Amazon Quantum Ledger Database (Amazon QLDB)

Developer Guide

Amazon Quantum Ledger Database (Amazon QLDB): Developer Guide

Table of Contents

What is Amazon QLDB?

Amazon Quantum Ledger Database (Amazon QLDB) is a fully managed ledger database that provides a transparent, immutable, and cryptographically verifiable transaction log owned by a central trusted authority. You can use Amazon QLDB to track all application data changes, and maintain a complete and verifiable history of changes over time.

Ledgers are typically used to record a history of economic and financial activity in an organization. Many organizations build applications with ledger-like functionality because they want to maintain an accurate history of their applications' data. For example, they might want to track the history of credits and debits in banking transactions, verify the data lineage of an insurance claim, or trace the movement of an item in a supply chain network. Ledger applications are often implemented using custom audit tables or audit trails created in relational databases.

Amazon QLDB is a new class of database that helps eliminate the need to engage in the complex development effort of building your own ledger-like applications. With QLDB, the history of changes to your data is immutable—it cannot be altered, updated, or deleted. And using cryptography, you can easily verify that there have been no unintended changes to your application's data. QLDB uses an immutable transactional log, known as a *journal*. The journal is append-only and is composed of a sequenced and hash-chained set of *blocks* that contain your committed data.

Amazon QLDB video

For an overview of Amazon QLDB and how it can benefit you, watch this QLDB overview video on YouTube.

Amazon QLDB pricing

With Amazon QLDB, you pay only for what you use with no minimum fees or mandatory service usage. You pay only for the resources your ledger database consumes, and you do not need to provision in advance.

For more information, see Amazon QLDB pricing.

Getting started with QLDB

We recommend that you begin by reading the following topics:

- **Overview of Amazon QLDB (p. 2)**—To get a high-level overview of QLDB.
- **Core concepts and terminology in Amazon QLDB (p. 7)**—To learn fundamental QLDB concepts and terminology.
- **Accessing Amazon QLDB (p. 15)**—To learn how to access QLDB using the AWS Management Console, API, or AWS Command Line Interface (AWS CLI).

To get started quickly with the QLDB console, see Getting started with the Amazon QLDB console (p. 25).

To learn about developing with QLDB using an AWS-provided driver, see Getting started with the Amazon QLDB driver (p. 40).

Overview of Amazon QLDB

The following sections provide a high-level overview of Amazon QLDB service components and how they interact.

Topics

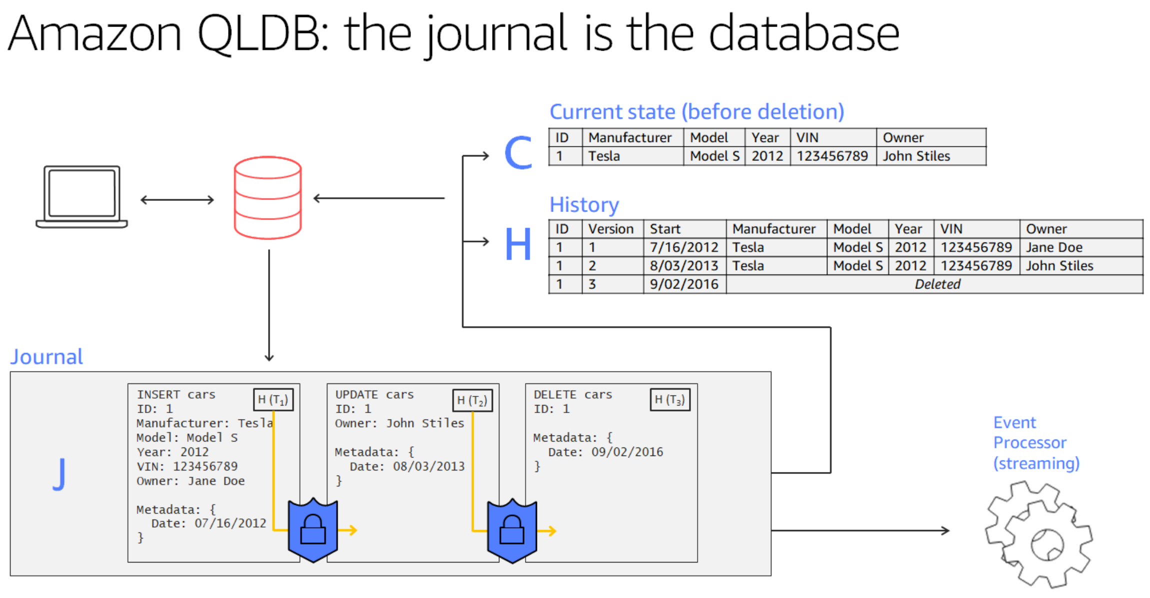
Journal first

In traditional database architecture, you generally write data in tables as part of a transaction. A transaction log—typically an internal implementation—records all of the transactions and the database modifications that they make. The transaction log is a critical component of the database. You need the log to replay transactions in the event of a system failure, disaster recovery, or data replication. However, database transaction logs are not immutable and are not designed to provide direct and easy access to users.

In Amazon QLDB, the journal is the core of the database. Structurally similar to a transaction log, the journal is an immutable, append-only data structure that stores your transaction data along with the associated metadata. All write transactions, including updates and deletes, are committed to the journal first.

QLDB uses the journal to determine the current state of your ledger data by materializing it into queryable, user-defined tables. These tables also provide an accessible history of all transaction data, including document revisions and metadata. In addition, the journal handles concurrency, sequencing, cryptographic verification, and availability of the ledger data.

The following diagram illustrates the QLDB journal architecture.


Amazon QLDB: the journal is the database
Current state (before deletion)
C
History
H
Journal
J
H (T1)
H (T2)
H (T3)
Event Processor (streaming)

ID	Manufacturer	Model	Year	VIN	Owner
1	Tesla	Model S	2012	123456789	John Stiles

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner
1	1	7/16/2012	Tesla	Model S	2012	123456789	Jane Doe
1	2	8/03/2013	Tesla	Model S	2012	123456789	John Stiles
1	3	9/02/2016	*Deleted*				

```
INSERT cars
ID: 1
Manufacturer: Tesla
Model: Model S
Year: 2012
VIN: 123456789
Owner: Jane Doe

Metadata: {
  Date: 07/16/2012
}
```

```
UPDATE cars
ID: 1
Owner: John Stiles

Metadata: {
  Date: 08/03/2013
}
```

```
DELETE cars
ID: 1

Metadata: {
  Date: 09/02/2016
}
```

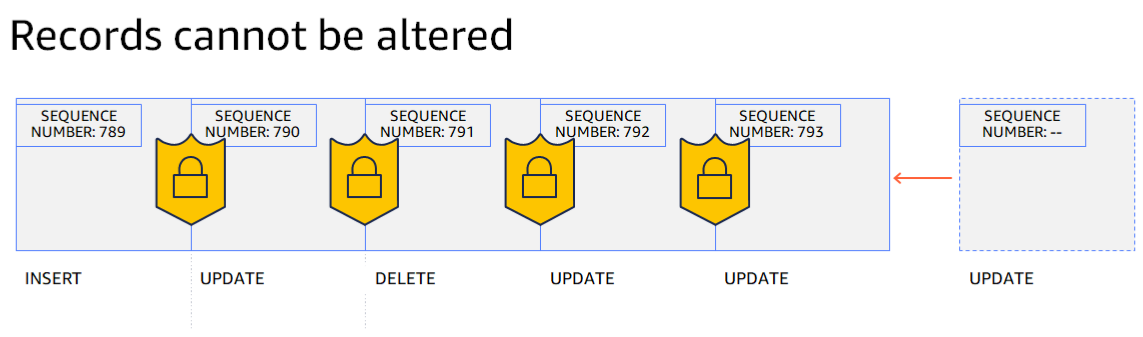
- In this example, an application connects to a ledger and executes transactions that insert, update, and delete a document into a table named `cars`.
- The data is first written to the journal in sequenced order.
- Then the data is materialized into the table with built-in views. These views let you query both the current state and the complete history of the car, with each revision assigned a version number.
- You can also export or stream data directly from the journal.

Immutable

Because the QLDB journal is append-only, it keeps a full record of all changes to your data that cannot be deleted, modified, or overwritten. There are no APIs or other methods to alter any committed data. This enables you to access and query the full history of your ledger.

QLDB writes one or more blocks to the journal in a transaction. Each block contains entry objects that represent the documents that you insert, update, and delete, along with the statements that you executed to commit them. These blocks are sequenced and hash-chained to guarantee data integrity.

The following diagram illustrates this journal structure.

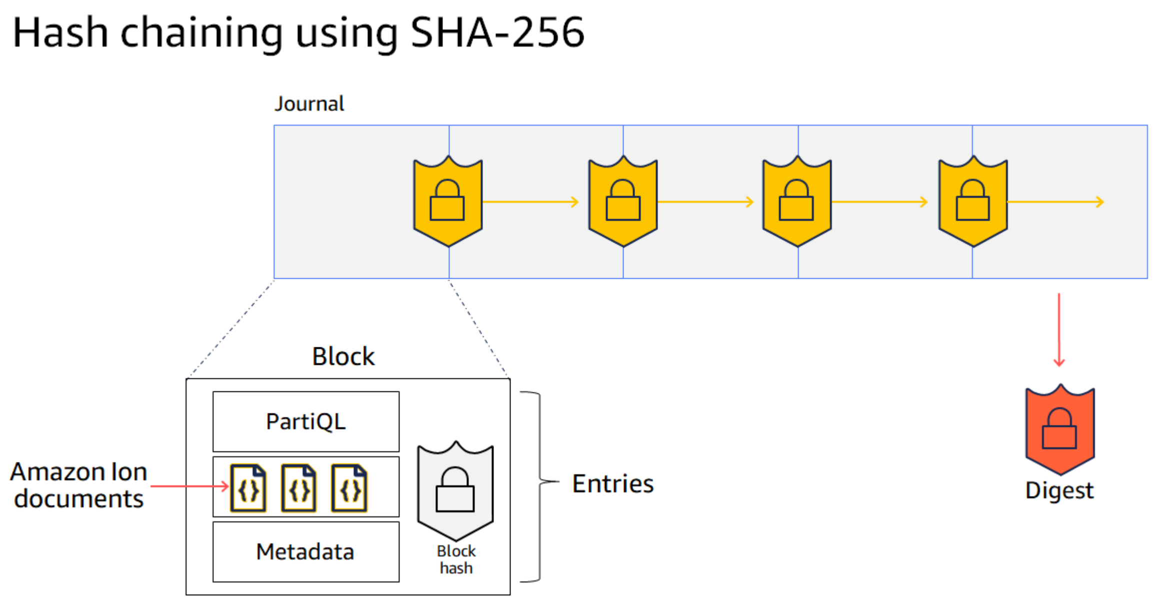


The diagram shows that transactions are committed to the journal as blocks that are hash-chained for verification. Each block has a sequence number to specify its address.

Cryptographically verifiable

Journal blocks are sequenced and chained together with cryptographic hashing techniques, similar to blockchains. This feature enables the journal to provide transactional data integrity using a cryptographic verification method. Using a *digest* (a hash value that represents a journal's full hash chain as of a point in time) and a *Merkle audit proof* (a mechanism that proves the validity of any node within a binary hash tree), you can verify that there have been no unintended changes to your data at any time.

The following diagram shows a digest that covers a journal's full hash chain at a point in time.



In this diagram, the journal blocks are hashed using the SHA-256 cryptographic hash function and are sequentially chained to subsequent blocks. Each block contains entries that include your data documents, metadata, and the PartiQL statements that were executed in the transaction.

For more information, see Data verification in Amazon QLDB (p. 338).

SQL compatible and document flexible

QLDB uses PartiQL as its query language and Amazon Ion as its document-oriented data model. PartiQL is an open source, SQL-compatible query language that has been extended to work with Ion. With PartiQL, you can insert, query, and manage your data with familiar SQL operators. When you're querying flat documents, the syntax is the same as using SQL to query relational tables. To learn more about PartiQL, see the Amazon QLDB PartiQL reference (p. 426).

Amazon Ion is a superset of JSON. Ion is an open source, document-based data format that gives you the flexibility of storing and processing structured, semistructured, and nested data. To learn more about Ion, see the Amazon Ion data format reference (p. 487).

For a high-level comparison of the core components and features in traditional relational databases versus QLDB, see From relational to ledger (p. 5).

Open source developer ecosystem

To simplify application development, QLDB provides open source drivers in various programming languages. You can use these drivers to interact with the transactional data API by running PartiQL statements on a ledger and processing the results of those statements. For information and tutorials about the driver languages that are currently supported, see Getting started with the Amazon QLDB driver (p. 40).

Amazon Ion also provides client libraries that process Ion data for you. For developer guides and code examples of processing Ion data, see the Amazon Ion documentation on GitHub.

Serverless and highly available

QLDB is serverless and highly available. The service automatically scales to support the demands of your application, and you don't need to provision instances or capacity. Multiple copies of your data are replicated within an Availability Zone and across Availability Zones in an AWS Region.

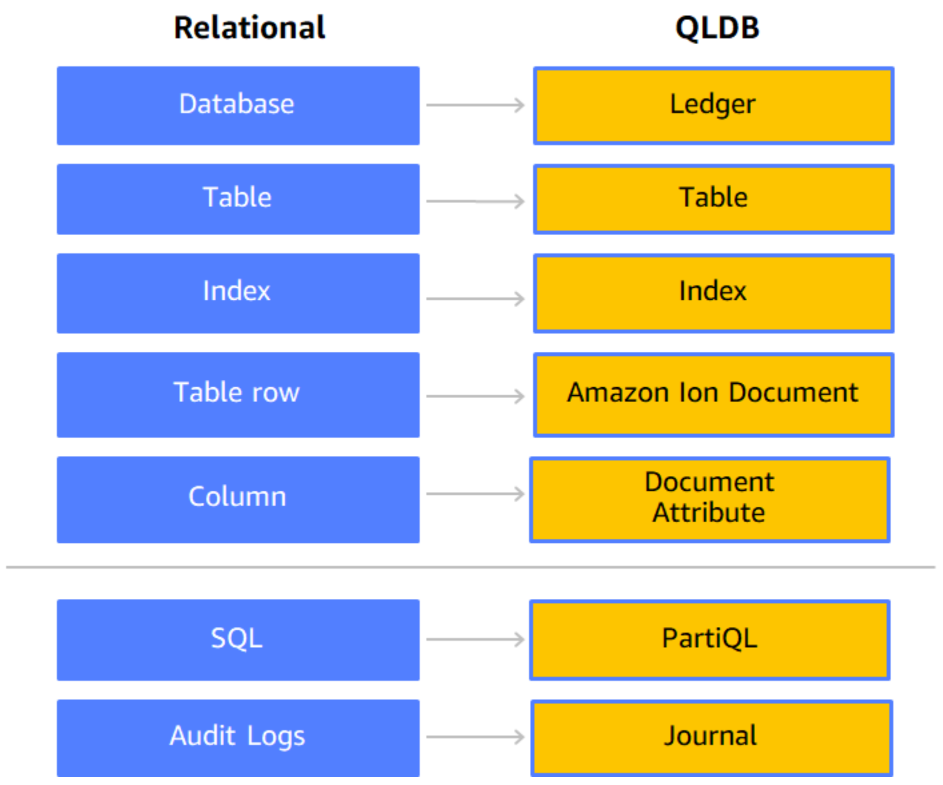
Enterprise grade

QLDB transactions are fully compliant with ACID (atomicity, consistency, isolation, and durability) properties. QLDB uses optimistic concurrency control (OCC), and transactions operate with full serializability—the highest level of isolation. This means that there's no risk of seeing phantom reads, dirty reads, write skew, or other similar concurrency issues. For more information, see Amazon QLDB concurrency model (p. 332).

From relational to ledger

If you are an application developer, you might have some experience using a relational database management system (RDBMS) and Structured Query Language (SQL). As you begin working with Amazon QLDB, you will encounter many similarities. As you progress to more advanced topics, you will also encounter powerful new features that QLDB has built on the RDBMS foundation. This section describes common database components and operations, comparing and contrasting them with their equivalents in QLDB.

The following diagram shows the mapping constructs of the core components between a traditional RDBMS and Amazon QLDB.



The following table shows the primary high-level similarities and differences of built-in operational features between a traditional RDBMS and QLDB.

Operation	RDBMS	QLDB
Creating tables	`CREATE TABLE` statement that defines all column names and data types	`CREATE TABLE` statement that doesn't define any table attributes or data types to allow schemaless and open content
Creating indexes	`CREATE INDEX` statement	`CREATE INDEX` statement that can be executed on empty tables and on a single field
Inserting data	`INSERT` statement that specifies values within a new row or tuple that adheres to the schema as defined by the table	`INSERT` statement that specifies values within a new document in any valid Amazon Ion format regardless of the existing documents in the table
Querying data	`SELECT-FROM-WHERE` statement	`SELECT-FROM-WHERE` statement in the same syntax as SQL when querying flat documents
Updating data	`UPDATE-SET-WHERE` statement	`UPDATE-SET-WHERE` statement in the same syntax as SQL when updating flat documents

Operation	RDBMS	QLDB
Deleting data	`DELETE-FROM-WHERE` statement	`DELETE-FROM-WHERE` statement in the same syntax as SQL when deleting flat documents
Nested and semistructured data	Flat rows or tuples only	Documents that can have any structured, semistructured, or nested data as supported by the Amazon Ion data format and the PartiQL query language
Querying metadata	No built-in metadata	`SELECT` statement that queries from the built-in committed view of a table
Querying revision history	No built-in data history	`SELECT` statement that queries from the built-in history function
Cryptographic verification	No built-in cryptography or immutability	APIs that return a digest of a journal and a proof that verifies the integrity of any document revision relative to that digest

For an overview of the core concepts and terminology in QLDB, see Core concepts (p. 7).

For detailed information about the process of creating, querying, and managing your data in a ledger, see Working with data and history (p. 317).

Core concepts and terminology in Amazon QLDB

This section provides an overview of the core concepts and terminology in Amazon QLDB, including ledger structure and how a ledger manages data. After you read this section, see Working with data and history (p. 317) and follow the examples that walk you through the process of creating tables, inserting data, and running basic queries.

As a ledger database, QLDB differs from other document-based databases when it comes to the following key concepts.

Topics

Ledger structure

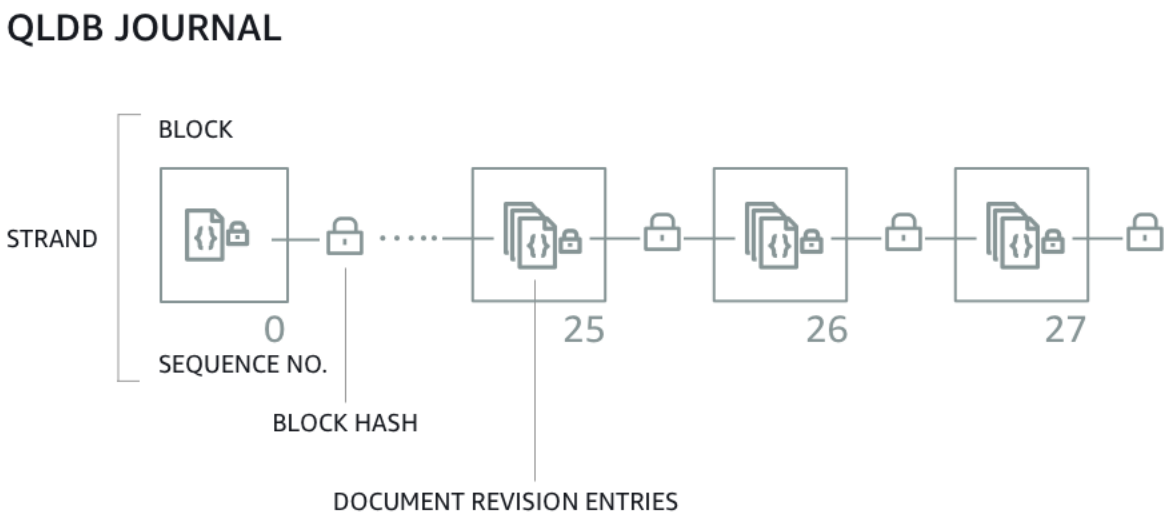
Fundamentally, QLDB data is organized into tables of Amazon Ion (p. 487) documents. More precisely, tables are collections of document revisions. A *document revision* represents a single iteration of the document's full dataset. Because QLDB stores the complete change history of your data, a table contains not only the latest revision of its documents, but also all prior iterations. For details on revisions, see Updating and deleting documents (p. 325) and Querying revision history (p. 326).

> **Note**
> For brevity, this guide often refers to *document revisions* as simply *documents*. This shorthand term keeps a consistent meaning because we typically think in terms of documents when inserting, updating, and deleting elements of a collection. However, when querying document history, it's important to specify that we are referring to *revisions*.

