txn =>
{
    IQuery<Table> query = txn.Query<Table>("CREATE TABLE Person");
    return await txn.Execute(query);
});

await foreach (var result in createResult)
{
    Console.WriteLine("{ tableId: " + result.TableId + " }");
    // The statement returns the created table ID:
    // { tableId: 4o5Uk09OcjC6PpJpLahceE }
}
```

Sync

```
IResult<Table> createResult = driver.Execute( txn =>
{
    IQuery<Table> query = txn.Query<Table>("CREATE TABLE Person");
    return txn.Execute(query);
});

foreach (var result in createResult)
{
    Console.WriteLine("{ tableId: " + result.TableId + " }");
    // The statement returns the created table ID:
    // { tableId: 4o5Uk09OcjC6PpJpLahceE }
}
```

Ion 라이브러리 사용

Async

```
// The result from driver.Execute() is buffered into memory because once the
// transaction is committed, streaming the result is no longer possible.
```

```
IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("CREATE TABLE Person");
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the created table ID:
    // {
    //    tableId: "4o5Uk09OcjC6PpJpLahceE"
    // }
}
```

Sync

```
// The result from driver.Execute() is buffered into memory because once the
// transaction is committed, streaming the result is no longer possible.
IResult result = driver.Execute(txn =>
{
    return txn.Execute("CREATE TABLE Person");
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the created table ID:
    // {
    //    tableId: "4o5Uk09OcjC6PpJpLahceE"
    // }
}
```

인덱스 생성

Async

```
IAsyncResult<Table> createResult = await driver.Execute(async txn =>
{
    IQuery<Table> query = txn.Query<Table>("CREATE INDEX ON Person(firstName)");
    return await txn.Execute(query);
});
```

```
await foreach (var result in createResult)
{
    Console.WriteLine("{ tableId: " + result.TableId + " }");
    // The statement returns the updated table ID:
    // { tableId: 4o5Uk09OcjC6PpJpLahceE }
}
```

Sync

```
IResult<Table> createResult = driver.Execute(txn =>
{
    IQuery<Table> query = txn.Query<Table>("CREATE INDEX ON Person(firstName)");
    return txn.Execute(query);
});

foreach (var result in createResult)
{
    Console.WriteLine("{ tableId: " + result.TableId + " }");
    // The statement returns the updated table ID:
    // { tableId: 4o5Uk09OcjC6PpJpLahceE }
}
```

Ion 라이브러리 사용

Async

```
IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("CREATE INDEX ON Person(GovId)");
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the updated table ID:
    // {
    //    tableId: "4o5Uk09OcjC6PpJpLahceE"
    // }
}
```

Sync

```
IResult result = driver.Execute(txn =>
{
    return txn.Execute("CREATE INDEX ON Person(GovId)");
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the updated table ID:
    // {
    //    tableId: "4o5Uk09OcjC6PpJpLahceE"
    // }
}
```

문서 읽기

```
// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }
// Person class is defined as follows:
// public class Person
// {
//     public string GovId { get; set; }
//     public string FirstName { get; set; }
//  }

IAsyncResult<Person> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Person>("SELECT * FROM Person WHERE GovId =
 'TOYENC486FH'"));
});

await foreach (Person person in result)
{
    Console.WriteLine(person.GovId); // Prints TOYENC486FH.
    Console.WriteLine(person.FirstName); // Prints Brent.
}
```

> **Note**
>
> 인덱싱된 조회 없이 쿼리를 실행하면 전체 테이블 스캔이 호출됩니다. 이 예제에서는 성능을 최적화하기 위해 GovId 필드에 인덱스를 사용하는 것이 좋습니다. GovId에 인덱스를 사용하지 않으면 쿼리에 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

쿼리 파라미터 사용

다음 코드 예제에서는 C# 형식 쿼리 파라미터를 사용합니다.

```
IAsyncResult<Person> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Person>("SELECT * FROM Person WHERE FirstName
 = ?", "Brent"));
});

await foreach (Person person in result)
{
    Console.WriteLine(person.GovId); // Prints TOYENC486FH.
    Console.WriteLine(person.FirstName); // Prints Brent.
}
```

다음 코드 예제에서는 여러 C# 형식 쿼리 파라미터를 사용합니다.

```
IAsyncResult<Person> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Person>("SELECT * FROM Person WHERE GovId = ?
 AND FirstName = ?", "TOYENC486FH", "Brent"));
});

await foreach (Person person in result)
{
    Console.WriteLine(person.GovId); // Prints TOYENC486FH.
    Console.WriteLine(person.FirstName); // Prints Brent.
}
```

다음 코드 예제에서는 C# 형식 쿼리 파라미터의 배열을 사용합니다.

```
// Assumes that Person table has documents as follows:
```

```
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }
// { "GovId": "ROEE1C1AABH", "FirstName" : "Jim" }
// { "GovId": "YH844DA7LDB", "FirstName" : "Mary" }

string[] ids = {
    "TOYENC486FH",
    "ROEE1C1AABH",
    "YH844DA7LDB"
};

IAsyncResult<Person> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Person>("SELECT * FROM Person WHERE GovId IN
 (?,?,?)", ids));
});

await foreach (Person person in result)
{
    Console.WriteLine(person.FirstName); // Prints Brent on first iteration.
    // Prints Jim on second iteration.
    // Prints Mary on third iteration.
}
```

다음 코드 예제에서는 C# 목록을 값으로 사용합니다.

```
// Assumes that Person table has document as follows:
// { "GovId": "TOYENC486FH",
//   "FirstName" : "Brent",
//   "Vehicles": [
//      { "Make": "Volkswagen",
//        "Model": "Golf"},
//      { "Make": "Honda",
//        "Model": "Civic"}
//   ]
// }
// Person class is defined as follows:
// public class Person
// {
//     public string GovId { get; set; }
//     public string FirstName { get; set; }
//     public List<Vehicle> Vehicles { get; set; }
// }
// Vehicle class is defined as follows:
```

```
// public class Vehicle
// {
//     public string Make { get; set; }
//     public string Model { get; set; }
// }

List<Vehicle> vehicles = new List<Vehicle>
{
    new Vehicle
    {
        Make = "Volkswagen",
        Model = "Golf"
    },
    new Vehicle
    {
        Make = "Honda",
        Model = "Civic"
    }
};

IAsyncResult<Person> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Person>("SELECT * FROM Person WHERE Vehicles
 = ?", vehicles));
});

await foreach (Person person in result)
{
    Console.WriteLine("{");
    Console.WriteLine($"  GovId: {person.GovId},");
    Console.WriteLine($"  FirstName: {person.FirstName},");
    Console.WriteLine("  Vehicles: [");
    foreach (Vehicle vehicle in person.Vehicles)
    {
        Console.WriteLine("  {");
        Console.WriteLine($"    Make: {vehicle.Make},");
        Console.WriteLine($"    Model: {vehicle.Model},");
        Console.WriteLine("  },");
    }
    Console.WriteLine("  ]");
    Console.WriteLine("}");
    // Prints:
    // {
    //     GovId: TOYENC486FH,
```

```
    //     FirstName: Brent,
    //     Vehicles: [
    //     {
    //         Make: Volkswagen,
    //         Model: Golf
    //     },
    //     {
    //         Make: Honda,
    //         Model: Civic
    //     },
    //     ]
    // }
}
```

Ion 라이브러리 사용

Async

```
// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE GovId = 'TOYENC486FH'");
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

Sync

```
// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }

IResult result = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE GovId = 'TOYENC486FH'");
});

foreach (IIonValue row in result)
```

```
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

> **Note**
>
> 인덱싱된 조회 없이 쿼리를 실행하면 전체 테이블 스캔이 호출됩니다. 이 예제에서는 성능을 최적화하기 위해 GovId 필드에 인덱스를 사용하는 것이 좋습니다. GovId에 인덱스를 사용하지 않으면 쿼리에 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

다음 코드 예제는 Ion 유형 쿼리 파라미터를 사용합니다.

Async

```
IValueFactory valueFactory = new ValueFactory();
IIonValue ionFirstName = valueFactory.NewString("Brent");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE FirstName = ?",
 ionFirstName);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

Sync

```
IValueFactory valueFactory = new ValueFactory();
IIonValue ionFirstName = valueFactory.NewString("Brent");

IResult result = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE FirstName = ?", ionFirstName);
});
```

```
foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

다음 코드 예제는 여러 쿼리 파라미터를 사용합니다.

Async

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");
IIonValue ionFirstName = valueFactory.NewString("Brent");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE GovId = ? AND FirstName
 = ?", ionGovId, ionFirstName);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

Sync

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");
IIonValue ionFirstName = valueFactory.NewString("Brent");

IResult result = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE GovId = ? AND FirstName = ?",
 ionGovId, ionFirstName);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("GovId").StringValue); // Prints TOYENC486FH.
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent.
}
```

다음 코드 예제는 쿼리 파라미터 목록을 사용합니다.

Async

```
// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }
// { "GovId": "ROEE1C1AABH", "FirstName" : "Jim" }
// { "GovId": "YH844DA7LDB", "FirstName" : "Mary" }

IIonValue[] ionIds = {
    valueFactory.NewString("TOYENC486FH"),
    valueFactory.NewString("ROEE1C1AABH"),
    valueFactory.NewString("YH844DA7LDB")
};

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE GovId IN (?,?,?)", ionIds);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent on
 first iteration.
                                                              // Prints Jim on
 second iteration.
                                                              // Prints Mary on
 third iteration.
}
```

Sync

```
// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName" : "Brent" }
// { "GovId": "ROEE1C1AABH", "FirstName" : "Jim" }
// { "GovId": "YH844DA7LDB", "FirstName" : "Mary" }

IIonValue[] ionIds = {
    valueFactory.NewString("TOYENC486FH"),
    valueFactory.NewString("ROEE1C1AABH"),
    valueFactory.NewString("YH844DA7LDB")
};
```

```
IResult result = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE GovId IN (?,?,?)", ionIds);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.GetField("FirstName").StringValue); // Prints Brent on
 first iteration.
                                                              // Prints Jim on
 second iteration.
                                                              // Prints Mary on
 third iteration.
}
```

다음 코드 예제에서는 Ion 목록을 값으로 사용합니다. 다른 Ion 유형 사용에 더 알아보려면 Amazon QLDB에서 Amazon Ion 데이터 타입을 사용한 작업 단원을 참조하십시오.

Async

```
// Assumes that Person table has document as follows:
// { "GovId": "TOYENC486FH",
//   "FirstName" : "Brent",
//   "Vehicles": [
//      { "Make": "Volkswagen",
//        "Model": "Golf"},
//      { "Make": "Honda",
//        "Model": "Civic"}
//   ]
// }

IIonValue ionVehicle1 = valueFactory.NewEmptyStruct();
ionVehicle1.SetField("Make", valueFactory.NewString("Volkswagen"));
ionVehicle1.SetField("Model", valueFactory.NewString("Golf"));

IIonValue ionVehicle2 = valueFactory.NewEmptyStruct();
ionVehicle2.SetField("Make", valueFactory.NewString("Honda"));
ionVehicle2.SetField("Model", valueFactory.NewString("Civic"));

IIonValue ionVehicles =  valueFactory.NewEmptyList();
ionVehicles.Add(ionVehicle1);
ionVehicles.Add(ionVehicle2);
```

```
IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM Person WHERE Vehicles = ?",
 ionVehicles);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // Prints:
    // {
    //     GovId: "TOYENC486FN",
    //     FirstName: "Brent",
    //     Vehicles: [
    //     {
    //         Make: "Volkswagen",
    //         Model: "Golf"
    //     },
    //     {
    //         Make: "Honda",
    //         Model: "Civic"
    //     }
    //     ]
    // }
}
```

Sync

```
// Assumes that Person table has document as follows:
// { "GovId": "TOYENC486FH",
//   "FirstName" : "Brent",
//   "Vehicles": [
//      { "Make": "Volkswagen",
//        "Model": "Golf"},
//      { "Make": "Honda",
//        "Model": "Civic"}
//   ]
// }

IIonValue ionVehicle1 = valueFactory.NewEmptyStruct();
ionVehicle1.SetField("Make", valueFactory.NewString("Volkswagen"));
ionVehicle1.SetField("Model", valueFactory.NewString("Golf"));
```

```
IIonValue ionVehicle2 = valueFactory.NewEmptyStruct();
ionVehicle2.SetField("Make", valueFactory.NewString("Honda"));
ionVehicle2.SetField("Model", valueFactory.NewString("Civic"));

IIonValue ionVehicles =  valueFactory.NewEmptyList();
ionVehicles.Add(ionVehicle1);
ionVehicles.Add(ionVehicle2);

IResult result = driver.Execute(txn =>
{
    return txn.Execute("SELECT * FROM Person WHERE Vehicles = ?", ionVehicles);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // Prints:
    // {
    //     GovId: "TOYENC486FN",
    //     FirstName: "Brent",
    //     Vehicles: [
    //     {
    //         Make: "Volkswagen",
    //         Model: "Golf"
    //     },
    //     {
    //         Make: "Honda",
    //         Model: "Civic"
    //     }
    //     ]
    // }
}
```

문서 삽입하기

다음 코드 예제는 Ion 데이터 유형을 삽입합니다.

```
string govId = "TOYENC486FH";

Person person = new Person
{
    GovId = "TOYENC486FH",
```

```
    FirstName = "Brent"
};

await driver.Execute(async txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    IAsyncResult<Person> result = await txn.Execute(txn.Query<Person>("SELECT * FROM
 Person WHERE GovId = ?", govId));

    // Check if there is a record in the cursor.
    int count = await result.CountAsync();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    await txn.Execute(txn.Query<Document>("INSERT INTO Person ?", person));
});
```

Ion 라이브러리 사용

Async

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("GovId", valueFactory.NewString("TOYENC486FH"));
ionPerson.SetField("FirstName", valueFactory.NewString("Brent"));

await driver.Execute(async txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    IAsyncResult result = await txn.Execute("SELECT * FROM Person WHERE GovId = ?",
 ionGovId);

    // Check if there is a record in the cursor.
    int count = await result.CountAsync();
    if (count > 0)
    {
```

```
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    await txn.Execute("INSERT INTO Person ?", ionPerson);
});
```

Sync

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("GovId", valueFactory.NewString("TOYENC486FH"));
ionPerson.SetField("FirstName", valueFactory.NewString("Brent"));

driver.Execute(txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    IResult result = txn.Execute("SELECT * FROM Person WHERE GovId = ?", ionGovId);

    // Check if there is a record in the cursor.
    int count = result.Count();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    txn.Execute("INSERT INTO Person ?", ionPerson);
});
```

이 트랜잭션은 문서를 Person 테이블에 삽입합니다. 삽입하기 전에 먼저 문서가 테이블에 이미 있는지 확인합니다. 이 검사를 통해 트랜잭션은 본질적으로 멱등성을 가지게 됩니다. 이 트랜잭션을 여러 번 실행하더라도 의도하지 않은 부작용이 발생하지는 않습니다.

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 `GovId` 필드에 인덱스를 사용하는 것이 좋습니다. `GovId`에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

하나의 명령문에 여러 문서 삽입

단일 INSERT 문을 사용하여 여러 문서를 삽입하려면 다음과 같이 문에 C# `List` 파라미터를 전달할 수 있습니다.

```
Person person1 = new Person
{
    FirstName = "Brent",
    GovId = "TOYENC486FH"
};

Person person2 = new Person
{
    FirstName = "Jim",
    GovId = "ROEE1C1AABH"
};

List<Person> people = new List<Person>();
people.Add(person1);
people.Add(person2);

IAsyncResult<Document> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Document>("INSERT INTO Person ?", people));
});

await foreach (Document row in result)
{
    Console.WriteLine("{ documentId: " + row.DocumentId + " }");
    // The statement returns the created documents' ID:
    // { documentId: 6BFt5eJQDFLBW2aR8LPw42 }
    // { documentId: K5Zrcb6N3gmIEHgGhwoyKF }
}
```

Ion 라이브러리 사용

단일 INSERT 문을 사용하여 여러 문서를 삽입하려면 다음과 같이 Ion 목록 타입의 파라미터를 해당 문에 전달할 수 있습니다.

Async

```
IIonValue ionPerson1 = valueFactory.NewEmptyStruct();
ionPerson1.SetField("FirstName", valueFactory.NewString("Brent"));
ionPerson1.SetField("GovId", valueFactory.NewString("TOYENC486FH"));

IIonValue ionPerson2 = valueFactory.NewEmptyStruct();
ionPerson2.SetField("FirstName", valueFactory.NewString("Jim"));
ionPerson2.SetField("GovId", valueFactory.NewString("ROEE1C1AABH"));

IIonValue ionPeople = valueFactory.NewEmptyList();
ionPeople.Add(ionPerson1);
ionPeople.Add(ionPerson2);

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("INSERT INTO Person ?", ionPeople);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the created documents' ID:
    // {
    //     documentId: "6BFt5eJQDFLBW2aR8LPw42"
    // }
    //
    // {
    //     documentId: "K5Zrcb6N3gmIEHgGhwoyKF"
    // }
}
```

Sync

```
IIonValue ionPerson1 = valueFactory.NewEmptyStruct();
ionPerson1.SetField("FirstName", valueFactory.NewString("Brent"));
ionPerson1.SetField("GovId", valueFactory.NewString("TOYENC486FH"));
```

```
IIonValue ionPerson2 = valueFactory.NewEmptyStruct();
ionPerson2.SetField("FirstName", valueFactory.NewString("Jim"));
ionPerson2.SetField("GovId", valueFactory.NewString("ROEE1C1AABH"));

IIonValue ionPeople = valueFactory.NewEmptyList();
ionPeople.Add(ionPerson1);
ionPeople.Add(ionPerson2);

IResult result = driver.Execute(txn =>
{
    return txn.Execute("INSERT INTO Person ?", ionPeople);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the created documents' ID:
    // {
    //     documentId: "6BFt5eJQDFLBW2aR8LPw42"
    // }
    //
    // {
    //     documentId: "K5Zrcb6N3gmIEHgGhwoyKF"
    // }
}
```

Ion 목록을 전달할 때는 변수 자리 표시자(?)를 이중 꺾쇠 괄호(<<...>>)로 묶지 마십시오. 수동 PartiQL 문에서 이중 꺾쇠 괄호는 백으로 알려진 정렬되지 않은 모음을 의미합니다.

문서 업데이트

```
string govId = "TOYENC486FH";
string firstName = "John";

IAsyncResult<Document> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Document>("UPDATE Person SET FirstName = ? WHERE
 GovId = ?", firstName , govId));
});

await foreach (Document row in result)
{
```

```
    Console.WriteLine("{ documentId: " + row.DocumentId + " }");
    // The statement returns the updated document ID:
    // { documentId: Djg30Zoltqy5M4BFsA2jSJ }
}
```

Ion 라이브러리 사용

Async

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");
IIonValue ionFirstName = valueFactory.NewString("John");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("UPDATE Person SET FirstName = ? WHERE GovId = ?",
 ionFirstName , ionGovId);
});

await foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the updated document ID:
    // {
    //     documentId: "Djg30Zoltqy5M4BFsA2jSJ"
    // }
}
```

Sync

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");
IIonValue ionFirstName = valueFactory.NewString("John");

IResult result = driver.Execute(txn =>
{
    return txn.Execute("UPDATE Person SET FirstName = ? WHERE GovId = ?",
 ionFirstName , ionGovId);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the updated document ID:
    // {
```

```
    //     documentId: "Djg30Zoltqy5M4BFsA2jSJ"
    // }
}
```

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 GovId 필드에 인덱스를 사용하는 것이 좋습니다. GovId에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

문서 삭제

```
string govId = "TOYENC486FH";

IAsyncResult<Document> result = await driver.Execute(async txn =>
{
    return await txn.Execute(txn.Query<Document>("DELETE FROM Person WHERE GovId = ?",
 govId));
});

await foreach (Document row in result)
{
    Console.WriteLine("{ documentId: " + row.DocumentId + " }");
    // The statement returns the updated document ID:
    // { documentId: Djg30Zoltqy5M4BFsA2jSJ }
}
```

Ion 라이브러리 사용

Async

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("DELETE FROM Person WHERE GovId = ?", ionGovId);
});

await foreach (IIonValue row in result)
{
```

```
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the deleted document ID:
    // {
    //     documentId: "Djg30Zoltqy5M4BFsA2jSJ"
    // }
}
```

Sync

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IResult result = driver.Execute(txn =>
{
    return txn.Execute("DELETE FROM Person WHERE GovId = ?", ionGovId);
});

foreach (IIonValue row in result)
{
    Console.WriteLine(row.ToPrettyString());
    // The statement returns the deleted document ID:
    // {
    //     documentId: "Djg30Zoltqy5M4BFsA2jSJ"
    // }
}
```

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 `GovId` 필드에 인덱스를 사용하는 것이 좋습니다. `GovId`에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

하나의 트랜잭션에서 여러 명령문 실행

```
// This code snippet is intentionally trivial. In reality you wouldn't do this because
 you'd
// set your UPDATE to filter on vin and insured, and check if you updated something or
 not.
public static async Task<bool> InsureVehicle(IAsyncQldbDriver driver, string vin)
{
    return await driver.Execute(async txn =>
```

```
    {
        // Check if the vehicle is insured.
        Amazon.QLDB.Driver.Generic.IAsyncResult<Vehicle> result = await txn.Execute(
            txn.Query<Vehicle>("SELECT insured FROM Vehicles WHERE vin = ? AND insured
 = FALSE", vin));

        if (await result.CountAsync() > 0)
        {
            // If the vehicle is not insured, insure it.
            await txn.Execute(
                txn.Query<Document>("UPDATE Vehicles SET insured = TRUE WHERE vin = ?",
 vin));
            return true;
        }
        return false;
    });
}
```

Ion 라이브러리 사용

Async

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
 because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
 or not.
public static async Task<bool> InsureVehicle(IAsyncQldbDriver driver, string vin)
{
    ValueFactory valueFactory = new ValueFactory();
    IIonValue ionVin = valueFactory.NewString(vin);

    return await driver.Execute(async txn =>
    {
        // Check if the vehicle is insured.
        Amazon.QLDB.Driver.IAsyncResult result = await txn.Execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
 ionVin);

        if (await result.CountAsync() > 0)
        {
            // If the vehicle is not insured, insure it.
            await txn.Execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", ionVin);
```

```
            return true;
        }
        return false;
    });
}
```

재시도 로직

드라이버의 내장 재시도 로직에 대한 자세한 내용은 Amazon QLDB의 드라이버를 사용한 재시도 정책에 대한 이해 섹션을 참조하세요.

고유성 제약 조건 구현

QLDB는 고유 인덱스를 지원하지 않지만 애플리케이션에서 이 동작을 구현할 수 있습니다.

Person 테이블의 GovId 필드에 고유성 제약 조건을 구현하려고 한다고 가정해 보겠습니다. 이렇게 하면 다음 작업을 수행하는 트랜잭션을 작성합니다.

1. 테이블에 지정된 GovId가 있는 기존 문서가 없는지 확인합니다.
2. 어설션이 통과하면 문서를 삽입합니다.

경쟁 트랜잭션이 어설션을 동시에 통과하면 트랜잭션 중 하나만 성공적으로 커밋됩니다. 다른 트랜잭션은 OCC 충돌 예외가 발생하여 실패합니다.

다음 코드 예제는 이 고유성 제약 조건 구현 방법을 보여줍니다.

```
string govId = "TOYENC486FH";

Person person = new Person
{
    GovId = "TOYENC486FH",
    FirstName = "Brent"
};

await driver.Execute(async txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    IAsyncResult<Person> result = await txn.Execute(txn.Query<Person>("SELECT * FROM
 Person WHERE GovId = ?", govId));
```

```
    // Check if there is a record in the cursor.
    int count = await result.CountAsync();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    await txn.Execute(txn.Query<Document>("INSERT INTO Person ?", person));
});
```

Ion 라이브러리 사용

Async

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("GovId", valueFactory.NewString("TOYENC486FH"));
ionPerson.SetField("FirstName", valueFactory.NewString("Brent"));

await driver.Execute(async txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    IAsyncResult result = await txn.Execute("SELECT * FROM Person WHERE GovId = ?",
 ionGovId);

    // Check if there is a record in the cursor.
    int count = await result.CountAsync();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    await txn.Execute("INSERT INTO Person ?", ionPerson);
});
```

Sync

```
IIonValue ionGovId = valueFactory.NewString("TOYENC486FH");

IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("GovId", valueFactory.NewString("TOYENC486FH"));
ionPerson.SetField("FirstName", valueFactory.NewString("Brent"));

driver.Execute(txn =>
{
    // Check if a document with GovId:TOYENC486FH exists
    // This is critical to make this transaction idempotent
    IResult result = txn.Execute("SELECT * FROM Person WHERE GovId = ?", ionGovId);

    // Check if there is a record in the cursor.
    int count = result.Count();
    if (count > 0)
    {
        // Document already exists, no need to insert
        return;
    }

    // Insert the document.
    txn.Execute("INSERT INTO Person ?", ionPerson);
});
```

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 `GovId` 필드에 인덱스를 사용하는 것이 좋습니다. `GovId`에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

Amazon Ion 작업

QLDB에서 Amazon Ion 데이터를 처리하는 방법은 여러 가지가 있습니다. Ion 라이브러리를 사용하여 Ion 값을 생성하고 수정할 수 있습니다. 또는 Ion 객체 매퍼를 사용하여 C# POCO(Plain Old CLR Object)를 Ion 값에 매핑하거나 그 반대로 매핑할 수 있습니다. .NET용 QLDB 드라이버 버전 1.3.0에서는 Ion 객체 매퍼에 대한 지원이 도입되었습니다.

다음 섹션에서는 두 기술을 모두 사용하여 Ion 데이터를 처리하는 코드 예제를 제공합니다.

목차

Ion 모듈 가져오기

```
using Amazon.IonObjectMapper;
```

Ion 라이브러리 사용

```
using Amazon.IonDotnet.Builders;
```

Ion 유형 생성

다음 코드 예제는 Ion 객체 매퍼를 사용하여 C# 객체에서 Ion 값을 만드는 방법을 보여줍니다.

```
// Assumes that Person class is defined as follows:
// public class Person
// {
//     public string FirstName { get; set; }
//     public int Age { get; set; }
// }

// Initialize the Ion Object Mapper
IonSerializer ionSerializer = new IonSerializer();

// The C# object to be serialized
Person person = new Person
{
    FirstName = "John",
    Age = 13
};

// Serialize the C# object into stream using the Ion Object Mapper
Stream stream = ionSerializer.Serialize(person);
```

```
// Load will take in stream and return a datagram; a top level container of Ion values.
IIonValue ionDatagram = IonLoader.Default.Load(stream);

// To get the Ion value within the datagram, we call GetElementAt(0).
IIonValue ionPerson = ionDatagram.GetElementAt(0);

Console.WriteLine(ionPerson.GetField("firstName").StringValue);
Console.WriteLine(ionPerson.GetField("age").IntValue);
```

Ion 라이브러리 사용

다음 코드 예제는 Ion 라이브러리를 사용하여 Ion 값을 생성하는 두 가지 방법을 보여줍니다.

`ValueFactory` 사용

```
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;

IValueFactory valueFactory = new ValueFactory();

IIonValue ionPerson = valueFactory.NewEmptyStruct();
ionPerson.SetField("firstName", valueFactory.NewString("John"));
ionPerson.SetField("age", valueFactory.NewInt(13));

Console.WriteLine(ionPerson.GetField("firstName").StringValue);
Console.WriteLine(ionPerson.GetField("age").IntValue);
```

`IonLoader` 사용

```
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;

// Load will take in Ion text and return a datagram; a top level container of Ion
 values.
IIonValue ionDatagram = IonLoader.Default.Load("{firstName: \"John\", age: 13}");

// To get the Ion value within the datagram, we call GetElementAt(0).
IIonValue ionPerson = ionDatagram.GetElementAt(0);

Console.WriteLine(ionPerson.GetField("firstName").StringValue);
Console.WriteLine(ionPerson.GetField("age").IntValue);
```

Ion 이진 덤프 가져오기

```
// Initialize the Ion Object Mapper with Ion binary serialization format
IonSerializer ionSerializer = new IonSerializer(new IonSerializationOptions
{
    Format = IonSerializationFormat.BINARY
});

// The C# object to be serialized
Person person = new Person
{
    FirstName = "John",
    Age = 13
};

MemoryStream stream = (MemoryStream) ionSerializer.Serialize(person);
Console.WriteLine(BitConverter.ToString(stream.ToArray()));
```

Ion 라이브러리 사용

```
// ionObject is an Ion struct
MemoryStream stream = new MemoryStream();
using (var writer = IonBinaryWriterBuilder.Build(stream))
{
    ionObject.WriteTo(writer);
    writer.Finish();
}

Console.WriteLine(BitConverter.ToString(stream.ToArray()));
```

Ion 텍스트 덤프 가져오기

```
// Initialize the Ion Object Mapper
IonSerializer ionSerializer = new IonSerializer(new IonSerializationOptions
{
    Format = IonSerializationFormat.TEXT
});

// The C# object to be serialized
Person person = new Person
{
    FirstName = "John",
```

```
    Age = 13
};

MemoryStream stream = (MemoryStream) ionSerializer.Serialize(person);
Console.WriteLine(System.Text.Encoding.UTF8.GetString(stream.ToArray()));
```

Ion 라이브러리 사용

```
// ionObject is an Ion struct
StringWriter sw = new StringWriter();
using (var writer = IonTextWriterBuilder.Build(sw))
{
    ionObject.WriteTo(writer);
    writer.Finish();
}

Console.WriteLine(sw.ToString());
```

Ion 작업에 대한 자세한 정보는 GitHub의 Amazon Ion 설명서를 참조하십시오. QLDB에서 Ion을 사용하는 방법에 대한 추가 코드 예제는 Amazon QLDB에서 Amazon Ion 데이터 타입을 사용한 작업 섹션을 참조하세요.

Go용 Amazon QLDB 드라이버

원장의 데이터로 작업하려면 AWS가 제공하는 드라이버를 사용하여 Go 애플리케이션에서 Amazon QLDB에 연결할 수 있습니다. 다음 주제에서는 Go용 QLDB 드라이버를 시작하는 방법에 대해 설명합니다.

주제

- 드라이버 리소스
- 필수 조건
- 설치
- Go용 Amazon QLDB 드라이버 - 빠른 시작 자습서
- Go용 Amazon QLDB 드라이버 - Cookbook 참조

드라이버 리소스

Go 드라이버에서 지원하는 기능에 대한 자세한 설명은 다음 리소스를 참조하세요.

- API 참조: 3.x, 2.x, 1.x
- 드라이버 소스 코드(GitHub)
- Amazon Ion Cookbook

필수 조건

Go용 QLDB 드라이버를 시작하기 전에 다음을 수행해야 합니다:

1. Amazon QLDB 액세스의 AWS 설정 지침을 따르십시오. 다음 내용이 포함됩니다:
    1. AWS에 가입합니다.
    2. 적절한 QLDB 권한을 가진 사용자를 생성합니다.
    3. 개발을 위한 프로그래밍 방식 액세스 권한을 부여합니다.
2. (선택 사항)원하는 통합 개발 환경(IDE)을 설치합니다. 일반적으로 Go에 사용되는 IDE 목록은 Go 웹 사이트의 편집기 플러그인 및 IDE를 참조하세요.
3. Go 다운로드 사이트에서 다음 버전의 Go를 다운로드하여 설치하세요.
    - 1.15 이상 - Go v3용 QLDB 드라이버
    - 1.14 - Go v1 또는 v2용 QLDB 드라이버
4. AWS SDK for Go를 위한 개발 환경 구성:
    1. AWS 보안 인증을 설정합니다. 공유 보안 인증 파일을 생성할 것을 권장합니다.

        지침은 AWS SDK for Go 개발자 가이드의 자격 증명 지정을 참조하세요.
    2. 기본 AWS 리전을 설정하십시오. 방법을 알아보려면 AWS 리전 지정하기를 참조하세요.

        사용 가능한 리전의 전체 목록은 AWS 일반 참조에서 Amazon QLDB 엔드포인트 및 할당량을 참조하십시오.

그런 다음 기본 샘플 애플리케이션을 설정하고 예의 단축 코드를 실행하거나 해당 드라이버를 기존 Go 프로젝트에 설치할 수 있습니다.

- 기존 프로젝트에 QLDB 드라이버와 AWS SDK for Go를 설치하려면 설치로 이동하세요.
- 프로젝트를 설정하고 원장에 대한 기본 데이터 트랜잭션을 보여주는 단축 코드 예제를 실행하려면 빠른 시작 자습서를 참조하십시오.

설치

Go용 QLDB 드라이버는 GitHub 리포지토리 awslabs/amazon-qldb-driver-go의 오픈 소스입니다. QLDB는 다음 드라이버 버전과 해당 Go 종속성을 지원합니다.

드라이버 버전	Go 버전	상태	최근 릴리스 날짜
1.x	1.14 이상	프로덕션 릴리스	2021년 6월 16일
2.x	1.14 이상	프로덕션 릴리스	2021년 7월 21일
3.x	1.15 이상	프로덕션 릴리스	2022년 11월 10일

드라이버를 설치하려면

1. 프로젝트에서 Go 모듈을 사용하여 프로젝트 종속성을 설치하는지 확인하세요.
2. 프로젝트 디렉터리에 다음 `go get` 명령을 입력합니다.

   3.x

   ```
   $ go get -u github.com/awslabs/amazon-qldb-driver-go/v3/qldbdriver
   ```

   2.x

   ```
   $ go get -u github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver
   ```

드라이버를 설치하면 AWS SDK for Go 또는 AWS SDK for Go v2 및 Amazon Ion 패키지를 포함한 종속 항목도 설치됩니다.

원장에서 기본 데이터 트랜잭션을 실행하는 방법에 대한 단축 코드 예제는 Cookbook 참조를 참조하십시오.

Go용 Amazon QLDB 드라이버 - 빠른 시작 자습서

이 자습서에서는 Go용 Amazon QLDB 드라이버의 최신 버전을 사용하여 간단한 애플리케이션을 설정하는 방법을 알아봅니다. 이 안내서에는 드라이버 설치 단계 및 기본적인 CRUD(생성, 읽기, 업데이트 및 삭제) 작업에 대한 단축 코드 예제가 포함되어 있습니다.

주제

필수 조건

시작하기 전에 다음을 수행해야 합니다.

1. 아직 Go 드라이버에 대한 필수 조건을 완료하지 않은 경우, 완료하세요. 여기에는 AWS 가입, 개발을 위한 프로그래밍 방식 액세스 권한 부여, Go 설치가 포함됩니다.
2. `quick-start`라는 명칭의 원장을 생성합니다.

   원장 생성 방법을 알아보려면 콘솔 시작하기의 Amazon QLDB 원장의 기본 작업 또는 1단계: 새 원장 생성 섹션을 참조하세요.

1단계: 드라이버 설치

프로젝트에서 Go 모듈을 사용하여 프로젝트 종속성을 설치하는지 확인하세요.

프로젝트 디렉터리에 다음 `go get` 명령을 입력합니다.

```
$ go get -u github.com/awslabs/amazon-qldb-driver-go/v3/qldbdriver
```

드라이버를 설치하면 AWS SDK for Go v2 및 Amazon Ion 패키지를 포함한 종속 항목도 설치됩니다.

2단계: 패키지 가져오기

다음 AWS 패키지를 가져옵니다.

```
import (
    "context"
    "fmt"

    "github.com/amzn/ion-go/ion"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/qldbSession"
    "github.com/awslabs/amazon-qldb-driver-go/v3/qldbdriver"
)
```

3단계: 드라이버 초기화

`quick-start`라는 명칭의 원장에 연결되는 드라이버의 인스턴스를 초기화합니다.

```
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
  panic(err)
}

qldbSession := qldbsession.NewFromConfig(cfg, func(options *qldbsession.Options) {
    options.Region = "us-east-1"
})
driver, err := qldbdriver.New(
    "quick-start",
    qldbSession,
    func(options *qldbdriver.DriverOptions) {
        options.LoggerVerbosity = qldbdriver.LogInfo
    })
if err != nil {
    panic(err)
}

defer driver.Shutdown(context.Background())
```

> **Note**
>
> 이 코드 예에서는 *us-east-1*을 원장을 생성한 AWS 리전으로 바꿉니다.

4단계: 테이블 및 인덱스 생성

다음 코드 예에서는 `CREATE TABLE` 및 `CREATE INDEX` 문을 실행하는 방법을 보여줍니다.

```
_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    _, err := txn.Execute("CREATE TABLE People")
    if err != nil {
        return nil, err
    }

    // When working with QLDB, it's recommended to create an index on fields we're
 filtering on.
    // This reduces the chance of OCC conflict exceptions with large datasets.
    _, err = txn.Execute("CREATE INDEX ON People (firstName)")
    if err != nil {
        return nil, err
    }

    _, err = txn.Execute("CREATE INDEX ON People (age)")
    if err != nil {
        return nil, err
    }

    return nil, nil
})
if err != nil {
    panic(err)
}
```

이 코드는 `People`이라는 테이블을 만들고 해당 테이블의 `age` 및 `firstName` 필드를 인덱싱합니다. 인덱스는 쿼리 성능을 최적화하고 OCC(낙관적 동시성 제어) 충돌 예외를 제한하는 데 필요합니다.

5단계: 문서 삽입

다음 코드 예에서는 `INSERT` 문을 실행하는 방법을 보여줍니다. QLDB는 PartiQL 쿼리 언어(SQL 호환) 및 Amazon Ion 데이터 형식(JSON의 상위 집합)을 지원합니다.

리터럴 PartiQL 사용

다음 코드는 문자열 리터럴 PartiQL 명령문을 사용하여 People 테이블에 문서를 삽입합니다.

```
_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    return txn.Execute("INSERT INTO People {'firstName': 'Jane', 'lastName': 'Doe',
 'age': 77}")
})
if err != nil {
    panic(err)
}
```

Ion 데이터 유형 사용

Go의 내장 JSON 패키지와 마찬가지로 Ion과 Go 데이터 유형을 마샬링하거나 Ion에서 마샬링 취소할 수 있습니다.

1. 다음과 같이 Person이라는 이름의 Go 구조가 있는 경우.

   ```
   type Person struct {
       FirstName string `ion:"firstName"`
       LastName  string `ion:"lastName"`
       Age       int    `ion:"age"`
   }
   ```

2. Person의 인스턴스를 만듭니다.

   ```
   person := Person{"John", "Doe", 54}
   ```

   드라이버는 Ion 인코딩된 person의 텍스트 표현을 자동으로 마샬링합니다.

   > **Important**
   >
   > 마샬링과 마샬링 취소가 제대로 작동하려면 Go 데이터 구조의 필드 이름을 내보내야 합니다(첫 글자를 대문자로 표시).

3. person 인스턴스를 트랜잭션의 Execute 메서드로 전달합니다.

   ```
   _, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
    (interface{}, error) {
   ```

```
    return txn.Execute("INSERT INTO People ?", person)
})
if err != nil {
    panic(err)
}
```

이 예에서는 물음표(?)를 변수 자리 표시자로 사용하여 문서 정보를 해당 문에 전달합니다. 자리 표시자를 사용할 때는 Ion 인코딩된 텍스트 값을 전달해야 합니다.

> **Tip**
>
> 단일 INSERT 문을 사용하여 여러 문서를 삽입하려면 다음과 같이 목록 타입의 파라미터를 해당 문에 전달할 수 있습니다.
>
> ```
> // people is a list
> txn.Execute("INSERT INTO People ?", people)
> ```
>
> 목록을 전달할 때 변수 자리 표시자(?)를 이중 꺾쇠 괄호(<<...>>)로 묶지 마세요. 수동 PartiQL 문에서 이중 꺾쇠 괄호는 백으로 알려진 정렬되지 않은 모음을 의미합니다.

6단계: 문서 쿼리

다음 코드 예에서는 SELECT 문을 실행하는 방법을 보여줍니다.

```
p, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    result, err := txn.Execute("SELECT firstName, lastName, age FROM People WHERE age =
 54")
    if err != nil {
        return nil, err
    }

    // Assume the result is not empty
    hasNext := result.Next(txn)
    if !hasNext && result.Err() != nil {
        return nil, result.Err()
    }

    ionBinary := result.GetCurrentData()
```

```
    temp := new(Person)
    err = ion.Unmarshal(ionBinary, temp)
    if err != nil {
        return nil, err
    }

    return *temp, nil
})
if err != nil {
    panic(err)
}

var returnedPerson Person
returnedPerson = p.(Person)

if returnedPerson != person {
    fmt.Print("Queried result does not match inserted struct")
}
```

이 예제에서는 People 테이블에서 문서를 쿼리하고 결과 세트가 비어 있지 않다고 가정하고 결과에서 문서를 반환합니다.

7단계: 문서 업데이트

다음 코드 예에서는 UPDATE 문을 실행하는 방법을 보여줍니다.

```
person.Age += 10

_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    return txn.Execute("UPDATE People SET age = ? WHERE firstName = ?", person.Age,
 person.FirstName)
})
if err != nil {
    panic(err)
}
```

8단계: 업데이트된 문서 쿼리

다음 코드 예제는 firstName으로 People 테이블을 쿼리하고 결과 세트의 모든 문서를 반환합니다.

```
p, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
```

```
    result, err := txn.Execute("SELECT firstName, lastName, age FROM People WHERE
 firstName = ?", person.FirstName)
    if err != nil {
        return nil, err
    }

    var people []Person
    for result.Next(txn) {
        ionBinary := result.GetCurrentData()

        temp := new(Person)
        err = ion.Unmarshal(ionBinary, temp)
        if err != nil {
            return nil, err
        }

        people = append(people, *temp)
    }
    if result.Err() != nil {
        return nil, result.Err()
    }

    return people, nil
})
if err != nil {
    panic(err)
}

var people []Person
people = p.([]Person)

updatedPerson := Person{"John", "Doe", 64}
if people[0] != updatedPerson {
    fmt.Print("Queried result does not match updated struct")
}
```

9단계: 테이블 삭제

다음 코드 예에서는 DROP TABLE 문을 실행하는 방법을 보여줍니다.

```
_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    return txn.Execute("DROP TABLE People")
```

```
})
if err != nil {
    panic(err)
}
```

전체 애플리케이션 실행

다음 코드 예는 애플리케이션의 전체 버전입니다. 이전 단계를 개별적으로 수행하는 대신 이 코드 예를 처음부터 끝까지 복사하여 실행할 수도 있습니다. 이 애플리케이션은 `quick-start`이라는 명칭의 원장에 대한 몇 가지 기본 CRUD 작업을 보여줍니다.

> **Note**
>
> 이 코드를 실행하기 전에 `quick-start` 원장에 `People`이라는 명칭의 활성 테이블이 아직 없는지 확인하세요.

```
package main

import (
    "context"
    "fmt"

    "github.com/amzn/ion-go/ion"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/qldbsession"
    "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)

func main() {
    awsSession := session.Must(session.NewSession(aws.NewConfig().WithRegion("us-
east-1")))
    qldbSession := qldbsession.New(awsSession)

    driver, err := qldbdriver.New(
        "quick-start",
        qldbSession,
        func(options *qldbdriver.DriverOptions) {
            options.LoggerVerbosity = qldbdriver.LogInfo
        })
    if err != nil {
```

```
        panic(err)
    }
    defer driver.Shutdown(context.Background())

    _, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
(interface{}, error) {
        _, err := txn.Execute("CREATE TABLE People")
        if err != nil {
            return nil, err
        }

        // When working with QLDB, it's recommended to create an index on fields we're
filtering on.
        // This reduces the chance of OCC conflict exceptions with large datasets.
        _, err = txn.Execute("CREATE INDEX ON People (firstName)")
        if err != nil {
            return nil, err
        }

        _, err = txn.Execute("CREATE INDEX ON People (age)")
        if err != nil {
            return nil, err
        }

        return nil, nil
    })
    if err != nil {
        panic(err)
    }

    _, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
(interface{}, error) {
        return txn.Execute("INSERT INTO People {'firstName': 'Jane', 'lastName': 'Doe',
'age': 77}")
    })
    if err != nil {
        panic(err)
    }

    type Person struct {
        FirstName string `ion:"firstName"`
        LastName  string `ion:"lastName"`
        Age       int    `ion:"age"`
    }
```

```
   person := Person{"John", "Doe", 54}

   _, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
(interface{}, error) {
       return txn.Execute("INSERT INTO People ?", person)
   })
   if err != nil {
       panic(err)
   }

   p, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
(interface{}, error) {
       result, err := txn.Execute("SELECT firstName, lastName, age FROM People WHERE
age = 54")
       if err != nil {
           return nil, err
       }

       // Assume the result is not empty
       hasNext := result.Next(txn)
       if !hasNext && result.Err() != nil {
           return nil, result.Err()
       }

       ionBinary := result.GetCurrentData()

       temp := new(Person)
       err = ion.Unmarshal(ionBinary, temp)
       if err != nil {
           return nil, err
       }

       return *temp, nil
   })
   if err != nil {
       panic(err)
   }

   var returnedPerson Person
   returnedPerson = p.(Person)

   if returnedPerson != person {
       fmt.Print("Queried result does not match inserted struct")
```

```
    }

    person.Age += 10

    _, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
(interface{}, error) {
        return txn.Execute("UPDATE People SET age = ? WHERE firstName = ?", person.Age,
person.FirstName)
    })
    if err != nil {
        panic(err)
    }

    p, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
(interface{}, error) {
        result, err := txn.Execute("SELECT firstName, lastName, age FROM People WHERE
firstName = ?", person.FirstName)
        if err != nil {
            return nil, err
        }

        var people []Person
        for result.Next(txn) {
            ionBinary := result.GetCurrentData()

            temp := new(Person)
            err = ion.Unmarshal(ionBinary, temp)
            if err != nil {
                return nil, err
            }

            people = append(people, *temp)
        }
        if result.Err() != nil {
            return nil, result.Err()
        }

        return people, nil
    })
    if err != nil {
        panic(err)
    }

    var people []Person
```

```
    people = p.([]Person)

    updatedPerson := Person{"John", "Doe", 64}
    if people[0] != updatedPerson {
        fmt.Print("Queried result does not match updated struct")
    }

    _, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
        return txn.Execute("DROP TABLE People")
    })
    if err != nil {
        panic(err)
    }
}
```

Go용 Amazon QLDB 드라이버 - Cookbook 참조

이 참조 가이드는 Go용 Amazon QLDB 드라이버의 일반적인 사용 사례를 보여줍니다. 이 Go 코드 예제는 드라이버를 사용하여 기본 CRUD(생성, 읽기, 업데이트, 삭제) 작업을 실행하는 방법을 보여줍니다. 또한 Amazon Ion 데이터를 처리하기 위한 코드 예제도 포함되어 있습니다. 또한 이 가이드에서는 트랜잭션에 멱등성을 부여하고 고유성 제약하는 모범 사례를 중점적으로 설명합니다.

> **Note**
>
> 해당하는 경우, 일부 사용 사례에서는 Go용 QLDB 드라이버의 지원되는 각 주요 버전마다 다른 코드 예제가 있습니다.

목차

드라이버 가져오기

다음 코드 예제는 드라이버 및 기타 필수 AWS 패키지를 가져옵니다.

3.x

```
import (

    "github.com/amzn/ion-go/ion"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/qldbSession"
    "github.com/awslabs/amazon-qldb-driver-go/v3/qldbdriver"
)
```

2.x

```
import (

    "github.com/amzn/ion-go/ion"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/qldbsession"
    "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)
```

> **Note**
>
> 이 예제에서는 Amazon Ion 패키지(amzn/ion-go/ion)도 가져옵니다. 이 참조에서 일부 데이터 작업을 실행할 때 Ion 데이터를 처리하려면 이 패키지가 필요합니다. 자세한 내용은 Amazon Ion 작업 섹션을 참조하세요.

드라이버 인스턴스화

다음 코드 예제는 지정된 AWS 리전의 지정된 원장 이름에 연결되는 드라이버 인스턴스를 만듭니다.

3.x

```
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic(err)
}

qldbSession := qldbsession.NewFromConfig(cfg, func(options *qldbsession.Options) {
    options.Region = "us-east-1"
})
driver, err := qldbdriver.New(
  "vehicle-registration",
  qldbSession,
  func(options *qldbdriver.DriverOptions) {
    options.LoggerVerbosity = qldbdriver.LogInfo
})
if err != nil {
  panic(err)
}

defer driver.Shutdown(context.Background())
```

2.x

```
awsSession := session.Must(session.NewSession(aws.NewConfig().WithRegion("us-
east-1")))
qldbSession := qldbsession.New(awsSession)

driver, err := qldbdriver.New(
  "vehicle-registration",
  qldbSession,
```

```
  func(options *qldbdriver.DriverOptions) {
    options.LoggerVerbosity = qldbdriver.LogInfo
  })
if err != nil {
  panic(err)
}
```

CRUD 작업

QLDB는 트랜잭션의 일부로 CRUD(생성, 읽기, 업데이트, 삭제) 작업을 실행합니다.

> **Warning**
>
> 가장 좋은 방법은 쓰기 트랜잭션이 완전한 멱등성을 부여하는 것입니다.

트랜잭션에 멱등성 부여하기

재시도 시 예상치 못한 부작용이 발생하지 않도록 쓰기 트랜잭션에 멱등성을 부여하는 것이 좋습니다. 여러 번 실행하여 매번 동일한 결과를 생성할 수 있는 트랜잭션은 멱등성을 가집니다.

이름이 `Person`인 테이블에 문서를 삽입하는 트랜잭션을 예로 들어 보겠습니다. 트랜잭션은 먼저 문서가 테이블에 이미 존재하는지 여부를 확인해야 합니다. 이렇게 확인하지 않으면 테이블에 문서가 중복될 수 있습니다.

QLDB가 서버 측에서 트랜잭션을 성공적으로 커밋했지만 응답을 기다리는 동안 클라이언트 제한 시간이 초과되었다고 가정해 보겠습니다. 트랜잭션이 멱등성을 가지지 않는 경우 재시도 시 동일한 문서가 두 번 이상 삽입될 수 있습니다.

인덱스를 사용하여 전체 테이블 스캔 방지

인덱싱된 필드 또는 문서 ID(예: `WHERE indexedField = 123` 또는 `WHERE indexedField IN (456, 789)`)에서 동등 연산자를 사용하여 `WHERE` 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 이 인덱싱된 조회가 없으면 QLDB는 테이블 스캔을 수행해야 하며, 이로 인해 트랜잭션 제한 시간이 초과되거나 OCC(낙관적 동시성 제어) 충돌이 발생할 수 있습니다.

OCC에 대한 자세한 내용은 Amazon QLDB 동시성 모델 단원을 참조하세요.

암시적으로 생성된 트랜잭션

QLDBDriver.Execute 함수는 트랜잭션의 인스턴스를 수신하는 Lambda 함수를 받아들이며, 이 함수를 사용하여 명령문을 실행할 수 있습니다. `Transaction`의 인스턴스는 암시적으로 생성된 트랜잭션을 래핑합니다.

`Transaction.Execute` 함수를 사용하여 Lambda 함수 내에서 명령문을 실행할 수 있습니다. 드라이버는 Lambda 함수가 반환될 때 트랜잭션을 암시적으로 커밋합니다.

다음 섹션에서는 기본 CRUD 작업을 실행하고, 사용자 지정 재시도 로직을 지정하고, 고유성 제약 조건을 구현하는 방법을 보여줍니다.

목차

테이블 생성

```
result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  return txn.Execute("CREATE TABLE Person")
})
```

인덱스 생성

```
result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  return txn.Execute("CREATE INDEX ON Person(GovId)")
})
```

문서 읽기

```
var decodedResult map[string]interface{}

// Assumes that Person table has documents as follows:
// { "GovId": "TOYENC486FH", "FirstName": "Brent" }
_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  result, err := txn.Execute("SELECT * FROM Person WHERE GovId = 'TOYENC486FH'")
  if err != nil {
    return nil, err
  }
  for result.Next(txn) {
    ionBinary := result.GetCurrentData()
    err = ion.Unmarshal(ionBinary, &decodedResult)
    if err != nil {
      return nil, err
    }
    fmt.Println(decodedResult) // prints map[GovId: TOYENC486FH FirstName:Brent]
  }
  if result.Err() != nil {
    return nil, result.Err()
  }
  return nil, nil
})
if err != nil {
  panic(err)
}
```

쿼리 파라미터 사용

다음 코드 예제는 네이티브 유형 쿼리 파라미터를 사용합니다.

```
result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  return txn.Execute("SELECT * FROM Person WHERE GovId = ?", "TOYENC486FH")
})
if err != nil {
  panic(err)
}
```

다음 코드 예제는 여러 쿼리 파라미터를 사용합니다.

```
result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  return txn.Execute("SELECT * FROM Person WHERE GovId = ? AND FirstName = ?",
 "TOYENC486FH", "Brent")
})
if err != nil {
  panic(err)
}
```

다음 코드 예제는 쿼리 파라미터 목록을 사용합니다.

```
govIDs := []string{}{"TOYENC486FH", "ROEE1", "YH844"}

result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  return txn.Execute("SELECT * FROM Person WHERE GovId IN (?,?,?)", govIDs...)
})
if err != nil {
  panic(err)
}
```

> **Note**
>
> 인덱싱된 조회 없이 쿼리를 실행하면 전체 테이블 스캔이 호출됩니다. 이 예제에서는 성능을 최적화하기 위해 GovId 필드에 인덱스를 사용하는 것이 좋습니다. GovId에 인덱스를 사용하지 않으면 쿼리에 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

문서 삽입하기

다음 코드 예제는 네이티브 데이터 유형을 삽입합니다.

```
_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  // Check if a document with a GovId of TOYENC486FH exists
  // This is critical to make this transaction idempotent
  result, err := txn.Execute("SELECT * FROM Person WHERE GovId = ?", "TOYENC486FH")
  if err != nil {
    return nil, err
  }
```

```
  // Check if there are any results
  if result.Next(txn) {
    // Document already exists, no need to insert
  } else {
    person := map[string]interface{}{
      "GovId": "TOYENC486FH",
      "FirstName": "Brent",
    }
    _, err = txn.Execute("INSERT INTO Person ?", person)
    if err != nil {
      return nil, err
    }
  }
  return nil, nil
})
```

이 트랜잭션은 문서를 `Person` 테이블에 삽입합니다. 삽입하기 전에 먼저 문서가 테이블에 이미 있는지 확인합니다. 이 검사를 통해 트랜잭션은 본질적으로 멱등성을 가지게 됩니다. 이 트랜잭션을 여러 번 실행하더라도 의도하지 않은 부작용이 발생하지는 않습니다.

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 `GovId` 필드에 인덱스를 사용하는 것이 좋습니다. `GovId`에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

하나의 명령문에 여러 문서 삽입

단일 INSERT 문을 사용하여 여러 문서를 삽입하려면 다음과 같이 목록 타입의 파라미터를 해당 문에 전달할 수 있습니다.

```
// people is a list
txn.Execute("INSERT INTO People ?", people)
```

목록을 전달할 때 변수 자리 표시자(?)를 이중 꺾쇠 괄호(`<<...>>`)로 묶지 마세요. 수동 PartiQL 문에서 이중 꺾쇠 괄호는 백으로 알려진 정렬되지 않은 모음을 의미합니다.

문서 업데이트

다음 코드 예제는 네이티브 데이터 유형을 사용합니다.

```
result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  return txn.Execute("UPDATE Person SET FirstName = ? WHERE GovId = ?", "John",
 "TOYENC486FH")
})
```

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 GovId 필드에 인덱스를 사용하는 것이 좋습니다. GovId에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

문서 삭제

다음 코드 예제는 네이티브 데이터 유형을 사용합니다.

```
result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  return txn.Execute("DELETE FROM Person WHERE GovId = ?", "TOYENC486FH")
})
```

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 GovId 필드에 인덱스를 사용하는 것이 좋습니다. GovId에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

하나의 트랜잭션에서 여러 명령문 실행

```
// This code snippet is intentionally trivial. In reality you wouldn't do this because
 you'd
// set your UPDATE to filter on vin and insured, and check if you updated something or
 not.
func InsureCar(driver *qldbdriver.QLDBDriver, vin string) (bool, error) {
    insured, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {

        result, err := txn.Execute(
```

```
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE", vin)
        if err != nil {
            return false, err
        }

        hasNext := result.Next(txn)
        if !hasNext && result.Err() != nil {
            return false, result.Err()
        }

        if hasNext {
            _, err = txn.Execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin)
            if err != nil {
                return false, err
            }
            return true, nil
        }
        return false, nil
    })
    if err != nil {
        panic(err)
    }

    return insured.(bool), err
}
```

재시도 로직

드라이버의 `Execute` 함수에는 재시도 가능한 예외(예: 시간 초과 또는 OCC 충돌)가 발생할 경우 트랜잭션을 재시도하는 재시도 메커니즘이 내장되어 있습니다. 최대 재시도 횟수와 백오프 전략을 구성할 수 있습니다.

기본 재시도 제한은 `4`이며, 기본 백오프 전략은 `10`밀리초 단위의 ExponentialBackoffStrategy입니다. RetryPolicy 인스턴스를 사용하여 드라이버 인스턴스별 및 트랜잭션별로 재시도 정책을 설정할 수 있습니다.

다음 코드 예제는 드라이버 인스턴스에 대한 사용자 지정 재시도 제한 및 사용자 지정 백오프 전략을 사용하여 재시도 로직을 지정합니다.

```
import (
  "github.com/aws/aws-sdk-go/aws"
  "github.com/aws/aws-sdk-go/aws/session"
```

```
  "github.com/aws/aws-sdk-go/service/qldbsession"
  "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)

func main() {
  awsSession := session.Must(session.NewSession(aws.NewConfig().WithRegion("us-
east-1")))
  qldbSession := qldbsession.New(awsSession)

  // Configuring retry limit to 2
  retryPolicy := qldbdriver.RetryPolicy{MaxRetryLimit: 2}

  driver, err := qldbdriver.New("test-ledger", qldbSession, func(options
 *qldbdriver.DriverOptions) {
    options.RetryPolicy = retryPolicy
  })
  if err != nil {
    panic(err)
  }

  // Configuring an exponential backoff strategy with base of 20 milliseconds
  retryPolicy = qldbdriver.RetryPolicy{
    MaxRetryLimit: 2,
    Backoff: qldbdriver.ExponentialBackoffStrategy{SleepBase: 20, SleepCap: 4000,
    }}

  driver, err = qldbdriver.New("test-ledger", qldbSession, func(options
 *qldbdriver.DriverOptions) {
    options.RetryPolicy = retryPolicy
  })
  if err != nil {
    panic(err)
  }
}
```

다음 코드 예제는 특정 익명 함수에 대한 사용자 지정 재시도 제한 및 사용자 지정 백오프 전략을 사용하여 재시도 로직을 지정합니다. `SetRetryPolicy` 함수는 드라이버 인스턴스에 설정된 재시도 정책을 재정의합니다.

```
import (
  "context"
  "github.com/aws/aws-sdk-go/aws"
  "github.com/aws/aws-sdk-go/aws/session"
```

```
  "github.com/aws/aws-sdk-go/service/qldbsession"
  "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)

func main() {
  awsSession := session.Must(session.NewSession(aws.NewConfig().WithRegion("us-
east-1")))
  qldbSession := qldbsession.New(awsSession)

  // Configuring retry limit to 2
  retryPolicy1 := qldbdriver.RetryPolicy{MaxRetryLimit: 2}

  driver, err := qldbdriver.New("test-ledger", qldbSession, func(options
 *qldbdriver.DriverOptions) {
    options.RetryPolicy = retryPolicy1
  })
  if err != nil {
    panic(err)
  }

  // Configuring an exponential backoff strategy with base of 20 milliseconds
  retryPolicy2 := qldbdriver.RetryPolicy{
    MaxRetryLimit: 2,
    Backoff: qldbdriver.ExponentialBackoffStrategy{SleepBase: 20, SleepCap: 4000,
    }}

  // Overrides the retry policy set by the driver instance
  driver.SetRetryPolicy(retryPolicy2)

  driver.Execute(context.Background(), func(txn qldbdriver.Transaction) (interface{},
 error) {
    return txn.Execute("CREATE TABLE Person")
  })
}
```

고유성 제약 조건 구현

QLDB는 고유 인덱스를 지원하지 않지만 애플리케이션에서 이 동작을 구현할 수 있습니다.

`Person` 테이블의 `GovId` 필드에 고유성 제약 조건을 구현하려고 한다고 가정해 보겠습니다. 이렇게 하면 다음 작업을 수행하는 트랜잭션을 작성합니다.

1. 테이블에 지정된 `GovId`가 있는 기존 문서가 없는지 확인합니다.

2. 어설션이 통과하면 문서를 삽입합니다.

경쟁 트랜잭션이 어설션을 동시에 통과하면 트랜잭션 중 하나만 성공적으로 커밋됩니다. 다른 트랜잭션은 OCC 충돌 예외가 발생하여 실패합니다.

다음 코드 예제는 이 고유성 제약 조건 구현 방법을 보여줍니다.

```
govID := "TOYENC486FH"

document := map[string]interface{}{
  "GovId":     "TOYENC486FH",
  "FirstName": "Brent",
}

result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
  // Check if doc with GovId = govID exists
  result, err := txn.Execute("SELECT * FROM Person WHERE GovId = ?", govID)
  if err != nil {
    return nil, err
  }
  // Check if there are any results
  if result.Next(txn) {
    // Document already exists, no need to insert
    return nil, nil
  }
  return txn.Execute("INSERT INTO Person ?", document)
})
if err != nil {
  panic(err)
}
```

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 `GovId` 필드에 인덱스를 사용하는 것이 좋습니다. `GovId`에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

Amazon Ion 작업

다음 섹션에서는 Amazon Ion 모듈을 사용하여 Ion 데이터를 처리하는 방법을 보여줍니다.

목차

Ion 모듈 가져오기

```
import "github.com/amzn/ion-go/ion"
```

Ion 유형 생성

Go용 Ion 라이브러리는 현재 문서 객체 모델(DOM)을 지원하지 않으므로 Ion 데이터 유형을 만들 수 없습니다. 하지만 QLDB를 사용할 때는 Go 네이티브 유형과 Ion 이진 사이를 마샬링 및 마샬링 취소할 수 있습니다.

Ion 이진 가져오기

```
aDict := map[string]interface{}{
  "GovId": "TOYENC486FH",
  "FirstName": "Brent",
}

ionBytes, err := ion.MarshalBinary(aDict)
if err != nil {
  panic(err)
}

fmt.Println(ionBytes) // prints [224 1 0 234 238 151 129 131 222 147 135 190 144 133 71
 111 118 73 100 137 70 105 114 115 116 78 97 109 101 222 148 138 139 84 79 89 69 78 67
 52 56 54 70 72 139 133 66 114 101 110 116]
```

Ion 텍스트 가져오기

```
aDict := map[string]interface{}{
  "GovId": "TOYENC486FH",
```

```
  "FirstName": "Brent",
}

ionBytes, err := ion.MarshalText(aDict)
if err != nil {
  panic(err)
}

fmt.Println(string(ionBytes)) // prints {FirstName:"Brent",GovId:"TOYENC486FH"}
```

Ion에 대한 자세한 정보는 GitHub의 Amazon Ion 설명서를 참조하십시오. QLDB에서 Ion을 사용하는 방법에 대한 추가 코드 예제는 Amazon QLDB에서 Amazon Ion 데이터 타입을 사용한 작업 섹션을 참조하세요.

Node.js용 Amazon QLDB 드라이버

원장의 데이터를 사용하려면 AWS가 제공하는 드라이버를 사용하여 Node.js 애플리케이션에서 Amazon QLDB에 연결할 수 있습니다. 다음 주제에서는 Node.js용 QLDB 드라이버를 시작하는 방법에 대해 설명합니다.

주제

- 드라이버 리소스
- 사전 조건
- 설치
- 설정 권장 사항
- Node.js용 Amazon QLDB 드라이버 - 빠른 시작 자습서
- Node.js용 Amazon QLDB 드라이버 - Cookbook 참조

드라이버 리소스

Node.js 드라이버에서 지원하는 기능에 대한 자세한 정보는 다음 리소스를 참조하십시오.

- API 참조: 3.x, 2.x, 1.x
- 드라이버 소스 코드(GitHub)
- 샘플 애플리케이션 소스 코드(GitHub)
- Amazon Ion 코드 예제
- AWS Lambda(AWS 블로그)를 사용하여 QLDB에 간단한 CRUD 작업 및 데이터 스트림 구축하기

사전 조건

Node.js용 QLDB 드라이버를 시작하기 전에 다음을 진행해야 합니다.

1. Amazon QLDB 액세스의 AWS 설정 지침을 따르십시오. 다음 내용이 포함됩니다:
    1. AWS에 가입합니다.
    2. 적절한 QLDB 권한을 가진 사용자를 생성합니다.
    3. 개발을 위한 프로그래밍 방식 액세스 권한을 부여합니다.
2. Node.js 다운로드 사이트에서 Node.js 버전 14.x 이상을 설치하십시오. (이전 버전의 드라이버는 Node.js 버전 10.x 이상을 지원합니다.)
3. Node.js의 JavaScript용 AWS SDK에 대한 개발 환경을 구성합니다.
    1. AWS 보안 인증을 설정합니다. 공유 보안 인증 파일을 생성할 것을 권장합니다.

        지침은 AWS SDK for JavaScript 개발자 안내서의 공유 보안 인증 파일에서 Node.js 보안 인증 로드를 참조하십시오.
    2. 기본 AWS 리전을 설정하십시오. 사용 방법을 배우려면 AWS 리전 설정을 참조하십시오.

        사용 가능한 리전의 전체 목록은 AWS 일반 참조에서 Amazon QLDB 엔드포인트 및 할당량을 참조하십시오.

다음으로 전체 자습서 샘플 애플리케이션을 다운로드하거나 Node.js 프로젝트에 드라이버만 설치하고 단축 코드 예제를 실행할 수 있습니다.

- 기존 프로젝트에서 QLDB 드라이버와 Node.js의 JavaScript용 AWS SDK를 설치하려면 설치로 진행하십시오.
- 프로젝트를 설정하고 원장에 대한 기본 데이터 트랜잭션을 보여주는 단축 코드 예제를 실행하려면 빠른 시작 자습서를 참조하십시오.
- 전체 자습서 샘플 애플리케이션에서 데이터 및 관리 API 작업에 대한 보다 심층적인 예제를 실행하려면 Node.js 자습서를 참조하십시오.

설치

QLDB는 다음 드라이버 버전과 해당 Node.js 종속성을 지원합니다.

드라이버 버전	Node.js 버전	상태 표시기	최근 릴리스 날짜
1.x	10.x 이상	프로덕션 릴리스	2020년 6월 5일
2.x	10.x 이상	프로덕션 릴리스	2021년 5월 6일
3.x	14.x 이상	프로덕션 릴리스	2023년 11월 10일

npm(Node.js 패키지 관리자)을 사용하여 QLDB 드라이버를 설치하려면 프로젝트 루트 디렉터리에서 다음 명령을 입력합니다.

3.x

```
npm install amazon-qldb-driver-nodejs
```

2.x

```
npm install amazon-qldb-driver-nodejs@2.2.0
```

1.x

```
npm install amazon-qldb-driver-nodejs@1.0.0
```

드라이버는 다음 패키지에 대한 피어 종속성을 가집니다. 또한 이러한 패키지를 프로젝트에 종속성으로 설치해야 합니다.

3.x

모듈식 집계 QLDB 클라이언트(관리 API)

```
npm install @aws-sdk/client-qldb
```

모듈식 집계 QLDB 세션 클라이언트(데이터 API)

```
npm install @aws-sdk/client-qldb-session
```

Amazon Ion 데이터 형식

```
npm install ion-js
```

`BigInt`의 순수 JavaScript 구현

```
npm install jsbi
```

2.x

AWS SDK for JavaScript

```
npm install aws-sdk
```

Amazon Ion 데이터 형식

```
npm install ion-js@4.0.0
```

`BigInt`의 순수 JavaScript 구현

```
npm install jsbi@3.1.1
```

1.x

AWS SDK for JavaScript

```
npm install aws-sdk
```

Amazon Ion 데이터 형식

```
npm install ion-js@4.0.0
```

`BigInt`의 순수 JavaScript 구현

```
npm install jsbi@3.1.1
```

드라이버를 사용하여 원장에 연결

그런 다음 드라이버를 가져와서 원장에 연결하는 데 사용할 수 있습니다. 다음 TypeScript 코드 예제에서는 지정된 원장 이름 및 AWS 리전에 대한 드라이버 인스턴스를 생성하는 방법을 보여줍니다.

3.x

```
import { Agent } from 'https';
import { QLDBSessionClientConfig } from "@aws-sdk/client-qldb-session";
import { QldbDriver, RetryConfig  } from 'amazon-qldb-driver-nodejs';
import { NodeHttpHandlerOptions } from "@aws-sdk/node-http-handler";

const maxConcurrentTransactions: number = 10;
const retryLimit: number = 4;

//Reuse connections with keepAlive
const lowLevelClientHttpOptions: NodeHttpHandlerOptions = {
    httpAgent: new Agent({
      maxSockets: maxConcurrentTransactions
    })
};

const serviceConfigurationOptions: QLDBSessionClientConfig = {
    region: "us-east-1"
};

//Use driver's default backoff function for this example (no second parameter
 provided to RetryConfig)
const retryConfig: RetryConfig = new RetryConfig(retryLimit);
const qldbDriver: QldbDriver = new QldbDriver("testLedger",
 serviceConfigurationOptions, lowLevelClientHttpOptions, maxConcurrentTransactions,
 retryConfig);

qldbDriver.getTableNames().then(function(tableNames: string[]) {
    console.log(tableNames);
});
```

2.x

```
import { Agent } from 'https';
import { QldbDriver, RetryConfig  } from 'amazon-qldb-driver-nodejs';

const maxConcurrentTransactions: number = 10;
const retryLimit: number = 4;

//Reuse connections with keepAlive
const agentForQldb: Agent = new Agent({
    keepAlive: true,
```

```
    maxSockets: maxConcurrentTransactions
});

const serviceConfigurationOptions = {
    region: "us-east-1",
    httpOptions: {
        agent: agentForQldb
    }
};

//Use driver's default backoff function for this example (no second parameter
 provided to RetryConfig)
const retryConfig: RetryConfig = new RetryConfig(retryLimit);
const qldbDriver: QldbDriver = new QldbDriver("testLedger",
 serviceConfigurationOptions, maxConcurrentTransactions, retryConfig);

qldbDriver.getTableNames().then(function(tableNames: string[]) {
    console.log(tableNames);
});
```

1.x

```
import { Agent } from 'https';
import { QldbDriver } from 'amazon-qldb-driver-nodejs';

const poolLimit: number = 10;
const retryLimit: number = 4;

//Reuse connections with keepAlive
const agentForQldb: Agent = new Agent({
    keepAlive: true,
    maxSockets: poolLimit
});

const serviceConfigurationOptions = {
    region: "us-east-1",
    httpOptions: {
        agent: agentForQldb
    }
};

const qldbDriver: QldbDriver = new QldbDriver("testLedger",
 serviceConfigurationOptions, retryLimit, poolLimit);
```

```
qldbDriver.getTableNames().then(function(tableNames: string[]) {
    console.log(tableNames);
});
```

원장에서 기본 데이터 트랜잭션을 실행하는 방법에 대한 단축 코드 예제는 Cookbook 참조를 참조하십시오.

설정 권장 사항

연결 유지를 통한 연결 재사용

QLDB Node.js 드라이버 v3

기본 Node.js HTTP/HTTPS 에이전트는 모든 새 요청에 대해 새로운 TCP 연결을 생성합니다. 새 연결을 설정하는 데 드는 비용을 피하기 위해 AWS SDK for JavaScript v3에서는 기본적으로 TCP 연결을 재사용합니다. 자세한 내용과 연결 재사용을 비활성화하는 방법을 알아보려면 AWS SDK for JavaScript 개발자 안내서의 Node.js에서 연결 유지를 이용해 연결 재사용을 참조하십시오.

Node.js용 QLDB 드라이버의 연결을 재사용하려면 기본 설정을 사용할 것을 권장합니다. 드라이버 초기화 중에 하위 수준 클라이언트 HTTP 옵션 `maxSockets`을 `maxConcurrentTransactions`에 설정한 값과 동일하게 설정합니다.

예를 들어, 다음 JavaScript 또는 TypeScript 코드를 참조하십시오.

JavaScript

```
const qldb = require('amazon-qldb-driver-nodejs');
const https = require('https');

//Replace this value as appropriate for your application
const maxConcurrentTransactions = 50;

const agentForQldb = new https.Agent({
    //Set this to the same value as `maxConcurrentTransactions`(previously called
 `poolLimit`)
    //Do not rely on the default value of `Infinity`
    "maxSockets": maxConcurrentTransactions
});

const lowLevelClientHttpOptions = {
```

```
    httpAgent: agentForQldb
}

let driver = new qldb.QldbDriver("testLedger", undefined, lowLevelClientHttpOptions,
 maxConcurrentTransactions);
```

TypeScript

```
import { Agent } from 'https';
import { NodeHttpHandlerOptions } from "@aws-sdk/node-http-handler";
import { QldbDriver } from 'amazon-qldb-driver-nodejs';

//Replace this value as appropriate for your application
const maxConcurrentTransactions: number = 50;

const agentForQldb: Agent = new Agent({
    //Set this to the same value as `maxConcurrentTransactions`(previously called
 `poolLimit`)
    //Do not rely on the default value of `Infinity`
    maxSockets: maxConcurrentTransactions
});

const lowLevelClientHttpOptions: NodeHttpHandlerOptions = {
    httpAgent: agentForQldb
};

let driver = new QldbDriver("testLedger", undefined, lowLevelClientHttpOptions,
 maxConcurrentTransactions);
```

QLDB Node.js 드라이버 v2

기본 Node.js HTTP/HTTPS 에이전트는 모든 새 요청에 대해 새로운 TCP 연결을 생성합니다. 새 연결 설정 비용이 발생하지 않게 하려면 기존 연결을 재사용할 것을 권장합니다.

Node.js용 QLDB 드라이버의 연결을 재사용하려면 다음 옵션 중 하나를 사용십시오.

- 드라이버 초기화 중에 다음과 같은 하위 수준 클라이언트 HTTP 옵션을 설정하십시오.
  - `keepAlive` – `true`
  - `maxSockets` - `maxConcurrentTransactions`에 설정한 것과 동일한 값

  예를 들어, 다음 JavaScript 또는 TypeScript 코드를 참조하십시오.

JavaScript

```
const qldb = require('amazon-qldb-driver-nodejs');
const https = require('https');

//Replace this value as appropriate for your application
const maxConcurrentTransactions = 50;

const agentForQldb = new https.Agent({
    "keepAlive": true,
    //Set this to the same value as `maxConcurrentTransactions`(previously called
 `poolLimit`)
    //Do not rely on the default value of `Infinity`
    "maxSockets": maxConcurrentTransactions
});

const serviceConfiguration = { "httpOptions": {
    "agent": agentForQldb
}};

let driver = new qldb.QldbDriver("testLedger", serviceConfiguration,
 maxConcurrentTransactions);
```

TypeScript

```
import { Agent } from 'https';
import { ClientConfiguration } from 'aws-sdk/clients/acm';
import { QldbDriver } from 'amazon-qldb-driver-nodejs';

//Replace this value as appropriate for your application
const maxConcurrentTransactions: number = 50;

const agentForQldb: Agent = new Agent({
    keepAlive: true,
    //Set this to the same value as `maxConcurrentTransactions`(previously called
 `poolLimit`)
    //Do not rely on the default value of `Infinity`
    maxSockets: maxConcurrentTransactions
});

const serviceConfiguration: ClientConfiguration = { httpOptions: {
    agent: agentForQldb
}};
```

```
let driver = new QldbDriver("testLedger", serviceConfiguration,
 maxConcurrentTransactions);
```

- 또는 AWS_NODEJS_CONNECTION_REUSE_ENABLED 환경 변수를 1로 설정할 수 있습니다. 자세한 정보는 AWS SDK for JavaScript 개발자 안내서의 Node.js에서 연결 유지를 통한 연결 재사용을 참조하십시오.

> **Note**
>
> 이 환경 변수를 설정하면 AWS SDK for JavaScript를 사용하는 모든 AWS 서비스에게 영향을 줍니다.

Node.js용 Amazon QLDB 드라이버 - 빠른 시작 자습서

이 자습서에서는 Node.js 용 Amazon QLDB 드라이버를 사용하여 간단한 애플리케이션을 설정하는 방법을 알아봅니다. 이 가이드에는 드라이버 설치 단계 및 기본적인 CRUD(생성, 읽기, 업데이트 및 삭제) 작업에 대한 간단한 JavaScript 및 TypeScript 코드 예가 포함되어 있습니다. 전체 샘플 애플리케이션에서 이러한 작업을 보여 주는 자세한 예를 보려면 Node.js 자습서 섹션을 참조하세요.

> **Note**
>
> 해당하는 경우 일부 단계에는 Node.js용 QLDB 드라이버의 지원되는 각 주요 버전마다 다른 코드 예가 있습니다.

주제

필수 조건

시작하기 전에 다음을 수행해야 합니다.

1. Node.js. 드라이버에 대한 사전 조건을 아직 완료하지 않은 경우, 완료하세요. 여기에는 AWS 가입, 개발을 위한 프로그램에 입각한 액세스 권한 부여, Node.js. 설치가 포함됩니다.
2. `quick-start`라는 명칭의 원장을 생성합니다.

   원장 생성 방법을 알아보려면 콘솔 시작하기의 Amazon QLDB 원장의 기본 작업 또는 1단계: 새 원장 생성 섹션을 참조하세요.

TypeScript를 사용하는 경우 다음 설정 단계도 수행해야 합니다.

TypeScript 사용

TypeScript를 설치하려면

1. TypeScript 패키지를 설치합니다. QLDB 드라이버는 TypeScript 3.8.x에서 실행됩니다.

   ```
   $ npm install --global typescript@3.8.0
   ```

2. 패키지를 설치한 후 다음 명령을 실행하여 TypeScript 컴파일러가 설치되었는지 확인합니다.

   ```
   $ tsc --version
   ```

다음 단계에서 코드를 실행하려면 먼저 다음과 같이 TypeScript 파일을 실행 가능한 JavaScript 코드로 변환해야 합니다.

```
$ tsc app.ts; node app.js
```

1단계: 프로젝트 설정

먼저 Node.js 프로젝트를 설정합니다.

1. 애플리케이션을 위한 폴더를 생성합니다.

   ```
   $ mkdir myproject
   $ cd myproject
   ```

2. 프로젝트를 초기화하려면 다음 npm 명령을 입력하고 설정 중에 나타나는 질문에 답하세요. 대부분의 질문에 기본값을 사용할 수 있습니다.

```
$ npm init
```

3. Node.js용 Amazon QLDB 드라이버를 설치합니다.

   - 버전 3.x 사용

```
$ npm install amazon-qldb-driver-nodejs --save
```

   - 버전 2.x 사용

```
$ npm install amazon-qldb-driver-nodejs@2.2.0 --save
```

   - 버전 1.x 사용

```
$ npm install amazon-qldb-driver-nodejs@1.0.0 --save
```

4. 드라이버의 피어 종속 항목을 설치합니다.

   - 버전 3.x 사용

```
$ npm install @aws-sdk/client-qldb-session --save
$ npm install ion-js --save
$ npm install jsbi --save
```

   - 버전 2.x 또는 1.x 사용

```
$ npm install aws-sdk --save
$ npm install ion-js@4.0.0 --save
$ npm install jsbi@3.1.1 --save
```

5. `app.js`라는 명칭의 JavaScript용 새 파일 또는 `app.ts`라는 명칭의 TypeScript용 새 파일을 생성합니다.

   그런 다음 다음 단계의 코드 예를 점진적으로 추가하여 몇 가지 기본 CRUD 작업을 시도해 보세요. 또는 단계별 자습서를 건너뛰고 전체 애플리케이션을 실행할 수도 있습니다.

2단계: 드라이버 초기화

`quick-start`라는 명칭의 원장에 연결되는 드라이버의 인스턴스를 초기화합니다. 다음 코드를 `app.js` 또는 `app.ts` 파일에 추가합니다.

버전 3.x 사용

JavaScript

```
var qldb = require('amazon-qldb-driver-nodejs');
var https = require('https');

function main() {
    const maxConcurrentTransactions = 10;
    const retryLimit = 4;

    const agentForQldb = new https.Agent({
        maxSockets: maxConcurrentTransactions
    });

    const lowLevelClientHttpOptions = {
       httpAgent: agentForQldb
    }

    const serviceConfigurationOptions = {
        region: "us-east-1"
    };

    // Use driver's default backoff function for this example (no second parameter
 provided to RetryConfig)
    var retryConfig = new qldb.RetryConfig(retryLimit);
    var driver = new qldb.QldbDriver("quick-start", serviceConfigurationOptions,
 lowlevelClientHttpOptions, maxConcurrentTransactions, retryConfig);
}

main();
```

TypeScript

```
import { Agent } from "https";
import { NodeHttpHandlerOptions } from "@aws-sdk/node-http-handler";
import { QLDBSessionClientConfig } from "@aws-sdk/client-qldb-session";
import { QldbDriver, RetryConfig } from "amazon-qldb-driver-nodejs";
```

```
function main(): void {
    const maxConcurrentTransactions: number = 10;
    const agentForQldb: Agent = new Agent({
        maxSockets: maxConcurrentTransactions
    });

    const lowLevelClientHttpOptions: NodeHttpHandlerOptions = {
      httpAgent: agentForQldb
    };

    const serviceConfigurationOptions: QLDBSessionClientConfig = {
        region: "us-east-1"
    };

    const retryLimit: number = 4;
    // Use driver's default backoff function for this example (no second parameter
 provided to RetryConfig)
    const retryConfig: RetryConfig = new RetryConfig(retryLimit);
    const driver: QldbDriver = new QldbDriver("quick-start",
 serviceConfigurationOptions, lowLevelClientHttpOptions, maxConcurrentTransactions,
 retryConfig);
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> - 이 코드 예에서는 *us-east-1*을 원장을 생성한 AWS 리전으로 바꿉니다.
> - 간소화를 위해 이 가이드의 나머지 코드 예에서는 다음 예 버전 1.x에 지정된 대로 기본 설정이 있는 드라이버를 사용합니다. 사용자 지정 `RetryConfig` 대신 자체 드라이버 인스턴스를 사용할 수도 있습니다.

버전 2.x 사용

JavaScript

```
var qldb = require('amazon-qldb-driver-nodejs');
var https = require('https');

function main() {
    var maxConcurrentTransactions = 10;
    var retryLimit = 4;

    var agentForQldb = new https.Agent({
        keepAlive: true,
        maxSockets: maxConcurrentTransactions
    });

    var serviceConfigurationOptions = {
        region: "us-east-1",
        httpOptions: {
            agent: agentForQldb
        }
    };

    // Use driver's default backoff function for this example (no second parameter
 provided to RetryConfig)
    var retryConfig = new qldb.RetryConfig(retryLimit);
    var driver = new qldb.QldbDriver("quick-start", serviceConfigurationOptions,
 maxConcurrentTransactions, retryConfig);
}

main();
```

TypeScript

```
import { QldbDriver, RetryConfig } from "amazon-qldb-driver-nodejs";
import { ClientConfiguration } from "aws-sdk/clients/acm";
import { Agent } from "https";

function main(): void {
    const maxConcurrentTransactions: number = 10;
    const agentForQldb: Agent = new Agent({
        keepAlive: true,
        maxSockets: maxConcurrentTransactions
```

```
    });
    const serviceConfigurationOptions: ClientConfiguration = {
        region: "us-east-1",
        httpOptions: {
            agent: agentForQldb
        }
    };
    const retryLimit: number = 4;
    // Use driver's default backoff function for this example (no second parameter
 provided to RetryConfig)
    const retryConfig: RetryConfig = new RetryConfig(retryLimit);
    const driver: QldbDriver = new QldbDriver("quick-start",
 serviceConfigurationOptions, maxConcurrentTransactions, retryConfig);
}

if (require.main === module) {
    main();
}
```

Note

- 이 코드 예에서는 *us-east-1*을 원장을 생성한 AWS 리전으로 바꿉니다.
- 버전 2.x에는 `QldbDriver` 초기화를 위한 새로운 옵션 파라미터 `RetryConfig`가 도입되었습니다.
- 간소화를 위해 이 가이드의 나머지 코드 예에서는 다음 예 버전 1.x에 지정된 대로 기본 설정이 있는 드라이버를 사용합니다. 사용자 지정 `RetryConfig` 대신 자체 드라이버 인스턴스를 사용할 수도 있습니다.
- 이 코드 예는 연결 유지 옵션을 설정하여 기존 연결을 재사용하는 드라이버를 초기화합니다. 자세한 설명은 Node.js 드라이버에 대한 설정 권장 사항 섹션을 참조하세요.

버전 1.x 사용

JavaScript

```
const qldb = require('amazon-qldb-driver-nodejs');

function main() {
    // Use default settings
```

```
    const driver = new qldb.QldbDriver("quick-start");
}

main();
```

TypeScript

```
import { QldbDriver } from "amazon-qldb-driver-nodejs";

function main(): void {
    // Use default settings
    const driver: QldbDriver = new QldbDriver("quick-start");
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> AWS_REGION 환경 변수를 설정하여 지역을 지정할 수 있습니다. 자세한 설명은 AWS SDK for JavaScript 개발자 가이드의 AWS 리전의 설정을 참조하세요.

3단계: 테이블 및 인덱스 생성

다음 코드 예는 CREATE TABLE 및 CREATE INDEX 문을 실행하는 방법을 보여줍니다.

1. People라는 일므의 표를 생성하는 다음 함수를 추가합니다.

   JavaScript

   ```
   async function createTable(txn) {
       await txn.execute("CREATE TABLE People");
   }
   ```

   TypeScript

   ```
   async function createTable(txn: TransactionExecutor): Promise<void> {
       await txn.execute("CREATE TABLE People");
   ```

```
}
```

2. People 표의 firstName 필드에 대한 인덱스를 만드는 다음 함수를 추가합니다. 인덱스는 쿼리 성능을 최적화하고 OCC(낙관적 동시성 제어) 충돌 예외를 제한하는 데 필요합니다.

   JavaScript

   ```
   async function createIndex(txn) {
       await txn.execute("CREATE INDEX ON People (firstName)");
   }
   ```

   TypeScript

   ```
   async function createIndex(txn: TransactionExecutor): Promise<void> {
       await txn.execute("CREATE INDEX ON People (firstName)");
   }
   ```

3. main 함수에서는 먼저 createTable를 호출한 다음 createIndex를 호출합니다.

   JavaScript

   ```
   const qldb = require('amazon-qldb-driver-nodejs');

   async function main() {
       // Use default settings
       const driver = new qldb.QldbDriver("quick-start");

       await driver.executeLambda(async (txn) => {
           console.log("Create table People");
           await createTable(txn);
           console.log("Create index on firstName");
           await createIndex(txn);
       });

       driver.close();
   }

   main();
   ```

   TypeScript

   ```
   import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";
   ```

```
async function main(): Promise<void> {
    // Use default settings
    const driver: QldbDriver = new QldbDriver("quick-start");

    await driver.executeLambda(async (txn: TransactionExecutor) => {
        console.log("Create table People");
        await createTable(txn);
        console.log("Create index on firstName");
        await createIndex(txn);
    });

    driver.close();
}

if (require.main === module) {
    main();
}
```

4. 코드를 실행하여 표와 인덱스를 생성합니다.

   JavaScript

   ```
   $ node app.js
   ```

   TypeScript

   ```
   $ tsc app.ts; node app.js
   ```

4단계: 문서 삽입

다음 코드 예에서는 `INSERT` 문을 실행하는 방법을 보여줍니다. QLDB는 PartiQL 쿼리 언어(SQL 호환) 및 Amazon Ion 데이터 형식(JSON의 상위 집합)을 지원합니다.

1. `People` 표에 문서를 삽입하는 다음 함수를 추가합니다.

   JavaScript

   ```
   async function insertDocument(txn) {
       const person = {
           firstName: "John",
   ```

```
        lastName: "Doe",
        age: 42
    };
    await txn.execute("INSERT INTO People ?", person);
}
```

TypeScript

```
async function insertDocument(txn: TransactionExecutor): Promise<void> {
    const person: Record<string, any> = {
        firstName: "John",
        lastName: "Doe",
        age: 42
    };
    await txn.execute("INSERT INTO People ?", person);
}
```

이 예에서는 물음표(?)를 변수 자리 표시자로 사용하여 문서 정보를 해당 문에 전달합니다. 이 `execute` 메서드는 Amazon Ion 타입과 Node.js 네이티브 타입 모두의 값을 지원합니다.

> **Tip**
>
> 단일 INSERT 문을 사용하여 여러 문서를 삽입하려면 다음과 같이 목록 타입의 파라미터를 해당 문에 전달할 수 있습니다.
>
> ```
> // people is a list
> txn.execute("INSERT INTO People ?", people);
> ```
>
> 목록을 전달할 때 변수 자리 표시자(?)를 이중 꺾쇠 괄호(`<<...>>`)로 묶지 마세요. 수동 PartiQL 문에서 이중 꺾쇠 괄호는 백으로 알려진 정렬되지 않은 모음을 의미합니다.

2. `main`함수에서 `createTable` 및 `createIndex` 호출을 제거하고 `insertDocument`에 대한 호출을 추가합니다.

JavaScript

```
const qldb = require('amazon-qldb-driver-nodejs');

async function main() {
    // Use default settings
```

```
    const driver = new qldb.QldbDriver("quick-start");

    await driver.executeLambda(async (txn) => {
        console.log("Insert document");
        await insertDocument(txn);
    });

    driver.close();
}

main();
```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";

async function main(): Promise<void> {
    // Use default settings
    const driver: QldbDriver = new QldbDriver("quick-start");

    await driver.executeLambda(async (txn: TransactionExecutor) => {
        console.log("Insert document");
        await insertDocument(txn);
    });

    driver.close();
}

if (require.main === module) {
    main();
}
```

5단계: 문서 쿼리

다음 코드 예에서는 SELECT 문을 실행하는 방법을 보여줍니다.

1. People 표에서 문서를 쿼리하는 다음 함수를 추가합니다.

   JavaScript

```
async function fetchDocuments(txn) {
```

```
    return await txn.execute("SELECT firstName, age, lastName FROM People WHERE
 firstName = ?", "John");
}
```

TypeScript

```
async function fetchDocuments(txn: TransactionExecutor): Promise<dom.Value[]> {
    return (await txn.execute("SELECT firstName, age, lastName FROM People WHERE
 firstName = ?", "John")).getResultList();
}
```

2. `main` 함수에서 `insertDocument` 호출 뒤에 다음 `fetchDocuments` 호출을 추가합니다.

JavaScript

```
const qldb = require('amazon-qldb-driver-nodejs');

async function main() {
    // Use default settings
    const driver = new qldb.QldbDriver("quick-start");

    var resultList = await driver.executeLambda(async (txn) => {
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        var result = await fetchDocuments(txn);
        return result.getResultList();
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}

main();
```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { dom } from "ion-js";

async function main(): Promise<void> {
```

```
    // Use default settings
    const driver: QldbDriver = new QldbDriver("quick-start");

    const resultList: dom.Value[] = await driver.executeLambda(async (txn:
 TransactionExecutor) => {
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        return await fetchDocuments(txn);
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}

if (require.main === module) {
    main();
}
```

6단계: 문서 업데이트

다음 코드 예에서는 UPDATE 문을 실행하는 방법을 보여줍니다.

1. lastName를 "Stiles"으로 변경함으로써 People 표의 문서를 업데이트하는 다음 함수를 추가합니다.

   JavaScript

   ```
   async function updateDocuments(txn) {
       await txn.execute("UPDATE People SET lastName = ? WHERE firstName = ?",
    "Stiles", "John");
   }
   ```

   TypeScript

   ```
   async function updateDocuments(txn: TransactionExecutor): Promise<void> {
       await txn.execute("UPDATE People SET lastName = ? WHERE firstName = ?",
    "Stiles", "John");
   }
   ```

2. `main` 함수에서 `fetchDocuments` 호출 뒤에 다음 `updateDocuments` 호출을 추가합니다. 그런 다음 `fetchDocuments`를 다시 호출하여 업데이트된 결과를 확인합니다.

JavaScript

```
const qldb = require('amazon-qldb-driver-nodejs');

async function main() {
    // Use default settings
    const driver = new qldb.QldbDriver("quick-start");

    var resultList = await driver.executeLambda(async (txn) => {
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        await fetchDocuments(txn);
        console.log("Update document");
        await updateDocuments(txn);
        console.log("Fetch document after update");
        var result = await fetchDocuments(txn);
        return result.getResultList();
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}

main();
```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { dom } from "ion-js";

async function main(): Promise<void> {
    // Use default settings
    const driver: QldbDriver = new QldbDriver("quick-start");

    const resultList: dom.Value[] = await driver.executeLambda(async (txn:
 TransactionExecutor) => {
        console.log("Insert document");
        await insertDocument(txn);
```

```
        console.log("Fetch document");
        await fetchDocuments(txn);
        console.log("Update document");
        await updateDocuments(txn);
        console.log("Fetch document after update");
        return await fetchDocuments(txn);
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}

if (require.main === module) {
    main();
}
```

3. 코드를 실행하여 문서를 삽입, 쿼리 및 업데이트합니다.

   JavaScript

   ```
   $ node app.js
   ```

   TypeScript

   ```
   $ tsc app.ts; node app.js
   ```

전체 애플리케이션 실행

다음 코드 예는 `app.js` 및 `app.ts`의 전체 버전입니다. 이전 단계를 개별적으로 수행하는 대신 이 코드를 처음부터 끝까지 실행할 수도 있습니다. 이 애플리케이션은 `quick-start`이라는 명칭의 원장에 대한 몇 가지 기본 CRUD 작업을 보여줍니다.

> **Note**
>
> 이 코드를 실행하기 전에 `quick-start` 원장에 `People`이라는 명칭의 활성 테이블이 아직 없는지 확인하세요.

JavaScript

```
const qldb = require('amazon-qldb-driver-nodejs');

async function createTable(txn) {
    await txn.execute("CREATE TABLE People");
}

async function createIndex(txn) {
    await txn.execute("CREATE INDEX ON People (firstName)");
}

async function insertDocument(txn) {
    const person = {
        firstName: "John",
        lastName: "Doe",
        age: 42
    };
    await txn.execute("INSERT INTO People ?", person);
}

async function fetchDocuments(txn) {
    return await txn.execute("SELECT firstName, age, lastName FROM People WHERE
 firstName = ?", "John");
}

async function updateDocuments(txn) {
    await txn.execute("UPDATE People SET lastName = ? WHERE firstName = ?",
 "Stiles", "John");
}

async function main() {
    // Use default settings
    const driver = new qldb.QldbDriver("quick-start");

    var resultList = await driver.executeLambda(async (txn) => {
        console.log("Create table People");
        await createTable(txn);
        console.log("Create index on firstName");
        await createIndex(txn);
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        await fetchDocuments(txn);
```

```
        console.log("Update document");
        await updateDocuments(txn);
        console.log("Fetch document after update");
        var result = await fetchDocuments(txn);
        return result.getResultList();
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}

main();
```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { dom } from "ion-js";

async function createTable(txn: TransactionExecutor): Promise<void> {
    await txn.execute("CREATE TABLE People");
}

async function createIndex(txn: TransactionExecutor): Promise<void> {
    await txn.execute("CREATE INDEX ON People (firstName)");
}

async function insertDocument(txn: TransactionExecutor): Promise<void> {
    const person: Record<string, any> = {
        firstName: "John",
        lastName: "Doe",
        age: 42
    };
    await txn.execute("INSERT INTO People ?", person);
}

async function fetchDocuments(txn: TransactionExecutor): Promise<dom.Value[]> {
    return (await txn.execute("SELECT firstName, age, lastName FROM People WHERE
 firstName = ?", "John")).getResultList();
}

async function updateDocuments(txn: TransactionExecutor): Promise<void> {
```

```
    await txn.execute("UPDATE People SET lastName = ? WHERE firstName = ?",
 "Stiles", "John");
};

async function main(): Promise<void> {
    // Use default settings
    const driver: QldbDriver = new QldbDriver("quick-start");

    const resultList: dom.Value[] = await driver.executeLambda(async (txn:
 TransactionExecutor) => {
        console.log("Create table People");
        await createTable(txn);
        console.log("Create index on firstName");
        await createIndex(txn);
        console.log("Insert document");
        await insertDocument(txn);
        console.log("Fetch document");
        await fetchDocuments(txn);
        console.log("Update document");
        await updateDocuments(txn);
        console.log("Fetch document after update");
        return await fetchDocuments(txn);
    });

    // Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    driver.close();
}

if (require.main === module) {
    main();
}
```

전체 애플리케이션을 실행하려면 다음 명령을 입력하십시오.

JavaScript

```
$ node app.js
```

TypeScript

```
$ tsc app.ts; node app.js
```

Node.js용 Amazon QLDB 드라이버 - Cookbook 참조

이 참조 가이드는 Node.js용 Amazon QLDB 드라이버의 일반적인 사용 사례를 보여줍니다. 이 가이드는 드라이버를 사용하여 기본 CRUD(생성, 읽기, 업데이트, 삭제) 작업을 실행하는 방법을 보여주는 JavaScript 및 TypeScript 코드 예제를 안내합니다. 또한 Amazon Ion 데이터를 처리하기 위한 코드 예제도 포함되어 있습니다. 또한 이 가이드에서는 트랜잭션에 멱등성을 부여하고 고유성 제약하는 모범 사례를 중점적으로 설명합니다.

목차

드라이버 가져오기

다음 코드 예제에서는 드라이브를 가져옵니다.

JavaScript

```
var qldb = require('amazon-qldb-driver-nodejs');
var ionjs = require('ion-js');
```

TypeScript

```
import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { dom, dumpBinary, load } from "ion-js";
```

> **Note**
>
> 이 예제에서는 Amazon Ion 패키지(`ion-js`)도 가져옵니다. 이 참조에서 일부 데이터 작업을 실행할 때 Ion 데이터를 처리하려면 이 패키지가 필요합니다. 자세한 내용은 Amazon Ion 작업 섹션을 참조하세요.

드라이버 인스턴스화

다음 코드 예제는 기본 설정을 사용하여 지정된 원장 이름에 연결하는 드라이버 인스턴스를 만듭니다.

JavaScript

```
const qldbDriver = new qldb.QldbDriver("vehicle-registration");
```

TypeScript

```
const qldbDriver: QldbDriver = new QldbDriver("vehicle-registration");
```

CRUD 작업

QLDB는 트랜잭션의 일부로 CRUD(생성, 읽기, 업데이트, 삭제) 작업을 실행합니다.

> **Warning**
>
> 가장 좋은 방법은 쓰기 트랜잭션이 완전한 멱등성을 부여하는 것입니다.

트랜잭션에 멱등성 부여하기

재시도 시 예상치 못한 부작용이 발생하지 않도록 쓰기 트랜잭션에 멱등성을 부여하는 것이 좋습니다. 여러 번 실행하여 매번 동일한 결과를 생성할 수 있는 트랜잭션은 멱등성을 가집니다.

이름이 `Person`인 테이블에 문서를 삽입하는 트랜잭션을 예로 들어 보겠습니다. 트랜잭션은 먼저 문서가 테이블에 이미 존재하는지 여부를 확인해야 합니다. 이렇게 확인하지 않으면 테이블에 문서가 중복될 수 있습니다.

QLDB가 서버 측에서 트랜잭션을 성공적으로 커밋했지만 응답을 기다리는 동안 클라이언트 제한 시간이 초과되었다고 가정해 보겠습니다. 트랜잭션이 멱등성을 가지지 않는 경우 재시도 시 동일한 문서가 두 번 이상 삽입될 수 있습니다.

인덱스를 사용하여 전체 테이블 스캔 방지

인덱싱된 필드 또는 문서 ID(예: `WHERE indexedField = 123` 또는 `WHERE indexedField IN (456, 789)`)에서 동등 연산자를 사용하여 `WHERE` 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 이 인덱싱된 조회가 없으면 QLDB는 테이블 스캔을 수행해야 하며, 이로 인해 트랜잭션 제한 시간이 초과되거나 OCC(낙관적 동시성 제어) 충돌이 발생할 수 있습니다.

OCC에 대한 자세한 내용은 Amazon QLDB 동시성 모델 단원을 참조하세요.

암시적으로 생성된 트랜잭션

QldbDriver.executeLambda 메서드는 TransactionExecutor의 인스턴스를 수신하는 Lambda 함수를 받아들이며, 이 함수를 사용하여 명령문을 실행할 수 있습니다. `TransactionExecutor`의 인스턴스는 암시적으로 생성된 트랜잭션을 래핑합니다.

트랜잭션 실행자의 execute 메서드를 사용하여 lambda 함수 내에서 명령문을 실행할 수 있습니다. 드라이버는 Lambda 함수가 반환될 때 트랜잭션을 암시적으로 커밋합니다.

> **Note**
>
> 이 `execute` 메서드는 Amazon Ion 유형과 Node.js 네이티브 유형을 모두 지원합니다. Node.js 네이티브 유형을 인수로 `execute`에 전달하면 드라이버가 `ion-js` 패키지를 사용하여 해당 형식을 Ion 유형으로 변환합니다(지정된 Node.js 데이터 유형에 대한 변환이 지원되는 경우). 지원되는 데이터 유형 및 변환 규칙은 Ion JavaScript DOM README를 참조하세요.

다음 섹션에서는 기본 CRUD 작업을 실행하고, 사용자 지정 재시도 로직을 지정하고, 고유성 제약 조건을 구현하는 방법을 보여줍니다.

목차

테이블 생성

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        await txn.execute("CREATE TABLE Person");
    });
})();
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        await txn.execute('CREATE TABLE Person');
    });
}());
```

인덱스 생성

JavaScript

```
(async function() {
```

```
    await qldbDriver.executeLambda(async (txn) => {
        await txn.execute("CREATE INDEX ON Person (GovId)");
    });
})();
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        await txn.execute('CREATE INDEX ON Person (GovId)');
    });
}());
```

문서 읽기

JavaScript

```
(async function() {
    // Assumes that Person table has documents as follows:
    // { "GovId": "TOYENC486FH", "FirstName": "Brent" }
    await qldbDriver.executeLambda(async (txn) => {
        const results = (await txn.execute("SELECT * FROM Person WHERE GovId =
 'TOYENC486FH'")).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    // Assumes that Person table has documents as follows:
    // { "GovId": "TOYENC486FH", "FirstName": "Brent" }
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const results: dom.Value[] = (await txn.execute("SELECT * FROM Person WHERE
 GovId = 'TOYENC486FH'")).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
```

```
        }
    });
}());
```

쿼리 파라미터 사용

다음 코드 예제는 네이티브 유형 쿼리 파라미터를 사용합니다.

JavaScript

```
(async function() {
    // Assumes that Person table has documents as follows:
    // { "GovId": "TOYENC486FH", "FirstName": "Brent" }
    await qldbDriver.executeLambda(async (txn) => {
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ?',
 'TOYENC486FH')).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    // Assumes that Person table has documents as follows:
    // { "GovId": "TOYENC486FH", "FirstName": "Brent" }
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
 GovId = ?', 'TOYENC486FH')).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
}());
```

다음 코드 예제는 Ion 유형 쿼리 파라미터를 사용합니다.

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        const govId = ionjs.load("TOYENC486FH");

        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ?',
 govId)).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const govId: dom.Value = load("TOYENC486FH");

        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
 GovId = ?', govId)).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
}());
```

다음 코드 예제는 여러 쿼리 파라미터를 사용합니다.

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ? AND
 FirstName = ?', 'TOYENC486FH', 'Brent')).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
```

```
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
 GovId = ? AND FirstName = ?', 'TOYENC486FH', 'Brent')).getResultList();
        for (let result of results) {
            console.log(result.get('GovId')); // prints [String: 'TOYENC486FH']
            console.log(result.get('FirstName')); // prints [String: 'Brent']
        }
    });
}());
```

다음 코드 예제는 쿼리 파라미터 목록을 사용합니다.

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        const govIds = ['TOYENC486FH','LOGANB486CG','LEWISR261LL'];
        /*
        Assumes that Person table has documents as follows:
        { "GovId": "TOYENC486FH", "FirstName": "Brent" }
        { "GovId": "LOGANB486CG", "FirstName": "Brent" }
        { "GovId": "LEWISR261LL", "FirstName": "Raul" }
        */
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId IN
 (?,?,?)', ...govIds)).getResultList();
        for (let result of results) {
            console.log(result.get('GovId'));
            console.log(result.get('FirstName'));
            /*
            prints:
            [String: 'TOYENC486FH']
            [String: 'Brent']
            [String: 'LOGANB486CG']
            [String: 'Brent']
            [String: 'LEWISR261LL']
            [String: 'Raul']
```

```
            */
        }
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const govIds: string[] = ['TOYENC486FH','LOGANB486CG','LEWISR261LL'];
        /*
        Assumes that Person table has documents as follows:
        { "GovId": "TOYENC486FH", "FirstName": "Brent" }
        { "GovId": "LOGANB486CG", "FirstName": "Brent" }
        { "GovId": "LEWISR261LL", "FirstName": "Raul" }
        */
        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
 GovId IN (?,?,?)', ...govIds)).getResultList();
        for (let result of results) {
            console.log(result.get('GovId'));
            console.log(result.get('FirstName'));
            /*
            prints:
            [String: 'TOYENC486FH']
            [String: 'Brent']
            [String: 'LOGANB486CG']
            [String: 'Brent']
            [String: 'LEWISR261LL']
            [String: 'Raul']
            */
        }
    });
}());
```

> **Note**
>
> 인덱싱된 조회 없이 쿼리를 실행하면 전체 테이블 스캔이 호출됩니다. 이 예제에서는 성능을 최적화하기 위해 `GovId` 필드에 인덱스를 사용하는 것이 좋습니다. `GovId`에 인덱스를 사용하지 않으면 쿼리에 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

문서 삽입하기

다음 코드 예제는 네이티브 데이터 유형을 삽입합니다.

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        // Check if doc with GovId:TOYENC486FH exists
        // This is critical to make this transaction idempotent
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ?',
 'TOYENC486FH')).getResultList();
        // Insert the document after ensuring it doesn't already exist
        if (results.length == 0) {
            const doc = {
                'FirstName': 'Brent',
                'GovId': 'TOYENC486FH',
            };
            await txn.execute('INSERT INTO Person ?', doc);
        }
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        // Check if doc with GovId:TOYENC486FH exists
        // This is critical to make this transaction idempotent
        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
 GovId = ?', 'TOYENC486FH')).getResultList();
        // Insert the document after ensuring it doesn't already exist
        if (results.length == 0) {
            const doc: Record<string, string> = {
                'FirstName': 'Brent',
                'GovId': 'TOYENC486FH',
            };
            await txn.execute('INSERT INTO Person ?', doc);
        }
    });
}());
```

다음 코드 예제는 Ion 데이터 유형을 삽입합니다.

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        // Check if doc with GovId:TOYENC486FH exists
        // This is critical to make this transaction idempotent
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ?',
 'TOYENC486FH')).getResultList();
        // Insert the document after ensuring it doesn't already exist
        if (results.length == 0) {
            const doc = {
                'FirstName': 'Brent',
                'GovId': 'TOYENC486FH',
            };
            // Create a sample Ion doc
            const ionDoc = ionjs.load(ionjs.dumpBinary(doc));

            await txn.execute('INSERT INTO Person ?', ionDoc);
        }
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        // Check if doc with GovId:TOYENC486FH exists
        // This is critical to make this transaction idempotent
        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
 GovId = ?', 'TOYENC486FH')).getResultList();
        // Insert the document after ensuring it doesn't already exist
        if (results.length == 0) {
            const doc: Record<string, string> = {
                'FirstName': 'Brent',
                'GovId': 'TOYENC486FH',
            };
            // Create a sample Ion doc
            const ionDoc: dom.Value = load(dumpBinary(doc));

            await txn.execute('INSERT INTO Person ?', ionDoc);
        }
```

```
    });
}());
```

이 트랜잭션은 문서를 Person 테이블에 삽입합니다. 삽입하기 전에 먼저 문서가 테이블에 이미 있는지 확인합니다. 이 검사를 통해 트랜잭션은 본질적으로 멱등성을 가지게 됩니다. 이 트랜잭션을 여러 번 실행하더라도 의도하지 않은 부작용이 발생하지는 않습니다.

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 GovId 필드에 인덱스를 사용하는 것이 좋습니다. GovId에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

하나의 명령문에 여러 문서 삽입

단일 INSERT 문을 사용하여 여러 문서를 삽입하려면 다음과 같이 목록 타입의 파라미터를 해당 문에 전달할 수 있습니다.

```
// people is a list
txn.execute("INSERT INTO People ?", people);
```

목록을 전달할 때 변수 자리 표시자(?)를 이중 꺾쇠 괄호(<<...>>)로 묶지 마세요. 수동 PartiQL 문에서 이중 꺾쇠 괄호는 백으로 알려진 정렬되지 않은 모음을 의미합니다.

문서 업데이트

다음 코드 예제는 네이티브 데이터 유형을 사용합니다.

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        await txn.execute('UPDATE Person SET FirstName = ? WHERE GovId = ?', 'John',
 'TOYENC486FH');
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        await txn.execute('UPDATE Person SET FirstName = ? WHERE GovId = ?', 'John',
 'TOYENC486FH');
    });
}());
```

다음 코드 예제는 Ion 데이터 유형을 사용합니다.

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        const firstName = ionjs.load("John");
        const govId = ionjs.load("TOYENC486FH");

        await txn.execute('UPDATE Person SET FirstName = ? WHERE GovId = ?',
 firstName, govId);
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const firstName: dom.Value = load("John");
        const govId: dom.Value = load("TOYENC486FH");

        await txn.execute('UPDATE Person SET FirstName = ? WHERE GovId = ?',
 firstName, govId);
    });
}());
```

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 `GovId` 필드에 인덱스를 사용하는 것이 좋습니다. `GovId`에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

문서 삭제

다음 코드 예제는 네이티브 데이터 유형을 사용합니다.

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        await txn.execute('DELETE FROM Person WHERE GovId = ?', 'TOYENC486FH');
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        await txn.execute('DELETE FROM Person WHERE GovId = ?', 'TOYENC486FH');
    });
}());
```

다음 코드 예제는 Ion 데이터 유형을 사용합니다.

JavaScript

```
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        const govId = ionjs.load("TOYENC486FH");

        await txn.execute('DELETE FROM Person WHERE GovId = ?', govId);
    });
}());
```

TypeScript

```
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        const govId: dom.Value = load("TOYENC486FH");

        await txn.execute('DELETE FROM Person WHERE GovId = ?', govId);
    });
}());
```

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 GovId 필드에 인덱스를 사용하는 것이 좋습니다. GovId에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

하나의 트랜잭션에서 여러 명령문 실행

TypeScript

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
 because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
 or not.
async function insureCar(driver: QldbDriver, vin: string): Promise<boolean> {

    return await driver.executeLambda(async (txn: TransactionExecutor) => {
        const results: dom.Value[] = (await txn.execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
 vin)).getResultList();

        if (results.length > 0) {
            await txn.execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin);
            return true;
        }
        return false;
    });
};
```

재시도 로직

드라이버의 executeLambda 메서드에는 재시도 가능한 예외(예: 시간 초과 또는 OCC 충돌)가 발생할 경우 트랜잭션을 재시도하는 재시도 메커니즘이 내장되어 있습니다. 최대 재시도 횟수와 백오프 전략을 구성할 수 있습니다.

기본 재시도 제한은 4이며, 기본 백오프 전략은 10밀리초 단위의 defaultBackoffFunction입니다. RetryConfig 인스턴스를 사용하여 드라이버 인스턴스별 및 트랜잭션별 재시도 구성을 설정할 수 있습니다.

다음 코드 예제는 드라이버 인스턴스에 대한 사용자 지정 재시도 제한 및 사용자 지정 백오프 전략을 사용하여 재시도 로직을 지정합니다.

JavaScript

```
var qldb = require('amazon-qldb-driver-nodejs');

// Configuring retry limit to 2
const retryConfig = new qldb.RetryConfig(2);
const qldbDriver = new qldb.QldbDriver("test-ledger", undefined, undefined,
 retryConfig);

// Configuring a custom backoff which increases delay by 1s for each attempt.
const customBackoff = (retryAttempt, error, transactionId) => {
    return 1000 * retryAttempt;
};

const retryConfigCustomBackoff = new qldb.RetryConfig(2, customBackoff);
const qldbDriverCustomBackoff = new qldb.QldbDriver("test-ledger", undefined,
 undefined, retryConfigCustomBackoff);
```

TypeScript

```
import { BackoffFunction, QldbDriver, RetryConfig } from "amazon-qldb-driver-nodejs"

// Configuring retry limit to 2
const retryConfig: RetryConfig = new RetryConfig(2);
const qldbDriver: QldbDriver = new QldbDriver("test-ledger", undefined, undefined,
 retryConfig);

// Configuring a custom backoff which increases delay by 1s for each attempt.
const customBackoff: BackoffFunction = (retryAttempt: number, error: Error,
 transactionId: string) => {
```

```
    return 1000 * retryAttempt;
};

const retryConfigCustomBackoff: RetryConfig = new RetryConfig(2, customBackoff);
const qldbDriverCustomBackoff: QldbDriver = new QldbDriver("test-ledger", undefined,
 undefined, retryConfigCustomBackoff);
```

다음 코드 예제는 특정 Lambda 실행에 대한 사용자 지정 재시도와 사용자 지정 백오프 전략을 사용하여 재시도 로직을 지정합니다. `executeLambda`에 대한 이 구성은 드라이버 인스턴스에 설정된 재시도 로직을 재정의합니다.

JavaScript

```
var qldb = require('amazon-qldb-driver-nodejs');

// Configuring retry limit to 2
const retryConfig1 = new qldb.RetryConfig(2);
const qldbDriver = new qldb.QldbDriver("test-ledger", undefined, undefined,
 retryConfig1);

// Configuring a custom backoff which increases delay by 1s for each attempt.
const customBackoff = (retryAttempt, error, transactionId) => {
    return 1000 * retryAttempt;
};

const retryConfig2 = new qldb.RetryConfig(2, customBackoff);

// The config `retryConfig1` will be overridden by `retryConfig2`
(async function() {
    await qldbDriver.executeLambda(async (txn) => {
        await txn.execute('CREATE TABLE Person');
    }, retryConfig2);
}());
```

TypeScript

```
import { BackoffFunction, QldbDriver, RetryConfig, TransactionExecutor } from
 "amazon-qldb-driver-nodejs"

// Configuring retry limit to 2
const retryConfig1: RetryConfig = new RetryConfig(2);
```

```
const qldbDriver: QldbDriver = new QldbDriver("test-ledger", undefined, undefined,
 retryConfig1);

// Configuring a custom backoff which increases delay by 1s for each attempt.
const customBackoff: BackoffFunction = (retryAttempt: number, error: Error,
 transactionId: string) => {
    return 1000 * retryAttempt;
};

const retryConfig2: RetryConfig = new RetryConfig(2, customBackoff);

// The config `retryConfig1` will be overridden by `retryConfig2`
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        await txn.execute('CREATE TABLE Person');
    }, retryConfig2);
}());
```

고유성 제약 조건 구현

QLDB는 고유 인덱스를 지원하지 않지만 애플리케이션에서 이 동작을 구현할 수 있습니다.

Person 테이블의 GovId 필드에 고유성 제약 조건을 구현하려고 한다고 가정해 보겠습니다. 이렇게 하면 다음 작업을 수행하는 트랜잭션을 작성합니다.

1. 테이블에 지정된 GovId가 있는 기존 문서가 없는지 확인합니다.
2. 어설션이 통과하면 문서를 삽입합니다.

경쟁 트랜잭션이 어설션을 동시에 통과하면 트랜잭션 중 하나만 성공적으로 커밋됩니다. 다른 트랜잭션은 OCC 충돌 예외가 발생하여 실패합니다.

다음 코드 예제는 이 고유성 제약 조건 구현 방법을 보여줍니다.

JavaScript

```
const govId = 'TOYENC486FH';
const document = {
    'FirstName': 'Brent',
    'GovId': 'TOYENC486FH',
};
(async function() {
```

```
    await qldbDriver.executeLambda(async (txn) => {
        // Check if doc with GovId = govId exists
        const results = (await txn.execute('SELECT * FROM Person WHERE GovId = ?',
 govId)).getResultList();
        // Insert the document after ensuring it doesn't already exist
        if (results.length == 0) {
            await txn.execute('INSERT INTO Person ?', document);
        }
    });
})();
```

TypeScript

```
const govId: string = 'TOYENC486FH';
const document: Record<string, string> = {
    'FirstName': 'Brent',
    'GovId': 'TOYENC486FH',
};
(async function(): Promise<void> {
    await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
        // Check if doc with GovId = govId exists
        const results: dom.Value[] = (await txn.execute('SELECT * FROM Person WHERE
 GovId = ?', govId)).getResultList();
        // Insert the document after ensuring it doesn't already exist
        if (results.length == 0) {
            await txn.execute('INSERT INTO Person ?', document);
        }
    });
})();
```

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 GovId 필드에 인덱스를 사용하는 것이 좋습니다. GovId에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

Amazon Ion 작업

다음 섹션에서는 Amazon Ion 모듈을 사용하여 Ion 데이터를 처리하는 방법을 보여줍니다.

목차

Ion 모듈 가져오기

JavaScript

```
var ionjs = require('ion-js');
```

TypeScript

```
import { dom, dumpBinary, dumpText, load } from "ion-js";
```

Ion 유형 생성

다음 코드 예제는 Ion 텍스트에서 Ion 객체를 만듭니다.

JavaScript

```
const ionText  = '{GovId: "TOYENC486FH", FirstName: "Brent"}';
const ionObj = ionjs.load(ionText);

console.log(ionObj.get('GovId')); // prints [String: 'TOYENC486FH']
console.log(ionObj.get('FirstName')); // prints [String: 'Brent']
```

TypeScript

```
const ionText: string = '{GovId: "TOYENC486FH", FirstName: "Brent"}';
const ionObj: dom.Value = load(ionText);

console.log(ionObj.get('GovId')); // prints [String: 'TOYENC486FH']
console.log(ionObj.get('FirstName')); // prints [String: 'Brent']
```

다음 코드 예제는 Node.js 사전에서 Ion 객체를 만듭니다.

JavaScript

```
const aDict = {
    'GovId': 'TOYENC486FH',
    'FirstName': 'Brent'
};
const ionObj = ionjs.load(ionjs.dumpBinary(aDict));
console.log(ionObj.get('GovId')); // prints [String: 'TOYENC486FH']
console.log(ionObj.get('FirstName')); // prints [String: 'Brent']
```

TypeScript

```
const aDict: Record<string, string> = {
    'GovId': 'TOYENC486FH',
    'FirstName': 'Brent'
};
const ionObj: dom.Value = load(dumpBinary(aDict));
console.log(ionObj.get('GovId')); // prints [String: 'TOYENC486FH']
console.log(ionObj.get('FirstName')); // prints [String: 'Brent']
```

Ion 이진 덤프 가져오기

JavaScript

```
// ionObj is an Ion struct
console.log(ionjs.dumpBinary(ionObj).toString()); // prints
 224,1,0,234,238,151,129,131,222,147,135,190,144,133,71,111,118,73,100,137,70,105,114,115,11
```

TypeScript

```
// ionObj is an Ion struct
console.log(dumpBinary(ionObj).toString()); // prints
 224,1,0,234,238,151,129,131,222,147,135,190,144,133,71,111,118,73,100,137,70,105,114,115,11
```

Ion 텍스트 덤프 가져오기

JavaScript

```
// ionObj is an Ion struct
```

```
console.log(ionjs.dumpText(ionObj)); // prints
 {GovId:"TOYENC486FH",FirstName:"Brent"}
```

TypeScript

```
// ionObj is an Ion struct
console.log(dumpText(ionObj)); // prints {GovId:"TOYENC486FH",FirstName:"Brent"}
```

Ion에 대한 자세한 정보는 GitHub의 Amazon Ion 설명서를 참조하십시오. QLDB에서 Ion을 사용하는 방법에 대한 추가 코드 예제는 Amazon QLDB에서 Amazon Ion 데이터 타입을 사용한 작업 섹션을 참조하세요.

Python용 Amazon QLDB 드라이버

원장의 데이터로 작업하려면 AWS가 제공하는 드라이버를 사용하여 Python 애플리케이션에서 Amazon QLDB에 연결할 수 있습니다. 다음 주제에서는 Python용 QLDB 드라이버를 시작하는 방법을 설명합니다.

주제

- 드라이버 리소스
- 필수 조건
- 설치
- Python용 Amazon QLDB 드라이버 - 빠른 시작 자습서
- Python용 Amazon QLDB 드라이버 - Cookbook 참조

드라이버 리소스

Python 드라이버에서 지원하는 기능에 대한 자세한 내용은 다음 리소스를 참조하십시오.

- API 참조: 3.x, 2.x
- 드라이버 소스 코드(GitHub)
- 샘플 애플리케이션 소스 코드(GitHub)
- Amazon Ion 코드 예제

필수 조건

Python용 QLDB 드라이버를 시작하기 전에 다음을 수행해야 합니다.

1. Amazon QLDB 액세스의 AWS 설정 지침을 따르십시오. 다음 내용이 해당됩니다.
   1. AWS에 가입합니다.
   2. 적절한 QLDB 권한을 가진 사용자를 생성합니다.
   3. 개발을 위한 프로그래밍 방식 액세스 권한을 부여합니다.
2. Python 다운로드 사이트에서 다음 버전의 Python 중 하나를 설치합니다.
   - 3.6 이상 — Python v3용 QLDB 드라이버
   - 3.4 이상 — Python v2용 QLDB 드라이버
3. AWS 보안 인증 정보와 기본 AWS 리전을 설정합니다. 지침은 AWS SDK for Python (Boto3) 설명서의 Quickstart을 참조하십시오.

   사용 가능한 리전의 전체 목록은 AWS 일반 참조에서 Amazon QLDB 엔드포인트 및 할당량을 참조하십시오.

다음으로 전체 자습서 샘플 애플리케이션을 다운로드하거나 Python 프로젝트에 드라이버만 설치하고 단축 코드 예제를 실행할 수 있습니다.

- 기존 프로젝트에 QLDB 드라이버와 AWS SDK for Python (Boto3)를 설치하려면 설치로 이동하세요.
- 프로젝트를 설정하고 원장에 대한 기본 데이터 트랜잭션을 보여주는 단축 코드 예제를 실행하려면 빠른 시작 자습서를 참조하십시오.
- 전체 자습서 샘플 애플리케이션에서 데이터 및 관리 API 작업에 대한 보다 심층적인 예제를 실행하려면 Python 자습서를 참조하십시오.

설치

QLDB는 다음 드라이버 버전과 해당 Python 종속성을 지원합니다.

드라이버 버전	Python 버전	상태	최근 릴리스 날짜
2.x	3.4 이상	프로덕션 릴리스	2020년 5월 7일

드라이버 버전	Python 버전	상태	최근 릴리스 날짜
3.x	3.6 이상	프로덕션 릴리스	2021년 10월 28일

pip(Python용 패키지 관리자)를 사용하여 PyPI에서 QLDB 드라이버를 설치하려면 명령줄에 다음을 입력합니다.

3.x

```
pip install pyqldb
```

2.x

```
pip install pyqldb==2.0.2
```

드라이버를 설치하면 AWS SDK for Python (Boto3) 및 Amazon Ion 패키지를 포함한 종속 항목도 설치됩니다.

드라이버를 사용하여 원장에 연결

그런 다음 드라이버를 가져와서 원장에 연결하는 데 사용할 수 있습니다. 다음 Python 코드 예제에서는 지정된 원장 이름에 대한 세션을 생성하는 방법을 보여줍니다.

3.x

```
from pyqldb.driver.qldb_driver import QldbDriver
qldb_driver = QldbDriver(ledger_name='testLedger')

for table in qldb_driver.list_tables():
    print(table)
```

2.x

```
from pyqldb.driver.pooled_qldb_driver import PooledQldbDriver

qldb_driver = PooledQldbDriver(ledger_name='testLedger')
qldb_session = qldb_driver.get_session()

for table in qldb_session.list_tables():
```

```
    print(table)
```

원장에서 기본 데이터 트랜잭션을 실행하는 방법에 대한 단축 코드 예제는 Cookbook 참조를 참조하십시오.

Python용 Amazon QLDB 드라이버 - 빠른 시작 자습서

이 자습서에서는 Python용 Amazon QLDB 드라이버의 최신 버전을 사용하여 간단한 애플리케이션을 설정하는 방법을 알아봅니다. 이 안내서에는 드라이버 설치 단계 및 기본적인 CRUD(생성, 읽기, 업데이트 및 삭제) 작업에 대한 단축 코드 예제가 포함되어 있습니다. 전체 샘플 애플리케이션에서 이러한 작업을 보여 주는 자세한 예를 보려면 Python 자습서 섹션을 참조하세요.

주제

필수 조건

시작하기 전에 다음을 수행해야 합니다.

1. Python 드라이버를 위한 필수 조건을 아직 완료하지 않은 경우, 완료하세요. 여기에는 AWS 가입, 개발을 위한 프로그램에 입각한 액세스 권한 부여, Python 버전 3.6 이상 설치가 포함됩니다.
2. `quick-start`라는 명칭의 원장을 생성합니다.

   원장 생성 방법을 알아보려면 콘솔 시작하기의 Amazon QLDB 원장의 기본 작업 또는 1단계: 새 원장 생성 섹션을 참조하세요.

1단계: 프로젝트 설정

먼저 Python 프로젝트를 설정합니다.

> **Note**
>
> 이러한 설정 단계를 자동화하는 기능이 있는 IDE를 사용하는 경우 2단계: 드라이버 초기화로 넘어가도 됩니다.

1. 애플리케이션을 위한 폴더를 생성합니다.

   ```
   $ mkdir myproject
   $ cd myproject
   ```

2. PyPI에서 Python용 QLDB 드라이버를 설치하려면 다음 `pip` 명령을 입력합니다.

   ```
   $ pip install pyqldb
   ```

   드라이버를 설치하면 AWS SDK for Python (Boto3) 및 Amazon Ion 패키지를 포함한 해당 종속 항목도 설치됩니다.

3. `app.py`라는 명칭의 새로운 파일을 만듭니다.

   그런 다음 다음 단계의 코드 예를 점진적으로 추가하여 몇 가지 기본 CRUD 작업을 시도해 보세요. 또는 단계별 자습서를 건너뛰고 전체 애플리케이션을 실행할 수도 있습니다.

2단계: 드라이버 초기화

`quick-start`라는 명칭의 원장에 연결되는 드라이버의 인스턴스를 초기화합니다. 다음 코드를 `app.py` 파일에 추가합니다.

```
from pyqldb.config.retry_config import RetryConfig
from pyqldb.driver.qldb_driver import QldbDriver

# Configure retry limit to 3
retry_config = RetryConfig(retry_limit=3)

# Initialize the driver
print("Initializing the driver")
qldb_driver = QldbDriver("quick-start", retry_config=retry_config)
```

3단계: 테이블 및 인덱스 생성

다음 코드 예에서는 `CREATE TABLE` 및 `CREATE INDEX` 문을 실행하는 방법을 보여줍니다.

`People`라는 명칭의 표와 해당 표의 `lastName` 필드 인덱스를 만드는 다음 코드를 추가합니다. 인덱스는 쿼리 성능을 최적화하고 OCC(낙관적 동시성 제어) 충돌 예외를 제한하는 데 필요합니다.

```
def create_table(transaction_executor):
    print("Creating a table")
    transaction_executor.execute_statement("Create TABLE People")

def create_index(transaction_executor):
    print("Creating an index")
    transaction_executor.execute_statement("CREATE INDEX ON People(lastName)")

# Create a table
qldb_driver.execute_lambda(lambda executor: create_table(executor))

# Create an index on the table
qldb_driver.execute_lambda(lambda executor: create_index(executor))
```

4단계: 문서 삽입

다음 코드 예에서는 `INSERT` 문을 실행하는 방법을 보여줍니다. QLDB는 PartiQL 쿼리 언어(SQL 호환) 및 Amazon Ion 데이터 형식(JSON의 상위 집합)을 지원합니다.

`People` 테이블에 문서를 삽입하는 다음 코드를 추가합니다.

```
def insert_documents(transaction_executor, arg_1):
    print("Inserting a document")
    transaction_executor.execute_statement("INSERT INTO People ?", arg_1)

# Insert a document
doc_1 = { 'firstName': "John",
          'lastName': "Doe",
          'age': 32,
        }

qldb_driver.execute_lambda(lambda x: insert_documents(x, doc_1))
```

이 예에서는 물음표(?)를 변수 자리 표시자로 사용하여 문서 정보를 해당 문에 전달합니다. 이 `execute_statement` 메서드는 Amazon Ion 타입과 Python 네이티브 타입 모두의 값을 지원합니다.

> **Tip**
>
> 단일 INSERT 문을 사용하여 여러 문서를 삽입하려면 다음과 같이 목록 타입의 파라미터를 해당 문에 전달할 수 있습니다.
>
> ```
> # people is a list
> transaction_executor.execute_statement("INSERT INTO Person ?", people)
> ```
>
> 목록을 전달할 때 변수 자리 표시자(?)를 이중 꺾쇠 괄호(<<...>>)로 묶지 마세요. 수동 PartiQL 문에서 이중 꺾쇠 괄호는 백으로 알려진 정렬되지 않은 모음을 의미합니다.

5단계: 문서 쿼리

다음 코드 예에서는 SELECT 문을 실행하는 방법을 보여줍니다.

People 테이블에서 문서를 쿼리하는 다음 코드를 추가합니다.

```
def read_documents(transaction_executor):
    print("Querying the table")
    cursor = transaction_executor.execute_statement("SELECT * FROM People WHERE
 lastName = ?", 'Doe')

    for doc in cursor:
        print(doc["firstName"])
        print(doc["lastName"])
        print(doc["age"])

# Query the table
qldb_driver.execute_lambda(lambda executor: read_documents(executor))
```

6단계: 문서 업데이트

다음 코드 예에서는 UPDATE 문을 실행하는 방법을 보여줍니다.

1. age를 42로 업데이트하여 People 표의 문서를 업데이트하는 다음 코드를 추가합니다.

   ```
   def update_documents(transaction_executor, age, lastName):
       print("Updating the document")
       transaction_executor.execute_statement("UPDATE People SET age = ? WHERE
    lastName = ?", age, lastName)
   ```

```
# Update the document
age = 42
lastName = 'Doe'

qldb_driver.execute_lambda(lambda x: update_documents(x, age, lastName))
```

2. 표를 다시 쿼리하여 업데이트된 값을 확인합니다.

```
# Query the updated document
qldb_driver.execute_lambda(lambda executor: read_documents(executor))
```

3. 애플리케이션을 실행하려면 프로젝트 디렉터리에서 다음 명령을 입력합니다.

```
$ python app.py
```

전체 애플리케이션 실행

다음 코드 예는 app.py 애플리케이션의 전체 버전입니다. 이전 단계를 개별적으로 수행하는 대신 이 코드 예를 처음부터 끝까지 복사하여 실행할 수도 있습니다. 이 애플리케이션은 quick-start이라는 명칭의 원장에 대한 몇 가지 기본 CRUD 작업을 보여줍니다.

> **Note**
>
> 이 코드를 실행하기 전에 quick-start 원장에 People이라는 명칭의 활성 테이블이 아직 없는지 확인하세요.

```
from pyqldb.config.retry_config import RetryConfig
from pyqldb.driver.qldb_driver import QldbDriver

def create_table(transaction_executor):
    print("Creating a table")
    transaction_executor.execute_statement("CREATE TABLE People")

def create_index(transaction_executor):
    print("Creating an index")
    transaction_executor.execute_statement("CREATE INDEX ON People(lastName)")

def insert_documents(transaction_executor, arg_1):
```

```
    print("Inserting a document")
    transaction_executor.execute_statement("INSERT INTO People ?", arg_1)

def read_documents(transaction_executor):
    print("Querying the table")
    cursor = transaction_executor.execute_statement("SELECT * FROM People WHERE
 lastName = ?", 'Doe')


    for doc in cursor:
        print(doc["firstName"])
        print(doc["lastName"])
        print(doc["age"])

def update_documents(transaction_executor, age, lastName):
    print("Updating the document")
    transaction_executor.execute_statement("UPDATE People SET age = ? WHERE lastName
 = ?", age, lastName)

# Configure retry limit to 3
retry_config = RetryConfig(retry_limit=3)

# Initialize the driver
print("Initializing the driver")
qldb_driver = QldbDriver("quick-start", retry_config=retry_config)

# Create a table
qldb_driver.execute_lambda(lambda executor: create_table(executor))

# Create an index on the table
qldb_driver.execute_lambda(lambda executor: create_index(executor))

# Insert a document
doc_1 = { 'firstName': "John",
          'lastName': "Doe",
          'age': 32,
        }

qldb_driver.execute_lambda(lambda x: insert_documents(x, doc_1))

# Query the table
qldb_driver.execute_lambda(lambda executor: read_documents(executor))

# Update the document
```

```
age = 42
lastName = 'Doe'

qldb_driver.execute_lambda(lambda x: update_documents(x, age, lastName))

# Query the table for the updated document
qldb_driver.execute_lambda(lambda executor: read_documents(executor))
```

전체 애플리케이션을 실행하려면 프로젝트 디렉터리에서 다음 명령을 입력합니다.

```
$ python app.py
```

Python용 Amazon QLDB 드라이버 - Cookbook 참조

이 참조 가이드는 Python용 Amazon QLDB 드라이버의 일반적인 사용 사례를 보여줍니다. 이 Python 코드 예제는 드라이버를 사용하여 기본 CRUD(생성, 읽기, 업데이트, 삭제) 작업을 실행하는 방법을 보여줍니다. 또한 Amazon Ion 데이터를 처리하기 위한 코드 예제도 포함되어 있습니다. 또한 이 가이드에서는 트랜잭션에 멱등성을 부여하고 고유성 제약하는 모범 사례를 중점적으로 설명합니다.

> **Note**
>
> 해당하는 경우, 일부 사용 사례에서는 Python용 QLDB 드라이버의 지원되는 각 주요 버전마다 다른 코드 예제가 있습니다.

목차

- 문서 삭제
- 하나의 트랜잭션에서 여러 명령문 실행
- 재시도 로직
- 고유성 제약 조건 구현

- Amazon Ion 작업
  - Ion 모듈 가져오기
  - Ion 유형 생성
  - Ion 이진 덤프 가져오기
  - Ion 텍스트 덤프 가져오기

드라이버 가져오기

다음 코드 예제에서는 드라이브를 가져옵니다.

3.x

```
from pyqldb.driver.qldb_driver import QldbDriver
import amazon.ion.simpleion as simpleion
```

2.x

```
from pyqldb.driver.pooled_qldb_driver import PooledQldbDriver
import amazon.ion.simpleion as simpleion
```

> **Note**
>
> 이 예제에서는 Amazon Ion 패키지(`amazon.ion.simpleion`)도 가져옵니다. 이 참조에서 일부 데이터 작업을 실행할 때 Ion 데이터를 처리하려면 이 패키지가 필요합니다. 자세한 내용은 Amazon Ion 작업 섹션을 참조하세요.

드라이버 인스턴스화

다음 코드 예제는 기본 설정을 사용하여 지정된 원장 이름에 연결하는 드라이버 인스턴스를 만듭니다.

3.x

```
qldb_driver = QldbDriver(ledger_name='vehicle-registration')
```

2.x

```
qldb_driver = PooledQldbDriver(ledger_name='vehicle-registration')
```

CRUD 작업

QLDB는 트랜잭션의 일부로 CRUD(생성, 읽기, 업데이트, 삭제) 작업을 실행합니다.

> **Warning**
>
> 가장 좋은 방법은 쓰기 트랜잭션이 완전한 멱등성을 부여하는 것입니다.

트랜잭션에 멱등성 부여하기

재시도 시 예상치 못한 부작용이 발생하지 않도록 쓰기 트랜잭션에 멱등성을 부여하는 것이 좋습니다. 여러 번 실행하여 매번 동일한 결과를 생성할 수 있는 트랜잭션은 멱등성을 가집니다.

이름이 `Person`인 테이블에 문서를 삽입하는 트랜잭션을 예로 들어 보겠습니다. 트랜잭션은 먼저 문서가 테이블에 이미 존재하는지 여부를 확인해야 합니다. 이렇게 확인하지 않으면 테이블에 문서가 중복될 수 있습니다.

QLDB가 서버 측에서 트랜잭션을 성공적으로 커밋했지만 응답을 기다리는 동안 클라이언트 제한 시간이 초과되었다고 가정해 보겠습니다. 트랜잭션이 멱등성을 가지지 않는 경우 재시도 시 동일한 문서가 두 번 이상 삽입될 수 있습니다.

인덱스를 사용하여 전체 테이블 스캔 방지

인덱싱된 필드 또는 문서 ID(예: `WHERE indexedField = 123` 또는 `WHERE indexedField IN (456, 789)`)에서 동등 연산자를 사용하여 `WHERE` 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 이 인덱싱된 조회가 없으면 QLDB는 테이블 스캔을 수행해야 하며, 이로 인해 트랜잭션 제한 시간이 초과되거나 OCC(낙관적 동시성 제어) 충돌이 발생할 수 있습니다.

OCC에 대한 자세한 내용은 Amazon QLDB 동시성 모델 단원을 참조하세요.

암시적으로 생성된 트랜잭션

pyqldb.driver.qldb_driver.execute_lambda 메서드는 pyqldb.execution.executor.Executor의 인스턴스를 수신하는 Lambda 함수를 받아들이며, 이 함수를 사용하여 명령문을 실행할 수 있습니다. `Executor`의 인스턴스는 암시적으로 생성된 트랜잭션을 래핑합니다.

트랜잭션 실행자의 execute_statement 메서드를 사용하여 Lambda 함수 내에서 명령문을 실행할 수 있습니다. 드라이버는 Lambda 함수가 반환될 때 트랜잭션을 암시적으로 커밋합니다.

> **Note**
>
> 이 `execute_statement` 메서드는 Amazon Ion 유형과 Python 네이티브 유형을 모두 지원합니다. Python 네이티브 유형을 인수로 `execute_statement`에 전달하면 드라이버가 `amazon.ion.simpleion` 모듈을 사용하여 Ion 유형으로 변환합니다(지정된 Python 데이터 유형에 대한 변환이 지원되는 경우). 지원되는 데이터 유형 및 변환 규칙은 simpleion 소스 코드를 참조하세요.

다음 섹션에서는 기본 CRUD 작업을 실행하고, 사용자 지정 재시도 로직을 지정하고, 고유성 제약 조건을 구현하는 방법을 보여줍니다.

목차

테이블 생성

```
def create_table(transaction_executor):
```

```
    transaction_executor.execute_statement("CREATE TABLE Person")

qldb_driver.execute_lambda(lambda executor: create_table(executor))
```

인덱스 생성

```
def create_index(transaction_executor):
    transaction_executor.execute_statement("CREATE INDEX ON Person(GovId)")

qldb_driver.execute_lambda(lambda executor: create_index(executor))
```

문서 읽기

```
# Assumes that Person table has documents as follows:
# { "GovId": "TOYENC486FH", "FirstName": "Brent" }

def read_documents(transaction_executor):
    cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId =
 'TOYENC486FH'")

    for doc in cursor:
        print(doc["GovId"]) # prints TOYENC486FH
        print(doc["FirstName"]) # prints Brent

qldb_driver.execute_lambda(lambda executor: read_documents(executor))
```

쿼리 파라미터 사용

다음 코드 예제는 네이티브 유형 쿼리 파라미터를 사용합니다.

```
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId = ?",
 'TOYENC486FH')
```

다음 코드 예제는 Ion 유형 쿼리 파라미터를 사용합니다.

```
name = ion.loads('Brent')
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE FirstName
 = ?", name)
```

다음 코드 예제는 여러 쿼리 파라미터를 사용합니다.

```
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId = ?
 AND FirstName = ?", 'TOYENC486FH', "Brent")
```

다음 코드 예제는 쿼리 파라미터 목록을 사용합니다.

```
gov_ids = ['TOYENC486FH','ROEE1','YH844']
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId IN
 (?,?,?)", *gov_ids)
```

> **Note**
>
> 인덱싱된 조회 없이 쿼리를 실행하면 전체 테이블 스캔이 호출됩니다. 이 예제에서는 성능을 최적화하기 위해 `GovId` 필드에 인덱스를 사용하는 것이 좋습니다. `GovId`에 인덱스를 사용하지 않으면 쿼리에 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

문서 삽입하기

다음 코드 예제는 네이티브 데이터 유형을 삽입합니다.

```
def insert_documents(transaction_executor, arg_1):
    # Check if doc with GovId:TOYENC486FH exists
    # This is critical to make this transaction idempotent
    cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId
 = ?", 'TOYENC486FH')
    # Check if there is any record in the cursor
    first_record = next(cursor, None)

    if first_record:
        # Record already exists, no need to insert
        pass
    else:
        transaction_executor.execute_statement("INSERT INTO Person ?", arg_1)

doc_1 = { 'FirstName': "Brent",
          'GovId': 'TOYENC486FH',
        }

qldb_driver.execute_lambda(lambda executor: insert_documents(executor, doc_1))
```

다음 코드 예제는 Ion 데이터 유형을 삽입합니다.

```
def insert_documents(transaction_executor, arg_1):
    # Check if doc with GovId:TOYENC486FH exists
    # This is critical to make this transaction idempotent
    cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId
 = ?", 'TOYENC486FH')
    # Check if there is any record in the cursor
    first_record = next(cursor, None)

    if first_record:
        # Record already exists, no need to insert
        pass
    else:
        transaction_executor.execute_statement("INSERT INTO Person ?", arg_1)

doc_1 = { 'FirstName': 'Brent',
          'GovId': 'TOYENC486FH',
        }

# create a sample Ion doc
ion_doc_1 = simpleion.loads(simpleion.dumps(doc_1)))

qldb_driver.execute_lambda(lambda executor: insert_documents(executor, ion_doc_1))
```

이 트랜잭션은 문서를 Person 테이블에 삽입합니다. 삽입하기 전에 먼저 문서가 테이블에 이미 있는지 확인합니다. 이 검사를 통해 트랜잭션은 본질적으로 멱등성을 가지게 됩니다. 이 트랜잭션을 여러 번 실행하더라도 의도하지 않은 부작용이 발생하지는 않습니다.

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 GovId 필드에 인덱스를 사용하는 것이 좋습니다. GovId에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

하나의 명령문에 여러 문서 삽입

단일 INSERT 문을 사용하여 여러 문서를 삽입하려면 다음과 같이 목록 타입의 파라미터를 해당 문에 전달할 수 있습니다.

```
# people is a list
```

```
transaction_executor.execute_statement("INSERT INTO Person ?", people)
```

목록을 전달할 때 변수 자리 표시자(?)를 이중 꺾쇠 괄호(<<...>>)로 묶지 마세요. 수동 PartiQL 문에서 이중 꺾쇠 괄호는 백으로 알려진 정렬되지 않은 모음을 의미합니다.

문서 업데이트

다음 코드 예제는 네이티브 데이터 유형을 사용합니다.

```
def update_documents(transaction_executor, gov_id, name):
    transaction_executor.execute_statement("UPDATE Person SET FirstName = ?  WHERE
 GovId = ?", name, gov_id)

gov_id = 'TOYENC486FH'
name = 'John'

qldb_driver.execute_lambda(lambda executor: update_documents(executor, gov_id, name))
```

다음 코드 예제는 Ion 데이터 유형을 사용합니다.

```
def update_documents(transaction_executor, gov_id, name):
    transaction_executor.execute_statement("UPDATE Person SET FirstName = ? WHERE GovId
 = ?", name, gov_id)

# Ion datatypes
gov_id = simpleion.loads('TOYENC486FH')
name = simpleion.loads('John')

qldb_driver.execute_lambda(lambda executor: update_documents(executor, gov_id, name))
```

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 GovId 필드에 인덱스를 사용하는 것이 좋습니다. GovId에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

문서 삭제

다음 코드 예제는 네이티브 데이터 유형을 사용합니다.

```
def delete_documents(transaction_executor, gov_id):
    cursor = transaction_executor.execute_statement("DELETE FROM Person WHERE GovId
 = ?", gov_id)

gov_id = 'TOYENC486FH'

qldb_driver.execute_lambda(lambda executor: delete_documents(executor, gov_id))
```

다음 코드 예제는 Ion 데이터 유형을 사용합니다.

```
def delete_documents(transaction_executor, gov_id):
    cursor = transaction_executor.execute_statement("DELETE FROM Person WHERE GovId
 = ?", gov_id)

# Ion datatypes
gov_id = simpleion.loads('TOYENC486FH')

qldb_driver.execute_lambda(lambda executor: delete_documents(executor, gov_id))
```

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 GovId 필드에 인덱스를 사용하는 것이 좋습니다. GovId에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

하나의 트랜잭션에서 여러 명령문 실행

```
# This code snippet is intentionally trivial. In reality you wouldn't do this because
 you'd
# set your UPDATE to filter on vin and insured, and check if you updated something or
 not.

def do_insure_car(transaction_executor, vin):
    cursor = transaction_executor.execute_statement(
        "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE", vin)
    first_record = next(cursor, None)
    if first_record:
        transaction_executor.execute_statement(
            "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin)
        return True
```

```
    else:
        return False

def insure_car(qldb_driver, vin_to_insure):
    return qldb_driver.execute_lambda(
        lambda executor: do_insure_car(executor, vin_to_insure))
```

재시도 로직

드라이버의 execute_lambda 메서드에는 재시도 가능한 예외(예: 시간 초과 또는 OCC 충돌)가 발생할 경우 트랜잭션을 재시도하는 재시도 메커니즘이 내장되어 있습니다.

3.x

최대 재시도 횟수와 백오프 전략을 구성할 수 있습니다.

기본 재시도 제한은 4이며, 기본 백오프 전략은 10밀리초 단위의 지수 백오프 및 Jitter입니다. pyqldb.config.retry_config.RetryConfig 인스턴스를 사용하여 드라이버 인스턴스별 및 트랜잭션별 재시도 구성을 설정할 수 있습니다.

다음 코드 예제는 드라이버 인스턴스에 대한 사용자 지정 재시도 제한 및 사용자 지정 백오프 전략을 사용하여 재시도 로직을 지정합니다.

```
from pyqldb.config.retry_config import RetryConfig
from pyqldb.driver.qldb_driver import QldbDriver

# Configuring retry limit to 2
retry_config = RetryConfig(retry_limit=2)
qldb_driver = QldbDriver("test-ledger", retry_config=retry_config)

# Configuring a custom backoff which increases delay by 1s for each attempt.
def custom_backoff(retry_attempt, error, transaction_id):
    return 1000 * retry_attempt

retry_config_custom_backoff = RetryConfig(retry_limit=2,
 custom_backoff=custom_backoff)
qldb_driver = QldbDriver("test-ledger", retry_config=retry_config_custom_backoff)
```

다음 코드 예제는 특정 Lambda 실행에 대한 사용자 지정 재시도와 사용자 지정 백오프 전략을 사용하여 재시도 로직을 지정합니다. execute_lambda에 대한 이 구성은 드라이버 인스턴스에 설정된 재시도 로직을 재정의합니다.

```
from pyqldb.config.retry_config import RetryConfig
from pyqldb.driver.qldb_driver import QldbDriver

# Configuring retry limit to 2
retry_config_1 = RetryConfig(retry_limit=4)
qldb_driver = QldbDriver("test-ledger", retry_config=retry_config_1)

# Configuring a custom backoff which increases delay by 1s for each attempt.
def custom_backoff(retry_attempt, error, transaction_id):
    return 1000 * retry_attempt

retry_config_2 = RetryConfig(retry_limit=2, custom_backoff=custom_backoff)

# The config `retry_config_1` will be overriden by `retry_config_2`
qldb_driver.execute_lambda(lambda txn: txn.execute_statement("CREATE TABLE Person"),
 retry_config_2)
```

2.x

최대 재시도 횟수는 구성 가능합니다. PooledQldbDriver를 초기화할 때 retry_limit 속성을 설정하여 재시도 제한을 구성할 수 있습니다.

기본 재시도 한도는 4입니다.

고유성 제약 조건 구현

QLDB는 고유 인덱스를 지원하지 않지만 애플리케이션에서 이 동작을 구현할 수 있습니다.

Person 테이블의 GovId 필드에 고유성 제약 조건을 구현하려고 한다고 가정해 보겠습니다. 이렇게 하면 다음 작업을 수행하는 트랜잭션을 작성합니다.

1. 테이블에 지정된 GovId가 있는 기존 문서가 없는지 확인합니다.
2. 어설션이 통과하면 문서를 삽입합니다.

경쟁 트랜잭션이 어설션을 동시에 통과하면 트랜잭션 중 하나만 성공적으로 커밋됩니다. 다른 트랜잭션은 OCC 충돌 예외가 발생하여 실패합니다.

다음 코드 예제는 이 고유성 제약 조건 구현 방법을 보여줍니다.

```
def insert_documents(transaction_executor, gov_id, document):
```

```
    # Check if doc with GovId = gov_id exists
    cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId
 = ?", gov_id)
    # Check if there is any record in the cursor
    first_record = next(cursor, None)

    if first_record:
        # Record already exists, no need to insert
        pass
    else:
        transaction_executor.execute_statement("INSERT INTO Person ?", document)

qldb_driver.execute_lambda(lambda executor: insert_documents(executor, gov_id,
 document))
```

> **Note**
>
> 이 예제에서는 성능을 최적화하기 위해 `GovId` 필드에 인덱스를 사용하는 것이 좋습니다. `GovId`에 인덱스를 설정하지 않으면 명령문의 지연 시간이 길어지고 OCC 충돌 예외 또는 트랜잭션 시간 초과가 발생할 수도 있습니다.

Amazon Ion 작업

다음 섹션에서는 Amazon Ion 모듈을 사용하여 Ion 데이터를 처리하는 방법을 보여줍니다.

목차

Ion 모듈 가져오기

```
import amazon.ion.simpleion as simpleion
```

Ion 유형 생성

다음 코드 예제는 Ion 텍스트에서 Ion 객체를 만듭니다.

```
ion_text = '{GovId: "TOYENC486FH", FirstName: "Brent"}'
ion_obj = simpleion.loads(ion_text)

print(ion_obj['GovId']) # prints TOYENC486FH
print(ion_obj['Name']) # prints Brent
```

다음 코드 예제는 Python `dict`에서 Ion 객체를 만듭니다.

```
a_dict = { 'GovId': 'TOYENC486FH',
           'FirstName': "Brent"
         }
ion_obj = simpleion.loads(simpleion.dumps(a_dict))

print(ion_obj['GovId']) # prints TOYENC486FH
print(ion_obj['FirstName']) # prints Brent
```

Ion 이진 덤프 가져오기

```
# ion_obj is an Ion struct
print(simpleion.dumps(ion_obj)) # b'\xe0\x01\x00\xea\xee\x97\x81\x83\xde\x93\x87\xbe
\x90\x85GovId\x89FirstName\xde\x94\x8a\x8bTOYENC486FH\x8b\x85Brent'
```

Ion 텍스트 덤프 가져오기

```
# ion_obj is an Ion struct
print(simpleion.dumps(ion_obj, binary=False)) # prints $ion_1_0
 {GovId:'TOYENC486FH',FirstName:"Brent"}
```

Ion 작업에 대한 자세한 정보는 GitHub의 Amazon Ion 설명서를 참조하십시오. QLDB에서 Ion을 사용하는 방법에 대한 추가 코드 예제는 Amazon QLDB에서 Amazon Ion 데이터 타입을 사용한 작업 섹션을 참조하세요.

Amazon QLDB에서 드라이버를 사용한 세션 관리에 대한 이해

관계형 데이터베이스 관리 시스템(RDBMS)을 사용해 본 경험이 있다면 동시 연결에 익숙할 것입니다. 트랜잭션이 HTTP 요청 및 응답 메시지로 실행되기 때문에 QLDB는 기존 RDBMS 연결과 같은 개념을 가지고 있지 않습니다.

QLDB에서 유사한 개념은 활성 세션입니다. 세션은 개념적으로 사용자 로그인과 비슷합니다. 즉, 원장에 대한 데이터 트랜잭션 요청에 대한 정보를 관리합니다. 활성 세션은 트랜잭션을 활발하게 실행하는

세션입니다. 또한 서비스가 즉시 다른 트랜잭션을 시작할 것으로 예상하는, 트랜잭션을 최근에 완료한 세션일 수도 있습니다. QLDB는 세션당 하나의 활성 실행 트랜잭션을 지원합니다.

원장당 동시 활성 세션의 한도는 Amazon QLDB 할당량 및 제한에 정의되어 있습니다. 이 한도에 도달한 후 트랜잭션을 시작하려는 모든 세션에서 오류(`LimitExceededException`)가 발생합니다.

QLDB 드라이버를 사용하여 애플리케이션에서 세션 풀을 구성하는 모범 사례는 Amazon QLDB 드라이버 권장 사항의 QLDBDriver 객체 구성 섹션을 참조하세요.

목차

세션 라이프사이클

다음 QLDB 세션 API 작업 시퀀스는 QLDB 세션의 일반적인 라이프사이클를 나타냅니다.

1. `StartSession`
2. `StartTransaction`
3. `ExecuteStatement`
4. `CommitTransaction`
5. 더 많은 트랜잭션을 시작하려면 2-4단계를 반복합니다(한 번에 한 트랜잭션씩).
6. `EndSession`

세션 만료

QLDB는 트랜잭션이 활발하게 실행되고 있는지 여부와 상관없이 총 13~17분이 지나면 세션이 만료되고 삭제됩니다. 세션은 하드웨어 장애, 네트워크 장애 또는 애플리케이션 재시작과 같은 여러 가지 이유로 손실되거나 손상될 수 있습니다. QLDB는 클라이언트 애플리케이션이 세션 장애에 대해 복원력을 유지할 수 있도록 세션에 최대 수명을 적용합니다.

QLDB 드라이버에서의 세션 처리

AWS SDK를 사용하여 QLDB 세션 API와 직접 상호 작용할 수 있지만, 이 경우 복잡성이 가중되고 체결 다이제스트를 계산해야 합니다. QLDB는 이 체결 다이제스트를 사용하여 트랜잭션 무결성을 보장합니다. 이 API와 직접 상호 작용하는 대신 QLDB 드라이버를 사용하는 것이 좋습니다.

드라이버는 트랜잭션 데이터 API 위에 높은 수준의 추상 계층을 제공합니다. SendCommand API 호출을 관리하여 원장 데이터에서 PartiQL 문을 실행하는 프로세스를 간소화합니다. 이러한 API 호출에는 세션, 트랜잭션 관리, 오류 발생 시 재시도 정책 등 드라이버가 대신 처리하는 여러 파라미터가 필요합니다.

주제

- 세션 풀링 개요
- 세션 풀링 및 트랜잭션 로직
- 풀로 세션 반환하기

세션 풀링 개요

이전 버전의 QLDB 드라이버(예: Java v1.1.0)에서는 드라이버 객체의 두 가지 구현, 즉 표준, 비풀링 `QldbDriver` 및 `PooledQldbDriver`를 제공했습니다. 명칭에서 알 수 있듯이 `PooledQldbDriver`는 여러 트랜잭션에서 재사용되는 세션 풀을 유지 관리합니다.

사용자 피드백에 따르면 개발자들은 표준 드라이버를 사용하는 대신 풀링 기능과 그 이점을 기본으로 사용하는 것을 선호합니다. 따라서 `PooledQldbDriver`를 제거하고 세션 풀링 기능을 `QldbDriver`로 옮겼습니다. 이 변경 사항은 각 드라이버의 최근 릴리스(예: Java v2.0.0)에 포함되어 있습니다.

드라이버는 세 가지 수준의 추상화를 제공합니다.

- 드라이버(풀링된 드라이버 구현) - 최상위 추상화입니다. 드라이버는 세션 풀을 유지 및 관리합니다. 드라이버에게 트랜잭션을 실행하도록 요청하면 드라이버는 풀에서 세션을 선택하고 이를 사용하여 트랜잭션을 실행합니다. 세션 오류(`InvalidSessionException`)로 인해 트랜잭션이 실패하는 경우 드라이버는 다른 세션을 사용하여 트랜잭션을 재시도합니다. 기본적으로 드라이버는 종합 관리형 세션 환경을 제공합니다.
- 세션 - 드라이버 추상화보다 한 수준 낮습니다. 세션은 풀에 포함되며 드라이버는 세션의 라이프사이클를 관리합니다. 트랜잭션이 실패하면 드라이버는 동일한 세션을 사용하여 지정된 횟수까지 트랜잭션을 재시도합니다. 그러나 세션에서 오류(`InvalidSessionException`)가 발생하면 QLDB는

해당 오류를 내부적으로 삭제합니다. 그러면 드라이버는 풀에서 다른 세션을 가져와 트랜잭션을 재시도해야 합니다.

- 트랜잭션 - 가장 낮은 추상화 수준입니다. 트랜잭션은 세션에 포함되며 세션은 트랜잭션의 라이프사이클를 관리합니다. 세션은 오류 발생 시 트랜잭션을 재시도합니다. 또한 세션은 체결되거나 취소되지 않은 열린 트랜잭션이 유출되지 않도록 합니다.

각 드라이버의 최신 버전에서는 드라이버 추상화 수준에서만 작업을 수행할 수 있습니다. 개별 세션과 트랜잭션을 직접 제어할 수 없습니다. (즉, 새 세션이나 트랜잭션을 수동으로 시작하는 API 작업이 없습니다.)

세션 풀링 및 트랜잭션 로직

각 드라이버의 최근 릴리스에서는 더 이상 드라이버 객체의 비풀링 구현을 제공하지 않습니다. 기본적으로 `QldbDriver` 객체는 세션 풀을 관리합니다. 트랜잭션을 실행하기 위해 호출을 하면 드라이버가 다음 단계를 수행합니다.

1. 드라이버는 세션 풀 한도에 도달했는지 확인합니다. 그럴 경우 드라이버는 즉시 `NoSessionAvailable` 예외를 발생시킵니다. 아니면 그 다음 단계로 이동합니다.
2. 드라이버는 풀에 사용 가능한 세션이 있는지 확인합니다.
   - 풀에서 세션을 사용할 수 있는 경우 드라이버는 해당 세션을 사용하여 트랜잭션을 실행합니다.
   - 풀에 사용 가능한 세션이 없는 경우 드라이버는 새 세션을 만들고 이를 사용하여 트랜잭션을 실행합니다.
3. 드라이버가 2단계에서 세션을 가져오면 드라이버가 세션 인스턴스에서 `execute` 작업을 호출합니다.
4. 세션의 `execute` 작업에서 드라이버는 `startTransaction`를 호출하여 트랜잭션을 시작하려고 합니다.
   - 세션이 유효하지 않으면 `startTransaction` 호출이 실패하고 드라이버는 1단계로 돌아갑니다.
   - `startTransaction` 호출이 성공하면 드라이버는 다음 단계로 진행됩니다.
5. 드라이버가 Lambda 표현식을 실행합니다. 이 Lambda 표현식에는 PartiQL 문을 실행하기 위한 하나 이상의 호출이 포함될 수 있습니다. Lambda 표현식이 오류 없이 실행을 마치면 드라이버는 트랜잭션 체결을 진행합니다.
6. 트랜잭션을 체결하면 다음 두 가지 결과 중 하나가 발생할 수 있습니다.
   - 체결이 성공하고 드라이버가 제어권을 애플리케이션 코드에 반환합니다.

- OCC(낙관적 동시성 제어) 충돌로 인해 체결이 실패합니다. 이 경우 드라이버는 동일한 세션을 사용하여 4~6단계를 재시도합니다. 최대 재시도 횟수는 애플리케이션 코드에서 구성할 수 있습니다. 기본 한도는 4입니다.

> **Note**
>
> 4~6단계에서 `InvalidSessionException`가 발생하면 드라이버는 세션을 닫힌 것으로 표시하고 1단계로 돌아가 트랜잭션을 재시도합니다.
> 4~6단계에서 다른 예외가 발생하는 경우 드라이버는 예외를 재시도할 수 있는지 확인합니다. 그럴 경우 지정된 재시도 횟수까지 트랜잭션을 재시도합니다. 그렇지 않으면 예외가 애플리케이션 코드에 전파(버블 업되어 발생)됩니다.

풀로 세션 반환하기

활성 트랜잭션이 진행되는 동안 언제든지 `InvalidSessionException`가 발생하는 경우 드라이버는 세션을 풀로 반환하지 않습니다. QLDB는 대신 세션을 삭제하고 드라이버는 풀에서 다른 세션을 가져옵니다. 다른 모든 경우에는 드라이버가 세션을 풀에 반환합니다.

Amazon QLDB 드라이버 권장 사항

이 섹션에서는 지원되는 모든 언어에 대해 Amazon QLDB 드라이버를 구성하고 사용하는 모범 사례를 설명합니다. 제공된 코드 예제는 Java 전용입니다.

이러한 권장 사항은 대부분의 일반적인 사용 사례에 적용되지만 한 가지 방법이 모든 사용 사례에 적합하지는 않습니다. 애플리케이션에 적합하다고 판단되는 대로 다음 권장 사항을 사용하세요.

주제

QLDBDriver 객체 구성

이 `QldbDriver` 객체는 트랜잭션 간에 재사용되는 세션 풀을 유지 관리하여 원장과의 연결을 관리합니다. 세션은 원장에 대한 단일 연결을 나타냅니다. QLDB는 세션당 하나의 활성 실행 트랜잭션을 지원합니다.

Important

이전 드라이버 버전의 경우 세션 풀링 기능은 `QldbDriver` 대신 여전히 `PooledQldbDriver` 객체에 있습니다. 다음 버전 중 하나를 사용하는 경우, 이 주제의 나머지에서는 `QldbDriver`에 대한 언급을 `PooledQldbDriver`로 바꿉니다.

드라이버	버전
Java	`1.1.0` 또는 이전
.NET	`0.1.0-beta`
Node.js	`1.0.0-rc.1` 또는 이전
Python	`2.0.2` 또는 이전

이 `PooledQldbDriver` 객체는 최신 버전의 드라이버에서 더 이상 사용되지 않습니다. 최신 버전으로 업그레이드하고 `PooledQldbDriver`의 모든 인스턴스를 `QldbDriver`로 변환하는 것이 좋습니다.

QldbDriver를 글로벌 객체로 구성

드라이버 및 세션 사용을 최적화하려면 애플리케이션 인스턴스에 드라이버의 글로벌 인스턴스가 하나만 있어야 합니다. 예를 들어, Java에서는 Spring, Google Guice 또는 Dagger와 같은 종속성 주입 프레임워크를 사용할 수 있습니다. 다음 코드 예제는 `QldbDriver`를 싱글톤으로 구성하는 방법을 보여줍니다.

```
@Singleton
public QldbDriver qldbDriver (AWSCredentialsProvider credentialsProvider,
                                    @Named(LEDGER_NAME_CONFIG_PARAM) String ledgerName)
 {
```

```
    QldbSessionClientBuilder builder = QldbSessionClient.builder();
    if (null != credentialsProvider) {
        builder.credentialsProvider(credentialsProvider);
    }
    return QldbDriver.builder()
            .ledger(ledgerName)
            .transactionRetryPolicy(RetryPolicy
                .builder()
                .maxRetries(3)
                .build())
            .sessionClientBuilder(builder)
            .build();
}
```

재시도 시도 구성

드라이버는 일반적인 일시적 예외(예: `SocketTimeoutException` 또는 `NoHttpResponseException`)가 발생할 경우 자동으로 트랜잭션을 재시도합니다. 최대 재시도 횟수를 설정하려면 `QldbDriver`의 인스턴스를 만들 때 `transactionRetryPolicy` 구성 객체의 `maxRetries` 파라미터를 사용할 수 있습니다. (이전 섹션에 나열된 이전 드라이버 버전의 경우 `PooledQldbDriver`의 `retryLimit` 파라미터를 사용하세요.)

`maxRetries`의 기본값은 4입니다.

`InvalidParameterException`와 같은 클라이언트 측 오류는 다시 시도할 수 없습니다. 오류가 발생하면 트랜잭션이 중단되고 세션이 풀로 반환되며 드라이버의 클라이언트에 예외가 발생합니다.

동시에 접속할 수 있는 최대 세션 및 트랜잭션 수 설정

`QldbDriver`의 인스턴스가 트랜잭션을 실행하는 데 사용하는 최대 원장 세션 수는 해당 `maxConcurrentTransactions` 파라미터에 의해 정의됩니다. (이전 섹션에 나열된 이전 드라이버 버전의 경우 이는 `PooledQldbDriver`의 `poolLimit` 파라미터로 정의됩니다.)

이 한도는 특정 AWS SDK에서 정의한 대로 0보다 크고 세션 클라이언트가 허용하는 최대 공개 HTTP 연결 수보다 작거나 같아야 합니다. 예를 들어, Java의 경우 최대 연결 수는 ClientConfiguration 객체에 설정됩니다.

`maxConcurrentTransactions`의 기본값은 AWS SDK의 최대 연결 설정입니다.

애플리케이션에서 `QldbDriver`를 구성할 때는 다음과 같은 크기 조정 고려 사항을 고려하세요.

- 풀에는 항상 계획한 동시 실행 트랜잭션 수만큼 많은 세션이 있어야 합니다.

- 감독자 스레드가 작업자 스레드에 위임하는 다중 스레드 모델에서는 드라이버에 최소한 작업자 스레드 수 만큼의 세션이 있어야 합니다. 그렇지 않으면 부하가 최대일 때 스레드가 사용 가능한 세션을 기다리게 됩니다.
- 원장당 동시 활성 세션의 서비스 한도는 Amazon QLDB 할당량 및 제한에 정의되어 있습니다. 모든 클라이언트의 단일 원장에 사용할 동시 세션 한도를 초과하여 구성하지 않도록 하십시오.

예외에 대한 재시도

QLDB에서 발생한 예외에 대해 재시도할 때는 다음 권장 사항을 고려하세요.

OccConflictException 재시도

OCC(낙관적 동시성 제어) 충돌 예외는 트랜잭션이 시작된 이후 트랜잭션에서 액세스하는 데이터가 변경된 경우 발생합니다. QLDB는 트랜잭션 커밋을 시도하는 동안 이 예외를 발생시킵니다. 드라이버는 `maxRetries`이 구성된 횟수만큼 트랜잭션을 재시도합니다.

OCC에 대한 자세한 내용과 인덱스를 사용하여 OCC 충돌을 제한하는 모범 사례에 대한 자세한 내용은 Amazon QLDB 동시성 모델 섹션을 참조하세요.

QldbDriver 외부의 다른 예외에 대한 재시도

런타임 중에 애플리케이션에서 정의한 사용자 지정 예외가 발생할 때 드라이버 외부에서 트랜잭션을 재시도하려면 트랜잭션을 래핑해야 합니다. 예를 들어, Java의 경우 다음 코드는 Reslience4J 라이브러리를 사용하여 QLDB에서 트랜잭션을 재시도하는 방법을 보여줍니다.

```
private final RetryConfig retryConfig = RetryConfig.custom()
        .maxAttempts(MAX_RETRIES)
        .intervalFunction(IntervalFunction.ofExponentialRandomBackoff())
        // Retry this exception
        .retryExceptions(InvalidSessionException.class, MyRetryableException.class)
        // But fail for any other type of exception extended from RuntimeException
        .ignoreExceptions(RuntimeException.class)
        .build();

// Method callable by a client
public void myTransactionWithRetries(Params params) {
    Retry retry = Retry.of("registerDriver", retryConfig);

    Function<Params, Void> transactionFunction = Retry.decorateFunction(
            retry,
```

```
            parameters ->  transactionNoReturn(params));
    transactionFunction.apply(params);
}

private Void transactionNoReturn(Params params) {
    try (driver.execute(txn -> {
            // Transaction code
        });
    }
    return null;
}
```

> **Note**
>
> QLDB 드라이버 외부에서 트랜잭션을 재시도하면 승수 효과를 누릴 수 있습니다. 예를 들어, `QldbDriver`가 세 번 재시도하도록 구성되어 있고, 사용자 지정 재시도 로직도 세 번 재시도하면 동일한 트랜잭션을 최대 9번까지 재시도할 수 있습니다.

트랜잭션에 멱등성 부여하기

재시도 시 예상치 못한 부작용이 발생하지 않도록 쓰기 트랜잭션에 멱등성을 부여하는 것이 가장 좋은 방법입니다. 여러 번 실행하여 매번 동일한 결과를 생성할 수 있는 트랜잭션은 멱등성을 가집니다.

자세한 내용은 Amazon QLDB 동시성 모델 섹션을 참조하세요.

성능 최적화

드라이버를 사용하여 트랜잭션을 실행할 때 성능을 최적화하려면 다음 사항을 고려하세요.

- 이 `execute` 작업은 항상 다음 명령을 포함하여 QLDB에 대해 최소 세 번의 SendCommand API 호출을 수행합니다.
  1. `StartTransaction`
  2. `ExecuteStatement`

     이 명령은 `execute` 블록에서 실행하는 각 PartiQL 명령문에 대해 호출됩니다.
  3. `CommitTransaction`

  애플리케이션의 전체 워크로드를 계산할 때 수행되는 총 API 호출 수를 고려하세요.

- 일반적으로 단일 스레드 작성기로 시작하여 단일 트랜잭션 내에서 여러 명령문을 일괄 처리하여 트랜잭션을 최적화하는 것이 좋습니다. Amazon QLDB 할당량 및 제한에 정의된 대로 트랜잭션 크기, 문서 크기 및 트랜잭션당 문서 수의 할당량을 최대화합니다.
- 대규모 트랜잭션 로드에 일괄 처리가 충분하지 않은 경우, 작성자를 추가하여 멀티스레딩을 시도할 수 있습니다. 그러나 문서 및 트랜잭션 시퀀싱에 대한 애플리케이션 요구 사항과 이로 인해 발생하는 추가적인 복잡성을 신중하게 고려해야 합니다.

트랜잭션당 여러 명령문 실행

이전 섹션에서 설명한 대로 트랜잭션당 여러 명령문을 실행하여 애플리케이션의 성능을 최적화할 수 있습니다. 다음 코드 예제에서는 테이블을 쿼리하고 트랜잭션 내에서 해당 테이블의 문서를 업데이트합니다. Lambda 표현식을 `execute` 작업에 전달하여 이를 수행할 수 있습니다.