Write transactions

When an application needs to modify data in a document, it does so in a database transaction. Within a transaction, data is read from the ledger, updated, and committed (p. 332) to the journal. The *journal* represents a complete and immutable history of all the changes to your data. QLDB writes one or more chained *blocks* to the journal in a transaction. Each block contains *entry* objects that represent the document revisions that you insert, update, and delete, along with the PartiQL (p. 426) statements that committed them.

The following diagram illustrates this journal structure.



The diagram shows that transactions are committed to the journal as blocks that contain document revision entries. Each block is hashed and chained to subsequent blocks for verification (p. 338). Each block has a sequence number to specify its address within the strand.

> **Note**
> In Amazon QLDB, a strand is a partition of your ledger's journal. QLDB currently supports journals with a single strand only.

For information about the data contents in a block, see Journal contents in Amazon QLDB (p. 9).

Querying your data

QLDB is intended to address the needs of high-performance online transaction processing (OLTP) workloads. A ledger provides queryable views of your data based on the transaction information that is committed to the journal. Similar to views in relational databases, a *view* in QLDB is a projection of the data in a table. Views are maintained in real time, so that they're always available for applications to query.

You can query the following views using PartiQL `SELECT` statements:

- *User*—The latest non-deleted revision of your application-defined data only (that is, the current state of your data). This is the default view in QLDB.
- *Committed*—The latest non-deleted revision of both your data and the system-generated metadata. This is the full system-defined table that corresponds directly to your user table.

In addition to these views, you can query the revision history of your data by using the built-in History function (p. 326). The history function returns both your data and the associated metadata in the same schema as the *committed view*.

Data storage

There are two types of data storage in QLDB:

- *Journal storage*—The disk space that is used by a ledger's journal. The journal is append-only and contains the complete, immutable, and verifiable history of all the changes to your data.
- *Indexed storage*—The disk space that is used by a ledger's tables, indexes, and indexed history. Indexed storage consists of ledger data that is optimized for high-performance queries.

After your data is committed to the journal, it is materialized into the tables that you define. These tables enable faster and more efficient queries. When an application reads data, it accesses the tables and indexes that are stored in your indexed storage.

Next steps

To help you understand how to use a ledger with your data, see Working with data and history in Amazon QLDB (p. 317). This guide explains how these concepts work in QLDB, using sample data and query examples for context.

To get started quickly with a sample application tutorial using the QLDB console, see Getting started with the Amazon QLDB console (p. 25).

For a list of the key terms and definitions described in this section, see the Amazon QLDB glossary (p. 12).

Journal contents in Amazon QLDB

In Amazon QLDB, the *journal* is the immutable transactional log that stores the complete and verifiable history of all the changes to your data. The journal is append-only and is composed of a sequenced and hash-chained set of *blocks* that contain your committed data and other system metadata. QLDB writes one or more chained blocks to the journal in a transaction.

This section provides an example of a journal block with sample data and describes the contents of a block.

Topics

Block example

A journal block contains transaction metadata along with entries that represent the document revisions that were committed in the transaction and the PartiQL (p. 426) statements that committed them.

The following is an example of a block with sample data.

Note
This block example is for informational purposes only. The hashes shown are not real calculated hash values.

```
{
  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"
      }
    }
  ]
}
```

In the `revisions` field, some revision objects might only contain a `hash` value and no other attributes. These are internal-only system revisions that don't contain user data. The hashes of these revisions are part of the journal's full hash chain, which is required for cryptographic verification.

Block contents

A journal block has the following fields:

`blockAddress`

The location of the block in the journal. An address is an Amazon Ion (p. 487) structure that has two fields: `strandId` and `sequenceNo`.

For example: `{strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:14}`

`transactionId`

The unique ID of the transaction that committed the block.

`blockTimestamp`

The timestamp when the block was committed to the journal.

`blockHash`

The 256-bit hash value that uniquely represents the block. This is the hash of the concatenation of `entriesHash` and `previousBlockHash`.

`entriesHash`

The hash that represents all the entries within the block, including internal-only system entries. This is the root hash of the Merkle tree (p. 341) in which the leaf nodes consist of all the hashes in `entriesHashList`.

`previousBlockHash`

The hash of the previous chained block in the journal.

`entriesHashList`

The list of hashes that represent each entry within the block. This list includes the following entry hashes:

- A hash that represents `transactionInfo`.
- The root hash of the Merkle tree in which the leaf nodes consist of all the hashes in `revisions`.
- Hashes that represent internal-only system metadata. These hashes might not exist in all blocks.

`transactionInfo`

An Amazon Ion structure that contains information about the statements in the transaction that committed the block. This structure has the following fields:

- `statements` – The list of PartiQL statements and the `startTime` when they were executed. Each statement has a `statementDigest` hash, which is required to calculate the hash of the `transactionInfo` structure.
- `documents` – The document IDs that were updated by the statements. Each document includes the `tableName` and `tableId` that it belongs to, and the index of the statement that updated it.

`revisions`

The list of document revisions that were committed in the block. Each revision structure contains all the fields from the *committed view* of the revision.

This can also include hashes that represent internal-only system revisions that are part of the full hash chain of a journal.

Sample application

For a Java code example that exports journal data from a ledger and uses the exported data to validate the journal's hash chain, see ValidateQldbHashChain.java in the GitHub repository `aws-samples/amazon-qldb-dmv-sample-java`. For instructions on how to download and install this complete sample application, see Installing the Amazon QLDB Java sample application (p. 45).

The `ValidateQldbHashChain` class contains tutorial code that validates the hash chain between journal blocks and also within each block. Before you run the tutorial code, make sure that you follow Steps 1–3 in the Java tutorial (p. 44) to set up a sample ledger and load it with sample data.

See also

For more information about journals in QLDB, see the following topics:

- Exporting journal data from Amazon QLDB (p. 354) – To learn how to export journal data to Amazon Simple Storage Service (Amazon S3).
- Streaming journal data from Amazon QLDB (p. 366) – To learn how to stream journal data to Amazon Kinesis Data Streams.
- Data verification in Amazon QLDB (p. 338) – To learn about cryptographic verification of journal data.

Amazon QLDB glossary

The following are definitions for key terms that you might encounter as you work with Amazon QLDB.

block (p. 13) | digest (p. 13) | document (p. 13) | document ID (p. 13) | document revision (p. 13) | entry (p. 13) | field (p. 13) | index (p. 13) | indexed storage (p. 13) | journal (p. 13) | journal block (p. 14) | journal storage (p. 14) | journal strand (p. 14) | journal

tip (p. 14) | ledger (p. 14) | proof (p. 14) | revision (p. 14) | session (p. 14) | strand (p. 14) | table (p. 14) | view (p. 14)

block

An object that is committed to the journal in a transaction. A single transaction writes one or more blocks in the journal, but a block can only be associated with one transaction. A block contains entries that represent the document revisions that were committed in the transaction, along with the PartiQL (p. 426) statements that committed them.

Each block also has a hash value for verification. A block hash is calculated from the entry hashes within that block combined with the hash of the previous chained block.

digest

A 256-bit hash value that uniquely represents your ledger's entire history of document revisions as of a point in time. A digest hash is calculated from your journal's full hash chain as of the latest committed block in the journal at that time.

QLDB enables you to generate a digest as a secure output file. Then, you can use that output file to verify the integrity of your document revisions relative to that hash.

document

A set of data in Amazon Ion (p. 487) `struct` format that can be inserted, updated, and deleted in a table. A QLDB document can have structured, semistructured, nested, and schema-less data.

document ID

The universally unique identifier (UUID) that QLDB assigns to each document that you insert into a table. This ID is a 128-bit number that is represented in a Base62-encoded alphanumeric string with a fixed length of 22 characters.

document revision

A structure that represents a single iteration of a document's full dataset. A revision includes both your application-defined data and QLDB-generated metadata. Each revision is stored in a table and is uniquely identified by a combination of the document ID and a zero-based version number.

entry

An object that is contained in a block. Entries represent document revisions that are inserted, updated, and deleted in a transaction, along with the PartiQL statements that committed them.

Each entry also has a hash value for verification. An entry hash is calculated from the revision hashes or the statement hashes within that entry.

field

A name-value pair that makes up each attribute of a QLDB document. The name is a symbol token, and the value is unrestricted.

index

A data structure that you can create on a table to improve the speed of data retrieval operations, similar to that of relational databases. For information about indexes in QLDB, see CREATE INDEX (p. 448) in the *Amazon QLDB PartiQL reference*.

indexed storage

The disk space that is used by a ledger's tables, indexes, and indexed history. Indexed storage consists of ledger data that is optimized for high-performance queries.

journal

The hash-chained set of all blocks that are committed in your ledger. The journal is append-only and represents a complete and immutable history of all the changes to your ledger data.

journal block

See block (p. 13).

journal storage

The disk space that is used by a ledger's journal.

journal strand

See strand (p. 14).

journal tip

The latest committed block in a journal at a point in time.

ledger

An instance of an Amazon QLDB ledger database resource. A ledger consists of both *journal storage* and *indexed storage*. Ledger data is organized into tables of Amazon Ion document revisions.

proof

The ordered list of 256-bit hash values that QLDB returns for a given digest and document revision. It consists of the hashes that are required by a Merkle tree model to chain the given revision hash to the digest hash. A proof enables you to verify the integrity of your revisions relative to the digest. For more information, see Data verification in Amazon QLDB (p. 338).

revision

See document revision (p. 13).

session

An object that manages information about your data transaction requests and responses to and from a ledger. An *active session* (one that is actively executing a transaction) represents a single connection to a ledger. QLDB supports one actively executing transaction per session.

strand

A partition of a journal. QLDB currently supports journals with a single strand only.

table

An unordered collection of document revisions.

view

A queryable projection of the data in a table, based on transactions committed to the journal. In a PartiQL statement, a view is denoted with a prefix qualifier (starting with `_ql_`) for a table name.

You can query the following views using `SELECT` statements:

- *User*—The latest non-deleted revision of your application-defined data only (that is, the current state of your data). This is the default view in QLDB.
- *Committed*—The latest non-deleted revision of both your data and the system-generated metadata. This is the full system-defined table that corresponds directly to your user table. For example: `_ql_committed_`*`TableName`*.

Accessing Amazon QLDB

You can access Amazon QLDB using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the QLDB API. Before accessing QLDB, you must do the following:

1. Sign up for an AWS account. (p. 15)
2. Create an AWS Identity and Access Management (IAM) user. (p. 15)
3. Get an IAM access key (p. 17) (used to access QLDB programmatically).

   Note
   If you plan to interact with QLDB only through the console, you don't need an AWS access key, and you can skip ahead to Accessing Amazon QLDB using the console (p. 18).
4. Configure your credentials (p. 18) (used to access QLDB programmatically).

After completing these steps, see the following topics to learn more about how to access QLDB:

- Using the console (p. 18)
- Using the AWS CLI (control plane only) (p. 19)
- Using the Amazon QLDB shell (data plane only) (p. 20)
- Using the API (p. 23)

Signing up for an AWS account

To use the QLDB service, you must have an AWS account. If you don't already have an account, you are prompted to create one when you sign up. You're not charged for any AWS services that you sign up for unless you use them.

To sign up for AWS

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Creating an IAM user

If your account already includes an IAM user with full AWS administrative permissions, you can skip this section.

When you first create an Amazon Web Services (AWS) account, you begin with a single sign-in identity. That identity has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user*. When you sign in, enter the email address and password that you used to create the account.

> **Important**
> We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks. To view the tasks that require you to sign in as the root user, see AWS Tasks That Require Root User.

To create an administrator user for yourself and add the user to an administrators group (console)

1. Sign in to the IAM console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   > **Note**
   > We strongly recommend that you adhere to the best practice of using the **`Administrator`** IAM user below and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.
2. In the navigation pane, choose **Users** and then choose **Add user**.
3. For **User name**, enter **`Administrator`**.
4. Select the check box next to **AWS Management Console access**. Then select **Custom password**, and then enter your new password in the text box.
5. (Optional) By default, AWS requires the new user to create a new password when first signing in. You can clear the check box next to **User must create a new password at next sign-in** to allow the new user to reset their password after they sign in.
6. Choose **Next: Permissions**.
7. Under **Set permissions**, choose **Add user to group**.
8. Choose **Create group**.
9. In the **Create group** dialog box, for **Group name** enter **`Administrators`**.
10. Choose **Filter policies**, and then select **AWS managed -job function** to filter the table contents.
11. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.

    > **Note**
    > You must activate IAM user and role access to Billing before you can use the `AdministratorAccess` permissions to access the AWS Billing and Cost Management console. To do this, follow the instructions in step 1 of the tutorial about delegating access to the billing console.
12. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
13. Choose **Next: Tags**.
14. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM Entities in the *IAM User Guide*.
15. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

You can use this same process to create more groups and users and to give your users access to your AWS account resources. To learn about using policies that restrict user permissions to specific AWS resources, see Access Management and Example Policies.

For information about using IAM to manage access to QLDB, see How Amazon QLDB works with IAM (p. 395).

Signing in as an IAM user

Sign in to the IAM console by choosing **IAM user** and entering your AWS account ID or account alias. On the next page, enter your IAM user name and your password.

> **Note**
> For your convenience, the AWS sign-in page uses a browser cookie to remember your IAM user name and account information. If you previously signed in as a different user, choose the sign-in link beneath the button to return to the main sign-in page. From there, you can enter your AWS account ID or account alias to be redirected to the IAM user sign-in page for your account.

Getting IAM user access keys

Before you can access Amazon QLDB programmatically or through the AWS CLI, you must have an AWS access key. You don't need an access key if you plan to use the QLDB console only.

Access keys consist of an access key ID and secret access key, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them from the AWS Management Console. As a best practice, do not use the AWS account root user access keys for any task where it's not required. Instead, create a new administrator IAM user with access keys for yourself.

The only time that you can view or download the secret access key is when you create the keys. You cannot recover them later. However, you can create new access keys at any time. You must also have permissions to perform the required IAM actions. For more information, see Permissions Required to Access IAM Resources in the *IAM User Guide*.

To create access keys for an IAM user

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. Choose the name of the user whose access keys you want to create, and then choose the **Security credentials** tab.
4. In the **Access keys** section, choose **Create access key**.
5. To view the new access key pair, choose **Show**. You will not have access to the secret access key again after this dialog box closes. Your credentials will look something like this:
   - Access key ID: AKIAIOSFODNN7EXAMPLE
   - Secret access key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
6. To download the key pair, choose **Download .csv file**. Store the keys in a secure location. You will not have access to the secret access key again after this dialog box closes.

   Keep the keys confidential in order to protect your AWS account and never email them. Do not share them outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.
7. After you download the `.csv` file, choose **Close**. When you create an access key, the key pair is active by default, and you can use the pair right away.

Related topics

- What Is IAM? in the *IAM User Guide*
- AWS Security Credentials in *AWS General Reference*

Configuring your credentials

Before you can access Amazon QLDB programmatically or through the AWS CLI, you must configure your credentials to enable authorization for your applications.

There are several ways to do this, including the following methods:

- Manually create a credentials file to store your AWS access key ID and secret access key.
- Use the `aws configure` command of the AWS CLI to automatically create the file.

  To install and configure the AWS CLI, see Accessing Amazon QLDB using the AWS CLI (control plane only) (p. 19).
- Use environment variables.

For instructions on configuring your credentials, see Getting started with the Amazon QLDB driver (p. 40). Each programming language has specific prerequisites with links to instructions in their respective AWS SDK developer guide.

Accessing Amazon QLDB using the console

You can access the AWS Management Console for Amazon QLDB here: https://console.aws.amazon.com/qldb

You can use the console to do the following in QLDB:

- Create, delete, describe, and list ledgers.
- Run PartiQL (p. 426) statements by using the *Query editor*.
- Manage tags for QLDB resources.
- Verify data cryptographically.
- Export or stream journal blocks.

To learn how to create an Amazon QLDB ledger and set it up with sample application data, see Getting started with the Amazon QLDB console (p. 25).

Query editor keyboard shortcuts

The *Query editor* on the QLDB console supports the following keyboard shortcuts.

Action	macOS	Windows
Run	**Cmd+Return**	**Ctrl+Enter**
Save	**Cmd+S**	**Ctrl+S**
Clear	**Cmd+Shift+Delete**	**Ctrl+Shift+Backspace**

Accessing Amazon QLDB using the AWS CLI (control plane only)

You can use the AWS Command Line Interface (AWS CLI) to control multiple AWS services from the command line and automate them through scripts. You can use the AWS CLI for ad hoc operations. You can also use it to embed Amazon QLDB operations within utility scripts.

Before you can use the AWS CLI with QLDB, you must get an access key ID and secret access key. For more information, see Getting IAM user access keys (p. 17).

For a complete listing of all the commands available for QLDB in the AWS CLI, see the AWS CLI Command Reference.

Note
The AWS CLI only supports the `qldb` control plane API operations that are listed in the Amazon QLDB API reference (p. 500). These actions are used only for managing ledgers and for non-transactional data operations.
To execute data transactions with the `qldb-session` API using a command line interface, see Accessing Amazon QLDB using the QLDB shell (data plane only) (p. 20).

Topics

Downloading and configuring the AWS CLI

The AWS CLI runs on Windows, macOS, or Linux. To download, install, and configure it, follow these steps:

1. Download the AWS CLI at http://aws.amazon.com/cli.
2. Follow the instructions for Installing the AWS CLI and Configuring the AWS CLI in the *AWS Command Line Interface User Guide*.

Using the AWS CLI with QLDB

The command line format consists of an Amazon QLDB operation name, followed by the parameters for that operation. The AWS CLI supports a shorthand syntax for the parameter values, in addition to JSON.

Use `help` to list all available commands in QLDB:

```
aws qldb help
```

You can also use `help` to describe a specific command and learn more about its usage:

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

For example, to create a ledger:

```
aws qldb create-ledger --name my-ledger --permissions-mode ALLOW_ALL
```

Accessing Amazon QLDB using the QLDB shell (data plane only)

Amazon QLDB provides a command line shell for interaction with the transactional data plane. The QLDB shell enables you to execute PartiQL (p. 426) statements on ledger data. This shell is written in Python and is open-sourced in the GitHub repository awslabs/amazon-qldb-shell.

Note
The Amazon QLDB shell only supports the `qldb-session` transactional data API. This API is used only for executing PartiQL statements on a QLDB ledger.
To interact with the `qldb` control plane API actions using a command line interface, see Accessing Amazon QLDB using the AWS CLI (control plane only) (p. 19).

This tool is not intended to be incorporated into an application or adopted for production purposes. The objective of this tool is to give developers, DevOps, database administrators, and others who are interested the opportunity to rapidly experiment with QLDB and PartiQL.

The following sections describe how to get started with the QLDB shell.

Topics

Prerequisites

Before you get started with the QLDB shell, you must do the following:

1. Follow the AWS setup instructions in Accessing Amazon QLDB (p. 15). This includes signing up for AWS and getting an AWS access key for development.
2. Install Python version 3.4 or later from the Python downloads site.
3. Set up your AWS credentials and your default AWS Region. For instructions, see Quickstart in the AWS SDK for Python (Boto3) documentation.

   For a complete list of available Regions, see Amazon QLDB endpoints and quotas in the *AWS General Reference*.

Setting up the shell

To install the QLDB shell from PyPI using `pip3` (a package manager for Python 3), run the following at your command line terminal.

```
$ pip3 install qldbshell
```

Installing the shell also installs the following required package dependencies:

- `pyqldb` – Requires version `2.0.2` or earlier.
- `amazon.ion`
- `argparse`
- `boto3`

The shell is not compatible with the latest version (3.x) of the QLDB driver for Python (`pyqldb`). If you have the latest version installed, see Resolving the driver dependency (p. 23) for additional setup instructions.

Invoking the shell

To invoke the QLDB shell on your command line terminal for a specific ledger, run the following command. Replace *`test-ledger`* with your ledger name.

```
$ qldbshell --ledger test-ledger
```

This command connects to your default AWS Region. To explicitly specify the Region, you can run the command with the `--region` parameter, as described in the following section.

Each successive line that you enter in your `qldbshell` session is treated as a separate PartiQL statement. In the current version of the QLDB shell, a PartiQL statement cannot span more than one line.

Connection parameters

To see a list of available input arguments, run the following command before you invoke a `qldbshell` session.

```
$ qldbshell --help
```

The following connection parameters are available for the `qldbshell` command. You can add the optional arguments to override the AWS Region, credentials profile, and endpoint that the shell uses.

Usage syntax:

```
$ qldbshell -l LEDGER_NAME [-v] [-p PROFILE] [-r REGION_CODE] [-s QLDB_SESSION_ENDPOINT]
```

Required arguments

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

Specifies the name of the ledger that the shell connects to. The ledger name must already exist and must be active.

Optional arguments

`-v`, `--verbose`

Increases output verbosity in your shell session.

`-p` *`PROFILE`*, `--profile` *`PROFILE`*

Specifies the location of your AWS credentials profile that the shell uses for authentication.

If not provided, the shell uses your default AWS profile, which is located at `~/.aws/credentials`.

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

Specifies the AWS Region code of the QLDB ledger that the shell connects to. For example: `us-east-1`.

If not provided, the shell connects to your default AWS Region as specified in your AWS profile.

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

Specifies the `qldb-session` API endpoint that the shell connects to.

For a complete list of available QLDB Regions and endpoints, see Amazon QLDB endpoints and quotas in the *AWS General Reference*.

Parameter file

As an alternative to passing arguments on the command line, you can also save input parameters in a file. Put each parameter on a separate line. Then, pass the file name to the `qldbshell` as follows.

```
$ qldbshell @params.conf
```

Exiting the shell

To exit the current `qldbshell` session and close the current ledger database connection, run the `exit` command.

```
qldbshell > exit
```

Example

For information about writing PartiQL statements in QLDB, see the Amazon QLDB PartiQL reference (p. 426).

Example

The following example shows a common sequence of basic commands.

Note
The QLDB shell executes each PartiQL statement in this example in its own transaction.
This example assumes that the ledger `test-ledger` already exists and is active.

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

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

Common errors

This section provides instructions to resolve common errors that you might encounter while using the QLDB shell.

Resolving the driver dependency

The QLDB shell requires version `2.0.2` or earlier of the QLDB driver for Python. If you have an incompatible version of the driver installed, you might see one of the following error messages.

```
ModuleNotFoundError: No module named 'pyqldb.driver.pooled_qldb_driver'
```

```
ERROR: pyqldb 3.0.0rc1 has requirement amazon.ion<0.6,>=0.5.0, but you'll have amazon-ion
 0.6.0 which is incompatible.
```

To resolve this driver dependency issue, you can downgrade to an earlier version. Or, you can run the shell in a virtual environment, which installs an older version of the driver.

To downgrade the driver to version 2.x

1. Uninstall the current version of the driver.

   ```
   $ pip3 uninstall pyqldb
   ```

2. Install version `2.0.2` of the driver.

   ```
   $ pip3 install pyqldb==2.0.2
   ```

To run the shell in a virtual environment

1. Install `virtualenv`.

   ```
   $ pip3 install virtualenv
   ```

2. Create your virtual environment. You can replace *`venv`* with your own environment name.

   ```
   $ virtualenv venv
   ```

3. Activate your virtual environment. Replace *`venv`* with the environment name that you created in the previous step.

   ```
   $ source venv/bin/activate
   ```

4. Install the QLDB shell.

   ```
   $ pip3 install qldbshell
   ```

5. Proceed to Invoking the shell (p. 21) to run the shell in your virtual environment.
6. After you finish using the shell, you can deactivate the virtual environment.

   ```
   $ deactivate
   ```

Accessing Amazon QLDB using the API

You can use the AWS Management Console and the AWS Command Line Interface (AWS CLI) to work interactively with Amazon QLDB. However, to get the most out of QLDB, you can write application code with a QLDB driver and its underlying AWS SDK to interact with your ledger using the APIs.

The QLDB driver provides support for QLDB in Java, .NET, Go, Node.js, and Python. To get started quickly with these languages, see Getting started with the Amazon QLDB driver (p. 40).

Before you can use the QLDB driver in your application, you must get an AWS access key ID and secret access key. For more information, see Getting IAM user access keys (p. 17).

Getting started with the Amazon QLDB console

This tutorial guides you through steps to create your first Amazon QLDB ledger and populate it with tables and sample data. The sample ledger you create in this scenario is a database for a department of motor vehicles (DMV) application that tracks the complete historical information about vehicle registrations.

The history of an asset is a common use case for QLDB because it involves a variety of scenarios and operations that highlight the usefulness of a ledger database. QLDB enables you to track and verify the full history of changes in an easily accessible and flexible database.

As you work through this tutorial, the following sections explain how to add vehicle registrations, modify them, and view the history of changes to those registrations. This guide also shows you how to cryptographically verify a registration document, and concludes by cleaning up resources and deleting the sample ledger.

Each step in the tutorial has instructions for using the AWS Management Console.

Topics

Tutorial prerequisites

Before you start this tutorial, you must follow the AWS setup instructions in Accessing Amazon QLDB (p. 15). These steps include signing up for AWS and creating an AWS Identity and Access Management (IAM) user with QLDB access.

Step 1: Create a new ledger

In this step, you create a new Amazon QLDB ledger named `vehicle-registration`. Then, you confirm that the status of the ledger is **Active**. You can also verify any tags that you added to the ledger.

When you create a ledger, *deletion protection* is enabled by default. Deletion protection is a feature in QLDB that prevents ledgers from being deleted by any user. You can disable deletion protection when you create a ledger using the QLDB API or the AWS Command Line Interface (AWS CLI).

To create a new ledger

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Getting started**.
3. On the **Create your first ledger** card, choose **Create Ledger**.
4. On the **Create Ledger** page, do the following:
   - **Ledger information**—The **Ledger name** should be pre-populated with **`vehicle-registration`**.
   - **Tags**—(Optional) Add metadata to the ledger by attaching tags as key-value pairs. You can add tags to your ledger to help organize and identify them. For more information, see Tagging Amazon QLDB resources (p. 387).

     Choose **Add tag**, and then enter any key-value pairs as appropriate.
5. When the settings are as you want them, choose **Create ledger**.

   Note
   You can access your QLDB ledger when its status becomes **Active**. This can take several minutes.
6. In the list of **Ledgers**, locate `vehicle-registration` and confirm that the ledger's status is **Active**.
7. (Optional) Choose the `vehicle-registration` ledger name. On the **vehicle-registration** ledger details page, confirm that any tags that you added to the ledger appear on the **Tags** card. You can also edit your ledger tags using this console page.

To create tables in the `vehicle-registration` ledger, proceed to Step 2: Create tables, indexes, and sample data (p. 26).

Step 2: Create tables, indexes, and sample data

When your Amazon QLDB ledger is active, you can start creating tables for data about vehicles, their owners, and their registration information. After creating the tables and indexes, you can load them with data.

In this step, you create four tables in the `vehicle-registration` ledger:

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

You also create the following indexes.

Table name	**Field**
`VehicleRegistration`	`VIN`
`VehicleRegistration`	`LicensePlateNumber`
`Vehicle`	`VIN`
`Person`	`GovId`

Table name	Field
`DriversLicense`	`LicenseNumber`
`DriversLicense`	`PersonId`

You can use the QLDB console to automatically create these tables with indexes and load them with sample data. Or, you can use the **Query editor** on the console to manually run each PartiQL (p. 426) statement step-by-step.

Automatic option

To create tables, indexes, and sample data

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Getting started**.
3. Under **Automatic option** on the **Sample application data** card, choose `vehicle-registration` in the list of ledgers.
4. Choose **Load sample data**.

   If the operation finishes successfully, the console displays the message **Sample data loaded**.

   This script runs all statements in a single transaction. If any part of the transaction fails, every statement is rolled back, and an appropriate error message is displayed. You can retry the operation after addressing any issues.

   Note

   - One possible cause for a transaction failure is attempting to create duplicate tables. Your request to load sample data will fail if any of the following table names already exist in your ledger: `VehicleRegistration`, `Vehicle`, `Person`, and `DriversLicense`.

     Instead, try loading this sample data in an empty ledger.
   - This script runs parameterized `INSERT` statements. So, these PartiQL statements are recorded in your journal blocks with bind parameters instead of the literal data. For example, you might see the following statement in a journal block, where the question mark (`?`) is a variable placeholder for the document contents.

   ```
   INSERT INTO Vehicle ?
   ```

Manual option

You insert documents into `VehicleRegistration` with an empty `PrimaryOwner` field, and into `DriversLicense` with an empty `PersonId` field. Later, you populate these fields with the system-assigned document `id` from the `Person` table.

Tip
As a best practice, use this document `id` metadata field as a foreign key. For more information, see Querying document metadata (p. 323).

To create tables, indexes, and sample data

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Query editor**.
3. Choose the `vehicle-registration` ledger.

4. Start by creating four tables. QLDB supports open content and does not enforce schema, so you don't specify attributes or data types.

   In the query editor window, enter the following statement, and then choose **Run**. To run the statement, you can also use the keyboard shortcut **Ctrl**+**Enter** for Windows, or **Cmd**+**Return** for macOS. For more keyboard shortcuts, see Accessing Amazon QLDB using the console (p. 18).

   ```
   CREATE TABLE VehicleRegistration
   ```

   Repeat this step for each of the following.

   ```
   CREATE TABLE Vehicle
   ```

   ```
   CREATE TABLE Person
   ```

   ```
   CREATE TABLE DriversLicense
   ```

5. Next, create indexes that help speed up queries against each table.

   In the query editor window, enter the following statement, and then choose **Run**.

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

   Repeat this step for the following.

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

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

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

   ```
   CREATE INDEX ON DriversLicense (LicenseNumber)
   ```

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

6. After creating your indexes, you can start loading data into your tables. In this step, insert documents into the `Person` table with personal information about owners of the vehicles that the ledger is tracking.

   In the query editor window, enter the following statement, and then choose **Run**.

   ```
   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. Then, populate the `DriversLicense` table with documents that include driver's license information for each vehicle owner.

   In the query editor window, enter the following statement, and then choose **Run**.

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

8. Now, populate the `VehicleRegistration` table with vehicle registration documents. These documents include a nested `Owners` structure that stores the primary and secondary owners.

   In the query editor window, enter the following statement, and then choose **Run**.

```
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. Lastly, populate the `Vehicle` table with documents describing the vehicles that are registered in your ledger.

   In the query editor window, enter the following statement, and then choose **Run**.

```
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'
} >>
```

Next, you can use `SELECT` statements to read data from the tables in the `vehicle-registration` ledger. Proceed to Step 3: Query the tables in a ledger (p. 32).

Step 3: Query the tables in a ledger

After creating tables in an Amazon QLDB ledger and loading them with data, you can run queries to review the vehicle registration data that you just inserted. QLDB uses PartiQL as its query language and Amazon Ion as its document-oriented data model.

PartiQL is an open source, SQL-compatible query language that has been extended to work with Ion. With PartiQL, you can insert, query, and manage your data with familiar SQL operators. Amazon Ion is a superset of JSON. Ion is an open source, document-based data format that gives you the flexibility of storing and processing structured, semistructured, and nested data.

In this step, you use `SELECT` statements to read data from the tables in the `vehicle-registration` ledger.

To query the tables

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Query editor**.
3. Choose the `vehicle-registration` ledger.
4. In the query editor window, enter the following statement to query the `Vehicle` table for a particular vehicle identification number (VIN) that you added to the ledger, and then choose **Run**.

   To run the statement, you can also use the keyboard shortcut **Ctrl**+**Enter** for Windows, or **Cmd**+**Return** for macOS. For more keyboard shortcuts, see Accessing Amazon QLDB using the console (p. 18).

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

5. You can write inner join queries. This query example joins `Vehicle` with `VehicleRegistration` and returns registration information along with attributes of the registered vehicle for a specified `VIN`.

   Enter the following statement, and then choose **Run**.

```
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
```

   You can also join the `Person` and `DriversLicense` tables to see attributes related to the drivers who were added to the ledger.

   Repeat this step for the following.

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

To learn about modifying documents in the tables in the `vehicle-registration` ledger, see Step 4: Modify documents in a ledger (p. 33).

Step 4: Modify documents in a ledger

Now that you have data to work with, you can start making changes to documents in the `vehicle-registration` ledger in Amazon QLDB. For example, consider the Audi A5 with VIN `1N4AL11D75C109151`. This car is initially owned by a driver named Raul Lewis in Seattle, WA.

Suppose that Raul sells the car to a resident in Everett, WA named Brent Logan. Then, Brent and Alexis Pena decide to get married. Brent wants to add Alexis as a secondary owner on the registration. In this step, the following data manipulation language (DML) statements demonstrate how to make the appropriate changes in your ledger to reflect these events.

Tip
As a best practice, use a document's system-assigned `id` as a foreign key. While you can define fields that are intended to be unique identifiers (for example, a vehicle's VIN), the true unique identifier of a document is its `id`. This field is included in the document's metadata, which you can query in the *committed view* (the system-defined view of a table).
For more information about views in QLDB, see Core concepts (p. 7). To learn more about metadata, see Querying document metadata (p. 323).

To modify documents

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Query editor**.
3. Choose the `vehicle-registration` ledger.

   Note
   If you set up your ledger using the console's automatic **Load sample data** feature, skip ahead to *Step 6*.
4. If you manually ran `INSERT` statements to load the sample data, continue with these steps.

   To initially register Raul as this vehicle's owner, start by finding his system-assigned document `id` in the `Person` table. This field is included in the document's metadata, which you can query in the system-defined view of the table, called the *committed view*.

   In the query editor window, enter the following statement, and then choose **Run**.

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

   The prefix `_ql_committed_` is a reserved prefix signifying that you want to query the committed view of the `Person` table. In this view, your data is nested in the `data` field, and metadata is nested in the `metadata` field.
5. Now, use this `id` in an `UPDATE` statement to modify the appropriate document in the `VehicleRegistration` table. Enter the following statement, and then choose **Run**.

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

   Confirm that you modified the `Owners` field by issuing this statement.

   ```
   SELECT r.Owners FROM VehicleRegistration AS r
   WHERE r.VIN = '1N4AL11D75C109151'
   ```
6. To transfer the vehicle's ownership to Brent in the city of Everett, first find his `id` from the `Person` table with the following statement.

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

Next, use this `id` to update the `PrimaryOwner` and the `City` in the `VehicleRegistration` table.

```
UPDATE VehicleRegistration AS r
SET r.Owners.PrimaryOwner.PersonId = 'IN7MvYtUjkp1GMZu0F6CG9', --replace with your id
r.City = 'Everett'
WHERE r.VIN = '1N4AL11D75C109151'
```

Confirm that you modified the `PrimaryOwner` and `City` fields by issuing this statement.

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

7. To add Alexis as a secondary owner of the car, find her `Person id`.

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

Then, insert this `id` into the `SecondaryOwners` list with the following FROM-INSERT (p. 452) DML statement.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners
    VALUE { 'PersonId' : '5UfgdlnjO6gF5CWcOIu64s' } --replace with your id
```

Confirm that you modified `SecondaryOwners` by issuing this statement.

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

To review these changes in the `vehicle-registration` ledger, see Step 5: View the revision history for a document (p. 34).

Step 5: View the revision history for a document

After modifying registration data for the car with VIN `1N4AL11D75C109151`, you can query the history of all its registered owners and any other updated fields. You can see all revisions of a document that you inserted, updated, and deleted by querying the built-in History function (p. 326).

The history function returns revisions from the *committed view* of your table, which includes both your application data and the associated metadata. The metadata shows exactly when each revision was made, in what order, and which transaction committed them.

In this step, you query the revision history of a document in the `VehicleRegistration` table in the `vehicle-registration` ledger.

To view the revision history

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.

2. In the navigation pane, choose **Query editor**.
3. Choose the vehicle-registration ledger.
4. To query the history of a document, start by finding its unique id. In addition to querying the committed view, another way of getting a document id is to use the BY keyword in the table's default user view. To learn more, see Using the BY clause to query document ID (p. 325).

   In the query editor window, enter the following statement, and then choose **Run**.

   ```
   SELECT r_id FROM VehicleRegistration AS r BY r_id
   WHERE r.VIN = '1N4AL11D75C109151'
   ```
5. Next, you can use this id value to query the history function in the following syntax.

   ```
   SELECT * FROM history( table [, `start-time`, `end-time`] ) AS h
   WHERE h.metadata.id = 'id'
   ```

   - Best practice is to qualify a history query with a document id. This helps to avoid inefficient queries.
   - The start-time and end-time are optional parameters that are both inclusive. They must be in ISO 8601 date and time format and in Universal Coordinated Time (UTC).
   - The start-time and end-time are Amazon Ion literal values that can be denoted with backticks (`...`). To learn more, see Querying Ion with PartiQL (p. 444).

   The history function returns documents with the same schema as the committed view. For example, enter the following statement, and then choose **Run**. Be sure to replace the id value with your own document ID as appropriate.

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

   This first history query projects your modified vehicle registration data. The output includes all revisions of the document and should look similar to the following.

VIN	City	Owners
"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
   The history query might not always return document revisions in sequential order.

Review the output and confirm that the changes reflect what you did in Step 4: Modify documents in a ledger (p. 33).

6. Then, you can inspect the document metadata of each revision. Enter the following statement, and then choose **Run**. Again, be sure to replace the `id` value with your own document ID as appropriate.

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

The output should look similar to the following.

version	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"

These metadata fields provide details on when each item was modified, and by which transaction. From this data, you can infer the following:

- The document is uniquely identified by its system-assigned `id`: `ADR2LQq48kB9neZDupQrMm`. This is a universally unique identifier (UUID) that is represented in a Base62-encoded string.
- The `txTime` shows that the initial revision of the document (version `0`) was created at `2019-05-23T19:20:360d-3Z`.
- Each subsequent transaction creates a new revision with the same document `id`, an incremented version number, and an updated `txId` and `txTime`.

To cryptographically verify a document revision in the `vehicle-registration` ledger, proceed to Step 6: Verify a document in a ledger (p. 36).

Step 6: Verify a document in a ledger

With Amazon QLDB, you can efficiently verify the integrity of a document in your ledger's journal by using cryptographic hashing with SHA-256. In this example, Alexis and Brent decide to upgrade to a new model by trading in the vehicle with VIN `1N4AL11D75C109151` at a car dealership. The dealership starts the process by verifying the vehicle's ownership with the registration office.

To learn more about how verification and cryptographic hashing work in QLDB, see Data verification in Amazon QLDB (p. 338).

In this step, you verify a document revision in the `vehicle-registration` ledger. First, you request a digest, which is returned as an output file and acts as a signature of your ledger's entire change history. Then, you request a proof for the revision relative to that digest. Using this proof, the integrity of your revision is verified if all validation checks pass.

To request a digest

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.

2. In the navigation pane, choose **Ledgers**.
3. In the list of ledgers, select `vehicle-registration`.
4. Choose **Get digest**. The **Get digest** dialog box displays the following digest details:

   - **Digest**—The SHA-256 hash value of the digest that you requested.
   - **Digest tip address**—The latest block (p. 338) location in the journal covered by the digest that you requested. An address has the following two fields:
     - `strandId`—The unique ID of the journal strand that contains the block.
     - `sequenceNo`—The index number that specifies the location of the block within the strand.
   - **Ledger**—The ledger name for which you requested a digest.
   - **Date**—The timestamp when you requested the digest.
5. Review the digest information. Then choose **Save**. You can keep the default file name, or enter a new name.

   This step saves a plaintext file with contents in Amazon Ion (p. 487) format. The file has a file name extension of `.ion.txt` and contains all the digest information that was listed on the preceding dialog box. The following is an example of a digest file's contents. The order of the fields can vary depending on your browser.

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

6. Save this file where you can access it later. In the following steps, you use this file as a fingerprint to verify a document revision against.

After you have a ledger digest saved, you can start the process of verifying a document revision against that digest.

Note
In a real scenario for verification, you use a digest that was previously saved rather than doing the two tasks consecutively. As a best practice, the digest is saved as soon as the document revision that you want to verify is written to the journal.

To verify a document revision

1. First, query your ledger for the `id` and `blockAddress` of the document revision that you want to verify. These fields are included in the document's metadata, which you can query in the committed view.

   The document `id` is a system-assigned unique ID string. The `blockAddress` is an Ion structure that specifies the block location where the revision was committed.

   In the navigation pane of the QLDB console, choose **Query editor**.
2. Choose the `vehicle-registration` ledger.
3. In the query editor window, enter the following statement, and then choose **Run**.

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

4. Copy and save the `id` and `blockAddress` values that your query returns. Be sure to omit the double quotes for the `id` field. In Amazon Ion, string data types are delimited with double quotes.
5. Now that you have a document revision selected, you can start the process of verifying it.

   In the navigation pane, choose **Verification**.
6. On the **Verify document** form, under **Specify the document that you want to verify**, enter the following input parameters:
   - **Ledger**—Choose `vehicle-registration`.
   - **Block address**—The `blockAddress` value returned by your query in *step 3*.
   - **Document ID**—The `id` value returned by your query in *step 3*.
7. Under **Specify the digest to use for verification**, select the digest that you previously saved by choosing **Choose digest**. If the file is valid, this auto-populates all the digest fields on your console. Or, you can manually copy and paste the following values directly from your digest file:
   - **Digest**—The `digest` value from your digest file.
   - **Digest tip address**—The `digestTipAddress` value from your digest file.
8. Review your document and digest input parameters, and then choose **Verify**.

   The console automates two steps for you:

   a. Request a proof from QLDB for your specified document.
   b. Use the proof returned by QLDB to call a client-side API, which verifies your document revision against the provided digest.

   The console displays the results of your request in the **Verification results** card. For more information, see Verification results (p. 346).
9. To test the verification logic, repeat steps 6–8 under **To verify a document revision**, but change a single character in the **Digest** input string. This should cause your **Verify** request to fail with an appropriate error message.

If you no longer need to use the `vehicle-registration` ledger, proceed to Step 7 (optional): Clean up resources (p. 38).

Step 7 (optional): Clean up resources

You can continue using the `vehicle-registration` ledger. However, if you no longer need it, you should delete it.

If deletion protection is enabled for your ledger, you must first disable it before you can delete the ledger using the QLDB API or the AWS Command Line Interface (AWS CLI). The QLDB console disables deletion protection for you when you use it to delete a ledger.

To delete the ledger

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Ledgers**.
3. In the list of ledgers, select `vehicle-registration`.
4. Choose **Delete ledger**. Confirm this by entering **`vehicle-registration`** in the field provided.
5. If deletion protection is enabled for this ledger, you must also choose the option to **Override deletion protection**.

To learn more about working with ledgers in QLDB, see Getting started with Amazon QLDB: Next steps (p. 39).

Getting started with Amazon QLDB: Next steps

For more information about using Amazon QLDB, see the following topics:

- Working with data and history in Amazon QLDB (p. 317)
- Getting started with the Amazon QLDB driver (p. 40)
- Amazon QLDB concurrency model (p. 332)
- Data verification in Amazon QLDB (p. 338)
- Exporting journal data from Amazon QLDB (p. 354)
- Amazon QLDB PartiQL reference (p. 426)

Getting started with the Amazon QLDB driver

This section contains hands-on tutorials to help you learn about developing with Amazon QLDB by using the QLDB driver. The driver is built on top of the AWS SDK, which enables your application to interact with the QLDB API (p. 500).

QLDB session abstraction

The driver provides a high-level abstraction layer above the transactional data API (*QLDB Session*). It streamlines the execution of PartiQL (p. 426) statements on ledger data by managing SendCommand API calls. These API calls require several parameters that the driver handles for you, including the management of sessions, transactions, and retry policy in case of errors.

Amazon Ion support

In addition, the driver uses Amazon Ion (p. 487) libraries to enable support for handling Ion data when executing transactions. These libraries also take care of calculating the hash of Ion values, which QLDB requires to check the integrity of data transaction requests.

Driver terminology

This tool is called a *driver* because it's comparable to other database drivers that provide developer-friendly interfaces. These drivers similarly encapsulate logic that converts a standard set of commands and functions into specific calls that are required by the service's low-level API.

The driver is open-sourced on GitHub and is available for the following programming languages.

Topics

Amazon QLDB driver for Java

To work with data in your ledger, you can connect to Amazon QLDB from your Java application by using an AWS-provided driver. The following sections describe how to get started with the QLDB driver for Java.

Topics

Driver resources

For more information about the functionality supported by the Java driver, see the following resources:

- API Reference: 2.x, 1.x
- Driver recommendations (p. 310)
- Common errors (p. 315)
- Ion code examples (p. 489)
- Driver source code (GitHub)
- Sample application source code (GitHub)

Prerequisites

Before you get started with the QLDB driver for Java, you must do the following:

1. Follow the AWS setup instructions in Accessing Amazon QLDB (p. 15).
2. Choose a Java integrated development environment (IDE):
   - **AWS Cloud9** *(recommended)* – Use AWS Cloud9 as your Java IDE and automate setup of the QLDB sample application with a provided AWS CloudFormation template. This option requires the least amount of manual installation and configuration work.

     For this option, skip the rest of this section and proceed to the Java tutorial (p. 44). Then, follow the *AWS Cloud9* steps in Installing the sample application (p. 46).
   - **Other IDE** – Use your own development resources to manually install your preferred IDE.

     For this option, continue the following setup steps.
3. Set up a Java development environment by downloading and installing the following:
   - Java SE Development Kit 8 (such as Amazon Corretto 8).
   - Java IDE (such as Eclipse or IntelliJ).
4. Configure your AWS credentials and Region for development:
   - Set up your AWS security credentials (p. 41) for use with the AWS SDK for Java.
   - Set your AWS Region (p. 42) to determine your default QLDB endpoint.

Next, you can download the complete tutorial sample application—or you can install only the driver in an existing Java project.

- To run the complete tutorial with the QLDB sample application, see the Java tutorial (p. 44).
- To install only the QLDB driver and the AWS SDK for Java in an existing project, proceed to Installation (driver only) (p. 42).

Setting your AWS credentials

The QLDB driver and the underlying AWS SDK for Java 2.x require that you provide AWS credentials to your application at runtime. The code examples in this guide assume that you are using an AWS credentials file, as described in Set up AWS credentials and Region for development in the *AWS SDK for Java 2.x Developer Guide*.

The following is an example of an AWS credentials file named `~/.aws/credentials`, where the tilde character (~) represents your home directory.

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

Setting the AWS Region and endpoint

As part of the previous step to set up your AWS credentials, you should also set your default AWS Region. The sample application connects to QLDB in your default AWS Region. You can also change the Region in your code by modifying the `AmazonQLDB` client object (control plane) or the `QldbSessionClientBuilder` object (transactional data plane).

The following code example instantiates a new `AmazonQLDB` client object.