Java

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
 because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
 or not.
public static boolean InsureCar(QldbDriver qldbDriver, final String vin) {
    final IonSystem ionSystem = IonSystemBuilder.standard().build();
    final IonString ionVin = ionSystem.newString(vin);

    return qldbDriver.execute(txn -> {
        Result result = txn.execute(
                "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
                ionVin);
        if (!result.isEmpty()) {
            txn.execute("UPDATE Vehicles SET insured = TRUE WHERE vin = ?", ionVin);
            return true;
        }
        return false;
    });
}
```

.NET

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
 because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
 or not.
```

```
public static async Task<bool> InsureVehicle(IAsyncQldbDriver driver, string vin)
{
    ValueFactory valueFactory = new ValueFactory();
    IIonValue ionVin = valueFactory.NewString(vin);

    return await driver.Execute(async txn =>
    {
        // Check if the vehicle is insured.
        Amazon.QLDB.Driver.IAsyncResult result = await txn.Execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
 ionVin);

        if (await result.CountAsync() > 0)
        {
            // If the vehicle is not insured, insure it.
            await txn.Execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", ionVin);
            return true;
        }
        return false;
    });
}
```

Go

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
 because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
 or not.
func InsureCar(driver *qldbdriver.QLDBDriver, vin string) (bool, error) {
    insured, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {

        result, err := txn.Execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE", vin)
        if err != nil {
            return false, err
        }

        hasNext := result.Next(txn)
        if !hasNext && result.Err() != nil {
            return false, result.Err()
        }
```

```
        if hasNext {
            _, err = txn.Execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin)
            if err != nil {
                return false, err
            }
            return true, nil
        }
        return false, nil
    })
    if err != nil {
        panic(err)
    }

    return insured.(bool), err
}
```

Node.js

```
// This code snippet is intentionally trivial. In reality you wouldn't do this
 because you'd
// set your UPDATE to filter on vin and insured, and check if you updated something
 or not.
async function insureCar(driver: QldbDriver, vin: string): Promise<boolean> {

    return await driver.executeLambda(async (txn: TransactionExecutor) => {
        const results: dom.Value[] = (await txn.execute(
            "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE",
 vin)).getResultList();

        if (results.length > 0) {
            await txn.execute(
                "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin);
            return true;
        }
        return false;
    });
};
```

Python

```
# This code snippet is intentionally trivial. In reality you wouldn't do this
 because you'd
# set your UPDATE to filter on vin and insured, and check if you updated something
 or not.

def do_insure_car(transaction_executor, vin):
    cursor = transaction_executor.execute_statement(
        "SELECT insured FROM Vehicles WHERE vin = ? AND insured = FALSE", vin)
    first_record = next(cursor, None)
    if first_record:
        transaction_executor.execute_statement(
            "UPDATE Vehicles SET insured = TRUE WHERE vin = ?", vin)
        return True
    else:
        return False

def insure_car(qldb_driver, vin_to_insure):
    return qldb_driver.execute_lambda(
        lambda executor: do_insure_car(executor, vin_to_insure))
```

드라이버의 `execute` 작업은 세션 및 해당 세션에서 트랜잭션을 암시적으로 시작합니다. Lambda 표현식에서 실행하는 각 명령문은 트랜잭션에 래핑됩니다. 모든 명령문이 실행된 후 드라이버는 트랜잭션을 자동 커밋합니다. 자동 재시도 한도를 모두 사용한 후 명령문이 하나라도 실패하면 트랜잭션이 중단됩니다.

트랜잭션에 예외 전파

트랜잭션당 여러 명령문을 실행하는 경우 일반적으로 트랜잭션 내에서 예외를 캐치하고 가리지 않는 것이 좋습니다.

예를 들어, Java에서 다음 프로그램은 `RuntimeException`의 모든 인스턴스를 캐치하고 오류를 기록하고 계속합니다. 이 코드 예제는 `UPDATE` 명령문이 실패하더라도 트랜잭션이 성공하기 때문에 잘못된 관행으로 간주됩니다. 따라서 클라이언트는 업데이트가 성공하지 못했지만 성공했다고 가정할 수 있습니다.

> **Warning**
>
> 이 코드 예제를 사용하지 마세요. 이 예제는 잘못된 관행으로 간주되는 안티 패턴 예제를 보여 주기 위해 제공되었습니다.

```
// DO NOT USE this code example because it is considered bad practice
public static void main(final String... args) {
    ConnectToLedger.getDriver().execute(txn -> {
        final Result selectTableResult = txn.execute("SELECT * FROM Vehicle WHERE VIN
 ='123456789'");
        // Catching an error inside the transaction is an anti-pattern because the
 operation might
        // not succeed.
        // In this example, the transaction succeeds even when the update statement
 fails.
        // So, the client might assume that the update succeeded when it didn't.
        try {
            processResults(selectTableResult);
            String model = // some code that extracts the model
            final Result updateResult = txn.execute("UPDATE Vehicle SET model = ? WHERE
 VIN = '123456789'",
                    Constants.MAPPER.writeValueAsIonValue(model));
        } catch (RuntimeException e) {
            log.error("Exception when updating the Vehicle table {}", e.getMessage());
        }
    });
    log.info("Vehicle table updated successfully.");
}
```

대신 예외를 전파(버블 업)하세요. 트랜잭션의 일부가 실패하는 경우, 클라이언트가 그에 따라 예외를 처리할 수 있도록 `execute` 작업을 통해 트랜잭션을 중단합니다.

Amazon QLDB의 드라이버를 사용한 재시도 정책에 대한 이해

Amazon QLDB 드라이버는 재시도 정책을 사용하여 실패한 트랜잭션을 투명하게 재시도하여 일시적인 예외를 처리합니다. 이러한 예외(예: `CapacityExceededException` 및 `RateExceededException`)는 일반적으로 일정 시간이 지나면 자동으로 수정됩니다. 예외와 함께 실패한 트랜잭션을 적절한 지연 후에 다시 시도하면 성공할 가능성이 높습니다. 이는 QLDB를 사용하는 애플리케이션의 안정성을 개선하는 데 도움이 됩니다.

주제

재시도 가능한 오류 타입

드라이버는 트랜잭션 내에서 작업을 수행하는 동안 다음과 같은 예외가 하나라도 발생하는 경우에만 트랜잭션을 자동으로 재시도합니다.

- CapacityExceededException - 요청이 원장의 처리 용량을 초과할 때 반환됩니다.
- InvalidSessionException - 세션이 더 이상 유효하지 않거나 세션이 존재하지 않을 때 반환됩니다.
- LimitExceededException - 활성 세션 수와 같은 리소스 제한을 초과할 경우 반환됩니다.
- OccConflictException - OCC(낙관적 동시성 제어)의 검증 단계에서 실패로 인해 트랜잭션을 저널에 기록할 수 없을 때 반환됩니다.
- RateExceededException - 요청 비율이 허용된 처리량을 초과할 때 반환됩니다.

기본 키 정책

재시도 정책은 재시도 조건과 백오프 전략으로 구성됩니다. 재시도 조건은 트랜잭션을 재시도해야 하는 시기를 정의하고, 백오프 전략은 트랜잭션을 재시도하기 전에 기다려야 하는 시간을 정의합니다.

드라이버 인스턴스를 만들 때 기본 재시도 정책은 최대 4회까지 재시도하고 지수 백오프 전략을 사용하도록 지정합니다. 지수 백오프 전략은 지터가 동일한 최소 10밀리초와 최대 5000밀리초의 지연을 사용합니다. 재시도 정책 내에서 트랜잭션을 성공적으로 체결할 수 없는 경우 다른 시간에 트랜잭션을 시도하는 것이 좋습니다.

지수 백오프의 개념은 오류 응답이 연속될 때마다 재시도 간 대기 시간을 점진적으로 늘린다는 것입니다. 자세한 설명은 AWS 블로그 게시물 지수 백오프 및 지터를 참조하세요.

Amazon QLDB 드라이버에서 발생하는 일반적인 오류

이 섹션에서는 QLDB 세션 API와 상호 작용할 때 Amazon QLDB 드라이버에서 발생할 수 있는 런타임 오류에 대해 설명합니다.

다음은 드라이버가 반환하는 일반적인 예외 목록입니다. 각 예외에는 특정 오류 메시지와 가능한 해결 방법에 대한 간단한 설명 및 제안이 포함됩니다.

CapacityExceededException

메시지: 용량 초과

Amazon QLDB는 원장의 처리 용량을 초과했기 때문에 요청을 거부했습니다. QLDB는 서비스의 상태와 성능을 유지하기 위해 원장당 내부 크기 조정 한도를 적용합니다. 이 한도는 각 개별 요청의 워크로드 크기에 따라 달라집니다. 예를 들어, 인덱스가 아닌 정규화된 쿼리로 인한 테이블 스캔과 같은 비효율적인 데이터 트랜잭션을 수행하는 요청의 경우 워크로드가 증가할 수 있습니다.

요청을 재시도하기 전에 잠시 기다릴 것을 권장합니다. 애플리케이션에서 이 예외가 지속적으로 발생하는 경우 문을 최적화하고 원장에 보내는 요청의 속도와 볼륨을 줄이십시오. 명령문 최적화의 예로는 트랜잭션당 더 적은 수의 명령문을 실행하고 테이블 인덱스를 조정하는 경우를 들 수 있습니다. 명령문을 최적화하고 테이블 스캔을 방지하는 방법을 알아보려면 쿼리 성능 최적화을 참조하십시오.

또한 최신 버전의 QLDB 드라이버를 사용할 것을 권장합니다. 드라이버에는 지수 백오프 및 Jitter를 사용하여 이와 같은 예외가 발생할 경우 자동으로 재시도하는 기본 재시도 정책이 있습니다. 지수 백오프의 개념은 오류 응답이 연속될 때마다 재시도 간 대기 시간을 점진적으로 늘린다는 것입니다.

InvalidSessionException

메시지: 트랜잭션 *transactionId*가 만료되었습니다

트랜잭션이 최대 수명을 초과했습니다. 트랜잭션은 커밋되기 전까지 최대 30초 동안 실행될 수 있습니다. 이 시간 초과 제한 이후에는 트랜잭션에 대해 수행된 모든 작업이 거부되고 QLDB는 세션을 삭제합니다. 이 제한은 트랜잭션을 시작하고 커밋하거나 취소하지 않기 때문에 클라이언트의 세션 유출을 방지합니다.

이것이 애플리케이션에서 일반적인 예외인 경우, 트랜잭션을 실행하는 데 시간이 너무 오래 걸릴 수 있습니다. 트랜잭션 런타임이 30초 이상 걸리는 경우 문을 최적화하여 트랜잭션 속도를 높이십시오. 명령문 최적화의 예로는 트랜잭션당 더 적은 수의 명령문을 실행하고 테이블 인덱스를 조정하는 경우를 들 수 있습니다. 자세한 내용은 쿼리 성능 최적화을 참조하십시오.

InvalidSessionException

메시지: 세션 *sessionId*가 만료되었습니다

최대 총 수명을 초과했기 때문에 QLDB가 세션을 삭제했습니다. QLDB는 활성 트랜잭션과 상관없이 13~17분 후에 세션을 삭제합니다. 세션은 하드웨어 장애, 네트워크 장애 또는 애플리케이션 재시작과 같은 여러 가지 이유로 손실되거나 손상될 수 있습니다. 따라서 QLDB는 클라이언트 소프트웨어가 세션 장애에 대해 복원력을 유지할 수 있도록 세션에 최대 수명을 적용합니다.

이 예외가 발생하는 경우 새 세션을 획득하고 트랜잭션을 다시 시도할 것을 권장합니다. 또한 애플리케이션을 대신하여 세션 풀과 해당 상태를 관리하는 최신 버전의 QLDB 드라이버를 사용할 것을 권장합니다.

InvalidSessionException

메시지: 해당 세션이 없습니다

클라이언트가 존재하지 않는 세션을 사용하여 QLDB와 트랜잭션을 시도했습니다. 클라이언트가 이전에 존재했던 세션을 사용하고 있다고 가정하면 다음 중 하나로 인해 세션이 더 이상 존재하지 않을 수 있습니다.

- 세션이 내부 서버 장애(즉, HTTP 응답 코드 500 오류)와 관련된 경우, QLDB는 고객이 불확실한 상태의 세션과 트랜잭션하도록 허용하는 대신 해당 세션을 완전히 삭제하도록 선택할 수 있습니다. 그러면 해당 세션에 대한 모든 재시도가 실패하고 이 오류가 발생합니다.
- QLDB는 만료된 세션을 결국 잊어버립니다. 그런 다음 세션을 계속 사용하려고 하면 초기 `InvalidSessionException`이 아닌 이 오류가 발생합니다.

이 예외가 발생하는 경우 새 세션을 획득하고 트랜잭션을 다시 시도할 것을 권장합니다. 또한 애플리케이션을 대신하여 세션 풀과 해당 상태를 관리하는 최신 버전의 QLDB 드라이버를 사용할 것을 권장합니다.

RateExceededException

메시지: 속도가 초과되었습니다

QLDB는 발신자의 ID를 기반으로 클라이언트를 제한했습니다. QLDB는 토큰 버킷 제한 알고리즘을 사용하여 리전별, 계정별로 제한을 적용합니다. QLDB는 서비스의 성과를 돕고 모든 QLDB 고객의 공정한 사용을 보장하기 위해 이를 수행합니다. 예를 들어, `StartSessionRequest` 작업을 사용하여 많은 수의 동시 세션을 획득하려고 하면 제한이 발생할 수 있습니다.

애플리케이션 상태를 유지하고 추가적인 제한을 완화하기 위해 지수 백오프 및 Jitter를 사용하여 이 예외를 다시 시도할 수 있습니다. 지수 백오프의 개념은 오류 응답이 연속될 때마다 재시도 간 대기 시간을 점진적으로 늘린다는 것입니다. 최신 버전의 QLDB 드라이버를 사용하는 것이 좋습니다. 드라이버에는 지수 백오프 및 Jitter를 사용하여 이와 같은 예외가 발생할 경우 자동으로 재시도하는 기본 재시도 정책이 있습니다.

최신 버전의 QLDB 드라이버는 애플리케이션이 `StartSessionRequest` 호출에 대해 QLDB에 의해 지속적으로 제한되는 경우에도 도움이 될 수 있습니다. 드라이버는 여러 트랜잭션에서 재사용되는 세션 풀을 유지 관리하므로 애플리케이션이 수행하는 `StartSessionRequest` 호출 수를 줄이는데 도움이 될 수 있습니다. API 호출 한도 증가를 요청하려면 AWS Support 센터로 문의하십시오.

LimitExceededException

메시지: 세션 제한을 초과했습니다

원장이 활성 세션 수의 할당량(한도라고도 함)을 초과했습니다. 이 할당량은 Amazon QLDB 할당량 및 제한에 정의되어 있습니다. 원장의 활성 세션 수는 최종적으로 일정하게 유지되며, 할당량 근처에서 지속적으로 실행되는 원장에서는 이 예외가 주기적으로 발생할 수 있습니다.

애플리케이션의 상태를 유지하려면 이 예외를 다시 시도할 것을 권장합니다. 이 예외를 피하려면 모든 클라이언트에서 단일 원장에 1,500개 이상의 동시 세션을 사용하도록 구성하지 않았는지 확인하십시오. 예를 들어, Java용 Amazon QLDB 드라이버의 MaxConcurrentTransactions 메서드를 사용하여 드라이버 인스턴스에서 사용 가능한 최대 세션 수를 구성할 수 있습니다.

QldbClientException

메시지: 스트리밍된 결과는 상위 트랜잭션이 열려 있을 때만 유효합니다

트랜잭션이 종료되었으며 QLDB에서 결과를 검색하는 데 사용할 수 없습니다. 트랜잭션은 커밋되거나 취소되면 종료됩니다.

이 예외는 클라이언트가 `Transaction` 객체로 직접 작업하고 트랜잭션을 커밋하거나 취소한 후 QLDB에서 결과를 검색하려고 할 때 발생합니다. 이 문제를 완화하려면 클라이언트가 트랜잭션을 닫기 전에 데이터를 읽어야 합니다.

샘플 애플리케이션 자습서를 사용하여 Amazon QLDB 시작하기

이 자습서에서는 Amazon QLDB 드라이버를 AWS SDK와 함께 사용하여 QLDB 원장을 생성하고 샘플 데이터로 채웁니다. 드라이버를 사용하면 애플리케이션이 트랜잭션 데이터 API를 사용하여 QLDB와 상호 작용할 수 있습니다. AWS SDK는 QLDB 리소스 관리 API와의 상호 작용을 지원합니다.

이 시나리오에서 생성하는 샘플 원장은 차량 등록에 대한 전체 기록 정보를 추적하는 자동차 부서(DMV) 데이터베이스입니다. 다음 항목에서는 차량 등록을 추가하고, 수정하고, 해당 등록의 변경 기록을 보는 방법을 설명합니다. 또한 이 안내서는 등록 문서를 암호적으로 검증하는 방법을 보여주며, 리소스를 정리하고 샘플 원장을 삭제하는 것으로 마무리합니다.

샘플 애플리케이션 자습서는 다음 프로그래밍 언어로 제공됩니다.

주제

- Amazon QLDB Java 자습서
- Amazon QLDB Node.js 자습서

- Amazon QLDB Python 자습서

Amazon QLDB Java 자습서

이 자습서 샘플 애플리케이션의 구현에서는 Amazon QLDB 드라이버를 AWS SDK for Java와 함께 사용하여 QLDB 원장을 생성하고 이를 샘플 데이터로 채웁니다.

이 자습서로 학습하면서 관리 API 작업에 대해서 AWS SDK for Java API 참조를 참조할 수 있습니다. 트랜잭션 데이터 작업의 경우 Java용 QLDB 드라이버 API 참조를 참조할 수 있습니다.

> **Note**
>
> 해당하는 경우 일부 자습서 단계에는 Java용 QLDB 드라이버의 지원되는 각 메이저 버전마다 서로 다른 명령 또는 코드 예제가 있습니다.

주제

- Amazon QLDB Java 샘플 애플리케이션 설치
- 1단계: 새 원장 생성
- 2단계: 원장과의 연결을 테스트
- 3단계: 테이블, 인덱스 및 샘플 데이터 생성
- 4단계: 원장에서 테이블 쿼리
- 5단계: 원장의 문서 수정
- 6단계: 문서의 개정 기록 보기
- 7단계: 원장에 있는 문서 검증
- 8단계: 원장의 저널 데이터 내보내기 및 검증
- 9단계(선택 사항): 리소스 정리

Amazon QLDB Java 샘플 애플리케이션 설치

이 섹션에서는 단계별 Java 자습서를 위해 제공된 Amazon QLDB 샘플 애플리케이션을 설치하고 실행하는 방법을 설명합니다. 이 샘플 애플리케이션의 사용 사례는 차량 등록에 대한 전체 기록 정보를 추적하는 자동차 부서(DMV) 데이터베이스입니다.

Java용 DMV 샘플 애플리케이션은 GitHub 리포지토리 aws-samples/amazon-qldb-dmv-sample-java의 오픈 소스입니다.

사전 조건

시작하기 전에 Java 사전 조건용 QLDB 드라이버를 완료했는지 확인합니다. 다음 내용이 해당됩니다.

1. AWS에 가입합니다.
2. 적절한 QLDB 권한을 가진 사용자를 생성합니다. 이 자습서의 모든 단계를 완료하려면 QLDB API를 통해 원장 리소스에 대한 전체 관리 액세스 권한이 필요합니다.
3. AWS Cloud9 이외의 IDE를 사용하는 경우 Java를 설치하고 개발을 위한 프로그래밍 방식 액세스 권한을 부여하십시오.

설치

다음 단계는 로컬 개발 환경에서 샘플 애플리케이션을 다운로드하고 설정하는 방법을 설명합니다. 또는 AWS Cloud9를 IDE로 사용하고 AWS CloudFormation 템플릿을 사용하여 개발 리소스를 프로비저닝하여 샘플 애플리케이션의 설정을 자동화할 수 있습니다.

로컬 개발 환경

이 지침은 자체 리소스 및 개발 환경을 사용하여 QLDB Java 샘플 애플리케이션을 다운로드하고 설치하는 방법을 설명합니다.

샘플 애플리케이션을 다운로드하여 실행하려면

1. 다음 명령을 입력하여 GitHub에서 샘플 애플리케이션을 복제합니다.

   2.x

   ```
   git clone https://github.com/aws-samples/amazon-qldb-dmv-sample-java.git
   ```

   1.x

   ```
   git clone -b v1.2.0 https://github.com/aws-samples/amazon-qldb-dmv-sample-
   java.git
   ```

   이 패키지에는 Gradle 구성 및 Java 자습서의 전체 코드가 포함되어 있습니다.
2. 제공된 애플리케이션을 로드하고 실행합니다.
   - Eclipse를 사용하는 경우:

a. Eclipse를 시작하고 Eclipse 메뉴에서 파일, 가져오기, 기존 Gradle 프로젝트를 차례로 선택합니다.

b. 프로젝트 루트 디렉터리에서 `build.gradle` 파일이 포함된 애플리케이션 디렉터리를 찾아 선택합니다. 그런 다음 완료를 선택하여 기본 Gradle 설정을 가져오기에 사용합니다.

c. 예를 들어, `ListLedgers` 프로그램을 실행해 볼 수 있습니다. `ListLedgers.java` 파일에 대한 컨텍스트 메뉴를 열고(마우스 오른쪽 버튼 클릭) Java 애플리케이션으로 실행을 선택합니다.

- IntelliJ를 사용하는 경우:

a. IntelliJ를 시작하고 IntelliJ 메뉴에서 파일을 선택한 다음 열기를 선택합니다.

b. 프로젝트 루트 디렉터리에서 `build.gradle` 파일이 포함된 애플리케이션 디렉터리를 찾아 선택합니다. 그 다음에 확인을 선택합니다. 기본 설정을 유지하고 확인을 다시 선택합니다.

c. 예를 들어, `ListLedgers` 프로그램을 실행해 볼 수 있습니다. `ListLedgers.java` 파일에 대한 컨텍스트 메뉴를 열고(마우스 오른쪽 버튼 클릭) 'ListLedgers' 실행을 선택합니다.

3. 1단계: 새 원장 생성로 진행하여 자습서를 시작하고 원장을 생성하십시오.

AWS Cloud9

이 지침에서는 AWS Cloud9을 IDE로 사용하여 Java용 Amazon QLDB 차량 등록 샘플 애플리케이션을 자동으로 설정하는 방법을 설명합니다. 이 안내서에서는 AWS CloudFormation 템플릿을 사용하여 개발 리소스를 프로비저닝합니다.

AWS Cloud9에 대한 추가 정보는 AWS Cloud9 사용 설명서를 참조하십시오. AWS CloudFormation에 대한 자세한 내용은 AWS CloudFormation 사용 설명서를 참조하십시오.

주제

- 1부: 리소스 프로비저닝
- 2부: IDE 설정
- 3부: QLDB DMV 샘플 애플리케이션 실행

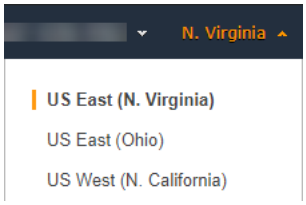
1부: 리소스 프로비저닝

이 첫 번째 단계에서는 AWS CloudFormation을 사용하여 Amazon QLDB 샘플 애플리케이션을 사용하여 개발 환경을 설정하는 데 필요한 리소스를 프로비저닝합니다.

AWS CloudFormation 콘솔을 열고 QLDB 샘플 애플리케이션 템플릿을 로드하려면

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/cloudformation에서 AWS CloudFormation 콘솔을 엽니다.

   QLDB를 지원하는 리전으로 전환합니다. 전체 목록은 AWS 일반 참조에서 Amazon QLDB 엔드포인트 및 할당량을 참조하십시오. 다음 AWS Management Console 스크린샷은 미국 동부(버지니아 북부)가 AWS 리전로 선택된 모습입니다.

2. AWS CloudFormation 콘솔에서 스택 생성을 선택한 다음 새 리소스 사용(표준)을 선택합니다.
3. 템플릿 지정 아래 스택 생성 페이지에서 Amazon S3 URL을 선택합니다.
4. 다음 URL을 입력하고 다음을 선택합니다.

   ```
   https://amazon-qldb-assets.s3.amazonaws.com/templates/QLDB-DMV-SampleApp.yml
   ```

5. **qldb-sample-app**와 같은 스택 이름을 입력한 후 다음을 선택합니다.
6. 원하는 대로 태그를 추가하고 기본 옵션을 유지할 수 있습니다. 이후 다음을 선택합니다.
7. 스택 설정을 검토하고 스택 생성을 선택합니다. AWS CloudFormation 스크립트를 완료하려면 몇 분 정도 걸릴 수 있습니다.

   이 스크립트는 이 자습서에서 QLDB 샘플 애플리케이션을 실행하는 데 사용하는 연결된 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스를 사용하여 AWS Cloud9 환경을 프로비저닝합니다. 또한 GitHub의 aws-samples/amazon-qldb-dmv-sample-java 리포지토리를 AWS Cloud9 개발 환경으로 복제합니다.

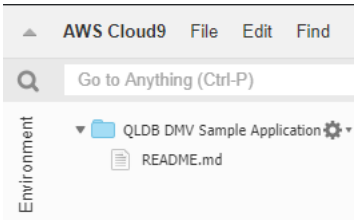
2부: IDE 설정

이 단계에서는 클라우드 개발 환경 설정을 완료합니다. 제공된 쉘 스크립트를 다운로드하고 실행하여 샘플 애플리케이션의 종속성을 가진 AWS Cloud9 IDE를 설정할 수 있습니다.

AWS Cloud9 환경을 설정하려면

1. https://console.aws.amazon.com/cloud9/에서 AWS Cloud9 콘솔을 엽니다.
2. 사용자 환경에서 QLDB DMV 샘플 애플리케이션이라는 환경 이름의 카드를 찾아 IDE 열기를 선택합니다. 기본 EC2 인스턴스가 시작될 때 환경을 로드하는 데 1분 정도 걸릴 수 있습니다.

   AWS Cloud9 환경은 자습서를 실행하는 데 필요한 시스템 종속성으로 미리 구성되어 있습니다. 콘솔의 환경 탐색 창에 `QLDB DMV Sample Application`로 이름이 지정된 폴더가 표시되는지 확인합니다. AWS Cloud9 콘솔의 다음 스크린샷은 QLDB DMV 샘플 애플리케이션 환경 폴더 창을 보여줍니다.

   탐색 창이 보이지 않으면 콘솔 왼쪽에 있는 환경 탭을 전환하십시오. 패널에 폴더가 보이지 않는 경우, 설정 아이콘
   ()
   을 사용하여 환경 루트 표시를 활성화하십시오.
3. 콘솔 하단 창에 열린 bash 터미널 창이 보일 것입니다. 이 화면이 보이지 않으면 콘솔 상단의 창 메뉴에서 새 터미널을 선택합니다.
4. 그런 다음 설치 스크립트를 다운로드하여 실행하여 OpenJDK 8을 설치하고 해당하는 경우 Git 리포지토리에서 적절한 브랜치를 확인합니다. 이전 단계에서 생성한 AWS Cloud9 터미널에서 다음 두 가지 명령을 순서대로 실행합니다.

   2.x

   ```
   aws s3 cp s3://amazon-qldb-assets/setup-scripts/dmv-setup-v2.sh .
   ```

   ```
   sh dmv-setup-v2.sh
   ```

   1.x

   ```
   aws s3 cp s3://amazon-qldb-assets/setup-scripts/dmv-setup.sh .
   ```

```
sh dmv-setup.sh
```

완료되면 터미널에서 다음 메시지가 출력됩니다.

```
** DMV Sample App setup completed , enjoy!! **
```

5. 잠시 시간을 내어 AWS Cloud9의 샘플 애플리케이션 코드, 특히 다음 디렉터리 경로 `src/main/java/software/amazon/qldb/tutorial`를 찾아보십시오.

3부: QLDB DMV 샘플 애플리케이션 실행

이 단계에서는 AWS Cloud9를 사용하여 Amazon QLDB DMV 샘플 애플리케이션 작업을 실행하는 방법을 알아봅니다. 샘플 코드를 실행하려면 2부: IDE 설정에서와 같이 AWS Cloud9 터미널로 돌아가거나 새 터미널 창을 만드십시오.

샘플 애플리케이션을 실행하려면

1. 터미널에서 다음 명령을 실행하여 프로젝트 루트 디렉터리로 전환합니다.

```
cd ~/environment/amazon-qldb-dmv-sample-java
```

다음 디렉터리 경로에서 예제를 실행하고 있는지 확인하십시오.

```
/home/ec2-user/environment/amazon-qldb-dmv-sample-java/
```

2. 다음 명령어는 각 작업을 실행하는 Gradle 구문을 보여줍니다.

```
./gradlew run -Dtutorial=Task
```

예를 들어, 다음 명령을 실행하여 AWS 계정 및 현재 리전의 모든 원장을 나열합니다.

```
./gradlew run -Dtutorial=ListLedgers
```

3. 1단계: 새 원장 생성로 진행하여 자습서를 시작하고 원장을 생성하십시오.
4. (선택 사항) 자습서를 완료한 후 AWS CloudFormation 리소스가 더 이상 필요하지 않으면 정리하십시오.

a. https://console.aws.amazon.com/cloudformation에서 AWS CloudFormation 콘솔을 열고 1부: 리소스 프로비저닝에서 만든 스택을 삭제합니다.

b. 또한 AWS CloudFormation 템플릿에서 생성한 AWS Cloud9 스택도 삭제하십시오.

1단계: 새 원장 생성

이 단계에서는 `vehicle-registration`라는 이름의 새 Amazon QLDB 원장을 생성합니다.

새 원장을 생성하려면

1. 이 자습서의 다른 모든 프로그램에서 사용하는 상수 값이 들어 있는 다음 파일(`Constants.java`)을 검토하십시오.

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */
```

```
package software.amazon.qldb.tutorial;

import com.amazon.ion.IonSystem;
import com.amazon.ion.system.IonSystemBuilder;
import com.fasterxml.jackson.databind.SerializationFeature;
import com.fasterxml.jackson.dataformat.ion.IonObjectMapper;
import com.fasterxml.jackson.dataformat.ion.ionvalue.IonValueMapper;

/**
 * Constant values used throughout this tutorial.
 */
public final class Constants {
    public static final int RETRY_LIMIT = 4;
    public static final String LEDGER_NAME = "vehicle-registration";
    public static final String STREAM_NAME = "vehicle-registration-stream";
    public static final String VEHICLE_REGISTRATION_TABLE_NAME =
 "VehicleRegistration";
    public static final String VEHICLE_TABLE_NAME = "Vehicle";
    public static final String PERSON_TABLE_NAME = "Person";
    public static final String DRIVERS_LICENSE_TABLE_NAME = "DriversLicense";
    public static final String VIN_INDEX_NAME = "VIN";
    public static final String PERSON_GOV_ID_INDEX_NAME = "GovId";
    public static final String
 VEHICLE_REGISTRATION_LICENSE_PLATE_NUMBER_INDEX_NAME = "LicensePlateNumber";
    public static final String DRIVER_LICENSE_NUMBER_INDEX_NAME =
 "LicenseNumber";
    public static final String DRIVER_LICENSE_PERSONID_INDEX_NAME = "PersonId";
    public static final String JOURNAL_EXPORT_S3_BUCKET_NAME_PREFIX = "qldb-
tutorial-journal-export";
    public static final String USER_TABLES = "information_schema.user_tables";
    public static final String LEDGER_NAME_WITH_TAGS = "tags";
    public static final IonSystem SYSTEM = IonSystemBuilder.standard().build();
    public static final IonObjectMapper MAPPER = new IonValueMapper(SYSTEM);

    private Constants() { }

    static {
        MAPPER.disable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);
    }
}
```

1.x

> **Important**
>
> Amazon Ion 패키지의 경우 애플리케이션에서 네임스페이스 `com.amazon.ion`을 사용해야 합니다. AWS SDK for Java은 네임스페이스 `software.amazon.ion` 아래의 다른 Ion 패키지에 따라 다르지만 이 패키지는 QLDB 드라이버와 호환되지 않는 레거시 패키지입니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonSystem;
import com.amazon.ion.system.IonSystemBuilder;
import com.fasterxml.jackson.databind.SerializationFeature;
```

```
import com.fasterxml.jackson.dataformat.ion.IonObjectMapper;
import com.fasterxml.jackson.dataformat.ion.ionvalue.IonValueMapper;

/**
 * Constant values used throughout this tutorial.
 */
public final class Constants {
    public static final int RETRY_LIMIT = 4;
    public static final String LEDGER_NAME = "vehicle-registration";
    public static final String STREAM_NAME = "vehicle-registration-stream";
    public static final String VEHICLE_REGISTRATION_TABLE_NAME =
 "VehicleRegistration";
    public static final String VEHICLE_TABLE_NAME = "Vehicle";
    public static final String PERSON_TABLE_NAME = "Person";
    public static final String DRIVERS_LICENSE_TABLE_NAME = "DriversLicense";
    public static final String VIN_INDEX_NAME = "VIN";
    public static final String PERSON_GOV_ID_INDEX_NAME = "GovId";
    public static final String
 VEHICLE_REGISTRATION_LICENSE_PLATE_NUMBER_INDEX_NAME = "LicensePlateNumber";
    public static final String DRIVER_LICENSE_NUMBER_INDEX_NAME =
 "LicenseNumber";
    public static final String DRIVER_LICENSE_PERSONID_INDEX_NAME = "PersonId";
    public static final String JOURNAL_EXPORT_S3_BUCKET_NAME_PREFIX = "qldb-
tutorial-journal-export";
    public static final String USER_TABLES = "information_schema.user_tables";
    public static final String LEDGER_NAME_WITH_TAGS = "tags";
    public static final IonSystem SYSTEM = IonSystemBuilder.standard().build();
    public static final IonObjectMapper MAPPER = new IonValueMapper(SYSTEM);

    private Constants() { }

    static {
        MAPPER.disable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);
    }
}
```

> **Note**
>
> 이 `Constants` 클래스에는 오픈 소스 Jackson `IonValueMapper` 클래스의 인스턴스가 포함되어 있습니다. 읽기 및 쓰기 트랜잭션을 수행할 때 이 매퍼를 사용하여 Amazon Ion 데이터를 처리할 수 있습니다.

또한 CreateLedger.java 파일은 원장의 현재 상태를 설명하는 다음 프로그램(DescribeLedger.java)에 대한 종속성을 가집니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.model.DescribeLedgerRequest;
import com.amazonaws.services.qldb.model.DescribeLedgerResult;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Describe a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
```

```
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class DescribeLedger {
    public static AmazonQLDB client = CreateLedger.getClient();
    public static final Logger log = LoggerFactory.getLogger(DescribeLedger.class);

    private DescribeLedger() { }

    public static void main(final String... args) {
        try {

            describe(Constants.LEDGER_NAME);

        } catch (Exception e) {
            log.error("Unable to describe a ledger!", e);
        }
    }

    /**
     * Describe a ledger.
     *
     * @param name
     *              Name of the ledger to describe.
     * @return {@link DescribeLedgerResult} from QLDB.
     */
    public static DescribeLedgerResult describe(final String name) {
        log.info("Let's describe ledger with name: {}...", name);
        DescribeLedgerRequest request = new DescribeLedgerRequest().withName(name);
        DescribeLedgerResult result = client.describeLedger(request);
        log.info("Success. Ledger description: {}", result);
        return result;
    }
}
```

2. `CreateLedger.java` 프로그램을 컴파일하고 실행하여 `vehicle-registration`라는 이름의 원장을 생성합니다.

   2.x

   ```
   /*
    * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
    * SPDX-License-Identifier: MIT-0
   ```

```
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.AmazonQLDBClientBuilder;
import com.amazonaws.services.qldb.model.CreateLedgerRequest;
import com.amazonaws.services.qldb.model.CreateLedgerResult;
import com.amazonaws.services.qldb.model.DescribeLedgerResult;
import com.amazonaws.services.qldb.model.LedgerState;
import com.amazonaws.services.qldb.model.PermissionsMode;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Create a ledger and wait for it to be active.
 * <p>
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class CreateLedger {
```

```
    public static final Logger log =
LoggerFactory.getLogger(CreateLedger.class);
    public static final Long LEDGER_CREATION_POLL_PERIOD_MS = 10_000L;
    public static String endpoint = null;
    public static String region = null;
    public static AmazonQLDB client = getClient();

    private CreateLedger() {
    }

    /**
     * Build a low-level QLDB client.
     *
     * @return {@link AmazonQLDB} control plane client.
     */
    public static AmazonQLDB getClient() {
        AmazonQLDBClientBuilder builder = AmazonQLDBClientBuilder.standard();
        if (null != endpoint && null != region) {
            builder.setEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(endpoint, region));
        }
        return builder.build();
    }

    public static void main(final String... args) throws Exception {
        try {
            client = getClient();

            create(Constants.LEDGER_NAME);

            waitForActive(Constants.LEDGER_NAME);

        } catch (Exception e) {
            log.error("Unable to create the ledger!", e);
            throw e;
        }
    }

    /**
     * Create a new ledger with the specified ledger name.
     *
     * @param ledgerName Name of the ledger to be created.
     * @return {@link CreateLedgerResult} from QLDB.
     */
```

```
    public static CreateLedgerResult create(final String ledgerName) {
        log.info("Let's create the ledger with name: {}...", ledgerName);
        CreateLedgerRequest request = new CreateLedgerRequest()
                .withName(ledgerName)
                .withPermissionsMode(PermissionsMode.ALLOW_ALL);
        CreateLedgerResult result = client.createLedger(request);
        log.info("Success. Ledger state: {}.", result.getState());
        return result;
    }

    /**
     * Wait for a newly created ledger to become active.
     *
     * @param ledgerName Name of the ledger to wait on.
     * @return {@link DescribeLedgerResult} from QLDB.
     * @throws InterruptedException if thread is being interrupted.
     */
    public static DescribeLedgerResult waitForActive(final String ledgerName)
 throws InterruptedException {
        log.info("Waiting for ledger to become active...");
        while (true) {
            DescribeLedgerResult result = DescribeLedger.describe(ledgerName);
            if (result.getState().equals(LedgerState.ACTIVE.name())) {
                log.info("Success. Ledger is active and ready to use.");
                return result;
            }
            log.info("The ledger is still creating. Please wait...");
            Thread.sleep(LEDGER_CREATION_POLL_PERIOD_MS);
        }
    }
}
```

1.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
```

```
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.AmazonQLDBClientBuilder;
import com.amazonaws.services.qldb.model.CreateLedgerRequest;
import com.amazonaws.services.qldb.model.CreateLedgerResult;
import com.amazonaws.services.qldb.model.DescribeLedgerResult;
import com.amazonaws.services.qldb.model.LedgerState;
import com.amazonaws.services.qldb.model.PermissionsMode;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Create a ledger and wait for it to be active.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class CreateLedger {
    public static final Logger log =
 LoggerFactory.getLogger(CreateLedger.class);
    public static final Long LEDGER_CREATION_POLL_PERIOD_MS = 10_000L;
    public static AmazonQLDB client = getClient();
```

```
    private CreateLedger() { }

    /**
     * Build a low-level QLDB client.
     *
     * @return {@link AmazonQLDB} control plane client.
     */
    public static AmazonQLDB getClient() {
        return AmazonQLDBClientBuilder.standard().build();
    }

    public static void main(final String... args) throws Exception {
        try {

            create(Constants.LEDGER_NAME);

            waitForActive(Constants.LEDGER_NAME);

        } catch (Exception e) {
            log.error("Unable to create the ledger!", e);
            throw e;
        }
    }

    /**
     * Create a new ledger with the specified ledger name.
     *
     * @param ledgerName
     *              Name of the ledger to be created.
     * @return {@link CreateLedgerResult} from QLDB.
     */
    public static CreateLedgerResult create(final String ledgerName) {
        log.info("Let's create the ledger with name: {}...", ledgerName);
        CreateLedgerRequest request = new CreateLedgerRequest()
                .withName(ledgerName)
                .withPermissionsMode(PermissionsMode.ALLOW_ALL);
        CreateLedgerResult result = client.createLedger(request);
        log.info("Success. Ledger state: {}.", result.getState());
        return result;
    }

    /**
     * Wait for a newly created ledger to become active.
     *
```

```
     * @param ledgerName
     *              Name of the ledger to wait on.
     * @return {@link DescribeLedgerResult} from QLDB.
     * @throws InterruptedException if thread is being interrupted.
     */
    public static DescribeLedgerResult waitForActive(final String ledgerName)
 throws InterruptedException {
        log.info("Waiting for ledger to become active...");
        while (true) {
            DescribeLedgerResult result = DescribeLedger.describe(ledgerName);
            if (result.getState().equals(LedgerState.ACTIVE.name())) {
                log.info("Success. Ledger is active and ready to use.");
                return result;
            }
            log.info("The ledger is still creating. Please wait...");
            Thread.sleep(LEDGER_CREATION_POLL_PERIOD_MS);
        }
    }
}
```

> **Note**
>
> - `createLedger` 호출 시 원장 이름과 권한 모드를 지정해야 합니다. 원장 데이터의 보안을 극대화하려면 `STANDARD` 권한 모드를 사용할 것을 권장합니다.
> - 원장을 생성할 때 기본적으로 삭제 방지가 활성화됩니다. 이 기능은 사용자가 원장을 삭제하는 것을 방지하는 QLDB의 기능입니다. QLDB API 또는 AWS Command Line Interface(AWS CLI)를 사용하여 원장 생성 시 삭제 보호를 비활성화할 수 있습니다.
> - 선택적으로, 원장에 첨부할 태그를 지정할 수도 있습니다.

새 원장과의 연결을 확인하려면 2단계: 원장과의 연결을 테스트로 이동하십시오.

2단계: 원장과의 연결을 테스트

이 단계에서는 트랜잭션 데이터 API 엔드포인트를 사용하여 Amazon QLDB의 `vehicle-registration` 원장에 연결할 수 있는지 확인합니다.

원장과의 연결을 테스트하려면

1. vehicle-registration 원장에 대한 데이터 세션 연결을 생성하는 다음 프로그램(ConnectToLedger.java)을 검토합니다.

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.net.URI;
import java.net.URISyntaxException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.auth.credentials.AwsCredentialsProvider;
import software.amazon.awssdk.services.qldbsession.QldbSessionClient;
import software.amazon.awssdk.services.qldbsession.QldbSessionClientBuilder;
import software.amazon.qldb.QldbDriver;
```

```
import software.amazon.qldb.RetryPolicy;

/**
 * Connect to a session for a given ledger using default settings.
 * <p>
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class ConnectToLedger {
    public static final Logger log =
 LoggerFactory.getLogger(ConnectToLedger.class);
    public static AwsCredentialsProvider credentialsProvider;
    public static String endpoint = null;
    public static String ledgerName = Constants.LEDGER_NAME;
    public static String region = null;
    public static QldbDriver driver;

    private ConnectToLedger() {
    }

    /**
     * Create a pooled driver for creating sessions.
     *
     * @param retryAttempts How many times the transaction will be retried in
     * case of a retryable issue happens like Optimistic Concurrency Control
 exception,
     * server side failures or network issues.
     * @return The pooled driver for creating sessions.
     */
    public static QldbDriver createQldbDriver(int retryAttempts) {
        QldbSessionClientBuilder builder = getAmazonQldbSessionClientBuilder();
        return QldbDriver.builder()
                             .ledger(ledgerName)
                             .transactionRetryPolicy(RetryPolicy
                                   .builder()
                                   .maxRetries(retryAttempts)
                                   .build())
                             .sessionClientBuilder(builder)
                             .build();
    }

    /**
     * Create a pooled driver for creating sessions.
```

```
     *
     * @return The pooled driver for creating sessions.
     */
    public static QldbDriver createQldbDriver() {
        QldbSessionClientBuilder builder = getAmazonQldbSessionClientBuilder();
        return QldbDriver.builder()
            .ledger(ledgerName)
            .transactionRetryPolicy(RetryPolicy.builder()

.maxRetries(Constants.RETRY_LIMIT).build())
            .sessionClientBuilder(builder)
            .build();
    }

    /**
     * Creates a QldbSession builder that is passed to the QldbDriver to connect
to the Ledger.
     *
     * @return An instance of the AmazonQLDBSessionClientBuilder
     */
    public static QldbSessionClientBuilder getAmazonQldbSessionClientBuilder() {
        QldbSessionClientBuilder builder = QldbSessionClient.builder();
        if (null != endpoint && null != region) {
            try {
                builder.endpointOverride(new URI(endpoint));
            } catch (URISyntaxException e) {
                throw new IllegalArgumentException(e);
            }
        }
        if (null != credentialsProvider) {
            builder.credentialsProvider(credentialsProvider);
        }
        return builder;
    }

    /**
     * Create a pooled driver for creating sessions.
     *
     * @return The pooled driver for creating sessions.
     */
    public static QldbDriver getDriver() {
        if (driver == null) {
            driver = createQldbDriver();
        }
```

```
        return driver;
    }


    public static void main(final String... args) {
        Iterable<String> tables = ConnectToLedger.getDriver().getTableNames();
        log.info("Existing tables in the ledger:");
        for (String table : tables) {
            log.info("- {} ", table);
        }
    }
}
```

> **Note**
>
> - 원장에서 데이터 작업을 실행하려면 특정 원장에 연결할 `QldbDriver` 클래스 인스턴스를 생성해야 합니다. 이는 이전 단계에서 원장을 생성할 때 사용한 `AmazonQLDB` 클라이언트와는 다른 클라이언트 객체입니다. 이전 클라이언트는 Amazon QLDB API 참조에 나열된 관리 API 작업을 실행하는 데만 사용됩니다.
> - 먼저 `QldbDriver` 객체를 생성해야 합니다. 이 드라이버를 생성할 때 원장 이름을 지정해야 합니다.
>
>   그런 다음 이 드라이버의 `execute` 메서드를 사용하여 PartiQL 명령문을 실행할 수 있습니다.
> - 선택적으로 트랜잭션 예외에 대한 최대 재시도 횟수를 지정할 수 있습니다. 이 `execute` 메서드는 OCC(낙관적 동시성 제어) 충돌 및 기타 일반적인 일시적 예외를 이 구성 가능한 한도까지 자동으로 재시도합니다. 기본값은 4입니다.
>
>   한도에 도달한 후에도 트랜잭션이 여전히 실패하면 드라이버에서 예외가 발생합니다. 자세한 내용은 Amazon QLDB의 드라이버를 사용한 재시도 정책에 대한 이해 단원을 참조하십시오.

1.x

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
```

```
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.qldbsession.AmazonQLDBSessionClientBuilder;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.PooledQldbDriver;
import software.amazon.qldb.QldbDriver;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.exceptions.QldbClientException;

/**
 * Connect to a session for a given ledger using default settings.
 * <p>
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class ConnectToLedger {
    public static final Logger log =
 LoggerFactory.getLogger(ConnectToLedger.class);
```

```
    public static AWSCredentialsProvider credentialsProvider;
    public static String endpoint = null;
    public static String ledgerName = Constants.LEDGER_NAME;
    public static String region = null;
    private static PooledQldbDriver driver;

    private ConnectToLedger() {
    }

    /**
     * Create a pooled driver for creating sessions.
     *
     * @return The pooled driver for creating sessions.
     */
    public static PooledQldbDriver createQldbDriver() {
        AmazonQLDBSessionClientBuilder builder =
AmazonQLDBSessionClientBuilder.standard();
        if (null != endpoint && null != region) {
            builder.setEndpointConfiguration(new
AwsClientBuilder.EndpointConfiguration(endpoint, region));
        }
        if (null != credentialsProvider) {
            builder.setCredentials(credentialsProvider);
        }
        return PooledQldbDriver.builder()
                .withLedger(ledgerName)
                .withRetryLimit(Constants.RETRY_LIMIT)
                .withSessionClientBuilder(builder)
                .build();
    }

    /**
     * Create a pooled driver for creating sessions.
     *
     * @return The pooled driver for creating sessions.
     */
    public static PooledQldbDriver getDriver() {
        if (driver == null) {
            driver = createQldbDriver();
        }
        return driver;
    }

    /**
```

```
     * Connect to a ledger through a {@link QldbDriver}.
     *
     * @return {@link QldbSession}.
     */
    public static QldbSession createQldbSession() {
        return getDriver().getSession();
    }

    public static void main(final String... args) {
        try (QldbSession qldbSession = createQldbSession()) {
            log.info("Listing table names ");
            for (String tableName : qldbSession.getTableNames()) {
                log.info(tableName);
            }
        } catch (QldbClientException e) {
            log.error("Unable to create session.", e);
        }
    }
}
```

Note

- 원장에서 데이터 작업을 실행하려면 특정 원장에 연결할 `PooledQldbDriver` 또는 `QldbDriver` 클래스 인스턴스를 생성해야 합니다. 이는 이전 단계에서 원장을 생성할 때 사용한 `AmazonQLDB` 클라이언트와는 다른 클라이언트 객체입니다. 이전 클라이언트는 Amazon QLDB API 참조에 나열된 관리 API 작업을 실행하는 데만 사용됩니다.

  `QldbDriver`로 사용자 지정 세션 풀을 구현해야 하는 경우가 아니라면 `PooledQldbDriver`를 사용하는 것이 좋습니다. `PooledQldbDriver`의 기본 풀 크기는 세션 클라이언트가 허용하는 최대 열린 HTTP 연결 수입니다.

- 먼저 `PooledQldbDriver` 객체를 생성해야 합니다. 이 드라이버를 생성할 때 원장 이름을 지정해야 합니다.

  그런 다음 이 드라이버의 `execute` 메서드를 사용하여 PartiQL 명령문을 실행할 수 있습니다. 또는 이 풀링된 드라이버 객체에서 세션을 수동으로 만들고 세션의 `execute` 메서드를 사용할 수 있습니다. 세션은 원장과의 단일 연결을 나타냅니다.

- 선택적으로 트랜잭션 예외에 대한 최대 재시도 횟수를 지정할 수 있습니다. 이 `execute` 메서드는 OCC(낙관적 동시성 제어) 충돌 및 기타 일반적인 일시적 예외를 이 구성 가능한 한도까지 자동으로 재시도합니다. 기본값은 4입니다.

  한도에 도달한 후에도 트랜잭션이 여전히 실패하면 드라이버에서 예외가 발생합니다. 자세한 내용은 Amazon QLDB의 드라이버를 사용한 재시도 정책에 대한 이해 단원을 참조하십시오.

2. `ConnectToLedger.java` 프로그램을 컴파일하고 실행하여 `vehicle-registration` 원장에 대한 데이터 세션 연결을 테스트하세요.

AWS 리전 재정의

샘플 애플리케이션은 기본 AWS 리전의 QLDB에 연결되며, 사전 조건 단계 기본 AWS 보안 인증 정보 및 리전 설정에 설명된 대로 설정할 수 있습니다. 또한 QLDB 세션 클라이언트 빌더 속성을 수정하여 리전을 변경할 수 있습니다.

2.x

다음 코드 예제에서는 새 `QldbSessionClientBuilder` 객체를 인스턴스화합니다.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.qldbsession.QldbSessionClientBuilder;

// This client builder will default to US East (Ohio)
QldbSessionClientBuilder builder = QldbSessionClient.builder()
    .region(Region.US_EAST_2);
```

`region` 메서드를 사용하면 사용할 수 있는 어떤 리전의 QLDB에 대해서도 코드를 실행할 수 있습니다. 전체 목록은 AWS 일반 참조에서 Amazon QLDB 엔드포인트 및 할당량을 참조하십시오.

1.x

다음 코드 예제에서는 새 `AmazonQLDBSessionClientBuilder` 객체를 인스턴스화합니다.

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.qldbsession.AmazonQLDBSessionClientBuilder;

// This client builder will default to US East (Ohio)
AmazonQLDBSessionClientBuilder builder = AmazonQLDBSessionClientBuilder.standard()
    .withRegion(Regions.US_EAST_2);
```

withRegion 메서드를 사용하면 사용할 수 있는 어떤 리전의 QLDB에 대해서도 코드를 실행할 수 있습니다. 전체 목록은 AWS 일반 참조에서 Amazon QLDB 엔드포인트 및 할당량을 참조하십시오.

vehicle-registration 원장에 테이블을 생성하려면 3단계: 테이블, 인덱스 및 샘플 데이터 생성로 진행하십시오.

3단계: 테이블, 인덱스 및 샘플 데이터 생성

Amazon QLDB 원장이 활성화되고 연결을 수락하면 차량, 소유자 및 등록 정보에 대한 데이터 테이블 생성을 시작할 수 있습니다. 테이블과 인덱스를 생성한 후 데이터를 로드할 수 있습니다.

이 단계에서는 vehicle-registration 원장에 4개의 테이블을 생성합니다.

- VehicleRegistration
- Vehicle
- Person
- DriversLicense

또한 다음과 같은 인덱스를 생성합니다.

테이블 이름	필드
VehicleRegistration	VIN
VehicleRegistration	LicensePlateNumber
Vehicle	VIN
Person	GovId
DriversLicense	LicenseNumber
DriversLicense	PersonId

샘플 데이터를 삽입할 때는 먼저 Person 테이블에 문서를 삽입합니다. 그런 다음 각 Person 문서에서 시스템이 할당한 id를 사용하여 적절한 VehicleRegistration 및 DriversLicense 문서의 해당 필드를 채웁니다.

> **Tip**
>
> 가장 좋은 방법은 문서의 시스템 할당 `id`를 외래 키로 사용하는 것입니다. 고유 식별자(예: 차량의 VIN)로 사용되는 필드를 정의할 수 있지만 문서의 실제 고유 식별자는 `id`입니다. 이 필드는 문서의 메타데이터에 포함되며 커밋된 뷰(테이블의 시스템 정의 뷰)에서 쿼리할 수 있습니다.
>
> QLDB 뷰에 대한 자세한 내용은 핵심 개념 섹션을 참조하세요. 메타데이터에 대해 자세히 알아보려면 문서 메타데이터 쿼리 섹션을 참조하세요.

샘플 데이터를 설정하려면

1. 다음 `.java` 파일을 검토합니다. 이러한 모델 클래스는 `vehicle-registration` 테이블에 저장하는 문서를 나타냅니다. Amazon Ion 형식으로 또는 Amazon Ion 형식에서 직렬화할 수 있습니다.

   > **Note**
   >
   > Amazon QLDB 문서는 JSON의 상위 집합인 Ion 형식으로 저장됩니다. 따라서 FasterXML Jackson 라이브러리를 사용하여 데이터를 JSON으로 모델링할 수 있습니다.

   1. `DriversLicense.java`

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
of this
 * software and associated documentation files (the "Software"), to deal in the
Software
 * without restriction, including without limitation the rights to use, copy,
modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
```

```
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.annotation.JsonDeserialize;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import software.amazon.qldb.tutorial.model.streams.RevisionData;

import java.time.LocalDate;

/**
 * Represents a driver's license, serializable to (and from) Ion.
 */
public final class DriversLicense implements RevisionData {
    private final String personId;
    private final String licenseNumber;
    private final String licenseType;

    @JsonSerialize(using = IonLocalDateSerializer.class)
    @JsonDeserialize(using = IonLocalDateDeserializer.class)
    private final LocalDate validFromDate;

    @JsonSerialize(using = IonLocalDateSerializer.class)
    @JsonDeserialize(using = IonLocalDateDeserializer.class)
    private final LocalDate validToDate;

    @JsonCreator
    public DriversLicense(@JsonProperty("PersonId") final String personId,
                          @JsonProperty("LicenseNumber") final String
 licenseNumber,
                          @JsonProperty("LicenseType") final String licenseType,
                          @JsonProperty("ValidFromDate") final LocalDate
 validFromDate,
                          @JsonProperty("ValidToDate") final LocalDate
 validToDate) {
        this.personId = personId;
```

```
        this.licenseNumber = licenseNumber;
        this.licenseType = licenseType;
        this.validFromDate = validFromDate;
        this.validToDate = validToDate;
    }

    @JsonProperty("PersonId")
    public String getPersonId() {
        return personId;
    }

    @JsonProperty("LicenseNumber")
    public String getLicenseNumber() {
        return licenseNumber;
    }

    @JsonProperty("LicenseType")
    public String getLicenseType() {
        return licenseType;
    }

    @JsonProperty("ValidFromDate")
    public LocalDate getValidFromDate() {
        return  validFromDate;
    }

    @JsonProperty("ValidToDate")
    public LocalDate getValidToDate() {
        return  validToDate;
    }

    @Override
    public String toString() {
        return "DriversLicense{" +
                "personId='" + personId + '\'' +
                ", licenseNumber='" + licenseNumber + '\'' +
                ", licenseType='" + licenseType + '\'' +
                ", validFromDate=" + validFromDate +
                ", validToDate=" + validToDate +
                '}';
    }
}
```

2. Person.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import java.time.LocalDate;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.annotation.JsonDeserialize;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;

import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.model.streams.RevisionData;

/**
 * Represents a person, serializable to (and from) Ion.
 */
public final class Person implements RevisionData {
    private final String firstName;
```

```
    private final String lastName;

    @JsonSerialize(using = IonLocalDateSerializer.class)
    @JsonDeserialize(using = IonLocalDateDeserializer.class)
    private final LocalDate dob;
    private final String govId;
    private final String govIdType;
    private final String address;

    @JsonCreator
    public Person(@JsonProperty("FirstName") final String firstName,
                  @JsonProperty("LastName") final String lastName,
                  @JsonProperty("DOB") final LocalDate dob,
                  @JsonProperty("GovId") final String govId,
                  @JsonProperty("GovIdType") final String govIdType,
                  @JsonProperty("Address") final String address) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.dob = dob;
        this.govId = govId;
        this.govIdType = govIdType;
        this.address = address;
    }

    @JsonProperty("Address")
    public String getAddress() {
        return address;
    }

    @JsonProperty("DOB")
    public LocalDate getDob() {
        return dob;
    }

    @JsonProperty("FirstName")
    public String getFirstName() {
        return firstName;
    }

    @JsonProperty("LastName")
    public String getLastName() {
        return lastName;
    }
```

```
    @JsonProperty("GovId")
    public String getGovId() {
        return govId;
    }

    @JsonProperty("GovIdType")
    public String getGovIdType() {
        return govIdType;
    }

    /**
     * This returns the unique document ID given a specific government ID.
     *
     * @param txn
     *              A transaction executor object.
     * @param govId
     *              The government ID of a driver.
     * @return the unique document ID.
     */
    public static String getDocumentIdByGovId(final TransactionExecutor txn,
 final String govId) {
        return SampleData.getDocumentId(txn, Constants.PERSON_TABLE_NAME,
 "GovId", govId);
    }

    @Override
    public String toString() {
        return "Person{" +
                "firstName='" + firstName + '\'' +
                ", lastName='" + lastName + '\'' +
                ", dob=" + dob +
                ", govId='" + govId + '\'' +
                ", govIdType='" + govIdType + '\'' +
                ", address='" + address + '\'' +
                '}';
    }
}
```

3. VehicleRegistration.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
```

```
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.annotation.JsonDeserialize;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.model.streams.RevisionData;

import java.math.BigDecimal;
import java.time.LocalDate;

/**
 * Represents a vehicle registration, serializable to (and from) Ion.
 */
public final class VehicleRegistration implements RevisionData {

    private final String vin;
    private final String licensePlateNumber;
    private final String state;
    private final String city;
    private final BigDecimal pendingPenaltyTicketAmount;
```

```
    private final LocalDate validFromDate;
    private final LocalDate validToDate;
    private final Owners owners;

    @JsonCreator
    public VehicleRegistration(@JsonProperty("VIN") final String vin,
                               @JsonProperty("LicensePlateNumber") final String
licensePlateNumber,
                               @JsonProperty("State") final String state,
                               @JsonProperty("City") final String city,
                               @JsonProperty("PendingPenaltyTicketAmount") final
BigDecimal pendingPenaltyTicketAmount,
                               @JsonProperty("ValidFromDate") final LocalDate
validFromDate,
                               @JsonProperty("ValidToDate") final LocalDate
validToDate,
                               @JsonProperty("Owners") final Owners owners) {
        this.vin = vin;
        this.licensePlateNumber = licensePlateNumber;
        this.state = state;
        this.city = city;
        this.pendingPenaltyTicketAmount = pendingPenaltyTicketAmount;
        this.validFromDate = validFromDate;
        this.validToDate = validToDate;
        this.owners = owners;
    }

    @JsonProperty("City")
    public String getCity() {
        return city;
    }

    @JsonProperty("LicensePlateNumber")
    public String getLicensePlateNumber() {
        return licensePlateNumber;
    }

    @JsonProperty("Owners")
    public Owners getOwners() {
        return owners;
    }

    @JsonProperty("PendingPenaltyTicketAmount")
    public BigDecimal getPendingPenaltyTicketAmount() {
```

```
        return pendingPenaltyTicketAmount;
    }

    @JsonProperty("State")
    public String getState() {
        return state;
    }

    @JsonProperty("ValidFromDate")
    @JsonSerialize(using = IonLocalDateSerializer.class)
    @JsonDeserialize(using = IonLocalDateDeserializer.class)
    public LocalDate getValidFromDate() {
        return validFromDate;
    }

    @JsonProperty("ValidToDate")
    @JsonSerialize(using = IonLocalDateSerializer.class)
    @JsonDeserialize(using = IonLocalDateDeserializer.class)
    public LocalDate getValidToDate() {
        return validToDate;
    }

    @JsonProperty("VIN")
    public String getVin() {
        return vin;
    }

    /**
     * Returns the unique document ID of a vehicle given a specific VIN.
     *
     * @param txn
     *              A transaction executor object.
     * @param vin
     *              The VIN of a vehicle.
     * @return the unique document ID of the specified vehicle.
     */
    public static String getDocumentIdByVin(final TransactionExecutor txn, final
String vin) {
        return SampleData.getDocumentId(txn,
Constants.VEHICLE_REGISTRATION_TABLE_NAME, "VIN", vin);
    }

    @Override
    public String toString() {
```

```
        return "VehicleRegistration{" +
                "vin='" + vin + '\'' +
                ", licensePlateNumber='" + licensePlateNumber + '\'' +
                ", state='" + state + '\'' +
                ", city='" + city + '\'' +
                ", pendingPenaltyTicketAmount=" + pendingPenaltyTicketAmount +
                ", validFromDate=" + validFromDate +
                ", validToDate=" + validToDate +
                ", owners=" + owners +
                '}';
    }
}
```

4. Vehicle.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
```

```
import software.amazon.qldb.tutorial.model.streams.RevisionData;

/**
 * Represents a vehicle, serializable to (and from) Ion.
 */
public final class Vehicle implements RevisionData {
    private final String vin;
    private final String type;
    private final int year;
    private final String make;
    private final String model;
    private final String color;

    @JsonCreator
    public Vehicle(@JsonProperty("VIN") final String vin,
                   @JsonProperty("Type") final String type,
                   @JsonProperty("Year") final int year,
                   @JsonProperty("Make") final String make,
                   @JsonProperty("Model") final String model,
                   @JsonProperty("Color") final String color) {
        this.vin = vin;
        this.type = type;
        this.year = year;
        this.make = make;
        this.model = model;
        this.color = color;
    }

    @JsonProperty("Color")
    public String getColor() {
        return color;
    }

    @JsonProperty("Make")
    public String getMake() {
        return make;
    }

    @JsonProperty("Model")
    public String getModel() {
        return model;
    }

    @JsonProperty("Type")
```

```
    public String getType() {
        return type;
    }

    @JsonProperty("VIN")
    public String getVin() {
        return vin;
    }

    @JsonProperty("Year")
    public int getYear() {
        return year;
    }

    @Override
    public String toString() {
        return "Vehicle{" +
                "vin='" + vin + '\'' +
                ", type='" + type + '\'' +
                ", year=" + year +
                ", make='" + make + '\'' +
                ", model='" + model + '\'' +
                ", color='" + color + '\'' +
                '}';
    }
}
```

5. Owner.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
```

```
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.fasterxml.jackson.annotation.JsonProperty;

/**
 * Represents a vehicle owner, serializable to (and from) Ion.
 */
public final class Owner {
    private final String personId;

    public Owner(@JsonProperty("PersonId") final String personId) {
        this.personId = personId;
    }

    @JsonProperty("PersonId")
    public String getPersonId() {
        return personId;
    }

    @Override
    public String toString() {
        return "Owner{" +
                "personId='" + personId + '\'' +
                '}';
    }
}
```

6. Owners.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
```

```
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.fasterxml.jackson.annotation.JsonProperty;

import java.util.List;

/**
 * Represents a set of owners for a given vehicle, serializable to (and from)
 Ion.
 */
public final class Owners {
    private final Owner primaryOwner;
    private final List<Owner> secondaryOwners;

    public Owners(@JsonProperty("PrimaryOwner") final Owner primaryOwner,
                  @JsonProperty("SecondaryOwners") final List<Owner>
 secondaryOwners) {
        this.primaryOwner = primaryOwner;
        this.secondaryOwners = secondaryOwners;
    }

    @JsonProperty("PrimaryOwner")
    public Owner getPrimaryOwner() {
```

```
        return primaryOwner;
    }

    @JsonProperty("SecondaryOwners")
    public List<Owner> getSecondaryOwners() {
        return secondaryOwners;
    }

    @Override
    public String toString() {
        return "Owners{" +
                "primaryOwner=" + primaryOwner +
                ", secondaryOwners=" + secondaryOwners +
                '}';
    }
}
```

7. DmlResultDocument.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */
```

```
package software.amazon.qldb.tutorial.qldb;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

/**
 * Contains information about an individual document inserted or modified
 * as a result of DML.
 */
public class DmlResultDocument {

    private String documentId;

    @JsonCreator
    public DmlResultDocument(@JsonProperty("documentId") final String documentId)
 {
        this.documentId = documentId;
    }

    public String getDocumentId() {
        return documentId;
    }

    @Override
    public String toString() {
        return "DmlResultDocument{"
            + "documentId='" + documentId + '\''
            + '}';
    }
}
```

8. RevisionData.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
```

```
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model.streams;

/**
 * Allows modeling the content of all revisions as a generic revision data. Used
 * in the {@link Revision} and extended by domain models in {@link
 * software.amazon.qldb.tutorial.model} to make it easier to write the {@link
 * Revision.RevisionDataDeserializer} that must deserialize the {@link
 * Revision#data} from different domain models.
 */
public interface RevisionData { }
```

9. RevisionMetadata.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
```

```
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.amazon.ion.IonInt;
import com.amazon.ion.IonString;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonTimestamp;
import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import com.fasterxml.jackson.dataformat.ion.IonTimestampSerializers;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Date;
import java.util.Objects;

/**
 * Represents the metadata field of a QLDB Document
 */
public class RevisionMetadata {
    private static final Logger log =
 LoggerFactory.getLogger(RevisionMetadata.class);
    private final String id;
    private final long version;
    @JsonSerialize(using =
 IonTimestampSerializers.IonTimestampJavaDateSerializer.class)
    private final Date txTime;
    private final String txId;

    @JsonCreator
    public RevisionMetadata(@JsonProperty("id") final String id,
                            @JsonProperty("version") final long version,
                            @JsonProperty("txTime") final Date txTime,
                            @JsonProperty("txId") final String txId) {
        this.id = id;
```

```
        this.version = version;
        this.txTime = txTime;
        this.txId = txId;
    }

    /**
     * Gets the unique ID of a QLDB document.
     *
     * @return the document ID.
     */
    public String getId() {
        return id;
    }

    /**
     * Gets the version number of the document in the document's modification
history.
     * @return the version number.
     */
    public long getVersion() {
        return version;
    }

    /**
     * Gets the time during which the document was modified.
     *
     * @return the transaction time.
     */
    public Date getTxTime() {
        return txTime;
    }

    /**
     * Gets the transaction ID associated with this document.
     *
     * @return the transaction ID.
     */
    public String getTxId() {
        return txId;
    }

    public static RevisionMetadata fromIon(final IonStruct ionStruct) {
        if (ionStruct == null) {
            throw new IllegalArgumentException("Metadata cannot be null");
```

```
        }
        try {
            IonString id = (IonString) ionStruct.get("id");
            IonInt version = (IonInt) ionStruct.get("version");
            IonTimestamp txTime = (IonTimestamp) ionStruct.get("txTime");
            IonString txId = (IonString) ionStruct.get("txId");
            if (id == null || version == null || txTime == null || txId == null)
{
                throw new IllegalArgumentException("Document is missing required
fields");
            }
            return new RevisionMetadata(id.stringValue(), version.longValue(),
new Date(txTime.getMillis()), txId.stringValue());
        } catch (ClassCastException e) {
            log.error("Failed to parse ion document");
            throw new IllegalArgumentException("Document members are not of the
correct type", e);
        }
    }

    /**
     * Converts a {@link RevisionMetadata} object to a string.
     *
     * @return the string representation of the {@link QldbRevision} object.
     */
    @Override
    public String toString() {
        return "Metadata{"
                + "id='" + id + '\''
                + ", version=" + version
                + ", txTime=" + txTime
                + ", txId='" + txId
                + '\''
                + '}';
    }

    /**
     * Check whether two {@link RevisionMetadata} objects are equivalent.
     *
     * @return {@code true} if the two objects are equal, {@code false}
otherwise.
     */
    @Override
    public boolean equals(Object o) {
```

```
        if (this == o) { return true; }
        if (o == null || getClass() != o.getClass()) { return false; }
        RevisionMetadata metadata = (RevisionMetadata) o;
        return version == metadata.version
                && id.equals(metadata.id)
                && txTime.equals(metadata.txTime)
                && txId.equals(metadata.txId);
    }

    /**
     * Generate a hash code for the {@link RevisionMetadata} object.
     *
     * @return the hash code.
     */
    @Override
    public int hashCode() {
        // CHECKSTYLE:OFF - Disabling as we are generating a hashCode of multiple
 properties.
        return Objects.hash(id, version, txTime, txId);
        // CHECKSTYLE:ON
    }
}
```

10QldbRevision.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
```

```
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.amazon.ion.IonBlob;
import com.amazon.ion.IonStruct;
import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.Verifier;

import java.io.IOException;
import java.util.Arrays;
import java.util.Objects;

/**
 * Represents a QldbRevision including both user data and metadata.
 */
public final class QldbRevision {
    private static final Logger log =
 LoggerFactory.getLogger(QldbRevision.class);

    private final BlockAddress blockAddress;
    private final RevisionMetadata metadata;
    private final byte[] hash;
    private final byte[] dataHash;
    private final IonStruct data;

    @JsonCreator
    public QldbRevision(@JsonProperty("blockAddress") final BlockAddress
 blockAddress,
                        @JsonProperty("metadata") final RevisionMetadata
 metadata,
                        @JsonProperty("hash") final byte[] hash,
                        @JsonProperty("dataHash") final byte[] dataHash,
                        @JsonProperty("data") final IonStruct data) {
        this.blockAddress = blockAddress;
        this.metadata = metadata;
```

```
        this.hash = hash;
        this.dataHash = dataHash;
        this.data = data;
    }

    /**
     * Gets the unique ID of a QLDB document.
     *
     * @return the {@link BlockAddress} object.
     */
    public BlockAddress getBlockAddress() {
        return blockAddress;
    }

    /**
     * Gets the metadata of the revision.
     *
     * @return the {@link RevisionMetadata} object.
     */
    public RevisionMetadata getMetadata() {
        return metadata;
    }

    /**
     * Gets the SHA-256 hash value of the revision.
     * This is equivalent to the hash of the revision metadata and data.
     *
     * @return the byte array representing the hash.
     */
    public byte[] getHash() {
        return hash;
    }

    /**
     * Gets the SHA-256 hash value of the data portion of the revision.
     * This is only present if the revision is redacted.
     *
     * @return the byte array representing the hash.
     */
    public byte[] getDataHash() {
        return dataHash;
    }

    /**
```

```
     * Gets the revision data.
     *
     * @return the revision data.
     */
    public IonStruct getData() {
        return data;
    }

    /**
     * Returns true if the revision has been redacted.
     * @return a boolean value representing the redaction status
     * of this revision.
     */
    public Boolean isRedacted() {
        return dataHash != null;
    }

    /**
     * Constructs a new {@link QldbRevision} from an {@link IonStruct}.
     *
     * The specified {@link IonStruct} must include the following fields
     *
     * - blockAddress -- a {@link BlockAddress},
     * - metadata -- a {@link RevisionMetadata},
     * - hash -- the revision's hash calculated by QLDB,
     * - dataHash -- the user data's hash calculated by QLDB (only present if
revision is redacted),
     * - data -- an {@link IonStruct} containing user data in the document.
     *
     * If any of these fields are missing or are malformed, then throws {@link
IllegalArgumentException}.
     *
     * If the document hash calculated from the members of the specified {@link
IonStruct} does not match
     * the hash member of the {@link IonStruct} then throws {@link
IllegalArgumentException}.
     *
     * @param ionStruct
     *              The {@link IonStruct} that contains a {@link QldbRevision}
object.
     * @return the converted {@link QldbRevision} object.
     * @throws IOException if failed to parse parameter {@link IonStruct}.
     */
```

```
   public static QldbRevision fromIon(final IonStruct ionStruct) throws
IOException {
       try {
           BlockAddress blockAddress =
Constants.MAPPER.readValue(ionStruct.get("blockAddress"), BlockAddress.class);
           IonBlob revisionHash = (IonBlob) ionStruct.get("hash");
           IonStruct metadataStruct = (IonStruct) ionStruct.get("metadata");
           IonStruct data = ionStruct.get("data") == null ||
ionStruct.get("data").isNullValue() ?
               null : (IonStruct) ionStruct.get("data");
           IonBlob dataHash = ionStruct.get("dataHash") == null ||
ionStruct.get("dataHash").isNullValue() ?
               null : (IonBlob) ionStruct.get("dataHash");
           if (revisionHash == null || metadataStruct == null) {
               throw new IllegalArgumentException("Document is missing required
fields");
           }
           byte[] dataHashBytes = dataHash != null ? dataHash.getBytes() :
QldbIonUtils.hashIonValue(data);
           verifyRevisionHash(metadataStruct, dataHashBytes,
revisionHash.getBytes());
           RevisionMetadata metadata = RevisionMetadata.fromIon(metadataStruct);
           return new QldbRevision(
                   blockAddress,
                   metadata,
                   revisionHash.getBytes(),
                   dataHash != null ? dataHash.getBytes() : null,
                   data
           );
       } catch (ClassCastException e) {
           log.error("Failed to parse ion document");
           throw new IllegalArgumentException("Document members are not of the
correct type", e);
       }
   }

   /**
    * Converts a {@link QldbRevision} object to string.
    *
    * @return the string representation of the {@link QldbRevision} object.
    */
   @Override
   public String toString() {
       return "QldbRevision{" +
```

```
                "blockAddress=" + blockAddress +
                ", metadata=" + metadata +
                ", hash=" + Arrays.toString(hash) +
                ", dataHash=" + Arrays.toString(dataHash) +
                ", data=" + data +
                '}';
    }

    /**
     * Check whether two {@link QldbRevision} objects are equivalent.
     *
     * @return {@code true} if the two objects are equal, {@code false}
otherwise.
     */
    @Override
    public boolean equals(final Object o) {
        if (this == o) {
            return true;
        }
        if (!(o instanceof QldbRevision)) {
            return false;
        }
        final QldbRevision that = (QldbRevision) o;
        return Objects.equals(getBlockAddress(), that.getBlockAddress())
                && Objects.equals(getMetadata(), that.getMetadata())
                && Arrays.equals(getHash(), that.getHash())
                && Arrays.equals(getDataHash(), that.getDataHash())
                && Objects.equals(getData(), that.getData());
    }

    /**
     * Create a hash code for the {@link QldbRevision} object.
     *
     * @return the hash code.
     */
    @Override
    public int hashCode() {
        // CHECKSTYLE:OFF - Disabling as we are generating a hashCode of multiple
properties.
        int result = Objects.hash(blockAddress, metadata, data);
        // CHECKSTYLE:ON
        result = 31 * result + Arrays.hashCode(hash);
        return result;
    }
```

```
    /**
     * Throws an IllegalArgumentException if the hash of the revision data and
 metadata
     * does not match the hash provided by QLDB with the revision.
     */
    public void verifyRevisionHash() {
        // Certain internal-only system revisions only contain a hash which
 cannot be
        // further computed. However, these system hashes still participate to
 validate
        // the journal block. User revisions will always contain values for all
 fields
        // and can therefore have their hash computed.
        if (blockAddress == null && metadata == null && data == null && dataHash
 == null) {
            return;
        }

        try {
            IonStruct metadataIon = (IonStruct)
 Constants.MAPPER.writeValueAsIonValue(metadata);
            byte[] dataHashBytes = isRedacted() ? dataHash :
 QldbIonUtils.hashIonValue(data);
            verifyRevisionHash(metadataIon, dataHashBytes, hash);
        } catch (IOException e) {
            throw new IllegalArgumentException("Could not encode revision
 metadata to ion.", e);
        }
    }

    private static void verifyRevisionHash(IonStruct metadata, byte[] dataHash,
 byte[] expectedHash) {
        byte[] metadataHash = QldbIonUtils.hashIonValue(metadata);
        byte[] candidateHash = Verifier.dot(metadataHash, dataHash);
        if (!Arrays.equals(candidateHash, expectedHash)) {
            throw new IllegalArgumentException("Hash entry of QLDB revision and
 computed hash "
                    + "of QLDB revision do not match");
        }
    }
}
```

11IonLocalDateDeserializer.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.amazon.ion.Timestamp;
import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.databind.DeserializationContext;
import com.fasterxml.jackson.databind.JsonDeserializer;

import java.io.IOException;
import java.time.LocalDate;

/**
 * Deserializes [java.time.LocalDate] from Ion.
 */
public class IonLocalDateDeserializer extends JsonDeserializer<LocalDate> {

    @Override
    public LocalDate deserialize(JsonParser jp, DeserializationContext ctxt)
 throws IOException {
```

```
        return timestampToLocalDate((Timestamp) jp.getEmbeddedObject());
    }

    private LocalDate timestampToLocalDate(Timestamp timestamp) {
        return LocalDate.of(timestamp.getYear(), timestamp.getMonth(),
 timestamp.getDay());
    }
}
```

12IonLocalDateSerializer.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.amazon.ion.Timestamp;
import com.fasterxml.jackson.core.JsonGenerator;
import com.fasterxml.jackson.databind.SerializerProvider;
import com.fasterxml.jackson.databind.ser.std.StdScalarSerializer;
import com.fasterxml.jackson.dataformat.ion.IonGenerator;
```

```
import java.io.IOException;
import java.time.LocalDate;

/**
 * Serializes [java.time.LocalDate] to Ion.
 */
public class IonLocalDateSerializer extends StdScalarSerializer<LocalDate> {

    public IonLocalDateSerializer() {
        super(LocalDate.class);
    }

    @Override
    public void serialize(LocalDate date, JsonGenerator jsonGenerator,
 SerializerProvider serializerProvider) throws IOException {
        Timestamp timestamp = Timestamp.forDay(date.getYear(),
 date.getMonthValue(), date.getDayOfMonth());
        ((IonGenerator) jsonGenerator).writeValue(timestamp);
    }
}
```

2. vehicle-registration 테이블에 삽입한 샘플 데이터를 나타내는 다음 파일(SampleData.java)을 검토하십시오.

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
```

```
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.amazon.ion.IonString;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonValue;
import java.io.IOException;
import java.math.BigDecimal;
import java.text.ParseException;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.ConnectToLedger;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.qldb.DmlResultDocument;
import software.amazon.qldb.tutorial.qldb.QldbRevision;

/**
 * Sample domain objects for use throughout this tutorial.
 */
public final class SampleData {
    public static final DateTimeFormatter DATE_TIME_FORMAT =
 DateTimeFormatter.ofPattern("yyyy-MM-dd");

    public static final List<VehicleRegistration> REGISTRATIONS =
 Collections.unmodifiableList(Arrays.asList(
            new VehicleRegistration("1N4AL11D75C109151", "LEWISR261LL", "WA",
 "Seattle",
                    BigDecimal.valueOf(90.25), convertToLocalDate("2017-08-21"),
 convertToLocalDate("2020-05-11"),
                    new Owners(new Owner(null), Collections.emptyList())),
```

```
            new VehicleRegistration("KM8SRDHF6EU074761", "CA762X", "WA", "Kent",
                    BigDecimal.valueOf(130.75),
convertToLocalDate("2017-09-14"), convertToLocalDate("2020-06-25"),
                    new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("3HGGK5G53FM761765", "CD820Z", "WA",
"Everett",
                    BigDecimal.valueOf(442.30),
convertToLocalDate("2011-03-17"), convertToLocalDate("2021-03-24"),
                    new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("1HVBBAANXWH544237", "LS477D", "WA",
"Tacoma",
                    BigDecimal.valueOf(42.20), convertToLocalDate("2011-10-26"),
convertToLocalDate("2023-09-25"),
                    new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("1C4RJFAG0FC625797", "TH393F", "WA",
"Olympia",
                    BigDecimal.valueOf(30.45), convertToLocalDate("2013-09-02"),
convertToLocalDate("2024-03-19"),
                    new Owners(new Owner(null), Collections.emptyList()))
    ));

    public static final List<Vehicle> VEHICLES =
Collections.unmodifiableList(Arrays.asList(
            new Vehicle("1N4AL11D75C109151", "Sedan", 2011,  "Audi", "A5",
"Silver"),
            new Vehicle("KM8SRDHF6EU074761", "Sedan", 2015, "Tesla", "Model S",
"Blue"),
            new Vehicle("3HGGK5G53FM761765", "Motorcycle", 2011, "Ducati",
"Monster 1200", "Yellow"),
            new Vehicle("1HVBBAANXWH544237", "Semi", 2009, "Ford", "F 150",
"Black"),
            new Vehicle("1C4RJFAG0FC625797", "Sedan", 2019, "Mercedes", "CLK
350", "White")
    ));

    public static final List<Person> PEOPLE =
Collections.unmodifiableList(Arrays.asList(
            new Person("Raul", "Lewis", convertToLocalDate("1963-08-19"),
                    "LEWISR261LL", "Driver License",  "1719 University Street,
Seattle, WA, 98109"),
            new Person("Brent", "Logan", convertToLocalDate("1967-07-03"),
                    "LOGANB486CG", "Driver License", "43 Stockert Hollow Road,
Everett, WA, 98203"),
            new Person("Alexis", "Pena", convertToLocalDate("1974-02-10"),
```

```
                    "744 849 301", "SSN", "4058 Melrose Street, Spokane Valley,
WA, 99206"),
            new Person("Melvin", "Parker", convertToLocalDate("1976-05-22"),
                    "P626-168-229-765", "Passport", "4362 Ryder Avenue, Seattle,
WA, 98101"),
            new Person("Salvatore", "Spencer", convertToLocalDate("1997-11-15"),
                    "S152-780-97-415-0", "Passport", "4450 Honeysuckle Lane,
Seattle, WA, 98101")
    ));

    public static final List<DriversLicense> LICENSES =
Collections.unmodifiableList(Arrays.asList(
            new DriversLicense(null, "LEWISR261LL", "Learner",
                    convertToLocalDate("2016-12-20"),
convertToLocalDate("2020-11-15")),
            new DriversLicense(null, "LOGANB486CG", "Probationary",
                    convertToLocalDate("2016-04-06"),
convertToLocalDate("2020-11-15")),
            new DriversLicense(null, "744 849 301", "Full",
                    convertToLocalDate("2017-12-06"),
convertToLocalDate("2022-10-15")),
            new DriversLicense(null, "P626-168-229-765", "Learner",
                    convertToLocalDate("2017-08-16"),
convertToLocalDate("2021-11-15")),
            new DriversLicense(null, "S152-780-97-415-0", "Probationary",
                    convertToLocalDate("2015-08-15"),
convertToLocalDate("2021-08-21"))
    ));

    private SampleData() { }

    /**
     * Converts a date string with the format 'yyyy-MM-dd' into a {@link
java.util.Date} object.
     *
     * @param date
     *              The date string to convert.
     * @return {@link java.time.LocalDate} or null if there is a {@link
ParseException}
     */
    public static synchronized LocalDate convertToLocalDate(String date) {
        return LocalDate.parse(date, DATE_TIME_FORMAT);
    }
```

```
    /**
     * Convert the result set into a list of IonValues.
     *
     * @param result
     *              The result set to convert.
     * @return a list of IonValues.
     */
    public static List<IonValue> toIonValues(Result result) {
        final List<IonValue> valueList = new ArrayList<>();
        result.iterator().forEachRemaining(valueList::add);
        return valueList;
    }

    /**
     * Get the document ID of a particular document.
     *
     * @param txn
     *              A transaction executor object.
     * @param tableName
     *              Name of the table containing the document.
     * @param identifier
     *              The identifier used to narrow down the search.
     * @param value
     *              Value of the identifier.
     * @return the list of document IDs in the result set.
     */
    public static String getDocumentId(final TransactionExecutor txn, final
String tableName,
                                       final String identifier, final String
value) {
        try {
            final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(value));
            final String query = String.format("SELECT metadata.id FROM
_ql_committed_%s AS p WHERE p.data.%s = ?",
                    tableName, identifier);
            Result result = txn.execute(query, parameters);
            if (result.isEmpty()) {
                throw new IllegalStateException("Unable to retrieve document ID
using " + value);
            }
            return getStringValueOfStructField((IonStruct)
result.iterator().next(), "id");
        } catch (IOException ioe) {
```

```
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Get the document by ID.
     *
     * @param tableName
     *              Name of the table to insert documents into.
     * @param documentId
     *              The unique ID of a document in the Person table.
     * @return a {@link QldbRevision} object.
     * @throws IllegalStateException if failed to convert parameter into {@link
IonValue}.
     */
    public static QldbRevision getDocumentById(String tableName, String
documentId) {
        try {
            final IonValue ionValue =
Constants.MAPPER.writeValueAsIonValue(documentId);
            Result result = ConnectToLedger.getDriver().execute(txn -> {
                return txn.execute("SELECT c.* FROM _ql_committed_" + tableName
+ " AS c BY docId "
                                   + "WHERE docId = ?", ionValue);
            });
            if (result.isEmpty()) {
                throw new IllegalStateException("Unable to retrieve document by
id " + documentId + " in table " + tableName);
            }
            return Constants.MAPPER.readValue(result.iterator().next(),
QldbRevision.class);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Return a list of modified document IDs as strings from a DML {@link
Result}.
     *
     * @param result
     *              The result set from a DML operation.
     * @return the list of document IDs modified by the operation.
     */
```

```
   public static List<String> getDocumentIdsFromDmlResult(final Result result)
{
       final List<String> strings = new ArrayList<>();
       result.iterator().forEachRemaining(row ->
strings.add(getDocumentIdFromDmlResultDocument(row)));
       return strings;
   }

   /**
    * Convert the given DML result row's document ID to string.
    *
    * @param dmlResultDocument
    *              The {@link IonValue} representing the results of a DML
operation.
    * @return a string of document ID.
    */
   public static String getDocumentIdFromDmlResultDocument(final IonValue
dmlResultDocument) {
       try {
           DmlResultDocument result =
Constants.MAPPER.readValue(dmlResultDocument, DmlResultDocument.class);
           return result.getDocumentId();
       } catch (IOException ioe) {
           throw new IllegalStateException(ioe);
       }
   }

   /**
    * Get the String value of a given {@link IonStruct} field name.
    * @param struct the {@link IonStruct} from which to get the value.
    * @param fieldName the name of the field from which to get the value.
    * @return the String value of the field within the given {@link IonStruct}.
    */
   public static String getStringValueOfStructField(final IonStruct struct,
final String fieldName) {
       return ((IonString) struct.get(fieldName)).stringValue();
   }

   /**
    * Return a copy of the given driver's license with updated person Id.
    *
    * @param oldLicense
    *              The old driver's license to update.
    * @param personId
```

```
     *              The PersonId of the driver.
     * @return the updated {@link DriversLicense}.
     */
    public static DriversLicense updatePersonIdDriversLicense(final
 DriversLicense oldLicense, final String personId) {
        return new DriversLicense(personId, oldLicense.getLicenseNumber(),
 oldLicense.getLicenseType(),
                oldLicense.getValidFromDate(), oldLicense.getValidToDate());
    }

    /**
     * Return a copy of the given vehicle registration with updated person Id.
     *
     * @param oldRegistration
     *              The old vehicle registration to update.
     * @param personId
     *              The PersonId of the driver.
     * @return the updated {@link VehicleRegistration}.
     */
    public static VehicleRegistration updateOwnerVehicleRegistration(final
 VehicleRegistration oldRegistration,
                                                                     final
 String personId) {
        return new VehicleRegistration(oldRegistration.getVin(),
 oldRegistration.getLicensePlateNumber(),
                oldRegistration.getState(), oldRegistration.getCity(),
 oldRegistration.getPendingPenaltyTicketAmount(),
                oldRegistration.getValidFromDate(),
 oldRegistration.getValidToDate(),
                new Owners(new Owner(personId), Collections.emptyList()));
    }
}
```

1.x

> **Important**
>
> Amazon Ion 패키지의 경우 애플리케이션에서 네임스페이스 `com.amazon.ion`을 사용해야 합니다. AWS SDK for Java은 네임스페이스 `software.amazon.ion` 아래의 다른 Ion 패키지에 따라 다르지만 이 패키지는 QLDB 드라이버와 호환되지 않는 레거시 패키지입니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.model;

import com.amazon.ion.IonString;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonValue;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.qldb.DmlResultDocument;
import software.amazon.qldb.tutorial.qldb.QldbRevision;

import java.io.IOException;

import java.math.BigDecimal;
import java.text.ParseException;
```

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

/**
 * Sample domain objects for use throughout this tutorial.
 */
public final class SampleData {
    public static final DateTimeFormatter DATE_TIME_FORMAT =
 DateTimeFormatter.ofPattern("yyyy-MM-dd");

    public static final List<VehicleRegistration> REGISTRATIONS =
 Collections.unmodifiableList(Arrays.asList(
            new VehicleRegistration("1N4AL11D75C109151", "LEWISR261LL", "WA",
 "Seattle",
                    BigDecimal.valueOf(90.25), convertToLocalDate("2017-08-21"),
 convertToLocalDate("2020-05-11"),
                    new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("KM8SRDHF6EU074761", "CA762X", "WA", "Kent",
                    BigDecimal.valueOf(130.75),
 convertToLocalDate("2017-09-14"), convertToLocalDate("2020-06-25"),
                    new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("3HGGK5G53FM761765", "CD820Z", "WA",
 "Everett",
                    BigDecimal.valueOf(442.30),
 convertToLocalDate("2011-03-17"), convertToLocalDate("2021-03-24"),
                    new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("1HVBBAANXWH544237", "LS477D", "WA",
 "Tacoma",
                    BigDecimal.valueOf(42.20), convertToLocalDate("2011-10-26"),
 convertToLocalDate("2023-09-25"),
                    new Owners(new Owner(null), Collections.emptyList())),
            new VehicleRegistration("1C4RJFAG0FC625797", "TH393F", "WA",
 "Olympia",
                    BigDecimal.valueOf(30.45), convertToLocalDate("2013-09-02"),
 convertToLocalDate("2024-03-19"),
                    new Owners(new Owner(null), Collections.emptyList()))
    ));

    public static final List<Vehicle> VEHICLES =
 Collections.unmodifiableList(Arrays.asList(
```

```
            new Vehicle("1N4AL11D75C109151", "Sedan", 2011,  "Audi", "A5",
"Silver"),
            new Vehicle("KM8SRDHF6EU074761", "Sedan", 2015, "Tesla", "Model S",
"Blue"),
            new Vehicle("3HGGK5G53FM761765", "Motorcycle", 2011, "Ducati",
"Monster 1200", "Yellow"),
            new Vehicle("1HVBBAANXWH544237", "Semi", 2009, "Ford", "F 150",
"Black"),
            new Vehicle("1C4RJFAG0FC625797", "Sedan", 2019, "Mercedes", "CLK
350", "White")
    ));

    public static final List<Person> PEOPLE =
Collections.unmodifiableList(Arrays.asList(
            new Person("Raul", "Lewis", convertToLocalDate("1963-08-19"),
                    "LEWISR261LL", "Driver License",  "1719 University Street,
Seattle, WA, 98109"),
            new Person("Brent", "Logan", convertToLocalDate("1967-07-03"),
                    "LOGANB486CG", "Driver License", "43 Stockert Hollow Road,
Everett, WA, 98203"),
            new Person("Alexis", "Pena", convertToLocalDate("1974-02-10"),
                    "744 849 301", "SSN", "4058 Melrose Street, Spokane Valley,
WA, 99206"),
            new Person("Melvin", "Parker", convertToLocalDate("1976-05-22"),
                    "P626-168-229-765", "Passport", "4362 Ryder Avenue, Seattle,
WA, 98101"),
            new Person("Salvatore", "Spencer", convertToLocalDate("1997-11-15"),
                    "S152-780-97-415-0", "Passport", "4450 Honeysuckle Lane,
Seattle, WA, 98101")
    ));

    public static final List<DriversLicense> LICENSES =
Collections.unmodifiableList(Arrays.asList(
            new DriversLicense(null, "LEWISR261LL", "Learner",
                    convertToLocalDate("2016-12-20"),
convertToLocalDate("2020-11-15")),
            new DriversLicense(null, "LOGANB486CG", "Probationary",
                    convertToLocalDate("2016-04-06"),
convertToLocalDate("2020-11-15")),
            new DriversLicense(null, "744 849 301", "Full",
                    convertToLocalDate("2017-12-06"),
convertToLocalDate("2022-10-15")),
            new DriversLicense(null, "P626-168-229-765", "Learner",
```

```
                    convertToLocalDate("2017-08-16"),
convertToLocalDate("2021-11-15")),
            new DriversLicense(null, "S152-780-97-415-0", "Probationary",
                    convertToLocalDate("2015-08-15"),
convertToLocalDate("2021-08-21"))
    ));

    private SampleData() { }

    /**
     * Converts a date string with the format 'yyyy-MM-dd' into a {@link
java.util.Date} object.
     *
     * @param date
     *              The date string to convert.
     * @return {@link LocalDate} or null if there is a {@link ParseException}
     */
    public static synchronized LocalDate convertToLocalDate(String date) {
        return LocalDate.parse(date, DATE_TIME_FORMAT);
    }

    /**
     * Convert the result set into a list of IonValues.
     *
     * @param result
     *              The result set to convert.
     * @return a list of IonValues.
     */
    public static List<IonValue> toIonValues(Result result) {
        final List<IonValue> valueList = new ArrayList<>();
        result.iterator().forEachRemaining(valueList::add);
        return valueList;
    }

    /**
     * Get the document ID of a particular document.
     *
     * @param txn
     *              A transaction executor object.
     * @param tableName
     *              Name of the table containing the document.
     * @param identifier
     *              The identifier used to narrow down the search.
     * @param value
```

```
     *              Value of the identifier.
     * @return the list of document IDs in the result set.
     */
    public static String getDocumentId(final TransactionExecutor txn, final
String tableName,
                                       final String identifier, final String
value) {
        try {
            final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(value));
            final String query = String.format("SELECT metadata.id FROM
_ql_committed_%s AS p WHERE p.data.%s = ?",
                    tableName, identifier);
            Result result = txn.execute(query, parameters);
            if (result.isEmpty()) {
                throw new IllegalStateException("Unable to retrieve document ID
using " + value);
            }
            return getStringValueOfStructField((IonStruct)
result.iterator().next(), "id");
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Get the document by ID.
     *
     * @param qldbSession
     *              A QLDB session.
     * @param tableName
     *              Name of the table to insert documents into.
     * @param documentId
     *              The unique ID of a document in the Person table.
     * @return a {@link QldbRevision} object.
     * @throws IllegalStateException if failed to convert parameter into {@link
IonValue}.
     */
    public static QldbRevision getDocumentById(QldbSession qldbSession, String
tableName, String documentId) {
        try {
            final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(documentId));
```

```
            final String query = String.format("SELECT c.* FROM _ql_committed_%s
AS c BY docId WHERE docId = ?", tableName);
            Result result = qldbSession.execute(query, parameters);
            if (result.isEmpty()) {
                throw new IllegalStateException("Unable to retrieve document by
id " + documentId + " in table " + tableName);
            }
            return Constants.MAPPER.readValue(result.iterator().next(),
QldbRevision.class);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Return a list of modified document IDs as strings from a DML {@link
Result}.
     *
     * @param result
     *              The result set from a DML operation.
     * @return the list of document IDs modified by the operation.
     */
    public static List<String> getDocumentIdsFromDmlResult(final Result result)
{
        final List<String> strings = new ArrayList<>();
        result.iterator().forEachRemaining(row ->
strings.add(getDocumentIdFromDmlResultDocument(row)));
        return strings;
    }

    /**
     * Convert the given DML result row's document ID to string.
     *
     * @param dmlResultDocument
     *              The {@link IonValue} representing the results of a DML
operation.
     * @return a string of document ID.
     */
    public static String getDocumentIdFromDmlResultDocument(final IonValue
dmlResultDocument) {
        try {
            DmlResultDocument result =
Constants.MAPPER.readValue(dmlResultDocument, DmlResultDocument.class);
            return result.getDocumentId();
```

```
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Get the String value of a given {@link IonStruct} field name.
     * @param struct the {@link IonStruct} from which to get the value.
     * @param fieldName the name of the field from which to get the value.
     * @return the String value of the field within the given {@link IonStruct}.
     */
    public static String getStringValueOfStructField(final IonStruct struct,
final String fieldName) {
        return ((IonString) struct.get(fieldName)).stringValue();
    }

    /**
     * Return a copy of the given driver's license with updated person Id.
     *
     * @param oldLicense
     *              The old driver's license to update.
     * @param personId
     *              The PersonId of the driver.
     * @return the updated {@link DriversLicense}.
     */
    public static DriversLicense updatePersonIdDriversLicense(final
DriversLicense oldLicense, final String personId) {
        return new DriversLicense(personId, oldLicense.getLicenseNumber(),
oldLicense.getLicenseType(),
                oldLicense.getValidFromDate(), oldLicense.getValidToDate());
    }

    /**
     * Return a copy of the given vehicle registration with updated person Id.
     *
     * @param oldRegistration
     *              The old vehicle registration to update.
     * @param personId
     *              The PersonId of the driver.
     * @return the updated {@link VehicleRegistration}.
     */
    public static VehicleRegistration updateOwnerVehicleRegistration(final
VehicleRegistration oldRegistration,
```

```
                                                                    final
 String personId) {
        return new VehicleRegistration(oldRegistration.getVin(),
 oldRegistration.getLicensePlateNumber(),
                oldRegistration.getState(), oldRegistration.getCity(),
 oldRegistration.getPendingPenaltyTicketAmount(),
                oldRegistration.getValidFromDate(),
 oldRegistration.getValidToDate(),
                new Owners(new Owner(personId), Collections.emptyList()));
    }
}
```

> **Note**
>
> - 이 클래스는 Ion 라이브러리를 사용하여 데이터를 Ion 형식으로 또는 Ion 형식에서 변환하는 도우미 메서드를 제공합니다.
> - 이 `getDocumentId` 메서드는 접두사 `_ql_committed_`가 있는 테이블에 대한 쿼리를 실행합니다. 이는 테이블의 커밋된 보기를 쿼리하려는 것을 나타내는 예약된 접두사입니다. 이 뷰에서는 데이터가 `data` 필드에 중첩되고 메타데이터는 `metadata` 필드에 중첩됩니다.

3. 다음 프로그램(`CreateTable.java`)을 컴파일하고 실행하여 앞서 언급한 테이블을 생성합니다.

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
```

```
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Create tables in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class CreateTable {
    public static final Logger log = LoggerFactory.getLogger(CreateTable.class);

    private CreateTable() { }

    /**
     * Registrations, vehicles, owners, and licenses tables being created in a
 single transaction.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *              Name of the table to be created.
     * @return the number of tables created.
     */
```

```
    public static int createTable(final TransactionExecutor txn, final String
 tableName) {
        log.info("Creating the '{}' table...", tableName);
        final String createTable = String.format("CREATE TABLE %s", tableName);
        final Result result = txn.execute(createTable);
        log.info("{} table created successfully.", tableName);
        return SampleData.toIonValues(result).size();
    }

    public static void main(final String... args) {
        ConnectToLedger.getDriver().execute(txn -> {
            createTable(txn, Constants.DRIVERS_LICENSE_TABLE_NAME);
            createTable(txn, Constants.PERSON_TABLE_NAME);
            createTable(txn, Constants.VEHICLE_TABLE_NAME);
            createTable(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME);
        });
    }
}
```

1.x

```
/*
 * Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
```

```
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Create tables in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class CreateTable {
    public static final Logger log = LoggerFactory.getLogger(CreateTable.class);

    private CreateTable() { }

    /**
     * Registrations, vehicles, owners, and licenses tables being created in a
 single transaction.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *              Name of the table to be created.
     * @return the number of tables created.
     */
    public static int createTable(final TransactionExecutor txn, final String
 tableName) {
        log.info("Creating the '{}' table...", tableName);
        final String createTable = String.format("CREATE TABLE %s", tableName);
        final Result result = txn.execute(createTable);
        log.info("{} table created successfully.", tableName);
        return SampleData.toIonValues(result).size();
    }
```

```
    public static void main(final String... args) {
        ConnectToLedger.getDriver().execute(txn -> {
            createTable(txn, Constants.DRIVERS_LICENSE_TABLE_NAME);
            createTable(txn, Constants.PERSON_TABLE_NAME);
            createTable(txn, Constants.VEHICLE_TABLE_NAME);
            createTable(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME);
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
    }
}
```

> **Note**
>
> 이 프로그램은 `TransactionExecutor` Lambda를 `execute` 메서드에 전달하는 방법을 보여줍니다. 이 예제에서는 Lambda 표현식을 사용하여 단일 트랜잭션에서 여러 `CREATE TABLE` PartiQL 명령문을 실행합니다.
> `execute` 메서드는 암시적으로 트랜잭션을 시작하고 Lambda에서 모든 문을 실행한 다음 트랜잭션을 자동 커밋합니다.

4. 앞에서 설명한 대로 다음 프로그램(`CreateIndex.java`)을 컴파일하고 실행하여 테이블에 인덱스를 생성합니다.

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
```

```
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;


import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Create indexes on tables in a particular ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class CreateIndex {
    public static final Logger log = LoggerFactory.getLogger(CreateIndex.class);

    private CreateIndex() { }

    /**
     * In this example, create indexes for registrations and vehicles tables.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *              Name of the table to be created.
     * @param indexAttribute
     *              The index attribute to use.
     * @return the number of tables created.
     */
    public static int createIndex(final TransactionExecutor txn, final String
 tableName, final String indexAttribute) {
```

```
        log.info("Creating an index on {}...", indexAttribute);
        final String createIndex = String.format("CREATE INDEX ON %s (%s)",
 tableName, indexAttribute);
        final Result r = txn.execute(createIndex);
        return SampleData.toIonValues(r).size();
    }

    public static void main(final String... args) {
        ConnectToLedger.getDriver().execute(txn -> {
            createIndex(txn, Constants.PERSON_TABLE_NAME,
 Constants.PERSON_GOV_ID_INDEX_NAME);
            createIndex(txn, Constants.VEHICLE_TABLE_NAME,
 Constants.VIN_INDEX_NAME);
            createIndex(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
 Constants.DRIVER_LICENSE_NUMBER_INDEX_NAME);
            createIndex(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
 Constants.DRIVER_LICENSE_PERSONID_INDEX_NAME);
            createIndex(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
 Constants.VIN_INDEX_NAME);
            createIndex(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,

 Constants.VEHICLE_REGISTRATION_LICENSE_PLATE_NUMBER_INDEX_NAME);
        });
        log.info("Indexes created successfully!");
    }
}
```

1.x

```
/*
 * Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
```

```
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Create indexes on tables in a particular ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class CreateIndex {
    public static final Logger log = LoggerFactory.getLogger(CreateIndex.class);

    private CreateIndex() { }

    /**
     * In this example, create indexes for registrations and vehicles tables.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *              Name of the table to be created.
     * @param indexAttribute
     *              The index attribute to use.
     * @return the number of tables created.
     */
```

```
    public static int createIndex(final TransactionExecutor txn, final String
 tableName, final String indexAttribute) {
        log.info("Creating an index on {}...", indexAttribute);
        final String createIndex = String.format("CREATE INDEX ON %s (%s)",
 tableName, indexAttribute);
        final Result r = txn.execute(createIndex);
        return SampleData.toIonValues(r).size();
    }

    public static void main(final String... args) {
        ConnectToLedger.getDriver().execute(txn -> {
            createIndex(txn, Constants.PERSON_TABLE_NAME,
 Constants.PERSON_GOV_ID_INDEX_NAME);
            createIndex(txn, Constants.VEHICLE_TABLE_NAME,
 Constants.VIN_INDEX_NAME);
            createIndex(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
 Constants.DRIVER_LICENSE_NUMBER_INDEX_NAME);
            createIndex(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
 Constants.DRIVER_LICENSE_PERSONID_INDEX_NAME);
            createIndex(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
 Constants.VIN_INDEX_NAME);
            createIndex(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,

 Constants.VEHICLE_REGISTRATION_LICENSE_PLATE_NUMBER_INDEX_NAME);
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
        log.info("Indexes created successfully!");
    }
}
```

5. 다음 프로그램(InsertDocument.java)을 컴파일하고 실행하여 샘플 데이터를 테이블에 삽입합니다.

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
```

```
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;

import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.DriversLicense;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.model.VehicleRegistration;

/**
 * Insert documents into a table in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class InsertDocument {
```

```
   public static final Logger log =
LoggerFactory.getLogger(InsertDocument.class);

   private InsertDocument() { }

   /**
    * Insert the given list of documents into the specified table and return
the document IDs of the inserted documents.
    *
    * @param txn
    *              The {@link TransactionExecutor} for lambda execute.
    * @param tableName
    *              Name of the table to insert documents into.
    * @param documents
    *              List of documents to insert into the specified table.
    * @return a list of document IDs.
    * @throws IllegalStateException if failed to convert documents into an
{@link IonValue}.
    */
   public static List<String> insertDocuments(final TransactionExecutor txn,
final String tableName,
                                              final List documents) {
       log.info("Inserting some documents in the {} table...", tableName);
       try {
           final String query = String.format("INSERT INTO %s ?", tableName);
           final IonValue ionDocuments =
Constants.MAPPER.writeValueAsIonValue(documents);

           return SampleData.getDocumentIdsFromDmlResult(txn.execute(query,
ionDocuments));
       } catch (IOException ioe) {
           throw new IllegalStateException(ioe);
       }
   }

   /**
    * Update PersonIds in driver's licenses and in vehicle registrations using
document IDs.
    *
    * @param documentIds
    *              List of document IDs representing the PersonIds in
DriversLicense and PrimaryOwners in VehicleRegistration.
    * @param licenses
    *              List of driver's licenses to update.
```

```
     * @param registrations
     *              List of registrations to update.
     */
    public static void updatePersonId(final List<String> documentIds, final
 List<DriversLicense> licenses,
                                      final List<VehicleRegistration>
 registrations) {
        for (int i = 0; i < documentIds.size(); ++i) {
            DriversLicense license = SampleData.LICENSES.get(i);
            VehicleRegistration registration = SampleData.REGISTRATIONS.get(i);
            licenses.add(SampleData.updatePersonIdDriversLicense(license,
 documentIds.get(i)));

 registrations.add(SampleData.updateOwnerVehicleRegistration(registration,
 documentIds.get(i)));
        }
    }

    public static void main(final String... args) {
        final List<DriversLicense> newDriversLicenses = new ArrayList<>();
        final List<VehicleRegistration> newVehicleRegistrations = new
 ArrayList<>();
        ConnectToLedger.getDriver().execute(txn -> {
            List<String> documentIds = insertDocuments(txn,
 Constants.PERSON_TABLE_NAME, SampleData.PEOPLE);
            updatePersonId(documentIds, newDriversLicenses,
 newVehicleRegistrations);
            insertDocuments(txn, Constants.VEHICLE_TABLE_NAME,
 SampleData.VEHICLES);
            insertDocuments(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
                    Collections.unmodifiableList(newVehicleRegistrations));
            insertDocuments(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
                    Collections.unmodifiableList(newDriversLicenses));
        });
        log.info("Documents inserted successfully!");
    }
}
```

1.x

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
```

```
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonValue;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.DriversLicense;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.model.VehicleRegistration;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * Insert documents into a table in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
```

```
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class InsertDocument {
    public static final Logger log =
 LoggerFactory.getLogger(InsertDocument.class);

    private InsertDocument() { }

    /**
     * Insert the given list of documents into the specified table and return
 the document IDs of the inserted documents.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *              Name of the table to insert documents into.
     * @param documents
     *              List of documents to insert into the specified table.
     * @return a list of document IDs.
     * @throws IllegalStateException if failed to convert documents into an
 {@link IonValue}.
     */
    public static List<String> insertDocuments(final TransactionExecutor txn,
 final String tableName,
                                               final List documents) {
        log.info("Inserting some documents in the {} table...", tableName);
        try {
            final String statement = String.format("INSERT INTO %s ?",
 tableName);
            final IonValue ionDocuments =
 Constants.MAPPER.writeValueAsIonValue(documents);
            final List<IonValue> parameters =
 Collections.singletonList(ionDocuments);
            return SampleData.getDocumentIdsFromDmlResult(txn.execute(statement,
 parameters));
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Update PersonIds in driver's licenses and in vehicle registrations using
 document IDs.
```

```
     *
     * @param documentIds
     *              List of document IDs representing the PersonIds in
 DriversLicense and PrimaryOwners in VehicleRegistration.
     * @param licenses
     *              List of driver's licenses to update.
     * @param registrations
     *              List of registrations to update.
     */
    public static void updatePersonId(final List<String> documentIds, final
 List<DriversLicense> licenses,
                                      final List<VehicleRegistration>
 registrations) {
        for (int i = 0; i < documentIds.size(); ++i) {
            DriversLicense license = SampleData.LICENSES.get(i);
            VehicleRegistration registration = SampleData.REGISTRATIONS.get(i);
            licenses.add(SampleData.updatePersonIdDriversLicense(license,
 documentIds.get(i)));

 registrations.add(SampleData.updateOwnerVehicleRegistration(registration,
 documentIds.get(i)));
        }
    }

    public static void main(final String... args) {
        final List<DriversLicense> newDriversLicenses = new ArrayList<>();
        final List<VehicleRegistration> newVehicleRegistrations = new
 ArrayList<>();
        ConnectToLedger.getDriver().execute(txn -> {
            List<String> documentIds = insertDocuments(txn,
 Constants.PERSON_TABLE_NAME, SampleData.PEOPLE);
            updatePersonId(documentIds, newDriversLicenses,
 newVehicleRegistrations);
            insertDocuments(txn, Constants.VEHICLE_TABLE_NAME,
 SampleData.VEHICLES);
            insertDocuments(txn, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
                    Collections.unmodifiableList(newVehicleRegistrations));
            insertDocuments(txn, Constants.DRIVERS_LICENSE_TABLE_NAME,
                    Collections.unmodifiableList(newDriversLicenses));
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
        log.info("Documents inserted successfully!");
    }
}
```

> **Note**
>
> - 이 프로그램은 파라미터화된 값을 사용하여 `execute` 메서드를 호출하는 방법을 보여 줍니다. 실행하려는 PartiQL 문 외에도 `IonValue` 유형의 데이터 파라미터를 전달할 수 있습니다. 명령문 문자열에서 물음표(?)를 변수 자리 표시자로 사용하세요.
> - `INSERT` 문이 성공하면 삽입된 각 문서의 `id`이 반환됩니다.

다음으로, `SELECT` 문을 사용하여 `vehicle-registration` 원장의 테이블에서 데이터를 읽을 수 있습니다. 4단계: 원장에서 테이블 쿼리 항목으로 이동합니다.

4단계: 원장에서 테이블 쿼리

Amazon QLDB 원장에 테이블을 생성하고 데이터를 로드한 후 쿼리를 실행하여 방금 삽입한 차량 등록 데이터를 검토할 수 있습니다. QLDB는 PartiQL을 쿼리 언어로 사용하고 Amazon Ion을 문서 지향 데이터 모델로 사용합니다.

PartiQL은 Ion과 함께 작동하도록 확장된 오픈 소스 SQL 호환 쿼리 언어입니다. PartiQL을 사용하면 익숙한 SQL 연산자를 사용하여 데이터를 삽입, 쿼리 및 관리할 수 있습니다. Amazon Ion은 JSON의 상위 집합입니다. Ion은 정형, 반정형 및 중첩 데이터를 저장하고 처리할 수 있는 유연성을 제공하는 오픈 소스 문서 기반 데이터 형식입니다.

이 단계에서는 `SELECT` 명령문을 사용하여 `vehicle-registration` 원장의 테이블에서 데이터를 읽습니다.

> **Warning**
>
> 인덱싱된 조회 없이 QLDB에서 쿼리를 실행하면 전체 테이블 스캔이 호출됩니다. PartiQL은 SQL과 호환되므로 이러한 쿼리를 지원합니다. 하지만 QLDB의 프로덕션 사용 사례에 대해서는 테이블 스캔을 실행하지 마세요. 테이블 스캔은 동시성 충돌 및 트랜잭션 시간 초과를 포함하여 대규모 테이블에서 성능 문제를 일으킬 수 있습니다.
> 인덱싱된 필드 또는 문서 ID(예: `WHERE indexedField = 123` 또는 `WHERE indexedField IN (456, 789)`)에서 동등 연산자를 사용하여 `WHERE` 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 자세한 내용은 쿼리 성능 최적화 섹션을 참조하세요.

테이블을 쿼리하려면

- 다음 프로그램(FindVehicles.java)을 컴파일하고 실행하여 원장에 있는 사람의 이름으로 등록된 모든 차량을 쿼리하십시오.

  2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.io.IOException;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;

import software.amazon.qldb.Result;
```

```
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Find all vehicles registered under a person.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class FindVehicles {
    public static final Logger log =
 LoggerFactory.getLogger(FindVehicles.class);

    private FindVehicles() { }

    /**
     * Find vehicles registered under a driver using their government ID.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param govId
     *              The government ID of the owner.
     * @throws IllegalStateException if failed to convert parameters into {@link
 IonValue}.
     */
    public static void findVehiclesForOwner(final TransactionExecutor txn, final
 String govId) {
        try {
            final String documentId = Person.getDocumentIdByGovId(txn, govId);
            final String query = "SELECT v FROM Vehicle AS v INNER JOIN
 VehicleRegistration AS r "
                    + "ON v.VIN = r.VIN WHERE r.Owners.PrimaryOwner.PersonId
 = ?";

            final Result result = txn.execute(query,
 Constants.MAPPER.writeValueAsIonValue(documentId));
            log.info("List of Vehicles for owner with GovId: {}...", govId);
            ScanTable.printDocuments(result);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }
```

```
    public static void main(final String... args) {
        final Person person = SampleData.PEOPLE.get(0);
        ConnectToLedger.getDriver().execute(txn -> {
            findVehiclesForOwner(txn, person.getGovId());
        });
    }
}
```

1.x

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.io.IOException;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import com.amazon.ion.IonValue;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Find all vehicles registered under a person.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class FindVehicles {
    public static final Logger log =
 LoggerFactory.getLogger(FindVehicles.class);

    private FindVehicles() { }

    /**
     * Find vehicles registered under a driver using their government ID.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param govId
     *              The government ID of the owner.
     * @throws IllegalStateException if failed to convert parameters into {@link
 IonValue}.
     */
    public static void findVehiclesForOwner(final TransactionExecutor txn, final
 String govId) {
        try {
            final String documentId = Person.getDocumentIdByGovId(txn, govId);
            final String query = "SELECT v FROM Vehicle AS v INNER JOIN
 VehicleRegistration AS r "
                    + "ON v.VIN = r.VIN WHERE r.Owners.PrimaryOwner.PersonId
 = ?";

            final Result result = txn.execute(query,
 Constants.MAPPER.writeValueAsIonValue(documentId));
            log.info("List of Vehicles for owner with GovId: {}...", govId);
            ScanTable.printDocuments(result);
```

```
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    public static void main(final String... args) {
        final Person person = SampleData.PEOPLE.get(0);
        ConnectToLedger.getDriver().execute(txn -> {
            findVehiclesForOwner(txn, person.getGovId());
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
    }
}
```

> **Note**
>
> 먼저 이 프로그램은 이 문서에 대한 `Person` 테이블을 `GovId LEWISR261LL`로 쿼리하여 `id` 메타데이터 필드를 가져옵니다.
> 그런 다음 이 문서 `id`를 외래 키로 사용하여 `VehicleRegistration` 테이블을 `PrimaryOwner.PersonId`로 쿼리합니다. 또한 `VIN` 필드의 `Vehicle` 테이블과 `VehicleRegistration`를 조인합니다.

`vehicle-registration` 원장의 테이블에 있는 문서를 수정하는 방법에 대한 자세한 내용은 5단계: 원장의 문서 수정을 참조하십시오.

5단계: 원장의 문서 수정

이제 작업할 데이터가 있으므로 Amazon QLDB에서 `vehicle-registration` 원장에 있는 문서를 변경할 수 있습니다. 이 단계에서 다음 코드 예제는 DML(데이터 조작 언어) 문을 실행하는 방법을 보여 줍니다. 이러한 명령문은 한 차량의 주 소유자를 업데이트하고 다른 차량에 보조 소유자를 추가합니다.

문서를 수정하려면

1. 다음 프로그램(`TransferVehicleOwnership.java`)을 컴파일하고 실행하여 차량의 기본 소유주를 원장의 VIN `1N4AL11D75C109151`으로 업데이트합니다.

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonReader;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonReaderBuilder;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Owner;
import software.amazon.qldb.tutorial.model.Person;
```

```
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Find primary owner for a particular vehicle's VIN.
 * Transfer to another primary owner for a particular vehicle's VIN.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class TransferVehicleOwnership {
    public static final Logger log =
 LoggerFactory.getLogger(TransferVehicleOwnership.class);

    private TransferVehicleOwnership() { }

    /**
     * Query a driver's information using the given ID.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param documentId
     *              The unique ID of a document in the Person table.
     * @return a {@link Person} object.
     * @throws IllegalStateException if failed to convert parameter into {@link
 IonValue}.
     */
    public static Person findPersonFromDocumentId(final TransactionExecutor txn,
 final String documentId) {
        try {
            log.info("Finding person for documentId: {}...", documentId);
            final String query = "SELECT p.* FROM Person AS p BY pid WHERE pid
 = ?";

            Result result = txn.execute(query,
 Constants.MAPPER.writeValueAsIonValue(documentId));
            if (result.isEmpty()) {
                throw new IllegalStateException("Unable to find person with ID:
 " + documentId);
            }

            return Constants.MAPPER.readValue(result.iterator().next(),
 Person.class);
        } catch (IOException ioe) {
```

```
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Find the primary owner for the given VIN.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              Unique VIN for a vehicle.
     * @return a {@link Person} object.
     * @throws IllegalStateException if failed to convert parameter into {@link
IonValue}.
     */
    public static Person findPrimaryOwnerForVehicle(final TransactionExecutor
txn, final String vin) {
        try {
            log.info("Finding primary owner for vehicle with Vin: {}...", vin);
            final String query = "SELECT Owners.PrimaryOwner.PersonId FROM
VehicleRegistration AS v WHERE v.VIN = ?";
            final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
            Result result = txn.execute(query, parameters);
            final List<IonStruct> documents = ScanTable.toIonStructs(result);
            ScanTable.printDocuments(documents);
            if (documents.isEmpty()) {
                throw new IllegalStateException("Unable to find registrations
with VIN: " + vin);
            }

            final IonReader reader =
IonReaderBuilder.standard().build(documents.get(0));
            final String personId = Constants.MAPPER.readValue(reader,
LinkedHashMap.class).get("PersonId").toString();
            return findPersonFromDocumentId(txn, personId);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Update the primary owner for a vehicle registration with the given
documentId.
```

```
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              Unique VIN for a vehicle.
     * @param documentId
     *              New PersonId for the primary owner.
     * @throws IllegalStateException if no vehicle registration was found using
the given document ID and VIN, or if failed
     * to convert parameters into {@link IonValue}.
     */
    public static void updateVehicleRegistration(final TransactionExecutor txn,
final String vin, final String documentId) {
        try {
            log.info("Updating primary owner for vehicle with Vin: {}...", vin);
            final String query = "UPDATE VehicleRegistration AS v SET
v.Owners.PrimaryOwner = ? WHERE v.VIN = ?";

            final List<IonValue> parameters = new ArrayList<>();
            parameters.add(Constants.MAPPER.writeValueAsIonValue(new
Owner(documentId)));
            parameters.add(Constants.MAPPER.writeValueAsIonValue(vin));

            Result result = txn.execute(query, parameters);
            ScanTable.printDocuments(result);
            if (result.isEmpty()) {
                throw new IllegalStateException("Unable to transfer vehicle,
could not find registration.");
            } else {
                log.info("Successfully transferred vehicle with VIN '{}' to new
owner.", vin);
            }
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    public static void main(final String... args) {
        final String vin = SampleData.VEHICLES.get(0).getVin();
        final String primaryOwnerGovId = SampleData.PEOPLE.get(0).getGovId();
        final String newPrimaryOwnerGovId = SampleData.PEOPLE.get(1).getGovId();

        ConnectToLedger.getDriver().execute(txn -> {
            final Person primaryOwner = findPrimaryOwnerForVehicle(txn, vin);
```

```
            if (!primaryOwner.getGovId().equals(primaryOwnerGovId)) {
                // Verify the primary owner.
                throw new IllegalStateException("Incorrect primary owner
 identified for vehicle, unable to transfer.");
            }

            final String newOwner = Person.getDocumentIdByGovId(txn,
 newPrimaryOwnerGovId);
            updateVehicleRegistration(txn, vin, newOwner);
        });
        log.info("Successfully transferred vehicle ownership!");
    }
}
```

1.x

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;
```

```
import com.amazon.ion.IonReader;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonReaderBuilder;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Owner;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Find primary owner for a particular vehicle's VIN.
 * Transfer to another primary owner for a particular vehicle's VIN.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class TransferVehicleOwnership {
    public static final Logger log =
 LoggerFactory.getLogger(TransferVehicleOwnership.class);

    private TransferVehicleOwnership() { }

    /**
     * Query a driver's information using the given ID.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param documentId
     *              The unique ID of a document in the Person table.
     * @return a {@link Person} object.
```

```
    * @throws IllegalStateException if failed to convert parameter into {@link
IonValue}.
    */
   public static Person findPersonFromDocumentId(final TransactionExecutor txn,
final String documentId) {
       try {
           log.info("Finding person for documentId: {}...", documentId);
           final String query = "SELECT p.* FROM Person AS p BY pid WHERE pid
= ?";

           Result result = txn.execute(query,
Constants.MAPPER.writeValueAsIonValue(documentId));
           if (result.isEmpty()) {
               throw new IllegalStateException("Unable to find person with ID:
" + documentId);
           }

           return Constants.MAPPER.readValue(result.iterator().next(),
Person.class);
       } catch (IOException ioe) {
           throw new IllegalStateException(ioe);
       }
   }

   /**
    * Find the primary owner for the given VIN.
    *
    * @param txn
    *              The {@link TransactionExecutor} for lambda execute.
    * @param vin
    *              Unique VIN for a vehicle.
    * @return a {@link Person} object.
    * @throws IllegalStateException if failed to convert parameter into {@link
IonValue}.
    */
   public static Person findPrimaryOwnerForVehicle(final TransactionExecutor
txn, final String vin) {
       try {
           log.info("Finding primary owner for vehicle with Vin: {}...", vin);
           final String query = "SELECT Owners.PrimaryOwner.PersonId FROM
VehicleRegistration AS v WHERE v.VIN = ?";
           final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
           Result result = txn.execute(query, parameters);
```

```
            final List<IonStruct> documents = ScanTable.toIonStructs(result);
            ScanTable.printDocuments(documents);
            if (documents.isEmpty()) {
                throw new IllegalStateException("Unable to find registrations
with VIN: " + vin);
            }

            final IonReader reader =
IonReaderBuilder.standard().build(documents.get(0));
            final String personId = Constants.MAPPER.readValue(reader,
LinkedHashMap.class).get("PersonId").toString();
            return findPersonFromDocumentId(txn, personId);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Update the primary owner for a vehicle registration with the given
documentId.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              Unique VIN for a vehicle.
     * @param documentId
     *              New PersonId for the primary owner.
     * @throws IllegalStateException if no vehicle registration was found using
the given document ID and VIN, or if failed
     * to convert parameters into {@link IonValue}.
     */
    public static void updateVehicleRegistration(final TransactionExecutor txn,
final String vin, final String documentId) {
        try {
            log.info("Updating primary owner for vehicle with Vin: {}...", vin);
            final String query = "UPDATE VehicleRegistration AS v SET
v.Owners.PrimaryOwner = ? WHERE v.VIN = ?";

            final List<IonValue> parameters = new ArrayList<>();
            parameters.add(Constants.MAPPER.writeValueAsIonValue(new
Owner(documentId)));
            parameters.add(Constants.MAPPER.writeValueAsIonValue(vin));

            Result result = txn.execute(query, parameters);
```

```
            ScanTable.printDocuments(result);
            if (result.isEmpty()) {
                throw new IllegalStateException("Unable to transfer vehicle,
 could not find registration.");
            } else {
                log.info("Successfully transferred vehicle with VIN '{}' to new
 owner.", vin);
            }
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    public static void main(final String... args) {
        final String vin = SampleData.VEHICLES.get(0).getVin();
        final String primaryOwnerGovId = SampleData.PEOPLE.get(0).getGovId();
        final String newPrimaryOwnerGovId = SampleData.PEOPLE.get(1).getGovId();

        ConnectToLedger.getDriver().execute(txn -> {
            final Person primaryOwner = findPrimaryOwnerForVehicle(txn, vin);
            if (!primaryOwner.getGovId().equals(primaryOwnerGovId)) {
                // Verify the primary owner.
                throw new IllegalStateException("Incorrect primary owner
 identified for vehicle, unable to transfer.");
            }

            final String newOwner = Person.getDocumentIdByGovId(txn,
 newPrimaryOwnerGovId);
            updateVehicleRegistration(txn, vin, newOwner);
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
        log.info("Successfully transferred vehicle ownership!");
    }
}
```

2. 다음 프로그램(AddSecondaryOwner.java)을 컴파일하고 실행하여 보조 소유자를 원장의 VIN KM8SRDHF6EU074761로 차량에 추가합니다.

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
```

```
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.io.IOException;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Owner;
import software.amazon.qldb.tutorial.model.Owners;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Finds and adds secondary owners for a vehicle.
 *
```

```
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class AddSecondaryOwner {
    public static final Logger log =
 LoggerFactory.getLogger(AddSecondaryOwner.class);

    private AddSecondaryOwner() { }

    /**
     * Check whether a secondary owner has already been registered for the given
 VIN.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              Unique VIN for a vehicle.
     * @param secondaryOwnerId
     *              The secondary owner to add.
     * @return {@code true} if the given secondary owner has already been
 registered, {@code false} otherwise.
     * @throws IllegalStateException if failed to convert VIN to an {@link
 IonValue}.
     */
    public static boolean isSecondaryOwnerForVehicle(final TransactionExecutor
 txn, final String vin,
                                                     final String
 secondaryOwnerId) {
        try {
            log.info("Finding secondary owners for vehicle with VIN: {}...",
 vin);
            final String query = "SELECT Owners.SecondaryOwners FROM
 VehicleRegistration AS v WHERE v.VIN = ?";
            final List<IonValue> parameters =
 Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
            final Result result = txn.execute(query, parameters);
            final Iterator<IonValue> itr = result.iterator();
            if (!itr.hasNext()) {
                return false;
            }

            final Owners owners = Constants.MAPPER.readValue(itr.next(),
 Owners.class);
```

```
            if (null != owners.getSecondaryOwners()) {
                for (Owner owner : owners.getSecondaryOwners()) {
                    if (secondaryOwnerId.equalsIgnoreCase(owner.getPersonId()))
{
                        return true;
                    }
                }
            }

            return false;
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Adds a secondary owner for the specified VIN.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              Unique VIN for a vehicle.
     * @param secondaryOwner
     *              The secondary owner to add.
     * @throws IllegalStateException if failed to convert parameter into an
{@link IonValue}.
     */
    public static void addSecondaryOwnerForVin(final TransactionExecutor txn,
final String vin,
                                               final String secondaryOwner) {
        try {
            log.info("Inserting secondary owner for vehicle with VIN: {}...",
vin);
            final String query = String.format("FROM VehicleRegistration AS v
WHERE v.VIN = ?" +
                    "INSERT INTO v.Owners.SecondaryOwners VALUE ?");
            final IonValue newOwner = Constants.MAPPER.writeValueAsIonValue(new
Owner(secondaryOwner));
            final IonValue vinAsIonValue =
Constants.MAPPER.writeValueAsIonValue(vin);
            Result result = txn.execute(query, vinAsIonValue, newOwner);
            log.info("VehicleRegistration Document IDs which had secondary
owners added: ");
            ScanTable.printDocuments(result);
```

```
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    public static void main(final String... args) {
        final String vin = SampleData.VEHICLES.get(1).getVin();
        final String govId = SampleData.PEOPLE.get(0).getGovId();

        ConnectToLedger.getDriver().execute(txn -> {
            final String documentId = Person.getDocumentIdByGovId(txn, govId);
            if (isSecondaryOwnerForVehicle(txn, vin, documentId)) {
                log.info("Person with ID {} has already been added as a
 secondary owner of this vehicle.", govId);
            } else {
                addSecondaryOwnerForVin(txn, vin, documentId);
            }
        });
        log.info("Secondary owners successfully updated.");
    }
}
```

1.x

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
```

```
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.io.IOException;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.Owner;
import software.amazon.qldb.tutorial.model.Owners;
import software.amazon.qldb.tutorial.model.Person;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Finds and adds secondary owners for a vehicle.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class AddSecondaryOwner {
    public static final Logger log =
 LoggerFactory.getLogger(AddSecondaryOwner.class);

    private AddSecondaryOwner() { }

    /**
     * Check whether a secondary owner has already been registered for the given
 VIN.
     *
     * @param txn
```

```
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              Unique VIN for a vehicle.
     * @param secondaryOwnerId
     *              The secondary owner to add.
     * @return {@code true} if the given secondary owner has already been
registered, {@code false} otherwise.
     * @throws IllegalStateException if failed to convert VIN to an {@link
IonValue}.
     */
    public static boolean isSecondaryOwnerForVehicle(final TransactionExecutor
txn, final String vin,
                                                     final String
secondaryOwnerId) {
        try {
            log.info("Finding secondary owners for vehicle with VIN: {}...",
vin);
            final String query = "SELECT Owners.SecondaryOwners FROM
VehicleRegistration AS v WHERE v.VIN = ?";
            final List<IonValue> parameters =
Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
            final Result result = txn.execute(query, parameters);
            final Iterator<IonValue> itr = result.iterator();
            if (!itr.hasNext()) {
                return false;
            }

            final Owners owners = Constants.MAPPER.readValue(itr.next(),
Owners.class);
            if (null != owners.getSecondaryOwners()) {
                for (Owner owner : owners.getSecondaryOwners()) {
                    if (secondaryOwnerId.equalsIgnoreCase(owner.getPersonId()))
{
                        return true;
                    }
                }
            }

            return false;
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }
```

```
   /**
    * Adds a secondary owner for the specified VIN.
    *
    * @param txn
    *              The {@link TransactionExecutor} for lambda execute.
    * @param vin
    *              Unique VIN for a vehicle.
    * @param secondaryOwner
    *              The secondary owner to add.
    * @throws IllegalStateException if failed to convert parameter into an
{@link IonValue}.
    */
   public static void addSecondaryOwnerForVin(final TransactionExecutor txn,
final String vin,
                                              final String secondaryOwner) {
       try {
           log.info("Inserting secondary owner for vehicle with VIN: {}...",
vin);
           final String query = String.format("FROM VehicleRegistration AS v
WHERE v.VIN = '%s' " +
                   "INSERT INTO v.Owners.SecondaryOwners VALUE ?", vin);
           final IonValue newOwner = Constants.MAPPER.writeValueAsIonValue(new
Owner(secondaryOwner));
           Result result = txn.execute(query, newOwner);
           log.info("VehicleRegistration Document IDs which had secondary
owners added: ");
           ScanTable.printDocuments(result);
       } catch (IOException ioe) {
           throw new IllegalStateException(ioe);
       }
   }

   public static void main(final String... args) {
       final String vin = SampleData.VEHICLES.get(1).getVin();
       final String govId = SampleData.PEOPLE.get(0).getGovId();

       ConnectToLedger.getDriver().execute(txn -> {
           final String documentId = Person.getDocumentIdByGovId(txn, govId);
           if (isSecondaryOwnerForVehicle(txn, vin, documentId)) {
               log.info("Person with ID {} has already been added as a
secondary owner of this vehicle.", govId);
           } else {
               addSecondaryOwnerForVin(txn, vin, documentId);
           }
```

```
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
        log.info("Secondary owners successfully updated.");
    }
}
```

vehicle-registration 원장의 이러한 변경 사항을 검토하려면 6단계: 문서의 개정 기록 보기을 참조하세요.

6단계: 문서의 개정 기록 보기

이전 단계에서 차량의 등록 데이터를 수정한 후 등록된 모든 소유자의 기록과 기타 업데이트된 필드를 쿼리할 수 있습니다. 이 단계에서는 vehicle-registration 원장의 VehicleRegistration 테이블에 있는 문서의 개정 기록을 쿼리합니다.

개정 기록을 보려면

1. 다음 프로그램(QueryHistory.java)을 검토합니다.

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
```

```
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.io.IOException;
import java.time.Instant;
import java.time.temporal.ChronoUnit;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonValue;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.model.VehicleRegistration;

/**
 * Query a table's history for a particular set of documents.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class QueryHistory {
    public static final Logger log =
 LoggerFactory.getLogger(QueryHistory.class);
    private static final int THREE_MONTHS = 90;

    private QueryHistory() { }

    /**
     * In this example, query the 'VehicleRegistration' history table to find
 all previous primary owners for a VIN.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              VIN to find previous primary owners for.
     * @param query
```

```
     *              The query to find previous primary owners.
     * @throws IllegalStateException if failed to convert document ID to an
 {@link IonValue}.
     */
    public static void previousPrimaryOwners(final TransactionExecutor txn,
 final String vin, final String query) {
        try {
            final String docId = VehicleRegistration.getDocumentIdByVin(txn,
 vin);

            log.info("Querying the 'VehicleRegistration' table's history using
 VIN: {}...", vin);
            final Result result = txn.execute(query,
 Constants.MAPPER.writeValueAsIonValue(docId));
            ScanTable.printDocuments(result);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    public static void main(final String... args) {
        final String threeMonthsAgo = Instant.now().minus(THREE_MONTHS,
 ChronoUnit.DAYS).toString();
        final String query = String.format("SELECT data.Owners.PrimaryOwner,
 metadata.version "
                                           + "FROM history(VehicleRegistration,
 `%s`) "
                                           + "AS h WHERE h.metadata.id = ?",
 threeMonthsAgo);
        ConnectToLedger.getDriver().execute(txn -> {
            final String vin = SampleData.VEHICLES.get(0).getVin();
            previousPrimaryOwners(txn, vin, query);
        });
        log.info("Successfully queried history.");
    }
}
```

1.x

```
/*
 * Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
```

```
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonValue;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.model.VehicleRegistration;

import java.io.IOException;
import java.time.Instant;
import java.time.temporal.ChronoUnit;
import java.util.Collections;
import java.util.List;

/**
 * Query a table's history for a particular set of documents.
 *
 * This code expects that you have AWS credentials setup per:
```

```
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class QueryHistory {
    public static final Logger log =
 LoggerFactory.getLogger(QueryHistory.class);
    private static final int THREE_MONTHS = 90;

    private QueryHistory() { }

    /**
     * In this example, query the 'VehicleRegistration' history table to find
 all previous primary owners for a VIN.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              VIN to find previous primary owners for.
     * @param query
     *              The query to find previous primary owners.
     * @throws IllegalStateException if failed to convert document ID to an
 {@link IonValue}.
     */
    public static void previousPrimaryOwners(final TransactionExecutor txn,
 final String vin, final String query) {
        try {
            final String docId = VehicleRegistration.getDocumentIdByVin(txn,
 vin);
            final List<IonValue> parameters =
 Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(docId));
            log.info("Querying the 'VehicleRegistration' table's history using
 VIN: {}...", vin);
            final Result result = txn.execute(query, parameters);
            ScanTable.printDocuments(result);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    public static void main(final String... args) {
        final String threeMonthsAgo = Instant.now().minus(THREE_MONTHS,
 ChronoUnit.DAYS).toString();
        final String query = String.format("SELECT data.Owners.PrimaryOwner,
 metadata.version "
```

```
                                                + "FROM history(VehicleRegistration,
 `%s`) "
                                                + "AS h WHERE h.metadata.id = ?",
 threeMonthsAgo);
        ConnectToLedger.getDriver().execute(txn -> {
            final String vin = SampleData.VEHICLES.get(0).getVin();
            previousPrimaryOwners(txn, vin, query);
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
        log.info("Successfully queried history.");
    }
}
```

> **Note**
>
> - 다음 구문에 내장된 기록 함수를 쿼리하여 문서의 개정 기록을 볼 수 있습니다
>
>   ```
>   SELECT * FROM history( table_name [, `start-time` [, `end-time` ] ] ) AS h
>   [ WHERE h.metadata.id = 'id' ]
>   ```
>
> - 시작 시간과 종료 시간은 모두 선택 사항입니다. 이 값은 백틱(`` `...` ``)으로 표시할 수 있는 Amazon Ion 리터럴 값입니다. 자세한 내용은 Amazon QLDB에서 PartiQL을 사용한 Ion 쿼리 단원을 참조하십시오.
> - 가장 좋은 방법은 날짜 범위(시작 시간 및 종료 시간) 와 문서 ID(`metadata.id`)를 모두 사용하여 기록 쿼리를 검증하는 것입니다. QLDB는 트랜잭션에서 SELECT 쿼리를 처리하며, 이 쿼리에는 트랜잭션 시간 초과 제한이 적용됩니다.
>
>   QLDB 기록은 문서 ID로 인덱싱되며, 이번에는 추가 기록 인덱스를 만들 수 없습니다. 시작 시간과 종료 시간을 포함하는 기록 쿼리는 날짜 범위 한정이라는 이점을 얻습니다.

2. `QueryHistory.java` 프로그램을 컴파일하고 실행하여 VIN `1N4AL11D75C109151`으로 `VehicleRegistration` 문서의 개정 기록을 쿼리합니다.

`vehicle-registration` 원장의 문서 개정을 암호화 방식으로 검증하려면 7단계: 원장에 있는 문서 검증으로 진행하세요.

7단계: 원장에 있는 문서 검증

Amazon QLDB를 사용하면 SHA-256 암호화 해싱을 사용하여 원장 저널에 있는 문서의 무결성을 효율적으로 검증할 수 있습니다. QLDB에서 검증 및 암호화 해싱이 작동하는 방식에 대한 자세한 내용은 Amazon QLDB에서의 데이터 확인을 참조하십시오.

이 단계에서는 `vehicle-registration` 원장의 `VehicleRegistration` 테이블에 있는 문서 개정을 검증합니다. 먼저 다이제스트를 요청합니다. 다이제스트는 출력 파일로 반환되며 원장의 전체 변경 내역에 대한 서명 역할을 합니다. 그런 다음 해당 다이제스트와 관련된 개정 증거를 요청합니다. 이 증거를 사용하면 모든 유효성 검사를 통과한 경우 개정 내용의 무결성을 확인할 수 있습니다.

문서 개정을 검증하려면

1. 검증에 필요한 QLDB 객체와 Ion 및 문자열 값에 대한 도우미 메서드가 포함된 유틸리티 클래스를 나타내는 다음 `.java` 파일을 검토하세요.

   1. `BlockAddress.java`

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */
```

```
package software.amazon.qldb.tutorial.qldb;

import java.util.Objects;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

/**
 * Represents the BlockAddress field of a QLDB document.
 */
public final class BlockAddress {

    private static final Logger log =
 LoggerFactory.getLogger(BlockAddress.class);

    private final String strandId;
    private final long sequenceNo;

    @JsonCreator
    public BlockAddress(@JsonProperty("strandId") final String strandId,
                        @JsonProperty("sequenceNo") final long sequenceNo) {
        this.strandId = strandId;
        this.sequenceNo = sequenceNo;
    }

    public long getSequenceNo() {
        return sequenceNo;
    }

    public String getStrandId() {
        return strandId;
    }

    @Override
    public String toString() {
        return "BlockAddress{"
                + "strandId='" + strandId + '\''
                + ", sequenceNo=" + sequenceNo
                + '}';
    }
```

```
    @Override
    public boolean equals(final Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        BlockAddress that = (BlockAddress) o;
        return sequenceNo == that.sequenceNo
                && strandId.equals(that.strandId);
    }

    @Override
    public int hashCode() {
        // CHECKSTYLE:OFF - Disabling as we are generating a hashCode of multiple
 properties.
        return Objects.hash(strandId, sequenceNo);
        // CHECKSTYLE:ON
    }
}
```

2. Proof.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
```

```
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.amazon.ion.IonReader;
import com.amazon.ion.IonSystem;
import com.amazon.ion.system.IonSystemBuilder;
import com.amazonaws.services.qldb.model.GetRevisionRequest;
import com.amazonaws.services.qldb.model.GetRevisionResult;

import java.util.ArrayList;
import java.util.List;

/**
 * A Java representation of the {@link Proof} object.
 * Returned from the {@link
 com.amazonaws.services.qldb.AmazonQLDB#getRevision(GetRevisionRequest)} api.
 */
public final class Proof {
    private static final IonSystem SYSTEM = IonSystemBuilder.standard().build();

    private List<byte[]> internalHashes;

    public Proof(final List<byte[]> internalHashes) {
        this.internalHashes = internalHashes;
    }

    public List<byte[]> getInternalHashes() {
        return internalHashes;
    }

    /**
     * Decodes a {@link Proof} from an ion text String. This ion text is returned
 in
     * a {@link GetRevisionResult#getProof()}
     *
     * @param ionText
     *              The ion text representing a {@link Proof} object.
     * @return {@link JournalBlock} parsed from the ion text.
```

```
     * @throws IllegalStateException if failed to parse the {@link Proof} object
 from the given ion text.
     */
    public static Proof fromBlob(final String ionText) {
        try {
            IonReader reader = SYSTEM.newReader(ionText);
            List<byte[]> list = new ArrayList<>();
            reader.next();
            reader.stepIn();
            while (reader.next() != null) {
                list.add(reader.newBytes());
            }
            return new Proof(list);
        } catch (Exception e) {
            throw new IllegalStateException("Failed to parse a Proof from byte
 array");
        }
    }
}
```

3. QldbIonUtils.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
```

```
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.amazon.ion.IonReader;
import com.amazon.ion.IonValue;
import com.amazon.ionhash.IonHashReader;
import com.amazon.ionhash.IonHashReaderBuilder;
import com.amazon.ionhash.MessageDigestIonHasherProvider;
import software.amazon.qldb.tutorial.Constants;

public class QldbIonUtils {

    private static MessageDigestIonHasherProvider ionHasherProvider = new
 MessageDigestIonHasherProvider("SHA-256");

    private QldbIonUtils() {}

    /**
     * Builds a hash value from the given {@link IonValue}.
     *
     * @param ionValue
     *              The {@link IonValue} to hash.
     * @return a byte array representing the hash value.
     */
    public static byte[] hashIonValue(final IonValue ionValue) {
        IonReader reader = Constants.SYSTEM.newReader(ionValue);
        IonHashReader hashReader = IonHashReaderBuilder.standard()
                .withHasherProvider(ionHasherProvider)
                .withReader(reader)
                .build();
        while (hashReader.next() != null) {  }
        return hashReader.digest();
    }

}
```

4. QldbStringUtils.java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
```

```
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.amazon.ion.IonWriter;
import com.amazon.ion.system.IonReaderBuilder;
import com.amazon.ion.system.IonTextWriterBuilder;
import com.amazonaws.services.qldb.model.GetBlockResult;
import com.amazonaws.services.qldb.model.GetDigestResult;
import com.amazonaws.services.qldb.model.ValueHolder;

import java.io.IOException;

/**
 * Helper methods to pretty-print certain QLDB response types.
 */
public class QldbStringUtils {

    private QldbStringUtils() {}

    /**
     * Returns the string representation of a given {@link ValueHolder}.
     * Adapted from the AWS SDK autogenerated {@code toString()} method, with
 sensitive values un-redacted.
```

```
    * Additionally, this method pretty-prints any IonText included in the {@link
ValueHolder}.
    *
    * @param valueHolder the {@link ValueHolder} to convert to a String.
    * @return the String representation of the supplied {@link ValueHolder}.
    */
   public static String toUnredactedString(ValueHolder valueHolder) {
       StringBuilder sb = new StringBuilder();
       sb.append("{");
       if (valueHolder.getIonText() != null) {

           sb.append("IonText: ");
           IonWriter prettyWriter = IonTextWriterBuilder.pretty().build(sb);
           try {

prettyWriter.writeValues(IonReaderBuilder.standard().build(valueHolder.getIonText()));
           } catch (IOException ioe) {
               sb.append("**Exception while printing this IonText**");
           }
       }

       sb.append("}");
       return sb.toString();
   }

   /**
    * Returns the string representation of a given {@link GetBlockResult}.
    * Adapted from the AWS SDK autogenerated {@code toString()} method, with
sensitive values un-redacted.
    *
    * @param getBlockResult the {@link GetBlockResult} to convert to a String.
    * @return the String representation of the supplied {@link GetBlockResult}.
    */
   public static String toUnredactedString(GetBlockResult getBlockResult) {
       StringBuilder sb = new StringBuilder();
       sb.append("{");
       if (getBlockResult.getBlock() != null) {
           sb.append("Block:
").append(toUnredactedString(getBlockResult.getBlock())).append(",");
       }

       if (getBlockResult.getProof() != null) {
           sb.append("Proof:
").append(toUnredactedString(getBlockResult.getProof()));
```

```
        }

        sb.append("}");
        return sb.toString();
    }

    /**
     * Returns the string representation of a given {@link GetDigestResult}.
     * Adapted from the AWS SDK autogenerated {@code toString()} method, with
 sensitive values un-redacted.
     *
     * @param getDigestResult the {@link GetDigestResult} to convert to a String.
     * @return the String representation of the supplied {@link GetDigestResult}.
     */
    public static String toUnredactedString(GetDigestResult getDigestResult) {
        StringBuilder sb = new StringBuilder();
        sb.append("{");
        if (getDigestResult.getDigest() != null) {
            sb.append("Digest:
 ").append(getDigestResult.getDigest()).append(",");
        }

        if (getDigestResult.getDigestTipAddress() != null) {
            sb.append("DigestTipAddress:
 ").append(toUnredactedString(getDigestResult.getDigestTipAddress()));
        }

        sb.append("}");
        return sb.toString();
    }
}
```

5. `Verifier.java`

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 copy of this
 * software and associated documentation files (the "Software"), to deal in
 the Software
```

```
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR
 A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazonaws.util.Base64;

import software.amazon.qldb.tutorial.qldb.Proof;

/**
 * Encapsulates the logic to verify the integrity of revisions or blocks in a
 QLDB ledger.
 *
 * The main entry point is {@link #verify(byte[], byte[], String)}.
```

```
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class Verifier {
    public static final Logger log = LoggerFactory.getLogger(Verifier.class);
    private static final int HASH_LENGTH = 32;
    private static final int UPPER_BOUND = 8;

    /**
     * Compares two hashes by their <em>signed</em> byte values in little-
endian order.
     */
    private static Comparator<byte[]> hashComparator = (h1, h2) -> {
        if (h1.length != HASH_LENGTH || h2.length != HASH_LENGTH) {
            throw new IllegalArgumentException("Invalid hash.");
        }
        for (int i = h1.length - 1; i >= 0; i--) {
            int byteEqual = Byte.compare(h1[i], h2[i]);
            if (byteEqual != 0) {
                return byteEqual;
            }
        }

        return 0;
    };

    private Verifier() { }

    /**
     * Verify the integrity of a document with respect to a QLDB ledger
 digest.
     *
     * The verification algorithm includes the following steps:
     *
     * 1. {@link #buildCandidateDigest(Proof, byte[])} build the candidate
 digest from the internal hashes
     * in the {@link Proof}.
     * 2. Check that the {@code candidateLedgerDigest} is equal to the {@code
 ledgerDigest}.
     *
     * @param documentHash
     *              The hash of the document to be verified.
```

```
    * @param digest
    *              The QLDB ledger digest. This digest should have been
retrieved using
    *              {@link com.amazonaws.services.qldb.AmazonQLDB#getDigest}
    * @param proofBlob
    *              The ion encoded bytes representing the {@link Proof}
associated with the supplied
    *              {@code digestTipAddress} and {@code address} retrieved
using
    *              {@link
com.amazonaws.services.qldb.AmazonQLDB#getRevision}.
    * @return {@code true} if the record is verified or {@code false} if it
is not verified.
    */
   public static boolean verify(
           final byte[] documentHash,
           final byte[] digest,
           final String proofBlob
   ) {
       Proof proof = Proof.fromBlob(proofBlob);

       byte[] candidateDigest = buildCandidateDigest(proof, documentHash);

       return Arrays.equals(digest, candidateDigest);
   }

   /**
    * Build the candidate digest representing the entire ledger from the
internal hashes of the {@link Proof}.
    *
    * @param proof
    *              A Java representation of {@link Proof}
    *              returned from {@link
com.amazonaws.services.qldb.AmazonQLDB#getRevision}.
    * @param leafHash
    *              Leaf hash to build the candidate digest with.
    * @return a byte array of the candidate digest.
    */
   private static byte[] buildCandidateDigest(final Proof proof, final byte[]
leafHash) {
       return calculateRootHashFromInternalHashes(proof.getInternalHashes(),
leafHash);
   }
```

```
   /**
    * Get a new instance of {@link MessageDigest} using the SHA-256
algorithm.
    *
    * @return an instance of {@link MessageDigest}.
    * @throws IllegalStateException if the algorithm is not available on the
current JVM.
    */
   static MessageDigest newMessageDigest() {
       try {
           return MessageDigest.getInstance("SHA-256");
       } catch (NoSuchAlgorithmException e) {
           log.error("Failed to create SHA-256 MessageDigest", e);
           throw new IllegalStateException("SHA-256 message digest is
unavailable", e);
       }
   }

   /**
    * Takes two hashes, sorts them, concatenates them, and then returns the
    * hash of the concatenated array.
    *
    * @param h1
    *              Byte array containing one of the hashes to compare.
    * @param h2
    *              Byte array containing one of the hashes to compare.
    * @return the concatenated array of hashes.
    */
   public static byte[] dot(final byte[] h1, final byte[] h2) {
       if (h1.length == 0) {
           return h2;
       }
       if (h2.length == 0) {
           return h1;
       }
       byte[] concatenated = new byte[h1.length + h2.length];
       if (hashComparator.compare(h1, h2) < 0) {
           System.arraycopy(h1, 0, concatenated, 0, h1.length);
           System.arraycopy(h2, 0, concatenated, h1.length, h2.length);
       } else {
           System.arraycopy(h2, 0, concatenated, 0, h2.length);
           System.arraycopy(h1, 0, concatenated, h2.length, h1.length);
       }
       MessageDigest messageDigest = newMessageDigest();
```

```
        messageDigest.update(concatenated);

        return messageDigest.digest();
    }

    /**
     * Starting with the provided {@code leafHash} combined with the provided
{@code internalHashes}
     * pairwise until only the root hash remains.
     *
     * @param internalHashes
     *              Internal hashes of Merkle tree.
     * @param leafHash
     *              Leaf hashes of Merkle tree.
     * @return the root hash.
     */
    private static byte[] calculateRootHashFromInternalHashes(final
List<byte[]> internalHashes, final byte[] leafHash) {
        return internalHashes.stream().reduce(leafHash, Verifier::dot);
    }

    /**
     * Flip a single random bit in the given byte array. This method is used
to demonstrate
     * QLDB's verification features.
     *
     * @param original
     *              The original byte array.
     * @return the altered byte array with a single random bit changed.
     */
    public static byte[] flipRandomBit(final byte[] original) {
        if (original.length == 0) {
            throw new IllegalArgumentException("Array cannot be empty!");
        }
        int alteredPosition =
ThreadLocalRandom.current().nextInt(original.length);
        int b = ThreadLocalRandom.current().nextInt(UPPER_BOUND);
        byte[] altered = new byte[original.length];
        System.arraycopy(original, 0, altered, 0, original.length);
        altered[alteredPosition] = (byte) (altered[alteredPosition] ^ (1 <<
b));
        return altered;
    }
```

```
    public static String toBase64(byte[] arr) {
        return new String(Base64.encode(arr), StandardCharsets.UTF_8);
    }

    /**
     * Convert a {@link ByteBuffer} into byte array.
     *
     * @param buffer
     *              The {@link ByteBuffer} to convert.
     * @return the converted byte array.
     */
    public static byte[] convertByteBufferToByteArray(final ByteBuffer buffer)
{
        byte[] arr = new byte[buffer.remaining()];
        buffer.get(arr);
        return arr;
    }

    /**
     * Calculates the root hash from a list of hashes that represent the base
of a Merkle tree.
     *
     * @param hashes
     *              The list of byte arrays representing hashes making up base
of a Merkle tree.
     * @return a byte array that is the root hash of the given list of hashes.
     */
    public static byte[] calculateMerkleTreeRootHash(List<byte[]> hashes) {
        if (hashes.isEmpty()) {
            return new byte[0];
        }

        List<byte[]> remaining = combineLeafHashes(hashes);
        while (remaining.size() > 1) {
            remaining = combineLeafHashes(remaining);
        }
        return remaining.get(0);
    }

    private static List<byte[]> combineLeafHashes(List<byte[]> hashes) {
        List<byte[]> combinedHashes = new ArrayList<>();
        Iterator<byte[]> it = hashes.stream().iterator();

        while (it.hasNext()) {
```

```
            byte[] left = it.next();
            if (it.hasNext()) {
                byte[] right = it.next();
                byte[] combined = dot(left, right);
                combinedHashes.add(combined);
            } else {
                combinedHashes.add(left);
            }
        }

        return combinedHashes;
    }
}
```

1.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 copy of this
 * software and associated documentation files (the "Software"), to deal in
 the Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR
 A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */
```

```
package software.amazon.qldb.tutorial;

import com.amazonaws.util.Base64;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.tutorial.qldb.Proof;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.*;
import java.util.concurrent.ThreadLocalRandom;

/**
 * Encapsulates the logic to verify the integrity of revisions or blocks in a
 QLDB ledger.
 *
 * The main entry point is {@link #verify(byte[], byte[], String)}.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class Verifier {
    public static final Logger log = LoggerFactory.getLogger(Verifier.class);
    private static final int HASH_LENGTH = 32;
    private static final int UPPER_BOUND = 8;

    /**
     * Compares two hashes by their <em>signed</em> byte values in little-
endian order.
     */
    private static Comparator<byte[]> hashComparator = (h1, h2) -> {
        if (h1.length != HASH_LENGTH || h2.length != HASH_LENGTH) {
            throw new IllegalArgumentException("Invalid hash.");
        }
        for (int i = h1.length - 1; i >= 0; i--) {
            int byteEqual = Byte.compare(h1[i], h2[i]);
            if (byteEqual != 0) {
                return byteEqual;
            }
        }
```

```
        return 0;
    };

    private Verifier() { }

    /**
     * Verify the integrity of a document with respect to a QLDB ledger
digest.
     *
     * The verification algorithm includes the following steps:
     *
     * 1. {@link #buildCandidateDigest(Proof, byte[])} build the candidate
digest from the internal hashes
     * in the {@link Proof}.
     * 2. Check that the {@code candidateLedgerDigest} is equal to the {@code
ledgerDigest}.
     *
     * @param documentHash
     *              The hash of the document to be verified.
     * @param digest
     *              The QLDB ledger digest. This digest should have been
retrieved using
     *              {@link com.amazonaws.services.qldb.AmazonQLDB#getDigest}
     * @param proofBlob
     *              The ion encoded bytes representing the {@link Proof}
associated with the supplied
     *              {@code digestTipAddress} and {@code address} retrieved
using
     *              {@link
com.amazonaws.services.qldb.AmazonQLDB#getRevision}.
     * @return {@code true} if the record is verified or {@code false} if it
is not verified.
     */
    public static boolean verify(
            final byte[] documentHash,
            final byte[] digest,
            final String proofBlob
    ) {
        Proof proof = Proof.fromBlob(proofBlob);

        byte[] candidateDigest = buildCandidateDigest(proof, documentHash);

        return Arrays.equals(digest, candidateDigest);
    }
```

```
   /**
    * Build the candidate digest representing the entire ledger from the
internal hashes of the {@link Proof}.
    *
    * @param proof
    *              A Java representation of {@link Proof}
    *              returned from {@link
com.amazonaws.services.qldb.AmazonQLDB#getRevision}.
    * @param leafHash
    *              Leaf hash to build the candidate digest with.
    * @return a byte array of the candidate digest.
    */
   private static byte[] buildCandidateDigest(final Proof proof, final byte[]
leafHash) {
       return calculateRootHashFromInternalHashes(proof.getInternalHashes(),
leafHash);
   }

   /**
    * Get a new instance of {@link MessageDigest} using the SHA-256
algorithm.
    *
    * @return an instance of {@link MessageDigest}.
    * @throws IllegalStateException if the algorithm is not available on the
current JVM.
    */
   static MessageDigest newMessageDigest() {
       try {
           return MessageDigest.getInstance("SHA-256");
       } catch (NoSuchAlgorithmException e) {
           log.error("Failed to create SHA-256 MessageDigest", e);
           throw new IllegalStateException("SHA-256 message digest is
unavailable", e);
       }
   }

   /**
    * Takes two hashes, sorts them, concatenates them, and then returns the
    * hash of the concatenated array.
    *
    * @param h1
    *              Byte array containing one of the hashes to compare.
    * @param h2
```

```
     *              Byte array containing one of the hashes to compare.
     * @return the concatenated array of hashes.
     */
    public static byte[] dot(final byte[] h1, final byte[] h2) {
        if (h1.length == 0) {
            return h2;
        }
        if (h2.length == 0) {
            return h1;
        }
        byte[] concatenated = new byte[h1.length + h2.length];
        if (hashComparator.compare(h1, h2) < 0) {
            System.arraycopy(h1, 0, concatenated, 0, h1.length);
            System.arraycopy(h2, 0, concatenated, h1.length, h2.length);
        } else {
            System.arraycopy(h2, 0, concatenated, 0, h2.length);
            System.arraycopy(h1, 0, concatenated, h2.length, h1.length);
        }
        MessageDigest messageDigest = newMessageDigest();
        messageDigest.update(concatenated);

        return messageDigest.digest();
    }

    /**
     * Starting with the provided {@code leafHash} combined with the provided
{@code internalHashes}
     * pairwise until only the root hash remains.
     *
     * @param internalHashes
     *              Internal hashes of Merkle tree.
     * @param leafHash
     *              Leaf hashes of Merkle tree.
     * @return the root hash.
     */
    private static byte[] calculateRootHashFromInternalHashes(final
List<byte[]> internalHashes, final byte[] leafHash) {
        return internalHashes.stream().reduce(leafHash, Verifier::dot);
    }

    /**
     * Flip a single random bit in the given byte array. This method is used
to demonstrate
     * QLDB's verification features.
```

```
     *
     * @param original
     *              The original byte array.
     * @return the altered byte array with a single random bit changed.
     */
    public static byte[] flipRandomBit(final byte[] original) {
        if (original.length == 0) {
            throw new IllegalArgumentException("Array cannot be empty!");
        }
        int alteredPosition =
ThreadLocalRandom.current().nextInt(original.length);
        int b = ThreadLocalRandom.current().nextInt(UPPER_BOUND);
        byte[] altered = new byte[original.length];
        System.arraycopy(original, 0, altered, 0, original.length);
        altered[alteredPosition] = (byte) (altered[alteredPosition] ^ (1 <<
b));
        return altered;
    }

    public static String toBase64(byte[] arr) {
        return new String(Base64.encode(arr), StandardCharsets.UTF_8);
    }

    /**
     * Convert a {@link ByteBuffer} into byte array.
     *
     * @param buffer
     *              The {@link ByteBuffer} to convert.
     * @return the converted byte array.
     */
    public static byte[] convertByteBufferToByteArray(final ByteBuffer buffer)
{
        byte[] arr = new byte[buffer.remaining()];
        buffer.get(arr);
        return arr;
    }

    /**
     * Calculates the root hash from a list of hashes that represent the base
of a Merkle tree.
     *
     * @param hashes
     *              The list of byte arrays representing hashes making up base
of a Merkle tree.
```

```
     * @return a byte array that is the root hash of the given list of hashes.
     */
    public static byte[] calculateMerkleTreeRootHash(List<byte[]> hashes) {
        if (hashes.isEmpty()) {
            return new byte[0];
        }

        List<byte[]> remaining = combineLeafHashes(hashes);
        while (remaining.size() > 1) {
            remaining = combineLeafHashes(remaining);
        }
        return remaining.get(0);
    }

    private static List<byte[]> combineLeafHashes(List<byte[]> hashes) {
        List<byte[]> combinedHashes = new ArrayList<>();
        Iterator<byte[]> it = hashes.stream().iterator();

        while (it.hasNext()) {
            byte[] left = it.next();
            if (it.hasNext()) {
                byte[] right = it.next();
                byte[] combined = dot(left, right);
                combinedHashes.add(combined);
            } else {
                combinedHashes.add(left);
            }
        }

        return combinedHashes;
    }
}
```

2. 두 개의 .java 파일(GetDigest.java 및 GetRevision.java)을 사용하여 다음 단계를 수행하세요.

   - vehicle-registration 원장에 새 다이제스트를 요청하세요.
   - VehicleRegistration 테이블에 있는 문서의 각 개정에 대한 증거를 요청합니다.
   - 반환된 다이제스트와 증거를 사용하여 다이제스트를 다시 계산하여 개정 내용을 검증합니다.

   GetDigest.java 프로그램에는 다음 코드가 포함되어 있습니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.model.GetDigestRequest;
import com.amazonaws.services.qldb.model.GetDigestResult;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.tutorial.qldb.QldbStringUtils;

/**
 * This is an example for retrieving the digest of a particular ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class GetDigest {
```

```
    public static final Logger log = LoggerFactory.getLogger(GetDigest.class);
    public static AmazonQLDB client = CreateLedger.getClient();

    private GetDigest() { }

    /**
     * Calls {@link #getDigest(String)} for a ledger.
     *
     * @param args
     *              Arbitrary command-line arguments.
     * @throws Exception if failed to get a ledger digest.
     */
    public static void main(final String... args) throws Exception {
        try {

            getDigest(Constants.LEDGER_NAME);

        } catch (Exception e) {
            log.error("Unable to get a ledger digest!", e);
            throw e;
        }
    }

    /**
     * Get the digest for the specified ledger.
     *
     * @param ledgerName
     *              The ledger to get digest from.
     * @return {@link GetDigestResult}.
     */
    public static GetDigestResult getDigest(final String ledgerName) {
        log.info("Let's get the current digest of the ledger named {}.",
 ledgerName);
        GetDigestRequest request = new GetDigestRequest()
                .withName(ledgerName);
        GetDigestResult result = client.getDigest(request);
        log.info("Success. LedgerDigest: {}.",
 QldbStringUtils.toUnredactedString(result));
        return result;
    }
}
```

> **Note**
>
> `getDigest` 메서드를 사용하여 원장에 있는 저널의 현재 팁을 포함하는 다이제스트를 요청합니다. 저널 팁은 QLDB가 요청을 수신한 시점을 기준으로 가장 최근에 커밋된 블록을 나타냅니다.

`GetRevision.java` 프로그램에는 다음 코드가 포함되어 있습니다.