```
import com.amazonaws.regions.Regions;
import com.amazonaws.services.qldb.AmazonQLDB;
import com.amazonaws.services.qldb.AmazonQLDBClientBuilder;

// This client will default to US East (Ohio)
AmazonQLDB client = AmazonQLDBClientBuilder.standard()
    .withRegion(Regions.US_EAST_2)
    .build();
```

You can use the `withRegion` method to run your code against QLDB in any Region where it is available. For a complete list, see Amazon QLDB endpoints and quotas in the *Amazon Web Services General Reference*.

Installation (driver only)

QLDB supports the following Java driver versions and their AWS SDK dependencies.

Driver version	AWS SDK	Status	Latest release date
1.x	AWS SDK for Java 1.x	Production release	March 20, 2020
2.x	AWS SDK for Java 2.x	Production release	August 28, 2020

To install the QLDB driver and the AWS SDK for Java, we recommend using a dependency management system, such as Gradle or Maven. Add the following artifacts as dependencies in your Java project:

2.x

1. amazon-qldb-driver-java – The QLDB driver for Java. The latest version is `2.0.0`.

   This artifact automatically includes the AWS SDK for Java 2.x core module, Amazon Ion (p. 487) libraries, and other required dependencies.
2. aws-java-sdk-qldb – The QLDB module of the AWS SDK for Java. The minimum QLDB-supported version is `1.11.785`.

   This module enables your application to interact with the control plane API operations listed in the Amazon QLDB API reference (p. 500).
3. jackson-dataformat-ion – (Optional) The open source Jackson data format module for Ion. You need this artifact to run the code in the Java tutorial (p. 44). The sample application requires version `2.10.0.pr1` or later.

   If you are using your own application to interact with QLDB, you don't need to add this Jackson dependency to your project.

For example, if you use Gradle or Maven as your build tool, add the following dependencies in your `build.gradle` (Gradle) or `pom.xml` (Maven) configuration file.

Example — Gradle

```
dependencies {
    compile group: 'software.amazon.qldb', name: 'amazon-qldb-driver-java', version:
 '2.0.0'
    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.pr1'
}
```

Example — Maven

```
<dependencies>
  <dependency>
    <groupId>software.amazon.qldb</groupId>
    <artifactId>amazon-qldb-driver-java</artifactId>
    <version>2.0.0</version>
  </dependency>
  <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.pr1</version>
  </dependency>
</dependencies>
```

1.x

1. amazon-qldb-driver-java – The QLDB driver for Java. The latest version is `1.1.0`.

   This artifact automatically includes the AWS SDK for Java core module, Amazon Ion (p. 487) libraries, and other required dependencies.

   > **Important**
   > **Amazon Ion namespace** – You must use the Amazon Ion package that is under the namespace `com.amazon.ion` in your application. The AWS SDK for Java depends on another Ion package under the namespace `software.amazon.ion`, but this is a legacy package that is not compatible with the QLDB driver.

2. aws-java-sdk-qldb – The QLDB module of the AWS SDK for Java. The minimum QLDB-supported version is `1.11.785`.

   This module enables your application to interact with the control plane API operations listed in the Amazon QLDB API reference (p. 500).

3. jackson-dataformat-ion – (Optional) The open source Jackson data format module for Ion. You need this artifact to run the code in the Java tutorial (p. 44). The sample application requires version `2.10.0.pr1` or later.

   If you are using your own application to interact with QLDB, you don't need to add this Jackson dependency to your project.

For example, if you use Gradle or Maven as your build tool, add the following dependencies in your `build.gradle` (Gradle) or `pom.xml` (Maven) configuration file.

Example — Gradle

```
dependencies {
    compile group: 'software.amazon.qldb', name: 'amazon-qldb-driver-java', version:
 '1.1.0'
    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.pr1'
}
```

Example — Maven

```
<dependencies>
  <dependency>
    <groupId>software.amazon.qldb</groupId>
    <artifactId>amazon-qldb-driver-java</artifactId>
    <version>1.1.0</version>
  </dependency>
  <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.pr1</version>
  </dependency>
</dependencies>
```

For code examples that demonstrate operations such as how to create a ledger and run data transactions on a ledger, see the Java tutorial (p. 44).

Amazon QLDB Java tutorial

In this tutorial, you use the Amazon QLDB driver with the AWS SDK for Java to create a QLDB ledger and populate it with sample data. The driver enables your application to interact with QLDB using the transactional data API. The AWS SDK for Java enables interaction with the QLDB control plane API.

The sample ledger that you create in this scenario is a department of motor vehicles (DMV) database that tracks the complete historical information about vehicle registrations. The following sections explain how to add vehicle registrations, modify them, and view the history of changes to those registrations. This guide also shows you how to verify a registration document cryptographically, and it concludes by cleaning up resources and deleting the sample ledger.

As you work through this tutorial, you can refer to the AWS SDK for Java API Reference for control plane operations. For transactional data operations, you can refer to the QLDB Driver for Java API Reference.

Note
Where applicable, some tutorial steps have different commands or code examples for each supported major version of the QLDB driver for Java.

Topics

- Step 5: Modify documents in a ledger (p. 115)
- Step 6: View the revision history for a document (p. 126)
- Step 7: Verify a document in a ledger (p. 130)
- Step 8 (optional): Clean up resources (p. 174)

Installing the Amazon QLDB Java sample application

This section describes how to install and run the provided Amazon QLDB sample application for the step-by-step Java tutorial. The use case for this sample application is a department of motor vehicles (DMV) database that tracks the complete historical information about vehicle registrations.

The DMV sample application for Java is open-sourced in the GitHub repository aws-samples/amazon-qldb-dmv-sample-java.

Prerequisites

Before you get started, make sure that you complete the Prerequisites (p. 41). This includes signing up for AWS and—if you're using an IDE other than AWS Cloud9—getting an AWS access key for development and installing Java.

Installation

The following steps describe how to download and set up the sample application with a local development environment. Or, you can automate setup of the sample application by using AWS Cloud9 as your IDE.

Local development environment

These instructions describe how to download and install the QLDB Java sample application using your own resources and development environment.

To download and run the sample application

1. Enter the following command to clone the sample application from GitHub.

   2.x

   ```
   git clone -b v2.0.0 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
   ```

   This package includes the Gradle configuration and complete code from the Java tutorial (p. 44).
2. Load and run the provided application.
   - If you are using Eclipse:

     a. Start Eclipse, and on the **Eclipse** menu, choose **File**, **Import**, and then **Existing Gradle Project**.
     b. In the project root directory, browse and select the application directory that contains the `build.gradle` file. Then, choose **Finish** to use the default Gradle settings for the import.
     c. You can try running the `ListLedgers` program as an example. Open the context (right-click) menu for the `ListLedgers.java` file, and choose **Run as Java Application**.
   - If you are using IntelliJ:

   a. Start IntelliJ, and on the **IntelliJ** menu, choose **File** and then **Open**.
   b. In the project root directory, browse and select the application directory that contains the `build.gradle` file. Then, choose **OK**. Keep the default settings, and choose **OK** again.
   c. You can try running the `ListLedgers` program as an example. Open the context (right-click) menu for the `ListLedgers.java` file, and choose **Run 'ListLedgers'**.
3. Proceed to Step 1: Create a new ledger (p. 48) to start the tutorial and create a ledger.

AWS Cloud9

These instructions describe how to automate setup of the Amazon QLDB vehicle registration sample application for Java, using AWS Cloud9 as your IDE. In this guide, you use an AWS CloudFormation template to provision your development resources.

For more information about AWS Cloud9, see the AWS Cloud9 User Guide. To learn more about AWS CloudFormation, see the AWS CloudFormation User Guide.

Topics

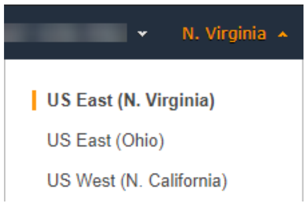
Part 1: Provision your resources

In this first step, you use AWS CloudFormation to provision the resources required to set up your development environment with the Amazon QLDB sample application.

To open the AWS CloudFormation console and load the QLDB sample application template

1. Sign in to the AWS Management Console and open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation.

   Switch to a Region that supports QLDB. For a complete list, see Amazon QLDB endpoints and quotas in the *AWS General Reference*.



2. On the AWS CloudFormation console, choose **Create stack**, and then choose **With new resources (standard)**.
3. On the **Create stack** page under **Specify template**, choose **Amazon S3 URL**.
4. Enter the following URL, and choose **Next**.

```
https://amazon-qldb-assets.s3.amazonaws.com/templates/QLDB-DMV-SampleApp.yml
```

5. Enter a **Stack name** (such as **`qldb-sample-app`**), and choose **Next**.
6. You can add any tags as appropriate and keep the default options. Then choose **Next**.
7. Review your stack settings, and choose **Create stack**. The AWS CloudFormation script might take a few minutes to finish.

   This script provisions your AWS Cloud9 environment with an associated Amazon Elastic Compute Cloud (Amazon EC2) instance that you use to run the QLDB sample application in this tutorial. It also clones the aws-samples/amazon-qldb-dmv-sample-java repository from GitHub into your AWS Cloud9 development environment.

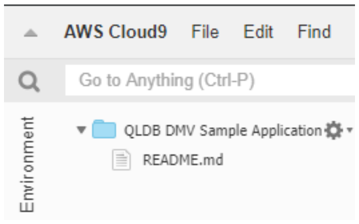
Part 2: Set up your IDE

In this step, you finish the setup of your cloud development environment. You download and run a provided shell script to set up your AWS Cloud9 IDE with sample application's dependencies.

To set up your AWS Cloud9 environment

1. Open the AWS Cloud9 console at https://console.aws.amazon.com/cloud9/.
2. Under **Your environments**, locate the card for the environment named **QLDB DMV Sample Application**, and choose **Open IDE**. Your environment might take a minute to load as the underlying EC2 instance launches.

   Your AWS Cloud9 environment is preconfigured with the system dependencies that you need to run the tutorial. In the **Environment** navigation pane of your console, confirm that you see a folder named `QLDB DMV Sample Application`.

   If you don't see a navigation pane, toggle the **Environment** tab on the left side of your console. If you don't see any folders in the pane, enable **Show Environment Root** using the settings icon.
3. On the bottom pane of your console, you should see an open `bash` terminal window. If you don't see this, choose **New Terminal** from the **Window** menu at the top of your console.
4. Next, download and run a setup script to install OpenJDK 8 and—if applicable—check out the appropriate branch from the Git repository. In the AWS Cloud9 terminal that you created in the previous step, run the following two commands in order:

   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
   ```

   Upon completion, you should see the following message printed in the terminal:

   ```
   ** DMV Sample App setup completed , enjoy!! **
   ```
5. Take a moment to browse the sample application code in AWS Cloud9, particularly in the following directory path: `src/main/java/software/amazon/qldb/tutorial`.

Part 3: Run the QLDB DMV sample application

In this step, you learn how to run the Amazon QLDB DMV sample application tasks using AWS Cloud9. To run the sample code, go back to your AWS Cloud9 terminal or create a new terminal window as you did in *Part 2: Set Up Your IDE*.

To run the sample application

1. Run the following command in your terminal to switch to the project root directory:

   ```
   cd ~/environment/amazon-qldb-dmv-sample-java
   ```

   Ensure that you are running the examples in the following directory path.

   ```
   /home/ec2-user/environment/amazon-qldb-dmv-sample-java/
   ```
2. The following command shows the Gradle syntax to run each task.

   ```
   ./gradlew run -Dtutorial=Task
   ```

   For example, run the following command to list all of the ledgers in your AWS account and current Region.

   ```
   ./gradlew run -Dtutorial=ListLedgers
   ```
3. Proceed to Step 1: Create a new ledger (p. 48) to start the tutorial and create a ledger.
4. (Optional) After you complete the tutorial, clean up your AWS CloudFormation resources if you no longer need them.

   a. Open the AWS CloudFormation console at https://console.aws.amazon.com/cloudformation, and delete the stack that you created in *Part 1: Provision Your Resources*.

   b. Also delete the AWS Cloud9 stack that the AWS CloudFormation template created for you.

Step 1: Create a new ledger

In this step, you create a new Amazon QLDB ledger named `vehicle-registration`.

To create a new ledger

1. Review the following file (`Constants.java`), which contains constant values that are used by all of the other programs in this tutorial.

   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

```
/*
 * 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);
    }
}
```

Important

For the Amazon Ion package, you must use the namespace `com.amazon.ion` in your application. The AWS SDK for Java depends on another Ion package under the namespace `software.amazon.ion`, but this is a legacy package that is not compatible with the QLDB driver.

Note
This `Constants` class includes an instance of the open source Jackson `IonValueMapper` class. You can use this mapper to process your Amazon Ion (p. 487) data when doing read and write transactions.

The `CreateLedger.java` file also has a dependency on the following program (`DescribeLedger.java`), which describes the current status of your ledger.

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.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;
    }
}
```

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.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. Compile and run the `CreateLedger.java` program to create a ledger named `vehicle-registration`.

   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

- In the `createLedger` call, you must specify a ledger name and a permissions mode. The only permissions mode that is currently supported for a ledger is `ALLOW_ALL`.
- When you create a ledger, *deletion protection* is enabled by default. This is a feature in QLDB that prevents ledgers from being deleted by any user. You have the option of disabling deletion protection on ledger creation using the QLDB API or the AWS Command Line Interface (AWS CLI).
- Optionally, you can also specify tags to attach to your ledger.

To verify your connection to the new ledger, proceed to Step 2: Test connectivity to the ledger (p. 57).

Step 2: Test connectivity to the ledger

In this step, you verify that you can connect to the `vehicle-registration` ledger in Amazon QLDB using the transactional data API endpoint.

To test connectivity to the ledger

1. Review the following program (`ConnectToLedger.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.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

- To run data operations on your ledger, you must create an instance of the `QldbDriver` class to connect to a specific ledger. This is a different client object than the `AmazonQLDB` client that you used in the previous step to create the ledger. That previous client is only used to run the control plane API operations listed in the Amazon QLDB API reference (p. 500).
- First, create a `QldbDriver` object. You must specify a ledger name when you create this driver.

  Then, you can use this driver's `execute` method to run PartiQL statements.
- Optionally, you can specify a maximum number of retry attempts for transaction exceptions. The `execute` method automatically retries optimistic concurrency control (OCC) conflicts and other common transient exceptions up to this configurable limit. If the transaction still fails after the limit is reached, then

the driver throws the exception. To learn more, see Handling OCC conflict exceptions (p. 335).

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

- To run data operations on your ledger, you must create an instance of the `PooledQldbDriver` or `QldbDriver` class to connect to a specific ledger. This is a different client object than the `AmazonQLDB` client that you used in the previous step to create the ledger. That previous client is only used to run the control plane API operations listed in the Amazon QLDB API reference (p. 500).

  We recommended that you use `PooledQldbDriver` unless you need to implement a custom session pool with `QldbDriver`. The default pool size for

`PooledQldbDriver` is the maximum number of open HTTP connections that the session client allows.

- First, create a `PooledQldbDriver` object. You must specify a ledger name when you create this driver.

  Then, you can use this driver's `execute` method to run PartiQL statements. Or, you can manually create a session from this pooled driver object and use the session's `execute` method. A session represents a single connection with the ledger.
- Optionally, you can specify a maximum number of retry attempts for transaction exceptions. The `execute` method automatically retries optimistic concurrency control (OCC) conflicts and other common transient exceptions up to this configurable limit. If the transaction still fails after the limit is reached, then the driver throws the exception. To learn more, see Handling OCC conflict exceptions (p. 335).

2. Compile and run the program to test your data session connectivity to the `vehicle-registration` ledger.

Overriding the AWS Region

The sample application connects to QLDB in your default AWS Region, which you can set as described in the prerequisite step Setting the AWS Region and endpoint (p. 42). You can also change the Region by modifying the QLDB session client builder properties.

2.x

The following code example instantiates a new `QldbSessionClientBuilder` object.