2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonReader;
import com.amazon.ion.IonStruct;
```

```
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.IonWriter;
import com.amazon.ion.system.IonReaderBuilder;
import com.amazon.ion.system.IonSystemBuilder;
import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.model.GetDigestResult;
import com.amazonaws.services.qldb.model.GetRevisionRequest;
import com.amazonaws.services.qldb.model.GetRevisionResult;
import com.amazonaws.services.qldb.model.ValueHolder;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.QldbDriver;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.qldb.BlockAddress;
import software.amazon.qldb.tutorial.qldb.QldbRevision;
import software.amazon.qldb.tutorial.qldb.QldbStringUtils;

/**
 * Verify the integrity of a document revision in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class GetRevision {
    public static final Logger log = LoggerFactory.getLogger(GetRevision.class);
    public static AmazonQLDB client = CreateLedger.getClient();
    private static final IonSystem SYSTEM = IonSystemBuilder.standard().build();

    private GetRevision() { }

    public static void main(String... args) throws Exception {

        final String vin = SampleData.REGISTRATIONS.get(0).getVin();
```

```
        verifyRegistration(ConnectToLedger.getDriver(), Constants.LEDGER_NAME,
 vin);
    }

    /**
     * Verify each version of the registration for the given VIN.
     *
     * @param driver
     *              A QLDB driver.
     * @param ledgerName
     *              The ledger to get digest from.
     * @param vin
     *              VIN to query the revision history of a specific registration
 with.
     * @throws Exception if failed to verify digests.
     * @throws AssertionError if document revision verification failed.
     */
    public static void verifyRegistration(final QldbDriver driver, final String
 ledgerName, final String vin)
            throws Exception {
        log.info(String.format("Let's verify the registration with VIN=%s, in
 ledger=%s.", vin, ledgerName));

        try {
            log.info("First, let's get a digest.");
            GetDigestResult digestResult = GetDigest.getDigest(ledgerName);

            ValueHolder digestTipAddress = digestResult.getDigestTipAddress();
            byte[] digestBytes =
 Verifier.convertByteBufferToByteArray(digestResult.getDigest());

            log.info("Got a ledger digest. Digest end address={}, digest={}.",
                QldbStringUtils.toUnredactedString(digestTipAddress),
                Verifier.toBase64(digestBytes));

            log.info(String.format("Next, let's query the registration with VIN=
%s. "
                    + "Then we can verify each version of the registration.",
 vin));
            List<IonStruct> documentsWithMetadataList = new ArrayList<>();
            driver.execute(txn -> {
                documentsWithMetadataList.addAll(queryRegistrationsByVin(txn,
 vin));
            });
```

```
            log.info("Registrations queried successfully!");

            log.info(String.format("Found %s revisions of the registration with
VIN=%s.",
                    documentsWithMetadataList.size(), vin));

            for (IonStruct ionStruct : documentsWithMetadataList) {

                QldbRevision document = QldbRevision.fromIon(ionStruct);
                log.info(String.format("Let's verify the document: %s",
document));

                log.info("Let's get a proof for the document.");
                GetRevisionResult proofResult = getRevision(
                        ledgerName,
                        document.getMetadata().getId(),
                        digestTipAddress,
                        document.getBlockAddress()
                );

                final IonValue proof =
Constants.MAPPER.writeValueAsIonValue(proofResult.getProof());
                final IonReader reader =
IonReaderBuilder.standard().build(proof);
                reader.next();
                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                IonWriter writer = SYSTEM.newBinaryWriter(baos);
                writer.writeValue(reader);
                writer.close();
                baos.flush();
                baos.close();
                byte[] byteProof = baos.toByteArray();

                log.info(String.format("Got back a proof: %s",
Verifier.toBase64(byteProof)));

                boolean verified = Verifier.verify(
                        document.getHash(),
                        digestBytes,
                        proofResult.getProof().getIonText()
                );

                if (!verified) {
```

```
                    throw new AssertionError("Document revision is not
verified!");
                } else {
                    log.info("Success! The document is verified");
                }

                byte[] alteredDigest = Verifier.flipRandomBit(digestBytes);
                log.info(String.format("Flipping one bit in the digest and
assert that the document is NOT verified. "
                        + "The altered digest is: %s",
Verifier.toBase64(alteredDigest)));
                verified = Verifier.verify(
                        document.getHash(),
                        alteredDigest,
                        proofResult.getProof().getIonText()
                );

                if (verified) {
                    throw new AssertionError("Expected document to not be
verified against altered digest.");
                } else {
                    log.info("Success! As expected flipping a bit in the digest
causes verification to fail.");
                }

                byte[] alteredDocumentHash =
Verifier.flipRandomBit(document.getHash());
                log.info(String.format("Flipping one bit in the document's hash
and assert that it is NOT verified. "
                        + "The altered document hash is: %s.",
Verifier.toBase64(alteredDocumentHash)));
                verified = Verifier.verify(
                        alteredDocumentHash,
                        digestBytes,
                        proofResult.getProof().getIonText()
                );

                if (verified) {
                    throw new AssertionError("Expected altered document hash to
not be verified against digest.");
                } else {
                    log.info("Success! As expected flipping a bit in the
document hash causes verification to fail.");
                }
```

```
            }

        } catch (Exception e) {
            log.error("Failed to verify digests.", e);
            throw e;
        }

        log.info(String.format("Finished verifying the registration with VIN=%s
in ledger=%s.", vin, ledgerName));
    }

    /**
     * Get the revision of a particular document specified by the given document
ID and block address.
     *
     * @param ledgerName
     *              Name of the ledger containing the document.
     * @param documentId
     *              Unique ID for the document to be verified, contained in the
committed view of the document.
     * @param digestTipAddress
     *              The latest block location covered by the digest.
     * @param blockAddress
     *              The location of the block to request.
     * @return the requested revision.
     */
    public static GetRevisionResult getRevision(final String ledgerName, final
String documentId,
                                                final ValueHolder
digestTipAddress, final BlockAddress blockAddress) {
        try {
            GetRevisionRequest request = new GetRevisionRequest()
                    .withName(ledgerName)
                    .withDigestTipAddress(digestTipAddress)
                    .withBlockAddress(new
ValueHolder().withIonText(Constants.MAPPER.writeValueAsIonValue(blockAddress)
                            .toString()))
                    .withDocumentId(documentId);
            return client.getRevision(request);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }
```

```
    /**
     * Query the registration history for the given VIN.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              The unique VIN to query.
     * @return a list of {@link IonStruct} representing the registration
 history.
     * @throws IllegalStateException if failed to convert parameters into {@link
 IonValue}
     */
    public static List<IonStruct> queryRegistrationsByVin(final
 TransactionExecutor txn, final String vin) {
        log.info(String.format("Let's query the 'VehicleRegistration' table for
 VIN: %s...", vin));
        log.info("Let's query the 'VehicleRegistration' table for VIN: {}...",
 vin);
        final String query = String.format("SELECT * FROM _ql_committed_%s WHERE
 data.VIN = ?",
                Constants.VEHICLE_REGISTRATION_TABLE_NAME);
        try {
            final List<IonValue> parameters =
 Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
            final Result result = txn.execute(query, parameters);
            List<IonStruct> list = ScanTable.toIonStructs(result);
            log.info(String.format("Found %d document(s)!", list.size()));
            return list;
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }
}
```

1.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
```

```
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonReader;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonValue;
import com.amazon.ion.IonWriter;
import com.amazon.ion.system.IonReaderBuilder;
import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.model.GetDigestResult;
import com.amazonaws.services.qldb.model.GetRevisionRequest;
import com.amazonaws.services.qldb.model.GetRevisionResult;
import com.amazonaws.services.qldb.model.ValueHolder;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;
import software.amazon.qldb.tutorial.model.SampleData;
import software.amazon.qldb.tutorial.qldb.BlockAddress;
import software.amazon.qldb.tutorial.qldb.QldbRevision;
import software.amazon.qldb.tutorial.qldb.QldbStringUtils;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
```

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * Verify the integrity of a document revision in a QLDB ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class GetRevision {
    public static final Logger log = LoggerFactory.getLogger(GetRevision.class);
    public static AmazonQLDB client = CreateLedger.getClient();

    private GetRevision() { }

    public static void main(String... args) throws Exception {

        final String vin = SampleData.REGISTRATIONS.get(0).getVin();

        try (QldbSession qldbSession = ConnectToLedger.createQldbSession()) {
            verifyRegistration(qldbSession, Constants.LEDGER_NAME, vin);
        }
    }

    /**
     * Verify each version of the registration for the given VIN.
     *
     * @param qldbSession
     *              A QLDB session.
     * @param ledgerName
     *              The ledger to get digest from.
     * @param vin
     *              VIN to query the revision history of a specific registration
 with.
     * @throws Exception if failed to verify digests.
     * @throws AssertionError if document revision verification failed.
     */
    public static void verifyRegistration(final QldbSession qldbSession, final
 String ledgerName, final String vin)
            throws Exception {
        log.info(String.format("Let's verify the registration with VIN=%s, in
 ledger=%s.", vin, ledgerName));
```

```
        try {
            log.info("First, let's get a digest.");
            GetDigestResult digestResult = GetDigest.getDigest(ledgerName);

            ValueHolder digestTipAddress = digestResult.getDigestTipAddress();
            byte[] digestBytes =
 Verifier.convertByteBufferToByteArray(digestResult.getDigest());

            log.info("Got a ledger digest. Digest end address={}, digest={}.",
                QldbStringUtils.toUnredactedString(digestTipAddress),
                Verifier.toBase64(digestBytes));

            log.info(String.format("Next, let's query the registration with VIN=
%s. "
                    + "Then we can verify each version of the registration.",
 vin));
            List<IonStruct> documentsWithMetadataList = new ArrayList<>();
            qldbSession.execute(txn -> {
                documentsWithMetadataList.addAll(queryRegistrationsByVin(txn,
 vin));
            }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
            log.info("Registrations queried successfully!");

            log.info(String.format("Found %s revisions of the registration with
 VIN=%s.",
                    documentsWithMetadataList.size(), vin));

            for (IonStruct ionStruct : documentsWithMetadataList) {

                QldbRevision document = QldbRevision.fromIon(ionStruct);
                log.info(String.format("Let's verify the document: %s",
 document));

                log.info("Let's get a proof for the document.");
                GetRevisionResult proofResult = getRevision(
                        ledgerName,
                        document.getMetadata().getId(),
                        digestTipAddress,
                        document.getBlockAddress()
                );

                final IonValue proof =
 Constants.MAPPER.writeValueAsIonValue(proofResult.getProof());
```

```
                final IonReader reader =
IonReaderBuilder.standard().build(proof);
                reader.next();
                ByteArrayOutputStream baos = new ByteArrayOutputStream();
                IonWriter writer = Constants.SYSTEM.newBinaryWriter(baos);
                writer.writeValue(reader);
                writer.close();
                baos.flush();
                baos.close();
                byte[] byteProof = baos.toByteArray();

                log.info(String.format("Got back a proof: %s",
Verifier.toBase64(byteProof)));

                boolean verified = Verifier.verify(
                        document.getHash(),
                        digestBytes,
                        proofResult.getProof().getIonText()
                );

                if (!verified) {
                    throw new AssertionError("Document revision is not
verified!");
                } else {
                    log.info("Success! The document is verified");
                }

                byte[] alteredDigest = Verifier.flipRandomBit(digestBytes);
                log.info(String.format("Flipping one bit in the digest and
assert that the document is NOT verified. "
                        + "The altered digest is: %s",
Verifier.toBase64(alteredDigest)));
                verified = Verifier.verify(
                        document.getHash(),
                        alteredDigest,
                        proofResult.getProof().getIonText()
                );

                if (verified) {
                    throw new AssertionError("Expected document to not be
verified against altered digest.");
                } else {
                    log.info("Success! As expected flipping a bit in the digest
causes verification to fail.");
```

```
                }

                byte[] alteredDocumentHash =
Verifier.flipRandomBit(document.getHash());
                log.info(String.format("Flipping one bit in the document's hash
and assert that it is NOT verified. "
                        + "The altered document hash is: %s.",
Verifier.toBase64(alteredDocumentHash)));
                verified = Verifier.verify(
                        alteredDocumentHash,
                        digestBytes,
                        proofResult.getProof().getIonText()
                );

                if (verified) {
                    throw new AssertionError("Expected altered document hash to
not be verified against digest.");
                } else {
                    log.info("Success! As expected flipping a bit in the
document hash causes verification to fail.");
                }
            }

        } catch (Exception e) {
            log.error("Failed to verify digests.", e);
            throw e;
        }

        log.info(String.format("Finished verifying the registration with VIN=%s
in ledger=%s.", vin, ledgerName));
    }

    /**
     * Get the revision of a particular document specified by the given document
ID and block address.
     *
     * @param ledgerName
     *              Name of the ledger containing the document.
     * @param documentId
     *              Unique ID for the document to be verified, contained in the
committed view of the document.
     * @param digestTipAddress
     *              The latest block location covered by the digest.
     * @param blockAddress
```

```
     *              The location of the block to request.
     * @return the requested revision.
     */
    public static GetRevisionResult getRevision(final String ledgerName, final
String documentId,
                                                final ValueHolder
digestTipAddress, final BlockAddress blockAddress) {
        try {
            GetRevisionRequest request = new GetRevisionRequest()
                    .withName(ledgerName)
                    .withDigestTipAddress(digestTipAddress)
                    .withBlockAddress(new
ValueHolder().withIonText(Constants.MAPPER.writeValueAsIonValue(blockAddress)
                            .toString()))
                    .withDocumentId(documentId);
            return client.getRevision(request);
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }

    /**
     * Query the registration history for the given VIN.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param vin
     *              The unique VIN to query.
     * @return a list of {@link IonStruct} representing the registration
history.
     * @throws IllegalStateException if failed to convert parameters into {@link
IonValue}
     */
    public static List<IonStruct> queryRegistrationsByVin(final
TransactionExecutor txn, final String vin) {
        log.info(String.format("Let's query the 'VehicleRegistration' table for
VIN: %s...", vin));
        log.info("Let's query the 'VehicleRegistration' table for VIN: {}...",
vin);
        final String query = String.format("SELECT * FROM _ql_committed_%s WHERE
data.VIN = ?",
                Constants.VEHICLE_REGISTRATION_TABLE_NAME);
        try {
```

```
            final List<IonValue> parameters =
 Collections.singletonList(Constants.MAPPER.writeValueAsIonValue(vin));
            final Result result = txn.execute(query, parameters);
            List<IonStruct> list = ScanTable.toIonStructs(result);
            log.info(String.format("Found %d document(s)!", list.size()));
            return list;
        } catch (IOException ioe) {
            throw new IllegalStateException(ioe);
        }
    }
}
```

> **Note**
>
> `getRevision` 메서드가 지정된 문서 개정에 대한 증거를 반환하면 이 프로그램은 클라이언트 측 API를 사용하여 해당 개정을 검증합니다. 이 API에서 사용하는 알고리즘에 대한 개요는 증명을 사용하여 다이제스트를 다시 계산하기 섹션을 참조하세요.

3. `GetRevision.java` 프로그램을 컴파일하고 실행하여 VIN `1N4AL11D75C109151`으로 `VehicleRegistration` 문서를 암호학적으로 검증합니다.

`vehicle-registration` 원장의 저널 데이터를 내보내고 검증하려면 8단계: 원장의 저널 데이터 내보내기 및 검증로 이동하세요.

8단계: 원장의 저널 데이터 내보내기 및 검증

Amazon QLDB에서는 데이터 보존, 분석 및 감사와 같은 다양한 목적으로 원장의 저널 콘텐츠에 액세스할 수 있습니다. 자세한 내용은 Amazon QLDB에서 저널 데이터 내보내기 섹션을 참조하세요.

이 단계에서는 `vehicle-registration` 원장의 저널 블록을 Amazon S3 버킷으로 내보냅니다. 그런 다음 내보낸 데이터를 사용하여 저널 블록과 각 블록 내 개별 해시 구성 요소 간의 해시 체인을 검증합니다.

사용하는 AWS Identity and Access Management(IAM) 주체 엔터티에는 AWS 계정에서 Amazon S3 버킷을 생성할 수 있는 충분한 IAM 권한이 있어야 합니다. 자세한 내용은 Amazon S3 사용 설명서의 Amazon S3의 정책 및 권한을 참조하세요. 또한 QLDB가 Amazon S3 버킷에 객체를 쓸 수 있도록 허용하는 권한 정책이 첨부된 IAM 역할을 생성할 권한이 있어야 합니다. 자세한 내용은 IAM 사용 설명서에서 IAM 리소스에 액세스하는 데 필요한 권한을 참조하세요.

저널 데이터를 내보내고 검증하려면

1. 저널 블록과 해당 데이터 내용을 나타내는 다음 파일(JournalBlock.java)을 검토하세요. 여기에는 블록 해시의 각 개별 구성 요소를 계산하는 방법을 보여주는 verifyBlockHash()라는 이름의 메서드가 포함되어 있습니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial.qldb;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import com.fasterxml.jackson.dataformat.ion.IonTimestampSerializers;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.tutorial.Constants;
import software.amazon.qldb.tutorial.Verifier;

import java.io.IOException;
```

```
import java.nio.ByteBuffer;
import java.util.Arrays;
import java.util.Date;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

import static java.nio.ByteBuffer.wrap;

/**
 * Represents a JournalBlock that was recorded after executing a transaction
 * in the ledger.
 */
public final class JournalBlock {
    private static final Logger log = LoggerFactory.getLogger(JournalBlock.class);

    private BlockAddress blockAddress;
    private String transactionId;
    @JsonSerialize(using =
 IonTimestampSerializers.IonTimestampJavaDateSerializer.class)
    private Date blockTimestamp;
    private byte[] blockHash;
    private byte[] entriesHash;
    private byte[] previousBlockHash;
    private byte[][] entriesHashList;
    private TransactionInfo transactionInfo;
    private RedactionInfo redactionInfo;
    private List<QldbRevision> revisions;

    @JsonCreator
    public JournalBlock(@JsonProperty("blockAddress") final BlockAddress
 blockAddress,
                        @JsonProperty("transactionId") final String transactionId,
                        @JsonProperty("blockTimestamp") final Date blockTimestamp,
                        @JsonProperty("blockHash") final byte[] blockHash,
                        @JsonProperty("entriesHash") final byte[] entriesHash,
                        @JsonProperty("previousBlockHash") final byte[]
 previousBlockHash,
                        @JsonProperty("entriesHashList") final byte[][]
 entriesHashList,
                        @JsonProperty("transactionInfo") final TransactionInfo
 transactionInfo,
```

```
                        @JsonProperty("redactionInfo") final RedactionInfo
redactionInfo,
                        @JsonProperty("revisions") final List<QldbRevision>
revisions) {
        this.blockAddress = blockAddress;
        this.transactionId = transactionId;
        this.blockTimestamp = blockTimestamp;
        this.blockHash = blockHash;
        this.entriesHash = entriesHash;
        this.previousBlockHash = previousBlockHash;
        this.entriesHashList = entriesHashList;
        this.transactionInfo = transactionInfo;
        this.redactionInfo = redactionInfo;
        this.revisions = revisions;
    }

    public BlockAddress getBlockAddress() {
        return blockAddress;
    }

    public String getTransactionId() {
        return transactionId;
    }

    public Date getBlockTimestamp() {
        return blockTimestamp;
    }

    public byte[][] getEntriesHashList() {
        return entriesHashList;
    }

    public TransactionInfo getTransactionInfo() {
        return transactionInfo;
    }

    public RedactionInfo getRedactionInfo() {
        return redactionInfo;
    }

    public List<QldbRevision> getRevisions() {
        return revisions;
    }
```

```
    public byte[] getEntriesHash() {
        return entriesHash;
    }

    public byte[] getBlockHash() {
        return blockHash;
    }

    public byte[] getPreviousBlockHash() {
        return previousBlockHash;
    }

    @Override
    public String toString() {
        return "JournalBlock{"
                + "blockAddress=" + blockAddress
                + ", transactionId='" + transactionId + '\''
                + ", blockTimestamp=" + blockTimestamp
                + ", blockHash=" + Arrays.toString(blockHash)
                + ", entriesHash=" + Arrays.toString(entriesHash)
                + ", previousBlockHash=" + Arrays.toString(previousBlockHash)
                + ", entriesHashList=" + Arrays.toString(entriesHashList)
                + ", transactionInfo=" + transactionInfo
                + ", redactionInfo=" + redactionInfo
                + ", revisions=" + revisions
                + '}';
    }

    @Override
    public boolean equals(final Object o) {
        if (this == o) {
            return true;
        }
        if (!(o instanceof JournalBlock)) {
            return false;
        }

        final JournalBlock that = (JournalBlock) o;

        if (!getBlockAddress().equals(that.getBlockAddress())) {
            return false;
        }
        if (!getTransactionId().equals(that.getTransactionId())) {
            return false;
```

```
        }
        if (!getBlockTimestamp().equals(that.getBlockTimestamp())) {
            return false;
        }
        if (!Arrays.equals(getBlockHash(), that.getBlockHash())) {
            return false;
        }
        if (!Arrays.equals(getEntriesHash(), that.getEntriesHash())) {
            return false;
        }
        if (!Arrays.equals(getPreviousBlockHash(), that.getPreviousBlockHash())) {
            return false;
        }
        if (!Arrays.deepEquals(getEntriesHashList(), that.getEntriesHashList())) {
            return false;
        }
        if (!getTransactionInfo().equals(that.getTransactionInfo())) {
            return false;
        }
        if (getRedactionInfo() != null ? !
getRedactionInfo().equals(that.getRedactionInfo()) : that.getRedactionInfo() !=
 null) {
            return false;
        }
        return getRevisions() != null ?
 getRevisions().equals(that.getRevisions()) : that.getRevisions() == null;
    }

    @Override
    public int hashCode() {
        int result = getBlockAddress().hashCode();
        result = 31 * result + getTransactionId().hashCode();
        result = 31 * result + getBlockTimestamp().hashCode();
        result = 31 * result + Arrays.hashCode(getBlockHash());
        result = 31 * result + Arrays.hashCode(getEntriesHash());
        result = 31 * result + Arrays.hashCode(getPreviousBlockHash());
        result = 31 * result + Arrays.deepHashCode(getEntriesHashList());
        result = 31 * result + getTransactionInfo().hashCode();
        result = 31 * result + (getRedactionInfo() != null ?
 getRedactionInfo().hashCode() : 0);
        result = 31 * result + (getRevisions() != null ?
 getRevisions().hashCode() : 0);
        return result;
    }
```

```
   /**
    * This method validates that the hashes of the components of a journal block
make up the block
    * hash that is provided with the block itself.
    *
    * The components that contribute to the hash of the journal block consist of
the following:
    *   - user transaction information (contained in [transactionInfo])
    *   - user redaction information (contained in [redactionInfo])
    *   - user revisions (contained in [revisions])
    *   - hashes of internal-only system metadata (contained in [revisions] and in
[entriesHashList])
    *   - the previous block hash
    *
    * If any of the computed hashes of user information cannot be validated or any
of the system
    * hashes do not result in the correct computed values, this method will throw
an IllegalArgumentException.
    *
    * Internal-only system metadata is represented by its hash, and can be present
in the form of certain
    * items in the [revisions] list that only contain a hash and no user data, as
well as some hashes
    * in [entriesHashList].
    *
    * To validate that the hashes of the user data are valid components of the
[blockHash], this method
    * performs the following steps:
    *
    * 1. Compute the hash of the [transactionInfo] and validate that it is
included in the [entriesHashList].
    * 2. Compute the hash of the [redactionInfo], if present, and validate that it
is included in the [entriesHashList].
    * 3. Validate the hash of each user revision was correctly computed and
matches the hash published
    * with that revision.
    * 4. Compute the hash of the [revisions] by treating the revision hashes as
the leaf nodes of a Merkle tree
    * and calculating the root hash of that tree. Then validate that hash is
included in the [entriesHashList].
    * 5. Compute the hash of the [entriesHashList] by treating the hashes as the
leaf nodes of a Merkle tree
```

```
     * and calculating the root hash of that tree. Then validate that hash matches
 [entriesHash].
     * 6. Finally, compute the block hash by computing the hash resulting from
 concatenating the [entriesHash]
     * and previous block hash, and validate that the result matches the
 [blockHash] provided by QLDB with the block.
     *
     * This method is called by ValidateQldbHashChain::verify for each journal
 block to validate its
     * contents before verifying that the hash chain between consecutive blocks is
 correct.
     */
    public void verifyBlockHash() {
        Set<ByteBuffer> entriesHashSet = new HashSet<>();
        Arrays.stream(entriesHashList).forEach(hash ->
 entriesHashSet.add(wrap(hash).asReadOnlyBuffer()));

        byte[] computedTransactionInfoHash = computeTransactionInfoHash();
        if (!
entriesHashSet.contains(wrap(computedTransactionInfoHash).asReadOnlyBuffer())) {
            throw new IllegalArgumentException(
                    "Block transactionInfo hash is not contained in the QLDB block
 entries hash list.");
        }

        if (redactionInfo != null) {
            byte[] computedRedactionInfoHash = computeRedactionInfoHash();
            if (!
entriesHashSet.contains(wrap(computedRedactionInfoHash).asReadOnlyBuffer())) {
                throw new IllegalArgumentException(
                        "Block redactionInfo hash is not contained in the QLDB
 block entries hash list.");
            }
        }

        if (revisions != null) {
            revisions.forEach(QldbRevision::verifyRevisionHash);
            byte[] computedRevisionsHash = computeRevisionsHash();
            if (!
entriesHashSet.contains(wrap(computedRevisionsHash).asReadOnlyBuffer())) {
                throw new IllegalArgumentException(
                        "Block revisions list hash is not contained in the QLDB
 block entries hash list.");
            }
```

```
        }

        byte[] computedEntriesHash = computeEntriesHash();
        if (!Arrays.equals(computedEntriesHash, entriesHash)) {
            throw new IllegalArgumentException("Computed entries hash does not
match entries hash provided in the block.");
        }

        byte[] computedBlockHash = Verifier.dot(computedEntriesHash,
previousBlockHash);
        if (!Arrays.equals(computedBlockHash, blockHash)) {
            throw new IllegalArgumentException("Computed block hash does not match
block hash provided in the block.");
        }
    }

    private byte[] computeTransactionInfoHash() {
        try {
            return
QldbIonUtils.hashIonValue(Constants.MAPPER.writeValueAsIonValue(transactionInfo));
        } catch (IOException e) {
            throw new IllegalArgumentException("Could not compute transactionInfo
hash to verify block hash.", e);
        }
    }

    private byte[] computeRedactionInfoHash() {
        try {
            return
QldbIonUtils.hashIonValue(Constants.MAPPER.writeValueAsIonValue(redactionInfo));
        } catch (IOException e) {
            throw new IllegalArgumentException("Could not compute redactionInfo
hash to verify block hash.", e);
        }
    }

    private byte[] computeRevisionsHash() {
        return
Verifier.calculateMerkleTreeRootHash(revisions.stream().map(QldbRevision::getHash).collect
    }

    private byte[] computeEntriesHash() {
        return
Verifier.calculateMerkleTreeRootHash(Arrays.asList(entriesHashList));
```

```
    }
}
```

2. 다음 프로그램(ValidateQldbHashChain.java)을 컴파일하고 실행하여 다음 단계를 수행합니다.

   1. vehicle-registration 원장의 저널 블록을 **qldb-tutorial-journal-export-*111122223333***라는 이름의 Amazon S3 버킷으로 내보냅니다(사용자의 AWS 계정 번호로 대체).
   2. verifyBlockHash()를 호출하여 각 블록 내의 개별 해시 구성 요소를 검증합니다.
   3. 저널 블록 간의 해시 체인을 검증합니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazonaws.services.qldb.model.ExportJournalToS3Result;
import com.amazonaws.services.qldb.model.S3EncryptionConfiguration;
```

```
import com.amazonaws.services.qldb.model.S3ObjectEncryptionType;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;

import java.time.Instant;
import java.util.Arrays;
import java.util.List;

import com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClientBuilder;
import com.amazonaws.services.securitytoken.model.GetCallerIdentityRequest;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import software.amazon.qldb.tutorial.qldb.JournalBlock;

/**
 * Validate the hash chain of a QLDB ledger by stepping through its S3 export.
 *
 * This code accepts an exportId as an argument, if exportId is passed the code
 * will use that or request QLDB to generate a new export to perform QLDB hash
 * chain validation.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class ValidateQldbHashChain {
    public static final Logger log =
 LoggerFactory.getLogger(ValidateQldbHashChain.class);
    private static final int TIME_SKEW = 20;

    private ValidateQldbHashChain() { }

    /**
     * Export journal contents to a S3 bucket.
     *
     * @return the ExportId of the journal export.
     * @throws InterruptedException if the thread is interrupted while waiting for
 export to complete.
     */
    private static String createExport() throws InterruptedException {
        String accountId = AWSSecurityTokenServiceClientBuilder.defaultClient()
            .getCallerIdentity(new GetCallerIdentityRequest()).getAccount();
        String bucketName = Constants.JOURNAL_EXPORT_S3_BUCKET_NAME_PREFIX + "-" +
 accountId;
```

```
        String prefix = Constants.LEDGER_NAME + "-" +
Instant.now().getEpochSecond() + "/";

        S3EncryptionConfiguration encryptionConfiguration = new
S3EncryptionConfiguration()
                .withObjectEncryptionType(S3ObjectEncryptionType.SSE_S3);
        ExportJournalToS3Result exportJournalToS3Result =

ExportJournal.createJournalExportAndAwaitCompletion(Constants.LEDGER_NAME,
                    bucketName, prefix, null, encryptionConfiguration,
ExportJournal.DEFAULT_EXPORT_TIMEOUT_MS);

        return exportJournalToS3Result.getExportId();
    }

    /**
     * Validates that the chain hash on the {@link JournalBlock} is valid.
     *
     * @param journalBlocks
     *              {@link JournalBlock} containing hashes to validate.
     * @throws IllegalStateException if previous block hash does not match.
     */
    public static void verify(final List<JournalBlock> journalBlocks) {
        if (journalBlocks.size() == 0) {
            return;
        }

        journalBlocks.stream().reduce(null, (previousJournalBlock, journalBlock) ->
{
            journalBlock.verifyBlockHash();
            if (previousJournalBlock == null) { return journalBlock; }
            if (!Arrays.equals(previousJournalBlock.getBlockHash(),
journalBlock.getPreviousBlockHash())) {
                throw new IllegalStateException("Previous block hash doesn't
match.");
            }
            byte[] blockHash = Verifier.dot(journalBlock.getEntriesHash(),
previousJournalBlock.getBlockHash());
            if (!Arrays.equals(blockHash, journalBlock.getBlockHash())) {
                throw new IllegalStateException("Block hash doesn't match
entriesHash dot previousBlockHash, the chain is "
                        + "broken.");
            }
            return journalBlock;
```

```
        });
    }

    public static void main(final String... args) throws InterruptedException {
        try {
            String exportId;
            if (args.length == 1) {
                exportId = args[0];
                log.info("Validating QLDB hash chain for exportId: " + exportId);
            } else {
                log.info("Requesting QLDB to create an export.");
                exportId = createExport();
            }
            List<JournalBlock> journalBlocks =

 JournalS3ExportReader.readExport(DescribeJournalExport.describeExport(Constants.LEDGER_NAM
                    exportId), AmazonS3ClientBuilder.defaultClient());
            verify(journalBlocks);
        } catch (Exception e) {
            log.error("Unable to perform hash chain verification.", e);
            throw e;
        }
    }

}
```

vehicle-registration 원장을 더 이상 사용할 필요가 없는 경우 9단계(선택 사항): 리소스 정리로 진행하십시오.

9단계(선택 사항): 리소스 정리

vehicle-registration 원장을 계속 사용할 수 있습니다. 그러나 더 이상 필요하지 않은 경우 삭제해야 합니다.

원장을 삭제하려면

1. 다음 프로그램(DeleteLedger.java)을 컴파일하고 실행하여 vehicle-registration 원장 및 모든 콘텐츠를 삭제합니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
```

```
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.model.DeleteLedgerRequest;
import com.amazonaws.services.qldb.model.DeleteLedgerResult;
import com.amazonaws.services.qldb.model.ResourceNotFoundException;
import com.amazonaws.services.qldb.model.UpdateLedgerRequest;
import com.amazonaws.services.qldb.model.UpdateLedgerResult;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Delete a ledger.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-
credentials.html
 */
public final class DeleteLedger {
    public static final Logger log = LoggerFactory.getLogger(DeleteLedger.class);
    public static final Long LEDGER_DELETION_POLL_PERIOD_MS = 20_000L;
```

```
    public static AmazonQLDB client = CreateLedger.getClient();

    private DeleteLedger() { }

    public static void main(String... args) throws Exception {
        try {
            setDeletionProtection(Constants.LEDGER_NAME, false);

            delete(Constants.LEDGER_NAME);

            waitForDeleted(Constants.LEDGER_NAME);

        } catch (Exception e) {
            log.error("Unable to delete the ledger.", e);
            throw e;
        }
    }

    /**
     * Send a request to the QLDB database to delete the specified ledger.
     *
     * @param ledgerName
     *              Name of the ledger to be deleted.
     * @return DeleteLedgerResult.
     */
    public static DeleteLedgerResult delete(final String ledgerName) {
        log.info("Attempting to delete the ledger with name: {}...", ledgerName);
        DeleteLedgerRequest request = new
DeleteLedgerRequest().withName(ledgerName);
        DeleteLedgerResult result = client.deleteLedger(request);
        log.info("Success.");
        return result;
    }

    /**
     * Wait for the ledger to be deleted.
     *
     * @param ledgerName
     *              Name of the ledger being deleted.
     * @throws InterruptedException if thread is being interrupted.
     */
    public static void waitForDeleted(final String ledgerName) throws
InterruptedException {
        log.info("Waiting for the ledger to be deleted...");
```

```
        while (true) {
            try {
                DescribeLedger.describe(ledgerName);
                log.info("The ledger is still being deleted. Please wait...");
                Thread.sleep(LEDGER_DELETION_POLL_PERIOD_MS);
            } catch (ResourceNotFoundException ex) {
                log.info("Success. The ledger is deleted.");
                break;
            }
        }
    }

    public static UpdateLedgerResult setDeletionProtection(String ledgerName,
 boolean deletionProtection) {
        log.info("Let's set deletionProtection to {} for the ledger with name {}",
 deletionProtection, ledgerName);
        UpdateLedgerRequest request = new UpdateLedgerRequest()
                .withName(ledgerName)
                .withDeletionProtection(deletionProtection);

        UpdateLedgerResult result = client.updateLedger(request);
        log.info("Success. Ledger updated: {}", result);
        return result;
    }
}
```

> **Note**
>
> 원장에 대해 삭제 방지가 활성화된 경우 QLDB API를 사용하여 원장을 삭제하기 전에 먼저 이를 비활성화해야 합니다.

2. 이전 단계에서 저널 데이터를 내보냈는데 더 이상 필요하지 않은 경우, Amazon S3 콘솔을 사용하여 S3 버킷을 삭제하세요.

   https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.

Amazon QLDB Node.js 자습서

이 자습서 샘플 애플리케이션 구현에서는 Node.js의 JavaScript용 AWS SDK와 함께 Amazon QLDB 드라이버를 사용하여 QLDB 원장을 생성하고 샘플 데이터로 채웁니다.

이 자습서로 학습하면서 관리 API 작업에 대해서 AWS SDK for JavaScript API 참조를 참조할 수 있습니다. 트랜잭션 데이터 작업의 경우 Node.js API용 QLDB 드라이버 참조를 참조할 수 있습니다.

> **Note**
>
> 해당하는 경우 일부 자습서 단계에는 지원되는 Node.js QLDB 드라이버의 각 주요 버전에 대해 서로 다른 명령 또는 코드 예제가 있습니다.

주제

Amazon QLDB Node.js 샘플 애플리케이션 설치

이 섹션에서는 단계별 Node.js 자습서를 위해 제공된 Amazon QLDB 샘플 애플리케이션을 설치하고 실행하는 방법을 설명합니다. 이 샘플 애플리케이션의 사용 사례는 차량 등록에 대한 전체 기록 정보를 추적하는 자동차 부서(DMV) 데이터베이스입니다.

Node.js용 DMV 샘플 애플리케이션은 GitHub 리포지토리 aws-samples/amazon-qldb-dmv-sample-nodejs의 오픈 소스입니다.

사전 조건

시작하기 전에 Node.js 사전 조건용 QLDB 드라이버를 완료했는지 확인합니다. 여기에는 Node.js 설치 및 다음 작업이 포함됩니다.

1. AWS에 가입합니다.

2. 적절한 QLDB 권한을 가진 사용자를 생성합니다.
3. 개발을 위한 프로그래밍 방식 액세스 권한을 부여합니다.

이 자습서의 모든 단계를 완료하려면 QLDB API를 통해 원장 리소스에 대한 전체 관리 액세스 권한이 필요합니다.

설치

샘플 애플리케이션을 설치하려면

1. 다음 명령을 입력하여 GitHub에서 샘플 애플리케이션을 복제합니다.

   2.x

   ```
   git clone https://github.com/aws-samples/amazon-qldb-dmv-sample-nodejs.git
   ```

   1.x

   ```
   git clone -b v1.0.0 https://github.com/aws-samples/amazon-qldb-dmv-sample-
   nodejs.git
   ```

   샘플 애플리케이션은 Node.js 드라이버 및 Node.js의 JavaScript용 AWS SDK를 포함하여 이 자습서의 전체 소스 코드와 해당 종속성을 패키징합니다. 이 애플리케이션은 TypeScript로 작성되었습니다.
2. `amazon-qldb-dmv-sample-nodejs` 패키지가 복제된 디렉토리로 전환합니다.

   ```
   cd amazon-qldb-dmv-sample-nodejs
   ```
3. 종속성을 클린 설치합니다.

   ```
   npm ci
   ```
4. 패키지를 트랜스파일합니다.

   ```
   npm run build
   ```

   트랜스파일된 JavaScript 파일은 `./dist` 디렉터리에 작성됩니다.
5. 1단계: 새 원장 생성로 진행하여 자습서를 시작하고 원장을 생성하십시오.

1단계: 새 원장 생성

이 단계에서는 `vehicle-registration`라는 이름의 새 Amazon QLDB 원장을 생성합니다

새 원장을 생성하려면

1. 이 자습서의 다른 모든 프로그램에서 사용하는 상수 값이 들어 있는 다음 파일(`Constants.ts`)을 검토하십시오.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

/**
 * Constant values used throughout this tutorial.
 */
export const LEDGER_NAME = "vehicle-registration";
export const LEDGER_NAME_WITH_TAGS = "tags";

export const DRIVERS_LICENSE_TABLE_NAME = "DriversLicense";
export const PERSON_TABLE_NAME = "Person";
export const VEHICLE_REGISTRATION_TABLE_NAME = "VehicleRegistration";
```

```
export const VEHICLE_TABLE_NAME = "Vehicle";

export const GOV_ID_INDEX_NAME = "GovId";
export const LICENSE_NUMBER_INDEX_NAME = "LicenseNumber";
export const LICENSE_PLATE_NUMBER_INDEX_NAME = "LicensePlateNumber";
export const PERSON_ID_INDEX_NAME = "PersonId";
export const VIN_INDEX_NAME = "VIN";

export const RETRY_LIMIT = 4;

export const JOURNAL_EXPORT_S3_BUCKET_NAME_PREFIX = "qldb-tutorial-journal-export";
export const USER_TABLES = "information_schema.user_tables";
```

2. 다음 프로그램(`CreateLedger.ts`)을 사용하여 이름이 `vehicle-registration`로 지정된 원장을 생성합니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QLDB } from "aws-sdk";
import {
```

```
    CreateLedgerRequest,
    CreateLedgerResponse,
    DescribeLedgerRequest,
    DescribeLedgerResponse
} from "aws-sdk/clients/qldb";

import { LEDGER_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { sleep } from "./qldb/Util";

const LEDGER_CREATION_POLL_PERIOD_MS = 10000;
const ACTIVE_STATE = "ACTIVE";

/**
 * Create a new ledger with the specified name.
 * @param ledgerName Name of the ledger to be created.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with a CreateLedgerResponse.
 */
export async function createLedger(ledgerName: string, qldbClient: QLDB):
 Promise<CreateLedgerResponse> {
    log(`Creating a ledger named: ${ledgerName}...`);
    const request: CreateLedgerRequest = {
        Name: ledgerName,
        PermissionsMode: "ALLOW_ALL"
    }
    const result: CreateLedgerResponse = await
 qldbClient.createLedger(request).promise();
    log(`Success. Ledger state: ${result.State}.`);
    return result;
}

/**
 * Wait for the newly created ledger to become active.
 * @param ledgerName Name of the ledger to be checked on.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with a DescribeLedgerResponse.
 */
export async function waitForActive(ledgerName: string, qldbClient: QLDB):
 Promise<DescribeLedgerResponse> {
    log(`Waiting for ledger ${ledgerName} to become active...`);
    const request: DescribeLedgerRequest = {
        Name: ledgerName
    }
```

```
    while (true) {
        const result: DescribeLedgerResponse = await
 qldbClient.describeLedger(request).promise();
        if (result.State === ACTIVE_STATE) {
            log("Success. Ledger is active and ready to be used.");
            return result;
        }
        log("The ledger is still creating. Please wait...");
        await sleep(LEDGER_CREATION_POLL_PERIOD_MS);
    }
}

/**
 * Create a ledger and wait for it to be active.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbClient: QLDB = new QLDB();
        await createLedger(LEDGER_NAME, qldbClient);
        await waitForActive(LEDGER_NAME, qldbClient);
    } catch (e) {
        error(`Unable to create the ledger: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> - `createLedger` 호출 시 원장 이름과 권한 모드를 지정해야 합니다. 원장 데이터의 보안을 극대화하려면 STANDARD 권한 모드를 사용할 것을 권장합니다.
> - 원장을 생성할 때 기본적으로 삭제 방지가 활성화됩니다. 이 기능은 사용자가 원장을 삭제하는 것을 방지하는 QLDB의 기능입니다. QLDB API 또는 AWS Command Line Interface(AWS CLI)를 사용하여 원장 생성 시 삭제 보호를 비활성화할 수 있습니다.
> - 선택적으로, 원장에 첨부할 태그를 지정할 수도 있습니다.

3. 트랜스파일된 프로그램을 실행하려면 다음 명령을 입력합니다.

```
node dist/CreateLedger.js
```

새 원장과의 연결을 확인하려면 2단계: 원장과의 연결을 테스트로 이동하십시오.

2단계: 원장과의 연결을 테스트

이 단계에서는 트랜잭션 데이터 API 엔드포인트를 사용하여 Amazon QLDB의 `vehicle-registration` 원장에 연결할 수 있는지 확인합니다.

원장과의 연결을 테스트하려면

1. 다음 프로그램(`ConnectToLedger.ts`)을 사용하여 `vehicle-registration` 원장에 대한 데이터 세션 연결을 생성합니다.

   2.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */
```

```
import { QldbDriver, RetryConfig  } from "amazon-qldb-driver-nodejs";
import { ClientConfiguration } from "aws-sdk/clients/qldbsession";

import { LEDGER_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";

const qldbDriver: QldbDriver = createQldbDriver();

/**
 * Create a driver for creating sessions.
 * @param ledgerName The name of the ledger to create the driver on.
 * @param serviceConfigurationOptions The configurations for the AWS SDK client
 that the driver uses.
 * @returns The driver for creating sessions.
 */
export function createQldbDriver(
    ledgerName: string = LEDGER_NAME,
    serviceConfigurationOptions: ClientConfiguration = {}
): QldbDriver {
    const retryLimit = 4;
    const maxConcurrentTransactions = 10;
    //Use driver's default backoff function (and hence, no second parameter
 provided to RetryConfig)
    const retryConfig: RetryConfig = new RetryConfig(retryLimit);
    const qldbDriver: QldbDriver = new QldbDriver(ledgerName,
 serviceConfigurationOptions, maxConcurrentTransactions, retryConfig);
    return qldbDriver;
}

export function getQldbDriver(): QldbDriver {
    return qldbDriver;
}

/**
 * Connect to a session for a given ledger using default settings.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        log("Listing table names...");
        const tableNames: string[] = await qldbDriver.getTableNames();
        tableNames.forEach((tableName: string): void => {
            log(tableName);
```

```
        });
    } catch (e) {
        error(`Unable to create session: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

1.x

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
 THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver  } from "amazon-qldb-driver-nodejs";
import { ClientConfiguration } from "aws-sdk/clients/qldbsession";

import { LEDGER_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
```

```
const qldbDriver: QldbDriver = createQldbDriver();

/**
 * Create a driver for creating sessions.
 * @param ledgerName The name of the ledger to create the driver on.
 * @param serviceConfigurationOptions The configurations for the AWS SDK client
 that the driver uses.
 * @returns The driver for creating sessions.
 */
export function createQldbDriver(
    ledgerName: string = LEDGER_NAME,
    serviceConfigurationOptions: ClientConfiguration = {}
): QldbDriver {
    const qldbDriver: QldbDriver = new QldbDriver(ledgerName,
 serviceConfigurationOptions);
    return qldbDriver;
}

export function getQldbDriver(): QldbDriver {
    return qldbDriver;
}

/**
 * Connect to a session for a given ledger using default settings.
 * @returns Promise which fulfills with void.
 */
var main = async function(): Promise<void> {
    try {
        log("Listing table names...");
        const tableNames: string[] = await qldbDriver.getTableNames();
        tableNames.forEach((tableName: string): void => {
            log(tableName);
        });
    } catch (e) {
        error(`Unable to create session: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> 원장에서 데이터 트랜잭션을 실행하려면 지정된 원장에 연결할 QLDB 드라이버 객체를 만들어야 합니다. 이는 이전 단계에서 원장을 생성할 때 사용한 `qldbClient` 객체와는 다른 클라이언트 객체입니다. 이전 클라이언트는 Amazon QLDB API 참조에 나열된 관리 API 작업을 실행하는 데만 사용됩니다.

2. 트랜스파일된 프로그램을 실행하려면 다음 명령을 입력합니다.

```
node dist/ConnectToLedger.js
```

`vehicle-registration` 원장에 테이블을 생성하려면 3단계: 테이블, 인덱스 및 샘플 데이터 생성로 진행하십시오.

3단계: 테이블, 인덱스 및 샘플 데이터 생성

Amazon QLDB 원장이 활성화되고 연결을 수락하면 차량, 소유자 및 등록 정보에 대한 데이터 테이블 생성을 시작할 수 있습니다. 테이블과 인덱스를 생성한 후 데이터를 로드할 수 있습니다.

이 단계에서는 `vehicle-registration` 원장에 4개의 테이블을 생성합니다.

- `VehicleRegistration`
- `Vehicle`
- `Person`
- `DriversLicense`

또한 다음과 같은 인덱스를 생성합니다.

테이블 이름	필드
`VehicleRegistration`	`VIN`
`VehicleRegistration`	`LicensePlateNumber`
`Vehicle`	`VIN`

테이블 이름	필드
Person	GovId
DriversLicense	LicenseNumber
DriversLicense	PersonId

샘플 데이터를 삽입할 때는 먼저 Person 테이블에 문서를 삽입합니다. 그런 다음 각 Person 문서에서 시스템이 할당한 id를 사용하여 적절한 VehicleRegistration 및 DriversLicense 문서의 해당 필드를 채웁니다.

> **Tip**
>
> 가장 좋은 방법은 문서의 시스템 할당 id를 외래 키로 사용하는 것입니다. 고유 식별자(예: 차량의 VIN)로 사용되는 필드를 정의할 수 있지만 문서의 실제 고유 식별자는 id입니다. 이 필드는 문서의 메타데이터에 포함되며 커밋된 뷰(테이블의 시스템 정의 뷰)에서 쿼리할 수 있습니다.
>
> QLDB 뷰에 대한 자세한 내용은 핵심 개념 섹션을 참조하세요. 메타데이터에 대해 자세히 알아보려면 문서 메타데이터 쿼리 섹션을 참조하세요.

테이블 및 인덱스를 생성하려면

1. 다음 프로그램(CreateTable.ts)을 사용하여 앞서 언급한 테이블을 생성합니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
```

```
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";

import { getQldbDriver } from "./ConnectToLedger";
import {
    DRIVERS_LICENSE_TABLE_NAME,
    PERSON_TABLE_NAME,
    VEHICLE_REGISTRATION_TABLE_NAME,
    VEHICLE_TABLE_NAME
} from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";

/**
 * Create multiple tables in a single transaction.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName Name of the table to create.
 * @returns Promise which fulfills with the number of changes to the database.
 */
export async function createTable(txn: TransactionExecutor, tableName: string):
 Promise<number> {
    const statement: string = `CREATE TABLE ${tableName}`;
    return await txn.execute(statement).then((result: Result) => {
        log(`Successfully created table ${tableName}.`);
        return result.getResultList().length;
    });
}

/**
 * Create tables in a QLDB ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
```

```
        const qldbDriver: QldbDriver = getQldbDriver();
        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            Promise.all([
                createTable(txn, VEHICLE_REGISTRATION_TABLE_NAME),
                createTable(txn, VEHICLE_TABLE_NAME),
                createTable(txn, PERSON_TABLE_NAME),
                createTable(txn, DRIVERS_LICENSE_TABLE_NAME)
            ]);
        });
    } catch (e) {
        error(`Unable to create tables: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> 이 프로그램은 QLDB 드라이버 인스턴스에서 `executeLambda` 함수를 사용하는 방법을 보여줍니다. 이 예제에서는 단일 Lambda 표현식을 사용하여 여러 `CREATE TABLE` PartiQL 문을 실행합니다.
>
> 이 실행 함수는 암시적으로 트랜잭션을 시작하고 Lambda에서 모든 문을 실행한 다음 트랜잭션을 자동 커밋합니다.

2. 트랜스파일된 프로그램을 실행하려면 다음 명령을 입력합니다.

```
node dist/CreateTable.js
```

3. 앞에서 설명한 대로 다음 프로그램(`CreateIndex.ts`)을 사용하여 테이블에 인덱스를 생성합니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
```

```
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";

import { getQldbDriver } from "./ConnectToLedger";
import {
    DRIVERS_LICENSE_TABLE_NAME,
    GOV_ID_INDEX_NAME,
    LICENSE_NUMBER_INDEX_NAME,
    LICENSE_PLATE_NUMBER_INDEX_NAME,
    PERSON_ID_INDEX_NAME,
    PERSON_TABLE_NAME,
    VEHICLE_REGISTRATION_TABLE_NAME,
    VEHICLE_TABLE_NAME,
    VIN_INDEX_NAME
} from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";

/**
 * Create an index for a particular table.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName Name of the table to add indexes for.
 * @param indexAttribute Index to create on a single attribute.
 * @returns Promise which fulfills with the number of changes to the database.
 */
export async function createIndex(
    txn: TransactionExecutor,
    tableName: string,
    indexAttribute: string
```

```
): Promise<number> {
    const statement: string = `CREATE INDEX on ${tableName} (${indexAttribute})`;
    return await txn.execute(statement).then((result) => {
        log(`Successfully created index ${indexAttribute} on table ${tableName}.`);
        return result.getResultList().length;
    });
}

/**
 * Create indexes on tables in a particular ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();
        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            Promise.all([
                createIndex(txn, PERSON_TABLE_NAME, GOV_ID_INDEX_NAME),
                createIndex(txn, VEHICLE_TABLE_NAME, VIN_INDEX_NAME),
                createIndex(txn, VEHICLE_REGISTRATION_TABLE_NAME, VIN_INDEX_NAME),
                createIndex(txn, VEHICLE_REGISTRATION_TABLE_NAME,
 LICENSE_PLATE_NUMBER_INDEX_NAME),
                createIndex(txn, DRIVERS_LICENSE_TABLE_NAME, PERSON_ID_INDEX_NAME),
                createIndex(txn, DRIVERS_LICENSE_TABLE_NAME,
 LICENSE_NUMBER_INDEX_NAME)
            ]);
        });
    } catch (e) {
        error(`Unable to create indexes: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

4. 트랜스파일된 프로그램을 실행하려면 다음 명령을 입력합니다.

```
node dist/CreateIndex.js
```

데이터를 테이블로 로드하려면

1. 다음 .ts 파일을 검토합니다.

    1. SampleData.ts – vehicle-registration 테이블에 삽입한 샘플 데이터를 포함합니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { Decimal } from "ion-js";

const EMPTY_SECONDARY_OWNERS: object[] = [];
export const DRIVERS_LICENSE = [
    {
        PersonId: "",
        LicenseNumber: "LEWISR261LL",
        LicenseType: "Learner",
        ValidFromDate: new Date("2016-12-20"),
        ValidToDate: new Date("2020-11-15")
    },
    {
```

```
        PersonId: "",
        LicenseNumber : "LOGANB486CG",
        LicenseType: "Probationary",
        ValidFromDate : new Date("2016-04-06"),
        ValidToDate : new Date("2020-11-15")
    },
    {
        PersonId: "",
        LicenseNumber : "744 849 301",
        LicenseType: "Full",
        ValidFromDate : new Date("2017-12-06"),
        ValidToDate : new Date("2022-10-15")
    },
    {
        PersonId: "",
        LicenseNumber : "P626-168-229-765",
        LicenseType: "Learner",
        ValidFromDate : new Date("2017-08-16"),
        ValidToDate : new Date("2021-11-15")
    },
    {
        PersonId: "",
        LicenseNumber : "S152-780-97-415-0",
        LicenseType: "Probationary",
        ValidFromDate : new Date("2015-08-15"),
        ValidToDate : new Date("2021-08-21")
    }
];
export const PERSON = [
    {
        FirstName : "Raul",
        LastName : "Lewis",
        DOB : new Date("1963-08-19"),
        Address : "1719 University Street, Seattle, WA, 98109",
        GovId : "LEWISR261LL",
        GovIdType : "Driver License"
    },
    {
        FirstName : "Brent",
        LastName : "Logan",
        DOB : new Date("1967-07-03"),
        Address : "43 Stockert Hollow Road, Everett, WA, 98203",
        GovId : "LOGANB486CG",
        GovIdType : "Driver License"
```

```
    },
    {
        FirstName : "Alexis",
        LastName : "Pena",
        DOB : new Date("1974-02-10"),
        Address : "4058 Melrose Street, Spokane Valley, WA, 99206",
        GovId : "744 849 301",
        GovIdType : "SSN"
    },
    {
        FirstName : "Melvin",
        LastName : "Parker",
        DOB : new Date("1976-05-22"),
        Address : "4362 Ryder Avenue, Seattle, WA, 98101",
        GovId : "P626-168-229-765",
        GovIdType : "Passport"
    },
    {
        FirstName : "Salvatore",
        LastName : "Spencer",
        DOB : new Date("1997-11-15"),
        Address : "4450 Honeysuckle Lane, Seattle, WA, 98101",
        GovId : "S152-780-97-415-0",
        GovIdType : "Passport"
    }
];
export const VEHICLE = [
    {
        VIN : "1N4AL11D75C109151",
        Type : "Sedan",
        Year : 2011,
        Make : "Audi",
        Model : "A5",
        Color : "Silver"
    },
    {
        VIN : "KM8SRDHF6EU074761",
        Type : "Sedan",
        Year : 2015,
        Make : "Tesla",
        Model : "Model S",
        Color : "Blue"
    },
    {
```

```
        VIN : "3HGGK5G53FM761765",
        Type : "Motorcycle",
        Year : 2011,
        Make : "Ducati",
        Model : "Monster 1200",
        Color : "Yellow"
    },
    {
        VIN : "1HVBBAANXWH544237",
        Type : "Semi",
        Year : 2009,
        Make : "Ford",
        Model : "F 150",
        Color : "Black"
    },
    {
        VIN : "1C4RJFAG0FC625797",
        Type : "Sedan",
        Year : 2019,
        Make : "Mercedes",
        Model : "CLK 350",
        Color : "White"
    }
];
export const VEHICLE_REGISTRATION = [
    {
        VIN : "1N4AL11D75C109151",
        LicensePlateNumber : "LEWISR261LL",
        State : "WA",
        City : "Seattle",
        ValidFromDate : new Date("2017-08-21"),
        ValidToDate : new Date("2020-05-11"),
        PendingPenaltyTicketAmount : new Decimal(9025, -2),
        Owners : {
            PrimaryOwner : { PersonId : "" },
            SecondaryOwners : EMPTY_SECONDARY_OWNERS
        }
    },
    {
        VIN : "KM8SRDHF6EU074761",
        LicensePlateNumber : "CA762X",
        State : "WA",
        City : "Kent",
        PendingPenaltyTicketAmount : new Decimal(13075, -2),
```

```
        ValidFromDate : new Date("2017-09-14"),
        ValidToDate : new Date("2020-06-25"),
        Owners : {
            PrimaryOwner : { PersonId : "" },
            SecondaryOwners : EMPTY_SECONDARY_OWNERS
        }
    },
    {
        VIN : "3HGGK5G53FM761765",
        LicensePlateNumber : "CD820Z",
        State : "WA",
        City : "Everett",
        PendingPenaltyTicketAmount : new Decimal(44230, -2),
        ValidFromDate : new Date("2011-03-17"),
        ValidToDate : new Date("2021-03-24"),
        Owners : {
            PrimaryOwner : { PersonId : "" },
            SecondaryOwners : EMPTY_SECONDARY_OWNERS
        }
    },
    {
        VIN : "1HVBBAANXWH544237",
        LicensePlateNumber : "LS477D",
        State : "WA",
        City : "Tacoma",
        PendingPenaltyTicketAmount : new Decimal(4220, -2),
        ValidFromDate : new Date("2011-10-26"),
        ValidToDate : new Date("2023-09-25"),
        Owners : {
            PrimaryOwner : { PersonId : "" },
            SecondaryOwners : EMPTY_SECONDARY_OWNERS
        }
    },
    {
        VIN : "1C4RJFAG0FC625797",
        LicensePlateNumber : "TH393F",
        State : "WA",
        City : "Olympia",
        PendingPenaltyTicketAmount : new Decimal(3045, -2),
        ValidFromDate : new Date("2013-09-02"),
        ValidToDate : new Date("2024-03-19"),
        Owners : {
            PrimaryOwner : { PersonId : "" },
            SecondaryOwners : EMPTY_SECONDARY_OWNERS
```

```
        }
    }
];
```

2. `Util.ts` – Amazon Ion 데이터를 변환, 구문 분석 및 인쇄하는 도우미 기능을 제공하기 위해 `ion-js` 패키지에서 가져오는 유틸리티 모듈입니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { Result, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { GetBlockResponse, GetDigestResponse, ValueHolder } from "aws-sdk/
clients/qldb";
import {
    Decimal,
    decodeUtf8,
    dom,
    IonTypes,
    makePrettyWriter,
    makeReader,
    Reader,
```

```
    Timestamp,
    toBase64,
    Writer
} from "ion-js";

import { error } from "./LogUtil";



/**
 * TODO: Replace this with json.stringify
 * Returns the string representation of a given BlockResponse.
 * @param blockResponse The BlockResponse to convert to string.
 * @returns The string representation of the supplied BlockResponse.
 */
export function blockResponseToString(blockResponse: GetBlockResponse): string {
    let stringBuilder: string = "";
    if (blockResponse.Block.IonText) {
        stringBuilder = stringBuilder + "Block: " + blockResponse.Block.IonText +
 ", ";
    }
    if (blockResponse.Proof.IonText) {
        stringBuilder = stringBuilder + "Proof: " + blockResponse.Proof.IonText;
    }
    stringBuilder = "{" + stringBuilder + "}";
    const writer: Writer = makePrettyWriter();
    const reader: Reader = makeReader(stringBuilder);
    writer.writeValues(reader);
    return decodeUtf8(writer.getBytes());
}

/**
 * TODO: Replace this with json.stringify
 * Returns the string representation of a given GetDigestResponse.
 * @param digestResponse The GetDigestResponse to convert to string.
 * @returns The string representation of the supplied GetDigestResponse.
 */
export function digestResponseToString(digestResponse: GetDigestResponse): string
 {
    let stringBuilder: string = "";
    if (digestResponse.Digest) {
        stringBuilder += "Digest: " + JSON.stringify(toBase64(<Uint8Array>
 digestResponse.Digest)) + ", ";
    }
```

```
    if (digestResponse.DigestTipAddress.IonText) {
        stringBuilder += "DigestTipAddress: " +
 digestResponse.DigestTipAddress.IonText;
    }
    stringBuilder = "{" + stringBuilder + "}";
    const writer: Writer = makePrettyWriter();
    const reader: Reader = makeReader(stringBuilder);
    writer.writeValues(reader);
    return decodeUtf8(writer.getBytes());
}

/**
 * Get the document IDs from the given table.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName The table name to query.
 * @param field A field to query.
 * @param value The key of the given field.
 * @returns Promise which fulfills with the document ID as a string.
 */
export async function getDocumentId(
    txn: TransactionExecutor,
    tableName: string,
    field: string,
    value: string
): Promise<string> {
    const query: string = `SELECT id FROM ${tableName} AS t BY id WHERE t.
${field} = ?`;
    let documentId: string = undefined;
    await txn.execute(query, value).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();
        if (resultList.length === 0) {
            throw new Error(`Unable to retrieve document ID using ${value}.`);
        }
        documentId = resultList[0].get("id").stringValue();

    }).catch((err: any) => {
        error(`Error getting documentId: ${err}`);
    });
    return documentId;
}

/**
 * Sleep for the specified amount of time.
 * @param ms The amount of time to sleep in milliseconds.
```

```
 * @returns Promise which fulfills with void.
 */
export function sleep(ms: number): Promise<void> {
    return new Promise(resolve => setTimeout(resolve, ms));
}

/**
 * Find the value of a given path in an Ion value. The path should contain a blob
 value.
 * @param value The Ion value that contains the journal block attributes.
 * @param path The path to a certain attribute.
 * @returns Uint8Array value of the blob, or null if the attribute cannot be
 found in the Ion value
 *                  or is not of type Blob
 */
export function getBlobValue(value: dom.Value, path: string): Uint8Array | null {
    const attribute: dom.Value = value.get(path);
    if (attribute !== null && attribute.getType() === IonTypes.BLOB) {
        return attribute.uInt8ArrayValue();
    }
    return null;
}

/**
 * TODO: Replace this with json.stringify
 * Returns the string representation of a given ValueHolder.
 * @param valueHolder The ValueHolder to convert to string.
 * @returns The string representation of the supplied ValueHolder.
 */
export function valueHolderToString(valueHolder: ValueHolder): string {
    const stringBuilder: string = `{ IonText: ${valueHolder.IonText}}`;
    const writer: Writer = makePrettyWriter();
    const reader: Reader = makeReader(stringBuilder);
    writer.writeValues(reader);
    return decodeUtf8(writer.getBytes());
}

/**
 * Converts a given value to Ion using the provided writer.
 * @param value The value to convert to Ion.
 * @param ionWriter The Writer to pass the value into.
 * @throws Error: If the given value cannot be converted to Ion.
 */
export function writeValueAsIon(value: any, ionWriter: Writer): void {
```

```
    switch (typeof value) {
        case "string":
            ionWriter.writeString(value);
            break;
        case "boolean":
            ionWriter.writeBoolean(value);
            break;
        case "number":
                ionWriter.writeInt(value);
                break;
        case "object":
            if (Array.isArray(value)) {
                // Object is an array.
                ionWriter.stepIn(IonTypes.LIST);

                for (const element of value) {
                    writeValueAsIon(element, ionWriter);
                }

                ionWriter.stepOut();
            } else if (value instanceof Date) {
                // Object is a Date.
                ionWriter.writeTimestamp(Timestamp.parse(value.toISOString()));
            } else if (value instanceof Decimal) {
                // Object is a Decimal.
                ionWriter.writeDecimal(value);
            } else if (value === null) {
                ionWriter.writeNull(IonTypes.NULL);
            } else {
                // Object is a struct.
                ionWriter.stepIn(IonTypes.STRUCT);

                for (const key of Object.keys(value)) {
                    ionWriter.writeFieldName(key);
                    writeValueAsIon(value[key], ionWriter);
                }
                ionWriter.stepOut();
            }
            break;
        default:
            throw new Error(`Cannot convert to Ion for type: ${(typeof
value)}.`);
    }
```

```
}
```

> **Note**
>
> 이 `getDocumentId` 함수는 테이블에서 시스템 할당 문서 ID를 반환하는 쿼리를 실행합니다. 자세한 내용은 BY 절을 사용하여 문서 ID 쿼리하기 섹션을 참조하세요.

2. 다음 프로그램(`InsertDocument.ts`)을 사용하여 샘플 데이터를 테이블에 삽입합니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";
import { dom } from "ion-js";

import { getQldbDriver } from "./ConnectToLedger";
import { DRIVERS_LICENSE, PERSON, VEHICLE, VEHICLE_REGISTRATION } from "./model/
SampleData";
import {
```

```
    DRIVERS_LICENSE_TABLE_NAME,
    PERSON_TABLE_NAME,
    VEHICLE_REGISTRATION_TABLE_NAME,
    VEHICLE_TABLE_NAME
} from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";

/**
 * Insert the given list of documents into a table in a single transaction.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName Name of the table to insert documents into.
 * @param documents List of documents to insert.
 * @returns Promise which fulfills with a {@linkcode Result} object.
 */
export async function insertDocument(
    txn: TransactionExecutor,
    tableName: string,
    documents: object[]
): Promise<Result> {
    const statement: string = `INSERT INTO ${tableName} ?`;
    const result: Result = await txn.execute(statement, documents);
    return result;
}

/**
 * Handle the insertion of documents and updating PersonIds all in a single
 transaction.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @returns Promise which fulfills with void.
 */
async function updateAndInsertDocuments(txn: TransactionExecutor): Promise<void> {
    log("Inserting multiple documents into the 'Person' table...");
    const documentIds: Result = await insertDocument(txn, PERSON_TABLE_NAME,
 PERSON);

    const listOfDocumentIds: dom.Value[] = documentIds.getResultList();
    log("Updating PersonIds for 'DriversLicense' and PrimaryOwner for
 'VehicleRegistration'...");
    updatePersonId(listOfDocumentIds);

    log("Inserting multiple documents into the remaining tables...");
    await Promise.all([
        insertDocument(txn, DRIVERS_LICENSE_TABLE_NAME, DRIVERS_LICENSE),
        insertDocument(txn, VEHICLE_REGISTRATION_TABLE_NAME, VEHICLE_REGISTRATION),
```

```
        insertDocument(txn, VEHICLE_TABLE_NAME, VEHICLE)
    ]);
}

/**
 * Update the PersonId value for DriversLicense records and the PrimaryOwner value
 for VehicleRegistration records.
 * @param documentIds List of document IDs.
 */
export function updatePersonId(documentIds: dom.Value[]): void {
    documentIds.forEach((value: dom.Value, i: number) => {
        const documentId: string = value.get("documentId").stringValue();
        DRIVERS_LICENSE[i].PersonId = documentId;
        VEHICLE_REGISTRATION[i].Owners.PrimaryOwner.PersonId = documentId;
    });
}

/**
 * Insert documents into a table in a QLDB ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();
        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await updateAndInsertDocuments(txn);
        });
    } catch (e) {
        error(`Unable to insert documents: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> - 이 프로그램은 파라미터화된 값을 사용하여 execute 함수를 호출하는 방법을 보여줍니다. 실행하려는 PartiQL 문 외에도 데이터 파라미터를 전달할 수 있습니다. 명령문 문자열에서 물음표(?)를 변수 자리 표시자로 사용하세요.

- INSERT 문이 성공하면 삽입된 각 문서의 id이 반환됩니다.

3. 트랜스파일된 프로그램을 실행하려면 다음 명령을 입력합니다.

```
node dist/InsertDocument.js
```

다음으로 SELECT 문을 사용하여 vehicle-registration 원장의 테이블에서 데이터를 읽을 수 있습니다. 4단계: 원장에서 테이블 쿼리로 이동합니다.

4단계: 원장에서 테이블 쿼리

Amazon QLDB 원장에 테이블을 생성하고 데이터를 로드한 후 쿼리를 실행하여 방금 삽입한 차량 등록 데이터를 검토할 수 있습니다. QLDB는 PartiQL을 쿼리 언어로 사용하고 Amazon Ion을 문서 지향 데이터 모델로 사용합니다.

PartiQL은 Ion과 함께 작동하도록 확장된 오픈 소스 SQL 호환 쿼리 언어입니다. PartiQL을 사용하면 익숙한 SQL 연산자를 사용하여 데이터를 삽입, 쿼리 및 관리할 수 있습니다. Amazon Ion은 JSON의 상위 집합입니다. Ion은 정형, 반정형 및 중첩 데이터를 저장하고 처리할 수 있는 유연성을 제공하는 오픈 소스 문서 기반 데이터 형식입니다.

이 단계에서는 SELECT 명령문을 사용하여 vehicle-registration 원장의 테이블에서 데이터를 읽습니다.

> **Warning**
>
> 인덱싱된 조회 없이 QLDB에서 쿼리를 실행하면 전체 테이블 스캔이 호출됩니다. PartiQL은 SQL과 호환되므로 이러한 쿼리를 지원합니다. 하지만 QLDB의 프로덕션 사용 사례에 대해서는 테이블 스캔을 실행하지 마세요. 테이블 스캔은 동시성 충돌 및 트랜잭션 시간 초과를 포함하여 대규모 테이블에서 성능 문제를 일으킬 수 있습니다.
> 인덱싱된 필드 또는 문서 ID(예: `WHERE indexedField = 123` 또는 `WHERE indexedField IN (456, 789)`)에서 동등 연산자를 사용하여 WHERE 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 자세한 내용은 쿼리 성능 최적화을 참조하십시오.

테이블을 쿼리하려면

1. 다음 프로그램(FindVehicles.ts)을 사용하여 원장에 있는 사람의 이름으로 등록된 모든 차량을 쿼리하세요.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";
import { dom } from "ion-js";

import { getQldbDriver } from "./ConnectToLedger";
import { PERSON } from "./model/SampleData";
import { PERSON_TABLE_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { getDocumentId } from "./qldb/Util";
import { prettyPrintResultList } from "./ScanTable";

/**
 * Query 'Vehicle' and 'VehicleRegistration' tables using a unique document ID in
 one transaction.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param govId The owner's government ID.
 * @returns Promise which fulfills with void.
 */
```

```
async function findVehiclesForOwner(txn: TransactionExecutor, govId: string):
 Promise<void> {
    const documentId: string = await getDocumentId(txn, PERSON_TABLE_NAME, "GovId",
 govId);
    const query: string = "SELECT Vehicle FROM Vehicle INNER JOIN
 VehicleRegistration AS r " +
                          "ON Vehicle.VIN = r.VIN WHERE
 r.Owners.PrimaryOwner.PersonId = ?";

    await txn.execute(query, documentId).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();
        log(`List of vehicles for owner with GovId: ${govId}`);
        prettyPrintResultList(resultList);
    });
}

/**
 * Find all vehicles registered under a person.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();
        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await findVehiclesForOwner(txn, PERSON[0].GovId);
        });
    } catch (e) {
        error(`Error getting vehicles for owner: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> 먼저 이 프로그램은 이 문서에 대한 Person 테이블을 GovId LEWISR261LL로 쿼리하여 id 메타데이터 필드를 가져옵니다.

그런 다음 이 문서 `id`를 외래 키로 사용하여 `VehicleRegistration` 테이블을 `PrimaryOwner.PersonId`로 쿼리합니다. 또한 `VIN` 필드의 `Vehicle` 테이블과 `VehicleRegistration`를 조인합니다.

2. 트랜스파일된 프로그램을 실행하려면 다음 명령을 입력합니다.

```
node dist/FindVehicles.js
```

`vehicle-registration` 원장의 테이블에 있는 문서를 수정하는 방법에 대한 자세한 내용은 5단계: 원장의 문서 수정을 참조하십시오.

5단계: 원장의 문서 수정

이제 작업할 데이터가 있으므로 Amazon QLDB에서 `vehicle-registration` 원장에 있는 문서를 변경할 수 있습니다. 이 단계에서 다음 코드 예제는 DML(데이터 조작 언어) 문을 실행하는 방법을 보여 줍니다. 이러한 명령문은 한 차량의 주 소유자를 업데이트하고 다른 차량에 보조 소유자를 추가합니다.

문서를 수정하려면

1. 다음 프로그램(`TransferVehicleOwnership.ts`)을 사용하여 차량의 주 소유자를 원장에 VIN `1N4AL11D75C109151`으로 업데이트하세요.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
```

```
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";
import { dom } from "ion-js";

import { getQldbDriver } from "./ConnectToLedger";
import { PERSON, VEHICLE } from "./model/SampleData";
import { PERSON_TABLE_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { getDocumentId } from "./qldb/Util";

/**
 * Query a driver's information using the given ID.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param documentId The unique ID of a document in the Person table.
 * @returns Promise which fulfills with an Ion value containing the person.
 */
export async function findPersonFromDocumentId(txn: TransactionExecutor,
 documentId: string): Promise<dom.Value> {
    const query: string = "SELECT p.* FROM Person AS p BY pid WHERE pid = ?";

    let personId: dom.Value;
    await txn.execute(query, documentId).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();
        if (resultList.length === 0) {
            throw new Error(`Unable to find person with ID: ${documentId}.`);
        }
        personId = resultList[0];
    });
    return personId;
}

/**
 * Find the primary owner for the given VIN.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin The VIN to find primary owner for.
 * @returns Promise which fulfills with an Ion value containing the primary owner.
```

```
 */
export async function findPrimaryOwnerForVehicle(txn: TransactionExecutor, vin:
 string): Promise<dom.Value> {
    log(`Finding primary owner for vehicle with VIN: ${vin}`);
    const query: string = "SELECT Owners.PrimaryOwner.PersonId FROM
 VehicleRegistration AS v WHERE v.VIN = ?";

    let documentId: string = undefined;
    await txn.execute(query, vin).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();
        if (resultList.length === 0) {
            throw new Error(`Unable to retrieve document ID using ${vin}.`);
        }
        const PersonIdValue: dom.Value = resultList[0].get("PersonId");
        if (PersonIdValue === null) {
            throw new Error(`Expected field name PersonId not found.`);
        }
        documentId = PersonIdValue.stringValue();
    });
    return findPersonFromDocumentId(txn, documentId);
}

/**
 * Update the primary owner for a vehicle using the given VIN.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin The VIN for the vehicle to operate on.
 * @param documentId New PersonId for the primary owner.
 * @returns Promise which fulfills with void.
 */
async function updateVehicleRegistration(txn: TransactionExecutor, vin: string,
 documentId: string): Promise<void> {
    const statement: string = "UPDATE VehicleRegistration AS r SET
 r.Owners.PrimaryOwner.PersonId = ? WHERE r.VIN = ?";

    log(`Updating the primary owner for vehicle with VIN: ${vin}...`);
    await txn.execute(statement, documentId, vin).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();
        if (resultList.length === 0) {
            throw new Error("Unable to transfer vehicle, could not find
 registration.");
        }
        log(`Successfully transferred vehicle with VIN ${vin} to new owner.`);
    });
}
```

```
/**
 * Validate the current owner of the given vehicle and transfer its ownership to a
 new owner in a single transaction.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin The VIN of the vehicle to transfer ownership of.
 * @param currentOwner The GovId of the current owner of the vehicle.
 * @param newOwner The GovId of the new owner of the vehicle.
 */
export async function validateAndUpdateRegistration(
    txn: TransactionExecutor,
    vin: string,
    currentOwner: string,
    newOwner: string
): Promise<void> {
    const primaryOwner: dom.Value = await findPrimaryOwnerForVehicle(txn, vin);
    const govIdValue: dom.Value = primaryOwner.get("GovId");
    if (govIdValue !== null && govIdValue.stringValue() !== currentOwner) {
        log("Incorrect primary owner identified for vehicle, unable to transfer.");
    }
    else {
        const documentId: string = await getDocumentId(txn, PERSON_TABLE_NAME,
 "GovId", newOwner);
        await updateVehicleRegistration(txn, vin, documentId);
        log("Successfully transferred vehicle ownership!");
    }
}

/**
 * Find primary owner for a particular vehicle's VIN.
 * Transfer to another primary owner for a particular vehicle's VIN.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();

        const vin: string = VEHICLE[0].VIN;
        const previousOwnerGovId: string = PERSON[0].GovId;
        const newPrimaryOwnerGovId: string = PERSON[1].GovId;

        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await validateAndUpdateRegistration(txn, vin, previousOwnerGovId,
 newPrimaryOwnerGovId);
```

```
        });
    } catch (e) {
        error(`Unable to connect and run queries: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

2. 트랜스파일된 프로그램을 실행하려면 다음 명령을 입력합니다.

```
node dist/TransferVehicleOwnership.js
```

3. 다음 프로그램(AddSecondaryOwner.ts)을 사용하여 차량의 보조 소유자를 원장에 VIN KM8SRDHF6EU074761으로 추가하세요.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */
```

```
import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";
import { dom } from "ion-js";

import { getQldbDriver } from "./ConnectToLedger";
import { PERSON, VEHICLE_REGISTRATION } from "./model/SampleData";
import { PERSON_TABLE_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { getDocumentId } from "./qldb/Util";
import { prettyPrintResultList } from "./ScanTable";

/**
 * Add a secondary owner into 'VehicleRegistration' table for a particular VIN.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin VIN of the vehicle to query.
 * @param secondaryOwnerId The secondary owner's person ID.
 * @returns Promise which fulfills with void.
 */
export async function addSecondaryOwner(
    txn: TransactionExecutor,
    vin: string,
    secondaryOwnerId: string
): Promise<void> {
    log(`Inserting secondary owner for vehicle with VIN: ${vin}`);
    const query: string =
        `FROM VehicleRegistration AS v WHERE v.VIN = ? INSERT INTO
 v.Owners.SecondaryOwners VALUE ?`;

    const personToInsert = {PersonId: secondaryOwnerId};
    await txn.execute(query, vin, personToInsert).then(async (result: Result) => {
        const resultList: dom.Value[] = result.getResultList();
        log("VehicleRegistration Document IDs which had secondary owners added: ");
        prettyPrintResultList(resultList);
    });
}

/**
 * Query for a document ID with a government ID.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param governmentId The government ID to query with.
 * @returns Promise which fulfills with the document ID as a string.
 */
export async function getDocumentIdByGovId(txn: TransactionExecutor, governmentId:
 string): Promise<string> {
```

```
    const documentId: string = await getDocumentId(txn, PERSON_TABLE_NAME, "GovId",
 governmentId);
    return documentId;
}

/**
 * Check whether a driver has already been registered for the given VIN.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin VIN of the vehicle to query.
 * @param secondaryOwnerId The secondary owner's person ID.
 * @returns Promise which fulfills with a boolean.
 */
export async function isSecondaryOwnerForVehicle(
    txn: TransactionExecutor,
    vin: string,
    secondaryOwnerId: string
): Promise<boolean> {
    log(`Finding secondary owners for vehicle with VIN: ${vin}`);
    const query: string = "SELECT Owners.SecondaryOwners FROM VehicleRegistration
 AS v WHERE v.VIN = ?";

    let doesExist: boolean = false;

    await txn.execute(query, vin).then((result: Result) => {
        const resultList: dom.Value[] = result.getResultList();

        resultList.forEach((value: dom.Value) => {
            const secondaryOwnersList: dom.Value[] =
 value.get("SecondaryOwners").elements();

            secondaryOwnersList.forEach((secondaryOwner) => {
                const personId: dom.Value = secondaryOwner.get("PersonId");
                if (personId !== null &&  personId.stringValue() ===
 secondaryOwnerId) {
                    doesExist = true;
                }
            });
        });
    });
    return doesExist;
}

/**
 * Finds and adds secondary owners for a vehicle.
```

```
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();
        const vin: string = VEHICLE_REGISTRATION[1].VIN;
        const govId: string = PERSON[0].GovId;

        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            const documentId: string = await getDocumentIdByGovId(txn, govId);

            if (await isSecondaryOwnerForVehicle(txn, vin, documentId)) {
                log(`Person with ID ${documentId} has already been added as a
 secondary owner of this vehicle.`);
            } else {
                await addSecondaryOwner(txn, vin, documentId);
            }
        });

        log("Secondary owners successfully updated.");
    } catch (e) {
        error(`Unable to add secondary owner: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

4. 트랜스파일된 프로그램을 실행하려면 다음 명령을 입력합니다.

```
node dist/AddSecondaryOwner.js
```

vehicle-registration 원장의 이러한 변경 사항을 검토하려면 6단계: 문서의 개정 기록 보기을 참조하세요.

6단계: 문서의 개정 기록 보기

이전 단계에서 차량의 등록 데이터를 수정한 후, 등록된 모든 소유자 및 기타 업데이트된 필드의 기록을 쿼리할 수 있습니다. 이 단계에서는 vehicle-registration 원장의 VehicleRegistration 테이블에 있는 문서의 개정 기록을 쿼리합니다.

개정 기록을 보려면

1. 다음 프로그램(QueryHistory.ts)을 사용하여 VIN 1N4AL11D75C109151로 VehicleRegistration 문서의 개정 기록을 쿼리합니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-
nodejs";
import { dom } from "ion-js";

import { getQldbDriver } from "./ConnectToLedger";
import { VEHICLE_REGISTRATION } from "./model/SampleData";
import { VEHICLE_REGISTRATION_TABLE_NAME } from "./qldb/Constants";
import { prettyPrintResultList } from "./ScanTable";
import { error, log } from "./qldb/LogUtil";
import { getDocumentId } from "./qldb/Util";

/**
 * Find previous primary owners for the given VIN in a single transaction.
```

```
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin The VIN to find previous primary owners for.
 * @returns Promise which fulfills with void.
 */
async function previousPrimaryOwners(txn: TransactionExecutor, vin: string):
 Promise<void> {
    const documentId: string = await getDocumentId(txn,
 VEHICLE_REGISTRATION_TABLE_NAME, "VIN", vin);
    const todaysDate: Date = new Date();
    // set todaysDate back one minute to ensure end time is in the past
    // by the time the request reaches our backend
    todaysDate.setMinutes(todaysDate.getMinutes() - 1);
    const threeMonthsAgo: Date = new Date(todaysDate);
    threeMonthsAgo.setMonth(todaysDate.getMonth() - 3);

    const query: string =
        `SELECT data.Owners.PrimaryOwner, metadata.version FROM history ` +
        `(${VEHICLE_REGISTRATION_TABLE_NAME}, \`${threeMonthsAgo.toISOString()}\`,
 \`${todaysDate.toISOString()}\`) ` +
        `AS h WHERE h.metadata.id = ?`;

    await txn.execute(query, documentId).then((result: Result) => {
        log(`Querying the 'VehicleRegistration' table's history using VIN:
 ${vin}.`);
        const resultList: dom.Value[] = result.getResultList();
        prettyPrintResultList(resultList);
    });
}

/**
 * Query a table's history for a particular set of documents.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();
        const vin: string = VEHICLE_REGISTRATION[0].VIN;
        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await previousPrimaryOwners(txn, vin);
        });
    } catch (e) {
        error(`Unable to query history to find previous owners: ${e}`);
    }
}
```

```
if (require.main === module) {
    main();
}
```

> **Note**
>
> - 다음 구문에 내장된 기록 함수를 쿼리하여 문서의 개정 기록을 볼 수 있습니다
>
>   ```
>   SELECT * FROM history( table_name [, `start-time` [, `end-time` ] ] ) AS h
>   [ WHERE h.metadata.id = 'id' ]
>   ```
>
> - 시작 시간과 종료 시간은 모두 선택 사항입니다. 이 값은 백틱(`...`)으로 표시할 수 있는 Amazon Ion 리터럴 값입니다. 자세한 내용은 Amazon QLDB에서 PartiQL을 사용한 Ion 쿼리을 참조하십시오.
> - 가장 좋은 방법은 날짜 범위(시작 시간 및 종료 시간)와 문서 ID(`metadata.id`)를 모두 사용하여 기록 쿼리를 한정하는 것입니다. QLDB는 트랜잭션에서 SELECT 쿼리를 처리하며, 이 쿼리에는 트랜잭션 시간 초과 제한이 적용됩니다.
>
>   QLDB 기록은 문서 ID로 인덱싱되며, 이번에는 추가 기록 인덱스를 만들 수 없습니다. 시작 시간과 종료 시간을 포함하는 기록 쿼리는 날짜 범위 한정이라는 이점을 갖습니다.

2. 트랜스파일된 프로그램을 실행하려면 다음 명령을 입력합니다.

```
node dist/QueryHistory.js
```

`vehicle-registration` 원장의 문서 개정을 암호화 방식으로 검증하려면 7단계: 원장에 있는 문서 검증으로 진행하세요.

7단계: 원장에 있는 문서 검증

Amazon QLDB를 사용하면 SHA-256 암호화 해싱을 사용하여 원장 저널에 있는 문서의 무결성을 효율적으로 검증할 수 있습니다. QLDB에서 검증 및 암호화 해싱이 작동하는 방식에 대한 자세한 내용은 Amazon QLDB에서의 데이터 확인을 참조하십시오.

이 단계에서는 `vehicle-registration` 원장의 `VehicleRegistration` 테이블에 있는 문서 개정을 검증합니다. 먼저 다이제스트를 요청합니다. 다이제스트는 출력 파일로 반환되며 원장의 전체 변경

내역에 대한 서명 역할을 합니다. 그런 다음 해당 다이제스트와 관련된 개정 증거를 요청합니다. 이 증거를 사용하면 모든 유효성 검사를 통과한 경우 개정 내용의 무결성을 확인할 수 있습니다.

문서 개정을 검증하려면

1. 검증에 필요한 QLDB 객체가 들어 있는 다음 `.ts` 파일을 검토하세요.

    1. `BlockAddress.ts`

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { ValueHolder } from "aws-sdk/clients/qldb";
import { dom, IonTypes } from "ion-js";

export class BlockAddress {
    _strandId: string;
    _sequenceNo: number;

    constructor(strandId: string, sequenceNo: number) {
        this._strandId = strandId;
```

```
        this._sequenceNo = sequenceNo;
    }
}

/**
 * Convert a block address from an Ion value into a ValueHolder.
 * Shape of the ValueHolder must be: {'IonText': "{strandId: <"strandId">,
 sequenceNo: <sequenceNo>}"}
 * @param value The Ion value that contains the block address values to convert.
 * @returns The ValueHolder that contains the strandId and sequenceNo.
 */
export function blockAddressToValueHolder(value: dom.Value): ValueHolder {
    const blockAddressValue : dom.Value = getBlockAddressValue(value);
    const strandId: string = getStrandId(blockAddressValue);
    const sequenceNo: number = getSequenceNo(blockAddressValue);
    const valueHolder: string = `{strandId: "${strandId}", sequenceNo:
 ${sequenceNo}}`;
    const blockAddress: ValueHolder = {IonText: valueHolder};
    return blockAddress;
}

/**
 * Helper method that to get the Metadata ID.
 * @param value The Ion value.
 * @returns The Metadata ID.
 */
export function getMetadataId(value: dom.Value): string {
    const metaDataId: dom.Value = value.get("id");
    if (metaDataId === null) {
        throw new Error(`Expected field name id, but not found.`);
    }
    return metaDataId.stringValue();
}

/**
 * Helper method to get the Sequence No.
 * @param value The Ion value.
 * @returns The Sequence No.
 */
export function getSequenceNo(value : dom.Value): number {
    const sequenceNo: dom.Value = value.get("sequenceNo");
    if (sequenceNo === null) {
        throw new Error(`Expected field name sequenceNo, but not found.`);
    }
```

```
    return sequenceNo.numberValue();
}

/**
 * Helper method to get the Strand ID.
 * @param value The Ion value.
 * @returns The Strand ID.
 */
export function getStrandId(value: dom.Value): string {
    const strandId: dom.Value = value.get("strandId");
    if (strandId === null) {
        throw new Error(`Expected field name strandId, but not found.`);
    }
    return strandId.stringValue();
}

export function getBlockAddressValue(value: dom.Value) : dom.Value {
    const type = value.getType();
    if (type !== IonTypes.STRUCT) {
        throw new Error(`Unexpected format: expected struct, but got IonType:
 ${type.name}`);
    }
    const blockAddress: dom.Value = value.get("blockAddress");
    if (blockAddress == null) {
        throw new Error(`Expected field name blockAddress, but not found.`);
    }
    return blockAddress;
}
```

2. Verifier.ts

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
 * permit persons to whom the Software is furnished to do so.
```

```
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { Digest, ValueHolder } from "aws-sdk/clients/qldb";
import { createHash } from "crypto";
import { dom, toBase64 } from "ion-js";

import { getBlobValue } from "./Util";

const HASH_LENGTH: number = 32;
const UPPER_BOUND: number = 8;

/**
 * Build the candidate digest representing the entire ledger from the Proof
 hashes.
 * @param proof The Proof object.
 * @param leafHash The revision hash to pair with the first hash in the Proof
 hashes list.
 * @returns The calculated root hash.
 */
function buildCandidateDigest(proof: ValueHolder, leafHash: Uint8Array):
 Uint8Array {
    const parsedProof: Uint8Array[] = parseProof(proof);
    const rootHash: Uint8Array = calculateRootHashFromInternalHash(parsedProof,
 leafHash);
    return rootHash;
}

/**
 * Combine the internal hashes and the leaf hash until only one root hash
 remains.
 * @param internalHashes An array of hash values.
 * @param leafHash The revision hash to pair with the first hash in the Proof
 hashes list.
 * @returns The root hash constructed by combining internal hashes.
```

```
 */
function calculateRootHashFromInternalHash(internalHashes: Uint8Array[],
 leafHash: Uint8Array): Uint8Array {
    const rootHash: Uint8Array = internalHashes.reduce(joinHashesPairwise,
 leafHash);
    return rootHash;
}

/**
 * Compare two hash values by converting each Uint8Array byte, which is unsigned
 by default,
 * into a signed byte, assuming they are little endian.
 * @param hash1 The hash value to compare.
 * @param hash2 The hash value to compare.
 * @returns Zero if the hash values are equal, otherwise return the difference of
 the first pair of non-matching bytes.
 */
function compareHashValues(hash1: Uint8Array, hash2: Uint8Array): number {
    if (hash1.length !== HASH_LENGTH || hash2.length !== HASH_LENGTH) {
        throw new Error("Invalid hash.");
    }
    for (let i = hash1.length-1; i >= 0; i--) {
        const difference: number = (hash1[i]<<24 >>24) - (hash2[i]<<24 >>24);
        if (difference !== 0) {
            return difference;
        }
    }
    return 0;
}

/**
 * Helper method that concatenates two Uint8Array.
 * @param arrays List of array to concatenate, in the order provided.
 * @returns The concatenated array.
 */
function concatenate(...arrays: Uint8Array[]): Uint8Array {
    let totalLength = 0;
    for (const arr of arrays) {
        totalLength += arr.length;
    }
    const result = new Uint8Array(totalLength);
    let offset = 0;
    for (const arr of arrays) {
        result.set(arr, offset);
```

```
        offset += arr.length;
    }
    return result;
}

/**
 * Flip a single random bit in the given hash value.
 * This method is intended to be used for purpose of demonstrating the QLDB
 verification features only.
 * @param original The hash value to alter.
 * @returns The altered hash with a single random bit changed.
 */
export function flipRandomBit(original: any): Uint8Array {
    if (original.length === 0) {
        throw new Error("Array cannot be empty!");
    }
    const bytePos: number = Math.floor(Math.random() * original.length);
    const bitShift: number = Math.floor(Math.random() * UPPER_BOUND);
    const alteredHash: Uint8Array = original;

    alteredHash[bytePos] = alteredHash[bytePos] ^ (1 << bitShift);
    return alteredHash;
}

/**
 * Take two hash values, sort them, concatenate them, and generate a new hash
 value from the concatenated values.
 * @param h1 Byte array containing one of the hashes to compare.
 * @param h2 Byte array containing one of the hashes to compare.
 * @returns The concatenated array of hashes.
 */
export function joinHashesPairwise(h1: Uint8Array, h2: Uint8Array): Uint8Array {
    if (h1.length === 0) {
        return h2;
    }
    if (h2.length === 0) {
        return h1;
    }
    let concat: Uint8Array;
    if (compareHashValues(h1, h2) < 0) {
        concat = concatenate(h1, h2);
    } else {
        concat = concatenate(h2, h1);
    }
```

```
    const hash = createHash('sha256');
    hash.update(concat);
    const newDigest: Uint8Array = hash.digest();
    return newDigest;
}

/**
 * Parse the Block object returned by QLDB and retrieve block hash.
 * @param valueHolder A structure containing an Ion string value.
 * @returns The block hash.
 */
export function parseBlock(valueHolder: ValueHolder): Uint8Array {
    const block: dom.Value = dom.load(valueHolder.IonText);
    const blockHash: Uint8Array = getBlobValue(block, "blockHash");
    return blockHash;
}

/**
 * Parse the Proof object returned by QLDB into an iterator.
 * The Proof object returned by QLDB is a dictionary like the following:
 * {'IonText': '[{{<hash>}},{{<hash>}}]'}
 * @param valueHolder A structure containing an Ion string value.
 * @returns A list of hash values.
 */
function parseProof(valueHolder: ValueHolder): Uint8Array[] {
    const proofs : dom.Value = dom.load(valueHolder.IonText);
    return proofs.elements().map(proof => proof.uInt8ArrayValue());
}

/**
 *  Verify document revision against the provided digest.
 * @param documentHash The SHA-256 value representing the document revision to be
 verified.
 * @param digest The SHA-256 hash value representing the ledger digest.
 * @param proof The Proof object retrieved from GetRevision.getRevision.
 * @returns If the document revision verifies against the ledger digest.
 */
export function verifyDocument(documentHash: Uint8Array, digest: Digest, proof:
 ValueHolder): boolean {
    const candidateDigest = buildCandidateDigest(proof, documentHash);
    return (toBase64(<Uint8Array> digest) === toBase64(candidateDigest));
}
```

2. 두 .ts 프로그램(GetDigest.ts 및 GetRevision.ts)을 사용하여 다음 단계를 수행하세요.

- vehicle-registration 원장에 새 다이제스트를 요청하세요.
- VehicleRegistration 테이블에서 VIN 1N4AL11D75C109151이 포함된 문서의 각 개정에 대한 증거를 요청합니다.
- 반환된 다이제스트와 증거를 사용하여 다이제스트를 다시 계산하여 개정 내용을 검증합니다.

GetDigest.ts 프로그램에는 다음 코드가 포함되어 있습니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QLDB } from "aws-sdk";
import { GetDigestRequest, GetDigestResponse } from "aws-sdk/clients/qldb";

import { LEDGER_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { digestResponseToString } from "./qldb/Util";

/**
 * Get the digest of a ledger's journal.
```

```
 * @param ledgerName Name of the ledger to operate on.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with a GetDigestResponse.
 */
export async function getDigestResult(ledgerName: string, qldbClient: QLDB):
 Promise<GetDigestResponse> {
    const request: GetDigestRequest = {
        Name: ledgerName
    };
    const result: GetDigestResponse = await
 qldbClient.getDigest(request).promise();
    return result;
}

/**
 * This is an example for retrieving the digest of a particular ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbClient: QLDB = new QLDB();
        log(`Retrieving the current digest for ledger: ${LEDGER_NAME}.`);
        const digest: GetDigestResponse = await getDigestResult(LEDGER_NAME,
 qldbClient);
        log(`Success. Ledger digest: \n${digestResponseToString(digest)}.`);
    } catch (e) {
        error(`Unable to get a ledger digest: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> `getDigest` 함수를 사용하여 원장에 있는 저널의 현재 팁을 포함하는 다이제스트를 요청합니다. 저널 팁은 QLDB가 요청을 수신한 시점을 기준으로 가장 최근에 커밋된 블록을 나타냅니다.

`GetRevision.ts` 프로그램에는 다음 코드가 포함되어 있습니다.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { QLDB } from "aws-sdk";
import { Digest, GetDigestResponse, GetRevisionRequest, GetRevisionResponse,
 ValueHolder } from "aws-sdk/clients/qldb";
import { dom, toBase64 } from "ion-js";

import { getQldbDriver } from "./ConnectToLedger";
import { getDigestResult } from './GetDigest';
import { VEHICLE_REGISTRATION } from "./model/SampleData"
import { blockAddressToValueHolder, getMetadataId } from './qldb/BlockAddress';
import { LEDGER_NAME } from './qldb/Constants';
import { error, log } from "./qldb/LogUtil";
import { getBlobValue, valueHolderToString } from "./qldb/Util";
import { flipRandomBit, verifyDocument } from "./qldb/Verifier";

/**
 * Get the revision data object for a specified document ID and block address.
 * Also returns a proof of the specified revision for verification.
```

```
 * @param ledgerName Name of the ledger containing the document to query.
 * @param documentId Unique ID for the document to be verified, contained in the
 committed view of the document.
 * @param blockAddress The location of the block to request.
 * @param digestTipAddress The latest block location covered by the digest.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with a GetRevisionResponse.
 */
async function getRevision(
    ledgerName: string,
    documentId: string,
    blockAddress: ValueHolder,
    digestTipAddress: ValueHolder,
    qldbClient: QLDB
): Promise<GetRevisionResponse> {
    const request: GetRevisionRequest = {
        Name: ledgerName,
        BlockAddress: blockAddress,
        DocumentId: documentId,
        DigestTipAddress: digestTipAddress
    };
    const result: GetRevisionResponse = await
 qldbClient.getRevision(request).promise();
    return result;
}

/**
 * Query the table metadata for a particular vehicle for verification.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param vin VIN to query the table metadata of a specific registration with.
 * @returns Promise which fulfills with a list of Ion values that contains the
 results of the query.
 */
export async function lookupRegistrationForVin(txn: TransactionExecutor, vin:
 string): Promise<dom.Value[]> {
    log(`Querying the 'VehicleRegistration' table for VIN: ${vin}...`);
    let resultList: dom.Value[];
    const query: string = "SELECT blockAddress, metadata.id FROM
 _ql_committed_VehicleRegistration WHERE data.VIN = ?";

    await txn.execute(query, vin).then(function(result) {
      resultList = result.getResultList();
    });
    return resultList;
```

```
}

/**
 * Verify each version of the registration for the given VIN.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param ledgerName The ledger to get the digest from.
 * @param vin VIN to query the revision history of a specific registration with.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with void.
 * @throws Error: When verification fails.
 */
export async function verifyRegistration(
    txn: TransactionExecutor,
    ledgerName: string,
    vin: string,
    qldbClient: QLDB
): Promise<void> {
    log(`Let's verify the registration with VIN = ${vin}, in ledger =
 ${ledgerName}.`);
    const digest: GetDigestResponse = await getDigestResult(ledgerName,
 qldbClient);
    const digestBytes: Digest = digest.Digest;
    const digestTipAddress: ValueHolder = digest.DigestTipAddress;

    log(
        `Got a ledger digest: digest tip address = \n
${valueHolderToString(digestTipAddress)},
        digest = \n${toBase64(<Uint8Array> digestBytes)}.`
    );
    log(`Querying the registration with VIN = ${vin} to verify each version of the
 registration...`);
    const resultList: dom.Value[] = await lookupRegistrationForVin(txn, vin);
    log("Getting a proof for the document.");

    for (const result of resultList) {
        const blockAddress: ValueHolder =  blockAddressToValueHolder(result);
        const documentId: string = getMetadataId(result);

        const revisionResponse: GetRevisionResponse = await getRevision(
            ledgerName,
            documentId,
            blockAddress,
            digestTipAddress,
            qldbClient
```

```
        );

        const revision: dom.Value = dom.load(revisionResponse.Revision.IonText);
        const documentHash: Uint8Array = getBlobValue(revision, "hash");
        const proof: ValueHolder = revisionResponse.Proof;
        log(`Got back a proof: ${valueHolderToString(proof)}.`);

        let verified: boolean = verifyDocument(documentHash, digestBytes, proof);
        if (!verified) {
           throw new Error("Document revision is not verified.");
        } else {
            log("Success! The document is verified.");
        }
        const alteredDocumentHash: Uint8Array = flipRandomBit(documentHash);

        log(
            `Flipping one bit in the document's hash and assert that the document
 is NOT verified.
            The altered document hash is: ${toBase64(alteredDocumentHash)}`
        );
        verified = verifyDocument(alteredDocumentHash, digestBytes, proof);

        if (verified) {
            throw new Error("Expected altered document hash to not be verified
 against digest.");
        } else {
            log("Success! As expected flipping a bit in the document hash causes
 verification to fail.");
        }
        log(`Finished verifying the registration with VIN = ${vin} in ledger =
 ${ledgerName}.`);
    }
}

/**
 * Verify the integrity of a document revision in a QLDB ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbClient: QLDB = new QLDB();
        const qldbDriver: QldbDriver = getQldbDriver();

        const registration = VEHICLE_REGISTRATION[0];
```

```
        const vin: string = registration.VIN;

        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await verifyRegistration(txn, LEDGER_NAME, vin, qldbClient);
        });
    } catch (e) {
        error(`Unable to verify revision: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> getRevision 함수가 지정된 문서 개정에 대한 증거를 반환하면 이 프로그램은 클라이언트 측 API를 사용하여 해당 개정본을 확인합니다.

3. 트랜스파일된 프로그램을 실행하려면 다음 명령을 입력합니다.

```
node dist/GetRevision.js
```

vehicle-registration 원장을 더 이상 사용할 필요가 없는 경우 8단계(선택 사항): 리소스 정리로 진행하십시오.

8단계(선택 사항): 리소스 정리

vehicle-registration 원장을 계속 사용할 수 있습니다. 그러나 더 이상 필요하지 않은 경우 삭제해야 합니다.

원장을 삭제하려면

1. 다음 프로그램(DeleteLedger.ts)을 사용하여 vehicle-registration 원장과 모든 내용을 삭제하세요.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
```

```
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { isResourceNotFoundException } from "amazon-qldb-driver-nodejs";
import { AWSError, QLDB } from "aws-sdk";
import { DeleteLedgerRequest, DescribeLedgerRequest } from "aws-sdk/clients/qldb";

import { setDeletionProtection } from "./DeletionProtection";
import { LEDGER_NAME } from "./qldb/Constants";
import { error, log } from "./qldb/LogUtil";
import { sleep } from "./qldb/Util";

const LEDGER_DELETION_POLL_PERIOD_MS = 20000;

/**
 * Send a request to QLDB to delete the specified ledger.
 * @param ledgerName Name of the ledger to be deleted.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with void.
 */
export async function deleteLedger(ledgerName: string, qldbClient: QLDB):
 Promise<void> {
    log(`Attempting to delete the ledger with name: ${ledgerName}`);
    const request: DeleteLedgerRequest = {
        Name: ledgerName
    };
```

```
    await qldbClient.deleteLedger(request).promise();
    log("Success.");
}

/**
 * Wait for the ledger to be deleted.
 * @param ledgerName Name of the ledger to be deleted.
 * @param qldbClient The QLDB control plane client to use.
 * @returns Promise which fulfills with void.
 */
export async function waitForDeleted(ledgerName: string, qldbClient: QLDB):
 Promise<void> {
    log("Waiting for the ledger to be deleted...");
    const request: DescribeLedgerRequest = {
        Name: ledgerName
    };
    let isDeleted: boolean = false;
    while (true) {
        await qldbClient.describeLedger(request).promise().catch((error: AWSError)
 => {
            if (isResourceNotFoundException(error)) {
                isDeleted = true;
                log("Success. Ledger is deleted.");
            }
        });
        if (isDeleted) {
            break;
        }
        log("The ledger is still being deleted. Please wait...");
        await sleep(LEDGER_DELETION_POLL_PERIOD_MS);
    }
}

/**
 * Delete a ledger.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbClient: QLDB = new QLDB();
        await setDeletionProtection(LEDGER_NAME, qldbClient, false);
        await deleteLedger(LEDGER_NAME, qldbClient);
        await waitForDeleted(LEDGER_NAME, qldbClient);
    } catch (e) {
```

```
        error(`Unable to delete the ledger: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

> **Note**
>
> 원장에 대해 삭제 방지가 활성화된 경우 QLDB API를 사용하여 원장을 삭제하기 전에 먼저 이를 비활성화해야 합니다.

2. 트랜스파일된 프로그램을 실행하려면 다음 명령을 입력합니다.

```
node dist/DeleteLedger.js
```

Amazon QLDB Python 자습서

이 자습서 샘플 애플리케이션의 구현에서는 Amazon QLDB 드라이버를 AWS SDK for Python (Boto3)와 함께 사용하여 QLDB 원장을 생성하고 이를 샘플 데이터로 채웁니다.

이 자습서로 학습하면서 Python용 AWS SDK (Boto3) API 참조에서 QLDB 하위 수준 클라이언트를 참조하여 관리 API 작업을 수행할 수 있습니다. 트랜잭션 데이터 작업의 경우 Python용 QLDB 드라이버 API 참조를 참조할 수 있습니다.

> **Note**
>
> 해당하는 경우 일부 자습서 단계에는 Python용 QLDB 드라이버의 지원되는 각 메이저 버전마다 서로 다른 명령 또는 코드 예제가 있습니다.

주제

- Amazon QLDB Python 샘플 애플리케이션 설치
- 1단계: 새 원장 생성
- 2단계: 원장과의 연결을 테스트
- 3단계: 테이블, 인덱스 및 샘플 데이터 생성

- 4단계: 원장에서 테이블 쿼리
- 5단계: 원장의 문서 수정
- 6단계: 문서의 개정 기록 보기
- 7단계: 원장에 있는 문서 검증
- 8단계(선택 사항): 리소스 정리

Amazon QLDB Python 샘플 애플리케이션 설치

이 섹션에서는 단계별 Python 자습서를 위해 제공된 Amazon QLDB 샘플 애플리케이션을 설치하고 실행하는 방법을 설명합니다. 이 샘플 애플리케이션의 사용 사례는 차량 등록에 대한 전체 기록 정보를 추적하는 자동차 부서(DMV) 데이터베이스입니다.

Python용 DMV 샘플 애플리케이션은 GitHub 리포지토리 aws-samples/amazon-qldb-dmv-sample-python 내 오픈 소스입니다.

사전 조건

시작하기 전에 Python 필수 조건용 QLDB 드라이버를 완료했는지 확인합니다. 여기에는 Python 설치 및 다음 작업이 포함됩니다.

1. AWS에 가입합니다.
2. 적절한 QLDB 권한을 가진 사용자를 생성합니다.
3. 개발을 위한 프로그래밍 방식 액세스 권한을 부여합니다.

이 자습서의 모든 단계를 완료하려면 QLDB API를 통해 원장 리소스에 대한 전체 관리 액세스 권한이 필요합니다.

설치

샘플 애플리케이션을 설치하려면

1. 다음 `pip` 명령을 입력하여 GitHub에서 샘플 애플리케이션을 복제하고 설치합니다.

   3.x

   ```
   pip install git+https://github.com/aws-samples/amazon-qldb-dmv-sample-python.git
   ```

2.x

```
pip install git+https://github.com/aws-samples/amazon-qldb-dmv-sample-
python.git@v1.0.0
```

샘플 애플리케이션은 이 자습서의 전체 소스 코드와 Python 드라이버 및 AWS SDK for Python (Boto3)을 포함한 해당 종속성을 패키징합니다.

2. 명령줄에서 코드를 실행하기 전에 현재 작업 디렉터리를 `pyqldbsamples` 패키지가 설치된 위치로 전환하십시오. 다음 명령을 입력합니다.

```
cd $(python -c "import pyqldbsamples; print(pyqldbsamples.__path__[0])")
```

3. 1단계: 새 원장 생성로 진행하여 자습서를 시작하고 원장을 생성하십시오.

1단계: 새 원장 생성

이 단계에서는 `vehicle-registration`라는 이름의 새 Amazon QLDB 원장을 생성합니다

새 원장을 생성하려면

1. 이 자습서의 다른 모든 프로그램에서 사용하는 상수 값이 들어 있는 다음 파일(`constants.py`)을 검토하십시오.

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
```

```
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.


class Constants:
    """
    Constant values used throughout this tutorial.
    """
    LEDGER_NAME = "vehicle-registration"

    VEHICLE_REGISTRATION_TABLE_NAME = "VehicleRegistration"
    VEHICLE_TABLE_NAME = "Vehicle"
    PERSON_TABLE_NAME = "Person"
    DRIVERS_LICENSE_TABLE_NAME = "DriversLicense"

    LICENSE_NUMBER_INDEX_NAME = "LicenseNumber"
    GOV_ID_INDEX_NAME = "GovId"
    VEHICLE_VIN_INDEX_NAME = "VIN"
    LICENSE_PLATE_NUMBER_INDEX_NAME = "LicensePlateNumber"
    PERSON_ID_INDEX_NAME = "PersonId"

    JOURNAL_EXPORT_S3_BUCKET_NAME_PREFIX = "qldb-tutorial-journal-export"
    USER_TABLES = "information_schema.user_tables"
    S3_BUCKET_ARN_TEMPLATE = "arn:aws:s3:::"
    LEDGER_NAME_WITH_TAGS = "tags"

    RETRY_LIMIT = 4
```

2. 다음 프로그램(`create_ledger.py`)을 사용하여 이름이 `vehicle-registration`로 지정된 원장을 생성합니다.

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
```

```
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO
from time import sleep

from boto3 import client

from pyqldbsamples.constants import Constants

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')

LEDGER_CREATION_POLL_PERIOD_SEC = 10
ACTIVE_STATE = "ACTIVE"


def create_ledger(name):
    """
    Create a new ledger with the specified name.

    :type name: str
    :param name: Name for the ledger to be created.

    :rtype: dict
    :return: Result from the request.
    """
    logger.info("Let's create the ledger named: {}...".format(name))
    result = qldb_client.create_ledger(Name=name, PermissionsMode='ALLOW_ALL')
    logger.info('Success. Ledger state: {}.'.format(result.get('State')))
    return result
```

```
def wait_for_active(name):
    """
    Wait for the newly created ledger to become active.

    :type name: str
    :param name: The ledger to check on.

    :rtype: dict
    :return: Result from the request.
    """
    logger.info('Waiting for ledger to become active...')
    while True:
        result = qldb_client.describe_ledger(Name=name)
        if result.get('State') == ACTIVE_STATE:
            logger.info('Success. Ledger is active and ready to use.')
            return result
        logger.info('The ledger is still creating. Please wait...')
        sleep(LEDGER_CREATION_POLL_PERIOD_SEC)


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Create a ledger and wait for it to be active.
    """
    try:
        create_ledger(ledger_name)
        wait_for_active(ledger_name)
    except Exception as e:
        logger.exception('Unable to create the ledger!')
        raise e


if __name__ == '__main__':
    main()
```

> **Note**
>
> - `create_ledger` 호출 시 원장 이름과 권한 모드를 지정해야 합니다. 원장 데이터의 보안을 극대화하려면 `STANDARD` 권한 모드를 사용할 것을 권장합니다.

- 원장을 생성할 때 기본적으로 삭제 방지가 활성화됩니다. 이 기능은 사용자가 원장을 삭제하는 것을 방지하는 QLDB의 기능입니다. QLDB API 또는 AWS Command Line Interface(AWS CLI)를 사용하여 원장 생성 시 삭제 보호를 비활성화할 수 있습니다.
- 선택적으로, 원장에 첨부할 태그를 지정할 수도 있습니다.

3. 프로그램을 실행하려면 다음 명령을 입력합니다.

```
python create_ledger.py
```

새 원장과의 연결을 확인하려면 2단계: 원장과의 연결을 테스트로 이동하십시오.

2단계: 원장과의 연결을 테스트

이 단계에서는 트랜잭션 데이터 API 엔드포인트를 사용하여 Amazon QLDB의 `vehicle-registration` 원장에 연결할 수 있는지 확인합니다.

원장과의 연결을 테스트하려면

1. 다음 프로그램(`connect_to_ledger.py`)을 사용하여 `vehicle-registration` 원장에 대한 데이터 세션 연결을 생성합니다.

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
```

```
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from botocore.exceptions import ClientError

from pyqldb.driver.qldb_driver import QldbDriver
from pyqldbsamples.constants import Constants

logger = getLogger(__name__)
basicConfig(level=INFO)


def create_qldb_driver(ledger_name=Constants.LEDGER_NAME, region_name=None,
 endpoint_url=None, boto3_session=None):
    """
    Create a QLDB driver for executing transactions.

    :type ledger_name: str
    :param ledger_name: The QLDB ledger name.

    :type region_name: str
    :param region_name: See [1].

    :type endpoint_url: str
    :param endpoint_url: See [1].

    :type boto3_session: :py:class:`boto3.session.Session`
    :param boto3_session: The boto3 session to create the client with (see [1]).

    :rtype: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :return: A QLDB driver object.

    [1]: `Boto3 Session.client Reference <https://
boto3.amazonaws.com/v1/documentation/api/latest/reference/core/
session.html#boto3.session.Session.client>`.
    """
```

```
    qldb_driver = QldbDriver(ledger_name=ledger_name, region_name=region_name,
 endpoint_url=endpoint_url,
                             boto3_session=boto3_session)
    return qldb_driver


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Connect to a given ledger using default settings.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            logger.info('Listing table names ')
            for table in driver.list_tables():
                logger.info(table)
    except ClientError as ce:
        logger.exception('Unable to list tables.')
        raise ce


if __name__ == '__main__':
    main()
```

> **Note**
>
> - 원장에서 데이터 트랜잭션을 실행하려면 특정 원장에 연결할 QLDB 드라이버 객체를 생성해야 합니다. 이는 이전 단계에서 원장을 생성할 때 사용한 `qldb_client` 객체와는 다른 클라이언트 객체입니다. 이전 클라이언트는 Amazon QLDB API 참조에 나열된 관리 API 작업을 실행하는 데만 사용됩니다.
> - 이 드라이버 객체를 생성할 때 원장 이름을 지정해야 합니다.

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
```

```
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from botocore.exceptions import ClientError

from pyqldb.driver.pooled_qldb_driver import PooledQldbDriver
from pyqldbsamples.constants import Constants

logger = getLogger(__name__)
basicConfig(level=INFO)


def create_qldb_driver(ledger_name=Constants.LEDGER_NAME, region_name=None,
 endpoint_url=None, boto3_session=None):
    """
    Create a QLDB driver for creating sessions.

    :type ledger_name: str
    :param ledger_name: The QLDB ledger name.

    :type region_name: str
    :param region_name: See [1].

    :type endpoint_url: str
    :param endpoint_url: See [1].

    :type boto3_session: :py:class:`boto3.session.Session`
```

```
    :param boto3_session: The boto3 session to create the client with (see [1]).

    :rtype: :py:class:`pyqldb.driver.pooled_qldb_driver.PooledQldbDriver`
    :return: A pooled QLDB driver object.

    [1]: `Boto3 Session.client Reference <https://
boto3.amazonaws.com/v1/documentation/api/latest/reference/core/
session.html#boto3.session.Session.client>`.
    """
    qldb_driver = PooledQldbDriver(ledger_name=ledger_name,
 region_name=region_name, endpoint_url=endpoint_url,
                                   boto3_session=boto3_session)
    return qldb_driver


def create_qldb_session():
    """
    Retrieve a QLDB session object.

    :rtype: :py:class:`pyqldb.session.pooled_qldb_session.PooledQldbSession`
    :return: A pooled QLDB session object.
    """
    qldb_session = pooled_qldb_driver.get_session()
    return qldb_session


pooled_qldb_driver = create_qldb_driver()


if __name__ == '__main__':
    """
    Connect to a session for a given ledger using default settings.
    """
    try:
        qldb_session = create_qldb_session()
        logger.info('Listing table names ')
        for table in qldb_session.list_tables():
            logger.info(table)
    except ClientError:
        logger.exception('Unable to create session.')
```

> **Note**
>
> - 원장에서 데이터 트랜잭션을 실행하려면 특정 원장에 연결할 QLDB 드라이버 객체를 생성해야 합니다. 이는 이전 단계에서 원장을 생성할 때 사용한 `qldb_client` 객체와는 다른 클라이언트 객체입니다. 이전 클라이언트는 Amazon QLDB API 참조에 나열된 관리 API 작업을 실행하는 데만 사용됩니다.
> - 먼저 풀링된 QLDB 드라이버 객체를 생성합니다. 이 드라이버를 생성할 때 원장 이름을 지정해야 합니다.
> - 그러면 이 풀링된 드라이버 객체에서 세션을 생성할 수 있습니다.

2. 프로그램을 실행하려면 다음 명령을 입력합니다.

```
python connect_to_ledger.py
```

`vehicle-registration` 원장에 테이블을 생성하려면 3단계: 테이블, 인덱스 및 샘플 데이터 생성로 진행하십시오.

3단계: 테이블, 인덱스 및 샘플 데이터 생성

Amazon QLDB 원장이 활성화되고 연결을 수락하면 차량, 소유자 및 등록 정보에 대한 데이터 테이블 생성을 시작할 수 있습니다. 테이블과 인덱스를 생성한 후 데이터를 로드할 수 있습니다.

이 단계에서는 `vehicle-registration` 원장에 4개의 테이블을 생성합니다.

- `VehicleRegistration`
- `Vehicle`
- `Person`
- `DriversLicense`

또한 다음과 같은 인덱스를 생성합니다.

테이블 이름	필드
`VehicleRegistration`	`VIN`

테이블 이름	필드
VehicleRegistration	LicensePlateNumber
Vehicle	VIN
Person	GovId
DriversLicense	LicenseNumber
DriversLicense	PersonId

샘플 데이터를 삽입할 때는 먼저 Person 테이블에 문서를 삽입합니다. 그런 다음 각 Person 문서에서 시스템이 할당한 id를 사용하여 적절한 VehicleRegistration 및 DriversLicense 문서의 해당 필드를 채웁니다.

> **Tip**
>
> 가장 좋은 방법은 문서의 시스템 할당 id를 외래 키로 사용하는 것입니다. 고유 식별자(예: 차량의 VIN)로 사용되는 필드를 정의할 수 있지만 문서의 실제 고유 식별자는 id입니다. 이 필드는 문서의 메타데이터에 포함되며 커밋된 뷰(테이블의 시스템 정의 뷰)에서 쿼리할 수 있습니다.
>
> QLDB 뷰에 대한 자세한 내용은 핵심 개념 섹션을 참조하세요. 메타데이터에 대해 자세히 알아보려면 문서 메타데이터 쿼리 섹션을 참조하세요.

테이블 및 인덱스를 생성하려면

1. 다음 프로그램(create_table.py)을 사용하여 앞서 언급한 테이블을 생성합니다.

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
```

```
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)


def create_table(driver, table_name):
    """
    Create a table with the specified name.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type table_name: str
    :param table_name: Name of the table to create.

    :rtype: int
    :return: The number of changes to the database.
    """
    logger.info("Creating the '{}' table...".format(table_name))
    statement = 'CREATE TABLE {}'.format(table_name)
    cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(statement))
```

```
    logger.info('{} table created successfully.'.format(table_name))
    return len(list(cursor))


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Create registrations, vehicles, owners, and licenses tables.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            create_table(driver, Constants.DRIVERS_LICENSE_TABLE_NAME)
            create_table(driver, Constants.PERSON_TABLE_NAME)
            create_table(driver, Constants.VEHICLE_TABLE_NAME)
            create_table(driver, Constants.VEHICLE_REGISTRATION_TABLE_NAME)
            logger.info('Tables created successfully.')
    except Exception as e:
        logger.exception('Errors creating tables.')
        raise e


if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
```

```
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)


def create_table(transaction_executor, table_name):
    """
    Create a table with the specified name using an Executor object.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type table_name: str
    :param table_name: Name of the table to create.

    :rtype: int
    :return: The number of changes to the database.
    """
    logger.info("Creating the '{}' table...".format(table_name))
    statement = 'CREATE TABLE {}'.format(table_name)
    cursor = transaction_executor.execute_statement(statement)
    logger.info('{} table created successfully.'.format(table_name))
    return len(list(cursor))


if __name__ == '__main__':
    """
    Create registrations, vehicles, owners, and licenses tables in a single
 transaction.
    """
    try:
        with create_qldb_session() as session:
```

```
            session.execute_lambda(lambda x: create_table(x,
Constants.DRIVERS_LICENSE_TABLE_NAME) and
                                   create_table(x, Constants.PERSON_TABLE_NAME)
and
                                   create_table(x, Constants.VEHICLE_TABLE_NAME)
and
                                   create_table(x,
Constants.VEHICLE_REGISTRATION_TABLE_NAME),
                                   lambda retry_attempt: logger.info('Retrying
due to OCC conflict...'))
            logger.info('Tables created successfully.')
    except Exception:
        logger.exception('Errors creating tables.')
```

> **Note**
>
> 이 프로그램은 `execute_lambda` 함수의 사용 방법을 설명합니다. 이 예제에서는 단일 Lambda 표현식을 사용하여 여러 `CREATE TABLE` PartiQL 문을 실행합니다.
> 이 실행 함수는 암시적으로 트랜잭션을 시작하고 Lambda에서 모든 문을 실행한 다음 트랜잭션을 자동 커밋합니다.

2. 프로그램을 실행하려면 다음 명령을 입력합니다.

```
python create_table.py
```

3. 앞에서 설명한 대로 다음 프로그램(`create_index.py`)을 사용하여 테이블에 인덱스를 생성합니다.

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
```

```
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)


def create_index(driver, table_name, index_attribute):
    """
    Create an index for a particular table.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type table_name: str
    :param table_name: Name of the table to add indexes for.

    :type index_attribute: str
    :param index_attribute: Index to create on a single attribute.

    :rtype: int
    :return: The number of changes to the database.
    """
    logger.info("Creating index on '{}'...".format(index_attribute))
    statement = 'CREATE INDEX on {} ({})'.format(table_name, index_attribute)
    cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(statement))
    return len(list(cursor))
```

```
def main(ledger_name=Constants.LEDGER_NAME):
    """
    Create indexes on tables in a particular ledger.
    """
    logger.info('Creating indexes on all tables...')
    try:
        with create_qldb_driver(ledger_name) as driver:
            create_index(driver, Constants.PERSON_TABLE_NAME,
 Constants.GOV_ID_INDEX_NAME)
            create_index(driver, Constants.VEHICLE_TABLE_NAME,
 Constants.VEHICLE_VIN_INDEX_NAME)
            create_index(driver, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
 Constants.LICENSE_PLATE_NUMBER_INDEX_NAME)
            create_index(driver, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
 Constants.VEHICLE_VIN_INDEX_NAME)
            create_index(driver, Constants.DRIVERS_LICENSE_TABLE_NAME,
 Constants.PERSON_ID_INDEX_NAME)
            create_index(driver, Constants.DRIVERS_LICENSE_TABLE_NAME,
 Constants.LICENSE_NUMBER_INDEX_NAME)
            logger.info('Indexes created successfully.')
    except Exception as e:
        logger.exception('Unable to create indexes.')
        raise e


if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
```

```
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)


def create_index(transaction_executor, table_name, index_attribute):
    """
    Create an index for a particular table.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type table_name: str
    :param table_name: Name of the table to add indexes for.

    :type index_attribute: str
    :param index_attribute: Index to create on a single attribute.

    :rtype: int
    :return: The number of changes to the database.
    """
    logger.info("Creating index on '{}'...".format(index_attribute))
    statement = 'CREATE INDEX on {} ({})'.format(table_name, index_attribute)
    cursor = transaction_executor.execute_statement(statement)
    return len(list(cursor))
```

```
if __name__ == '__main__':
    """
    Create indexes on tables in a particular ledger.
    """
    logger.info('Creating indexes on all tables in a single transaction...')
    try:
        with create_qldb_session() as session:
            session.execute_lambda(lambda x: create_index(x,
 Constants.PERSON_TABLE_NAME,

 Constants.GOV_ID_INDEX_NAME)
                                   and create_index(x,
 Constants.VEHICLE_TABLE_NAME,

 Constants.VEHICLE_VIN_INDEX_NAME)
                                   and create_index(x,
 Constants.VEHICLE_REGISTRATION_TABLE_NAME,

 Constants.LICENSE_PLATE_NUMBER_INDEX_NAME)
                                   and create_index(x,
 Constants.VEHICLE_REGISTRATION_TABLE_NAME,

 Constants.VEHICLE_VIN_INDEX_NAME)
                                   and create_index(x,
 Constants.DRIVERS_LICENSE_TABLE_NAME,

 Constants.PERSON_ID_INDEX_NAME)
                                   and create_index(x,
 Constants.DRIVERS_LICENSE_TABLE_NAME,

 Constants.LICENSE_NUMBER_INDEX_NAME),
                                   lambda retry_attempt: logger.info('Retrying
 due to OCC conflict...'))
            logger.info('Indexes created successfully.')
    except Exception:
        logger.exception('Unable to create indexes.')
```

4. 프로그램을 실행하려면 다음 명령을 입력합니다.

```
python create_index.py
```

데이터를 테이블로 로드하려면

1. `vehicle-registration` 테이블에 삽입한 샘플 데이터를 나타내는 다음 파일(`sample_data.py`)을 검토하십시오. 또한 이 파일은 `amazon.ion` 패키지에서 가져와서 Amazon Ion 데이터를 변환, 구문 분석 및 인쇄하는 도우미 함수를 제공합니다.

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
from datetime import datetime
from decimal import Decimal
from logging import basicConfig, getLogger, INFO

from amazon.ion.simple_types import IonPyBool, IonPyBytes, IonPyDecimal, IonPyDict,
 IonPyFloat, IonPyInt, IonPyList, \
    IonPyNull, IonPySymbol, IonPyText, IonPyTimestamp
from amazon.ion.simpleion import dumps, loads

logger = getLogger(__name__)
basicConfig(level=INFO)
IonValue = (IonPyBool, IonPyBytes, IonPyDecimal, IonPyDict, IonPyFloat, IonPyInt,
 IonPyList, IonPyNull, IonPySymbol,
            IonPyText, IonPyTimestamp)
```

```
class SampleData:
    """
    Sample domain objects for use throughout this tutorial.
    """
    DRIVERS_LICENSE = [
        {
            'PersonId': '',
            'LicenseNumber': 'LEWISR261LL',
            'LicenseType': 'Learner',
            'ValidFromDate': datetime(2016, 12, 20),
            'ValidToDate': datetime(2020, 11, 15)
        },
        {
            'PersonId': '',
            'LicenseNumber': 'LOGANB486CG',
            'LicenseType': 'Probationary',
            'ValidFromDate': datetime(2016, 4, 6),
            'ValidToDate': datetime(2020, 11, 15)
        },
        {
            'PersonId': '',
            'LicenseNumber': '744 849 301',
            'LicenseType': 'Full',
            'ValidFromDate': datetime(2017, 12, 6),
            'ValidToDate': datetime(2022, 10, 15)
        },
        {
            'PersonId': '',
            'LicenseNumber': 'P626-168-229-765',
            'LicenseType': 'Learner',
            'ValidFromDate': datetime(2017, 8, 16),
            'ValidToDate': datetime(2021, 11, 15)
        },
        {
            'PersonId': '',
            'LicenseNumber': 'S152-780-97-415-0',
            'LicenseType': 'Probationary',
            'ValidFromDate': datetime(2015, 8, 15),
            'ValidToDate': datetime(2021, 8, 21)
        }
    ]
    PERSON = [
        {
            'FirstName': 'Raul',
```

```
            'LastName': 'Lewis',
            'Address': '1719 University Street, Seattle, WA, 98109',
            'DOB': datetime(1963, 8, 19),
            'GovId': 'LEWISR261LL',
            'GovIdType': 'Driver License'
        },
        {
            'FirstName': 'Brent',
            'LastName': 'Logan',
            'DOB': datetime(1967, 7, 3),
            'Address': '43 Stockert Hollow Road, Everett, WA, 98203',
            'GovId': 'LOGANB486CG',
            'GovIdType': 'Driver License'
        },
        {
            'FirstName': 'Alexis',
            'LastName': 'Pena',
            'DOB': datetime(1974, 2, 10),
            'Address': '4058 Melrose Street, Spokane Valley, WA, 99206',
            'GovId': '744 849 301',
            'GovIdType': 'SSN'
        },
        {
            'FirstName': 'Melvin',
            'LastName': 'Parker',
            'DOB': datetime(1976, 5, 22),
            'Address': '4362 Ryder Avenue, Seattle, WA, 98101',
            'GovId': 'P626-168-229-765',
            'GovIdType': 'Passport'
        },
        {
            'FirstName': 'Salvatore',
            'LastName': 'Spencer',
            'DOB': datetime(1997, 11, 15),
            'Address': '4450 Honeysuckle Lane, Seattle, WA, 98101',
            'GovId': 'S152-780-97-415-0',
            'GovIdType': 'Passport'
        }
    ]
    VEHICLE = [
        {
            'VIN': '1N4AL11D75C109151',
            'Type': 'Sedan',
            'Year': 2011,
```

```
            'Make': 'Audi',
            'Model': 'A5',
            'Color': 'Silver'
        },
        {
            'VIN': 'KM8SRDHF6EU074761',
            'Type': 'Sedan',
            'Year': 2015,
            'Make': 'Tesla',
            'Model': 'Model S',
            'Color': 'Blue'
        },
        {
            'VIN': '3HGGK5G53FM761765',
            'Type': 'Motorcycle',
            'Year': 2011,
            'Make': 'Ducati',
            'Model': 'Monster 1200',
            'Color': 'Yellow'
        },
        {
            'VIN': '1HVBBAANXWH544237',
            'Type': 'Semi',
            'Year': 2009,
            'Make': 'Ford',
            'Model': 'F 150',
            'Color': 'Black'
        },
        {
            'VIN': '1C4RJFAG0FC625797',
            'Type': 'Sedan',
            'Year': 2019,
            'Make': 'Mercedes',
            'Model': 'CLK 350',
            'Color': 'White'
        }
    ]
    VEHICLE_REGISTRATION = [
        {
            'VIN': '1N4AL11D75C109151',
            'LicensePlateNumber': 'LEWISR261LL',
            'State': 'WA',
            'City': 'Seattle',
            'ValidFromDate': datetime(2017, 8, 21),
```

```
            'ValidToDate': datetime(2020, 5, 11),
            'PendingPenaltyTicketAmount': Decimal('90.25'),
            'Owners': {
                'PrimaryOwner': {'PersonId': ''},
                'SecondaryOwners': []
            }
        },
        {
            'VIN': 'KM8SRDHF6EU074761',
            'LicensePlateNumber': 'CA762X',
            'State': 'WA',
            'City': 'Kent',
            'PendingPenaltyTicketAmount': Decimal('130.75'),
            'ValidFromDate': datetime(2017, 9, 14),
            'ValidToDate': datetime(2020, 6, 25),
            'Owners': {
                'PrimaryOwner': {'PersonId': ''},
                'SecondaryOwners': []
            }
        },
        {
            'VIN': '3HGGK5G53FM761765',
            'LicensePlateNumber': 'CD820Z',
            'State': 'WA',
            'City': 'Everett',
            'PendingPenaltyTicketAmount': Decimal('442.30'),
            'ValidFromDate': datetime(2011, 3, 17),
            'ValidToDate': datetime(2021, 3, 24),
            'Owners': {
                'PrimaryOwner': {'PersonId': ''},
                'SecondaryOwners': []
            }
        },
        {
            'VIN': '1HVBBAANXWH544237',
            'LicensePlateNumber': 'LS477D',
            'State': 'WA',
            'City': 'Tacoma',
            'PendingPenaltyTicketAmount': Decimal('42.20'),
            'ValidFromDate': datetime(2011, 10, 26),
            'ValidToDate': datetime(2023, 9, 25),
            'Owners': {
                'PrimaryOwner': {'PersonId': ''},
                'SecondaryOwners': []
```

```
            }
        },
        {
            'VIN': '1C4RJFAG0FC625797',
            'LicensePlateNumber': 'TH393F',
            'State': 'WA',
            'City': 'Olympia',
            'PendingPenaltyTicketAmount': Decimal('30.45'),
            'ValidFromDate': datetime(2013, 9, 2),
            'ValidToDate': datetime(2024, 3, 19),
            'Owners': {
                'PrimaryOwner': {'PersonId': ''},
                'SecondaryOwners': []
            }
        }
    ]


def convert_object_to_ion(py_object):
    """
    Convert a Python object into an Ion object.

    :type py_object: object
    :param py_object: The object to convert.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyValue`
    :return: The converted Ion object.
    """
    ion_object = loads(dumps(py_object))
    return ion_object


def to_ion_struct(key, value):
    """
    Convert the given key and value into an Ion struct.

    :type key: str
    :param key: The key which serves as an unique identifier.

    :type value: str
    :param value: The value associated with a given key.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyDict`
    :return: The Ion dictionary object.
```

```
    """
    ion_struct = dict()
    ion_struct[key] = value
    return loads(str(ion_struct))


def get_document_ids(transaction_executor, table_name, field, value):
    """
    Gets the document IDs from the given table.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type table_name: str
    :param table_name: The table name to query.

    :type field: str
    :param field: A field to query.

    :type value: str
    :param value: The key of the given field.

    :rtype: list
    :return: A list of document IDs.
    """
    query = "SELECT id FROM {} AS t BY id WHERE t.{} = ?".format(table_name, field)
    cursor = transaction_executor.execute_statement(query,
 convert_object_to_ion(value))
    return list(map(lambda table: table.get('id'), cursor))


def get_document_ids_from_dml_results(result):
    """
    Return a list of modified document IDs as strings from DML results.

    :type result: :py:class:`pyqldb.cursor.buffered_cursor.BufferedCursor`
    :param: result: The result set from DML operation.

    :rtype: list
    :return: List of document IDs.
    """
    ret_val = list(map(lambda x: x.get('documentId'), result))
    return ret_val
```

```
def print_result(cursor):
    """
    Pretty print the result set. Returns the number of documents in the result set.

    :type cursor: :py:class:`pyqldb.cursor.stream_cursor.StreamCursor`/
                  :py:class:`pyqldb.cursor.buffered_cursor.BufferedCursor`
    :param cursor: An instance of the StreamCursor or BufferedCursor class.

    :rtype: int
    :return: Number of documents in the result set.
    """
    result_counter = 0
    for row in cursor:
        # Each row would be in Ion format.
        print_ion(row)
        result_counter += 1
    return result_counter


def print_ion(ion_value):
    """
    Pretty print an Ion Value.

    :type ion_value: :py:class:`amazon.ion.simple_types.IonPySymbol`
    :param ion_value: Any Ion Value to be pretty printed.
    """
    logger.info(dumps(ion_value, binary=False, indent='  ',
 omit_version_marker=True))
```

> **Note**
>
> 이 `get_document_ids` 함수는 테이블에서 시스템 할당 문서 ID를 반환하는 쿼리를 실행합니다. 자세한 내용은 BY 절을 사용하여 문서 ID 쿼리하기 섹션을 참조하세요.

2. 다음 프로그램(`insert_document.py`)을 사용하여 샘플 데이터를 테이블에 삽입합니다.

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
```

```
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.constants import Constants
from pyqldbsamples.model.sample_data import convert_object_to_ion, SampleData,
 get_document_ids_from_dml_results
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)


def update_person_id(document_ids):
    """
    Update the PersonId value for DriversLicense records and the PrimaryOwner
 value for VehicleRegistration records.

    :type document_ids: list
    :param document_ids: List of document IDs.

    :rtype: list
```

```
    :return: Lists of updated DriversLicense records and updated
 VehicleRegistration records.
    """
    new_drivers_licenses = SampleData.DRIVERS_LICENSE.copy()
    new_vehicle_registrations = SampleData.VEHICLE_REGISTRATION.copy()
    for i in range(len(SampleData.PERSON)):
        drivers_license = new_drivers_licenses[i]
        registration = new_vehicle_registrations[i]
        drivers_license.update({'PersonId': str(document_ids[i])})
        registration['Owners']['PrimaryOwner'].update({'PersonId':
 str(document_ids[i])})
    return new_drivers_licenses, new_vehicle_registrations


def insert_documents(driver, table_name, documents):
    """
    Insert the given list of documents into a table in a single transaction.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type table_name: str
    :param table_name: Name of the table to insert documents into.

    :type documents: list
    :param documents: List of documents to insert.

    :rtype: list
    :return: List of documents IDs for the newly inserted documents.
    """
    logger.info('Inserting some documents in the {}
 table...'.format(table_name))
    statement = 'INSERT INTO {} ?'.format(table_name)
    cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(statement,

 convert_object_to_ion(documents)))
    list_of_document_ids = get_document_ids_from_dml_results(cursor)

    return list_of_document_ids


def update_and_insert_documents(driver):
    """
```

```
    Handle the insertion of documents and updating PersonIds.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.
    """
    list_ids = insert_documents(driver, Constants.PERSON_TABLE_NAME,
 SampleData.PERSON)

    logger.info("Updating PersonIds for 'DriversLicense' and PrimaryOwner for
 'VehicleRegistration'...")
    new_licenses, new_registrations = update_person_id(list_ids)

    insert_documents(driver, Constants.VEHICLE_TABLE_NAME, SampleData.VEHICLE)
    insert_documents(driver, Constants.VEHICLE_REGISTRATION_TABLE_NAME,
 new_registrations)
    insert_documents(driver, Constants.DRIVERS_LICENSE_TABLE_NAME, new_licenses)


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Insert documents into a table in a QLDB ledger.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            # An INSERT statement creates the initial revision of a document
 with a version number of zero.
            # QLDB also assigns a unique document identifier in GUID format as
 part of the metadata.
            update_and_insert_documents(driver)
            logger.info('Documents inserted successfully!')
    except Exception as e:
        logger.exception('Error inserting or updating documents.')
        raise e


if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
```

```
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.constants import Constants
from pyqldbsamples.model.sample_data import convert_object_to_ion, SampleData,
 get_document_ids_from_dml_results
from pyqldbsamples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)


def update_person_id(document_ids):
    """
    Update the PersonId value for DriversLicense records and the PrimaryOwner
 value for VehicleRegistration records.

    :type document_ids: list
    :param document_ids: List of document IDs.

    :rtype: list
    :return: Lists of updated DriversLicense records and updated
 VehicleRegistration records.
```

```
    """
    new_drivers_licenses = SampleData.DRIVERS_LICENSE.copy()
    new_vehicle_registrations = SampleData.VEHICLE_REGISTRATION.copy()
    for i in range(len(SampleData.PERSON)):
        drivers_license = new_drivers_licenses[i]
        registration = new_vehicle_registrations[i]
        drivers_license.update({'PersonId': str(document_ids[i])})
        registration['Owners']['PrimaryOwner'].update({'PersonId':
 str(document_ids[i])})
    return new_drivers_licenses, new_vehicle_registrations


def insert_documents(transaction_executor, table_name, documents):
    """
    Insert the given list of documents into a table in a single transaction.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type table_name: str
    :param table_name: Name of the table to insert documents into.

    :type documents: list
    :param documents: List of documents to insert.

    :rtype: list
    :return: List of documents IDs for the newly inserted documents.
    """
    logger.info('Inserting some documents in the {}
 table...'.format(table_name))
    statement = 'INSERT INTO {} ?'.format(table_name)
    cursor = transaction_executor.execute_statement(statement,
 convert_object_to_ion(documents))
    list_of_document_ids = get_document_ids_from_dml_results(cursor)

    return list_of_document_ids


def update_and_insert_documents(transaction_executor):
    """
    Handle the insertion of documents and updating PersonIds all in a single
 transaction.
```

```
    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
statements within a transaction.
    """
    list_ids = insert_documents(transaction_executor,
Constants.PERSON_TABLE_NAME, SampleData.PERSON)

    logger.info("Updating PersonIds for 'DriversLicense' and PrimaryOwner for
'VehicleRegistration'...")
    new_licenses, new_registrations = update_person_id(list_ids)

    insert_documents(transaction_executor, Constants.VEHICLE_TABLE_NAME,
SampleData.VEHICLE)
    insert_documents(transaction_executor,
Constants.VEHICLE_REGISTRATION_TABLE_NAME, new_registrations)
    insert_documents(transaction_executor, Constants.DRIVERS_LICENSE_TABLE_NAME,
new_licenses)


if __name__ == '__main__':
    """
    Insert documents into a table in a QLDB ledger.
    """
    try:
        with create_qldb_session() as session:
            # An INSERT statement creates the initial revision of a document
with a version number of zero.
            # QLDB also assigns a unique document identifier in GUID format as
part of the metadata.
            session.execute_lambda(lambda executor:
update_and_insert_documents(executor),
                                   lambda retry_attempt: logger.info('Retrying
due to OCC conflict...'))
            logger.info('Documents inserted successfully!')
    except Exception:
        logger.exception('Error inserting or updating documents.')
```

> **Note**
>
> - 이 프로그램은 파라미터화된 값을 사용하여 `execute_statement` 함수를 호출하는 방법을 보여줍니다. 실행하려는 PartiQL 문 외에도 데이터 파라미터를 전달할 수 있습니다. 명령문 문자열에서 물음표(?)를 변수 자리 표시자로 사용하세요.
> - `INSERT` 문이 성공하면 삽입된 각 문서의 `id`이 반환됩니다.

3. 프로그램을 실행하려면 다음 명령을 입력합니다.

```
python insert_document.py
```

다음으로 `SELECT` 문을 사용하여 `vehicle-registration` 원장의 테이블에서 데이터를 읽을 수 있습니다. 4단계: 원장에서 테이블 쿼리로 이동합니다.

4단계: 원장에서 테이블 쿼리

Amazon QLDB 원장에 테이블을 생성하고 데이터를 로드한 후 쿼리를 실행하여 방금 삽입한 차량 등록 데이터를 검토할 수 있습니다. QLDB는 PartiQL을 쿼리 언어로 사용하고 Amazon Ion을 문서 지향 데이터 모델로 사용합니다.

PartiQL은 Ion과 함께 작동하도록 확장된 오픈 소스 SQL 호환 쿼리 언어입니다. PartiQL을 사용하면 익숙한 SQL 연산자를 사용하여 데이터를 삽입, 쿼리 및 관리할 수 있습니다. Amazon Ion은 JSON의 상위 집합입니다. Ion은 정형, 반정형 및 중첩 데이터를 저장하고 처리할 수 있는 유연성을 제공하는 오픈 소스 문서 기반 데이터 형식입니다.

이 단계에서는 `SELECT` 명령문을 사용하여 `vehicle-registration` 원장의 테이블에서 데이터를 읽습니다.

> **Warning**
>
> 인덱싱된 조회 없이 QLDB에서 쿼리를 실행하면 전체 테이블 스캔이 호출됩니다. PartiQL은 SQL과 호환되므로 이러한 쿼리를 지원합니다. 하지만 QLDB의 프로덕션 사용 사례에 대해서는 테이블 스캔을 실행하지 마세요. 테이블 스캔은 동시성 충돌 및 트랜잭션 시간 초과를 포함하여 대규모 테이블에서 성능 문제를 일으킬 수 있습니다.

인덱싱된 필드 또는 문서 ID(예: `WHERE indexedField = 123` 또는 `WHERE indexedField IN (456, 789)`)에서 동등 연산자를 사용하여 `WHERE` 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 자세한 내용은 쿼리 성능 최적화을 참조하십시오.

테이블을 쿼리하려면

1. 다음 프로그램(`find_vehicles.py`)을 사용하여 원장에 있는 사람의 이름으로 등록된 모든 차량을 쿼리하세요.

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.model.sample_data import get_document_ids, print_result,
 SampleData
from pyqldbsamples.constants import Constants
```

```
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)


def find_vehicles_for_owner(driver, gov_id):
    """
    Find vehicles registered under a driver using their government ID.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type gov_id: str
    :param gov_id: The owner's government ID.
    """
    document_ids = driver.execute_lambda(lambda executor:
 get_document_ids(executor, Constants.PERSON_TABLE_NAME,

 'GovId', gov_id))

    query = "SELECT Vehicle FROM Vehicle INNER JOIN VehicleRegistration AS r " \
            "ON Vehicle.VIN = r.VIN WHERE r.Owners.PrimaryOwner.PersonId = ?"

    for ids in document_ids:
        cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(query, ids))
        logger.info('List of Vehicles for owner with GovId:
 {}...'.format(gov_id))
        print_result(cursor)


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Find all vehicles registered under a person.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            # Find all vehicles registered under a person.
            gov_id = SampleData.PERSON[0]['GovId']
            find_vehicles_for_owner(driver, gov_id)
    except Exception as e:
        logger.exception('Error getting vehicles for owner.')
        raise e
```

```
if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.model.sample_data import get_document_ids, print_result,
 SampleData
from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)
```

```
def find_vehicles_for_owner(transaction_executor, gov_id):
    """
    Find vehicles registered under a driver using their government ID.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type gov_id: str
    :param gov_id: The owner's government ID.
    """
    document_ids = get_document_ids(transaction_executor,
 Constants.PERSON_TABLE_NAME, 'GovId', gov_id)

    query = "SELECT Vehicle FROM Vehicle INNER JOIN VehicleRegistration AS r " \
            "ON Vehicle.VIN = r.VIN WHERE r.Owners.PrimaryOwner.PersonId = ?"

    for ids in document_ids:
        cursor = transaction_executor.execute_statement(query, ids)
        logger.info('List of Vehicles for owner with GovId:
 {}...'.format(gov_id))
        print_result(cursor)


if __name__ == '__main__':
    """
    Find all vehicles registered under a person.
    """
    try:
        with create_qldb_session() as session:
            # Find all vehicles registered under a person.
            gov_id = SampleData.PERSON[0]['GovId']
            session.execute_lambda(lambda executor:
 find_vehicles_for_owner(executor, gov_id),
                                   lambda retry_attempt: logger.info('Retrying
 due to OCC conflict...'))
    except Exception:
        logger.exception('Error getting vehicles for owner.')
```

> **Note**
>
> 먼저 이 프로그램은 이 문서에 대한 `Person` 테이블을 `GovId LEWISR261LL`로 쿼리하여 `id` 메타데이터 필드를 가져옵니다.
> 그런 다음 이 문서 `id`를 외래 키로 사용하여 `VehicleRegistration` 테이블을 `PrimaryOwner.PersonId`로 쿼리합니다. 또한 `VIN` 필드의 `Vehicle` 테이블과 `VehicleRegistration`를 조인합니다.

2. 프로그램을 실행하려면 다음 명령을 입력합니다.

```
python find_vehicles.py
```

`vehicle-registration` 원장의 테이블에 있는 문서를 수정하는 방법에 대한 자세한 내용은 5단계: 원장의 문서 수정을 참조하십시오.

5단계: 원장의 문서 수정

이제 작업할 데이터가 있으므로 Amazon QLDB에서 `vehicle-registration` 원장에 있는 문서를 변경할 수 있습니다. 이 단계에서 다음 코드 예제는 DML(데이터 조작 언어) 문을 실행하는 방법을 보여 줍니다. 이러한 명령문은 한 차량의 주 소유자를 업데이트하고 다른 차량에 보조 소유자를 추가합니다.

문서를 수정하려면

1. 다음 프로그램(`transfer_vehicle_ownership.py`)을 사용하여 차량의 주 소유자를 원장에 VIN `1N4AL11D75C109151`으로 업데이트하세요.

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
```

```
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.add_secondary_owner import get_document_ids, print_result,
 SampleData
from pyqldbsamples.constants import Constants
from pyqldbsamples.model.sample_data import convert_object_to_ion
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)


def find_person_from_document_id(transaction_executor, document_id):
    """
    Query a driver's information using the given ID.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type document_id: :py:class:`amazon.ion.simple_types.IonPyText`
    :param document_id: The document ID required to query for the person.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyDict`
    :return: The resulting document from the query.
    """
    query = 'SELECT p.* FROM Person AS p BY pid WHERE pid = ?'
    cursor = transaction_executor.execute_statement(query, document_id)
```

```
    return next(cursor)


def find_primary_owner_for_vehicle(driver, vin):
    """
    Find the primary owner of a vehicle given its VIN.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: The VIN to find primary owner for.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyDict`
    :return: The resulting document from the query.
    """
    logger.info('Finding primary owner for vehicle with VIN: {}.'.format(vin))
    query = "SELECT Owners.PrimaryOwner.PersonId FROM VehicleRegistration AS v
 WHERE v.VIN = ?"
    cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(query, convert_object_to_ion(vin)))
    try:
        return driver.execute_lambda(lambda executor:
 find_person_from_document_id(executor,

   next(cursor).get('PersonId')))
    except StopIteration:
        logger.error('No primary owner registered for this vehicle.')
        return None


def update_vehicle_registration(driver, vin, document_id):
    """
    Update the primary owner for a vehicle using the given VIN.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: The VIN for the vehicle to operate on.

    :type document_id: :py:class:`amazon.ion.simple_types.IonPyText`
    :param document_id: New PersonId for the primary owner.
```

```
    :raises RuntimeError: If no vehicle registration was found using the given
 document ID and VIN.
    """
    logger.info('Updating the primary owner for vehicle with Vin:
 {}...'.format(vin))
    statement = "UPDATE VehicleRegistration AS r SET
 r.Owners.PrimaryOwner.PersonId = ? WHERE r.VIN = ?"
    cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(statement, document_id,

 convert_object_to_ion(vin)))
    try:
        print_result(cursor)
        logger.info('Successfully transferred vehicle with VIN: {} to new
 owner.'.format(vin))
    except StopIteration:
        raise RuntimeError('Unable to transfer vehicle, could not find
 registration.')


def validate_and_update_registration(driver, vin, current_owner, new_owner):
    """
    Validate the current owner of the given vehicle and transfer its ownership
 to a new owner.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: The VIN of the vehicle to transfer ownership of.

    :type current_owner: str
    :param current_owner: The GovId of the current owner of the vehicle.

    :type new_owner: str
    :param new_owner: The GovId of the new owner of the vehicle.

    :raises RuntimeError: If unable to verify primary owner.
    """
    primary_owner = find_primary_owner_for_vehicle(driver, vin)
    if primary_owner is None or primary_owner['GovId'] != current_owner:
        raise RuntimeError('Incorrect primary owner identified for vehicle,
 unable to transfer.')
```

```
    document_ids = driver.execute_lambda(lambda executor:
 get_document_ids(executor, Constants.PERSON_TABLE_NAME,

 'GovId', new_owner))
    update_vehicle_registration(driver, vin, document_ids[0])


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Find primary owner for a particular vehicle's VIN.
    Transfer to another primary owner for a particular vehicle's VIN.
    """
    vehicle_vin = SampleData.VEHICLE[0]['VIN']
    previous_owner = SampleData.PERSON[0]['GovId']
    new_owner = SampleData.PERSON[1]['GovId']

    try:
        with create_qldb_driver(ledger_name) as driver:
            validate_and_update_registration(driver, vehicle_vin,
 previous_owner, new_owner)
    except Exception as e:
        logger.exception('Error updating VehicleRegistration.')
        raise e


if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
```

```
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.add_secondary_owner import get_document_ids, print_result,
 SampleData
from pyqldbsamples.constants import Constants
from pyqldbsamples.model.sample_data import convert_object_to_ion
from pyqldbsamples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)


def find_person_from_document_id(transaction_executor, document_id):
    """
    Query a driver's information using the given ID.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type document_id: :py:class:`amazon.ion.simple_types.IonPyText`
    :param document_id: The document ID required to query for the person.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyDict`
    :return: The resulting document from the query.
    """
    query = 'SELECT p.* FROM Person AS p BY pid WHERE pid = ?'
    cursor = transaction_executor.execute_statement(query, document_id)
    return next(cursor)


def find_primary_owner_for_vehicle(transaction_executor, vin):
```

```
    """
    Find the primary owner of a vehicle given its VIN.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type vin: str
    :param vin: The VIN to find primary owner for.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyDict`
    :return: The resulting document from the query.
    """
    logger.info('Finding primary owner for vehicle with VIN: {}.'.format(vin))
    query = "SELECT Owners.PrimaryOwner.PersonId FROM VehicleRegistration AS v
 WHERE v.VIN = ?"
    cursor = transaction_executor.execute_statement(query,
 convert_object_to_ion(vin))
    try:
        return find_person_from_document_id(transaction_executor,
 next(cursor).get('PersonId'))
    except StopIteration:
        logger.error('No primary owner registered for this vehicle.')
        return None


def update_vehicle_registration(transaction_executor, vin, document_id):
    """
    Update the primary owner for a vehicle using the given VIN.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type vin: str
    :param vin: The VIN for the vehicle to operate on.

    :type document_id: :py:class:`amazon.ion.simple_types.IonPyText`
    :param document_id: New PersonId for the primary owner.

    :raises RuntimeError: If no vehicle registration was found using the given
 document ID and VIN.
    """
```

```
    logger.info('Updating the primary owner for vehicle with Vin:
 {}...'.format(vin))
    statement = "UPDATE VehicleRegistration AS r SET
 r.Owners.PrimaryOwner.PersonId = ? WHERE r.VIN = ?"
    cursor = transaction_executor.execute_statement(statement, document_id,
 convert_object_to_ion(vin))
    try:
        print_result(cursor)
        logger.info('Successfully transferred vehicle with VIN: {} to new
 owner.'.format(vin))
    except StopIteration:
        raise RuntimeError('Unable to transfer vehicle, could not find
 registration.')


def validate_and_update_registration(transaction_executor, vin, current_owner,
 new_owner):
    """
    Validate the current owner of the given vehicle and transfer its ownership
 to a new owner in a single transaction.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type vin: str
    :param vin: The VIN of the vehicle to transfer ownership of.

    :type current_owner: str
    :param current_owner: The GovId of the current owner of the vehicle.

    :type new_owner: str
    :param new_owner: The GovId of the new owner of the vehicle.

    :raises RuntimeError: If unable to verify primary owner.
    """
    primary_owner = find_primary_owner_for_vehicle(transaction_executor, vin)
    if primary_owner is None or primary_owner['GovId'] != current_owner:
        raise RuntimeError('Incorrect primary owner identified for vehicle,
 unable to transfer.')

    document_id = next(get_document_ids(transaction_executor,
 Constants.PERSON_TABLE_NAME, 'GovId', new_owner))
```

```
    update_vehicle_registration(transaction_executor, vin, document_id)


if __name__ == '__main__':
    """
    Find primary owner for a particular vehicle's VIN.
    Transfer to another primary owner for a particular vehicle's VIN.
    """
    vehicle_vin = SampleData.VEHICLE[0]['VIN']
    previous_owner = SampleData.PERSON[0]['GovId']
    new_owner = SampleData.PERSON[1]['GovId']

    try:
        with create_qldb_session() as session:
            session.execute_lambda(lambda executor:
 validate_and_update_registration(executor, vehicle_vin,

     previous_owner, new_owner),
                                   retry_indicator=lambda retry_attempt:
 logger.info('Retrying due to OCC conflict...'))
    except Exception:
        logger.exception('Error updating VehicleRegistration.')
```

2. 프로그램을 실행하려면 다음 명령을 입력합니다.

```
python transfer_vehicle_ownership.py
```

3. 다음 프로그램(add_secondary_owner.py)을 사용하여 차량의 보조 소유자를 원장에 VIN KM8SRDHF6EU074761으로 추가하세요.

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
```

```
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.model.sample_data import to_ion_struct, get_document_ids,
 print_result, SampleData, \
    convert_object_to_ion
from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)


def get_document_id_by_gov_id(driver, government_id):
    """
    Find a driver's person ID using the given government ID.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type government_id: str
    :param government_id: A driver's government ID.

    :rtype: list
    :return: A list of document IDs.
    """
    logger.info("Finding secondary owner's person ID using given government ID:
 {}.".format(government_id))
    return driver.execute_lambda(lambda executor: get_document_ids(executor,
 Constants.PERSON_TABLE_NAME, 'GovId',
```

```
 government_id))


def is_secondary_owner_for_vehicle(driver, vin, secondary_owner_id):
    """
    Check whether a secondary owner has already been registered for the given
 VIN.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: VIN of the vehicle to query.

    :type secondary_owner_id: str
    :param secondary_owner_id: The secondary owner's person ID.

    :rtype: bool
    :return: If the driver has already been registered.
    """
    logger.info('Finding secondary owners for vehicle with VIN:
 {}...'.format(vin))
    query = 'SELECT Owners.SecondaryOwners FROM VehicleRegistration AS v WHERE
 v.VIN = ?'
    rows = driver.execute_lambda(lambda executor:
 executor.execute_statement(query, convert_object_to_ion(vin)))

    for row in rows:
        secondary_owners = row.get('SecondaryOwners')
        person_ids = map(lambda owner: owner.get('PersonId').text,
 secondary_owners)
        if secondary_owner_id in person_ids:
            return True
    return False


def add_secondary_owner_for_vin(driver, vin, parameter):
    """
    Add a secondary owner into `VehicleRegistration` table for a particular VIN.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.
```

```
    :type vin: str
    :param vin: VIN of the vehicle to add a secondary owner for.

    :type parameter: :py:class:`amazon.ion.simple_types.IonPyValue`
    :param parameter: The Ion value or Python native type that is convertible to
 Ion for filling in parameters of the
                      statement.
    """
    logger.info('Inserting secondary owner for vehicle with VIN:
 {}...'.format(vin))
    statement = "FROM VehicleRegistration AS v WHERE v.VIN = ? INSERT INTO
 v.Owners.SecondaryOwners VALUE ?"

    cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(statement, convert_object_to_ion(vin),

 parameter))
    logger.info('VehicleRegistration Document IDs which had secondary owners
 added: ')
    print_result(cursor)


def register_secondary_owner(driver, vin, gov_id):
    """
    Register a secondary owner for a vehicle if they are not already registered.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: VIN of the vehicle to register a secondary owner for.

    :type gov_id: str
    :param gov_id: The government ID of the owner.
    """
    logger.info('Finding the secondary owners for vehicle with VIN:
 {}.'.format(vin))

    document_ids = get_document_id_by_gov_id(driver, gov_id)

    for document_id in document_ids:
        if is_secondary_owner_for_vehicle(driver, vin, document_id):
            logger.info('Person with ID {} has already been added as a secondary
 owner of this vehicle.'.format(gov_id))
```

```
        else:
            add_secondary_owner_for_vin(driver, vin, to_ion_struct('PersonId',
 document_id))


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Finds and adds secondary owners for a vehicle.
    """
    vin = SampleData.VEHICLE[1]['VIN']
    gov_id = SampleData.PERSON[0]['GovId']
    try:
        with create_qldb_driver(ledger_name) as driver:
            register_secondary_owner(driver, vin, gov_id)
            logger.info('Secondary owners successfully updated.')
    except Exception as e:
        logger.exception('Error adding secondary owner.')
        raise e


if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
```

```
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.model.sample_data import to_ion_struct, get_document_ids,
 print_result, SampleData, \
    convert_object_to_ion
from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)


def get_document_id_by_gov_id(transaction_executor, government_id):
    """
    Find a driver's person ID using the given government ID.
    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.
    :type government_id: str
    :param government_id: A driver's government ID.
    :rtype: list
    :return: A list of document IDs.
    """
    logger.info("Finding secondary owner's person ID using given government ID:
 {}.".format(government_id))
    return get_document_ids(transaction_executor, Constants.PERSON_TABLE_NAME,
 'GovId', government_id)


def is_secondary_owner_for_vehicle(transaction_executor, vin,
 secondary_owner_id):
    """
    Check whether a secondary owner has already been registered for the given
 VIN.
    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.
```

```
    :type vin: str
    :param vin: VIN of the vehicle to query.
    :type secondary_owner_id: str
    :param secondary_owner_id: The secondary owner's person ID.
    :rtype: bool
    :return: If the driver has already been registered.
    """
    logger.info('Finding secondary owners for vehicle with VIN:
{}...'.format(vin))
    query = 'SELECT Owners.SecondaryOwners FROM VehicleRegistration AS v WHERE
v.VIN = ?'
    rows = transaction_executor.execute_statement(query,
convert_object_to_ion(vin))

    for row in rows:
        secondary_owners = row.get('SecondaryOwners')
        person_ids = map(lambda owner: owner.get('PersonId').text,
secondary_owners)
        if secondary_owner_id in person_ids:
            return True
    return False


def add_secondary_owner_for_vin(transaction_executor, vin, parameter):
    """
    Add a secondary owner into `VehicleRegistration` table for a particular VIN.
    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
statements within a transaction.
    :type vin: str
    :param vin: VIN of the vehicle to add a secondary owner for.
    :type parameter: :py:class:`amazon.ion.simple_types.IonPyValue`
    :param parameter: The Ion value or Python native type that is convertible to
Ion for filling in parameters of the
                      statement.
    """
    logger.info('Inserting secondary owner for vehicle with VIN:
{}...'.format(vin))
    statement = "FROM VehicleRegistration AS v WHERE v.VIN = '{}' INSERT INTO
v.Owners.SecondaryOwners VALUE ?"\
        .format(vin)

    cursor = transaction_executor.execute_statement(statement, parameter)
```

```
    logger.info('VehicleRegistration Document IDs which had secondary owners
 added: ')
    print_result(cursor)


def register_secondary_owner(transaction_executor, vin, gov_id):
    """
    Register a secondary owner for a vehicle if they are not already registered.
    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.
    :type vin: str
    :param vin: VIN of the vehicle to register a secondary owner for.
    :type gov_id: str
    :param gov_id: The government ID of the owner.
    """
    logger.info('Finding the secondary owners for vehicle with VIN:
 {}.'.format(vin))
    document_ids = get_document_id_by_gov_id(transaction_executor, gov_id)

    for document_id in document_ids:
        if is_secondary_owner_for_vehicle(transaction_executor, vin,
 document_id):
            logger.info('Person with ID {} has already been added as a secondary
 owner of this vehicle.'.format(gov_id))
        else:
            add_secondary_owner_for_vin(transaction_executor, vin,
 to_ion_struct('PersonId', document_id))


if __name__ == '__main__':
    """
    Finds and adds secondary owners for a vehicle.
    """
    vin = SampleData.VEHICLE[1]['VIN']
    gov_id = SampleData.PERSON[0]['GovId']
    try:
        with create_qldb_session() as session:
            session.execute_lambda(lambda executor:
 register_secondary_owner(executor, vin, gov_id),
                                   lambda retry_attempt: logger.info('Retrying
 due to OCC conflict...'))
            logger.info('Secondary owners successfully updated.')
    except Exception:
```

```
        logger.exception('Error adding secondary owner.')
```

4. 프로그램을 실행하려면 다음 명령을 입력합니다.

```
python add_secondary_owner.py
```

vehicle-registration 원장의 이러한 변경 사항을 검토하려면 6단계: 문서의 개정 기록 보기을 참조하세요.

6단계: 문서의 개정 기록 보기

이전 단계에서 차량의 등록 데이터를 수정한 후 등록된 모든 소유자의 기록과 기타 업데이트된 필드를 쿼리할 수 있습니다. 이 단계에서는 vehicle-registration 원장의 VehicleRegistration 테이블에 있는 문서의 개정 기록을 쿼리합니다.

개정 기록을 보려면

1. 다음 프로그램(query_history.py)을 사용하여 VIN 1N4AL11D75C109151로 VehicleRegistration 문서의 개정 기록을 쿼리합니다.

   3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
```

```
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from datetime import datetime, timedelta
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.model.sample_data import print_result, get_document_ids,
 SampleData
from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)


def format_date_time(date_time):
    """
    Format the given date time to a string.

    :type date_time: :py:class:`datetime.datetime`
    :param date_time: The date time to format.

    :rtype: str
    :return: The formatted date time.
    """
    return date_time.strftime('`%Y-%m-%dT%H:%M:%S.%fZ`')


def previous_primary_owners(driver, vin):
    """
    Find previous primary owners for the given VIN in a single transaction.
    In this example, query the `VehicleRegistration` history table to find all
 previous primary owners for a VIN.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: VIN to find previous primary owners for.
    """
```

```
    person_ids = driver.execute_lambda(lambda executor:
 get_document_ids(executor,

 Constants.VEHICLE_REGISTRATION_TABLE_NAME,
                                                                    'VIN',
 vin))

    todays_date = datetime.utcnow() - timedelta(seconds=1)
    three_months_ago = todays_date - timedelta(days=90)
    query = 'SELECT data.Owners.PrimaryOwner, metadata.version FROM history({},
 {}, {}) AS h WHERE h.metadata.id = ?'.\
        format(Constants.VEHICLE_REGISTRATION_TABLE_NAME,
 format_date_time(three_months_ago),
               format_date_time(todays_date))

    for ids in person_ids:
        logger.info("Querying the 'VehicleRegistration' table's history using
 VIN: {}.".format(vin))
        cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(query, ids))
        if not (print_result(cursor)) > 0:
            logger.info('No modification history found within the given time
 frame for document ID: {}'.format(ids))


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Query a table's history for a particular set of documents.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            vin = SampleData.VEHICLE_REGISTRATION[0]['VIN']
            previous_primary_owners(driver, vin)
            logger.info('Successfully queried history.')
    except Exception as e:
        logger.exception('Unable to query history to find previous owners.')
        raise e


if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from datetime import datetime, timedelta
from logging import basicConfig, getLogger, INFO

from pyqldbsamples.model.sample_data import print_result, get_document_ids,
 SampleData
from pyqldbsamples.constants import Constants
from pyqldbsamples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)


def format_date_time(date_time):
    """
    Format the given date time to a string.
```

```
    :type date_time: :py:class:`datetime.datetime`
    :param date_time: The date time to format.

    :rtype: str
    :return: The formatted date time.
    """
    return date_time.strftime('`%Y-%m-%dT%H:%M:%S.%fZ`')


def previous_primary_owners(transaction_executor, vin):
    """
    Find previous primary owners for the given VIN in a single transaction.
    In this example, query the `VehicleRegistration` history table to find all
 previous primary owners for a VIN.

    :type transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :type vin: str
    :param vin: VIN to find previous primary owners for.
    """
    person_ids = get_document_ids(transaction_executor,
 Constants.VEHICLE_REGISTRATION_TABLE_NAME, 'VIN', vin)

    todays_date = datetime.utcnow() - timedelta(seconds=1)
    three_months_ago = todays_date - timedelta(days=90)
    query = 'SELECT data.Owners.PrimaryOwner, metadata.version FROM history({},
 {}, {}) AS h WHERE h.metadata.id = ?'.\
        format(Constants.VEHICLE_REGISTRATION_TABLE_NAME,
 format_date_time(three_months_ago),
               format_date_time(todays_date))

    for ids in person_ids:
        logger.info("Querying the 'VehicleRegistration' table's history using
 VIN: {}.".format(vin))
        cursor = transaction_executor.execute_statement(query, ids)
        if not (print_result(cursor)) > 0:
            logger.info('No modification history found within the given time
 frame for document ID: {}'.format(ids))


if __name__ == '__main__':
    """
```

```
    Query a table's history for a particular set of documents.
    """
    try:
        with create_qldb_session() as session:
            vin = SampleData.VEHICLE_REGISTRATION[0]['VIN']
            session.execute_lambda(lambda lambda_executor:
previous_primary_owners(lambda_executor, vin),
                                   lambda retry_attempt: logger.info('Retrying
due to OCC conflict...'))
            logger.info('Successfully queried history.')
    except Exception:
        logger.exception('Unable to query history to find previous owners.')
```

> **Note**
>
> - 다음 구문에 내장된 기록 함수를 쿼리하여 문서의 개정 기록을 볼 수 있습니다
>
>   ```
>   SELECT * FROM history( table_name [, `start-time` [, `end-time` ] ] ) AS h
>   [ WHERE h.metadata.id = 'id' ]
>   ```
>
> - 시작 시간과 종료 시간은 모두 선택 사항입니다. 이 값은 백틱(`` `...` ``)으로 표시할 수 있는 Amazon Ion 리터럴 값입니다. 자세한 내용은 Amazon QLDB에서 PartiQL을 사용한 Ion 쿼리 섹션을 참조하세요.
> - 가장 좋은 방법은 날짜 범위(시작 시간 및 종료 시간)와 문서 ID(`metadata.id`)를 모두 사용하여 기록 쿼리를 한정하는 것입니다. QLDB는 트랜잭션에서 SELECT 쿼리를 처리하며, 이 쿼리에는 트랜잭션 시간 초과 제한이 적용됩니다.
>
>   QLDB 기록은 문서 ID로 인덱싱되며, 이번에는 추가 기록 인덱스를 만들 수 없습니다. 시작 시간과 종료 시간을 포함하는 기록 쿼리는 날짜 범위 한정이라는 이점을 갖습니다.

2. 프로그램을 실행하려면 다음 명령을 입력합니다.

```
python query_history.py
```

`vehicle-registration` 원장의 문서 개정을 암호화 방식으로 검증하려면 7단계: 원장에 있는 문서 검증으로 진행하세요.

7단계: 원장에 있는 문서 검증

Amazon QLDB를 사용하면 SHA-256 암호화 해싱을 사용하여 원장 저널에 있는 문서의 무결성을 효율적으로 검증할 수 있습니다. QLDB에서 검증 및 암호화 해싱이 작동하는 방식에 대한 자세한 내용은 Amazon QLDB에서의 데이터 확인을 참조하십시오.

이 단계에서는 `vehicle-registration` 원장의 `VehicleRegistration` 테이블에 있는 문서 개정을 검증합니다. 먼저 다이제스트를 요청합니다. 다이제스트는 출력 파일로 반환되며 원장의 전체 변경 내역에 대한 서명 역할을 합니다. 그런 다음 해당 다이제스트와 관련된 개정 증거를 요청합니다. 이 증거를 사용하면 모든 유효성 검사를 통과한 경우 개정 내용의 무결성을 확인할 수 있습니다.

문서 개정을 검증하려면

1. 검증에 필요한 QLDB 객체와 QLDB 응답 유형을 문자열로 변환하는 도우미 함수가 있는 유틸리티 모듈을 나타내는 다음 `.py` 파일을 검토하십시오.

   1. `block_address.py`

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

```
def block_address_to_dictionary(ion_dict):
    """
    Convert a block address from IonPyDict into a dictionary.
    Shape of the dictionary must be: {'IonText': "{strandId: <"strandId">,
 sequenceNo: <sequenceNo>}"}

    :type ion_dict: :py:class:`amazon.ion.simple_types.IonPyDict`/str
    :param ion_dict: The block address value to convert.

    :rtype: dict
    :return: The converted dict.
    """
    block_address = {'IonText': {}}
    if not isinstance(ion_dict, str):
        py_dict = '{{strandId: "{}", sequenceNo:
{}}}'.format(ion_dict['strandId'], ion_dict['sequenceNo'])
        ion_dict = py_dict
    block_address['IonText'] = ion_dict
    return block_address
```

2. verifier.py

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

```
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from array import array
from base64 import b64encode
from functools import reduce
from hashlib import sha256
from random import randrange

from amazon.ion.simpleion import loads

HASH_LENGTH = 32
UPPER_BOUND = 8


def parse_proof(value_holder):
    """
    Parse the Proof object returned by QLDB into an iterator.

    The Proof object returned by QLDB is a dictionary like the following:
    {'IonText': '[{{<hash>}},{{<hash>}}]'}

    :type value_holder: dict
    :param value_holder: A structure containing an Ion string value.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyList`
    :return: A list of hash values.
    """
    value_holder = value_holder.get('IonText')
    proof_list = loads(value_holder)
    return proof_list


def parse_block(value_holder):
    """
    Parse the Block object returned by QLDB and retrieve block hash.

    :type value_holder: dict
    :param value_holder: A structure containing an Ion string value.

    :rtype: :py:class:`amazon.ion.simple_types.IonPyBytes`
    :return: The block hash.
    """
    value_holder = value_holder.get('IonText')
```

```
    block = loads(value_holder)
    block_hash = block.get('blockHash')
    return block_hash


def flip_random_bit(original):
    """
    Flip a single random bit in the given hash value.
    This method is used to demonstrate QLDB's verification features.

    :type original: bytes
    :param original: The hash value to alter.

    :rtype: bytes
    :return: The altered hash with a single random bit changed.
    """
    assert len(original) != 0, 'Invalid bytes.'

    altered_position = randrange(len(original))
    bit_shift = randrange(UPPER_BOUND)
    altered_hash = bytearray(original).copy()

    altered_hash[altered_position] = altered_hash[altered_position] ^ (1 <<
 bit_shift)
    return bytes(altered_hash)


def compare_hash_values(hash1, hash2):
    """
    Compare two hash values by converting them into byte arrays, assuming they
 are little endian.

    :type hash1: bytes
    :param hash1: The hash value to compare.

    :type hash2: bytes
    :param hash2: The hash value to compare.

    :rtype: int
    :return: Zero if the hash values are equal, otherwise return the difference
 of the first pair of non-matching bytes.
    """
    assert len(hash1) == HASH_LENGTH
    assert len(hash2) == HASH_LENGTH
```

```
    hash_array1 = array('b', hash1)
    hash_array2 = array('b', hash2)

    for i in range(len(hash_array1) - 1, -1, -1):
        difference = hash_array1[i] - hash_array2[i]
        if difference != 0:
            return difference
    return 0


def join_hash_pairwise(hash1, hash2):
    """
    Take two hash values, sort them, concatenate them, and generate a new hash
 value from the concatenated values.

    :type hash1: bytes
    :param hash1: Hash value to concatenate.

    :type hash2: bytes
    :param hash2: Hash value to concatenate.

    :rtype: bytes
    :return: The new hash value generated from concatenated hash values.
    """
    if len(hash1) == 0:
        return hash2
    if len(hash2) == 0:
        return hash1

    concatenated = hash1 + hash2 if compare_hash_values(hash1, hash2) < 0 else
 hash2 + hash1
    new_hash_lib = sha256()
    new_hash_lib.update(concatenated)
    new_digest = new_hash_lib.digest()
    return new_digest


def calculate_root_hash_from_internal_hashes(internal_hashes, leaf_hash):
    """
    Combine the internal hashes and the leaf hash until only one root hash
 remains.

    :type internal_hashes: map
```

```
    :param internal_hashes: An iterable over a list of hash values.

    :type leaf_hash: bytes
    :param leaf_hash: The revision hash to pair with the first hash in the Proof
 hashes list.

    :rtype: bytes
    :return: The root hash constructed by combining internal hashes.
    """
    root_hash = reduce(join_hash_pairwise, internal_hashes, leaf_hash)
    return root_hash


def build_candidate_digest(proof, leaf_hash):
    """
    Build the candidate digest representing the entire ledger from the Proof
 hashes.

    :type proof: dict
    :param proof: The Proof object.

    :type leaf_hash: bytes
    :param leaf_hash: The revision hash to pair with the first hash in the Proof
 hashes list.

    :rtype: bytes
    :return: The calculated root hash.
    """
    parsed_proof = parse_proof(proof)
    root_hash = calculate_root_hash_from_internal_hashes(parsed_proof, leaf_hash)
    return root_hash


def verify_document(document_hash, digest, proof):
    """
    Verify document revision against the provided digest.

    :type document_hash: bytes
    :param document_hash: The SHA-256 value representing the document revision to
 be verified.

    :type digest: bytes
    :param digest: The SHA-256 hash value representing the ledger digest.
```

```
    :type proof: dict
    :param proof: The Proof object retrieved
 from :func:`pyqldbsamples.get_revision.get_revision`.

    :rtype: bool
    :return: If the document revision verify against the ledger digest.
    """
    candidate_digest = build_candidate_digest(proof, document_hash)
    return digest == candidate_digest


def to_base_64(input):
    """
    Encode input in base64.

    :type input: bytes
    :param input: Input to be encoded.

    :rtype: string
    :return: Return input that has been encoded in base64.
    """
    encoded_value = b64encode(input)
    return str(encoded_value, 'UTF-8')
```

3. qldb_string_utils.py

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
```

```
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
from amazon.ion.simpleion import dumps, loads


def value_holder_to_string(value_holder):
    """
    Returns the string representation of a given `value_holder`.

    :type value_holder: dict
    :param value_holder: The `value_holder` to convert to string.

    :rtype: str
    :return: The string representation of the supplied `value_holder`.
    """
    ret_val = dumps(loads(value_holder), binary=False, indent='  ',
 omit_version_marker=True)
    val = '{{ IonText: {}}}'.format(ret_val)
    return val


def block_response_to_string(block_response):
    """
    Returns the string representation of a given `block_response`.

    :type block_response: dict
    :param block_response: The `block_response` to convert to string.

    :rtype: str
    :return: The string representation of the supplied `block_response`.
    """
    string = ''
    if block_response.get('Block', {}).get('IonText') is not None:
        string += 'Block: ' + value_holder_to_string(block_response['Block']
['IonText']) + ', '

    if block_response.get('Proof', {}).get('IonText') is not None:
        string += 'Proof: ' + value_holder_to_string(block_response['Proof']
['IonText'])

    return '{' + string + '}'
```

```
def digest_response_to_string(digest_response):
    """
    Returns the string representation of a given `digest_response`.

    :type digest_response: dict
    :param digest_response: The `digest_response` to convert to string.

    :rtype: str
    :return: The string representation of the supplied `digest_response`.
    """
    string = ''
    if digest_response.get('Digest') is not None:
        string += 'Digest: ' + str(digest_response['Digest']) + ', '

    if digest_response.get('DigestTipAddress', {}).get('IonText') is not None:
        string += 'DigestTipAddress: ' +
 value_holder_to_string(digest_response['DigestTipAddress']['IonText'])

    return '{' + string + '}'
```

2. 두 .py 프로그램(get_digest.py 및 get_revision.py)을 사용하여 다음 단계를 수행하세요.

   - vehicle-registration 원장에 새 다이제스트를 요청하세요.
   - VehicleRegistration 테이블에서 VIN 1N4AL11D75C109151이 포함된 문서의 각 개정에 대한 증거를 요청합니다.
   - 반환된 다이제스트와 증거를 사용하여 다이제스트를 다시 계산하여 개정 내용을 검증합니다.

   get_digest.py 프로그램에는 다음 코드가 포함되어 있습니다.

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
```

```
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from boto3 import client

from pyqldbsamples.constants import Constants
from pyqldbsamples.qldb.qldb_string_utils import digest_response_to_string

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')


def get_digest_result(name):
    """
    Get the digest of a ledger's journal.

    :type name: str
    :param name: Name of the ledger to operate on.

    :rtype: dict
    :return: The digest in a 256-bit hash value and a block address.
    """
    logger.info("Let's get the current digest of the ledger named {}".format(name))
    result = qldb_client.get_digest(Name=name)
    logger.info('Success. LedgerDigest:
 {}.'.format(digest_response_to_string(result)))
    return result


def main(ledger_name=Constants.LEDGER_NAME):
    """
    This is an example for retrieving the digest of a particular ledger.
```

```
    """
    try:
        get_digest_result(ledger_name)
    except Exception as e:
        logger.exception('Unable to get a ledger digest!')
        raise e


if __name__ == '__main__':
    main()
```

> **Note**
>
> `get_digest_result` 함수를 사용하여 원장에 있는 저널의 현재 팁을 포함하는 다이제스트를 요청합니다. 저널 팁은 QLDB가 요청을 수신한 시점을 기준으로 가장 최근에 커밋된 블록을 나타냅니다.