```
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);
```

You can use the `region` method to run your code against QLDB in any Region where it is available. For a complete list, see Amazon QLDB endpoints and quotas in the *AWS General Reference*.

1.x

The following code example instantiates a new `AmazonQLDBSessionClientBuilder` object.

```
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);
```

You can use the `withRegion` method to run your code against QLDB in any Region where it is available. For a complete list, see Amazon QLDB endpoints and quotas in the *AWS General Reference*.

To create tables in the `vehicle-registration` ledger, proceed to Step 3: Create tables, indexes, and sample data (p. 63).

Step 3: Create tables, indexes, and sample data

When your Amazon QLDB ledger is active and accepts connections, you can start creating tables for data about vehicles, their owners, and their registration information. After creating the tables and indexes, you can load them with data.

In this step, you create four tables in the `vehicle-registration` ledger:

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

You also create the following indexes.

Table name	Field
`VehicleRegistration`	`VIN`
`VehicleRegistration`	`LicensePlateNumber`
`Vehicle`	`VIN`
`Person`	`GovId`
`DriversLicense`	`LicenseNumber`
`DriversLicense`	`PersonId`

When inserting sample data, you first insert documents into the `Person` table. Then, you use the system-assigned `id` from each `Person` document to populate the corresponding fields in the appropriate `VehicleRegistration` and `DriversLicense` documents.

Tip

As a best practice, use a document's system-assigned `id` as a foreign key. While you can define fields that are intended to be unique identifiers (for example, a vehicle's VIN), the true unique identifier of a document is its `id`. This field is included in the document's metadata, which you can query in the *committed view* (the system-defined view of a table).

For more information about views in QLDB, see Core concepts (p. 7). To learn more about metadata, see Querying document metadata (p. 323).

To set up the sample data

1. Review the following `.java` files. These model classes represent documents that you store in the `vehicle-registration` tables. They are serializable to and from Amazon Ion format.

   Note

   QLDB documents (p. 427) are stored in Ion format. So, you can use the open source Jackson library to model the data in JSON.

   2.x

   1. `DriversLicense.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.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. `IonLocalDateDeserializer.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());
    }
}
```

9. `IonLocalDateSerializer.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);
    }
}
```

10. `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 { }
```

1.x

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;

import java.time.LocalDate;

/**
 * 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. `IonLocalDateDeserializer.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());
    }
}
```

9. `IonLocalDateSerializer.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);
    }
}
```

10RevisionData.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 { }
```

2. Review the following file (`SampleData.java`), which represents the sample data that you insert into the `vehicle-registration` tables.

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

```
/*
 * 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()));
    }
}
```

Important
For the Amazon Ion package, you must use the namespace `com.amazon.ion` in your application. The AWS SDK for Java depends on another Ion package under the namespace `software.amazon.ion`, but this is a legacy package that is not compatible with the QLDB driver.

Note

- This class uses the Amazon Ion library to provide helper methods that convert your data to and from Ion format.
- The `getDocumentId` method runs a query on a table with the prefix `_ql_committed_`. This is a reserved prefix signifying that you want to query the *committed view* of a table. In this view, your data is nested in the `data` field, and metadata is nested in the `metadata` field.

3. Compile and run the following program (`CreateTable.java`) to create the previously mentioned tables.

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
This program demonstrates how to pass a `TransactionExecutor` lambda to the `execute` method. In this example, you run multiple `CREATE TABLE` PartiQL statements in a single transaction by using a lambda expression.
The `execute` method implicitly starts a transaction, runs all of the statements in the lambda, and then auto-commits the transaction.

4. Compile and run the following program (`CreateIndex.java`) to create indexes on the tables, as previously described.

   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. Compile and run the following program (`InsertDocument.java`) to insert the sample data into your tables.

   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

- This program demonstrates how to call the `execute` method with parameterized values. You can pass data parameters of type `IonValue` in addition to the PartiQL statement that you want to run. Use a question mark (`?`) as a variable placeholder in your statement string.
- If an `INSERT` statement succeeds, it returns the `id` of each inserted document.

Next, you can use `SELECT` statements to read data from the tables in the `vehicle-registration` ledger. Proceed to Step 4: Query the tables in a ledger (p. 108).

Step 4: Query the tables in a ledger

After creating tables in an Amazon QLDB ledger and loading them with data, you can run queries to review the vehicle registration data that you just inserted. QLDB uses PartiQL (p. 426) as its query language and Amazon Ion (p. 487) as its document-oriented data model.

PartiQL is an open source, SQL-compatible query language that has been extended to work with Ion. With PartiQL, you can insert, query, and manage your data with familiar SQL operators. Amazon Ion is a superset of JSON. Ion is an open source, document-based data format that gives you the flexibility of storing and processing structured, semistructured, and nested data.

In this step, you use `SELECT` statements to read data from the tables in the `vehicle-registration` ledger.

To query the tables

1. Compile and run the following program (`ScanTable.java`) to scan all documents in all tables in your ledger.

   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.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.amazon.ion.IonStruct;

import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;

/**
 * Scan for all the documents in a table.
 *
 * 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 ScanTable {
    private static final Logger log = LoggerFactory.getLogger(ScanTable.class);

    private ScanTable() { }

    /**
     * Scan the table with the given {@code tableName} for all documents.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *              Name of the table to scan.
     * @return a list of documents in {@link IonStruct} .
     */
    public static List<IonStruct> scanTableForDocuments(final TransactionExecutor
 txn, final String tableName) {
        log.info("Scanning '{}'...", tableName);
        final String scanTable = String.format("SELECT * FROM %s", tableName);
        List<IonStruct> documents = toIonStructs(txn.execute(scanTable));
        log.info("Scan successful!");
        printDocuments(documents);
        return documents;
    }

    /**
     * Pretty print all elements in the provided {@link Result}.
     *
     * @param result
     *              {@link Result} from executing a query.
     */
    public static void printDocuments(final Result result) {
        result.iterator().forEachRemaining(row -> log.info(row.toPrettyString()));
    }

    /**
     * Pretty print all elements in the provided list of {@link IonStruct}.
     *
     * @param documents
     *              List of documents to print.
     */
    public static void printDocuments(final List<IonStruct> documents) {
        documents.forEach(row -> log.info(row.toPrettyString()));
    }

    /**
     * Convert the result set into a list of {@link IonStruct}.
     *
     * @param result
     *              {@link Result} from executing a query.
     * @return a list of documents in IonStruct.
     */
    public static List<IonStruct> toIonStructs(final Result result) {
        final List<IonStruct> documentList = new ArrayList<>();
        result.iterator().forEachRemaining(row -> documentList.add((IonStruct)
 row));
        return documentList;
    }

    public static void main(final String... args) {
        ConnectToLedger.getDriver().execute(txn -> {
            List<String> tableNames = scanTableForDocuments(txn,
 Constants.USER_TABLES)
                .stream()
                .map((s) -> s.get("name").toString())
                .collect(Collectors.toList());
            for (String tableName : tableNames) {
                scanTableForDocuments(txn, tableName);
            }
        });
    }
}
```

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.IonStruct;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.qldb.QldbSession;
import software.amazon.qldb.Result;
import software.amazon.qldb.TransactionExecutor;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

/**
 * Scan for all the documents in a table.
 *
 * 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 ScanTable {
    private static final Logger log = LoggerFactory.getLogger(ScanTable.class);

    private ScanTable() { }

    /**
     * Scan the table with the given {@code tableName} for all documents.
     *
     * @param txn
     *              The {@link TransactionExecutor} for lambda execute.
     * @param tableName
     *              Name of the table to scan.
     * @return a list of documents in {@link IonStruct} .
     */
    public static List<IonStruct> scanTableForDocuments(final TransactionExecutor
 txn, final String tableName) {
        log.info("Scanning '{}'...", tableName);
        final String scanTable = String.format("SELECT * FROM %s", tableName);
        List<IonStruct> documents = toIonStructs(txn.execute(scanTable));
        log.info("Scan successful!");
        printDocuments(documents);
        return documents;
    }

    /**
     * Pretty print all elements in the provided {@link Result}.
     *
     * @param result
     *              {@link Result} from executing a query.
     */
    public static void printDocuments(final Result result) {
        result.iterator().forEachRemaining(row -> log.info(row.toPrettyString()));
    }

    /**
     * Pretty print all elements in the provided list of {@link IonStruct}.
     *
     * @param documents
     *              List of documents to print.
     */
    public static void printDocuments(final List<IonStruct> documents) {
        documents.forEach(row -> log.info(row.toPrettyString()));
    }

    /**
     * Convert the result set into a list of {@link IonStruct}.
     *
     * @param result
     *              {@link Result} from executing a query.
     * @return a list of documents in IonStruct.
     */
    public static List<IonStruct> toIonStructs(final Result result) {
        final List<IonStruct> documentList = new ArrayList<>();
        result.iterator().forEachRemaining(row -> documentList.add((IonStruct)
 row));
        return documentList;
    }

    public static void main(final String... args) {
        ConnectToLedger.getDriver().execute(txn -> {
            List<String> tableNames = scanTableForDocuments(txn,
 Constants.USER_TABLES)
                .stream()
                .map((s) -> s.get("name").toString())
                .collect(Collectors.toList());
            for (String tableName : tableNames) {
                scanTableForDocuments(txn, tableName);
            }
        }, (retryAttempt) -> log.info("Retrying due to OCC conflict..."));
    }
}
```

2. Compile and run the following program (`FindVehicles.java`) to query all vehicles registered under a person in your ledger.

   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
First, this program queries the `Person` table for the document with `GovId LEWISR261LL` to get its `id` metadata field.
Then, it uses this document `id` as a foreign key to query the `VehicleRegistration` table by `PrimaryOwner.PersonId`. It also joins `VehicleRegistration` with the `Vehicle` table on the `VIN` field.

To learn about modifying documents in the tables in the `vehicle-registration` ledger, see Step 5: Modify documents in a ledger (p. 115).

Step 5: Modify documents in a ledger

Now that you have data to work with, you can start making changes to documents in the `vehicle-registration` ledger in Amazon QLDB. In this step, the following code examples demonstrate how to run data manipulation language (DML) statements. These statements update the primary owner of one vehicle and add a secondary owner to another vehicle.

To modify documents

1. Compile and run the following program (`TransferVehicleOwnership.java`) to update the primary owner of the vehicle with VIN `1N4AL11D75C109151` in your ledger.

   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. Compile and run the following program (`AddSecondaryOwner.java`) to add a secondary owner to the vehicle with VIN `KM8SRDHF6EU074761` in your ledger.

   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 = '%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);
            }
        });
        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.");
    }
}
```

To review these changes in the `vehicle-registration` ledger, see Step 6: View the revision history for a document (p. 126).

Step 6: View the revision history for a document

After modifying registration data for a vehicle in the previous step, you can query the history of all its registered owners and any other updated fields. In this step, you query the revision history of a document in the `VehicleRegistration` table in your `vehicle-registration` ledger.

To view the revision history

1. Review the following program (`QueryHistory.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.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

- You can view the revision history of a document by querying the built-in History function (p. 326) in the following syntax.

```
SELECT * FROM history( table [, `start-time`, `end-time`] ) AS h
WHERE h.metadata.id = 'id'
```

- Best practice is to qualify a history query with a document `id`. This helps to avoid inefficient queries.

- The `start-time` and `end-time` are both optional. They are Amazon Ion literal values that can be denoted with backticks (`` `...` ``). To learn more, see Querying Ion with PartiQL (p. 444).

2. Compile and run the program to query the revision history of the `VehicleRegistration` document with VIN `1N4AL11D75C109151`.

To cryptographically verify a document revision in the `vehicle-registration` ledger, proceed to Step 7: Verify a document in a ledger (p. 130).

Step 7: Verify a document in a ledger

With Amazon QLDB, you can efficiently verify the integrity of a document in your ledger's journal by using cryptographic hashing with SHA-256. To learn more about how verification and cryptographic hashing work in QLDB, see Data verification in Amazon QLDB (p. 338).

In this step, you verify a document revision in the `VehicleRegistration` table in your `vehicle-registration` ledger. First, you request a digest, which is returned as an output file and acts as a signature of your ledger's entire change history. Then, you request a proof for the revision relative to that digest. Using this proof, the integrity of your revision is verified if all validation checks pass.

To verify a document revision

1. Review the following `.java` files, which represent QLDB objects that are required for verification and a utility class with helper methods to pretty-print certain QLDB response types.

   2.x

   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. `QldbRevision.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 IonStruct data;
    @JsonCreator
    public QldbRevision(@JsonProperty("blockAddress") final BlockAddress
blockAddress,
                        @JsonProperty("metadata") final RevisionMetadata
metadata,
                        @JsonProperty("hash") final byte[] hash,
                        @JsonProperty("data") final IonStruct data) {
        this.blockAddress = blockAddress;
        this.metadata = metadata;
        this.hash = hash;
        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 data.
     *
     * @return the byte array representing the hash.
     */
    public byte[] getHash() {
        return hash;
    }

    /**
     * Gets the revision data.
     *
     * @return the revision data.
     */
    public IonStruct getData() {
        return data;
    }

    /**
     * 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 document's hash calculated by QLDB,
     * - 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 hash = (IonBlob) ionStruct.get("hash");
            IonStruct metadataStruct = (IonStruct) ionStruct.get("metadata");
            IonStruct data = (IonStruct) ionStruct.get("data");
            if (hash == null || data == null) {
                throw new IllegalArgumentException("Document is missing required
fields");
            }
            verifyRevisionHash(metadataStruct, data, hash.getBytes());
            RevisionMetadata metadata = RevisionMetadata.fromIon(metadataStruct);
            return new QldbRevision(blockAddress, metadata, hash.getBytes(),
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) +
                ", 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()) &&
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) {
            return;
        }

        try {
            IonStruct metadataIon = (IonStruct)
 Constants.MAPPER.writeValueAsIonValue(metadata);
            verifyRevisionHash(metadataIon, data, hash);
        } catch (IOException e) {
            throw new IllegalArgumentException("Could not encode revision
 metadata to ion.", e);
        }
    }

    private static void verifyRevisionHash(IonStruct metadata, IonStruct
 revisionData, byte[] expectedHash) {
        byte[] metadataHash = QldbIonUtils.hashIonValue(metadata);
        byte[] dataHash = QldbIonUtils.hashIonValue(revisionData);
        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");
        }
    }
}
```

4. `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
    }
}
```

5. `Verifier.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 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;
    }
}
```

6. `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();
    }

}
```

7. `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();
    }
}
```

1.x

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. `QldbRevision.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 IonStruct data;

    @JsonCreator
    public QldbRevision(@JsonProperty("blockAddress") final BlockAddress
 blockAddress,
                        @JsonProperty("metadata") final RevisionMetadata
 metadata,
                        @JsonProperty("hash") final byte[] hash,
                        @JsonProperty("data") final IonStruct data) {
        this.blockAddress = blockAddress;
        this.metadata = metadata;
        this.hash = hash;
        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 data.
     *
     * @return the byte array representing the hash.
     */
    public byte[] getHash() {
        return hash;
    }

    /**
     * Gets the revision data.
     *
     * @return the revision data.
     */
    public IonStruct getData() {
        return data;
    }

    /**
     * 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 document's hash calculated by QLDB,
     * - 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 hash = (IonBlob) ionStruct.get("hash");
            IonStruct metadataStruct = (IonStruct) ionStruct.get("metadata");
            IonStruct data = (IonStruct) ionStruct.get("data");
            if (hash == null || data == null) {
                throw new IllegalArgumentException("Document is missing required
fields");
            }
            verifyRevisionHash(metadataStruct, data, hash.getBytes());
            RevisionMetadata metadata = RevisionMetadata.fromIon(metadataStruct);
            return new QldbRevision(blockAddress, metadata, hash.getBytes(),
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) +
                ", 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()) &&
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) {
            return;
        }

        try {
            IonStruct metadataIon = (IonStruct)
Constants.MAPPER.writeValueAsIonValue(metadata);
            verifyRevisionHash(metadataIon, data, hash);
        } catch (IOException e) {
            throw new IllegalArgumentException("Could not encode revision
metadata to ion.", e);
        }
    }

    private static void verifyRevisionHash(IonStruct metadata, IonStruct
revisionData, byte[] expectedHash) {
        byte[] metadataHash = QldbIonUtils.hashIonValue(metadata);
        byte[] dataHash = QldbIonUtils.hashIonValue(revisionData);
        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");
        }
    }
}
```

4. `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
    }
}
```

5. `Verifier.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.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;
    }
}
```

6. `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();
    }

}
```

7. `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();
    }
}
```

2. Review the following two `.java` programs, which contain code examples that demonstrate the following steps:

   - Request a new digest from the `vehicle-registration` ledger.
   - Request a proof for each revision of a document from the `VehicleRegistration` table.

- Verify the revisions using the returned digest and proof by recalculating the digest.

1. `GetDigest.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 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;
    }
}
```

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.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
Use the `getDigest` method to request a digest that covers the current *tip* of the journal in your ledger. The tip of the journal refers to the latest committed block as of the time that QLDB receives your request.

2. `GetRevision.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 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
After the `getRevision` method returns a proof for the specified document revision, this program uses a client-side API to verify that revision. For an overview of the algorithm used by this API, see Using a proof to recalculate your digest (p. 347).

3. Compile and run the `GetRevision.java` program to cryptographically verify the `VehicleRegistration` document with VIN `1N4AL11D75C109151`.

If you no longer need to use the `vehicle-registration` ledger, proceed to Step 8 (optional): Clean up resources (p. 174).

Step 8 (optional): Clean up resources

You can continue using the `vehicle-registration` ledger. However, if you no longer need it, you should delete it.

To delete the ledger

- Compile and run the following program (`DeleteLedger.java`) to delete your `vehicle-registration` ledger and all of its contents.

  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.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;
    }
}
```

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.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
If deletion protection is enabled for your ledger, you must first disable it before you can delete the ledger using the QLDB API.

Amazon QLDB driver for .NET

To work with data in your ledger, you can connect to Amazon QLDB from your Microsoft .NET application by using an AWS-provided driver. The driver targets **.NET Standard 2.0**. More specifically, it supports **.NET Core (LTS) 2.1+** and **.NET Framework 4.5.2+**. For information on compatibility, see .NET Standard on the *Microsoft Docs* site.

The following sections describe how to get started with the QLDB driver for .NET.

Topics

Driver resources

For more information about the functionality supported by the .NET driver, see the following resources:

- API reference
- Driver recommendations (p. 310)
- Common errors (p. 315)
- Driver source code (GitHub)
- Amazon Ion Cookbook

Prerequisites

Before you get started with the QLDB driver for .NET, you must do the following:

1. Follow the AWS setup instructions in Accessing Amazon QLDB (p. 15). This includes signing up for AWS and getting an AWS access key for development.
2. Download and install the .NET Core SDK version 2.1 or later from the Microsoft .NET downloads site.
3. (Optional) Install an integrated development environment (IDE) of your choice, such as Visual Studio, Visual Studio for Mac, or Visual Studio Code. You can download these from the Microsoft Visual Studio site.
4. Configure your development environment for the AWS SDK for .NET:

   - Set up your AWS credentials. We recommend that you create a shared credentials file.

     For instructions, see Configuring AWS credentials using a credentials file in the *AWS SDK for .NET Developer Guide*.
   - Set your default AWS Region. To learn how, see AWS Region selection.

     For a complete list of available Regions, see Amazon QLDB endpoints and quotas in the *AWS General Reference*.

Next, you can set up a basic sample application and run short code examples—or you can install the driver in an existing .NET project.

- To install the QLDB driver and the AWS SDK for .NET in an existing project, proceed to Installation (p. 179).
- To run short code examples that execute basic data transactions on a ledger, see the Quick start (p. 180) guide.

Installation

Use the NuGet package manager to install the QLDB driver for .NET. We recommended that you use Visual Studio or an IDE of your choice to add project dependencies. The driver package name is Amazon.QLDB.Driver.

For example in Visual Studio, open the **NuGet Package Manager Console** on the **Tools** menu. Then, enter the following command at the `PM>` prompt.

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

Installing the driver also installs its dependencies, including the AWS SDK for .NET and Amazon Ion (p. 487) packages.

For short code examples of how to run basic data transactions on a ledger, see the Cookbook (p. 185) reference.

Amazon QLDB driver for .NET – Quick start

In this section, you learn how to set up a simple application using the Amazon QLDB driver for .NET. This guide includes steps for installing the driver and short code examples of basic *create, read, update, and delete* (CRUD) operations.

Topics

Prerequisites

Before you get started, make sure that you do the following:

1. Complete the Prerequisites (p. 179) for the .NET driver, if you haven't already done so. This includes signing up for AWS, getting an AWS access key for development, and installing the .NET Core SDK.
2. Create a ledger named `quick-start`.