`get_revision.py` 프로그램에는 다음 코드가 포함되어 있습니다.

3.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
```

```
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from amazon.ion.simpleion import loads
from boto3 import client

from pyqldbsamples.constants import Constants
from pyqldbsamples.get_digest import get_digest_result
from pyqldbsamples.model.sample_data import SampleData, convert_object_to_ion
from pyqldbsamples.qldb.block_address import block_address_to_dictionary
from pyqldbsamples.verifier import verify_document, flip_random_bit, to_base_64
from pyqldbsamples.connect_to_ledger import create_qldb_driver
from pyqldbsamples.qldb.qldb_string_utils import value_holder_to_string

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')


def get_revision(ledger_name, document_id, block_address, digest_tip_address):
    """
    Get the revision data object for a specified document ID and block address.
    Also returns a proof of the specified revision for verification.

    :type ledger_name: str
    :param ledger_name: Name of the ledger containing the document to query.

    :type document_id: str
    :param document_id: Unique ID for the document to be verified, contained in
 the committed view of the document.

    :type block_address: dict
    :param block_address: The location of the block to request.

    :type digest_tip_address: dict
    :param digest_tip_address: The latest block location covered by the digest.

    :rtype: dict
    :return: The response of the request.
    """
```

```
    result = qldb_client.get_revision(Name=ledger_name,
 BlockAddress=block_address, DocumentId=document_id,
                                       DigestTipAddress=digest_tip_address)
    return result


def lookup_registration_for_vin(driver, vin):
    """
    Query revision history for a particular vehicle for verification.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type vin: str
    :param vin: VIN to query the revision history of a specific registration
 with.

    :rtype: :py:class:`pyqldb.cursor.buffered_cursor.BufferedCursor`
    :return: Cursor on the result set of the statement query.
    """
    logger.info("Querying the 'VehicleRegistration' table for VIN:
 {}...".format(vin))
    query = 'SELECT * FROM _ql_committed_VehicleRegistration WHERE data.VIN = ?'
    return driver.execute_lambda(lambda txn: txn.execute_statement(query,
 convert_object_to_ion(vin)))


def verify_registration(driver, ledger_name, vin):
    """
    Verify each version of the registration for the given VIN.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type ledger_name: str
    :param ledger_name: The ledger to get digest from.

    :type vin: str
    :param vin: VIN to query the revision history of a specific registration
 with.

    :raises AssertionError: When verification failed.
    """
```

```
    logger.info("Let's verify the registration with VIN = {}, in ledger =
{}.".format(vin, ledger_name))
    digest = get_digest_result(ledger_name)
    digest_bytes = digest.get('Digest')
    digest_tip_address = digest.get('DigestTipAddress')

    logger.info('Got a ledger digest: digest tip address = {}, digest =
{}.'.format(
        value_holder_to_string(digest_tip_address.get('IonText')),
to_base_64(digest_bytes)))

    logger.info('Querying the registration with VIN = {} to verify each version
of the registration...'.format(vin))
    cursor = lookup_registration_for_vin(driver, vin)
    logger.info('Getting a proof for the document.')

    for row in cursor:
        block_address = row.get('blockAddress')
        document_id = row.get('metadata').get('id')

        result = get_revision(ledger_name, document_id,
block_address_to_dictionary(block_address), digest_tip_address)
        revision = result.get('Revision').get('IonText')
        document_hash = loads(revision).get('hash')

        proof = result.get('Proof')
        logger.info('Got back a proof: {}.'.format(proof))

        verified = verify_document(document_hash, digest_bytes, proof)
        if not verified:
            raise AssertionError('Document revision is not verified.')
        else:
            logger.info('Success! The document is verified.')

        altered_document_hash = flip_random_bit(document_hash)
        logger.info("Flipping one bit in the document's hash and assert that the
document is NOT verified. "
                    "The altered document hash is:
{}.".format(to_base_64(altered_document_hash)))
        verified = verify_document(altered_document_hash, digest_bytes, proof)
        if verified:
            raise AssertionError('Expected altered document hash to not be
verified against digest.')
        else:
```

```
            logger.info('Success! As expected flipping a bit in the document
 hash causes verification to fail.')

        logger.info('Finished verifying the registration with VIN = {} in ledger
 = {}.'.format(vin, ledger_name))


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Verify the integrity of a document revision in a QLDB ledger.
    """
    registration = SampleData.VEHICLE_REGISTRATION[0]
    vin = registration['VIN']
    try:
        with create_qldb_driver(ledger_name) as driver:
            verify_registration(driver, ledger_name, vin)
    except Exception as e:
        logger.exception('Unable to verify revision.')
        raise e


if __name__ == '__main__':
    main()
```

2.x

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
 of this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software,
 and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
```

```
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
 ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from amazon.ion.simpleion import loads
from boto3 import client

from pyqldbsamples.constants import Constants
from pyqldbsamples.get_digest import get_digest_result
from pyqldbsamples.model.sample_data import SampleData, convert_object_to_ion
from pyqldbsamples.qldb.block_address import block_address_to_dictionary
from pyqldbsamples.verifier import verify_document, flip_random_bit, to_base_64
from pyqldbsamples.connect_to_ledger import create_qldb_session
from pyqldbsamples.qldb.qldb_string_utils import value_holder_to_string

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')


def get_revision(ledger_name, document_id, block_address, digest_tip_address):
    """
    Get the revision data object for a specified document ID and block address.
    Also returns a proof of the specified revision for verification.

    :type ledger_name: str
    :param ledger_name: Name of the ledger containing the document to query.

    :type document_id: str
    :param document_id: Unique ID for the document to be verified, contained in
 the committed view of the document.

    :type block_address: dict
    :param block_address: The location of the block to request.

    :type digest_tip_address: dict
    :param digest_tip_address: The latest block location covered by the digest.
```

```
    :rtype: dict
    :return: The response of the request.
    """
    result = qldb_client.get_revision(Name=ledger_name,
 BlockAddress=block_address, DocumentId=document_id,
                                      DigestTipAddress=digest_tip_address)
    return result


def lookup_registration_for_vin(qldb_session, vin):
    """
    Query revision history for a particular vehicle for verification.

    :type qldb_session: :py:class:`pyqldb.session.qldb_session.QldbSession`
    :param qldb_session: An instance of the QldbSession class.

    :type vin: str
    :param vin: VIN to query the revision history of a specific registration
 with.

    :rtype: :py:class:`pyqldb.cursor.buffered_cursor.BufferedCursor`
    :return: Cursor on the result set of the statement query.
    """
    logger.info("Querying the 'VehicleRegistration' table for VIN:
 {}...".format(vin))
    query = 'SELECT * FROM _ql_committed_VehicleRegistration WHERE data.VIN = ?'
    parameters = [convert_object_to_ion(vin)]
    cursor = qldb_session.execute_statement(query, parameters)
    return cursor


def verify_registration(qldb_session, ledger_name, vin):
    """
    Verify each version of the registration for the given VIN.

    :type qldb_session: :py:class:`pyqldb.session.qldb_session.QldbSession`
    :param qldb_session: An instance of the QldbSession class.

    :type ledger_name: str
    :param ledger_name: The ledger to get digest from.

    :type vin: str
```

```
    :param vin: VIN to query the revision history of a specific registration
with.

    :raises AssertionError: When verification failed.
    """
    logger.info("Let's verify the registration with VIN = {}, in ledger =
{}.".format(vin, ledger_name))
    digest = get_digest_result(ledger_name)
    digest_bytes = digest.get('Digest')
    digest_tip_address = digest.get('DigestTipAddress')

    logger.info('Got a ledger digest: digest tip address = {}, digest =
{}.'.format(
        value_holder_to_string(digest_tip_address.get('IonText')),
to_base_64(digest_bytes)))

    logger.info('Querying the registration with VIN = {} to verify each version
of the registration...'.format(vin))
    cursor = lookup_registration_for_vin(qldb_session, vin)
    logger.info('Getting a proof for the document.')

    for row in cursor:
        block_address = row.get('blockAddress')
        document_id = row.get('metadata').get('id')

        result = get_revision(ledger_name, document_id,
block_address_to_dictionary(block_address), digest_tip_address)
        revision = result.get('Revision').get('IonText')
        document_hash = loads(revision).get('hash')

        proof = result.get('Proof')
        logger.info('Got back a proof: {}.'.format(proof))

        verified = verify_document(document_hash, digest_bytes, proof)
        if not verified:
            raise AssertionError('Document revision is not verified.')
        else:
            logger.info('Success! The document is verified.')

        altered_document_hash = flip_random_bit(document_hash)
        logger.info("Flipping one bit in the document's hash and assert that the
document is NOT verified. "
                    "The altered document hash is:
{}.".format(to_base_64(altered_document_hash)))
```

```
        verified = verify_document(altered_document_hash, digest_bytes, proof)
        if verified:
            raise AssertionError('Expected altered document hash to not be
 verified against digest.')
        else:
            logger.info('Success! As expected flipping a bit in the document
 hash causes verification to fail.')

        logger.info('Finished verifying the registration with VIN = {} in ledger
 = {}.'.format(vin, ledger_name))


if __name__ == '__main__':
    """
    Verify the integrity of a document revision in a QLDB ledger.
    """
    registration = SampleData.VEHICLE_REGISTRATION[0]
    vin = registration['VIN']
    try:
        with create_qldb_session() as session:
            verify_registration(session, Constants.LEDGER_NAME, vin)
    except Exception:
        logger.exception('Unable to verify revision.')
```

> **Note**
>
> `get_revision` 함수가 지정된 문서 개정에 대한 증거를 반환하면 이 프로그램은 클라이언트 측 API를 사용하여 해당 개정본을 확인합니다.

3. 프로그램을 실행하려면 다음 명령을 입력합니다.

```
python get_revision.py
```

`vehicle-registration` 원장을 더 이상 사용할 필요가 없는 경우 8단계(선택 사항): 리소스 정리로 진행하십시오.

8단계(선택 사항): 리소스 정리

`vehicle-registration` 원장을 계속 사용할 수 있습니다. 그러나 더 이상 필요하지 않은 경우 삭제해야 합니다.

원장을 삭제하려면

1. 다음 프로그램(delete_ledger.py)을 사용하여 vehicle-registration 원장과 모든 내용을 삭제하세요.

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO
from time import sleep

from boto3 import client

from pyqldbsamples.constants import Constants
from pyqldbsamples.describe_ledger import describe_ledger

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')

LEDGER_DELETION_POLL_PERIOD_SEC = 20
```

```
def delete_ledger(ledger_name):
    """
    Send a request to QLDB to delete the specified ledger.

    :type ledger_name: str
    :param ledger_name: Name for the ledger to be deleted.

    :rtype: dict
    :return: Result from the request.
    """
    logger.info('Attempting to delete the ledger with name:
 {}...'.format(ledger_name))
    result = qldb_client.delete_ledger(Name=ledger_name)
    logger.info('Success.')
    return result


def wait_for_deleted(ledger_name):
    """
    Wait for the ledger to be deleted.

    :type ledger_name: str
    :param ledger_name: The ledger to check on.
    """
    logger.info('Waiting for the ledger to be deleted...')
    while True:
        try:
            describe_ledger(ledger_name)
            logger.info('The ledger is still being deleted. Please wait...')
            sleep(LEDGER_DELETION_POLL_PERIOD_SEC)
        except qldb_client.exceptions.ResourceNotFoundException:
            logger.info('Success. The ledger is deleted.')
            break


def set_deletion_protection(ledger_name, deletion_protection):
    """
    Update an existing ledger's deletion protection.

    :type ledger_name: str
    :param ledger_name: Name of the ledger to update.

    :type deletion_protection: bool
```

```
    :param deletion_protection: Enable or disable the deletion protection.

    :rtype: dict
    :return: Result from the request.
    """
    logger.info("Let's set deletion protection to {} for the ledger with name
 {}.".format(deletion_protection,

       ledger_name))
    result = qldb_client.update_ledger(Name=ledger_name,
 DeletionProtection=deletion_protection)
    logger.info('Success. Ledger updated: {}'.format(result))


def main(ledger_name=Constants.LEDGER_NAME):
    """
    Delete a ledger.
    """
    try:
        set_deletion_protection(ledger_name, False)
        delete_ledger(ledger_name)
        wait_for_deleted(ledger_name)
    except Exception as e:
        logger.exception('Unable to delete the ledger.')
        raise e


if __name__ == '__main__':
    main()
```

> **Note**
>
> 원장에 대해 삭제 방지가 활성화된 경우 QLDB API를 사용하여 원장을 삭제하기 전에 먼저 이를 비활성화해야 합니다.

또한 `delete_ledger.py` 파일은 다음 프로그램(`describe_ledger.py`)에 대한 종속성을 갖습니다.

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
```

```
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy,
 modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from logging import basicConfig, getLogger, INFO

from boto3 import client

from pyqldbsamples.constants import Constants

logger = getLogger(__name__)
basicConfig(level=INFO)
qldb_client = client('qldb')


def describe_ledger(ledger_name):
    """
    Describe a ledger.

    :type ledger_name: str
    :param ledger_name: Name of the ledger to describe.
    """
    logger.info('describe ledger with name: {}.'.format(ledger_name))
    result = qldb_client.describe_ledger(Name=ledger_name)
    result.pop('ResponseMetadata')
    logger.info('Success. Ledger description: {}.'.format(result))
    return result
```

```
def main(ledger_name=Constants.LEDGER_NAME):
    """
    Describe a QLDB ledger.
    """
    try:
        describe_ledger(ledger_name)
    except Exception as e:
        logger.exception('Unable to describe a ledger.')
        raise e


if __name__ == '__main__':
    main()
```

2. 프로그램을 실행하려면 다음 명령을 입력합니다.

```
python delete_ledger.py
```

Amazon QLDB에서 Amazon Ion 데이터 타입을 사용한 작업

Amazon QLDB는 데이터를 Amazon Ion 형식으로 저장합니다. QLDB에서 데이터를 사용하려면 지원되는 프로그래밍 언어의 종속 항목으로 Ion 라이브러리를 사용해야 합니다.

이 섹션에서는 데이터를 네이티브 타입에서 해당 타입의 Ion 형식으로 변환하는 방법과 그 반대의 방법을 알아봅니다. 이 참조 가이드는 QLDB 드라이버를 사용하여 QLDB 원장의 Ion 데이터를 처리하는 코드 예를 보여줍니다. 여기에는 Java, .NET(C#)Go, Node.js(TypeScript) 및 Python의 코드 예가 포함됩니다.

주제

- 필수 조건
- Bool
- 정수
- Float
- 10진수
- 타임스탬프
- 문자열

- Blob
- 목록
- 구조체
- Null 값 및 동적 타입
- JSON으로 하향 변환

필수 조건

다음 코드 예는 `ExampleTable`라는 명칭의 활성 원장에 연결된 QLDB 드라이버 인스턴스가 있다고 가정합니다. 이 표에는 다음과 같은 8개 필드가 있는 기존 문서 하나가 포함되어 있습니다:

- ExampleBool
- ExampleInt
- ExampleFloat
- ExampleDecimal
- ExampleTimestamp
- ExampleString
- ExampleBlob
- ExampleList

> **Note**
>
> 이 참조에서는 각 필드에 저장된 형식이 해당 명칭과 일치한다고 가정합니다. 실제로 QLDB는 문서 필드에 스키마 또는 데이터 타입 정의를 적용하지 않습니다.

Bool

다음 코드 예에서는 Ion 부울 타입을 처리하는 방법을 보여줍니다.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();
```

```
boolean exampleBoolean = driver.execute((txn) -> {
    // Transforming a Java boolean to Ion
    boolean aBoolean = true;
    IonValue ionBool = ionSystem.newBool(aBoolean);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleBool = ?", ionBool);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleBool from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java boolean
        // Cast IonValue to IonBool first
        aBoolean = ((IonBool)ionValue).booleanValue();
    }

    // exampleBoolean is now the value fetched from QLDB
    return aBoolean;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();

// Transforming a C# bool to Ion.
bool nativeBool  = true;
IIonValue ionBool = valueFactory.NewBool(nativeBool);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleBool = ?", ionBool);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleBool from ExampleTable");
});

bool? retrievedBool = null;
```

```
// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# bool.
    retrievedBool = ionValue.BoolValue;
}
```

> **Note**
>
> 동기 코드로 변환하려면 `await` 및 `async` 키워드를 제거하고 `IAsyncResult` 유형을 `IResult`로 변경하세요.

Go

```
exampleBool, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aBool := true

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleBool = ?", aBool)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleBool FROM ExampleTable")
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  var decodedResult bool
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleBool is now the value fetched from QLDB
  return decodedResult, nil
```

```
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonBoolean(driver: QldbDriver): Promise<boolean> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleBool = ?", true);

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleBool FROM ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Transforming Ion to a TypeScript Boolean
            const boolValue: boolean = ionValue.booleanValue();
            return boolValue;
        })
    );
}
```

Python

```
def update_and_query_ion_bool(txn):
    # QLDB can take in a Python bool
    a_bool = True

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleBool = ?", a_bool)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleBool FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python bool is a child class of Python bool
        a_bool = ion_value

    # example_bool is now the value fetched from QLDB
    return a_bool
```

```
example_bool = driver.execute_lambda(lambda txn: update_and_query_ion_bool(txn))
```

정수

다음 코드 예에서는 Ion 정수형을 처리하는 방법을 보여줍니다.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

int exampleInt = driver.execute((txn) -> {
    // Transforming a Java int to Ion
    int aInt = 256;
    IonValue ionInt = ionSystem.newInt(aInt);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleInt = ?", ionInt);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleInt from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java int
        // Cast IonValue to IonInt first
        aInt = ((IonInt)ionValue).intValue();
    }

    // exampleInt is now the value fetched from QLDB
    return aInt;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();
```

```
// Transforming a C# int to Ion.
int nativeInt  = 256;
IIonValue ionInt = valueFactory.NewInt(nativeInt);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleInt = ?", ionInt);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleInt from ExampleTable");
});

int? retrievedInt = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# int.
    retrievedInt = ionValue.IntValue;
}
```

> **Note**
>
> 동기 코드로 변환하려면 `await` 및 `async` 키워드를 제거하고 `IAsyncResult` 유형을 `IResult`로 변경하세요.

Go

```
exampleInt, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aInt := 256

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleInt = ?", aInt)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleInt FROM ExampleTable")
```

```
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  var decodedResult int
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleInt is now the value fetched from QLDB
  return decodedResult, nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonInt(driver: QldbDriver): Promise<number> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleInt = ?", 256);

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleInt FROM ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Transforming Ion to a TypeScript Number
            const intValue: number = ionValue.numberValue();
            return intValue;
        })
    );
}
```

Python

```
def update_and_query_ion_int(txn):
    # QLDB can take in a Python int
    a_int = 256
```

```
    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleInt = ?", a_int)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleInt FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python int is a child class of Python int
        a_int = ion_value

    # example_int is now the value fetched from QLDB
    return a_int


example_int = driver.execute_lambda(lambda txn: update_and_query_ion_int(txn))
```

Float

다음 코드 예에서는 Ion 플로트 타입을 처리하는 방법을 보여줍니다.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

float exampleFloat = driver.execute((txn) -> {
    // Transforming a Java float to Ion
    float aFloat = 256;
    IonValue ionFloat = ionSystem.newFloat(aFloat);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleFloat = ?", ionFloat);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleFloat from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java float
        // Cast IonValue to IonFloat first
```

```
        aFloat = ((IonFloat)ionValue).floatValue();
    }

    // exampleFloat is now the value fetched from QLDB
    return aFloat;
});
```

.NET

C# 플로트 사용

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();

// Transforming a C# float to Ion.
float nativeFloat = 256;
IIonValue ionFloat = valueFactory.NewFloat(nativeFloat);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleFloat = ?", ionFloat);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleFloat from ExampleTable");
});

float? retrievedFloat = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# float. We cast ionValue.DoubleValue to a float
    // but be cautious, this is a down-cast and can lose precision.
    retrievedFloat = (float)ionValue.DoubleValue;
}
```

> **Note**
>
> 동기 코드로 변환하려면 `await` 및 `async` 키워드를 제거하고 `IAsyncResult` 유형을 `IResult`로 변경하세요.

C# 더블 사용

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();

// Transforming a C# double to Ion.
double nativeDouble = 256;
IIonValue ionFloat = valueFactory.NewFloat(nativeDouble);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleFloat = ?", ionFloat);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleFloat from ExampleTable");
});

double? retrievedDouble = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# double.
    retrievedDouble = ionValue.DoubleValue;
}
```

> **Note**
>
> 동기 코드로 변환하려면 `await` 및 `async` 키워드를 제거하고 `IAsyncResult` 유형을 `IResult`로 변경하세요.

Go

```
exampleFloat, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aFloat := float32(256)

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleFloat = ?", aFloat)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleFloat FROM ExampleTable")
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  // float64 would work as well
  var decodedResult float32
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleFloat is now the value fetched from QLDB
  return decodedResult, nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonFloat(driver: QldbDriver): Promise<number> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleFloat = ?", 25.6);

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleFloat FROM ExampleTable")).getResultList();
```

```
            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Transforming Ion to a TypeScript Number
            const floatValue: number = ionValue.numberValue();
            return floatValue;
        })
    );
}
```

Python

```
def update_and_query_ion_float(txn):
    # QLDB can take in a Python float
    a_float = float(256)

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleFloat = ?", a_float)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleFloat FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python float is a child class of Python float
        a_float = ion_value

    # example_float is now the value fetched from QLDB
    return a_float


example_float = driver.execute_lambda(lambda txn: update_and_query_ion_float(txn))
```

10진수

다음 코드 예에서는 Ion 10진수 타입을 처리하는 방법을 보여줍니다.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();
```

```
double exampleDouble = driver.execute((txn) -> {
    // Transforming a Java double to Ion
    double aDouble = 256;
    IonValue ionDecimal = ionSystem.newDecimal(aDouble);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleDecimal = ?", ionDecimal);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleDecimal from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java double
        // Cast IonValue to IonDecimal first
        aDouble = ((IonDecimal)ionValue).doubleValue();
    }

    // exampleDouble is now the value fetched from QLDB
    return aDouble;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();

// Transforming a C# decimal to Ion.
decimal nativeDecimal = 256.8723m;
IIonValue ionDecimal = valueFactory.NewDecimal(nativeDecimal);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleDecimal = ?", ionDecimal);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleDecimal from ExampleTable");
});

decimal? retrievedDecimal = null;
```

```
// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# decimal.
    retrievedDecimal = ionValue.DecimalValue;
}
```

> **Note**
>
> 동기 코드로 변환하려면 await 및 async 키워드를 제거하고 IAsyncResult 유형을 IResult로 변경하세요.

Go

```
exampleDecimal, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aDecimal, err := ion.ParseDecimal("256")
 if err != nil {
  return nil, err
 }

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleDecimal = ?", aDecimal)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleDecimal FROM ExampleTable")
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  var decodedResult ion.Decimal
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }
```

```
  // exampleDecimal is now the value fetched from QLDB
  return decodedResult, nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonDecimal(driver: QldbDriver): Promise<Decimal> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
            // Creating a Decimal value. Decimal is an Ion Type with high precision
            let ionDecimal: Decimal = dom.load("2.5d-6").decimalValue();
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleDecimal = ?",
 ionDecimal);

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleDecimal FROM ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Get the Ion Decimal
            ionDecimal = ionValue.decimalValue();
            return ionDecimal;
        })
    );
}
```

> **Note**
>
> `ionValue.numberValue()`를 사용하여 Ion 10진수를 JavaScript 숫자로 변환할 수도 있습니다. `ionValue.numberValue()`를 사용하면 성능이 더 좋지만 정확도는 떨어집니다.

Python

```
def update_and_query_ion_decimal(txn):
    # QLDB can take in a Python decimal
    a_decimal = Decimal(256)
```

```
    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleDecimal = ?", a_decimal)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleDecimal FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python decimal is a child class of Python decimal
        a_decimal = ion_value

    # example_decimal is now the value fetched from QLDB
    return a_decimal


example_decimal = driver.execute_lambda(lambda txn:
 update_and_query_ion_decimal(txn))
```

타임스탬프

다음 코드 예에서는 Ion 타임스탬프 타입을 처리하는 방법을 보여줍니다.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

Date exampleDate = driver.execute((txn) -> {
    // Transforming a Java Date to Ion
    Date aDate = new Date();
    IonValue ionTimestamp = ionSystem.newUtcTimestamp(aDate);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleTimestamp = ?", ionTimestamp);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleTimestamp from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java Date
        // Cast IonValue to IonTimestamp first
```

```
        aDate = ((IonTimestamp)ionValue).dateValue();
    }

    // exampleDate is now the value fetched from QLDB
    return aDate;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
using Amazon.IonDotnet;
...

IValueFactory valueFactory = new ValueFactory();

// Convert C# native DateTime to Ion.
DateTime nativeDateTime = DateTime.Now;

// First convert it to a timestamp object from Ion.
Timestamp timestamp = new Timestamp(nativeDateTime);
// Then convert to Ion timestamp.
IIonValue ionTimestamp = valueFactory.NewTimestamp(timestamp);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleTimestamp = ?", ionTimestamp);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleTimestamp from ExampleTable");
});

DateTime? retrievedDateTime = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# DateTime.
    retrievedDateTime = ionValue.TimestampValue.DateTimeValue;
}
```

> **Note**
>
> 동기 코드로 변환하려면 `await` 및 `async` 키워드를 제거하고 `IAsyncResult` 유형을 `IResult`로 변경하세요.

Go

```
exampleTimestamp, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aTimestamp := time.Date(2006, time.May, 20, 12, 30, 0, 0, time.UTC)
 if err != nil {
  return nil, err
 }

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleTimestamp = ?", aTimestamp)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleTimestamp FROM ExampleTable")
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  var decodedResult ion.Timestamp
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleTimestamp is now the value fetched from QLDB
  return decodedResult.GetDateTime(), nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonTimestamp(driver: QldbDriver): Promise<Date> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
            let exampleDateTime: Date = new Date(Date.now());
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleTimestamp = ?",
 exampleDateTime);

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleTimestamp FROM ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Transforming Ion to a TypeScript Date
            exampleDateTime = ionValue.timestampValue().getDate();
            return exampleDateTime;
        })
    );
}
```

Python

```
def update_and_query_ion_timestamp(txn):
    # QLDB can take in a Python timestamp
    a_datetime = datetime.now()

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleTimestamp = ?",
 a_datetime)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleTimestamp FROM
 ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python timestamp is a child class of Python datetime
        a_timestamp = ion_value

    # example_timestamp is now the value fetched from QLDB
    return a_timestamp
```

```
example_timestamp = driver.execute_lambda(lambda txn:
 update_and_query_ion_timestamp(txn))
```

문자열

다음 코드 예에서는 Ion 문자열 타입을 처리하는 방법을 보여줍니다.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

String exampleString = driver.execute((txn) -> {
    // Transforming a Java String to Ion
    String aString = "Hello world!";
    IonValue ionString = ionSystem.newString(aString);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleString = ?", ionString);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleString from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java String
        // Cast IonValue to IonString first
        aString = ((IonString)ionValue).stringValue();
    }

    // exampleString is now the value fetched from QLDB
    return aString;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...

IValueFactory valueFactory = new ValueFactory();
```

```
// Convert C# string to Ion.
String nativeString = "Hello world!";
IIonValue ionString = valueFactory.NewString(nativeString);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleString = ?", ionString);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleString from ExampleTable");
});

String retrievedString = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# string.
    retrievedString = ionValue.StringValue;
}
```

> **Note**
>
> 동기 코드로 변환하려면 `await` 및 `async` 키워드를 제거하고 `IAsyncResult` 유형을 `IResult`로 변경하세요.

Go

```
exampleString, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aString := "Hello World!"

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleString = ?", aString)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
```

```
 result, err := txn.Execute("SELECT VALUE ExampleString FROM ExampleTable")
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  var decodedResult string
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleString is now the value fetched from QLDB
  return decodedResult, nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonString(driver: QldbDriver): Promise<string> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleString = ?", "Hello
 World!");

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleString FROM ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Transforming Ion to a TypeScript String
            const stringValue: string = ionValue.stringValue();
            return stringValue;
        })
    );
}
```

Python

```
def update_and_query_ion_string(txn):
```

```
    # QLDB can take in a Python string
    a_string = "Hello world!"

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleString = ?", a_string)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleString FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python string is a child class of Python string
        a_string = ion_value

    # example_string is now the value fetched from QLDB
    return a_string


example_string = driver.execute_lambda(lambda txn: update_and_query_ion_string(txn))
```

Blob

다음 코드 예에서는 Ion blob 타입을 처리하는 방법을 보여줍니다.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

byte[] exampleBytes = driver.execute((txn) -> {
    // Transforming a Java byte array to Ion
    // Transform any arbitrary data to a byte array to store in QLDB
    String aString = "Hello world!";
    byte[] aByteArray = aString.getBytes();
    IonValue ionBlob = ionSystem.newBlob(aByteArray);

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleBlob = ?", ionBlob);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleBlob from ExampleTable");
    // Assume there is only one document in ExampleTable
```

```
    for (IonValue ionValue : result)
    {
        // Transforming Ion to a Java byte array
        // Cast IonValue to IonBlob first
        aByteArray = ((IonBlob)ionValue).getBytes();
    }

    // exampleBytes is now the value fetched from QLDB
    return aByteArray;
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
using System.Text;
...

IValueFactory valueFactory = new ValueFactory();

// Transform any arbitrary data to a byte array to store in QLDB.
string nativeString = "Hello world!";
byte[] nativeByteArray = Encoding.UTF8.GetBytes(nativeString);
// Transforming a C# byte array to Ion.
IIonValue ionBlob = valueFactory.NewBlob(nativeByteArray);

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleBlob = ?", ionBlob);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleBlob from ExampleTable");
});

byte[] retrievedByteArray = null;

// Assume there is only one document in ExampleTable.
await foreach (IIonValue ionValue in selectResult)
{
    // Transforming Ion to a C# byte array.
    retrievedByteArray = ionValue.Bytes().ToArray();
}
```

> **Note**
>
> 동기 코드로 변환하려면 `await` 및 `async` 키워드를 제거하고 `IAsyncResult` 유형을 `IResult`로 변경하세요.

Go

```
exampleBlob, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aBlob := []byte("Hello World!")

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleBlob = ?", aBlob)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleBlob FROM ExampleTable")
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  var decodedResult []byte
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleBlob is now the value fetched from QLDB
  return decodedResult, nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonBlob(driver: QldbDriver): Promise<Uint8Array> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
```

```
            const enc = new TextEncoder();
            let blobValue: Uint8Array = enc.encode("Hello World!");
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleBlob = ?", blobValue);

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleBlob FROM ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Transforming Ion to TypeScript Uint8Array
            blobValue = ionValue.uInt8ArrayValue();
            return blobValue;
        })
    );
}
```

Python

```
def update_and_query_ion_blob(txn):
    # QLDB can take in a Python byte array
    a_string = "Hello world!"
    a_byte_array = str.encode(a_string)

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleBlob = ?", a_byte_array)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleBlob FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python blob is a child class of Python byte array
        a_blob = ion_value

    # example_blob is now the value fetched from QLDB
    return a_blob


example_blob = driver.execute_lambda(lambda txn: update_and_query_ion_blob(txn))
```

목록

다음 코드 예는 Ion 리스트 타입을 처리하는 방법을 보여줍니다.

Java

```
// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

List<Integer> exampleList = driver.execute((txn) -> {
    // Transforming a Java List to Ion
    List<Integer> aList = new ArrayList<>();
    // Add 5 Integers to the List for the sake of example
    for (int i = 0; i < 5; i++) {
        aList.add(i);
    }
    // Create an empty Ion List
    IonList ionList = ionSystem.newEmptyList();
    // Add the 5 Integers to the Ion List
    for (Integer i : aList) {
        // Convert each Integer to Ion ints first to add it to the Ion List
        ionList.add(ionSystem.newInt(i));
    }

    // Insertion into QLDB
    txn.execute("UPDATE ExampleTable SET ExampleList = ?", (IonValue) ionList);

    // Fetching from QLDB
    Result result = txn.execute("SELECT VALUE ExampleList from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Iterate through the Ion List to map it to a Java List
        List<Integer> intList = new ArrayList<>();
        for (IonValue ionInt : (IonList)ionValue) {
            // Convert the 5 Ion ints to Java Integers
            intList.add(((IonInt)ionInt).intValue());
        }
        aList = intList;
    }

    // exampleList is now the value fetched from QLDB
    return aList;
```

```
});
```

.NET

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
using System.Collections.Generic;
...

IValueFactory valueFactory = new ValueFactory();

// Transforming a C# list to Ion.
IIonValue ionList = valueFactory.NewEmptyList();
foreach (int i in new List<int> {0, 1, 2, 3, 4})
{
    // Convert to Ion int and add to Ion list.
    ionList.Add(valueFactory.NewInt(i));
}

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleList = ?", ionList);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleList from ExampleTable");
});


List<int> retrievedList = new List<int>();
await foreach (IIonValue ionValue in selectResult)
{
    // Iterate through the Ion List to map it to a C# list.
    foreach (IIonValue ionInt in ionValue)
    {
        retrievedList.Add(ionInt.IntValue);
    }
}
```

> **Note**
>
> 동기 코드로 변환하려면 `await` 및 `async` 키워드를 제거하고 `IAsyncResult` 유형을 `IResult`로 변경하세요.

Go

```
exampleList, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aList := []int{1, 2, 3, 4, 5}

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleList = ?", aList)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleList FROM ExampleTable")
 if err != nil {
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  var decodedResult []int
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleList is now the value fetched from QLDB
  return decodedResult, nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonList(driver: QldbDriver): Promise<number[]> {
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
            let listOfNumbers: number[] = [1, 2, 3, 4, 5];
            // Updating QLDB
            await txn.execute("UPDATE ExampleTable SET ExampleList = ?",
 listOfNumbers);

            // Fetching from QLDB
```

```
            const resultList: dom.Value[] = (await txn.execute("SELECT VALUE
 ExampleList FROM ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // Get Ion List
            const ionList: dom.Value[] = ionValue.elements();
            // Iterate through the Ion List to map it to a JavaScript Array
            let intList: number[] = [];
            ionList.forEach(item => {
                // Transforming Ion to a TypeScript Number
                const intValue: number = item.numberValue();
                intList.push(intValue);
            });
            listOfNumbers = intList;
            return listOfNumbers;
        })
    );
}
```

Python

```
def update_and_query_ion_list(txn):
    # QLDB can take in a Python list
    a_list = list()
    for i in range(0, 5):
        a_list.append(i)

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleList = ?", a_list)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleList FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python blob is a child class of Python list
        a_list = ion_value

    # example_list is now the value fetched from QLDB
    return a_list
```

```
example_list = driver.execute_lambda(lambda txn: update_and_query_ion_list(txn))
```

구조체

QLDB에서 `struct` 데이터 타입은 다른 Ion 타입과 특히 다릅니다. 표에 삽입하는 최상위 문서는 `struct` 타입이어야 합니다. 문서 필드는 중첩된 `struct`를 저장할 수도 있습니다.

단순화를 위해 다음 예에서는 `ExampleString` 및 `ExampleInt` 필드만 있는 문서를 정의합니다.

Java

```
class ExampleStruct {
    public String exampleString;
    public int exampleInt;

    public ExampleStruct(String exampleString, int exampleInt) {
        this.exampleString = exampleString;
        this.exampleInt = exampleInt;
    }
}

// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();

ExampleStruct examplePojo = driver.execute((txn) -> {
    // Transforming a POJO to Ion
    ExampleStruct aPojo = new ExampleStruct("Hello world!", 256);
    // Create an empty Ion struct
    IonStruct ionStruct = ionSystem.newEmptyStruct();
    // Map the fields of the POJO to Ion values and put them in the Ion struct
    ionStruct.add("ExampleString", ionSystem.newString(aPojo.exampleString));
    ionStruct.add("ExampleInt", ionSystem.newInt(aPojo.exampleInt));

    // Insertion into QLDB
    txn.execute("INSERT INTO ExampleTable ?", ionStruct);

    // Fetching from QLDB
    Result result = txn.execute("SELECT * from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Map the fields of the Ion struct to Java values and construct a new POJO
```

```
        ionStruct = (IonStruct)ionValue;
        IonString exampleString = (IonString)ionStruct.get("ExampleString");
        IonInt exampleInt = (IonInt)ionStruct.get("ExampleInt");
        aPojo = new ExampleStruct(exampleString.stringValue(),
 exampleInt.intValue());
    }

    // examplePojo is now the document fetched from QLDB
    return aPojo;
});
```

또는 Jackson 라이브러리를 사용하여 Ion에 데이터 타입을 매핑하거나 Ion에서 데이터 타입을 매핑할 수 있습니다. 라이브러리는 다른 Ion 데이터 타입을 지원하지만 이 예에서는 `struct` 타입에 초점을 맞춥니다.

```
class ExampleStruct {
    public String exampleString;
    public int exampleInt;

    @JsonCreator
    public ExampleStruct(@JsonProperty("ExampleString") String exampleString,
                         @JsonProperty("ExampleInt") int exampleInt) {
        this.exampleString = exampleString;
        this.exampleInt = exampleInt;
    }

    @JsonProperty("ExampleString")
    public String getExampleString() {
        return this.exampleString;
    }

    @JsonProperty("ExampleInt")
    public int getExampleInt() {
        return this.exampleInt;
    }
}

// Instantiate an IonSystem from the Ion library
IonSystem ionSystem = IonSystemBuilder.standard().build();
// Instantiate an IonObjectMapper from the Jackson library
IonObjectMapper ionMapper = new IonValueMapper(ionSystem);

ExampleStruct examplePojo = driver.execute((txn) -> {
```

```
    // Transforming a POJO to Ion
    ExampleStruct aPojo = new ExampleStruct("Hello world!", 256);
    IonValue ionStruct;
    try  {
        // Use the mapper to convert Java objects into Ion
        ionStruct = ionMapper.writeValueAsIonValue(aPojo);
    } catch (IOException e) {
        // Wrap the exception and throw it for the sake of simplicity in this
 example
        throw new RuntimeException(e);
    }

    // Insertion into QLDB
    txn.execute("INSERT INTO ExampleTable ?", ionStruct);

    // Fetching from QLDB
    Result result = txn.execute("SELECT * from ExampleTable");
    // Assume there is only one document in ExampleTable
    for (IonValue ionValue : result)
    {
        // Use the mapper to convert Ion to Java objects
        try {
            aPojo = ionMapper.readValue(ionValue, ExampleStruct.class);
        } catch (IOException e) {
            // Wrap the exception and throw it for the sake of simplicity in this
 example
            throw new RuntimeException(e);
        }
    }

    // examplePojo is now the document fetched from QLDB
    return aPojo;
});
```

.NET

Amazon.QLDB.Driver.Serialization 라이브러리를 사용하여 네이티브 C# 데이터 타입을 Ion과 매핑하거나 Ion에서 네이티브 C# 데이터 타입을 매핑할 수 있습니다. 라이브러리는 다른 Ion 데이터 타입을 지원하지만 이 예에서는 `struct` 타입에 초점을 맞춥니다.

```
using Amazon.QLDB.Driver.Generic;
using Amazon.QLDB.Driver.Serialization;
...
```

```
IAsyncQldbDriver driver = AsyncQldbDriver.Builder()
    .WithLedger("vehicle-registration")
    // Add Serialization library
    .WithSerializer(new ObjectSerializer())
    .Build();

// Creating a C# POCO.
ExampleStruct exampleStruct = new ExampleStruct
{
    ExampleString = "Hello world!",
    ExampleInt = 256
};

IAsyncResult<ExampleStruct> selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute(txn.Query<Document>("UPDATE ExampleTable SET ExampleStruct
 = ?", exampleStruct));

    // Fetching from QLDB.
    return await txn.Execute(txn.Query<ExampleStruct>("SELECT VALUE ExampleStruct
 from ExampleTable"));
});

await foreach (ExampleStruct row in selectResult)
{
    Console.WriteLine(row.ExampleString);
    Console.WriteLine(row.ExampleInt);
}
```

> **Note**
>
> 동기 코드로 변환하려면 `await` 및 `async` 키워드를 제거하고 `IAsyncResult` 유형을 `IResult`로 변경하세요.

또는 Amazon.IonDotnet.Builders 라이브러리를 사용하여 Ion 데이터 타입을 처리할 수도 있습니다.

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;
...
```

```
IValueFactory valueFactory = new ValueFactory();

// Creating Ion struct.
IIonValue ionStruct = valueFactory.NewEmptyStruct();
ionStruct.SetField("ExampleString", valueFactory.NewString("Hello world!"));
ionStruct.SetField("ExampleInt", valueFactory.NewInt(256));

IAsyncResult selectResult = await driver.Execute(async txn =>
{
    // Insertion into QLDB.
    await txn.Execute("UPDATE ExampleTable SET ExampleStruct = ?", ionStruct);

    // Fetching from QLDB.
    return await txn.Execute("SELECT VALUE ExampleStruct from ExampleTable");
});

string retrievedString = null;
int? retrievedInt = null;

await foreach (IIonValue ionValue in selectResult)
{
    retrievedString = ionValue.GetField("ExampleString").StringValue;
    retrievedInt = ionValue.GetField("ExampleInt").IntValue;
}
```

Go

```
exampleStruct, err := driver.Execute(context.Background(), func(txn
 qldbdriver.Transaction) (interface{}, error) {
 aStruct := map[string]interface{} {
  "ExampleString": "Hello World!",
  "ExampleInt": 256,
 }

 // Insertion into QLDB
 _, err = txn.Execute("UPDATE ExampleTable SET ExampleStruct = ?", aStruct)
 if err != nil {
  return nil, err
 }

 // Fetching from QLDB
 result, err := txn.Execute("SELECT VALUE ExampleStruct FROM ExampleTable")
 if err != nil {
```

```
  return nil, err
 }

 // Assume there is only one document in ExampleTable
 if result.Next(txn) {
  var decodedResult map[string]interface{}
  err := ion.Unmarshal(result.GetCurrentData(), &decodedResult)
  if err != nil {
   return nil, err
  }

  // exampleStruct is now the value fetched from QLDB
  return decodedResult, nil
 }
 return nil, result.Err()
})
```

Node.js

```
async function queryIonStruct(driver: QldbDriver): Promise<any> {
    let exampleStruct: any = {stringValue: "Hello World!", intValue: 256};
    return (driver.executeLambda(async (txn: TransactionExecutor) => {
            // Inserting into QLDB
            await txn.execute("INSERT INTO ExampleTable ?", exampleStruct);

            // Fetching from QLDB
            const resultList: dom.Value[] = (await txn.execute("SELECT * FROM
 ExampleTable")).getResultList();

            // Assume there is only one document in ExampleTable
            const ionValue: dom.Value = resultList[0];
            // We can get all the keys of Ion struct and their associated values
            const ionFieldNames: string[] = ionValue.fieldNames();

            // Getting key and value of Ion struct to TypeScript String and Number
            const nativeStringVal: string =
 ionValue.get(ionFieldNames[0]).stringValue();
            const nativeIntVal: number =
 ionValue.get(ionFieldNames[1]).numberValue();
            // Alternatively, we can access to Ion struct fields, using their
 literal field names:
            //   const nativeStringVal = ionValue.get("stringValue").stringValue();
            //   const nativeIntVal = ionValue.get("intValue").numberValue();
```

```
            exampleStruct = {[ionFieldNames[0]]: nativeStringVal,
 [ionFieldNames[1]]: nativeIntVal};
            return exampleStruct;
        })
    );
}
```

Python

```
def update_and_query_ion_struct(txn):
    # QLDB can take in a Python struct
    a_struct = {"ExampleString": "Hello world!", "ExampleInt": 256}

    # Insertion into QLDB
    txn.execute_statement("UPDATE ExampleTable SET ExampleStruct = ?", a_struct)

    # Fetching from QLDB
    cursor = txn.execute_statement("SELECT VALUE ExampleStruct FROM ExampleTable")

    # Assume there is only one document in ExampleTable
    for ion_value in cursor:
        # Ion Python struct is a child class of Python struct
        a_struct = ion_value

    # example_struct is now the value fetched from QLDB
    return a_struct


example_struct = driver.execute_lambda(lambda txn: update_and_query_ion_struct(txn))
```

Null 값 및 동적 타입

QLDB는 개방형 콘텐츠를 지원하며 문서 필드에 대한 스키마 또는 데이터 타입 정의를 적용하지 않습니다. QLDB 문서에 Ion null 값을 저장할 수도 있습니다. 위의 모든 예에서는 반환된 각 데이터 타입이 알려져 있고 null이 아니라고 가정합니다. 다음 예에서는 데이터 타입을 알 수 없거나 null일 가능성이 있는 경우 Ion을 사용하는 방법을 보여줍니다.

Java

```
// Empty variables
```

```
String exampleString = null;
Integer exampleInt = null;

// Assume ionValue is some queried data from QLDB
IonValue ionValue = null;

// Check the value type and assign it to the variable if it is not null
if (ionValue.getType() == IonType.STRING) {
    if (ionValue.isNullValue()) {
        exampleString = null;
    } else {
        exampleString = ((IonString)ionValue).stringValue();
    }
} else if (ionValue.getType() == IonType.INT) {
    if (ionValue.isNullValue()) {
        exampleInt = null;
    } else {
        exampleInt = ((IonInt)ionValue).intValue();
    }
};

// Creating null values
IonSystem ionSystem = IonSystemBuilder.standard().build();

// A null value still has an Ion type
IonString ionString;
if (exampleString == null) {
    // Specifically a null string
    ionString = ionSystem.newNullString();
} else {
    ionString = ionSystem.newString(exampleString);
}

IonInt ionInt;
if (exampleInt == null) {
    // Specifically a null int
    ionInt = ionSystem.newNullInt();
} else {
    ionInt = ionSystem.newInt(exampleInt);
}

// Special case regarding null!
// There is a generic null type which has Ion type 'Null'.
// The above null values still have the types 'String' and 'Int'
```

```
IonValue specialNull = ionSystem.newNull();
if (specialNull.getType() == IonType.NULL) {
    // This is true!
}
if (specialNull.isNullValue()) {
    // This is also true!
}
if (specialNull.getType() == IonType.STRING || specialNull.getType() == IonType.INT)
 {
    // This is false!
}
```

.NET

```
// Empty variables.
string exampleString = null;
int? exampleInt = null;

// Assume ionValue is some queried data from QLDB.
IIonValue ionValue;

if (ionValue.Type() == IonType.String)
{
    exampleString = ionValue.StringValue;
}
else if (ionValue.Type() == IonType.Int)
{
    if (ionValue.IsNull)
    {
        exampleInt = null;
    }
    else
    {
        exampleInt = ionValue.IntValue;
    }
};

// Creating null values.
IValueFactory valueFactory = new ValueFactory();

// A null value still has an Ion type.
IIonValue ionString = valueFactory.NewString(exampleString);
```

```
IIonValue ionInt;
if (exampleInt == null)
{
    // Specifically a null int.
    ionInt = valueFactory.NewNullInt();
}
else
{
    ionInt = valueFactory.NewInt(exampleInt.Value);
}

// Special case regarding null!
// There is a generic null type which has Ion type 'Null'.
IIonValue specialNull = valueFactory.NewNull();
if (specialNull.Type() == IonType.Null) {
    // This is true!
}
if (specialNull.IsNull) {
    // This is also true!
}
```

Go

수렴 제한

Go에서 `nil` 값을 수렴한 다음 수렴 취소하면 해당 값의 타입이 유지되지 않습니다. `nil` 값은 Ion null로 수렴되지만, Ion null은 `nil`이 아닌 값 0으로 수렴 취소됩니다.

```
ionNull, err := ion.MarshalText(nil) // ionNull is set to ion null
if err != nil {
    return
}

var result int
err = ion.Unmarshal(ionNull, &result) // result unmarshals to 0
if err != nil {
    return
}
```

Node.js

```
// Empty variables
let exampleString: string;
```

```
let exampleInt: number;

// Assume ionValue is some queried data from QLDB
// Check the value type and assign it to the variable if it is not null
if (ionValue.getType() === IonTypes.STRING) {
    if (ionValue.isNull()) {
        exampleString = null;
    } else {
        exampleString = ionValue.stringValue();
    }
} else if (ionValue.getType() === IonTypes.INT) {
    if (ionValue.isNull()) {
        exampleInt = null;
    } else {
        exampleInt = ionValue.numberValue();
    }
}

// Creating null values
if (exampleString === null) {
    ionString = dom.load('null.string');
} else {
    ionString = dom.load.of(exampleString);
}

if (exampleInt === null) {
    ionInt = dom.load('null.int');
} else {
    ionInt = dom.load.of(exampleInt);
}

// Special case regarding null!
// There is a generic null type which has Ion type 'Null'.
// The above null values still have the types 'String' and 'Int'
specialNull: dom.Value = dom.load("null.null");
if (specialNull.getType() === IonType.NULL) {
    // This is true!
}
if (specialNull.getType() === IonType.STRING || specialNull.getType() ===
 IonType.INT) {
    // This is false!
}
```

Python

```
# Empty variables
example_string = None
example_int = None

# Assume ion_value is some queried data from QLDB
# Check the value type and assign it to the variable if it is not null
if ion_value.ion_type == IonType.STRING:
    if isinstance(ion_value, IonPyNull):
        example_string = None
    else:
        example_string = ion_value
elif ion_value.ion_type == IonType.INT:
    if isinstance(ion_value, IonPyNull):
        example_int = None
    else:
        example_int = ion_value

# Creating Ion null values
if example_string is None:
    # Specifically a null string
    ion_string = loads("null.string")
else:
    # QLDB can take in Python string
    ion_string = example_string
if example_int is None:
    # Specifically a null int
    ion_int = loads("null.int")
else:
    # QLDB can take in Python int
    ion_int = example_int


# Special case regarding null!
# There is a generic null type which has Ion type 'Null'.
# The above null values still have the types 'String' and 'Int'
special_null = loads("null.null")
if special_null.ion_type == IonType.NULL:
    # This is true!
if special_null.ion_type == IonType.STRING or special_null.ion_type == IonType.INT:
    # This is false!
```

JSON으로 하향 변환

애플리케이션에 JSON 병용성이 필요한 경우, Amazon Ion 데이터를 JSON으로 하향 변환할 수 있습니다. 하지만 JSON에 없는 풍부한 Ion 유형을 데이터에 사용하는 경우 Ion을 JSON으로 변환해도 손실이 발생합니다.

Ion에서 JSON으로의 변환 규칙에 대한 자세한 내요은 Amazon Ion Cookbook의 JSON으로 하향 변환을 참조하세요.

Amazon QLDB에서 데이터 및 기록 작업

다음 항목에서는 CRUD(생성/읽기/업데이트/삭제) 명령문의 기본 예제를 제공합니다. QLDB 콘솔의 PartiQL 편집기 또는 QLDB 쉘을 사용하여 이러한 명령문을 수동으로 실행할 수 있습니다. 또한 이 가이드는 원장을 변경할 때 QLDB가 데이터를 처리하는 프로세스를 안내합니다.

QLDB는 PartiQL 쿼리 언어를 지원합니다.

QLDB 드라이버를 사용하여 유사한 명령문을 프로그래밍 방식으로 실행하는 방법을 보여주는 코드 예제는 드라이버 시작하기의 자습서를 참조하세요.

> **Tip**
>
> 다음은 QLDB에서 PartiQL을 사용하기 위한 팁과 모범 사례를 간략하게 요약한 것입니다.
>
> - 동시성 및 트랜잭션 제한 이해 - `SELECT` 쿼리를 포함한 모든 명령문은 30초의 트랜잭션 시간 제한을 포함하여 OCC(Optimistic Concurrency Control) 충돌 및 트랜잭션 제한의 영향을 받습니다.
> - 인덱스 사용 – 카디널리티가 높은 인덱스를 사용하고 대상 쿼리를 실행하여 명령문을 최적화하고 전체 테이블 스캔을 방지합니다. 자세한 내용은 쿼리 성능 최적화 섹션을 참조하세요.
> - 동등 조건자 사용 – 인덱싱된 조회에는 동등 연산자(`=` 또는 `IN`)가 필요합니다. 부등 연산자(`<`, `>`, `LIKE`, `BETWEEN`)는 인덱싱된 조회에 적합하지 않으며 테이블 전체를 스캔해야 합니다.
> - 내부 조인만 사용 — QLDB는 내부 조인만 지원합니다. 가장 좋은 방법은 조인하려는 각 테이블에 대해 인덱싱된 필드를 조인하는 것입니다. 조인 기준과 동등 조건자 모두에 대해 높은 카디널리티 인덱스를 선택하세요.

주제

- 문서 개정본 수정하기
- 쿼리 성능 최적화
- PartiQL 문 통계 가져오기
- 시스템 카탈로그 쿼리
- 테이블 관리
- 인덱스 관리
- Amazon QLDB의 고유 ID

인덱스가 포함된 테이블 생성 및 문서 삽입

Amazon QLDB 원장을 생성한 후 첫 번째 단계는 기본 CREATE TABLE 명령문이 포함된 테이블을 생성하는 것입니다. 테이블은 Amazon Ion `struct`의 데이터 세트인 QLDB 문서로 구성됩니다.

주제

- 테이블 및 인덱스 생성
- 문서 삽입하기

테이블 및 인덱스 생성

테이블은 네임스페이스 없이 대소문자를 구분하는 간단한 이름을 사용합니다. QLDB는 개방형 콘텐츠를 지원하며 스키마를 적용하지 않으므로 테이블을 만들 때 속성이나 데이터 유형을 정의하지 않습니다.

```
CREATE TABLE VehicleRegistration
```

```
CREATE TABLE Vehicle
```

`CREATE TABLE` 명령문은 새 테이블의 시스템 할당 ID를 반환합니다. QLDB의 모든 시스템 할당 ID는 Base62로 인코딩된 문자열로 각각 표시되는 범용 고유 식별자(UUID)입니다.

> **Note**
>
> 테이블을 생성하는 동안 선택적으로 테이블 리소스의 태그를 정의할 수 있습니다. 자세한 방법은 테이블 생성 시 태그 지정 섹션을 참조하세요.

테이블에 인덱스를 생성하여 쿼리 성능을 최적화할 수도 있습니다.

```
CREATE INDEX ON VehicleRegistration (VIN)
```

```
CREATE INDEX ON VehicleRegistration (LicensePlateNumber)
```

```
CREATE INDEX ON Vehicle (VIN)
```

> **Important**
>
> QLDB는 문서를 효율적으로 조회하기 위한 인덱스가 필요합니다. 인덱스가 없으면 QLDB는 문서를 읽을 때 전체 테이블 스캔을 수행해야 합니다. 이로 인해 동시성 충돌 및 트랜잭션 시간 초과를 포함하여 대규모 테이블에서 성능 문제가 발생할 수 있습니다.
> 인덱싱된 필드 또는 문서 ID(예: = 또는 IN)에서 동등 연산자를 사용하여 WHERE 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 자세한 내용은 쿼리 성능 최적화을 참조하십시오.

인덱스를 생성할 때는 다음 제약 조건에 유의하세요.

- 인덱스는 단일 최상위 필드에만 생성할 수 있습니다. 복합, 중첩, 고유 및 함수 기반 인덱스는 지원되지 않습니다.
- `list` 및 `struct`를 비롯한 모든 Ion 데이터 유형에 대해 인덱스를 생성할 수 있습니다. 그러나 Ion 유형에 관계없이 전체 Ion 값이 같아야만 인덱스 조회를 수행할 수 있습니다. 예를 들어 `list` 형식을 인덱스로 사용하는 경우 목록 내에서 한 항목씩 인덱스 검색을 수행할 수 없습니다.
- 동등 조건자(예: `WHERE indexedField = 123` 또는 `WHERE indexedField IN (456, 789)`)를 사용할 때만 쿼리 성능이 향상됩니다.

  QLDB는 쿼리 조건자의 부등을 인정하지 않습니다. 따라서 범위 필터링된 스캔은 구현되지 않습니다.
- 인덱싱된 필드 이름은 대소문자를 구분하며 최대 128자입니다.
- QLDB에서의 인덱스 생성은 비동기적으로 이루어집니다. 비어 있지 않은 테이블에서 인덱스 빌드를 완료하는 데 걸리는 시간은 테이블 크기에 따라 다릅니다. 자세한 내용은 인덱스 관리을 참조하십시오.

문서 삽입하기

그러면 테이블에 문서를 삽입할 수 있습니다. QLDB 문서는 Amazon Ion 형식으로 저장됩니다. 다음 PartiQL INSERT 명령문에는 Amazon QLDB 콘솔 시작하기에서 사용된 차량 등록 샘플 데이터의 하위 집합이 포함됩니다.

```
INSERT INTO VehicleRegistration
<< {
    'VIN' : '1N4AL11D75C109151',
    'LicensePlateNumber' : 'LEWISR261LL',
    'State' : 'WA',
    'City' : 'Seattle',
    'PendingPenaltyTicketAmount' : 90.25,
    'ValidFromDate' : `2017-08-21T`,
    'ValidToDate' : `2020-05-11T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId' : '294jJ3YUoH1IEEm8GSabOs' },
        'SecondaryOwners' : [ { 'PersonId' : '5Ufgdlnj06gF5CWcOIu64s' } ]
    }
},
{
    'VIN' : 'KM8SRDHF6EU074761',
    'LicensePlateNumber' : 'CA762X',
    'State' : 'WA',
    'City' : 'Kent',
    'PendingPenaltyTicketAmount' : 130.75,
    'ValidFromDate' : `2017-09-14T`,
    'ValidToDate' : `2020-06-25T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId': 'IN7MvYtUjkp1GMZu0F6CG9' },
        'SecondaryOwners' : []
    }
} >>
```

```
INSERT INTO Vehicle
<< {
    'VIN' : '1N4AL11D75C109151',
    'Type' : 'Sedan',
    'Year' : 2011,
    'Make' : 'Audi',
    'Model' : 'A5',
    'Color' : 'Silver'
```

```
} ,
{
    'VIN' : 'KM8SRDHF6EU074761',
    'Type' : 'Sedan',
    'Year' : 2015,
    'Make' : 'Tesla',
    'Model' : 'Model S',
    'Color' : 'Blue'
} >>
```

PartiQL 구문 및 시맨틱

- 필드 이름은 작은 따옴표로 묶습니다(`'...'`).
- 문자열 값도 작은 따옴표로 묶습니다(`'...'`).
- 타임스탬프는 백틱으로 묶습니다(`` `...` ``). 백틱은 모든 Ion 리터럴을 나타내는 데 사용할 수 있습니다.
- 정수와 소수는 표시할 필요가 없는 리터럴 값입니다.

PartiQL의 구문 및 시맨틱에 대한 자세한 내용은 Amazon QLDB에서 PartiQL을 사용한 Ion 쿼리 섹션을 참조하세요.

`INSERT` 명령문은 버전 번호가 0인 문서의 초기 개정을 생성합니다. 각 문서를 고유하게 식별하기 위해 QLDB는 문서 ID를 메타데이터의 일부로 할당합니다. Insert 문은 삽입된 각 문서의 ID를 반환합니다.

> **Important**
>
> QLDB는 스키마를 적용하지 않으므로 동일한 문서를 테이블에 여러 번 삽입할 수 있습니다. 각 Insert 명령문은 저널에 별도의 문서 항목을 커밋하고 QLDB는 각 문서에 고유한 ID를 할당합니다.

테이블에 삽입한 문서를 쿼리하는 방법을 알아보려면 데이터 쿼리로 이동합니다.

데이터 쿼리

사용자 뷰는 사용자 데이터의 삭제되지 않은 최신 버전만 반환합니다. 이는 Amazon QLDB의 기본 뷰입니다. 즉, 데이터만 쿼리하려는 경우에는 특별한 한정자가 필요하지 않습니다.

다음 쿼리 예제의 구문 및 파라미터에 대한 자세한 내용은 Amazon QLDB PartiQL 참조의 SELECT 섹션을 참조하세요.

주제

- 기본 쿼리
- 프로젝션 및 필터
- 조인
- 중첩된 데이터

기본 쿼리

기본 SELECT 쿼리는 테이블에 삽입한 문서를 반환합니다.

> **Warning**
>
> 인덱싱된 조회 없이 QLDB에서 쿼리를 실행하면 전체 테이블 스캔이 호출됩니다. PartiQL은 SQL과 호환되므로 이러한 쿼리를 지원합니다. 하지만 QLDB의 프로덕션 사용 사례에 대해서는 테이블 스캔을 실행하지 마세요. 테이블 스캔은 동시성 충돌 및 트랜잭션 시간 초과를 포함하여 대규모 테이블에서 성능 문제를 일으킬 수 있습니다.
>
> 인덱싱된 필드 또는 문서 ID(예: `WHERE indexedField = 123` 또는 `WHERE indexedField IN (456, 789)`)에서 동등 연산자를 사용하여 WHERE 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 자세한 내용은 쿼리 성능 최적화을 참조하십시오.

다음 쿼리는 이전에 인덱스가 포함된 테이블 생성 및 문서 삽입에 삽입한 차량 등록 문서에 대한 결과를 보여줍니다. 결과 순서는 구체적이지 않으며 각 SELECT 쿼리마다 다를 수 있습니다. QLDB의 모든 쿼리에 대해 결과 순서에 의존해서는 안 됩니다.

```
SELECT * FROM VehicleRegistration
WHERE LicensePlateNumber IN ('LEWISR261LL', 'CA762X')
```

```
{
    VIN: "1N4AL11D75C109151",
    LicensePlateNumber: "LEWISR261LL",
    State: "WA",
    City: "Seattle",
```

```
    PendingPenaltyTicketAmount: 90.25,
    ValidFromDate: 2017-08-21T,
    ValidToDate: 2020-05-11T,
    Owners: {
        PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSabOs" },
        SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }]
    }
},
{
    VIN: "KM8SRDHF6EU074761",
    LicensePlateNumber: "CA762X",
    State: "WA",
    City: "Kent",
    PendingPenaltyTicketAmount: 130.75,
    ValidFromDate: 2017-09-14T,
    ValidToDate: 2020-06-25T,
    Owners: {
        PrimaryOwner: { PersonId: "IN7MvYtUjkp1GMZu0F6CG9" },
        SecondaryOwners: []
    }
}
```

```
SELECT * FROM Vehicle
WHERE VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

```
{
    VIN: "1N4AL11D75C109151",
    Type: "Sedan",
    Year: 2011,
    Make: "Audi",
    Model: "A5",
    Color: "Silver"
},
{
    VIN: "KM8SRDHF6EU074761",
    Type: "Sedan",
    Year: 2015,
    Make: "Tesla",
    Model: "Model S",
    Color: "Blue"
}
```

> **Important**
>
> PartiQL에서는 작은따옴표를 사용하여 DML(데이터 조작 언어) 또는 쿼리문의 문자열을 나타냅니다. 하지만 QLDB 콘솔과 QLDB 쉘은 Amazon Ion 텍스트 형식으로 쿼리 결과를 반환하므로 문자열이 큰따옴표로 묶여 있습니다.
> 이 구문을 사용하면 PartiQL 쿼리 언어가 SQL 호환성을 유지하고 Amazon Ion 텍스트 형식이 JSON 호환성을 유지할 수 있습니다.

프로젝션 및 필터

프로젝션(대상 `SELECT`) 및 기타 표준 필터(`WHERE` 절)를 수행할 수 있습니다. 다음 쿼리는 `VehicleRegistration` 테이블에 있는 문서 필드의 하위 집합을 반환합니다. 이는 다음 기준에 따라 차량을 필터링합니다.

- 문자열 필터 — 시애틀에 등록되어 있습니다.
- 십진수 필터 — 보류 중인 페널티 티켓 금액이 `100.0`보다 적습니다.
- 날짜 필터 — 2019년 9월 4일 이후에 유효한 등록일을 갖습니다.

```
SELECT r.VIN, r.PendingPenaltyTicketAmount, r.Owners
FROM VehicleRegistration AS r
WHERE r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
AND r.City = 'Seattle' --string
AND r.PendingPenaltyTicketAmount < 100.0 --decimal
AND r.ValidToDate >= `2019-09-04T` --timestamp with day precision
```

```
{
    VIN: "1N4AL11D75C109151",
    PendingPenaltyTicketAmount: 90.25,
    Owners: {
        PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSabOs" },
        SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }]
    }
}
```

조인

내부 조인 쿼리를 작성할 수도 있습니다. 다음 예제는 등록된 차량의 속성과 함께 모든 등록 문서를 반환하는 암시적 내부 조인 쿼리를 보여줍니다.

```
SELECT * FROM VehicleRegistration AS r, Vehicle AS v
WHERE r.VIN = v.VIN
AND r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

```
{
    VIN: "1N4AL11D75C109151",
    LicensePlateNumber: "LEWISR261LL",
    State: "WA",
    City: "Seattle",
    PendingPenaltyTicketAmount: 90.25,
    ValidFromDate: 2017-08-21T,
    ValidToDate: 2020-05-11T,
    Owners: {
        PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSabOs" },
        SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }]
    },
    Type: "Sedan",
    Year: 2011,
    Make: "Audi",
    Model: "A5",
    Color: "Silver"
},
{
    VIN: "KM8SRDHF6EU074761",
    LicensePlateNumber: "CA762X",
    State: "WA",
    City: "Kent",
    PendingPenaltyTicketAmount: 130.75,
    ValidFromDate: 2017-09-14T,
    ValidToDate: 2020-06-25T,
    Owners: {
        PrimaryOwner: { PersonId: "IN7MvYtUjkp1GMZu0F6CG9" },
        SecondaryOwners: []
    },
    Type: "Sedan",
    Year: 2015,
    Make: "Tesla",
    Model: "Model S",
```

```
    Color: "Blue"
}
```

또는 다음과 같이 명시적 구문으로 동일한 내부 조인 쿼리를 작성할 수 있습니다.

```
SELECT * FROM VehicleRegistration AS r INNER JOIN Vehicle AS v
ON r.VIN = v.VIN
WHERE r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

중첩된 데이터

QLDB의 PartiQL을 사용하여 문서의 중첩된 데이터를 쿼리할 수 있습니다. 다음 예제는 중첩된 데이터를 평면화하는 상관 하위 쿼리를 보여줍니다. 여기서 이 @ 문자는 엄밀히 따지자면 선택 사항입니다. 하지만 이는 Owners이라는 다른 컬렉션(있는 경우)이 아니라 VehicleRegistration 안에 Owners 구조를 적용하고자 한다는 것을 명시적으로 나타냅니다.

```
SELECT
    r.VIN,
    o.SecondaryOwners
FROM
    VehicleRegistration AS r, @r.Owners AS o
WHERE
    r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

```
{
    VIN: "1N4AL11D75C109151",
    SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }]
},
{
    VIN: "KM8SRDHF6EU074761",
    SecondaryOwners: []
}
```

다음은 내부 조인 외에도 중첩된 데이터를 프로젝션하는 SELECT 목록의 하위 쿼리를 보여줍니다.

```
SELECT
    v.Make,
    v.Model,
    (SELECT VALUE o.PrimaryOwner.PersonId FROM @r.Owners AS o) AS PrimaryOwner
FROM
    VehicleRegistration AS r, Vehicle AS v
```

```
WHERE
    r.VIN = v.VIN AND r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

```
{
    Make: "Audi",
    Model: "A5",
    PrimaryOwner: ["294jJ3YUoH1IEEm8GSabOs"]
},
{
    Make: "Tesla",
    Model: "Model S",
    PrimaryOwner: ["IN7MvYtUjkp1GMZu0F6CG9"]
}
```

다음 쿼리는 `VehicleRegistration` 문서에 대해 `Owners.SecondaryOwners` 목록에 있는 각 사람의 `PersonId` 및 인덱스(서수) 번호를 반환합니다.

```
SELECT s.PersonId, owner_idx
FROM VehicleRegistration AS r, @r.Owners.SecondaryOwners AS s AT owner_idx
WHERE r.VIN = '1N4AL11D75C109151'
```

```
{
    PersonId: "5Ufgdlnj06gF5CWcOIu64s",
    owner_idx: 0
}
```

문서 메타데이터를 쿼리하는 방법을 알아보려면 문서 메타데이터 쿼리 섹션으로 이동하세요.

문서 메타데이터 쿼리

`INSERT` 명령문은 버전 번호가 0인 문서의 초기 개정을 생성합니다. 각 문서를 고유하게 식별하기 위해 Amazon QLDB는 메타데이터의 일부로 문서ID를 할당합니다.

QLDB는 문서 ID 및 버전 번호 외에도 각 문서에 대해 시스템에서 생성한 다른 메타데이터를 테이블에 저장합니다. 이 메타데이터에는 트랜잭션 정보, 저널 속성 및 문서의 해시 값이 포함됩니다.

시스템에서 할당한 모든 ID는 Base62로 인코딩된 문자열로 각각 표시되는 범용 고유 식별자(UUID)입니다. 자세한 내용은 Amazon QLDB의 고유 ID을 참조하십시오.

주제

커밋된 뷰

커밋된 뷰를 쿼리하여 문서 메타데이터에 액세스할 수 있습니다. 이 뷰는 사용자 테이블에 직접 해당하는 시스템 정의 테이블의 문서를 반환합니다. 여기에는 데이터와 시스템에서 생성한 메타데이터의 최신 커밋되고 삭제되지 않은 개정이 포함됩니다. 이 뷰를 쿼리하려면 쿼리의 테이블 이름에 접두사 `_ql_committed_`를 추가합니다. (접두사 `_ql_`는 QLDB에서 시스템 객체용으로 예약되어 있습니다.)

```
SELECT * FROM _ql_committed_VehicleRegistration AS r
WHERE r.data.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

이전에 인덱스가 포함된 테이블 생성 및 문서 삽입에 삽입한 데이터를 사용하여 이 쿼리의 출력에는 삭제되지 않은 각 문서의 최신 개정 버전에 대한 시스템 내용이 표시됩니다. 시스템 문서의 메타데이터는 `metadata` 필드에 중첩되어 있고 사용자 데이터는 `data` 필드에 중첩되어 있습니다.

```
{
    blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:14
    },
    hash:{{wCsmM6qD4STxz0WYmE+47nZvWtcCz9D6zNtCiM5GoWg=}},
    data:{
        VIN: "1N4AL11D75C109151",
        LicensePlateNumber: "LEWISR261LL",
        State: "WA",
        City: "Seattle",
        PendingPenaltyTicketAmount: 90.25,
        ValidFromDate: 2017-08-21T,
        ValidToDate: 2020-05-11T,
        Owners: {
            PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSabOs" },
            SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }]
        }
    },
    metadata:{
        id:"3Qv67yjXEwB9SjmvkuG6Cp",
        version:0,
        txTime:2019-06-05T20:53:321d-3Z,
        txId:"HgXAkLjAtV0HQ4lNYdzX60"
```

```
    }
},
{
    blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:14
    },
    hash:{{wPuwH60TtcCvg/23BFp+redRXuCALkbDihkEvCX22Jk=}},
    data:{
        VIN: "KM8SRDHF6EU074761",
        LicensePlateNumber: "CA762X",
        State: "WA",
        City: "Kent",
        PendingPenaltyTicketAmount: 130.75,
        ValidFromDate: 2017-09-14T,
        ValidToDate: 2020-06-25T,
        Owners: {
            PrimaryOwner: { PersonId: "IN7MvYtUjkp1GMZu0F6CG9" },
            SecondaryOwners: []
        }
    },
    metadata:{
        id:"JOzfB3lWqGU727mpPeWyxg",
        version:0,
        txTime:2019-06-05T20:53:321d-3Z,
        txId:"HgXAkLjAtV0HQ4lNYdzX60"
    }
}
```

커밋된 뷰 필드

- `blockAddress` — 문서 개정이 커밋된 원장 저널의 블록 위치입니다. 암호화 검증에 사용할 수 있는 주소에는 다음과 같은 두 개의 필드가 있습니다.
  - `strandId` - 블록을 포함하는 저널 스트랜드의 고유 ID입니다.
  - `sequenceNo` - 스트랜드 내 블록 위치를 지정하는 인덱스 번호입니다.

> **Note**
>
> 이 예제의 두 문서는 동일한 `sequenceNo`의 `blockAddress`를 가지고 있습니다. 이러한 문서는 단일 트랜잭션(이 경우에는 단일 명령문) 내에 삽입되었으므로 동일한 블록에서 커밋되었습니다.

- `hash` - 문서 개정을 고유하게 나타내는 SHA-256 Ion 해시 값입니다. 해시는 개정판의 `data` 및 `metadata` 필드를 포함하며 암호화 검증에 사용할 수 있습니다.
- `data` - 문서의 사용자 데이터 속성입니다.

  개정을 수정하면 이 `data` 구조가 `dataHash` 필드로 대체되며 필드의 값은 제거된 `data` 구조의 Ion 해시입니다.
- `metadata` - 문서의 메타데이터 속성입니다.
  - `id` - 시스템에서 할당한 문서의 고유 ID입니다.
  - `version` - 문서의 버전 번호입니다. 이는 0으로 시작하는 정수로, 문서가 개정될 때마다 증가합니다.
  - `txTime` - 문서 개정이 저널에 커밋된 시점의 타임스탬프입니다.
  - `txId` - 문서 개정을 커밋한 트랜잭션의 고유 ID입니다.

커밋된 뷰와 사용자 뷰 조인

커밋된 뷰의 테이블을 사용자 뷰의 테이블과 조인하는 쿼리를 작성할 수 있습니다. 예를 들어 한 테이블의 `id` 문서를 다른 테이블의 사용자 정의 필드와 조인하고자 할 수 있습니다.

다음 쿼리는 `PersonId` 및 문서 `id` 필드에서 각각 `DriversLicense` 및 `Person`라는 두 테이블을 조인하며, 후자에 대해 커밋된 뷰를 사용합니다.

```
SELECT * FROM DriversLicense AS d INNER JOIN _ql_committed_Person AS p
ON d.PersonId = p.metadata.id
WHERE p.metadata.id = '1CWScY2qHYI9G88C2SjvtH'
```

기본 사용자 뷰에서 문서 ID 필드를 쿼리하는 방법을 알아보려면 BY 절을 사용하여 문서 ID 쿼리하기을 참조하세요.

BY 절을 사용하여 문서 ID 쿼리하기

고유 식별자(예: 차량의 VIN)로 의도된 필드를 정의할 수 있지만 문서의 진정한 고유 식별자는 문서 삽입하기에 설명된 대로 `id` 메타데이터 필드입니다. 이러한 이유로 `id` 필드를 사용하여 테이블 간의 관계를 만들 수 있습니다.

문서 `id` 필드는 커밋된 뷰에서만 직접 액세스할 수 있지만, `BY` 절을 사용하여 기본 사용자 뷰로 프로젝션할 수도 있습니다. 예를 들어 다음 쿼리와 해당 결과를 참조하세요.

```
SELECT r_id, r.VIN, r.LicensePlateNumber, r.State, r.City, r.Owners
FROM VehicleRegistration AS r BY r_id
WHERE r_id = '3Qv67yjXEwB9SjmvkuG6Cp'
```

```
{
    r_id: "3Qv67yjXEwB9SjmvkuG6Cp",
    VIN: "1N4AL11D75C109151",
    LicensePlateNumber: "LEWISR261LL",
    State: "WA",
    City: "Seattle",
    Owners: {
        PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSabOs" },
        SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }]
    }
}
```

이 쿼리에서 `r_id`는 `FROM` 키워드를 사용하여 `BY` 절에 선언되는 사용자 정의 별칭입니다. 이 `r_id` 별칭은 쿼리 결과 집합에 있는 각 문서의 `id` 메타데이터 필드에 바인딩됩니다. 이 별칭은 `SELECT` 절에 사용할 수 있으며 사용자 뷰의 쿼리 `WHERE` 절에도 사용할 수 있습니다.

하지만 다른 메타데이터 속성에 액세스하려면 커밋된 뷰를 쿼리해야 합니다.

문서 ID에 조인하기

한 테이블의 문서 `id`를 다른 테이블의 사용자 정의 필드에서 외래 키로 사용하고 있다고 가정해 보겠습니다. 이 `BY` 절을 사용하여 이러한 필드에 있는 두 테이블의 내부 조인 쿼리를 작성할 수 있습니다(이전 항목의 커밋된 뷰와 사용자 뷰 조인과 유사).

다음 예제에서는 `PersonId` 및 문서 `id` 필드에서 각각 `DriversLicense` 및 `Person`라는 두 테이블을 조인하며, 후자에 대해 `BY` 절을 사용합니다.

```
SELECT * FROM DriversLicense AS d INNER JOIN Person AS p BY pid
ON d.PersonId = pid
WHERE pid = '1CWScY2qHYI9G88C2SjvtH'
```

테이블의 문서를 변경하는 방법을 알아보려면 문서 업데이트 및 삭제 섹션으로 이동하세요.

문서 업데이트 및 삭제

Amazon QLDB에서 문서 개정은 고유한 문서 ID로 식별되는 문서 시퀀스의 단일 버전을 나타내는 Amazon Ion 구조입니다. 모든 개정에는 사용자 데이터와 시스템 생성 메타데이터를 포함한 문서의 전체 데이터 세트가 포함됩니다. 각 개정은 문서 ID와 0으로 시작하는 버전 번호의 조합으로 고유하게 식별됩니다.

문서를 업데이트하면 QLDB는 동일한 문서 ID와 증가된 버전 번호를 사용하여 새 개정을 생성합니다. 테이블에서 문서를 삭제하면 문서 수명 주기가 종료됩니다. 즉, 동일한 문서 ID로 문서 개정본을 다시 만들 수 없습니다.

문서 개정하기

예를 들어, 다음 명령문은 새 차량 등록을 삽입하고 등록 도시를 업데이트한 다음 등록을 삭제합니다. 이로 인해 한 문서가 세 번 개정됩니다.

```
INSERT INTO VehicleRegistration
{
    'VIN' : '1HVBBAANXWH544237',
    'LicensePlateNumber' : 'LS477D',
    'State' : 'WA',
    'City' : 'Tacoma',
    'PendingPenaltyTicketAmount' : 42.20,
    'ValidFromDate' : `2011-10-26T`,
    'ValidToDate' : `2023-09-25T`,
    'Owners' : {
        'PrimaryOwner' : { 'PersonId': 'KmA3XPKKFqYCP2zhR3d0Ho' },
        'SecondaryOwners' : []
    }
}
```

> **Note**
>
> Insert 명령문 및 기타 DML 명령문은 영향을 받는 각 문서의 ID를 반환합니다. 다음 항목의 기록 함수에 필요하므로 계속하기 전에 이 ID를 저장해 두세요. 다음 쿼리로도 문서 ID를 찾을 수 있습니다.
>
> ```
> SELECT r_id FROM VehicleRegistration AS r BY r_id
> WHERE r.VIN = '1HVBBAANXWH544237'
> ```

```
UPDATE VehicleRegistration AS r
SET r.City = 'Bellevue'
WHERE r.VIN = '1HVBBAANXWH544237'
```

```
DELETE FROM VehicleRegistration AS r
WHERE r.VIN = '1HVBBAANXWH544237'
```

이러한 DML 명령문의 구문에 대한 추가 예제와 정보는 Amazon QLDB PartiQL 참조의 UPDATE 및 DELETE를 참조하십시오.

문서에 특정 요소를 삽입하거나 제거하려면 FROM 키워드로 시작하는 UPDATE 명령문이나 기타 DML 문을 사용할 수 있습니다. 자세한 내용과 예제는 FROM (INSERT, REMOVE, 또는 SET) 참조를 참조하세요.

문서를 삭제한 후에는 커밋된 뷰나 사용자 뷰에서 문서를 더 이상 쿼리할 수 없습니다. 내장된 기록 함수를을 사용하여 이 문서의 수정 기록을 쿼리하는 방법을 알아보려면 개정 기록 쿼리을 참조하세요.

개정 기록 쿼리

Amazon QLDB는 모든 문서의 전체 기록을 테이블에 저장합니다. 내장된 기록 함수를 쿼리하면 문서 업데이트 및 삭제에서 이전에 삽입, 업데이트 및 삭제한 차량 등록 문서의 세 가지 개정본을 모두 볼 수 있습니다.

주제

- 기록 함수
- 기록 쿼리 예제

기록 함수

QLDB의 기록 함수는 테이블의 시스템 정의 뷰에서 개정 내용을 반환하는 PartiQL 확장입니다. 따라서 커밋된 뷰와 동일한 스키마에 데이터와 관련 메타데이터가 모두 포함됩니다.

구문

```
SELECT * FROM history( table_name | 'table_id' [, `start-time` [, `end-time` ] ] ) AS h
[ WHERE h.metadata.id = 'id' ]
```

인수

table_name | '***table_id***'

테이블 이름 또는 테이블 ID입니다. 테이블 이름은 큰따옴표로 표시하거나 따옴표 없이 표시할 수 있는 PartiQL 식별자입니다. 테이블 ID는 작은따옴표로 묶여야 하는 문자열 리터럴입니다. 테이블 ID 사용에 대한 자세한 내용은 비활성 테이블의 기록 쿼리 섹션을 참조하세요.

`***start-time***`, `***end-time***`

(선택 사항)모든 개정이 활성화된 시간 범위를 지정합니다. 이러한 파라미터는 개정이 트랜잭션에서 저널에 커밋된 시간 범위를 지정하지 않습니다.

시작 및 종료 시간은 백틱(`` `...` ``)으로 표시할 수 있는 Ion 타임스탬프 리터럴입니다. 자세한 내용은 Amazon QLDB에서 PartiQL을 사용한 Ion 쿼리 섹션을 참조하세요.

이 시간 파라미터는 다음과 같은 동작을 합니다.

- 시작 시간과 종료 시간이 모두 포함됩니다. 이는 ISO 8601 날짜 및 시간 형식이어야 하며 협정 세계시(UTC)여야 합니다.
- 시작 시간은 종료 시간보다 작거나 같아야 하며 임의의 과거 날짜가 될 수 있습니다.
- 종료 시간은 현재 UTC 날짜 및 시간보다 작거나 같아야 합니다.
- 시작 시간은 지정하고 종료 시간은 지정하지 않는 경우 쿼리는 종료 시간을 현재 날짜 및 시간으로 기본 설정합니다. 둘 다 지정하지 않으면 쿼리는 전체 기록을 반환합니다.

'***id***'

(선택 사항)수정 기록을 쿼리하려는 문서 ID로, 작은 따옴표로 표시됩니다.

> **Tip**
>
> 가장 좋은 방법은 날짜 범위(시작 시간 및 종료 시간)와 문서 ID(`metadata.id`)를 모두 사용하여 기록 쿼리를 한정하는 것입니다. QLDB에서는 모든 `SELECT` 쿼리가 트랜잭션에서 처리되며 트랜잭션 시간 초과 제한이 적용됩니다.
> 기록 쿼리는 테이블에 만든 인덱스를 사용하지 않습니다. QLDB 기록은 문서 ID로만 인덱싱되며, 이번에는 추가 기록 인덱스를 만들 수 없습니다. 시작 시간과 종료 시간을 포함하는 기록 쿼리는 날짜 범위 한정이라는 이점을 갖습니다.

기록 쿼리 예제

차량 등록 문서의 내역을 쿼리하려면 이전에 문서 업데이트 및 삭제에 저장한 id을 사용하세요. 예를 들어, 다음 기록 쿼리는 2019-06-05T00:00:00Z와 2019-06-05T23:59:59Z 사이에 ADR2Ll1fGsU4Jr4EqTdnQF 활성화된 문서 ID의 모든 개정 내용을 반환합니다.

> **Note**
>
> 시작 및 종료 시간 파라미터는 개정이 트랜잭션에서 저널에 커밋된 시간 범위를 지정하지 않는다는 점을 기억하세요. 예를 들어 2019-06-05T00:00:00Z 이전에 개정이 커밋되었고 해당 시작 시간 이후에도 활성 상태를 유지한 경우 이 예제 쿼리는 결과에 해당 개정 내용을 반환합니다.

id, 시작 시간 및 종료 시간을 적절하게 사용자 고유의 값으로 바꾸세요.

```
SELECT * FROM history(VehicleRegistration, `2019-06-05T00:00:00Z`,
 `2019-06-05T23:59:59Z`) AS h
WHERE h.metadata.id = 'ADR2Ll1fGsU4Jr4EqTdnQF' --replace with your id
```

쿼리 결과는 다음과 비슷한 모습이어야 합니다.

```
{
    blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:14
    },
    hash:{{B2wYwrHKOWsmIBmxUgPRrTx9lv36tMlod2xVvWNiTbo=}},
    data: {
        VIN: "1HVBBAANXWH544237",
        LicensePlateNumber: "LS477D",
        State: "WA",
        City: "Tacoma",
        PendingPenaltyTicketAmount: 42.20,
        ValidFromDate: 2011-10-26T,
        ValidToDate: 2023-09-25T,
        Owners: {
            PrimaryOwner: { PersonId: "KmA3XPKKFqYCP2zhR3d0Ho" },
            SecondaryOwners: []
        }
    },
```

```
    metadata:{
        id:"ADR2Ll1fGsU4Jr4EqTdnQF",
        version:0,
        txTime:2019-06-05T20:53:321d-3Z,
        txId:"HgXAkLjAtV0HQ4lNYdzX60"
    }
},
{
    blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:17
    },
    hash:{{LGSFZ4iEYWZeMwmAqcxxNyT4wbCtuMOmFCj8pEd6Mp0=}},
    data: {
        VIN: "1HVBBAANXWH544237",
        LicensePlateNumber: "LS477D",
        State: "WA",
        PendingPenaltyTicketAmount: 42.20,
        ValidFromDate: 2011-10-26T,
        ValidToDate: 2023-09-25T,
        Owners: {
            PrimaryOwner: { PersonId: "KmA3XPKKFqYCP2zhR3d0Ho" },
            SecondaryOwners: []
        },
        City: "Bellevue"
    },
    metadata:{
        id:"ADR2Ll1fGsU4Jr4EqTdnQF",
        version:1,
        txTime:2019-06-05T21:01:442d-3Z,
        txId:"9cArhIQV5xf5Tf5vtsPwPq"
    }
},
{
    blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:19
    },
    hash:{{7bm5DUwpqJFGrmZpb7h9wAxtvggYLPcXq+LAobi9fDg=}},
    metadata:{
        id:"ADR2Ll1fGsU4Jr4EqTdnQF",
        version:2,
        txTime:2019-06-05T21:03:76d-3Z,
        txId:"9GslbtDtpVHAgYghR5FXbZ"
```

```
    }
}
```

출력에는 각 항목이 수정된 시기와 어떤 트랜잭션에 의해 수정되었는지에 대한 세부 정보를 제공하는 메타데이터 속성이 포함됩니다. 이 데이터에서 다음을 확인할 수 있습니다.

- 이 문서는 시스템이 할당한 `id`인 `ADR2Ll1fGsU4Jr4EqTdnQF`로 고유하게 식별됩니다 이는 Base62로 인코딩된 문자열로 표시되는 UUID입니다.
- `INSERT` 명령문을 사용하면 문서의 초기 개정본(버전 `0`)이 만들어집니다.
- 이후 업데이트될 때마다 동일한 문서 `id`와 증가된 버전 번호를 사용하여 새 개정을 생성합니다.
- 이 `txId` 필드는 각 개정을 커밋한 트랜잭션을 나타내며 `txTime`은 각 개정이 커밋된 시기를 보여줍니다.
- `DELETE` 명령문은 문서의 새로운 최종 개정본을 생성합니다. 이 최종 개정본에는 메타데이터만 있습니다.

개정 버전을 영구 삭제하는 방법을 알아보려면 문서 개정본 수정하기으로 이동하세요.

문서 개정본 수정하기

Amazon QLDB에서 `DELETE` 문은 문서를 삭제된 것으로 표시하는 새 개정본을 생성하여 논리적으로만 문서를 삭제합니다. QLDB는 테이블 기록에서 비활성 문서 개정을 영구적으로 삭제할 수 있는 데이터 수정 작업도 지원합니다.

> **Note**
>
> 2021년 7월 22일 이전에 생성된 원장은 현재 수정할 수 없습니다. Amazon QLDB 콘솔에서 원장 생성 시간을 볼 수 있습니다.

수정 작업을 수행하면 지정된 정 버전의 사용자 데이터만 삭제되고 저널 시퀀스와 문서 메타데이터는 변경되지 않은 상태로 둡니다. 이렇게 하면 원장의 전체 데이터 무결성이 유지됩니다.

QLDB에서 데이터 수정을 시작하기 전에 먼저 Amazon QLDB PartiQL 참조에서 수정 고려 사항 및 제한를 검토해야 합니다.

주제

저장된 프로시저로 수정

REDACT_REVISION 저장 프로시저를 사용하여 원장에서 비활성 상태인 개별 개정본을 영구적으로 삭제할 수 있습니다. 이 저장 프로시저는 인덱싱된 저장소와 저널 저장소 모두에서 지정된 개정본의 모든 사용자 데이터를 삭제합니다. 하지만 저널 시퀀스와 문서 ID 및 해시를 포함한 문서 메타데이터는 변경되지 않습니다. 이 작업은 되돌릴 수 없습니다.

지정된 문서 수정본은 기록에서 비활성 상태여야 합니다. 문서의 최신 활성 수정본은 수정할 수 없습니다.

여러 수정본을 수정하려면 각 수정본마다 한 번씩 저장 프로시저를 실행해야 합니다. 트랜잭션당 하나의 수정본을 수정할 수 있습니다.

구문

```
EXEC REDACT_REVISION `block-address`, 'table-id', 'document-id'
```

인수

`block-address`

수정할 문서의 수정본의 저널 블록 위치입니다. 주소는 strandId 및 sequenceNo라는 두 개의 필드로 구성된 Amazon Ion 구조입니다.

이 값은 백틱으로 표시되는 Ion 리터럴 값입니다. 예:

```
`{strandId:"JdxjkR9bSYB5jMHWcI464T", sequenceNo:17}`
```

'table-id'

수정하고자 하는 문서 개정본이 있는 테이블의 고유 ID로, 작은 따옴표로 표시됩니다.

'*document-id*'

수정할 개정본의 고유한 문서 ID로, 작은 따옴표로 표시됩니다.

수정 완료 여부 확인

저장된 프로시저를 실행하여 수정 요청을 제출하면 QLDB는 데이터 수정을 비동기적으로 처리합니다. 완료되면 개정의 사용자 데이터(data 구조로 표시됨)가 영구적으로 제거됩니다. 수정 요청이 완료되었는지 확인하려면 다음 중 하나를 사용할 수 있습니다.

- 저널 내보내기
- 저널 스트림
- GetBlock API 작업
- GetRevision API 작업
- 기록 함수— 참고: 저널에서 편집을 완료한 후 기록 쿼리에 편집 결과가 표시되기까지 다소 시간이 걸릴 수 있습니다. 비동기 편집이 완료되면 일부 개정 사항이 다른 개정보다 먼저 편집되는 것을 볼 수 있지만 기록 쿼리에는 결국 완료된 결과가 표시됩니다.

개정본 수정이 완료되면 개정본의 data 구조가 새 dataHash 필드로 대체됩니다. 이 필드의 값은 다음 예와 같이 제거된 data 구조의 Ion 해시입니다. 따라서 원장은 전반적인 데이터 무결성을 유지하고 기존 검증 API 작업을 통해 암호화 방식으로 검증 가능한 상태를 유지합니다. 검증에 대한 자세한 정보는 Amazon QLDB에서의 데이터 확인을 참조하세요.

수정 예제

이전에 개정 기록 쿼리에서 검토한 차량 등록 문서를 생각해 보세요. 두 번째 개정(version:1)을 삭제하려 한다고 가정합니다. 다음 쿼리 예제는 수정 전의 이 개정본을 보여줍니다. 쿼리 결과에서, 수정될 data 구조는 *### ####*로 강조 표시됩니다.

```
SELECT * FROM history(VehicleRegistration) AS h
WHERE h.metadata.id = 'ADR2Ll1fGsU4Jr4EqTdnQF' --replace with your id
AND h.metadata.version = 1
```

```
{
    blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:17
    },
```

```
    hash:{{LGSFZ4iEYWZeMwmAqcxxNyT4wbCtuMOmFCj8pEd6Mp0=}},
    data: {
        VIN: "1HVBBAANXWH544237",
        LicensePlateNumber: "LS477D",
        State: "WA",
        PendingPenaltyTicketAmount: 42.20,
        ValidFromDate: 2011-10-26T,
        ValidToDate: 2023-09-25T,
        Owners: {
            PrimaryOwner: { PersonId: "KmA3XPKKFqYCP2zhR3d0Ho" },
            SecondaryOwners: []
        },
        City: "Bellevue"
    },
    metadata:{
        id:"ADR2Ll1fGsU4Jr4EqTdnQF",
        version:1,
        txTime:2019-06-05T21:01:442d-3Z,
        txId:"9cArhIQV5xf5Tf5vtsPwPq"
    }
}
```

이 값을 REDACT_REVISION 저장 프로시저로 전달해야 하므로 쿼리 결과에서 blockAddress를 기록해 두세요. 그런 다음 다음과 같이 시스템 카탈로그를 쿼리하여 VehicleRegistration 테이블의 고유 ID를 찾습니다.

```
SELECT tableId FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

이 테이블 ID를 문서 ID 및 블록 주소와 함께 사용하여 REDACT_REVISION를 실행합니다. 테이블 ID와 문서 ID는 작은따옴표로 묶어야 하는 문자열 리터럴이고, 블록 주소는 백틱으로 묶인 Ion 리터럴입니다. 이 인수는 필요에 따라 사용자 고유의 값으로 바꿉니다.

```
EXEC REDACT_REVISION `{strandId:"JdxjkR9bSYB5jMHWcI464T", sequenceNo:17}`,
 '5PLf9SXwndd63lPaSIa0O6', 'ADR2Ll1fGsU4Jr4EqTdnQF'
```

> **Tip**
>
> QLDB 콘솔 또는 QLDB 쉘을 사용하여 테이블 ID 또는 문서 ID(또는 문자열 리터럴 값)를 쿼리할 때 반환된 값은 큰따옴표로 묶입니다. 하지만 REDACT_REVISION 저장 프로시저의 테이블 ID 및 문서 ID 인수를 지정할 때는 값을 작은 따옴표로 묶어야 합니다.

이는 명령문을 PartiQL 형식으로 작성하지만 QLDB는 결과를 Amazon Ion 형식으로 반환하기 때문입니다. QLDB에서 PartiQL의 구문 및 시맨틱에 대한 자세한 내용은 PartiQL을 사용하여 Ion 쿼리하기 섹션을 참조하세요.

유효한 수정 요청은 다음과 같이 수정 중인 문서 개정본을 나타내는 Ion 구조를 반환합니다.

```
{
  blockAddress: {
    strandId: "JdxjkR9bSYB5jMHWcI464T",
    sequenceNo: 17
  },
  tableId: "5PLf9SXwndd63lPaSIa0O6",
  documentId: "ADR2Ll1fGsU4Jr4EqTdnQF",
  version: 1
}
```

이 저장된 프로시저를 실행하면 QLDB는 수정 요청을 비동기적으로 처리합니다. 수정이 완료되면 `data` 구조가 영구적으로 제거되고 새 *`dataHash`* 필드로 대체됩니다. 이 필드의 값은 다음과 같이 제거된 `data` 구조의 Ion 해시입니다.

> **Note**
>
> 이 `dataHash` 예제는 정보 제공용으로만 제공되며 실제 계산된 해시 값이 아닙니다.

```
{
    blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:17
    },
    hash:{{LGSFZ4iEYWZeMwmAqcxxNyT4wbCtuMOmFCj8pEd6Mp0=}},
    dataHash: {{s83jd7sfhsdfhksj7hskjdfjfpIPP/DP2hvionas2d4=}},
    metadata:{
        id:"ADR2Ll1fGsU4Jr4EqTdnQF",
        version:1,
        txTime:2019-06-05T21:01:442d-3Z,
        txId:"9cArhIQV5xf5Tf5vtsPwPq"
    }
}
```

활성 개정본 삭제 및 수정

활성 문서 개정본(즉, 각 문서의 삭제되지 않은 최신 개정)은 데이터 수정 대상이 아닙니다. 활성 개정본을 수정하려면 먼저 이를 업데이트하거나 삭제해야 합니다. 이렇게 하면 이전에 활성 개정본이었던 것이 기록으로 이동하고 수정이 가능해집니다.

사용 사례에서 전체 문서를 삭제된 것으로 표시해야 하는 경우에는 먼저 DELETE 명령문을 사용합니다. 예를 들어, 다음 명령문은 VIN이 1HVBBAANXWH544237인 VehicleRegistration 문서를 논리적으로 삭제합니다.

```
DELETE FROM VehicleRegistration AS r
WHERE r.VIN = '1HVBBAANXWH544237'
```

그런 다음 앞에서 설명한 대로 이 삭제 전에 이전 개정본을 수정합니다. 필요한 경우 이전 개정본을 개별적으로 수정할 수도 있습니다.

사용 사례에 따라 문서를 활성 상태로 유지해야 하는 경우 먼저 UPDATE 또는 FROM 문을 사용하여 수정하려는 필드를 모호하게 하거나 제거해야 합니다. 이 프로세스는 다음 섹션에서 설명합니다.

개정본 내 특정 필드 수정하기

QLDB는 문서 개정 내 특정 필드의 수정을 지원하지 않습니다. 이렇게 하려면 먼저 UPDATE-REMOVE 또는 FROM-REMOVE 문을 사용하여 개정본에서 기존 필드를 제거할 수 있습니다. 예를 들어, 다음 명령문은 VIN이 1HVBBAANXWH544237인 VehicleRegistration 문서에서 LicensePlateNumber 필드를 제거합니다.

```
UPDATE VehicleRegistration AS r
REMOVE r.LicensePlateNumber
WHERE r.VIN = '1HVBBAANXWH544237'
```

그런 다음 앞에서 설명한 대로 이 제거 전에 이전 개정본을 수정합니다. 필요한 경우 현재 제거된 이 필드를 포함하는 이전 개정본을 개별적으로 수정할 수도 있습니다.

쿼리를 최적화하는 방법을 알아보려면 쿼리 성능 최적화으로 이동하세요.

쿼리 성능 최적화

Amazon QLDB는 고성능 온라인 트랜잭션 처리(OLTP) 워크로드의 니즈를 해결하기 위한 것입니다. 즉, QLDB는 SQL과 유사한 쿼리 기능을 지원하지만 특정 쿼리 패턴 세트에 최적화되어 있습니다. 이러

한 쿼리 패턴과 함께 작동하도록 애플리케이션과 해당 데이터 모델을 설계하는 것이 중요합니다. 그렇지 않으면 테이블이 커질수록 쿼리 지연 시간, 트랜잭션 시간 초과, 동시성 충돌 등 심각한 성능 문제가 발생할 수 있습니다.

이 섹션에서는 QLDB의 쿼리 제약 조건에 대해 설명하고 이러한 제약 조건을 고려하여 최적의 쿼리를 작성하기 위한 지침을 제공합니다.

주제

트랜잭션 시간 초과 제한

QLDB에서는 모든 PartiQL 문(모든 SELECT 쿼리 포함)이 트랜잭션에서 처리되며, 트랜잭션 시간 초과 제한이 적용됩니다. 트랜잭션은 커밋되기 전까지 최대 30초 동안 실행될 수 있습니다. 이 제한을 초과하면 QLDB는 트랜잭션에 대한 모든 작업을 거부하고 트랜잭션을 실행한 세션을 삭제합니다. 이 제한은 트랜잭션을 시작하고 커밋하거나 취소하지 않음으로써 서비스 클라이언트의 세션 유출을 방지합니다.

동시성 충돌

QLDB는 낙관적 동시성 제어(OCC)를 사용하여 동시성 제어를 구현합니다. 쿼리가 최적이 아닌 경우 OCC 충돌이 더 많이 발생할 수도 있습니다. OCC에 대한 자세한 내용은 Amazon QLDB 동시성 모델 섹션을 참조하세요.

최적의 쿼리 패턴

인덱싱된 필드 또는 문서 ID를 기준으로 필터링하는 WHERE 조건자 절을 사용하여 명령문을 실행하는 것이 가장 좋습니다. QLDB에서 문서를 효율적으로 검색하려면 인덱싱된 필드에 동등 연산자(= 또는 IN)가 필요합니다.

다음은 사용자 뷰에서 가장 적합한 쿼리 패턴의 예입니다.

```
--Indexed field (VIN) lookup using the = operator
SELECT * FROM VehicleRegistration
```

```
WHERE VIN = '1N4AL11D75C109151'

--Indexed field (VIN) AND non-indexed field (City) lookup
SELECT * FROM VehicleRegistration
WHERE VIN = '1N4AL11D75C109151' AND City = 'Seattle'

--Indexed field (VIN) lookup using the IN operator
SELECT * FROM VehicleRegistration
WHERE VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')

--Document ID (r_id) lookup using the BY clause
SELECT * FROM VehicleRegistration BY r_id
WHERE r_id = '3Qv67yjXEwB9SjmvkuG6Cp'
```

이러한 패턴을 따르지 않는 모든 쿼리는 전체 테이블 스캔을 호출합니다. 테이블 스캔으로 인해 큰 테이블에 대한 쿼리 또는 큰 결과 집합을 반환하는 쿼리의 경우 트랜잭션 제한 시간이 초과될 수 있습니다. 또한 이로 인해 경쟁 트랜잭션과 OCC 충돌이 발생할 수 있습니다.

하이 카디널리티 인덱스

카디널리티 값이 높은 필드를 인덱싱하는 것이 좋습니다. 예를 들어, `VehicleRegistration` 테이블의 `VIN` 및 `LicensePlateNumber` 필드는 고유하도록 인덱싱된 필드입니다.

상태 코드, 주소 주 또는 지방, 우편 번호와 같이 카디널리티가 낮은 필드는 인덱싱하지 마세요. 이러한 필드를 인덱싱하면 쿼리에서 큰 결과 집합이 생성되어 트랜잭션 시간 초과가 발생하거나 의도하지 않은 OCC 충돌이 발생할 가능성이 높습니다.

커밋된 뷰 쿼리

커밋된 뷰에서 실행하는 쿼리는 사용자 뷰 쿼리와 동일한 최적화 가이드라인을 따릅니다. 테이블에 만든 인덱스는 커밋된 뷰의 쿼리에도 사용됩니다.

기록 함수 쿼리

기록 함수 쿼리는 테이블에 생성한 인덱스를 사용하지 않습니다. QLDB 기록은 문서 ID로만 인덱싱되며, 이번에는 추가 기록 인덱스를 만들 수 없습니다.

가장 좋은 방법은 날짜 범위(시작 시간 및 종료 시간)와 문서 ID(`metadata.id`)를 모두 사용하여 기록 쿼리를 한정하는 것입니다. 시작 시간과 종료 시간을 포함하는 기록 쿼리는 날짜 범위 한정이라는 이점을 갖습니다.

내부 조인 쿼리

내부 조인 쿼리의 경우 조인 오른쪽에 있는 테이블에 대해 최소한 하나의 인덱싱된 필드가 포함된 조인 기준을 사용합니다. 조인 인덱스가 없는 경우 조인 쿼리는 여러 테이블 스캔을 호출합니다. 즉, 조인의 왼쪽 테이블에 있는 모든 문서에 대해 쿼리는 오른쪽 테이블을 완전히 스캔합니다. 가장 좋은 방법은 하나 이상의 테이블에 WHERE 동등 조건을 지정하는 것 외에도 조인하는 각 테이블에 대해 인덱싱된 필드를 조인하는 것입니다.

예를 들어 다음 쿼리는 둘 다 인덱싱된 각 VIN 필드의 VehicleRegistration 및 Vehicle 테이블을 조인합니다. 이 쿼리에는 VehicleRegistration.VIN에 동등 조건자도 있습니다.

```
SELECT * FROM VehicleRegistration AS r INNER JOIN Vehicle AS v
ON r.VIN = v.VIN
WHERE r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

조인 쿼리의 조인 기준과 동등 조건자 모두에 대해 카디널리티가 높은 인덱스를 선택하세요.

피해야 하는 쿼리 패턴

다음은 QLDB의 더 큰 테이블에 맞게 확장되지 않는 차선책 명령문의 몇 가지 예입니다. 쿼리로 인해 결국 트랜잭션 제한 시간이 초과되므로, 시간이 지남에 따라 증가하는 테이블에 대해서는 이러한 유형의 쿼리를 사용하지 않는 것이 좋습니다. 테이블에는 크기가 다양한 문서가 포함되어 있기 때문에 인덱싱되지 않은 쿼리에 대한 정확한 제한을 정의하기는 어렵습니다.

```
--No predicate clause
SELECT * FROM Vehicle

--COUNT() is not an optimized function
SELECT COUNT(*) FROM Vehicle

--Low-cardinality predicate
SELECT * FROM Vehicle WHERE Color = 'Silver'

--Inequality (>) does not qualify for indexed lookup
SELECT * FROM Vehicle WHERE "Year" > 2019

--Inequality (LIKE)
SELECT * FROM Vehicle WHERE VIN LIKE '1N4AL%'

--Inequality (BETWEEN)
SELECT SUM(PendingPenaltyTicketAmount) FROM VehicleRegistration
WHERE ValidToDate BETWEEN `2020-01-01T` AND `2020-07-01T`
```

```
--No predicate clause
DELETE FROM Vehicle

--No document id, and no date range for the history() function
SELECT * FROM history(Vehicle)
```

일반적으로 QLDB의 프로덕션 사용 사례에서는 다음과 같은 유형의 쿼리 패턴을 실행하지 않는 것이 좋습니다.

- 온라인 분석 처리(OLAP) 쿼리
- 조건자 절이 없는 탐색적 쿼리
- 쿼리 보고
- 텍스트 검색

대신 분석적인 사용 사례에 최적화된 목적별 데이터베이스 서비스로 데이터를 스트리밍하는 것이 좋습니다. 예를 들어 QLDB 데이터를 Amazon OpenSearch Service로 스트리밍하여 문서에 대한 전체 텍스트 검색 기능을 제공할 수 있습니다. 이 사용 사례를 보여주는 샘플 애플리케이션을 보려면 GitHub 리포지토리 aws-samples/amazon-qldb-streaming-amazon-opensearch-service-sample-python을 참조하세요. QLDB 스트림에 대한 자세한 내용은 Amazon QLDB에서 저널 데이터 스트리밍 섹션을 참조하세요.

모니터링 성능

QLDB 드라이버는 사용된 I/O 사용량 및 타이밍 정보를 명령문의 결과 개체에 제공합니다. 이러한 지표를 사용하여 비효율적인 PartiQL 문을 식별할 수 있습니다. 자세히 알아보려면 PartiQL 문 통계 가져오기 섹션을 참조하세요.

Amazon CloudWatch를 사용하여 데이터 작업에 대한 원장의 성능을 추적할 수도 있습니다. 지정된 `LedgerName` 및 `CommandType`의 `CommandLatency` 지표를 모니터링합니다. 자세한 내용은 아마존을 통한 모니터링 CloudWatch을 참조하십시오. QLDB가 명령을 사용하여 데이터 작업을 관리하는 방법을 알아보려면 드라이버를 사용한 세션 관리 섹션을 참조하세요.

PartiQL 문 통계 가져오기

Amazon QLDB는 보다 효율적인 PartiQL 문을 실행하여 QLDB 사용을 최적화하는 데 도움이 되는 문 실행 통계를 제공합니다. QLDB는 명령문의 결과와 함께 이러한 통계를 반환합니다. 여기에는 소비된 I/O 사용량과 서버 측 처리 시간을 정량화하는 지표가 포함되며, 이를 통해 비효율적인 명령문을 식별할 수 있습니다.

이 기능은 현재 QLDB 콘솔의 PartiQL 편집기, QLDB 쉘 및 지원되는 모든 언어에 대한 최신 버전의 QLDB 드라이버에서 사용할 수 있습니다. 콘솔에서 쿼리 기록에 대한 명령문 통계를 볼 수도 있습니다.

주제

- I/O 사용량
- 타이밍 정보

I/O 사용량

I/O 사용량 지표는 읽기 I/O 요청 수를 설명합니다. 읽기 I/O 요청 수가 예상보다 많으면 인덱스 부족 등 명령문이 최적화되지 않았음을 나타냅니다. 이전 항목인 쿼리 성능 최적화에서 최적의 쿼리 패턴를 검토하는 것이 좋습니다.

> **Note**
>
> 비어 있지 않은 테이블에서 `CREATE INDEX` 명령문을 실행하면 I/O 사용량 지표에는 동기 인덱스 생성 호출에 대한 읽기 요청만 포함됩니다.
> QLDB는 테이블의 기존 문서에 대한 인덱스를 비동기적으로 빌드합니다. 이러한 비동기 읽기 요청은 명령문 결과의 I/O 사용량 지표에 포함되지 않습니다. 비동기 읽기 요청은 별도로 요금이 부과되며 인덱스 빌드가 완료된 후 총 읽기 I/O에 추가됩니다.

QLDB 콘솔 사용

QLDB 콘솔을 사용하여 명령문의 읽기 I/O 사용량을 확인하려면 다음 단계를 수행하세요.

1. https://console.aws.amazon.com/qldb에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 PartiQL 편집기를 선택합니다.
3. 원장 드롭다운 목록에서 원장을 선택합니다.
4. 쿼리 편집기 창에서 선택한 문을 입력한 다음 실행을 선택합니다. 다음은 쿼리 예제입니다.

   ```
   SELECT * FROM testTable WHERE firstName = 'Jim'
   ```

   명령문을 실행하려면 키보드 단축키 (Windows의 경우 Ctrl+Enter, macOS의 경우 Cmd+Return)를 사용할 수도 있습니다. 키보드 단축키에 대한 자세한 내용은 PartiQL 편집기 키보드 바로 가기 섹션을 참조하세요.

5. 쿼리 편집기 창 아래에 쿼리 결과에는 명령문에 의해 수행된 읽기 요청 횟수인 읽기 I/O가 포함됩니다.

다음 단계를 수행하여 쿼리 기록의 읽기 I/O를 볼 수도 있습니다.

1. 탐색 창의 PartiQL 편집기에서 최근 쿼리를 선택합니다.
2. 읽기 I/O 열에는 각 명령문에서 이루어진 읽기 요청 수가 표시됩니다.

QLDB 드라이버 사용

QLDB 드라이버를 사용하여 명령문의 I/O 사용량을 가져오려면 결과의 스트림 커서 또는 버퍼링된 커서의 `getConsumedIOs` 작업을 호출하세요.

다음 코드 예제는 문 결과의 스트림 커서에서 읽기 I/O를 가져오는 방법을 보여줍니다.

Java

```
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;
import software.amazon.qldb.IOUsage;
import software.amazon.qldb.Result;

IonSystem ionSystem = IonSystemBuilder.standard().build();
IonValue ionFirstName = ionSystem.newString("Jim");

driver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM testTable WHERE firstName = ?",
 ionFirstName);

    for (IonValue ionValue : result) {
        // User code here to handle results
    }

    IOUsage ioUsage = result.getConsumedIOs();
    long readIOs = ioUsage.getReadIOs();
});
```

.NET

```
using Amazon.IonDotnet.Builders;
```

```
using Amazon.IonDotnet.Tree;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

// This is one way of creating Ion values. We can also use a ValueFactory.
// For more details, see: https://docs.aws.amazon.com/qldb/latest/developerguide/
driver-cookbook-dotnet.html#cookbook-dotnet.ion
IIonValue ionFirstName = IonLoader.Default.Load("Jim");

await driver.Execute(async txn =>
{
    IAsyncResult result = await txn.Execute("SELECT * FROM testTable WHERE firstName
 = ?", ionFirstName);

    // Iterate through stream cursor to accumulate read IOs.
    await foreach (IIonValue ionValue in result)
    {
        // User code here to handle results.
        // Warning: It is bad practice to rely on results within a lambda block,
 unless
        // it is to check the state of a result. This is because lambdas are
 retryable.
    }

    var ioUsage = result.GetConsumedIOs();
    var readIOs = ioUsage?.ReadIOs;
});
```

> **Note**
>
> 동기 코드로 변환하려면 `await` 및 `async` 키워드를 제거하고 `IAsyncResult` 유형을 `IResult`로 변경하세요.

Go

```
import (
    "context"
    "fmt"
    "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)
```

```
driver.Execute(context.Background(), func(txn qldbdriver.Transaction) (interface{},
 error) {
    result, err := txn.Execute("SELECT * FROM testTable WHERE firstName = ?", "Jim")

    if err != nil {
        panic(err)
    }

    for result.Next(txn) {
        // User code here to handle results
    }

    ioUsage := result.GetConsumedIOs()
    readIOs := *ioUsage.GetReadIOs()
    fmt.Println(readIOs)
    return nil,nil
})
```

Node.js

```
import { IOUsage, ResultReadable, TransactionExecutor } from "amazon-qldb-driver-
nodejs";

await driver.executeLambda(async (txn: TransactionExecutor) => {
    const result: ResultReadable = await txn.executeAndStreamResults("SELECT * FROM
 testTable WHERE firstName = ?", "Jim");

    for await (const chunk of result) {
        // User code here to handle results
    }

    const ioUsage: IOUsage = result.getConsumedIOs();
    const readIOs: number = ioUsage.getReadIOs();
});
```

Python

```
def get_read_ios(transaction_executor):
    cursor = transaction_executor.execute_statement("SELECT * FROM testTable WHERE
 firstName = ?", "Jim")

    for row in cursor:
        # User code here to handle results
```

```
        pass

    consumed_ios = cursor.get_consumed_ios()
    read_ios = consumed_ios.get('ReadIOs')

qldb_driver.execute_lambda(lambda txn: get_read_ios(txn))
```

다음 코드 예제는 문 결과의 버퍼된 커서에서 읽기 I/O를 가져오는 방법을 보여줍니다. 그러면 `ExecuteStatement` 및 `FetchPage` 요청의 총 읽기 I/O가 반환됩니다.

Java

```
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;
import software.amazon.qldb.IOUsage;
import software.amazon.qldb.Result;

IonSystem ionSystem = IonSystemBuilder.standard().build();
IonValue ionFirstName = ionSystem.newString("Jim");

Result result = driver.execute(txn -> {
    return txn.execute("SELECT * FROM testTable WHERE firstName = ?", ionFirstName);
});

IOUsage ioUsage = result.getConsumedIOs();
long readIOs = ioUsage.getReadIOs();
```

.NET

```
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

IIonValue ionFirstName = IonLoader.Default.Load("Jim");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM testTable WHERE firstName = ?",
 ionFirstName);
});
```

```
var ioUsage = result.GetConsumedIOs();
var readIOs = ioUsage?.ReadIOs;
```

> **Note**
>
> 동기 코드로 변환하려면 `await` 및 `async` 키워드를 제거하고 `IAsyncResult` 유형을 `IResult`로 변경하세요.

Go

```
import (
    "context"
    "fmt"
    "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)

result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    result, err :=  txn.Execute("SELECT * FROM testTable WHERE firstName = ?",
 "Jim")
    if err != nil {
        return nil, err
    }
    return txn.BufferResult(result)
})

if err != nil {
    panic(err)
}

qldbResult := result.(*qldbdriver.BufferedResult)
ioUsage := qldbResult.GetConsumedIOs()
readIOs := *ioUsage.GetReadIOs()
fmt.Println(readIOs)
```

Node.js

```
import { IOUsage, Result, TransactionExecutor } from "amazon-qldb-driver-nodejs";
```

```
const result: Result = await driver.executeLambda(async (txn: TransactionExecutor)
 => {
    return await txn.execute("SELECT * FROM testTable WHERE firstName = ?", "Jim");
});

const ioUsage: IOUsage = result.getConsumedIOs();
const readIOs: number = ioUsage.getReadIOs();
```

Python

```
cursor = qldb_driver.execute_lambda(
    lambda txn: txn.execute_statement("SELECT * FROM testTable WHERE firstName = ?",
 "Jim"))

consumed_ios = cursor.get_consumed_ios()
read_ios = consumed_ios.get('ReadIOs')
```

> **Note**
>
> 스트림 커서는 결과 집합의 페이지를 매기므로 상태를 추적할 수 있습니다. 따라서 `getConsumedIOs` 및 `getTimingInformation` 연산은 호출한 시점부터 누적된 지표를 반환합니다.
> 버퍼링된 커서는 결과 집합을 메모리에 버퍼링하고 총 누적된 지표를 반환합니다.

타이밍 정보

타이밍 정보 지표는 서버 측 처리 시간을 밀리초 단위로 설명합니다. 서버측 처리 시간은 QLDB가 명령문을 처리하는 데 소비하는 시간으로 정의됩니다. 네트워크 호출 또는 일시 중지에 소요된 시간은 여기에 포함되지 않습니다. 이 메트릭은 QLDB 서비스 측의 처리 시간과 클라이언트 측의 처리 시간을 구분합니다.

QLDB 콘솔 사용

QLDB 콘솔을 사용하여 명령문의 타이밍 정보를 가져오려면 다음 단계를 수행하세요.

1. https://console.aws.amazon.com/qldb에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 PartiQL 편집기를 선택합니다.
3. 원장 드롭다운 목록에서 원장을 선택합니다.

4. 쿼리 편집기 창에서 선택한 문을 입력한 다음 실행을 선택합니다. 다음은 쿼리 예제입니다.

```
SELECT * FROM testTable WHERE firstName = 'Jim'
```

   명령문을 실행하려면 키보드 단축키 (Windows의 경우 Ctrl+Enter, macOS의 경우 Cmd+Return)를 사용할 수도 있습니다. 키보드 단축키에 대한 자세한 내용은 PartiQL 편집기 키보드 바로 가기 섹션을 참조하세요.

5. 쿼리 편집기 창 아래에 쿼리 결과에는 QLDB가 명령문 요청을 수신한 시점과 응답을 보낸 시점 사이의 시간인 서버 측 지연 시간이 포함됩니다. 이는 총 쿼리 기간의 하위 집합입니다.

다음 단계를 수행하여 쿼리 기록의 타이밍 정보를 볼 수도 있습니다.

1. 탐색 창의 PartiQL 편집기에서 최근 쿼리를 선택합니다.
2. 실행 시간(밀리초) 열에는 각 명령문에 대한 이 타이밍 정보가 표시됩니다.

QLDB 드라이버 사용

QLDB 드라이버를 사용하여 명령문의 타이밍 정보를 가져오려면 결과의 스트림 커서 또는 버퍼링된 커서의 `getTimingInformation` 작업을 호출하세요.

다음 코드 예제는 문 결과의 스트림 커서에서 처리 시간을 가져오는 방법을 보여줍니다.

Java

```
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;
import software.amazon.qldb.Result;
import software.amazon.qldb.TimingInformation;

IonSystem ionSystem = IonSystemBuilder.standard().build();
IonValue ionFirstName = ionSystem.newString("Jim");

driver.execute(txn -> {
    Result result = txn.execute("SELECT * FROM testTable WHERE firstName = ?",
 ionFirstName);

    for (IonValue ionValue : result) {
        // User code here to handle results
```

```
    }

    TimingInformation timingInformation = result.getTimingInformation();
    long processingTimeMilliseconds =
 timingInformation.getProcessingTimeMilliseconds();
});
```

.NET

```
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;
using Amazon.QLDB.Driver;
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

IIonValue ionFirstName = IonLoader.Default.Load("Jim");

await driver.Execute(async txn =>
{
    IAsyncResult result = await txn.Execute("SELECT * FROM testTable WHERE firstName
 = ?", ionFirstName);

    // Iterate through stream cursor to accumulate processing time.
    await foreach(IIonValue ionValue in result)
    {
        // User code here to handle results.
        // Warning: It is bad practice to rely on results within a lambda block,
 unless
        // it is to check the state of a result. This is because lambdas are
 retryable.
    }

    var timingInformation = result.GetTimingInformation();
    var processingTimeMilliseconds = timingInformation?.ProcessingTimeMilliseconds;
});
```

> **Note**
>
> 동기 코드로 변환하려면 `await` 및 `async` 키워드를 제거하고 `IAsyncResult` 유형을 `IResult`로 변경하세요.

Go

```
import (
    "context"
    "fmt"
    "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)

driver.Execute(context.Background(), func(txn qldbdriver.Transaction) (interface{},
 error) {
    result, err := txn.Execute("SELECT * FROM testTable WHERE firstName = ?", "Jim")

    if err != nil {
        panic(err)
    }

    for result.Next(txn) {
        // User code here to handle results
    }

    timingInformation := result.GetTimingInformation()
    processingTimeMilliseconds := *timingInformation.GetProcessingTimeMilliseconds()
    fmt.Println(processingTimeMilliseconds)
    return nil, nil
})
```

Node.js

```
import { ResultReadable, TimingInformation, TransactionExecutor } from "amazon-qldb-
driver-nodejs";

await driver.executeLambda(async (txn: TransactionExecutor) => {
    const result: ResultReadable = await txn.executeAndStreamResults("SELECT * FROM
 testTable WHERE firstName = ?", "Jim");

    for await (const chunk of result) {
        // User code here to handle results
    }

    const timingInformation: TimingInformation = result.getTimingInformation();
    const processingTimeMilliseconds: number =
 timingInformation.getProcessingTimeMilliseconds();
});
```

Python

```
def get_processing_time_milliseconds(transaction_executor):
    cursor = transaction_executor.execute_statement("SELECT * FROM testTable WHERE
 firstName = ?", "Jim")

    for row in cursor:
        # User code here to handle results
        pass

    timing_information = cursor.get_timing_information()
    processing_time_milliseconds =
 timing_information.get('ProcessingTimeMilliseconds')

qldb_driver.execute_lambda(lambda txn: get_processing_time_milliseconds(txn))
```

다음 코드 예제는 문 결과의 버퍼된 커서에서 처리 시간을 가져오는 방법을 보여줍니다. 그러면 `ExecuteStatement` 및 `FetchPage` 요청의 총 처리 시간이 반환됩니다.

Java

```
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;
import software.amazon.qldb.Result;
import software.amazon.qldb.TimingInformation;

IonSystem ionSystem = IonSystemBuilder.standard().build();
IonValue ionFirstName = ionSystem.newString("Jim");

Result result = driver.execute(txn -> {
    return txn.execute("SELECT * FROM testTable WHERE firstName = ?", ionFirstName);
});

TimingInformation timingInformation = result.getTimingInformation();
long processingTimeMilliseconds = timingInformation.getProcessingTimeMilliseconds();
```

.NET

```
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;
using Amazon.QLDB.Driver;
```

```
using IAsyncResult = Amazon.QLDB.Driver.IAsyncResult;

IIonValue ionFirstName = IonLoader.Default.Load("Jim");

IAsyncResult result = await driver.Execute(async txn =>
{
    return await txn.Execute("SELECT * FROM testTable WHERE firstName = ?",
 ionFirstName);
});

var timingInformation = result.GetTimingInformation();
var processingTimeMilliseconds = timingInformation?.ProcessingTimeMilliseconds;
```

> **Note**
>
> 동기 코드로 변환하려면 `await` 및 `async` 키워드를 제거하고 `IAsyncResult` 유형을 `IResult`로 변경하세요.

Go

```
import (
    "context"
    "fmt"
    "github.com/awslabs/amazon-qldb-driver-go/v2/qldbdriver"
)

result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    result, err :=  txn.Execute("SELECT * FROM testTable WHERE firstName = ?",
 "Jim")
    if err != nil {
        return nil, err
    }
    return txn.BufferResult(result)
})

if err != nil {
    panic(err)
}

qldbResult := result.(*qldbdriver.BufferedResult)
```

```
timingInformation := qldbResult.GetTimingInformation()
processingTimeMilliseconds := *timingInformation.GetProcessingTimeMilliseconds()
fmt.Println(processingTimeMilliseconds)
```

Node.js

```
import { Result, TimingInformation, TransactionExecutor } from "amazon-qldb-driver-
nodejs";

const result: Result = await driver.executeLambda(async (txn: TransactionExecutor)
 => {
    return await txn.execute("SELECT * FROM testTable WHERE firstName = ?", "Jim");
});

const timingInformation: TimingInformation = result.getTimingInformation();
const processingTimeMilliseconds: number =
 timingInformation.getProcessingTimeMilliseconds();
```

Python

```
cursor = qldb_driver.execute_lambda(
    lambda txn: txn.execute_statement("SELECT * FROM testTable WHERE firstName = ?",
 "Jim"))

timing_information = cursor.get_timing_information()
processing_time_milliseconds = timing_information.get('ProcessingTimeMilliseconds')
```

> **Note**
>
> 스트림 커서는 결과 집합의 페이지를 매기므로 상태를 추적할 수 있습니다. 따라서 `getConsumedIOs` 및 `getTimingInformation` 연산은 호출한 시점부터 누적된 지표를 반환합니다.
> 버퍼링된 커서는 결과 집합을 메모리에 버퍼링하고 총 누적된 지표를 반환합니다.

시스템 카탈로그를 쿼리하는 방법을 알아보려면 시스템 카탈로그 쿼리 섹션을 참조하세요.

시스템 카탈로그 쿼리

Amazon QLDB 원장에 생성하는 각 테이블에는 시스템에서 할당한 고유 ID가 있습니다. 시스템 카탈로그 테이블 `information_schema.user_tables`을 쿼리하여 테이블 ID, 인덱스 목록 및 기타 메타데이터를 찾을 수 있습니다.

시스템에서 할당한 모든 ID는 Base62로 인코딩된 문자열로 각각 표시되는 범용 고유 식별자(UUID)입니다. 자세한 내용은 Amazon QLDB의 고유 ID을 참조하십시오.

다음 예제는 `VehicleRegistration` 테이블의 메타데이터 속성을 반환하는 쿼리 결과를 보여줍니다.

```
SELECT * FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

```
{
    tableId: "5PLf9SXwndd63lPaSIa0O6",
    name: "VehicleRegistration",
    indexes: [
        { indexId: "Djg2nt0yIs2GY0T29Kud1z", expr: "[VIN]", status: "ONLINE" },
        { indexId: "4tPW3fUhaVhDinRgKRLhGU", expr: "[LicensePlateNumber]", status:
 "BUILDING" }
    ],
    status: "ACTIVE"
}
```

테이블 메타데이터 필드

- `tableId` – 테이블의 고유 ID입니다.
- `name` – 테이블 이름입니다.
- `indexes` – 테이블의 인덱스 목록입니다.
  - `indexId` – 인덱스의 고유 ID입니다.
  - `expr` – 인덱싱된 문서 경로입니다. 이 필드는 `[fieldName]` 형식의 문자열입니다.
  - `status` - 인덱스의 현재 상태(`BUILDING`, `FINALIZING`, `ONLINE`, `FAILED`, 또는`DELETING`)입니다. QLDB는 상태가 `ONLINE`이면 쿼리에 인덱스를 사용하지 않습니다.
  - `message` – 인덱스에 `FAILED` 상태가 있는 이유를 설명하는 오류 메시지입니다. 이 필드는 실패한 인덱스에만 포함됩니다.

- `status` – 테이블의 현재 상태(`ACTIVE` 또는`INACTIVE`)입니다. 테이블을 `DROP` 할 경우 `INACTIVE`이 됩니다.

`DROP TABLE` 및 `UNDROP TABLE` 문을 사용하여 테이블을 관리하는 방법을 알아보려면 테이블 관리으로 이동하세요.

테이블 관리

이 단원에서는 Amazon QLDB에서 `DROP TABLE` 및 `UNDROP TABLE` 문을 사용하여 테이블을 관리하는 방법을 설명합니다. 또한 테이블을 생성할 때 태그를 지정하는 방법도 설명합니다. 생성할 수 있는 활성 테이블 수와 총 테이블 수에 대한 할당량은 Amazon QLDB 할당량 및 제한에 정의되어 있습니다.

주제

- 테이블 생성 시 태그 지정
- 테이블 삭제
- 비활성 테이블의 기록 쿼리
- 테이블 재활성화

테이블 생성 시 태그 지정

> **Note**
>
> 테이블 생성 시 태그 지정은 현재 `STANDARD` 권한 모드의 원장에만 지원됩니다.

테이블 리소스에 태그를 지정할 수 있습니다. 기존 테이블의 태그를 관리하려면 AWS Management Console 또는 API 작업 `TagResource`, `UntagResource`, `ListTagsForResource`을 사용하세요. 자세한 내용은 Amazon QLDB 리소스 태그 지정을 참조하십시오.

테이블을 생성하는 동안 QLDB 콘솔을 사용하거나 `CREATE TABLE` PartiQL 문에 테이블 태그를 지정하여 테이블 태그를 정의할 수도 있습니다. 다음 예제에서는 `environment=production` 태그가 있는 `Vehicle`이라는 테이블을 생성합니다.

```
CREATE TABLE Vehicle WITH (aws_tags = `{'environment': 'production'}`)
```

리소스를 생성하는 동안 태그를 지정하면 리소스 생성 후 사용자 지정 태그 지정 스크립트를 실행할 필요가 없습니다. 테이블에 태그가 지정된 후 해당 태그를 기반으로 테이블에 대한 액세스를 제어할 수 있습니다. 예를 들어 특정 태그가 있는 테이블에만 전체 액세스 권한을 부여할 수 있습니다. JSON 정책의 예는 테이블 태그를 기반으로 하는 모든 작업에 대한 전체 액세스 권한 섹션을 참조하세요.

테이블 삭제

테이블을 삭제하려면 기본 DROP TABLE 명령문을 사용하세요. QLDB에 테이블을 삭제하면 해당 테이블이 비활성화됩니다.

예를 들어 다음 명령문은 `VehicleRegistration` 테이블을 비활성화합니다.

```
DROP TABLE VehicleRegistration
```

`DROP TABLE` 명령문은 테이블의 시스템 할당 ID를 반환합니다. 이제 `VehicleRegistration`의 상태는 시스템 카탈로그 테이블 information_schema.user_tables에서 `INACTIVE`여야 합니다.

```
SELECT status FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

비활성 테이블의 기록 쿼리

테이블 이름 외에도 테이블 ID를 첫 번째 입력 인수로 사용하여 QLDB 기록 함수를 쿼리할 수도 있습니다. 비활성 테이블의 기록을 쿼리하려면 테이블 ID를 사용해야 합니다. 테이블이 비활성화되면 더 이상 테이블 이름으로 테이블 기록을 쿼리할 수 없습니다.

먼저 시스템 카탈로그 테이블을 쿼리하여 테이블 ID를 찾으세요. 예를 들어 다음 쿼리는 `VehicleRegistration` 테이블의 `tableId`를 반환합니다.

```
SELECT tableId FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

그러면 이 ID를 사용하여 개정 기록 쿼리에서 동일한 기록 쿼리를 실행할 수 있습니다. 다음은 테이블 ID `5PLf9SXwndd63lPaSIa0O6`에서 문서 ID `ADR2Ll1fGsU4Jr4EqTdnQF` 기록을 쿼리하는 예제입니다. 테이블 ID는 작은 따옴표로 묶여야 하는 문자열 리터럴입니다.

```
--replace both the table and document IDs with your values
SELECT * FROM history('5PLf9SXwndd63lPaSIa0O6', `2000T`, `2019-06-05T23:59:59Z`) AS h
WHERE h.metadata.id = 'ADR2Ll1fGsU4Jr4EqTdnQF'
```

테이블 재활성화

QLDB에서 테이블을 비활성화한 후 UNDROP TABLE 명령문을 사용하여 테이블을 다시 활성화할 수 있습니다.

먼저 `information_schema.user_tables`에서 테이블 ID를 찾으세요. 예를 들어 다음 쿼리는 `VehicleRegistration` 테이블의 `tableId`를 반환합니다. 상태는 `INACTIVE`과 같아야 합니다.

```
SELECT tableId FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

그런 다음 이 ID를 사용하여 테이블을 다시 활성화하세요. 다음은 테이블 ID `5PLf9SXwndd63lPaSIa0O6`의 삭제를 취소하는 예제입니다. 이 경우 테이블 ID는 큰따옴표로 묶는 고유 식별자입니다.

```
UNDROP TABLE "5PLf9SXwndd63lPaSIa0O6"
```

이제 `VehicleRegistration`의 상태가 `ACTIVE`여야 합니다.

인덱스를 생성하고, 설명하고, 삭제하는 방법을 알아보려면 인덱스 관리으로 이동하세요.

인덱스 관리

이 섹션에서는 Amazon QLDB에서 인덱스를 생성, 설명 및 삭제하는 방법을 설명합니다. 생성할 수 있는 테이블당 인덱스 수에 대한 할당량은 Amazon QLDB 할당량 및 제한에 정의되어 있습니다.

주제

- 인덱스 생성
- 인덱스 설명
- 인덱스 삭제
- 일반적인 오류

인덱스 생성

테이블 및 인덱스 생성에도 설명된 대로 CREATE INDEX 문을 사용하여 다음과 같이 지정된 최상위 필드의 인덱스를 테이블에 생성할 수 있습니다. 테이블 이름과 인덱싱된 필드 이름은 모두 대소문자를 구분합니다.

```
CREATE INDEX ON VehicleRegistration (VIN)
```

```
CREATE INDEX ON VehicleRegistration (LicensePlateNumber)
```

테이블에 생성하는 각 인덱스에는 시스템에서 할당한 고유 ID가 있습니다. 이 인덱스 ID를 찾으려면 다음 인덱스 설명 섹션을 참조하세요.

> **Important**
>
> QLDB는 문서를 효율적으로 조회하기 위한 인덱스가 필요합니다. 인덱스가 없으면 QLDB는 문서를 읽을 때 전체 테이블 스캔을 수행해야 합니다. 이로 인해 동시성 충돌 및 트랜잭션 시간 초과를 포함하여 대규모 테이블에서 성능 문제가 발생할 수 있습니다.
> 인덱싱된 필드 또는 문서 ID(예: = 또는 IN)에서 동등 연산자를 사용하여 WHERE 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 자세한 내용은 쿼리 성능 최적화을 참조하십시오.

인덱스를 생성할 때는 다음 제약 조건에 유의하세요.

- 인덱스는 단일 최상위 필드에만 생성할 수 있습니다. 복합, 중첩, 고유 및 함수 기반 인덱스는 지원되지 않습니다.
- `list` 및 `struct`를 비롯한 모든 Ion 데이터 유형에 대해 인덱스를 생성할 수 있습니다. 그러나 Ion 유형에 관계없이 전체 Ion 값이 같아야만 인덱스 조회를 수행할 수 있습니다. 예를 들어 `list` 형식을 인덱스로 사용하는 경우 목록 내에서 한 항목씩 인덱스 검색을 수행할 수 없습니다.
- 동등 조건자(예: `WHERE indexedField = 123` 또는 `WHERE indexedField IN (456, 789)`)를 사용할 때만 쿼리 성능이 향상됩니다.

  QLDB는 쿼리 조건자의 부등을 인정하지 않습니다. 따라서 범위 필터링된 스캔은 구현되지 않습니다.
- 인덱싱된 필드 이름은 대소문자를 구분하며 최대 128자입니다.
- QLDB에서의 인덱스 생성은 비동기적으로 이루어집니다. 비어 있지 않은 테이블에서 인덱스 빌드를 완료하는 데 걸리는 시간은 테이블 크기에 따라 다릅니다. 자세한 내용은 인덱스 관리을 참조하십시오.

인덱스 설명

QLDB에서의 인덱스 생성은 비동기적으로 이루어집니다. 비어 있지 않은 테이블에서 인덱스 빌드를 완료하는 데 걸리는 시간은 테이블 크기에 따라 다릅니다. 인덱스 빌드 상태를 확인하려면 시스템 카탈로그 테이블 information_schema.user_tables을 쿼리하면 됩니다.

예를 들어, 다음 명령문은 시스템 카탈로그에서 VehicleRegistration 테이블의 모든 인덱스를 쿼리합니다.

```
SELECT VALUE indexes
FROM information_schema.user_tables info, info.indexes indexes
WHERE info.name = 'VehicleRegistration'
```

```
{
    indexId: "Djg2nt0yIs2GY0T29Kud1z",
    expr: "[VIN]",
    status: "ONLINE"
},
{
    indexId: "4tPW3fUhaVhDinRgKRLhGU",
    expr: "[LicensePlateNumber]",
    status: "FAILED",
    message: "aws.ledger.errors.InvalidEntityError: Document contains multiple values
 for indexed field: LicensePlateNumber"
}
```

인덱스 필드

- indexId – 인덱스의 고유 ID입니다.
- expr – 인덱싱된 문서 경로입니다. 이 필드는 [fieldName] 형식의 문자열입니다.
- status – 인덱스의 현재 상태입니다. 인덱스의 상태는 다음 값 중 하나일 수 있습니다.
  - BUILDING – 테이블의 인덱스를 적극적으로 빌드하고 있습니다.
  - FINALIZING – 인덱스 빌드를 완료하고 활성화하여 사용하기 시작합니다.
  - ONLINE – 활성 상태이며 쿼리에 사용할 준비가 되었습니다. QLDB는 상태가 온라인 상태가 될 때까지 쿼리에 인덱스를 사용하지 않습니다.
  - FAILED – 복구할 수 없는 오류로 인해 인덱스를 빌드할 수 없습니다. 이 상태의 인덱스는 여전히 테이블당 인덱스 할당량에 포함됩니다. 자세한 내용은 일반적인 오류을 참조하십시오.
  - DELETING – 사용자가 인덱스를 삭제한 후 인덱스를 적극적으로 삭제하고 있습니다.

- `message` – 인덱스에 `FAILED` 상태가 있는 이유를 설명하는 오류 메시지입니다. 이 필드는 실패한 인덱스에만 포함됩니다.

콘솔 사용

AWS Management Console를 사용하여 인덱스 상태를 확인할 수도 있습니다.

인덱스의 상태를 확인하려면(콘솔)

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/qldb에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 원장을 선택합니다.
3. 원장 목록에서 관리하려는 인덱스가 있는 원장 이름을 선택합니다.
4. 원장 세부정보 페이지의 테이블 탭에서 인덱스를 확인할 테이블 이름을 선택합니다.
5. 테이블 세부정보 페이지에서 인덱싱된 필드 카드를 찾습니다. 인덱스 상태 열은 테이블의 각 인덱스의 현재 상태를 표시합니다.

인덱스 삭제

DROP INDEX 명령문을 사용하여 인덱스를 삭제합니다. 인덱스를 삭제하면 테이블에서 영구적으로 삭제됩니다.

먼저 인덱스 ID를 `information_schema.user_tables`에서 찾으세요. 예를 들어, 다음 쿼리는 `VehicleRegistration` 테이블에서 인덱싱된 `LicensePlateNumber` 필드의 `indexId`을 반환합니다.

```
SELECT indexes.indexId
FROM information_schema.user_tables info, info.indexes indexes
WHERE info.name = 'VehicleRegistration' and indexes.expr = '[LicensePlateNumber]'
```

그런 다음 이 ID를 사용하여 인덱스를 삭제하세요. 다음은 인덱스 ID `4tPW3fUhaVhDinRgKRLhGU`를 삭제하는 예제입니다. 인덱스 ID는 큰따옴표로 묶어야 하는 고유 식별자입니다.

```
DROP INDEX "4tPW3fUhaVhDinRgKRLhGU" ON VehicleRegistration WITH (purge = true)
```

> **Note**
>
> 이 `WITH (purge = true)` 절은 모든 `DROP INDEX` 명령문에 필수이며 `true`는 현재 지원되는 유일한 값입니다.
> `purge` 키워드는 대소문자를 구분하며 모두 소문자여야 합니다.

콘솔 사용

AWS Management Console를 사용하여 인덱스를 삭제할 수도 있습니다.

인덱스를 삭제하려면(콘솔)

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/qldb에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 원장을 선택합니다.
3. 원장 목록에서 관리하려는 인덱스가 있는 원장 이름을 선택합니다.
4. 원장 세부정보 페이지의 테이블 탭에서 인덱스를 삭제할 테이블 이름을 선택합니다.
5. 테이블 세부정보 페이지에서 인덱싱된 필드 카드를 찾습니다. 삭제하려는 인덱스를 선택한 다음 인덱스 삭제를 선택합니다.

일반적인 오류

이 섹션에서는 인덱스를 만들 때 발생할 수 있는 일반적인 오류를 설명하고 가능한 해결 방법을 제안합니다.

> **Note**
>
> `FAILED` 상태의 인덱스는 여전히 테이블당 인덱스 할당량에 포함됩니다. 또한 실패한 인덱스는 테이블에서 인덱스 생성 실패를 초래한 문서를 수정하거나 삭제하는 것을 방지합니다. 할당량에서 인덱스를 제거하려면 명시적으로 인덱스를 제거해야 합니다.

문서에 인덱싱된 필드 ***`fieldName`***에 대한 여러 값이 포함되어 있습니다.

테이블에 동일한 필드에 대해 여러 값(즉, 중복된 필드 이름)이 있는 문서가 포함되어 있기 때문에 QLDB는 지정된 필드 이름에 대한 인덱스를 생성할 수 없습니다.

먼저 실패한 인덱스를 삭제해야 합니다. 그런 다음 인덱스 생성을 다시 시도하기 전에 테이블의 모든 문서에 각 필드 이름에 대해 하나의 값만 있는지 확인하세요. 중복이 없는 다른 필드에 대한 인덱스를 생성할 수도 있습니다.

또한 테이블에 이미 인덱싱된 필드에 대해 여러 값이 포함된 문서를 삽입하려고 할 경우 QLDB는 이 오류를 반환합니다.

인덱스 제한 초과: 테이블 ***TableName##*** 이미 ***n#***의 인덱스가 있으며 더 만들 수 없습니다.

QLDB는 실패한 인덱스를 포함하여 테이블당 인덱스를 5개로 제한합니다. 새 인덱스를 생성하기 전에 기존 인덱스를 삭제해야 합니다.

식별자가 ***indexId***인 정의된 인덱스가 없습니다.

지정된 테이블과 인덱스 ID 조합에 대해 존재하지 않는 인덱스를 삭제하려고 했습니다. 기존 인덱스를 확인하는 방법을 알아보려면 인덱스 설명을 참조하세요.

Amazon QLDB의 고유 ID

이 섹션에서는 Amazon QLDB에 있는 시스템 할당 고유 식별자의 속성 및 사용 지침을 설명합니다. 또한 QLDB 고유 ID의 몇 가지 예를 제공합니다.

주제

속성

QLDB의 모든 시스템 할당 ID는 범용 고유 식별자(UUID)입니다. 각 ID에는 다음 속성이 있습니다.

- 128비트 UUID 숫자
- Base62로 인코딩된 텍스트로 표시됨
- 22자의 고정 길이 영숫자 문자열(예:`3Qv67yjXEwB9SjmvkuG6Cp`)

사용량

애플리케이션에서 QLDB 고유 ID를 사용하는 경우 다음 지침에 유의하세요.

해야 할 일

- ID를 문자열로 취급합니다.

하지 않아야 할 일

- 문자열을 디코딩하려고 시도합니다.
- 문자열에 의미론적 의미를 부여합니다(예: 시간 구성 요소 도출).
- 문자열을 의미론적 순서로 정렬합니다.

예시

다음 속성은 QLDB의 고유 ID의 몇 가지 예입니다.

- 문서 ID
- 인덱스 ID
- 스트랜드 ID
- 테이블 ID
- 트랜잭션 ID

Amazon QLDB 동시성 모델

Amazon QLDB는 고성능 온라인 트랜잭션 처리(OLTP) 워크로드의 니즈를 해결하기 위한 것입니다. QLDB는 SQL과 유사한 쿼리 기능을 지원하며 전체 ACID 트랜잭션을 제공합니다. 또한 QLDB 데이터 항목은 문서이므로 스키마 유연성과 직관적인 데이터 모델링을 제공합니다. 저널을 중심으로 하면 QLDB를 사용하여 데이터에 대한 모든 변경 사항의 완전하고 검증 가능한 기록에 액세스하고, 필요에 따라 다른 데이터 서비스에 일관된 트랜잭션을 스트리밍할 수 있습니다.

주제

낙관적 동시성 제어

QLDB에서는 OCC(낙관적 동시성 제어)를 사용하여 동시성 제어를 구현합니다. OCC는 여러 트랜잭션이 서로 간섭하지 않고 자주 완료될 수 있다는 원칙에 따라 작동합니다.

OCC를 사용하면 QLDB의 트랜잭션이 데이터베이스 리소스에 대한 잠금을 획득하지 않고, 완전한 직렬화가 가능한 격리 상태로 작동합니다. QLDB는 동시 트랜잭션을 순차적으로 실행하므로 해당 트랜잭션이 순차적으로 시작된 것과 동일한 결과를 생성합니다.

각 트랜잭션은 체결하기 전에 검증 검사를 수행하여 다른 체결된 트랜잭션이 액세스 중인 데이터를 수정하지는 않았는지 확인합니다. 이 검사에서 충돌하는 수정 사항이 발견되거나 데이터 상태가 변경되면 체결된 트랜잭션이 거부됩니다. 하지만 트랜잭션을 다시 시작할 수 있습니다.

트랜잭션이 QLDB에 기록하 때 OCC 모델의 검증 검사는 QLDB 자체에서 구현됩니다. OCC의 검증 단계에서 오류가 발생하여 저널에 트랜잭션을 기록할 수 없는 경우 QLDB는 를 애플리케이션 계층에 `OccConflictException`를 반환합니다. 애플리케이션 소프트웨어는 트랜잭션이 다시 시작되도록 할 책임이 있습니다. 애플리케이션은 거부된 트랜잭션을 중단한 다음 처음부터 전체 트랜잭션을 다시 시도해야 합니다.

QLDB 드라이버가 OCC 충돌 및 기타 일시적 예외를 처리하고 재시도하는 방법을 알아보려면 Amazon QLDB의 드라이버를 사용한 재시도 정책에 대한 이해 섹션을 참조하세요.

인덱스를 사용하여 전체 테이블 스캔 방지

QLDB에서는 모든 PartiQL 문(모든 SELECT 쿼리 포함)이 트랜잭션에서 처리되며, 트랜잭션 시간 초과 제한이 적용됩니다.

인덱싱된 필드 또는 문서 ID를 기준으로 필터링하는 WHERE 조건자 절을 사용하여 명령문을 실행하는 것이 가장 좋습니다. QLDB에서 문서를 효율적으로 조회하려면 인덱싱된 필드에 같음 연산자가 필요합니다(예: `WHERE indexedField = 123` 또는`WHERE indexedField IN (456, 789)`).

이 인덱스 조회가 없으면 QLDB는 문서를 읽을 때 전체 표 스캔을 수행해야 합니다. 이로 인해 쿼리 지연 및 트랜잭션 시간 초과가 발생할 수 있으며 경쟁 트랜잭션과 OCC가 충돌할 가능성도 높아집니다.

VIN 필드에만 인덱스가 있는 Vehicle라는 명칭의 표를 예로 들어 보겠습니다. 여기에는 다음 문서가 포함됩니다 .

VIN	Make	모델	색상
"1N4AL11D 75C109151"	"Audi"	"A5"	"Silver"
"KM8SRDHF 6EU074761"	"Tesla"	"Model S"	"Blue"
"3HGGK5G5 3FM761765"	"Ducati"	"Monster 1200"	"Yellow"
"1HVBBAAN XWH544237"	"Ford"	"F 150"	"Black"
"1C4RJFAG 0FC625797"	"Mercedes"	"CLK 350"	"White"

Alice와 Bob이라는 두 명의 동시 사용자가 원장에 있는 같은 표를 사용하고 있습니다. 그들은 다음과 같이 서로 다른 두 문서를 업데이트하려고 합니다.

Alice:

```
UPDATE Vehicle AS v
SET v.Color = 'Blue'
WHERE v.VIN = '1N4AL11D75C109151'
```

Bob:

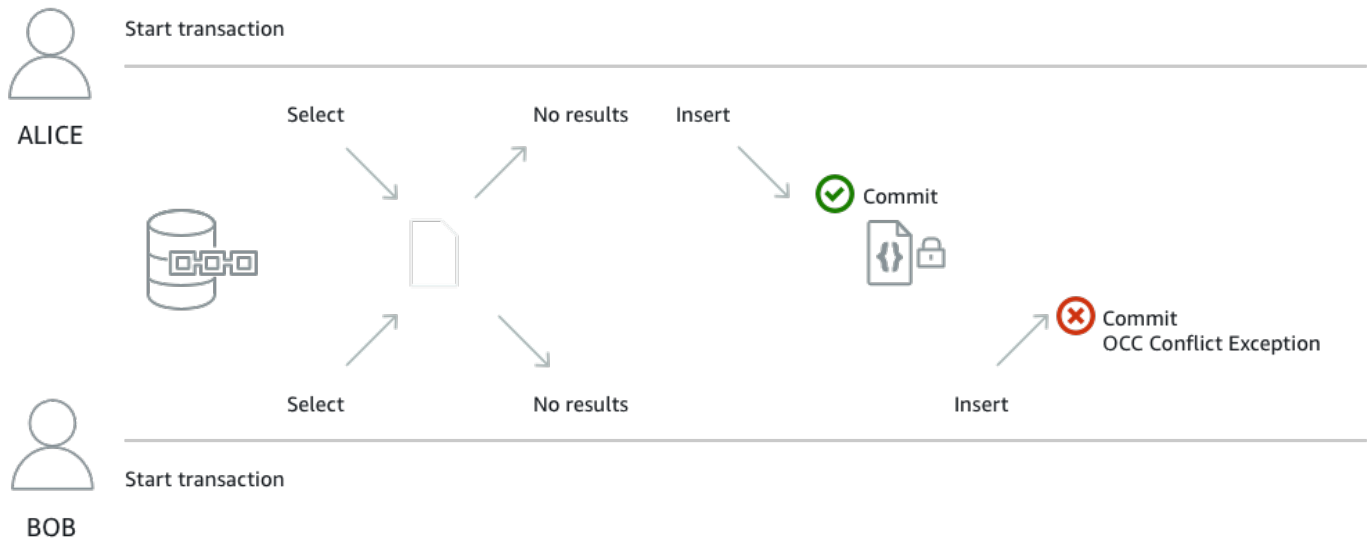
```
UPDATE Vehicle AS v
SET v.Color = 'Red'
WHERE v.Make = 'Tesla' AND v.Model = 'Model S'
```

Alice와 Bob이 동시에 트랜잭션을 시작한다고 가정해 봅시다. Alice의 UPDATE 문은 VIN 필드에서 인덱싱된 조회를 수행하므로 해당 문서 하나만 읽으면 됩니다. Alice가 먼저 트랜잭션을 완료하고 트랜잭션을 성공적으로 체결합니다.

Bob의 명령문은 인덱싱되지 않은 필드를 필터링하므로 표 스캔을 수행한 후 `OccConflictException`가 발생합니다. 그 이유는 Alice가 체결한 트랜잭션이 Bob의 문이 액세스하고 있는 데이터를 수정했기 때문입니다. 이러한 데이터에넌 Bob이 업데이트하는 문서뿐만 아니라 표의 모든 문서가 포함됩니다.

삽입 OCC 충돌

OCC 충돌에는 이전에 존재했던 문서뿐만 아니라 새로 삽입된 문서가 포함될 수 있습니다. 두 명의 동시 사용자(Alice와 Bob)가 한 원장의 동일한 표로 작업하고 있는 다음 다이어그램을 생각해 봅시다. 둘 경우 조건자 값이 아직 존재하지 않는 경우에만 새 문서를 삽입하려고 합니다.



이 예에서 Alice와 Bob은 모두 단일 트랜잭션 내에서 다음 SELECT 및 INSERT 문을 실행합니다. 애플리케이션은 SELECT 문이 결과를 반환하지 않는 경우에만 INSERT 문을 실행합니다.

```
SELECT * FROM Vehicle v WHERE v.VIN = 'ABCDE12345EXAMPLE'
```

```
INSERT INTO Vehicle VALUE
{
    'VIN' : 'ABCDE12345EXAMPLE',
    'Type' : 'Wagon',
    'Year' : 2019,
    'Make' : 'Subaru',
    'Model' : 'Outback',
    'Color' : 'Gray'
}
```

Alice와 Bob이 동시에 트랜잭션을 시작한다고 가정해 봅시다. 두 SELECT 쿼리 모두 ABCDE12345EXAMPLE의 VIN이 있는 기존 문서를 반환하지 않습니다. 따라서 애플리케이션은 INSERT 문을 따라 진행됩니다.

Alice가 먼저 트랜잭션을 완료하고 트랜잭션을 성공적으로 체결합니다. 그런 다음 Bob은 트랜잭션을 체결하려고 하지만, QLDB는 이럴 거절하고 OccConflictException가 발생합니다. Alice의 체결된 트랜잭션이 Bob의 SELECT 쿼리 결과 세트를 수정했고, OCC는 Bob의 트랜잭션을 체결하기 전에 이 충돌을 감지하기 때문입니다.

이 트랜잭션 예가 멱등성을 가지려면 SELECT 쿼리가 필요합니다. 그러면 Bob은 전체 트랜잭션을 처음부터 다시 시도할 수 있습니다. 하지만 다음 SELECT 쿼리는 Alice가 삽입한 문서를 반환하므로 Bob의 애플리케이션은 해당 INSERT를 실행하지 않습니다.

트랜잭션에 멱등성 부여하기

이전 섹션의 삽입 트랜잭션도 멱등성 트랜잭션의 한 예입니다. 즉, 동일한 트랜잭션을 여러 번 실행하면 동일한 결과가 생성됩니다. Bob이 특정 VIN가 이미 존재하는지 먼저 확인하지 않고 INSERT를 실행하면 표에 중복된 VIN 값이 있는 문서가 생성될 수 있습니다.

OCC 충돌 외에도 다른 재시도 시나리오도 살펴보세요. 예컨대, QLDB가 서버측에서 트랜잭션을 성공적으로 체결했지만 응답을 기다리는 동안 클라이언트 제한 시간이 초과될 수 있습니다. 가장 좋은 방법은 동시 실행 또는 재시도 시 예상치 못한 부작용이 발생하지 않도록 쓰기 트랜잭션에 멱득성을 부여하는 것입니다.

수정 OCC 충돌

QLDB는 동일한 저널 블록에서 수정본 수정이 동시에 발생하는 것을 방지합니다. 두 명의 동시 사용자(Alice와 Bob)가 원장의 동일한 블록에서 체결된 서로 다른 두 개의 문서 수정본을 수정하려는 경우를 예로 들어 보겠습니다. 먼저 Alice는 다음과 같이 REDACT_REVISION 저장 프로시저를 실행하여 한 수정본의 수정을 요청합니다.

```
EXEC REDACT_REVISION `{strandId:"JdxjkR9bSYB5jMHWcI464T", sequenceNo:17}`,
 '5PLf9SXwndd63lPaSIa0O6', 'ADR2Ll1fGsU4Jr4EqTdnQF'
```

그러면 Alice의 요청이 아직 처리 중인 동안 Bob은 다음과 같이 다른 수정본의 수정을 요청합니다.