   To learn how to create a ledger, see Basic operations for Amazon QLDB ledgers (p. 382) or Step 1: Create a new ledger (p. 25) in *Getting Started with the Console*.

Step 1: Set up your project

First, set up your .NET project.

1. To create and run a template application, enter the following `dotnet` commands on a terminal such as *bash*, *PowerShell*, or *Command Prompt*.

   ```
   $ dotnet new console --output qldb-quickstart
   $ dotnet run --project qldb-quickstart
   ```

   This template creates a folder named `qldb-quickstart`. In that folder, it creates a project with the same name and a file named `Program.cs`. The program contains code that displays the output `Hello World!`.
2. Use the NuGet package manager to install the QLDB driver for .NET. We recommended that you use Visual Studio or an IDE of your choice to add dependencies to your project. The driver package name is Amazon.QLDB.Driver.
   - For example in Visual Studio, open the **NuGet Package Manager Console** on the **Tools** menu. Then, enter the following command at the `PM>` prompt.

     ```
     PM> Install-Package Amazon.QLDB.Driver
     ```
   - Or, you can enter the following commands on your terminal.

     ```
     $ cd qldb-quickstart
     ```

```
$ dotnet add package Amazon.QLDB.Driver -v 1.0.0
```

Installing the driver also installs its dependencies, including the AWS SDK for .NET and Amazon Ion (p. 487) libraries.

3. Open the `Program.cs` file.

   Then, incrementally add the code examples in the following sections to try some basic CRUD operations. Or, you can skip the step-by-step tutorial and instead run the complete application (p. 184).

Step 2: Initialize the driver

Initialize an instance of the driver that connects to the ledger named `quick-start`. Add the following code to your `Program.cs` file.

```
using System;
using Amazon.QLDB.Driver;
using Amazon.QLDBSession;

namespace qldb_quickstart
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello QLDB!");
            AmazonQLDBSessionConfig amazonQldbSessionConfig = new
 AmazonQLDBSessionConfig();
            var ledgerName = "quick-start";
            Console.WriteLine("Create the QLDB Driver");
            IQldbDriver driver = QldbDriver.Builder()
                .WithQLDBSessionConfig(amazonQldbSessionConfig)
                .WithLedger(ledgerName)
                .Build();
        }
    }
}
```

Step 3: Create a table and an index

The following code example shows how to run `CREATE TABLE` and `CREATE INDEX` statements.

Add the following code that creates a table named `People` and an index for the `firstName` field on that table. Indexes (p. 448) can improve your query performance.

```
// ...
using System.Threading;
// ...

namespace qldb_quickstart
{
    class Program
    {
        static void Main(string[] args)
        {
            // ...
            // Creates the table and the index in the same transaction
            Console.WriteLine("Creating the tables and index);
            driver.Execute(txn =>
            {
                txn.Execute("CREATE TABLE People");
                txn.Execute("CREATE INDEX ON People(firstName)");
            });
            Thread.Sleep(2000);
        }
    }
}
```

Step 4: Insert a document

The following code example shows how to run an `INSERT` statement. QLDB supports the PartiQL (p. 426) query language (SQL compatible) and the Amazon Ion (p. 487) data format (superset of JSON).

1. Add the following code that inserts a document into the `People` table.

```
// ...
using System.IO;
using Amazon.IonDotnet;
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;
// ...

namespace qldb_quickstart
{
    class Program
    {
        static void Main(string[] args)
        {
            // ...
            Console.WriteLine("Inserting a document");
            IIonValue insertResult = driver.Execute(txn =>
            {
                IResult result = txn.Execute("INSERT INTO People {'firstName': 'John',
 'lastName': 'Doe', 'age':32}");
                foreach (var row in result)
                {
                    return row;
                }
                return null;
            });
        }
    }
}
```

2. To print the results, you can use a `TextWriter` combined with an `IonTextWriterBuilder` and an `IIonReader`.

```
// Printing the results of the document inserted
using TextWriter tw = new StringWriter();
using IIonWriter writer = IonTextWriterBuilder.Build(tw, new IonTextOptions
 { PrettyPrint = true });
using IIonReader reader = IonReaderBuilder.Build(insertResult);
writer.WriteValues(reader);
writer.Finish();
Console.WriteLine($"Document inserted {tw.ToString()}");
```

At a high level, this code example does the following:

1. The `IonTextWriterBuilder` asks for a `TextWriter` to store the results.

2. The `IonReaderBuilder` asks for the value to be read, which is the result of the insert statement.
3. The writer gets the value from the reader and stores it in the `TextWriter`.

Step 5: Query the document

The following code example shows how to run a `SELECT` statement.

Add the following code that queries a document from the `People` table.

```
// Selecting a document
IIonValue ionFirstName = IonLoader.Default.Load("John");
Console.WriteLine("Querying a table");
IIonValue selectResult = driver.Execute(txn =>
{
    IResult result = txn.Execute("SELECT firstName, age, lastName FROM People WHERE
 firstName = ?", ionFirstName);
    foreach (var row in result)
    {
        return row;
    }
    return null;
});
Console.WriteLine($"Person found {selectResult.GetField("firstName").StringValue}");
```

This example uses a question mark (`?`) as a variable placeholder to pass the document information to the statement. When you use placeholders, you must pass a value of type `IonValue`.

Step 6: Update the document

The following code example shows how to run an `UPDATE` statement.

1. Add the following code that updates a document in the `People` table by changing `age` to `50`.

```
// Updating a document
IIonValue ionIntAge = new ValueFactory().NewInt(50);
Console.WriteLine("Updating a document");
IIonValue updateResult = driver.Execute(txn =>
{
    IResult result = txn.Execute("UPDATE People SET age = ? WHERE firstName = ?",
 ionIntAge, ionFirstName);
    foreach (var row in result)
    {
        return row;
    }
    return null;
});
```

2. You can also print the `id` of the updated document by using the same technique that you used when printing the insert results.

```
// Printing the id of the updated document
using TextWriter twUpdate = new StringWriter();
using IIonWriter writerUpdate = IonTextWriterBuilder.Build(twUpdate, new IonTextOptions
 { PrettyPrint = true });
using IIonReader readerUpdate = IonReaderBuilder.Build(updateResult);
writerUpdate.WriteValues(readerUpdate);
writerUpdate.Finish();
Console.WriteLine($"Document updated {twUpdate.ToString()}");
```

3. To run the application, enter the following command from the parent directory of your `qldb-quickstart` project directory.

```
$ dotnet run --project qldb-quickstart
```

Running the complete application

The following example code is the complete version of the `Program.cs` application. Instead of doing the previous steps individually, you can also copy and run this code example end to end. This application demonstrates some basic CRUD operations on the ledger named `quick-start`.

> **Note**
> Before you run this code, make sure that you don't already have an active table named `People` in the `quick-start` ledger.

```
using System;
using System.IO;
using System.Threading;
using Amazon.IonDotnet;
using Amazon.IonDotnet.Builders;
using Amazon.IonDotnet.Tree;
using Amazon.IonDotnet.Tree.Impl;
using Amazon.QLDB.Driver;
using Amazon.QLDBSession;

namespace qldb_quickstart
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello QLDB!");
            AmazonQLDBSessionConfig amazonQldbSessionConfig = new
 AmazonQLDBSessionConfig();
            var ledgerName = "quick-start";
            Console.WriteLine("Create the QLDB Driver");
            IQldbDriver driver = QldbDriver.Builder()
                .WithQLDBSessionConfig(amazonQldbSessionConfig)
                .WithLedger(ledgerName)
                .Build();

            // Creates the table and the index in the same transaction
            Console.WriteLine("Creating the tables");
            driver.Execute(txn =>
            {
                txn.Execute("CREATE TABLE People");
                txn.Execute("CREATE INDEX ON People(firstName)");
            });
            Thread.Sleep(2000);

            Console.WriteLine("Inserting a document");
            IIonValue insertResult = driver.Execute(txn =>
            {
                IResult result = txn.Execute("INSERT INTO People {'firstName': 'John',
 'lastName': 'Doe', 'age':32}");
                foreach (var row in result)
                {
                    return row;
                }
                return null;
            });
            // Printing the results of the document inserted
            using TextWriter tw = new StringWriter();
            using IIonWriter writer = IonTextWriterBuilder.Build(tw, new IonTextOptions
 { PrettyPrint = true });
            using IIonReader reader = IonReaderBuilder.Build(insertResult);
            writer.WriteValues(reader);
            writer.Finish();
            Console.WriteLine($"Document inserted {tw.ToString()}");

            // Selecting a document
            IIonValue ionFirstName = IonLoader.Default.Load("John");
            Console.WriteLine("Querying a table");
            IIonValue selectResult = driver.Execute(txn =>
            {
                IResult result = txn.Execute("SELECT firstName, age, lastName FROM People
 WHERE firstName = ?", ionFirstName);
                foreach (var row in result)
                {
                    return row;
                }
                return null;
            });
            Console.WriteLine($"Person found
 {selectResult.GetField("firstName").StringValue}");

            // Updating a document
            IIonValue ionIntAge = new ValueFactory().NewInt(50);
            Console.WriteLine("Updating a document");
            IIonValue updateResult = driver.Execute(txn =>
            {
                IResult result = txn.Execute("UPDATE People SET age = ? WHERE firstName
 = ?", ionIntAge, ionFirstName);
                foreach (var row in result)
                {
                    return row;
                }
                return null;
            });

            // Printing the id of the updated document
            using TextWriter twUpdate = new StringWriter();
            using IIonWriter writerUpdate = IonTextWriterBuilder.Build(twUpdate, new
 IonTextOptions { PrettyPrint = true });
            using IIonReader readerUpdate = IonReaderBuilder.Build(updateResult);
            writerUpdate.WriteValues(readerUpdate);
            writerUpdate.Finish();
            Console.WriteLine($"Document updated {twUpdate.ToString()}");
        }
    }
}
```

To run the complete application, enter the following command from the parent directory of your `qldb-quickstart` project directory.

```
$ dotnet run --project qldb-quickstart
```

Amazon QLDB driver for .NET – Cookbook

This section is a reference guide for common use cases of the Amazon QLDB driver for .NET. It provides C# code examples that show how to use the driver to run basic *create, read, update, and delete* (CRUD) operations. It also includes code examples for processing Amazon Ion data. In addition, this guide highlights best practices for making transactions idempotent and implementing uniqueness constraints.

Topics

Working with Amazon Ion

The following sections show how to use the Amazon Ion module to process Ion data.

Topics

Importing the Ion module

```
using Amazon.IonDotnet.Builders;
```

Creating Ion types

The following code example creates an Ion object from Ion text.

```
var ionText = "{GovId: \"TOYENC486FH\", FirstName: \"Brent\"}";
var ionObject = IonLoader.Default.Load(ionText).GetElementAt(0);

Console.WriteLine(ionObject.GetField("GovId").StringValue); //prints TOYENC486FH
Console.WriteLine(ionObject.GetField("FirstName").StringValue); //prints Brent
```

Getting an Ion binary dump

```
// 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()));
```

Getting an Ion text dump

```
// ionObject is an Ion struct
StringWriter sw = new StringWriter();
using (var writer = IonTextWriterBuilder.Build(sw))
{
    ionObject.WriteTo(writer);
    writer.Finish();
}

Console.WriteLine(sw.ToString());
```

For more information about working with Ion, see the Amazon Ion documentation on GitHub.

Importing the driver

The following code example imports the driver.

```
using Amazon.QLDB.Driver;
```

Instantiating the driver

The following code example creates an instance of the driver that connects to a specified ledger name.

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

CRUD operations

QLDB executes *create, read, update, and delete* (CRUD) operations as part of a transaction.

> **Warning**
> As a best practice, make your write transactions strictly idempotent.

Making transactions idempotent

We recommend that you make write transactions idempotent to avoid any unexpected side effects in the case of retries. A transaction is *idempotent* if it can be executed multiple times and produce identical results each time.

For example, consider a transaction that inserts a document into a table named `Person`. The transaction should first check whether or not the document already exists in the table. Without this check, the table might end up with duplicate documents.

Suppose that QLDB successfully commits the transaction on the server side, but the client times out while waiting for a response. If the transaction is not idempotent, the same document could be inserted more than once in the case of a retry.

Using indexes to limit concurrency conflicts

We highly recommend that you run statements with a `WHERE` predicate clause that filters on an indexed field or a document `id`. Without an index, QLDB needs to do a table scan, which can lead to more *optimistic concurrency control* (OCC) conflicts.

For more information about OCC, see Amazon QLDB concurrency model (p. 332).

Implicitly created transactions

The Amazon.QLDB.Driver.IQldbDriver.Execute method accepts a lambda function that receives an instance of Amazon.QLDB.Driver.TransactionExecutor, which you can use to run statements. The instance of `TransactionExecutor` wraps an implicitly created transaction.

You can run statements within the lambda function by using the `Execute` method of the transaction executor. The driver implicitly commits the transaction when the lambda function returns.

The following sections show how to run basic CRUD operations, specify custom retry logic, and implement uniqueness constraints.

Topics

Creating tables

```
driver.Execute(t => { t.Execute("CREATE TABLE Person"); });
```

Creating indexes

```
driver.Execute(t => { t.Execute("CREATE INDEX ON Person(GovId)"); });
```

Reading documents

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

driver.Execute(t =>
{
    var result = t.Execute("SELECT * FROM Person");

    foreach (var row in result)
    {
        Console.WriteLine(row.GetField("GovId").StringValue); //prints TOYENC486FH
        Console.WriteLine(row.GetField("FirstName").StringValue); //prints Brent
    }
});
```

Using query parameters

The following code example uses an Ion type query parameter.

```
IIonValue name = IonLoader.Default.Load("Brent");
driver.Execute(t =>
{
    var result = t.Execute("SELECT * FROM Person WHERE FirstName = ?", name);

    foreach (var row in result)
    {
        Console.WriteLine(row.GetField("GovId").StringValue); //prints TOYENC486FH
        Console.WriteLine(row.GetField("FirstName").StringValue); //prints Brent
    }
});
```

The following code example uses multiple query parameters.

```
IIonValue id = IonLoader.Default.Load("TOYENC486FH");
IIonValue firstName = IonLoader.Default.Load("Brent");

driver.Execute(t =>
{
    var result = t.Execute("SELECT * FROM Person WHERE GovId = ? AND FirstName = ?", id,
 firstName);

    foreach (var row in result)
    {
        Console.WriteLine(row.GetField("GovId").StringValue); //prints TOYENC486FH
        Console.WriteLine(row.GetField("FirstName").StringValue); //prints Brent
    }
});
```

The following code example uses a list of query parameters.

```
var ids = new IIonValue[] {
     IonLoader.Default.Load("TOYENC486FH"),
     IonLoader.Default.Load("ROEE1"),
     IonLoader.Default.Load("YH844")
};

driver.Execute(t =>
{
    t.Execute("SELECT * FROM Person WHERE GovId IN (?,?,?)", ids);
});
```

Note
When you run a query that doesn't filter on an indexed field, it results in a full table scan. In this example, we recommend having an index (p. 448) on the `GovId` field to optimize performance. Without an index on `GovId`, queries can have more latency and can also lead to more OCC conflict exceptions.

Inserting documents

The following code example inserts Ion data types.

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

// Check if a document with a GovId of TOYENC486FH exists
// This is critical to make this transaction idempotent
driver.Execute(t =>
{
    var id = IonLoader.Default.Load("TOYENC486FH");

    var result = t.Execute("SELECT * FROM Person WHERE GovId = ?", id);

    // Check if there is a record in the cursor
    if (result.Count() > 0)
    {
        return;
    }

    // Insert the document
    var doc = IonLoader.Default.Load("{GovId: \"TOYENC486FH\", FirstName: \"Brent\"}");
    t.Execute("INSERT INTO Person ?", doc);
});
```

This transaction inserts a document into the `Person` table. Before inserting, it first checks if the document already exists in the table. **This check makes the transaction idempotent in nature.** Even if you execute this transaction multiple times, it will not cause any unintended side effects.

Note
In this example, we recommend having an index on the `GovId` field to optimize performance. Without an index on `GovId`, statements can have more latency and can also lead to more OCC conflict exceptions.

Updating documents

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

driver.Execute(t =>
{
    var id = IonLoader.Default.Load("TOYENC486FH");
    var firstName = IonLoader.Default.Load("John");

    t.Execute("UPDATE Person SET FirstName = ?  WHERE GovId = ?", firstName, id);
});
```

Note
In this example, we recommend having an index on the `GovId` field to optimize performance. Without an index on `GovId`, statements can have more latency and can also lead to more OCC conflict exceptions.

Deleting documents

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

driver.Execute(t =>
{
    var id = IonLoader.Default.Load("TOYENC486FH");

    t.Execute("DELETE FROM Person WHERE GovId = ?", id);
});
```

Note
In this example, we recommend having an index on the `GovId` field to optimize performance. Without an index on `GovId`, statements can have more latency and can also lead to more OCC conflict exceptions.

Retry logic

The driver's `Execute` methods have a built-in retry mechanism that retries the transaction if a retryable exception occurs (such as `InvalidSessionException` or `OccConflictException`). The maximum number of retry attempts and the backoff strategy are configurable.

The default retry limit is `4`, and the default backoff strategy is ExponentBackoffStrategy. You can override the retry configuration per transaction by passing an instance of Amazon.QLDB.Driver.RetryPolicy to the `Execute` methods that take a customized `RetryPolicy`.

The following code example specifies retry logic with a custom retry limit.

```
driver.Execute(t => { t.Execute("SELECT * FROM Person"); },
 RetryPolicy.Builder().WithMaxRetries(2).Build());
```

The following code example specifies retry logic with a custom backoff strategy.

```
public void Retry()
{
    var driver = QldbDriver.Builder().WithLedger("vehicle-registration").Build();

    driver.Execute(t =>
    {
        var result = t.Execute("SELECT * FROM Person");
    },
    RetryPolicy.Builder()
        .WithBackoffStrategy(new FixedBackOff())
        .Build());
}

private class FixedBackOff : IBackoffStrategy
{
    public TimeSpan CalculateDelay(RetryPolicyContext retryPolicyContext)
    {
        return TimeSpan.FromMilliseconds(200);
    }
}
```

Implementing uniqueness constraints

QLDB does not currently support unique indexes. But it's easy to implement this behavior in your application.

Suppose that you want to implement a uniqueness constraint on the `GovId` field in the `Person` table. To do this, you can write a transaction that does the following:

1. Assert that the table has no existing documents with a specified `GovId`.
2. Insert the document if the assertion passes.

If a competing transaction concurrently passes the assertion, only one of the transactions will commit successfully. The other transaction will fail with an OCC conflict exception.

The following code example shows how to implement this uniqueness constraint logic.

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

driver.Execute(t =>
{
    var id = IonLoader.Default.Load("TOYENC486FH");

    var result = t.Execute("SELECT * FROM Person WHERE GovId = ?", id);

    // Check if there is a record in the cursor
    if (result.Count() > 0)
    {
        return;
    }

    // Insert the document
    var doc = IonLoader.Default.Load("{GovId: \"TOYENC486FH\", FirstName: \"Brent\"}");
    t.Execute("INSERT INTO Person ?", doc);
});
```

Note
In this example, we recommend having an index on the `GovId` field to optimize performance. Without an index on `GovId`, statements can have more latency and can also lead to more OCC conflict exceptions.

Amazon QLDB driver for Go (preview)

This is prerelease documentation for a feature in preview release. It is subject to change.

To work with data in your ledger, you can connect to Amazon QLDB from your Go application by using an AWS-provided driver. The following sections describe how to get started with the QLDB driver for Go.

Topics

Driver resources

For more information about the functionality supported by the Go driver, see the following resources:

- API reference
- Driver recommendations (p. 310)
- Common errors (p. 315)
- Driver source code (GitHub)
- Amazon Ion Cookbook

Prerequisites

Before you get started with the QLDB driver for Go, you must do the following:

1. Follow the AWS setup instructions in Accessing Amazon QLDB (p. 15). This includes signing up for AWS and getting an AWS access key for development.
2. (Optional) Install an integrated development environment (IDE) of your choice. For a list of commonly used IDEs from the Go ecosystem, see Editor plugins and IDEs on the Go website.
3. Download and install Go version 1.14 or later from the Go downloads site.
4. Configure your development environment for the AWS SDK for Go:

    - Set up your AWS credentials. We recommend that you create a shared credentials file.

      For instructions, see Specifying Credentials in the *AWS SDK for Go Developer Guide*.
    - Set your default AWS Region. To learn how, see Specifying the AWS Region.

      For a complete list of available Regions, see Amazon QLDB endpoints and quotas in the *AWS General Reference*.

Next, you can set up a basic sample application and run short code examples—or you can install the driver in an existing Go project.

- To install the QLDB driver and the AWS SDK for Go in an existing project, proceed to Installation (p. 192).
- To run short code examples that execute basic data transactions on a ledger, see the Quick start (preview) (p. 193) guide.

Installation

The QLDB driver for Go is open-sourced in the GitHub repository awslabs/amazon-qldb-driver-go.

To install the driver, enter the following `go get` command.