```
EXEC REDACT_REVISION `{strandId:"JdxjkR9bSYB5jMHWcI464T", sequenceNo:17}`,
 '8F0TPCmdNQ6JTRpiLj2TmW', '05K8zpGYWynDlEOK5afDRc'
```

이들이 수정하려고 하는 두 개의수정본은 다르지만 QLDB는 `OccConflictException` 이 있는 Bob의 요청을 거부합니다. 이는 Bob의 수정본이 Alice가 수정 중인 수정본과 동일한 블록에 있기 때문입니다. Alice의 요청 처리가 완료되면 Bob은 수정 요청을 다시 시도할 수 있습니다.

마찬가지로, 두 개의 동시 트랜잭션이 동일한 수정 내용을 수정하려고 시도하는 경우 하나의 요청만 처리될 수 있습니다. 수정이 완료될 때까지 다른 요청은 OCC 충돌 예외로 실패합니다. 이후에 동일한 수정본을 수정해 달라는 요청을 하면 해당 수정본이 이미 수정되었음을 나타내는 오류가 발생합니다.

동시 세션 관리

관계형 데이터베이스 관리 시스템(RDBMS)을 사용해 본 경험이 있다면, 동시 연결 한도에 익숙할 것입니다. 트랜잭션이 HTTP 요청 및 응답 메시지로 실행되기 때문에 QLDB는 기존 RDBMS 연결과 같은 개념을 가지고 있지 않습니다.

QLDB에서 유사한 개념은 활성 세션입니다. 세션은 개념적으로 사용자 로그인과 비슷합니다. 즉, 원장에 대한 데이터 트랜잭션 요청에 대한 정보를 관리합니다. 활성 세션은 트랜잭션을 활발하게 실행하는 세션입니다. 또한 서비스가 즉시 다른 트랜잭션을 시작할 것으로 예상하는, 트랜잭션을 최근에 완료한 세션일 수도 있습니다. QLDB는 세션당 하나의 활성 실행 트랜잭션을 지원합니다.

원장당 동시 활성 세션의 한도는 Amazon QLDB 할당량 및 제한에 정의되어 있습니다. 이 한도에 도달한 후 트랜잭션을 시작하려는 모든 세션에서 오류(`LimitExceededException`)가 발생합니다.

세션의 라이프사이클와 데이터 트랜잭션 실행 시 QLDB 드라이버가 세션을 처리하는 방법에 대한 자세한 설명은 드라이버를 사용한 세션 관리 세션을 참조하세요. QLDB 드라이버를 사용하여 애플리케

이션에서 세션 풀을 구성하는 모범 사례는 Amazon QLDB 드라이버 권장 사항의 QLDBDriver 객체 구성 섹션을 참조하세요.

Amazon QLDB에서의 데이터 확인

Amazon QLDB를 사용하면 정확한 애플리케이션 데이터 변경 기록을 확인하고 신뢰할 수 있습니다. QLDB는 저널이라고 하는 변경 불가능한 트랜잭션 로그를 데이터 스토리지에 사용합니다. 저널은 커밋된 데이터의 모든 변경 내용을 추적하고 시간이 지나면서 완전하고 확인 가능한 시간대별 변경 기록을 유지합니다.

QLDB는 Merkle 트리 기반 모델과 함께 SHA-256 해시 함수를 사용하여 다이제스트라고하는 저널의 암호화 표현을 생성합니다. 다이제스트는 특정 시점을 기준으로 데이터의 전체 변경 기록에 대한 고유한 서명 역할을 합니다. 다이제스트를 사용하여 해당 서명과 관련된 문서 수정본의 무결성을 확인할 수 있습니다.

주제

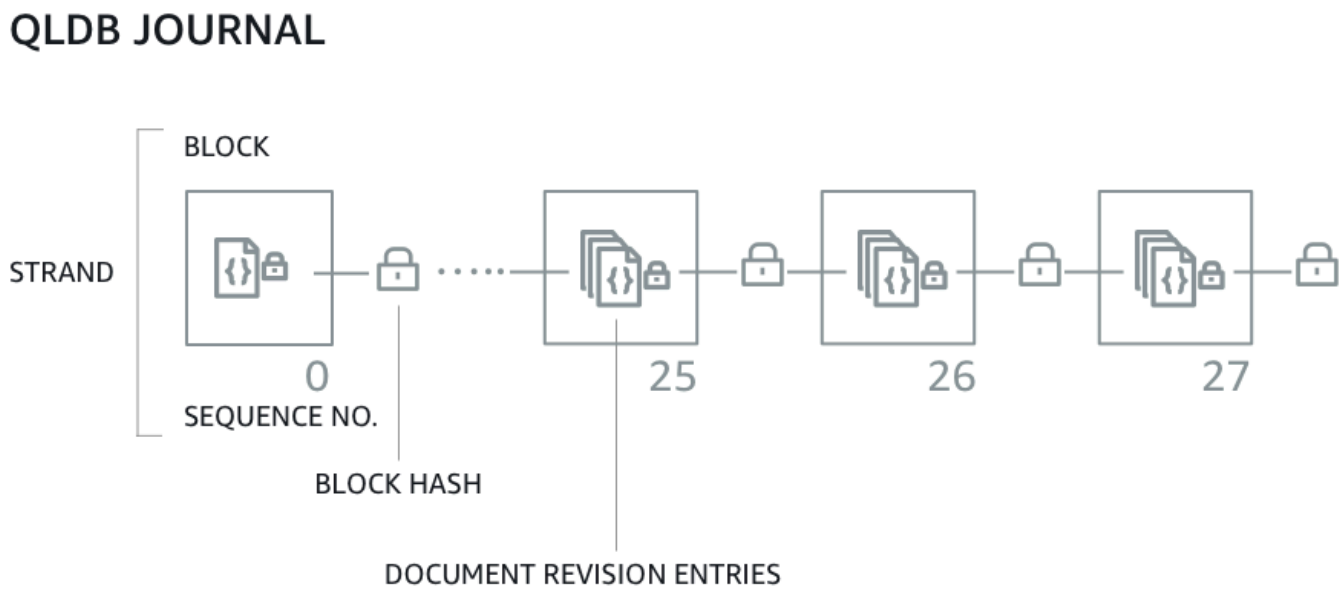
QLDB에서 확인할 수 있는 데이터 종류는 무엇입니까?

QLDB에서 각 원장에는 정확히 하나의 저널이 있습니다. 저널에는 저널의 파티션인 스트랜드가 여러 개 있을 수 있습니다.

> **Note**
>
> QLDB는 현재 단일 스트랜드가 포함된 저널만 지원합니다.

블록은 트랜잭션 중에 저널 스트랜드에 커밋되는 객체입니다. 이 블록에는 트랜잭션으로 인한 문서 수정본을 나타내는 항목 객체가 들어 있습니다. QLDB에서 개별 수정 또는 전체 저널 블록을 확인할 수 있습니다.

다음 다이어그램은 이 저널 구조를 보여줍니다.



다이어그램은 트랜잭션이 문서 수정본 항목이 포함된 블록 형태로 저널에 커밋됨을 보여줍니다. 또한 각 블록은 후속 블록에 해시 체인으로 연결되며 스트랜드 내에서 해당 주소를 지정하는 시퀀스 번호가 있다는 것을 보여줍니다.

블록의 데이터 내용에 대해 자세히 알아보려면 Amazon QLDB의 저널 콘텐츠 섹션을 참조하세요.

데이터 무결성이란 무엇을 의미하나요?

QLDB의 데이터 무결성은 원장의 저널이 사실상 변경이 불가능하다는 것을 의미합니다. 즉, 데이터(특히 각 문서 수정본)는 다음과 같은 상태에 있습니다.

1. 처음 작성된 저널 내 동일한 위치에 있습니다.
2. 작성된 이후로 어떤 식으로든 변경되지 않았습니다.

확인은 어떻게 작동하나요?

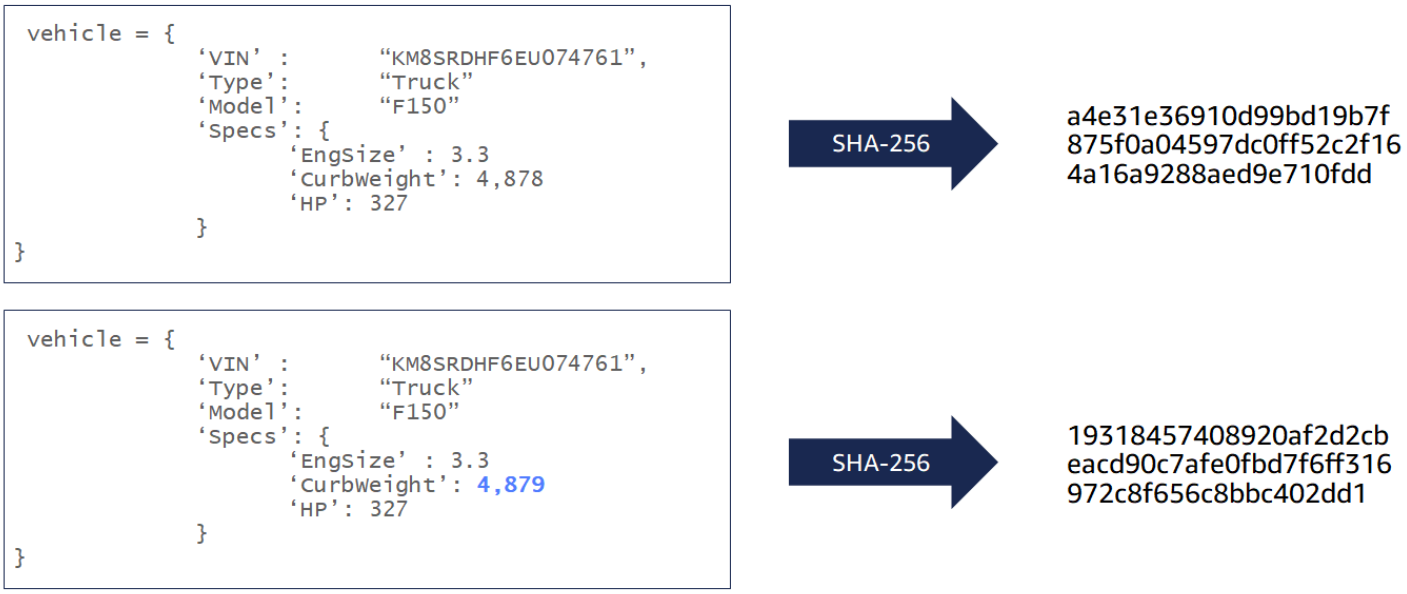
Amazon QLDB에서 확인이 작동하는 방식을 이해하기 위해 개념을 네 가지 기본 구성 요소로 나눌 수 있습니다.

- 해싱
- 다이제스트
- Merkle 트리
- 증명

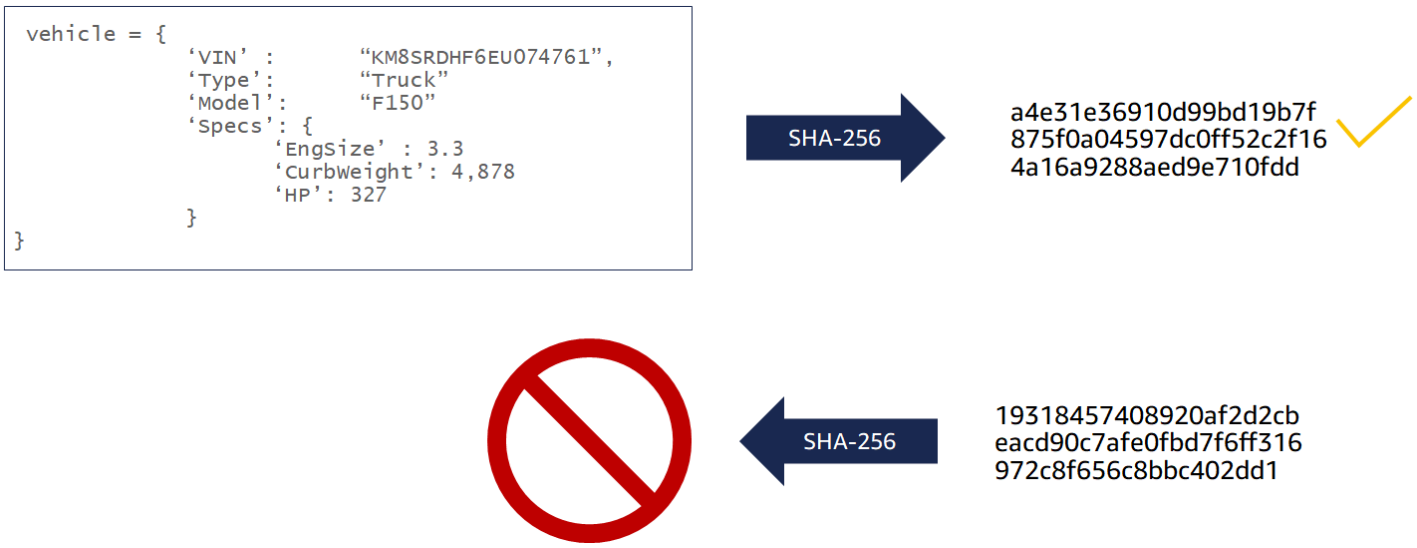
해싱

QLDB는 SHA-256 암호화 해시 함수를 사용하여 256비트 해시 값을 생성합니다. 해시는 임의의 양의 입력 데이터에 대해 고정된 길이의 고유한 서명 역할을 합니다. 단일 문자나 비트라도 입력의 일부를 변경하면 출력 해시가 완전히 변경됩니다.

다음 다이어그램은 SHA-256 해시 함수가 한 자릿수만 다른 두 QLDB 문서에 대해 완전히 고유한 해시 값을 생성하는 것을 보여줍니다.



SHA-256 해시 함수는 단방향으로, 출력이 주어졌을 때 입력을 계산하는 것이 수학적으로 불가능합니다. 다음 다이어그램은 출력 해시 값이 주어지면 입력 QLDB 문서를 계산하는 것이 불가능하다는 것을 보여줍니다.



다음 데이터 입력은 확인을 위해 QLDB에서 해싱됩니다.

- 문서 수정
- PATIL 문
- 수정 항목
- 저널 블록

다이제스트

다이제스트는 특정 시점의 원장 전체 일지를 암호로 표현한 것입니다. 저널은 첨부만 가능하며 저널 블록은 블록체인과 유사하게 시퀀싱되고 해시 체인으로 연결됩니다.

언제든지 원장에 대한 다이제스트를 요청할 수 있습니다. QLDB는 다이제스트를 생성하여 보안 출력 파일로 사용자에게 반환합니다. 그런 다음 해당 다이제스트를 사용하여 이전 시점에 커밋된 문서 수정본의 무결성을 확인합니다. 수정본으로 시작하여 다이제스트로 끝나는 방식으로 해시를 다시 계산하면 그 사이에 데이터가 변경되지 않았음을 증명할 수 있습니다.

Merkle 트리

원장 규모가 커질수록 확인을 위해 저널의 전체 해시 체인을 다시 계산하는 것은 점점 비효율적입니다. QLDB는 Merkle 트리 모델을 사용하여 이러한 비효율성을 해결합니다.

Merkle 트리는 각 리프 노드가 데이터 블록의 해시를 나타내는 트리 데이터 구조입니다. 리프가 아닌 각 노드는 해당 하위 노드의 해시입니다. 블록체인에서 일반적으로 사용되는 Merkle 트리는 감사 증명

메커니즘을 통해 대규모 데이터 세트를 효율적으로 확인하는 데 도움이 됩니다. Merkle 트리에 대한 자세한 내용은 Merkle 트리 Wikipedia 페이지를 참조하세요. Merkle 감사 증명에 대해 자세히 알아보고 인증서 투명성 사이트에서 로그 증명의 작동 방식 섹션을 참조하여 사용 사례 예시에 대해 알아보세요.

Merkle 트리의 QLDB 구현은 저널의 전체 해시 체인으로 구성됩니다. 이 모델에서 리프 노드는 모든 개별 문서 수정본 해시의 집합입니다. 루트 노드는 특정 시점을 기준으로 한 전체 저널의 다이제스트를 나타냅니다.

Merkle 감사 증명을 사용하면 원장 수정 기록의 일부 하위 집합만 확인하여 수정 내역을 확인할 수 있습니다. 주어진 리프 노드(수정본)에서 루트(다이제스트)까지 트리를 순회하여 이러한 작업을 수행할 수 있습니다. 이 순회 경로를 따라 노드 시블링 쌍을 재귀적으로 해싱하여 다이제스트가 끝날 때까지 상위 해시를 계산합니다. 이 순회는 트리에 있는 `log(n)` 노드의 시간 복잡도를 초래합니다.

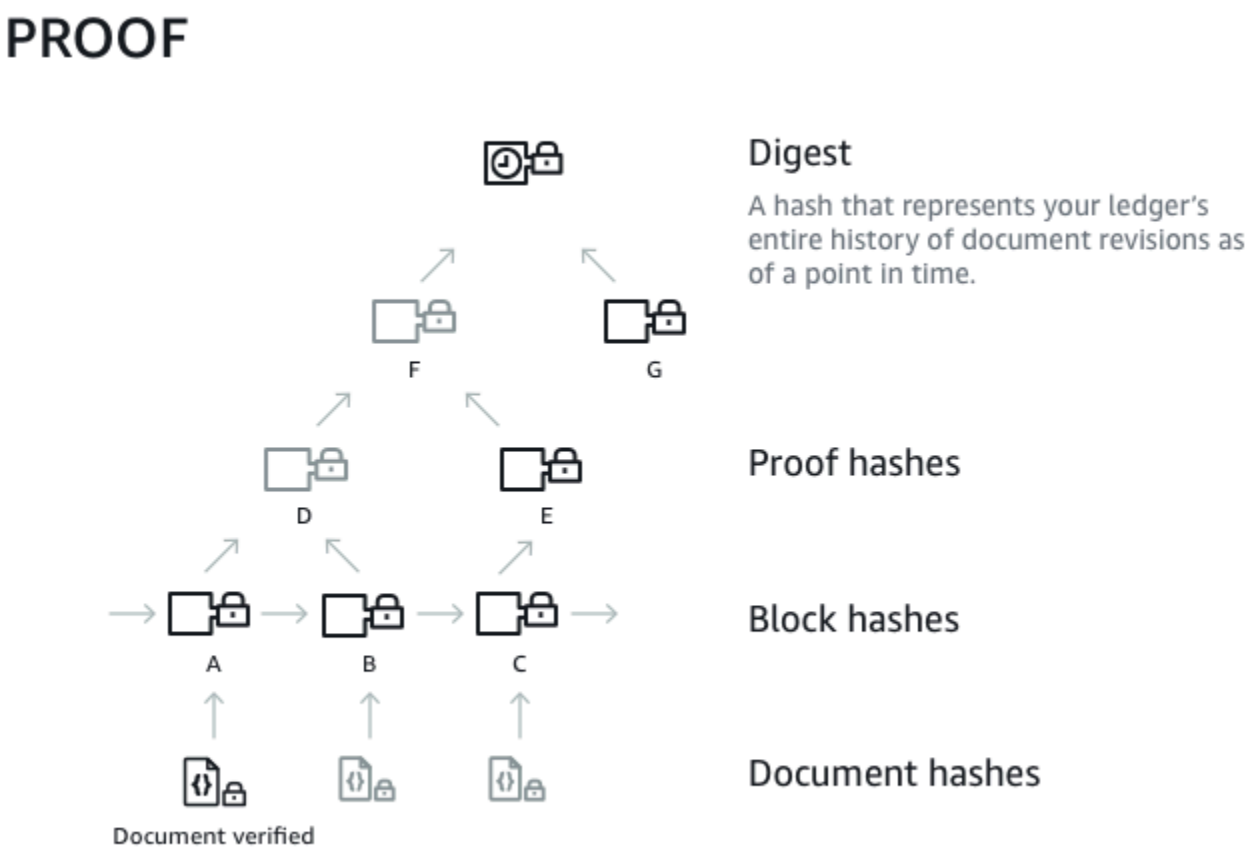
증명

증명은 주어진 다이제스트 및 문서 수정본에 대해 QLDB가 반환하는 노드 해시의 정렬된 목록입니다. 이는 Merkle 트리 모델이 주어진 리프 노드 해시(수정본)를 루트 해시(다이제스트)에 연결하는 데 필요한 해시로 구성됩니다.

수정본과 다이제스트 사이에 커밋된 데이터를 변경하면 저널의 해시 체인이 손상되어 증명을 생성할 수 없게 됩니다.

확인 예

다음 다이어그램은 Amazon QLDB 해시 트리 모델을 보여 줍니다. 최상위 루트 노드로 롤업되는 블록 해시 세트를 보여 주며, 이는 저널 스트랜드의 다이제스트를 나타냅니다. 단일 스트랜드 저널이 있는 원장에서 이 루트 노드는 전체 원장의 다이제스트이기도 합니다.



노드 A가 해시를 확인하고자 하는 문서 수정본 버전이 들어 있는 블록이라고 가정해 보겠습니다. B, E, G는 노드는 QLDB가 증명에 제공하는 순서가 지정된 해시 목록을 나타냅니다. 이러한 해시는 해시 A의 다이제스트를 다시 계산하는 데 필요합니다.

다이제스트를 다시 계산하려면 다음을 수행합니다.

1. 해시 A로 시작하여 해시 B와 연결합니다. 그런 다음 결과를 해싱하여 D를 계산합니다.
2. D와 E를 사용하여 F를 계산합니다.
3. F와 G를 사용하여 다이제스트를 계산합니다.

재계산된 다이제스트가 예상 값과 일치하면 확인이 성공한 것입니다. 수정 해시와 다이제스트를 고려하면 증명의 해시를 리버스 엔지니어링하는 것은 불가능합니다. 따라서 이 연습은 해당 수정본이 다이제스트를 기준으로 이 저널 위치에 실제로 작성되었음을 증명합니다.

데이터 수정은 확인에 어떤 영향을 미치나요?

Amazon QLDB에서 DELETE 문은 문서를 삭제된 것으로 표시하는 새 개정본을 생성하여 논리적으로만 문서를 삭제합니다. QLDB는 테이블 기록에서 비활성 문서 개정을 영구적으로 삭제할 수 있는 데이터 수정 작업도 지원합니다.

수정 작업은 지정된 수정 버전의 사용자 데이터만 삭제하고 저널 시퀀스와 문서 메타데이터는 변경되지 않습니다. 수정이 수정된 후에는 수정본의 사용자 데이터(data 구조로 표시됨)는 새 dataHash 필드로 대체됩니다. 이 필드의 값은 제거된 data 구조의 Amazon Ion 해시입니다. 수정 작업에 대한 자세한 내용과 예제는 문서 개정본 수정하기을 참조하십시오.

따라서 원장은 전반적인 데이터 무결성을 유지하고 기존 검증 API 작업을 통해 암호화 방식으로 검증 가능한 상태를 유지합니다. 예상대로 이러한 API 작업을 사용하여 다이제스트 (GetDigest) 를 요청하고 증명 (GetBlock또는 GetRevision) 을 요청한 다음 반환된 객체를 사용하여 확인 알고리즘을 실행할 수 있습니다.

수정 해시 재계산

해시를 다시 계산하여 개별 문서 수정본을 확인하려는 경우, 조건부로 수정본이 수정되었는지 여부를 확인해야 합니다. 수정본이 수정된 경우 dataHash 필드에 제공된 해시 값을 사용할 수 있습니다. 수정되지 않은 경우 data 필드를 사용하여 해시를 다시 계산할 수 있습니다.

이 조건부 검사를 통해 수정된 수정 내용을 식별하고 적절한 조치를 취할 수 있습니다. 예를 들어 모니터링을 위해 데이터 조작 이벤트를 기록할 수 있습니다.

알림 시작하기

데이터를 확인하려면 먼저 원장에서 다이제스트를 요청하고 나중을 위해 저장해야 합니다. 다이제스트에서 다루는 최신 블록 이전에 커밋된 문서 수정본은 해당 다이제스트에 대한 확인을 받을 수 있습니다.

그런 다음 Amazon QLDB에 확인하려는 적격 수정본에 대한 증명을 요청합니다. 이 증명을 사용하면 클라이언트측 API를 호출하여 수정 해시부터 시작하여 다이제스트를 다시 계산합니다. 이전에 저장한 다이제스트를 QLDB 외부에서 알고 신뢰할 수 있는 한, 다시 계산된 다이제스트 해시가 저장된 다이제스트 해시와 일치하면 문서의 무결성이 입증됩니다.

> **Important**
>
> - 이 다이제스트를 저장한 시점과 확인을 실행한 시점 사이에 문서 수정본이 변경되지 않았다는 것을 구체적으로 증명해야 합니다. 나중에 확인하려는 수정본이 저널에 커밋되는 즉시 다이제스트를 요청하고 저장할 수 있습니다.
> - 가장 좋은 방법은 다이제스트를 정기적으로 요청하여 원장과 떨어진 곳에 보관하는 것입니다. 원장의 수정본을 커밋하는 빈도에 따라 다이제스트 요청 빈도를 결정합니다.
>
> 실제 사용 사례의 맥락에서 암호화 검증의 가치를 설명하는 자세한 AWS 블로그 게시물은 Amazon QLDB를 사용한 실제 암호화 검증을 참조하십시오.

원장에 요약을 요청하고 데이터를 검증하는 방법에 대한 step-by-step 지침은 다음을 참조하십시오.

- 1단계: QLDB에서 다이제스트 요청
- 2단계: QLDB에서 데이터 확인

1단계: QLDB에서 다이제스트 요청

Amazon QLDB는 원장에 있는 저널의 현재 팁을 다루는 다이제스트를 요청하는 API를 제공합니다. 저널 팁은 QLDB가 요청을 수신한 시점을 기준으로 가장 최근에 커밋된 블록을 나타냅니다. AWS Management Console, AWS SDK 또는 AWS Command Line Interface (AWS CLI) 를 사용하여 다이제스트를 가져올 수 있습니다.

주제

- AWS Management Console
- QLDB API

AWS Management Console

다음 단계에 따라 QLDB 콘솔을 사용하여 리소스를 복원합니다.

다이제스트를 요청하려면(콘솔)

1. 에 AWS Management Console로그인하고 https://console.aws.amazon.com/qldb 에서 Amazon QLDB 콘솔을 엽니다.

2. 탐색 창에서 원장을 선택합니다.
3. 원장 목록에서 다이제스트를 요청하려는 원장명을 선택합니다.
4. 다이제스트 가져오기를 선택합니다. 다이제스트 가져오기 대화 상자에는 다음과 같은 다이제스트 세부 정보가 표시됩니다.

   - 다이제스트 – 요청한 다이제스트의 SHA-256 해시 값.
   - 다이제스트 팁 주소 – 요청한 다이제스트에 포함된 저널의 최신 블록 위치. 주소에는 다음과 같은 두 가지 필드가 있습니다.
     - `strandId` - 블록을 포함하는 저널 스트랜드의 고유 ID입니다.
     - `sequenceNo` – 스트랜드 내 블록의 위치를 지정하는 인덱스 번호.
   - 원장 – 다이제스트를 요청한 원장 이름.
   - 날짜 – 다이제스트를 요청한 시점의 타임스탬프.
5. 다이제스트 정보를 검토합니다. 그런 다음 저장을 선택합니다. 기본 파일 이름을 유지하거나 새 이름을 입력할 수 있습니다.

   > **Note**
   >
   > 원장의 데이터를 수정하지 않아도 다이제스트 해시 및 팁 주소 값이 변경되는 것을 확인할 수 있습니다. 이는 PartiQL 편집기에서 쿼리를 실행할 때마다 콘솔이 원장의 시스템 카탈로그를 검색하기 때문입니다. 이는 저널에 커밋되어 최신 블록 주소를 변경하는 읽기 트랜잭션입니다.

   이 단계는 콘텐츠가 포함된 일반 텍스트 파일을 Amazon Ion 형식으로 저장합니다. 파일의 파일 이름 확장자는 `.ion.txt`이며 이전 대화 상자에 나열된 모든 다이제스트 정보를 포함합니다. 다음은 다이제스트 파일의 내용을 보여주는 예제입니다. 필드 순서는 브라우저에 따라 다를 수 있습니다.

   ```
   {
     "digest": "42zaJOfV8iGutVGNaIuzQWhD5Xb/5B9lScHnvxPXm9E=",
     "digestTipAddress": "{strandId:\"BlFTjlSXze9BIh1KOszcE3\",sequenceNo:73}",
     "ledger": "my-ledger",
     "date": "2019-04-17T16:57:26.749Z"
   }
   ```
6. 나중에 액세스할 수 있는 위치에 이 파일을 저장하세요. 나중에 이 파일을 사용하여 문서 수정본을 확인할 수 있습니다.

> **Important**
>
> 나중에 확인하는 문서 수정본은 저장한 다이제스트에 포함되어야 합니다. 즉, 문서 주소의 시퀀스 번호는 다이제스트 팁 주소의 시퀀스 번호보다 작거나 같아야 합니다.

QLDB API

Amazon QLDB API를 AWS SDK 또는 AWS CLI와 함께 사용하여 원장의 다이제스트를 요청할 수도 있습니다. 애플리케이션 프로그램에서 사용할 수 있는 다음과 같은 기능이 API에서 제공됩니다.

- GetDigest— 저널에서 가장 최근에 커밋된 블록의 원장 요약을 반환합니다. 응답에는 256비트 해시 값과 블록 주소가 포함됩니다.

를 사용하여 다이제스트를 요청하는 방법에 대한 자세한 내용은 명령 참조의 AWS CLIget-digest 명령을 참조하십시오.AWS CLI

샘플 애플리케이션

자바 코드 예제는 aws-samples/ -java 리포지토리를 참조하십시오. GitHub amazon-qldb-dmv-sample 이 샘플 애플리케이션을 다운로드하여 설치하는 방법에 대한 자세한 내용은 Amazon QLDB Java 샘플 애플리케이션 설치 섹션을 참조하세요. 다이제스트를 요청하기 전에 Java 자습서의 1~3단계에 따라 샘플 원장을 생성하고 샘플 데이터와 함께 로드해야 합니다.

클래스의 자습서 코드는 샘플 원장에 다이제스트를 요청하는 예를 GetDigest제공합니다. `vehicle-registration`

저장한 다이제스트를 사용하여 문서 수정본을 확인하려면 2단계: QLDB에서 데이터 확인로 진행합니다.

2단계: QLDB에서 데이터 확인

Amazon QLDB는 지정된 문서 ID 및 연결된 블록에 대한 증명을 요청하는 API를 제공합니다. 1단계: QLDB에서 다이제스트 요청에 설명된 대로 이전에 저장한 다이제스트의 팁 주소도 제공해야 합니다. AWS Management Console, AWS SDK 또는 를 사용하여 증거를 얻을 수 있습니다. AWS CLI

그런 다음 QLDB에서 반환한 증명을 사용하여 클라이언트측 API를 사용해 저장된 다이제스트와 비교하여 문서 수정본을 확인할 수 있습니다. 이를 통해 데이터 확인에 사용하는 알고리즘을 제어할 수 있습니다.

주제

- AWS Management Console
- QLDB API

AWS Management Console

이 단원에서는 Amazon QLDB 콘솔을 사용하여 이전에 저장한 다이제스트와 비교하여 문서 수정본을 확인하는 단계를 설명합니다.

시작하기 전에 먼저 1단계: QLDB에서 다이제스트 요청의 단계를 따라야 합니다. 확인을 위해서는 확인하려는 수정본을 포함하는 이전에 저장한 다이제스트가 필요합니다.

문서 수정본을 확인하려면(콘솔)

1. https://console.aws.amazon.com/qldb에서 Amazon QLDB 콘솔을 엽니다.
2. 먼저 원장에서 확인하려는 수정본의 `id` 및 `blockAddress`를 쿼리합니다. 이 필드는 문서의 메타데이터에 포함되며, 커밋된 뷰에서 쿼리할 수 있습니다.

   문서 `id`는 시스템에서 할당한 고유 ID 문자열입니다. `blockAddress`는 개정이 커밋된 블록 위치를 지정하는 Ion 구조입니다.

   탐색 창에서 PartiQL 편집기를 선택합니다.
3. 수정본을 확인하려는 원장 이름을 선택합니다.
4. 쿼리 편집기에 다음 구문에서 `SELECT` 문을 입력한 다음 실행을 선택합니다.

```
SELECT metadata.id, blockAddress FROM _ql_committed_table_name
WHERE criteria
```

   예를 들어, 다음 쿼리는 Amazon QLDB 콘솔 시작하기에서 생성된 샘플 원장의 `VehicleRegistration` 테이블에서 문서를 반환합니다.

```
SELECT r.metadata.id, r.blockAddress FROM _ql_committed_VehicleRegistration AS r
WHERE r.data.VIN = 'KM8SRDHF6EU074761'
```

5. 쿼리가 반환하는 `id` 및 `blockAddress` 값을 복사하여 저장합니다. `id` 필드의 큰따옴표는 반드시 생략하십시오 Amazon Ion에서 문자열 데이터 유형이 큰따옴표로 구분됩니다. 예를 들어, 다음 스니펫의 영숫자 텍스트만 복사해야 합니다.

   `"LtMNJYNjSwzBLgf7sLifrG"`

6. 이제 문서 수정본을 선택했으니 확인 프로세스를 시작할 수 있습니다.

   탐색 창에서 검증을 선택합니다.

7. 문서 검증 양식의 검증하려는 문서 지정에 다음 입력 파라미터를 입력합니다.

   - 원장 – 수정본을 확인하려는 원장.
   - 블록 주소 – 4단계에서 쿼리를 통해 반환된 `blockAddress` 값.
   - 문서 ID – 4단계에서 쿼리를 통해 반환된 `id` 값.

8. 확인에 사용할 다이제스트 지정에서 다이제스트 선택을 선택하여 이전에 저장한 다이제스트를 선택합니다. 파일이 유효하면 콘솔의 모든 다이제스트 필드가 자동으로 채워집니다. 또는 다이제스트 파일에서 직접 다음 값을 수동으로 복사하여 붙여넣을 수 있습니다.

   - 다이제스트 – 다이제스트 파일의 `digest` 값.
   - 다이제스트 팁 주소 – 다이제스트 파일의 `digestTipAddress` 값.

9. 문서 및 다이제스트 입력 파라미터를 검토한 다음 검증을 선택합니다.

   콘솔은 다음 두 단계를 자동화합니다.

   a. 지정된 문서에 대한 증거를 QLDB에 요청합니다.

   b. QLDB에서 반환한 증거를 사용하여 제공된 다이제스트에 대해 문서 개정을 검증하는 클라이언트 측 API를 호출합니다. 이 확인 알고리즘을 검사하려면 다음 QLDB API 섹션을 참조하여 코드 예제를 다운로드하세요.

   콘솔은 검증 결과 카드에 요청 결과를 표시합니다. 자세한 내용은 확인 결과을 참조하십시오.

QLDB API

Amazon QLDB API를 AWS SDK 또는 AWS CLI와 함께 사용하여 문서 수정본을 확인할 수도 있습니다. 애플리케이션 프로그램에서 사용할 수 있는 다음과 같은 기능이 QLDB API에서 제공됩니다.

- `GetDigest` – 저널에서 가장 최근에 커밋된 블록의 원장 다이제스트를 반환합니다. 응답에는 256비트 해시 값과 블록 주소가 포함됩니다.

- `GetBlock` – 저널의 지정된 주소에 있는 블록 객체를 반환합니다. 또한 `DigestTipAddress`가 제공된 경우 검증을 위해 지정된 블록의 증명을 반환합니다.
- `GetRevision` – 지정된 문서 ID 및 블록 주소의 수정본 데이터 객체를 반환합니다. 또한 `DigestTipAddress`가 제공된 경우 검증을 위해 지정된 수정본의 증명을 반환합니다.

이러한 API작업 설명 전체를 보려면 Amazon QLDB API 참조 섹션을 참조하세요.

를 사용한 데이터 검증에 대한 자세한 내용은 AWS CLI 명령 참조를 참조하십시오. AWS CLI

샘플 애플리케이션

자바 코드 예제는 amazon-qldb-dmv-sampleaws-samples/ -java GitHub 리포지토리를 참조하십시오. 이 샘플 애플리케이션을 다운로드하여 설치하는 방법에 대한 자세한 내용은 Amazon QLDB Java 샘플 애플리케이션 설치 섹션을 참조하세요. 확인을 수행하기 전에 Java 자습서의 1~3단계에 따라 샘플 원장을 생성하고 샘플 데이터와 함께 로드해야 합니다.

클래스의 자습서 코드는 문서 개정에 대한 증명을 요청한 다음 해당 개정을 확인하는 예를 GetRevision제공합니다. 이 클래스는 다음 단계를 실행합니다.

1. 샘플 원장 `vehicle-registration`에서 새 다이제스트를 요청합니다.
2. `vehicle-registration` 원장의 `VehicleRegistration` 테이블에서 샘플 문서 수정본에 대한 증명을 요청합니다.
3. 반환된 다이제스트과 증명을 사용하여 샘플 수정을 확인합니다.

확인 결과

이 섹션에서는 AWS Management Console에서 Amazon QLDB 데이터 확인 요청을 통해 반환된 결과를 설명합니다. 확인 요청을 제출하는 방법에 대한 자세한 단계는 2단계: QLDB에서 데이터 확인 섹션을 참조하세요.

QLDB 콘솔의 확인 페이지에서 요청 결과가 확인 결과 카드에 표시됩니다. 증명 탭에는 지정된 문서 수정본 및 다이제스트에 대해 QLDB에서 반환한 증명의 콘텐츠가 표시됩니다. 여기에는 다음 이벤트가 포함됩니다.

- 수정 해시 – 확인 중인 문서 수정본을 고유하게 나타내는 SHA-256 값.
- 증명 해시 – 지정된 다이제스트를 다시 계산하는 데 사용되는 QLDB에서 제공하는 해시의 정렬된 목록. 콘솔은 수정 해시로 시작하여 다시 계산된 다이제스트로 끝날 때까지 각 증명 해시와 순차적으로 결합합니다.

목록은 기본적으로 축소되어 있으므로 목록을 확장하여 해시 값을 표시할 수 있습니다. 필요에 따라 증명을 사용하여 다이제스트를 다시 계산하기에 설명된 단계에 따라 직접 해시 계산을 시도할 수도 있습니다.

- 다이제스트 계산 - 수정 해시에서 수행된 일련의 해시 계산의 결과로 생성된 해시. 이 값이 이전에 저장한 다이제스트와 일치하면 확인이 성공합니다.

블록 탭에는 확인 중인 수정본이 포함된 블록의 콘텐츠가 표시됩니다. 여기에는 다음 이벤트가 포함됩니다.

- 거래 ID – 이 블록을 커밋한 거래의 고유 ID.
- 트랜잭션 시간 – 이 블록이 스트랜드에 커밋된 시점의 타임스탬프.
- 블록 해시 - 이 블록과 모든 콘텐츠를 고유하게 나타내는 SHA-256 값.
- 블록 주소 – 이 블록이 커밋된 원장 일지 내 위치. 주소에는 다음과 같은 두 가지 필드가 있습니다.
  - 스트랜드 ID – 이 블록을 포함하는 저널 스트랜드의 고유 ID.
  - 시퀀스 번호 – 스트랜드 내에서 이 블록의 위치를 지정하는 인덱스 번호.
- 문 – 이 블록의 항목을 커밋하기 위해 수행된 PartiQL 문.

> **Note**
>
> 파라미터화된 문을 프로그래밍 방식으로 실행하면 리터럴 데이터 대신 바인드 파라미터를 사용하여 저널 블록에 기록됩니다. 예를 들어, 저널 블록에서 다음 명령문을 볼 수 있는데, 여기서 물음표(?)는 문서 콘텐츠의 변수 자리 표시자입니다.
>
> ```
> INSERT INTO Vehicle ?
> ```

- 문서 항목 – 이 블록에서 커밋된 문서 수정본.

요청에서 문서 수정본 확인에 실패한 경우 가능한 원인에 대한 자세한 내용은 확인을 위한 일반적인 오류 섹션을 참조하세요.

증명을 사용하여 다이제스트를 다시 계산하기

QLDB가 문서 확인 요청에 대한 증명을 반환한 후 사용자가 직접 해시 계산을 시도할 수 있습니다. 이 섹션에서는 제공된 증명을 사용하여 다이제스트를 다시 계산하는 상위 단계를 설명합니다.

먼저 수정 해시를 증명 해시 목록의 첫 번째 해시와 페어링합니다. 그리고 다음을 수행합니다.

1. 두 해시를 정렬합니다. 리틀 엔디안 순서의 서명된 바이트 값을 기준으로 해시를 비교합니다.
2. 두 해시를 정렬된 순서대로 연결합니다.
3. SHA-256 해시 생성기를 사용하여 연결된 쌍을 해싱합니다.
4. 새 해시를 증명의 다음 해시와 페어링하고 1~3단계를 반복합니다. 마지막 증명 해시를 처리한 후에는 새 해시가 재계산된 다이제스트가 됩니다.

재계산된 다이제스트가 이전에 저장한 다이제스트와 일치하면 문서가 성공적으로 확인됩니다.

이러한 검증 단계를 보여주는 코드 예제가 포함된 step-by-step 자습서를 보려면 을 자습서: AWS SDK를 사용한 데이터 검증 참조하십시오.

자습서: AWS SDK를 사용한 데이터 검증

이 자습서에서는 SDK를 통해 QLDB API를 사용하여 Amazon QLDB 원장의 문서 수정 해시와 저널 블록 해시를 확인합니다. AWS 또한 QLDB 드라이버를 사용하여 문서 수정본을 쿼리할 수 있습니다.

예를 들어 차량 식별 번호(VIN)가 `KM8SRDHF6EU074761`인 차량의 데이터가 포함된 문서 수정본이 있다고 가정해 보겠습니다. 문서 수정본은 `vehicle-registration`라는 이름의 원장에 있는 `VehicleRegistration` 테이블에 있습니다. 이 차량에 대한 문서 수정본과 수정본이 포함된 저널 블록 모두의 무결성을 확인한다고 가정해 보겠습니다.

> **Note**
>
> 실제 사용 사례의 맥락에서 암호화 검증의 가치를 설명하는 자세한 AWS 블로그 게시물은 Amazon QLDB를 사용한 실제 암호화 검증을 참조하십시오.

주제

필수 조건

시작하기 전에 다음을 수행해야 합니다.

1. Amazon QLDB 드라이버 시작하기에서 해당 사전 조건을 완료하여 원하는 언어에 맞게 QLDB 드라이버를 설정합니다. 여기에는 가입, 개발을 위한 프로그래밍 방식 액세스 권한 부여 AWS, 개발 환경 구성 등이 포함됩니다.
2. Amazon QLDB 콘솔 시작하기의 1~2단계에 따라 `vehicle-registration`라는 이름의 원장을 생성하고 사전 정의된 샘플 데이터와 함께 로드하세요.

그런 다음, 다음 단계를 검토하여 확인 작동 방식을 알아본 다음 전체 코드 예제를 처음부터 끝까지 실행합니다.

1단계: 다이제스트 요청

데이터를 확인하려면 먼저 나중에 사용할 수 있도록 원장 `vehicle-registration`에 다이제스트를 요청해야 합니다.

Java

```
// Get a digest
GetDigestRequest digestRequest = new GetDigestRequest().withName(ledgerName);
GetDigestResult digestResult = client.getDigest(digestRequest);

java.nio.ByteBuffer digest = digestResult.getDigest();

// expectedDigest is the buffer we will use later to compare against our calculated
 digest
byte[] expectedDigest = new byte[digest.remaining()];
digest.get(expectedDigest);
```

.NET

```
// Get a digest
```

```
GetDigestRequest getDigestRequest = new GetDigestRequest
{
    Name = ledgerName
};
GetDigestResponse getDigestResponse =
 client.GetDigestAsync(getDigestRequest).Result;

// expectedDigest is the buffer we will use later to compare against our calculated
 digest
MemoryStream digest = getDigestResponse.Digest;
byte[] expectedDigest = digest.ToArray();
```

Go

```
// Get a digest
currentLedgerName := ledgerName
input := qldb.GetDigestInput{Name: &currentLedgerName}
digestOutput, err := client.GetDigest(&input)
if err != nil {
    panic(err)
}

// expectedDigest is the buffer we will later use to compare against our calculated
 digest
expectedDigest := digestOutput.Digest
```

Node.js

```
// Get a digest
const getDigestRequest: GetDigestRequest = {
    Name: ledgerName
};
const getDigestResponse: GetDigestResponse = await
 qldbClient.getDigest(getDigestRequest).promise();

// expectedDigest is the buffer we will later use to compare against our calculated
 digest
const expectedDigest: Uint8Array = <Uint8Array>getDigestResponse.Digest;
```

Python

```
# Get a digest
```

```
get_digest_response = qldb_client.get_digest(Name=ledger_name)

# expected_digest is the buffer we will later use to compare against our calculated
 digest
expected_digest = get_digest_response.get('Digest')
digest_tip_address = get_digest_response.get('DigestTipAddress')
```

2단계: 문서 수정본 쿼리

QLDB 드라이버를 사용하여 VIN KM8SRDHF6EU074761과 연결된 블록 주소, 해시 및 문서 ID를 쿼리합니다.

Java

```
// Retrieve info for the given vin's document revisions
Result result = driver.execute(txn -> {
    final String query = String.format("SELECT blockAddress, hash, metadata.id FROM
 _ql_committed_%s WHERE data.VIN = '%s'", tableName, vin);
    return txn.execute(query);
});
```

.NET

```
// Retrieve info for the given vin's document revisions
var result = driver.Execute(txn => {
    string query = $"SELECT blockAddress, hash, metadata.id FROM
 _ql_committed_{tableName} WHERE data.VIN = '{vin}'";
    return txn.Execute(query);
});
```

Go

```
// Retrieve info for the given vin's document revisions
result, err := driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    statement := fmt.Sprintf(
            "SELECT blockAddress, hash, metadata.id FROM _ql_committed_%s WHERE
 data.VIN = '%s'",
            tableName,
            vin)
    result, err := txn.Execute(statement)
```

```
    if err != nil {
        return nil, err
    }

    results := make([]map[string]interface{}, 0)

    // Convert the result set into a map
    for result.Next(txn) {
        var doc map[string]interface{}
        err := ion.Unmarshal(result.GetCurrentData(), &doc)
        if err != nil {
            return nil, err
        }
        results = append(results, doc)
    }
    return results, nil
})
if err != nil {
    panic(err)
}
resultSlice := result.([]map[string]interface{})
```

Node.js

```
const result: dom.Value[] = await driver.executeLambda(async (txn:
 TransactionExecutor): Promise<dom.Value[]> => {
    const query: string = `SELECT blockAddress, hash, metadata.id FROM
 _ql_committed_${tableName} WHERE data.VIN = '${vin}'`;
    const queryResult: Result = await txn.execute(query);
    return queryResult.getResultList();
});
```

Python

```
def query_doc_revision(txn):
    query = "SELECT blockAddress, hash, metadata.id FROM _ql_committed_{} WHERE
 data.VIN = '{}'".format(table_name, vin)
    return txn.execute_statement(query)

# Retrieve info for the given vin's document revisions
result = qldb_driver.execute_lambda(query_doc_revision)
```

3단계: 수정에 대한 증명 요청

쿼리 결과를 반복해서 살펴보고 각 블록 주소 및 문서 ID를 원장 이름과 함께 사용하여 `GetRevision` 요청을 제출합니다. 수정에 대한 증명을 가져오려면 이전에 저장한 다이제스트의 팁 주소도 제공해야 합니다. 이 API 작업은 문서 수정본과 수정본에 대한 증명이 포함된 객체를 반환합니다.

저널 블록 구조 및 내용에 대한 자세한 내용은 문서 메타데이터 쿼리 섹션을 참조하세요.

Java

```
for (IonValue ionValue : result) {
    IonStruct ionStruct = (IonStruct)ionValue;

    // Get the requested fields
    IonValue blockAddress = ionStruct.get("blockAddress");
    IonBlob hash = (IonBlob)ionStruct.get("hash");
    String metadataId = ((IonString)ionStruct.get("id")).stringValue();

    System.out.printf("Verifying document revision for id '%s'%n", metadataId);

    String blockAddressText = blockAddress.toString();

    // Submit a request for the revision
    GetRevisionRequest revisionRequest = new GetRevisionRequest()
            .withName(ledgerName)
            .withBlockAddress(new ValueHolder().withIonText(blockAddressText))
            .withDocumentId(metadataId)
            .withDigestTipAddress(digestResult.getDigestTipAddress());

    // Get a result back
    GetRevisionResult revisionResult = client.getRevision(revisionRequest);

    ...
}
```

.NET

```
foreach (IIonValue ionValue in result)
{
    IIonStruct ionStruct = ionValue;

    // Get the requested fields
```

```
    IIonValue blockAddress = ionStruct.GetField("blockAddress");
    IIonBlob hash = ionStruct.GetField("hash");
    String metadataId = ionStruct.GetField("id").StringValue;

    Console.WriteLine($"Verifying document revision for id '{metadataId}'");

    // Use an Ion Reader to convert block address to text
    IIonReader reader = IonReaderBuilder.Build(blockAddress);
    StringWriter sw = new StringWriter();
    IIonWriter textWriter = IonTextWriterBuilder.Build(sw);
    textWriter.WriteValues(reader);
    string blockAddressText = sw.ToString();

    // Submit a request for the revision
    GetRevisionRequest revisionRequest = new GetRevisionRequest
    {
        Name = ledgerName,
        BlockAddress = new ValueHolder
        {
            IonText = blockAddressText
        },
        DocumentId = metadataId,
        DigestTipAddress = getDigestResponse.DigestTipAddress
    };

    // Get a response back
    GetRevisionResponse revisionResponse =
 client.GetRevisionAsync(revisionRequest).Result;

    ...
}
```

Go

```
for _, value := range resultSlice {
    // Get the requested fields
    ionBlockAddress, err := ion.MarshalText(value["blockAddress"])
    if err != nil {
        panic(err)
    }
    blockAddress := string(ionBlockAddress)
    metadataId := value["id"].(string)
    documentHash := value["hash"].([]byte)
```

```
    fmt.Printf("Verifying document revision for id '%s'\n", metadataId)

    // Submit a request for the revision
    revisionInput := qldb.GetRevisionInput{
        BlockAddress:     &qldb.ValueHolder{IonText: &blockAddress},
        DigestTipAddress: digestOutput.DigestTipAddress,
        DocumentId:       &metadataId,
        Name:             &currentLedgerName,
    }

    // Get a result back
    revisionOutput, err := client.GetRevision(&revisionInput)
    if err != nil {
        panic(err)
    }

    ...
}
```

Node.js

```
for (let value of result) {
    // Get the requested fields
    const blockAddress: dom.Value = value.get("blockAddress");
    const hash: dom.Value = value.get("hash");
    const metadataId: string = value.get("id").stringValue();

    console.log(`Verifying document revision for id '${metadataId}'`);

    // Submit a request for the revision
    const revisionRequest: GetRevisionRequest = {
        Name: ledgerName,
        BlockAddress: {
            IonText: dumpText(blockAddress)
        },
        DocumentId: metadataId,
        DigestTipAddress: getDigestResponse.DigestTipAddress
    };

    // Get a response back
    const revisionResponse: GetRevisionResponse = await
 qldbClient.getRevision(revisionRequest).promise();
```

```
    ...
}
```

Python

```
for value in result:
    # Get the requested fields
    block_address = value['blockAddress']
    document_hash = value['hash']
    metadata_id = value['id']

    print("Verifying document revision for id '{}'".format(metadata_id))

    # Submit a request for the revision and get a result back
    proof_response = qldb_client.get_revision(Name=ledger_name,

 BlockAddress=block_address_to_dictionary(block_address),
                                                DocumentId=metadata_id,
                                                DigestTipAddress=digest_tip_address)
```

그런 다음 요청된 수정본에 대한 증명을 검색하세요.

QLDB API는 정렬된 노드 해시 목록을 문자열로 표현하여 증명을 반환합니다. 이 문자열을 노드 해시의 이진수 표현 목록으로 변환하려면 Amazon Ion 라이브러리의 Ion 리더를 사용합니다. Ion 라이브러리 사용에 대한 자세한 내용은 Amazon Ion Cookbook을 참조하세요.

Java

이 예제에서는 `IonReader`를 사용하여 이진수 변환을 수행합니다.

```
String proofText = revisionResult.getProof().getIonText();

// Take the proof and convert it to a list of byte arrays
List<byte[]> internalHashes = new ArrayList<>();
IonReader reader = SYSTEM.newReader(proofText);
reader.next();
reader.stepIn();
while (reader.next() != null) {
    internalHashes.add(reader.newBytes());
}
```

.NET

이 예제에서는 IonLoader를 사용하여 증명을 Ion 데이터그램으로 로드합니다.

```
string proofText = revisionResponse.Proof.IonText;
IIonDatagram proofValue = IonLoader.Default.Load(proofText);
```

Go

이 예제에서는 Ion 리더를 사용하여 증명을 이진수로 변환하고 증명의 노드 해시 목록을 반복해서 살펴봅니다.

```
proofText := revisionOutput.Proof.IonText

// Use ion.Reader to iterate over the proof's node hashes
reader := ion.NewReaderString(*proofText)
// Enter the struct containing node hashes
reader.Next()
if err := reader.StepIn(); err != nil {
    panic(err)
}
```

Node.js

이 예제에서는 load 함수를 사용하여 이진 변환을 수행합니다.

```
let proofValue: dom.Value = load(revisionResponse.Proof.IonText);
```

Python

이 예제에서는 loads 함수를 사용하여 이진 변환을 수행합니다.

```
proof_text = proof_response.get('Proof').get('IonText')
proof_hashes = loads(proof_text)
```

4단계: 수정본에서 다이제스트 다시 계산하기

증명의 해시 목록을 사용하여 수정 해시부터 시작하여 다이제스트를 다시 계산합니다. 이전에 저장된 다이제스트가 QLDB 외부에서 알려져 있고 신뢰할 수 있는 한, 다시 계산된 다이제스트 해시가 저장된 다이제스트 해시와 일치하면 문서 수정본의 무결성이 입증됩니다.

Java

```
// Calculate digest
byte[] calculatedDigest = internalHashes.stream().reduce(hash.getBytes(),
 BlockHashVerification::dot);

boolean verified = Arrays.equals(expectedDigest, calculatedDigest);

if (verified) {
    System.out.printf("Successfully verified document revision for id '%s'!%n",
 metadataId);
} else {
    System.out.printf("Document revision for id '%s' verification failed!%n",
 metadataId);
    return;
}
```

.NET

```
byte[] documentHash = hash.Bytes().ToArray();
foreach (IIonValue proofHash in proofValue.GetElementAt(0))
{
    // Calculate the digest
    documentHash = Dot(documentHash, proofHash.Bytes().ToArray());
}

bool verified = expectedDigest.SequenceEqual(documentHash);

if (verified)
{
    Console.WriteLine($"Successfully verified document revision for id
 '{metadataId}'!");
}
else
{
    Console.WriteLine($"Document revision for id '{metadataId}' verification
 failed!");
    return;
}
```

Go

```
// Going through nodes and calculate digest
```

```
for reader.Next() {
    val, _ := reader.ByteValue()
    documentHash, err = dot(documentHash, val)
}

// Compare documentHash with the expected digest
verified := reflect.DeepEqual(documentHash, expectedDigest)

if verified {
    fmt.Printf("Successfully verified document revision for id '%s'!\n", metadataId)
} else {
    fmt.Printf("Document revision for id '%s' verification failed!\n", metadataId)
    return
}
```

Node.js

```
let documentHash: Uint8Array = hash.uInt8ArrayValue();
proofValue.elements().forEach((proofHash: dom.Value) => {
    // Calculate the digest
    documentHash = dot(documentHash, proofHash.uInt8ArrayValue());
});

let verified: boolean = isEqual(expectedDigest, documentHash);

if (verified) {
    console.log(`Successfully verified document revision for id '${metadataId}'!`);
} else {
    console.log(`Document revision for id '${metadataId}' verification failed!`);
    return;
}
```

Python

```
# Calculate digest
calculated_digest = reduce(dot, proof_hashes, document_hash)

verified = calculated_digest == expected_digest
if verified:
    print("Successfully verified document revision for id
 '{}'!".format(metadata_id))
else:
    print("Document revision for id '{}' verification failed!".format(metadata_id))
```

5단계: 저널 블록에 대한 증명 요청

다음으로, 문서 수정본이 포함된 저널 블록을 확인합니다.

1단계에서 저장한 다이제스트의 블록 주소와 팁 주소를 사용하여 `GetBlock` 요청을 제출합니다. 2단계의 `GetRevision` 요청과 마찬가지로, 저장된 다이제스트의 팁 주소를 다시 입력해야 블록에 대한 증명을 받을 수 있습니다. 이 API 작업은 블록과 블록 증명이 포함된 객체를 반환합니다.

저널 블록 구조 및 콘텐츠에 대한 자세한 콘텐츠는 Amazon QLDB의 저널 콘텐츠 섹션을 참조하세요.

Java

```
// Submit a request for the block
GetBlockRequest getBlockRequest = new GetBlockRequest()
        .withName(ledgerName)
        .withBlockAddress(new ValueHolder().withIonText(blockAddressText))
        .withDigestTipAddress(digestResult.getDigestTipAddress());

// Get a result back
GetBlockResult getBlockResult = client.getBlock(getBlockRequest);
```

.NET

```
// Submit a request for the block
GetBlockRequest getBlockRequest = new GetBlockRequest
{
    Name = ledgerName,
    BlockAddress = new ValueHolder
    {
        IonText = blockAddressText
    },
    DigestTipAddress = getDigestResponse.DigestTipAddress
};

// Get a response back
GetBlockResponse getBlockResponse = client.GetBlockAsync(getBlockRequest).Result;
```

Go

```
// Submit a request for the block
blockInput := qldb.GetBlockInput{
    Name:            &currentLedgerName,
    BlockAddress:    &qldb.ValueHolder{IonText: &blockAddress},
```

```
    DigestTipAddress: digestOutput.DigestTipAddress,
}

// Get a result back
blockOutput, err := client.GetBlock(&blockInput)
if err != nil {
    panic(err)
}
```

Node.js

```
// Submit a request for the block
const getBlockRequest: GetBlockRequest = {
    Name: ledgerName,
    BlockAddress: {
        IonText: dumpText(blockAddress)
    },
    DigestTipAddress: getDigestResponse.DigestTipAddress
};

// Get a response back
const getBlockResponse: GetBlockResponse = await
 qldbClient.getBlock(getBlockRequest).promise();
```

Python

```
def block_address_to_dictionary(ion_dict):
"""
Convert a block address from IonPyDict into a dictionary.
Shape of the dictionary must be: {'IonText': "{strandId: <"strandId">, sequenceNo:
 <sequenceNo>}"}

:type ion_dict: :py:class:`amazon.ion.simple_types.IonPyDict`/str
:param ion_dict: The block address value to convert.

:rtype: dict
:return: The converted dict.
"""
block_address = {'IonText': {}}
if not isinstance(ion_dict, str):
    py_dict = '{{strandId: "{}", sequenceNo:{}}}'.format(ion_dict['strandId'],
 ion_dict['sequenceNo'])
    ion_dict = py_dict
```

```
block_address['IonText'] = ion_dict
return block_address

# Submit a request for the block and get a result back
block_response = qldb_client.get_block(Name=ledger_name,
 BlockAddress=block_address_to_dictionary(block_address),
                                        DigestTipAddress=digest_tip_address)
```

그런 다음 결과에서 블록 해시와 증명을 검색합니다.

Java

이 예제에서는 IonLoader를 사용하여 블록 객체를 IonDatagram 컨테이너에 로드합니다.

```
String blockText = getBlockResult.getBlock().getIonText();

IonDatagram datagram = SYSTEM.getLoader().load(blockText);
ionStruct = (IonStruct)datagram.get(0);

final byte[] blockHash = ((IonBlob)ionStruct.get("blockHash")).getBytes();
```

IonLoader를 사용하여 증명을 IonDatagram에 로드할 수도 있습니다.

```
proofText = getBlockResult.getProof().getIonText();

// Take the proof and create a list of hash binary data
datagram = SYSTEM.getLoader().load(proofText);
ListIterator<IonValue> listIter =
 ((IonList)datagram.iterator().next()).listIterator();

internalHashes.clear();
while (listIter.hasNext()) {
    internalHashes.add(((IonBlob)listIter.next()).getBytes());
}
```

.NET

이 예제에서는 IonLoader를 사용하여 블록과 증명을 각각의 Ion 데이터그램에 로드합니다.

```
string blockText = getBlockResponse.Block.IonText;
IIonDatagram blockValue = IonLoader.Default.Load(blockText);
```

```
// blockValue is a IonDatagram, and the first value is an IonStruct containing the
 blockHash
byte[] blockHash =
 blockValue.GetElementAt(0).GetField("blockHash").Bytes().ToArray();

proofText = getBlockResponse.Proof.IonText;
proofValue = IonLoader.Default.Load(proofText);
```

Go

이 예제에서는 Ion 리더를 사용하여 증명을 이진수로 변환하고 증명의 노드 해시 목록을 반복해서 살펴봅니다.

```
proofText = blockOutput.Proof.IonText

block := new(map[string]interface{})
err = ion.UnmarshalString(*blockOutput.Block.IonText, block)
if err != nil {
    panic(err)
}

blockHash := (*block)["blockHash"].([]byte)

// Use ion.Reader to iterate over the proof's node hashes
reader = ion.NewReaderString(*proofText)
// Enter the struct containing node hashes
reader.Next()
if err := reader.StepIn(); err != nil {
    panic(err)
}
```

Node.js

이 예제에서는 `load` 함수를 사용하여 블록과 증명을 이진수로 변환합니다.

```
const blockValue: dom.Value = load(getBlockResponse.Block.IonText)
let blockHash: Uint8Array = blockValue.get("blockHash").uInt8ArrayValue();

proofValue = load(getBlockResponse.Proof.IonText);
```

Python

이 예제에서는 `loads` 함수를 사용하여 블록과 증명을 이진수로 변환합니다.

```
block_text = block_response.get('Block').get('IonText')
block = loads(block_text)

block_hash = block.get('blockHash')

proof_text = block_response.get('Proof').get('IonText')
proof_hashes = loads(proof_text)
```

6단계: 블록의 다이제스트 다시 계산하기

증명의 해시 목록을 사용하여 블록 해시부터 시작하여 다이제스트를 다시 계산합니다. 이전에 저장된 다이제스트가 QLDB 외부에서 알려져 있고 신뢰할 수 있는 한, 다시 계산된 다이제스트 해시가 저장된 다이제스트 해시와 일치하면 블록의 무결성이 입증됩니다.

Java

```
// Calculate digest
calculatedDigest = internalHashes.stream().reduce(blockHash,
 BlockHashVerification::dot);

verified = Arrays.equals(expectedDigest, calculatedDigest);

if (verified) {
    System.out.printf("Block address '%s' successfully verified!%n",
 blockAddressText);
} else {
    System.out.printf("Block address '%s' verification failed!%n",
 blockAddressText);
}
```

.NET

```
foreach (IIonValue proofHash in proofValue.GetElementAt(0))
{
    // Calculate the digest
    blockHash = Dot(blockHash, proofHash.Bytes().ToArray());
}

verified = expectedDigest.SequenceEqual(blockHash);

if (verified)
```

```
{
    Console.WriteLine($"Block address '{blockAddressText}' successfully verified!");
}
else
{
    Console.WriteLine($"Block address '{blockAddressText}' verification failed!");
}
```

Go

```
// Going through nodes and calculate digest
for reader.Next() {
    val, err := reader.ByteValue()
    if err != nil {
        panic(err)
    }
    blockHash, err = dot(blockHash, val)
}

// Compare blockHash with the expected digest
verified = reflect.DeepEqual(blockHash, expectedDigest)

if verified {
    fmt.Printf("Block address '%s' successfully verified!\n", blockAddress)
} else {
    fmt.Printf("Block address '%s' verification failed!\n", blockAddress)
    return
}
```

Node.js

```
proofValue.elements().forEach((proofHash: dom.Value) => {
    // Calculate the digest
    blockHash = dot(blockHash, proofHash.uInt8ArrayValue());
});

verified = isEqual(expectedDigest, blockHash);

if (verified) {
    console.log(`Block address '${dumpText(blockAddress)}' successfully verified!`);
} else {
    console.log(`Block address '${dumpText(blockAddress)}' verification failed!`);
}
```

Python

```
# Calculate digest
calculated_digest = reduce(dot, proof_hashes, block_hash)

verified = calculated_digest == expected_digest
if verified:
    print("Block address '{}' successfully verified!".format(dumps(block_address,
                                                                   binary=False,

 omit_version_marker=True)))
else:
    print("Block address '{}' verification failed!".format(block_address))
```

이전 코드 예제에서는 다이제스트를 다시 계산할 때 dot 함수를 사용합니다. 이 함수는 두 해시를 입력받아 두 해시를 정렬하고 연결한 다음 연결된 배열의 해시를 반환합니다.

Java

```
/**
 * Takes two hashes, sorts them, concatenates them, and then returns the
 * hash of the concatenated array.
 *
 * @param h1
 *              Byte array containing one of the hashes to compare.
 * @param h2
 *              Byte array containing one of the hashes to compare.
 * @return the concatenated array of hashes.
 */
public static byte[] dot(final byte[] h1, final byte[] h2) {
    if (h1.length != HASH_LENGTH || h2.length != HASH_LENGTH) {
        throw new IllegalArgumentException("Invalid hash.");
    }

    int byteEqual = 0;
    for (int i = h1.length - 1; i >= 0; i--) {
        byteEqual = Byte.compare(h1[i], h2[i]);
        if (byteEqual != 0) {
            break;
        }
    }
```

```
    byte[] concatenated = new byte[h1.length + h2.length];
    if (byteEqual < 0) {
        System.arraycopy(h1, 0, concatenated, 0, h1.length);
        System.arraycopy(h2, 0, concatenated, h1.length, h2.length);
    } else {
        System.arraycopy(h2, 0, concatenated, 0, h2.length);
        System.arraycopy(h1, 0, concatenated, h2.length, h1.length);
    }

    MessageDigest messageDigest;
    try {
        messageDigest = MessageDigest.getInstance("SHA-256");
    } catch (NoSuchAlgorithmException e) {
        throw new IllegalStateException("SHA-256 message digest is unavailable", e);
    }

    messageDigest.update(concatenated);
    return messageDigest.digest();
}
```

.NET

```
/// <summary>
/// Takes two hashes, sorts them, concatenates them, and then returns the
/// hash of the concatenated array.
/// </summary>
/// <param name="h1">Byte array containing one of the hashes to compare.</param>
/// <param name="h2">Byte array containing one of the hashes to compare.</param>
/// <returns>The concatenated array of hashes.</returns>
private static byte[] Dot(byte[] h1, byte[] h2)
{
    if (h1.Length == 0)
    {
        return h2;
    }

    if (h2.Length == 0)
    {
        return h1;
    }

    HashAlgorithm hashAlgorithm = HashAlgorithm.Create("SHA256");
    HashComparer comparer = new HashComparer();
```

```
    if (comparer.Compare(h1, h2) < 0)
    {
        return hashAlgorithm.ComputeHash(h1.Concat(h2).ToArray());
    }
    else
    {
        return hashAlgorithm.ComputeHash(h2.Concat(h1).ToArray());
    }
}

private class HashComparer : IComparer<byte[]>
{
    private static readonly int HASH_LENGTH = 32;

    public int Compare(byte[] h1, byte[] h2)
    {
        if (h1.Length != HASH_LENGTH || h2.Length != HASH_LENGTH)
        {
            throw new ArgumentException("Invalid hash");
        }

        for (var i = h1.Length - 1; i >= 0; i--)
        {
            var byteEqual = (sbyte)h1[i] - (sbyte)h2[i];
            if (byteEqual != 0)
            {
                return byteEqual;
            }
        }

        return 0;
    }
}
```

Go

```
// Takes two hashes, sorts them, concatenates them, and then returns the hash of the
 concatenated array.
func dot(h1, h2 []byte) ([]byte, error) {
    compare, err := hashComparator(h1, h2)
    if err != nil {
        return nil, err
    }
```

```
    var concatenated []byte
    if compare < 0 {
        concatenated = append(h1, h2...)
    } else {
        concatenated = append(h2, h1...)
    }

    newHash := sha256.Sum256(concatenated)
    return newHash[:], nil
}

func hashComparator(h1 []byte, h2 []byte) (int16, error) {
    if len(h1) != hashLength || len(h2) != hashLength {
        return 0, errors.New("invalid hash")
    }
    for i := range h1 {
        // Reverse index for little endianness
        index := hashLength - 1 - i

        // Handle byte being unsigned and overflow
        h1Int := int16(h1[index])
        h2Int := int16(h2[index])
        if h1Int > 127 {
            h1Int = 0 - (256 - h1Int)
        }
        if h2Int > 127 {
            h2Int = 0 - (256 - h2Int)
        }

        difference := h1Int - h2Int
        if difference != 0 {
            return difference, nil
        }
    }
    return 0, nil
}
```

Node.js

```
/**
 * Takes two hashes, sorts them, concatenates them, and calculates a digest based on
 the concatenated hash.
```

```
 * @param h1 Byte array containing one of the hashes to compare.
 * @param h2 Byte array containing one of the hashes to compare.
 * @returns The digest calculated from the concatenated hash values.
 */
function dot(h1: Uint8Array, h2: Uint8Array): Uint8Array {
    if (h1.length === 0) {
        return h2;
    }
    if (h2.length === 0) {
        return h1;
    }

    const newHashLib = createHash("sha256");

    let concatenated: Uint8Array;
    if (hashComparator(h1, h2) < 0) {
        concatenated = concatenate(h1, h2);
    } else {
        concatenated = concatenate(h2, h1);
    }
    newHashLib.update(concatenated);
    return newHashLib.digest();
}

/**
 * Compares two hashes by their **signed** byte values in little-endian order.
 * @param hash1 The hash value to compare.
 * @param hash2 The hash value to compare.
 * @returns Zero if the hash values are equal, otherwise return the difference of
 the first pair of non-matching
 *          bytes.
 * @throws RangeError When the hash is not the correct hash size.
 */
function hashComparator(hash1: Uint8Array, hash2: Uint8Array): number {
    if (hash1.length !== HASH_SIZE || hash2.length !== HASH_SIZE) {
        throw new RangeError("Invalid hash.");
    }
    for (let i = hash1.length-1; i >= 0; i--) {
        const difference: number = (hash1[i]<<24 >>24) - (hash2[i]<<24 >>24);
        if (difference !== 0) {
            return difference;
        }
    }
    return 0;
```

```
}

/**
 * Helper method that concatenates two Uint8Array.
 * @param arrays List of arrays to concatenate, in the order provided.
 * @returns The concatenated array.
 */
function concatenate(...arrays: Uint8Array[]): Uint8Array {
    let totalLength = 0;
    for (const arr of arrays) {
        totalLength += arr.length;
    }
    const result = new Uint8Array(totalLength);
    let offset = 0;
    for (const arr of arrays) {
        result.set(arr, offset);
        offset += arr.length;
    }
    return result;
}

/**
 * Helper method that checks for equality between two Uint8Array.
 * @param expected Byte array containing one of the hashes to compare.
 * @param actual Byte array containing one of the hashes to compare.
 * @returns Boolean indicating equality between the two Uint8Array.
 */
function isEqual(expected: Uint8Array, actual: Uint8Array): boolean {
    if (expected === actual) return true;
    if (expected == null || actual == null) return false;
    if (expected.length !== actual.length) return false;

    for (let i = 0; i < expected.length; i++) {
        if (expected[i] !== actual[i]) {
            return false;
        }
    }
    return true;
}
```

Python

```
def dot(hash1, hash2):
```

```
    """
    Takes two hashes, sorts them, concatenates them, and then returns the
    hash of the concatenated array.

    :type hash1: bytes
    :param hash1: The hash value to compare.

    :type hash2: bytes
    :param hash2: The hash value to compare.

    :rtype: bytes
    :return: The new hash value generated from concatenated hash values.
    """
    if len(hash1) != hash_length or len(hash2) != hash_length:
        raise ValueError('Illegal hash.')

    hash_array1 = array('b', hash1)
    hash_array2 = array('b', hash2)

    difference = 0
    for i in range(len(hash_array1) - 1, -1, -1):
        difference = hash_array1[i] - hash_array2[i]
        if difference != 0:
            break

    if difference < 0:
        concatenated = hash1 + hash2
    else:
        concatenated = hash2 + hash1

    new_hash_lib = sha256()
    new_hash_lib.update(concatenated)
    new_digest = new_hash_lib.digest()
    return new_digest
```

전체 코드 예제 실행

다음과 같이 전체 코드 예제를 실행하여 이전 단계를 처음부터 끝까지 모두 수행합니다.

Java

```
import com.amazon.ion.IonBlob;
```

```
import com.amazon.ion.IonDatagram;
import com.amazon.ion.IonList;
import com.amazon.ion.IonReader;
import com.amazon.ion.IonString;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSystem;
import com.amazon.ion.IonValue;
import com.amazon.ion.system.IonSystemBuilder;
import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.AmazonQLDBClientBuilder;
import com.amazonaws.services.qldb.model.GetBlockRequest;
import com.amazonaws.services.qldb.model.GetBlockResult;
import com.amazonaws.services.qldb.model.GetDigestRequest;
import com.amazonaws.services.qldb.model.GetDigestResult;
import com.amazonaws.services.qldb.model.GetRevisionRequest;
import com.amazonaws.services.qldb.model.GetRevisionResult;
import com.amazonaws.services.qldb.model.ValueHolder;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.ListIterator;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.qldbsession.QldbSessionClient;
import software.amazon.awssdk.services.qldbsession.QldbSessionClientBuilder;
import software.amazon.qldb.QldbDriver;
import software.amazon.qldb.Result;

public class BlockHashVerification {
    private static final IonSystem SYSTEM = IonSystemBuilder.standard().build();
    private static final QldbDriver driver = createQldbDriver();
    private static final AmazonQLDB client =
 AmazonQLDBClientBuilder.standard().build();
    private static final String region = "us-east-1";
    private static final String ledgerName = "vehicle-registration";
    private static final String tableName = "VehicleRegistration";
    private static final String vin = "KM8SRDHF6EU074761";
    private static final int HASH_LENGTH = 32;

    /**
     * Create a pooled driver for creating sessions.
     *
```

```
     * @return The pooled driver for creating sessions.
     */
    public static QldbDriver createQldbDriver() {
        QldbSessionClientBuilder sessionClientBuilder = QldbSessionClient.builder();
        sessionClientBuilder.region(Region.of(region));

        return QldbDriver.builder()
                .ledger(ledgerName)
                .sessionClientBuilder(sessionClientBuilder)
                .build();
    }

    /**
     * Takes two hashes, sorts them, concatenates them, and then returns the
     * hash of the concatenated array.
     *
     * @param h1
     *              Byte array containing one of the hashes to compare.
     * @param h2
     *              Byte array containing one of the hashes to compare.
     * @return the concatenated array of hashes.
     */
    public static byte[] dot(final byte[] h1, final byte[] h2) {
        if (h1.length != HASH_LENGTH || h2.length != HASH_LENGTH) {
            throw new IllegalArgumentException("Invalid hash.");
        }

        int byteEqual = 0;
        for (int i = h1.length - 1; i >= 0; i--) {
            byteEqual = Byte.compare(h1[i], h2[i]);
            if (byteEqual != 0) {
                break;
            }
        }

        byte[] concatenated = new byte[h1.length + h2.length];
        if (byteEqual < 0) {
            System.arraycopy(h1, 0, concatenated, 0, h1.length);
            System.arraycopy(h2, 0, concatenated, h1.length, h2.length);
        } else {
            System.arraycopy(h2, 0, concatenated, 0, h2.length);
            System.arraycopy(h1, 0, concatenated, h2.length, h1.length);
        }
```

```
        MessageDigest messageDigest;
        try {
            messageDigest = MessageDigest.getInstance("SHA-256");
        } catch (NoSuchAlgorithmException e) {
            throw new IllegalStateException("SHA-256 message digest is unavailable",
e);
        }

        messageDigest.update(concatenated);
        return messageDigest.digest();
    }

    public static void main(String[] args) {
        // Get a digest
        GetDigestRequest digestRequest = new
GetDigestRequest().withName(ledgerName);
        GetDigestResult digestResult = client.getDigest(digestRequest);

        java.nio.ByteBuffer digest = digestResult.getDigest();

        // expectedDigest is the buffer we will use later to compare against our
calculated digest
        byte[] expectedDigest = new byte[digest.remaining()];
        digest.get(expectedDigest);

        // Retrieve info for the given vin's document revisions
        Result result = driver.execute(txn -> {
            final String query = String.format("SELECT blockAddress, hash,
metadata.id FROM _ql_committed_%s WHERE data.VIN = '%s'", tableName, vin);
            return txn.execute(query);
        });

        System.out.printf("Verifying document revisions for vin '%s' in table '%s'
in ledger '%s'%n", vin, tableName, ledgerName);

        for (IonValue ionValue : result) {
            IonStruct ionStruct = (IonStruct)ionValue;

            // Get the requested fields
            IonValue blockAddress = ionStruct.get("blockAddress");
            IonBlob hash = (IonBlob)ionStruct.get("hash");
            String metadataId = ((IonString)ionStruct.get("id")).stringValue();
```

```
            System.out.printf("Verifying document revision for id '%s'%n",
metadataId);

            String blockAddressText = blockAddress.toString();

            // Submit a request for the revision
            GetRevisionRequest revisionRequest = new GetRevisionRequest()
                    .withName(ledgerName)
                    .withBlockAddress(new
ValueHolder().withIonText(blockAddressText))
                    .withDocumentId(metadataId)
                    .withDigestTipAddress(digestResult.getDigestTipAddress());

            // Get a result back
            GetRevisionResult revisionResult = client.getRevision(revisionRequest);

            String proofText = revisionResult.getProof().getIonText();

            // Take the proof and convert it to a list of byte arrays
            List<byte[]> internalHashes = new ArrayList<>();
            IonReader reader = SYSTEM.newReader(proofText);
            reader.next();
            reader.stepIn();
            while (reader.next() != null) {
                internalHashes.add(reader.newBytes());
            }

            // Calculate digest
            byte[] calculatedDigest =
internalHashes.stream().reduce(hash.getBytes(), BlockHashVerification::dot);

            boolean verified = Arrays.equals(expectedDigest, calculatedDigest);

            if (verified) {
                System.out.printf("Successfully verified document revision for id
'%s'!%n", metadataId);
            } else {
                System.out.printf("Document revision for id '%s' verification
failed!%n", metadataId);
                return;
            }

            // Submit a request for the block
            GetBlockRequest getBlockRequest = new GetBlockRequest()
```

```
                    .withName(ledgerName)
                    .withBlockAddress(new
ValueHolder().withIonText(blockAddressText))
                    .withDigestTipAddress(digestResult.getDigestTipAddress());

            // Get a result back
            GetBlockResult getBlockResult = client.getBlock(getBlockRequest);

            String blockText = getBlockResult.getBlock().getIonText();

            IonDatagram datagram = SYSTEM.getLoader().load(blockText);
            ionStruct = (IonStruct)datagram.get(0);

            final byte[] blockHash =
((IonBlob)ionStruct.get("blockHash")).getBytes();

            proofText = getBlockResult.getProof().getIonText();

            // Take the proof and create a list of hash binary data
            datagram = SYSTEM.getLoader().load(proofText);
            ListIterator<IonValue> listIter =
((IonList)datagram.iterator().next()).listIterator();

            internalHashes.clear();
            while (listIter.hasNext()) {
                internalHashes.add(((IonBlob)listIter.next()).getBytes());
            }

            // Calculate digest
            calculatedDigest = internalHashes.stream().reduce(blockHash,
BlockHashVerification::dot);

            verified = Arrays.equals(expectedDigest, calculatedDigest);

            if (verified) {
                System.out.printf("Block address '%s' successfully verified!%n",
blockAddressText);
            } else {
                System.out.printf("Block address '%s' verification failed!%n",
blockAddressText);
            }
        }
    }
```

```
}
```

.NET

```
using Amazon.IonDotnet;
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;
using Amazon.QLDB;
using Amazon.QLDB.Driver;
using Amazon.QLDB.Model;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.Cryptography;

namespace BlockHashVerification
{
    class BlockHashVerification
    {
        private static readonly string ledgerName = "vehicle-registration";
        private static readonly string tableName = "VehicleRegistration";
        private static readonly string vin = "KM8SRDHF6EU074761";
        private static readonly IQldbDriver driver =
 QldbDriver.Builder().WithLedger(ledgerName).Build();
        private static readonly IAmazonQLDB client = new AmazonQLDBClient();

        /// <summary>
        /// Takes two hashes, sorts them, concatenates them, and then returns the
        /// hash of the concatenated array.
        /// </summary>
        /// <param name="h1">Byte array containing one of the hashes to compare.</
param>
        /// <param name="h2">Byte array containing one of the hashes to compare.</
param>
        /// <returns>The concatenated array of hashes.</returns>
        private static byte[] Dot(byte[] h1, byte[] h2)
        {
            if (h1.Length == 0)
            {
                return h2;
            }
```

```
            if (h2.Length == 0)
            {
                return h1;
            }

            HashAlgorithm hashAlgorithm = HashAlgorithm.Create("SHA256");
            HashComparer comparer = new HashComparer();
            if (comparer.Compare(h1, h2) < 0)
            {
                return hashAlgorithm.ComputeHash(h1.Concat(h2).ToArray());
            }
            else
            {
                return hashAlgorithm.ComputeHash(h2.Concat(h1).ToArray());
            }
        }

        private class HashComparer : IComparer<byte[]>
        {
            private static readonly int HASH_LENGTH = 32;

            public int Compare(byte[] h1, byte[] h2)
            {
                if (h1.Length != HASH_LENGTH || h2.Length != HASH_LENGTH)
                {
                    throw new ArgumentException("Invalid hash");
                }

                for (var i = h1.Length - 1; i >= 0; i--)
                {
                    var byteEqual = (sbyte)h1[i] - (sbyte)h2[i];
                    if (byteEqual != 0)
                    {
                        return byteEqual;
                    }
                }

                return 0;
            }
        }

        static void Main()
        {
            // Get a digest
```

```
            GetDigestRequest getDigestRequest = new GetDigestRequest
            {
                Name = ledgerName
            };
            GetDigestResponse getDigestResponse =
client.GetDigestAsync(getDigestRequest).Result;

            // expectedDigest is the buffer we will use later to compare against our
calculated digest
            MemoryStream digest = getDigestResponse.Digest;
            byte[] expectedDigest = digest.ToArray();

            // Retrieve info for the given vin's document revisions
            var result = driver.Execute(txn => {
                string query = $"SELECT blockAddress, hash, metadata.id FROM
_ql_committed_{tableName} WHERE data.VIN = '{vin}'";
                return txn.Execute(query);
            });

            Console.WriteLine($"Verifying document revisions for vin '{vin}' in
table '{tableName}' in ledger '{ledgerName}'");

            foreach (IIonValue ionValue in result)
            {
                IIonStruct ionStruct = ionValue;

                // Get the requested fields
                IIonValue blockAddress = ionStruct.GetField("blockAddress");
                IIonBlob hash = ionStruct.GetField("hash");
                String metadataId = ionStruct.GetField("id").StringValue;

                Console.WriteLine($"Verifying document revision for id
'{metadataId}'");

                // Use an Ion Reader to convert block address to text
                IIonReader reader = IonReaderBuilder.Build(blockAddress);
                StringWriter sw = new StringWriter();
                IIonWriter textWriter = IonTextWriterBuilder.Build(sw);
                textWriter.WriteValues(reader);
                string blockAddressText = sw.ToString();

                // Submit a request for the revision
                GetRevisionRequest revisionRequest = new GetRevisionRequest
                {
```

```
                    Name = ledgerName,
                    BlockAddress = new ValueHolder
                    {
                        IonText = blockAddressText
                    },
                    DocumentId = metadataId,
                    DigestTipAddress = getDigestResponse.DigestTipAddress
                };

                // Get a response back
                GetRevisionResponse revisionResponse =
client.GetRevisionAsync(revisionRequest).Result;

                string proofText = revisionResponse.Proof.IonText;
                IIonDatagram proofValue = IonLoader.Default.Load(proofText);

                byte[] documentHash = hash.Bytes().ToArray();
                foreach (IIonValue proofHash in proofValue.GetElementAt(0))
                {
                    // Calculate the digest
                    documentHash = Dot(documentHash, proofHash.Bytes().ToArray());
                }

                bool verified = expectedDigest.SequenceEqual(documentHash);

                if (verified)
                {
                    Console.WriteLine($"Successfully verified document revision for
id '{metadataId}'!");
                }
                else
                {
                    Console.WriteLine($"Document revision for id '{metadataId}'
verification failed!");
                    return;
                }

                // Submit a request for the block
                GetBlockRequest getBlockRequest = new GetBlockRequest
                {
                    Name = ledgerName,
                    BlockAddress = new ValueHolder
                    {
                        IonText = blockAddressText
```

```
                    },
                    DigestTipAddress = getDigestResponse.DigestTipAddress
                };

                // Get a response back
                GetBlockResponse getBlockResponse =
 client.GetBlockAsync(getBlockRequest).Result;

                string blockText = getBlockResponse.Block.IonText;
                IIonDatagram blockValue = IonLoader.Default.Load(blockText);

                // blockValue is a IonDatagram, and the first value is an IonStruct
 containing the blockHash
                byte[] blockHash =
 blockValue.GetElementAt(0).GetField("blockHash").Bytes().ToArray();

                proofText = getBlockResponse.Proof.IonText;
                proofValue = IonLoader.Default.Load(proofText);

                foreach (IIonValue proofHash in proofValue.GetElementAt(0))
                {
                    // Calculate the digest
                    blockHash = Dot(blockHash, proofHash.Bytes().ToArray());
                }

                verified = expectedDigest.SequenceEqual(blockHash);

                if (verified)
                {
                    Console.WriteLine($"Block address '{blockAddressText}'
 successfully verified!");
                }
                else
                {
                    Console.WriteLine($"Block address '{blockAddressText}'
 verification failed!");
                }
            }
        }
    }
}
```

Go

```
package main

import (
    "context"
    "crypto/sha256"
    "errors"
    "fmt"
    "reflect"

    "github.com/amzn/ion-go/ion"
    "github.com/aws/aws-sdk-go/aws"
    AWSSession "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/qldb"
    "github.com/aws/aws-sdk-go/service/qldbsession"
    "github.com/awslabs/amazon-qldb-driver-go/qldbdriver"
)

const (
    hashLength = 32
    ledgerName = "vehicle-registration"
    tableName  = "VehicleRegistration"
    vin        = "KM8SRDHF6EU074761"
)

// Takes two hashes, sorts them, concatenates them, and then returns the hash of the
 concatenated array.
func dot(h1, h2 []byte) ([]byte, error) {
    compare, err := hashComparator(h1, h2)
    if err != nil {
        return nil, err
    }

    var concatenated []byte
    if compare < 0 {
        concatenated = append(h1, h2...)
    } else {
        concatenated = append(h2, h1...)
    }

    newHash := sha256.Sum256(concatenated)
    return newHash[:], nil
}
```

```
func hashComparator(h1 []byte, h2 []byte) (int16, error) {
    if len(h1) != hashLength || len(h2) != hashLength {
        return 0, errors.New("invalid hash")
    }
    for i := range h1 {
        // Reverse index for little endianness
        index := hashLength - 1 - i

        // Handle byte being unsigned and overflow
        h1Int := int16(h1[index])
        h2Int := int16(h2[index])
        if h1Int > 127 {
            h1Int = 0 - (256 - h1Int)
        }
        if h2Int > 127 {
            h2Int = 0 - (256 - h2Int)
        }

        difference := h1Int - h2Int
        if difference != 0 {
            return difference, nil
        }
    }
    return 0, nil
}

func main() {
    driverSession := AWSSession.Must(AWSSession.NewSession(aws.NewConfig()))
    qldbSession := qldbsession.New(driverSession)
    driver, err := qldbdriver.New(ledgerName, qldbSession, func(options
 *qldbdriver.DriverOptions) {})
    if err != nil {
        panic(err)
    }
    client := qldb.New(driverSession)

    // Get a digest
    currentLedgerName := ledgerName
    input := qldb.GetDigestInput{Name: &currentLedgerName}
    digestOutput, err := client.GetDigest(&input)
    if err != nil {
        panic(err)
    }
```

```
   // expectedDigest is the buffer we will later use to compare against our
calculated digest
   expectedDigest := digestOutput.Digest

   // Retrieve info for the given vin's document revisions
   result, err := driver.Execute(context.Background(), func(txn
qldbdriver.Transaction) (interface{}, error) {
       statement := fmt.Sprintf(
               "SELECT blockAddress, hash, metadata.id FROM _ql_committed_%s WHERE
data.VIN = '%s'",
               tableName,
               vin)
       result, err := txn.Execute(statement)
       if err != nil {
           return nil, err
       }

       results := make([]map[string]interface{}, 0)

       // Convert the result set into a map
       for result.Next(txn) {
           var doc map[string]interface{}
           err := ion.Unmarshal(result.GetCurrentData(), &doc)
           if err != nil {
               return nil, err
           }
           results = append(results, doc)
       }
       return results, nil
   })
   if err != nil {
       panic(err)
   }
   resultSlice := result.([]map[string]interface{})

   fmt.Printf("Verifying document revisions for vin '%s' in table '%s' in ledger
'%s'\n", vin, tableName, ledgerName)

   for _, value := range resultSlice {
       // Get the requested fields
       ionBlockAddress, err := ion.MarshalText(value["blockAddress"])
       if err != nil {
           panic(err)
```

```
        }
        blockAddress := string(ionBlockAddress)
        metadataId := value["id"].(string)
        documentHash := value["hash"].([]byte)

        fmt.Printf("Verifying document revision for id '%s'\n", metadataId)

        // Submit a request for the revision
        revisionInput := qldb.GetRevisionInput{
            BlockAddress:     &qldb.ValueHolder{IonText: &blockAddress},
            DigestTipAddress: digestOutput.DigestTipAddress,
            DocumentId:       &metadataId,
            Name:             &currentLedgerName,
        }

        // Get a result back
        revisionOutput, err := client.GetRevision(&revisionInput)
        if err != nil {
            panic(err)
        }

        proofText := revisionOutput.Proof.IonText

        // Use ion.Reader to iterate over the proof's node hashes
        reader := ion.NewReaderString(*proofText)
        // Enter the struct containing node hashes
        reader.Next()
        if err := reader.StepIn(); err != nil {
            panic(err)
        }

        // Going through nodes and calculate digest
        for reader.Next() {
            val, _ := reader.ByteValue()
            documentHash, err = dot(documentHash, val)
        }

        // Compare documentHash with the expected digest
        verified := reflect.DeepEqual(documentHash, expectedDigest)

        if verified {
            fmt.Printf("Successfully verified document revision for id '%s'!\n",
metadataId)
        } else {
```

```
            fmt.Printf("Document revision for id '%s' verification failed!\n",
metadataId)
            return
        }

        // Submit a request for the block
        blockInput := qldb.GetBlockInput{
            Name:             &currentLedgerName,
            BlockAddress:     &qldb.ValueHolder{IonText: &blockAddress},
            DigestTipAddress: digestOutput.DigestTipAddress,
        }

        // Get a result back
        blockOutput, err := client.GetBlock(&blockInput)
        if err != nil {
            panic(err)
        }

        proofText = blockOutput.Proof.IonText

        block := new(map[string]interface{})
        err = ion.UnmarshalString(*blockOutput.Block.IonText, block)
        if err != nil {
            panic(err)
        }

        blockHash := (*block)["blockHash"].([]byte)

        // Use ion.Reader to iterate over the proof's node hashes
        reader = ion.NewReaderString(*proofText)
        // Enter the struct containing node hashes
        reader.Next()
        if err := reader.StepIn(); err != nil {
            panic(err)
        }

        // Going through nodes and calculate digest
        for reader.Next() {
            val, err := reader.ByteValue()
            if err != nil {
                panic(err)
            }
            blockHash, err = dot(blockHash, val)
        }
```

```
        // Compare blockHash with the expected digest
        verified = reflect.DeepEqual(blockHash, expectedDigest)

        if verified {
            fmt.Printf("Block address '%s' successfully verified!\n", blockAddress)
        } else {
            fmt.Printf("Block address '%s' verification failed!\n", blockAddress)
            return
        }
    }
}
```

Node.js

```
import { QldbDriver, Result, TransactionExecutor} from "amazon-qldb-driver-nodejs";
import { QLDB } from "aws-sdk"
import { GetBlockRequest, GetBlockResponse, GetDigestRequest, GetDigestResponse,
 GetRevisionRequest, GetRevisionResponse } from "aws-sdk/clients/qldb";
import { createHash } from "crypto";
import { dom, dumpText, load } from "ion-js"

const ledgerName: string = "vehicle-registration";
const tableName: string = "VehicleRegistration";
const vin: string = "KM8SRDHF6EU074761";
const driver: QldbDriver = new QldbDriver(ledgerName);
const qldbClient: QLDB = new QLDB();
const HASH_SIZE = 32;

/**
 * Takes two hashes, sorts them, concatenates them, and calculates a digest based on
 the concatenated hash.
 * @param h1 Byte array containing one of the hashes to compare.
 * @param h2 Byte array containing one of the hashes to compare.
 * @returns The digest calculated from the concatenated hash values.
 */
function dot(h1: Uint8Array, h2: Uint8Array): Uint8Array {
    if (h1.length === 0) {
        return h2;
    }
    if (h2.length === 0) {
        return h1;
    }
```

```
    const newHashLib = createHash("sha256");

    let concatenated: Uint8Array;
    if (hashComparator(h1, h2) < 0) {
        concatenated = concatenate(h1, h2);
    } else {
        concatenated = concatenate(h2, h1);
    }
    newHashLib.update(concatenated);
    return newHashLib.digest();
}

/**
 * Compares two hashes by their **signed** byte values in little-endian order.
 * @param hash1 The hash value to compare.
 * @param hash2 The hash value to compare.
 * @returns Zero if the hash values are equal, otherwise return the difference of
 the first pair of non-matching
 *          bytes.
 * @throws RangeError When the hash is not the correct hash size.
 */
function hashComparator(hash1: Uint8Array, hash2: Uint8Array): number {
    if (hash1.length !== HASH_SIZE || hash2.length !== HASH_SIZE) {
        throw new RangeError("Invalid hash.");
    }
    for (let i = hash1.length-1; i >= 0; i--) {
        const difference: number = (hash1[i]<<24 >>24) - (hash2[i]<<24 >>24);
        if (difference !== 0) {
            return difference;
        }
    }
    return 0;
}

/**
 * Helper method that concatenates two Uint8Array.
 * @param arrays List of arrays to concatenate, in the order provided.
 * @returns The concatenated array.
 */
function concatenate(...arrays: Uint8Array[]): Uint8Array {
    let totalLength = 0;
    for (const arr of arrays) {
        totalLength += arr.length;
```

```
    }
    const result = new Uint8Array(totalLength);
    let offset = 0;
    for (const arr of arrays) {
        result.set(arr, offset);
        offset += arr.length;
    }
    return result;
}

/**
 * Helper method that checks for equality between two Uint8Array.
 * @param expected Byte array containing one of the hashes to compare.
 * @param actual Byte array containing one of the hashes to compare.
 * @returns Boolean indicating equality between the two Uint8Array.
 */
function isEqual(expected: Uint8Array, actual: Uint8Array): boolean {
    if (expected === actual) return true;
    if (expected == null || actual == null) return false;
    if (expected.length !== actual.length) return false;

    for (let i = 0; i < expected.length; i++) {
        if (expected[i] !== actual[i]) {
            return false;
        }
    }
    return true;
}

const main = async function (): Promise<void> {
    // Get a digest
    const getDigestRequest: GetDigestRequest = {
        Name: ledgerName
    };
    const getDigestResponse: GetDigestResponse = await
 qldbClient.getDigest(getDigestRequest).promise();

    // expectedDigest is the buffer we will later use to compare against our
 calculated digest
    const expectedDigest: Uint8Array = <Uint8Array>getDigestResponse.Digest;

    const result: dom.Value[] = await driver.executeLambda(async (txn:
 TransactionExecutor): Promise<dom.Value[]> => {
```

```
        const query: string = `SELECT blockAddress, hash, metadata.id FROM
_ql_committed_${tableName} WHERE data.VIN = '${vin}'`;
        const queryResult: Result = await txn.execute(query);
        return queryResult.getResultList();
    });

    console.log(`Verifying document revisions for vin '${vin}' in table
'${tableName}' in ledger '${ledgerName}'`);

    for (let value of result) {
        // Get the requested fields
        const blockAddress: dom.Value = value.get("blockAddress");
        const hash: dom.Value = value.get("hash");
        const metadataId: string = value.get("id").stringValue();

        console.log(`Verifying document revision for id '${metadataId}'`);

        // Submit a request for the revision
        const revisionRequest: GetRevisionRequest = {
            Name: ledgerName,
            BlockAddress: {
                IonText: dumpText(blockAddress)
            },
            DocumentId: metadataId,
            DigestTipAddress: getDigestResponse.DigestTipAddress
        };

        // Get a response back
        const revisionResponse: GetRevisionResponse = await
qldbClient.getRevision(revisionRequest).promise();

        let proofValue: dom.Value = load(revisionResponse.Proof.IonText);

        let documentHash: Uint8Array = hash.uInt8ArrayValue();
        proofValue.elements().forEach((proofHash: dom.Value) => {
            // Calculate the digest
            documentHash = dot(documentHash, proofHash.uInt8ArrayValue());
        });

        let verified: boolean = isEqual(expectedDigest, documentHash);

        if (verified) {
            console.log(`Successfully verified document revision for id
'${metadataId}'!`);
```

```
        } else {
            console.log(`Document revision for id '${metadataId}' verification
 failed!`);
            return;
        }

        // Submit a request for the block
        const getBlockRequest: GetBlockRequest = {
            Name: ledgerName,
            BlockAddress: {
                IonText: dumpText(blockAddress)
            },
            DigestTipAddress: getDigestResponse.DigestTipAddress
        };

        // Get a response back
        const getBlockResponse: GetBlockResponse = await
 qldbClient.getBlock(getBlockRequest).promise();

        const blockValue: dom.Value = load(getBlockResponse.Block.IonText)
        let blockHash: Uint8Array = blockValue.get("blockHash").uInt8ArrayValue();

        proofValue = load(getBlockResponse.Proof.IonText);

        proofValue.elements().forEach((proofHash: dom.Value) => {
            // Calculate the digest
            blockHash = dot(blockHash, proofHash.uInt8ArrayValue());
        });

        verified = isEqual(expectedDigest, blockHash);

        if (verified) {
            console.log(`Block address '${dumpText(blockAddress)}' successfully
 verified!`);
        } else {
            console.log(`Block address '${dumpText(blockAddress)}' verification
 failed!`);
        }
    }
};

if (require.main === module) {
    main();
```

```
}
```

Python

```
from amazon.ion.simpleion import dumps, loads
from array import array
from boto3 import client
from functools import reduce
from hashlib import sha256
from pyqldb.driver.qldb_driver import QldbDriver

ledger_name = 'vehicle-registration'
table_name = 'VehicleRegistration'
vin = 'KM8SRDHF6EU074761'
qldb_client = client('qldb')
hash_length = 32


def query_doc_revision(txn):
    query = "SELECT blockAddress, hash, metadata.id FROM _ql_committed_{} WHERE
 data.VIN = '{}'".format(table_name, vin)
    return txn.execute_statement(query)


def block_address_to_dictionary(ion_dict):
    """
    Convert a block address from IonPyDict into a dictionary.
    Shape of the dictionary must be: {'IonText': "{strandId: <"strandId">,
 sequenceNo: <sequenceNo>}"}

    :type ion_dict: :py:class:`amazon.ion.simple_types.IonPyDict`/str
    :param ion_dict: The block address value to convert.

    :rtype: dict
    :return: The converted dict.
    """
    block_address = {'IonText': {}}
    if not isinstance(ion_dict, str):
        py_dict = '{{strandId: "{}", sequenceNo:{}}}'.format(ion_dict['strandId'],
 ion_dict['sequenceNo'])
        ion_dict = py_dict
    block_address['IonText'] = ion_dict
    return block_address
```

```
def dot(hash1, hash2):
    """
    Takes two hashes, sorts them, concatenates them, and then returns the
    hash of the concatenated array.

    :type hash1: bytes
    :param hash1: The hash value to compare.

    :type hash2: bytes
    :param hash2: The hash value to compare.

    :rtype: bytes
    :return: The new hash value generated from concatenated hash values.
    """
    if len(hash1) != hash_length or len(hash2) != hash_length:
        raise ValueError('Illegal hash.')

    hash_array1 = array('b', hash1)
    hash_array2 = array('b', hash2)

    difference = 0
    for i in range(len(hash_array1) - 1, -1, -1):
        difference = hash_array1[i] - hash_array2[i]
        if difference != 0:
            break

    if difference < 0:
        concatenated = hash1 + hash2
    else:
        concatenated = hash2 + hash1

    new_hash_lib = sha256()
    new_hash_lib.update(concatenated)
    new_digest = new_hash_lib.digest()
    return new_digest


# Get a digest
get_digest_response = qldb_client.get_digest(Name=ledger_name)

# expected_digest is the buffer we will later use to compare against our calculated
 digest
```

```
expected_digest = get_digest_response.get('Digest')
digest_tip_address = get_digest_response.get('DigestTipAddress')

qldb_driver = QldbDriver(ledger_name=ledger_name)

# Retrieve info for the given vin's document revisions
result = qldb_driver.execute_lambda(query_doc_revision)

print("Verifying document revisions for vin '{}' in table '{}' in ledger
 '{}'".format(vin, table_name, ledger_name))

for value in result:
    # Get the requested fields
    block_address = value['blockAddress']
    document_hash = value['hash']
    metadata_id = value['id']

    print("Verifying document revision for id '{}'".format(metadata_id))

    # Submit a request for the revision and get a result back
    proof_response = qldb_client.get_revision(Name=ledger_name,

 BlockAddress=block_address_to_dictionary(block_address),
                                              DocumentId=metadata_id,
                                              DigestTipAddress=digest_tip_address)

    proof_text = proof_response.get('Proof').get('IonText')
    proof_hashes = loads(proof_text)

    # Calculate digest
    calculated_digest = reduce(dot, proof_hashes, document_hash)

    verified = calculated_digest == expected_digest
    if verified:
        print("Successfully verified document revision for id
 '{}'!".format(metadata_id))
    else:
        print("Document revision for id '{}' verification
 failed!".format(metadata_id))

    # Submit a request for the block and get a result back
    block_response = qldb_client.get_block(Name=ledger_name,
 BlockAddress=block_address_to_dictionary(block_address),
                                           DigestTipAddress=digest_tip_address)
```

```
    block_text = block_response.get('Block').get('IonText')
    block = loads(block_text)

    block_hash = block.get('blockHash')

    proof_text = block_response.get('Proof').get('IonText')
    proof_hashes = loads(proof_text)

    # Calculate digest
    calculated_digest = reduce(dot, proof_hashes, block_hash)

    verified = calculated_digest == expected_digest
    if verified:
        print("Block address '{}' successfully
verified!".format(dumps(block_address,
                                                                binary=False,

omit_version_marker=True)))
    else:
        print("Block address '{}' verification failed!".format(block_address))
```

확인을 위한 일반적인 오류

이 섹션에서는 확인 요청 시 Amazon QLDB에서 발생하는 런타임 오류를 설명합니다.

다음은 서비스에서 반환하는 일반적인 예외 목록입니다. 각 예외에는 특정 오류 메시지와 이를 발생시킬 수 있는 API 작업, 간단한 설명, 가능한 해결 방법에 대한 제안이 포함됩니다.

IllegalArgumentException

메시지: 제공된 Ion 값이 유효하지 않아 구문 분석할 수 없습니다.

API 작업: `GetDigest, GetBlock, GetRevision`

요청을 재시도하기 전에 유효한 Amazon Ion 값을 제공했는지 확인하세요.

IllegalArgumentException

메시지: 제공된 블록 주소가 유효하지 않습니다.

API 작업: `GetDigest, GetBlock, GetRevision`

요청을 다시 시도하기 전에 유효한 블록 주소를 입력했는지 확인하세요. 블록 주소는 `strandId` 및 `sequenceNo`라는 두 개의 필드를 갖는 Amazon Ion 구조입니다.

예: `{strandId:"BlFTjlSXze9BIh1K0szcE3",sequenceNo:14}`

IllegalArgumentException

메시지: 제공된 다이제스트 팁 주소의 시퀀스 번호가 스트랜드의 최근 커밋된 레코드를 벗어났습니다.

API 작업: `GetDigest, GetBlock, GetRevision`

제공하는 다이제스트 팁 주소의 시퀀스 번호는 저널 스트랜드의 최근 커밋된 레코드의 시퀀스 번호보다 작거나 같아야 합니다. 요청을 다시 시도하기 전에 유효한 시퀀스 번호가 있는 다이제스트 팁 주소를 제공해야 합니다.

IllegalArgumentException

메시지: 제공된 블록 주소의 스트랜드 ID가 유효하지 않습니다.

API 작업: `GetDigest, GetBlock, GetRevision`

제공하는 블록 주소에는 저널의 스트랜드 ID와 일치하는 스트랜드 ID가 있어야 합니다. 요청을 다시 시도하기 전에 유효한 스트랜드 ID를 가진 블록 주소를 제공해야 합니다.

IllegalArgumentException

메시지: 제공된 블록 주소의 시퀀스 번호가 스트랜드의 최신 커밋 레코드를 벗어났습니다.

API 작업: `GetBlock, GetRevision`

제공하는 블록 주소는 스트랜드의 최근 커밋된 레코드의 시퀀스 번호보다 작거나 같은 시퀀스 번호를 가져야 합니다. 요청을 재시도하기 전에 유효한 시퀀스 번호가 있는 블록 주소를 제공해야 합니다.

IllegalArgumentException

메시지: 제공된 블록 주소의 스트랜드 ID는 제공된 다이제스트 팁 주소의 스트랜드 ID와 일치해야 합니다.

API 작업: `GetBlock, GetRevision`

제공한 다이제스트와 동일한 저널 스트랜드에 문서 개정 또는 블록이 존재하는 경우에만 문서 수정 또는 차단을 확인할 수 있습니다.

IllegalArgumentException

메시지: 제공된 블록 주소의 시퀀스 번호가 스트랜드의 최신 커밋 레코드를 벗어났습니다.

API 작업: `GetBlock, GetRevision`

제공한 다이제스트에 포함된 경우에만 문서 수정 또는 차단을 확인할 수 있습니다. 즉, 다이제스트 팁 주소보다 먼저 저널에 커밋되었다는 뜻입니다.

IllegalArgumentException

메시지: 지정한 블록 주소의 블록에서 제공된 문서 ID를 찾을 수 없습니다.

API 작업: `GetRevision`

제공하는 문서 ID는 입력한 블록 주소에 있어야 합니다. 요청을 재시도하기 전에 이 두 파라미터가 일치하는지 확인하세요.

Amazon QLDB에서 저널 데이터 내보내기

Amazon QLDB는 저널이라고 하는 변경 불가능한 트랜잭션 로그를 데이터 스토리지에 사용합니다. 저널은 커밋된 데이터의 모든 변경 내용을 추적하고 시간이 지나면서 완전하고 확인 가능한 시간대별 변경 기록을 유지합니다.

분석, 감사, 데이터 보존, 확인, 다른 시스템으로 내보내기 등 다양한 목적으로 원장에 있는 저널 콘텐츠에 액세스할 수 있습니다. 다음 주제에서는 원장의 저널 블록을 AWS 계정의 Amazon Simple Storage Service(S3) 버킷으로 내보내는 방법을 설명합니다. 저널 내보내기 작업은 Amazon S3의 데이터를 Amazon Ion 형식의 텍스트 또는 이진수 표현이나 JSON Lines 텍스트 형식의 객체로 기록합니다.

JSON Lines 형식에서 내보낸 데이터 객체의 각 블록은 줄 바꿈으로 구분된 유효한 JSON 객체입니다. 이 형식을 사용하여 JSON 내보내기를 Amazon AWS Glue Athena와 같은 분석 도구와 직접 통합할 수 있습니다. 이러한 서비스는 줄바꿈으로 구분된 JSON을 자동으로 파싱할 수 있기 때문입니다. 데이터 형식에 대한 자세한 정보는 JSON Lines을 참조하세요.

Amazon S3에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서를 참조하세요.

> **Note**
>
> 내보내기 작업의 출력 형식으로 JSON을 지정하는 경우, QLDB는 내보낸 데이터 객체의 Ion 저널 데이터를 JSON으로 하향 변환합니다. 자세한 내용은 JSON으로 하향 변환을 참조하십시오.

주제

QLDB에서 저널 내보내기 요청

Amazon QLDB는 지정된 날짜 및 시간 범위와 지정된 Amazon S3 버킷 대상으로 저널 블록 내보내기를 요청하는 API를 제공합니다. 저널 내보내기 작업은 Amazon Ion 형식의 텍스트 또는 이진 표현이나 JSON Lines 텍스트 형식으로 데이터 객체를 쓸 수 있습니다. AWS Management Console, AWS SDK 또는 AWS Command Line Interface (AWS CLI) 를 사용하여 내보내기 작업을 생성할 수 있습니다.

주제

- AWS Management Console
- QLDB API
- 내보내기 작업 만료

AWS Management Console

다음 단계에 따라 QLDB 콘솔을 사용하여 QLDB에서 저널 내보내기 요청을 제출하세요.

내보내기를 요청하려면(콘솔)

1. 에 AWS Management Console로그인하고 https://console.aws.amazon.com/qldb 에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 내보내기를 선택합니다.
3. 내보내기 작업 생성을 선택합니다.
4. 내보내기 작업 생성 페이지에서 다음 내보내기 설정을 입력합니다.

   - 원장 – 내보내기하려는 저널 블록이 있는 원장입니다.
   - 시작 날짜 및 시간 – 내보내기할 저널 블록 범위의 포함되는 시작 타임스탬프로, 협정 세계시(UTC) 기준입니다. 이 타임스탬프는 종료 날짜 및 시간보다 이전이어야 합니다. 원장의 `CreationDateTime`보다 이전인 시작 타임스탬프를 제공하는 경우 QLDB는 기본값을 원장의 `CreationDateTime`으로 설정합니다.
   - 종료 날짜 및 시간 – 내보낼 저널 블록 범위의 불포함되는 종료 타임스탬프(UTC)입니다. 이 날짜와 시간은 미래일 수 없습니다.
   - 저널 블록 대상 - 내보내기 작업이 데이터 객체를 기록하는 Amazon S3 버킷 및 접두사 이름. 다음 Amazon S3 URI 형식을 사용하세요.

     ```
     s3://DOC-EXAMPLE-BUCKET/prefix/
     ```

     출력 객체에 대해 S3 버킷 이름과 선택적 접두사 이름을 지정해야 합니다. 다음은 예입니다.

     ```
     s3://DOC-EXAMPLE-BUCKET/journalExport/
     ```

     버킷 이름과 접두사는 모두 Amazon S3 이름 지정 규칙 및 규칙을 준수해야 합니다. 버킷 이름 지정에 대한 자세한 내용은 Amazon S3 개발자 안내서의 버킷 규제 및 제한을 참조하세요. 객체 이름 접두사에 대한 자세한 내용은 객체 키와 메타데이터 섹션을 참조하세요.

> **Note**
>
> 크로스 리전 쿼리는 지원되지 않습니다. 지정된 Amazon S3 버킷은 원장과 AWS 리전 동일해야 합니다.

- S3 암호화 - 내보내기 작업에서 Amazon S3 버킷에 데이터를 쓰기 위해 사용하는 암호화 설정입니다. Amazon S3의 서버 측 암호화 옵션에 대한 자세한 내용은 Amazon S3 개발자 안내서의 서버 측 암호화를 사용한 데이터 보호를 참조하세요.
  - 버킷 기본 암호화 - 지정된 Amazon S3 버킷의 기본 암호화 설정을 사용합니다.
  - AES-256 - Amazon S3 관리형 키(SSE-S3)로 서버 측 암호화를 사용.
  - AWS-KMS — AWS KMS 관리형 키를 사용한 서버 측 암호화 (SSE-KMS) 를 사용합니다.

    다른 AWS KMS key선택 옵션과 함께 이 유형을 선택하는 경우, 다음 Amazon 리소스 이름(ARN)형식으로 대칭 암호화 KMS 키도 지정해야 합니다.

    ```
    arn:aws:kms:aws-region:account-id:key/key-id
    ```

- 서비스 액세스 – Amazon S3 버킷에 QLDB 쓰기 권한을 부여하는 IAM 역할입니다. 해당하는 경우 IAM 역할은 QLDB에 KMS 키를 사용할 수 있는 권한도 부여해야 합니다.

  저널 내보내기를 요청할 때 QLDB에 역할을 전달하려면 IAM 역할 리소스에서 *iam:PassRole* 작업을 수행할 수 있는 권한이 있어야 합니다.
  - 새 서비스 역할 생성 및 사용 – 콘솔에서 지정된 Amazon S3 버킷에 필요한 권한을 가진 새 역할을 생성하도록 합니다.
  - 기존 서비스 역할 사용 – IAM에서 이 역할을 수동으로 생성하는 방법을 알아보려면 내보내기 권한 섹션을 참조하세요.
- 출력 형식 - 내보낸 저널 데이터의 출력 형식
  - Ion 텍스트 – (기본값)Amazon Ion의 텍스트 표현
  - Ion 이진수 – Amazon 이온의 이진수 표현
  - JSON – 줄바꿈으로 구분된 JSON 텍스트 형식

    JSON을 선택하면 QLDB는 내보낸 데이터 객체의 Ion 저널 데이터를 JSON으로 하향 변환합니다. 자세한 내용은 JSON으로 하향 변환을 참조하십시오.

5. 원하는 대로 설정되었으면 내보내기 작업 생성을 선택합니다.

내보내기 작업을 완료하는 데 걸리는 시간은 데이터 크기에 따라 다릅니다. 요청을 성공적으로 제출하면 콘솔이 기본 내보내기 페이지로 돌아가고 내보내기 작업이 현재 상태와 함께 나열됩니다.

6. Amazon S3 콘솔에서 내보내기 객체를 볼 수 있습니다.

   https://console.aws.amazon.com/s3/에서 S3 콘솔을 엽니다.

   이러한 출력 객체의 형식에 대한 자세한 내용은 QLDB의 저널 내보내기 출력 섹션을 참조하세요.

> **Note**
>
> 내보내기 작업은 완료 후 7일이 지나면 만료됩니다. 자세한 내용은 내보내기 작업 만료을 참조하십시오.

QLDB API

Amazon QLDB API를 SDK 또는 와 함께 AWS 사용하여 저널 내보내기를 요청할 수도 있습니다. AWS CLI애플리케이션 프로그램에서 사용할 수 있는 다음과 같은 기능이 QLDB API에서 제공됩니다.

- `ExportJournalToS3` – 주어진 원장에서 날짜 및 시간 범위 내의 저널 콘텐츠를 주어진 Amazon S3 버킷으로 내보냅니다. 내보내기 작업은 Amazon Ion 형식의 텍스트 또는 이진수 표현이나 JSON Lines 텍스트 형식의 객체로 데이터를 쓸 수 있습니다.
- `DescribeJournalS3Export` – 저널 내보내기 작업에 대한 자세한 정보를 반환합니다. 출력에는 현재 상태, 작성 시간, 원본 내보내기 요청의 파라미터가 포함됩니다.
- `ListJournalS3Exports` – 현재 AWS 계정 및 리전과 관련된 모든 원장에 대한 저널 내보내기 작업 설명 목록을 반환합니다. 각 내보내기 작업 설명의 출력에는 `DescribeJournalS3Export`에서 반환한 것과 동일한 세부 정보가 포함됩니다.
- `ListJournalS3ExportsForLedger` – 주어진 원장에 대한 저널 내보내기 작업 설명 목록을 반환합니다. 각 내보내기 작업 설명의 출력에는 `DescribeJournalS3Export`에서 반환한 것과 동일한 세부 정보가 포함됩니다.

이러한 API작업 설명 전체를 보려면 Amazon QLDB API 참조 섹션을 참조하세요.

를 사용하여 저널 데이터를 내보내는 방법에 대한 자세한 내용은 명령 AWS CLI참조를 참조하십시오.AWS CLI

샘플 애플리케이션(Java)

기본 내보내기 작업의 자바 코드 예제는 amazon-qldb-dmv-sampleaws-samples/ -java GitHub 리포지토리를 참조하십시오. 이 샘플 애플리케이션을 다운로드하여 설치하는 방법에 대한 자세한 내용은 Amazon QLDB Java 샘플 애플리케이션 설치 섹션을 참조하세요. 내보내기를 요청하기 전에 Java 자습서의 1~3단계에 따라 샘플 원장을 생성하고 샘플 데이터와 함께 로드해야 합니다.

다음 클래스의 자습서 코드는 내보내기 생성, 내보내기 상태 확인, 내보내기 출력 처리의 예를 제공합니다.

Class	설명
ExportJournal	`vehicle-registration` 샘플 원장의 저널 블록을 10분 전부터 지금까지의 타임스탬프 범위로 내보냅니다. 지정된 S3 버킷에 출력 객체를 쓰거나, 지정되지 않은 경우 고유한 버킷을 생성합니다.
DescribeJournalExport	`vehicle-registration` 샘플 원장의 지정된 `exportId`에 대한 저널 내보내기 작업에 대해 설명합니다.
ListJournalExports	`vehicle-registration` 샘플 원장에 대한 저널 내보내기 작업 설명 목록을 반환합니다.
ValidateQldbHashChain	주어진 `exportId`를 사용하여 `vehicle-registration` 샘플 원장의 해시 체인을 확인합니다. 제공되지 않은 경우 해시 체인 확인에 사용할 새 내보내기를 요청합니다.

내보내기 작업 만료

완료된 저널 내보내기 작업에는 7일의 보존 기간이 적용됩니다. 이 한도가 만료되면 해당 파일은 자동으로 영구 삭제됩니다. 이 만료 기간은 고정 한도이며 변경할 수 없습니다.

완료된 내보내기 작업이 삭제된 후에는 더 이상 QLDB 콘솔 또는 다음 API 작업을 사용하여 작업에 대한 메타데이터를 검색할 수 없습니다.

- `DescribeJournalS3Export`

- `ListJournalS3Exports`
- `ListJournalS3ExportsForLedger`

하지만 이 만료는 내보낸 데이터 자체에는 영향을 주지 않습니다. 모든 메타데이터는 내보내기로 작성된 매니페스트 파일에 보존됩니다. 이 만료는 저널 내보내기 작업을 나열하는 API 작업을 보다 원활하게 수행할 수 있도록 설계되었습니다. QLDB는 이전 내보내기 작업을 제거하여 여러 페이지의 작업을 파싱하지 않고도 최근 내보내기만 볼 수 있도록 합니다.

QLDB의 저널 내보내기 출력

Amazon QLDB 저널 내보내기 작업은 저널 블록을 포함하는 데이터 객체 외에도 두 개의 매니페스트 파일을 작성합니다. 해당 파일은 모두 내보내기 요청에서 지정하는 Amazon S3 버킷에 저장됩니다. 다음 섹션에서는 각 출력 객체의 형식과 내용을 설명합니다.

> **Note**
>
> 내보내기 작업의 출력 형식으로 JSON을 지정하는 경우 QLDB는 내보낸 데이터 객체의 Amazon Ion 저널 데이터를 JSON으로 하향 변환합니다. 자세한 정보는 JSON으로 하향 변환 섹션으로 이동하세요.

주제

- 매니페스트 파일
- 데이터 객체
- JSON으로 하향 변환
- 내보내기 프로세서 라이브러리(Java)

매니페스트 파일

Amazon QLDB는 제공된 S3 버킷에 각 내보내기 요청에 대해 매니페스트 파일을 두 개 생성합니다. 초기 매니페스트 파일은 내보내기 요청을 제출하는 즉시 생성됩니다. 최종 매니페스트 파일은 내보내기가 완료된 후에 작성됩니다. 이러한 파일을 사용하여 Amazon S3의 내보내기 작업 상태를 확인할 수 있습니다.

매니페스트 파일의 콘텐츠 형식은 요청된 내보내기 출력 형식과 일치합니다.

초기 매니페스트

초기 매니페스트는 내보내기 작업이 시작되었음을 나타냅니다. 여기에는 요청에 전달한 입력 파라미터가 포함되어 있습니다. 이 파일에는 Amazon S3 대상, 내보내기 시작 및 종료 시간 파라미터 외에도 `exportId`가 포함되어 있습니다. `exportId`는 QLDB가 각 내보내기 작업에 할당하는 고유 ID입니다.

파일 명명 규칙은 다음과 같습니다.

```
s3://DOC-EXAMPLE-BUCKET/prefix/exportId.started.manifest
```

다음은 Ion 텍스트 형식의 초기 매니페스트 파일과 해당 콘텐츠의 예입니다.

```
s3://DOC-EXAMPLE-BUCKET/journalExport/8UyXulxccYLAsbN1aon7e4.started.manifest
```

```
{
  ledgerName:"my-example-ledger",
  exportId:"8UyXulxccYLAsbN1aon7e4",
  inclusiveStartTime:2019-04-15T00:00:00.000Z,
  exclusiveEndTime:2019-04-15T22:00:00.000Z,
  bucket:"DOC-EXAMPLE-BUCKET",
  prefix:"journalExport",
  objectEncryptionType:"NO_ENCRYPTION",
  outputFormat:"ION_TEXT"
}
```

초기 매니페스트는 내보내기 요청에 지정된 경우에만 `outputFormat`을 포함합니다. 출력 형식을 지정하지 않으면 내보내는 데이터의 기본값은 `ION_TEXT` 형식입니다.

DescribeJournalS3Export API 작업 및 내보낸 Amazon S3 객체의 콘텐츠 유형도 출력 형식을 나타냅니다.

최종 매니페스트

최종 매니페스트는 특정 저널 스트랜드의 내보내기 작업이 완료되었음을 나타냅니다. 내보내기 작업은 각 스트랜드에 대해 별도의 최종 매니페스트 파일을 작성합니다.

> **Note**
>
> Amazon QLDB에서 스트랜드는 원장 저널의 파티션입니다. QLDB는 현재 단일 스트랜드가 포함된 저널만 지원합니다.

최종 매니페스트에는 내보내기 중에 기록된 데이터 객체 키의 정렬된 목록이 포함됩니다. 파일 명명 규칙은 다음과 같습니다.

```
s3://DOC-EXAMPLE-BUCKET/prefix/exportId.strandId.completed.manifest
```

strandId는 QLDB가 스트랜드에 할당하는 고유 ID입니다. 다음은 Ion 텍스트 형식의 최종 매니페스트 파일과 해당 콘텐츠의 예입니다.

```
s3://DOC-EXAMPLE-BUCKET/
journalExport/8UyXulxccYLAsbN1aon7e4.JdxjkR9bSYB5jMHWcI464T.completed.manifest
```

```
{
  keys:[
    "2019/04/15/22/JdxjkR9bSYB5jMHWcI464T.1-4.ion",
    "2019/04/15/22/JdxjkR9bSYB5jMHWcI464T.5-10.ion",
    "2019/04/15/22/JdxjkR9bSYB5jMHWcI464T.11-12.ion",
    "2019/04/15/22/JdxjkR9bSYB5jMHWcI464T.13-20.ion",
    "2019/04/15/22/JdxjkR9bSYB5jMHWcI464T.21-21.ion"
  ]
}
```

데이터 객체

Amazon QLDB는 제공된 Amazon S3 버킷에 Amazon Ion 형식의 텍스트 또는 이진수 표현이나JSON Lines 텍스트 형식으로 저널 데이터 객체를 기록합니다.

JSON Lines 형식에서 내보낸 데이터 객체의 각 블록은 줄 바꿈으로 구분된 유효한 JSON 객체입니다. 이 형식을 사용하여 JSON 내보내기를 Amazon AWS Glue Athena와 같은 분석 도구와 직접 통합할 수 있습니다. 이러한 서비스는 줄바꿈으로 구분된 JSON을 자동으로 파싱할 수 있기 때문입니다. 데이터 형식에 대한 자세한 정보는 JSON Lines을 참조하세요.

데이터 객체 이름

저널 내보내기 작업은 다음과 같은 명명 규칙에 따라 이러한 데이터 객체를 작성합니다.

```
s3://DOC-EXAMPLE-BUCKET/prefix/yyyy/mm/dd/hh/strandId.startSn-endSn.ion|.json
```

- 각 내보내기 작업의 출력 데이터는 청크로 분할됩니다.
- yyyy/mm/dd/hh – 내보내기 요청을 제출한 날짜 및 시간. 동일한 시간 내에 내보낸 객체는 동일한 Amazon S3 접두사로 그룹화됩니다.

- `strandId` – 내보내는 저널 블록을 포함하는 특정 스트랜드의 고유 ID.
- `startSn-endSn` – 객체에 포함된 시퀀스 번호 범위. 시퀀스 번호는 스트랜드 내 블록의 위치를 지정합니다.

예를 들어 다음과 같은 경로를 지정한다고 가정하겠습니다.

```
s3://DOC-EXAMPLE-BUCKET/journalExport/
```

내보내기 작업을 수행하면 다음과 비슷한 Amazon S3 데이터 객체가 생성됩니다. 이 예제는 객체 이름을 Ion 형식으로 보여줍니다.

```
s3://DOC-EXAMPLE-BUCKET/journalExport/2019/04/15/22/JdxjkR9bSYB5jMHWcI464T.1-5.ion
```

데이터 객체 콘텐츠

각 데이터 객체에는 다음 형식의 저널 블록 객체가 포함되어 있습니다.

```
{
  blockAddress: {
    strandId: String,
    sequenceNo: Int
  },
  transactionId: String,
  blockTimestamp: Datetime,
  blockHash: SHA256,
  entriesHash: SHA256,
  previousBlockHash: SHA256,
  entriesHashList: [ SHA256 ],
  transactionInfo: {
    statements: [
      {
        //PartiQL statement object
      }
    ],
    documents: {
      //document-table-statement mapping object
    }
  },
  revisions: [
    {
      //document revision object
```

```
    }
  ]
}
```

블록은 트랜잭션 중에 저널에 커밋되는 객체입니다. 블록에는 트랜잭션 메타데이터가 함께 트랜잭션에서 커밋된 문서 수정본과 이를 커밋한 PartiQL 문을 나타내는 항목이 포함되어 있습니다.

다음은 Ion 텍스트 형식의 샘플 데이터가 있는 블록의 예입니다. 객체 ACL 필드에 대한 자세한 내용은 Amazon QLDB의 저널 콘텐츠 섹션을 참조하세요.

> **Note**
>
> 이 블록 예제는 정보 제공 목적으로만 제공됩니다. 표시된 해시는 실제 계산된 해시 값이 아닙니다.

```
{
  blockAddress:{
    strandId:"JdxjkR9bSYB5jMHWcI464T",
    sequenceNo:1234
  },
  transactionId:"D35qctdJRU1L1N2VhxbwSn",
  blockTimestamp:2019-10-25T17:20:21.009Z,
  blockHash:{{WYLOfZClk0lYWT3lUsSr0ONXh+Pw8MxxB+9zvTgSvlQ=}},
  entriesHash:{{xN9X96atkMvhvF3nEy6jMSVQzKjHJfz1H3bsNeg8GMA=}},
  previousBlockHash:{{IAfZ0h22ZjvcuHPSBCDy/6XNQTsqEmeY3GW0gBae8mg=}},
  entriesHashList:[
      {{F7rQIKCNn0vXVWPexilGfJn5+MCrtsSQqqVdlQxXpS4=}},
      {{C+L8gRhkzVcxt3qRJpw8w6hVEqA5A6ImGne+E7iHizo=}}
  ],
  transactionInfo:{
    statements:[
      {
        statement:"CREATE TABLE VehicleRegistration",
        startTime:2019-10-25T17:20:20.496Z,
        statementDigest:{{3jeSdejOgp6spJ8huZxDRUtp2fRXRqpOMtG43V0nXg8=}}
      },
      {
        statement:"CREATE INDEX ON VehicleRegistration (VIN)",
        startTime:2019-10-25T17:20:20.549Z,
        statementDigest:{{099D+5ZWDgA7r+aWeNUrWhc8ebBTXjgscq+mZ2dVibI=}}
      },
```

```
      {
        statement:"CREATE INDEX ON VehicleRegistration (LicensePlateNumber)",
        startTime:2019-10-25T17:20:20.560Z,
        statementDigest:{{B73tVJzVyVXicnH4n96NzU2L2JFY8e9Tjg895suWMew=}}
      },
      {
        statement:"INSERT INTO VehicleRegistration ?",
        startTime:2019-10-25T17:20:20.595Z,
        statementDigest:{{ggpon5qCXLo95K578YVhAD8ix0A0M5CcBx/W40Ey/Tk=}}
      }
    ],
    documents:{
      '8F0TPCmdNQ6JTRpiLj2TmW':{
        tableName:"VehicleRegistration",
        tableId:"BPxNiDQXCIB5l5F68KZoOz",
        statements:[3]
      }
    }
  },
  revisions:[
    {
      hash:{{FR1IWcWew0yw1TnRklo2YMF/qtwb7ohsu5FD8A4DSVg=}}
    },
    {
      blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:1234
      },
      hash:{{t8Hj6/VC4SBitxnvBqJbOmrGytF2XAA/1c0AoSq2NQY=}},
      data:{
        VIN:"1N4AL11D75C109151",
        LicensePlateNumber:"LEWISR261LL",
        State:"WA",
        City:"Seattle",
        PendingPenaltyTicketAmount:90.25,
        ValidFromDate:2017-08-21,
        ValidToDate:2020-05-11,
        Owners:{
          PrimaryOwner:{
            PersonId:"GddsXfIYfDlKCEprOLOwYt"
          },
          SecondaryOwners:[]
        }
      },
```

```
      metadata:{
        id:"8F0TPCmdNQ6JTRpiLj2TmW",
        version:0,
        txTime:2019-10-25T17:20:20.618Z,
        txId:"D35qctdJRU1L1N2VhxbwSn"
      }
    }
  ]
}
```

`revisions` 필드에서 일부 개정 객체에는 `hash` 값만 포함되고 기타 속성은 포함되지 않을 수 있습니다. 이는 사용자 데이터를 포함하지 않는 내부 전용 시스템 수정본입니다. 이러한 수정본의 해시는 저널의 전체 해시 체인의 일부이기 때문에 내보내기 작업에는 해당 블록에 이러한 수정본이 포함됩니다. 암호화 확인에는 전체 해시 체인이 필요합니다.

JSON으로 하향 변환

내보내기 작업의 출력 형식으로 JSON을 지정하는 경우 QLDB는 내보낸 데이터 객체의 Amazon Ion 저널 데이터를 JSON으로 하향 변환합니다. 하지만 JSON에 없는 풍부한 Ion 유형을 데이터에 사용하는 경우 Ion을 JSON으로 변환해도 손실이 발생합니다.

Ion에서 JSON으로의 변환 규칙에 대한 자세한 내용은 Amazon Ion Cookbook의 JSON으로 하향 변환을 참조하세요.

내보내기 프로세서 라이브러리(Java)

QLDB는 Amazon S3에서의 내보내기 처리를 간소화하는 Java용 확장 가능한 프레임워크를 제공합니다. 이 프레임워크 라이브러리는 내보내기의 출력을 읽고 내보낸 블록을 순차적으로 반복하는 작업을 처리합니다. 이 익스포트 프로세서를 사용하려면 awslabs/ -java 리포지토리를 참조하십시오. GitHub amazon-qldb-export-processor

QLDB의 저널 내보내기 권한

Amazon QLDB에서 저널 내보내기 요청을 제출하기 전에 지정된 Amazon S3 버킷에 대한 쓰기 권한을 QLDB에 제공해야 합니다. Amazon S3 버킷의 객체 암호화 유형으로 고객 관리형 AWS KMS key를 선택하는 경우, QLDB에 지정된 대칭 암호화 키를 사용할 수 있는 권한도 제공해야 합니다. Amazon S3에서는 비대칭 KMS 키가 지원되지 않습니다.

내보내기 작업에 필요한 권한을 제공하려면 적절한 권한 정책을 통해 QLDB가 IAM 서비스 역할을 맡도록 합니다. 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역

할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용자 설명서의 역할을 생성하여 AWS 서비스에게 권한 위임을 참조하세요.

> **Note**
>
> 저널 내보내기를 요청할 때 QLDB에 역할을 전달하려면 IAM 역할 리소스에서 `iam:PassRole` 작업을 수행할 수 있는 권한이 있어야 합니다. 이는 QLDB 원장 리소스에 대한 `qldb:ExportJournalToS3` 권한에 추가됩니다.
> IAM을 사용하여 QLDB에 대한 액세스를 제어하는 방법을 알아보려면 Amazon QLDB에서 IAM을 사용하는 방법 섹션을 참조하세요. QLDB 정책 예제는 Amazon QLDB의 자격 증명 기반 정책 예 섹션을 참조하세요.

이 예에서는 QLDB에서 사용자 대신 Amazon S3 버킷에 객체를 기록하도록 허용하는 역할을 만듭니다. 자세한 내용은 IAM 사용자 설명서의 역할을 생성하여 AWS 서비스에게 권한 위임을 참조하세요.

QLDB 저널을 처음으로 내보내는 경우, 먼저 다음을 수행하여 적절한 정책이 적용된 IAM 역할을 생성해야 합니다. AWS 계정 또는 QLDB 콘솔을 사용하여 자동으로 역할을 생성할 수 있습니다. 또는 이전에 생성한 역할을 선택할 수 있습니다.

주제

- 권한 정책 생성
- IAM 역할 생성

권한 정책 생성

다음 단계를 완료하여 QLDB 저널 내보내기 작업에 대한 권한 정책을 만드세요. 이 예제는 지정된 버킷에 객체를 쓸 수 있는 권한을 QLDB에 부여하는 Amazon S3 버킷 정책을 보여줍니다. 해당하는 경우 이 예제에서는 QLDB가 대칭 암호화 KMS 키를 사용하도록 허용하는 키 정책도 보여줍니다.

Amazon S3의 버킷 정책에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버킷 정책 및 사용자 정책 사용을 참조하세요. AWS KMS 키 정책에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서에서 AWS KMS에서 키 정책 사용을 참조하세요.

> **Note**
>
> Amazon S3 버킷과 KMS 키는 모두 QLDB 원장과 AWS 리전 동일해야 합니다.

JSON 정책 편집기를 사용하여 정책을 생성하려면

1. 에 AWS Management Console 로그인하고 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 왼쪽의 탐색 열에서 정책을 선택합니다.

   정책을 처음으로 선택하는 경우 관리형 정책 소개 페이지가 나타납니다. 시작하기를 선택합니다.
3. 페이지 상단에서 정책 생성을 선택합니다.
4. JSON 탭을 선택합니다.
5. JSON 정책 문서를 입력합니다.
   - Amazon S3 객체 암호화에 고객 관리형 KMS 키를 사용하는 경우 다음 예제 정책 문서를 사용하세요. 이 정책을 사용하려면 예제의 *DOC-EXAMPLE-BUCKET*, *us-east-1*, *123456789012*, *1234abcd-12ab-34cd-56ef-1234567890ab*를 사용자 고유의 정보로 바꾸세요.

     ```
     {
         "Version": "2012-10-17",
         "Statement": [
             {
                 "Sid": "QLDBJournalExportS3Permission",
                 "Action": [
                     "s3:PutObjectAcl",
                     "s3:PutObject"
                 ],
                 "Effect": "Allow",
                 "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
             },
             {
                 "Sid": "QLDBJournalExportKMSPermission",
                 "Action": [ "kms:GenerateDataKey" ],
                 "Effect": "Allow",
                 "Resource": "arn:aws:kms:us-
     east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
             }
         ]
     }
     ```
   - 다른 암호화 유형의 경우 다음 예제 정책 문서를 사용합니다. 이 정책을 사용하려면 예제에 있는 *DOC-EXAMPLE-BUCKET*을 사용자의 Amazon S3 버킷 이름으로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBJournalExportS3Permission",
            "Action": [
                "s3:PutObjectAcl",
                "s3:PutObject"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
        }
    ]
}
```

6. 정책 검토를 선택합니다.

> **Note**
>
> 언제든지 시각적 편집기 및 JSON 탭을 전환할 수 있습니다. 그러나 변경을 수행하거나 시각적 편집기 탭에서 정책 검토를 선택한 경우 IAM은 시각적 편집기에 최적화되도록 정책을 재구성할 수 있습니다. 자세한 내용은 IAM 사용자 설명서의 정책 재구성을 참조하세요.

7. 정책 검토 페이지에서 생성하려는 정책의 이름과 설명(선택 사항)을 입력합니다. 정책 요약을 검토하여 정책이 부여한 권한을 확인합니다. 그런 다음 정책 생성을 선택하여 작업을 저장합니다.

IAM 역할 생성

QLDB 저널 내보내기 작업에 대한 권한 정책을 생성한 후 IAM 역할을 생성하고 정책을 여기에 연결할 수 있습니다.

QLDB에 대한 서비스 역할을 생성하는 방법(IAM 콘솔)

1. https://console.aws.amazon.com/iam/ 에서 AWS Management Console 로그인하고 IAM 콘솔을 엽니다.
2. IAM 콘솔의 탐색 창에서 역할을 선택하고 역할 생성을 선택합니다.
3. 신뢰할 수 있는 엔터티 유형에 AWS 서비스를 선택합니다.
4. 서비스 또는 사용 사례의 경우 QLDB를 선택한 다음 QLDB 사용 사례를 선택합니다.

5. 다음을 선택합니다.
6. 앞에서 생성한 정책 옆의 상자를 선택합니다.
7. (선택 사항)권한 경계로서 설정됩니다. 이는 서비스 역할에서 가능한 고급 기능이며 서비스 링크된 역할은 아닙니다.

   a. 권한 경계 설정 섹션을 열고 최대 역할 권한을 관리하기 위한 권한 경계 사용을 선택합니다.

      IAM에는 계정의 AWS 관리형 및 고객 관리형 정책 목록이 포함되어 있습니다.

   b. 정책을 선택하여 권한 경계를 사용하세요.
8. 다음을 선택합니다.
9. 역할의 목적을 식별하는 데 도움이 되는 역할 이름이나 역할 이름 접미사를 입력합니다.

   > **Important**
   >
   > 역할 이름을 지정할 때는 다음 사항에 유의하세요.
   >
   > - 역할 이름은 사용자 내에서 고유해야 AWS 계정하며 대소문자를 구분하여 고유할 수 없습니다.
   >
   >   예를 들어, 이름이 **PRODROLE**과 **prodrole**, 두 가지로 지정된 역할을 만들지 마십시오. 역할 이름이 정책 또는 ARN의 일부로 사용되는 경우 역할 이름은 대소문자를 구분합니다. 그러나 로그인 프로세스와 같이 콘솔에서 역할 이름이 고객에게 표시되는 경우에는 역할 이름이 대소문자를 구분하지 않습니다.
   > - 다른 엔터티가 역할을 참조할 수 있기 때문에 역할이 생성된 후에는 역할 이름을 편집할 수 없습니다.
10. (선택 사항)설명에 역할에 대한 설명을 입력합니다.
11. (선택 사항) 역할에 대한 사용 사례와 권한을 편집하려면 1단계: 신뢰할 수 있는 엔터티 선택 또는 2단계: 권한 추가 섹션에서 편집을 선택합니다.
12. (선택 사항) 태그를 키-값 페어로 연결하여 역할을 식별, 구성 또는 검색합니다. IAM에서 태그 사용에 대한 자세한 내용을 알아보려면 IAM 사용 설명서의 IAM 리소스에 태그 지정을 참조하세요.
13. 역할을 검토한 다음 역할 생성을 선택합니다.

다음 JSON 문서는 QLDB가 특정 권한이 부여된 IAM 역할을 맡도록 허용하는 신뢰 정책의 예입니다.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "qldb.amazonaws.com"
            },
            "Action": [ "sts:AssumeRole" ],
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:qldb:us-east-1:123456789012:*"
                },
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                }
            }
        }
    ]
}
```

> **Note**
>
> 이 신뢰 정책 예시에서 `aws:SourceArn` 및 `aws:SourceAccount` 전역 조건 컨텍스트 키를 사용하여 혼동된 대리자 문제를 방지하는 방법을 보여줍니다. 이 신뢰 정책을 사용하면 QLDB는 계정 123456789012의 모든 QLDB 리소스에 대한 역할만 맡을 수 있습니다.
> 자세한 내용은 교차 서비스 혼동된 대리자 예방을 참조하십시오.

IAM 역할을 생성한 후 QLDB 콘솔로 돌아가서 내보내기 작업 생성 페이지를 새로 고쳐 새 역할을 찾을 수 있도록 합니다.

저널 내보내기의 일반적인 오류

이 섹션에서는 저널 내보내기 요청 시 Amazon QLDB에서 발생하는 런타임 오류를 설명합니다.

다음은 서비스에서 반환하는 일반적인 예외 목록입니다. 각 예외에는 특정 오류 메시지와 가능한 해결 방법에 대한 간단한 설명 및 제안이 포함됩니다.

AccessDeniedException

###: ###: UserArn# ### ## ### ### ### ####. iam: PassRole on ###: RoLearn

IAM 역할을 QLDB 서비스에 전달할 권한이 없습니다. QLDB에는 모든 저널 내보내기 요청에 대한 역할이 필요하며 이 역할을 QLDB에 전달할 권한이 있어야 합니다. 이 역할은 지정된 Amazon S3 버킷에 대한 쓰기 권한을 QLDB에 제공합니다.

QLDB 서비스(`qldb.amazonaws.com`)의 지정된 IAM 역할 리소스에서 `PassRole` API 작업을 수행할 권한을 부여하는 IAM 정책을 정의했는지 확인하세요. 정책 예제는 Amazon QLDB의 자격 증명 기반 정책 예을 참조하십시오.

IllegalArgumentException

메시지: QLDB encountered an error validating S3 configuration: *errorCodeerrorMessage*

이 오류의 가능한 원인은 제공된 Amazon S3 버킷이 Amazon S3에 존재하지 않기 때문입니다. 또는 QLDB에 지정된 Amazon S3 버킷에 객체를 쓸 수 있는 충분한 권한이 없습니다.

내보내기 작업 요청에 제공한 S3 버킷 이름이 정확한지 확인하세요. 버킷 이름 지정에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버킷 규제 및 제한을 참조하세요.

또한 QLDB 서비스(`qldb.amazonaws.com`)에 대한 `PutObject` 및 `PutObjectAcl` 권한을 부여하는 지정된 버킷에 대한 정책을 정의했는지 확인합니다. 자세한 내용은 내보내기 권한 섹션을 참조하세요.

IllegalArgumentException

메시지: Unexpected response from Amazon S3 while validating the S3 configuration. S3로부터의 응답: *errorCode errorMessage*

제공된 Amazon S3 오류 응답으로 인해 제공된 S3 버킷에 저널 내보내기 데이터를 쓰려는 시도가 실패했습니다. 발생 가능한 원인에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 Amazon S3 문제 해결을 참조하십시오.

IllegalArgumentException

메시지: Amazon S3 버킷 접두사는 128자를 초과할 수 없습니다

저널 내보내기 요청에 제공된 접두사가 128자를 초과합니다.

IllegalArgumentException

메시지: 시작 날짜는 종료 날짜보다 이후일 수 없습니다.

`InclusiveStartTime` 및 `ExclusiveEndTime`는 ISO 8601 날짜 및 시간 형식이어야 하며 협정 세계시(UTC)여야 합니다.

IllegalArgumentException

메시지: 종료일은 미래일 수 없습니다

`InclusiveStartTime` 및 `ExclusiveEndTime`은 UTC 기준의 `ISO 8601` 날짜 및 시간 형식이어야 합니다.

IllegalArgumentException

메시지: 제공된 객체 암호화 설정 (S3EncryptionConfiguration) 이 () 키와 호환되지 않습니다. AWS Key Management Service AWS KMS

`KMSKeyArn`에 `NO_ENCRYPTION` 또는 `SSE_S3`의 `ObjectEncryptionType`를 제공했습니다. `SSE_KMS`의 객체 암호화 유형에 고객 관리형 AWS KMS key 만 제공할 수 있습니다. Amazon S3의 서버 측 암호화 옵션에 대한 자세한 내용은 Amazon S3 개발자 안내서의 서버 측 암호화를 사용한 데이터 보호를 참조하세요.

LimitExceededException

메시지: 동시 실행 저널 내보내기 작업 제한인 2개를 초과했습니다

QLDB은 기본적으로 동시 저널 내보내기 작업을 2개로 제한합니다.

Amazon QLDB에서 저널 데이터 스트리밍

Amazon QLDB는 저널이라고 하는 변경 불가능한 트랜잭션 로그를 데이터 스토리지에 사용합니다. 저널은 커밋된 데이터의 모든 변경 내용을 추적하고 시간이 지나면서 완전하고 확인 가능한 시간대별 변경 기록을 유지합니다.

QLDB에서 저널에 커밋된 모든 문서 수정본을 수집하고 이 데이터를 Amazon Kinesis Data Streams에 거의 실시간으로 전송할 수 있는 스트림을 QLDB에서 생성할 수 있습니다. QLDB 스트림은 원장의 저널에서 Kinesis 데이터 스트림 리소스로의 지속적인 데이터 흐름입니다.

그런 다음 Kinesis 스트리밍 플랫폼 또는 Kinesis Client Library를 사용하여 스트림을 사용하고, 데이터 레코드를 처리하고, 데이터 콘텐츠를 분석할 수 있습니다. QLDB 스트림은 제어, 블록 요약, 수정 세부 정보 등 세 가지 유형의 레코드로 Kinesis Data Streams에 데이터를 씁니다. 자세한 내용은 Kinesis의 QLDB 스트림 레코드을 참조하십시오.

주제

일반 사용 사례

스트리밍을 사용하면 저널 데이터를 다른 서비스와 통합하면서 QLDB를 확인 가능한 단일 진실 공급원으로 사용할 수 있습니다. 다음은 QLDB 저널 스트림이 지원하는 몇 가지 일반적인 사용 사례입니다.

- 이벤트 기반 아키텍처 – 분리된 구성 요소를 사용하여 이벤트 기반 아키텍처 스타일로 애플리케이션을 빌드합니다. 예를 들어, 은행은 AWS Lambda 함수를 사용하여 계정 잔액이 임계값 아래로 떨어

지면 고객에게 알리는 알림 시스템을 구현할 수 있습니다. 이러한 시스템에서는 계정 잔액이 QLDB 원장에 유지되며 잔액 변경 사항은 저널에 기록됩니다. 이 AWS Lambda 함수는 저널에 커밋되어 Kinesis 데이터 스트림으로 전송되는 밸런스 업데이트 이벤트를 소비하면 알림 로직을 트리거할 수 있습니다.

- 실시간 분석 – 이벤트 데이터에 대한 실시간 분석을 실행하는 Kinesis 소비자 애플리케이션을 빌드합니다. 이 기능을 사용하면 거의 실시간으로 통찰력을 얻고 변화하는 비즈니스 환경에 신속하게 대응할 수 있습니다. 예를 들어, 전자 상거래 웹 사이트는 제품 판매 데이터를 분석하고 판매량이 한도에 도달하는 즉시 할인된 제품에 대한 광고를 중단할 수 있습니다.
- 기록 분석– 과거 이벤트 데이터를 재생하여 Amazon QLDB의 저널 지향 아키텍처를 활용합니다. 과거 어느 시점에서든 QLDB 스트림을 시작하도록 선택할 수 있으며, 이 경우 해당 시점 이후의 모든 수정본이 Kinesis Data Streams에 전달됩니다. 이 기능을 사용하면 기록 데이터에 대한 분석 작업을 실행하는 Kinesis 소비자 애플리케이션을 빌드할 수 있습니다. 예를 들어 전자 상거래 웹 사이트는 필요에 따라 분석을 실행하여 이전에 포착되지 않았던 과거 판매 지표를 생성할 수 있습니다.
- 목적별 데이터베이스로 복제 – QLDB 저널 스트림을 사용하여 QLDB 원장을 다른 목적별 데이터 스토어에 연결합니다. 예를 들어 Kinesis 스트리밍 데이터 플랫폼을 사용하여 QLDB 문서에 대한 전체 텍스트 검색 기능을 제공할 수 있는 Amazon OpenSearch Service와 통합할 수 있습니다. 또한 사용자 지정 Kinesis 소비자 애플리케이션을 빌드하여 저널 데이터를 다양한 구체화된 뷰를 제공하는 다른 목적별 데이터베이스에 복제할 수 있습니다. 예를 들어 관계형 데이터의 경우 Amazon Aurora로, 그래프 기반 데이터의 경우 Amazon Neptune으로 복제합니다.

스트림 사용

Kinesis Data Streams를 사용하여 대량의 데이터 레코드 스트림을 지속적으로 소비, 처리 및 분석할 수 있습니다. Kinesis 스트리밍 데이터 플랫폼에는 Kinesis Data Streams 외에도 아마존 데이터 파이어호스 및 아파치 플링크용 아마존 매니지드 서비스가 포함되어 있습니다. 이 플랫폼을 사용하여 Amazon OpenSearch 서비스, Amazon Redshift, Amazon S3 또는 Splunk와 같은 서비스에 데이터 레코드를 직접 보낼 수 있습니다. 자세한 내용을 알아보려면 Amazon Kinesis Data Streams 개발자 안내서의 Amazon Kinesis Streams 주요 개념을 참조하세요.

Kinesis Client Library(KCL)를 사용하여 사용자 지정 방식으로 데이터 레코드를 처리하는 스트림 소비자 애플리케이션을 빌드할 수도 있습니다. KCL은 하위 수준의 Kinesis Data Streams API에 유용한 추상화를 제공하여 코딩을 단순화합니다. KCL에 대한 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 Kinesis Client Library를 사용을 참조하세요.

전송 보증

QLDB 스트림은 전송을 보장합니다. at-least-once QLDB 스트림에서 생성된 각 데이터 레코드는 Kinesis Data Streams에 한 번 이상 전송됩니다. Kinesis 데이터 스트림에는 동일한 레코드가 여러 번 표시될 수 있습니다. 따라서 사용 사례에 필요한 경우 소비자 애플리케이션 계층에 중복 제거 로직이 있어야 합니다.

순서 또한 보장되지 않습니다. 경우에 따라 Kinesis 데이터 스트림에서 QLDB 블록 및 수정본이 순서대로 생성되지 않을 수 있습니다. 자세한 내용은 중복 및 out-of-order 레코드 처리을 참조하십시오.

전송 지연 시간 고려 사항

QLDB 스트림은 일반적으로 거의 실시간으로 Kinesis Data Streams에 업데이트를 전송합니다. 하지만 다음 시나리오에서는 새로 커밋된 QLDB 데이터가 Kinesis 데이터 스트림으로 전송되기 전에 추가 지연이 발생할 수 있습니다.

- Kinesis는 Kinesis Data Streams 프로비저닝에 따라 QLDB에서 스트리밍되는 데이터를 제한할 수 있습니다. 예를 들어, 단일 Kinesis 데이터 스트림에 쓰는 QLDB 스트림이 여러 개 있고 QLDB의 요청률이 Kinesis 스트림 리소스의 용량을 초과하는 경우, 이 문제가 발생할 수 있습니다. Kinesis의 제한은 온디맨드 프로비저닝을 사용할 때도 15분 이내에 처리량이 이전 최고치의 두 배 이상으로 증가하는 경우 발생할 수 있습니다.

  Kinesis 지표 `WriteProvisionedThroughputExceeded`를 모니터링하여 초과 처리량을 측정할 수 있습니다. 자세한 내용과 가능한 해결 방법은 Kinesis Data Streams에서 제한 오류를 해결하려면 어떻게 해야 하나요? 섹션을 참조하세요.
- QLDB 스트림을 사용하면 시작 날짜 및 시간이 과거이고 종료 날짜 및 시간이 없는 무기한 스트림을 만들 수 있습니다. 설계상 QLDB는 지정된 시작 날짜 및 시간의 모든 이전 데이터가 성공적으로 전송된 후에만 Kinesis Data Streams에 새로 커밋된 데이터를 내보내기 시작합니다. 이 시나리오에서 추가 지연이 감지되면 이전 데이터가 전송될 때까지 기다리거나 나중의 시작 날짜 및 시간부터 스트림을 시작할 수도 있습니다.

데이터 스트림 시작하기

다음은 Kinesis Data Streams로 저널 데이터 스트리밍을 시작하는 데 필요한 단계에 대한 상위 수준의 개요입니다.

1. Kinesis Data Streams 리소스를 생성합니다. 지침은 Amazon Kinesis Data Streams 개발자 안내서의 데이터 스트림 만들기 및 업데이트를 참조하세요.
2. QLDB가 Kinesis 데이터 스트림에 대한 쓰기 권한을 맡을 수 있도록 허용하는 IAM 역할을 생성합니다. 지침은 QLDB의 스트림 권한 섹션을 참조하세요.
3. QLDB 저널 스트림을 생성합니다. 지침은 QLDB에서 스트림 생성 및 관리을 참조하세요.
4. 이전 섹션 스트림 사용에서 설명한 대로 Kinesis 데이터 스트림을 사용합니다. Kinesis 클라이언트 라이브러리 사용 방법을 보여주는 코드 예제는 또는 AWS Lambda을 참조하십시오. QLDB에서 스트림을 사용한 개발

QLDB에서 스트림 생성 및 관리

Amazon QLDB는 원장에서 Amazon Kinesis Data Streams로 전달되는 저널 데이터 스트림을 생성하고 관리하기 위한 API 작업을 제공합니다. QLDB 스트림은 저널에 커밋된 모든 문서 수정본을 캡처하여 Kinesis 데이터 스트림으로 보냅니다.

AWS Management Console, AWS SDK 또는 AWS Command Line Interface (AWS CLI) 를 사용하여 저널 스트림을 생성할 수 있습니다. 또한 AWS CloudFormation 템플릿을 사용하여 스트림을 생성할 수도 있습니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 AWS::QLDB::Stream리소스를 참조하십시오.

주제

스트림 파라미터

QLDB 저널 스트림을 생성하려면 다음 구성 파라미터를 제공해야 합니다.

원장 이름

저널 데이터를 Kinesis Data Streams로 스트리밍하려는 QLDB 원장.

스트림 이름

QLDB 저널 스트림에 할당할 명칭입니다. 사용자 정의 명칭은 스트림의 목적을 식별하고 나타내는 데 도움이 될 수 있습니다.

주어진 원장의 경우, 스트림 명칭은 다른 활성 스트림들 사이에서 고유해야 합니다. 스트림 이름은 Amazon QLDB 할당량 및 제한에 정의된 원장 이름과 동일한 명명 제약 조건을 갖습니다.

QLDB는 스트림 이름 외에도 사용자가 생성하는 각 QLDB 스트림에 스트림 ID를 할당합니다. 스트림 ID는 상태에 관계없이 주어진 원장의 모든 스트림에서 고유합니다.

현재 날짜 및 시간

스트리밍 저널 데이터를 시작할 시작 날짜 및 시간(경계값 포함)입니다. 이 값은 과거의 모든 날짜 및 시간일 수 있지만, 미래에는 사용할 수 없습니다.

현재 날짜 및 시간

스트림이 끝날 때를 지정하는 독점 날짜 및 시간입니다.

종료 시간이 없는 무제한 스트림을 생성하는 경우 스트림을 종료하려면 수동으로 취소해야 합니다. 지정된 종료 날짜 및 시간에 아직 도달하지 않은 유한한 활성 스트림을 취소할 수도 있습니다.

대상 Kinesis 데이터 스트림

스트림이 데이터 레코드를 쓰는 Kinesis Data Streams 대상 리소스입니다. Kinesis 데이터 스트림 생성 방법을 알아보려면 Amazon Kinesis Data Streams 개발자 안내서의 데이터 스트림 만들기 및 업데이트를 참조하세요.

> **Important**
>
> - 크로스 리전 및 크로스 계정 간 스트림은 지원되지 않습니다. 지정된 Kinesis 데이터 스트림은 원장과 동일한 AWS 리전 과 계정에 있어야 합니다.
> - Kinesis Data Streams의 레코드 집계는 기본적으로 활성화되어 있습니다. QLDB에서 단일 Kinesis Data Streams 레코드에 여러 데이터 레코드를 게시하여 API 직접 호출당 전송되는 레코드 수를 늘릴 수 있도록 합니다.
>
>   레코드 집계는 레코드 처리에 중요한 영향을 미치며 스트림 소비자의 집계를 해제해야 합니다. 자세한 내용을 알아보려면 Amazon Kinesis Data Streams 개발자 안내서의 KPL 주요 개념과 소비자 분해를 참조하세요.

IAM 역할

QLDB가 Kinesis 데이터 스트림에 대한 쓰기 권한을 맡을 수 있도록 허용하는 IAM 역할입니다. QLDB 콘솔을 사용하여 이 역할을 자동으로 생성하거나 IAM에서 수동으로 생성할 수 있습니다. 수동으로 생성하는 방법을 알아보려면 스트림 권한 섹션을 참조하세요.

저널 스트림을 요청할 때 QLDB에 역할을 전달하려면 IAM 역할 리소스에서 `iam:PassRole` 작업을 수행할 수 있는 권한이 있어야 합니다.

스트림 ARN

모든 QLDB 저널 스트림은 원장의 하위 리소스이며 Amazon 리소스 이름(ARN)으로 고유하게 식별됩니다. 다음은 이름이 `exampleLedger`인 원장의 스트림 ID가 `IiPT4brpZCqCq3f4MTHbYy`인 QLDB 스트림의 예제 ARN입니다.

```
arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/IiPT4brpZCqCq3f4MTHbYy
```

다음 섹션에서는 AWS Management Console을 사용하여 QLDB 스트림을 생성 및 취소하는 방법에 대해 설명합니다.

AWS Management Console

QLDB 콘솔을 사용하여 QLDB 스트림을 생성하거나 취소하려면 다음 단계에 따르세요.

태그를 만들려면(콘솔)

1. 에 AWS Management Console로그인하고 https://console.aws.amazon.com/qldb 에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 스트림을 선택합니다.
3. QLDB 스트림 생성을 선택합니다.
4. QLDB 스트림 생성 페이지에서 다음 설정을 입력합니다.
   - 스트림 이름 – QLDB 저널 스트림에 할당할 이름입니다.
   - 원장 – 스트리밍하려는 저널 데이터가 있는 원장입니다.
   - 시작 날짜 및 시간 – 저널 데이터 스트리밍을 시작할 포함되는 타임스탬프로, 협정 세계시(UTC) 기준입니다. 이 타임스탬프의 기본값은 현재 날짜 및 시간입니다. 이 날짜는 미래일 수 없으며 종료 날짜 및 시간보다 이전이어야 합니다.

- 종료 날짜 및 시간 – (선택 사항)스트림이 종료되는 시기를 지정하는 불포함되는 타임스탬프(UTC)입니다. 이 파라미터를 정의하지 않으면 취소하기 전까지 스트림이 무기한 실행됩니다.
- 대상 스트림 - 스트림이 데이터 레코드를 쓰는 Kinesis Data Streams 대상 리소스입니다. ARN 형식을 사용합니다.

  ```
  arn:aws:kinesis:aws-region:account-id:stream/kinesis-stream-name
  ```

  다음은 예입니다.

  ```
  arn:aws:kinesis:us-east-1:123456789012:stream/stream-for-qldb
  ```

  크로스 리전 및 크로스 계정 간 스트림은 지원되지 않습니다. 지정된 Kinesis 데이터 스트림은 AWS 리전 원장과 동일한 계정 내에 있어야 합니다.
- Kinesis Data Streams에서 레코드 집계 활성화 - (기본적으로 활성화됨) QLDB가 단일 Kinesis Data Streams 레코드에 여러 데이터 레코드를 게시하여 API 직접 호출당 전송되는 레코드 수를 늘릴 수 있도록 합니다.
- 서비스 액세스 – Kinesis 데이터 스트림에 QLDB 쓰기 권한을 부여하는 IAM 역할입니다.

  저널 스트림을 요청할 때 QLDB에 역할을 전달하려면 IAM 역할 리소스에서 *iam:PassRole* 작업을 수행할 수 있는 권한이 있어야 합니다.
  - 새 서비스 역할 생성 및 사용 - 콘솔에서 지정된 Kinesis 데이터 스트림에 필요한 권한을 가진 새 역할을 생성하도록 합니다.
  - 기존 서비스 역할 사용 – IAM에서 이 역할을 수동으로 생성하는 방법을 알아보려면 스트림 권한 섹션을 참조하세요.
- 태그 – (선택 사항) 태그를 키 값 페어로 연결하여 메타데이터를 역할에 추가합니다. 태그를 구성하고 식별하는 데 도움이 되는 태그를 스트림에 추가할 수 있습니다. 자세한 내용은 Amazon QLDB 리소스 태그 지정을 참조하십시오.

  태그 추가를 선택한 다음 키-값 쌍을 적절히 입력합니다.

5. 원하는 대로 설정되었으면 QLDB 스트림 생성을 선택합니다.

   요청이 성공적으로 제출되면 콘솔이 기본 스트림 페이지로 돌아가고 QLDB 스트림이 현재 상태와 함께 나열됩니다.

6. 스트림이 활성화되면 Kinesis를 사용하여 소비자 애플리케이션으로 스트림 데이터를 처리합니다.

https://console.aws.amazon.com/kinesis에서 Kinesis Data Streams 콘솔을 엽니다.

스트림 데이터 레코드의 형식에 대한 자세한 내용은 Kinesis의 QLDB 스트림 레코드 섹션을 참조하세요.

오류가 발생하는 스트림을 처리하는 방법을 알아보려면 손상된 스트림 처리 섹션을 참조하세요.

스트림을 취소하려면(콘솔)

QLDB 스트림을 취소한 후에는 다시 시작할 수 없습니다. Kinesis Data Streams로의 데이터 전송을 재개하려면 새 QLDB 스트림을 생성하면 됩니다.

1. https://console.aws.amazon.com/qldb에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 스트림을 선택합니다.
3. QLDB 스트림 목록에서 취소하려는 활성 스트림을 선택합니다.
4. 스트림 취소를 선택합니다. 제공된 상자에 **`cancel stream`**를 입력하여 확인합니다.

QLDB API를 SDK와 함께 AWS 사용하거나 저널 스트림을 생성 및 관리하는 방법에 대한 자세한 내용은 을 참조하십시오. AWS CLI QLDB에서 스트림을 사용한 개발

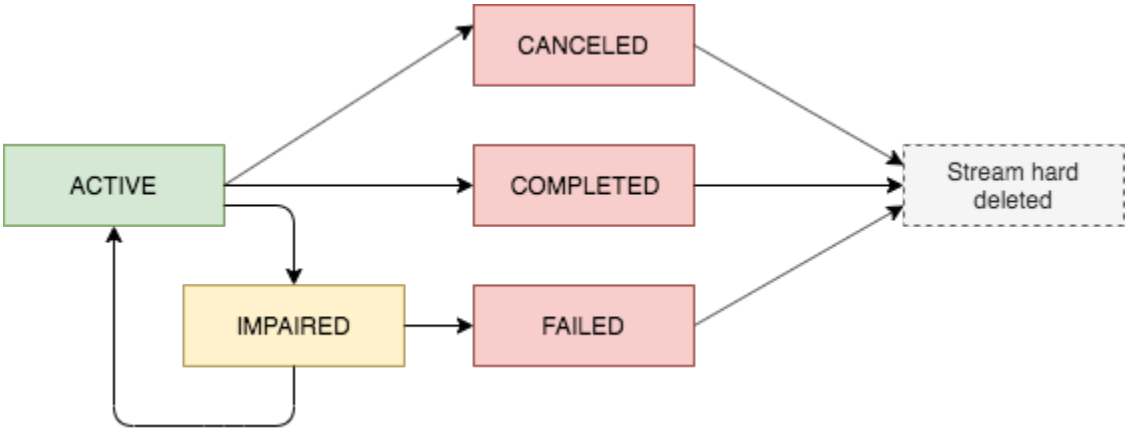
스트림 상태

QLDB 스트림의 상태는 다음 중 하나일 수 있습니다.

- `ACTIVE` – 현재 스트리밍 중이거나 데이터 스트리밍을 위해 대기 중입니다(종료 시간이 없는 무제한 스트림용).
- `COMPLETED` – 지정된 시간 범위 내에 모든 저널 블록 스트리밍을 성공적으로 마쳤습니다. 이것은 종료 상태입니다.
- `CANCELED` – 지정된 종료 시간 전에 사용자 요청으로 종료되었으며, 더 이상 데이터를 스트리밍하지 않습니다. 이것은 종료 상태입니다.
- `IMPAIRED` – 사용자 조치가 필요한 오류 때문에 Kinesis에 레코드를 쓸 수 없습니다. 이는 복구 가능한 터미널이 아닌 상태입니다.

  1시간 이내에 오류를 해결하면 스트림은 자동으로 `ACTIVE` 상태로 전환됩니다. 1시간 후에도 오류가 해결되지 않으면 스트림은 자동으로 `FAILED` 상태로 전환됩니다.
- `FAILED` – 오류로 인해 Kinesis에 레코드를 쓸 수 없으며 복구할 수 없는 터미널 상태입니다.

다음 다이어그램은 QLDB 스트림 리소스가 상태 사이를 전환하는 방법을 보여줍니다.



터미널 스트림 만료

터미널 상태(CANCELED, COMPLETED, FAILED)에 있는 스트림 리소스에는 7일의 보존 기간이 적용됩니다. 이 한도가 만료되면 해당 파일은 자동으로 영구 삭제됩니다.

터미널 스트림이 삭제된 후에는 더 이상 QLDB 콘솔 또는 QLDB API를 사용하여 스트림 리소스를 설명하거나 나열할 수 없습니다.

손상된 스트림 처리

스트림에 오류가 발생하면 먼저 IMPAIRED 상태로 전환됩니다. QLDB는 최대 1시간 동안 IMPAIRED 스트림을 계속 재시도합니다.

1시간 이내에 오류를 해결하면 스트림은 자동으로 ACTIVE 상태로 전환됩니다. 1시간 후에도 오류가 해결되지 않으면 스트림은 자동으로 FAILED 상태로 전환됩니다.

손상되거나 장애가 발생한 스트림의 오류 원인은 다음 오류 원인 중 하나일 수 있습니다.

- KINESIS_STREAM_NOT_FOUND – 대상 Kinesis Data Streams 리소스가 존재하지 않습니다. QLDB 스트림 요청에서 제공한 Kinesis 데이터 스트림이 올바른지 확인하세요. 그런 다음 Kinesis로 이동하여 지정한 데이터 스트림을 생성합니다.
- IAM_PERMISSION_REVOKED – QLDB에 지정된 Kinesis 데이터 스트림에 데이터 레코드를 쓸 수 있는 충분한 권한이 없습니다. 지정된 Kinesis 데이터 스트림에 대해 QLDB 서비스(qldb.amazonaws.com)에 다음 작업에 대한 권한을 부여하는 정책을 정의했는지 확인하세요.
  - kinesis:PutRecord
  - kinesis:PutRecords
  - kinesis:DescribeStream
  - kinesis:ListShards

손상된 스트림 모니터링

스트림이 손상되면 QLDB 콘솔은 스트림에 대한 세부 정보와 발생한 오류를 보여주는 배너를 표시합니다. 또한 `DescribeJournalKinesisStream` API 작업을 사용하여 스트림의 상태와 기본 오류 원인을 확인할 수 있습니다.

또한 CloudWatch Amazon을 사용하여 스트림의 `IsImpaired` 지표를 모니터링하는 경보를 생성할 수 있습니다. 를 사용하여 QLDB CloudWatch 메트릭을 모니터링하는 방법에 대한 자세한 내용은 을 참조하십시오. Amazon QLDB 차원 및 지표

QLDB에서 스트림을 사용한 개발

이 섹션에서는 AWS SDK와 함께 사용하거나 Amazon QLDB에서 저널 스트림을 생성 및 관리하는 AWS CLI 데 사용할 수 있는 API 작업을 요약합니다. 또한 이러한 작업을 시연하고 Kinesis Client Library(KCL) 또는 AWS Lambda 를 사용하거나 스트림 소비자를 구현하는 샘플 애플리케이션에 대해서도 설명합니다.

KCL을 사용하여 Amazon Kinesis Data Streams의 소비자 애플리케이션을 빌드할 수 있습니다. KCL은 하위 수준의 Kinesis Data Streams API에 유용한 추상화를 제공하여 코딩을 단순화합니다. KCL에 대한 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 Kinesis Client Library를 사용을 참조하세요.

목차

QLDB 저널 스트림 API

애플리케이션 프로그램에서 사용할 수 있는 다음과 같은 기능이 QLDB API에서 제공됩니다.

- `StreamJournalToKinesis` – 주어진 QLDB 원장에 대한 저널 스트림을 생성합니다. 스트림은 원장의 저널에 커밋된 모든 문서 개정본을 캡처하고 지정된 Kinesis Data Streams 리소스로 데이터를 전송합니다.

  - Kinesis Data Streams의 레코드 집계는 기본적으로 활성화되어 있습니다. QLDB에서 단일 Kinesis Data Streams 레코드에 여러 데이터 레코드를 게시하여 API 직접 호출당 전송되는 레코드 수를 늘릴 수 있도록 합니다.

    레코드 집계는 레코드 처리에 중요한 영향을 미치며 스트림 소비자의 집계를 해제해야 합니다. 자세한 내용을 알아보려면 Amazon Kinesis Data Streams 개발자 안내서의 KPL 주요 개념과 소비자 분해를 참조하세요.

- `DescribeJournalKinesisStream` - 주어진 QLDB 저널 스트림에 대한 세부 정보를 반환합니다. 출력에는 ARN, 스트림 이름, 현재 상태, 생성 시간, 원래 스트림 생성 요청의 파라미터가 포함됩니다.
- `ListJournalKinesisStreamsForLedger` – 주어진 원장의 모든 QLDB 저널 스트림 설명자 목록을 반환합니다. 각 스트림 설명자의 출력에는 `DescribeJournalKinesisStream`에서 반환한 것과 동일한 세부 정보가 포함됩니다.
- `CancelJournalKinesisStream` – 주어진 QLDB 저널 스트림을 종료합니다. 스트림을 취소하려면 스트림의 현재 상태가 `ACTIVE`이어야 합니다.

  스트리밍을 취소한 후에는 다시 시작할 수 없습니다. Kinesis Data Streams로의 데이터 전송을 재개하려면 새 QLDB 스트림을 생성하면 됩니다.

이러한 API작업 설명 전체를 보려면 Amazon QLDB API 참조 섹션을 참조하세요.

를 사용하여 저널 스트림을 생성하고 관리하는 방법에 대한 자세한 내용은 명령 참조를 AWS CLI참조하십시오.AWS CLI

샘플 애플리케이션

QLDB는 저널 스트림을 사용한 다양한 작업을 시연하는 샘플 애플리케이션을 제공합니다. 이러한 애플리케이션은 AWS 샘플 GitHub 사이트의 오픈 소스입니다.

주제

- 기본 작업(Java)
- OpenSearch 서비스와의 통합 (Python)
- Amazon SNS 및 Amazon SQS와의 통합(Python)

기본 작업(Java)

QLDB 저널 스트림의 기본 작업을 보여주는 자바 코드 예제는 aws-samples/ -java 리포지토리를 참조하십시오 GitHub . amazon-qldb-dmv-sample 이 샘플 애플리케이션을 다운로드하여 설치하는 방법에 대한 자세한 내용은 Amazon QLDB Java 샘플 애플리케이션 설치 섹션을 참조하세요.

> **Note**
>
> 애플리케이션을 설치한 후에는 원장을 만들기 위해 Java 자습서의 1단계로 진행하지 마세요. 이 스트리밍용 샘플 애플리케이션은 `vehicle-registration` 원장을 자동으로 생성합니다.

이 샘플 애플리케이션은 다음 모듈을 포함하여 Java 자습서와 관련 종속성의 전체 소스 코드를 패키징합니다.

- AWS SDK for Java – 원장, QLDB 저널 스트림, Kinesis 데이터 스트림을 비롯한 QLDB 및 Kinesis Data Streams 리소스를 모두 생성하고 삭제합니다.
- Java용 Amazon QLDB 드라이버 – PartiQL 문을 사용하여 원장에 대한 데이터 트랜잭션을 실행합니다(테이블 생성 및 문서 삽입 포함).
- Kinesis Client Library – Kinesis 데이터 스트림의 데이터를 사용하고 처리합니다.

코드 실행

이 StreamJournal클래스에는 다음 작업을 보여주는 자습서 코드가 포함되어 있습니다.

1. `vehicle-registration`라는 이름의 원장을 만들고 테이블을 만든 다음 샘플 데이터와 함께 로드합니다.

   > **Note**
   >
   > 이 코드를 실행하기 전에 이미 `vehicle-registration`라는 이름이 지정된 활성화된 원장이 없는지 확인하세요.

2. Kinesis 데이터 스트림, QLDB가 Kinesis 데이터 스트림에 대한 쓰기 권한을 맡을 수 있도록 허용하는 Kinesis 데이터 스트림, IAM 역할을 생성합니다.
3. KCL을 사용하여 Kinesis 데이터 스트림을 처리하고 각 QLDB 데이터 레코드를 기록하는 스트림 리더를 시작할 수 있습니다.

4. 스트림 데이터를 사용하여 `vehicle-registration` 샘플 원장의 해시 체인을 확인할 수 있습니다.
5. 스트림 리더를 중지하고, QLDB 저널 스트림을 취소하고, 원장을 삭제하고, Kinesis 데이터 스트림을 삭제하여 모든 리소스를 정리합니다.

`StreamJournal` 자습서 코드를 실행하려면 프로젝트 루트 디렉터리에서 다음 Gradle 명령을 입력합니다.

```
./gradlew run -Dtutorial=streams.StreamJournal
```

OpenSearch 서비스와의 통합 (Python)

QLDB 스트림을 OpenSearch Amazon 서비스와 통합하는 방법을 보여주는 Python 샘플 애플리케이션은 aws-samples/ - 리포지토리를 GitHub 참조하십시오. amazon-qldb-streaming-amazon opensearch-service-sample-python 이 애플리케이션은 AWS Lambda 함수를 사용하여 Kinesis Data Streams 소비자를 구현합니다.

다음 `git` 명령을 입력하여 리포지토리를 복제합니다.

```
git clone https://github.com/aws-samples/amazon-qldb-streaming-amazon-opensearch-service-sample-python.git
```

샘플 애플리케이션을 실행하려면 자세한 지침은 GitHub README를 참조하십시오.

Amazon SNS 및 Amazon SQS와의 통합(Python)

QLDB 스트림을 아마존 심플 알림 서비스 (Amazon SNS) 와 통합하는 방법을 보여주는 Python 샘플 애플리케이션은 aws-samples/ - 리포지토리를 참조하십시오. GitHub amazon-qldb-streams-dmv sample-lambda-python

이 애플리케이션은 AWS Lambda 함수를 사용하여 Kinesis Data Streams 소비자를 구현합니다. Amazon Simple Queue Service(Amazon SQS) 대기열이 구독되어 있는 Amazon SNS 주제로 메시지를 전송합니다.

다음 `git` 명령을 입력하여 리포지토리를 복제합니다.

```
git clone https://github.com/aws-samples/amazon-qldb-streams-dmv-sample-lambda-python.git
```

샘플 애플리케이션을 실행하려면 자세한 지침은 GitHub README를 참조하십시오.

Kinesis의 QLDB 스트림 레코드

Amazon QLDB 스트림은 주어진 Amazon Kinesis Data Streams 리소스에 control, 블록 요약, 수정 세부 정보 등 세 가지 유형의 데이터 레코드를 기록합니다. 세 가지 레코드 유형은 모두 Amazon Ion 형식의 이진수 표현으로 작성됩니다.

제어 레코드는 QLDB 스트림의 시작과 완료를 나타냅니다. 수정본이 저널에 커밋될 때마다 QLDB 스트림은 연결된 모든 저널 블록 데이터를 블록 요약 및 수정 세부 정보 레코드에 기록합니다.

세 가지 레코드 유형은 다형성입니다. 이들은 모두 QLDB 스트림 ARN, 레코드 유형 및 레코드 페이로드를 포함하는 공통 최상위 레코드로 구성됩니다. 이 최상위 수준 레코드의 형식은 다음과 같습니다.

```
{
  qldbStreamArn: string,
  recordType: string,
  payload: {
    //control | block summary | revision details record
  }
}
```

`recordType` 필드는 다음 세 가지 값 중 하나를 가질 수 있습니다.

- `CONTROL`
- `BLOCK_SUMMARY`
- `REVISION_DETAILS`

다음 섹션에서는 각 개별 페이로드 레코드의 형식과 콘텐츠를 설명합니다.

> **Note**
>
> QLDB는 모든 스트림 레코드를 Amazon Ion의 이진수 표현으로 Kinesis Data Streams에 씁니다. 다음 예는 레코드 콘텐츠를 읽기 쉬운 형식으로 설명하기 위해 Ion의 텍스트 표현으로 제공됩니다.

주제

제어 레코드

QLDB 스트림은 제어 레코드를 기록하여 시작 및 완료 이벤트를 표시합니다. 다음은 각 `controlRecordType`에 대한 샘플 데이터가 포함된 제어 레코드의 예입니다.

- `CREATED` – QLDB 스트림이 새로 생성한 스트림이 활성 상태임을 나타내기 위해 Kinesis에 쓰는 첫 번째 레코드입니다.

```
{
  qldbStreamArn:"arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/
IiPT4brpZCqCq3f4MTHbYy",
  recordType:"CONTROL",
  payload:{
    controlRecordType:"CREATED"
  }
}
```

- `COMPLETED` – QLDB 스트림이 스트림이 지정된 종료 날짜 및 시간에 도달했음을 나타내기 위해 Kinesis에 쓰는 마지막 레코드입니다. 스트림을 취소하면 이 레코드가 기록되지 않습니다.

```
{
  qldbStreamArn:"arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/
IiPT4brpZCqCq3f4MTHbYy",
  recordType:"CONTROL",
  payload:{
    controlRecordType:"COMPLETED"
  }
}
```

블록 요약 레코드

블록 요약 레코드는 문서 수정본이 커밋되는 저널 블록을 나타냅니다. 블록은 트랜잭션 중에 QLDB 저널에 커밋되는 객체입니다.

블록 요약 레코드의 페이로드에는 블록을 커밋한 트랜잭션의 블록 주소, 타임스탬프 및 기타 메타데이터가 포함됩니다. 또한 블록 내 수정본의 요약 속성과 이를 커밋한 PartiQL 문도 포함됩니다. 다음은 샘플 데이터가 포함된 블록 요약 레코드의 예입니다.

> **Note**
>
> 이 블록 요약 예제는 정보 제공 목적으로만 제공됩니다. 표시된 해시는 실제 계산된 해시 값이 아닙니다.

```
{
  qldbStreamArn:"arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/
IiPT4brpZCqCq3f4MTHbYy",
  recordType:"BLOCK_SUMMARY",
  payload:{
    blockAddress:{
      strandId:"ElYL30RGoqrFCbbaQn3K6m",
      sequenceNo:60807
    },
    transactionId:"9RWohCo7My4GGkxRETAJ6M",
    blockTimestamp:2019-09-18T17:00:14.601000001Z,
    blockHash:{{6Pk9KDYJd38ci09oaHxx0D2grtgh4QBBqbDS6i9quX8=}},
    entriesHash:{{r5YoH6+NXDXxgoRzPREGAWJfn73KlZE0eTfbTxZWUDU=}},
    previousBlockHash:{{K3ti0Agk7DEponywKcQCPRYVHb5RuyxdmQFTfrloptA=}},
    entriesHashList:[
      {{pbzvz6ofJC7mD2jvgfyrY/VtR01zIZHoWy8T1VcxlGo=}},
      {{k2brC23DLMercmiOWHiURaGwHu0mQtLzdNPuviE2rcs=}},
      {{hvw1EV8k4oOkIO36kblO/+UUSFUQqCanKuDGr0aP9nQ=}},
      {{ZrLbkyzDcpJ9KWsZMZqRuKUkG/czLIJ4US+K5E31b+Q=}}
    ],
    transactionInfo:{
      statements:[
        {
          statement:"SELECT * FROM Person WHERE GovId = ?",
          startTime:2019-09-18T17:00:14.587Z,
          statementDigest:{{p4Dn0DiuYD3Xm9UQQ75YLwmoMbSfJmopOmTfMnXs26M=}}
        },
        {
          statement:"INSERT INTO Person ?",
          startTime:2019-09-18T17:00:14.594Z,
          statementDigest:{{klMLkLfa5VJqk6JUPtHkQpOsDdG4HmuUaq/VaApQflU=}}
        },
```

```
        {
          statement:"INSERT INTO VehicleRegistration ?",
          startTime:2019-09-18T17:00:14.598Z,
          statementDigest:{{B0gO9BWVNrzRYFoe7t+GVLpJ6uZcLKf5t/chkfRhspI=}}
        }
      ],
      documents:{
        '7z2OpEBgVCvCtwvx4a2JGn':{
          tableName:"Person",
          tableId:"LSkFkQvkIOjCmpTZpkfpn9",
          statements:[1]
        },
        'K0FpsSLpydLDr7hi6KUzqk':{
          tableName:"VehicleRegistration",
          tableId:"Ad3A07z0ZffC7Gpso7BXyO",
          statements:[2]
        }
      }
    },
    revisionSummaries:[
      {
        hash:{{uDthuiqSy4FwjZssyCiyFd90XoPSlIwomHBdF/OrmkE=}},
        documentId:"7z2OpEBgVCvCtwvx4a2JGn"
      },
      {
        hash:{{qJID/amu0gN3dpG5Tg0FfIFTh/U5yFkfT+g/O6k5sPM=}},
        documentId:"K0FpsSLpydLDr7hi6KUzqk"
      }
    ]
  }
}
```

revisionSummaries 필드의 일부 수정본에는 documentId가 없을 수 있습니다. 이는 사용자 데이터를 포함하지 않는 내부 전용 시스템 수정본입니다. 이러한 수정본의 해시는 저널의 전체 해시 체인의 일부이기 때문에 QLDB 스트림은 해당 블록 요약 레코드에 이러한 수정본을 포함합니다. 암호화 확인에는 전체 해시 체인이 필요합니다.

다음 섹션에 설명된 대로 문서 ID가 있는 수정본만 별도의 수정본 세부 정보 레코드에 게시됩니다.

수정본 세부 정보 레코드

수정 세부 정보 레코드는 저널에 커밋된 문서 수정본을 나타냅니다. 페이로드에는 수정본의 커밋된 보기의 모든 속성이 관련 테이블 이름 및 테이블 ID와 함께 포함되어 있습니다. 다음은 샘플 데이터가 포함된 수정 레코드의 예입니다.

```
{
  qldbStreamArn:"arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/
IiPT4brpZCqCq3f4MTHbYy",
  recordType:"REVISION_DETAILS",
  payload:{
    tableInfo:{
      tableName:"VehicleRegistration",
      tableId:"Ad3A07z0ZffC7Gpso7BXyO"
    },
    revision:{
      blockAddress:{
        strandId:"ElYL30RGoqrFCbbaQn3K6m",
        sequenceNo:60807
      },
      hash:{{qJID/amu0gN3dpG5Tg0FfIFTh/U5yFkfT+g/O6k5sPM=}},
      data:{
        VIN:"1N4AL11D75C109151",
        LicensePlateNumber:"LEWISR261LL",
        State:"WA",
        City:"Seattle",
        PendingPenaltyTicketAmount:90.25,
        ValidFromDate:2017-08-21,
        ValidToDate:2020-05-11,
        Owners:{
          PrimaryOwner:{PersonId:"7z2OpEBgVCvCtwvx4a2JGn"},
          SecondaryOwners:[]
        }
      },
      metadata:{
        id:"K0FpsSLpydLDr7hi6KUzqk",
        version:0,
        txTime:2019-09-18T17:00:14.602Z,
        txId:"9RWohCo7My4GGkxRETAJ6M"
      }
    }
  }
```

```
}
```

중복 및 out-of-order 레코드 처리

QLDB 스트림은 Kinesis Data Streams에 중복 out-of-order 및 레코드를 게시할 수 있습니다. 따라서 소비자 애플리케이션은 이러한 시나리오를 식별하고 처리하기 위해 자체 로직을 구현해야 할 수 있습니다. 블록 요약 및 수정 세부 정보 레코드에는 이러한 목적으로 사용할 수 있는 필드가 포함되어 있습니다. 이러한 필드를 다운스트림 서비스의 기능과 함께 사용하면 고유한 ID와 레코드의 엄격한 순서를 모두 나타낼 수 있습니다.

QLDB를 인덱스와 OpenSearch 통합하여 문서에 대한 전체 텍스트 검색 기능을 제공하는 스트림을 예로 들어 보겠습니다. 이 사용 사례에서는 문서의 오래된 (out-of-order) 수정본을 인덱싱하지 않도록 해야 합니다. 주문 및 중복 제거를 적용하려면 개정 세부 정보 레코드의 문서 ID 및 버전 필드와 함께 의 OpenSearch 문서 ID 및 외부 버전 필드를 사용할 수 있습니다.

QLDB를 OpenSearch Amazon Service와 통합하는 샘플 애플리케이션의 중복 제거 로직 예는 aws-samples/ - 리포지토리를 참조하십시오. GitHub amazon-qldb-streaming-amazon opensearch-service-sample-python

QLDB의 스트림 권한

Amazon QLDB 스트림을 생성하기 전에 지정된 Amazon Kinesis Data Streams 리소스에 대한 쓰기 권한을 QLDB에 제공해야 합니다. Kinesis 스트림의 서버 측 암호화에 고객 관리형 AWS KMS key 를 사용하는 경우 QLDB에 지정된 대칭 암호화 키를 사용할 수 있는 권한도 제공해야 합니다. Kinesis Data Streams는 비대칭 KMS 키를 지원하지 않습니다.

QLDB 스트림에 필요한 권한을 제공하려면 적절한 권한 정책을 사용하여 QLDB가 IAM 서비스 역할을 맡도록 합니다. 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용자 설명서의 역할을 생성하여 AWS 서비스에게 권한 위임을 참조하세요.

> **Note**
>
> 저널 스트림을 요청할 때 QLDB에 역할을 전달하려면 IAM 역할 리소스에서 `iam:PassRole` 작업을 수행할 수 있는 권한이 있어야 합니다. 이는 QLDB 스트림 하위 리소스에 대한 `qldb:StreamJournalToKinesis` 권한에 추가됩니다.

IAM을 사용하여 QLDB에 대한 액세스를 제어하는 방법을 알아보려면 Amazon QLDB에서 IAM을 사용하는 방법 섹션을 참조하세요. QLDB 정책 예제는 Amazon QLDB의 자격 증명 기반 정책 예 섹션을 참조하세요.

이 예시에서는 QLDB가 사용자 대신 Kinesis 데이터 스트림에 데이터 레코드를 기록하도록 허용하는 역할을 생성합니다. 자세한 내용은 IAM 사용자 설명서의 역할을 생성하여 AWS 서비스에게 권한 위임을 참조하세요.

QLDB 저널을 처음으로 스트리밍하는 경우 먼저 다음을 수행하여 적절한 정책을 사용하여 IAM 역할을 생성해야 합니다. AWS 계정 또는 QLDB 콘솔을 사용하여 자동으로 역할을 생성할 수 있습니다. 또는 이전에 생성한 역할을 선택할 수 있습니다.

주제

- 권한 정책 생성
- IAM 역할 생성

권한 정책 생성

QLDB 스트림에 대한 권한 정책을 생성하려면 다음 단계를 완료합니다. 이 예제는 지정된 Kinesis 데이터 스트림에 데이터 레코드를 쓸 수 있는 권한을 QLDB에 부여하는 Kinesis Data Streams 정책을 보여줍니다. 해당하는 경우 이 예제에서는 QLDB가 대칭 암호화 KMS 키를 사용하도록 허용하는 키 정책도 보여줍니다.

Kinesis Data Streams 정책에 대한 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 IAM을 사용하여 Amazon Kinesis Data Streams 리소스에 대한 액세스 제어 및 사용자 생성 KMS 키를 사용할 수 있는 권한을 참조하세요. AWS KMS 키 정책에 대해 자세히 알아보려면 개발자 안내서의 키 정책 사용을 참조하십시오. AWS KMSAWS Key Management Service

> **Note**
>
> Kinesis 데이터 스트림과 KMS 키는 모두 QLDB AWS 리전 원장과 동일한 계정 내에 있어야 합니다.

JSON 정책 편집기를 사용하여 정책을 생성하려면

1. 에 AWS Management Console 로그인하고 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 왼쪽의 탐색 열에서 정책을 선택합니다.

   정책을 처음으로 선택하는 경우 관리형 정책 소개 페이지가 나타납니다. 시작하기를 선택합니다.
3. 페이지 상단에서 정책 생성을 선택합니다.
4. JSON 탭을 선택합니다.
5. JSON 정책 문서를 입력합니다.
   - Kinesis 스트림의 서버 측 암호화에 고객 관리형 KMS 키를 사용하는 경우 다음 예제 정책 문서를 사용하세요. *# ### ##### ### us-east-1, 123456789012 # kinesis-stream-name1234abcd-12ab-34cd-56ef-1234567890ab# ### ## ### ######.*

     ```
     {
         "Version": "2012-10-17",
         "Statement": [
             {
                 "Sid": "QLDBStreamKinesisPermissions",
                 "Action": [ "kinesis:PutRecord*", "kinesis:DescribeStream",
      "kinesis:ListShards" ],
                 "Effect": "Allow",
                 "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/kinesis-
     stream-name"
             },
             {
                 "Sid": "QLDBStreamKMSPermission",
                 "Action": [ "kms:GenerateDataKey" ],
                 "Effect": "Allow",
                 "Resource": "arn:aws:kms:us-
     east-1:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
             }
         ]
     }
     ```
   - 다음은 정책 문서의 예입니다. 이 정책을 사용하려면 예제에서 *us-east-1,* 123456789012를 사용자 고유의 정보로 바꾸십시오. *kinesis-stream-name*

     ```
     {
         "Version": "2012-10-17",
     ```

```
    "Statement": [
        {
            "Sid": "QLDBStreamKinesisPermissions",
            "Action": [ "kinesis:PutRecord*", "kinesis:DescribeStream",
 "kinesis:ListShards" ],
            "Effect": "Allow",
            "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/kinesis-
stream-name"
        }
    ]
}
```

6. 정책 검토를 선택합니다.

> **Note**
>
> 언제든지 시각적 편집기 및 JSON 탭을 전환할 수 있습니다. 그러나 변경을 수행하거나 시각적 편집기 탭에서 정책 검토를 선택한 경우 IAM은 시각적 편집기에 최적화되도록 정책을 재구성할 수 있습니다. 자세한 내용은 IAM 사용자 설명서의 정책 재구성을 참조하세요.

7. 정책 검토 페이지에서 생성하려는 정책의 이름과 설명(선택 사항)을 입력합니다. 정책 요약을 검토하여 정책이 부여한 권한을 확인합니다. 그런 다음 정책 생성을 선택하여 작업을 저장합니다.

IAM 역할 생성

QLDB 스트림에 대한 권한 정책을 생성한 후 IAM 역할을 생성하고 정책을 여기에 연결할 수 있습니다.

QLDB에 대한 서비스 역할을 생성하는 방법(IAM 콘솔)

1. 로그인하고 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다. AWS Management Console
2. IAM 콘솔의 탐색 창에서 역할을 선택하고 역할 생성을 선택합니다.
3. 신뢰할 수 있는 엔터티 유형에 AWS 서비스를 선택합니다.
4. 서비스 또는 사용 사례의 경우 QLDB를 선택한 다음 QLDB 사용 사례를 선택합니다.
5. 다음을 선택합니다.
6. 앞에서 생성한 정책 옆의 상자를 선택합니다.
7. (선택 사항)권한 경계로서 설정됩니다. 이는 서비스 역할에서 가능한 고급 기능이며 서비스 링크된 역할은 아닙니다.

a. 권한 경계 설정 섹션을 열고 최대 역할 권한을 관리하기 위한 권한 경계 사용을 선택합니다.

IAM에는 계정의 AWS 관리형 및 고객 관리형 정책 목록이 포함되어 있습니다.

b. 정책을 선택하여 권한 경계를 사용하세요.

8. 다음을 선택합니다.
9. 역할의 목적을 식별하는 데 도움이 되는 역할 이름이나 역할 이름 접미사를 입력합니다.

> **Important**
>
> 역할 이름을 지정할 때는 다음 사항에 유의하세요.
>
> - 역할 이름은 사용자 내에서 고유해야 AWS 계정하며 대소문자를 구분하여 고유할 수 없습니다.
>
> 예를 들어, 이름이 **PRODROLE**과 **prodrole**, 두 가지로 지정된 역할을 만들지 마십시오. 역할 이름이 정책 또는 ARN의 일부로 사용되는 경우 역할 이름은 대소문자를 구분합니다. 그러나 로그인 프로세스와 같이 콘솔에서 역할 이름이 고객에게 표시되는 경우에는 역할 이름이 대소문자를 구분하지 않습니다.
> - 다른 엔터티가 역할을 참조할 수 있기 때문에 역할이 생성된 후에는 역할 이름을 편집할 수 없습니다.

10. (선택 사항)설명에 역할에 대한 설명을 입력합니다.
11. (선택 사항) 역할에 대한 사용 사례와 권한을 편집하려면 1단계: 신뢰할 수 있는 엔터티 선택 또는 2단계: 권한 추가 섹션에서 편집을 선택합니다.
12. (선택 사항) 태그를 키-값 페어로 연결하여 역할을 식별, 구성 또는 검색합니다. IAM에서 태그 사용에 대한 자세한 내용을 알아보려면 IAM 사용 설명서의 IAM 리소스에 태그 지정을 참조하세요.
13. 역할을 검토한 다음 역할 생성을 선택합니다.

다음 JSON 문서는 QLDB가 특정 권한이 부여된 IAM 역할을 맡도록 허용하는 신뢰 정책의 예입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "qldb.amazonaws.com"
```

```
            },
            "Action": [ "sts:AssumeRole" ],
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:qldb:us-
east-1:123456789012:stream/myExampleLedger/*"
                },
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                }
            }
        }
    ]
}
```

> **Note**
>
> 이 신뢰 정책 예시에서 `aws:SourceArn` 및 `aws:SourceAccount` 전역 조건 컨텍스트 키를 사용하여 혼동된 대리자 문제를 방지하는 방법을 보여줍니다. 이 신뢰 정책을 사용하면 QLDB는 원장 `myExampleLedger`에 대한 계정 `123456789012`의 모든 QLDB 스트림에 대한 역할만 맡을 수 있습니다.
> 자세한 내용은 교차 서비스 혼동된 대리자 예방을 참조하십시오.

IAM 역할을 생성한 후 QLDB 콘솔로 돌아가서 QLDB 스트림 생성 페이지를 새로 고쳐 새 역할을 찾을 수 있도록 합니다.

QLDB의 저널 스트림에 대한 일반적인 오류

이 섹션에서는 저널 스트림 요청에 대해 Amazon QLDB에서 발생하는 런타임 오류를 설명합니다.

다음은 서비스에서 반환하는 일반적인 예외 목록입니다. 각 예외에는 특정 오류 메시지와 가능한 해결 방법에 대한 간단한 설명 및 제안이 포함됩니다.

AccessDeniedException

###: ###: UserArn# ### ## ### ### ### ####. iam: PassRole on ###: RoLearn

IAM 역할을 QLDB 서비스에 전달할 권한이 없습니다. QLDB에는 모든 저널 스트림 요청에 대한 역할이 필요하며 이 역할을 QLDB에 전달할 수 있는 권한이 있어야 합니다. 이 역할은 지정된 Amazon Kinesis Data Streams 리소스에 대한 쓰기 권한을 QLDB에 제공합니다.

QLDB 서비스(`qldb.amazonaws.com`)의 지정된 IAM 역할 리소스에서 `PassRole` API 작업을 수행할 권한을 부여하는 IAM 정책을 정의했는지 확인하세요. 정책 예제는 Amazon QLDB의 자격 증명 기반 정책 예을 참조하십시오.

IllegalArgumentException

메시지: QLDB encountered an error validating Kinesis Data Streams: Response from Kinesis: *errorCodeerrorMessage*

이 오류의 가능한 원인은 제공된 Kinesis Data Streams 리소스가 존재하지 않기 때문입니다. 또는 QLDB에 지정된 Kinesis 데이터 스트림에 데이터 레코드를 쓸 수 있는 충분한 권한이 없습니다.

스트림 요청에서 제공하는 Kinesis 데이터 스트림이 올바른지 확인하세요. 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 데이터 스트림 생성 및 업데이트를 참조하세요.

또한 다음 작업에 QLDB 서비스(`qldb.amazonaws.com`) 권한을 부여하는 지정된 Kinesis 데이터 스트림에 대한 정책을 정의했는지 확인합니다. 자세한 정보는 스트림 권한을 참조하세요.

- `kinesis:PutRecord`
- `kinesis:PutRecords`
- `kinesis:DescribeStream`
- `kinesis:ListShards`

IllegalArgumentException

메시지: Kinesis 구성을 검증하는 동안 Kinesis 데이터 스트림에서 예상치 못한 응답이 발생했습니다. Kinesis의 응답: *errorCode errorMessage*

제공된 Kinesis 오류 응답으로 인해 제공된 Kinesis 데이터 스트림에 데이터 레코드를 쓰려는 시도가 실패했습니다. 발생 가능한 원인에 대한 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 Amazon Kinesis Data Streams 생산자 문제 해결을 참조하십시오.

IllegalArgumentException

메시지: 시작 날짜는 종료 날짜보다 이후일 수 없습니다.

`InclusiveStartTime` 및 `ExclusiveEndTime`는 ISO 8601 날짜 및 시간 형식이어야 하며 협정 세계시(UTC)여야 합니다.

IllegalArgumentException

메시지: 종료일은 미래일 수 없습니다

`InclusiveStartTime` 및 `ExclusiveEndTime`은 UTC 기준의 `ISO 8601` 날짜 및 시간 형식이어야 합니다.

LimitExceededException

메시지: Kinesis Data Streams에 대한 동시 실행 저널 스트림 제한인 5개를 초과했습니다

QLDB는 기본적으로 동시 저널 스트림을 5개로 제한합니다.

Amazon QLDB의 원장 관리

이 장에서는 QLDB API, AWS Command Line Interface(AWS CLI), AWS CloudFormation를 사용하여 Amazon QLDB에서 원장 관리 작업을 수행하는 방법을 설명합니다.

AWS Management Console을 사용하여 이와 동일한 작업을 수행할 수도 있습니다. 자세한 내용은 콘솔을 사용하여 Amazon QLDB에 액세스을 참조하십시오.

주제

Amazon QLDB 원장의 기본 작업

QLDB API 또는 AWS Command Line Interface(AWS CLI)를 사용하여 Amazon QLDB에서 원장을 생성, 업데이트 및 삭제할 수 있습니다. 계정에 속한 모든 원장을 나열하거나 특정 원장에 대한 정보를 가져올 수도 있습니다.

다음 항목에서는 AWS SDK for Java 및 AWS CLI를 사용하는 원장 작업의 일반적인 단계를 보여주는 간단한 코드 예를 제공합니다.

주제

전체 샘플 애플리케이션에서 이러한 작업을 보여주는 코드 예는 다음 드라이버 시작하기 자습서 및 GitHub 리포지토리를 참조하세요.

- Java: 자습서 | GitHub 리포지토리

- Node.js: 자습서 | GitHub 리포지토리
- Python: 자습서 | GitHub 리포지토리

원장 생성

AWS 계정 작업을 사용하여 `CreateLedger`에서 원장을 생성합니다. 다음 정보를 제공해야 합니다.

- 원장 명칭 - 계정에 생성하려는 원장의 명칭입니다. 이 명칭은 현재 AWS 리전에 있는 귀하의 모든 원장에 고유해야 합니다.

  원장 명칭에 대한 명명 제약 조건은 Amazon QLDB 할당량 및 제한에서 정의됩니다.
- 권한 모드 - 원장에 할당할 권한 모드입니다. 다음 옵션 중 하나를 선택하십시오:
  - 모두 허용 – 원장에 대한 API 수준 세분화로 액세스 제어를 가능하게 하는 레거시 권한 모드입니다

    이 모드를 사용하면 이 원장에 대한 `SendCommand` API 권한이 있는 사용자가 지정된 원장의 모든 표에서 모든 PartiQL 명령(따라서 `ALLOW_ALL`)을 실행할 수 있습니다. 이 모드는 원장에 대해 생성하는 모든 표 수준 또는 명령 수준 IAM 권한 정책을 무시합니다.
  - 표준 – (권장) 원장, 테이블 및 PartiQL 명령에 대해 보다 세분화된 액세스 제어를 가능하게 하는 권한 모드입니다.. 원장 데이터의 보안을 극대화하기 위해 이 권한 모드를 사용하는 것을 강력 권장합니다.

    기본적으로 이 모드는 이 원장의 모든 테이블에서 PartiQL 명령을 실행하려는 모든 요청을 거부합니다. PartiQL 명령을 허용하려면 원장에 대한 `SendCommand` API 권한 외에도 특정 테이블 리소스 및 PartiQL 작업에 대한 IAM 권한 정책을 생성해야 합니다. 자세한 내용은 Amazon QLDB에서 표준 권한 모드로 시작하기을 참조하세요.
- 삭제 보호 – (옵션) 사용자가 원장을 삭제하는 것을 방지하는 플래그입니다. 원장 생성 중 이를 정의하지 않는 경우에도 이 기능은 기본적으로 활성화됩니다(`true`).

  삭제 방지가 활성화된 경우 원장을 삭제하려면 먼저 이 기능을 비활성화해야 합니다. `UpdateLedger` 작업을 사용하여 플래그를 `false`로 설정함으로써 비활성화할 수 있습니다.
- AWS KMS key - (옵션) 저장 데이터를 암호화하는 데 사용할 AWS Key Management Service(AWS KMS)의 키입니다. 다음 AWS KMS keys 타입 중 하나를 선택하십시오:
  - AWS 소유 KMS 키 – AWS가 사용자를 대신하여 소유하고 관리하는 KMS 키를 사용합니다.

    원장 생성 중에 이 파라미터를 정의하지 않으면 원장이 기본적으로 이 타입의 키를 사용합니다. 문자열 `AWS_OWNED_KMS_KEY`를 사용하여 이 키 타입을 지정할 수 있습니다. 이 옵션은 추가 설정이 필요하지 않습니다.

- 고객 관리형 KMS 키 – 귀하가 생성, 소유 및 관리하는 계정에서 대칭 암호화 KMS 키를 사용하십시오. QLDB는 비대칭 키는 지원하지 않습니다.

  이 옵션을 사용하려면 KMS 키를 만들거나 계정의 기존 키를 사용해야 합니다. 고객 관리형 키 생성에 대한 지침은 AWS Key Management Service 개발자 가이드에서 대칭 암호화 KMS 키 생성을 참조하세요.

  ID, 별칭 또는 Amazon 리소스 이름(ARN)을 사용하여 고객 관리형 KMS 키를 지정할 수 있습니다. 자세한 설명은 AWS Key Management Service 개발자 가이드에서 키 식별자(KeyId)를 참조하세요.

  > **Note**
  >
  > 교차 리전 키는 지원되지 않습니다. 지정된 KMS 키는 원장과 동일한 AWS 리전에 있어야 합니다.

  자세한 내용은 Amazon QLDB에서의 저장 시 암호화을 참조하세요.
- 태그 – (선택 사항) 태그를 키-값 쌍으로 연결하여 메타데이터를 원장에 추가합니다. 원장에 태그를 추가함으로써 쉽게 원장을 조직화하고 식별할 수 있습니다. 자세한 내용은 Amazon QLDB 리소스 태그 지정을 참조하십시오.

QLDB가 원장을 생성하고 그 상태를 `ACTIVE`로 설정할 때까지 원장은 사용 준비가 되지 않습니다.

원장 생성 (Java)

AWS SDK for Java를 사용해 원장을 생성하려면

1. `AmazonQLDB` 클래스의 인스턴스를 만듭니다.
2. `CreateLedgerRequest` 클래스 인스턴스를 만들어 요청 정보를 입력합니다.

   원장 명칭과 권한 모드를 제공해야 합니다.
3. 요청 객체를 파라미터로 입력하여 `createLedger` 메서드를 실행합니다.

`createLedger` 요청은 원장에 대한 정보가 있는 `CreateLedgerResult` 객체를 반환합니다. 원장 생성 후 `DescribeLedger` 작업을 사용하여 원장 상태를 확인하는 예는 다음 섹션을 참조하세요.

다음 예는 이전 단계를 설명합니다.

Example - 기본 구성 설정 사용

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
CreateLedgerRequest request = new CreateLedgerRequest()
        .withName(ledgerName)
        .withPermissionsMode(PermissionsMode.STANDARD);
CreateLedgerResult result = client.createLedger(request);
```

> **Note**
>
> 지정하지 않을 경우 원장은 다음과 같은 기본 설정을 사용합니다.
>
> - 삭제 보호 – 활성화 (`true`).
> - KMS 키 - AWS 소유 KMS 키.

Example - 삭제 보호를 비활성화하고, 고객 관리형 KMS 키를 사용하고, 태그를 첨부합니다.

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();

Map<String, String> tags = new HashMap<>();
tags.put("IsTest", "true");
tags.put("Domain", "Test");

CreateLedgerRequest request = new CreateLedgerRequest()
        .withName(ledgerName)
        .withPermissionsMode(PermissionsMode.STANDARD)
        .withDeletionProtection(false)
        .withKmsKey("arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab")
        .withTags(tags);
CreateLedgerResult result = client.createLedger(request);
```

원장 생성 (AWS CLI)

기본 구성 설정을 사용하여 `vehicle-registration`라는 명칭의 새 원장을 생성합니다.

Example

```
aws qldb create-ledger --name vehicle-registration --permissions-mode STANDARD
```

> **Note**
>
> 지정하지 않을 경우 원장은 다음과 같은 기본 설정을 사용합니다.
>
> - 삭제 보호 – 활성화 (`true`).
> - KMS 키 - AWS 소유 KMS 키.

또는 삭제 방지 기능이 비활성화되고 지정된 고객 관리형 KMS 키와 지정된 태그를 사용하여 `vehicle-registration`라는 명칭의 새 원장을 생성할 수 있습니다.

Example

```
aws qldb create-ledger \
    --name vehicle-registration \
    --no-deletion-protection \
    --permissions-mode STANDARD \
    --kms-key arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab \
    --tags IsTest=true,Domain=Test
```

원장 생성 (AWS CloudFormation)

AWS CloudFormation 템플릿을 사용하여 원장을 생성할 수도 있습니다. 자세한 설명은 AWS CloudFormation 사용자 가이드의 AWS::QLDB::Ledger 리소스를 참조하세요.

원장 설명

원장에 대한 세부 정보를 보려면 `DescribeLedger` 작업을 사용하십시오. 원장 명칭을 입력해야 합니다. `DescribeLedger` 출력은 `CreateLedger` 출력의 형식과 동일합니다. 여기에는 다음 정보가 포함됩니다.

- 원장 명칭 - 설명하려는 원장의 명칭입니다.
- ARN – 다음 형식의 원장을 위한 Amazon 리소스 이름(ARN).

  ```
  arn:aws:qldb:aws-region:account-id:ledger/ledger-name
  ```

- 삭제 보호 - 삭제 보호 기능이 활성화되었는지 여부를 나타내는 플래그입니다.
- 생성 날짜 및 시간 - Epoch 시간 형식의 원장이 생성된 날짜 및 시간입니다.

- 상태 - 원장의 현재 상태입니다. 다음 값 중 하나일 수 있습니다.
  - `CREATING`
  - `ACTIVE`
  - `DELETING`
  - `DELETED`
- 권한 모드 - 원장에 할당된 권한 모드입니다. 다음 값 중 하나일 수 있습니다.
  - `ALLOW_ALL` – 원장에 대한 API 수준 세분화로 액세스 제어를 가능하게 하는 레거시 권한 모드입니다
  - `STANDARD` – 원장, 테이블 및 PartiQL 명령에 대해 보다 세분화된 액세스 제어를 가능하게 하는 권한 모드입니다..
- 암호화 설명 - 원장의 저장 데이터 암호화에 대한 정보입니다. 다음 항목이 해당됩니다:
  - AWS KMS key ARN - 원장이 저장된 암호화에 사용하는 고객 관리형 KMS 키의 ARN입니다. 정의하지 않으면 원장은 암호화에 AWS 소유 KMS 키를 사용합니다.
  - 암호화 상태 - 원장에 대한 저장된 암호화의 현재 상태입니다. 다음 값 중 하나일 수 있습니다:
    - `ENABLED` - 지정된 키를 사용하여 암호화가 완전히 활성화되었습니다.
    - `UPDATING` - 지정된 키 변경이 활발히 처리되고 있습니다.

      QLDB의 주요 변경 사항은 비동기적으로 이루어집니다. 키 변경이 처리되는 동안 성능에 영향을 주지 않고 원장에 완전히 액세스할 수 있습니다. 키를 업데이트하는 데 걸리는 시간은 원장 크기에 따라 다릅니다
    - `KMS_KEY_INACCESSIBLE` - 지정된 고객 관리형 KMS 키에 액세스할 수 없으며 원장이 손상되었습니다. 키가 비활성화 또는 삭제되었거나 키에 대한 권한 부여가 취소되었습니다. 원장이 손상되면 해당 원장에 접근할 수 없으며 읽기 또는 쓰기 요청을 수락할 수 없습니다.

      손상된 원장은 키에 대한 권한 부여를 복원한 후 또는 비활성화된 키를 다시 활성화하면 자동으로 활성 상태로 돌아갑니다. 하지만 고객 관리형 KMS 키 삭제 작업은 되돌릴 수 없습니다. 키가 삭제된 후에는 해당 키로 보호되는 원장에 더 이상 액세스할 수 없으며 데이터를 영구적으로 복구할 수 없게 됩니다.
  - 액세스할 수 없는 AWS KMS key - 오류가 발생한 경우 KMS 키에 처음 액세스할 수 없게 된 Epoch 시간 형식의 날짜 및 시간입니다.

    KMS 키에 액세스할 수 있는 경우에는 이 값이 정의되지 않습니다.

자세한 내용은 Amazon QLDB에서의 저장 시 암호화을 참조하십시오.

> **Note**
>
> QLDB 원장을 생성한 후 상태가 CREATING에서 ACTIVE로 변경되면 사용할 준비가 된 것입니다.

원장 설명 (Java)

AWS SDK for Java를 사용하여 원장을 설명하려면

1. AmazonQLDB 클래스의 인스턴스를 만듭니다. 또는 CreateLedger 요청을 위해 인스턴스화한 것과 동일한 AmazonQLDB 클라이언트 인스턴스를 사용할 수 있습니다.
2. DescribeLedgerRequest 클래스의 인스턴스를 생성하고, 삭제하려는 원장 명칭을 입력하십시오.
3. 요청 객체를 파라미터로 입력하여 describeLedger 메서드를 실행합니다.
4. describeLedger 요청은 원장에 대한 최신 정보가 있는 DescribeLedgerResult 객체를 반환합니다.

다음 코드 예에서는 이전 단계를 설명합니다. 클라이언트의 describeLedger 메서드를 호출하여 언제든지 원장 정보를 가져올 수 있습니다.

Example

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
DescribeLedgerRequest request = new DescribeLedgerRequest().withName(ledgerName);
DescribeLedgerResult result = client.describeLedger(request);
System.out.printf("%s: ARN: %s \t State: %s \t CreationDateTime: %s \t
 DeletionProtection: %s
                   \t PermissionsMode: %s \t EncryptionDescription: %s",
        result.getName(),
        result.getArn(),
        result.getState(),
        result.getCreationDateTime(),
        result.getDeletionProtection(),
        result.getPermissionsMode(),
        result.getEncryptionDescription());
```

원장 설명 (AWS CLI)

방금 생성한 `vehicle-registration` 원장에 대해 설명합니다.

Example

```
aws qldb describe-ledger --name vehicle-registration
```

원장 업데이트

현재 `UpdateLedger` 작업을 통해 기존 원장에 대해 다음과 같은 구성 설정을 변경할 수 있습니다.

- 삭제 보호 – 사용자가 원장을 삭제하는 것을 방지하는 플래그입니다. 이 기능이 활성화된 경우 원장을 삭제하려면 먼저 플래그를 `false`로 설정하여 이 기능을 비활성화해야 합니다.

  이 파라미터를 정의하지 않으면 원장의 삭제 보호 설정이 변경되지 않습니다.
- AWS KMS key - 저장 데이터를 암호화하는 데 사용할 AWS Key Management Service(AWS KMS)의 키입니다. 이 파라미터를 정의하지 않으면 원장의 KMS 키가 변경되지 않습니다.

  > **Note**
  >
  > Amazon QLDB는 2021년 7월 22일에 고객 관리형 AWS KMS keys에 대한 지원을 시작했습니다. 출시 이전에 생성된 모든 원장은 기본적으로 AWS 소유 키로 보호되지만, 현재는 고객 관리형 키를 사용하여 저장 시 암호화를 할 수 없습니다.
  > QLDB 콘솔에서 원장 생성 시간을 볼 수 있습니다.

  다음 옵션 중 하나를 사용하십시오.

  - AWS 소유 KMS 키 – AWS가 사용자를 대신하여 소유하고 관리하는 KMS 키를 사용합니다. 이 타입의 키를 사용하려면 이 파라미터에 대해 `AWS_OWNED_KMS_KEY` 문자열을 지정하세요. 이 옵션은 추가 설정이 필요하지 않습니다.
  - 고객 관리형 KMS 키 – 귀하가 생성, 소유 및 관리하는 계정에서 대칭 암호화 KMS 키를 사용하십시오. QLDB는 비대칭 키는 지원하지 않습니다.

    이 옵션을 사용하려면 KMS 키를 만들거나 계정의 기존 키를 사용해야 합니다. 고객 관리형 키 생성에 대한 지침은 AWS Key Management Service 개발자 가이드에서 대칭 암호화 KMS 키 생성을 참조하세요.

ID, 별칭 또는 Amazon 리소스 이름(ARN)을 사용하여 고객 관리형 KMS 키를 지정할 수 있습니다. 자세한 설명은 AWS Key Management Service 개발자 가이드에서 키 식별자(KeyId)를 참조하세요.

> **Note**
>
> 교차 리전 키는 지원되지 않습니다. 지정된 KMS 키는 원장과 동일한 AWS 리전에 있어야 합니다.

QLDB의 주요 변경 사항은 비동기적으로 이루어집니다. 키 변경이 처리되는 동안 성능에 영향을 주지 않고 원장에 완전히 액세스할 수 있습니다.

필요한 만큼 자주 키를 전환할 수 있지만 키를 업데이트하는 데 걸리는 시간은 원장 크기에 따라 다릅니다. `DescribeLedger` 작업을 사용하여 저장 상태의 암호화를 확인할 수 있습니다.

자세한 내용은 Amazon QLDB에서의 저장 시 암호화을 참조하십시오.

`UpdateLedger` 출력은 `CreateLedger` 출력의 형식과 동일합니다.

원장 업데이트 (Java)

AWS SDK for Java를 사용하여 원장을 업데이트하려면

1. `AmazonQLDB` 클래스의 인스턴스를 만듭니다.
2. `UpdateLedgerRequest` 클래스 인스턴스를 만들어 요청 정보를 입력합니다.

   삭제 방지를 위한 새 부울 값 또는 KMS 키의 새 문자열 값과 함께 원장 명칭을 제공해야 합니다.
3. 요청 객체를 파라미터로 입력하여 `updateLedger` 메서드를 실행합니다.

다음 코드 예는 이전 단계를 설명합니다. `UpdateLedgerResult` 요청은 원장에 대한 정보가 업데이트된 `updateLedger` 객체를 반환합니다.

Example – 삭제 보호 비활성화

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
UpdateLedgerRequest request = new UpdateLedgerRequest()
        .withName(ledgerName)
        .withDeletionProtection(false);
```

```
UpdateLedgerResult result = client.updateLedger(request);
```

Example – 고객 관리형 KMS 키 사용

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
UpdateLedgerRequest request = new UpdateLedgerRequest()
        .withName(ledgerName)
        .withKmsKey("arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab")
UpdateLedgerResult result = client.updateLedger(request);
```

Example - AWS 소유 KMS 키 사용

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
UpdateLedgerRequest request = new UpdateLedgerRequest()
        .withName(ledgerName)
        .withKmsKey("AWS_OWNED_KMS_KEY")
UpdateLedgerResult result = client.updateLedger(request);
```

원장 업데이트 (AWS CLI)

`vehicle-registration` 원장에 삭제 보호가 활성화된 경우, 원장을 삭제하려면 먼저 이 기능을 비활성화해야 합니다.

Example

```
aws qldb update-ledger --name vehicle-registration --no-deletion-protection
```

고객 관리형 KMS 키를 사용하도록 원장의 저장된 암호화 설정을 변경할 수도 있습니다.

Example

```
aws qldb update-ledger --name vehicle-registration --kms-key arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

또는 저장된 암호화 설정을 변경하여 AWS 소유 KMS 키를 사용할 수 있습니다.

Example

```
aws qldb update-ledger --name vehicle-registration --kms-key AWS_OWNED_KMS_KEY
```

원장 권한 모드 업데이트

`UpdateLedgerPermissionsMode` 작업을 통해 기존 원장의 권한 모드를 변경할 수 있습니다. 다음 옵션 중 하나를 선택하십시오:

- 모두 허용 – 원장에 대한 API 수준 세분화로 액세스 제어를 가능하게 하는 레거시 권한 모드입니다

  이 모드를 사용하면 이 원장에 대한 `SendCommand` API 권한이 있는 사용자가 지정된 원장의 모든 표에서 모든 PartiQL 명령(따라서 `ALLOW_ALL`)을 실행할 수 있습니다. 이 모드는 원장에 대해 생성하는 모든 표 수준 또는 명령 수준 IAM 권한 정책을 무시합니다.
- 표준 – (권장) 원장, 테이블 및 PartiQL 명령에 대해 보다 세분화된 액세스 제어를 가능하게 하는 권한 모드입니다.. 원장 데이터의 보안을 극대화하기 위해 이 권한 모드를 사용하는 것을 강력 권장합니다.

  기본적으로 이 모드는 이 원장의 모든 테이블에서 PartiQL 명령을 실행하려는 모든 요청을 거부합니다. PartiQL 명령을 허용하려면 원장에 대한 `SendCommand` API 권한 외에도 특정 테이블 리소스 및 PartiQL 작업에 대한 IAM 권한 정책을 생성해야 합니다. 자세한 내용은 Amazon QLDB에서 표준 권한 모드로 시작하기을 참조하세요.

> **Important**
>
> `STANDARD` 권한 모드로 전환하기 전에 사용자에게 방해가 되지 않도록 먼저 요구되는 모든 IAM 정책과 표 태그를 생성해야 합니다. 자세히 알아보려면 표준 권한 모드로 마이그레이션 섹션을 참조하세요.

원장 권한 모드 업데이트 (Java)

AWS SDK for Java를 사용하여 원장 권한 모드를 업데이트하려면

1. `AmazonQLDB` 클래스의 인스턴스를 만듭니다.
2. `UpdateLedgerPermissionsModeRequest` 클래스 인스턴스를 만들어 요청 정보를 입력합니다.

   권한 모드의 새 문자열 값과 함께 원장 명칭을 제공해야 합니다.
3. 요청 객체를 파라미터로 입력하여 `updateLedgerPermissionsMode` 메서드를 실행합니다.

다음 코드 예는 이전 단계를 설명합니다. `UpdateLedgerPermissionsModeResult` 요청은 원장에 대한 정보가 업데이트된 `updateLedgerPermissionsMode` 객체를 반환합니다.

Example - 표준 권한 모드 할당

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
UpdateLedgerPermissionsModeRequest request = new UpdateLedgerPermissionsModeRequest()
        .withName(ledgerName)
        .withPermissionsMode(PermissionsMode.STANDARD);
UpdateLedgerPermissionsModeResult result = client.updateLedgerPermissionsMode(request);
```

원장 권한 모드 업데이트 (AWS CLI)

`vehicle-registration` 원장에 `STANDARD` 권한 모드를 할당합니다.

Example

```
aws qldb update-ledger-permissions-mode --name vehicle-registration --permissions-mode
 STANDARD
```

표준 권한 모드로 마이그레이션

`STANDARD` 권한 모드로 마이그레이션하려면 QLDB 액세스 패턴을 분석하고 사용자에게 리소스에 액세스할 수 있는 적절한 권한을 부여하는 IAM 정책을 추가하는 것이 좋습니다.

`STANDARD` 권한 모드로 전환하기 전에 먼저 필요한 모든 IAM 정책과 표 태그를 생성해야 합니다. 그렇지 않으면 올바른 IAM 정책을 생성하거나 권한 모드를 다시 `ALLOW_ALL`로 되돌릴 때까지 권한 모드를 전환하면 사용자 작업이 중단될 수 있습니다. 이러한 정책 생성에 대한 자세한 설명은 Amazon QLDB에서 표준 권한 모드로 시작하기 섹션을 참조하세요.

또한 AWS 관리형 정책을 사용하여 모든 QLDB 리소스에 대한 전체 액세스 권한을 부여할 수 있습니다. `AmazonQLDBFullAccess` 및 `AmazonQLDBConsoleFullAccess` 관리형 정책에는 모든 PartiQL 작업을 포함한 모든 QLDB 작업이 포함됩니다. 이러한 정책 중 하나를 주체에 연결하는 것은 해당 주체에 대한 `ALLOW_ALL` 권한 모드와 동일합니다. 자세한 내용은 AWS 아마존 QLDB에 대한 관리형 정책을 참조하십시오.

원장 삭제

이 `DeleteLedger` 작업을 사용하여 원장과 모든 콘텐츠를 삭제합니다. 원장 삭제 작업은 취소할 수 없습니다.

원장에 대해 삭제 보호가 활성화된 경우, 먼저 이를 비활성화해야 원장을 삭제할 수 있습니다.

`DeleteLedger` 요청을 발행하면 원장의 상태가 `ACTIVE`에서 `DELETING`으로 바뀝니다. 사용하는 스토리지 용량에 따라 원장을 삭제하는 데 시간이 걸릴 수 있습니다. `DeleteLedger` 작업이 완료되면 QLDB에 해당 원장이 더 이상 존재하지 않습니다.

원장 삭제 (Java)

AWS SDK for Java를 사용하여 원장을 삭제하려면

1. `AmazonQLDB` 클래스의 인스턴스를 만듭니다.
2. `DeleteLedgerRequest` 클래스의 인스턴스를 생성하고 삭제하려는 원장의 명칭을 입력합니다.
3. 요청 객체를 파라미터로 입력하여 `deleteLedger` 메서드를 실행합니다.

다음 코드 예에서는 이전 단계를 설명합니다.

Example

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
DeleteLedgerRequest request = new DeleteLedgerRequest().withName(ledgerName);
DeleteLedgerResult result = client.deleteLedger(request);
```

원장 삭제 (AWS CLI)

`vehicle-registration` 원장을 삭제합니다.

Example

```
aws qldb delete-ledger --name vehicle-registration
```

원장 등재

이 `ListLedgers` 작업은 현재 AWS 계정 및 지역에 대한 모든 QLDB 원장의 요약 정보를 반환합니다.

원장 등재 (Java)

AWS SDK for Java을 사용하여 계정의 원장을 나열하려면

1. `AmazonQLDB` 클래스의 인스턴스를 만듭니다.
2. `ListLedgersRequest` 클래스의 인스턴스를 만듭니다.

이전 `ListLedgers` 호출의 응답에서 `NextToken` 값을 받은 경우 이 요청에 해당 값을 제공해야 다음 결과 페이지를 확인할 수 있습니다.

3. 요청 객체를 파라미터로 입력하여 `listLedgers` 메서드를 실행합니다.
4. `listLedgers` 요청은 `ListLedgersResult` 객체를 반환합니다. 이 객체에는 `LedgerSummary` 객체 목록과 더 많은 결과가 있는지 여부를 나타내는 페이지 매김 토큰이 있습니다.

   - `NextToken`이 비어 있는 경우 결과의 마지막 페이지가 처리된 것이며, 더 이상 결과가 없습니다.
   - `NextToken`이 비어 있지 않으면 더 많은 결과를 사용할 수 있습니다. 결과의 그 다음 페이지를 검색하려면 후속 `ListLedgers` 호출에서 `NextToken`의 값을 사용합니다.

다음 코드 예에서는 이전 단계를 설명합니다.

Example

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
List<LedgerSummary> ledgerSummaries = new ArrayList<>();
String nextToken = null;
do {
    ListLedgersRequest request = new ListLedgersRequest().withNextToken(nextToken);
    ListLedgersResult result = client.listLedgers(request);
    ledgerSummaries.addAll(result.getLedgers());
    nextToken = result.getNextToken();
} while (nextToken != null);
```

원장 등재 (AWS CLI)

현재 AWS 계정 및 지역의 모든 원장을 열거합니다.

Example

```
aws qldb list-ledgers
```

AWS CloudFormation를 사용하여 Amazon QLDB 리소스 생성

Amazon QLDB는 AWS 리소스를 모델링하고 설정하여 리소스 및 인프라를 생성하고 관리하는 데 더 적은 시간을 할애할 수 있도록 지원하는 서비스인 AWS CloudFormation와 통합되어 있습니다. 원하는

모든 AWS 리소스(예: QLDB 원장)를 설명하는 템플릿을 생성하면 AWS CloudFormation는 해당 리소스를 프로비저닝하고 구성합니다.

AWS CloudFormation을 사용하면 템플릿을 재사용하여 QLDB 리소스를 일관되고 반복적으로 설정할 수 있습니다. 리소스를 한 번 설명한 다음 여러 AWS 계정 및 리전에서 동일한 리소스를 반복해서 프로비저닝합니다.

QLDB 및 AWS CloudFormation 템플릿

QLDB 및 관련 서비스에 대한 리소스를 프로비저닝하고 구성하려면 AWS CloudFormation 템플릿을 이해해야 합니다. 템플릿은 JSON 또는 YAML로 서식 지정된 텍스트 파일입니다. 이 템플릿은 AWS CloudFormation 스택에서 프로비저닝할 리소스에 대해 설명합니다. JSON 또는 YAML에 익숙하지 않은 경우 AWS CloudFormation Designer를 사용하면 AWS CloudFormation 템플릿을 시작하는 데 도움이 됩니다. 자세한 내용은 AWS CloudFormation 사용 설명서에서 AWS CloudFormation Designer란 무엇입니까?를 참조하세요.

QLDB는 AWS CloudFormation에서 원장 및 저널 스트림 생성을 지원합니다. 원장 및 스트림에 대한 JSON 및 YAML 템플릿의 예를 포함한 자세한 내용은 AWS CloudFormation 사용 설명서의 다음 리소스 유형 참조를 참조하십시오.

- AWS::QLDB::원장 참조
- AWS::QLDB::스트림 참조

AWS CloudFormation에 대해 자세히 알아보기

AWS CloudFormation에 대한 자세한 내용은 다음 리소스를 참조하십시오.

- AWS CloudFormation
- AWS CloudFormation 사용 설명서
- AWS CloudFormation API 참조
- AWS CloudFormation 명령줄 인터페이스 사용 설명서

Amazon QLDB 리소스 태그 지정

태그는 사용자 또는 AWS가 AWS 리소스에 할당하는 사용자 지정 속성 레이블입니다. 각 태그에는 다음 두 가지 부분이 있습니다.

- 태그 키 (예: `CostCenter`, `Environment` 또는 `Project`) 태그 키는 대/소문자를 구별합니다.
- 태그 값(예: `111122223333` 또는 `Production`)으로 알려진 선택적 필드 태그 값을 생략하는 것은 빈 문자열을 사용하는 것과 같습니다. 태그 키처럼 태그 값은 대/소문자를 구별합니다.

태그는 다음을 지원합니다.

- AWS 리소스를 식별하고 정리합니다. 많은 AWS 서비스가 태그 지정을 지원하므로 다른 서비스의 리소스에 동일한 태그를 할당하여 해당 리소스의 관련 여부를 나타낼 수 있습니다. 예컨대, Amazon S3 버킷에 할당한 것과 동일한 태그를 Amazon QLDB 원장에 할당할 수 있습니다.
- AWS 비용을 추적합니다. AWS Billing and Cost Management 대시보드에서 이러한 태그를 활성화합니다. AWS는 태그를 사용하여 비용을 분류하고 월별 비용 할당 보고서를 전달합니다. 자세한 설명은 AWS Billing 사용자 가이드에서 비용 할당 태그 사용을 참조하세요.
- AWS Identity and Access Management (IAM)을 사용하여 AWS 리소스에 대한 액세스를 통제합니다. 자세한 설명은 이 개발자 가이드의 QLDB와 함께하는 ABAC(속성 기반 액세스 제어) 섹션과 IAM 사용자 가이드의 IAM 태그를 사용한 액세스 통제를 참조하세요.

태그 사용에 대한 팁은 AWS Answers 블로그의 게시글 AWS Tagging Strategies를 참조하세요.

다음 섹션에서는 Amazon QLDB용 태그에 대한 추가 정보를 제공합니다.

주제

- Amazon QLDB에서 지원되는 리소스
- 태그 명칭 지정 및 사용 규칙
- 태그 관리
- 생성 시 리소스 태그 지정

Amazon QLDB에서 지원되는 리소스

Amazon QLDB의 다음 리소스는 태깅을 지원합니다.

- 원장
- 테이블
- 저널 스트림

태그 추가 및 관리에 대한 자세한 설명은 태그 관리 섹션을 참조하세요.

태그 명칭 지정 및 사용 규칙

다음 기본 명명 및 사용 규칙은 Amazon QLDB 리소스와 함께 태그를 사용할 때 적용됩니다:

- 각 리소스는 최대 50개의 태그를 보유할 수 있습니다.
- 각 리소스에 대해 각 태그 키는 고유하며 하나의 값만 가질 수 있습니다.
- 태그 키의 최대 길이는 UTF-8 형식의 유니코드 문자 128자입니다.
- 태그 값의 최대 길이는 UTF-8 형식의 유니코드 문자 256자입니다.
- 허용되는 문자는 UTF-8로 표현할 수 있는 문자, 숫자, 공백 및 . : + = @ _ / -(하이픈) 문자도 있습니다.
- 태그 키와 값은 대/소문자를 구분합니다. 태그를 대문자로 사용하는 전략을 세우고 이러한 전략을 모든 리소스 타입에 대해 일관되게 구현하는 것이 가장 좋습니다. 예컨대, `Costcenter`, `costcenter` 또는 `CostCenter`를 사용할지 결정하고 모든 태그에 대해 동일한 규칙을 사용합니다. 대/소문자가 일치하지 않는 유사한 태그를 사용하지 마십시오.
- `aws:` 접두사는 AWS용으로 예약되어 있습니다. 태그에 `aws:` 접두사가 있는 태그 키가 있는 경우 태그의 키 또는 값을 편집하거나 삭제할 수 없습니다. 이 접두사가 지정된 태그는 리소스당 태그 수 제한에 포함되지 않습니다.

태그 관리

태그는 리소스의 `Key` 및 `Value` 속성으로 구성됩니다. Amazon QLDB 콘솔, AWS CLI 또는 QLDB API를 사용하여 이러한 속성의 값을 추가, 편집 또는 삭제할 수 있습니다. AWS Resource Groups 태그 편집기를 사용하여 태그를 관리할 수도 있습니다.

태그 작업에 대한 자세한 설명은 다음 API 작업을 참조하세요:

- Amazon QLDB API 참조의 ListTagsForResource
- Amazon QLDB API 참조의 TagResource
- Amazon QLDB API 참조의 UntagResource

QLDB 태그 지정 패널을 사용하려면(콘솔)

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/qldb에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 원장을 선택합니다.

3. 원장 목록에서 태그를 관리하려는 원장 명칭을 선택합니다.
4. 원장 세부 정보 페이지에서 태그 카드를 찾아 태그 관리를 선택합니다.
5. 태그 관리 페이지에서 원장에 적합한 태그를 추가, 편집 또는 제거할 수 있습니다. 태그 키와 값을 원하는 대로 설정했으면 저장을 선택합니다.

생성 시 리소스 태그 지정

태그 지정을 지원하는 QLDB 리소스의 경우, AWS Management Console, AWS CLI 또는 QLDB API를 사용하여 리소스를 생성하는 동안 태그를 정의할 수 있습니다. 생성하는 동안 리소스에 태그를 지정하면 리소스 생성 후 사용자 지정 태그 지정 스크립트를 실행할 필요가 없습니다.

리소스에 태그가 지정되면 해당 태그를 기반으로 리소스에 대한 액세스를 제어할 수 있습니다. 예컨대 특정 태그가 있는 표 리소스에만 전체 액세스 권한을 부여할 수 있습니다. JSON 정책의 예는 테이블 태그를 기반으로 하는 모든 작업에 대한 전체 액세스 권한 섹션을 참조하세요.

> **Note**
>
> 표 및 스트림 리소스는 루트 원장 리소스의 태그를 상속하지 않습니다.

또한 `CREATE TABLE` PartiQL 문에 표 태그를 지정하여 표 태그를 정의할 수 있습니다. 자세한 내용은 태그 지정 테이블 섹션을 참조하세요.

Amazon QLDB의 보안

클라우드 AWS 보안이 최우선 과제입니다. AWS 고객은 가장 보안에 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다.

보안은 기업과 기업 간의 AWS 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드 보안 — AWS AWS 서비스 클라우드에서 실행되는 인프라를 보호하는 역할을 합니다 AWS 클라우드. AWS 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사원은 정기적으로 AWS 규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 확인합니다. Amazon ECR에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 AWS 규정 준수 프로그램별 범위 내 서비스를 참조하세요.
- 클라우드에서의 보안 — AWS 서비스 사용하는 항목에 따라 책임이 결정됩니다. 또한 귀하는 데이터의 민감도, 회사 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 QLDB 사용 시 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 QLDB를 구성하는 방법을 보여줍니다. 또한 QLDB 리소스를 모니터링하고 보호하는 데 도움이 AWS 서비스 되는 다른 방법을 사용하는 방법도 알아봅니다.

주제

- Amazon QLDB의 데이터 보호
- Amazon QLDB용 ID 및 액세스 관리
- Amazon QLDB의 로깅 및 모니터링
- Amazon QLDB에 대한 규정 준수 확인
- Amazon QLDB의 복원성
- Amazon MQ의 인프라 보안

Amazon QLDB의 데이터 보호

AWS 공동 책임 모델 Amazon QLDB의 데이터 보호에 적용됩니다. 이 모델에 설명된 대로 AWS 는 모든 모델을 실행하는 글로벌 인프라를 보호할 책임이 있습니다. AWS 클라우드사용자는 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스 의 보안 구성과 관리 작업에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를 참조하세

요. 유럽의 데이터 보호에 대한 자세한 내용은AWS 보안 블로그에서 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터 보호를 위해 AWS 계정 자격 증명을 보호하고 AWS IAM Identity Center OR AWS Identity and Access Management (IAM) 을 사용하여 개별 사용자를 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 리소스와 통신할 수 있습니다. AWS TLS 1.2는 필수이며 TLS 1.3를 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다. AWS CloudTrail
- 포함된 모든 기본 보안 제어와 함께 AWS 암호화 솔루션을 사용하십시오 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 명령줄 인터페이스 또는 API를 AWS 통해 액세스할 때 FIPS 140-2로 검증된 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용하십시오. 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 콘솔 AWS CLI, API 또는 SDK를 사용하여 QLDB 또는 AWS 서비스 다른 것으로 작업하는 경우가 포함됩니다. AWS 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 확인하기 위해 보안 인증 정보를 URL에 포함시켜서는 안 됩니다.

> **Note**
>
> 민감한 정보에 대한 태그 또는 자유 형식 필드 사용에 대한 이 지침은 원장에 저장된 데이터가 아닌 QLDB 원장 리소스의 메타데이터를 참조합니다. QLDB 원장 리소스에 저장된 데이터는 청구 또는 진단 로그에 사용되지 않습니다.

주제

- Amazon QLDB에서의 저장 시 암호화
- Amazon QLDB의 전송 중 암호화

Amazon QLDB에서의 저장 시 암호화

Amazon QLDB에 저장된 모든 사용자 데이터는 기본적으로 저장 시 완전히 암호화됩니다. 저장된 QLDB 암호화는 () 의 암호화 키를 사용하여 저장된 모든 원장 데이터를 암호화하여 보안을 강화합니다. AWS Key Management Service AWS KMS이 기능을 사용하면 중요한 데이터 보호와 관련된 운영 부담 및 복잡성을 줄일 수 있습니다. 저장 시 암호화를 사용하면 엄격한 암호화 규정 준수 및 규제 요구 사항이 필요한, 보안에 민감한 원장 애플리케이션을 구축할 수 있습니다.

저장 중 암호화는 QLDB 원장을 보호하는 데 사용되는 암호화 키 관리를 AWS KMS 위해 통합됩니다. 에 대한 자세한 내용은 개발자 AWS KMS안내서의 AWS Key Management Service 개념을 참조하십시오.AWS Key Management Service

QLDB에서는 각 원장 AWS KMS key 리소스의 유형을 지정할 수 있습니다. 새 원장을 생성하거나 기존 원장을 업데이트할 때 다음 유형의 KMS 키 중 하나를 선택하여 원장 데이터를 보호할 수 있습니다.

- AWS 소유 키 - 기본 암호화 유형 키는 QLDB가 소유합니다(추가 비용 없음).
- 고객 관리형 키 - 사용자의 AWS 계정 에 키가 저장되며 사용자가 생성, 소유, 관리하는 유형입니다. 키를 완전히 제어할 수 있습니다 (AWS KMS 요금 적용).

> **Note**
>
> Amazon QLDB는 2021년 7월 22일에 고객 AWS KMS keys 관리형 고객을 위한 지원을 시작했습니다. 출시 이전에 생성된 모든 원장은 AWS 소유 키 기본적으로 보호되지만, 현재는 고객 관리 키를 사용하여 저장된 데이터를 암호화할 수 없습니다.
> QLDB 콘솔에서 원장 생성 시간을 볼 수 있습니다.

원장에 액세스하면 QLDB가 데이터를 투명하게 해독합니다. 언제든지 고객 관리 AWS 소유 키 키와 고객 관리 키 사이를 전환할 수 있습니다. 암호화된 데이터를 사용 또는 관리하기 위해 코드나 애플리케이션을 변경할 필요가 없습니다.

QLDB API 또는 () 를 사용하여 새 원장을 생성하거나 기존 원장의 암호화 키를 변경할 때 암호화 키를 지정할 수 있습니다. AWS Management Console AWS Command Line Interface AWS CLI자세한 정보는 Amazon QLDB에서 고객 관리형 키 사용을 참조하세요.

> **Note**
>
> 기본적으로 Amazon QLDB는 추가 비용 없이 AWS 소유 키 를 사용하여 저장 시 암호화를 자동으로 활성화합니다. 하지만 고객 관리 AWS KMS 키 사용에는 요금이 부과됩니다. 요금에 대한 자세한 정보는 AWS Key Management Service 요금을 참조하세요.
> 저장된 QLDB 암호화는 QLDB를 사용할 수 AWS 리전 있는 모든 곳에서 사용할 수 있습니다.

주제

- 저장 데이터 암호화: Amazon QLDB 작동 방식
- Amazon QLDB에서 고객 관리형 키 사용

저장 데이터 암호화: Amazon QLDB 작동 방식

QLDB 저장 시 암호화는 256비트 고급 암호화 표준(AES-256)을 사용하여 데이터를 암호화합니다. 이는 기본 스토리지에 대한 무단 액세스로부터 데이터의 보안을 유지하는 데 도움을 줍니다. QLDB 원장에 저장된 모든 데이터는 기본적으로 저장 시 암호화됩니다. 서버 측 암호화는 투명하므로 애플리케이션을 변경할 필요가 없습니다.

저장 중 암호화는 QLDB 원장을 보호하는 데 사용되는 암호화 키를 관리하기 위해 AWS Key Management Service (AWS KMS) 와 통합됩니다. 신규 원장을 생성하거나 기존 원장을 갱신할 때 다음 유형의 AWS KMS 키 중 하나를 선택할 수 있습니다.

- AWS 소유 키 - 기본 암호화 유형 키는 QLDB가 소유합니다(추가 비용 없음).
- 고객 관리형 키 - 사용자의 AWS 계정 에 키가 저장되며 사용자가 생성, 소유, 관리하는 유형입니다. 키를 완전히 제어할 수 있습니다 (AWS KMS 요금 적용).

주제

- AWS 소유 키
- 고객 관리형 키
- AWS KMS에서 Amazon QLDB가 권한 부여를 사용하는 방법
- AWS KMS에서 권한 부여 복원하기
- 저장 시 암호화 고려 사항

AWS 소유 키

AWS 소유 키 귀하의 계정에는 저장되지 않습니다 AWS 계정. 여러 키에서 사용할 수 있도록 AWS 소유하고 관리하는 KMS 키 컬렉션의 일부입니다. AWS 계정 AWS 서비스 데이터를 보호하는 AWS 소유 키 데 사용할 수 있습니다.

AWS 소유 키는 생성하거나 관리할 필요가 없습니다. 하지만 데이터 사용을 보거나 AWS 소유 키추적하거나 감사할 수는 없습니다. 월 사용료나 사용료는 AWS 소유 키청구되지 않으며 계정 AWS KMS 할당량에도 포함되지 않습니다.

자세한 내용은AWS Key Management Service 개발자 안내서의 AWS 소유 키을 참조하세요.

고객 관리형 키

고객 관리 키는 AWS 계정 사용자가 만들고 소유하고 관리하는 KMS 키입니다. 이러한 KMS 키를 완전히 제어할 수 있습니다. QLDB는 대칭 암호화 KMS 키만 지원합니다.

고객 관리형 키를 사용하여 다음과 같은 기능을 얻을 수 있습니다.

- 키 정책, IAM 정책 및 권한 부여를 사용하여 KMS 키에 대한 액세스를 제어합니다.
- 키 활성화 및 비활성화
- 키에 대한 암호화 자료 순환
- 키 태그 및 별칭 생성
- 키 삭제 예약
- 자체 키 구성 요소 가져오기 또는 사용자가 소유하고 관리하는 사용자 지정 키 스토어 사용
- Amazon AWS CloudTrail CloudWatch Logs를 사용하여 QLDB가 사용자를 대신하여 보내는 AWS KMS 요청을 추적합니다.

자세한 내용은 AWS Key Management Service 개발자 안내서의 고객 관리형 키를 참조하십시오.

고객 관리형 키는 각 API 호출에 대해 요금이 부과되며 이러한 KMS 키에는 AWS KMS 할당량이 적용됩니다. 자세한 내용은 AWS KMS 리소스 또는 요청 할당량을 참조하세요.

고객 관리형 키를 원장의 KMS 키로 지정하면 저널 스토리지와 인덱싱된 스토리지의 모든 원장 데이터가 동일한 고객 관리형 키로 보호됩니다.

액세스할 수 없는 고객 관리형 키

고객 관리형 키를 비활성화하거나, 키 삭제를 예약하거나, 키에 대한 권한 부여를 취소하면 원장 암호화 상태가 `KMS_KEY_INACCESSIBLE`이 됩니다. 이 상태에서는 원장이 손상되어 읽기 또는 쓰기 요청

을 수락하지 않습니다. 액세스할 수 없는 키는 모든 사용자와 QLDB 서비스가 데이터를 암호화하거나 해독할 수 없게 하고 원장에서 읽기 및 쓰기 작업을 수행할 수 없게 합니다. QLDB는 사용자가 원장에 지속적으로 액세스하고 데이터 유실을 방지하기 위해 KMS 키에 액세스할 수 있어야 합니다.

> **Important**
>
> 손상된 원장은 키에 대한 권한 부여를 복원한 후 또는 비활성화된 키를 다시 활성화하면 자동으로 활성 상태로 돌아갑니다.
> 하지만 고객 관리형 키를 삭제하면 되돌릴 수 없습니다. 키가 삭제된 후에는 해당 키로 보호되는 원장에 더 이상 액세스할 수 없으며 데이터를 영구적으로 복구할 수 없게 됩니다.

원장의 암호화 상태를 확인하려면 또는 API 작업을 사용하십시오. AWS Management Console DescribeLedger

AWS KMS에서 Amazon QLDB가 권한 부여를 사용하는 방법

QLDB에서 고객 관리형 키를 사용하려면 권한 부여가 필요합니다. 고객 관리 키로 보호되는 원장을 생성하면 QLDB가 요청을 전송하여 사용자를 대신하여 권한을 생성합니다. CreateGrant AWS KMS권한 AWS KMS 부여는 QLDB에 고객의 KMS 키에 대한 액세스 권한을 부여하는 데 사용됩니다. AWS 계정 자세한 내용은 AWS Key Management Service 개발자 가이드에서 권한 부여 사용을 참조하세요.

QLDB는 다음 AWS KMS 작업에 대해 고객 관리형 키를 사용할 수 있는 권한이 필요합니다.

- DescribeKey— 지정된 대칭 암호화 KMS 키가 유효한지 확인하십시오.
- GenerateDataKey— QLDB가 원장에 저장된 데이터를 암호화하는 데 사용하는 고유한 대칭 데이터 키를 생성합니다.
- Decrypt – 고객 관리형 키로 암호화된 데이터 키를 해독합니다.
- Encrypt – 고객 관리형 키를 사용하여 일반 텍스트를 사이퍼텍스트로 암호화합니다.

언제든지 권한 부여에 대한 액세스 권한을 취소하거나 고객 관리형 키에 대한 서비스 액세스를 제거할 수 있습니다. 이렇게 하면 키에 액세스할 수 없게 되고 QLDB는 고객 관리형 키로 보호되는 원장 데이터에 액세스할 수 없게 됩니다. 이 상태에서는 원장이 손상되어 키에 대한 권한을 복원할 때까지 어떠한 읽기 또는 쓰기 요청도 수락하지 않습니다.

AWS KMS에서 권한 부여 복원하기

고객 관리형 키에 대한 권한을 복원하고 QLDB에서 원장에 대한 액세스를 복구하려면 원장을 업데이트하고 동일한 KMS 키를 지정합니다. 지침은 기존 원장의 AWS KMS key 업데이트을 참조하세요.

저장 시 암호화 고려 사항

에서 저장 시 암호화를 사용할 때 다음을 고려하세요.

- 저장 시 서버 측 암호화는 기본적으로 모든 QLDB 원장 데이터에서 활성화되며 비활성화할 수 없습니다. 원장에서 항목의 하위 집합만 암호화할 수 없습니다.
- 저장 시 암호화는 영구 스토리지 미디어에서 정적(저장 시) 상태인 데이터만 암호화합니다. 전송 중 데이터 또는 사용 중인 데이터에 대한 데이터 보안 문제가 있는 경우 다음과 같은 추가 조치를 취해야 합니다.
  - 전송 중 데이터: QLDB의 모든 데이터는 전송 중에 암호화됩니다. 기본적으로 QLDB와의 통신은 Secure Sockets Layer(SSL)/전송 계층 보안(TLS) 암호화를 사용하여 네트워크 트래픽을 보호하는 HTTPS 프로토콜을 사용합니다.
  - 사용 중인 데이터: 에 보내기 전에 클라이언트 측 암호화를 사용하여 데이터를 보호합니다.

원장에 대해 고객 관리형 키를 구현하는 방법을 알아보려면 Amazon QLDB에서 고객 관리형 키 사용 섹션으로 이동하세요.

Amazon QLDB에서 고객 관리형 키 사용

AWS Management Console, AWS Command Line Interface (AWS CLI) 또는 QLDB API를 사용하여 Amazon QLDB의 새 원장과 기존 원장을 지정할 AWS KMS key 수 있습니다. 다음 주제에서는 QLDB에서 고객 관리형 키의 사용을 관리하고 모니터링하는 방법을 설명합니다.

주제

- 필수 조건
- 새 원장에 AWS KMS key 지정하기
- 기존 원장의 AWS KMS key 업데이트
- AWS KMS keys모니터링

필수 조건

고객 관리 키로 QLDB 원장을 보호하려면 먼저 () 에서 키를 생성해야 합니다. AWS Key Management Service AWS KMS또한 QLDB가 사용자를 AWS KMS key 대신하여 권한 부여를 생성할 수 있도록 하는 키 정책을 지정해야 합니다.

고객 관리형 키 생성

고객 관리형 키를 생성하려면 AWS Key Management Service 개발자 안내서의 대칭 암호화 KMS 키 생성의 단계를 따르세요. QLDB는 비대칭 키는 지원하지 않습니다.

키 정책 설정

키 정책은 고객 관리 키에 대한 액세스를 제어하는 기본 방법입니다. AWS KMS모든 고객 관리형 키에는 정확히 하나의 키 정책이 있어야 합니다. 키 정책 문서의 설명은 KMS 키를 사용하는 권한을 가진 사람이 누구고 이를 어떻게 사용할 수 있는지 결정합니다. 자세한 내용은 에서 키 정책 사용을 참조하십시오 AWS KMS.

고객 관리형 키를 생성할 때 키 정책을 지정할 수 있습니다. 기존 고객 관리형 키의 키 정책을 변경하려면 키 정책 변경을 참조하세요.

QLDB가 고객 관리 키를 사용하도록 허용하려면 키 정책에 다음 작업에 대한 권한이 포함되어야 합니다. AWS KMS

- kms: CreateGrant — 고객 관리 키에 권한 부여를 추가합니다. 지정된 KMS 키에 제어 액세스 권한을 부여합니다.

  지정된 고객 관리형 키로 원장을 생성하거나 업데이트하면 QLDB는 필요한 권한 부여 작업에 액세스할 수 있는 권한을 생성합니다. 권한 부여 작업에는 다음이 포함됩니다.

  - GenerateDataKey
  - Decrypt
  - 암호화
- kms: DescribeKey — 고객 관리 키에 대한 세부 정보를 반환합니다. QLDB는 이 정보를 사용하여 키를 확인합니다.

예제 키 정책

다음은 QLDB에 사용할 수 있는 키 정책 예입니다. 이 정책은 계정 `111122223333`에서 QLDB를 사용할 권한이 있는 보안 주체가 리소스 `DescribeKey`에서 `CreateGrant` 및 `arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab` 작업을 호출할 수 있도록 허용합니다.

이 정책을 사용하려면 예제의 *us-east-1*, *111122223333*, *1234abcd-12ab-34cd-56ef-1234567890ab*를 사용자 고유의 정보로 바꾸세요.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid" : "Allow access to principals authorized to use Amazon QLDB",
            "Effect" : "Allow",
            "Principal" : {
                "AWS" : "*"
            },
            "Action" : [
                "kms:DescribeKey",
                "kms:CreateGrant"
            ],
            "Resource" : "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "Condition" : {
                "StringEquals" : {
                    "kms:ViaService" : "qldb.us-east-1.amazonaws.com",
                    "kms:CallerAccount" : "111122223333"
                }
            }
        }
    ]
}
```

새 원장에 AWS KMS key 지정하기

다음 단계에 따라 QLDB 콘솔 또는 AWS CLI를 사용하여 새 원장을 생성할 때 KMS 키를 지정합니다.

ID, 별칭 또는 Amazon 리소스 이름(ARN)을 사용하여 고객 관리형 키를 지정할 수 있습니다. 자세히 알아보려면 AWS Key Management Service 개발자 안내서의 키 식별자 (KeyId) 를 참조하십시오.

> **Note**
>
> 교차 리전 키는 지원되지 않습니다. 지정된 KMS 키는 원장 내에 있어야 합니다. AWS 리전

원장 생성 (콘솔)

1. 에 AWS Management Console로그인하고 https://console.aws.amazon.com/qldb 에서 Amazon QLDB 콘솔을 엽니다.
2. 원장 생성을 선택합니다.

3. 원장 생성 페이지에서 다음을 수행합니다.

   - 원장 정보 — 현재 및 지역의 모든 원장 중에서 고유한 원장 이름을 입력합니다. AWS 계정
   - 권한 모드 – 원장에 할당할 권한 모드를 선택합니다.
     - 모두 허용
     - 표준(권장)
   - 저장 데이터 암호화 – 저장된 데이터 암호화에 사용할 KMS 키 유형을 선택합니다.
     - AWS 소유한 KMS 키 사용 - 사용자를 대신하여 소유하고 관리하는 KMS 키를 사용합니다. AWS 이는 기본 옵션이며 추가 설정이 필요하지 않습니다.
     - 다른 AWS KMS 키 선택 — 생성, 소유, 관리하는 계정에서 대칭 암호화 KMS 키를 사용합니다.

       AWS KMS 콘솔을 사용하여 새 키를 생성하려면 키 생성을 선택합니다. AWS KMS 자세한 내용은 AWS Key Management Service 개발자 안내서의 대칭 암호화 KMS 키 생성을 참조하십시오.

       기존 KMS 키를 사용하려면 드롭다운 목록에서 하나를 선택하거나 KMS 키 ARN을 지정합니다.

4. 원하는 대로 설정되었으면 원장 생성을 선택합니다.

   QLDB 원장 상태가 활성 상태가 되면 해당 원장에 액세스할 수 있습니다. 몇 분 정도 걸릴 수 있습니다.

원장 생성 (AWS CLI)

AWS CLI 를 사용하여 QLDB에서 AWS 소유 키 기본 또는 고객 관리 키로 원장을 만들 수 있습니다.

Example – 기본 AWS 소유 키를 사용하여 원장 생성

```
aws qldb create-ledger --name my-example-ledger --permissions-mode STANDARD
```

Example – 고객 관리형 키를 사용하여 원장 생성

```
aws qldb create-ledger \
    --name my-example-ledger \
    --permissions-mode STANDARD \
    --kms-key arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

기존 원장의 AWS KMS key 업데이트

또한 QLDB 콘솔 또는 AWS CLI 를 사용하여 언제든지 기존 원장의 KMS 키를 또는 고객 관리 키로 업데이트할 수 AWS 소유 키 있습니다.

> **Note**
>
> Amazon QLDB는 2021년 7월 22일에 고객 AWS KMS keys 관리형 고객을 위한 지원을 시작했습니다. 출시 이전에 생성된 모든 원장은 AWS 소유 키 기본적으로 보호되지만, 현재는 고객 관리 키를 사용하여 저장된 데이터를 암호화할 수 없습니다.
> QLDB 콘솔에서 원장 생성 시간을 볼 수 있습니다.

QLDB의 주요 변경 사항은 비동기적으로 이루어집니다. 키 변경이 처리되는 동안 성능에 영향을 주지 않고 원장에 완전히 액세스할 수 있습니다. 키를 업데이트하는 데 걸리는 시간은 원장 크기에 따라 다릅니다

ID, 별칭 또는 Amazon 리소스 이름(ARN)을 사용하여 고객 관리형 키를 지정할 수 있습니다. 자세히 알아보려면 AWS Key Management Service 개발자 안내서의 키 식별자 (KeyId) 를 참조하십시오.

> **Note**
>
> 교차 리전 키는 지원되지 않습니다. 지정된 KMS 키는 원장 내에 있어야 합니다. AWS 리전

원장 업데이트 (콘솔)

1. 에 AWS Management Console로그인하고 https://console.aws.amazon.com/qldb 에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 원장을 선택합니다.
3. 원장 목록에서 갱신하려는 원장을 선택한 다음 원장 편집을 선택합니다.
4. 원장 편집 페이지에서 저장 시 암호화에 사용할 KMS 키 유형을 선택합니다.

   - AWS 소유한 KMS 키 사용 - 사용자를 대신하여 AWS 소유하고 관리하는 KMS 키를 사용합니다. 이는 기본 옵션이며 추가 설정이 필요하지 않습니다.
   - 다른 AWS KMS 키 선택 — 생성, 소유, 관리하는 계정에서 대칭 암호화 KMS 키를 사용합니다.

     AWS KMS 콘솔을 사용하여 새 키를 생성하려면 키 생성을 선택합니다. AWS KMS 자세한 내용은 AWS Key Management Service 개발자 안내서의 대칭 암호화 KMS 키 생성을 참조하십시오.

기존 KMS 키를 사용하려면 드롭다운 목록에서 하나를 선택하거나 KMS 키 ARN을 지정합니다.

5. 변경 확인을 선택합니다.

원장 업데이트 (AWS CLI)

AWS CLI 를 사용하여 QLDB의 기존 원장을 AWS 소유 키 기본 또는 고객 관리 키로 업데이트할 수 있습니다.

Example – 기본 AWS 소유 키를 사용하여 원장 업데이트

```
aws qldb update-ledger --name my-example-ledger --kms-key AWS_OWNED_KMS_KEY
```

Example – 고객 관리형 키를 사용하여 원장 업데이트

```
aws qldb update-ledger \
    --name my-example-ledger \
    --kms-key arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

AWS KMS keys모니터링

고객 관리 키를 사용하여 Amazon QLDB 원장을 보호하는 경우 Amazon Logs를 사용하여 AWS CloudTrailQLDB가 사용자를 대신하여 보내는 요청을 추적할 수 있습니다. CloudWatch AWS KMS 자세한 내용을 알아보려면 AWS Key Management Service 개발자 안내서의 AWS KMS keys모니터링를 참조하세요.

다음 예는`CreateGrant`,, `GenerateDataKey` 및 작업에 대한 CloudTrail 로그 항목입니다. `Decrypt` `Encrypt` `DescribeKey`

CreateGrant

원장 보호를 위해 고객 관리 키를 지정하면 QLDB는 사용자를 대신하여 KMS 키에 대한 액세스를 AWS KMS 허용하라는 요청을 `CreateGrant` 보냅니다. 또한 QLDB는 원장을 삭제할 때 `RetireGrant` 작업을 사용하여 부여한 권한을 제거합니다.

QLDB가 생성하는 권한 부여는 원장마다 다릅니다. `CreateGrant` 요청에 있는 보안 주체는 테이블을 생성한 사용자입니다.

`CreateGrant` 작업을 기록하는 이벤트는 다음 예시 이벤트와 유사합니다. 파라미터에는 테이블에 대한 CMK의 Amazon 리소스 이름(ARN), 피부여자 주체 및 삭제 보안 주체(QLDB 서비스), 권한 부여가 적용되는 작업이 포함됩니다.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAIOSFODNN7EXAMPLE:sample-user",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/sample-user",
        "accountId": "111122223333",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AKIAIOSFODNN7EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-06-04T21:37:11Z"
            }
        },
        "invokedBy": "qldb.amazonaws.com"
    },
    "eventTime": "2021-06-04T21:40:00Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "qldb.amazonaws.com",
    "userAgent": "qldb.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "granteePrincipal": "qldb.us-west-2.amazonaws.com",
        "operations": [
            "DescribeKey",
            "GenerateDataKey",
            "Decrypt",
            "Encrypt"
```

```
        ],
        "retiringPrincipal": "qldb.us-west-2.amazonaws.com"
    },
    "responseElements": {
        "grantId":
 "b3c83f999187ccc0979ef2ff86a1572237b6bba309c0ebce098c34761f86038a"
    },
    "requestID": "e99188d7-3b82-424e-b63e-e086d848ed60",
    "eventID": "88dc7ba5-4952-4d36-9ca8-9ab5d9598bab",
    "readOnly": false,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333"
}
```

GenerateDataKey

원장 보호를 위해 고객 관리형 키를 지정하면 QLDB는 고유한 데이터 키를 생성합니다. 원장의 고객 관리 키를 AWS KMS 지정하는 `GenerateDataKey` 요청을 보냅니다.

`GenerateDataKey` 작업을 기록하는 이벤트는 다음 예시 이벤트와 유사합니다. 사용자는 QLDB 서비스 계정입니다. 파라미터에는 고객 관리형 키의 ARN, 32바이트 길이가 필요한 데이터 키 지정자, 내부 키 계층 구조 노드를 식별하는 암호화 컨텍스트가 포함됩니다.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "qldb.amazonaws.com"
    },
    "eventTime": "2021-06-04T21:40:01Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-west-2",
```

```
    "sourceIPAddress": "qldb.amazonaws.com",
    "userAgent": "qldb.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "numberOfBytes": 32,
        "encryptionContext": {
            "key-hierarchy-node-id": "LY4HWMnkeZWKYi6MlitVJC",
            "key-hierarchy-node-version": "1"
        }
    },
    "responseElements": null,
    "requestID": "786977c9-e77c-467a-bff5-9ad5124a4462",
    "eventID": "b3f082cb-3e75-454e-bf0a-64be13075436",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333",
    "sharedEventID": "26688de5-0b1c-43d3-bc4f-a18029b08446"
}
```

Decrypt

원장에 액세스하면 QLDB는 원장의 암호화된 데이터에 액세스할 수 있도록 원장에 저장된 데이터 키를 해독하는 `Decrypt` 작업을 호출합니다.

`Decrypt` 작업을 기록하는 이벤트는 다음 예시 이벤트와 유사합니다. 사용자는 QLDB 서비스 계정입니다. 파라미터에는 고객 관리형 키의 ARN과 내부 키 계층 구조 노드를 식별하는 암호화 컨텍스트가 포함됩니다.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
```

```
        "invokedBy": "qldb.amazonaws.com"
    },
    "eventTime": "2021-06-04T21:40:56Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "qldb.amazonaws.com",
    "userAgent": "qldb.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "encryptionContext": {
            "key-hierarchy-node-id": "LY4HWMnkeZWKYi6MlitVJC",
            "key-hierarchy-node-version": "1"
        }
    },
    "responseElements": null,
    "requestID": "28f2dd18-3cc1-4fe2-82f7-5154f4933ebf",
    "eventID": "603ad5d4-4744-4505-9c21-bd4a6cbd4b20",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333",
    "sharedEventID": "7b6ce3e3-a764-42ec-8f90-5418c97ec411"
}
```


암호화

QLDB는 고객 관리형 키를 사용하여 일반 텍스트를 사이퍼텍스트로 암호화하는 `Encrypt` 작업을 호출합니다.

Encrypt 작업을 기록하는 이벤트는 다음 예시 이벤트와 유사합니다. 사용자는 QLDB 서비스 계정입니다. 파라미터에는 고객 관리형 키의 ARN과 원장의 내부 고유 ID를 지정하는 암호화 컨텍스트가 포함됩니다.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "qldb.amazonaws.com"
    },
    "eventTime": "2021-06-04T21:40:01Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Encrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "qldb.amazonaws.com",
    "userAgent": "qldb.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "encryptionContext": {
            "LedgerId": "F6qRNziJLUXA4Vy2ZUv8YY"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "b2daca7d-4606-4302-a2d7-5b3c8d30c64d",
    "eventID": "b8aace05-2e37-4fed-ae6f-a45a1c6098df",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333",
    "sharedEventID": "ce420ab0-288e-4b4f-aee8-541e18a28aa5"
}
```

DescribeKey

QLDB는 `DescribeKey` 작업을 호출하여 지정된 KMS 키가 AWS 계정 및 리전에 있는지 확인합니다.

`DescribeKey` 작업을 기록하는 이벤트는 다음 예시 이벤트와 유사합니다. 보안 주체는 KMS 키를 지정한 AWS 계정 사용자입니다. 파라미터는 고객 관리형 키의 ARN을 포함합니다.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAIOSFODNN7EXAMPLE:sample-user",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/sample-user",
        "accountId": "111122223333",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AKIAIOSFODNN7EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-06-04T21:37:11Z"
            }
        },
        "invokedBy": "qldb.amazonaws.com"
    },
    "eventTime": "2021-06-04T21:40:00Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "qldb.amazonaws.com",
    "userAgent": "qldb.amazonaws.com",
    "requestParameters": {
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
```

```
    "requestID": "a30586af-c783-4d25-8fda-33152c816c36",
    "eventID": "7a9caf07-2b27-44ab-afe4-b259533ebb88",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "111122223333"
}
```

Amazon QLDB의 전송 중 암호화

Amazon QLDB는 Secure Sockets Layer(SSL)/전송 계층 보안(TLS)을 사용하여 네트워크 트래픽을 보호하는 HTTPS 프로토콜을 사용하는 보안 연결만 허용합니다. 전송 중 암호화는 QLDB를 오가는 데이터를 암호화하여 데이터 보호를 한층 강화합니다. 조직의 정책, 업계나 정부 규범 및 규정 준수 요건에 따라 유휴 시 암호화를 사용하여 애플리케이션의 데이터 보안을 강화해야 할 수 있습니다.

또한 QLDB는 일부 지역에서 FIPS 엔드포인트를 제공합니다. 표준 AWS 엔드포인트와 달리 FIPS 엔드포인트에서는 Federal Information Processing Standard(FIPS) 140-2을 준수하는 TLS 소프트웨어 라이브러리를 사용합니다. 이러한 엔드포인트는 미국 정부와 상호 작용하는 기업에 필요할 수 있습니다. 자세한 내용은 AWS 일반 참조의 FIPS 엔드포인트를 참조하세요. QLDB에 사용할 수 있는 리전 및 엔드포인트의 전체 목록은 Amazon QLDB 엔드포인트 및 할당량을 참조하세요.

Amazon QLDB용 ID 및 액세스 관리

AWS Identity and Access Management (IAM) 은 관리자가 리소스에 대한 액세스를 안전하게 제어할 수 AWS 서비스 있도록 AWS 도와줍니다. IAM 관리자는 누가 QLDB 리소스를 사용하도록 인증되고(로그인됨) 권한이 부여되는지(권한 있음)를 제어합니다. IAM은 추가 AWS 서비스 비용 없이 사용할 수 있습니다.

주제

고객

사용 방법 AWS Identity and Access Management (IAM) 은 QLDB에서 수행하는 작업에 따라 다릅니다.

서비스 사용자 - QLDB 서비스를 사용하여 작업을 수행하는 경우 필요한 보안 인증과 권한을 관리자가 제공합니다. 더 많은 QLDB 기능을 사용하여 작업을 수행하게 되면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. QLDB의 기능에 액세스할 수 없는 경우 Amazon QLDB 자격 증명 및 액세스 문제 해결 섹션을 참조하세요.

서비스 관리자 - 회사에서 QLDB 리소스를 책임지고 있는 경우 QLDB에 대한 전체 액세스 권한을 가지고 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 QLDB 기능과 리소스를 결정합니다. 그런 다음, IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해합니다. 회사가 QLDB에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 Amazon QLDB에서 IAM을 사용하는 방법 섹션을 참조하세요.

IAM 관리자 - IAM 관리자라면 QLDB에 대한 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶을 것입니다. IAM에서 사용할 수 있는 Amazon QLDB의 자격 증명 기반 정책 예 자격 증명 기반 정책 예제를 보려면 섹션을 참조하세요.

자격 증명을 통한 인증

인증은 ID 자격 증명을 AWS 사용하여 로그인하는 방법입니다. IAM 사용자로 인증 (로그인 AWS) 하거나 IAM 역할을 맡아 인증 (로그인) 해야 합니다. AWS 계정 루트 사용자

ID 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 ID로 로그인할 수 있습니다. AWS AWS IAM Identity Center (IAM ID 센터) 사용자, 회사의 싱글 사인온 인증, Google 또는 Facebook 자격 증명이 페

더레이션 ID의 예입니다. 페더레이션형 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 액세스하는 경우 AWS 간접적으로 역할을 맡게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 로그인에 대한 자세한 내용은 AWS 로그인 사용 설명서의 내 로그인 방법을 참조하십시오. AWS AWS 계정

AWS 프로그래밍 방식으로 액세스하는 경우 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트 (SDK) 와 명령줄 인터페이스 (CLI) 를 AWS 제공합니다. AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 직접 요청에 서명하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 AWS API 요청 서명을 참조하십시오.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS 계정의 보안을 강화하기 위해 다단계 인증 (MFA) 을 사용할 것을 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 다중 인증 및 IAM 사용자 설명서의 AWS에서 다중 인증(MFA) 사용을 참조합니다.

AWS 계정 루트 사용자

계정을 AWS 계정만들 때는 먼저 계정의 모든 AWS 서비스 리소스에 대한 완전한 액세스 권한을 가진 하나의 로그인 ID로 시작합니다. 이 ID를 AWS 계정 루트 사용자라고 하며, 계정을 만들 때 사용한 이메일 주소와 비밀번호로 로그인하여 액세스할 수 있습니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 보안 인증 정보를 보호하고 루트 사용자만 수행할 수 있는 태스크를 수행하는 데 사용하세요. 루트 사용자로 로그인해야 하는 태스크의 전체 목록은 IAM 사용자 안내서의 루트 사용자 보안 인증이 필요한 태스크를 참조하세요.

페더레이션 자격 증명

가장 좋은 방법은 관리자 액세스가 필요한 사용자를 비롯한 수동 AWS 서비스 사용자가 ID 공급자와의 페더레이션을 사용하여 임시 자격 증명을 사용하여 액세스하도록 하는 것입니다.

페더레이션 ID는 기업 사용자 디렉토리, 웹 ID 공급자, Identity Center 디렉터리의 사용자 또는 ID 소스를 통해 제공된 자격 증명을 사용하여 액세스하는 AWS 서비스 모든 사용자를 말합니다. AWS Directory Service페더레이션 ID에 AWS 계정액세스하면 이들이 역할을 맡고 역할은 임시 자격 증명을 제공합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center을 사용하는 것이 좋습니다. IAM Identity Center에서 사용자 및 그룹을 생성하거나 자체 ID 소스의 사용자 및 그룹 집합에 연결하고 동기화하여 모든 사용자 및 애플리케이션에서 사용할 수 있습니다. AWS 계정 IAM Identity Center에 대한 자세한 내용은AWS IAM Identity Center 사용 설명서에서 IAM Identity Center란 무엇입니까?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 사용자 또는 애플리케이션에 대한 특정 권한을 AWS 계정 가진 사용자 내 자격 증명입니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명이 있는 IAM 사용자를 생성하는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우 정기적으로 액세스 키 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 자격 증명을 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 정보는 IAM 사용 설명서의 IAM 사용자를 만들어야 하는 경우(역할이 아님)를 참조하세요.

IAM 역할

IAM 역할은 특정 권한을 가진 사용자 AWS 계정 내의 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 역할을 AWS Management Console 전환하여 에서 일시적으로 IAM 역할을 맡을 수 있습니다. AWS CLI 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 정보는 IAM 사용 설명서의 IAM 역할 사용을 참조하세요.

임시 보안 인증 정보가 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 아이덴티티에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 연동 자격 증명이 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 역할에 대한 자세한 내용은 IAM 사용 설명서의 타사 자격 증명 공급자의 역할 만들기를 참조하세요. IAM Identity Center를 사용하는 경우 권한 세트를 구성합니다. 인증 후 자격 증명이 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 세트를 IAM의 역할과 연관짓습니다. 권한 세트에 대한 자세한 내용은AWS IAM Identity Center 사용 설명서의 권한 세트를 참조하세요.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 태스크에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 크로스 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입

니다. 그러나 일부 AWS 서비스경우에는 역할을 프록시로 사용하는 대신 정책을 리소스에 직접 연결할 수 있습니다. 크로스 계정 액세스를 위한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조하세요.

- 서비스 간 액세스 — 일부는 다른 AWS 서비스서비스의 기능을 AWS 서비스 사용합니다. 예컨대, 어떤 서비스에서 호출을 수행하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 순방향 액세스 세션 (FAS) — IAM 사용자 또는 역할을 사용하여 작업을 수행하는 경우 보안 AWS 주체로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 전화를 거는 주체의 권한을 다운스트림 AWS 서비스서비스에 AWS 서비스 요청하기 위한 요청과 결합하여 사용합니다. FAS 요청은 다른 서비스 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 서비스가 수신한 경우에만 이루어집니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 태스크를 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용자 설명서의 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조합니다.
  - 서비스 연결 역할 — 서비스 연결 역할은 에 연결된 서비스 역할의 한 유형입니다. AWS 서비스서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은 사용자에게 AWS 계정 표시되며 해당 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.
- Amazon EC2에서 실행되는 애플리케이션 — IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 API 요청을 AWS CLI 하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. AWS 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있게 하려면 인스턴스에 연결된 인스턴스 프로필을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.

IAM 역할을 사용할지 또는 IAM 사용자를 사용할지를 알아보려면 IAM 사용자 설명서의 IAM 역할(사용자 대신)을 생성하는 경우를 참조합니다.

정책을 사용한 액세스 관리

정책을 생성하고 이를 AWS ID 또는 리소스에 AWS 연결하여 액세스를 제어할 수 있습니다. 정책은 ID 또는 리소스와 연결될 때 AWS 해당 권한을 정의하는 객체입니다. AWS 주도자 (사용자, 루트 사용자

또는 역할 세션) 가 요청할 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은 JSON 문서로 AWS 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 JSON 정책 개요를 참조하세요.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자와 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole`태스크를 허용하는 정책이 있다고 가정합니다. 해당 정책을 사용하는 사용자는 AWS Management Console, AWS CLI, 또는 AWS API에서 역할 정보를 가져올 수 있습니다.

ID 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 자격 증명에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는 지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성을 참조하세요.

자격 증명 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 내 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립형 정책입니다. AWS 계정관리형 정책에는 AWS 관리형 정책과 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책과 인라인 정책의 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 보안 주체가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는 지를 정의합니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 연동 사용자 등이 포함될 수 있습니다. AWS 서비스

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. IAM의 AWS 관리형 정책은 리소스 기반 정책에 사용할 수 없습니다.

액세스 제어 목록(ACLs)

액세스 제어 목록(ACLs)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는 지를 제어합니다. ACLs는 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

ACL을 지원하는 서비스의 예로는 아마존 S3와 아마존 VPC가 있습니다. AWS WAF ACL에 대해 자세히 알아보려면 Amazon Simple Storage Service 개발자 안내서의 액세스 제어 목록(ACL) 개요를 참조하세요.

기타 정책 타입

AWS 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 타입에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 보안 인증 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 엔터티의 자격 증명 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 보안 주체로 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용자 설명서의 IAM 엔터티에 대한 권한 경계를 참조합니다.
- 서비스 제어 정책 (SCP) - SCP는 조직 또는 조직 단위 (OU) 에 대한 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations AWS Organizations 사업체가 소유한 여러 AWS 계정 개를 그룹화하고 중앙에서 관리하는 서비스입니다. 조직에서 모든 기능을 활성화할 경우 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 구성원 계정의 엔티티 (각 엔티티 포함) 에 대한 권한을 제한합니다. AWS 계정 루트 사용자조직 및 SCP에 대한 자세한 정보는AWS Organizations 사용 설명서의 SCP 작동 방식을 참조하세요.
- 세션 정책 – 세션 정책은 역할 또는 연합된 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할 자격 증명 기반 정책의 교차 및 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용자 설명서의 세션 정책을 참조합니다.

여러 정책 타입

여러 정책 타입이 요청에 적용되는 경우 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련되어 있을 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하십시오.

Amazon QLDB에서 IAM을 사용하는 방법

IAM을 사용하여 QLDB에 대한 액세스를 관리하기 전에 QLDB와 함께 사용할 수 있는 IAM 기능을 알아보세요.

Amazon QLDB에서 사용할 수 있는 IAM 기능

IAM 특성	QLDB 지원
ID 기반 정책	예
리소스 기반 정책	아니요
정책 작업	예
정책 리소스	예
정책 조건 키	예
ACLs	아니요
ABAC(정책의 태그)	예
임시 보안 인증	예
보안 주체 권한	아니요
서비스 역할	예
서비스 연결 역할	아니요

QLDB 및 AWS 서비스 기타 기능이 대부분의 IAM 기능과 어떻게 작동하는지 자세히 알아보려면 IAM 사용 설명서의 IAM과AWS 서비스 호환되는 기능을 참조하십시오.

QLDB에 대한 자격 증명 기반 정책

ID 기반 정책 지원	예

자격 증명 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 자격 증명에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는 지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성을 참조하세요.

IAM 자격 증명 기반 정책을 사용하면 허용되거나 거부되는 작업와 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. 자격 증명 기반 정책에서는 보안 주체가 연결된 사용자 또는 역할에 적용되므로 보안 주체를 지정할 수 없습니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용자 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

QLDB 자격 증명 기반 정책 예제

QLDB 자격 증명 기반 정책의 예를 보려면 Amazon QLDB의 자격 증명 기반 정책 예 섹션을 참조하세요.

QLDB 내 리소스 기반 정책

리소스 기반 정책 지원	아니요

리소스 기반 정책은 리소스에 연결하는 JSON 정책 문서입니다. 리소스 기반 정책의 예는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 보안 주체가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는 지를 정의합니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 연동 사용자 등이 포함될 수 있습니다. AWS 서비스

계정 간 액세스를 활성화하려는 경우 전체 계정이나 다른 계정의 IAM 엔터티를 리소스 기반 정책의 보안 주체로 지정할 수 있습니다. 리소스 기반 정책에 크로스 계정 보안 주체를 추가하는 것은 트러스트 관계 설정의 절반밖에 되지 않는다는 것을 유념하세요. 보안 주체와 리소스가 다른 AWS 계정경우 신뢰할 수 있는 계정의 IAM 관리자는 보안 주체 개체 (사용자 또는 역할) 에게 리소스에 액세스할 수 있는 권한도 부여해야 합니다. 개체에 자격 증명 기반 정책을 연결하여 권한을 부여합니다. 하지만 리소스 기반 정책이 동일 계정의 보안 주체에 액세스를 부여하는 경우 추가 자격 증명 기반 정책이 필요하지 않습니다. 자세한 정보는 IAM 사용자 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조하세요.

QLDB 정책 작업

정책 작업 지원	예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action`요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 태스크를 설명합니다. 정책 작업은 일반적으로 관련 AWS API 작업과 이름이 같습니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하십시오.

QLDB 작업 목록을 보려면 서비스 승인 참조의 Amazon QLDB에서 정의한 작업을 참조하세요.

QLDB의 정책 작업은 작업 앞에 다음 접두사를 사용합니다.

```
qldb
```

단일 문에서 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "qldb:action1",
      "qldb:action2"
]
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, `Describe`라는 단어로 시작하는 모든 태스크를 지정하려면 다음 태스크를 포함합니다.

```
"Action": "qldb:Describe*"
```

원장에서 PartiQL 문을 실행하여 QLDB 트랜잭션 데이터 API(QLDB 세션)와 상호 작용하려면 다음과 같이 `SendCommand` 작업에 권한을 부여해야 합니다.

```
"Action": "qldb:SendCommand"
```

STANDARD 권한 모드의 원장에 대해서는 각 PartiQL 명령에 필요한 추가 권한에 대한 PartiQL 권한 참조를 참조하세요.

QLDB 자격 증명 기반 정책의 예를 보려면 Amazon QLDB의 자격 증명 기반 정책 예 섹션을 참조하세요.

QLDB 정책 리소스

정책 리소스 지원	예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 개체를 지정합니다. 문장에는 `Resource`또는 `NotResource`요소가 반드시 추가되어야 합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 타입을 지원하는 작업에 대해 이 작업을 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

QLDB 리소스 유형 및 해당 ARN 목록을 보려면 서비스 승인 참조에서 Amazon QLDB에서 정의한 리소스를 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 Amazon QLDB에서 정의한 작업을 참조하세요.

QLDB에서 기본 리소스는 원장입니다. QLDB는 테이블과 스트림과 같은 추가 리소스 유형도 지원합니다. 하지만 기존 원장의 컨텍스트에서만 테이블과 스트림을 생성할 수 있습니다.

QLDB 테이블은 원장 저널에 있는 정렬되지 않은 문서 수정본 모음의 구체화된 뷰입니다. 원장의 `STANDARD` 권한 모드에서는 이 테이블 리소스에서 PartiQL 문을 실행할 권한을 부여하는 IAM 정책을 생성해야 합니다. 테이블 리소스에 대한 권한이 있으면 테이블의 현재 상태에 액세스하는 문을 실행할 수 있습니다. 또한 내장 `history()` 함수를 사용하여 테이블의 개정 이력을 쿼리할 수 있습니다. 자세한 내용은 Amazon QLDB에서 표준 권한 모드로 시작하기 섹션을 참조하세요.

> **Note**
>
> `CREATE TABLE` 문은 고유한 ID와 제공된 테이블 이름을 사용하여 테이블을 생성합니다. 제공된 테이블 이름은 활성된 모든 테이블에서 고유해야 합니다. 하지만 QLDB를 사용하면 테이블을 비활성화할 수 있으므로 동일한 테이블 이름을 공유하는 비활성 테이블이 여러 개 있을 수

있습니다. 따라서 테이블 리소스 ARN은 사용자 정의 테이블 이름이 아닌 시스템에서 할당한 고유 ID를 참조합니다.

또한 각 원장은 원장의 모든 테이블과 인덱스를 나열하기 위해 쿼리할 수 있는 시스템 정의 카탈로그 리소스를 제공합니다. QLDB 데이터 객체 모델에 대한 자세한 내용은 Amazon QLDB의 핵심 개념 및 용어 섹션을 참조하세요.

다음 표와 같이 리소스에는 고유한 ARN이 연결되어 있습니다.

리소스 유형	ARN
ledger	`arn:${Partition}:qldb:${Region}:${Account}:ledger/${LedgerName}`
table	`arn:${Partition}:qldb:${Region}:${Account}:ledger/${LedgerName}/table/${TableId}`
catalog	`arn:${Partition}:qldb:${Region}:${Account}:ledger/${LedgerName}/information_schema/user_tables`
stream	`arn:${Partition}:qldb:${Region}:${Account}:stream/${LedgerName}/${StreamId}`

예를 들어, `myExampleLedger` 리소스를 문에 지정하려면 다음 ARN을 사용합니다.

```
"Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
```

단일 문에서 여러 리소스를 지정하려면 ARN을 쉼표로 구분합니다.

```
"Resource": [
      "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger1",
      "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger2"
]
```

QLDB 자격 증명 기반 정책의 예를 보려면 Amazon QLDB의 자격 증명 기반 정책 예 섹션을 참조하세요.

QLDB 정책 조건 키

서비스별 정책 조건 키 지원	예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지 지정할 수 있습니다.

`Condition` 요소(또는 `Condition` 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. `Condition` 요소는 옵션입니다. 같거나 작음과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 `Condition`요소를 지정하거나 단일 `Condition`요소에서 여러 키를 지정하는 경우 AWS 는 논리적 `AND`태스크를 사용하여 평가합니다. 단일 조건 키에 여러 값을 지정하는 경우 는 논리적 `OR` 연산을 사용하여 조건을 AWS 평가합니다. 명문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예를 들어, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그를 참조하세요.

AWS 글로벌 조건 키 및 서비스별 조건 키를 지원합니다. 모든 AWS 글로벌 조건 키를 보려면 IAM 사용 AWS 설명서의 글로벌 조건 컨텍스트 키를 참조하십시오.

QLDB 조건 키 목록을 보려면 서비스 승인 참조에서 Amazon QLDB의 조건 키를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 Amazon QLDB에서 정의한 작업을 참조하세요.

`PartiQLDropIndex` 및 `PartiQLDropTable` 작업은 `qldb:Purge` 조건 키를 지원합니다. 이 조건 키는 PartiQL `DROP` 문에 지정된 `purge` 값을 기준으로 액세스를 필터링합니다. 하지만 QLDB는 현재 `DROP INDEX` 문의 `purge = true`과 `DROP TABLE` 문의 `purge = false`만 지원합니다. 다른 QLDB 작업은 일부 전역 조건 키를 지원합니다.

QLDB 자격 증명 기반 정책의 예를 보려면 Amazon QLDB의 자격 증명 기반 정책 예 섹션을 참조하세요.

액세스 제어 목록(ACL)

ACL 지원	아니요

액세스 제어 목록(ACLs)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는 지를 제어합니다. ACLs는 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

QLDB와 함께하는 ABAC(속성 기반 액세스 제어)

ABAC 지원(정책의 태그)	예

ABAC(속성 기반 액세스 제어)는 속성을 기반으로 권한을 정의하는 권한 부여 전략입니다. AWS에서는 이러한 속성을 태그라고 합니다. IAM 개체 (사용자 또는 역할) 및 여러 AWS 리소스에 태그를 첨부할 수 있습니다. ABAC의 첫 번째 단계로 개체 및 리소스에 태그를 지정합니다. 그런 다음 보안 주체의 태그가 액세스하려는 리소스의 태그와 일치할 때 작업을 허용하도록 ABAC 정책을 설계합니다.

ABAC는 빠르게 성장하는 환경에서 유용하며 정책 관리가 번거로운 상황에 도움이 됩니다.

태그를 기반으로 액세스를 제어하려면 `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`* 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

서비스가 모든 리소스 타입에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 타입에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

ABAC에 대한 자세한 정보는 IAM 사용 설명서의 ABAC란 무엇인가요?를 참조하세요. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용자 설명서의 ABAC(속성 기반 액세스 제어) 사용을 참조하세요.

QLDB 리소스 태깅에 대한 자세한 내용은 Amazon QLDB 리소스 태그 지정 섹션을 참조하세요.

리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예제는 태그를 기준으로 QLDB 원장 업데이트 섹션에서 확인할 수 있습니다.

QLDB와 함께 임시 보안 인증 정보 사용

임시 보안 인증 지원	예

임시 자격 증명을 사용하여 로그인하면 작동하지 AWS 서비스 않는 것도 있습니다. 임시 자격 증명을 사용하는 방법을 AWS 서비스 비롯한 추가 정보는 IAM 사용 설명서의 IAM과AWS 서비스 연동되는 내용을 참조하십시오.

사용자 이름과 암호를 제외한 다른 방법을 AWS Management Console 사용하여 로그인하면 임시 자격 증명을 사용하는 것입니다. 예를 들어 회사의 SSO (Single Sign-On) 링크를 AWS 사용하여 액세스

하는 경우 이 프로세스에서 자동으로 임시 자격 증명을 생성합니다. 또한 콘솔에 사용자로 로그인한 다음 역할을 전환할 때 임시 보안 인증을 자동으로 생성합니다. 역할 전환에 대한 자세한 정보는 IAM 사용 설명서의 역할로 전환(콘솔)을 참조하세요.

또는 API를 사용하여 임시 자격 증명을 수동으로 생성할 수 있습니다 AWS CLI . AWS 그런 다음 해당 임시 자격 증명을 사용하여 액세스할 수 AWS있습니다. AWS 장기 액세스 키를 사용하는 대신 임시 자격 증명을 동적으로 생성할 것을 권장합니다. 자세한 정보는 IAM의 임시 보안 보안 인증 섹션을 참조하세요.

QLDB의 교차 서비스 간 보안 주체 권한

전달 액세스 세션(FAS) 지원	아니요

IAM 사용자 또는 역할을 사용하여 작업을 수행하는 AWS경우 보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 전화를 거는 주체의 권한을 다운스트림 서비스에 AWS 서비스 요청하기 위한 요청과 함께 사용합니다. AWS 서비스 FAS 요청은 다른 서비스 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 서비스가 수신한 경우에만 이루어집니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.

QLDB에 대한 서비스 역할

서비스 역할 지원	예

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용자 설명서의 역할을 생성하여 AWS 서비스에게 권한 위임을 참조하세요.

> **Warning**
>
> 서비스 역할에 대한 권한을 변경하면 ACM 기능이 중단될 수 있습니다. ACM에서 관련 지침을 제공하는 경우에만 서비스 역할을 편집하세요.

QLDB는 다음 섹션에 설명된 대로 `ExportJournalToS3` 및 `StreamJournalToKinesis` API 작업에 대한 서비스 역할을 지원합니다.

QLDB에서 IAM 역할 선택

QLDB에서 저널 블록을 내보내거나 스트리밍하는 경우 QLDB가 사용자를 대신하여 주어진 대상에 객체를 쓸 수 있는 역할을 선택해야 합니다. 이전에 서비스 역할을 생성한 경우 QLDB는 선택할 수 있는 역할 목록을 제공합니다. 내보내기의 경우 지정된 Amazon S3 버킷에, 스트림의 경우 지정된 Amazon Kinesis Data Streams 리소스에 쓰기 액세스를 허용하는 역할을 선택하는 것이 중요합니다. 자세한 내용은 QLDB의 저널 내보내기 권한 또는 QLDB의 스트림 권한을 참조하세요.

> **Note**
>
> 저널 스트림을 요청할 때 QLDB에 역할을 전달하려면 IAM 역할 리소스에서 `iam:PassRole` 작업을 수행할 수 있는 권한이 있어야 합니다. 이는 QLDB 원장 리소스에서 `qldb:ExportJournalToS3`를, 또는 QLDB 스트림 하위 리소스에서 `qldb:StreamJournalToKinesis`를 수행할 수 있는 권한에 추가됩니다.

QLDB에 대한 서비스 연결 역할

서비스 연결 역할 지원	아니요

서비스 연결 역할은 에 연결된 서비스 역할의 한 유형입니다. AWS 서비스서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은 사용자에게 AWS 계정 표시되며 해당 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 IAM으로 작업하는AWS 서비스를 참조하세요. 서비스 연결 역할 열에서 `Yes`가 포함된 서비스를 테이블에서 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 Yes(네) 링크를 선택합니다.

Amazon QLDB에서 표준 권한 모드로 시작하기

이 섹션을 사용하여 Amazon QLDB의 표준 권한 모드를 시작하세요. 이 섹션에서는 AWS Identity and Access Management (IAM)에서 QLDB의 PartiQL 작업 및 테이블 리소스에 대한 ID 기반 정책을 작성할 때 도움이 되는 참조 테이블을 제공합니다. 또한 IAM에서 권한 정책을 생성하는 방법에 대한 step-by-step 자습서와 QLDB에서 테이블 ARN을 찾고 테이블 태그를 생성하는 방법에 대한 지침도 포함되어 있습니다.

주제

- STANDARD 권한 모드
- PartiQL 권한 참조
- 테이블 ID 및 ARN 찾기
- 태그 지정 테이블
- 빠른 시작 자습서: 권한 정책 생성

STANDARD 권한 모드

QLDB는 이제 원장 리소스에 대한 STANDARD 권한 모드를 지원합니다. 원장, 테이블 및 PartiQL 명령에 대해 보다 세분화된 액세스 제어를 가능하게 하는 표준 권한 모드입니다. 기본적으로 이 모드는 이 원장의 모든 테이블에서 PartiQL 명령을 실행하려는 모든 요청을 거부합니다.

> **Note**
>
> 이전에는 원장에 사용할 수 있는 유일한 권한 모드는 ALLOW_ALL이었습니다. ALLOW_ALL 모드를 사용하면 원장에 대한 API 수준 세분화로 액세스 제어가 가능하며 QLDB 원장의 경우 계속 지원되지만 권장되지는 않습니다. 이 모드를 사용하면 SendCommand API 권한이 있는 사용자가 권한 정책에 의해 지정된 원장의 모든 테이블에서 모든 PartiQL 명령을 실행할 수 있습니다(따라서 '모두 허용' PartiQL 명령).
>
> 기존 원장의 권한 모드를 ALLOW_ALL에서 STANDARD로 변경할 수 있습니다. 자세한 내용은 표준 권한 모드로 마이그레이션을 참조하세요.

표준 모드에서 명령을 허용하려면 IAM에서 특정 테이블 리소스 및 PartiQL 작업에 대한 권한 정책을 생성해야 합니다. 이는 원장에 대한 SendCommand API 권한에 추가됩니다. 이 모드에서 정책을 용이하게 하기 위해 QLDB는 PartiQL 명령에 대한 IAM 작업 세트와 QLDB 테이블에 대한 Amazon 리소스 이름(ARN)을 도입했습니다. QLDB 데이터 객체 모델에 대한 자세한 내용은 Amazon QLDB의 핵심 개념 및 용어 섹션을 참조하세요.

PartiQL 권한 참조

다음 표에는 각 QLDB PartiQL 명령, 명령을 수행하기 위해 권한을 부여해야 하는 해당 IAM 작업, 권한을 부여할 수 있는 리소스가 AWS 나열되어 있습니다. 정책의 Action필드에서 작업을 지정하고, 정책의 Resource필드에서 리소스 값을 지정합니다.

Important

- 이러한 PartiQL 명령에 권한을 부여하는 IAM 정책은 STANDARD 권한 모드가 원장에 할당된 경우에만 원장에 적용됩니다. 이러한 정책은 권한 모드의 ALLOW_ALL 원장에는 적용되지 않습니다.

  원장을 생성하거나 업데이트할 때 권한 모드를 지정하는 방법을 알아보려면 콘솔 시작하기의 Amazon QLDB 원장의 기본 작업 또는 1단계: 새 원장 생성 섹션을 참조하세요.
- 원장에서 PartiQL 명령을 실행하려면 원장 리소스에 대한 SendCommand API 작업에 대한 권한도 부여해야 합니다. 이는 다음 테이블에 나열된 PartiQL 작업 및 테이블 리소스에 추가됩니다. 자세한 내용은 데이터 트랜잭션 실행을 참조하십시오.

Amazon QLDB PartiQL 명령 및 필요한 권한

Command	필요한 권한(IAM 작업)	리소스	종속 작업
CREATE TABLE	qldb:PartiQLCreateTable	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*	qldb:TagResource (생성 시 태그 지정용)
DROP TABLE	qldb:PartiQLDropTable	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	
UNDROP TABLE	qldb:PartiQLUndropTable	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	
CREATE INDEX	qldb:PartiQLCreateIndex	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	

Command	필요한 권한(IAM 작업)	리소스	종속 작업
DROP INDEX	qldb:PartiQLDropIndex	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	
DELETE FROM-REMOVE(전체 문서용)	qldb:PartiQLDelete	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	qldb:PartiQLSelect
INSERT	qldb:PartiQLInsert	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	
UPDATE FROM (INSERT, REMOVE, 또는 SET)	qldb:PartiQLUpdate	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	qldb:PartiQLSelect
REDACT_REVISION(저장 프로시저)	qldb:PartiQLRedact	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	
SELECT FROM table_name	qldb:PartiQLSelect	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	

Command	필요한 권한(IAM 작업)	리소스	종속 작업
SELECT FROM information_schema.user_tables	qldb:PartiQLSelect	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /information_schema/user_tables	
SELECT FROM history(table_name)	qldb:PartiQLHistoryFunction	arn:aws:qldb: *region*:*account-id* :ledger/*ledger-name* /table/*table-id*	

이러한 PartiQL 명령에 권한을 부여하는 IAM 정책 문서의 예는 빠른 시작 자습서: 권한 정책 생성 또는 Amazon QLDB의 자격 증명 기반 정책 예를 참조하세요.

테이블 ID 및 ARN 찾기

를 사용하거나 테이블 정보_schema.user_tables를 쿼리하여 테이블 ID를 찾을 수 AWS Management Console 있습니다. 콘솔에서 테이블 세부 정보를 보거나 이 시스템 카탈로그 테이블을 쿼리하려면 시스템 카탈로그 리소스에 대한 SELECT 권한이 있어야 합니다. 예를 들어, 다음 문을 실행하여 Vehicle 테이블의 테이블 ID를 찾을 수 있습니다.

```
SELECT * FROM information_schema.user_tables
WHERE name = 'Vehicle'
```

이 쿼리는 다음 예와 비슷한 형식으로 결과를 반환합니다.

```
{
    tableId: "Au1EiThbt8s0z9wM26REZN",
    name: "Vehicle",
    indexes: [
        { indexId: "Djg2nt0yIs2GY0T29Kud1z", expr: "[VIN]", status: "ONLINE" },
        { indexId: "4tPW3fUhaVhDinRgKRLhGU", expr: "[LicensePlateNumber]", status:
 "BUILDING" }
```

```
    ],
    status: "ACTIVE"
}
```

테이블에서 PartiQL 문을 실행할 권한을 부여하려면 다음 ARN 형식으로 테이블 리소스를 지정합니다.

```
arn:aws:qldb:${region}:${account-id}:ledger/${ledger-name}/table/${table-id}
```

다음은 테이블 ID `Au1EiThbt8s0z9wM26REZN`에 대한 테이블 ARN의 예입니다.

```
arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/Au1EiThbt8s0z9wM26REZN
```

콘솔 사용

QLDB 콘솔을 사용하여 테이블을 생성할 수도 있습니다.

테이블의 ARN을 찾으려면(콘솔)

1. 에 AWS Management Console로그인하고 https://console.aws.amazon.com/qldb 에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 원장을 선택합니다.
3. 원장 목록에서 찾으려면 테이블 ARN이 포함된 원장 이름을 선택합니다.
4. 원장 세부 정보 페이지의 테이블 탭에서 찾으려는 ARN이 포함된 테이블 이름을 찾습니다. ARN을 복사하려면 옆에 있는 복사 아이콘 ( ) 을 선택합니다.

태그 지정 테이블

테이블 리소스에 태그를 지정할 수 있습니다. 기존 테이블의 태그를 관리하려면 AWS Management Console 또는 API 작업 `TagResource`, `UntagResource`, 및 을 사용하십시오. `ListTagsForResource` 자세한 정보는 Amazon QLDB 리소스 태그 지정을 참조하세요.

> **Note**
>
> 테이블 리소스는 루트 원장 리소스의 태그를 상속하지 않습니다.
> 테이블 생성 시 태그 지정은 현재 `STANDARD` 권한 모드의 원장에만 지원됩니다.

테이블을 만드는 동안 QLDB 콘솔을 사용하거나 CREATE TABLE PartiQL 문에 테이블 태그를 지정하여 테이블 태그를 정의할 수도 있습니다. 다음 예제에서는 environment=production 태그가 있는 Vehicle이라는 테이블을 생성합니다.

```
CREATE TABLE Vehicle WITH (aws_tags = `{'environment': 'production'}`)
```

생성 시 테이블에 태그를 지정하려면 qldb:PartiQLCreateTable 및 qldb:TagResource 작업 모두에 액세스할 수 있어야 합니다.

리소스를 생성하는 동안 태그를 지정하면 리소스 생성 후 사용자 지정 태그 지정 스크립트를 실행할 필요가 없습니다. 테이블에 태그가 지정된 후 해당 태그를 기반으로 테이블에 대한 액세스를 제어할 수 있습니다. 예를 들어 특정 태그가 있는 테이블에만 전체 액세스 권한을 부여할 수 있습니다. JSON 정책의 예는 테이블 태그를 기반으로 하는 모든 작업에 대한 전체 액세스 권한 섹션을 참조하세요.

콘솔 사용

테이블을 생성하는 동안 QLDB 콘솔을 사용하여 테이블 태그를 정의할 수도 있습니다.

생성 시 테이블 태그 지정하기(콘솔)

1. 에 AWS Management Console로그인하고 https://console.aws.amazon.com/qldb 에서 Amazon QLDB 콘솔을 엽니다.
2. 탐색 창에서 원장을 선택합니다.
3. 원장 목록에서 테이블을 생성할 원장 이름을 선택합니다.
4. 원장 세부 정보 페이지의 테이블 탭에서 테이블 생성을 선택합니다.
5. 테이블 생성 페이지에서 다음 작업을 수행합니다.
   - 테이블 이름 – 테이블 이름을 입력합니다.
   - 태그 – 태그를 키-값 쌍으로로 연결하여 메타데이터를 테이블에 추가합니다. 테이블을 정리하고 식별하는 데 도움이 되도록 테이블에 태그를 추가할 수 있습니다.

     태그 추가를 선택한 다음 키-값 쌍을 적절히 입력합니다.
6. 원하는 대로 설정되었으면 테이블 생성을 선택합니다.

빠른 시작 자습서: 권한 정책 생성

이 자습서는 STANDARD 권한 모드에서 Amazon QLDB 원장에 대한 IAM의 권한 정책을 생성하는 단계를 안내합니다. 그런 다음 사용자, 그룹 또는 역할에 권한을 할당할 수 있습니다.

PartiQL 명령 및 테이블 리소스에 권한을 부여하는 IAM 정책 문서의 추가 예는 Amazon QLDB의 자격 증명 기반 정책 예 섹션을 참조하세요.

주제

- 필수 조건
- 읽기 전용 정책 생성
- 전체 액세스 정책 생성
- 특정 테이블에 대한 읽기 전용 정책 생성
- 권한 할당

필수 조건

시작하기 전에 다음을 수행해야 합니다.

1. 아직 설정하지 Amazon QLDB 액세스 않았다면 에 있는 AWS 설정 지침을 따르십시오. 이 단계에는 관리 사용자 등록 AWS 및 생성이 포함됩니다.
2. 새 원장을 생성하고 해당 원장의 STANDARD 권한 모드를 선택합니다. 방법을 알아보려면 콘솔 시작하기의 1단계: 새 원장 생성 또는 Amazon QLDB 원장의 기본 작업 섹션을 참조하세요.

읽기 전용 정책 생성

JSON 정책 편집기를 사용하여 표준 권한 모드에서 원장의 모든 테이블에 대한 읽기 전용 정책을 생성하려면 다음 작업을 수행합니다.

1. https://console.aws.amazon.com/iam/ 에서 AWS Management Console 로그인하고 IAM 콘솔을 엽니다.
2. 왼쪽의 탐색 열에서 정책을 선택합니다.

   정책을 처음으로 선택하는 경우 관리형 정책 소개 페이지가 나타납니다. 시작하기를 선택합니다.
3. 페이지 상단에서 정책 생성을 선택합니다.
4. JSON 탭을 선택합니다.
5. 다음 JSON 정책 문서를 복사하여 붙여 넣습니다. 이 예제 정책은 원장의 모든 테이블에 읽기 전용 액세스 권한을 부여합니다.

   이 정책을 사용하려면 예제에서 *us-east-1,* 123456789012를 사용자 고유의 정보로 바꾸십시오. *myExampleLedger*

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBSendCommandPermission",
      "Effect": "Allow",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    },
    {
      "Sid": "QLDBPartiQLReadOnlyPermissions",
      "Effect": "Allow",
      "Action": [
        "qldb:PartiQLSelect",
        "qldb:PartiQLHistoryFunction"
      ],
      "Resource": [
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
      ]
    }
  ]
}
```

6. 정책 검토를 선택합니다.

> **Note**
>
> 언제든지 시각적 편집기 및 JSON 탭을 전환할 수 있습니다. 그러나 변경을 수행하거나 시각적 편집기 탭에서 정책 검토를 선택한 경우 IAM은 시각적 편집기에 최적화되도록 정책을 재구성할 수 있습니다. 자세한 내용은 IAM 사용자 설명서의 정책 재구성을 참조하세요.

7. 정책 검토 페이지에서 생성하려는 정책의 이름과 설명(선택 사항)을 입력합니다. 정책 요약을 검토하여 정책이 부여한 권한을 확인합니다. 그런 다음 정책 생성을 선택하여 작업을 저장합니다.

전체 액세스 정책 생성

표준 권한 모드에서 QLDB 원장의 모든 테이블에 대한 전체 액세스 정책을 만들려면 다음 작업을 수행합니다.

- 다음 정책 문서를 사용하여 이전 단계를 반복합니다. 이 예제 정책은 와일드카드(*)를 사용하여 원장 내의 모든 PartiQL 작업과 모든 리소스를 포함함으로써 원장의 모든 테이블의 모든 PartiQL 명령에 대한 액세스 권한을 부여합니다.

> **Warning**
>
> 이 예제에서는 와일드카드 문자(*)를 사용하여 QLDB 원장의 모든 테이블에 대한 관리 및 읽기/쓰기 작업을 비롯한 모든 PartiQL 작업을 허용합니다. 대신 허용할 각 작업과 해당 사용자, 역할 또는 그룹에 필요한 작업만 명시적으로 지정하는 것이 좋습니다.

이 정책을 사용하려면 예제에서 *us-east-1,* 123456789012를 사용자 고유의 정보로 바꾸십시오. *myExampleLedger*

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBSendCommandPermission",
      "Effect": "Allow",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    },
    {
      "Sid": "QLDBPartiQLFullPermissions",
      "Effect": "Allow",
      "Action": [
        "qldb:PartiQL*"
      ],
      "Resource": [
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
      ]
    }
  ]
}
```

특정 테이블에 대한 읽기 전용 정책 생성

표준 권한 모드에서 QLDB 원장의 특정 테이블에 대한 읽기 전용 액세스 정책을 만들려면 다음 작업을 수행합니다.

1. 를 사용하거나 시스템 카탈로그 테이블을 쿼리하여 테이블의 ARN을 찾습니다. AWS Management Console `information_schema.user_tables` 지침은 테이블 ID 및 ARN 찾기을 참조하세요.
2. 테이블 ARN을 사용하여 테이블에 대한 읽기 전용 액세스를 허용하는 정책을 생성합니다. 이렇게 하려면 다음 정책 문서를 사용하여 이전 단계를 반복합니다.

   이 예제 정책은 지정된 테이블 전용 읽기 전용 액세스를 부여합니다. 이 예시에서의 테이블 ID는 Au1EiThbt8s0z9wM26REZN입니다. *# ### ##### ### us-east-1, 123456789012 # Au1 myExampleLedger8s0z9WM26Rezn# ### ## ### ######. EiThbt*

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "QLDBSendCommandPermission",
      "Effect": "Allow",
      "Action": "qldb:SendCommand",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
    },
    {
      "Sid": "QLDBPartiQLReadOnlyPermissions",
      "Effect": "Allow",
      "Action": [
        "qldb:PartiQLSelect",
        "qldb:PartiQLHistoryFunction"
      ],
      "Resource": [
        "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
table/Au1EiThbt8s0z9wM26REZN"
      ]
    }
  ]
}
```

권한 할당

QLDB 권한 정책을 생성한 후 다음과 같이 권한을 할당합니다.

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- 다음과 AWS IAM Identity Center같은 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 따르십시오.

Amazon QLDB의 자격 증명 기반 정책 예

기본적으로 사용자 및 역할에는 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS Command Line Interface (AWS CLI) 또는 AWS API를 사용하여 작업을 수행할 수 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 맡을 수 있습니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용자 설명서의 IAM 정책 생성을 참조하세요.

각 리소스 유형에 대한 ARN 형식을 포함하여 QLDB에서 정의한 작업 및 리소스 유형에 대한 자세한 내용은 서비스 권한 부여 참조에서 Amazon QLDB에 대한 작업, 리소스 및 조건 키를 참조하세요.

목차

- 쿼리 기록 권한
- 쿼리 기록이 없는 전체 액세스 콘솔 권한
- 사용자가 자신의 고유한 권한을 볼 수 있도록 허용
- 데이터 트랜잭션 실행
  - PartiQL 작업 및 테이블 리소스에 대한 표준 권한
    - 모든 작업에 대한 모든 액세스 허용
    - 테이블 태그를 기반으로 하는 모든 작업에 대한 전체 액세스 권한
    - 읽기/쓰기 액세스
    - 읽기 전용 액세스
    - 특정 테이블에 대한 읽기 전용 액세스
    - 테이블 생성 액세스 허용
    - 요청 태그를 기반으로 테이블 생성 액세스 허용
- Amazon S3 버킷으로 저널 내보내기
- Kinesis Data Streams로 저널 스트리밍
- 태그를 기준으로 QLDB 원장 업데이트

정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 QLDB 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따르세요.

- AWS 관리형 정책으로 시작하고 최소 권한 권한으로 이동 — 사용자와 워크로드에 권한을 부여하려면 여러 일반적인 사용 사례에 권한을 부여하는 AWS 관리형 정책을 사용하세요. 해당 내용은 에서 사용할 수 있습니다. AWS 계정사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 더 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용 설명서의 AWS managed policies(관리형 정책) 또는 AWS managed policies for job functions(직무에 대한 관리형 정책)를 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우 태스크를 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 Policies and permissions in IAM(IAM의 정책 및 권한)을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어 SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책

조건을 작성할 수 있습니다. 예를 AWS 서비스들어 특정 작업을 통해 서비스 작업을 사용하는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 AWS CloudFormation있습니다. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.

- IAM Access Analyzer를 통해 IAM 정책을 검증하여 안전하고 기능적인 권한 보장 – IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 신규 및 기존 정책을 검증합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 정보는 IAM 사용 설명서의 IAM Access Analyzer 정책 검증을 참조하tpdy.
- 멀티 팩터 인증 (MFA) 필요 - IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 AWS 계정 MFA를 활성화하십시오. API 작업을 직접 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 정보는 IAM 사용 설명서의 Configuring MFA-protected API access(MFA 보호 API 액세스 구성)를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용자 설명서의 IAM의 보안 모범 사례를 참조하세요.

QLDB 콘솔 사용

Amazon QLDB 콘솔에 액세스하려면 최소한의 권한 집합이 있어야 합니다. 이러한 권한을 통해 QLDB 리소스의 세부 정보를 나열하고 볼 수 있어야 합니다. AWS 계정최소 필수 권한보다 더 제한적인 자격 증명 기반 정책을 만들면 콘솔이 해당 정책에 연결된 엔터티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

AWS CLI 또는 API만 호출하는 사용자에게 최소 콘솔 권한을 허용할 필요는 없습니다. AWS 그 대신, 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자와 역할이 QLDB 콘솔 및 모든 기능에 대한 전체 액세스 권한을 갖도록 하려면 AWS 다음 관리형 정책을 엔티티에 연결하십시오. 자세한 내용은 IAM 사용자 설명서의 AWS 아마존 QLDB에 대한 관리형 정책 및 사용자에게 권한 추가를 참조하세요.

```
AmazonQLDBConsoleFullAccess
```

쿼리 기록 권한

QLDB 권한 외에도 일부 콘솔 기능에는 Database Query Metadata Service(서비스 접두사: dbqms)에 대한 권한이 필요합니다. 이 서비스는 QLDB 및 기타 AWS 서비스에 대한 콘솔 쿼리 편집기에서 최근 쿼리와 저장된 쿼리를 관리하는 내부 전용 서비스입니다. DBQMS API 작업의 전체 목록은 서비스 승인 참조의 Database Query Metadata Service를 참조하세요.

쿼리 기록 권한을 허용하려면 AmazonQLDB의 AWS 관리형 정책을 사용할 수 있습니다. ConsoleFullAccess 이 정책은 와일드카드(dbqms:*)를 사용하여 모든 리소스에 대해 모든 DBQMS 작업을 허용합니다.

또는 사용자 지정 IAM 정책을 생성하고 다음 DBQMS 작업을 포함할 수 있습니다. QLDB 콘솔의 PartiQL 쿼리 편집기에는 쿼리 기록 기능에 이러한 작업을 사용할 수 있는 권한이 필요합니다.

```
dbqms:CreateFavoriteQuery
dbqms:CreateQueryHistory
dbqms:DeleteFavoriteQueries
dbqms:DeleteQueryHistory
dbqms:DescribeFavoriteQueries
dbqms:DescribeQueryHistory
dbqms:UpdateFavoriteQuery
```

쿼리 기록이 없는 전체 액세스 콘솔 권한

쿼리 기록 권한 없이 QLDB 콘솔에 대한 전체 액세스를 허용하려면 모든 DBQMS 작업을 제외하는 사용자 지정 IAM 정책을 만들 수 있습니다. 예를 들어, 다음 정책 문서는 서비스 접두사로 시작하는 작업을 제외하고 AWS 관리형 정책 ConsoleFullAccessAmazonQLDB에서 부여한 것과 동일한 권한을 허용합니다. dbqms

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "qldb:CreateLedger",
        "qldb:UpdateLedger",
        "qldb:UpdateLedgerPermissionsMode",
        "qldb:DeleteLedger",
        "qldb:ListLedgers",
        "qldb:DescribeLedger",
        "qldb:ExportJournalToS3",
        "qldb:ListJournalS3Exports",
        "qldb:ListJournalS3ExportsForLedger",
        "qldb:DescribeJournalS3Export",
        "qldb:CancelJournalKinesisStream",
        "qldb:DescribeJournalKinesisStream",
        "qldb:ListJournalKinesisStreamsForLedger",
        "qldb:StreamJournalToKinesis",
        "qldb:GetBlock",
```

```
        "qldb:GetDigest",
        "qldb:GetRevision",
        "qldb:TagResource",
        "qldb:UntagResource",
        "qldb:ListTagsForResource",
        "qldb:SendCommand",
        "qldb:ExecuteStatement",
        "qldb:ShowCatalog",
        "qldb:InsertSampleData",
        "qldb:PartiQLCreateIndex",
        "qldb:PartiQLDropIndex",
        "qldb:PartiQLCreateTable",
        "qldb:PartiQLDropTable",
        "qldb:PartiQLUndropTable",
        "qldb:PartiQLDelete",
        "qldb:PartiQLInsert",
        "qldb:PartiQLUpdate",
        "qldb:PartiQLSelect",
        "qldb:PartiQLHistoryFunction"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Action": [
        "kinesis:ListStreams",
        "kinesis:DescribeStream"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "qldb.amazonaws.com"
        }
      }
    }
  ]
}
```

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예시는 IAM 사용자가 자신의 사용자 자격 증명에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 API를 사용하여 프로그래밍 방식으로 이 작업을 완료할 수 있는 권한이 포함됩니다. AWS CLI AWS

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

데이터 트랜잭션 실행

원장에서 PartiQL 문을 실행하여 QLDB 트랜잭션 데이터 API(QLDB 세션션)와 상호 작용하려면 SendCommand API 작업에 권한을 부여해야 합니다. 다음 JSON 문서는 원장 myExampleLedger의 SendCommand API 작업에만 권한을 부여하는 정책의 예입니다.

이 정책을 사용하려면 예제에서 *us-east-1,* 123456789012를 사용자 고유의 정보로 바꾸십시오. *myExampleLedger*

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        }
    ]
}
```

myExampleLedger가 ALLOW_ALL 권한 모드를 사용하는 경우 이 정책은 원장의 모든 테이블에 모든 PartiQL 명령을 실행할 수 있는 권한을 부여합니다.

또한 AWS 관리형 정책을 사용하여 모든 QLDB 리소스에 대한 전체 액세스 권한을 부여할 수 있습니다. 자세한 정보는 AWS 아마존 QLDB에 대한 관리형 정책을 참조하세요.

PartiQL 작업 및 테이블 리소스에 대한 표준 권한

STANDARD 권한 모드의 원장의 경우 적절한 PartiQL 권한 부여의 예로 다음 IAM 정책 문서를 참조할 수 있습니다. 각 PartiQL 명령에 필요한 권한 목록은 PartiQL 권한 참조 섹션을 참조하세요.

주제

- 모든 작업에 대한 모든 액세스 허용
- 테이블 태그를 기반으로 하는 모든 작업에 대한 전체 액세스 권한
- 읽기/쓰기 액세스
- 읽기 전용 액세스
- 특정 테이블에 대한 읽기 전용 액세스

- 테이블 생성 액세스 허용
- 요청 태그를 기반으로 테이블 생성 액세스 허용

모든 작업에 대한 모든 액세스 허용

다음 JSON 정책 문서는 myExampleLedger의 모든 테이블에 모든 PartiQL 명령을 사용할 수 있는 전체 액세스 권한을 부여합니다. 이 정책은 원장의 ALLOW_ALL 권한 모드를 사용하는 것과 동일한 효과를 생성합니다.

이 정책을 사용하려면 예제에서 *us-east-1,* 123456789012를 사용자 고유의 정보로 바꾸십시오. *myExampleLedger*

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
        {
            "Sid": "QLDBPartiQLFullPermissions",
            "Effect": "Allow",
            "Action": [
                "qldb:PartiQLCreateIndex",
                "qldb:PartiQLDropIndex",
                "qldb:PartiQLCreateTable",
                "qldb:PartiQLDropTable",
                "qldb:PartiQLUndropTable",
                "qldb:PartiQLDelete",
                "qldb:PartiQLInsert",
                "qldb:PartiQLUpdate",
                "qldb:PartiQLRedact",
                "qldb:PartiQLSelect",
                "qldb:PartiQLHistoryFunction"
            ],
            "Resource": [
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
            ]
```

```
        }
    ]
}
```

테이블 태그를 기반으로 하는 모든 작업에 대한 전체 액세스 권한

다음 JSON 정책 문서는 테이블 리소스 태그를 기반으로 하는 조건을 사용하여 myExampleLedger의 모든 테이블에 모든 PartiQL 명령을 사용할 수 있는 전체 액세스 권한을 부여합니다. 테이블 태그 environment에 값 development이 있는 경우에만 권한이 부여됩니다.

> **Warning**
>
> 이 예제에서는 와일드카드 문자(*)를 사용하여 QLDB 원장의 모든 테이블에 대한 관리 및 읽기/쓰기 작업을 비롯한 모든 PartiQL 작업을 허용합니다. 대신 허용할 각 작업과 해당 사용자, 역할 또는 그룹에 필요한 작업만 명시적으로 지정하는 것이 좋습니다.

이 정책을 사용하려면 예제에서 *us-east-1,* 123456789012를 사용자 고유의 정보로 바꾸십시오. *myExampleLedger*

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
        {
            "Sid": "QLDBPartiQLFullPermissionsBasedOnTags",
            "Effect": "Allow",
            "Action": [
                "qldb:PartiQL*"
            ],
            "Resource": [
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
            ],
            "Condition": {
```

```
                "StringEquals": { "aws:ResourceTag/environment": "development" }
            }
        }
    ]
}
```

읽기/쓰기 액세스

다음 JSON 정책 문서는 myExampleLedger에 있는 모든 테이블의 데이터를 선택, 삽입, 업데이트 및 삭제할 수 있는 권한을 부여합니다. 이 정책은 데이터를 삭제하거나 스키마를 변경할 권한(예: 테이블 및 인덱스 생성 및 삭제)을 부여하지 않습니다.

> **Note**
>
> UPDATE 문에는 수정 중인 테이블에 대한 qldb:PartiQLUpdate 및 qldb:PartiQLSelect 작업 모두에 대한 권한이 필요합니다. UPDATE 문을 실행하면 업데이트 작업 외에 읽기 작업도 수행됩니다. 두 작업을 모두 요구하면 테이블의 콘텐츠를 읽을 수 있는 사용자에게만 UPDATE 권한이 부여됩니다.
> 마찬가지로 DELETE 문에는 qldb:PartiQLDelete 및 qldb:PartiQLSelect 작업 모두에 대한 권한이 필요합니다.

이 정책을 사용하려면 예제에서 *us-east-1,* 123456789012를 사용자 고유의 정보로 바꾸십시오. *myExampleLedger*

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
        {
            "Sid": "QLDBPartiQLReadWritePermissions",
            "Effect": "Allow",
            "Action": [
                "qldb:PartiQLDelete",
                "qldb:PartiQLInsert",
                "qldb:PartiQLUpdate",
```

```
                "qldb:PartiQLSelect",
                "qldb:PartiQLHistoryFunction"
            ],
            "Resource": [
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
            ]
        }
    ]
}
```

읽기 전용 액세스

다음 JSON 정책 문서는 myExampleLedger의 모든 테이블에 읽기 전용 권한을 부여합니다. 이 정책을 사용하려면 예제에서 *us-east-1,* 123456789012를 사용자 고유의 정보로 바꾸십시오. *myExampleLedger*

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
        {
            "Sid": "QLDBPartiQLReadOnlyPermissions",
            "Effect": "Allow",
            "Action": [
                "qldb:PartiQLSelect",
                "qldb:PartiQLHistoryFunction"
            ],
            "Resource": [
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
            ]
        }
    ]
}
```

특정 테이블에 대한 읽기 전용 액세스

다음 JSON 정책 문서는 myExampleLedger의 특정 테이블에 대한 읽기 전용 권한을 부여합니다. 이 예시에서의 테이블 ID는 Au1EiThbt8s0z9wM26REZN입니다.

 # ### ##### ### us-east-1, 123456789012 # Au1 myExampleLedger8s0z9WM26Rezn# ### ## ### ######. EiThbt

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
        {
            "Sid": "QLDBPartiQLReadOnlyPermissionsOnTable",
            "Effect": "Allow",
            "Action": [
                "qldb:PartiQLSelect",
                "qldb:PartiQLHistoryFunction"
            ],
            "Resource": [
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
table/Au1EiThbt8s0z9wM26REZN"
            ]
        }
    ]
}
```

테이블 생성 액세스 허용

다음 JSON 정책 문서는 myExampleLedger에서 테이블을 생성할 수 있는 권한을 부여합니다. qldb:PartiQLCreateTable 작업을 수행하려면 테이블 리소스 유형에 대한 권한이 필요합니다. 하지만 CREATE TABLE 문을 실행할 당시에는 새 테이블의 테이블 ID를 알 수 없습니다. 따라서 qldb:PartiQLCreateTable 권한을 부여하는 정책은 테이블 ARN에서 와일드카드(*)를 사용하여 리소스를 지정해야 합니다.

이 정책을 사용하려면 예제에서 *us-east-1,* 123456789012를 사용자 고유의 정보로 바꾸십시오. *myExampleLedger*

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
        {
            "Sid": "QLDBPartiQLCreateTablePermission",
            "Effect": "Allow",
            "Action": [
                "qldb:PartiQLCreateTable"
            ],
            "Resource": [
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*"
            ]
        }
    ]
}
```

요청 태그를 기반으로 테이블 생성 액세스 허용

다음 JSON 정책 문서는 `aws:RequestTag` 컨텍스트 키 기반 조건을 사용하여 myExampleLedger에서 테이블을 생성할 수 있는 권한을 부여합니다. 요청 태그 `environment`에 값 `development`이 있는 경우에만 권한이 부여됩니다. 생성 시 테이블에 태그를 지정하려면 `qldb:PartiQLCreateTable` 및 `qldb:TagResource` 작업 모두에 액세스할 수 있어야 합니다. 테이블을 생성할 때 태그를 지정하는 방법을 알아보려면 태그 지정 테이블 섹션을 참조하세요.

이 정책을 사용하려면 예제에서 *us-east-1*, 123456789012를 사용자 고유의 정보로 바꾸십시오. *myExampleLedger*

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBSendCommandPermission",
            "Effect": "Allow",
            "Action": "qldb:SendCommand",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
```

```
        {
            "Sid": "QLDBPartiQLCreateTablePermission",
            "Effect": "Allow",
            "Action": [
                "qldb:PartiQLCreateTable",
                "qldb:TagResource"
            ],
            "Resource": [
                "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*"
            ],
            "Condition": {
                "StringEquals": { "aws:RequestTag/environment": "development" }
            }
        }
    ]
}
```

Amazon S3 버킷으로 저널 내보내기

1단계: QLDB 저널 내보내기 권한

다음 예제에서는 사용자에게 QLDB 원장 리소스에서 작업을 수행할 수 있는 AWS 계정 권한을 부여합니다. `qldb:ExportJournalToS3` 또한 QLDB 서비스에 전달하려는 IAM 역할 리소스에서 `iam:PassRole` 작업을 수행할 수 있는 권한도 부여합니다. 이는 모든 저널 내보내기 요청에 필요합니다.

 # ### ##### ### us-east-1, 123456789012 myExampleLedger# qldb-s3-export# ## # ## ### ######.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBJournalExportPermission",
            "Effect": "Allow",
            "Action": "qldb:ExportJournalToS3",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
        },
        {
            "Sid": "IAMPassRolePermission",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::123456789012:role/qldb-s3-export",
```

```
            "Condition": {
               "StringEquals": {
                  "iam:PassedToService": "qldb.amazonaws.com"
               }
            }
        }
    ]
}
```

2단계: Amazon S3 버킷에 대한 권한 설정

다음 예제에서는 IAM 역할을 사용하여 QLDB에 Amazon S3 버킷 중 하나인 DOC-EXAMPLE-BUCKET에 쓸 수 있는 액세스 권한을 부여합니다. 이는 모든 QLDB 저널 내보내기에도 필요합니다.

정책은 s3:PutObject 권한을 부여할 뿐만 아니라 객체에 대한 액세스 제어 목록(ACL) 권한을 설정할 수 있는 기능에 대한 s3:PutObjectAcl 권한도 부여합니다.

이 정책을 사용하려면 예제에서 DOC-EXAMPLE-BUCKET을 Amazon S3 버킷 이름으로 대체합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBJournalExportS3Permissions",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
        }
    ]
}
```

그런 다음 이 권한 정책을 QLDB가 Amazon S3 버킷에 액세스하기 위해 맡을 수 있는 IAM 역할에 연결합니다. 다음 JSON 문서는 QLDB가 계정 123456789012의 모든 QLDB 리소스에 대해서만 IAM 역할을 맡도록 허용하는 신뢰 정책의 예입니다.

이 정책을 사용하려면 예제의 *us-east-1* 및 *123456789012*를 사용자 고유의 정보로 바꾸세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "qldb.amazonaws.com"
            },
            "Action": [ "sts:AssumeRole" ],
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:qldb:us-east-1:123456789012:*"
                },
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                }
            }
        }
    ]
}
```

Kinesis Data Streams로 저널 스트리밍

1단계: QLDB 저널 스트림 권한

다음 예제에서는 원장의 모든 QLDB 스트림 하위 리소스에서 `qldb:StreamJournalToKinesis` 작업을 수행할 수 있는 AWS 계정 권한을 사용자에게 부여합니다. 또한 QLDB 서비스에 전달하려는 IAM 역할 리소스에서 `iam:PassRole` 작업을 수행할 수 있는 권한도 부여합니다. 이는 모든 저널 스트림 요청에 필요합니다.

이 정책을 사용하려면 예시에서 *us-east-1, myExampleLedger*123456789012 및 를 사용자 고유의 정보로 바꾸십시오. *qldb-kinesis-stream*

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBJournalStreamPermission",
            "Effect": "Allow",
            "Action": "qldb:StreamJournalToKinesis",
            "Resource": "arn:aws:qldb:us-east-1:123456789012:stream/myExampleLedger/*"
        },
        {
            "Sid": "IAMPassRolePermission",
            "Effect": "Allow",
            "Action": "iam:PassRole",
```

```
            "Resource": "arn:aws:iam::123456789012:role/qldb-kinesis-stream",
            "Condition": {
               "StringEquals": {
                  "iam:PassedToService": "qldb.amazonaws.com"
               }
            }
        }
    ]
}
```

2단계: Kinesis Data Streams 권한

다음 예시에서는 IAM 역할을 사용하여 QLDB에 Amazon Kinesis 데이터 스트림에 데이터 레코드를 쓸 수 있는 액세스 권한을 부여합니다. *stream-for-qldb* 이는 모든 저널 스트림 요청에 필요합니다.

이 정책을 사용하려면 예제에서 *us-east-1,* 123456789012를 사용자 고유의 정보로 바꾸십시오. *stream-for-qldb*

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBStreamKinesisPermissions",
            "Action": [ "kinesis:PutRecord*", "kinesis:DescribeStream",
 "kinesis:ListShards" ],
            "Effect": "Allow",
            "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/stream-for-qldb"
        }
    ]
}
```

그런 다음 이 권한 정책을 QLDB가 Kinesis 데이터 스트림에 액세스하기 위해 맡을 수 있는 IAM 역할에 연결합니다. 다음 JSON 문서는 QLDB가 원장 123456789012의 계정 myExampleLedger의 모든 QLDB 스트림에 대해서만 IAM 역할을 맡을 수 있도록 허용하는 신뢰 정책의 예입니다.

이 정책을 사용하려면 예제에서 *us-east-1,* 123456789012를 사용자 고유의 정보로 바꾸십시오. *myExampleLedger*

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Principal": {
                "Service": "qldb.amazonaws.com"
            },
            "Action": [ "sts:AssumeRole" ],
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": "arn:aws:qldb:us-
east-1:123456789012:stream/myExampleLedger/*"
                },
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                }
            }
        }
    ]
}
```

태그를 기준으로 QLDB 원장 업데이트

자격 증명 기반 정책의 조건을 사용하여 태그를 기반으로 QLDB 리소스에 대한 액세스를 제어할 수 있습니다. 이 예제에서는 원장을 업데이트하도록 허용하는 정책을 생성할 수 있는 방법을 보여줍니다. 하지만 원장 태그 `Owner`에 해당 사용자의 사용자 이름 값이 있는 경우에만 권한이 부여됩니다. 이 정책은 콘솔에서 이 작업을 완료하는 데 필요한 권한도 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListLedgersInConsole",
            "Effect": "Allow",
            "Action": "qldb:ListLedgers",
            "Resource": "*"
        },
        {
            "Sid": "UpdateLedgerIfOwner",
            "Effect": "Allow",
            "Action": "qldb:UpdateLedger",
            "Resource": "arn:aws:qldb:*:*:ledger/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
            }
        }
```

```
    ]
}
```

이 정책을 계정의 사용자에게 연결할 수 있습니다. `richard-roe`라는 사용자가 QLDB 원장을 업데이트하려는 경우 원장에 `Owner=richard-roe` 또는 `owner=richard-roe` 태그를 지정해야 합니다. 그렇지 않으면 액세스가 거부됩니다. 조건 키 이름은 대소문자를 구분하지 않기 때문에 조건 태그 키 `Owner`는 `Owner` 및 `owner` 모두와 일치합니다. 자세한 정보는 IAM 사용자 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.

교차 서비스 혼동된 대리자 예방

혼동된 대리자 문제는 작업을 수행할 권한이 없는 엔터티가 권한이 더 많은 엔터티에게 작업을 수행하도록 강요할 수 있는 보안 문제입니다. 에서 서비스 간 사칭은 대리인 AWS혼동을 야기할 수 있습니다.

교차 서비스 가장은 한 서비스(직접 호출하는 서비스)가 다른 서비스(직접 호출되는 서비스)를 직접 호출할 때 발생할 수 있습니다. 직접 호출하는 서비스는 다른 고객의 리소스에 대해 액세스 권한이 없는 방식으로 작동하게 권한을 사용하도록 조작될 수 있습니다. 대리인이 혼동하는 문제를 방지하기 위해 에서는 계정의 리소스에 대한 액세스 권한이 부여된 서비스 보안 주체를 통해 모든 서비스의 데이터를 보호하는 데 도움이 되는 도구를 AWS 제공합니다.

Amazon QLDB가 리소스에 다른 서비스를 제공하는 권한을 제한하려면 리소스 정책에서 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키를 사용하는 것이 좋습니다. 두 글로벌 조건 컨텍스트 키를 모두 사용하는 경우 `aws:SourceAccount` 값과 `aws:SourceArn` 값의 계정은 동일한 정책 문에서 사용할 경우 동일한 계정 ID를 사용해야 합니다.

다음 표에는 `ExportJournalToS3` 및 `StreamsJournalToKinesis` QLDB API 작업에 사용할 수 있는 값 `aws:SourceArn`이 나와 있습니다. 이러한 작업은 지정한 IAM 역할을 맡기 위해 AWS Security Token Service (AWS STS) 를 호출하기 때문에 이 보안 문제의 범위에 속합니다.

API 작업	호출된 서비스	aws: SourceArn
`ExportJournalToS3`	AWS STS (`AssumeRole`)	QLDB가 계정의 모든 QLDB 리소스에 대한 역할을 맡을 수 있도록 허용합니다. `arn:aws:qldb: us-east-1 :123456789012 :*`

API 작업	호출된 서비스	aws: SourceArn
		현재 QLDB는 저널 내보내기에 대해 이 와일드카드 ARN만 지원합니다.

StreamsJournalToKinesis	AWS STS (AssumeRole)	QLDB가 특정 QLDB 스트림에 대한 역할을 맡을 수 있도록 허용합니다.
		`arn:aws:qldb: us-east-1 :123456789012 :stream/myExample Ledger /IiPT4brpZCqCq3f4MTHbYy`
		참고: 스트림 리소스가 생성된 후에만 ARN에서 스트림 ID를 지정할 수 있습니다. 이 ARN을 사용하면 단일 QLDB 스트림에만 역할을 사용하도록 허용할 수 있습니다.
		QLDB가 원장의 모든 QLDB 스트림에 대한 역할을 맡을 수 있도록 허용합니다.
		`arn:aws:qldb: us-east-1 :123456789012 :stream/myExample Ledger /*`
		QLDB가 계정의 모든 QLDB 스트림에 대한 역할을 맡을 수 있도록 허용합니다.
		`arn:aws:qldb: us-east-1 :123456789012 :stream/*`
		QLDB가 계정의 모든 QLDB 리소스에 대한 역할을 맡을 수 있도록 허용합니다.
		`arn:aws:qldb: us-east-1 :123456789012 :*`

혼동된 대리인 문제로부터 보호하는 가장 효과적인 방법은 리소스의 전체 ARN이 포함된 `aws:SourceArn` 글로벌 조건 컨텍스트 키를 사용하는 것입니다. 리소스의 전체 ARN을 모르거나 여러 리소스를 지정하는 경우, ARN의 알 수 없는 부분(예: `arn:aws:qldb:us-east-1:123456789012:*`)에 대해 와일드카드 문자(*)를 포함한 `aws:SourceArn` 글로벌 조건 컨텍스트 키를 사용합니다.

다음 예는 신뢰 정책에서 IAM 역할이 `aws:SourceArn` 및 `aws:SourceAccount` 전역 조건 컨텍스트 키를 사용하여 혼동된 대리자 문제를 방지하는 방법을 보여줍니다. 이 신뢰 정책을 사용하면 QLDB는 원장 `myExampleLedger`에 대한 계정 123456789012의 모든 QLDB 스트림에 대한 역할만 맡을 수 있습니다.

이 정책을 사용하려면 예제에서 *us-east-1,* 123456789012를 사용자 고유의 정보로 바꾸십시오. *myExampleLedger*

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "qldb.amazonaws.com"
    },
    "Action": [ "sts:AssumeRole" ],
    "Condition": {
      "ArnEquals": {
        "aws:SourceArn": "arn:aws:qldb:us-east-1:123456789012:stream/myExampleLedger/*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

AWS 아마존 QLDB에 대한 관리형 정책

AWS 관리형 정책은 에서 생성하고 관리하는 독립 실행형 정책입니다. AWS AWS 관리형 정책은 많은 일반 사용 사례에 대한 권한을 제공하도록 설계되었으므로 사용자, 그룹 및 역할에 권한을 할당하기 시작할 수 있습니다.

AWS 관리형 정책은 모든 AWS 고객이 사용할 수 있으므로 특정 사용 사례에 대해 최소 권한 권한을 부여하지 않을 수도 있다는 점에 유의하세요. 사용 사례에 고유한 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다.

관리형 정책에 정의된 권한은 변경할 수 없습니다. AWS AWS 관리형 정책에 정의된 권한을 업데이트하는 경우 AWS 해당 업데이트는 정책이 연결된 모든 주체 ID (사용자, 그룹, 역할) 에 영향을 미칩니다.

AWS 새 API 작업이 시작되거나 기존 서비스에 새 AWS 서비스 API 작업을 사용할 수 있게 되면 AWS 관리형 정책을 업데이트할 가능성이 가장 높습니다.

자세한 내용은 IAM 사용자 설명서의 AWS 관리형 정책을 참조하세요.

AWS 이러한 관리형 정책의 QLDB API 작업에 대한 자세한 내용은 를 참조하십시오. Amazon QLDB API 참조

주제

- AWS 관리형 정책: AmazonQLDB ReadOnly
- AWS 관리형 정책: AmazonQLDB FullAccess
- AWS 관리형 정책: AmazonQLDB ConsoleFullAccess
- 관리형 정책에 대한 QLDB 업데이트 AWS

AWS 관리형 정책: AmazonQLDB ReadOnly

AmazonQLDB ReadOnly 정책을 사용하여 모든 QLDB 리소스에 읽기 전용 권한을 부여할 수 있습니다. 이 정책을 IAM 자격 증명에 연결할 수 있습니다.

권한 세부 정보

이 정책에는 `qldb` 서비스에 대한 다음 권한이 포함되어 있습니다.

- 주체가 모든 QLDB 리소스와 해당 태그를 설명하고 나열할 수 있도록 허용합니다. 이러한 리소스에는 원장, Amazon S3 내보내기 작업, Kinesis Data Streams로의 스트림이 포함됩니다.
- 보안 주체가 모든 원장의 저널에서 블록, 다이제스트 또는 수정본을 가져와 암호를 통해 데이터를 확인할 수 있도록 허용합니다.
- 주체가 원장의 모든 테이블에서 PartiQL 명령을 실행하는 것을 허용하지 않습니다.

AWS 관리형 정책: AmazonQLDB FullAccess

Amazon QLDB FullAccess 정책을 사용하면 QLDB API 또는 를 통해 모든 QLDB 리소스에 전체 관리 권한을 부여할 수 있습니다. AWS CLI이 정책을 IAM 자격 증명에 연결할 수 있습니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `qldb`
  - 보안 주체가 모든 QLDB 리소스 및 해당 태그를 생성, 설명, 나열 및 관리할 수 있도록 허용합니다. 이러한 리소스에는 원장, Amazon S3 내보내기 작업, Kinesis Data Streams로의 스트림이 포함됩니다.
  - 보안 주체가 QLDB 드라이버 또는 QLDB 쉘을 사용하여 모든 원장의 모든 테이블에서 모든 PartiQL 명령을 실행할 수 있도록 허용합니다.
  - 보안 주체가 모든 원장의 저널에서 블록, 다이제스트 또는 수정본을 가져와 암호를 통해 데이터를 확인할 수 있도록 허용합니다.
- `iam` – 주체가 사용자 계정의 모든 IAM 역할 리소스를 QLDB 서비스에 전달할 수 있도록 허용합니다. 이는 모든 저널 내보내기 및 스트림 요청을 위해 요구됩니다.

AWS 관리형 정책: AmazonQLDB ConsoleFullAccess

Amazon QLDB ConsoleFullAccess 정책을 사용하면 AWS Management Console, QLDB API 또는 를 통해 모든 QLDB 리소스에 전체 관리 권한을 부여할 수 있습니다. AWS CLI이 정책을 IAM 자격 증명에 연결할 수 있습니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `qldb`
  - 보안 주체가 모든 QLDB 리소스 및 해당 태그를 생성, 설명, 나열 및 관리할 수 있도록 허용합니다. 이러한 리소스에는 원장, Amazon S3 내보내기 작업, Kinesis Data Streams로의 스트림이 포함됩니다.
  - 보안 주체가 QLDB 콘솔, QLDB 드라이버 또는 QLDB 쉘을 사용하여 모든 원장의 모든 테이블에서 모든 PartiQL 명령을 실행할 수 있도록 허용합니다.
  - 보안 주체가 QLDB 콘솔을 사용하여 모든 원장에 샘플 애플리케이션 데이터를 삽입할 수 있도록 허용합니다.
  - 보안 주체가 모든 원장의 저널에서 블록, 다이제스트 또는 수정본을 가져와 암호를 통해 데이터를 확인할 수 있도록 허용합니다.
- `dbqms`– 보안 주체가 Database Query Metadata Service의 모든 작업을 사용할 수 있도록 허용합니다. 이 서비스는 QLDB 콘솔에서 PartiQL 쿼리 편집기에 대한 최근 및 저장된 쿼리를 만들고, 설명하고, 관리하는 데 필요한 내부 전용 서비스입니다.
- `kinesis` – 보안 주체가 Amazon Kinesis Data Streams 리소스를 설명하고 나열할 수 있도록 허용합니다. 이러한 리소스는 QLDB 스트림 리소스가 데이터를 쓸 수 있는 목표 대상입니다.

- `iam` – 주체가 사용자 계정의 모든 IAM 역할 리소스를 QLDB 서비스에 전달할 수 있도록 허용합니다. 이는 모든 저널 내보내기 및 스트림 요청을 위해 요구됩니다.

관리형 정책에 대한 QLDB 업데이트 AWS

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후 QLDB의 AWS 관리형 정책 업데이트에 대한 세부 정보를 확인하십시오. 이 페이지의 변경 사항에 대한 자동 알림을 받아보려면 QLDB 릴리스 기록 페이지에서 RSS 피드를 구독하세요.

변경 사항	설명	날짜
AmazonQLDBFullAccess, AmazonQLDB — 기존 정책에 대한 업데이트 ConsoleFullAccess	QLDB는 주체가 `STANDARD` 권한 모드에서 모든 원장의 문서 수정본 내용을 수정할 수 있도록 허용하는 새로운 권한을 추가했습니다.	2022년 11월 4일
아마존 QLDB, 아마존 QLDB — 기존 정책에 대한 업데이트 FullAccess ConsoleFullAccess	QLDB는 주체가 사용자 계정의 모든 IAM 역할 리소스를 QLDB 서비스에 전달할 수 있도록 허용하는 새로운 권한을 추가했습니다. 이는 모든 저널 내보내기 및 스트림 요청을 위해 요구됩니다.	2021년 9월 2일
AmazonQLDB ReadOnly — 기존 정책에 대한 업데이트	QLDB는 이전에 두 번 나열된 중복 `qldb:GetBlock` 작업을 제거하고 `"Action"` 필드 앞에 나타나도록 `"Effect"` 필드를 재정렬했습니다.	2021년 7월 1일
아마존 QLDB, 아마존 QLDB — 기존 정책에 대한 업데이트 FullAccess ConsoleFullAccess	QLDB는 주체가 모든 원장의 권한 모드를 업데이트하고 새 `STANDARD` 권한 모드에서 모든 원장의 모든 PartiQL 명령을 실	2021년 5월 27일

변경 사항	설명	날짜
	행할 수 있도록 허용하는 새로운 권한을 추가했습니다. STANDARD 권한 모드는 PartiQL 명령에 대한 테이블 수준의 액세스 제어 및 세분 수준을 지원합니다. 새로운 권한 모드를 용이하게 하기 위해 QLDB는 PartiQL 명령 유형에 대한 IAM 작업 세트와 QLDB 테이블 리소스에 대한 Amazon 리소스 이름(ARN)을 도입했습니다. 이 두 정책은 STANDARD 원장에 대한 전체 액세스 권한을 부여하는 새로운 PartiQL 작업을 포함하도록 업데이트되었습니다.	
QLDB이 변경 사항 추적 시작	QLDB는 관리형 정책의 변경 사항을 추적하기 시작했습니다. AWS	2021년 3월 1일

Amazon QLDB 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 QLDB 및 IAM에서 발생할 수 있는 공통적인 문제를 진단하고 수정할 수 있습니다.

주제

- QLDB에서 작업을 수행할 권한이 없음
- 저는 IAM을 수행할 권한이 없습니다. PassRole
- 외부 사용자가 내 QLDB 리소스에 액세스할 AWS 계정 수 있도록 허용하고 싶습니다.

QLDB에서 작업을 수행할 권한이 없음

작업을 수행할 권한이 없다는 AWS Management Console 메시지가 표시되면 관리자에게 도움을 요청해야 합니다. 관리자는 로그인 보안 인증 정보를 제공한 사람입니다.

다음 예제 오류는 `mateojackson` 사용자가 콘솔을 사용하여 가상 *myExampleLedger* 리소스에 대한 세부 정보를 보려고 하지만 가상 `qldb:`*DescribeLedger* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 qldb:DescribeLedger on resource: myExampleLedger
```

이 경우 Mateo는 *myExampleLedger* 작업을 사용하여 `qldb:`*DescribeLedger* 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

저는 IAM을 수행할 권한이 없습니다. PassRole

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

새 서비스 역할 또는 서비스 연결 역할을 만드는 대신 기존 역할을 해당 서비스에 전달할 AWS 서비스 수 있는 기능도 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예시 오류는 `marymajor`라는 IAM 사용자가 콘솔을 사용하여 QLDB에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우 Mary가 `iam:PassRole`작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요하면 관리자에게 문의하세요. AWS 관리자는 로그인 자격 증명을 제공한 사람입니다.

저널 내보내기 또는 스트림 작업과 관련된 이 오류에 대한 문제 해결 지침은 Amazon QLDB 문제 해결 섹션을 참조하세요.

외부 사용자가 내 QLDB 리소스에 액세스할 AWS 계정 수 있도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제

어 목록(ACL)을 지원하는 서비스의 경우 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- QLDB에서 이러한 기능을 지원하는지 여부를 알아보려면 Amazon QLDB에서 IAM을 사용하는 방법 섹션을 참조하세요.
- 소유한 리소스에 대한 액세스 권한을 AWS 계정 부여하는 방법을 알아보려면 IAM 사용 설명서의 다른 AWS 계정 IAM 사용자에게 액세스 권한 제공을 참조하십시오.
- 제3자에게 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정AWS 계정 소유에 대한 액세스 제공을 참조하십시오.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(자격 증명 연동)을 참조하세요.
- 크로스 계정 액세스를 위한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용자 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조하세요.

Amazon QLDB의 로깅 및 모니터링

모니터링은 Amazon QLDB 및 솔루션의 안정성, 가용성 및 성능을 유지하는 데 있어 중요한 부분입니다. AWS 다중 지점 장애가 발생할 경우 이를 보다 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분에서 모니터링 데이터를 수집해야 합니다. 하지만 QLDB 모니터링을 시작하기 전에 다음 질문에 대한 답변을 포함하는 모니터링 계획을 작성해야 합니다.

- 모니터링의 목표
- 모니터링할 리소스
- 이러한 리소스를 모니터링하는 빈도
- 사용할 모니터링 도구
- 모니터링 작업을 수행할 사람
- 문제 발생 시 알려야 할 대상

다음 단계에서는 다양한 시간과 다양한 부하 조건에서 성능을 측정하여 환경에서 일반 성능의 기준선을 설정합니다. QLDB가 과거 모니터링 데이터를 저장하는 것을 모니터링하면서 현재 성능 데이터를 이 과거 데이터와 비교하면 일반적인 성능 패턴과 성능 이상을 식별하고 이를 해결할 방법을 고안할 수 있습니다.

기준선을 설정하려면 최소한 다음 항목을 모니터링해야 합니다.

- I/O 및 스토리지를 읽고 쓸 수 있으므로 청구 목적으로 원장의 소비 패턴을 추적할 수 있습니다.
- 명령 지연 시간, 데이터 작업을 실행할 때 원장의 성능을 추적할 수 있습니다.
- 예외, 오류가 발생한 요청이 있는지 확인할 수 있습니다.

주제

모니터링 도구

AWS Amazon QLDB를 모니터링하는 데 사용할 수 있는 다양한 도구를 제공합니다. 이들 도구 중에는 모니터링을 자동으로 수행하도록 구성할 수 있는 도구도 있지만, 수동 작업이 필요한 도구도 있습니다. 모니터링 작업은 최대한 자동화하는 것이 좋습니다.

주제

자동 모니터링 도구

다음과 같은 자동 모니터링 도구를 사용하여 를 관찰하고 문제 발생 시 보고할 수 있습니다.

- Amazon CloudWatch Alarms — 지정한 기간 동안 단일 지표를 관찰하고 일정 기간 동안 지정된 임계값을 기준으로 지표의 값을 기준으로 하나 이상의 작업을 수행합니다. 작업은 아마존 심플 알림 서비스 (Amazon SNS) 주제 또는 Amazon EC2 Auto Scaling 정책으로 전송되는 알림입니다. CloudWatch 경보가 특정 상태에 있다는 이유만으로 경보가 작업을 호출하는 것은 아닙니다. 상태가 변경되고 지정된 기간 동안 유지되어야 합니다. 자세한 정보는 아마존을 통한 모니터링 CloudWatch을 참조하세요.
- Amazon CloudWatch Logs — AWS CloudTrail 또는 다른 소스에서 로그 파일을 모니터링, 저장 및 액세스합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 로그 파일 모니터링을 참조하십시오.

- Amazon CloudWatch Events — 이벤트를 매칭하고 하나 이상의 대상 함수 또는 스트림으로 라우팅하여 변경하고, 상태 정보를 캡처하고, 수정 조치를 취합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서의 Amazon CloudWatch Events란 무엇입니까? 를 참조하십시오.
- AWS CloudTrail 로그 모니터링 — 계정 간에 로그 파일을 공유하고, CloudTrail 로그 파일을 CloudWatch Logs로 전송하여 실시간으로 모니터링하고, Java로 로그 처리 애플리케이션을 작성하고, 전송 후 로그 파일이 변경되지 않았는지 확인합니다 CloudTrail. 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 로그 파일 작업을 참조하십시오.

수동 모니터링 도구

QLDB 모니터링의 또 다른 중요한 부분은 경보에서 다루지 않는 항목을 수동으로 모니터링하는 CloudWatch 것입니다. QLDB CloudWatch, Trusted Advisor, 및 AWS Management Console 기타 at-a-glance 대시보드는 환경 상태를 보여줍니다. AWS 또한 Amazon QLDB에서 로그 파일을 확인하는 것이 좋습니다.

- QLDB 대시보드는 다음을 보여줍니다.
  - I/O 읽기 및 쓰기
  - 저널 및 인덱싱된 스토리지
  - 명령 지연 시간
  - 예외
- CloudWatch 홈 페이지에는 다음이 표시됩니다.
  - 현재 경보 및 상태
  - 경보 및 리소스 그래프
  - 서비스 상태

  또한 를 CloudWatch 사용하여 다음 작업을 수행할 수 있습니다.
  - 맞춤 대시보드를 생성하여 관심 있는 서비스 모니터링
  - 지표 데이터를 그래프로 작성하여 문제를 해결하고 추세 파악
  - 모든 AWS 리소스 메트릭을 검색하고 찾아보십시오.
  - 문제에 대해 알려주는 경보 생성 및 편집

아마존을 통한 모니터링 CloudWatch

Amazon QLDB의 원시 데이터를 수집하여 읽을 수 있는 지표로 처리하는 CloudWatch 를 사용하여 Amazon QLDB를 모니터링할 수 있습니다. near-real-time 이러한 통계는 2주간 기록되므로 기록 정보에 액세스하고 웹 애플리케이션 또는 서비스가 어떻게 실행되고 있는지 전체적으로 더 잘 파악할 수 있습니다. 기본적으로 QLDB 메트릭 데이터는 1분 또는 15분 간격으로 자동 CloudWatch 전송됩니다. 자세한 내용은 Amazon, Amazon CloudWatch 이벤트 및 Amazon CloudWatch CloudWatch 로그란 무엇입니까? 를 참조하십시오. Amazon CloudWatch 사용 설명서에서 확인할 수 있습니다.

주제

- QLDB 지표는 어떻게 사용하나요?
- Amazon QLDB 지표 및 측정기준
- Amazon QLDB를 모니터링하기 위한 CloudWatch 경보 생성

QLDB 지표는 어떻게 사용하나요?

QLDB에서 보고하는 지표는 다양한 방법으로 분석이 기능한 정보를 제공합니다. 다음 목록은 몇 가지 일반적인 지표 사용 사례를 보여 줍니다. 모든 사용 사례를 망라한 것은 아니지만 시작하는 데 참고가 될 것입니다.

- 지정된 기간 동안 `JournalStorage` 및 `IndexedStorage`를 모니터링하여 원장이 사용하는 디스크 공간을 추적할 수 있습니다.
- 지정된 기간 동안 `ReadIOs` 및 `WriteIOs`를 모니터링하여 원장에서 처리하고 있는 요청 수를 추적할 수 있습니다.
- `CommandLatency`를 모니터링하여 원장의 데이터 작업 성능을 추적하고, 지연 시간이 가장 많이 발생하는 명령 유형을 분석할 수 있습니다.

Amazon QLDB 지표 및 측정기준

Amazon QLDB와 상호 작용하면 다음 지표와 차원이 로 전송됩니다. CloudWatch 스토리지 지표는 15분마다 보고되며, 다른 모든 지표는 1분마다 집계 및 보고됩니다. 다음 절차에 따라 QLDB에 대한 지표를 볼 수 있습니다.

콘솔을 사용하여 지표를 보려면 CloudWatch

지표는 먼저 서비스 네임스페이스별로 그룹화된 다음 각 네임스페이스 내에서 다양한 차원 조합별로 그룹화됩니다.

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 필요한 경우, 지역을 변경합니다. 탐색 모음에서 AWS 리소스가 상주하는 리전을 선택합니다. 자세한 내용은 리전 및 엔드포인트를 참조하세요.
3. 탐색 창에서 지표를 선택합니다.
4. 모든 지표 탭에서 QLDB를 선택합니다.

를 사용하여 지표를 보려면 AWS CLI

- 명령 프롬프트에서 다음 명령을 사용합니다.

```
aws cloudwatch list-metrics --namespace "AWS/QLDB"
```

CloudWatch QLDB에 대한 다음 메트릭을 표시합니다.

Amazon QLDB 차원 및 지표

Amazon QLDB가 Amazon에 전송하는 지표 및 측정기준은 다음과 같습니다 CloudWatch .

QLDB 지표

지표	설명
JournalStorage	원장 저널에서 사용한 디스크 공간의 총량으로, 15분 간격으로 보고됩니다. 저널에는 데이터에 대한 모든 변경 사항에 대한 완전하고 변경 불가능하며 확인 가능한 기록이 포함됩니다. 단위: Bytes 차원: LedgerName
IndexedStorage	원장의 테이블, 인덱스 및 인덱싱된 내역에서 사용한 디스크 공간의 총량으로, 15분 간격으로 보고됩니다. 인덱싱된 스토리지는 고성능 쿼리에 최적화된 원장 데이터로 구성됩니다. 단위: Bytes

지표	설명
	차원: LedgerName
ReadIOs	읽기 I/O 요청 수로, 1분 간격으로 보고됩니다. 데이터 트랜잭션, 확인 요청, 저널 내보내기, 저널 스트림 등 모든 유형의 읽기 작업을 수집합니다. 단위: Count 차원: LedgerName
WriteIOs	쓰기 I/O 요청 수로, 1분 간격으로 보고됩니다. 단위: Count 차원: LedgerName
CommandLatency	데이터 작업에 소요된 시간으로, 1분 간격으로 보고됩니다. 단위: Milliseconds 차원: CommandType, LedgerName
IsImpaired	Kinesis Data Streams에 대한 저널 스트림이 손상되었는지 여부를 나타내는 플래그로, 1분 간격으로 보고됩니다. 값이 1이면 스트림이 손상된 상태임을 나타내고 그렇지 않은 경우 0입니다. 단위: Boolean(0 또는 1) 차원: LedgerName, StreamId
OccConflictExceptions	OccConflictException 를 생성하는 QLDB에 대한 요청 수입니다. OCC(낙관적 동시성 제어)에 대한 자세한 내용은 Amazon QLDB 동시성 모델 섹션을 참조하세요. 단위: Count

지표	설명
Session4xxExceptions	HTTP 4xx 오류를 생성하는 QLDB에 대한 요청 수입니다. 단위: Count
Session5xxExceptions	HTTP 5xx 오류를 생성하는 QLDB에 대한 요청 수입니다. 단위: Count
SessionRateExceededExceptions	SessionRateExceededException 를 생성하는 QLDB에 대한 요청 수입니다. 단위: Count

QLDB 지표에 대한 차원

QLDB의 지표는 계정, 원장 이름, 스트림 ID 또는 명령 유형의 값으로 정규화됩니다. CloudWatch 콘솔을 사용하여 다음 표의 모든 차원을 따라 QLDB 데이터를 검색할 수 있습니다.

측정기준	설명
LedgerName	이 차원은 특정 원장에 대한 데이터를 제한합니다. 이 값은 AWS 리전 현재와 현재의 모든 원장 이름일 수 있습니다. AWS 계정
StreamId	이 차원은 특정 저널 스트림에 대한 데이터를 제한합니다. 이 값은 현재 AWS 리전 및 현재 원장의 모든 스트림 ID가 될 수 있습니다. AWS 계정
CommandType	이 차원은 다음의 QLDB 데이터 API 명령 중 하나에 대한 데이터를 제한합니다. • AbortTransaction • CommitTransaction • EndSession • ExecuteStatement • FetchPage • StartSession

측정기준	설명
	• `StartTransaction` QLDB가 이러한 명령을 사용하여 데이터 작업을 관리하는 방법을 알아보려면 드라이버를 사용한 세션 관리 섹션을 참조하세요.

Amazon QLDB를 모니터링하기 위한 CloudWatch 경보 생성

CloudWatch 알람 상태가 변경될 때 Amazon Simple Service (Amazon SNS) 메시지를 보내는 Amazon 경보를 생성할 수 있습니다. 경보는 지정한 기간 동안 단일 지표를 감시합니다. 기간 수에 대한 주어진 임계값과 지표 값을 비교하여 하나 이상의 작업을 수행합니다. 이 작업은 Amazon SNS 주제 또는 Auto Scaling 정책으로 전송되는 알림입니다.

경보는 지속적인 상태 변경에 대한 조치만 호출합니다. CloudWatch 경보가 특정 상태에 있다는 이유만으로 경보가 조치를 호출하지는 않습니다. 상태가 변경되어 지정된 기간 수 동안 유지되어야 합니다.

CloudWatch 경보 생성에 대한 자세한 내용은 Amazon 사용 CloudWatch 설명서의 Amazon CloudWatch 경보 사용을 참조하십시오.

이벤트를 이용한 Amazon QLDB 자동화 CloudWatch

Amazon CloudWatch Events를 사용하면 애플리케이션 가용성 문제 또는 리소스 AWS 서비스 변경과 같은 시스템 이벤트를 자동화하고 이에 자동으로 대응할 수 있습니다. 의 AWS 서비스 이벤트는 거의 실시간으로 CloudWatch Events로 전송됩니다. 원하는 이벤트만 표시하도록 간단한 규칙을 작성한 후 규칙과 일치하는 이벤트 발생 시 실행할 자동화 작업을 지정할 수 있습니다. 자동으로 트리거할 수 있는 태스크는 다음과 같습니다.