```
$ go get -u github.com/awslabs/amazon-qldb-driver-go/qldbdriver
```

The driver has dependencies on the AWS SDK for Go (`aws-sdk-go`) and Amazon Ion (p. 487) (`ion-go`). You must also install these packages as dependencies in your project.

```
$ go get -u github.com/aws/aws-sdk-go
```

```
$ go get -u github.com/amzn/ion-go/ion
```

For short code examples of how to run basic data transactions on a ledger, proceed to the Quick start (preview) (p. 193) guide.

Amazon QLDB driver for Go – Quick start (preview)

This is prerelease documentation for a feature in preview release. It is subject to change.

In this section, you learn how to set up a simple application using the Amazon QLDB driver for Go. This guide includes steps for installing the driver and short code examples of basic *create, read, update, and delete* (CRUD) operations.

Topics

Prerequisites

Before you get started, make sure that you do the following:

1. Complete the Prerequisites (p. 192) for the Go driver, if you haven't already done so. This includes signing up for AWS, getting an AWS access key for development, and installing Go.
2. Create a ledger named `quick-start`.

   To learn how to create a ledger, see Basic operations for Amazon QLDB ledgers (p. 382) or Step 1: Create a new ledger (p. 25) in *Getting Started with the Console*.

Step 1: Install the driver

To install the driver and its dependencies, enter the following `go get` commands on your terminal.

```
$ go get -u github.com/awslabs/amazon-qldb-driver-go/qldbdriver
$ go get -u github.com/aws/aws-sdk-go
$ go get -u github.com/amzn/ion-go/ion
```

The driver has dependencies on the AWS SDK for Go and Amazon Ion packages.

Step 2: Import the packages

Import the following AWS packages.

```
import (
    "context"

    "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/qldbdriver"
)
```

Step 3: Initialize the driver

Initialize an instance of the driver that connects to the ledger named `quick-start`.

```
awsSession := session.Must(session.NewSession(aws.NewConfig().WithRegion("us-east-1")))
qldbSession := qldbsession.New(awsSession)

driver := qldbdriver.New(
    "quick-start",
    qldbSession,
    func(options *qldbdriver.DriverOptions) {
        options.LoggerVerbosity = qldbdriver.LogInfo
    })

defer driver.Close(context.Background())
```

Note
In this code example, replace *`us-east-1`* with the AWS Region where you created your ledger.

Step 4: Create a table and index

The following code example shows how to run `CREATE TABLE` and `CREATE INDEX` statements.

```
_, 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 conflicts 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
})
```

This code creates a table named `People` and indexes for the `firstName` and `age` fields on that table. Indexes (p. 448) can improve your query performance and help to limit *optimistic concurrency control* (OCC) conflict exceptions.

Step 5: Insert a document

The following code examples show how to run an `INSERT` statement. QLDB supports the PartiQL (p. 426) query language (SQL compatible) and the Amazon Ion (p. 487) data format (superset of JSON).

Using literal PartiQL

The following code inserts a document into the `People` table using a string literal PartiQL statement.

```
_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    return txn.Execute("INSERT INTO People {'firstName': 'Jane', 'lastName': 'Doe', 'age':
 77}")
})
```

Using Ion data types

Similar to Go's built-in JSON package, you can marshal and unmarshal Go data types to Ion.

1. Suppose that you have the following Go structure named `Person`.

   ```
   type Person struct {
       FirstName string `ion:"firstName"`
       LastName  string `ion:"lastName"`
       Age       int    `ion:"age"`
   }
   ```

2. Create an instance of `Person`.

   ```
   hh := Person{"John", "Doe", 54}
   ```

   The driver marshals an Ion-encoded text representation of `hh` for you.

   Important
   For marshal and unmarshal to work properly, the field names of the Go data structure must be exported (first letter capitalized).

3. Pass the `hh` instance to the transaction's `Execute` method.

   ```
   _, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
    (interface{}, error) {
       return txn.Execute("INSERT INTO People ?", hh)
   })
   ```

   This example uses a question mark (`?`) as a variable placeholder to pass the document information to the statement. When you use placeholders, you must pass an Ion-encoded text value.

Step 6: Query the document

The following code example shows how to run a `SELECT` statement.

```
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
    ionBinary, err := result.Next(txn)
    if err != nil {
        return nil, err
    }

    temp := new(Person)
    err = ion.Unmarshal(ionBinary, temp)
    if err != nil {
        return nil, err
    }

    return *temp, nil
})

var person Person
person = p.(Person)
```

This example queries your document from the `People` table, assumes that the result is not empty, and returns that single document from the result.

Step 7: Update the document

The following code example shows how to run an `UPDATE` statement.

```
hh.Age += 10

_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    return txn.Execute("UPDATE People SET age = ? WHERE firstName = ?", hh.Age,
 hh.FirstName)
})
```

Step 8: Query the updated document

The following code example does a full table scan on `People` and returns all of the documents in the result set.

> **Tip**
> We don't recommend running table scans on large tables. The best practice is to run statements that filter on an indexed field or a document `id`. For more information, see Using indexes to limit OCC conflicts (p. 332) in the *Concurrency model* topic.

```
p, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    result, err := txn.Execute("SELECT firstName, lastName, age FROM People")
    if err != nil {
        return nil, err
    }

    var people []Person
    for result.HasNext() {
        ionBinary, err := result.Next(txn)
        if err != nil {
            return nil, err
        }

        temp := new(Person)
        err = ion.Unmarshal(ionBinary, temp)
        if err != nil {
            return nil, err
        }

        people = append(*people, temp)
    }
    return people, nil
})

var people []Person
people := p.([]Person)
```

Step 9: Drop the table

The following code example shows how to run a `DROP TABLE` statement.

```
_, err = driver.Execute(context.Background(), func(txn qldbdriver.Transaction)
 (interface{}, error) {
    return txn.Execute("DROP TABLE People")
})
```

Amazon QLDB driver for Node.js

To work with data in your ledger, you can connect to Amazon QLDB from your Node.js application by using an AWS-provided driver. The following sections describe how to get started with the QLDB driver for Node.js.

Topics

Driver resources

For more information about the functionality supported by the Node.js driver, see the following resources:

- API reference: 2.x, 1.x
- Driver recommendations (p. 310)
- Common errors (p. 315)
- Ion code examples (p. 489)
- Driver source code (GitHub)
- Sample application source code (GitHub)

Prerequisites

Before you get started with the QLDB driver for Node.js, you must do the following:

1. Follow the AWS setup instructions in Accessing Amazon QLDB (p. 15). This includes signing up for AWS and getting an AWS access key for development.
2. Install Node.js version 10.x or later from the Node.js downloads site.
3. Configure your development environment for the AWS SDK for JavaScript in Node.js:

   - Set up your AWS credentials. We recommend that you create a shared credentials file.

     For instructions, see Loading credentials in Node.js from the shared credentials file in the *AWS SDK for JavaScript Developer Guide*.
   - Set your default AWS Region. To learn how, see Setting the AWS Region.

     For a complete list of available Regions, see Amazon QLDB endpoints and quotas in the *AWS General Reference*.

Next, you can download the complete tutorial sample application—or you can install only the driver in an existing Node.js project and run some code examples.

- To install only the QLDB driver and the AWS SDK for JavaScript in Node.js in an existing project, proceed to Installation (driver only) (p. 198).
- To run short code examples that execute basic data transactions on a ledger, see the Quick start (p. 199) guide.
- To run more in-depth examples of both data and control plane operations in the complete tutorial sample application, see the Node.js tutorial (p. 205).

Installation (driver only)

QLDB supports the following driver versions and their Node.js dependencies.

Driver version	Node.js version	Status	Latest release date
1.x	10.x or later	Production release	June 5, 2020
2.x	10.x or later	Production release	August 27, 2020

To install the QLDB driver using npm (the Node.js package manager), enter the following command from your project root directory.

2.x

```
npm install amazon-qldb-driver-nodejs@2.0.0
```

1.x

```
npm install amazon-qldb-driver-nodejs@1.0.0
```

Installing the driver also installs TypeScript 3.5.x as a dependency.

In addition, the driver has peer dependencies on `aws-sdk` (AWS SDK for JavaScript), `ion-js` (Amazon Ion data format), and `jsbi` (pure JavaScript implementation of `BigInt`). You must also install these packages as dependencies in your project.

```
npm install aws-sdk
```

```
npm install ion-js@4.0.0
```

```
npm install jsbi@3.1.2
```

Using the driver to connect to a ledger

Then you can import the driver and use it to connect to a ledger. The following TypeScript code example shows how to create a driver instance for a specified ledger name and AWS Region.

2.x

```
import { QldbDriver, RetryConfig  } from "amazon-qldb-driver-nodejs";

const serviceConfigurationOptions = {
    region: "us-east-1"
};

const maxConcurrentTransactions: number = 10;
const retryLimit: number = 4;

//Use driver's default backoff function for this example (so, 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 { QldbDriver } from "amazon-qldb-driver-nodejs";

const serviceConfigurationOptions = {
    region: "us-east-1"
};

const qldbDriver: QldbDriver = new QldbDriver("testLedger",
 serviceConfigurationOptions);
qldbDriver.getTableNames().then(function(tableNames: string[]) {
    console.log(tableNames);
});
```

For short code examples of how to run basic data transactions on a ledger, proceed to the Quick start (p. 199) guide. For more in-depth examples that demonstrate both data and control plane operations in a full sample application, see the Node.js tutorial (p. 205).

Amazon QLDB driver for Node.js – Quick start

In this section, you learn how to set up a simple application using the Amazon QLDB driver for Node.js. This guide includes steps for installing the driver and short *JavaScript* code examples of basic *create,* *read, update, and delete* (CRUD) operations. For more in-depth examples that demonstrate these operations in a full sample application, see the Node.js tutorial (p. 205).

> **Note**
> Where applicable, some steps have different code examples for each supported major version of the QLDB driver for Node.js.

Topics

Prerequisites

Before you get started, make sure that you do the following:

1. Complete the Prerequisites (p. 198) for the Node.js driver, if you haven't already done so. This includes signing up for AWS, getting an AWS access key for development, and installing Node.js.
2. Create a ledger named `quick-start`.

   To learn how to create a ledger, see Basic operations for Amazon QLDB ledgers (p. 382) or Step 1: Create a new ledger (p. 25) in *Getting started with the console*.

Step 1: Set up your project

First, set up your Node.js project.

1. Create a folder for your application.

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

2. To initialize your project, enter the following `npm` command and answer the questions that are asked during the setup. You can use defaults for most of the questions.

```
$ npm init
```

3. Install the Amazon QLDB driver for Node.js.

   2.x

```
$ npm install amazon-qldb-driver-nodejs@2.0.0 --save
```

   1.x

```
$ npm install amazon-qldb-driver-nodejs@1.0.0 --save
```

4. Install the peer dependencies of the driver.

```
$ npm install aws-sdk --save
$ npm install ion-js@4.0.0 --save
$ npm install jsbi@3.1.2 --save
```

5. Create a new file named `app.js`.

   Then, incrementally add the code examples in the following sections to try some basic CRUD operations. Or, you can skip the step-by-step tutorial and instead run the complete application (p. 204).

Step 2: Initialize the driver

Initialize an instance of the driver that connects to the ledger named `quick-start`. Add the following code to your `app.js` file.

2.x

```
var qldb = require('amazon-qldb-driver-nodejs');

const main = function() {
  var serviceConfigurationOptions = {
    "region": "us-east-1"
  };
  var maxConcurrentTransactions = 10;
  var retryLimit = 4;

  //Use driver's default backoff function for this example (so, no second parameter
 provided to RetryConfig)
  var retryConfig = new qldb.RetryConfig(retryLimit);
  var driver = new qldb.QldbDriver("quick-start", serviceConfigurationOptions,
 maxConcurrentTransactions, retryConfig);
};
main();
```

Note

- In this code example, replace *`us-east-1`* with the AWS Region where you created your ledger.
- Version 2.x introduces the new optional parameter `RetryConfig` for initializing `QldbDriver`.
- For simplicity, the remaining code examples in this section use a driver with default settings, as specified in this code example for version 1.x. You can also use your own driver instance with a custom `RetryConfig` instead.

1.x

```
var qldb = require('amazon-qldb-driver-nodejs');

const main = function() {
  //Use default settings
  const driver = new qldb.QldbDriver("quick-start");
});
main();
```

Step 3: Create a table and an index

The following code examples show how to run `CREATE TABLE` and `CREATE INDEX` statements.

1. Add the following function that creates a table named `People`.

```
const createTable = function(driver, callback) {
  driver.executeLambda(async (txn) => {
    txn.execute("CREATE TABLE People");
  }).then(() => {
    setTimeout(callback, 3000);
  });
}
```

2. Add the following function that creates an index for the `firstName` field on the `People` table. Indexes (p. 448) can improve your query performance.

```
const createIndex = function(driver, callback) {
  driver.executeLambda(async (txn) => {
    txn.execute("CREATE INDEX ON People (firstName)");
  }).then(callback);
}
```

3. In the `main` function, add the following call to `createTable`. In this code example, you first call `createTable`, and then pass `createIndex` as a callback to that function call.

```
var qldb = require('amazon-qldb-driver-nodejs');

const main = function() {
  //Use default settings
  const driver = new qldb.QldbDriver("quick-start");

  console.log("Create table People");
  createTable(driver, function() {
    console.log("Create index on firstName");
    createIndex(driver, function() {
      driver.close();
    });
  });
});
main();
```

4. Run `app.js` to create the table and index.

```
$ node app.js
```

Step 4: Insert a document

The following code example shows how to run an `INSERT` statement.

1. Add the following function that inserts a document into the `People` table.

```
const insertDocument = function(driver, callback) {
  const person = {firstName: "John", lastName: "Doe", age: 42}
  driver.executeLambda((txn) => {
    txn.execute("INSERT INTO People ?", person);
  }).then(callback);
}
```

2. In the `main` function, remove the `createTable` call and add a call to `insertDocument`.

```
var qldb = require('amazon-qldb-driver-nodejs');

const main = function() {
  //Use default settings
  const driver = new qldb.QldbDriver("quick-start");

  console.log("Insert document");
  insertDocument(driver, function() {
    driver.close();
  });
});
main();
```

Step 5: Query the document

The following code example shows how to run a `SELECT` statement.

1. Add the following function that queries a document from the `People` table.

```
const fetchDocuments = function(driver, callback) {
  driver.executeLambda(async (txn) => {
    return txn.execute("SELECT firstName, age, lastName FROM People WHERE firstName
 = ?", "John")
  }).then((result) => {
    const resultList = result.getResultList();
    //Pretty print the result list
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    callback();
  });
}
```

2. In the `main` function, pass `fetchDocuments` as a callback to the `insertDocument` function call.

```
var qldb = require('amazon-qldb-driver-nodejs');

const main = function() {
  //Use default settings
  const driver = new qldb.QldbDriver("quick-start");

  console.log("Insert document");
  insertDocument(driver, function() {
    console.log("Fetch document");
    fetchDocuments(driver, function() {
      driver.close();
    });
  });
});
main();
```

Step 6: Update the document

The following code example shows how to run an `UPDATE` statement.

1. Add the following function that updates a document in the `People` table by changing `lastName` to `"Stiles"`.

```
const updateDocuments = function(driver, callback) {
  driver.executeLambda(async (txn) => {
    txn.execute("UPDATE People SET lastName = ? WHERE firstName = ?", "Stiles",
 "John");
  }).then(callback);
}
```

2. In the `main` function, pass `updateDocuments` as a callback to the `fetchDocuments` function call. Then, you can call `fetchDocuments` again to see the updated results.

```
var qldb = require('amazon-qldb-driver-nodejs');

const main = function() {
  //Use default settings
  const driver = new qldb.QldbDriver("quick-start");

  console.log("Insert document");
  insertDocument(driver, function() {
    console.log("Fetch document");
    fetchDocuments(driver, function() {
      console.log("Update document");
      updateDocuments(driver, function() {
        console.log("Fetch document after update");
        fetchDocuments(driver, function() {
          driver.close();
        });
      });
    });
  });
});
main();
```

3. Run `app.js` to insert, query, and update a document.

```
$ node app.js
```

Running the complete application

The following example code is the complete version of the `app.js` application. Instead of doing the previous steps individually, you can also run this code example end to end. This application demonstrates some basic CRUD operations on the ledger named `quick-start`.

> **Note**
> Before you run this code, make sure that you don't already have an active table named `People` in the `quick-start` ledger.

```
const qldb = require('amazon-qldb-driver-nodejs');

const createTable = function(driver, callback) {
  driver.executeLambda(async (txn) => {
    txn.execute("CREATE TABLE People");
  }).then(() => {
    setTimeout(callback, 3000);
  });
}

const createIndex = function(driver, callback) {
  driver.executeLambda(async (txn) => {
    txn.execute("CREATE INDEX ON People (firstName)");
  }).then(callback);
}

const insertDocument = function(driver, callback) {
  const person = {firstName: "John", lastName: "Doe", age: 42}
  driver.executeLambda((txn) => {
    txn.execute("INSERT INTO People ?", person);
  }).then(callback);
}

const fetchDocuments = function(driver, callback) {
  driver.executeLambda(async (txn) => {
    return txn.execute("SELECT firstName, age, lastName FROM People  WHERE firstName = ?",
 "John")
  }).then((result) => {
    const resultList = result.getResultList();
    console.log("The result List is ", JSON.stringify(resultList, null, 2));
    callback();
  });
}

const updateDocuments = function(driver, callback) {
  driver.executeLambda(async (txn) => {
    txn.execute("UPDATE People SET lastName= ?  WHERE firstName = ?", "Stiles", "John");
  }).then(callback);
}

const main = function() {
  //Use default settings
  const driver = new qldb.QldbDriver("quick-start");
  console.log("Create table People");
  createTable(driver, function() {
    console.log("Create index on firstName");
    createIndex(driver, function() {
      console.log("Insert document");
      insertDocument(driver, function() {
        console.log("Fetch document");
        fetchDocuments(driver, function() {
          console.log("Update document");
          updateDocuments(driver, function() {
            console.log("Fetch document after update");
            fetchDocuments(driver, function() {
              driver.close();
            })
          });
        });
      });
    });
  });
}
main();
```

To run the complete application, enter the following command.

```
$ node app.js
```

Amazon QLDB Node.js tutorial

In this tutorial, you use the Amazon QLDB driver with the AWS SDK for JavaScript in Node.js to create a QLDB ledger and populate it with sample data. The driver enables your application to interact with QLDB using the transactional data API. The AWS SDK for JavaScript in Node.js enables interaction with the QLDB control plane API.

The sample ledger that you create in this scenario is a department of motor vehicles (DMV) database that tracks the complete historical information about vehicle registrations. The following sections explain how to add vehicle registrations, modify them, and view the history of changes to those registrations. This guide also shows you how to verify a registration document cryptographically, and it concludes by cleaning up resources and deleting the sample ledger.

As you work through this tutorial, you can refer to the AWS SDK for JavaScript API Reference for control plane operations. For transactional data operations, you can refer to the QLDB Driver for Node.js API Reference.

Note
Where applicable, some tutorial steps have different commands or code examples for each supported major version of the QLDB driver for Node.js.

Topics

Installing the Amazon QLDB Node.js sample application

This section describes how to install and run the provided Amazon QLDB sample application for the step-by-step Node.js tutorial. The use case for this sample application is a department of motor vehicles (DMV) database that tracks the complete historical information about vehicle registrations.

The DMV sample application for Node.js is open-sourced in the GitHub repository aws-samples/amazon-qldb-dmv-sample-nodejs.

Prerequisites

Before you get started, make sure that you complete the Prerequisites (p. 198). This includes signing up for AWS, getting an AWS access key for development, and installing Node.js.

Installation

To install the sample application

1. Enter the following command to clone the sample application from GitHub.

   2.x

   ```
   git clone -b v2.0.0 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
   ```

The sample application packages the complete source code from this tutorial and its dependencies, including the Node.js driver and the AWS SDK for JavaScript in Node.js. This application is written in TypeScript.

2. Switch to the directory where the `amazon-qldb-dmv-sample-nodejs` package is cloned.

```
cd amazon-qldb-dmv-sample-nodejs
```

3. Do a clean install of the dependencies.

```
npm ci
```

4. Transpile the package.

```
npm run build
```

   The transpiled JavaScript files are written in the `./dist` directory.

5. Proceed to Step 1: Create a new ledger (p. 207) to start the tutorial and create a ledger.

Step 1: Create a new ledger

In this step, you create a new Amazon QLDB ledger named `vehicle-registration`.

To create a new ledger

1. Review the following file (`Constants.ts`), which contains constant values that are used by all of the other programs in this tutorial.