- 함수 호출 AWS Lambda
- Amazon EC2 Run Command 호출
- Amazon Kinesis Data Streams로 이벤트 릴레이
- 스테이트 머신 활성화 AWS Step Functions
- SNS 주제 또는 Amazon SQS 대기열 알림

Amazon QLDB는 원장 리소스의 상태가 변경될 때마다 이벤트에 CloudWatch 이벤트를 보고합니다. AWS 계정 이벤트는 현재 QLDB 원장 리소스에 대해서만 보장된 at-least-once 기반으로 생성됩니다.

다음은 QLDB가 보고한 이벤트의 예입니다.이 이벤트에서 원장 상태가 DELETING로 변경되었습니다.

```
{
    "version" : "0",
    "id" : "2f6557eb-e361-54ef-0f9f-99dd9f171c62",
    "detail-type" : "QLDB Ledger State Change",
    "source" : "aws.qldb",
    "account" : "123456789012",
    "time" : "2019-07-24T21:59:17Z",
    "region" : "us-east-1",
    "resources" : ["arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger"],
    "detail" : {
        "ledgerName" : "exampleLedger",
        "state" : "DELETING"
    }
}
```

QLDB와 함께 CloudWatch 이벤트를 사용하는 몇 가지 예에는 다음이 포함되지만 이에 국한되지는 않습니다.

- 새로운 원장이 CREATING 상태에서 처음 생성되고 최종적으로 ACTIVE 상태가 되면 Lambda 함수가 활성화합니다.
- 원장 상태가 DELETING로 변경되었다가 DELETED로 변경되면 Amazon SNS 주제에 알립니다.

자세한 내용은 Amazon CloudWatch Events 사용 설명서를 참조하십시오.

를 사용하여 Amazon QLDB API 호출을 로깅합니다. AWS CloudTrail

Amazon QLDB는 QLDB에서 사용자, 역할 또는 사용자가 수행한 작업의 기록을 제공하는 서비스와 AWS CloudTrail통합됩니다. AWS 서비스 CloudTrail QLDB에 대한 모든 리소스 관리 API 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 QLDB 콘솔에서 수행한 호출과 QLDB API 작업에 대한 코드 호출이 포함됩니다. 트레일을 생성하면 QLDB에 대한 CloudTrail 이벤트를 포함하여 Amazon Simple Storage Service (Amazon S3) 버킷으로 이벤트를 지속적으로 전송할 수 있습니다. 트레일을 구성하지 않아도 CloudTrail 콘솔의 이벤트 기록에서 가장 최근 이벤트를 계속 볼 수 있습니다. 에서 수집한 정보를 사용하여 QLDB에 이루어진 요청 CloudTrail, 요청이 이루어진 IP 주소, 요청한 사람, 요청 시기 및 추가 세부 정보를 확인할 수 있습니다.

구성 및 활성화 방법을 CloudTrail 포함하여 자세한 내용은 사용 AWS CloudTrail 설명서를 참조하십시오.

QLDB 정보는 다음과 같습니다. CloudTrail

CloudTrail 계정을 만들 AWS 계정 때 활성화됩니다. QLDB에서 지원되는 활동이 발생하면 해당 활동이 이벤트 기록의 CloudTrail AWS 서비스 다른 이벤트와 함께 이벤트에 기록됩니다. 에서 최근 이벤트를 보고, 검색하고, 다운로드할 수 있습니다. AWS 계정자세한 내용은 이벤트 기록으로 CloudTrail 이벤트 보기를 참조하십시오.

QLDB용 이벤트를 포함하여 내 이벤트의 진행 중인 기록을 보려면 트레일을 생성하십시오 AWS 계정. 트레일을 사용하면 CloudTrail Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 트레일은 AWS 파티션에 있는 모든 지역의 이벤트를 기록하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 이에 따라 조치를 AWS 서비스 취하도록 기타를 구성할 수 있습니다.

자세한 내용은AWS CloudTrail 사용 설명서에서 다음 주제를 참조하세요.

- 추적 생성 개요
- CloudTrail 지원되는 서비스 및 통합
- 에 대한 Amazon SNS 알림 구성 CloudTrail
- 여러 지역에서 CloudTrail 로그 파일 수신
- 여러 계정에서 CloudTrail 로그 파일 받기

모든 QLDB 리소스 관리 및 비트랜잭션 데이터 API 작업은 Amazon QLDB API 참조에 의해 CloudTrail 기록되고 문서화됩니다. 예를 들어,`DescribeLedger`, 및 작업에 대한 호출은 로그 `CreateLedger` 파일에 항목을 생성합니다. `DeleteLedger` CloudTrail

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 신원 정보를 이용하면 다음을 쉽게 알아볼 수 있습니다.

- 요청을 루트로 했는지 아니면 사용자 자격 증명으로 했는지 여부
- 역할 또는 연합된 사용자에 대한 임시 보안 인증을 사용하여 요청이 생성되었는지 여부
- 다른 사람이 요청한 것인지 여부 AWS 서비스

자세한 내용은 CloudTrail UserIdentity 요소를 참조하십시오.

QLDB 로그 파일 항목 이해

트레일은 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 전송할 수 있는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함되어 있습니다. 이벤트는 모든 소스의 단일 요청을 나타내며 요청된 작업, 작업 날짜 및 시간, 요청 매개 변수 등에 대한 정보를 포함합니다. CloudTrail 로그 파일은 공개 API 호출의 정렬된 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음 예제는 이러한 작업을 보여주는 CloudTrail 로그 항목을 보여줍니다.

- `CreateLedger`
- `DescribeLedger`
- `ListTagsForResource`
- `TagResource`
- `UntagResource`
- `ListLedgers`
- `GetDigest`
- `GetBlock`
- `GetRevision`
- `ExportJournalToS3`
- `DescribeJournalS3Export`
- `ListJournalS3ExportsForLedger`
- `ListJournalS3Exports`
- `DeleteLedger`

```
{
  "endTime": 1561497717208,
  "startTime": 1561497687254,
  "calls": [
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:27Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "CreateLedger",
```

```
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "Name": "CloudtrailTest",
          "PermissionsMode": "ALLOW_ALL"
        },
        "responseElements": {
          "CreationDateTime": 1.561497687403E9,
          "Arn": "arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest",
          "State": "CREATING",
          "Name": "CloudtrailTest"
        },
        "requestID": "3135aec7-978f-11e9-b313-1dd92a14919e",
        "eventID": "bf703ff9-676f-41dd-be6f-5f666c9f7852",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:27Z\",\"eventSource
\":\"qldb.amazonaws.com\",\"eventName\":\"CreateLedger\",\"awsRegion\":\"us-
east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-
sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08
 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":
{\"Name\":\"CloudtrailTest\",\"PermissionsMode\":\"ALLOW_ALL\"},\"responseElements
\":{\"CreationDateTime\":1.561497687403E9,\"Arn\":\"arn:aws:qldb:us-
east-2:123456789012:ledger/CloudtrailTest\",\"State\":\"CREATING\",\"Name\":
\"CloudtrailTest\"},\"requestID\":\"3135aec7-978f-11e9-b313-1dd92a14919e\",\"eventID\":
\"bf703ff9-676f-41dd-be6f-5f666c9f7852\",\"readOnly\":false,\"eventType\":\"AwsApiCall
\",\"recipientAccountId\":\"123456789012\"}",
      "name": "CreateLedger",
      "request": [
        "com.amazonaws.services.qldb.model.CreateLedgerRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest",
          "tags": null,
```

```
          "permissionsMode": "ALLOW_ALL"
        }
      ],
      "requestId": "3135aec7-978f-11e9-b313-1dd92a14919e"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:43Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "DescribeLedger",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "name": "CloudtrailTest"
        },
        "responseElements": null,
        "requestID": "3af51ba0-978f-11e9-8ae6-837dd17a19f8",
        "eventID": "be128e61-3e38-4503-83de-49fdc7fc0afb",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity
\":{\"type\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-
user\",\"arn\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",
\"accountId\":\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",
\"sessionContext\":{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate
\":\"2019-06-25T21:21:25Z\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":
\"AKIAIOSFODNN7EXAMPLE\",\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId
\":\"123456789012\",\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:43Z
\",\"eventSource\":\"qldb.amazonaws.com\",\"eventName\":\"DescribeLedger\",
\"awsRegion\":\"us-east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":
\"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-
Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation
\",\"requestParameters\":{\"name\":\"CloudtrailTest\"},\"responseElements
\":null,\"requestID\":\"3af51ba0-978f-11e9-8ae6-837dd17a19f8\",\"eventID\":
\"be128e61-3e38-4503-83de-49fdc7fc0afb\",\"readOnly\":true,\"eventType\":\"AwsApiCall
\",\"recipientAccountId\":\"123456789012\"}",
      "name": "DescribeLedger",
      "request": [
        "com.amazonaws.services.qldb.model.DescribeLedgerRequest",
```

```
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest"
        }
      ],
      "requestId": "3af51ba0-978f-11e9-8ae6-837dd17a19f8"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:44Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "TagResource",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "resourceArn": "arn%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger
%2FCloudtrailTest",
          "Tags": {
            "TagKey": "TagValue"
          }
        },
        "responseElements": null,
        "requestID": "3b1d6371-978f-11e9-916c-b7d64ec76521",
        "eventID": "6101c94a-7683-4431-812b-9a91afb8c849",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:44Z\",\"eventSource\":
\"qldb.amazonaws.com\",\"eventName\":\"TagResource\",\"awsRegion\":\"us-east-2\",
\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575
 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202
 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"resourceArn\":\"arn
```

%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger%2FCloudtrailTest\",\"Tags\":{\"TagKey\":\"TagValue\"}},\"responseElements\":null,\"requestID\":\"3b1d6371-978f-11e9-916c-b7d64ec76521\",\"eventID\":\"6101c94a-7683-4431-812b-9a91afb8c849\",\"readOnly\":false,\"eventType\":\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}",

```
      "name": "TagResource",
      "request": [
        "com.amazonaws.services.qldb.model.TagResourceRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "resourceArn": "arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest",
          "tags": {
            "TagKey": "TagValue"
          }
        }
      ],
      "requestId": "3b1d6371-978f-11e9-916c-b7d64ec76521"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:44Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "ListTagsForResource",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "resourceArn": "arn%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger
%2FCloudtrailTest"
        },
        "responseElements": null,
        "requestID": "3b56c321-978f-11e9-8527-2517d5bfa8fd",
        "eventID": "375e57d7-cf94-495a-9a48-ac2192181c02",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity
\":{\"type\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-
user\",\"arn\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",
\"accountId\":\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",
\"sessionContext\":{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate
```

\":\"2019-06-25T21:21:25Z\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId
\":\"AKIAIOSFODNN7EXAMPLE\",\"arn\":\"arn:aws:iam::123456789012:role/Admin
\",\"accountId\":\"123456789012\",\"userName\":\"Admin\"}}},\"eventTime\":
\"2019-06-25T21:21:44Z\",\"eventSource\":\"qldb.amazonaws.com\",\"eventName
\":\"ListTagsForResource\",\"awsRegion\":\"us-east-2\",\"sourceIPAddress\":
\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6
 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/
Oracle_Corporation\",\"requestParameters\":{\"resourceArn\":\"arn%3Aaws%3Aqldb
%3Aus-east-2%3A123456789012%3Aledger%2FCloudtrailTest\"},\"responseElements\":null,
\"requestID\":\"3b56c321-978f-11e9-8527-2517d5bfa8fd\",\"eventID\":\"375e57d7-
cf94-495a-9a48-ac2192181c02\",\"readOnly\":true,\"eventType\":\"AwsApiCall\",
\"recipientAccountId\":\"123456789012\"}",
      "name": "ListTagsForResource",
      "request": [
        "com.amazonaws.services.qldb.model.ListTagsForResourceRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "resourceArn": "arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest"
        }
      ],
      "requestId": "3b56c321-978f-11e9-8527-2517d5bfa8fd"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:44Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "UntagResource",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "tagKeys": "TagKey",
          "resourceArn": "arn%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger
%2FCloudtrailTest"
        },
        "responseElements": null,
        "requestID": "3b87e59b-978f-11e9-8b9a-bb6dc3a800a9",
        "eventID": "bcdcdca3-699f-4363-b092-88242780406f",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"

```
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:44Z\",\"eventSource\":
\"qldb.amazonaws.com\",\"eventName\":\"UntagResource\",\"awsRegion\":\"us-east-2\",
\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575
 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202
 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"tagKeys\":
\"TagKey\",\"resourceArn\":\"arn%3Aaws%3Aqldb%3Aus-east-2%3A123456789012%3Aledger
%2FCloudtrailTest\"},\"responseElements\":null,\"requestID\":\"3b87e59b-978f-11e9-8b9a-
bb6dc3a800a9\",\"eventID\":\"bcdcdca3-699f-4363-b092-88242780406f\",\"readOnly\":false,
\"eventType\":\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}",
      "name": "UntagResource",
      "request": [
        "com.amazonaws.services.qldb.model.UntagResourceRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "resourceArn": "arn:aws:qldb:us-east-2:123456789012:ledger/CloudtrailTest",
          "tagKeys": [
            "TagKey"
          ]
        }
      ],
      "requestId": "3b87e59b-978f-11e9-8b9a-bb6dc3a800a9"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:44Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "ListLedgers",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": null,
        "responseElements": null,
        "requestID": "3bafb877-978f-11e9-a6de-dbe6464b9dec",
```

```
        "eventID": "6ebe7d49-af59-4f29-aaa2-beffe536e20c",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:44Z\",\"eventSource
\":\"qldb.amazonaws.com\",\"eventName\":\"ListLedgers\",\"awsRegion\":\"us-
east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-
sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08
 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":null,
\"responseElements\":null,\"requestID\":\"3bafb877-978f-11e9-a6de-dbe6464b9dec\",
\"eventID\":\"6ebe7d49-af59-4f29-aaa2-beffe536e20c\",\"readOnly\":true,\"eventType\":
\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}",
      "name": "ListLedgers",
      "request": [
        "com.amazonaws.services.qldb.model.ListLedgersRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "maxResults": null,
          "nextToken": null
        }
      ],
      "requestId": "3bafb877-978f-11e9-a6de-dbe6464b9dec"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:49Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "GetDigest",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "name": "CloudtrailTest"
```

```
        },
        "responseElements": null,
        "requestID": "3cddd8a1-978f-11e9-a6de-dbe6464b9dec",
        "eventID": "a5cb60db-e6c5-4f5e-a5fc-0712249622b3",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:49Z\",\"eventSource\":
\"qldb.amazonaws.com\",\"eventName\":\"GetDigest\",\"awsRegion\":\"us-east-2\",
\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575
 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202
 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"name\":
\"CloudtrailTest\"},\"responseElements\":null,\"requestID\":\"3cddd8a1-978f-11e9-a6de-
dbe6464b9dec\",\"eventID\":\"a5cb60db-e6c5-4f5e-a5fc-0712249622b3\",\"readOnly\":true,
\"eventType\":\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}",
      "name": "GetDigest",
      "request": [
        "com.amazonaws.services.qldb.model.GetDigestRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest",
          "digestTipAddress": null
        }
      ],
      "requestId": "3cddd8a1-978f-11e9-a6de-dbe6464b9dec"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:50Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "GetBlock",
        "awsRegion": "us-east-2",
```

```
        "errorCode": null,
        "requestParameters": {
          "BlockAddress": {
            "IonText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:0}"
          },
          "name": "CloudtrailTest",
          "DigestTipAddress": {
            "IonText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:0}"
          }
        },
        "responseElements": null,
        "requestID": "3eaea09f-978f-11e9-bdc2-c1e55368155e",
        "eventID": "1f7da83f-d829-4e35-953d-30b925ceee66",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:50Z\",\"eventSource
\":\"qldb.amazonaws.com\",\"eventName\":\"GetBlock\",\"awsRegion\":\"us-east-2\",
\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575
 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202
 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"BlockAddress
\":{\"IonText\":\"{strandId:\\\"2P2nsG3K2RwHQccUbnAMAj\\\",sequenceNo:0}\"},
\"name\":\"CloudtrailTest\",\"DigestTipAddress\":{\"IonText\":\"{strandId:
\\\"2P2nsG3K2RwHQccUbnAMAj\\\",sequenceNo:0}\"}},\"responseElements\":null,
\"requestID\":\"3eaea09f-978f-11e9-bdc2-c1e55368155e\",\"eventID\":\"1f7da83f-
d829-4e35-953d-30b925ceee66\",\"readOnly\":true,\"eventType\":\"AwsApiCall\",
\"recipientAccountId\":\"123456789012\"}",
      "name": "GetBlock",
      "request": [
        "com.amazonaws.services.qldb.model.GetBlockRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest",
          "blockAddress": {
            "ionText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:0}"
```

```
          },
          "digestTipAddress": {
            "ionText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:0}"
          }
        }
      ],
      "requestId": "3eaea09f-978f-11e9-bdc2-c1e55368155e"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:55Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "GetRevision",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "BlockAddress": {
            "IonText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:1}"
          },
          "name": "CloudtrailTest",
          "DocumentId": "8UyXvDw6ApoFfVOA2HPfUE",
          "DigestTipAddress": {
            "IonText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:1}"
          }
        },
        "responseElements": null,
        "requestID": "41e19139-978f-11e9-aaed-dfe1dafe37ab",
        "eventID": "43bf2661-5046-41ec-a1d3-87706954aa10",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:55Z\",\"eventSource\":
\"qldb.amazonaws.com\",\"eventName\":\"GetRevision\",\"awsRegion\":\"us-east-2\",
```

\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575
 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202
 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"BlockAddress
\":{\"IonText\":\"{strandId:\\\"2P2nsG3K2RwHQccUbnAMAj\\\",sequenceNo:1}\"},\"name
\":\"CloudtrailTest\",\"DocumentId\":\"8UyXvDw6ApoFfVOA2HPfUE\",\"DigestTipAddress
\":{\"IonText\":\"{strandId:\\\"2P2nsG3K2RwHQccUbnAMAj\\\",sequenceNo:1}\"}},
\"responseElements\":null,\"requestID\":\"41e19139-978f-11e9-aaed-dfe1dafe37ab\",
\"eventID\":\"43bf2661-5046-41ec-a1d3-87706954aa10\",\"readOnly\":true,\"eventType\":
\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}",


```
      "name": "GetRevision",
      "request": [
        "com.amazonaws.services.qldb.model.GetRevisionRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest",
          "blockAddress": {
            "ionText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:1}"
          },
          "documentId": "8UyXvDw6ApoFfVOA2HPfUE",
          "digestTipAddress": {
            "ionText": "{strandId:\"2P2nsG3K2RwHQccUbnAMAj\",sequenceNo:1}"
          }
        }
      ],
      "requestId": "41e19139-978f-11e9-aaed-dfe1dafe37ab"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:56Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "ExportJournalToS3",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "InclusiveStartTime": 1.561497687254E9,
          "name": "CloudtrailTest",
          "S3ExportConfiguration": {
            "Bucket": "cloudtrailtests-123456789012-us-east-2",
            "Prefix": "CloudtrailTestsJournalExport",
            "EncryptionConfiguration": {
```

```
              "ObjectEncryptionType": "SSE_S3"
            }
          },
          "ExclusiveEndTime": 1.561497715795E9
        },
        "responseElements": {
          "ExportId": "BabQhsmJRYDCGMnA2xYBDG"
        },
        "requestID": "423815f8-978f-11e9-afcf-55f7d0f3583d",
        "eventID": "1b5abdc4-52fa-435f-857e-8995ef7a19b7",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:56Z\",\"eventSource\":
\"qldb.amazonaws.com\",\"eventName\":\"ExportJournalToS3\",\"awsRegion\":\"us-east-2\",
\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-sdk-java/1.11.575
 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08 java/1.8.0_202
 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"InclusiveStartTime
\":1.561497687254E9,\"name\":\"CloudtrailTest\",\"S3ExportConfiguration\":{\"Bucket\":
\"cloudtrailtests-123456789012-us-east-2\",\"Prefix\":\"CloudtrailTestsJournalExport\",
\"EncryptionConfiguration\":{\"ObjectEncryptionType\":\"SSE_S3\"}},\"ExclusiveEndTime
\":1.561497715795E9},\"responseElements\":{\"ExportId\":\"BabQhsmJRYDCGMnA2xYBDG
\"},\"requestID\":\"423815f8-978f-11e9-afcf-55f7d0f3583d\",\"eventID\":
\"1b5abdc4-52fa-435f-857e-8995ef7a19b7\",\"readOnly\":false,\"eventType\":\"AwsApiCall
\",\"recipientAccountId\":\"123456789012\"}",
      "name": "ExportJournalToS3",
      "request": [
        "com.amazonaws.services.qldb.model.ExportJournalToS3Request",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest",
          "inclusiveStartTime": 1561497687254,
          "exclusiveEndTime": 1561497715795,
          "s3ExportConfiguration": {
            "bucket": "cloudtrailtests-123456789012-us-east-2",
```

```
            "prefix": "CloudtrailTestsJournalExport",
            "encryptionConfiguration": {
              "objectEncryptionType": "SSE_S3",
              "kmsKeyArn": null
            }
          }
        }
      ],
      "requestId": "423815f8-978f-11e9-afcf-55f7d0f3583d"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:56Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "DescribeJournalS3Export",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "name": "CloudtrailTest",
          "exportId": "BabQhsmJRYDCGMnA2xYBDG"
        },
        "responseElements": null,
        "requestID": "427ebbbc-978f-11e9-8888-e9894c9c4bb9",
        "eventID": "ca8ffc88-16ff-45f5-9042-d94fadb389c3",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:56Z\",\"eventSource\":
\"qldb.amazonaws.com\",\"eventName\":\"DescribeJournalS3Export\",\"awsRegion\":
\"us-east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3
 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08
 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":{\"name
\":\"CloudtrailTest\",\"exportId\":\"BabQhsmJRYDCGMnA2xYBDG\"},\"responseElements
```

\":null,\"requestID\":\"427ebbbc-978f-11e9-8888-e9894c9c4bb9\",\"eventID\":
\"ca8ffc88-16ff-45f5-9042-d94fadb389c3\",\"readOnly\":true,\"eventType\":\"AwsApiCall
\",\"recipientAccountId\":\"123456789012\"}",

```
      "name": "DescribeJournalS3Export",
      "request": [
        "com.amazonaws.services.qldb.model.DescribeJournalS3ExportRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest",
          "exportId": "BabQhsmJRYDCGMnA2xYBDG"
        }
      ],
      "requestId": "427ebbbc-978f-11e9-8888-e9894c9c4bb9"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:56Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "ListJournalS3ExportsForLedger",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "name": "CloudtrailTest"
        },
        "responseElements": null,
        "requestID": "429ca40c-978f-11e9-8c4b-d13a8018a286",
        "eventID": "34f0e76b-58a5-45be-881c-786d22e34e96",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:56Z\",\"eventSource
\":\"qldb.amazonaws.com\",\"eventName\":\"ListJournalS3ExportsForLedger\",
```

\"awsRegion\":\"us-east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":
\"aws-internal/3 aws-sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-
Bit_Server_VM/25.202-b08 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation
\",\"requestParameters\":{\"name\":\"CloudtrailTest\"},\"responseElements
\":null,\"requestID\":\"429ca40c-978f-11e9-8c4b-d13a8018a286\",\"eventID\":
\"34f0e76b-58a5-45be-881c-786d22e34e96\",\"readOnly\":true,\"eventType\":\"AwsApiCall
\",\"recipientAccountId\":\"123456789012\"}",
      "name": "ListJournalS3ExportsForLedger",
      "request": [
        "com.amazonaws.services.qldb.model.ListJournalS3ExportsForLedgerRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest",
          "maxResults": null,
          "nextToken": null
        }
      ],
      "requestId": "429ca40c-978f-11e9-8c4b-d13a8018a286"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:56Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "ListJournalS3Exports",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": null,
        "responseElements": null,
        "requestID": "42cc1814-978f-11e9-befb-f5dbaa142118",
        "eventID": "4c24d7d6-810c-4cf4-884e-00482278b6ce",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",

\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:56Z\",\"eventSource\":
\"qldb.amazonaws.com\",\"eventName\":\"ListJournalS3Exports\",\"awsRegion\":\"us-
east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-
sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08
 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters\":null,
\"responseElements\":null,\"requestID\":\"42cc1814-978f-11e9-befb-f5dbaa142118\",
\"eventID\":\"4c24d7d6-810c-4cf4-884e-00482278b6ce\",\"readOnly\":true,\"eventType\":
\"AwsApiCall\",\"recipientAccountId\":\"123456789012\"}",
      "name": "ListJournalS3Exports",
      "request": [
        "com.amazonaws.services.qldb.model.ListJournalS3ExportsRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "maxResults": null,
          "nextToken": null
        }
      ],
      "requestId": "42cc1814-978f-11e9-befb-f5dbaa142118"
    },
    {
      "cloudtrailEvent": {
        "userIdentity": {
          "arn": "arn:aws:sts::123456789012:assumed-role/Admin/test-user"
        },
        "eventTime": "2019-06-25T21:21:57Z",
        "eventSource": "qldb.amazonaws.com",
        "eventName": "DeleteLedger",
        "awsRegion": "us-east-2",
        "errorCode": null,
        "requestParameters": {
          "name": "CloudtrailTest"
        },
        "responseElements": null,
        "requestID": "42f439b9-978f-11e9-8b2c-69ef598d66e9",
        "eventID": "429f5163-cba5-4d86-bd7e-f606e057c6cf",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
      },
      "rawCloudtrailEvent": "{\"eventVersion\":\"1.05\",\"userIdentity\":{\"type
\":\"AssumedRole\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE:test-user\",\"arn
\":\"arn:aws:sts::123456789012:assumed-role/Admin/test-user\",\"accountId\":

```
\"123456789012\",\"accessKeyId\":\"AKIAI44QH8DHBEXAMPLE\",\"sessionContext\":
{\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2019-06-25T21:21:25Z
\"},\"sessionIssuer\":{\"type\":\"Role\",\"principalId\":\"AKIAIOSFODNN7EXAMPLE\",
\"arn\":\"arn:aws:iam::123456789012:role/Admin\",\"accountId\":\"123456789012\",
\"userName\":\"Admin\"}}},\"eventTime\":\"2019-06-25T21:21:57Z\",\"eventSource
\":\"qldb.amazonaws.com\",\"eventName\":\"DeleteLedger\",\"awsRegion\":\"us-
east-2\",\"sourceIPAddress\":\"192.0.2.01\",\"userAgent\":\"aws-internal/3 aws-
sdk-java/1.11.575 Mac_OS_X/10.13.6 Java_HotSpot(TM)_64-Bit_Server_VM/25.202-b08
 java/1.8.0_202 kotlin/1.3.21 vendor/Oracle_Corporation\",\"requestParameters
\":{\"name\":\"CloudtrailTest\"},\"responseElements\":null,\"requestID\":
\"42f439b9-978f-11e9-8b2c-69ef598d66e9\",\"eventID\":\"429f5163-cba5-4d86-bd7e-
f606e057c6cf\",\"readOnly\":false,\"eventType\":\"AwsApiCall\",\"recipientAccountId\":
\"123456789012\"}",
      "name": "DeleteLedger",
      "request": [
        "com.amazonaws.services.qldb.model.DeleteLedgerRequest",
        {
          "customRequestHeaders": null,
          "customQueryParameters": null,
          "name": "CloudtrailTest"
        }
      ],
      "requestId": "42f439b9-978f-11e9-8b2c-69ef598d66e9"
    }
  ]
}
```


Amazon QLDB에 대한 규정 준수 확인

타사 감사자는 다음을 포함하되 이에 국한되지 않는 AWS 여러 규정 준수 프로그램의 일환으로 Amazon QLDB의 보안 및 규정 준수를 평가합니다.

- SOC(시스템 및 조직 제어)
- 신용카드 업계(PCI)
- 국제표준화기구(ISO)
- 정보 시스템 보안 관리 및 평가 프로그램(ISMAP)
- HIPAA(미국 건강 보험 양도 및 책임에 관한 법)

> **Note**
>
> 다음은 Amazon QLDB 인증의 전체 목록이 아닙니다.

특정 규정 준수 프로그램의 범위 내에 AWS 서비스 있는지 알아보려면AWS 서비스 규정 준수 프로그램 범위 내 규정 준수AWS 서비스 프로그램별 규정 관심 있는 규정 준수 프로그램을 선택하십시오. 일반 정보는 AWS 규정 준수 프로그램AWS 보증 프로그램 규정AWS 참조하십시오.

를 사용하여 AWS Artifact타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 의 보고서 https://docs.aws.amazon.com/artifact/latest/ug/downloading-documents.html 참조하십시오 AWS Artifact.

사용 시 규정 준수 AWS 서비스 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 결정됩니다. AWS 규정 준수에 도움이 되는 다음 리소스를 제공합니다.

- 보안 및 규정 준수 퀵 스타트 가이드 - 이 배포 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수에 AWS 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다.
- Amazon Web Services의 HIPAA 보안 및 규정 준수를 위한 설계 — 이 백서에서는 기업이 HIPAA 적격 애플리케이션을 만드는 AWS 데 사용할 수 있는 방법을 설명합니다.

  > **Note**
  >
  > 모든 AWS 서비스 사람이 HIPAA 자격을 갖춘 것은 아닙니다. 자세한 내용은 HIPAA 적격 서비스 참조를 참조하십시오.

- AWS 규정 준수 리소스AWS — 이 워크북 및 가이드 모음은 해당 산업 및 지역에 적용될 수 있습니다.
- AWS 고객 규정 준수 가이드 — 규정 준수의 관점에서 공동 책임 모델을 이해하십시오. 이 가이드에서는 보안을 유지하기 위한 모범 사례를 AWS 서비스 요약하고 여러 프레임워크 (미국 표준 기술 연구소 (NIST), 결제 카드 산업 보안 표준 위원회 (PCI), 국제 표준화기구 (ISO) 등) 에서 보안 제어에 대한 지침을 매핑합니다.
- AWS Config 개발자 안내서의 규칙을 사용하여 리소스 평가 — 이 AWS Config 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- AWS Security Hub— 이를 AWS 서비스 통해 내부 AWS보안 상태를 포괄적으로 파악할 수 있습니다. Security Hub는 보안 제어를 사용하여 AWS 리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 Security Hub 제어 참조를 참조하십시오.

- Amazon GuardDuty — 환경에 의심스럽고 악의적인 활동이 있는지 AWS 계정모니터링하여 워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 AWS 서비스 탐지합니다. GuardDuty 특정 규정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족하여 PCI DSS와 같은 다양한 규정 준수 요구 사항을 해결하는 데 도움이 될 수 있습니다.
- AWS Audit Manager— 이를 AWS 서비스 통해 AWS 사용량을 지속적으로 감사하여 위험을 관리하고 규정 및 업계 표준을 준수하는 방법을 단순화할 수 있습니다.

Amazon QLDB의 복원성

AWS 글로벌 인프라는 가용 영역을 중심으로 구축됩니다. AWS 리전 AWS 리전 물리적으로 분리되고 격리된 여러 가용 영역을 제공합니다. 이 가용 영역은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹으로 연결됩니다. 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.

가용 영역에 대한 AWS 리전 자세한 내용은 글로벌 인프라를 참조하십시오AWS .

메시지 내구성

QLDB 저널 스토리지는 트랜잭션 커밋 시 여러 가용 영역으로의 동기 복제 기능을 제공합니다. 따라서 저널 스토리지 전체 가용 영역 장애가 발생하더라도 데이터 무결성이나 활성 서비스를 유지 관리하는 기능이 손상되지 않습니다. 또한 QLDB 저널에는 내결함성 스토리지에 대한 비동기식 아카이브가 있습니다. 이 기능은 여러 가용 영역에서 동시에 스토리지 장애가 발생할 가능성이 매우 낮은 경우에 재해 복구를 지원합니다.

QLDB 인덱싱된 스토리지는 여러 가용 영역으로의 복제를 통해 지원됩니다. 따라서 인덱싱된 스토리지에 전체 가용 영역 장애가 발생하더라도 데이터 무결성이나 활성 서비스를 유지 관리하는 기능이 손상되지 않습니다.

데이터 내구성 기능

QLDB는 AWS 글로벌 인프라 외에도 데이터 복원력 및 백업 요구 사항을 지원하는 데 도움이 되는 다음과 같은 기능을 제공합니다.

QLDB 서비스 기능

온디맨드 저널 내보내기

QLDB는 온디맨드 저널 내보내기 기능을 제공합니다. 원장의 저널 블록을 Amazon S3 버킷으로 내보내 저널의 콘텐츠에 액세스합니다. 이 데이터는 데이터 보존, 분석 및 감사와 같은 다양한 목적으로 사용할 수 있습니다. 자세한 내용은 Amazon QLDB에서 저널 데이터 내보내기을 참조하십시오.

백업 및 복원

내보내기에 대한 자동 복원은 현재 지원되지 않습니다. 내보내기는 정의된 빈도로 추가 데이터 중복성을 생성할 수 있는 기본 기능을 제공합니다. 그러나 복원 시나리오는 애플리케이션에 따라 다르며, 내보낸 레코드는 동일하거나 유사한 수집 방법을 활용하여 새 원장에 다시 쓰는 것으로 추정됩니다.

QLDB는 현재 전용 백업 및 관련 복원 기능을 제공하지 않습니다.

저널 스트림

QLDB는 연속 저널 스트림 기능도 제공합니다. QLDB 저널 스트림을 Amazon Kinesis 스트리밍 플랫폼과 통합하여 실시간 저널 데이터를 처리할 수 있습니다. 자세한 내용은 Amazon QLDB에서 저널 데이터 스트리밍을 참조하십시오.

QLDB 설계 기능

QLDB는 논리적 손상에 대한 복원력을 갖도록 설계되었습니다. QLDB 저널은 변경할 수 없으므로 커밋된 모든 트랜잭션이 저널에 유지됩니다. 또한 모든 커밋된 문서 변경 사항이 기록되므로 원장 데이터의 의도하지 않은 변경 사항을 point-in-time파악할 수 있습니다.

QLDB는 현재 논리적 손상 시나리오에 대한 자동 복구 기능을 제공하지 않습니다.

Amazon MQ의 인프라 보안

관리형 서비스인 Amazon QLDB는 Amazon Web Services: 보안 프로세스 개요 백서에 설명된 글로벌 네트워크 보안 절차에 따라 보호됩니다. AWS

AWS 게시된 API 호출을 사용하여 네트워크를 통해 QLDB에 액세스합니다. 클라이언트가 전송 계층 보안(TLS) 1.0 이상을 지원해야 합니다. TLS 1.2 이상을 권장합니다. 클라이언트는 Ephemeral Diffie-Hellman(DHE) 또는 Elliptic Curve Ephemeral Diffie-Hellman(ECDHE)과 같은 PFS(전달 완전 보안,

Perfect Forward Secrecy)가 포함된 암호 제품군도 지원해야 합니다. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 IAM 보안 주체와 관련된 프로그래밍 보안 인증을 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 보안 인증을 생성하여 요청에 서명할 수 있습니다.

QLDB에 Virtual Private Cloud(VPC) 엔드포인트를 사용할 수도 있습니다. 인터페이스 VPC 엔드포인트를 사용하면 Amazon VPC 리소스가 프라이빗 IP 주소를 사용하여 퍼블릭 인터넷에 노출되지 않고 QLDB에 액세스할 수 있습니다. 자세한 내용은 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 사용하여 Amazon QLDB에 액세스을 참조하십시오.

인터페이스 VPC 엔드포인트(AWS PrivateLink)를 사용하여 Amazon QLDB에 액세스

를 AWS PrivateLink 사용하여 VPC와 Amazon QLDB 간에 프라이빗 연결을 생성할 수 있습니다. 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 연결을 사용하지 않고도 VPC에 있는 것처럼 QLDB에 액세스할 수 있습니다. AWS Direct Connect VPC의 인스턴스에서 QLDB에 액세스하는 데 퍼블릭 IP 주소가 필요하지 않습니다.

AWS PrivateLink에서 제공되는 인터페이스 엔드포인트를 생성하여 이 프라이빗 연결을 설정합니다. 인터페이스 엔드포인트에 대해 사용 설정하는 각 서브넷에서 엔드포인트 네트워크 인터페이스를 생성합니다. 이는 QLDB로 향하는 트래픽의 진입점 역할을 하는 요청자 관리형 네트워크 인터페이스입니다.

자세한 내용은AWS PrivateLink 가이드의 AWS PrivateLink를 통해 AWS 서비스 에 액세스를 참조하세요.

주제

- QLDB에 대한 고려 사항
- QLDB에 대한 인터페이스 엔드포인트 생성
- 인터페이스 엔드포인트에 엔드포인트 정책 생성
- QLDB에 대한 인터페이스 엔드포인트 가용성

QLDB에 대한 고려 사항

Amazon EKS에 대한 인터페이스 엔드포인트를 설정하려면 먼저 AWS PrivateLink 가이드의 고려 사항을 검토합니다.

> **Note**
>
> QLDB는 인터페이스 엔드포인트를 통한 QLDB 세션 트랜잭션 데이터 API 호출만 지원합니다. 이 API에는 작업만 포함됩니다. SendCommand 원장의 STANDARD 권한 모드에서는 이 API의 특정 PartiQL 작업에 대한 권한을 제어할 수 있습니다.

QLDB에 대한 인터페이스 엔드포인트 생성

Amazon VPC 콘솔 또는 () 를 사용하여 QLDB용 인터페이스 엔드포인트를 생성할 수 있습니다. AWS Command Line Interface AWS CLI자세한 내용은AWS PrivateLink 가이드의 인터페이스 엔드포인트 생성을 참조하세요.

다음과 같은 서비스 이름을 사용하여 QLDB의 인터페이스 엔드포인트를 생성합니다.

```
com.amazonaws.region.qldb.session
```

인터페이스 엔드포인트에 프라이빗 DNS를 사용하도록 설정하는 경우, 기본 리전 DNS 이름을 사용하여 QLDB에 API 요청을 할 수 있습니다. 예를 들어 `session.qldb.us-east-1.amazonaws.com`입니다.

인터페이스 엔드포인트에 엔드포인트 정책 생성

엔드포인트 정책은 인터페이스 엔드포인트에 연결할 수 있는 IAM 리소스입니다. 기본 엔드포인트 정책을 사용하면 인터페이스 엔드포인트를 통해 QLDB에 대한 전체 액세스를 허용합니다. VPC에서 QLDB에 허용되는 액세스를 제어하려면 사용자 지정 엔드포인트 정책을 인터페이스 엔드포인트에 연결합니다.

엔드포인트 정책은 다음 정보를 지정합니다.

- 작업을 수행할 수 있는 보안 주체(AWS 계정, 사용자, 역할).
- 수행할 수 있는 작업.
- 작업을 수행할 수 있는 리소스.

자세한 내용은AWS PrivateLink 가이드의 엔드포인트 정책을 사용하여 서비스에 대한 액세스 제어를 참조하세요.

또한 사용자, 그룹 또는 역할에 연결된 정책의 Condition 필드를 사용하여 지정된 인터페이스 엔드포인트에서만 액세스를 허용할 수 있습니다. 엔드포인트 정책과 IAM 정책을 함께 사용하면 지정된 원장의 특정 QLDB 작업에 대한 액세스를 지정된 인터페이스 엔드포인트로 제한할 수 있습니다.

엔드포인트 정책 예: 특정 QLDB 원장에 대한 액세스 제한

다음은 QLDB에 대한 사용자 지정 엔드포인트 정책의 예제입니다. 이 정책을 인터페이스 엔드포인트에 연결하면 지정된 원장 리소스의 모든 보안 주체에 대한 SendCommand 작업 및 PartiQL 읽기 전용 작업에 대한 액세스 권한이 부여됩니다. 이 예시에서는 원장이 STANDARD 권한 모드에 있어야 합니다.

이 정책을 사용하려면 예제에서 *us-east-1,* 123456789012를 사용자 고유의 정보로 바꾸십시오. *myExampleLedger*

```
{
   "Statement": [
      {
         "Sid": "QLDBSendCommandPermission",
         "Principal": "*",
         "Effect": "Allow",
         "Action": "qldb:SendCommand",
         "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger"
      },
      {
         "Sid": "QLDBPartiQLReadOnlyPermissions",
         "Principal": "*",
         "Effect": "Allow",
         "Action": [
            "qldb:PartiQLSelect",
            "qldb:PartiQLHistoryFunction"
         ],
         "Resource": [
            "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/table/*",
            "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger/
information_schema/user_tables"
         ]
      }
   ]
}
```

IAM 정책 예제: 특정 인터페이스 엔드포인트에서만 QLDB 원장에 대한 액세스만 제한

다음은 QLDB에 대한 IAM 자격 증명 기반 정책 예제입니다. 이 정책을 사용자, 역할 또는 그룹에 연결하면 지정된 인터페이스 엔드포인트에서만 원장 리소스에 대한 SendCommand 액세스 권한이 허용됩니다.

 # ### ##### ### us-east-1, 123456789012 # myExampleLedgervpce-1a2b3c4d# ### ### #####.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AccessFromSpecificInterfaceEndpoint",
         "Effect": "Deny",
         "Action": "qldb:SendCommand",
         "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/myExampleLedger",
         "Condition": {
            "StringNotEquals": {
               "aws:sourceVpce": "vpce-1a2b3c4d"
            }
         }
      }
   ]
}
```

QLDB에 대한 인터페이스 엔드포인트 가용성

Amazon QLDB는 QLDB를 사용할 수 있는 모든 AWS 리전 에서 정책으로 인터페이스 엔드포인트를 지원합니다. 사용 가능한 리전의 전체 목록은 AWS 일반 참조에서 Amazon QLDB 엔드포인트 및 할당량을 참조하십시오.

Amazon QLDB 문제 해결

다음 섹션은 Amazon QLDB를 사용할 때 발생할 수 있는 일반적인 오류의 전체 목록과 이러한 오류를 해결하는 방법에 대한 지침을 제공합니다.

IAM 액세스 관련 문제 해결 지침은 Amazon QLDB 자격 증명 및 액세스 문제 해결을 참조하십시오.

PartiQL 문 조정에 대한 모범 사례는 쿼리 성능 최적화을 참조하십시오.

주제

- QLDB 드라이버를 사용하여 트랜잭션 실행
- 저널 데이터 내보내기
- 저널 데이터 스트리밍
- 저널 데이터 확인

QLDB 드라이버를 사용하여 트랜잭션 실행

이 섹션에는 Amazon QLDB 드라이버를 사용하여 원장에서 PartiQL 트랜잭션을 실행할 때 반환할 수 있는 일반적인 예외가 나열되어 있습니다. 이 기능에 대한 자세한 내용은 드라이버 시작하기을 참조하십시오. 드라이버 구성 및 사용에 대한 모범 사례는 드라이버 권장 사항을 참조하십시오.

각 예외에는 특정 오류 메시지와 가능한 해결 방법에 대한 간단한 설명 및 제안이 포함됩니다.

CapacityExceededException

메시지: 용량 초과

Amazon QLDB는 원장의 처리 용량을 초과했기 때문에 요청을 거부했습니다. QLDB는 서비스의 상태와 성능을 유지하기 위해 원장당 내부 크기 조정 한도를 적용합니다. 이 한도는 각 개별 요청의 워크로드 크기에 따라 달라집니다. 예를 들어, 인덱스가 아닌 정규화된 쿼리로 인한 테이블 스캔과 같은 비효율적인 데이터 트랜잭션을 수행하는 요청의 경우 워크로드가 증가할 수 있습니다.

요청을 재시도하기 전에 잠시 기다릴 것을 권장합니다. 애플리케이션에서 이 예외가 지속적으로 발생하는 경우 문을 최적화하고 원장에 보내는 요청의 속도와 볼륨을 줄이십시오. 명령문 최적화의 예로는 트랜잭션당 더 적은 수의 명령문을 실행하고 테이블 인덱스를 조정하는 경우를 들 수 있습니다. 명령문을 최적화하고 테이블 스캔을 방지하는 방법을 알아보려면 쿼리 성능 최적화을 참조하십시오.

또한 최신 버전의 QLDB 드라이버를 사용할 것을 권장합니다. 드라이버에는 지수 백오프 및 Jitter를 사용하여 이와 같은 예외가 발생할 경우 자동으로 재시도하는 기본 재시도 정책이 있습니다. 지수 백오프의 개념은 오류 응답이 연속될 때마다 재시도 간 대기 시간을 점진적으로 늘린다는 것입니다.

InvalidSessionException

메시지: 트랜잭션 *transactionId*가 만료되었습니다

트랜잭션이 최대 수명을 초과했습니다. 트랜잭션은 커밋되기 전까지 최대 30초 동안 실행될 수 있습니다. 이 시간 초과 제한 이후에는 트랜잭션에 대해 수행된 모든 작업이 거부되고 QLDB는 세션을 삭제합니다. 이 제한은 트랜잭션을 시작하고 커밋하거나 취소하지 않기 때문에 클라이언트의 세션 유출을 방지합니다.

이것이 애플리케이션에서 일반적인 예외인 경우, 트랜잭션을 실행하는 데 시간이 너무 오래 걸릴 수 있습니다. 트랜잭션 런타임이 30초 이상 걸리는 경우 문을 최적화하여 트랜잭션 속도를 높이십시오. 명령문 최적화의 예로는 트랜잭션당 더 적은 수의 명령문을 실행하고 테이블 인덱스를 조정하는 경우를 들 수 있습니다. 자세한 내용은 쿼리 성능 최적화을 참조하십시오.

InvalidSessionException

메시지: 세션 *sessionId*가 만료되었습니다

최대 총 수명을 초과했기 때문에 QLDB가 세션을 삭제했습니다. QLDB는 활성 트랜잭션과 상관없이 13~17분 후에 세션을 삭제합니다. 세션은 하드웨어 장애, 네트워크 장애 또는 애플리케이션 재시작과 같은 여러 가지 이유로 손실되거나 손상될 수 있습니다. 따라서 QLDB는 클라이언트 소프트웨어가 세션 장애에 대해 복원력을 유지할 수 있도록 세션에 최대 수명을 적용합니다.

이 예외가 발생하는 경우 새 세션을 획득하고 트랜잭션을 다시 시도할 것을 권장합니다. 또한 애플리케이션을 대신하여 세션 풀과 해당 상태를 관리하는 최신 버전의 QLDB 드라이버를 사용할 것을 권장합니다.

InvalidSessionException

메시지: 해당 세션이 없습니다

클라이언트가 존재하지 않는 세션을 사용하여 QLDB와 트랜잭션을 시도했습니다. 클라이언트가 이전에 존재했던 세션을 사용하고 있다고 가정하면 다음 중 하나로 인해 세션이 더 이상 존재하지 않을 수 있습니다.

- 세션이 내부 서버 장애(즉, HTTP 응답 코드 500 오류)와 관련된 경우, QLDB는 고객이 불확실한 상태의 세션과 트랜잭션하도록 허용하는 대신 해당 세션을 완전히 삭제하도록 선택할 수 있습니다. 그러면 해당 세션에 대한 모든 재시도가 실패하고 이 오류가 발생합니다.

- QLDB는 만료된 세션을 결국 잊어버립니다. 그런 다음 세션을 계속 사용하려고 하면 초기 `InvalidSessionException`이 아닌 이 오류가 발생합니다.

이 예외가 발생하는 경우 새 세션을 획득하고 트랜잭션을 다시 시도할 것을 권장합니다. 또한 애플리케이션을 대신하여 세션 풀과 해당 상태를 관리하는 최신 버전의 QLDB 드라이버를 사용할 것을 권장합니다.

RateExceededException

메시지: 속도가 초과되었습니다

QLDB는 발신자의 ID를 기반으로 클라이언트를 제한했습니다. QLDB는 토큰 버킷 제한 알고리즘을 사용하여 리전별, 계정별로 제한을 적용합니다. QLDB는 서비스의 성과를 돕고 모든 QLDB 고객의 공정한 사용을 보장하기 위해 이를 수행합니다. 예를 들어, `StartSessionRequest` 작업을 사용하여 많은 수의 동시 세션을 획득하려고 하면 제한이 발생할 수 있습니다.

애플리케이션 상태를 유지하고 추가적인 제한을 완화하기 위해 지수 백오프 및 Jitter를 사용하여 이 예외를 다시 시도할 수 있습니다. 지수 백오프의 개념은 오류 응답이 연속될 때마다 재시도 간 대기 시간을 점진적으로 늘린다는 것입니다. 최신 버전의 QLDB 드라이버를 사용하는 것이 좋습니다. 드라이버에는 지수 백오프 및 Jitter를 사용하여 이와 같은 예외가 발생할 경우 자동으로 재시도하는 기본 재시도 정책이 있습니다.

최신 버전의 QLDB 드라이버는 애플리케이션이 `StartSessionRequest` 호출에 대해 QLDB에 의해 지속적으로 제한되는 경우에도 도움이 될 수 있습니다. 드라이버는 여러 트랜잭션에서 재사용되는 세션 풀을 유지 관리하므로 애플리케이션이 수행하는 `StartSessionRequest` 호출 수를 줄이는데 도움이 될 수 있습니다. API 호출 한도 증가를 요청하려면 AWS Support 센터로 문의하십시오.

LimitExceededException

메시지: 세션 제한을 초과했습니다

원장이 활성 세션 수의 할당량(한도라고도 함)을 초과했습니다. 이 할당량은 Amazon QLDB 할당량 및 제한에 정의되어 있습니다. 원장의 활성 세션 수는 최종적으로 일정하게 유지되며, 할당량 근처에서 지속적으로 실행되는 원장에서는 이 예외가 주기적으로 발생할 수 있습니다.

애플리케이션의 상태를 유지하려면 이 예외를 다시 시도할 것을 권장합니다. 이 예외를 피하려면 모든 클라이언트에서 단일 원장에 1,500개 이상의 동시 세션을 사용하도록 구성하지 않았는지 확인하십시오. 예를 들어, Java용 Amazon QLDB 드라이버의 MaxConcurrentTransactions 메서드를 사용하여 드라이버 인스턴스에서 사용 가능한 최대 세션 수를 구성할 수 있습니다.

QldbClientException

메시지: 스트리밍된 결과는 상위 트랜잭션이 열려 있을 때만 유효합니다

트랜잭션이 종료되었으며 QLDB에서 결과를 검색하는 데 사용할 수 없습니다. 트랜잭션은 커밋되거나 취소되면 종료됩니다.

이 예외는 클라이언트가 `Transaction` 객체로 직접 작업하고 트랜잭션을 커밋하거나 취소한 후 QLDB에서 결과를 검색하려고 할 때 발생합니다. 이 문제를 완화하려면 클라이언트가 트랜잭션을 닫기 전에 데이터를 읽어야 합니다.

저널 데이터 내보내기

이 섹션에는 원장에서 Amazon S3 버킷으로 저널 데이터를 내보낼 때 QLDB가 반환할 수 있는 일반적인 예외가 나와 있습니다. 이 기능에 대한 자세한 내용은 Amazon QLDB에서 저널 데이터 내보내기을 참조하십시오.

각 예외에는 특정 오류 메시지와 가능한 해결 방법에 대한 간단한 설명 및 제안이 포함됩니다.

AccessDeniedException

메시지: User: *userARN* is not authorized to perform: iam:PassRole on resource: *roleARN*

IAM 역할을 QLDB 서비스에 전달할 권한이 없습니다. QLDB에는 모든 저널 내보내기 요청에 대한 역할이 필요하며 이 역할을 QLDB에 전달할 권한이 있어야 합니다. 이 역할은 지정된 Amazon S3 버킷에 대한 쓰기 권한을 QLDB에 제공합니다.

QLDB 서비스(`qldb.amazonaws.com`)의 지정된 IAM 역할 리소스에서 `PassRole` API 작업을 수행할 권한을 부여하는 IAM 정책을 정의했는지 확인하세요. 정책 예제는 Amazon QLDB의 자격 증명 기반 정책 예을 참조하십시오.

IllegalArgumentException

메시지: QLDB encountered an error validating S3 configuration: *errorCodeerrorMessage*

이 오류의 가능한 원인은 제공된 Amazon S3 버킷이 Amazon S3에 존재하지 않기 때문입니다. 또는 QLDB에 지정된 Amazon S3 버킷에 객체를 쓸 수 있는 충분한 권한이 없습니다.

내보내기 작업 요청에 제공한 S3 버킷 이름이 정확한지 확인하세요. 버킷 이름 지정에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 버킷 규제 및 제한을 참조하세요.

또한 QLDB 서비스(`qldb.amazonaws.com`)에 대한 `PutObject` 및 `PutObjectAcl` 권한을 부여하는 지정된 버킷에 대한 정책을 정의했는지 확인합니다. 자세한 내용은 내보내기 권한 섹션을 참조하세요.

IllegalArgumentException

메시지: Unexpected response from Amazon S3 while validating the S3 configuration. S3로부터의 응답: *errorCode errorMessage*

제공된 Amazon S3 오류 응답으로 인해 제공된 S3 버킷에 저널 내보내기 데이터를 쓰려는 시도가 실패했습니다. 발생 가능한 원인에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 Amazon S3 문제 해결을 참조하십시오.

IllegalArgumentException

메시지: Amazon S3 버킷 접두사는 128자를 초과할 수 없습니다

저널 내보내기 요청에 제공된 접두사가 128자를 초과합니다.

IllegalArgumentException

메시지: 시작 날짜는 종료 날짜보다 이후일 수 없습니다.

`InclusiveStartTime` 및 `ExclusiveEndTime`는 ISO 8601 날짜 및 시간 형식이어야 하며 협정 세계시(UTC)여야 합니다.

IllegalArgumentException

메시지: 종료일은 미래일 수 없습니다

`InclusiveStartTime` 및 `ExclusiveEndTime`은 UTC 기준의 `ISO 8601` 날짜 및 시간 형식이어야 합니다.

IllegalArgumentException

메시지: The supplied object encryption setting (S3EncryptionConfiguration) is not compatible with an AWS Key Management Service (AWS KMS) key

`KMSKeyArn`에 `NO_ENCRYPTION` 또는 `SSE_S3`의 `ObjectEncryptionType`를 제공했습니다. `SSE_KMS`의 객체 암호화 유형에 고객 관리형 AWS KMS key만 제공할 수 있습니다. Amazon S3의 서버 측 암호화 옵션에 대한 자세한 내용은 Amazon S3 개발자 안내서의 서버 측 암호화를 사용한 데이터 보호를 참조하세요.

LimitExceededException

메시지: 동시 실행 저널 내보내기 작업 제한인 2개를 초과했습니다

QLDB은 기본적으로 동시 저널 내보내기 작업을 2개로 제한합니다.

저널 데이터 스트리밍

이 섹션에는 원장에서 Amazon Kinesis Data Streams로 저널 데이터를 스트리밍할 때 QLDB가 반환할 수 있는 일반적인 예외가 나와 있습니다. 이 기능에 대한 자세한 내용은 Amazon QLDB에서 저널 데이터 스트리밍을 참조하십시오.

각 예외에는 특정 오류 메시지와 가능한 해결 방법에 대한 간단한 설명 및 제안이 포함됩니다.

AccessDeniedException

메시지: User: *userARN* is not authorized to perform: iam:PassRole on resource: *roleARN*

IAM 역할을 QLDB 서비스에 전달할 권한이 없습니다. QLDB에는 모든 저널 스트림 요청에 대한 역할이 필요하며 이 역할을 QLDB에 전달할 수 있는 권한이 있어야 합니다. 이 역할은 지정된 Amazon Kinesis Data Streams 리소스에 대한 쓰기 권한을 QLDB에 제공합니다.

QLDB 서비스(`qldb.amazonaws.com`)의 지정된 IAM 역할 리소스에서 `PassRole` API 작업을 수행할 권한을 부여하는 IAM 정책을 정의했는지 확인하세요. 정책 예제는 Amazon QLDB의 자격 증명 기반 정책 예을 참조하십시오.

IllegalArgumentException

메시지: QLDB encountered an error validating Kinesis Data Streams: Response from Kinesis: *errorCodeerrorMessage*

이 오류의 가능한 원인은 제공된 Kinesis Data Streams 리소스가 존재하지 않기 때문입니다. 또는 QLDB에 지정된 Kinesis 데이터 스트림에 데이터 레코드를 쓸 수 있는 충분한 권한이 없습니다.

스트림 요청에서 제공하는 Kinesis 데이터 스트림이 올바른지 확인하세요. 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 데이터 스트림 생성 및 업데이트를 참조하세요.

또한 다음 작업에 QLDB 서비스(`qldb.amazonaws.com`) 권한을 부여하는 지정된 Kinesis 데이터 스트림에 대한 정책을 정의했는지 확인합니다. 자세한 내용은 스트림 권한을 참조하십시오.

- `kinesis:PutRecord`
- `kinesis:PutRecords`

- `kinesis:DescribeStream`
- `kinesis:ListShards`

IllegalArgumentException

메시지: Kinesis 구성을 검증하는 동안 Kinesis 데이터 스트림에서 예상치 못한 응답이 발생했습니다. Kinesis의 응답: *errorCode errorMessage*

제공된 Kinesis 오류 응답으로 인해 제공된 Kinesis 데이터 스트림에 데이터 레코드를 쓰려는 시도가 실패했습니다. 발생 가능한 원인에 대한 자세한 내용은 Amazon Kinesis Data Streams 개발자 안내서의 Amazon Kinesis Data Streams 생산자 문제 해결을 참조하십시오.

IllegalArgumentException

메시지: 시작 날짜는 종료 날짜보다 이후일 수 없습니다.

`InclusiveStartTime` 및 `ExclusiveEndTime`는 ISO 8601 날짜 및 시간 형식이어야 하며 협정 세계시(UTC)여야 합니다.

IllegalArgumentException

메시지: 종료일은 미래일 수 없습니다

`InclusiveStartTime` 및 `ExclusiveEndTime`은 UTC 기준의 `ISO 8601` 날짜 및 시간 형식이어야 합니다.

LimitExceededException

메시지: Kinesis Data Streams에 대한 동시 실행 저널 스트림 제한인 5개를 초과했습니다

QLDB는 기본적으로 동시 저널 스트림을 5개로 제한합니다.

저널 데이터 확인

이 섹션에는 원장의 저널 데이터를 확인할 때 QLDB가 반환할 수 있는 일반적인 예외가 나열되어 있습니다. 이 기능에 대한 자세한 내용은 Amazon QLDB에서의 데이터 확인을 참조하십시오.

각 예외에는 특정 오류 메시지와 이를 발생시킬 수 있는 API 작업, 간단한 설명, 가능한 해결 방법에 대한 제안이 포함됩니다.

IllegalArgumentException

메시지: 제공된 Ion 값이 유효하지 않아 구문 분석할 수 없습니다.

API 작업: `GetDigest, GetBlock, GetRevision`

요청을 재시도하기 전에 유효한 Amazon Ion 값을 제공했는지 확인하세요.

IllegalArgumentException

메시지: 제공된 블록 주소가 유효하지 않습니다.

API 작업: `GetDigest, GetBlock, GetRevision`

요청을 다시 시도하기 전에 유효한 블록 주소를 입력했는지 확인하세요. 블록 주소는 `strandId` 및 `sequenceNo`라는 두 개의 필드를 갖는 Amazon Ion 구조입니다.

예: `{strandId:"BlFTjlSXze9BIh1K0szcE3",sequenceNo:14}`

IllegalArgumentException

메시지: 제공된 다이제스트 팁 주소의 시퀀스 번호가 스트랜드의 최근 커밋된 레코드를 벗어났습니다.

API 작업: `GetDigest, GetBlock, GetRevision`

제공하는 다이제스트 팁 주소의 시퀀스 번호는 저널 스트랜드의 최근 커밋된 레코드의 시퀀스 번호보다 작거나 같아야 합니다. 요청을 다시 시도하기 전에 유효한 시퀀스 번호가 있는 다이제스트 팁 주소를 제공해야 합니다.

IllegalArgumentException

메시지: 제공된 블록 주소의 스트랜드 ID가 유효하지 않습니다.

API 작업: `GetDigest, GetBlock, GetRevision`

제공하는 블록 주소에는 저널의 스트랜드 ID와 일치하는 스트랜드 ID가 있어야 합니다. 요청을 다시 시도하기 전에 유효한 스트랜드 ID를 가진 블록 주소를 제공해야 합니다.

IllegalArgumentException

메시지: 제공된 블록 주소의 시퀀스 번호가 스트랜드의 최신 커밋 레코드를 벗어났습니다.

API 작업: `GetBlock, GetRevision`

제공하는 블록 주소는 스트랜드의 최근 커밋된 레코드의 시퀀스 번호보다 작거나 같은 시퀀스 번호를 가져야 합니다. 요청을 재시도하기 전에 유효한 시퀀스 번호가 있는 블록 주소를 제공해야 합니다.

IllegalArgumentException

메시지: 제공된 블록 주소의 스트랜드 ID는 제공된 다이제스트 팁 주소의 스트랜드 ID와 일치해야 합니다.

API 작업: `GetBlock, GetRevision`

제공한 다이제스트와 동일한 저널 스트랜드에 문서 개정 또는 블록이 존재하는 경우에만 문서 수정 또는 차단을 확인할 수 있습니다.

IllegalArgumentException

메시지: 제공된 블록 주소의 시퀀스 번호가 스트랜드의 최신 커밋 레코드를 벗어났습니다.

API 작업: `GetBlock, GetRevision`

제공한 다이제스트에 포함된 경우에만 문서 수정 또는 차단을 확인할 수 있습니다. 즉, 다이제스트 팁 주소보다 먼저 저널에 커밋되었다는 뜻입니다.

IllegalArgumentException

메시지: 지정한 블록 주소의 블록에서 제공된 문서 ID를 찾을 수 없습니다.

API 작업: `GetRevision`

제공하는 문서 ID는 입력한 블록 주소에 있어야 합니다. 요청을 재시도하기 전에 이 두 파라미터가 일치하는지 확인하세요.

Amazon QLDB PartiQL 참조

Amazon QLDB는 PartiQL 쿼리 언어의 하위 집합을 지원합니다. 다음 항목에서는 PartiQL의 QLDB 구현을 설명합니다.

> **Note**
>
> - QLDB는 일부 PartiQL 작업을 지원하지 않습니다.
> - QLDB의 모든 PartiQL 문에는 Amazon QLDB 할당량 및 제한에 정의된 대로 트랜잭션 한도가 적용됩니다.
> - 이 참조는 QLDB 콘솔 또는 QLDB 쉘에서 수동으로 실행하는 PartiQL 문의 기본 구문과 사용 예제를 제공합니다. QLDB 드라이버를 사용하여 유사한 명령문을 프로그래밍 방식으로 실행하는 방법을 보여주는 코드 예제는 드라이버 시작하기의 자습서를 참조하세요.

주제

PartiQL이란?

PartiQL은 구조화 데이터, 반구조화 데이터 및 중첩 데이터를 포함하는 여러 데이터 저장소에 걸쳐 SQL 호환 쿼리 액세스를 제공합니다. 이는 Amazon 내에서 널리 사용되며 현재 QLDB를 비롯한 여러 AWS 서비스 서비스의 일부로 제공됩니다.

PartiQL 사양과 핵심 쿼리 언어에 대한 자습서는 PartiQL 설명서를 참조하세요.

PartiQL은 SQL-92 버전을 확장하여 Amazon Ion 데이터 형식의 문서를 지원합니다. Amazon Ion에 대한 자세한 내용은 Amazon QLDB의 Amazon Ion 데이터 형식 참조 섹션을 참조하세요.

Amazon QLDB의 PartiQL

QLDB에서 PartiQL 쿼리를 실행하려면 다음 중 하나를 사용할 수 있습니다.

- QLDB용 AWS Management Console의 PartiQL 편집기
- 명령줄 QLDB 쉘
- 프로그래밍 방식으로 쿼리를 실행하기 위한 AWS 제공 QLDB 드라이버

이러한 메서드를 사용하여 QLDB에 액세스하는 방법에 대한 자세한 내용은 Amazon QLDB 액세스 섹션을 참조하세요.

특정 테이블에서 각 PartiQL 명령을 실행하기 위한 액세스를 제어하는 방법을 알아보려면 Amazon QLDB에서 표준 권한 모드로 시작하기을 참조하세요.

QLDB의 PartiQL 빠른 팁

다음은 QLDB에서 PartiQL을 사용하기 위한 팁과 모범 사례를 간략하게 요약한 것입니다.

- 동시성 및 트랜잭션 제한 이해 - `SELECT` 쿼리를 포함한 모든 명령문은 30초의 트랜잭션 시간 제한을 포함하여 OCC(Optimistic Concurrency Control) 충돌 및 트랜잭션 제한의 영향을 받습니다.
- 인덱스 사용 – 카디널리티가 높은 인덱스를 사용하고 대상 쿼리를 실행하여 명령문을 최적화하고 전체 테이블 스캔을 방지합니다. 자세한 내용은 쿼리 성능 최적화 섹션을 참조하세요.
- 동등 조건자 사용 – 인덱싱된 조회에는 동등 연산자(= 또는 `IN`)가 필요합니다. 부등 연산자(<, >, `LIKE`, `BETWEEN`)는 인덱싱된 조회에 적합하지 않으며 테이블 전체를 스캔해야 합니다.

- 내부 조인만 사용 — QLDB는 내부 조인만 지원합니다. 가장 좋은 방법은 조인하려는 각 테이블에 대해 인덱싱된 필드를 조인하는 것입니다. 조인 기준과 동등 조건자 모두에 대해 높은 카디널리티 인덱스를 선택하세요.

Amazon QLDB PartiQL 참조 규칙

이번 섹션에서는 Amazon QLDB PartiQL 참조에서 설명한 PartiQL 명령, 함수 및 표현식의 구문을 작성하는 데 사용되는 규칙에 대해 설명합니다. 이러한 규칙을 PartiQL 쿼리 언어 자체의 구문 및 시맨틱과 혼동하지 마세요.

문자	설명
CAPS	대문자로 된 단어는 키워드입니다.
[ ]	대괄호는 선택적 인수 또는 절을 나타냅니다. 다수의 인수가 대괄호로 묶이면 인수를 얼마든지 선택할 수 있다는 것을 의미합니다. 또한 별도의 라인에서 대괄호로 묶이는 인수는 QLDB 구문 분석기에서 인수가 구문에 나열된 순서를 그대로 따른다는 것을 의미합니다.
|	파이프는 인수 중에서 하나를 선택할 수 있다는 것을 의미합니다.
### ####	빨간색 기울임꼴 단어는 자리 표시자를 나타냅니다. 빨간색 기울임꼴 단어 자리에 적절한 값을 넣습니다.
. . .	줄임표는 앞의 요소를 반복할 수 있음을 나타냅니다.
'	작은따옴표로 표시된 값은 작은따옴표를 입력해야 함을 나타냅니다. PartiQL에서 작은따옴표는 Amazon Ion 구조의 문자열 값 또는 필드 이름을 나타냅니다.
"	큰따옴표로 묶인 값은 큰따옴표를 입력해야 함을 나타냅니다. PartiQL에서 큰따옴표는 따옴표로 묶인 식별자를 나타냅니다.
`	백틱 값은 백틱을 입력해야 함을 나타냅니다. PartiQL에서 백틱은 Ion 리터럴 값을 나타냅니다.

Amazon QLDB의 데이터 유형

Amazon QLDB는 문서를 Amazon Ion 형식으로 저장합니다. Amazon Ion은 JSON의 상위 집합인 데이터 직렬화 형식(텍스트 형식과 이진 인코딩 형식 모두)입니다. 다음 표는 QLDB 문서에서 사용할 수 있는 Ion 데이터 유형을 나열합니다.

데이터 형식	설명
`null`	일반적인 null 값
`bool`	부울 값
`int`	임의 크기의 부호 있는 정수
`decimal`	임의 정밀도의 십진 부호화 실수
`float`	이진 인코딩된 부동 소수점 숫자(64비트 IEEE)
`timestamp`	임의 정밀도의 날짜/시간/시간대 모멘트
`string`	유니코드 텍스트 리터럴
`symbol`	유니코드 기호 원자(식별자)
`blob`	사용자 정의 인코딩의 이진 데이터
`clob`	사용자 정의 인코딩의 텍스트 데이터
`struct`	정렬되지 않은 이름-값 쌍 컬렉션
`list`	정렬된 서로 다른 유형의 값 컬렉션

전체 설명 및 값 형식 세부 정보가 포함된 Ion 코어 데이터 유형의 전체 목록은 Amazon GitHub 사이트의 Ion 사양 문서를 참조하세요.

Amazon QLDB 문서

Amazon QLDB는 데이터 레코드를 문서로 저장합니다. 문서는 테이블에 삽입되는 Amazon Ion `struct` 객체입니다. Ion 사양은 Amazon Ion GitHub 사이트를 참조하세요.

주제

Ion 문서 구조

JSON과 마찬가지로 QLDB 문서는 다음과 같은 구조의 이름-값 쌍으로 구성됩니다.

```
{
  name1: value1,
  name2: value2,
  name3: value3,
  ...
  nameN: valueN
}
```

이름은 심볼 토큰이며 값에는 제한이 없습니다. 각 이름-값 쌍을 필드라고 합니다. 필드 값은 컨테이너 유형(중첩 구조체, 목록, 구조체 목록)을 비롯한 모든 Ion 데이터 유형이 될 수 있습니다.

또한 JSON과 마찬가지로 `struct`는 중괄호(`{...}`)로 표시되고 `list`는 대괄호(`[...]`)로 표시됩니다. 다음 예제는 Amazon QLDB 콘솔 시작하기의 표본 데이터에서 다양한 유형의 값이 포함된 문서입니다.

```
{
    VIN: "1N4AL11D75C109151",
    LicensePlateNumber: "LEWISR261LL",
    State: "WA",
    City: "Seattle",
    PendingPenaltyTicketAmount: 90.25,
    ValidFrom: 2017-08-21T,
    ValidTo: 2020-05-11T,
    Owners: {
        PrimaryOwner: { PersonId: "294jJ3YUoH1IEEm8GSabOs" },
        SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }]
    }
}
```

> **Important**
>
> Ion에서 큰따옴표는 문자열 값을 나타내고 따옴표가 없는 기호는 필드 이름을 나타냅니다. 그러나 PartiQL에서 작은따옴표는 문자열과 필드 이름을 모두 나타냅니다.
> 이러한 구문 차이로 인해 PartiQL 쿼리 언어는 SQL 호환성을 유지하고 Amazon Ion 데이터 형식은 JSON 호환성을 유지할 수 있습니다. QLDB에서 PartiQL의 구문 및 시맨틱에 대한 자세한 내용은 PartiQL을 사용하여 Ion 쿼리하기 섹션을 참조하세요.

PartiQL-Ion 유형 매핑

QLDB에서 PartiQL은 SQL의 유형 시스템을 확장하여 Ion 데이터 모델을 포괄합니다. 이 매핑은 다음과 같이 설명됩니다.

- SQL 스칼라 유형은 해당 Ion 유형에 의해 적용됩니다. 예:
  - `CHAR` 및 `VARCHAR`는 Ion `string` 유형에 매핑되는 유니코드 시퀀스입니다.
  - `NUMBER`는 Ion `decimal` 유형에 매핑됩니다.
- Ion의 `struct` 타입은 전통적으로 테이블 행을 나타내는 SQL 튜플과 동일합니다.
  - 그러나 콘텐츠가 열려 있고 스키마가 없는 경우 SQL 튜플의 순서가 지정된 특성(예: `SELECT *` 출력 순서)에 의존하는 쿼리는 지원되지 않습니다.
- `NULL` 외에도 PartiQL에는 `MISSING` 유형이 있습니다. 이는 `NULL`의 전문화로 필드가 부족함을 나타냅니다. Ion `struct` 필드가 희소할 수 있으므로 이 유형이 필요합니다.

문서 ID

QLDB는 테이블에 삽입하는 각 문서에 문서 ID를 할당합니다. 모든 시스템 할당 ID는 Base62로 인코딩된 문자열(예: `3Qv67yjXEwB9SjmvkuG6Cp`)로 표시되는 범용 고유 식별자(UUID)입니다. 자세한 내용은 Amazon QLDB의 고유 ID을 참조하십시오.

각 문서 개정은 문서 ID와 0으로 시작하는 버전 번호의 조합으로 고유하게 식별됩니다.

문서 ID 및 버전 필드는 문서의 메타데이터에 포함되며, 커밋된 보기(테이블의 시스템 정의 뷰)에서 쿼리할 수 있습니다. QLDB 뷰에 대한 자세한 내용은 핵심 개념 섹션을 참조하세요. 메타데이터에 대해 자세히 알아보려면 문서 메타데이터 쿼리 섹션을 참조하세요.

Amazon QLDB에서 PartiQL을 사용한 Ion 쿼리

Amazon QLDB에서 데이터를 쿼리할 때는 PartiQL 형식으로 명령문을 작성하지만 QLDB는 Amazon Ion 형식으로 결과를 반환합니다. PartiQL은 SQL과 호환되도록 고안된 반면, Ion은 JSON의 확장입니다. 이로 인해 쿼리 명령문의 데이터를 표기하는 방식과 쿼리 결과가 표시되는 방식 간에 구문상의 차이가 발생합니다.

이 섹션에서는 QLDB 콘솔 또는 QLDB 쉘을 사용하여 PartiQL 문을 수동으로 실행하기 위한 기본 구문과 의미를 설명합니다.

> **Tip**
>
> 프로그래밍 방식으로 PartiQL 쿼리를 실행하는 경우 가장 좋은 방법은 매개 변수가 있는 명령문을 사용하는 것입니다. 명령문에서 물음표(?)를 바인드 변수 자리 표시자로 사용하면 이러한 구문 규칙을 피할 수 있습니다. 또한 이 방법이 더 안전하고 효율적입니다.
> 자세한 내용은 드라이버 시작하기에서 다음 자습서를 참조하세요.
>
> - Java: 빠른 시작 자습서 | Cookbook 참조
> - .NET: 빠른 시작 자습서 | Cookbook 참조
> - Go: 빠른 시작 자습서 | Cookbook 참조
> - Node.js: 빠른 시작 자습서 | Cookbook 참조
> - Python: 빠른 시작 자습서 | Cookbook 참조

주제

- 구문 및 시맨틱
- 백틱 표기법
- 경로 탐색
- 별칭
- PartiQL 사양

구문 및 시맨틱

QLDB 콘솔 또는 QLDB 쉘을 사용하여 Ion 데이터를 쿼리하는 경우 PartiQL의 기본 구문과 시맨틱은 다음과 같습니다.

대소문자 구분

필드 이름, 테이블 이름, 원장 이름을 비롯한 모든 QLDB 시스템 개체 이름은 대소문자를 구분합니다.

문자열 값

Ion에서 큰따옴표(`"..."`)는 문자열을 나타냅니다.

PartiQL에서 작은따옴표(`'...'`)는 문자열을 나타냅니다.

기호 및 식별자

Ion에서 작은따옴표(`'...'`)는 기호를 나타냅니다. Ion에서 식별자라고 하는 기호 중 일부는 따옴표가 없는 텍스트로 표시됩니다.

PartiQL에서 큰따옴표(`"..."`)는 따옴표로 묶인 PartiQL 식별자(예: 테이블 이름으로 사용되는 예약어)를 나타냅니다. 따옴표가 없는 텍스트는 예약어가 아닌 테이블 이름과 같은 일반 PartiQL 식별자를 나타냅니다.

Ion 리터럴

모든 Ion 리터럴은 PartiQL 문에서 백틱(`` `...` ``)으로 표시할 수 있습니다.

필드 이름

Ion 필드 이름은 대소문자를 구분합니다. PartiQL을 사용하면 DML 문에서 필드 이름을 작은따옴표로 표시할 수 있습니다. 이는 PartiQL의 `cast` 함수를 사용하여 심볼을 정의하는 것에 대한 간단한 대안입니다. 또한 백틱을 사용하여 리터럴 Ion 심볼을 나타내는 것보다 직관적입니다.

Literal

PartiQL 쿼리 언어의 리터럴은 다음과 같이 Ion 데이터 유형에 해당합니다.

Scalar

해당하는 경우 PartiQL-Ion 유형 매핑 섹션에 설명된 대로 SQL 구문을 따르세요. 예:

- `5`
- `'foo'`
- `null`

Struct

다양한 형식 및 기타 데이터 모델에서 튜플 또는 객체라고도 합니다.

중괄호({...})로 표시되며 struct 요소는 쉼표로 구분됩니다.

- `{ 'id' : 3, 'arr': [1, 2] }`

List

배열이라고도 합니다.

대괄호([...])로 표시되며 목록 요소는 쉼표로 구분됩니다.

- `[ 1, 'foo' ]`

Bag

PartiQL의 순서가 지정되지 않은 컬렉션입니다.

양각 괄호(<<...>>)로 표시되며, 백 요소는 쉼표로 구분됩니다. QLDB에서는 테이블을 백으로 생각할 수 있습니다. 하지만 백은 테이블의 문서 내에 중첩될 수 없습니다.

- `<< 1, 'foo' >>`

예

다음은 다양한 Ion 유형을 사용하는 INSERT 문 구문의 예입니다.

```
INSERT INTO VehicleRegistration VALUE
{
    'VIN' : 'KM8SRDHF6EU074761', --string
    'RegNum' : 1722, --integer
    'State' : 'WA',
    'City' : 'Kent',
    'PendingPenaltyTicketAmount' : 130.75, --decimal
    'Owners' : { --nested struct
        'PrimaryOwner' : { 'PersonId': '294jJ3YUoH1IEEm8GSabOs' },
        'SecondaryOwners' : [ --list of structs
            { 'PersonId' : '1nmeDdLo3AhGswBtyM1eYh' },
            { 'PersonId': 'IN7MvYtUjkp1GMZu0F6CG9' }
        ]
    },
    'ValidFromDate' : `2017-09-14T`, --Ion timestamp literal with day precision
    'ValidToDate' : `2020-06-25T`
}
```

백틱 표기법

PartiQL은 모든 Ion 데이터 유형을 완벽하게 지원하므로 백틱을 사용하지 않고도 어떤 명령문이든 작성할 수 있습니다. 하지만 이 Ion 리터럴 구문을 사용하면 명령문을 더 명확하고 간결하게 만들 수 있는 경우도 있습니다.

예를 들어, Ion 타임스탬프와 기호 값이 있는 문서를 삽입하려면 순수 PartiQL 구문만 사용하여 다음 명령문을 작성할 수 있습니다.

```
INSERT INTO myTable VALUE
{
    'myTimestamp': to_timestamp('2019-09-04T'),
    'mySymbol': cast('foo' as symbol)
}
```

이는 매우 장황하므로 대신 백틱을 사용하여 문을 단순화할 수 있습니다.

```
INSERT INTO myTable VALUE
{
    'myTimestamp': `2019-09-04T`,
    'mySymbol': `foo`
}
```

전체 구조를 백틱으로 묶어 키 입력을 몇 번 더 줄일 수도 있습니다.

```
INSERT INTO myTable VALUE
`{
    myTimestamp: 2019-09-04T,
    mySymbol: foo
}`
```

Important

문자열과 기호는 PartiQL에서 서로 다른 클래스입니다. 즉, 텍스트가 같더라도 동일하지는 않습니다. 예를 들어, 다음 PartiQL 식은 다른 Ion 값으로 평가됩니다.

```
'foo'
```

```
`foo`
```

경로 탐색

DML(데이터 조작 언어) 또는 쿼리 문을 작성할 때 경로 단계를 사용하여 중첩 구조 내의 필드에 액세스할 수 있습니다. PartiQL은 상위 구조의 필드 이름에 액세스하는 데 사용할 수 있는 점 표기법을 지원합니다. 다음 예제에서는 상위 `Vehicle`의 `Model` 필드에 액세스합니다.

```
Vehicle.Model
```

목록의 특정 요소에 접근하려면 대괄호 연산자를 사용하여 0으로 시작하는 서수를 표시할 수 있습니다. 다음 예제에서는 서수 2를 사용하여 `SecondaryOwners` 요소에 액세스합니다. 즉, 이 요소가 목록의 세 번째 요소입니다.

```
SecondaryOwners[2]
```

별칭

QLDB는 개방형 콘텐츠 및 스키마를 지원합니다. 따라서 명령문의 특정 필드에 액세스할 때 기대하는 결과를 얻을 수 있는 가장 좋은 방법은 별칭을 사용하는 것입니다. 예를 들어 명시적 별칭을 지정하지 않으면 시스템은 `FROM` 소스에 대해 암시적 별칭을 생성합니다.

```
SELECT VIN FROM Vehicle
--is rewritten to
SELECT Vehicle.VIN FROM Vehicle AS Vehicle
```

하지만 필드 이름 충돌로 인한 결과는 예측할 수 없습니다. `VIN`로 이름이 지정된 다른 필드가 문서 내 중첩된 구조에 있는 경우 이 쿼리에서 반환되는 `VIN` 값을 보면 놀라울 수 있습니다. 가장 좋은 방법은 다음 문을 대신 작성하는 것입니다. 이 쿼리는 `v`를 `Vehicle` 테이블 범위의 별칭으로 선언합니다. `AS` 키워드는 선택 사항입니다.

```
SELECT v.VIN FROM Vehicle [ AS ] v
```

별칭은 문서 내 중첩된 컬렉션에 경로를 지정할 때 특히 유용합니다. 예를 들어, 다음 명령문은 `o`를 컬렉션 `VehicleRegistration.Owners` 범위의 별칭으로 선언합니다.

```
SELECT o.SecondaryOwners
FROM VehicleRegistration AS r, @r.Owners AS o
```

여기서 이 @ 문자는 엄밀히 따지자면 선택 사항입니다. 하지만 이는 Owners이라는 다른 컬렉션(있는 경우)이 아니라 VehicleRegistration 안에 Owners 구조를 적용하고자 한다는 것을 명시적으로 나타냅니다.

PartiQL 사양

PartiQL 쿼리 언어에 대한 자세한 내용은 PartiQL 사양을 참조하세요.

Amazon QLDB의 PartiQL 명령

PartiQL은 SQL-92 버전을 확장하여 Amazon Ion 데이터 형식의 문서를 지원합니다. Amazon QLDB에서는 다음 PartiQL 명령을 지원합니다.

특정 테이블에서 각 PartiQL 명령을 실행하기 위한 액세스를 제어하는 방법을 알아보려면 Amazon QLDB에서 표준 권한 모드로 시작하기을 참조하세요.

> **Note**
>
> - QLDB는 모든 PartiQL 명령을 지원하지 않습니다.
> - QLDB의 모든 PartiQL 문에는 Amazon QLDB 할당량 및 제한에 정의된 대로 트랜잭션 한도가 적용됩니다.
> - 이 참조는 QLDB 콘솔 또는 QLDB 쉘에서 수동으로 실행하는 PartiQL 문의 기본 구문과 사용 예제를 제공합니다. 지원되는 프로그래밍 언어를 사용하여 유사한 명령문을 실행하는 방법을 보여주는 코드 예제는 드라이버 시작하기의 자습서를 참조하세요.

DDL(데이터 정의 언어) 문

DDL(데이터 정의 언어) 는 테이블 및 인덱스와 같은 데이터베이스 개체를 관리하는 데 사용하는 PartiQL 문 집합입니다. DDL을 사용하여 이러한 객체를 만들고 삭제합니다.

- CREATE INDEX
- CREATE TABLE
- DROP INDEX
- DROP TABLE
- UNDROP TABLE

DML(데이터 조작 언어) 문

DML(데이터 조작 언어)는 QLDB 테이블의 데이터를 관리하는 데 사용하는 PartiQL 문 집합입니다. DML 문을 사용하여 테이블의 데이터를 추가, 수정 또는 삭제할 수 있습니다.

지원되는 DML 및 쿼리 언어 문은 다음과 같습니다.

- DELETE
- FROM (INSERT, REMOVE, 또는 SET)
- INSERT
- SELECT
- UPDATE

Amazon QLDB에서 CREATE INDEX 명령

Amazon QLDB에서는 `CREATE INDEX` 명령을 사용하여 테이블의 문서 필드에 대한 인덱스를 생성합니다.

특정 테이블에서 이 PartiQL 명령을 실행하기 위한 액세스를 제어하는 방법을 알아보려면 Amazon QLDB에서 표준 권한 모드로 시작하기 섹션을 참조하세요.

> **Important**
>
> QLDB는 문서를 효율적으로 조회하기 위한 인덱스가 필요합니다. 인덱스가 없으면 QLDB는 문서를 읽을 때 전체 테이블 스캔을 수행해야 합니다. 이로 인해 동시성 충돌 및 트랜잭션 시간 초과를 포함하여 대규모 테이블에서 성능 문제가 발생할 수 있습니다.
> 인덱싱된 필드 또는 문서 ID(예: `=` 또는 `IN`)에서 동등 연산자를 사용하여 `WHERE` 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 자세한 내용은 쿼리 성능 최적화을 참조하십시오.

인덱스를 생성할 때는 다음 제약 조건에 유의하세요.

- 인덱스는 단일 최상위 필드에만 생성할 수 있습니다. 복합, 중첩, 고유 및 함수 기반 인덱스는 지원되지 않습니다.
- `list` 및 `struct`를 비롯한 모든 Ion 데이터 유형에 대해 인덱스를 생성할 수 있습니다. 그러나 Ion 유형에 관계없이 전체 Ion 값이 같아야만 인덱스 조회를 수행할 수 있습니다. 예를 들어 `list` 형식을 인덱스로 사용하는 경우 목록 내에서 한 항목씩 인덱스 검색을 수행할 수 없습니다.

- 동등 조건자(예: `WHERE indexedField = 123` 또는 `WHERE indexedField IN (456, 789)`)를 사용할 때만 쿼리 성능이 향상됩니다.

  QLDB는 쿼리 조건자의 부등을 인정하지 않습니다. 따라서 범위 필터링된 스캔은 구현되지 않습니다.
- 인덱싱된 필드 이름은 대소문자를 구분하며 최대 128자입니다.
- QLDB에서의 인덱스 생성은 비동기적으로 이루어집니다. 비어 있지 않은 테이블에서 인덱스 빌드를 완료하는 데 걸리는 시간은 테이블 크기에 따라 다릅니다. 자세한 내용은 인덱스 관리을 참조하십시오.

주제

구문

```
CREATE INDEX ON table_name (field)
```

파라미터

table_name

인덱스를 생성하려는 테이블의 이름입니다. 이미 있는 테이블이어야 합니다.

테이블 이름은 대/소문자를 구분합니다.

##

인덱스를 생성할 문서 필드 이름입니다. 필드는 최상위 속성이어야 합니다.

인덱싱된 필드 이름은 대소문자를 구분하며 최대 128자입니다.

`list`및 `struct`를 비롯한 모든 Amazon Ion 데이터 유형에 인덱스를 생성할 수 있습니다. 그러나 Ion 유형에 관계없이 전체 Ion 값이 같아야만 인덱스 조회를 수행할 수 있습니다. 예를 들어 `list` 형식을 인덱스로 사용하는 경우 목록 내에서 한 항목씩 인덱스 검색을 수행할 수 없습니다.

반환 값

`tableId` – 인덱스를 생성한 테이블의 고유 ID입니다.

예시

```
CREATE INDEX ON VehicleRegistration (LicensePlateNumber)
```

```
CREATE INDEX ON Vehicle (VIN)
```

드라이버를 사용하여 프로그래밍 방식으로 실행

QLDB 드라이버를 사용하여 이 명령문을 프로그래밍 방식으로 실행하는 방법을 알아보려면 드라이버 시작하기에서 다음 자습서를 참조하세요.

- Java: 빠른 시작 자습서 | Cookbook 참조
- .NET: 빠른 시작 자습서 | Cookbook 참조
- Go: 빠른 시작 자습서 | Cookbook 참조
- Node.js: 빠른 시작 자습서 | Cookbook 참조
- Python: 빠른 시작 자습서 | Cookbook 참조

Amazon QLDB에서 CREATE TABLE

Amazon QLDB에서는 `CREATE TABLE` 명령을 사용하여 새 테이블을 생성합니다.

테이블에는 네임스페이스가 없는 단순한 이름이 있습니다. QLDB는 개방형 콘텐츠를 지원하며 스키마를 적용하지 않으므로 테이블을 만들 때 속성이나 데이터 유형을 정의하지 않습니다.

> **Note**
>
> 원장에서 이 PartiQL 명령을 실행하기 위한 액세스를 제어하는 방법을 알아보려면 Amazon QLDB에서 표준 권한 모드로 시작하기을 참조하세요.

주제

- 구문

구문

```
CREATE TABLE table_name [ WITH (aws_tags = `{'key': 'value'}`) ]
```

파라미터

table_name

생성할 테이블의 고유 이름입니다. 같은 이름의 활성 테이블이 이미 존재하지 않아야 합니다. 다음은 명명 제약 조건입니다.

- 1~128자의 영숫자 또는 밑줄만 포함해야 합니다.
- 첫 번째 글자에는 문자 또는 밑줄이 있어야 합니다.
- 영숫자와 밑줄을 조합하여 나머지 문자에 사용할 수 있습니다.
- 대소문자를 구분합니다.
- QLDB PartiQL 예약어가 아니어야 합니다.

'***key***': '***value***'

(선택 사항)생성 중에 테이블 리소스에 연결할 태그입니다. 각 태그는 키-값 쌍으로 정의되며, 키와 값은 각각 작은 따옴표로 표시됩니다. 각 키-값 쌍은 백틱으로 표시되는 Amazon Ion 구조 내에 정의됩니다.

테이블 생성 시 태그 지정은 현재 *STANDARD* 권한 모드의 원장에만 지원됩니다.

반환 값

`tableId` – 생성한 테이블의 고유 ID입니다.

테이블 생성 시 태그 지정

> **Note**
>
> 테이블 생성 시 태그 지정은 현재 STANDARD 권한 모드의 원장에만 지원됩니다.

선택적으로 CREATE TABLE 문에 태그를 지정하여 테이블 리소스에 태그를 지정할 수 있습니다. 태그에 대한 자세한 내용은 Amazon QLDB 리소스 태그 지정 섹션을 참조하세요. 다음 예제에서는 environment=production 태그가 있는 Vehicle이라는 테이블을 생성합니다.

```
CREATE TABLE Vehicle WITH (aws_tags = `{'environment': 'production'}`)
```

생성 시 테이블에 태그를 지정하려면 qldb:PartiQLCreateTable 및 qldb:TagResource 작업 모두에 액세스할 수 있어야 합니다. QLDB 리소스 권한에 대한 자세한 내용은 Amazon QLDB에서 IAM을 사용하는 방법 섹션을 참조하세요.

리소스를 생성하는 동안 태그를 지정하면 리소스 생성 후 사용자 지정 태그 지정 스크립트를 실행할 필요가 없습니다. 테이블에 태그가 지정된 후 해당 태그를 기반으로 테이블에 대한 액세스를 제어할 수 있습니다. 예를 들어 특정 태그가 있는 테이블에만 전체 액세스 권한을 부여할 수 있습니다. JSON 정책의 예는 테이블 태그를 기반으로 하는 모든 작업에 대한 전체 액세스 권한 섹션을 참조하세요.

예시

```
CREATE TABLE VehicleRegistration
```

```
CREATE TABLE Vehicle WITH (aws_tags = `{'environment': 'development'}`)
```

```
CREATE TABLE Vehicle WITH (aws_tags = `{'key1': 'value1', 'key2': 'value2'}`)
```

드라이버를 사용하여 프로그래밍 방식으로 실행

QLDB 드라이버를 사용하여 이 명령문을 프로그래밍 방식으로 실행하는 방법을 알아보려면 드라이버 시작하기에서 다음 자습서를 참조하세요.

- Java: 빠른 시작 자습서 | Cookbook 참조
- .NET: 빠른 시작 자습서 | Cookbook 참조
- Go: 빠른 시작 자습서 | Cookbook 참조

- Node.js: 빠른 시작 자습서 | Cookbook 참조
- Python: 빠른 시작 자습서 | Cookbook 참조

Amazon QLDB의 DELETE 명령

Amazon QLDB에서는 DELETE 명령을 사용하여 문서의 새로운 최종 개정본을 생성하여 활성 문서를 테이블에서 삭제된 것으로 표시합니다. 이 최종 개정본은 문서가 삭제되었음을 나타냅니다. 이 작업을 수행하면 문서의 수명 주기가 종료되므로 동일한 문서 ID로 더 이상 문서 개정본을 만들 수 없습니다.

이 작업은 되돌릴 수 없습니다. 기록 함수를 사용하여 삭제된 문서의 개정 기록을 계속 쿼리할 수 있습니다.

> **Note**
>
> 특정 테이블에서 이 PartiQL 명령을 실행하기 위한 액세스를 제어하는 방법을 알아보려면 Amazon QLDB에서 표준 권한 모드로 시작하기 섹션을 참조하세요.

주제

구문

```
DELETE FROM table_name [ AS table_alias ] [ BY id_alias ]
[ WHERE condition ]
```

파라미터

table_name

삭제할 데이터가 포함된 사용자 테이블 이름입니다. DML 문은 기본 사용자 뷰에서만 지원됩니다. 각 명령문은 단일 테이블에서만 실행할 수 있습니다.

AS *table_alias*

(선택 사항)삭제할 테이블에 속하는 사용자 정의 별칭입니다. AS 키워드는 선택 사항입니다.

BY *id_alias*

(선택 사항)결과 집합에 있는 각 문서의 id 메타데이터 필드에 바인딩되는 사용자 정의 별칭입니다. BY 키워드를 사용하여 FROM 절에서 별칭을 선언해야 합니다. 이는 기본 사용자 뷰를 쿼리하면서 문서 ID를 기준으로 필터링하려는 경우에 유용합니다. 자세한 내용은 BY 절을 사용하여 문서 ID 쿼리하기을 참조하십시오.

WHERE *condition*

삭제할 문서의 선택 기준입니다.

> **Note**
>
> WHERE 절을 생략하면 테이블 내의 모든 문서가 삭제됩니다.

반환 값

documentId – 삭제한 각 문서의 고유 ID입니다.

예시

```
DELETE FROM VehicleRegistration AS r
WHERE r.VIN = '1HVBBAANXWH544237'
```

드라이버를 사용하여 프로그래밍 방식으로 실행

QLDB 드라이버를 사용하여 이 명령문을 프로그래밍 방식으로 실행하는 방법을 알아보려면 드라이버 시작하기에서 다음 자습서를 참조하세요.

- Java: 빠른 시작 자습서 | Cookbook 참조
- .NET: 빠른 시작 자습서 | Cookbook 참조
- Go: 빠른 시작 자습서 | Cookbook 참조
- Node.js: 빠른 시작 자습서 | Cookbook 참조

- Python: 빠른 시작 자습서 | Cookbook 참조

Amazon QLDB의 DROP INDEX 명령

Amazon QLDB에서는 `DROP INDEX` 명령을 사용하여 테이블의 인덱스를 삭제합니다.

> **Note**
>
> 특정 테이블에서 이 PartiQL 명령을 실행하기 위한 액세스를 제어하는 방법을 알아보려면 Amazon QLDB에서 표준 권한 모드로 시작하기 섹션을 참조하세요.

주제

구문

```
DROP INDEX "indexId" ON table_name WITH (purge = true)
```

> **Note**
>
> 이 `WITH (purge = true)` 절은 모든 `DROP INDEX` 명령문에 필수이며 `true`는 현재 지원되는 유일한 값입니다.
> `purge` 키워드는 대소문자를 구분하며 모두 소문자여야 합니다.

파라미터

"*indexId*"

삭제할 인덱스의 고유 ID로, 큰따옴표로 표시됩니다.

ON *table_name*

인덱스를 삭제할 테이블의 이름입니다.

반환 값

`tableId` – 인덱스를 삭제한 테이블의 고유 ID입니다.

예시

```
DROP INDEX "4tPW3fUhaVhDinRgKRLhGU" ON VehicleRegistration WITH (purge = true)
```

Amazon QLDB의 DROP TABLE 명령

Amazon QLDB에서는 `DROP TABLE` 명령을 사용하여 기존 테이블을 비활성화합니다. UNDROP TABLE 명령문을 사용하여 테이블을 다시 활성화할 수 있습니다. 테이블을 비활성화하거나 다시 활성화해도 해당 문서나 인덱스에는 영향을 주지 않습니다.

> **Note**
>
> 특정 테이블에서 이 PartiQL 명령을 실행하기 위한 액세스를 제어하는 방법을 알아보려면 Amazon QLDB에서 표준 권한 모드로 시작하기 섹션을 참조하세요.

주제

구문

```
DROP TABLE table_name
```

파라미터

table_name

비활성화할 테이블 이름입니다. 테이블이 이미 존재하고 `ACTIVE` 상태여야 합니다.

반환 값

`tableId` – 비활성화한 테이블의 고유 ID입니다.

예시

```
DROP TABLE VehicleRegistration
```

Amazon QLDB의 FROM (INSERT, REMOVE, 또는 SET) 명령

Amazon QLDB에서 `FROM`로 시작하는 문은 문서 내에 특정 요소를 삽입하고 제거할 수 있는 PartiQL 확장입니다. UPDATE 명령과 마찬가지로 이 명령문을 사용하여 문서의 기존 요소를 업데이트할 수도 있습니다.

> **Note**
>
> 특정 테이블에서 이 PartiQL 명령을 실행하기 위한 액세스를 제어하는 방법을 알아보려면 Amazon QLDB에서 표준 권한 모드로 시작하기 섹션을 참조하세요.

주제

구문

FROM-INSERT

기존 문서에 새 요소를 삽입합니다. 새 최상위 문서를 테이블에 삽입하려면 INSERT를 사용해야 합니다.

```
FROM table_name [ AS table_alias ] [ BY id_alias ]
[ WHERE condition ]
```

```
INSERT INTO element VALUE data [ AT key_name ]
```

FROM-REMOVE

문서 내의 기존 요소를 제거하거나 최상위 문서 전체를 제거합니다. 후자는 의미상 기존 DELETE 구문과 동일합니다.

```
FROM table_name [ AS table_alias ] [ BY id_alias ]
[ WHERE condition ]
REMOVE element
```

FROM-SET

문서 내 하나 이상의 요소를 업데이트합니다. 요소가 없으면 해당 요소가 삽입됩니다. 이는 기존 UPDATE 구문과 의미상 동일합니다.

```
FROM table_name [ AS table_alias ] [ BY id_alias ]
[ WHERE condition ]
SET element = data [, element = data, ... ]
```

파라미터

table_name

수정할 데이터가 포함된 사용자 테이블의 이름입니다. DML 문은 기본 사용자 뷰에서만 지원됩니다. 각 명령문은 단일 테이블에서만 실행할 수 있습니다.

이 절에는 지정된 테이블 내에 중첩된 컬렉션을 하나 이상 포함할 수도 있습니다. 자세한 내용은 중첩된 컬렉션 섹션을 참조하세요.

AS ***table_alias***

(선택 사항)수정할 테이블 전체에 적용되는 사용자 정의 별칭입니다. SET, REMOVE, INSERT INTO 또는 WHERE 절에 사용되는 모든 테이블 별칭은 FROM 절에서 선언되어야 합니다. AS 키워드는 선택 사항입니다.

BY ***id_alias***

(선택 사항)결과 집합에 있는 각 문서의 id 메타데이터 필드에 바인딩되는 사용자 정의 별칭입니다. BY 키워드를 사용하여 FROM 절에서 별칭을 선언해야 합니다. 이는 기본 사용자 뷰를 쿼리하면서 문서 ID를 기준으로 필터링하려는 경우에 유용합니다. 자세한 내용은 BY 절을 사용하여 문서 ID 쿼리하기을 참조하십시오.

WHERE *condition*

수정할 문서의 선택 기준입니다.

> **Note**
>
> WHERE 절을 생략하면 테이블의 모든 문서가 수정됩니다.

element

생성하거나 수정할 문서 요소입니다.

data

요소의 새 값입니다.

AT *key_name*

수정할 문서에 추가할 키 이름입니다. 키 이름과 함께 해당하는 VALUE을 지정해야 합니다. 이는 문서 내 특정 위치에 새 값 AT을 삽입하는 데 필요합니다.

중첩된 컬렉션

단일 테이블에서만 DML 문을 실행할 수 있지만 해당 테이블의 문서 내에 있는 중첩된 컬렉션을 추가 소스로 지정할 수 있습니다. 중첩 컬렉션에 대해 선언한 각 별칭은 WHERE 절과 SET, INSERT INTO 또는 REMOVE 절에서 사용할 수 있습니다.

예를 들어, 다음 명령문의 FROM 소스에는 VehicleRegistration 테이블과 중첩된 Owners.SecondaryOwners 구조가 모두 포함됩니다.

```
FROM VehicleRegistration r, @r.Owners.SecondaryOwners o
WHERE r.VIN = '1N4AL11D75C109151' AND o.PersonId = 'abc123'
SET o.PersonId = 'def456'
```

이 예제는 '1N4AL11D75C109151'의 VIN가 있는 VehicleRegistration 문서 내에서 'abc123'의 PersonId가 있는 SecondaryOwners 목록의 특정 요소를 업데이트합니다. 이 표현식을 사용하면 인덱스가 아닌 값을 기준으로 목록의 요소를 지정할 수 있습니다.

반환 값

documentId – 업데이트 또는 삭제한 각 문서의 고유 ID입니다.

예시

문서 내 요소를 수정합니다. 요소가 존재하지 않는 경우 해당 요소가 삽입됩니다.

```
FROM Vehicle AS v
WHERE v.VIN = '1N4AL11D75C109151' AND v.Color = 'Silver'
SET v.Color = 'Shiny Gray'
```

시스템에서 할당한 문서 `id` 메타데이터 필드에 요소와 필터를 수정하거나 삽입합니다.

```
FROM Vehicle AS v BY v_id
WHERE v_id = 'documentId'
SET v.Color = 'Shiny Gray'
```

문서 내 `Owners.SecondaryOwners` 목록에 있는 첫 번째 요소의 `PersonId` 필드를 수정합니다.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
SET r.Owners.SecondaryOwners[0].PersonId = 'abc123'
```

문서 내 기존 요소를 제거합니다.

```
FROM Person AS p
WHERE p.GovId = '111-22-3333'
REMOVE p.Address
```

테이블에서 전체 문서를 제거합니다.

```
FROM Person AS p
WHERE p.GovId = '111-22-3333'
REMOVE p
```

`VehicleRegistration` 테이블의 문서 내 `Owners.SecondaryOwners` 목록의 첫 번째 요소를 제거합니다.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
REMOVE r.Owners.SecondaryOwners[0]
```

`{'Mileage':26500}`를 `Vehicle` 테이블의 문서 내에 최상위 이름-값 쌍으로 삽입합니다.

```
FROM Vehicle AS v
WHERE v.VIN = '1N4AL11D75C109151'
INSERT INTO v VALUE 26500 AT 'Mileage'
```

`{'PersonId':'abc123'}`를 `VehicleRegistration` 테이블의 문서 `Owners.SecondaryOwners` 필드에 이름-값 쌍으로 추가합니다. 이 명령문이 유효하려면 `Owners.SecondaryOwners`이 이미 존재하고 목록 데이터 유형이어야 합니다. 그렇지 않으면 `INSERT INTO` 절에 `AT` 키워드가 필요합니다.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners VALUE { 'PersonId' : 'abc123' }
```

`{'PersonId':'abc123'}`를 문서 내 기존 `Owners.SecondaryOwners` 목록의 첫 번째 요소로 삽입합니다.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners VALUE {'PersonId' : 'abc123'} AT 0
```

문서 내 기존 `Owners.SecondaryOwners` 목록에 여러 개의 이름-값 쌍을 추가합니다.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners << {'PersonId' : 'abc123'}, {'PersonId' :
 'def456'} >>
```

드라이버를 사용하여 프로그래밍 방식으로 실행

QLDB 드라이버를 사용하여 이 명령문을 프로그래밍 방식으로 실행하는 방법을 알아보려면 드라이버 시작하기에서 다음 자습서를 참조하세요.

- Java: 빠른 시작 자습서 | Cookbook 참조
- .NET: 빠른 시작 자습서 | Cookbook 참조
- Go: 빠른 시작 자습서 | Cookbook 참조
- Node.js: 빠른 시작 자습서 | Cookbook 참조
- Python: 빠른 시작 자습서 | Cookbook 참조

Amazon QLDB의 INSERT 명령

Amazon QLDB에서는 `INSERT` 명령을 사용하여 하나 이상의 Amazon Ion 문서를 테이블에 추가합니다.

> **Note**
>
> 특정 테이블에서 이 PartiQL 명령을 실행하기 위한 액세스를 제어하는 방법을 알아보려면 Amazon QLDB에서 표준 권한 모드로 시작하기 섹션을 참조하세요.

주제

구문

단일 문서를 삽입합니다.

```
INSERT INTO table_name VALUE document
```

여러 문서를 삽입합니다.

```
INSERT INTO table_name << document, document, ... >>
```

파라미터

table_name

데이터를 삽입할 사용자 테이블의 이름입니다. 이미 있는 테이블이어야 합니다. DML 문은 기본 사용자 뷰에서만 지원됩니다.

document

유효한 QLDB 문서입니다. 문서를 최소 하나 이상 지정해야 합니다. 여러 문서는 쉼표로 구분해야 합니다.

문서는 중괄호(`{...}`)로 표시해야 합니다.

문서의 각 필드 이름은 PartiQL에서 작은 따옴표(`'...'`)로 표시할 수 있는 대소문자를 구분하는 Ion 기호입니다.

문자열 값도 PartiQL에서 작은 따옴표(`'...'`)로 표시될 수 있습니다.

모든 Ion 리터럴은 백틱(`` `...` ``)으로 표시할 수 있습니다.

> **Note**
>
> 이중 꺾쇠 괄호(`<<...>>`)는 정렬되지 않은 컬렉션(PartiQL에서는 백이라고 함)을 나타내며 여러 문서를 삽입하려는 경우에만 필요합니다.

반환 값

`documentId` – 삽입한 각 문서의 고유 ID입니다.

예시

단일 문서를 삽입합니다.

```
INSERT INTO VehicleRegistration VALUE
{
    'VIN' : 'KM8SRDHF6EU074761', --string
    'RegNum' : 1722, --integer
    'State' : 'WA',
    'City' : 'Kent',
    'PendingPenaltyTicketAmount' : 130.75, --decimal
    'Owners' : { --nested struct
        'PrimaryOwner' : { 'PersonId': '294jJ3YUoH1IEEm8GSabOs' },
        'SecondaryOwners' : [ --list of structs
            { 'PersonId' : '1nmeDdLo3AhGswBtyM1eYh' },
            { 'PersonId': 'IN7MvYtUjkp1GMZu0F6CG9' }
        ]
    },
    'ValidFromDate' : `2017-09-14T`, --Ion timestamp literal with day precision
    'ValidToDate' : `2020-06-25T`
}
```

이 명령문은 삽입한 문서의 고유 ID를 다음과 같이 반환합니다.

```
{
    documentId: "2kKuOPNB07D2iTPBrUTWGl"
}
```

여러 문서를 삽입합니다.

```
INSERT INTO Person <<
{
    'FirstName' : 'Raul',
    'LastName' : 'Lewis',
    'DOB' : `1963-08-19T`,
    'GovId' : 'LEWISR261LL',
    'GovIdType' : 'Driver License',
    'Address' : '1719 University Street, Seattle, WA, 98109'
},
{
    'FirstName' : 'Brent',
    'LastName' : 'Logan',
    'DOB' : `1967-07-03T`,
    'GovId' : 'LOGANB486CG',
    'GovIdType' : 'Driver License',
    'Address' : '43 Stockert Hollow Road, Everett, WA, 98203'
},
{
    'FirstName' : 'Alexis',
    'LastName' : 'Pena',
    'DOB' : `1974-02-10T`,
    'GovId' : '744 849 301',
    'GovIdType' : 'SSN',
    'Address' : '4058 Melrose Street, Spokane Valley, WA, 99206'
}
>>
```

이 명령문은 삽입한 각 문서의 고유 ID를 다음과 같이 반환합니다.

```
{
    documentId: "6WXzLscsJ3bDWW97Dy8nyp"
},
{
    documentId: "35e0ToZyTGJ7LGvcwrkX65"
},
```

```
{
    documentId: "BVHPcH612o7JROQ4yP8jiH"
}
```

드라이버를 사용하여 프로그래밍 방식으로 실행

QLDB 드라이버를 사용하여 이 명령문을 프로그래밍 방식으로 실행하는 방법을 알아보려면 드라이버 시작하기에서 다음 자습서를 참조하세요.

- Java: 빠른 시작 자습서 | Cookbook 참조
- .NET: 빠른 시작 자습서 | Cookbook 참조
- Go: 빠른 시작 자습서 | Cookbook 참조
- Node.js: 빠른 시작 자습서 | Cookbook 참조
- Python: 빠른 시작 자습서 | Cookbook 참조

Amazon QLDB의 SELECT 명령

Amazon QLDB에서는 `SELECT` 명령을 사용하여 하나 이상의 테이블에서 데이터를 검색합니다. QLDB의 모든 `SELECT` 쿼리는 트랜잭션에서 처리되며 트랜잭션 시간 초과 제한이 적용됩니다.

결과 순서는 구체적이지 않으며 각 `SELECT` 쿼리마다 다를 수 있습니다. QLDB의 모든 쿼리에 대해 결과 순서에 의존해서는 안 됩니다.

특정 테이블에서 이 PartiQL 명령을 실행하기 위한 액세스를 제어하는 방법을 알아보려면 Amazon QLDB에서 표준 권한 모드로 시작하기 섹션을 참조하세요.

> **Warning**
>
> 인덱싱된 조회 없이 QLDB에서 쿼리를 실행하면 전체 테이블 스캔이 호출됩니다. PartiQL은 SQL과 호환되므로 이러한 쿼리를 지원합니다. 하지만 QLDB의 프로덕션 사용 사례에 대해서는 테이블 스캔을 실행하지 마세요.. 테이블 스캔은 동시성 충돌 및 트랜잭션 시간 초과를 포함하여 대규모 테이블에서 성능 문제를 일으킬 수 있습니다.
>
> 인덱싱된 필드 또는 문서 ID(예: `WHERE indexedField = 123` 또는 `WHERE indexedField IN (456, 789)`)에서 동등 연산자를 사용하여 `WHERE` 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 자세한 내용은 쿼리 성능 최적화을 참조하십시오.

주제

구문

```
SELECT [ VALUE ] expression [ AS field_alias ] [, expression, ... ]
FROM source [ AS source_alias ] [ AT idx_alias ] [ BY id_alias ] [, source, ... ]
[ WHERE condition ]
```

파라미터

VALUE

값을 튜플 구조로 래핑하는 대신 쿼리에서 원시 데이터 유형 값을 반환하도록 하는 표현식의 한정자입니다.

###

* 와일드카드에서 형성된 프로젝션 또는 결과 집합에 있는 하나 이상의 필드로 구성된 프로젝션 목록입니다. 표현은 PartiQL 함수에 대한 호출 또는 PartiQL 연산자에서 수정되는 필드로 구성될 수 있습니다.

AS ***field_alias***

(선택 사항)최종 결과 집합에 사용되는 필드의 임시 사용자 정의 별칭입니다. AS 키워드는 선택 사항입니다.

단순 필드 이름이 아닌 표현식에 별칭을 지정하지 않으면 결과 집합은 해당 필드에 기본 이름을 적용합니다.

FROM ***source***

쿼리할 소스입니다. 현재 지원되는 소스는 테이블 이름, 테이블 간 내부 조인, 중첩된 SELECT 쿼리(중첩된 쿼리 한도에 적용), 테이블에 대한 기록 함수 호출 뿐입니다.

소스를 최소 하나 이상 지정해야 합니다. 여러 소스는 쉼표로 구분해야 합니다.

AS *source_alias*

(선택 사항)쿼리할 소스의 범위를 지정하는 사용자 정의 별칭입니다. SELECT 또는 WHERE 절에 사용되는 모든 소스 별칭은 FROM 절에 선언되어야 합니다. AS 키워드는 선택 사항입니다.

AT *idx_alias*

(선택 사항)소스 목록 내 각 요소의 인덱스(서수) 번호에 바인딩되는 사용자 정의 별칭입니다. AT 키워드를 사용하여 FROM 절에서 별칭을 선언해야 합니다.

BY *id_alias*

(선택 사항)결과 집합에 있는 각 문서의 id 메타데이터 필드에 바인딩되는 사용자 정의 별칭입니다. BY 키워드를 사용하여 FROM 절에서 별칭을 선언해야 합니다. 이는 기본 사용자 뷰를 쿼리하면서 문서 ID를 기반으로 프로젝션하거나 필터링하려는 경우에 유용합니다. 자세한 내용은 BY 절을 사용하여 문서 ID 쿼리하기을 참조하십시오.

WHERE *condition*

쿼리의 선택 기준 및 조인 기준(해당하는 경우)입니다.

> **Note**
>
> WHERE 절을 생략하면 테이블의 모든 문서가 검색됩니다.

조인

현재 내부 조인만 지원됩니다. 다음과 같이 명시적 INNER JOIN 절을 사용하여 내부 조인 쿼리를 작성할 수 있습니다. 이 구문에서는 JOIN와 ON가 쌍을 이루어야 하며 INNER 키워드는 선택 사항입니다.

```
SELECT expression
FROM table1 AS t1 [ INNER ] JOIN table2 AS t2
ON t1.element = t2.element
```

또는 다음과 같이 암시적 구문을 사용하여 내부 조인을 작성할 수 있습니다.

```
SELECT expression
FROM table1 AS t1, table2 AS t2
WHERE t1.element = t2.element
```

중첩된 쿼리 한도

SELECT 표현식과 FROM 소스 내에서 중첩된 쿼리(하위 쿼리)를 작성할 수 있습니다. 주요 제한 사항은 가장 바깥쪽 쿼리만 글로벌 데이터베이스 환경에 액세스할 수 있다는 것입니다. 예를 들어, VehicleRegistration 및 Person 테이블이 있는 원장이 있다고 가정해 보겠습니다. 다음 중첩 쿼리는 내부 SELECT가 Person에 액세스를 시도하므로 유효하지 않습니다.

```
SELECT r.VIN,
    (SELECT p.PersonId FROM Person AS p WHERE p.PersonId =
 r.Owners.PrimaryOwner.PersonId) AS PrimaryOwner
FROM VehicleRegistration AS r
```

반면 다음 중첩 쿼리는 유효합니다.

```
SELECT r.VIN, (SELECT o.PrimaryOwner.PersonId FROM @r.Owners AS o) AS PrimaryOwner
FROM VehicleRegistration AS r
```

예시

다음 쿼리는 IN 연산자를 사용하는 표준 WHERE 술어 절이 포함된 기본 SELECT 전체 와일드카드를 보여줍니다.

```
SELECT * FROM Vehicle
WHERE VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

다음은 문자열 필터를 사용한 SELECT 프로젝션을 보여줍니다.

```
SELECT FirstName, LastName, Address
FROM Person
WHERE Address LIKE '%Seattle%'
AND GovId = 'LEWISR261LL'
```

다음은 중첩된 데이터를 평면화하는 상관 하위 쿼리를 보여줍니다. 참고로 이 @ 문자는 기술적으로 선택 사항입니다. 하지만 이 표현은 Owners(존재하는 경우)이라는 이름의 다른 컬렉션이 아니라 VehicleRegistration 내에 중첩된 Owners 구조를 적용하고 싶다는 것을 명시적으로 나타냅니다. 자세한 내용은 데이터 및 기록 다루기 장의 중첩된 데이터 섹션을 참조하세요.

```
SELECT
    r.VIN,
    o.SecondaryOwners
```

```
FROM
    VehicleRegistration AS r, @r.Owners AS o
WHERE
    r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

다음은 중첩된 데이터를 프로젝션하는 SELECT 목록의 하위 쿼리와 암시적 내부 조인을 보여줍니다.

```
SELECT
    v.Make,
    v.Model,
    (SELECT VALUE o.PrimaryOwner.PersonId FROM @r.Owners AS o) AS PrimaryOwner
FROM
    VehicleRegistration AS r, Vehicle AS v
WHERE
    r.VIN = v.VIN AND r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

다음은 명시적 내부 조인을 보여줍니다.

```
SELECT
    v.Make,
    v.Model,
    r.Owners
FROM
    VehicleRegistration AS r JOIN Vehicle AS v
ON
    r.VIN = v.VIN
WHERE
    r.VIN IN ('1N4AL11D75C109151', 'KM8SRDHF6EU074761')
```

다음은 BY 절을 사용한 문서 id 메타데이터 필드의 프로젝션을 보여줍니다.

```
SELECT
    r_id,
    r.VIN
FROM
    VehicleRegistration AS r BY r_id
WHERE
    r_id = 'documentId'
```

다음은 BY 절을 사용하여 PersonId 및 문서 id 필드의 DriversLicense 및 Person 테이블에 각각 조인합니다.

```
SELECT * FROM DriversLicense AS d INNER JOIN Person AS p BY pid
ON d.PersonId = pid
WHERE pid = 'documentId'
```

다음은 커밋된 뷰을 사용하여 PersonId 및 문서 id 필드의 DriversLicense 및 Person 테이블에 각각 조인합니다.

```
SELECT * FROM DriversLicense AS d INNER JOIN _ql_committed_Person AS cp
ON d.PersonId = cp.metadata.id
WHERE cp.metadata.id = 'documentId'
```

다음은 테이블 VehicleRegistration 내 문서에 대한 Owners.SecondaryOwners 목록에 있는 각 사람의 PersonId 및 인덱스(서수)를 반환합니다.

```
SELECT s.PersonId, owner_idx
FROM VehicleRegistration AS r, @r.Owners.SecondaryOwners AS s AT owner_idx
WHERE r.VIN = 'KM8SRDHF6EU074761'
```

드라이버를 사용하여 프로그래밍 방식으로 실행

QLDB 드라이버를 사용하여 이 명령문을 프로그래밍 방식으로 실행하는 방법을 알아보려면 드라이버 시작하기에서 다음 자습서를 참조하세요.

- Java: 빠른 시작 자습서 | Cookbook 참조
- .NET: 빠른 시작 자습서 | Cookbook 참조
- Go: 빠른 시작 자습서 | Cookbook 참조
- Node.js: 빠른 시작 자습서 | Cookbook 참조
- Python: 빠른 시작 자습서 | Cookbook 참조

Amazon QLDB의 UPDATE 명령

Amazon QLDB에서는 UPDATE 명령을 사용하여 문서 내 하나 이상의 요소의 값을 수정합니다. 요소가 없으면 해당 요소가 삽입됩니다.

FROM (INSERT, REMOVE, 또는 SET) 명령문처럼 이 명령을 사용하여 문서 내에 특정 요소를 명시적으로 삽입하고 제거할 수도 있습니다.

> **Note**
>
> 특정 테이블에서 이 PartiQL 명령을 실행하기 위한 액세스를 제어하는 방법을 알아보려면 Amazon QLDB에서 표준 권한 모드로 시작하기 섹션을 참조하세요.

주제

구문

UPDATE-SET

문서 내 하나 이상의 요소를 업데이트합니다. 요소가 없으면 해당 요소가 삽입됩니다. 이는 FROM-SET 문과 의미상 동일합니다.

```
UPDATE table_name [ AS table_alias ] [ BY id_alias ]
SET element = data [, element = data, ... ]
[ WHERE condition ]
```

UPDATE-INSERT

기존 문서에 새 요소를 삽입합니다. 새 최상위 문서를 테이블에 삽입하려면 INSERT를 사용해야 합니다.

```
UPDATE table_name [ AS table_alias ] [ BY id_alias ]
INSERT INTO element VALUE data [ AT key_name ]
[ WHERE condition ]
```

UPDATE-REMOVE

문서 내의 기존 요소를 제거하거나 최상위 문서 전체를 제거합니다. 후자는 의미상 기존 DELETE 구문과 동일합니다.

```
UPDATE table_name [ AS table_alias ] [ BY id_alias ]
REMOVE element
[ WHERE condition ]
```

파라미터

table_name

수정할 데이터가 포함된 사용자 테이블의 이름입니다. DML 문은 기본 사용자 뷰에서만 지원됩니다. 각 명령문은 단일 테이블에서만 실행할 수 있습니다.

AS ***table_alias***

(선택 사항)업데이트할 테이블에 속하는 사용자 정의 별칭입니다. AS 키워드는 선택 사항입니다.

BY ***id_alias***

(선택 사항)결과 집합에 있는 각 문서의 `id` 메타데이터 필드에 바인딩되는 사용자 정의 별칭입니다. BY 키워드를 사용하여 UPDATE 절에서 별칭을 선언해야 합니다. 이는 기본 사용자 뷰를 쿼리하면서 문서 ID를 기준으로 필터링하려는 경우에 유용합니다. 자세한 내용은 BY 절을 사용하여 문서 ID 쿼리하기을 참조하십시오.

element

생성하거나 수정할 문서 요소입니다.

data

요소의 새 값입니다.

AT ***key_name***

수정할 문서에 추가할 키 이름입니다. 키 이름과 함께 해당하는 VALUE을 지정해야 합니다. 이는 문서 내 특정 위치에 새 값 AT을 삽입하는 데 필요합니다.

WHERE ***condition***

수정할 문서의 선택 기준입니다.

> **Note**
>
> WHERE 절을 생략하면 테이블의 모든 문서가 수정됩니다.

반환 값

`documentId` – 업데이트한 각 문서의 고유 ID입니다.

예시

문서의 필드를 업데이트합니다. 필드가 존재하지 않는 경우 해당 필드가 삽입됩니다.

```
UPDATE Person AS p
SET p.LicenseNumber = 'HOLLOR123ZZ'
WHERE p.GovId = '111-22-3333'
```

시스템에서 할당한 문서 `id` 메타데이터 필드를 기준으로 필터링합니다.

```
UPDATE Person AS p BY pid
SET p.LicenseNumber = 'HOLLOR123ZZ'
WHERE pid = 'documentId'
```

문서 전체를 덮어씁니다.

```
UPDATE Person AS p
SET p = {
    'FirstName' : 'Rosemarie',
    'LastName' : 'Holloway',
    'DOB' : `1977-06-18T`,
    'GovId' : '111-22-3333',
    'GovIdType' : 'Driver License',
    'Address' : '4637 Melrose Street, Ellensburg, WA, 98926'
}
WHERE p.GovId = '111-22-3333'
```

문서 내 `Owners.SecondaryOwners` 목록에 있는 첫 번째 요소의 `PersonId` 필드를 수정합니다.

```
UPDATE VehicleRegistration AS r
SET r.Owners.SecondaryOwners[0].PersonId = 'abc123'
WHERE r.VIN = '1N4AL11D75C109151'
```

`{'Mileage':26500}`를 `Vehicle` 테이블의 문서 내에 최상위 이름-값 쌍으로 삽입합니다.

```
UPDATE Vehicle AS v
INSERT INTO v VALUE 26500 AT 'Mileage'
```

```
WHERE v.VIN = '1N4AL11D75C109151'
```

{'PersonId':'abc123'}를 VehicleRegistration 테이블의 문서 Owners.SecondaryOwners 필드에 이름-값 쌍으로 추가합니다. 이 명령문이 유효하려면 Owners.SecondaryOwners이 이미 존재하고 목록 데이터 유형이어야 합니다. 그렇지 않으면 INSERT INTO 절에 AT 키워드가 필요합니다.

```
UPDATE VehicleRegistration AS r
INSERT INTO r.Owners.SecondaryOwners VALUE { 'PersonId' : 'abc123' }
WHERE r.VIN = '1N4AL11D75C109151'
```

{'PersonId':'abc123'}를 문서 내 기존 Owners.SecondaryOwners 목록의 첫 번째 요소로 삽입합니다.

```
UPDATE VehicleRegistration AS r
INSERT INTO r.Owners.SecondaryOwners VALUE {'PersonId' : 'abc123'} AT 0
WHERE r.VIN = '1N4AL11D75C109151'
```

문서 내 기존 Owners.SecondaryOwners 목록에 여러 개의 이름-값 쌍을 추가합니다.

```
UPDATE VehicleRegistration AS r
INSERT INTO r.Owners.SecondaryOwners << {'PersonId' : 'abc123'}, {'PersonId' :
 'def456'} >>
WHERE r.VIN = '1N4AL11D75C109151'
```

문서 내 기존 요소를 제거합니다.

```
UPDATE Person AS p
REMOVE p.Address
WHERE p.GovId = '111-22-3333'
```

테이블에서 전체 문서를 제거합니다.

```
UPDATE Person AS p
REMOVE p
WHERE p.GovId = '111-22-3333'
```

VehicleRegistration 테이블의 문서 내 Owners.SecondaryOwners 목록의 첫 번째 요소를 제거합니다.

```
UPDATE VehicleRegistration AS r
REMOVE r.Owners.SecondaryOwners[0]
WHERE r.VIN = '1N4AL11D75C109151'
```

드라이버를 사용하여 프로그래밍 방식으로 실행

QLDB 드라이버를 사용하여 이 명령문을 프로그래밍 방식으로 실행하는 방법을 알아보려면 드라이버 시작하기에서 다음 자습서를 참조하세요.

- Java: 빠른 시작 자습서 | Cookbook 참조
- .NET: 빠른 시작 자습서 | Cookbook 참조
- Go: 빠른 시작 자습서 | Cookbook 참조
- Node.js: 빠른 시작 자습서 | Cookbook 참조
- Python: 빠른 시작 자습서 | Cookbook 참조

Amazon QLDB의 UNDROP TABLE 명령

Amazon QLDB에서 `UNDROP TABLE` 명령을 사용하여 이전에 삭제한(즉, 비활성화된) 테이블을 다시 활성화합니다. 테이블을 비활성화하거나 다시 활성화해도 해당 문서나 인덱스에는 영향을 주지 않습니다.

> **Note**
>
> 특정 테이블에서 이 PartiQL 명령을 실행하기 위한 액세스를 제어하는 방법을 알아보려면 Amazon QLDB에서 표준 권한 모드로 시작하기 섹션을 참조하세요.

주제

- 구문
- 파라미터
- 반환 값
- 예시

구문

```
UNDROP TABLE "tableId"
```

파라미터

"*tableId*"

다시 활성화할 테이블의 고유 ID로, 큰따옴표로 표시됩니다.

테이블은 이전에 삭제된 적이 있어야 합니다. 즉, 해당 테이블은 시스템 카탈로그 테이블 `information_schema.user_tables`에 존재하며 `INACTIVE` 상태여야 합니다. 또한 동일한 이름을 가진 활성 기존 테이블이 없어야 합니다.

반환 값

`tableId` – 재활성화한 테이블의 고유 ID입니다.

예시

```
UNDROP TABLE "5PLf9SXwndd63lPaSIa0O6"
```

Amazon QLDB의 PartiQL 함수

Amazon QLDB의 PartiQL은 SQL 표준 함수의 다음과 같은 기본 제공 변형을 지원합니다.

> **Note**
>
> 이 목록에 포함되지 않은 SQL 함수는 현재 QLDB에 지원되지 않습니다.
> 이 함수 참조는 PartiQL 설명서 기본 제공 함수를 기반으로 합니다.

알 수 없는 타입(null 및 누락) 전파

달리 명시되지 않는 한, 이러한 함수는 모두 null 및 누락된 인수 값을 전파합니다. `NULL` 또는 `MISSING`의 전파는 함수 인수가 `NULL` 또는 `MISSING` 중 하나일 경우 반환하는 `NULL`로 정의됩니다. 다음은 이러한 전파의 예입니다.

```
CHAR_LENGTH(null)    -- null
CHAR_LENGTH(missing) -- null (also returns null)
```

집계 함수

- AVG
- COUNT
- MAX
- MIN
- SIZE
- SUM

조건 함수

- COALESCE
- exists
- nullIF

날짜 및 시간 함수

- DATE_ADD
- DATE_DIFF
- EXTRACT
- UTCNOW

스칼라 함수

- TXID

문자열 함수

- CHAR_LENGTH
- CHARACTER_LENGTH

- LOWER
- SUBSTRING
- TRIM
- UPPER

데이터 형식 지정 함수

- CAST
- TO_STRING
- TO_TIMESTAMP

Amazon QLDB의 AVG 함수

Amazon QLDB에서는 AVG 함수를 사용해 입력 표현식 값의 평균(산술 평균)을 반환합니다. 이 함수는 수치를 사용하고 null 또는 누락 값은 무시합니다.

조건

```
AVG ( expression )
```

인수

###

함수가 실행되는 숫자 데이터 타입의 필드 명칭 또는 표현식입니다.

데이터 타입

지원되는 인수 타입:

- `int`
- `decimal`
- `float`

반환 타입: `decimal`

예

```
SELECT AVG(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r  -- 147.19
SELECT AVG(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >>         -- 2.
```

관련 함수

- COUNT
- MAX
- MIN
- SIZE
- SUM

Amazon QLDB의 CAST 함수

Amazon QLDB에서는 CAST 함수를 사용하여 주어진 표현식을 값으로 평가하고 값을 지정된 대상 데이터 타입으로 변환합니다. 변환할 수 없는 경우 함수는 오류를 반환합니다.

조건

```
CAST ( expression AS type )
```

인수

###

함수가 변환하는 값으로 평가되는 파일 명칭 또는 표현식입니다. null 값을 변환하면 마찬가지로 null이 반환됩니다. 이 파라미터는 지원되는 데이터 유형 중 하나일 수 있습니다.

##

변환할 대상 데이터 타입의 명칭입니다. 이 파라미터는 지원되는 데이터 유형 중 하나일 수 있습니다.

반환 타입

type 인수에 의해 지정된 데이터 타입.

예

다음 예는 알 수 없는 타입(NULL 또는MISSING)의 전파를 보여줍니다.

```
CAST(null    AS null)    -- null
CAST(missing AS null)    -- null
CAST(missing AS missing) -- missing
CAST(null    AS missing) -- missing
CAST(null    AS boolean) -- null (null AS any data type name results in null)
CAST(missing AS boolean) -- missing (missing AS any data type name results in missing)
```

알 수 없는 형식이 아닌 값은 NULL 또는 MISSING로 캐스팅할 수 없습니다.

```
CAST(true AS null)    -- error
CAST(true AS missing) -- error
CAST(1    AS null)    -- error
CAST(1    AS missing) -- error
```

다음 예는 캐스팅 AS boolean을 보여줍니다.

```
CAST(true      AS boolean) -- true no-op
CAST(0         AS boolean) -- false
CAST(1         AS boolean) -- true
CAST(`1e0`     AS boolean) -- true (float)
CAST(`1d0`     AS boolean) -- true (decimal)
CAST('a'       AS boolean) -- false
CAST('true'    AS boolean) -- true (SqlName string 'true')
CAST(`'true'`  AS boolean) -- true (Ion symbol `'true'`)
CAST(`'false'` AS boolean) -- false (Ion symbol `'false'`)
```

다음 예는 캐스팅 AS integer을 보여줍니다.

```
CAST(true   AS integer) -- 1
CAST(false  AS integer) -- 0
CAST(1      AS integer) -- 1
CAST(`1d0`  AS integer) -- 1
CAST(`1d3`  AS integer) -- 1000
CAST(1.00   AS integer) -- 1
CAST(1.45   AS integer) -- 1
CAST(1.75   AS integer) -- 1
CAST('12'   AS integer) -- 12
CAST('aa'   AS integer) -- error
```

```
CAST(``'22'`` AS integer) -- 22
CAST(``'x'``  AS integer) -- error
```

다음 예는 캐스팅 AS float을 보여줍니다.

```
CAST(true   AS float) -- 1e0
CAST(false  AS float) -- 0e0
CAST(1      AS float) -- 1e0
CAST(`1d0`  AS float) -- 1e0
CAST(`1d3`  AS float) -- 1000e0
CAST(1.00   AS float) -- 1e0
CAST('12'   AS float) -- 12e0
CAST('aa'   AS float) -- error
CAST(``'22'`` AS float) -- 22e0
CAST(``'x'``  AS float) -- error
```

다음 예는 캐스팅 AS decimal을 보여줍니다.

```
CAST(true   AS decimal) -- 1.
CAST(false  AS decimal) -- 0.
CAST(1      AS decimal) -- 1.
CAST(`1d0`  AS decimal) -- 1. (REPL printer serialized to 1.)
CAST(`1d3`  AS decimal) -- 1d3
CAST(1.00   AS decimal) -- 1.00
CAST('12'   AS decimal) -- 12.
CAST('aa'   AS decimal) -- error
CAST(``'22'`` AS decimal) -- 22.
CAST(``'x'``  AS decimal) -- error
```

다음 예는 캐스팅 AS timestamp을 보여줍니다.

```
CAST(`2001T`                      AS timestamp) -- 2001T
CAST('2001-01-01T'                AS timestamp) -- 2001-01-01T
CAST(``'2010-01-01T00:00:00.000Z'`` AS timestamp) -- 2010-01-01T00:00:00.000Z
CAST(true                         AS timestamp) -- error
CAST(2001                         AS timestamp) -- error
```

다음 예는 캐스팅 AS symbol을 보여줍니다.

```
CAST(``'xx'``                    AS symbol) -- xx (``'xx'`` is an Ion symbol)
CAST('xx'                        AS symbol) -- xx ('xx' is a string)
CAST(42                          AS symbol) -- '42'
```

```
CAST(`1e0`                      AS symbol) -- '1.0'
CAST(`1d0`                      AS symbol) -- '1'
CAST(true                       AS symbol) -- 'true'
CAST(false                      AS symbol) -- 'false'
CAST(`2001T`                    AS symbol) -- '2001T'
CAST(`2001-01-01T00:00:00.000Z` AS symbol) -- '2001-01-01T00:00:00.000Z'
```

다음 예는 캐스팅 `AS string`을 보여줍니다.

```
CAST(``'xx'``                   AS string) -- "xx" (``'xx'`` is an Ion symbol)
CAST('xx'                       AS string) -- "xx" ('xx' is a string)
CAST(42                         AS string) -- "42"
CAST(`1e0`                      AS string) -- "1.0"
CAST(`1d0`                      AS string) -- "1"
CAST(true                       AS string) -- "true"
CAST(false                      AS string) -- "false"
CAST(`2001T`                    AS string) -- "2001T"
CAST(`2001-01-01T00:00:00.000Z` AS string) -- "2001-01-01T00:00:00.000Z"
```

다음 예는 캐스팅 `AS struct`을 보여줍니다.

```
CAST(`{ a: 1 }` AS struct) -- {a:1}
CAST(true       AS struct) -- err
```

다음 예는 캐스팅 `AS list`을 보여줍니다.

```
CAST(`[1, 2, 3]`        AS list) -- [1,2,3]
CAST(<<'a', { 'b':2 }>> AS list) -- ["a",{'b':2}]
CAST({ 'b':2 }          AS list) -- error
```

다음 예는 이전 예 중 일부를 포함하는 실행 가능한 명령문입니다.

```
SELECT CAST(true AS integer) FROM << 0 >>            -- 1
SELECT CAST('2001-01-01T' AS timestamp) FROM << 0 >> -- 2001-01-01T
SELECT CAST('xx' AS symbol) FROM << 0 >>             -- xx
SELECT CAST(42 AS string) FROM << 0 >>               -- "42"
```

관련 함수

- TO_STRING
- TO_TIMESTAMP

Amazon QLDB의 CHAR_LENGTH 함수

Amazon QLDB에서는 CHAR_LENGTH 함수를 사용하여 지정된 문자열의 문자 수를 반환합니다. 여기서 문자는 단일 유니코드 코드 포인트로 정의됩니다.

조건

```
CHAR_LENGTH ( string )
```

CHAR_LENGTH는 Amazon QLDB의 CHARACTER_LENGTH 함수의 동의어입니다.

인수

string

함수가 평가하는 데이터 타입 string의 필드 명칭 또는 표현식입니다.

반환 타입

int

예

```
SELECT CHAR_LENGTH('') FROM << 0 >>        -- 0
SELECT CHAR_LENGTH('abcdefg') FROM << 0 >> -- 7
SELECT CHAR_LENGTH('e#') FROM << 0 >>      -- 2 (because 'e#' is two code points: the
 letter 'e' and combining character U+032B)
```

관련 함수

- LOWER
- SUBSTRING
- TRIM
- UPPER

Amazon QLDB의 CHARACTER_LENGTH 함수

CHAR_LENGTH 함수의 동의어입니다.

Amazon QLDB의 CHAR_LENGTH 함수 섹션을 참조하세요.

Amazon QLDB의 COALESCE 함수

Amazon QLDB에서 하나 이상의 인수 목록이 주어지면 COALESCE 함수를 사용하여 왼쪽에서 오른쪽 순으로 인수를 평가하고 알 수 없는 타입(NULL 또는MISSING)이 아닌 첫 번째 값을 반환합니다. 모든 인수 타입을 알 수 없는 경우 결과는 NULL입니다.

COALESCE 함수는 NULL 및 MISSING을 전파하지 않습니다.

조건

```
COALESCE ( expression [, expression, ... ] )
```

인수

###

함수가 평가하는 하나 이상의 필드 명칭 또는 표현식 목록입니다. 각 인수는 지원되는 데이터 유형 중 하나일 수 있습니다.

반환 타입

지원되는 모든 데이터 타입입니다. 반환 타입은 NULL이거낙 null이 아니고 누락되지 않은 값으로 평가되는 첫 번째 표현식의 타입과 동일합니다.

예

```
SELECT COALESCE(1, null) FROM << 0 >>          -- 1
SELECT COALESCE(null, null, 1) FROM << 0 >>    -- 1
SELECT COALESCE(null, 'string') FROM << 0 >>   -- "string"
```

관련 함수

- exists
- nullIF

Amazon QLDB의 COUNT 함수

Amazon QLDB에서는 COUNT 함수를 사용하여 주어진 표현식으로 정의된 문서 수를 반환합니다. 이 함수는 2가지 변형이 있습니다:

- COUNT(*) - null 또는 누락 값의 포함 여부에 관계없이 대상 표의 모든 문서를 계수합니다.
- COUNT(expression) - 특정 기존 필드 또는 표현식에서 null이 아닌 값을 가진 문서 수를 계수합니다.

> **Warning**
>
> COUNT 함수는 최적화되지 않았으므로 인덱싱된 조회 없이는 사용하지 않는 것이 좋습니다. 인덱싱된 조회 없이 QLDB에서 쿼리를 실행하면 전체 테이블 스캔이 호출됩니다. 이로 인해 동시성 충돌 및 트랜잭션 시간 초과를 포함하여 대규모 테이블에서 성능 문제가 발생할 수 있습니다.
> 인덱싱된 필드 또는 문서 ID(예: = 또는 IN)에서 동등 연산자를 사용하여 WHERE 조건자 절이 포함된 문을 실행하는 것이 좋습니다. 자세한 내용은 쿼리 성능 최적화을 참조하십시오.

조건

```
COUNT ( * | expression )
```

인수

###

함수가 실행되는 필드 명칭 또는 표현식. 이 파라미터는 지원되는 데이터 유형 중 하나일 수 있습니다.

반환 타입

`int`

예

```
SELECT COUNT(*) FROM VehicleRegistration r WHERE r.LicensePlateNumber = 'CA762X' -- 1
SELECT COUNT(r.VIN) FROM Vehicle r WHERE r.VIN = '1N4AL11D75C109151'             -- 1
```

```
SELECT COUNT(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >>                    -- 3
```

관련 함수

- AVG
- MAX
- MIN
- SIZE
- SUM

Amazon QLDB의 DATE_ADD 함수

Amazon QLDB에서는 DATE_ADD 함수를 사용하여 주어진 타임스탬프 값을 지정된 간격만큼 증가시킵니다.

조건

```
DATE_ADD( datetimepart, interval, timestamp )
```

인수

datetimepart

함수가 실행되는 날짜 또는 시간 부분입니다. 이 파라미터는 다음 중 하나일 수 있습니다:

- year
- month
- day
- hour
- minute
- second

interval

주어진 *timestamp*에 추가할 간격을 지정하는 정수입니다. 음의 정수는 간격을 감산합니다.

#####

함수가 증분하는 데이터 타입 timestamp의 필드 명칭 또는 표현식.

Ion 타임스탬프 리터럴 값은 백틱(`` `...` ``)으로 표시할 수 있습니다. 형식 지정 세부 정보 및 타임스탬프 값의 예는 Amazon Ion 사양 문서의 타임스탬프를 참조하세요.

반환 타입

`timestamp`

예

```
DATE_ADD(year, 5, `2010-01-01T`)                -- 2015-01-01T
DATE_ADD(month, 1, `2010T`)                     -- 2010-02T (result adds precision as
 necessary)
DATE_ADD(month, 13, `2010T`)                    -- 2011-02T (2010T is equivalent to
 2010-01-01T00:00:00.000Z)
DATE_ADD(day, -1, `2017-01-10T`)                -- 2017-01-09T
DATE_ADD(hour, 1, `2017T`)                      -- 2017-01-01T01:00Z
DATE_ADD(hour, 1, `2017-01-02T03:04Z`)          -- 2017-01-02T04:04Z
DATE_ADD(minute, 1, `2017-01-02T03:04:05.006Z`) -- 2017-01-02T03:05:05.006Z
DATE_ADD(second, 1, `2017-01-02T03:04:05.006Z`) -- 2017-01-02T03:04:06.006Z

-- Runnable statements
SELECT DATE_ADD(year, 5, `2010-01-01T`) FROM << 0 >> -- 2015-01-01T
SELECT DATE_ADD(day, -1, `2017-01-10T`) FROM << 0 >> -- 2017-01-09T
```

관련 함수

- DATE_DIFF
- EXTRACT
- TO_STRING
- TO_TIMESTAMP
- UTCNOW

Amazon QLDB의 DATE_DIFF 함수

Amazon QLDB에서 `DATE_DIFF` 함수를 사용하여 주어진 두 타임스탬프의 지정된 날짜 부분 간의 차이를 반환합니다.

조건

```
DATE_DIFF( datetimepart, timestamp1, timestamp2 )
```

인수

datetimepart

함수가 실행되는 날짜 또는 시간 부분입니다. 이 파라미터는 다음 중 하나일 수 있습니다:

- year
- month
- day
- hour
- minute
- second

timestamp1, *timestamp2*

함수가 비교하는 데이터 타입 timestamp의 두 개의 필드 명칭 또는 표현식입니다. *timestamp2*가 *timestamp1*보다 이후인 경우 결과는 양수입니다. *timestamp2*가 *timestamp1*보다 이전인 경우 결과는 음수입니다.

Ion 타임스탬프 리터럴 값은 백틱(`` `...` ``)으로 표시할 수 있습니다. 형식 지정 세부 정보 및 타임스탬프 값의 예는 Amazon Ion 사양 문서의 타임스탬프를 참조하세요.

반환 타입

int

예

```
DATE_DIFF(year, `2010-01-01T`, `2011-01-01T`)           -- 1
DATE_DIFF(year, `2010-12T`, `2011-01T`)                 -- 0 (must be at least 12
 months apart to evaluate as a 1 year difference)
DATE_DIFF(month, `2010T`, `2010-05T`)                   -- 4 (2010T is equivalent to
 2010-01-01T00:00:00.000Z)
DATE_DIFF(month, `2010T`, `2011T`)                      -- 12
DATE_DIFF(month, `2011T`, `2010T`)                      -- -12
```

```
DATE_DIFF(month, `2010-12-31T`, `2011-01-01T`)           -- 0 (must be at least a full
 month apart to evaluate as a 1 month difference)
DATE_DIFF(day, `2010-01-01T23:00Z`, `2010-01-02T01:00Z`) -- 0 (must be at least 24
 hours apart to evaluate as a 1 day difference)

-- Runnable statements
SELECT DATE_DIFF(year, `2010-01-01T`, `2011-01-01T`) FROM << 0 >> -- 1
SELECT DATE_DIFF(month, `2010T`, `2010-05T`) FROM << 0 >>         -- 4
```

관련 함수

- DATE_ADD
- EXTRACT
- TO_STRING
- TO_TIMESTAMP
- UTCNOW

Amazon QLDB의 EXISTS 함수

Amazon QLDB에서 PartiQL 값이 주어진된 경우 값이 비어 있지 않은 컬렉션이면 EXISTS 함수를 사용하여 TRUE을 반환합니다. 그렇지 않으면 이 함수가 FALSE를 반환합니다. EXISTS에 대한 입력이 컨테이너가 아닌 경우 결과는 FALSE입니다.

EXISTS 함수는 NULL 및 MISSING을 전파하지 않습니다.

조건

```
EXISTS ( value )
```

인수

USD ##

함수가 평가하는 필드 명칭 또는 표현식입니다. 이 파라미터는 지원되는 데이터 유형 중 하나일 수 있습니다.

반환 타입

bool

예

```
EXISTS(`[]`)        -- false (empty list)
EXISTS(`[1, 2, 3]`) -- true (non-empty list)
EXISTS(`[missing]`) -- true (non-empty list)
EXISTS(`{}`)        -- false (empty struct)
EXISTS(`{ a: 1 }`)  -- true (non-empty struct)
EXISTS(`()`)        -- false (empty s-expression)
EXISTS(`(+ 1 2)`)   -- true (non-empty s-expression)
EXISTS(1)           -- false
EXISTS(`2017T`)     -- false
EXISTS(null)        -- false
EXISTS(missing)     -- error

-- Runnable statements
SELECT EXISTS(`[]`) FROM << 0 >>        -- false
SELECT EXISTS(`[1, 2, 3]`) FROM << 0 >> -- true
```

관련 함수

- COALESCE
- nullIF

Amazon QLDB의 EXTRACT 함수

Amazon QLDB에서는 EXTRACT 함수를 사용하여 주어진 타임스탬프에서 지정된 날짜 또는 시간 부분의 정수 값을 반환합니다.

조건

```
EXTRACT ( datetimepart FROM timestamp )
```

인수

datetimepart

함수가 추출하는 날짜 또는 시간 부분입니다. 이 파라미터는 다음 중 하나일 수 있습니다:

- year
- month

- day
- hour
- minute
- second
- timezone_hour
- timezone_minute

#####

함수가 추출하는 데이터 타입 timestamp의 필드 명칭 또는 표현식입니다. 이 파라미터가 알 수 없는 타입(NULL 또는MISSING)인 경우 함수는 NULL을 반환합니다.

Ion 타임스탬프 리터럴 값은 백틱(`...`)으로 표시할 수 있습니다. 형식 지정 세부 정보 및 타임스탬프 값의 예는 Amazon Ion 사양 문서의 타임스탬프를 참조하세요.

반환 타입

int

예

```
EXTRACT(YEAR FROM `2010-01-01T`)                              -- 2010
EXTRACT(MONTH FROM `2010T`)                                   -- 1 (equivalent to
 2010-01-01T00:00:00.000Z)
EXTRACT(MONTH FROM `2010-10T`)                                -- 10
EXTRACT(HOUR FROM `2017-01-02T03:04:05+07:08`)                -- 3
EXTRACT(MINUTE FROM `2017-01-02T03:04:05+07:08`)              -- 4
EXTRACT(TIMEZONE_HOUR FROM `2017-01-02T03:04:05+07:08`)       -- 7
EXTRACT(TIMEZONE_MINUTE FROM `2017-01-02T03:04:05+07:08`)     -- 8

-- Runnable statements
SELECT EXTRACT(YEAR FROM `2010-01-01T`) FROM << 0 >> -- 2010
SELECT EXTRACT(MONTH FROM `2010T`) FROM << 0 >>      -- 1
```

관련 함수

- DATE_ADD
- DATE_DIFF

- TO_STRING
- TO_TIMESTAMP
- UTCNOW

Amazon QLDB의 LOWER 함수

Amazon QLDB에서는 LOWER 함수를 사용하여 주어진 문자열의 모든 대문자를 소문자로 변환합니다.

조건

```
LOWER ( string )
```

인수

string

함수가 변환하는 데이터 타입 string의 필드 명칭 또는 표현식입니다.

반환 타입

string

예

```
SELECT LOWER('AbCdEfG!@#$') FROM << 0 >> -- 'abcdefg!@#$'
```

관련 함수

- CHAR_LENGTH
- SUBSTRING
- TRIM
- UPPER

Amazon QLDB의 MAX 함수

Amazon QLDB에서는 MAX 함수를 사용하여 숫자 값 세트에서 최대값을 반환합니다.

조건

```
MAX ( expression )
```

인수

###

함수가 실행되는 숫자 데이터 타입의 필드 명칭 또는 표현식입니다.

데이터 타입

지원되는 인수 타입:

- `int`
- `decimal`
- `float`

지원되는 반환 타입:

- `int`
- `decimal`
- `float`

예

```
SELECT MAX(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r  -- 442.30
SELECT MAX(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >>         -- 3
```

관련 함수

- AVG
- COUNT
- MIN
- SIZE
- SUM

Amazon QLDB의 MIN 함수

Amazon QLDB에서는 `MIN` 함수를 사용하여 수치 세트에서 최소값을 반환합니다.

조건

```
MIN ( expression )
```

인수

###

함수가 실행되는 숫자 데이터 타입의 필드 명칭 또는 표현식입니다.

데이터 타입

지원되는 인수 타입:

- `int`
- `decimal`
- `float`

지원되는 반환 타입:

- `int`
- `decimal`
- `float`

예

```
SELECT MIN(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r  -- 30.45
SELECT MIN(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >>         -- 1
```

관련 함수

- AVG

- COUNT
- MAX
- SIZE
- SUM

Amazon QLDB의 nullIF 함수

Amazon QLDB에서 두 개의 표현식 주어지면 NULLIF 함수를 사용하여 두 개의 표현식이 동일한 값으로 평가되는 경우 NULL를 반환합니다. 그렇지 않다면 이 함수는 첫 번째 표현식을 평가한 결과를 반환합니다.

NULLIF 함수는 NULL 및 MISSING을 전파하지 않습니다.

조건

```
NULLIF ( expression1, expression2 )
```

인수

expression1, *expression2*

함수가 비교하는 두 개의 필드 명칭 또는 표현식입니다. 이러한 파라미터는 지원되는 데이터 유형 중 하나일 수 있습니다.

반환 타입

지원되는 모든 데이터 타입입니다. 반환 타입은 NULL이거나 첫 번째 표현식의 타입과 같습니다.

예

```
NULLIF(1, 1)             -- null
NULLIF(1, 2)             -- 1
NULLIF(1.0, 1)           -- null
NULLIF(1, '1')           -- 1
NULLIF([1], [1])         -- null
NULLIF(1, NULL)          -- 1
NULLIF(NULL, 1)          -- null
```

```
NULLIF(null, null)       -- null
NULLIF(missing, null)    -- null
NULLIF(missing, missing) -- null

-- Runnable statements
SELECT NULLIF(1, 1) FROM << 0 >>   -- null
SELECT NULLIF(1, '1') FROM << 0 >> -- 1
```

관련 함수

- COALESCE
- exists

Amazon QLDB의 SIZE 함수

Amazon QLDB에서는 `SIZE` 함수를 사용하여 주어진 컨테이너 데이터 타입(목록, 구조 또는 백)의 요소 수를 반환합니다.

조건

```
SIZE ( container )
```

인수

####

함수가 실행되는 컨테이너 필드 명칭 또는 표현식.

데이터 타입

지원되는 인수 타입:

- list
- 구조
- 백

반환 타입: `int`

SIZE에 대한 입력이 컨테이너가 아닌 경우 함수에서 오류가 발생합니다.

예

```
SIZE(`[]`)                  -- 0
SIZE(`[null]`)              -- 1
SIZE(`[1,2,3]`)             -- 3
SIZE(<<'foo', 'bar'>>)      -- 2
SIZE(`{foo: bar}`)          -- 1 (number of key-value pairs)
SIZE(`[{foo: 1}, {foo: 2}]`) -- 2
SIZE(12)                    -- error

-- Runnable statements
SELECT SIZE(`[]`) FROM << 0 >>      -- 0
SELECT SIZE(`[1,2,3]`) FROM << 0 >> -- 3
```

관련 함수

- AVG
- COUNT
- MAX
- MIN
- SUM

Amazon QLDB의 SUBSTRING 함수

Amazon QLDB에서는 SUBSTRING 함수를 사용하여 주어진 문자열에서 하위 문자열을 반환합니다. 하위 문자열은 지정된 시작 인덱스에서 시작하여 문자열의 마지막 문자에서 끝나거나 지정된 길이에서 끝납니다.

조건

```
SUBSTRING ( string, start-index [, length ] )
```

인수

string

하위 문자열을 추출할 데이터 타입 string의 필드 명칭 또는 표현식입니다.

start-index

*###*내에서 추출을 시작할 시작 위치. 이 수는 음의 값이 될 수 있습니다.

*###*의 첫 번째 문자가 인덱스 1을 갖습니다.

length

(옵션) *start-index*에서 시작하여(*start-index* + *length*) - 1에서 끝나는 *string*에서 추출할 문자(코드 포인트)의 수입니다. 다시 말해 하위 문자열의 길이입니다. 이 수는 음의 값이 될 수 없습니다.

이 파라미터를 제공하지 않으면 함수는 *string* 끝까지 진행됩니다.

반환 타입

`string`

예

```
SUBSTRING('123456789', 0)      -- '123456789'
SUBSTRING('123456789', 1)      -- '123456789'
SUBSTRING('123456789', 2)      -- '23456789'
SUBSTRING('123456789', -4)     -- '123456789'
SUBSTRING('123456789', 0, 999) -- '123456789'
SUBSTRING('123456789', 0, 2)   -- '1'
SUBSTRING('123456789', 1, 999) -- '123456789'
SUBSTRING('123456789', 1, 2)   -- '12'
SUBSTRING('1', 1, 0)           -- ''
SUBSTRING('1', 1, 0)           -- ''
SUBSTRING('1', -4, 0)          -- ''
SUBSTRING('1234', 10, 10)      -- ''

-- Runnable statements
SELECT SUBSTRING('123456789', 1) FROM << 0 >>    -- "123456789"
SELECT SUBSTRING('123456789', 1, 2) FROM << 0 >> -- "12"
```

관련 함수

- CHAR_LENGTH
- LOWER

- TRIM
- UPPER

Amazon QLDB의 SUM 함수

Amazon QLDB에서는 SUM 함수를 사용하여 입력 필드 또는 표현식 값의 합계를 반환합니다. 이 함수는 수치를 사용하고 null 또는 누락 값은 무시합니다.

조건

```
SUM ( expression )
```

인수

###

함수가 실행되는 숫자 데이터 타입의 필드 명칭 또는 표현식입니다.

데이터 타입

지원되는 인수 타입:

- `int`
- `decimal`
- `float`

지원되는 반환 타입:

- `int` - 정수 인수의 경우
- `decimal` - 십진수 또는 부동 소수점 인수의 경우

예

```
SELECT SUM(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r  -- 735.95
SELECT SUM(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >>         -- 6
```

관련 함수

- AVG
- COUNT
- MAX
- MIN
- SIZE

Amazon QLDB의 TO_STRING 함수

Amazon QLDB에서는 `TO_STRING` 함수를 사용하여 주어진 타임스탬프의 문자열 표현을 지정된 형식 패턴으로 반환합니다.

조건

```
TO_STRING ( timestamp, 'format' )
```

인수

#####

함수가 문자열로 변환하는 데이터 타입 `timestamp`의 필드 명칭 또는 표현식입니다.

Ion 타임스탬프 리터럴 값은 백틱(`` `...` ``)으로 표시할 수 있습니다. 형식 지정 세부 정보 및 타임스탬프 값의 예는 Amazon Ion 사양 문서의 타임스탬프를 참조하세요.

format

날짜 부분을 기준으로 결과의 형식 패턴을 지정하는 문자열 리터럴입니다. 유효한 형식은 타임스탬프 형식 문자열 섹션을 참조하세요.

반환 타입

`string`

예

```
TO_STRING(`1969-07-20T20:18Z`, 'MMMM d, y')                    -- "July 20, 1969"
```

```
TO_STRING(`1969-07-20T20:18Z`, 'MMM d, yyyy')                    -- "Jul 20, 1969"
TO_STRING(`1969-07-20T20:18Z`, 'M-d-yy')                         -- "7-20-69"
TO_STRING(`1969-07-20T20:18Z`, 'MM-d-y')                         -- "07-20-1969"
TO_STRING(`1969-07-20T20:18Z`, 'MMMM d, y h:m a')                -- "July 20, 1969 8:18
 PM"
TO_STRING(`1969-07-20T20:18Z`, 'y-MM-dd''T''H:m:ssX')            --
 "1969-07-20T20:18:00Z"
TO_STRING(`1969-07-20T20:18+08:00Z`, 'y-MM-dd''T''H:m:ssX')      --
 "1969-07-20T20:18:00Z"
TO_STRING(`1969-07-20T20:18+08:00`, 'y-MM-dd''T''H:m:ssXXXX')    --
 "1969-07-20T20:18:00+0800"
TO_STRING(`1969-07-20T20:18+08:00`, 'y-MM-dd''T''H:m:ssXXXXX')   --
 "1969-07-20T20:18:00+08:00"

-- Runnable statements
SELECT TO_STRING(`1969-07-20T20:18Z`, 'MMMM d, y') FROM << 0 >>           -- "July 20,
 1969"
SELECT TO_STRING(`1969-07-20T20:18Z`, 'y-MM-dd''T''H:m:ssX') FROM << 0 >> --
 "1969-07-20T20:18:00Z"
```

관련 함수

- CAST
- DATE_ADD
- DATE_DIFF
- EXTRACT
- TO_TIMESTAMP
- UTCNOW

Amazon QLDB의 TO_TIMESTAMP 함수

Amazon QLDB에서 타임스탬프를 나타내는 문자열이 주어지면 `TO_TIMESTAMP` 함수를 사용하여 문자열을 `timestamp` 데이터 타입으로 변환합니다. 이것은 `TO_STRING`의 역연산입니다.

조건

```
TO_TIMESTAMP ( string [, 'format' ] )
```

인수

string

함수가 타임스탬프로 변환하는 데이터 타입 string의 필드 명칭 또는 표현식입니다.

format

(옵션) 입력 *string*의 형식 패턴을 날짜 부분 면에서 정의하는 문자열 리터럴입니다. 유효한 형식은 타임스탬프 형식 문자열 섹션을 참조하세요.

이 인수를 생략하면 함수는 *string*이 표준 Ion 타임스탬프 형식이라고 가정합니다. 이 함수를 사용하여 Ion 타임스탬프를 파싱하는 데 권장되는 방법입니다.

제로 패딩은 단일 문자 형식 기호(예:y, M, d, H, h, m, s)를 사용할 때는 선택 사항이지만, 제로 패딩 변형(예: yyyy, MM, dd, HH, hh, mm, ss)에는 필요합니다.

두 자리 연도(형식 기호 yy)에는 특별한 처리가 적용됩니다. 70보다 크거나 같은 값에는 1900이 추가되고 70보다 작은 값에는 2000이 추가됩니다.

월 명칭과 AM 또는 PM 표시기는 대소문자를 구분하지 않습니다.

반환 타입

timestamp

예

```
TO_TIMESTAMP('2007T')                          -- `2007T`
TO_TIMESTAMP('2007-02-23T12:14:33.079-08:00') -- `2007-02-23T12:14:33.079-08:00`
TO_TIMESTAMP('2016', 'y')                      -- `2016T`
TO_TIMESTAMP('2016', 'yyyy')                   -- `2016T`
TO_TIMESTAMP('02-2016', 'MM-yyyy')             -- `2016-02T`
TO_TIMESTAMP('Feb 2016', 'MMM yyyy')           -- `2016-02T`
TO_TIMESTAMP('February 2016', 'MMMM yyyy')     -- `2016-02T`

-- Runnable statements
SELECT TO_TIMESTAMP('2007T') FROM << 0 >>              -- 2007T
SELECT TO_TIMESTAMP('02-2016', 'MM-yyyy') FROM << 0 >> -- 2016-02T
```

관련 함수

- CAST

- DATE_ADD
- DATE_DIFF
- EXTRACT
- TO_STRING
- UTCNOW

Amazon QLDB의 TRIM 함수

Amazon QLDB에서는 TRIM 함수를 사용하여 선행 및 후행 공백 또는 지정된 문자 세트를 제거하여 지정된 문자열을 잘라냅니다.

조건

```
TRIM ( [ LEADING | TRAILING | BOTH [ characters ] FROM ] string)
```

인수

LEADING

(옵션) ###의 시작 부분에서 공백 또는 지정된 문자를 제거할 것임을 나타냅니다. 지정되지 않은 경우, 기본 작업은 BOTH입니다.

TRAILING

(옵션) ###의 끝 부분에서 공백 또는 지정된 문자를 제거할 것임을 나타냅니다. 지정되지 않은 경우, 기본 작업은 BOTH입니다.

BOTH

(옵션) *string*의 처음과 끝 부분에서 선행 및 후행 공백 또는 지정된 문자를 모두 제거할 것임을 나타냅니다.

###

(옵션) 제거할 문자 세트로, string로 지정됩니다.

이 파라미터를 제공하지 않으면 공백이 제거됩니다.

string

함수가 잘라내는 데이터 타입 string의 필드 명칭 또는 표현식입니다.

반환 타입

`string`

예

```
TRIM('       foobar         ')               -- 'foobar'
TRIM('      \tfoobar\t         ')            -- '\tfoobar\t'
TRIM(LEADING FROM '       foobar         ')  -- 'foobar         '
TRIM(TRAILING FROM '       foobar         ') -- '       foobar'
TRIM(BOTH FROM '       foobar         ')     -- 'foobar'
TRIM(BOTH '1' FROM '11foobar11')             -- 'foobar'
TRIM(BOTH '12' FROM '1112211foobar22211122') -- 'foobar'

-- Runnable statements
SELECT TRIM('       foobar         ') FROM << 0 >>              -- "foobar"
SELECT TRIM(LEADING FROM '       foobar         ') FROM << 0 >> -- "foobar         "
```

관련 함수

- CHAR_LENGTH
- LOWER
- SUBSTRING
- UPPER

Amazon QLDB의 TXID 함수

Amazon QLDB에서는 `TXID` 함수를 사용하여 실행 중인 현재 명령문의 고유한 트랜잭션 ID를 반환합니다. 이 값은 현재 트랜잭션이 저널에 체결될 때 문서의 `txId` 메타데이터 필드에 할당되는 값입니다.

조건

```
TXID()
```

인수

없음

반환 타입

`string`

예

```
SELECT TXID() FROM << 0 >>  -- "L7S9iJqcn9W2M4qOEn27ay"
SELECT TXID() FROM Person WHERE GovId = 'LEWISR261LL'  -- "BKeMb48PNyvHWJGZHkaodG"
```

Amazon QLDB의 UPPER 함수

Amazon QLDB에서는 UPPER 함수를 사용하여 주어진 문자열의 모든 소문자를 대문자로 변환합니다.

조건

```
UPPER ( string )
```

인수

string

함수가 변환하는 데이터 타입 `string`의 필드 명칭 또는 표현식입니다.

반환 타입

`string`

예

```
SELECT UPPER('AbCdEfG!@#$') FROM << 0 >> -- 'ABCDEFG!@#$'
```

관련 함수

- CHAR_LENGTH
- LOWER
- SUBSTRING
- TRIM

Amazon QLDB의 UTCNOW 함수

Amazon QLDB에서는 `UTCNOW` 함수를 사용하여 UTC(협정 세계시) 기준 현재 시간을 `timestamp`로 반환합니다.

조건

```
UTCNOW()
```

이 함수를 사용하려면 `SELECT` 쿼리에 `FROM` 절을 지정해야 합니다.

인수

없음

반환 타입

`timestamp`

예

```
SELECT UTCNOW() FROM << 0 >>  -- 2019-12-27T20:12:16.999Z
SELECT UTCNOW() FROM Person WHERE GovId = 'LEWISR261LL'  -- 2019-12-27T20:12:26.999Z

INSERT INTO Person VALUE { 'firstName': 'Jane', 'createdAt': UTCNOW() }
UPDATE Person p SET p.updatedAt = UTCNOW() WHERE p.firstName = 'John'
```

관련 함수

- DATE_ADD
- DATE_DIFF
- EXTRACT
- TO_STRING
- TO_TIMESTAMP

타임스탬프 형식 문자열

이 섹션에서는 타임스탬프 형식 문자열에 대한 참조 정보를 제공합니다.

타임스탬프 형식 문자열은 TO_STRING 및 TO_TIMESTAMP 함수에 적용됩니다. 이러한 문자열에는 날짜 부분 구분 기호(예: '-', '/' 또는 ':')와 다음 형식 기호가 포함될 수 있습니다.

형식	예	설명
yy	70	두 자리 수 연도
y	1970	4자리 수 연도
yyyy	1970	제로 패딩된 4자리 수 연도
M	1	월 정수
MM	01	제로 패딩된 월 정수
MMM	1월	단축된 월명
MMMM	1월	완전한 월명
d	2	일 (1~31)
dd	02	제로 패딩된 일 (01~31)
a	AM 또는 PM	오전/오후 표시자 (12시 방식일 때)
h	3	시간 (12시 방식, 1~12)
hh	03	제로 패딩 시간 (12시 방식, 01~12)
H	3	시간 (24시 방식, 0~23)
HH	03	제로 패딩 시간 (24시 방식, 00~23)
m	4	분 (0~59)
mm	04	제로 패딩 분 (00~59)
s	5	초 (0~59)

형식	예	설명
ss	05	제로 패딩 초 (00-59)
S	0	1초의 분할 (정밀도: 0.1, 범위: 0.0~0.9)
SS	06	1초의 분할 (정밀도: 0.01, 범위: 0.0~0.99)
SSS	060	1초의 분할 (정밀도: 0.001, 범위: 0.0~0.999)
X	+07 또는 Z	시 단위로 표시한 UTC로부터의 오프셋. 또는 오프셋이 0인 경우 "Z"
XX	+0700 또는 Z	시와 분 단위로 표시한 UTC로부터의 오프셋. 또는 오프셋이 0인 경우 "Z"
XXX	+ 07:00 또는 Z	시와 분 단위로 표시한 UTC로부터의 오프셋. 또는 오프셋이 0인 경우 "Z"
x	+07	UTC로부터의 오프셋 (시간 단위)
xx	+0700	시간과 분 단위로 표시한 UTC로부터의 오프셋
xxx	+07:00	시간과 분 단위로 표시한 UTC로부터의 오프셋

Amazon QLDB의 PartiQL 저장 프로시저

Amazon QLDB에서는 EXEC 명령을 사용하여 다음 구문으로 PartiQL 저장 프로시저를 실행할 수 있습니다.

```
EXEC stored_procedure_name argument [, ... ]
```

QLDB는 다음 시스템 저장 프로시저를 지원합니다.

주제

- Amazon QLDB의 REDACT_REVISION 저장 프로시저

Amazon QLDB의 REDACT_REVISION 저장 프로시저

> **Note**
>
> 2021년 7월 22일 이전에 생성된 원장은 현재 수정할 수 없습니다. Amazon QLDB 콘솔에서 원장 생성 시간을 볼 수 있습니다.

Amazon QLDB에서는 `REDACT_REVISION` 저장 프로시저를 사용하여 인덱싱된 스토리지와 저널 스토리지 모두에서 비활성 상태의 개별 문서 수정본을 영구적으로 삭제합니다. 이 저장 프로시저는 지정된 수정본의 모든 사용자 데이터를 삭제합니다. 하지만 저널 시퀀스와 문서 ID 및 해시를 포함한 문서 메타데이터는 변경되지 않습니다. 이 작업은 되돌릴 수 없습니다.

지정된 문서 수정본은 기록에서 비활성 상태여야 합니다. 문서의 최신 활성 수정본은 수정할 수 없습니다.

이 저장 프로시저를 실행하여 수정 요청을 제출하면 QLDB는 데이터 삭제를 비동기적으로 처리합니다. 수정이 완료되면 지정된 수정본(`data` 구조로 표시됨)의 사용자 데이터가 새 `dataHash` 필드로 대체됩니다. 이 필드의 값은 제거된 `data` 구조의 Amazon Ion 해시입니다. 따라서 원장은 전반적인 데이터 무결성을 유지하고 기존 검증 API 작업을 통해 암호화 방식으로 검증 가능한 상태를 유지합니다.

샘플 데이터를 사용한 수정 작업의 예는 문서 개정본 수정하기의 수정 예제 항목을 참조하세요.

> **Note**
>
> 특정 테이블에서 이 PartiQL 명령을 실행하기 위한 액세스를 제어하는 방법을 알아보려면 Amazon QLDB에서 표준 권한 모드로 시작하기 섹션을 참조하세요.

주제

수정 고려 사항 및 제한

Amazon QLDB에서 데이터 수정을 시작하기 전에 다음 고려 사항과 제한 사항을 검토하세요.

- REDACT_REVISION 저장 프로시저는 비활성 상태의 개별 문서 수정본의 사용자 데이터를 대상으로 합니다. 여러 수정본을 수정하려면 각 수정본마다 한 번씩 저장 프로시저를 실행해야 합니다. 트랜잭션당 하나의 수정본을 수정할 수 있습니다.
- 문서 수정본 내의 특정 필드를 수정하려면 먼저 별도의 데이터 조작 언어(DML) 문을 사용하여 수정본을 수정해야 합니다. 자세한 내용은 개정본 내 특정 필드 수정하기 섹션을 참조하세요.
- QLDB가 수정 요청을 받은 후에는 요청을 취소하거나 변경할 수 없습니다. 수정 완료 여부를 확인하려면 수정본의 data 구조가 dataHash 필드로 대체되었는지 확인하면 됩니다. 자세한 내용은 수정 완료 여부 확인 단원을 참조하십시오.
- 수정은 QLDB 서비스 외부에 복제된 QLDB 데이터에는 영향을 주지 않습니다. 여기에는 Amazon S3로의 모든 내보내기와 Amazon Kinesis Data Streams로의 스트림이 포함됩니다. QLDB 외부에 저장된 모든 데이터를 관리하려면 다른 데이터 보존 방법을 사용해야 합니다.
- 수정은 저널에 기록된 PartiQL 명령문의 리터럴 값에는 영향을 주지 않습니다. 모범 사례에 따라 파라미터화된 명령문은 리터럴 값 대신 변수 자리 표시자를 사용하여 프로그래밍 방식으로 실행해야 합니다. 저널에서 자리 표시자는 수정이 필요할 수 있는 민감한 정보 대신 물음표(?)로 기록됩니다.

  QLDB 드라이버를 사용하여 PartiQL 명령문을 프로그래밍 방식으로 실행하는 방법을 알아보려면 드라이버 시작하기에서 지원되는 각 프로그래밍 언어의 자습서를 참조하세요.

구문

```
EXEC REDACT_REVISION `block-address`, 'table-id', 'document-id'
```

인수

`` `block-address` ``

수정할 문서의 수정본의 저널 블록 위치입니다. 주소는 `strandId` 및 `sequenceNo`라는 두 개의 필드로 구성된 Amazon Ion 구조입니다.

이 값은 백틱으로 표시되는 Ion 리터럴 값입니다. 예:

```
`{strandId:"JdxjkR9bSYB5jMHWcI464T", sequenceNo:17}`
```

블록 주소를 찾는 방법을 알아보려면 문서 메타데이터 쿼리 섹션을 참조하세요.

'*table-id*'

수정하고자 하는 문서 개정본이 있는 테이블의 고유 ID로, 작은 따옴표로 표시됩니다.

테이블 ID를 찾는 방법에 대한 자세한 내용은 시스템 카탈로그 쿼리 섹션을 참조하세요.

'*document-id*'

수정할 개정본의 고유한 문서 ID로, 작은 따옴표로 표시됩니다.

문서 ID를 찾는 방법에 대한 자세한 내용은 문서 메타데이터 쿼리 섹션을 참조하세요.

반환 값

수정해야 할 문서 개정을 다음 형식으로 나타내는 Amazon Ion 구조입니다.

```
{
  blockAddress: {
    strandId: String,
    sequenceNo: Int
  },
  tableId: String,
  documentId: String,
  version: Int
}
```

반환 구조 필드

- `blockAddress` - 수정할 수정본의 저널 블록 위치입니다. 주소에는 다음과 같은 두 가지 필드가 있습니다.

  - `strandId` - 블록을 포함하는 저널 스트랜드의 고유 ID입니다.
  - `sequenceNo` - 스트랜드 내 블록 위치를 지정하는 인덱스 번호입니다.
- `tableId` - 수정하려는 수정본이 있는 테이블의 고유 ID입니다.
- `documentId` - 수정할 수정본의 고유한 문서 ID입니다.
- `version` - 수정할 문서 수정본의 버전 번호입니다.

다음은 샘플 데이터를 사용한 반환 구조의 예입니다.

```
{
  blockAddress: {
    strandId: "CsRnx0RDoNK6ANEEePa1ov",
    sequenceNo: 134
  },
  tableId: "6GZumdHggklLdMGyQq9DNX",
  documentId: "IXlQPSbfyKMIIsygePeKrZ",
  version: 0
}
```

예시

```
EXEC REDACT_REVISION `{strandId:"7z2P0AyQKWD8oFYmGNhi8D", sequenceNo:7}`,
 '8F0TPCmdNQ6JTRpiLj2TmW', '05K8zpGYWynDlEOK5afDRc'
```

Amazon QLDB의 PartiQL 연산자

Amazon QLDB의 PartiQL에서는 다음과 같은 SQL 표준 연산자를 지원합니다.

> **Note**
>
> 이 목록에 없는 모든 SQL 연산자는 현재 QLDB에서 지원되지 않습니다.

산술 연산자

연산자	설명
+	더하기

연산자	설명
-	빼기
*	곱하기
/	나누기
%	모듈로

비교 연산자

연산자	설명
=	같음
>	초과
<	미만
>=	크거나 같음
<=	작거나 같음
<>	같지 않음

논리 연산자

연산자	설명
AND	AND로 구분된 조건이 모두 TRUE이면 TRUE
BETWEEN	피연산자가 비교 범위 이내이면 TRUE
IN	피연산자가 표현식 목록 중 하나와 같으면 TRUE
IS	피연산자가 지정된 데이터 유형(NULL 또는 MISSING 포함)이면 TRUE

연산자	설명
LIKE	피연산자가 패턴과 일치하면 TRUE
NOT	지정된 부울 식의 값을 반대로 바꿈
OR	OR로 구분된 조건 중 하나라도 TRUE이면 TRUE

문자열 연산자

연산자	설명
||	|| 연산자의 양쪽으로 두 문자열을 연결하여 연결된 문자열을 반환합니다. 문자열 하나 또는 둘 모두가 NULL인 경우 연결 결과는 null입니다.

Amazon QLDB의 예약어

다음은 Amazon QLDB의 PartiQL 예약 키워드 목록입니다. 예약어를 큰따옴표가 있는 따옴표로 묶은 식별자로 사용할 수 있습니다(예: `"user"`). QLDB의 PartiQL 인용 규칙에 대한 자세한 내용은 PartiQL을 사용하여 Ion 쿼리하기 섹션을 참조하세요.

> **Important**
>
> PartiQL은 SQL-92 이전 버전과 호환되므로 이 목록의 키워드는 모두 예약된 것으로 간주됩니다. 하지만 QLDB는 이러한 예약어 중 일부만 지원합니다. QLDB에서 현재 지원하는 SQL 키워드 목록은 다음 주제를 참조하세요.
>
> - PartiQL 함수
> - PartiQL 연산자
> - PartiQL 명령

```
ABSOLUTE
ACTION
ADD
```

```
ALL
ALLOCATE
ALTER
AND
ANY
ARE
AS
ASC
ASSERTION
AT
AUTHORIZATION
AVG
BAG
BEGIN
BETWEEN
BIT
BIT_LENGTH
BLOB
BOOL
BOOLEAN
BOTH
BY
CASCADE
CASCADED
CASE
CAST
CATALOG
CHAR
CHARACTER
CHARACTER_LENGTH
CHAR_LENGTH
CHECK
CLOB
CLOSE
COALESCE
COLLATE
COLLATION
COLUMN
COMMIT
CONNECT
CONNECTION
CONSTRAINT
CONSTRAINTS
CONTINUE
```

```
CONVERT
CORRESPONDING
COUNT
CREATE
CROSS
CURRENT
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURSOR
DATE
DATE_ADD
DATE_DIFF
DAY
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DEFERRABLE
DEFERRED
DELETE
DESC
DESCRIBE
DESCRIPTOR
DIAGNOSTICS
DISCONNECT
DISTINCT
DOMAIN
DOUBLE
DROP
ELSE
END
END-EXEC
ESCAPE
EXCEPT
EXCEPTION
EXEC
EXECUTE
EXISTS
EXTERNAL
EXTRACT
FALSE
```

```
FETCH
FIRST
FLOAT
FOR
FOREIGN
FOUND
FROM
FULL
GET
GLOBAL
GO
GOTO
GRANT
GROUP
HAVING
HOUR
IDENTITY
IMMEDIATE
IN
INDEX
INDICATOR
INITIALLY
INNER
INPUT
INSENSITIVE
INSERT
INT
INTEGER
INTERSECT
INTERVAL
INTO
IS
ISOLATION
JOIN
KEY
LANGUAGE
LAST
LEADING
LEFT
LEVEL
LIKE
LIMIT
LIST
LOCAL
```

```
LOWER
MATCH
MAX
MIN
MINUTE
MISSING
MODULE
MONTH
NAMES
NATIONAL
NATURAL
NCHAR
NEXT
NO
NOT
NULL
NULLIF
NUMERIC
OCTET_LENGTH
OF
ON
ONLY
OPEN
OPTION
OR
ORDER
OUTER
OUTPUT
OVERLAPS
PAD
PARTIAL
PIVOT
POSITION
PRECISION
PREPARE
PRESERVE
PRIMARY
PRIOR
PRIVILEGES
PROCEDURE
PUBLIC
READ
REAL
REFERENCES
```

```
RELATIVE
REMOVE
RESTRICT
REVOKE
RIGHT
ROLLBACK
ROWS
SCHEMA
SCROLL
SECOND
SECTION
SELECT
SESSION
SESSION_USER
SET
SEXP
SIZE
SMALLINT
SOME
SPACE
SQL
SQLCODE
SQLERROR
SQLSTATE
STRING
STRUCT
SUBSTRING
SUM
SYMBOL
SYSTEM_USER
TABLE
TEMPORARY
THEN
TIME
TIMESTAMP
TIMEZONE_HOUR
TIMEZONE_MINUTE
TO
TO_STRING
TO_TIMESTAMP
TRAILING
TRANSACTION
TRANSLATE
TRANSLATION
```

```
TRIM
TRUE
TUPLE
TXID
UNDROP
UNION
UNIQUE
UNKNOWN
UNPIVOT
UPDATE
UPPER
USAGE
USER
USING
UTCNOW
VALUE
VALUES
VARCHAR
VARYING
VIEW
WHEN
WHENEVER
WHERE
WITH
WORK
WRITE
YEAR
ZONE
```

Amazon QLDB의 Amazon Ion 데이터 형식 참조

Amazon QLDB는 Amazon Ion을 PartiQL 유형의 하위 집합과 통합하는 데이터 표기법 모델을 사용합니다. 이 섹션에서는 PartiQL과의 통합과 별도로 Ion 문서 데이터 형식에 대한 참조 개요를 제공합니다.

Amazon QLDB에서 PartiQL을 사용한 Ion 쿼리

QLDB에서 PartiQL을 사용하여 Ion 데이터를 쿼리하는 구문 및 시맨틱은 Amazon QLDB PartiQL 참조의 PartiQL을 사용하여 Ion 쿼리하기를 참조하세요.

QLDB 원장에서 Ion 데이터를 쿼리하고 처리하는 코드 예제는 Amazon Ion 코드 예제 및 Amazon Ion 작업을 참조하세요.

주제

Amazon Ion이란 무엇입니까?

Ion은 풍부한 유형의 자체 설명이 가능한 오픈 소스 계층적 데이터 직렬화 형식으로, 원래 Amazon 내부에서 개발되었습니다. 이는 구조화된 데이터와 구조화되지 않은 데이터를 모두 저장할 수 있는 추상 데이터 모델을 기반으로 합니다. 이는 JSON의 상위 집합입니다. 즉, 유효한 JSON 문서는 모두 유효한 Ion 문서이기도 합니다. 이 가이드에서는 JSON에 대한 기본적인 실무 지식을 전제로 합니다. JSON에 아직 익숙하지 않은 경우 자세한 내용은 JSON 소개를 참조하세요.

Ion 문서를 사람이 읽을 수 있는 텍스트 형식이나 이진 인코딩 형식으로 바꿔서 표기할 수 있습니다. JSON과 마찬가지로 텍스트 형식은 읽고 쓰기가 쉬우며 신속한 프로토타이핑을 지원합니다. 이진 인코딩은 더욱 간결하고 지속, 전송 및 구문 분석에 효율적입니다. Ion 프로세서는 두 형식 사이를 트랜스코딩하여 데이터 손실 없이 정확히 동일한 데이터 구조 집합을 표현할 수 있습니다. 이 기능을 통해 애플리케이션은 다양한 사용 사례에 맞게 데이터를 처리하는 방법을 최적화할 수 있습니다.

> **Note**
>
> Ion 데이터 모델은 엄격하게 값을 기반으로 하며 참조를 지원하지 않습니다. 따라서 데이터 모델은 방향성 그래프가 아닌 임의의 깊이로 중첩될 수 있는 데이터 계층 구조를 나타낼 수 있습니다.

Ion 사양

전체 설명 및 값 형식 지정 세부 정보가 포함된 Ion 코어 데이터 유형의 전체 목록은 Amazon GitHub 사이트의 Ion 사양 문서를 참조하세요.

애플리케이션 개발을 간소화하기 위해 Amazon Ion은 Ion 데이터를 처리하는 클라이언트 라이브러리를 제공합니다. Ion 데이터를 처리하는 일반적인 사용 사례의 코드 예제는 GitHub의 Amazon Ion Cookbook을 참조하세요.

JSON 호환

JSON과 마찬가지로, 원시 데이터 유형 집합과 재귀적으로 정의된 컨테이너 유형 집합을 사용하여 Amazon Ion 문서를 작성합니다. Ion에는 다음과 같은 기존 JSON 데이터 유형이 포함됩니다.

- `null`: 형식이 지정되지 않은 일반 null(빈) 값입니다. 또한 다음 섹션에 설명된 대로 Ion은 각 원시 유형에 대해 고유한 null 유형을 지원합니다.
- `bool`: 부울 값입니다.
- `string`: 유니코드 텍스트 리터럴입니다.
- `list`: 서로 다른 유형의 값 컬렉션을 정렬했습니다.
- `struct`: 정렬된 서로 다른 유형의 값 컬렉션입니다. JSON과 마찬가지로 `struct`은 이름당 여러 값을 허용하지만 일반적으로 권장되지 않습니다.

JSON의 확장

숫자 유형

Amazon Ion은 숫자를 모호한 JSON `number` 유형 대신 다음 유형 중 하나로 엄격하게 정의합니다.

- `int`: 임의 크기의 부호 있는 정수입니다.
- `decimal`: 십진 인코딩된 임의 정밀도의 실수입니다.
- `float`: 이진 인코딩된 부동 소수점 숫자(64비트 IEEE)입니다.

문서를 구문 분석할 때 Ion 프로세서는 다음과 같이 숫자 유형을 할당합니다.

- `int`: 지수나 소수점이 없는 숫자(예: `100200`)입니다.
- `decimal`: 소수점이 있고 지수는 없는 숫자(예: `0.00001`, `200.0`)입니다.
- `float`: 과학 표기법 또는 전자 표기법과 같이 지수가 있는 숫자(예: `2e0`, `3.1e-4`)입니다.

새로운 데이터 유형

Amazon Ion은 다음과 같은 데이터 유형을 추가합니다.

- `timestamp`: 임의 정밀도의 날짜/시간/시간대 모멘트입니다.

- `symbol`: 유니코드 기호 원자(예: 식별자)입니다.
- `blob`: 사용자 정의 인코딩의 이진 데이터입니다.
- `clob`: 사용자 정의 인코딩의 텍스트 데이터입니다.
- `sexp`: 애플리케이션 정의 시맨틱을 사용하여 정렬된 값 컬렉션입니다.

Null 유형

Amazon Ion은 JSON으로 정의된 일반적인 null 유형 외에도 각 원시 유형에 대해 고유한 null 유형을 지원합니다. 이는 엄격한 데이터 유형을 유지하면서도 값이 부족하다는 것을 나타냅니다.

```
null
null.null       // Identical to untyped null
null.bool
null.int
null.float
null.decimal
null.timestamp
null.string
null.symbol
null.blob
null.clob
null.struct
null.list
null.sexp
```

Ion 텍스트 예시

```
// Here is a struct, which is similar to a JSON object.
{
    // Field names don't always have to be quoted.
    name: "fido",

    // This is an integer.
    age: 7,

    // This is a timestamp with day precision.
    birthday: 2012-03-01T,

    // Here is a list, which is like a JSON array.
    toys: [
```

```
        // These are symbol values, which are like strings,
        // but get encoded as integers in binary.
        ball,
        rope
    ],
}
```

API 참조

- ion-go
- ion-java
- ion-js
- ion-python

QLDB의 Amazon Ion 코드 예제

이 섹션에서는 Amazon QLDB 원장에서 문서 값을 읽고 쓰는 방식으로 Amazon Ion 데이터를 처리하는 코드 예제를 제공합니다. 코드 예제는 QLDB 드라이버를 사용하여 원장에서 PartiQL 문을 실행합니다. 이 예제는 샘플 애플리케이션 자습서를 사용하여 Amazon QLDB 시작하기의 샘플 애플리케이션의 일부이며 AWS 샘플 GitHub 사이트의 오픈 소스입니다.

Ion 데이터 처리의 일반적인 사용 사례를 보여주는 일반적인 코드 예제는 GitHub의 Amazon Ion Cookbook을 참조하세요.

코드 실행

각 프로그래밍 언어에 대한 자습서 코드는 다음 단계를 수행합니다.

1. `vehicle-registration` 샘플 원장에 연결합니다.
2. `IonTypes`이라는 테이블을 생성합니다.
3. 단일 `Name` 필드를 사용하여 테이블에 문서를 삽입합니다.
4. 지원되는 각 Ion 데이터 유형에 대해:
   a. 문서의 `Name` 필드를 데이터 유형의 리터럴 값으로 업데이트합니다.
   b. 테이블을 쿼리하여 문서의 최신 개정판을 가져올 수 있습니다.
   c. `Name` 값이 예상 유형과 일치하는지 확인하여 원래 데이터 유형 속성을 유지하는지 확인합니다.
5. `IonTypes` 테이블을 삭제합니다.

> **Note**
>
> 이 자습서 코드를 실행하기 전에 `vehicle-registration`이라는 이름의 원장을 생성해야 합니다.

Java

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

package software.amazon.qldb.tutorial;

import com.amazon.ion.IonBlob;
import com.amazon.ion.IonBool;
import com.amazon.ion.IonClob;
import com.amazon.ion.IonDecimal;
import com.amazon.ion.IonFloat;
import com.amazon.ion.IonInt;
import com.amazon.ion.IonList;
import com.amazon.ion.IonNull;
import com.amazon.ion.IonSexp;
```

```
import com.amazon.ion.IonString;
import com.amazon.ion.IonStruct;
import com.amazon.ion.IonSymbol;
import com.amazon.ion.IonTimestamp;
import com.amazon.ion.IonValue;
import com.amazon.ion.Timestamp;
import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;
import java.util.Collections;
import java.util.List;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;

/**
 * Insert all the supported Ion types into a ledger and verify that they are stored
 and can be retrieved properly, retaining
 * their original properties.
 *
 * This code expects that you have AWS credentials setup per:
 * http://docs.aws.amazon.com/java-sdk/latest/developer-guide/setup-credentials.html
 */
public class InsertIonTypes {
    public static final Logger log = LoggerFactory.getLogger(InsertIonTypes.class);
    public static final String TABLE_NAME = "IonTypes";

    private InsertIonTypes() {}

    /**
     * Update a document's Name value in the database. Then, query the value of the
 Name key and verify the expected Ion type was
     * saved.
     *
     * @param txn
     *              The {@link TransactionExecutor} for statement execution.
     * @param ionValue
     *              The {@link IonValue} to set the document's Name value to.
     *
     * @throws AssertionError when no value is returned for the Name key or if the
 value does not match the expected type.
     */
    public static void updateRecordAndVerifyType(final TransactionExecutor txn,
 final IonValue ionValue) {
```

```
        final String updateStatement = String.format("UPDATE %s SET Name = ?",
TABLE_NAME);
        final List<IonValue> parameters = Collections.singletonList(ionValue);
        txn.execute(updateStatement, parameters);
        log.info("Updated document.");

        final String searchQuery = String.format("SELECT VALUE Name FROM %s",
TABLE_NAME);
        final Result result = txn.execute(searchQuery);

        if (result.isEmpty()) {
            throw new AssertionError("Did not find any values for the Name key.");
        }
        for (IonValue value : result) {
            if (!ionValue.getClass().isInstance(value)) {
                throw new AssertionError(String.format("The queried value, %s, is
not an instance of %s.",
                        value.getClass().toString(),
ionValue.getClass().toString()));
            }
            if (!value.getType().equals(ionValue.getType())) {
                throw new AssertionError(String.format("The queried value type, %s,
does not match %s.",
                        value.getType().toString(), ionValue.getType().toString()));
            }
        }

        log.info("Successfully verified value is instance of {} with type {}.",
ionValue.getClass().toString(),
                ionValue.getType().toString());
    }

    /**
     * Delete a table.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *              The name of the table to delete.
     */
    public static void deleteTable(final TransactionExecutor txn, final String
tableName) {
        log.info("Deleting {} table...", tableName);
        final String statement = String.format("DROP TABLE %s", tableName);
```

```
        txn.execute(statement);
        log.info("{} table successfully deleted.", tableName);
    }

    public static void main(final String... args) {
        final IonBlob ionBlob = Constants.SYSTEM.newBlob("hello".getBytes());
        final IonBool ionBool = Constants.SYSTEM.newBool(true);
        final IonClob ionClob = Constants.SYSTEM.newClob("{{'This is a CLOB of
text.'}}".getBytes());
        final IonDecimal ionDecimal = Constants.SYSTEM.newDecimal(0.1);
        final IonFloat ionFloat = Constants.SYSTEM.newFloat(0.2);
        final IonInt ionInt = Constants.SYSTEM.newInt(1);
        final IonList ionList = Constants.SYSTEM.newList(new int[]{1, 2});
        final IonNull ionNull = Constants.SYSTEM.newNull();
        final IonSexp ionSexp = Constants.SYSTEM.newSexp(new int[]{2, 3});
        final IonString ionString = Constants.SYSTEM.newString("string");
        final IonStruct ionStruct = Constants.SYSTEM.newEmptyStruct();
        ionStruct.put("brand", Constants.SYSTEM.newString("ford"));
        final IonSymbol ionSymbol = Constants.SYSTEM.newSymbol("abc");
        final IonTimestamp ionTimestamp =
Constants.SYSTEM.newTimestamp(Timestamp.now());

        final IonBlob ionNullBlob = Constants.SYSTEM.newNullBlob();
        final IonBool ionNullBool = Constants.SYSTEM.newNullBool();
        final IonClob ionNullClob = Constants.SYSTEM.newNullClob();
        final IonDecimal ionNullDecimal = Constants.SYSTEM.newNullDecimal();
        final IonFloat ionNullFloat = Constants.SYSTEM.newNullFloat();
        final IonInt ionNullInt = Constants.SYSTEM.newNullInt();
        final IonList ionNullList = Constants.SYSTEM.newNullList();
        final IonSexp ionNullSexp = Constants.SYSTEM.newNullSexp();
        final IonString ionNullString = Constants.SYSTEM.newNullString();
        final IonStruct ionNullStruct = Constants.SYSTEM.newNullStruct();
        final IonSymbol ionNullSymbol = Constants.SYSTEM.newNullSymbol();
        final IonTimestamp ionNullTimestamp = Constants.SYSTEM.newNullTimestamp();


        ConnectToLedger.getDriver().execute(txn -> {
            CreateTable.createTable(txn, TABLE_NAME);
            final Document document = new
Document(Constants.SYSTEM.newString("val"));
            InsertDocument.insertDocuments(txn, TABLE_NAME,
Collections.singletonList(document));

            updateRecordAndVerifyType(txn, ionBlob);
```

```
            updateRecordAndVerifyType(txn, ionBool);
            updateRecordAndVerifyType(txn, ionClob);
            updateRecordAndVerifyType(txn, ionDecimal);
            updateRecordAndVerifyType(txn, ionFloat);
            updateRecordAndVerifyType(txn, ionInt);
            updateRecordAndVerifyType(txn, ionList);
            updateRecordAndVerifyType(txn, ionNull);
            updateRecordAndVerifyType(txn, ionSexp);
            updateRecordAndVerifyType(txn, ionString);
            updateRecordAndVerifyType(txn, ionStruct);
            updateRecordAndVerifyType(txn, ionSymbol);
            updateRecordAndVerifyType(txn, ionTimestamp);

            updateRecordAndVerifyType(txn, ionNullBlob);
            updateRecordAndVerifyType(txn, ionNullBool);
            updateRecordAndVerifyType(txn, ionNullClob);
            updateRecordAndVerifyType(txn, ionNullDecimal);
            updateRecordAndVerifyType(txn, ionNullFloat);
            updateRecordAndVerifyType(txn, ionNullInt);
            updateRecordAndVerifyType(txn, ionNullList);
            updateRecordAndVerifyType(txn, ionNullSexp);
            updateRecordAndVerifyType(txn, ionNullString);
            updateRecordAndVerifyType(txn, ionNullStruct);
            updateRecordAndVerifyType(txn, ionNullSymbol);
            updateRecordAndVerifyType(txn, ionNullTimestamp);

            deleteTable(txn, TABLE_NAME);
        });
    }

    /**
     * This class represents a simple document with a single key, Name, to use for
the IonTypes table.
     */
    private static class Document {
        private final IonValue name;

        @JsonCreator
        private Document(@JsonProperty("Name") final IonValue name) {
            this.name = name;
        }

        @JsonProperty("Name")
        private IonValue getName() {
```

```
            return name;
        }
    }
}
```

Node.js

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: MIT-0
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
 * software and associated documentation files (the "Software"), to deal in the
 Software
 * without restriction, including without limitation the rights to use, copy,
 modify,
 * merge, publish, distribute, sublicense, and/or sell copies of the Software, and
 to
 * permit persons to whom the Software is furnished to do so.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
 COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
 * OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

import { QldbDriver, Result, TransactionExecutor } from "amazon-qldb-driver-nodejs";
import { AssertionError } from "assert";
import { dom, IonType, IonTypes } from "ion-js";

import { insertDocument } from "./InsertDocument";
import { getQldbDriver } from "./ConnectToLedger";
import { createTable } from "./CreateTable";
import { error, log } from "./qldb/LogUtil";

const TABLE_NAME: string = "IonTypes";

/**
```

```
 * Delete a table.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName Name of the table to delete.
 * @returns Promise which fulfills with void.
 */
export async function deleteTable(txn: TransactionExecutor, tableName: string):
 Promise<void> {
    log(`Deleting ${tableName} table...`);
    const statement: string = `DROP TABLE ${tableName}`;
    await txn.execute(statement);
    log(`${tableName} table successfully deleted.`);
}

/**
 * Update a document's Name value in QLDB. Then, query the value of the Name key and
 verify the expected Ion type was
 * saved.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param parameter The IonValue to set the document's Name value to.
 * @param ionType The Ion type that the Name value should be.
 * @returns Promise which fulfills with void.
 */
async function updateRecordAndVerifyType(
    txn: TransactionExecutor,
    parameter: any,
    ionType: IonType
): Promise<void> {
    const updateStatement: string = `UPDATE ${TABLE_NAME} SET Name = ?`;
    await txn.execute(updateStatement, parameter);
    log("Updated record.");

    const searchStatement: string = `SELECT VALUE Name FROM ${TABLE_NAME}`;
    const result: Result = await txn.execute(searchStatement);

    const results: dom.Value[] = result.getResultList();

    if (0 === results.length) {
        throw new AssertionError({
            message: "Did not find any values for the Name key."
        });
    }

    results.forEach((value: dom.Value) => {
        if (value.getType().binaryTypeId !== ionType.binaryTypeId) {
```

```
            throw new AssertionError({
                message: `The queried value type, ${value.getType().name}, does not
 match expected type, ${ionType.name}.`
            });
        }
    });

    log(`Successfully verified value is of type ${ionType.name}.`);
}

/**
 * Insert all the supported Ion types into a table and verify that they are stored
 and can be retrieved properly,
 * retaining their original properties.
 * @returns Promise which fulfills with void.
 */
const main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();
        await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
            await createTable(txn, TABLE_NAME);
            await insertDocument(txn, TABLE_NAME, [{ "Name": "val" }]);
            await updateRecordAndVerifyType(txn, dom.load("null"), IonTypes.NULL);
            await updateRecordAndVerifyType(txn, true, IonTypes.BOOL);
            await updateRecordAndVerifyType(txn, 1, IonTypes.INT);
            await updateRecordAndVerifyType(txn, 3.2, IonTypes.FLOAT);
            await updateRecordAndVerifyType(txn, dom.load("5.5"), IonTypes.DECIMAL);
            await updateRecordAndVerifyType(txn, dom.load("2020-02-02"),
 IonTypes.TIMESTAMP);
            await updateRecordAndVerifyType(txn, dom.load("abc123"),
 IonTypes.SYMBOL);
            await updateRecordAndVerifyType(txn, dom.load("\"string\""),
 IonTypes.STRING);
            await updateRecordAndVerifyType(txn, dom.load("{{ \"clob\" }}"),
 IonTypes.CLOB);
            await updateRecordAndVerifyType(txn, dom.load("{{ blob }}"),
 IonTypes.BLOB);
            await updateRecordAndVerifyType(txn, dom.load("(1 2 3)"),
 IonTypes.SEXP);
            await updateRecordAndVerifyType(txn, dom.load("[1, 2, 3]"),
 IonTypes.LIST);
            await updateRecordAndVerifyType(txn, dom.load("{brand: ford}"),
 IonTypes.STRUCT);
            await deleteTable(txn, TABLE_NAME);
```

```
        });
    } catch (e) {
        error(`Error updating and validating Ion types: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

Python

```
# Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: MIT-0
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of
 this
# software and associated documentation files (the "Software"), to deal in the
 Software
# without restriction, including without limitation the rights to use, copy, modify,
# merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
# permit persons to whom the Software is furnished to do so.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 IMPLIED,
# INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
# PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
# HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
# OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
# SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# This code expects that you have AWS credentials setup per:
# https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
from datetime import datetime
from decimal import Decimal
from logging import basicConfig, getLogger, INFO

from amazon.ion.simple_types import IonPyBool, IonPyBytes, IonPyDecimal, IonPyDict,
 IonPyFloat, IonPyInt, IonPyList, \
    IonPyNull, IonPySymbol, IonPyText, IonPyTimestamp
from amazon.ion.simpleion import loads
from amazon.ion.symbols import SymbolToken
from amazon.ion.core import IonType
```

```
from pyqldbsamples.create_table import create_table
from pyqldbsamples.constants import Constants
from pyqldbsamples.insert_document import insert_documents
from pyqldbsamples.model.sample_data import convert_object_to_ion
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)

TABLE_NAME = 'IonTypes'


def update_record_and_verify_type(driver, parameter, ion_object, ion_type):
    """
    Update a record in the database table. Then query the value of the record and
 verify correct ion type saved.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type parameter: :py:class:`amazon.ion.simple_types.IonPyValue`
    :param parameter: The Ion value or Python native type that is convertible to Ion
 for filling in parameters of the
                      statement.

    :type
 ion_object: :py:obj:`IonPyBool`/:py:obj:`IonPyBytes`/:py:obj:`IonPyDecimal`/:py:obj:`IonPyD

 /:py:obj:`IonPyFloat`/:py:obj:`IonPyInt`/:py:obj:`IonPyList`/:py:obj:`IonPyNull`

 /:py:obj:`IonPySymbol`/:py:obj:`IonPyText`/:py:obj:`IonPyTimestamp`
    :param ion_object: The Ion object to verify against.

    :type ion_type: :py:class:`amazon.ion.core.IonType`
    :param ion_type: The Ion type to verify against.

    :raises TypeError: When queried value is not an instance of Ion type.
    """
    update_query = 'UPDATE {} SET Name = ?'.format(TABLE_NAME)
    driver.execute_lambda(lambda executor: executor.execute_statement(update_query,
 parameter))
    logger.info('Updated record.')
```

```
    search_query = 'SELECT VALUE Name FROM {}'.format(TABLE_NAME)
    cursor = driver.execute_lambda(lambda executor:
 executor.execute_statement(search_query))

    for c in cursor:
        if not isinstance(c, ion_object):
            raise TypeError('The queried value is not an instance of
 {}'.format(ion_object.__name__))

        if c.ion_type is not ion_type:
            raise TypeError('The queried value type does not match
 {}'.format(ion_type))

    logger.info("Successfully verified value is instance of '{}' with type
 '{}'.".format(ion_object.__name__, ion_type))
    return cursor


def delete_table(driver, table_name):
    """
    Delete a table.

    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: An instance of the QldbDriver class.

    :type table_name: str
    :param table_name: Name of the table to delete.

    :rtype: int
    :return: The number of changes to the database.
    """
    logger.info("Deleting '{}' table...".format(table_name))
    cursor = driver.execute_lambda(lambda executor: executor.execute_statement('DROP
 TABLE {}'.format(table_name)))
    logger.info("'{}' table successfully deleted.".format(table_name))
    return len(list(cursor))


def insert_and_verify_ion_types(driver):
    """
    Insert all the supported Ion types and Python values that are convertible to Ion
 into a ledger and verify that they
    are stored and can be retrieved properly, retaining their original properties.
```

```
    :type driver: :py:class:`pyqldb.driver.qldb_driver.QldbDriver`
    :param driver: A QLDB Driver object.
    """
    python_bytes = str.encode('hello')
    python_bool = True
    python_float = float('0.2')
    python_decimal = Decimal('0.1')
    python_string = "string"
    python_int = 1
    python_null = None
    python_datetime = datetime(2016, 12, 20, 5, 23, 43)
    python_list = [1, 2]
    python_dict = {"brand": "Ford"}

    ion_clob = convert_object_to_ion(loads('{{"This is a CLOB of text."}}'))
    ion_blob = convert_object_to_ion(python_bytes)
    ion_bool = convert_object_to_ion(python_bool)
    ion_decimal = convert_object_to_ion(python_decimal)
    ion_float = convert_object_to_ion(python_float)
    ion_int = convert_object_to_ion(python_int)
    ion_list = convert_object_to_ion(python_list)
    ion_null = convert_object_to_ion(python_null)
    ion_sexp = convert_object_to_ion(loads('(cons 1 2)'))
    ion_string = convert_object_to_ion(python_string)
    ion_struct = convert_object_to_ion(python_dict)
    ion_symbol = convert_object_to_ion(SymbolToken(text='abc', sid=123))
    ion_timestamp = convert_object_to_ion(python_datetime)

    ion_null_clob = convert_object_to_ion(loads('null.clob'))
    ion_null_blob = convert_object_to_ion(loads('null.blob'))
    ion_null_bool = convert_object_to_ion(loads('null.bool'))
    ion_null_decimal = convert_object_to_ion(loads('null.decimal'))
    ion_null_float = convert_object_to_ion(loads('null.float'))
    ion_null_int = convert_object_to_ion(loads('null.int'))
    ion_null_list = convert_object_to_ion(loads('null.list'))
    ion_null_sexp = convert_object_to_ion(loads('null.sexp'))
    ion_null_string = convert_object_to_ion(loads('null.string'))
    ion_null_struct = convert_object_to_ion(loads('null.struct'))
    ion_null_symbol = convert_object_to_ion(loads('null.symbol'))
    ion_null_timestamp = convert_object_to_ion(loads('null.timestamp'))

    create_table(driver, TABLE_NAME)
    insert_documents(driver, TABLE_NAME, [{'Name': 'val'}])
    update_record_and_verify_type(driver, python_bytes, IonPyBytes, IonType.BLOB)
```

```
    update_record_and_verify_type(driver, python_bool, IonPyBool, IonType.BOOL)
    update_record_and_verify_type(driver, python_float, IonPyFloat, IonType.FLOAT)
    update_record_and_verify_type(driver, python_decimal, IonPyDecimal,
IonType.DECIMAL)
    update_record_and_verify_type(driver, python_string, IonPyText, IonType.STRING)
    update_record_and_verify_type(driver, python_int, IonPyInt, IonType.INT)
    update_record_and_verify_type(driver, python_null, IonPyNull, IonType.NULL)
    update_record_and_verify_type(driver, python_datetime, IonPyTimestamp,
IonType.TIMESTAMP)
    update_record_and_verify_type(driver, python_list, IonPyList, IonType.LIST)
    update_record_and_verify_type(driver, python_dict, IonPyDict, IonType.STRUCT)
    update_record_and_verify_type(driver, ion_clob, IonPyBytes, IonType.CLOB)
    update_record_and_verify_type(driver, ion_blob, IonPyBytes, IonType.BLOB)
    update_record_and_verify_type(driver, ion_bool, IonPyBool, IonType.BOOL)
    update_record_and_verify_type(driver, ion_decimal, IonPyDecimal,
IonType.DECIMAL)
    update_record_and_verify_type(driver, ion_float, IonPyFloat, IonType.FLOAT)
    update_record_and_verify_type(driver, ion_int, IonPyInt, IonType.INT)
    update_record_and_verify_type(driver, ion_list, IonPyList, IonType.LIST)
    update_record_and_verify_type(driver, ion_null, IonPyNull, IonType.NULL)
    update_record_and_verify_type(driver, ion_sexp, IonPyList, IonType.SEXP)
    update_record_and_verify_type(driver, ion_string, IonPyText, IonType.STRING)
    update_record_and_verify_type(driver, ion_struct, IonPyDict, IonType.STRUCT)
    update_record_and_verify_type(driver, ion_symbol, IonPySymbol, IonType.SYMBOL)
    update_record_and_verify_type(driver, ion_timestamp, IonPyTimestamp,
IonType.TIMESTAMP)
    update_record_and_verify_type(driver, ion_null_clob, IonPyNull, IonType.CLOB)
    update_record_and_verify_type(driver, ion_null_blob, IonPyNull, IonType.BLOB)
    update_record_and_verify_type(driver, ion_null_bool, IonPyNull, IonType.BOOL)
    update_record_and_verify_type(driver, ion_null_decimal, IonPyNull,
IonType.DECIMAL)
    update_record_and_verify_type(driver, ion_null_float, IonPyNull, IonType.FLOAT)
    update_record_and_verify_type(driver, ion_null_int, IonPyNull, IonType.INT)
    update_record_and_verify_type(driver, ion_null_list, IonPyNull, IonType.LIST)
    update_record_and_verify_type(driver, ion_null_sexp, IonPyNull, IonType.SEXP)
    update_record_and_verify_type(driver, ion_null_string, IonPyNull,
IonType.STRING)
    update_record_and_verify_type(driver, ion_null_struct, IonPyNull,
IonType.STRUCT)
    update_record_and_verify_type(driver, ion_null_symbol, IonPyNull,
IonType.SYMBOL)
    update_record_and_verify_type(driver, ion_null_timestamp, IonPyNull,
IonType.TIMESTAMP)
    delete_table(driver, TABLE_NAME)
```

```
def main(ledger_name=Constants.LEDGER_NAME):
    """
    Insert all the supported Ion types and Python values that are convertible to Ion
 into a ledger and verify that they
    are stored and can be retrieved properly, retaining their original properties.
    """
    try:
        with create_qldb_driver(ledger_name) as driver:
            insert_and_verify_ion_types(driver)
    except Exception as e:
        logger.exception('Error updating and validating Ion types.')
        raise e


if __name__ == '__main__':
    main()
```

Amazon QLDB API 참조

이 장에서는 HTTP, AWS Command Line Interface(AWS CLI), 또는 AWS SDK를 통해 액세스할 수 있는 Amazon QLDB의 하위 수준 API 작업에 대해 설명합니다.

- Amazon QLDB - QLDB 리소스 관리 API(컨트롤 플레인이라고도 함). 이 API는 원장 리소스 관리 및 비트랜잭션 데이터 작업에만 사용됩니다. 이러한 작업을 사용하여 원장을 생성, 삭제, 설명, 나열 및 업데이트할 수 있습니다. 또한 저널 데이터를 암호화 방식으로 검증하고 저널 블록을 내보내거나 스트리밍할 수 있습니다.
- Amazon QLDB 세션 - QLDB 트랜잭션 데이터 API. 이 API를 사용하여 PartiQL 문을 사용하여 원장에서 데이터 트랜잭션을 실행할 수 있습니다.

> **Important**
>
> QLDB 세션 API와 직접 상호 작용하는 대신 QLDB 드라이버 또는 QLDB 쉘을 사용하여 원장에서 데이터 트랜잭션을 실행하는 것이 좋습니다.
>
> - AWS SDK로 작업하는 경우 QLDB 드라이버를 사용하세요. 드라이버는 이 QLDB 세션 데이터 API 위에 높은 수준의 추상화 계층을 제공하고 사용자를 대신하여 SendCommand 작업을 관리합니다. 지원되는 프로그래밍 언어에 대한 자세한 내용 및 목록은 드라이버 시작하기 섹션을 참조하세요.
> - AWS CLI를 사용하여 작업하는 경우 QLDB 쉘을 사용하세요. 쉘은 QLDB 드라이버를 사용하여 원장과 상호 작용하는 명령줄 인터페이스입니다. 자세한 내용은 Amazon QLDB 쉘 사용(데이터 API만 해당)을 참조하세요.

주제

작업

Amazon QLDB에서는 다음 작업을 지원합니다.

- CancelJournalKinesisStream
- CreateLedger
- DeleteLedger
- DescribeJournalKinesisStream
- DescribeJournalS3Export
- DescribeLedger
- ExportJournalToS3
- GetBlock
- GetDigest
- GetRevision
- ListJournalKinesisStreamsForLedger
- ListJournalS3Exports
- ListJournalS3ExportsForLedger
- ListLedgers
- ListTagsForResource
- StreamJournalToKinesis
- TagResource
- UntagResource
- UpdateLedger
- UpdateLedgerPermissionsMode

Amazon QLDB 세션에서는 다음 작업을 지원합니다.

- SendCommand

Amazon QLDB

Amazon QLDB에서는 다음 작업을 지원합니다.

- CancelJournalKinesisStream
- CreateLedger
- DeleteLedger

- DescribeJournalKinesisStream
- DescribeJournalS3Export
- DescribeLedger
- ExportJournalToS3
- GetBlock
- GetDigest
- GetRevision
- ListJournalKinesisStreamsForLedger
- ListJournalS3Exports
- ListJournalS3ExportsForLedger
- ListLedgers
- ListTagsForResource
- StreamJournalToKinesis
- TagResource
- UntagResource
- UpdateLedger
- UpdateLedgerPermissionsMode

CancelJournalKinesisStream

서비스: Amazon QLDB

지정된 Amazon QLDB 저널 스트림을 종료합니다. 스트림을 취소하려면 스트림의 현재 상태가 `ACTIVE`이어야 합니다.

스트리밍을 취소한 후에는 다시 시작할 수 없습니다. 취소된 QLDB 스트림 리소스는 7일의 보존 기간이 적용되므로 이 한도가 만료되면 자동으로 삭제됩니다.

Request Syntax

```
DELETE /ledgers/name/journal-kinesis-streams/streamId HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

원장의 명칭입니다.

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

streamId

취소할 QLDB 저널 스트림의 UUID(Base62 인코딩된 텍스트로 표시됨)입니다.

길이 제약 조건: 고정 길이는 22입니다.

패턴: `^[A-Za-z-0-9]+$`

필수 사항 여부: Yes

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
```

```
Content-type: application/json

{
   "StreamId": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

StreamId

취소된 QLDB 저널 스트림의 UUID(Base62 인코딩된 텍스트)입니다.

타입: 문자열

길이 제약 조건: 고정 길이는 22입니다.

패턴: ^[A-Za-z-0-9]+$

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidParameterException

요청에서 하나 이상의 파라미터가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

ResourcePreconditionNotMetException

조건이 미리 충족되지 않아 작업이 실패했습니다.

HTTP 상태 코드: 412

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

CreateLedger

서비스: Amazon QLDB

현재 지역의 새 원장을 생성합니다. AWS 계정

Request Syntax

```
POST /ledgers HTTP/1.1
Content-type: application/json

{
   "DeletionProtection": boolean,
   "KmsKey": "string",
   "Name": "string",
   "PermissionsMode": "string",
   "Tags": {
      "string" : "string"
   }
}
```

URI 요청 파라미터

요청은 URI 파라미터를 사용하지 않습니다.

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

DeletionProtection

사용자에 의해 삭제되지 않도록 원장을 보호할지 여부를 지정합니다. 원장 생성 중 정의하지 않을 경우 이 기능이 기본적으로 활성화됩니다(`true`).

삭제 방지가 활성화된 경우 원장을 삭제하려면 먼저 이 기능을 비활성화해야 합니다. `UpdateLedger` 작업을 호출하여 이 파라미터를 `false`로 설정하면 비활성화할 수 있습니다.

타입: 부울

필수 항목 여부: 아니요

KmsKey

원장에 저장된 데이터를 암호화하는 데 사용할 키 in AWS Key Management Service (AWS KMS). 자세한 내용을 알아보려면 Amazon QLDB 개발자 안내서의 저장된 암호화를 참조하세요.

다음 옵션 중 하나를 사용하여 이 파라미터를 지정합니다.

- `AWS_OWNED_KMS_KEY`: 본인 대신 소유하고 AWS 관리하는 AWS KMS 키를 사용하십시오.
- 미정: 기본적으로 AWS 소유한 KMS 키를 사용합니다.
- 유효한 대칭 고객 관리형 KMS 키: 사용자가 생성, 소유 및 관리하는 계정의 지정된 대칭 암호화 KMS 키를 사용합니다.

  Amazon QLDB에서는 비대칭 키가 지원되지 않습니다. 자세한 내용은 개발자 안내서의 대칭 및 비대칭 키 사용을 참조하십시오. AWS Key Management Service

고객 관리형 KMS 키를 지정하려면 해당 키 ID, Amazon 리소스 이름(ARN), 별칭 이름 또는 별칭 ARN을 사용할 수 있습니다. 별칭 이름을 사용할 때 `"alias/"`를 접두사를 사용합니다. 다른 AWS 계정키를 지정하려면 키 ARN 또는 별칭 ARN을 사용해야 합니다.

예:

- 키 ID: `1234abcd-12ab-34cd-56ef-1234567890ab`
- 키 ARN: `arn:aws:kms:us-east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab`
- 별칭 이름: `alias/ExampleAlias`
- 별칭 ARN: `arn:aws:kms:us-east-2:111122223333:alias/ExampleAlias`

자세한 내용은 개발자 안내서의 키 식별자 (KeyId) 를 참조하십시오. AWS Key Management Service

타입: 문자열

길이 제약 조건: 최대 길이는 1,600입니다.

필수 여부: 아니요

Name

생성할 원장의 이름입니다. 이름은 현재 지역의 모든 원장 중에서 고유해야 합니다 AWS 계정 .

원장 이름에 대한 이름 지정 제약 조건은 Amazon QLDB 개발자 안내서의 Amazon QLDB의 할당량에 정의되어 있습니다.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

PermissionsMode

생성하려는 원장에 할당할 권한 모드입니다. 이 파라미터는 다음 값 중 하나를 가질 수 있습니다.

- `ALLOW_ALL`: 원장에 대한 API 수준 세분화로 액세스 통제를 가능하게 하는 레거시 권한 모드입니다.

  이 모드를 사용하면 이 원장에 대한 SendCommand API 권한이 있는 사용자가 지정된 원장의 모든 표에서 모든 PartiQL 명령(따라서 `ALLOW_ALL`)을 실행할 수 있습니다. 이 모드는 원장에 대해 생성하는 모든 표 수준 또는 명령 수준 IAM 권한 정책을 무시합니다.

- `STANDARD`: (권장) 원장, 테이블 및 PartiQL 명령에 대해 보다 세분화된 액세스 제어를 가능하게 하는 권한 모드입니다..

  기본적으로 이 모드는 이 원장의 모든 표에서 PartiQL 명령을 실행하려는 모든 사용자 요청을 거부합니다. PartiQL 명령 실행을 허용하려면 원장에 대한 SendCommand API 권한 외에도 특정 테이블 리소스 및 PartiQL 작업에 대한 IAM 권한 정책을 생성해야 합니다. 자세한 내용은 Amazon QLDB 개발자 안내서의 표준 권한 모드로 시작하기를 참조하세요.

> **Note**
>
> 원장 데이터의 보안을 극대화하려면 `STANDARD` 권한 모드를 사용하는 것이 좋습니다.

타입: 문자열

유효 값: `ALLOW_ALL | STANDARD`

필수 사항 여부: 예

Tags

생성할 원장에 태그로 추가하려는 키-값 쌍입니다. 태그 키는 대소문자를 구별합니다. 태그 값은 대소문자를 구분하며 null일 수 있습니다.

유형: 문자열 간 맵

맵 항목: 최소 항목 수는 0개입니다. 최대 항목 수 200개.

키 길이 제약 조건: 최소 길이는 1. 최대 길이 128.

값 길이 제약 조건: 최소 길이는 0입니다. 최대 길이는 256입니다.

필수 여부: 아니요

응답 구문

```
HTTP/1.1 200
Content-type: application/json

{
   "Arn": "string",
   "CreationDateTime": number,
   "DeletionProtection": boolean,
   "KmsKeyArn": "string",
   "Name": "string",
   "PermissionsMode": "string",
   "State": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

Arn

원장의 Amazon 리소스 이름(ARN)입니다.

타입: 문자열

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

CreationDateTime

원장이 생성된 날짜와 시간(epoch 시간 형식)입니다. (epoch 시간 형식은 UTC 기준으로 1970년 1월 1일 자정 12:00:00부터 경과된 초 단위의 시간입니다.)

유형: 타임스탬프

DeletionProtection

사용자에 의해 삭제되지 않도록 원장을 보호할지 여부를 지정합니다. 원장 생성 중 정의하지 않을 경우 이 기능이 기본적으로 활성화됩니다(`true`).

삭제 방지가 활성화된 경우 원장을 삭제하려면 먼저 이 기능을 비활성화해야 합니다. `UpdateLedger` 작업을 호출하여 이 파라미터를 `false`로 설정하면 비활성화할 수 있습니다.

타입: 부울

KmsKeyArn

원장이 유휴 시 암호화에 사용하는 고객 관리형 KMS 키의 ARN입니다. 이 매개변수가 정의되지 않은 경우 원장은 AWS 소유한 KMS 키를 암호화에 사용합니다.

타입: 문자열

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

Name

원장의 명칭입니다.

타입: 문자열

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

PermissionsMode

생성한 원장의 권한 모드입니다.

타입: 문자열

유효 값: `ALLOW_ALL | STANDARD`

State

원장의 현재 상태입니다.

타입: 문자열

유효 값: `CREATING | ACTIVE | DELETING | DELETED`

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidParameterException

요청에서 하나 이상의 파라미터가 유효하지 않습니다.

HTTP 상태 코드: 400

LimitExceededException

허용되는 최대 리소스 수 제한에 도달했습니다.

HTTP 상태 코드: 400

ResourceAlreadyExistsException

지정된 리소스가 이미 있습니다.

HTTP 상태 코드: 409

ResourceInUseException

지금은 지정된 리소스를 수정할 수 없습니다.

HTTP 상태 코드: 409

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

DeleteLedger

서비스: Amazon QLDB

원장과 모든 해당 콘텐츠를 삭제합니다. 이 작업은 되돌릴 수 없습니다.

삭제 방지가 활성화된 경우 원장을 삭제하려면 먼저 이 기능을 비활성화해야 합니다. `UpdateLedger` 작업을 호출하여 이 파라미터를 `false`로 설정하면 비활성화할 수 있습니다.

Request Syntax

```
DELETE /ledgers/name HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

삭제할 원장의 이름입니다.

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
```

Response Elements

작업이 성공하면 서비스가 비어 있는 HTTP 본문과 함께 HTTP 200 응답을 반송합니다.

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidParameterException

요청에서 하나 이상의 파라미터가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceInUseException

지금은 지정된 리소스를 수정할 수 없습니다.

HTTP 상태 코드: 409

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

ResourcePreconditionNotMetException

조건이 미리 충족되지 않아 작업이 실패했습니다.

HTTP 상태 코드: 412

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

DescribeJournalKinesisStream

서비스: Amazon QLDB

지정된 Amazon QLDB 저널 스트림에 대한 세부 정보를 반환합니다. 출력에는 Amazon 리소스 이름(ARN), 스트림 이름, 현재 상태, 생성 시간, 원본 스트림 생성 요청의 파라미터가 포함됩니다.

이 작업을 수행해도 만료된 저널 스트림은 반환되지 않습니다. 자세한 내용은 Amazon QLDB 개발자 안내서의 터미널 스트림 만료를 참조하십시오.

Request Syntax

```
GET /ledgers/name/journal-kinesis-streams/streamId HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

원장의 명칭입니다.

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

streamId

설명할 QLDB 저널 스트림의 UUID(Base62 인코딩된 텍스트로 표시됨)입니다.

길이 제약 조건: 고정 길이는 22입니다.

패턴: `^[A-Za-z-0-9]+$`

필수 사항 여부: Yes

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
```

```
Content-type: application/json

{
   "Stream": {
      "Arn": "string",
      "CreationTime": number,
      "ErrorCause": "string",
      "ExclusiveEndTime": number,
      "InclusiveStartTime": number,
      "KinesisConfiguration": {
         "AggregationEnabled": boolean,
         "StreamArn": "string"
      },
      "LedgerName": "string",
      "RoleArn": "string",
      "Status": "string",
      "StreamId": "string",
      "StreamName": "string"
   }
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

Stream

`DescribeJournalS3Export` 요청에 의해 반환된 QLDB 저널 스트림에 대한 정보입니다.

유형: JournalKinesisStreamDescription객체

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidParameterException

요청에서 하나 이상의 파라미터가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

ResourcePreconditionNotMetException

조건이 미리 충족되지 않아 작업이 실패했습니다.

HTTP 상태 코드: 412

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

DescribeJournalS3Export

서비스: Amazon QLDB

원장 이름, 내보내기 ID, 생성 시간, 현재 상태, 원본 내보내기 생성 요청의 파라미터 등 저널 내보내기 작업에 대한 정보를 반환합니다.

이 작업을 수행해도 만료된 내보내기 작업은 반환되지 않습니다. 자세한 내용을 알아보려면 Amazon QLDB 개발자 안내서의 내보내기 작업 만료를 참조하십시오.

지정된 `ExportId`의 내보내기 작업이 존재하지 않으면 `ResourceNotFoundException`을 발생시킵니다.

지정된 `Name`의 원장이 존재하지 않으면 `ResourceNotFoundException`을 발생시킵니다.

Request Syntax

```
GET /ledgers/name/journal-s3-exports/exportId HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

exportId

설명할 저널 내보내기 작업의 UUID(Base62 인코딩된 텍스트로 표시됨)입니다.

길이 제약 조건: 고정 길이는 22입니다.

패턴: `^[A-Za-z-0-9]+$`

필수 사항 여부: Yes

name

원장의 명칭입니다.

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "ExportDescription": {
      "ExclusiveEndTime": number,
      "ExportCreationTime": number,
      "ExportId": "string",
      "InclusiveStartTime": number,
      "LedgerName": "string",
      "OutputFormat": "string",
      "RoleArn": "string",
      "S3ExportConfiguration": {
         "Bucket": "string",
         "EncryptionConfiguration": {
            "KmsKeyArn": "string",
            "ObjectEncryptionType": "string"
         },
         "Prefix": "string"
      },
      "Status": "string"
   }
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

ExportDescription

`DescribeJournalS3Export` 요청에 의해 반환된 저널 내보내기 작업에 대한 정보입니다.

유형: JournalS3ExportDescription객체

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

DescribeLedger

서비스: Amazon QLDB

상태, 권한 모드, 저장 시 암호화 설정, 생성 날짜 등 원장에 대한 정보를 반환합니다.

Request Syntax

```
GET /ledgers/name HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

설명할 원장의 이름입니다.

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "Arn": "string",
   "CreationDateTime": number,
   "DeletionProtection": boolean,
   "EncryptionDescription": {
      "EncryptionStatus": "string",
      "InaccessibleKmsKeyDateTime": number,
      "KmsKeyArn": "string"
   },
   "Name": "string",
```

```
    "PermissionsMode": "string",
    "State": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

Arn

원장의 Amazon 리소스 이름(ARN)입니다.

타입: 문자열

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

CreationDateTime

원장이 생성된 날짜와 시간(epoch 시간 형식)입니다. (epoch 시간 형식은 UTC 기준으로 1970년 1월 1일 자정 12:00:00부터 경과된 초 단위의 시간입니다.)

유형: 타임스탬프

DeletionProtection

사용자에 의해 삭제되지 않도록 원장을 보호할지 여부를 지정합니다. 원장 생성 중 정의하지 않을 경우 이 기능이 기본적으로 활성화됩니다(`true`).

삭제 방지가 활성화된 경우 원장을 삭제하려면 먼저 이 기능을 비활성화해야 합니다. `UpdateLedger` 작업을 호출하여 이 파라미터를 `false`로 설정하면 비활성화할 수 있습니다.

타입: 부울

EncryptionDescription

원장의 저장 데이터 암호화에 대한 정보. 여기에는 현재 상태, AWS KMS 키, 키에 액세스할 수 없게 된 시점 (오류 발생 시) 이 포함됩니다. 이 매개변수가 정의되지 않은 경우 원장은 AWS 소유한 KMS 키를 암호화에 사용합니다.

유형: LedgerEncryptionDescription객체

Name

원장의 명칭입니다.

타입: 문자열

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

PermissionsMode

원장의 권한 모드입니다.

타입: 문자열

유효 값: `ALLOW_ALL | STANDARD`

State

원장의 현재 상태입니다.

타입: 문자열

유효 값: `CREATING | ACTIVE | DELETING | DELETED`

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidParameterException

요청에서 하나 이상의 파라미터가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET

- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

ExportJournalToS3

서비스: Amazon QLDB

원장에서 날짜 및 시간 범위 내의 저널 콘텐츠를 지정된 Amazon Simple Storage Service(S3) 버킷에 내보냅니다. 저널 내보내기 작업은 Amazon Ion 형식의 텍스트 또는 이진 표현이나 JSON Lines 텍스트 형식으로 데이터 객체를 쓸 수 있습니다.

지정된 `Name`의 원장이 존재하지 않으면 `ResourceNotFoundException`을 발생시킵니다.

지정된 `Name`의 원장이 `CREATING` 상태이면 `ResourcePreconditionNotMetException`을 발생시킵니다.

각 원장에 대해 최대 두 개의 동시 저널 내보내기 요청을 시작할 수 있습니다. 이 한도를 초과하면 저널 내보내기 요청에 `LimitExceededException`가 발생합니다.

Request Syntax

```
POST /ledgers/name/journal-s3-exports HTTP/1.1
Content-type: application/json

{
   "ExclusiveEndTime": number,
   "InclusiveStartTime": number,
   "OutputFormat": "string",
   "RoleArn": "string",
   "S3ExportConfiguration": {
      "Bucket": "string",
      "EncryptionConfiguration": {
         "KmsKeyArn": "string",
         "ObjectEncryptionType": "string"
      },
      "Prefix": "string"
   }
}
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

원장의 명칭입니다.

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

ExclusiveEndTime

내보내기할 저널 콘텐츠 범위에 대한 불포함 종료 날짜 및 시간.

ISO 8601은 `ExclusiveEndTime` 날짜 및 시간 형식이어야 하며 협정 세계시(UTC)여야 합니다. 예를 들면 `2019-06-13T21:36:34Z`입니다.

`ExclusiveEndTime`는 현재 UTC 날짜 및 시간보다 작거나 같아야 합니다.

유형: 타임스탬프

필수 여부: 예

InclusiveStartTime

내보기할 저널 콘텐츠 범위에 대한 포함 시작 날짜 및 시간.

ISO 8601은 `InclusiveStartTime` 날짜 및 시간 형식이어야 하며 협정 세계시(UTC)여야 합니다. 예를 들면 `2019-06-13T21:36:34Z`입니다.

`InclusiveStartTime`는 `ExclusiveEndTime` 이전이어야 합니다.

원장의 `CreationDateTime` 이전인 `InclusiveStartTime`을 제공하는 경우, Amazon QLDB는 기본값을 원장의 `CreationDateTime`으로 설정합니다.

유형: 타임스탬프

필수 여부: 예

OutputFormat

내보내기할 저널 데이터의 출력 형식. 저널 내보내기 작업은 Amazon Ion 형식의 텍스트 또는 이진 표현이나 JSON Lines 텍스트 형식으로 데이터 객체를 쓸 수 있습니다.

기본값: `ION_TEXT`

JSON Lines 형식에서 내보낸 데이터 객체의 각 저널 블록은 줄 바꿈으로 구분된 유효한 JSON 객체입니다. 이 형식을 사용하여 JSON 내보내기를 Amazon Athena 및 AWS Glue 와 같은 분석 도구와 직접 통합할 수 있습니다. 이러한 서비스는 줄바꿈으로 구분된 JSON을 자동으로 파싱할 수 있기 때문입니다.

타입: 문자열

유효 값: `ION_BINARY | ION_TEXT | JSON`

필수 여부: 아니요

RoleArn

다음을 수행하기 위해 저널 내보내기 작업에 대한 QLDB 권한을 부여하는 IAM 역할의 Amazon 리소스 이름(ARN)입니다.

- Amazon S3 버킷에 객체를 작성합니다.
- (선택 사항) 내보낸 데이터의 서버 측 암호화에 고객 관리 키 in AWS Key Management Service (AWS KMS) 을 사용하십시오.

저널 내보내기를 요청할 때 QLDB에 역할을 전달하려면 IAM 역할 리소스에서 `iam:PassRole` 작업을 수행할 수 있는 권한이 있어야 합니다. 이는 모든 저널 내보내기 요청에 필요합니다.

타입: 문자열

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

필수 여부: 예

S3ExportConfiguration

내보내기 요청에 대한 Amazon S3 버킷 대상의 구성 설정입니다.

유형: S3ExportConfiguration객체

필수 여부: 예

응답 구문

```
HTTP/1.1 200
Content-type: application/json
```

```
{
   "ExportId": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

ExportId

QLDB가 각 저널 내보내기 작업에 할당하는 UUID(Base62로 인코딩된 텍스트로 표시됨).

ExportId를 사용하여 DescribeJournalS3Export를 호출하여 내보내기 요청을 설명하고 작업 상태를 확인할 수 있습니다.

타입: 문자열

길이 제약 조건: 고정 길이는 22입니다.

패턴: ^[A-Za-z-0-9]+$

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

ResourcePreconditionNotMetException

조건이 미리 충족되지 않아 작업이 실패했습니다.

HTTP 상태 코드: 412

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

GetBlock

서비스: Amazon QLDB

저널의 지정된 주소에 있는 블록 객체를 반환합니다. 또한 `DigestTipAddress`가 제공된 경우 검증을 위해 지정된 블록의 증명을 반환합니다.

블록의 데이터 콘텐츠에 대한 자세한 내용은 Amazon QLDB 개발자 안내서의 저널 콘텐츠를 참조하세요.

지정된 원장이 존재하지 않거나 `DELETING` 상태에 있는 경우 `ResourceNotFoundException`이 발생합니다.

지정된 원장이 `CREATING` 상태에 있으면 `ResourcePreconditionNotMetException`이 발생합니다.

지정된 주소를 가진 블록이 없으면 `InvalidParameterException`이 발생합니다.

Request Syntax

```
POST /ledgers/name/block HTTP/1.1
Content-type: application/json

{
   "BlockAddress": {
      "IonText": "string"
   },
   "DigestTipAddress": {
      "IonText": "string"
   }
}
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

원장의 명칭입니다.

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

BlockAddress

요청하려는 블록의 위치. 주소는 `strandId` 및 `sequenceNo`라는 두 개의 필드로 구성된 Amazon Ion 구조입니다.

예를 들면 `{strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:14}`입니다.

유형: ValueHolder객체

필수 여부: 예

DigestTipAddress

증명을 요청할 다이제스트에 포함된 최신 블록 위치. 주소는 `strandId` 및 `sequenceNo`라는 두 개의 필드로 구성된 Amazon Ion 구조입니다.

예를 들면 `{strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:49}`입니다.

유형: ValueHolder객체

필수 항목 여부: 아니요

응답 구문

```
HTTP/1.1 200
Content-type: application/json

{
   "Block": {
      "IonText": "string"
   },
   "Proof": {
      "IonText": "string"
   }
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

Block

Amazon Ion 형식의 블록 데이터 객체입니다.

유형: ValueHolder객체

Proof

`GetBlock` 요청을 통해 반환된 Amazon Ion 형식의 증명 객체입니다. 증명은 지정된 블록부터 시작하여 Merkle 트리를 사용하여 지정된 다이제스트를 다시 계산하는 데 필요한 해시 값 목록을 포함합니다.

유형: ValueHolder객체

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidParameterException

요청에서 하나 이상의 파라미터가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

ResourcePreconditionNotMetException

조건이 미리 충족되지 않아 작업이 실패했습니다.

HTTP 상태 코드: 412

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

GetDigest

서비스: Amazon QLDB

저널에서 가장 최근에 커밋된 블록의 원장 다이제스트를 반환합니다. 응답에는 256비트 해시 값과 블록 주소가 포함됩니다.

Request Syntax

```
POST /ledgers/name/digest HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

원장의 명칭입니다.

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "Digest": blob,
   "DigestTipAddress": {
      "IonText": "string"
   }
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

Digest

`GetDigest` 요청에서 반환된 다이제스트를 나타내는 256비트 해시 값입니다.

타입: Base64로 인코딩된 이진 데이터 객체

길이 제약 조건: 고정 길이는 32입니다.

DigestTipAddress

요청한 다이제스트에 포함된 최신 블록 위치. 주소는 `strandId` 및 `sequenceNo`라는 두 개의 필드로 구성된 Amazon Ion 구조입니다.

유형: ValueHolder객체

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidParameterException

요청에서 하나 이상의 파라미터가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

ResourcePreconditionNotMetException

조건이 미리 충족되지 않아 작업이 실패했습니다.

HTTP 상태 코드: 412

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

GetRevision

서비스: Amazon QLDB

지정된 문서 ID 및 블록 주소의 수정본 데이터 객체를 반환합니다. 또한 `DigestTipAddress`가 제공된 경우 검증을 위해 지정된 수정본의 증명을 반환합니다.

Request Syntax

```
POST /ledgers/name/revision HTTP/1.1
Content-type: application/json

{
   "BlockAddress": {
      "IonText": "string"
   },
   "DigestTipAddress": {
      "IonText": "string"
   },
   "DocumentId": "string"
}
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

원장의 명칭입니다.

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

BlockAddress

검증할 문서 수정본의 블록 위치. 주소는 `strandId` 및 `sequenceNo`라는 두 개의 필드로 구성된 Amazon Ion 구조입니다.

예를 들면 {strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:14}입니다.

유형: ValueHolder객체

필수 여부: 예

DigestTipAddress

증명을 요청할 다이제스트에 포함된 최신 블록 위치. 주소는 strandId 및 sequenceNo라는 두 개의 필드로 구성된 Amazon Ion 구조입니다.

예를 들면 {strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:49}입니다.

유형: ValueHolder객체

필수 항목 여부: 아니요

DocumentId

검증할 문서의 UUID(Base62로 인코딩된 텍스트로 표시).

타입: 문자열

길이 제약 조건: 고정 길이는 22입니다.

패턴: ^[A-Za-z-0-9]+$

필수 항목 여부: 예

응답 구문

```
HTTP/1.1 200
Content-type: application/json

{
   "Proof": {
      "IonText": "string"
   },
   "Revision": {
      "IonText": "string"
   }
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

Proof

`GetRevision` 요청을 통해 반환된 Amazon Ion 형식의 증명 객체입니다. 증명은 지정된 문서 개정부터 시작하여 Merkle 트리를 사용하여 지정된 다이제스트를 다시 계산하는 데 필요한 해시 값 목록을 포함합니다.

유형: ValueHolder객체

Revision

Amazon Ion 형식의 문서 수정본 데이터 객체입니다.

유형: ValueHolder객체

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidParameterException

요청에서 하나 이상의 파라미터가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

ResourcePreconditionNotMetException

조건이 미리 충족되지 않아 작업이 실패했습니다.

HTTP 상태 코드: 412

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

ListJournalKinesisStreamsForLedger

서비스: Amazon QLDB

주어진 원장에 대한 모든 Amazon QLDB 저널 스트림을 반환합니다.

이 작업을 수행해도 만료된 저널 스트림은 반환되지 않습니다. 자세한 내용은 Amazon QLDB 개발자 안내서의 터미널 스트림 만료를 참조하십시오.

이 작업을 수행하면 최대 `MaxResults`개의 항목이 반환됩니다. 페이지가 매겨져 있으므로 `ListJournalKinesisStreamsForLedger`를 여러 번 호출하여 모든 항목을 검색할 수 있습니다.

Request Syntax

```
GET /ledgers/name/journal-kinesis-streams?max_results=MaxResults&next_token=NextToken
 HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

원장의 명칭입니다.

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

MaxResults

단일 `ListJournalKinesisStreamsForLedger` 요청에서 반환할 결과의 최대 수입니다. (반환되는 결과의 실제 수는 더 작을 수 있습니다.)

유효 범위: 최소값은 1입니다. 최댓값은 100입니다.

NextToken

페이지 매김 토큰으로, 다음 결과 페이지를 검색할 것임을 나타냅니다. 이전 `ListJournalKinesisStreamsForLedger` 호출의 응답에서 `NextToken`의 값을 수신한 경우, 그 값을 여기에 입력으로 사용해야 합니다.

길이 제약: 최소 길이는 4입니다. 최대 길이는 1024입니다.

패턴: ^[A-Za-z-0-9+/=]+$

요청 본문

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "NextToken": "string",
   "Streams": [
      {
         "Arn": "string",
         "CreationTime": number,
         "ErrorCause": "string",
         "ExclusiveEndTime": number,
         "InclusiveStartTime": number,
         "KinesisConfiguration": {
            "AggregationEnabled": boolean,
            "StreamArn": "string"
         },
         "LedgerName": "string",
         "RoleArn": "string",
         "Status": "string",
         "StreamId": "string",
         "StreamName": "string"
      }
   ]
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

NextToken

- NextToken이 비어 있는 경우, 결과의 마지막 페이지가 처리된 것이며 더 이상 검색할 결과가 없습니다.

- `NextToken`이 비어 있지 않으면 더 많은 결과를 사용할 수 있습니다. 결과의 그 다음 페이지를 검색하려면 후속 `ListJournalKinesisStreamsForLedger` 호출에서 `NextToken`의 값을 사용합니다.

  타입: 문자열

  길이 제약: 최소 길이는 4입니다. 최대 길이는 1024입니다.

  패턴: `^[A-Za-z-0-9+/=]+$`

Streams

현재 주어진 원장과 연결된 QLDB 저널 스트림.

타입: JournalKinesisStreamDescription 객체 배열

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidParameterException

요청에서 하나 이상의 파라미터가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

ResourcePreconditionNotMetException

조건이 미리 충족되지 않아 작업이 실패했습니다.

HTTP 상태 코드: 412

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET

- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

ListJournalS3Exports

서비스: Amazon QLDB

현재 AWS 계정 및 리전과 관련된 모든 원장에 대한 모든 저널 내보내기 작업을 반환합니다.

이 작업은 최대 `MaxResults` 항목을 반환하며 `ListJournalS3Exports` 여러 번 호출하여 모든 항목을 검색할 수 있도록 페이지가 매겨집니다.

이 작업을 수행해도 만료된 내보내기 작업은 반환되지 않습니다. 자세한 내용을 알아보려면 Amazon QLDB 개발자 안내서의 내보내기 작업 만료를 참조하십시오.

Request Syntax

```
GET /journal-s3-exports?max_results=MaxResults&next_token=NextToken HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

MaxResults

단일 `ListJournalS3Exports` 요청에서 반환할 결과의 최대 수입니다. (반환되는 결과의 실제 수는 더 작을 수 있습니다.)

유효 범위: 최소값은 1입니다. 최댓값은 100입니다.

NextToken

페이지 매김 토큰으로, 다음 결과 페이지를 검색할 것임을 나타냅니다. 이전 `ListJournalS3Exports` 호출의 응답에서 `NextToken`의 값을 수신한 경우, 그 값을 여기에 입력으로 사용해야 합니다.

길이 제약: 최소 길이는 4입니다. 최대 길이는 1024입니다.

패턴: `^[A-Za-z-0-9+/=]+$`

요청 본문

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
```

```
Content-type: application/json

{
   "JournalS3Exports": [
      {
         "ExclusiveEndTime": number,
         "ExportCreationTime": number,
         "ExportId": "string",
         "InclusiveStartTime": number,
         "LedgerName": "string",
         "OutputFormat": "string",
         "RoleArn": "string",
         "S3ExportConfiguration": {
            "Bucket": "string",
            "EncryptionConfiguration": {
               "KmsKeyArn": "string",
               "ObjectEncryptionType": "string"
            },
            "Prefix": "string"
         },
         "Status": "string"
      }
   ],
   "NextToken": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

JournalS3Exports

현재 AWS 계정 및 리전과 관련된 모든 원장에 대한 저널 내보내기 작업입니다.

유형: JournalS3ExportDescription객체 어레이

NextToken

- NextToken이 비어 있는 경우, 결과의 마지막 페이지가 처리된 것이며 더 이상 검색할 결과가 없습니다.
- NextToken이 비어 있지 않으면 더 많은 결과를 입수할 수 있습니다. 결과의 그 다음 페이지를 검색하려면 후속 ListJournalS3Exports 호출에서 NextToken의 값을 사용합니다.

타입: 문자열

길이 제약: 최소 길이는 4입니다. 최대 길이는 1024입니다.

패턴: `^[A-Za-z-0-9+/=]+$`

Errors

모든 작업에서 발생하는 일반적인 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하세요.

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

ListJournalS3ExportsForLedger

서비스: Amazon QLDB

지정된 원장에 대해 저널 내보내기 작업을 모두 반환합니다.

이 작업은 최대 `MaxResults` 항목을 반환하며 `ListJournalS3ExportsForLedger` 여러 번 호출하여 모든 항목을 검색할 수 있도록 페이지가 매겨집니다.

이 작업을 수행해도 만료된 내보내기 작업은 반환되지 않습니다. 자세한 내용을 알아보려면 Amazon QLDB 개발자 안내서의 내보내기 작업 만료를 참조하십시오.

Request Syntax

```
GET /ledgers/name/journal-s3-exports?max_results=MaxResults&next_token=NextToken
 HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

MaxResults

단일 `ListJournalS3ExportsForLedger` 요청에서 반환할 결과의 최대 수입니다. (반환되는 결과의 실제 수는 더 작을 수 있습니다.)

유효 범위: 최소값은 1입니다. 최대값 100.

name

원장의 명칭입니다.

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

NextToken

페이지 매김 토큰으로, 다음 결과 페이지를 검색할 것임을 나타냅니다. 이전 `ListJournalS3ExportsForLedger` 호출의 응답에서 `NextToken`의 값을 수신한 경우, 그 값을 여기에 입력으로 사용해야 합니다.

길이 제약: 최소 길이는 4입니다. 최대 길이는 1024입니다.

패턴: ^[A-Za-z-0-9+/=]+$

요청 본문

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "JournalS3Exports": [
      {
         "ExclusiveEndTime": number,
         "ExportCreationTime": number,
         "ExportId": "string",
         "InclusiveStartTime": number,
         "LedgerName": "string",
         "OutputFormat": "string",
         "RoleArn": "string",
         "S3ExportConfiguration": {
            "Bucket": "string",
            "EncryptionConfiguration": {
               "KmsKeyArn": "string",
               "ObjectEncryptionType": "string"
            },
            "Prefix": "string"
         },
         "Status": "string"
      }
   ],
   "NextToken": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

JournalS3Exports

현재 지정된 원장과 연관된 저널 내보내기 작업입니다.

유형: JournalS3ExportDescription객체 어레이

NextToken

- NextToken이 비어 있는 경우, 결과의 마지막 페이지가 처리된 것이며 더 이상 검색할 결과가 없습니다.
- NextToken이 비어 있지 않으면 더 많은 결과를 입수할 수 있습니다. 결과의 그 다음 페이지를 검색하려면 후속 ListJournalS3ExportsForLedger 호출에서 NextToken의 값을 사용합니다.

타입: 문자열

길이 제약: 최소 길이는 4입니다. 최대 길이는 1024입니다.

패턴: ^[A-Za-z-0-9+/=]+$

Errors

모든 작업에서 발생하는 일반적인 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하세요.

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

ListLedgers

서비스: Amazon QLDB

현재 AWS 계정 및 지역과 관련된 모든 원장을 반환합니다.

이 작업은 최대 `MaxResults`개의 항목을 반환하며 `ListLedgers`를 여러 번 호출함으로써 모든 항목을 검색할 수 있도록 페이지가 매겨집니다.

Request Syntax

```
GET /ledgers?max_results=MaxResults&next_token=NextToken HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

MaxResults

단일 `ListLedgers` 요청에서 반환할 결과의 최대 수입니다. (반환되는 결과의 실제 수는 더 작을 수 있습니다.)

유효 범위: 최소값은 1입니다. 최댓값은 100입니다.

NextToken

페이지 매김 토큰으로, 다음 결과 페이지를 검색할 것임을 나타냅니다. 이전 `ListLedgers` 호출의 응답에서 `NextToken`의 값을 수신한 경우, 그 값을 여기에 입력으로 사용해야 합니다.

길이 제약: 최소 길이는 4입니다. 최대 길이는 1024입니다.

패턴: `^[A-Za-z-0-9+/=]+$`

요청 본문

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
```

```
   "Ledgers": [
      {
         "CreationDateTime": number,
         "Name": "string",
         "State": "string"
      }
   ],
   "NextToken": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

Ledgers

현재 AWS 계정 및 지역과 관련된 원장.

타입: LedgerSummary 객체 배열

NextToken

더 많은 결과가 있는지 여부를 나타내는 페이지 매김 토큰:

- NextToken이 비어 있는 경우, 결과의 마지막 페이지가 처리된 것이며 더 이상 검색할 결과가 없습니다.
- NextToken이 비어 있지 않으면 더 많은 결과를 입수할 수 있습니다. 결과의 그 다음 페이지를 검색하려면 후속 ListLedgers 호출에서 NextToken의 값을 사용합니다.

타입: 문자열

길이 제약: 최소 길이는 4입니다. 최대 길이는 1024입니다.

패턴: ^[A-Za-z-0-9+/=]+$

Errors

모든 작업에서 발생하는 일반적인 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하세요.

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

ListTagsForResource

서비스: Amazon QLDB

지정된 Amazon QLDB 리소스의 모든 태그를 반환합니다.

Request Syntax

```
GET /tags/resourceArn HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

resourceArn

태그를 열거할 Amazon 리소스 이름(ARN)입니다. 예:

`arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger`

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "Tags": {
      "string" : "string"
   }
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

Tags

지정된 Amazon QLDB 리소스와 현재 연계되어 있는 태그.

유형: 문자열 간 맵

맵 항목: 최소 항목 수는 0개입니다. 최대 항목 수 200개.

키 길이 제약 조건: 최소 길이는 1. 최대 길이 128.

값 길이 제약 조건: 최소 길이는 0입니다. 최대 길이는 256입니다.

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidParameterException

요청에서 하나 이상의 파라미터가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK

- AWS Python용 SDK
- AWS 루비 V3용 SDK

StreamJournalToKinesis

서비스: Amazon QLDB

주어진 Amazon QLDB 원장에 대한 저널 스트림을 생성합니다. 스트림은 원장의 저널에 체결된 모든 문서 개정본을 캡처하고 지정된 Amazon Kinesis Data Streams 리소스로 데이터를 전송합니다.

Request Syntax

```
POST /ledgers/name/journal-kinesis-streams HTTP/1.1
Content-type: application/json

{
   "ExclusiveEndTime": number,
   "InclusiveStartTime": number,
   "KinesisConfiguration": {
      "AggregationEnabled": boolean,
      "StreamArn": "string"
   },
   "RoleArn": "string",
   "StreamName": "string",
   "Tags": {
      "string" : "string"
   }
}
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

원장의 명칭입니다.

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

ExclusiveEndTime

스트림이 끝날 때를 지정하는 독점 날짜 및 시간입니다. 이 파라미터를 정의하지 않으면 취소하기 전까지 스트림이 무기한 실행됩니다.

ISO 8601은 ExclusiveEndTime 날짜 및 시간 형식이어야 하며 협정 세계시(UTC)여야 합니다. 예를 들면 2019-06-13T21:36:34Z입니다.

타입: Timestamp

필수 여부: 아니요

InclusiveStartTime

스트리밍 저널 데이터를 시작할 시작 날짜 및 시간(경계값 포함)입니다. 이 파라미터는 ISO 8601 날짜 및 시간 형식이어야 하며 협정 세계시(UTC)여야 합니다. 예를 들면 2019-06-13T21:36:34Z입니다.

InclusiveStartTime은 미래 시간일 수 없으며 ExclusiveEndTime 이전이어야 합니다.

원장의 CreationDateTime 이전인 InclusiveStartTime을 제공하는 경우 , QLDB는 실제로 기본값을 원장의 CreationDateTime으로 설정합니다.

타입: Timestamp

필수 여부: 예

KinesisConfiguration

스트림 요청에 대한 Kinesis Data Streams 대상의 구성 설정입니다.

타입: KinesisConfiguration 객체

필수 여부: 예

RoleArn

저널 스트림에 Kinesis Data Streams 리소스에 데이터 레코드를 쓸 수 있는 QLDB 권한을 부여하는 IAM 역할의 Amazon 리소스 이름(ARN)입니다.

저널 스트림을 요청할 때 QLDB에 역할을 전달하려면 IAM 역할 리소스에서 iam:PassRole 작업을 수행할 수 있는 권한이 있어야 합니다. 이는 모든 저널 스트림 요청에 필요합니다.

타입: 문자열

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

필수 여부: 예

StreamName

QLDB 저널 스트림에 할당할 명칭입니다. 사용자 정의 명칭은 스트림의 목적을 식별하고 나타내는 데 도움이 될 수 있습니다.

주어진 원장의 경우, 스트림 명칭은 다른 활성 스트림들 사이에서 고유해야 합니다. 스트림 명칭은 Amazon QLDB 개발자 가이드의 Amazon QLDB의 할당량에 정의된 원장 명칭과 동일한 명명 제약 조건을 갖습니다.

타입: 문자열

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 여부: 예

Tags

생성하려는 스트림에 태그로 추가하려는 키-값 쌍입니다. 태그 키는 대소문자를 구별합니다. 태그 값은 대소문자를 구분하며 null일 수 있습니다.

유형: 문자열 간 맵

맵 항목: 최소 항목 수는 0개입니다. 최대 항목 수 200개.

키 길이 제약 조건: 최소 길이는 1. 최대 길이 128.

값 길이 제약 조건: 최소 길이는 0입니다. 최대 길이는 256입니다.

필수 여부: 아니요

응답 구문

```
HTTP/1.1 200
Content-type: application/json

{
   "StreamId": "string"
```

```
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

StreamId

QLDB가 각 QLDB 저널 스트림에 할당하는 UUID(Base62로 인코딩된 텍스트로 표시됨)입니다.

타입: 문자열

길이 제약 조건: 고정 길이는 22입니다.

패턴: `^[A-Za-z-0-9]+$`

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidParameterException

요청에서 하나 이상의 파라미터가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

ResourcePreconditionNotMetException

조건이 미리 충족되지 않아 작업이 실패했습니다.

HTTP 상태 코드: 412

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

TagResource

서비스: Amazon QLDB

지정된 Amazon QLDB 리소스에 하나 이상의 태그를 추가합니다.

리소스는 최대 50개의 태그를 가질 수 있습니다. 리소스에 대해 50개가 넘는 태그를 만들려고 하면 요청이 실패하고 오류가 반환됩니다.

Request Syntax

```
POST /tags/resourceArn HTTP/1.1
Content-type: application/json

{
   "Tags": {
      "string" : "string"
   }
}
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

resourceArn

태그를 추가할 Amazon 리소스 이름(ARN)입니다. 예:

`arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger`

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

필수 여부: 예

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

Tags

지정된 QLDB 리소스에 태그로 추가할 키-값 쌍입니다. 태그 키는 대소문자를 구별합니다. 리소스에 이미 있는 키를 지정하면 요청이 실패하고 오류가 반환됩니다. 태그 값은 대소문자를 구분하며 null일 수 있습니다.

유형: 문자열 간 맵

맵 항목: 최소 항목 수는 0개입니다. 최대 항목 수 200개.

키 길이 제약 조건: 최소 길이는 1. 최대 길이 128.

값 길이 제약 조건: 최소 길이는 0입니다. 최대 길이는 256.

필수 여부: 예

응답 구문

```
HTTP/1.1 200
```

Response Elements

작업이 성공하면 서비스가 비어 있는 HTTP 본문과 함께 HTTP 200 응답을 반송합니다.

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidParameterException

요청에서 하나 이상의 파라미터가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++

- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

UntagResource

서비스: Amazon QLDB

지정된 Amazon QLDB 리소스에서 하나 이상의 태그를 제거합니다. 삭제할 태그 키를 최대 50개 지정할 수 있습니다.

Request Syntax

```
DELETE /tags/resourceArn?tagKeys=TagKeys HTTP/1.1
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

resourceArn

태그를 제거할 Amazon 리소스 이름(ARN)입니다. 예:

`arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger`

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

필수 여부: 예

TagKeys

제거할 태그 키의 목록입니다.

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

길이 제약: 최소 길이 1. 최대 길이는 128.

필수 여부: 예

Request Body

해당 요청에는 본문이 없습니다.

Response Syntax

```
HTTP/1.1 200
```

Response Elements

작업이 성공하면 서비스가 비어 있는 HTTP 본문과 함께 HTTP 200 응답을 반송합니다.

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidParameterException

요청에서 하나 이상의 파라미터가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

UpdateLedger

서비스: Amazon QLDB

원장의 속성을 업데이트합니다.

Request Syntax

```
PATCH /ledgers/name HTTP/1.1
Content-type: application/json

{
   "DeletionProtection": boolean,
   "KmsKey": "string"
}
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

원장의 명칭입니다.

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

DeletionProtection

사용자에 의해 삭제되지 않도록 원장을 보호할지 여부를 지정합니다. 원장 생성 중 정의하지 않을 경우 이 기능이 기본적으로 활성화됩니다(`true`).

삭제 방지가 활성화된 경우 원장을 삭제하려면 먼저 이 기능을 비활성화해야 합니다. `UpdateLedger` 작업을 호출하여 이 파라미터를 `false`로 설정하면 비활성화할 수 있습니다.

타입: 부울

필수 항목 여부: 아니요

KmsKey

원장에 저장된 데이터를 암호화하는 데 사용할 키 in AWS Key Management Service (AWS KMS) 입니다. 자세한 내용을 알아보려면 Amazon QLDB 개발자 안내서의 저장된 암호화를 참조하세요.

다음 옵션 중 하나를 사용하여 이 파라미터를 지정합니다.

- `AWS_OWNED_KMS_KEY`: 본인 대신 소유하고 AWS 관리하는 AWS KMS 키를 사용하십시오.
- 정의되지 않음: 원장의 KMS 키를 변경하지 않습니다.
- 유효한 대칭 고객 관리형 KMS 키: 사용자가 생성, 소유 및 관리하는 계정의 지정된 대칭 암호화 KMS 키를 사용합니다.

  Amazon QLDB에서는 비대칭 키가 지원되지 않습니다. 자세한 내용은 개발자 안내서의 대칭 및 비대칭 키 사용을 참조하십시오. AWS Key Management Service

고객 관리형 KMS 키를 지정하려면 해당 키 ID, Amazon 리소스 이름(ARN), 별칭 이름 또는 별칭 ARN을 사용할 수 있습니다. 별칭 이름을 사용할 때 `"alias/"`를 접두사를 사용합니다. 다른 AWS 계정키를 지정하려면 키 ARN 또는 별칭 ARN을 사용해야 합니다.

예:

- 키 ID: `1234abcd-12ab-34cd-56ef-1234567890ab`
- 키 ARN: `arn:aws:kms:us-east-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab`
- 별칭 이름: `alias/ExampleAlias`
- 별칭 ARN: `arn:aws:kms:us-east-2:111122223333:alias/ExampleAlias`

자세한 내용은 개발자 안내서의 키 식별자 (KeyId) 를 참조하십시오. AWS Key Management Service

타입: 문자열

길이 제약 조건: 최대 길이는 1,600입니다.

필수 여부: 아니요

응답 구문

```
HTTP/1.1 200
```

```
Content-type: application/json

{
   "Arn": "string",
   "CreationDateTime": number,
   "DeletionProtection": boolean,
   "EncryptionDescription": {
      "EncryptionStatus": "string",
      "InaccessibleKmsKeyDateTime": number,
      "KmsKeyArn": "string"
   },
   "Name": "string",
   "State": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

Arn

원장의 Amazon 리소스 이름(ARN)입니다.

타입: 문자열

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

CreationDateTime

원장이 생성된 날짜와 시간(epoch 시간 형식)입니다. (epoch 시간 형식은 UTC 기준으로 1970년 1월 1일 자정 12:00:00부터 경과된 초 단위의 시간입니다.)

유형: 타임스탬프

DeletionProtection

사용자에 의해 삭제되지 않도록 원장을 보호할지 여부를 지정합니다. 원장 생성 중 정의하지 않을 경우 이 기능이 기본적으로 활성화됩니다(`true`).

삭제 방지가 활성화된 경우 원장을 삭제하려면 먼저 이 기능을 비활성화해야 합니다. `UpdateLedger` 작업을 호출하여 이 파라미터를 `false`로 설정하면 비활성화할 수 있습니다.

타입: 부울

EncryptionDescription

원장의 저장 데이터 암호화에 대한 정보. 여기에는 현재 상태, AWS KMS 키, 키에 액세스할 수 없게 된 시점 (오류 발생 시) 이 포함됩니다.

유형: LedgerEncryptionDescription객체

Name

원장의 명칭입니다.

타입: 문자열

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

State

원장의 현재 상태입니다.

타입: 문자열

유효 값: `CREATING | ACTIVE | DELETING | DELETED`

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidParameterException

요청에서 하나 이상의 파라미터가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

UpdateLedgerPermissionsMode

서비스: Amazon QLDB

원장의 권한 모드를 업데이트합니다.

> **Important**
>
> STANDARD 권한 모드로 전환하기 전에 먼저 필요한 모든 IAM 정책과 표 태그를 생성하여 사용자에게 방해가 되지 않도록 해야 합니다. 자세히 알아보려면 Amazon QLDB 개발자 가이드의 표준 권한 모드로 마이그레이션을 참조하세요.

Request Syntax

```
PATCH /ledgers/name/permissions-mode HTTP/1.1
Content-type: application/json

{
   "PermissionsMode": "string"
}
```

URI 요청 파라미터

요청은 다음 URI 파라미터를 사용합니다.

name

원장의 명칭입니다.

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

요청 본문

요청은 JSON 형식으로 다음 데이터를 받습니다.

PermissionsMode

원장에 할당할 권한 모드입니다. 이 파라미터는 다음 값 중 하나를 가질 수 있습니다.

- ALLOW_ALL: 원장에 대한 API 수준 세분화로 액세스 통제를 가능하게 하는 레거시 권한 모드입니다.

  이 모드를 사용하면 이 원장에 대한 SendCommand API 권한이 있는 사용자가 지정된 원장의 모든 표에서 모든 PartiQL 명령(따라서 ALLOW_ALL)을 실행할 수 있습니다. 이 모드는 원장에 대해 생성하는 모든 표 수준 또는 명령 수준 IAM 권한 정책을 무시합니다.
- STANDARD: (권장) 원장, 테이블 및 PartiQL 명령에 대해 보다 세분화된 액세스 제어를 가능하게 하는 권한 모드입니다..

  기본적으로 이 모드는 이 원장의 모든 표에서 PartiQL 명령을 실행하려는 모든 사용자 요청을 거부합니다. PartiQL 명령 실행을 허용하려면 원장에 대한 SendCommand API 권한 외에도 특정 테이블 리소스 및 PartiQL 작업에 대한 IAM 권한 정책을 생성해야 합니다. 자세한 내용은 Amazon QLDB 개발자 안내서의 표준 권한 모드로 시작하기를 참조하세요.

> **Note**
>
> 원장 데이터의 보안을 극대화하려면 STANDARD 권한 모드를 사용하는 것이 좋습니다.

타입: 문자열

유효 값: ALLOW_ALL | STANDARD

필수 여부: 예

응답 구문

```
HTTP/1.1 200
Content-type: application/json

{
   "Arn": "string",
   "Name": "string",
   "PermissionsMode": "string"
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

Arn

원장의 Amazon 리소스 이름(ARN)입니다.

타입: 문자열

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

Name

원장의 명칭입니다.

타입: 문자열

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

PermissionsMode

원장의 현재 권한 모드입니다.

타입: 문자열

유효 값: `ALLOW_ALL | STANDARD`

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidParameterException

요청에서 하나 이상의 파라미터가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

지정된 리소스가 존재하지 않습니다.

HTTP 상태 코드: 404

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

Amazon QLDB Strese

Amazon QLDB 세션에서는 다음 작업을 지원합니다.

- SendCommand

SendCommand

서비스: Amazon QLDB Session

Amazon QLDB 원장에 명령을 보냅니다.

> **Note**
>
> 이 API와 직접 상호 작용하는 대신 QLDB 드라이버 또는 QLDB 쉘을 사용하여 원장에서 데이터 트랜잭션을 실행하는 것이 좋습니다.
>
> - AWS SDK로 작업하는 경우 QLDB 드라이버를 사용하십시오. 드라이버는 이 QLDB 세션 데이터 API 위에 높은 수준의 추상화 계층을 제공하고 사용자를 대신하여 SendCommand 작업을 관리합니다. 지원되는 프로그래밍 언어에 대한 자세한 내용과 목록은 Amazon QLDB 개발자 가이드의 드라이버 시작하기를 참조하세요.
> - AWS Command Line Interface (AWS CLI) 로 작업하는 경우 QLDB 셸을 사용하십시오. 쉘은 QLDB 드라이버를 사용하여 원장과 상호 작용하는 명령줄 인터페이스입니다. 자세한 설명은 Accessing Amazon QLDB using the QLDB shell 섹션을 참조하세요.

구문 요청

```
{
   "AbortTransaction": {
   },
   "CommitTransaction": {
      "CommitDigest": blob,
      "TransactionId": "string"
   },
   "EndSession": {
   },
   "ExecuteStatement": {
      "Parameters": [
         {
            "IonBinary": blob,
            "IonText": "string"
         }
      ],
      "Statement": "string",
      "TransactionId": "string"
   },
   "FetchPage": {
```

```
        "NextPageToken": "string",
        "TransactionId": "string"
    },
    "SessionToken": "string",
    "StartSession": {
        "LedgerName": "string"
    },
    "StartTransaction": {
    }
}
```

요청 파라미터

모든 작업에서 사용하는 파라미터에 대한 자세한 내용은 범용 파라미터를 참조하세요.

요청은 JSON 형식으로 다음 데이터를 받습니다.

AbortTransaction

현재 트랜잭션을 중지하는 명령입니다.

타입: AbortTransactionRequest 객체

필수 여부: 아니요

CommitTransaction

지정된 트랜잭션을 체결하는 명령입니다.

타입: CommitTransactionRequest 객체

필수 여부: 아니요

EndSession

현재 세션을 종료하는 명령입니다.

타입: EndSessionRequest 객체

필수 여부: 아니요

ExecuteStatement

지정된 트랜잭션에서 문을 실행하는 명령입니다.

타입: ExecuteStatementRequest 객체

필수 여부: 아니요

FetchPage

페이지를 가져오는 명령입니다.

타입: FetchPageRequest 객체

필수 여부: 아니요

SessionToken

현재 명령의 세션 토큰을 지정합니다. 세션 토큰은 세션 수명 내내 일정합니다.

세션 토큰을 가져오려면 `StartSession` 명령을 실행합니다. 이 `SessionToken`는 현재 세션 중에 실행되는 모든 후속 명령에 필요합니다.

타입: 문자열

길이 제약: 최소 길이는 4입니다. 최대 길이는 1024입니다.

패턴: `^[A-Za-z-0-9+/=]+$`

필수 여부: 아니요

StartSession

새 세션을 시작하는 명령입니다. 응답의 일부로 세션 토큰을 얻습니다.

타입: StartSessionRequest 객체

필수 여부: 아니요

StartTransaction

새 트랜잭션을 시작하는 명령입니다.

타입: StartTransactionRequest 객체

필수 항목 여부: 아니요

응답 구문

```
{
   "AbortTransaction": {
      "TimingInformation": {
         "ProcessingTimeMilliseconds": number
```

```
      }
   },
   "CommitTransaction": {
      "CommitDigest": blob,
      "ConsumedIOs": {
         "ReadIOs": number,
         "WriteIOs": number
      },
      "TimingInformation": {
         "ProcessingTimeMilliseconds": number
      },
      "TransactionId": "string"
   },
   "EndSession": {
      "TimingInformation": {
         "ProcessingTimeMilliseconds": number
      }
   },
   "ExecuteStatement": {
      "ConsumedIOs": {
         "ReadIOs": number,
         "WriteIOs": number
      },
      "FirstPage": {
         "NextPageToken": "string",
         "Values": [
            {
               "IonBinary": blob,
               "IonText": "string"
            }
         ]
      },
      "TimingInformation": {
         "ProcessingTimeMilliseconds": number
      }
   },
   "FetchPage": {
      "ConsumedIOs": {
         "ReadIOs": number,
         "WriteIOs": number
      },
      "Page": {
         "NextPageToken": "string",
         "Values": [
```

```
            {
               "IonBinary": blob,
               "IonText": "string"
            }
         ]
      },
      "TimingInformation": {
         "ProcessingTimeMilliseconds": number
      }
   },
   "StartSession": {
      "SessionToken": "string",
      "TimingInformation": {
         "ProcessingTimeMilliseconds": number
      }
   },
   "StartTransaction": {
      "TimingInformation": {
         "ProcessingTimeMilliseconds": number
      },
      "TransactionId": "string"
   }
}
```

응답 요소

작업이 성공하면 서비스가 HTTP 200 응답을 반송합니다.

다음 데이터는 서비스에 의해 JSON 형식으로 반환됩니다.

AbortTransaction

중단된 트랜잭션의 세부 정보를 포함합니다.

타입: AbortTransactionResult 객체

CommitTransaction

커밋된 트랜잭션의 세부 정보를 포함합니다.

타입: CommitTransactionResult 객체

EndSession

종료된 세션의 세부 정보를 포함합니다.

타입: EndSessionResult 객체

ExecuteStatement

실행된 명령문의 세부 정보를 포함합니다.

타입: ExecuteStatementResult 객체

FetchPage

가져온 페이지의 세부 정보를 포함합니다.

타입: FetchPageResult 객체

StartSession

세션 토큰이 포함된 시작된 세션의 세부 정보를 포함합니다. 이 `SessionToken`는 현재 세션 중에 실행되는 모든 후속 명령에 필요합니다.

타입: StartSessionResult 객체

StartTransaction

시작된 트랜잭션의 세부 정보를 포함합니다.

타입: StartTransactionResult 객체

Errors

모든 작업에서 발생하는 흔한 오류에 대한 자세한 내용은 일반적인 오류 섹션을 참조하십시오.

BadRequestException

요청이 잘못되었거나 잘못된 파라미터 값 또는 필수 파라미터 누락 등의 오류가 있는 경우 반환됩니다.

HTTP 상태 코드: 400

CapacityExceededException

요청이 원장의 처리 용량을 초과할 때 반환됩니다.

HTTP 상태 코드: 400

InvalidSessionException

제한 시간이 초과되었거나 만료되어 세션이 더 이상 존재하지 않는 경우 반환됩니다.

HTTP 상태 코드: 400

LimitExceededException

활성 세션 수와 같은 리소스 제한을 초과할 경우 반환됩니다.

HTTP 상태 코드: 400

OccConflictException

OCC(낙관적 동시성 제어)의 검증 단계에서 실패로 인해 트랜잭션을 저널에 기록할 수 없을 때 반환됩니다.

HTTP 상태 코드: 400

RateExceededException

요청율이 허용된 처리량을 초과할 때 반환됩니다.

HTTP 상태 코드: 400

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS Go v2를 위한 SDK
- AWS Java V2용 SDK
- AWS V3용 SDK JavaScript
- AWS PHP V3용 SDK
- AWS Python용 SDK
- AWS 루비 V3용 SDK

데이터 유형

Amazon QLDB는 다음과 같은 데이터 유형을 지원합니다.

- JournalKinesisStreamDescription

- JournalS3ExportDescription
- KinesisConfiguration
- LedgerEncryptionDescription
- LedgerSummary
- S3EncryptionConfiguration
- S3ExportConfiguration
- ValueHolder

Amazon QLDB 세션은 다음과 같은 데이터 유형을 지원합니다.

- AbortTransactionRequest
- AbortTransactionResult
- CommitTransactionRequest
- CommitTransactionResult
- EndSessionRequest
- EndSessionResult
- ExecuteStatementRequest
- ExecuteStatementResult
- FetchPageRequest
- FetchPageResult
- IOUsage
- Page
- StartSessionRequest
- StartSessionResult
- StartTransactionRequest
- StartTransactionResult
- TimingInformation
- ValueHolder

Amazon QLDB

Amazon QLDB는 다음과 같은 데이터 유형을 지원합니다.

- JournalKinesisStreamDescription
- JournalS3ExportDescription
- KinesisConfiguration
- LedgerEncryptionDescription
- LedgerSummary
- S3EncryptionConfiguration
- S3ExportConfiguration
- ValueHolder

JournalKinesisStreamDescription

서비스: Amazon QLDB

Amazon 리소스 이름(ARN), 스트림 이름, 생성 시간, 현재 상태, 원본 스트림 생성 요청의 파라미터 등 Amazon QLDB 저널 스트림에 대한 정보입니다.

내용

KinesisConfiguration

QLDB 저널 스트림에 대한 Amazon Kinesis Data Streams 대상의 구성 설정입니다.

유형: KinesisConfiguration객체

필수 여부: 예

LedgerName

원장의 명칭입니다.

타입: 문자열

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

RoleArn

저널 스트림에 Kinesis Data Streams 리소스에 데이터 레코드를 쓸 수 있는 QLDB 권한을 부여하는 IAM 역할의 Amazon 리소스 이름(ARN)입니다.

타입: 문자열

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

필수 여부: 예

Status

QLDB 저널 스트림의 현재 상태입니다.

타입: 문자열

유효 값: `ACTIVE | COMPLETED | CANCELED | FAILED | IMPAIRED`

필수 사항 여부: 예

StreamId

QLDB 저널 스트림의 UUID(Base62 인코딩된 텍스트로 표시됨)입니다.

타입: 문자열

길이 제약 조건: 고정 길이는 22입니다.

패턴: `^[A-Za-z-0-9]+$`

필수 사항 여부: Yes

StreamName

QLDB 저널 스트림의 사용자 정의 이름입니다.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

Arn

QLDB 저널 스트림의 Amazon 리소스 이름(ARN)입니다.

타입: 문자열

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

필수 여부: 아니요

CreationTime

QLDB 저널 스트림이 생성된 날짜 및 시간(epoch 시간 형식)입니다. (epoch 시간 형식은 UTC 기준으로 1970년 1월 1일 자정 12:00:00부터 경과된 초 단위의 시간입니다.)

유형: 타임스탬프

필수 여부: 아니요

ErrorCause

스트림의 상태가 `IMPAIRED` 또는 `FAILED` 상태인 이유를 설명하는 오류 메시지입니다. 다른 상태 값이 있는 스트림에는 해당되지 않습니다.

타입: 문자열

유효 값: `KINESIS_STREAM_NOT_FOUND | IAM_PERMISSION_REVOKED`

필수 여부: 아니요

ExclusiveEndTime

스트림이 끝날 때를 지정하는 독점 날짜 및 시간입니다. 이 파라미터를 정의하지 않으면 취소하기 전까지 스트림이 무기한 실행됩니다.

유형: 타임스탬프

필수 여부: 아니요

InclusiveStartTime

스트리밍 저널 데이터를 시작할 시작 날짜 및 시간(경계값 포함)입니다.

유형: 타임스탬프

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

JournalS3ExportDescription

서비스: Amazon QLDB

원장 이름, 내보내기 ID, 생성 시간, 현재 상태 및 원래 내보내기 생성 요청의 파라미터를 포함하는 저널 내보내기 작업에 대한 정보입니다.

내용

ExclusiveEndTime

원본 내보내기 요청에 지정된 저널 콘텐츠 범위에 대한 독점 종료 날짜 및 시간입니다.

유형: 타임스탬프

필수 여부: 예

ExportCreationTime

내보내기 작업이 생성된 날짜 및 시간(epoch 시간 형식)입니다. (epoch 시간 형식은 UTC 기준으로 1970년 1월 1일 자정 12:00:00부터 경과된 초 단위의 시간입니다.)

유형: 타임스탬프

필수 여부: 예

ExportId

저널 내보내기 작업의 UUID(Base62 인코딩된 텍스트로 표시됨)입니다.

타입: 문자열

길이 제약 조건: 고정 길이는 22입니다.

패턴: `^[A-Za-z-0-9]+$`

필수 사항 여부: Yes

InclusiveStartTime

원본 내보내기 요청에 지정된 저널 콘텐츠 범위에 대한 시작 날짜 및 시간을 포함한 시간입니다.

유형: 타임스탬프

필수 여부: 예

LedgerName

원장의 명칭입니다.

타입: 문자열

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 사항 여부: Yes

RoleArn

다음을 수행하기 위해 저널 내보내기 작업에 대한 QLDB 권한을 부여하는 IAM 역할의 Amazon 리소스 이름(ARN)입니다.

- Amazon Simple Storage Service(S3) 버킷에 개체를 기록합니다.
- (선택 사항) 내보낸 데이터의 서버 측 암호화에 고객 관리 키 in AWS Key Management Service (AWS KMS) 를 사용하십시오.

타입: 문자열

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

필수 여부: 예

S3ExportConfiguration

저널 내보내기 작업이 저널 콘텐츠를 작성하는 Amazon Simple Storage Service(S3) 버킷 위치입니다.

유형: S3ExportConfiguration객체

필수 여부: 예

Status

저널 내보내기 작업의 현재 상태입니다.

타입: 문자열

유효 값: `IN_PROGRESS | COMPLETED | CANCELLED`

필수 사항 여부: 예

OutputFormat

내보낸 저널 데이터의 출력 형식입니다.

타입: 문자열

유효 값: `ION_BINARY | ION_TEXT | JSON`

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

KinesisConfiguration

서비스: Amazon QLDB

Amazon QLDB 저널 스트림에 대한 Amazon Kinesis Data Streams 대상의 구성 설정입니다.

내용

StreamArn

Kinesis Data Streams의 Amazon 리소스 이름(ARN)입니다.

타입: 문자열

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

필수 여부: 예

AggregationEnabled

QLDB에서 단일 Kinesis Data Streams 레코드에 여러 데이터 레코드를 게시하여 API 호출당 전송되는 레코드 수를 늘릴 수 있도록 합니다.

기본값: `True`

> **Important**
>
> 레코드 집계는 레코드 처리에 중요한 영향을 미치며 스트림 소비자의 집계를 해제해야 합니다. 자세한 내용을 알아보려면 Amazon Kinesis Data Streams 개발자 안내서의 KPL 주요 개념과 소비자 분해를 참조하세요.

타입: 부울

필수 항목 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK

- AWS 루비 V3용 SDK

LedgerEncryptionDescription

서비스: Amazon QLDB

Amazon QLDB 원장에 있는 저장 데이터의 암호화에 대한 정보입니다. 여기에는 현재 상태, 키 입력 AWS Key Management Service (AWS KMS), 키에 액세스할 수 없게 된 시점 (오류 발생 시) 이 포함됩니다.

자세한 내용을 알아보려면 Amazon QLDB 개발자 안내서의 저장된 암호화를 참조하세요.

내용

EncryptionStatus

원장에 저장된 암호화의 현재 상태입니다. 다음 값 중 하나일 수 있습니다:

- ENABLED: 지정된 키를 사용하여 암호화가 완전히 활성화되었습니다.
- UPDATING: 원장이 지정된 키 변경을 능동적으로 처리하고 있습니다.

  QLDB의 주요 변경 사항은 비동기적으로 이루어집니다. 키 변경이 처리되는 동안 성능에 영향을 주지 않고 원장에 완전히 액세스할 수 있습니다. 키를 업데이트하는 데 걸리는 시간은 원장 크기에 따라 다릅니다

- KMS_KEY_INACCESSIBLE: 지정된 고객 관리형 KMS 키에 액세스할 수 없으며 원장이 손상되었습니다. 키가 비활성화 또는 삭제되었거나 키에 대한 권한 부여가 취소되었습니다. 원장이 손상되면 해당 원장에 액세스할 수 없으며 읽기 또는 쓰기 요청을 수락하지 않습니다.

  키에 부여된 권한을 복원하거나 비활성화된 키를 다시 활성화하면 손상된 원장은 자동으로 활성 상태로 돌아갑니다. 하지만 고객 관리형 KMS 키 삭제 작업은 되돌릴 수 없습니다. 키가 삭제된 후에는 해당 키로 보호되는 원장에 더 이상 액세스할 수 없으며 데이터를 영구적으로 복구할 수 없게 됩니다.

타입: 문자열

유효 값: ENABLED | UPDATING | KMS_KEY_INACCESSIBLE

필수 사항 여부: 예

KmsKeyArn

원장이 저장 중 암호화에 사용할 고객 관리형 KMS 키의 Amazon 리소스 이름(ARN)입니다. 이 매개변수가 정의되지 않은 경우 원장은 AWS 소유한 KMS 키를 암호화에 사용합니다. 원장의 암호화 구성을 소유한 KMS 키로 업데이트할 AWS_OWNED_KMS_KEY 때 표시됩니다. AWS

타입: 문자열

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

필수 여부: 예

InaccessibleKmsKeyDateTime

오류가 발생한 경우 AWS KMS 키에 처음 액세스할 수 없게 된 날짜 및 시간 (에포크 타임 형식) (epoch 시간 형식은 UTC 기준으로 1970년 1월 1일 자정 12:00:00부터 경과된 초 단위의 시간입니다.)

AWS KMS 키에 액세스할 수 있는 경우 이 매개변수는 정의되지 않습니다.

유형: 타임스탬프

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

LedgerSummary

서비스: Amazon QLDB

이름, 상태 및 생성 시기를 포함한 원장에 대한 정보입니다.

내용

CreationDateTime

원장이 생성된 날짜와 시간(epoch 시간 형식)입니다. (epoch 시간 형식은 UTC 기준으로 1970년 1월 1일 자정 12:00:00부터 경과된 초 단위의 시간입니다.)

유형: 타임스탬프

필수 여부: 아니요

Name

원장의 명칭입니다.

타입: 문자열

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: No

State

원장의 현재 상태입니다.

타입: 문자열

유효 값: `CREATING | ACTIVE | DELETING | DELETED`

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK

- AWS 루비 V3용 SDK

S3EncryptionConfiguration

서비스: Amazon QLDB

Amazon Simple Storage Service(S3) 버킷에 데이터를 기록하기 위해 저널 내보내기 작업에서 사용하는 암호화 설정입니다.

내용

ObjectEncryptionType

Amazon S3 객체 암호화 타입입니다.

Amazon S3의 서버 측 암호화 옵션에 대한 자세한 내용은 Amazon S3 개발자 안내서의 서버 측 암호화를 사용한 데이터 보호를 참조하세요.

타입: 문자열

유효 값: `SSE_KMS | SSE_S3 | NO_ENCRYPTION`

필수 사항 여부: 예

KmsKeyArn

() 에 있는 대칭 암호화 키의 Amazon 리소스 이름 (ARN). AWS Key Management Service AWS KMS Amazon S3는 비대칭 KMS 키를 지원하지 않습니다.

`SSE_KMS`를 `ObjectEncryptionType`로 지정하는 경우 `KmsKeyArn`를 제공해야 합니다.

`SSE_S3`를 `ObjectEncryptionType`으로 지정하는 경우 `KmsKeyArn`는 요구되지 않습니다.

타입: 문자열

길이 제약 조건: 최소 길이는 20입니다. 최대 길이는 1,600입니다.

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK

- AWS 루비 V3용 SDK

S3ExportConfiguration

서비스: Amazon QLDB

저널 내보내기 작업이 저널 콘텐츠를 작성하는 Amazon Simple Storage Service(S3) 버킷 위치입니다.

내용

Bucket

저널 내보내기 작업이 저널 콘텐츠를 작성하는 Amazon S3 버킷 위치입니다.

버킷 이름은 Amazon S3 버킷 명명 규칙을 준수해야 합니다. 자세한 내용은 Amazon S3 개발자 안내서의 버킷 규제 및 제한을 참조하십시오.

타입: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 255.

패턴: `^[A-Za-z-0-9-_.]+$`

필수 사항 여부: Yes

EncryptionConfiguration

저널 내보내기 작업에서 Amazon S3 버킷에 데이터를 쓰기 위해 사용하는 암호화 설정입니다.

유형: S3EncryptionConfiguration객체

필수 여부: 예

Prefix

저널 내보내기 작업이 저널 콘텐츠를 기록하는 Amazon S3 버킷의 접두사입니다.

접두사는 Amazon S3 키 명명 규칙 및 제한 사항을 준수해야 합니다. 자세한 내용은 Amazon S3 개발자 안내서의 객체 키 및 메타데이터를 참조하십시오.

다음은 유효한 `Prefix` 값의 예입니다.

- `JournalExports-ForMyLedger/Testing/`
- `JournalExports`
- `My:Tests/`

타입: 문자열

길이 제약 조건: 최소 길이는 0입니다. 최대 길이는 128입니다.

필수 여부: 예

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

ValueHolder

서비스: Amazon QLDB

여러 인코딩 형식의 값을 포함할 수 있는 구조입니다.

내용

IonText

`ValueHolder` 구조에 포함된 Amazon Ion 일반 텍스트 값입니다.

타입: 문자열

길이 제약: 최소 길이 1. 최대 길이는 1048576입니다.

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

Amazon QLDB Strese

Amazon QLDB 세션은 다음과 같은 데이터 유형을 지원합니다.

- AbortTransactionRequest
- AbortTransactionResult
- CommitTransactionRequest
- CommitTransactionResult
- EndSessionRequest
- EndSessionResult
- ExecuteStatementRequest
- ExecuteStatementResult

- FetchPageRequest
- FetchPageResult
- IOUsage
- Page
- StartSessionRequest
- StartSessionResult
- StartTransactionRequest
- StartTransactionResult
- TimingInformation
- ValueHolder

AbortTransactionRequest

서비스: Amazon QLDB Session

중단할 트랜잭션의 세부 정보를 포함합니다.

내용

이 예외 구조의 구성원은 컨텍스트에 따라 다릅니다.

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

AbortTransactionResult

서비스: Amazon QLDB Session

중단된 트랜잭션의 세부 정보를 포함합니다.

내용

TimingInformation

명령에 대한 서버 측 성능 정보가 포함되어 있습니다.

타입: TimingInformation 객체

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

CommitTransactionRequest

서비스: Amazon QLDB Session

커밋할 트랜잭션의 세부 정보가 포함되어 있습니다.

내용

CommitDigest

커밋할 트랜잭션의 커밋 다이제스트를 지정합니다. 모든 활성 트랜잭션에 대해 커밋 다이제스트를 전달해야 합니다. QLDB는 `CommitDigest`을 검증하고 클라이언트에서 계산된 다이제스트가 QLDB에서 계산한 다이제스트와 일치하지 않으면 오류와 함께 커밋을 거부합니다.

이 `CommitDigest` 파라미터의 목적은 서버가 클라이언트가 보낸 명령문 세트를 클라이언트가 전송한 것과 동일한 순서로 중복 없이 처리한 경우에만 QLDB가 트랜잭션을 커밋하도록 하는 것입니다.

타입: Base64로 인코딩된 이진 데이터 객체

필수 여부: 예

TransactionId

커밋할 트랜잭션의 트랜잭션 ID를 지정하십시오.

타입: 문자열

길이 제약 조건: 고정 길이는 22입니다.

패턴: `^[A-Za-z-0-9]+$`

필수 여부: 예

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

CommitTransactionResult

서비스: Amazon QLDB Session

커밋된 트랜잭션의 세부 정보를 포함합니다.

내용

CommitDigest

커밋된 트랜잭션의 커밋 다이제스트.

유형: Base64로 인코딩된 이진 데이터 객체

필수 여부: 아니요

ConsumedIOs

사용된 I/O 요청 수에 대한 지표가 포함되어 있습니다.

유형: IOUsage객체

필수 항목 여부: 아니요

TimingInformation

명령에 대한 서버 측 성능 정보가 포함되어 있습니다.

타입: TimingInformation 객체

필수 항목 여부: 아니요

TransactionId

커밋된 트랜잭션의 트랜잭션 ID.

타입: 문자열

길이 제약 조건: 고정 길이는 22입니다.

패턴: `^[A-Za-z-0-9]+$`

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

EndSessionRequest

서비스: Amazon QLDB Session

세션 종료 요청을 지정합니다.

내용

이 예외 구조의 구성원은 컨텍스트에 따라 다릅니다.

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

EndSessionResult

서비스: Amazon QLDB Session

종료된 세션의 세부 정보를 포함합니다.

내용

TimingInformation

명령에 대한 서버 측 성능 정보가 포함되어 있습니다.

타입: TimingInformation 객체

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

ExecuteStatementRequest

서비스: Amazon QLDB Session

명령문 실행 요청을 지정합니다.

내용

Statement

요청의 명령문을 지정합니다.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 100000입니다.

필수 여부: 예

TransactionId

요청의 트랜잭션 ID를 지정합니다.

타입: 문자열

길이 제약 조건: 고정 길이는 22입니다.

패턴: `^[A-Za-z-0-9]+$`

필수 사항 여부: Yes

Parameters

요청에 파라미터화된 명령문의 파라미터를 지정합니다.

타입: ValueHolder객체 배열

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK

- AWS 루비 V3용 SDK

ExecuteStatementResult

서비스: Amazon QLDB Session

실행된 명령문의 세부 정보를 포함합니다.

내용

ConsumedIOs

사용된 I/O 요청 수에 대한 지표가 포함되어 있습니다.

유형: IOUsage객체

필수 항목 여부: 아니요

FirstPage

가져온 첫 번째 페이지의 세부 정보가 들어 있습니다.

유형: Page객체

필수 항목 여부: 아니요

TimingInformation

명령에 대한 서버 측 성능 정보가 포함되어 있습니다.

타입: TimingInformation 객체

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

FetchPageRequest

서비스: Amazon QLDB Session

가져올 페이지의 세부 정보를 지정합니다.

내용

NextPageToken

가져올 페이지의 다음 페이지 토큰을 지정합니다.

타입: 문자열

길이 제약: 최소 길이는 4입니다. 최대 길이는 1024입니다.

패턴: `^[A-Za-z-0-9+/=]+$`

필수 사항 여부: Yes

TransactionId

가져올 페이지의 트랜잭션 ID를 지정합니다.

타입: 문자열

길이 제약 조건: 고정 길이는 22입니다.

패턴: `^[A-Za-z-0-9]+$`

필수 여부: 예

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

FetchPageResult

서비스: Amazon QLDB Session

가져온 페이지를 포함합니다.

내용

ConsumedIOs

사용된 I/O 요청 수에 대한 지표가 포함되어 있습니다.

유형: IOUsage객체

필수 항목 여부: 아니요

Page

가져온 페이지의 세부 정보가 포함되어 있습니다.

유형: Page객체

필수 항목 여부: 아니요

TimingInformation

명령에 대한 서버 측 성능 정보가 포함되어 있습니다.

타입: TimingInformation 객체

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

IOUsage

서비스: Amazon QLDB Session

호출된 명령에 대한 I/O 사용량 지표가 포함되어 있습니다.

내용

ReadIOs

명령에서 수행된 읽기 I/O 요청의 수입니다.

유형: Long

필수 여부: 아니요

WriteIOs

명령에서 수행된 쓰기 I/O 요청의 수입니다.

유형: Long

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

Page

서비스: Amazon QLDB Session

가져온 페이지의 세부 정보가 포함되어 있습니다.

내용

NextPageToken

다음 페이지의 토큰입니다.

타입: 문자열

길이 제약: 최소 길이는 4입니다. 최대 길이는 1024입니다.

패턴: `^[A-Za-z-0-9+/=]+$`

필수 여부: 아니요

Values

여러 인코딩 형식의 값을 포함하는 구조입니다.

타입: ValueHolder 객체 배열

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

StartSessionRequest

서비스: Amazon QLDB Session

새 세션 시작 요청을 지정합니다.

내용

LedgerName

새 세션을 시작할 원장의 명칭입니다.

타입: 문자열

길이 제약: 최소 길이 1. 최대 길이 32.

패턴: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

필수 여부: 예

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

StartSessionResult

서비스: Amazon QLDB Session

시작된 세션의 세부 정보를 포함합니다.

내용

SessionToken

시작된 세션의 세션 토큰입니다. 이 SessionToken는 현재 세션 중에 실행되는 모든 후속 명령에 필요합니다.

타입: 문자열

길이 제약: 최소 길이는 4입니다. 최대 길이는 1024입니다.

패턴: ^[A-Za-z-0-9+/=]+$

필수 여부: 아니요

TimingInformation

명령에 대한 서버 측 성능 정보가 포함되어 있습니다.

타입: TimingInformation 객체

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

StartTransactionRequest

서비스: Amazon QLDB Session

트랜잭션 시작 요청을 지정합니다.

내용

이 예외 구조의 구성원은 컨텍스트에 따라 다릅니다.

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

StartTransactionResult

서비스: Amazon QLDB Session

시작된 트랜잭션의 세부 정보를 포함합니다.

내용

TimingInformation

명령에 대한 서버 측 성능 정보가 포함되어 있습니다.

타입: TimingInformation 객체

필수 항목 여부: 아니요

TransactionId

시작된 트랜잭션의 트랜잭션 ID.

타입: 문자열

길이 제약 조건: 고정 길이는 22입니다.

패턴: `^[A-Za-z-0-9]+$`

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

TimingInformation

서비스: Amazon QLDB Session

명령에 대한 서버측 수행 정보가 포함되어 있습니다. Amazon QLDB는 요청을 받는 시간과 해당 응답을 보내는 시간 사이의 타이밍 정보를 캡처합니다.

내용

ProcessingTimeMilliseconds

QLDB가 명령을 처리하는 데 소비한 시간(밀리초).

타입: Long

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

ValueHolder

서비스: Amazon QLDB Session

여러 인코딩 형식의 값을 포함할 수 있는 구조입니다.

내용

IonBinary

`ValueHolder` 구조에 포함된 Amazon Ion 이진 값입니다.

타입: Base64로 인코딩된 이진 데이터 객체

길이 제약: 최소 길이 1. 최대 길이는 131072입니다.

필수 여부: 아니요

IonText

`ValueHolder` 구조에 포함된 Amazon Ion 일반 텍스트 값입니다.

타입: 문자열

길이 제약: 최소 길이 1. 최대 길이는 1048576입니다.

필수 여부: 아니요

참고

언어별 AWS SDK 중 하나에서 이 API를 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.

- AWS SDK for C++
- AWS Java V2용 SDK
- AWS 루비 V3용 SDK

일반적인 오류

이 단원에는 모든 AWS 서비스의 API 작업에 대한 일반 오류가 나와 있습니다. 이 서비스의 API 작업에 대한 오류는 해당 API 작업 항목을 참조하십시오.

AccessDeniedException

이 작업을 수행할 수 있는 충분한 액세스 권한이 없습니다.

HTTP 상태 코드: 400

IncompleteSignature

요청 서명이 AWS 표준을 준수하지 않습니다.

HTTP 상태 코드: 400

InternalFailure

알 수 없는 오류, 예외 또는 장애 때문에 요청 처리가 실패했습니다.

HTTP 상태 코드: 500

InvalidAction

요청된 동작 또는 작업이 유효하지 않습니다. 작업을 올바로 입력했는지 확인합니다.

HTTP 상태 코드: 400

InvalidClientTokenId

제공된 X.509 인증서 또는 AWS 액세스 키 ID가 AWS의 레코드에 존재하지 않습니다.

HTTP 상태 코드: 403

NotAuthorized

이 작업을 수행하려면 권한이 있어야 합니다.

HTTP 상태 코드: 400

OptInRequired

AWS 액세스 키 ID는 서비스에 대한 구독이 필요합니다.

HTTP 상태 코드: 403

RequestExpired

요청이 요청상의 날짜 스탬프로부터 15분 이상, 또는 요청 만료 날짜(예: 미리 서명된 URL)로부터 15분 이상 경과한 후 서비스에 도달했거나, 요청상의 날짜 스탬프가 15분 이상 미래입니다.

HTTP 상태 코드: 400

ServiceUnavailable

서버의 일시적 장애로 인해 요청이 실패하였습니다.

HTTP 상태 코드: 503

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationError

입력이 AWS 서비스에서 지정한 제약에 충족되지 않습니다.

HTTP 상태 코드: 400

공통 파라미터

다음 목록에는 모든 작업이 쿼리 문자열을 사용하여 Signature Version 4 요청에 서명하는 데 사용하는 파라미터가 포함되어 있습니다. 작업별 파라미터는 그 작업에 대한 항목에 나열되어 있습니다. Signature Version 4에 대한 자세한 내용은 IAM 사용 설명서의 AWS API 요청에 서명을 참조하세요.

Action

수행할 작업입니다.

유형: 문자열

필수 항목 여부: 예

Version

요청이 작성되는 API 버전으로 YYYY-MM-DD 형식으로 표시됩니다.

유형: 문자열

필수 항목 여부: 예

X-Amz-Algorithm

요청 서명을 생성하는 데 사용된 해시 알고리즘입니다.

조건: HTTP 권한 부여 헤더 대신 쿼리 문자열에 인증 정보를 포함하는 경우 이 파라미터를 지정합니다.

유형: 문자열

유효한 값: `AWS4-HMAC-SHA256`

필수 항목 여부: 조건부

X-Amz-Credential

자격 증명 범위 값이며 액세스 키, 날짜, 대상으로 하는 리전, 요청하는 서비스 및 종료 문자열("aws4_request")이 포함된 문자열입니다. 값은 다음 형식으로 표시됩니다. access_key/YYYYMMDD/region/service/aws4_request.

자세한 내용은 IAM 사용 설명서의 서명된 AWS API 요청 생성을 참조하세요.

조건: HTTP 권한 부여 헤더 대신 쿼리 문자열에 인증 정보를 포함하는 경우 이 파라미터를 지정합니다.

유형: 문자열

필수 항목 여부: 조건부

X-Amz-Date

서명을 만드는 데 사용되는 날짜입니다. 형식은 ISO 8601 기본 형식(YYYYMMDD'T'HHMMSS'Z') 이어야 합니다. 예를 들어 다음 날짜 시간은 유효한 X-Amz-Date 값: `20120325T120000Z`.

조건: X-Amz-Date는 모든 요청에서 옵션이지만 서명 요청에 사용되는 날짜보다 우선할 때 사용됩니다. 날짜 헤더가 ISO 8601 기본 형식으로 지정된 경우 X-Amz-Date가 필요하지 않습니다. X-Amz-Date를 사용하는 경우 항상 Date 헤더의 값을 재정의합니다. 자세한 내용은 IAM 사용 설명서의 AWS API 요청 서명의 요소를 참조하세요.

유형: 문자열

필수 항목 여부: 조건부

X-Amz-Security-Token

AWS Security Token Service(AWS STS)에 대한 호출을 통해 받은 임시 보안 토큰입니다. AWS STS의 임시 보안 인증 정보를 지원하는 서비스 목록은 IAM 사용 설명서의 IAM으로 작업하는 AWS 서비스를 참조하세요.

조건: AWS STS의 임시 보안 인증 정보를 사용하는 경우 보안 토큰을 포함시켜야 합니다.

유형: 문자열

필수 항목 여부: 조건부

X-Amz-Signature

서명할 문자열과 파생된 서명 키에서 계산된 16진수로 인코딩된 서명을 지정합니다.

조건: HTTP 권한 부여 헤더 대신 쿼리 문자열에 인증 정보를 포함하는 경우 이 파라미터를 지정합니다.

유형: 문자열

필수 항목 여부: 조건부

X-Amz-SignedHeaders

표준 요청의 일부로 포함된 모든 HTTP 헤더를 지정합니다. 서명된 헤더 지정에 대한 자세한 내용은 IAM 사용 설명서의 서명된 AWS API 요청 생성을 참조하세요.

조건: HTTP 권한 부여 헤더 대신 쿼리 문자열에 인증 정보를 포함하는 경우 이 파라미터를 지정합니다.

유형: 문자열

필수 항목 여부: 조건부

Amazon QLDB 할당량 및 제한

이 섹션에서는 Amazon QLDB 내의 현재 할당량(제한이라고도 함)에 대해 설명합니다.

주제

기본 할당량

QLDB에는 AWS 일반 참조의 Amazon QLDB 엔드포인트 및 할당량에도 나열된 대로 다음과 같은 기본 할당량이 있습니다. 이러한 할당량은 리전당 AWS 계정당 적용됩니다. 리전의 사용자 계정에 대한 할당량 증가를 요청하려면 Service Quotas 콘솔을 사용합니다.

AWS Management Console에 로그인하고 https://console.aws.amazon.com/servicequotas/에서 Service Quotas 콘솔을 엽니다.

리소스	기본 할당량
현재 리전의 이 계정에서 생성할 수 있는 활성 원장의 최대 수	5
원장당 Amazon S3로 내보내는 활성 저널의 최대 수	2
원장당 Kinesis Data Streams에 대한 최대 활성 저널 스트림 수	5

고정 할당량

기본 할당량 외에도 QLDB에는 원장당 다음과 같은 고정 할당량이 있습니다. 이러한 할당량은 Service Quotas를 사용하여 늘릴 수 없습니다.

리소스	고정 할당량
동시 활성 세션 수	1500
활성 테이블 수	20
총 테이블 수(활성 및 비활성) **Note** QLDB에서 삭제된 테이블은 비활성 상태로 간주되며 이 총 할당량에 포함됩니다.	40
테이블당 인덱스 수	5
트랜잭션에 포함된 문서 수	40
트랜잭션에서 수정해야 할 개정 수	1
문서 크기(`IonBinary` 형식으로 인코딩)	128KB
명령문 파라미터 크기(`IonBinary` 형식)	128KB
명령문 파라미터 크기(`IonText` 형식)	1MB
문의 문자열 길이	100,000자
트랜잭션 크기	4MB
트랜잭션 제한 시간	30초
완료된 저널 내보내기 작업의 만료 기간	7일
터미널 저널 스트림의 만료 기간	7일

원장 할당량

리전 내 사용자 계정의 원장 할당량 증가를 요청하려면 Service Quotas 콘솔을 사용하면 됩니다.

https://console.aws.amazon.com/servicequotas/에서 Service Quotas 콘솔을 엽니다.

일부 QLDB 사용 사례에서는 비즈니스 성장에 따라 리전당 AWS 계정당 원장 수를 늘려야 합니다. 예를 들어, 고객 또는 데이터를 격리하기 위해 전용 원장을 만들어야 할 수 있습니다. 이 경우 다중 계정 아키텍처를 활용하여 QLDB 할당량을 처리하는 것을 고려해 보십시오. 추가 정보는 AWS SaaS 테넌트 격리 전략 백서의 계정 사일로 격리를 참조하십시오.

문서 크기

`IonBinary` 형식으로 인코딩된 문서의 최대 크기는 128KB입니다. 텍스트에서 바이너리로의 변환은 각 문서의 구조에 따라 크게 달라지기 때문에 `IonText` 형식의 문서 크기를 정확히 제한할 수는 없습니다. QLDB는 콘텐츠가 열려 있는 문서를 지원하므로 각각의 고유한 문서 구조에 따라 크기 계산이 달라집니다.

트랜잭션 크기

QLDB 트랜잭션의 최대 크기는 4MB입니다. 트랜잭션 크기는 다음 요소의 합계를 기준으로 계산됩니다.

델타

트랜잭션 내의 모든 문에 의해 생성되는 문서 변경 사항입니다. 여러 문서에 영향을 미치는 트랜잭션에서 총 델타 크기는 영향을 받는 각 문서의 개별 델타의 합계입니다.

메타데이터

영향을 받는 각 문서와 관련된 시스템 생성 트랜잭션 메타데이터입니다.

인덱스

트랜잭션의 영향을 받는 테이블에 인덱스가 정의된 경우, 연결된 인덱스 항목도 델타를 생성합니다.

기록

모든 문서 개정 내용이 QLDB에 유지되므로 모든 트랜잭션도 기록에 추가됩니다.

삽입 — 테이블에 삽입된 모든 문서에는 기록 테이블에도 사본이 삽입됩니다. 예를 들어, 새로 삽입한 100KB 문서는 트랜잭션 하나에 최소 200KB의 델타를 생성합니다. (이는 메타데이터나 인덱스를 포함하지 않는 대략적인 예상치입니다.)

업데이트 — 단일 필드의 경우도 문서가 업데이트되면 기록에 업데이트 델타를 더하거나 뺀 전체 문서의 새 개정을 생성합니다. 즉, 큰 문서를 조금만 업데이트해도 여전히 큰 트랜잭션 델타를 생성한다는 것을 의미합니다. 예를 들어, 기존 100KB 문서에 2KB의 데이터를 추가하면 기록에 102KB 개정본이 새로 생성됩니다. 이렇게 하면 한 트랜잭션에 최소 104KB의 총 델타가 추가됩니다. (다시 말하지만, 이 추정치에는 메타데이터나 인덱스가 포함되지 않습니다.)

삭제 — 업데이트와 마찬가지로 모든 삭제 트랜잭션은 기록에 새 문서 개정을 생성합니다. 하지만 새로 만든 `DELETE` 개정은 사용자 데이터가 null이고 메타데이터만 포함되어 있기 때문에 원본 문서보다 크기가 작습니다.

명명 제약 조건

다음 테이블은 Amazon QLDB의 명명 제약 조건을 설명한 것입니다.

원장 이름	• 1~32자의 영숫자 또는 하이픈만으로 구성되어야 합니다. • 첫 번째 글자와 마지막 문자는 문자나 숫자가 있어야 합니다. • 숫자로만 이루어져서는 안 됩니다. • 하이픈 2개가 연속될 수 없습니다. • 대소문자를 구분합니다.
저널 스트림 이름	
테이블 이름	• 1~128자의 영숫자 또는 밑줄만 포함해야 합니다. • 첫 번째 글자에는 문자 또는 밑줄이 있어야 합니다. • 나머지 문자에는 영숫자와 밑줄을 조합하여 사용할 수 있습니다. • 대소문자를 구분합니다. • QLDB PartiQL 예약어가 아니어야 합니다.

Amazon QLDB 관련 정보

다음의 관련 리소스는 이 서비스를 이용할 때 도움이 될 수 있습니다.

주제

- 기술 설명서
- GitHub 리포지토리
- AWS 블로그 게시물 및 기사
- 미디어
- 일반 AWS 리소스

기술 설명서

- Amazon QLDB FAQ - 제품에 대해 자주 묻는 질문입니다.
- Amazon QLDB 요금 - AWS 요금 정보 및 예제입니다.
- AWS re:Post - 질문과 답변(Q&A)을 위한 AWS 커뮤니티 포럼입니다.
- Amazon Ion - Amazon Ion 데이터 형식에 대한 개발자 가이드, 사용 설명서 및 참고 자료입니다.
- PartiQL- PartiQL 쿼리 언어에 대한 사양 문서 및 일반 자습서입니다.
- QLDB 워크숍 - Amazon QLDB를 사용하여 레코드 시스템 애플리케이션을 구축하는 실제 사례를 제공하는 워크숍으로, 다음 실습을 포함합니다.
  - QLDB 기본 사항 배우기
  - Amazon Ion을 사용한 작업 및 Ion을 JSON으로 또는 JSON에서 변환하기(Java)
  - AWS Glue 및 Amazon Athena를 사용하여 데이터 레이크용 QLDB 데이터를 활성화하기
  - Amazon Aurora MySQL DB 인스턴스로 QLDB 데이터 스트리밍
- Amazon QLDB를 사용한 변조 방지 품질 데이터 - QLDB를 사용하여 정확한 데이터 변경 기록을 유지함으로써 공격자가 품질 데이터를 조작하지 못하도록 방지하는 방법을 보여주는 AWS 솔루션 구현입니다. AWS 솔루션 구현을 통해 일반적인 문제를 해결하고 AWS를 사용하여 더 빠르게 빌드할 수 있습니다.

GitHub 리포지토리

드라이버

- .NET 드라이버 - QLDB 드라이버의 .NET 구현입니다.
- https://github.com/awslabs/amazon-qldb-driver-goGo 드라이버 - QLDB 드라이버의 Go 구현입니다.
- Java 드라이버 - QLDB 드라이버의 Java 구현입니다.
- Node.js 드라이버 - QLDB 드라이버의 Node.js 구현입니다.
- Python 드라이버 - QLDB 드라이버의 Python 구현입니다.

명령줄 셸

- QLDB 셸 - QLDB 트랜잭션 데이터 API를 위한 명령줄 인터페이스의 Python 및 Rust 구현.

샘플 애플리케이션

- Java DMV 애플리케이션 - 자동차 부서(DMV) 사용 사례를 기반으로 하는 자습서 애플리케이션. QLDB 및 Java용 QLDB 드라이버 사용에 대한 기본 작업 및 모범 사례를 보여줍니다.
- .NET DMV 애플리케이션 - QLDB 및 .NET용 QLDB 드라이버 사용에 대한 기본 작업 및 모범 사례를 보여주는 DMV 기반 자습서 애플리케이션입니다.
- Node.js DMV 애플리케이션 - QLDB 및 Node.js 용 QLDB 드라이버 사용에 대한 기본 작업 및 모범 사례를 보여주는 DMV 기반 자습서 애플리케이션입니다.
- Python DMV 애플리케이션 - QLDB 및 Python용 QLDB 드라이버 사용에 대한 기본 작업과 모범 사례를 보여주는 DMV 기반 자습서 애플리케이션입니다.
- 원장 로더 - 지원되는 전송 채널(AWS DMS, Amazon SQS, Amazon SNS, Kinesis Data Streams, Amazon MSK 또는 EventBridge)을 사용하여 QLDB 원장에 데이터를 비동기적으로 고속으로 로드하기 위한 Java 프레임워크입니다.
- 내보내기 프로세서 - 내보내기 출력을 읽고 내보낸 블록을 순서대로 반복하여 Amazon S3에서 QLDB 내보내기를 처리하는 작업을 처리하는 확장 가능한 Java 프레임워크입니다.
- Python의 QLDB 스트림 샘플 Lambda - Amazon SQS 대기열이 구독되어 있는 Amazon SNS 주제에 QLDB 데이터를 전송하는 AWS Lambda 함수를 사용하여 QLDB 스트림을 사용하는 방법을 보여주는 애플리케이션입니다.
- QLDB 스트림 OpenSearch 통합 샘플 - 스트림을 사용하여 Amazon OpenSearch Service를 QLDB와 통합하는 방법을 보여주는 Python 애플리케이션입니다.

- 이중 입력 애플리케이션 - QLDB를 사용하여 이중 입력 재무 원장 애플리케이션을 모델링하는 방법을 보여주는 Java 애플리케이션입니다.
- Node.js용 QLDB KVS - 문서 검증을 위한 추가 기능을 갖춘 QLDB용 간단한 키-값 스토어 인터페이스 라이브러리입니다.

Amazon Ion 및 PartiQL

- Amazon Ion 라이브러리 - Ion 팀이 지원하는 라이브러리, 도구 및 설명서.
- PartiQL 구현 - PartiQL의 구현, 사양 및 자습서.

AWS 블로그 게시물 및 기사

- How Earnin built their ledger service using Amazon QLDB (2023년 2월 16일) - Earnin.com이 QLDB를 사용하여 사용자에게 최신 기능을 갖춘 모바일 금융 애플리케이션을 제공하는 원장 서비스를 구축한 방법을 설명합니다.
- Export and analyze Amazon QLDB journal data using AWS Glue and Amazon Athena (2022년 12월 19일) - AWS Glue 및 Athena와 함께 QLDB의 내보내기 기능을 사용하여 원장 기반 아키텍처에 보고 및 분석 기능을 제공하는 방법에 대해 설명합니다.
- How Shinsegae International enhances customer experience and prevents counterfeiting with Amazon QLDB (2022년 8월 3일) - Shinsegae International이 Amazon QLDB를 사용하여 고객에게 명품의 정품 여부를 알리고 제품의 구매 및 유통 내역을 제공하는 디지털 정품 확인 서비스를 구축한 방법을 설명합니다.
- BungkusIT uses Amazon QLDB and VeriDoc Global's ISV technology to improve the customer and delivery agent experience (2022년 4월 29일) - 물류 회사인 BungkusIT이 QLDB를 사용하여 변경 불가능하고 암호로 검증 가능한 트랜잭션 로그를 사용하여 업계 간 커뮤니케이션을 위한 중앙 집중식 플랫폼을 구현한 방법을 보여줍니다.
- Use Amazon QLDB as an immutable key-value store with a REST API and JSON (2022년 2월 14일) - QLDB를 변경 불가능한 키-값 스토어로 사용하고 REST API를 통해 QLDB의 감사 기능을 사용하는 방법을 소개합니다.
- Building a core banking system with Amazon QLDB (2022년 1월 21일) - QLDB를 사용하여 코어 뱅킹 원장 시스템을 구축하는 방법을 보여줍니다. 또한 QLDB가 금융 서비스 업계의 요구 사항에 적합한 데이터베이스인 이유도 보여줍니다.

- How Specright uses Amazon QLDB to create a traceable supply chain network (2022년 1월 21일) - Specright가 QLDB를 사용하여 도매 브랜드, 소매업체 및 제조업체가 중요한 공급망 데이터 및 포장 사양 데이터를 공유할 수 있도록 하는 추적 가능한 공급망 네트워크를 구축하는 방법을 보여줍니다.
- How fEMR Delivers Cryptographically Secure and Verifiable Medical Data with Amazon QLDB (2021년 12월 23일) - fEMR 팀이 QLDB 및 기타 AWS 관리형 서비스를 사용하여 구호 활동을 지원한 방법을 살펴봅니다.
- Monitor Amazon QLDB query access patterns (2021년 11월 8일) - 트랜잭션에서 실행된 QLDB 트랜잭션 및 PartiQL 명령문을 Amazon API Gateway를 통해 수신된 API 요청과 연결하는 방법을 보여줍니다.
- Build a simple CRUD operation and data stream on Amazon QLDB using AWS Lambda (2021년 9월 28일) - AWS Lambda 함수를 사용하여 QLDB에서 CRUD(생성, 읽기, 업데이트, 삭제) 작업을 수행하는 방법을 보여줍니다.
- Real-world cryptographic verification with Amazon QLDB (2021년 8월 26일) - 실제 사용 사례의 맥락에서 QLDB 원장에서의 암호화 검증의 가치에 대해 설명합니다.
- Verify delivery conditions with the Accord Project and Amazon QLDB: Part 1, Part 2 (2021년 6월 28일) - 오픈 소스 Accord Project 및 QLDB를 사용하여 배송 조건을 확인하기 위해 스마트 법률 계약 기술을 적용하는 방법을 설명합니다.
- Amazon QLDB data streaming via AWS CDK (2021년 6월 7일) - AWS Cloud Development Kit (AWS CDK)를 사용하여 QLDB를 설정하고, AWS Lambda 함수를 사용하여 QLDB 데이터를 채우고, QLDB 스트리밍을 설정하여 원장 데이터의 데이터 복원력을 제공하는 방법을 보여줍니다.
- Demo fine grained access control in QLDB (2021년 6월 1일) - QLDB 원장에 대한 세분화된 AWS Identity and Access Management(IAM)권한을 시작하는 방법을 보여줍니다.
- Stream Processing with DynamoDB Streams and QLDB Streams (2020년 11월 23일) - DynamoDB 스트림과 QLDB 스트림을 간략하게 비교하고 이를 시작하는 방법에 대한 팁을 제공합니다.
- Streaming data from Amazon QLDB to OpenSearch (2020년 8월 19일) - QLDB에서 Amazon OpenSearch Service로 데이터를 스트리밍하여 서식 있는 텍스트 검색과 다운스트림 분석(예: 레코드 간 집계 또는 지표)을 지원하는 방법을 설명합니다.
- How I streamed data from Amazon QLDB to DynamoDB using Nodejs in near-real time (2020년 7월 7일) - QLDB에서 DynamoDB로 데이터를 스트리밍하여 한 자릿수 지연 시간과 무한히 확장 가능한 키-값 쿼리를 지원하는 방법을 설명합니다.
- Building a GraphQL interface to Amazon QLDB with AWS AppSync: Part 1, Part 2 (2020년 5월 4일) - QLDB와 AWS AppSync를 통합하여 QLDB 원장에 활용도 높은 GraphQL기반 API를 제공하는 방법에 대해 설명합니다.

미디어

AWS 동영상

- AWS Tutorials & Demos: QLDB to Aurora Streaming (2023년 3월 17일, 21분) - 이 동영상에서는 QLDB 스트리밍 기능을 구현하기 위한 기본 개념을 설명하고, QLDB에서 Amazon Aurora MySQL 다운스트림 DB로 데이터 스트리밍을 설정하는 방법을 보여줍니다.
- ArcBlock: Leveraging Amazon QLDB to Build a Decentralized Identity Solution (2022년 5월 31일, 4분) - ArcBlock이 QLDB를 사용하여 구축한 분산형 ID 솔루션에 대해 설명합니다.
- AWS re:Invent 2020: Using Amazon QLDB as a system-of-trust database for core business apps (2021년 2월 5일, 30분) - Amazon QLDB 초기 사용자가 데이터 출처 및 암호화 검증에 대한 원장 데이터베이스의 고유한 속성을 적용하여 데이터 무결성이 내장된 레코드 시스템을 구현한 방법을 알아봅니다.
- AWS re:Invent 2020: Building out a serverless application with Amazon QLDB (2021년 2월 5일, 28분) - Amazon QLDB를 AWS Lambda, Amazon Kinesis 및 Amazon DynamoDB와 같은 서비스와 결합하여 모든 기능을 갖춘 서버리스 애플리케이션을 빌드, 테스트 및 최적화하는 방법을 알아봅니다.
- AWS re:Invent 2020: Building audit-based apps that maintain data integrity with Amazon QLDB (2021년 2월 5일, 18분) - 이 세션에서는 Amazon QLDB가 해결할 수 있는 문제를 자세히 살펴보고, 원장 데이터베이스를 언제, 왜 사용할지에 대한 질문에 답하고, Osano와 같은 고객의 사용 사례를 공유합니다.
- How to Centrally Store Immutable Logs using Amazon QLDB with .NET Applications (2020년 12월 7일, 10분) -.NET 애플리케이션과 함께 QLDB를 사용하여 변경할 수 없는 로그를 중앙에 저장하는 방법을 보여줍니다.
- Virtual Workshop: Building Ledger-Based Systems of Record with QLDB and Java - AWS Online Tech Talks (2020년 7월 29일, 87분) - 강사가 진행하는 워크숍으로, Working With Ion Immersion Day 실습을 통해 Amazon Ion 라이브러리와 Java용 QLDB 드라이버를 사용하여 데이터 로더 프로그램을 빌드합니다.
- Developer Workshop: Using AWS's Immutable Ledger Database QLDB on ABT Node (2020년 6월 22일, 34분) - ABT 노드에서 QLDB를 설정하고 구성하는 방법에 대한 단계별 가이드입니다. 또한 QLDB에 연결하기 위해 ArcBlock의 Blockchain Explorer와 Boarding Gate Blocklets을 설치하고 구성하는 방법도 설명합니다.
- AWS Tech Talk: A Customer's Perspective on Building an Event-Triggered System-of-Record Application with Amazon QLDB (2020년 3월 31일, 29분) - AWS Data Hero이자 영국 DVLA(운전자 및 차량 면허 기관)의 수석 설계자인 Matt Lewis의 테크 토크로, 이 테크 토크에서는 DVLA의 QLDB 사용 사례와 해당 애플리케이션의 이벤트 기반 아키텍처를 설명합니다.

- AWS re:Invent 2019: Why you need a ledger database: BMW, DVLA, & Sage discuss use cases (2019년 12월 5일, 47분) - 원장 데이터베이스를 사용해야 하는 이유를 설명하고 QLDB의 사용 사례를 공유하는 BMW, 영국 정부 기관 DVLA 및 Sage 고객이 진행하는 프레젠테이션입니다.
- AWS re:Invent 2019: Amazon QLDB: An engineer's deep dive on why this is a game changer (2019년 12월 5일, 50분) - Andrew Certain(AWS 저명한 엔지니어)이 QLDB의 고유한 저널 우선 아키텍처와 다양한 혁신 기술에 대해 설명하는 프레젠테이션입니다. 여기에는 암호화 해싱, Merkle 트리, 다중 가용 영역 복제, PartiQL 지원이 포함됩니다.
- Building Applications with Amazon QLDB, a First-of-Its-Kind Ledger Database - AWS Online Tech Talks (2019년 11월 19일, 51분) - QLDB의 고유한 기능과 핵심 기능을 사용하는 방법을 구체적으로 설명하는 심층 테크 토크입니다. 여기에는 데이터의 전체 기록을 쿼리하고, 문서를 암호학적으로 검증하고, 데이터 모델을 설계하는 작업이 포함됩니다.

팟캐스트

- Why are customers choosing Amazon QLDB? (2020년 7월 5일, 33분) - 원장 데이터베이스의 정의, 블록체인과 어떻게 다른지, 오늘날 고객이 이를 어떻게 사용하고 있는지에 대해 설명하는 토론입니다.

일반 AWS 리소스

- 교육 및 워크숍 – 역할 기반의 과정 및 전문 과정은 물론 자습형 실습에 대한 링크를 통해 AWS 기술을 연마하고 실무에 도움이 되는 경험을 쌓을 수 있습니다.
- AWS 개발자 센터 – 자습서를 살펴보고, 도구를 다운로드하고, AWS 개발자 이벤트에 대해 알아보세요.
- AWS 개발자 도구 – AWS 애플리케이션을 개발 및 관리하기 위한 개발자 도구, SDK, IDE 도구 키트 및 명령줄 도구 링크입니다.
- 시작하기 리소스 센터 - AWS 계정을(를) 설정하고 AWS 커뮤니티에 가입하고 첫 번째 애플리케이션을 시작하는 방법을 알아보세요.
- 실습 자습서 – 단계별 자습서에 따라 AWS에서 첫 번째 애플리케이션을 시작하세요.
- AWS 백서 – AWS 솔루션 아키텍트 또는 기타 기술 전문가가 아키텍처, 보안 및 경제 등의 주제에 대해 작성한 포괄적 AWS 기술 백서 목록의 링크입니다.
- AWS Support 센터 – AWS Support 사례를 생성하고 관리할 수 있는 허브입니다. 또한 포럼, 기술 FAQ, 서비스 상태 및 AWS Trusted Advisor 등의 기타 유용한 자료에 대한 링크가 있습니다.

- AWS Support – 클라우드에서 1대 1로 애플리케이션을 구축 및 실행하도록 지원하는 빠른 응답 지원 채널인 AWS Support에 대한 정보가 포함된 기본 웹 페이지입니다.
- 문의처 - AWS 결제, 계정, 이벤트, 침해 및 기타 문제에 대해 문의할 수 있는 중앙 연락 창구입니다.
- AWS 사이트 약관 – 저작권 및 상표, 사용자 계정, 라이선스 및 사이트 액세스와 기타 주제에 대한 세부 정보입니다.

Amazon QLDB 릴리스 이력

다음 표는 Amazon QLDB의 각 릴리스에서 변경된 중요 사항과 Amazon QLDB 개발자 가이드의 해당 업데이트를 설명합니다. 이 설명서에 대한 업데이트 알림을 받으려면 RSS 피드에 가입하면 됩니다.

- API 버전: 2019-01-02
- 최근 설명서 업데이트: 2023년 1월 3일

변경 사항	설명	날짜
IAM 지침을 업데이트함	IAM 모범 실무에 따라 가이드가 업데이트되었습니다. 자세한 설명은 IAM의 보안 모범 사례를 참조하세요.	2023년 1월 3일
AWS 관리형 정책으로 업데이트	Amazon QLDB는 기존 관리형 AWS 정책 `AmazonQLDBFullAccess` 및 `AmazonQLDBConsoleFullAccess` 을 업데이트했습니다. 이러한 정책은 주체가 PartiQL 저장 프로시저를 사용하여 문서 수정본을 수정할 수 있는 새로운 권한이 있습니다. 자세한 설명은 Amazon QLDB를 위한 AWS 관리형 정책을 참조하세요.	2022년 11월 4일
데이터 개정	Amazon QLDB는 이제 2021년 7월 22일 또는 그 이후에 생성된 원장에 대해 `REDACT_REVISION` PartiQL 저장 프로시저를 지원합니다. 이 저장 프로시저를 사용하면 기록에서 비활성 문서 수정 내용을 영구적으로 삭제하면서도 원장의 전	2022년 11월 3일

	체 데이터 무결성을 유지할 수 있습니다. 자세한 설명은 문서 수정본 수정을 참조하세요.	
Node.js 드라이버 v3	Node.js 버전 3.0용 Amazon QLDB 드라이버가 이제 일반 공개되었습니다. 이 버전에서는 AWS SDK for JavaScript v3를 위한 지원이 도입되었습니다. 릴리스 노트는 GitHub 리포지토리 awslabs/amazon-qldb-driver-nodejs를 참조하세요.	2022년 9월 26일
Go 드라이버 v3	Go 버전 3.0용 Amazon QLDB 드라이버가 이제 일반 공개되었습니다. 이 버전에서는 AWS SDK for Go v2를 위한 지원이 도입되었습니다. 릴리스 노트는 GitHub 리포지토리 awslabs/amazon-qldb-driver-go를 참조하세요.	2022년 8월 11일
새로운 PartiQL 쿼리 편집기	Amazon QLDB 콘솔의 새로운 PartiQL 쿼리 편집기가 이제 일반 공개되었습니다. 새로운 QLDB PartiQL 편집기는 쿼리 작성, 트랜잭션 디버깅 및 결과 탐색을 위한 향상된 인터페이스를 제공합니다. 편집기 열기 및 사용에 대한 자세한 설명은 Accessing Amazon QLDB using the console을 참조하세요.	2022년 6월 22일

.NET 드라이버용 Ion 객체 매퍼	.NET 버전 1.3용 Amazon QLDB 드라이버에 Ion 객체 매퍼에 대한 지원이 도입되었습니다. 이 기능을 사용하면 Amazon Ion 타입과 네이티브 C# 타입 사이를 수동으로 변환할 필요가 전혀 없습니다. Ion 객체 매퍼의 전체 변경 내역은 GitHub 리포지토리 `amzn/ion-object-mapper-dotnet` 의 CHANGELOG.md 파일을 참조하세요.	2022년 1월 19일
JSON 저널 내보내기 형식	Amazon QLDB는 이제 저널 내보내기를 위한 JSON Lines 출력 형식을 지원합니다. 자세한 설명은 Amazon QLDB로부터 저널 데이터 내보내기를 참조하세요.	2021년 12월 21일
Amazon QLDB 문제 해결	Amazon QLDB를 사용할 때 발생할 수 있는 일반적인 오류의 전체 목록에 대한 지침을 제공하는 새로운 문제 해결 주제가 추가되었습니다.	2021년 12월 8일
새로운 지역 출범	Amazon QLDB를 이제 캐나다(중부) 지역에서 사용할 수 있습니다. 사용 가능한 리전의 전체 목록은 Amazon Web Services 일반 참조에서 Amazon QLDB 엔드포인트 및 할당량을 참조하십시오.	2021년 11월 11일

교차 서비스 혼동된 대리자 예방	Amazon QLDB는 이제 혼동된 대리자 문제를 방지하기 위해 IAM 리소스 정책에서 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키 사용을 지원합니다. 자세한 설명은 교차 서비스 혼동된 대리자 예방을 참조하세요.	2021년 11월 8일
Amazon QLDB 쉘 v2	Rust에서 작성된 Amazon QLDB 쉘 버전 2.0가 이제 일반 공개되었습니다. 릴리스 노트는 GitHub 리포지토리 awslabs/amazon-qldb-shell을 참조하세요.	2021년 10월 14일
AWS 관리형 정책으로 업데이트	Amazon QLDB는 기존 관리형 AWS 정책 `AmazonQLDBFullAccess` 및 `AmazonQLDBConsoleFullAccess` 을 업데이트했습니다. 이러한 정책에는 주체가 사용자 계정의 모든 IAM 역할 리소스를 QLDB 서비스에 전달할 수 있는 권한이 새로 추가되었습니다. 이는 모든 저널 내보내기 및 스트림 요청을 위해 요구됩니다. 자세한 설명은 Amazon QLDB를 위한 AWS 관리형 정책을 참조하세요.	2021년 9월 2일

고객 관리형 AWS KMS 키	Amazon QLDB는 이제 새 원장 리소스에 대한 AWS Key Management Service(AWS KMS)의 고객 관리형 키를 사용하여 저장된 암호화를 지원합니다. 자세한 설명은 Amazon QLDB에서의 저장 시 암호화를 참조하세요.	2021년 7월 22일
AWS 관리형 정책으로 업데이트	Amazon QLDB는 이전에 두 번 나열된 중복 `qldb:GetBlock` 작업을 제거하기 위해 기존 AWS 관리형 정책 `AmazonQLDBReadOnly` 을 업데이트했습니다. 자세한 설명은 Amazon QLDB를 위한 AWS 관리형 정책을 참조하세요.	2021년 7월 1일
새로운 지역 출범	Amazon QLDB를 이제 유럽(런던) 지역에서 사용할 수 있습니다. 사용 가능한 리전의 전체 목록은 Amazon Web Services 일반 참조에서 Amazon QLDB 엔드포인트 및 할당량을 참조하십시오.	2021년 6월 24일

AWS 관리형 정책으로 업데이트	Amazon QLDB는 기존 관리형 AWS 정책 `AmazonQLDBFullAccess` 및 `AmazonQLDBConsoleFullAccess` 을 업데이트했습니다. 이러한 정책에는 주체가 모든 원장의 권한 모드를 업데이트하고 모든 `STANDARD` 권한 원장에서 모든 PartiQL 명령을 실행할 수 있는 권한이 새로 추가되었습니다. 자세한 설명은 Amazon QLDB를 위한 AWS 관리형 정책을 참조하세요.	2021년 5월 27일
표준 권한 모드	Amazon QLDB는 이제 원장 리소스를 위한 `STANDARD` 권한 모드를 지원합니다. 표준 권한 모드를 사용하면 원장, 표 및 PartiQL 명령에 대해 보다 세분화된 액세스를 제어할 수 있습니다. 자세한 내용은 표준 권한 모드로 시작하기를 참조하세요.	2021년 5월 27일
PartiQL 문 통계	이제 Amazon QLDB 콘솔의 쿼리 편집기에서 PartiQL 명령문 통계 기능을 사용할 수 있습니다. 자세한 설명은 문 통계 받기를 참조하십시오.	2021년 5월 24일

자습서: AWS SDK를 사용한 데이터 검증	AWS SDK를 통해 QLDB API를 사용하여 Amazon QLDB의 수정본 해시 및 블록 해시를 확인하는 방법을 보여주는 코드 예가 포함된 단계별 자습서가 추가되었습니다. 자세한 설명은 자습서: AWS SDK를 사용한 데이터 검증을 참조하세요.	2021년 5월 6일
.NET 드라이버 v1.2	.NET 버전 1.2용 Amazon QLDB 드라이버가 이제 일반 공개되었습니다. 이 버전에는 비동기 API가 도입되었습니다. 릴리스 노트는 GitHub 리포지토리 awslabs/amazon-qldb-driver-dotnet을 참조하세요.	2021년 4월 1일
PartiQL `DROP INDEX` 문	Amazon QLDB의 PartiQL은 이제 DROP INDEX 문을 지원합니다. 자세한 설명은 인덱스 떨어뜨리기를 참조하세요.	2021년 3월 3일
PartiQL 문 통계	이제 최신 버전의 Amazon QLDB 드라이버에서 .NET, Go 및 Python을 포함하여 지원되는 모든 언어에 대해 PartiQL 명령문 통계 기능을 사용할 수 있습니다. 자세한 설명은 문 통계 받기를 참조하십시오.	2021년 2월 25일
TypeScript 빠른 시작 자습서	Node.js 용 Amazon QLDB 드라이버용 빠른 시작 자습서에 TypeScript 코드 예가 추가되었습니다.	2020년 12월 28일

PartiQL 문 통계	이제 Java 및 Node.js 용 Amazon QLDB 드라이버의 최신 버전은 더 효율적인 PartiQL 문을 실행하는 데 도움이 되는 명령문 실행 통계를 제공합니다. 자세한 설명은 문 통계 받기를 참조하십시오.	2020년 12월 22일
드라이버 쿡북 참조 및 빠른 시작 자습서	Go 및 Node.js 드라이버에 대한 쿡북 참조, Java 및 Python 드라이버에 대한 빠른 시작 자습서, QLDB에서 Amazon Ion을 사용하기 위한 가이드가 추가되었습니다. 자세한 설명은 Amazon QLDB 드라이버 시작하기를 참조하세요.	2020년 11월 24일
Amazon QLDB 쉘 v1.1	Amazon QLDB 쉘 버전 1.1이 이제 일반 공개되었습니다. 릴리스 노트는 GitHub 리포지토리 awslabs/amazon-qldb-shell을 참조하세요.	2020년 10월 26일
Go용 Amazon QLDB 드라이버	Go용 Amazon QLDB 드라이버가 이제 일반 공개되었습니다. 이 드라이버는 GitHub의 오픈 소스이며, AWS SDK for Go를 사용하여 QLDB의 트랜잭션 데이터 API와 상호 작용할 수 있습니다. 릴리스 노트는 GitHub 리포지토리 awslabs/amazon-qldb-driver-go를 참조하세요.	2020년 10월 20일

Node.js 드라이버 설정 권장 사항	연결 유지를 통한 연결을 재사용하여 지연 시간을 행이는 방법을 포함하여 Node.js용 Amazon QLDB 드라이버에 대한 설정 권장 사항을 제공하는 새 섹션이 추가되었습니다.	2020년 10월 16일
비어 있지 않은 표에 인덱스 생성	Amazon QLDB는 이제 비어 있지 않은 표에 대한 인덱스 생성을 지원합니다. 자세한 설명은 인덱스 관리를 참조하세요.	2020년 9월 30일
쿼리 성능 최적화	Amazon QLDB의 쿼리 제약 조건을 설명하고 이러한 제약 조건을 고려하여 쿼리 성능 최적화에 대한 지침을 제공하는 새 섹션이 추가되었습니다.	2020년 9월 18일
Java 드라이버에 대한 쿡북 참조	Java용 Amazon QLDB 드라이버의 일반적인 사용 사례의 코드 예를 보여주는 쿡북 참조가 추가되었습니다.	2020년 9월 3일
드라이버를 사용한 세션 관리	Amazon QLDB에서 드라이버를 사용한 세션 관리에 대한 개요를 제공하고 데이터 트랜잭션을 실행할 때 QLDB 드라이버가 세션을 처리하는 방법을 설명하는 새 섹션이 추가되었습니다.	2020년 9월 1일
Java 드라이버 v2.0	Java 버전 2.0용 Amazon QLDB 드라이버가 이제 일반 공개되었습니다. 릴리스 노트는 GitHub 리포지토리 awslabs/amazon-qldb-driver-java를 참조하세요.	2020년 8월 28일

Node.js 드라이버 v2.0	Node.js 버전 2.0용 Amazon QLDB 드라이버가 이제 일반 공개되었습니다. 릴리스 노트는 GitHub 리포지토리 awslabs/amazon-qldb-driver-nodejs를 참조하세요.	2020년 8월 27일
관련 정보	Amazon QLDB를 이해하고 사용하는 데 도움이 되는 관련 정보와 추가 리소스 링크가 포함된 새 주제가 추가되었습니다.	2020년 8월 24일
Python 드라이버 v3.0	Python 버전 3.0용 Amazon QLDB 드라이버가 이제 일반 공개되었습니다. 릴리스 노트는 GitHub 리포지토리 awslabs/amazon-qldb-driver-python을 참조하세요.	2020년 8월 20일
Go용 Amazon QLDB 드라이버	Go용 Amazon QLDB 드라이버의 미리 보기 릴리스를 이제 사용할 수 있습니다. 이 드라이버를 사용하면 AWS SDK for Go를 사용하여 QLDB의 트랜잭션 데이터 API와 상호 작용할 수 있습니다. 자세한 설명은 Go용 Amazon QLDB 드라이버(미리 보기)를 참조하세요.	2020년 8월 6일
Python 드라이버에 대한 쿡북 참조	Python용 Amazon QLDB 드라이버의 일반적인 사용 사례의 코드 예를 보여 주는 쿡북 참조가 추가되었습니다.	2020년 7월 24일
Amazon QLDB의 고유 ID	Amazon QLDB의 고유 ID 속성 및 사용 지침을 설명하는 새 섹션이 추가되었습니다.	2020년 7월 9일

저널 스트림에 대한 AWS CloudFormation 리소스	Amazon QLDB는 이제 AWS CloudFormation 템플릿을 사용하여 저널 스트림을 생성하기 위한 리소스를 지원합니다. 자세한 설명은 AWS CloudFormation 사용자 가이드의 AWS::QLDB::Stream 리소스를 참조하세요.	2020년 7월 9일
.NET 드라이버에 대한 쿡북 참조	.NET용 Amazon QLDB 드라이버의 일반적인 사용 사례의 코드 예를 보여주는 쿡북 참조가 추가되었습니다.	2020년 7월 1일
.NET용 Amazon QLDB 드라이버	.NET용 Amazon QLDB 드라이버가 이제 일반 공개되었습니다. 이 드라이버는 GitHub의 오픈 소스이며, AWS SDK for .NET를 사용하여 QLDB의 트랜잭션 데이터 API와 상호 작용할 수 있습니다. 릴리스 노트는 GitHub 리포지토리 awslabs/amazon-qldb-driver-dotnet을 참조하세요.	2020년 6월 26일
Node.js용 Amazon QLDB 드라이버	Node.js용 Amazon QLDB 드라이버가 이제 일반 공개되었습니다. 이 드라이버는 GitHub의 오픈 소스이며, Node.js의 JavaScript용 AWS SDK를 사용하여 QLDB의 트랜잭션 데이터 API와 상호 작용할 수 있습니다. 릴리스 노트는 awslabs/amazon-qldb-driver-nodejs GitHub 리포지토리를 참조하세요.	2020년 6월 5일

Amazon QLDB 저널 스트림	이제 Amazon QLDB를 사용하여 저널에 체결된 모든 문서 수정본을 수집하고 이 데이터를 Amazon Kinesis Data Streams에 거의 실시간으로 전송하는 스트림을 생성할 수 있습니다. 자세한 설명은 Amazon QLDB에서 저널 데이터 스트리밍을 참조하세요.	2020년 5월 19일
Amazon Ion 코드 예제	Java, Node.js 및 Python용 QLDB 드라이버를 사용하여 Amazon QLDB 원장의 Amazon Ion 데이터를 쿼리하고 처리하는 코드 예가 추가되었습니다. 자세한 설명은 Amazon QLDB의 Ion 코드 예를 참조하세요.	2020년 5월 12일
동시성 모델 및 저널 콘텐츠	인덱스를 사용하여 OCC(낙관적 동시성 제어) 충돌을 제한하는 방법에 대한 정보를 추가하도록 Amazon QLDB 동시성 모델 주제가 확장되었습니다. Amazon QLDB의 저널 콘텐츠를 설명하는 새 섹션이 추가되었습니다.	2020년 5월 4일
Node.js 드라이버용 빠른 시작 가이드	Node.js용 Amazon QLDB 드라이버에 대한 빠른 시작 가이드가 추가되었습니다.	2020년 5월 1일

Amazon QLDB 쉘	Amazon QLDB 쉘이 이제 일반 공개되었습니다. 이 쉘은 GitHub의 오픈 소스이며, 원장 데이터에서 PartiQL 문을 실행할 수 있는 명령행 인터페이스를 제공합니다. 릴리스 노트는 awslabs/amazon-qldb-shell GitHub 리포지토리를 참조하세요.	2020년 4월 20일
규정 준수 인증	이제 Amazon QLDB는 HIPAA 및 ISO를 포함한 AWS 규정 준수 프로그램에 대한 인증을 받았습니다. 자세한 설명은 Amazon QLDB를 위한 규정 준수 확인을 참조하십시오.	2020년 4월 3일
확장된 드라이버 권장 사항	지원되는 모든 프로그래밍 언어에 적용할 수 있도록 Amazon QLDB 드라이버 권장 사항 섹션이 확장되었습니다.	2020년 4월 2일
Amazon QLDB 쉘	이제 Amazon QLDB 쉘의 미리 보기 릴리스를 사용할 수 있습니다. 이 쉘은 원장 데이터에 대해 PartiQL 문을 실행할 수 있는 명령행 인터페이스를 제공합니다. 자세한 설명은 QLDB 쉘을 사용하여 Amazon QLDB에 액세스(데이터 API만 해당)(미리 보기)를 참조하세요.	2020년 3월 23일

Java 드라이버 v1.1	Java 버전 1.1용 Amazon QLDB 드라이버가 이제 일반 공개되었습니다. 릴리스 노트는 awslabs/amazon-qldb-driver-java GitHub 리포지토리를 참조하세요.	2020년 3월 20일
.NET용 Amazon QLDB 드라이버	Amazon QLDB는 이제 .NET 드라이버의 미리 보기 릴리스를 제공합니다. 이 드라이버를 사용하면 AWS SDK for .NET를 사용하여 QLDB의 트랜잭션 데이터 API와 상호 작용할 수 있습니다. 자세한 설명은 .NET용 Amazon QLDB 드라이버(미리 보기)를 참조하세요.	2020년 3월 13일
Python용 Amazon QLDB 드라이버	Python용 Amazon QLDB 드라이버가 이제 일반 공개되었습니다. 이 드라이버는 GitHub의 오픈 소스이며, AWS SDK for Python (Boto3)를 사용하여 QLDB의 트랜잭션 데이터 API와 상호 작용할 수 있습니다. 릴리스 노트는 awslabs/amazon-qldb-driver-python GitHub 리포지토리를 참조하세요.	2020년 3월 11일

PartiQL UNDROP TABLE 문 및 중첩된 DML	Amazon QLDB의 PartiQL은 이제 중첩된 컬렉션을 FROM 절의 소스로 지정하는 데이터 조작 언어(DML) 문을 지원합니다. 자세한 설명은 PartiQL 참조의 FROM 문을 참조하세요. QLDB PartiQL은 UNDROP TABLE 문도 지원합니다. 자세한 설명은 표 떨어뜨리기 취소를 참조하세요.	2020년 2월 26일
확장된 소개 주제	새로운 섹션 Amazon QLDB 개요 및 관계형에서 원장까지를 포함하도록 입문용 Amazon QLDB란 무엇입니까? 주제가 확장되었습니다.	2020년 1월 24일
지원되는 함수에 대한 PartiQL 참조	지원되는 SQL 함수에 대한 자세한 사용 정보를 포함하도록 Amazon QLDB PartiQL 참조가 확장되었습니다. 자세한 설명은 PartiQL 함수를 참조하십시오.	2020년 1월 2일
드라이버 권장 사항 및 흔한 오류	Java용 Amazon QLDB 드라이버를 위한 드라이버 권장 사항 및 모든 드라이버 언어를 위한 흔한 오류를 설명하는 새 섹션들이 추가되었습니다.	2020년 1월 2일

새로운 지역 출범	이제 Amazon QLDB는 아시아 태평양(서울), 아시아 태평양(싱가포르), 아시아 태평양(시드니), 유럽(프랑크푸르트) 지역에서 사용 가능합니다. 사용 가능한 리전의 전체 목록은 Amazon Web Services 일반 참조에서 Amazon QLDB 엔드포인트 및 할당량을 참조하십시오.	2019년 11월 19일
Node.js용 Amazon QLDB 드라이버	Amazon QLDB는 이제 Node.js 드라이버의 미리 보기 릴리스를 제공합니다. 이 드라이버를 사용하면 Node.js의 JavaScript용 AWS SDK를 사용하여 QLDB의 트랜잭션 데이터 API와 상호 작용할 수 있습니다. 자세한 설명은 Node.js용 Amazon QLDB 드라이버(미리 보기)를 참조하세요.	2019년 11월 13일
Python용 Amazon QLDB 드라이버	Amazon QLDB는 이제 Python 드라이버의 미리 보기 릴리스를 제공합니다. 이 드라이버를 사용하면 AWS SDK for Python (Boto3)를 사용하여 QLDB의 트랜잭션 데이터 API와 상호 작용할 수 있습니다. 자세한 설명은 Python용 Amazon QLDB 드라이버(미리 보기)를 참조하세요.	2019년 10월 29일
공개 릴리스	Amazon QLDB의 최초 공개 릴리스입니다. 이 릴리스에는 개발자 가이드 및 통합 원장 관리 API 참조 문서가 포함됩니다.	2019년 9월 10일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.