```
/*
 * 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. Use the following program (`CreateLedger.ts`) to create a ledger named `vehicle-registration`.

```
/*
 * 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.
 */
var 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

- In the `createLedger` call, you must specify a ledger name and a permissions mode. The only permissions mode that is currently supported for a ledger is `ALLOW_ALL`.
- When you create a ledger, *deletion protection* is enabled by default. This is a feature in QLDB that prevents ledgers from being deleted by any user. You have the option of disabling deletion protection on ledger creation using the QLDB API or the AWS Command Line Interface (AWS CLI).
- Optionally, you can also specify tags to attach to your ledger.

3. To run the transpiled program, enter the following command.

```
node dist/CreateLedger.js
```

To verify your connection to the new ledger, proceed to Step 2: Test connectivity to the ledger (p. 210).

Step 2: Test connectivity to the ledger

In this step, you verify that you can connect to the `vehicle-registration` ledger in Amazon QLDB using the transactional data API endpoint.

To test connectivity to the ledger

1. Use the following program (`ConnectToLedger.ts`) to create a data session connection to the `vehicle-registration` ledger.

   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.
 */
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();
}
```

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
To run data transactions on your ledger, you must create a QLDB driver object to connect to a specified ledger. This is a different client object than the `qldbClient` object that you used in the previous step (p. 207) to create the ledger. That previous client is only used to run the control plane API operations listed in the Amazon QLDB API reference (p. 500).

2. To run the transpiled program, enter the following command.

```
node dist/ConnectToLedger.js
```

To create tables in the `vehicle-registration` ledger, proceed to Step 3: Create tables, indexes, and sample data (p. 212).

Step 3: Create tables, indexes, and sample data

When your Amazon QLDB ledger is active and accepts connections, you can start creating tables for data about vehicles, their owners, and their registration information. After creating the tables and indexes, you can load them with data.

In this step, you create four tables in the `vehicle-registration` ledger:

- `VehicleRegistration`
- `Vehicle`

- `Person`
- `DriversLicense`

You also create the following indexes.

Table name	Field
`VehicleRegistration`	`VIN`
`VehicleRegistration`	`LicensePlateNumber`
`Vehicle`	`VIN`
`Person`	`GovId`
`DriversLicense`	`LicenseNumber`
`DriversLicense`	`PersonId`

When inserting sample data, you first insert documents into the `Person` table. Then, you use the system-assigned `id` from each `Person` document to populate the corresponding fields in the appropriate `VehicleRegistration` and `DriversLicense` documents.

> **Tip**
> As a best practice, use a document's system-assigned `id` as a foreign key. While you can define fields that are intended to be unique identifiers (for example, a vehicle's VIN), the true unique identifier of a document is its `id`. This field is included in the document's metadata, which you can query in the *committed view* (the system-defined view of a table).
> For more information about views in QLDB, see Core concepts (p. 7). To learn more about metadata, see Querying document metadata (p. 323).

To create tables and indexes

1. Use the following program (`CreateTable.ts`) to create the previously mentioned tables.

```
/*
 * 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.
 */
var 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
This program demonstrates how to use the `executeLambda` function in a QLDB driver instance. In this example, you run multiple `CREATE TABLE` PartiQL statements with a single lambda expression.
This execute function implicitly starts a transaction, runs all of the statements in the lambda, and then auto-commits the transaction.

2. To run the transpiled program, enter the following command.

```
node dist/CreateTable.js
```

3. Use the following program (`CreateIndex.ts`) to create indexes on the tables, as previously described.

```
/*
 * 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.
 */
var 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. To run the transpiled program, enter the following command.

```
node dist/CreateIndex.js
```

To load data into the tables

1. Review the following `.ts` files.

    1. `SampleData.ts` — Contains the sample data that you insert into the `vehicle-registration` tables.

```
/*
 * 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` — A utility module that imports from the `ion-js` package to provide helper functions that convert, parse, and print Amazon Ion (p. 487) data.

```
/*
 * 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 covert 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
The `getDocumentId` function runs a query that returns system-assigned document IDs from a table. To learn more, see Using the BY clause to query document ID (p. 325).

2. Use the following program (`InsertDocument.ts`) to insert the sample data into your tables.

```
/*
 * 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} ?`;
    let 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.
 */
var 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

- This program demonstrates how to call the `execute` function with parameterized values. You can pass data parameters in addition to the PartiQL statement that you want to run. Use a question mark (`?`) as a variable placeholder in your statement string.
- If an `INSERT` statement succeeds, it returns the `id` of each inserted document.

3. To run the transpiled program, enter the following command.

```
node dist/InsertDocument.js
```

Next, you can use `SELECT` statements to read data from the tables in the `vehicle-registration` ledger. Proceed to Step 4: Query the tables in a ledger (p. 224).

Step 4: Query the tables in a ledger

After creating tables in an Amazon QLDB ledger and loading them with data, you can run queries to review the vehicle registration data that you just inserted. QLDB uses PartiQL (p. 426) as its query language and Amazon Ion (p. 487) as its document-oriented data model.

PartiQL is an open source, SQL-compatible query language that has been extended to work with Ion. With PartiQL, you can insert, query, and manage your data with familiar SQL operators. Amazon Ion is a superset of JSON. Ion is an open source, document-based data format that gives you the flexibility of storing and processing structured, semistructured, and nested data.

In this step, you use `SELECT` statements to read data from the tables in the `vehicle-registration` ledger.

To query the tables

1. Use the following program (`ScanTable.ts`) to scan all the documents in all the tables in your ledger.

```
/*
 * 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 { log } from "./qldb/LogUtil";

/**
 * Pretty print Ion values in the provided result list.
 * @param resultList The result list containing Ion values to pretty print.
 */
export function prettyPrintResultList(resultList: dom.Value[]): void {
    log(JSON.stringify(resultList, null, 2));
}

/**
 * Scan for all the documents in a table.
 * @param txn The {@linkcode TransactionExecutor} for lambda execute.
 * @param tableName The name of the table to operate on.
 * @returns Promise which fulfills with a {@linkcode Result} object.
 */
export async function scanTableForDocuments(txn: TransactionExecutor, tableName:
 string): Promise<Result> {
    log(`Scanning ${tableName}...`);
    const query: string = `SELECT * FROM ${tableName}`;
    return await txn.execute(query).then((result: Result) => {
        return result;
    });
}

/**
 * Scan for all the documents in a table.
 * @returns Promise which fulfills with void.
 */
var main = async function(): Promise<void> {
    try {
        const qldbDriver: QldbDriver = getQldbDriver();
        await qldbDriver.getTableNames().then(async (listOfTables: string[]) => {
            for (const tableName of listOfTables) {
                await qldbDriver.executeLambda(async (txn: TransactionExecutor) => {
                    const result: Result = await scanTableForDocuments(txn, tableName);
                    prettyPrintResultList(result.getResultList());
                });
            }
        });
    } catch (e) {
        log(`Error displaying documents: ${e}`);
    }
}

if (require.main === module) {
    main();
}
```

2. To run the transpiled program, enter the following command.

```
node dist/ScanTable.js
```

3. Use the following program (`FindVehicles.ts`) to query all vehicles registered under a person in your ledger.

```
/*
 * 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.
 */
var 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
First, this program queries the `Person` table for the document with `GovId LEWISR261LL` to get its `id` metadata field.
Then, it uses this document `id` as a foreign key to query the `VehicleRegistration` table by `PrimaryOwner.PersonId`. It also joins `VehicleRegistration` with the `Vehicle` table on the `VIN` field.

4. To run the transpiled program, enter the following command.

```
node dist/FindVehicles.js
```

To learn about modifying documents in the tables in the `vehicle-registration` ledger, see Step 5: Modify documents in a ledger (p. 227).

Step 5: Modify documents in a ledger

Now that you have data to work with, you can start making changes to documents in the `vehicle-registration` ledger in Amazon QLDB. In this step, the following code examples demonstrate how to run data manipulation language (DML) statements. These statements update the primary owner of one vehicle and add a secondary owner to another vehicle.

To modify documents

1. Use the following program (`TransferVehicleOwnership.ts`) to update the primary owner of the vehicle with VIN `1N4AL11D75C109151` in your ledger.

```
/*
 * 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.
 */
var 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. To run the transpiled program, enter the following command.

```
node dist/TransferVehicleOwnership.js
```

3. Use the following program (`AddSecondaryOwner.ts`) to add a secondary owner to the vehicle with VIN `KM8SRDHF6EU074761` in your ledger.

```
/*
 * 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: ${vin}`);
    const query: string =
        `FROM VehicleRegistration AS v WHERE v.VIN = '${vin}' INSERT INTO
 v.Owners.SecondaryOwners VALUE ?`;

    let personToInsert = {PersonId: secondaryOwnerId};
    await txn.execute(query, 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.
 */
var 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. To run the transpiled program, enter the following command.

```
node dist/AddSecondaryOwner.js
```

To review these changes in the `vehicle-registration` ledger, see Step 6: View the revision history for a document (p. 232).

Step 6: View the revision history for a document

After modifying the registration data for a vehicle in the previous step (p. 227), you can query the history of all its registered owners and any other updated fields. In this step, you query the revision history of a document in the `VehicleRegistration` table in your `vehicle-registration` ledger.

To view the revision history

1. Use the following program (`QueryHistory.ts`) to query the revision history of the `VehicleRegistration` document with VIN `1N4AL11D75C109151`.

```
/*
 * 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();
    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.
 */
var 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

- You can view the revision history of a document by querying the built-in History function (p. 326) in the following syntax.

```
SELECT * FROM history( table [, `start-time`, `end-time`] ) AS h
WHERE h.metadata.id = 'id'
```

- Best practice is to qualify a history query with a document `id`. This helps to avoid inefficient queries.
- The `start-time` and `end-time` are both optional. They are Amazon Ion literal values that can be denoted with backticks (`` `...` ``). For more information, see Querying Ion with PartiQL (p. 444).

2. To run the transpiled program, enter the following command.

```
node dist/QueryHistory.js
```

To cryptographically verify a document revision in the `vehicle-registration` ledger, proceed to Step 7: Verify a document in a ledger (p. 234).

Step 7: Verify a document in a ledger

With Amazon QLDB, you can efficiently verify the integrity of a document in your ledger's journal by using cryptographic hashing with SHA-256. To learn more about how verification and cryptographic hashing work in QLDB, see Data verification in Amazon QLDB (p. 338).

In this step, you verify a document revision in the `VehicleRegistration` table in your `vehicle-registration` ledger. First, you request a digest, which is returned as an output file and acts as a signature of your ledger's entire change history. Then, you request a proof for the revision relative to that digest. Using this proof, the integrity of your revision is verified if all validation checks pass.

To verify a document revision

1. Review the following `.ts` files, which contain the QLDB objects that are required for verification.

   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 {
    let 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[] {
    let 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. Use two `.ts` programs (`GetDigest.ts` and `GetRevision.ts`) to perform the following steps:

   - Request a new digest from the `vehicle-registration` ledger.
   - Request a proof for each revision of the document with VIN `1N4AL11D75C109151` from the `VehicleRegistration` table.
   - Verify the revisions using the returned digest and proof by recalculating the digest.

   The `GetDigest.ts` program contains the following code.

```
/*
 * 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.
 */
var 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
Use the `getDigest` function to request a digest that covers the current *tip* of the journal in your ledger. The tip of the journal refers to the latest committed block as of the time that QLDB receives your request.

The `GetRevision.ts` program contains the following code.

```
/*
 * 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.
 */
var 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
After the `getRevision` function returns a proof for the specified document revision, this program uses a client-side API to verify that revision.

3. To run the transpiled program, enter the following command.

```
node dist/GetRevision.js
```

If you no longer need to use the `vehicle-registration` ledger, proceed to Step 8 (optional): Clean up resources (p. 243).

Step 8 (optional): Clean up resources

You can continue using the `vehicle-registration` ledger. However, if you no longer need it, you should delete it.

To delete the ledger

1. Use the following program (`DeleteLedger.ts`) to delete your `vehicle-registration` ledger and all of its contents.

```
/*
 * 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 { 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
    };
    while (true) {
        try {
            await qldbClient.describeLedger(request).promise();
            log("The ledger is still being deleted. Please wait...");
            await sleep(LEDGER_DELETION_POLL_PERIOD_MS);
        } catch (e) {
            if (isResourceNotFoundException(e)) {
                log("Success. Ledger is deleted.");
                break;
            } else {
                throw e;
            }
        }
    }
}

/**
 * Delete a ledger.
 * @returns Promise which fulfills with void.
 */
var 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
If deletion protection is enabled for your ledger, you must first disable it before you can delete the ledger using the QLDB API.

2. To run the transpiled program, enter the following command.

```
node dist/DeleteLedger.js
```

Amazon QLDB driver for Python

To work with data in your ledger, you can connect to Amazon QLDB from your Python application by using an AWS-provided driver. The following sections describe how to get started with the QLDB driver for Python.

Topics

Driver resources

For more information about the functionality supported by the Python driver, see the following resources:

- API reference: 3.x, 2.x
- Driver recommendations (p. 310)
- Common errors (p. 315)
- Ion code examples (p. 489)
- Driver source code (GitHub)
- Sample application source code (GitHub)

Prerequisites

Before you get started with the QLDB driver for Python, you must do the following:

1. Follow the AWS setup instructions in Accessing Amazon QLDB (p. 15). This includes signing up for AWS and getting an AWS access key for development.
2. Install Python version 3.4 or later from the Python downloads site.
3. Set up your AWS credentials and your default AWS Region. For instructions, see Quickstart in the AWS SDK for Python (Boto3) documentation.

   For a complete list of available Regions, see Amazon QLDB endpoints and quotas in the *AWS General Reference*.

Next, you can download the complete tutorial sample application—or you can install only the driver in an existing Python project.

- To run the complete tutorial with the QLDB sample application, see the Python tutorial (p. 254).
- To install only the QLDB driver and the AWS SDK for Python (Boto3) in an existing project, proceed to Installation (driver only) (p. 245).

Installation (driver only)

QLDB supports the following driver versions and their Python dependencies.

Driver version	Python version	Status	Latest release date
2.x	3.4 or later	Production release	May 7, 2020
3.x	3.6 or later	Production release	August 20, 2020

To install the QLDB driver from PyPI using `pip` (a package manager for Python), enter the following at the command line.

3.x

```
pip install pyqldb==3.0.0
```

2.x

```
pip install pyqldb==2.0.2
```

Installing the driver also installs its dependencies, including the AWS SDK for Python (Boto3) and Amazon Ion (p. 487) packages.

Using the driver to connect to a ledger

Then you can import the driver and use it to connect to a ledger. The following Python code example shows how to create a session for a specified ledger name.

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)
```

For short code examples of how to run basic data transactions on a ledger, proceed to the Cookbook (p. 246) reference. For more in-depth examples that demonstrate both data and control plane operations in a full sample application, see the Python tutorial (p. 254).

Amazon QLDB driver for Python – Cookbook

This section is a reference guide for common use cases of the Amazon QLDB driver for Python. It provides Python code examples that show how to use the driver to run basic *create, read, update, and delete* (CRUD) operations. It also includes code examples for processing Amazon Ion data. In addition, this guide highlights best practices for making transactions idempotent and implementing uniqueness constraints.

Note
Where applicable, some use cases have different code examples for each supported major version of the QLDB driver for Python.

Topics

Working with Amazon Ion

The following sections show how to use the Amazon Ion module to process Ion data.

Topics

Importing the Ion module

```
import amazon.ion.simpleion as simpleion
```

Creating Ion types

The following code example creates an Ion object from Ion text.

```
ion_text = '{GovId: "TOYENC486FH", FirstName: "Brent"}'
ion_obj = simpleion.loads(ion_text)

print(ion_obj['GovId']) # prints TOYENC486FH
print(ion_obj['Name']) # prints Brent
```

The following code example creates an Ion object from a Python `dict`.

```
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
```

Getting an Ion binary dump

```
# 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'
```

Getting an Ion text dump

```
# ion_obj is an Ion struct
```

```
print(simpleion.dumps(ion_obj, binary=False)) # prints $ion_1_0
 {GovId:'TOYENC486FH',FirstName:"Brent"}
```

For more information about working with Ion, see the Amazon Ion documentation on GitHub.

Importing the driver

The following code example imports the driver.

3.x

```
from pyqldb.driver.qldb_driver import QldbDriver
```

2.x

```
from pyqldb.driver.pooled_qldb_driver import PooledQldbDriver
```

Instantiating the driver

The following code example creates an instance of the driver that connects to a specified ledger name.

3.x

```
qldb_driver = QldbDriver(ledger_name='vehicle-registration')
```

2.x

```
qldb_driver = PooledQldbDriver(ledger_name='vehicle-registration')
```

CRUD operations

QLDB executes *create, read, update, and delete* (CRUD) operations as part of a transaction.

> **Warning**
> As a best practice, make your write transactions strictly idempotent.

Making transactions idempotent

We recommend that you make write transactions idempotent to avoid any unexpected side effects in the case of retries. A transaction is *idempotent* if it can be executed multiple times and produce identical results each time.

For example, consider a transaction that inserts a document into a table named `Person`. The transaction should first check whether or not the document already exists in the table. Without this check, the table might end up with duplicate documents.

Suppose that QLDB successfully commits the transaction on the server side, but the client times out while waiting for a response. If the transaction is not idempotent, the same document could be inserted more than once in the case of a retry.

Using indexes to limit concurrency conflicts

We highly recommend that you run statements with a `WHERE` predicate clause that filters on an indexed field or a document `id`. Without an index, QLDB needs to do a table scan, which can lead to more *optimistic concurrency control* (OCC) conflicts.

For more information about OCC, see Amazon QLDB concurrency model (p. 332).

Implicitly created transactions

The pyqldb.driver.qldb_driver.execute_lambda method accepts a lambda function that receives an instance of pyqldb.execution.executor.Executor, which you can use to run statements. The instance of `Executor` wraps an implicitly created transaction.

You can run statements within the lambda function by using the execute_statement method of the transaction executor. The driver implicitly commits the transaction when the lambda function returns.

> **Note**
> The `execute_statement` method supports both Amazon Ion types and Python native types. If you pass a Python native type as an argument to `execute_statement`, the driver converts it to an Ion type using the `amazon.ion.simpleion` module (provided that conversion for the given Python data type is supported). For supported data types and conversion rules, see the simpleion source code.

The following sections show how to run basic CRUD operations, specify custom retry logic, and implement uniqueness constraints.

Topics

Creating tables

```
def create_table(transaction_executor):
    transaction_executor.execute_statement("Create TABLE Person")

qldb_driver.execute_lambda(lambda executor: create_table(executor))
```

Creating indexes

```
def create_index(transaction_executor):
    transaction_executor.execute_statement("CREATE INDEX ON Person(GovId)")

qldb_driver.execute_lambda(lambda executor: create_index(executor))
```

Reading documents

```
# 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")

    for doc in cursor:
        print(doc["GovId"]) # prints TOYENC486FH
        print(doc["FirstName"]) # prints Brent

qldb_driver.execute_lambda(lambda executor: read_documents(executor))
```

Using query parameters

The following code example uses a native type query parameter.

```
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId = ?",
 'TOYENC486FH')
```

The following code example uses an Ion type query parameter.

```
name_with_annotation = ion.loads("LegalName::Brent")
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE FirstName = ?",
 name_with_annotation)
```

The following code example uses multiple query parameters.

```
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId = ? AND
 FirstName = ?", 'TOYENC486FH', "Brent")
```

The following code example uses a list of query parameters.

```
gov_ids = ['TOYENC486FH','ROEE1','YH844']
cursor = transaction_executor.execute_statement("SELECT * FROM Person WHERE GovId IN
 (?,?,?)", *gov_ids)
```

Note
When you run a query that doesn't filter on an indexed field, it results in a full table scan. In this example, we recommend having an index (p. 448) on the `GovId` field to optimize performance. Without an index on `GovId`, queries can have more latency and can also lead to more OCC conflict exceptions.

Inserting documents

The following code example inserts native data types.

```
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 x: insert_documents(x, doc_1))
```

The following code example inserts Ion data types.

```
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 x: insert_documents(x, ion_doc_1))
```

This transaction inserts a document into the `Person` table. Before inserting, it first checks if the document already exists in the table. **This check makes the transaction idempotent in nature.** Even if you execute this transaction multiple times, it will not cause any unintended side effects.

Note
In this example, we recommend having an index on the `GovId` field to optimize performance. Without an index on `GovId`, statements can have more latency and can also lead to more OCC conflict exceptions.

Updating documents

The following code example uses native data types.

```
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 x: update_documents(x, gov_id, name))
```

The following code example uses Ion data types.

```
def update_documents(transaction_executor, gov_id, name):
    cursor = transaction_executor.execute_statement("UPDATE Person SET FirstName = ? WHERE
 GovId = ?", name, gov_id)

# ion datatypes
gov_id = simpleion.loads('TOYENC486FH')
name = simpleion.loads(simpleion.dumps('John'))

qldb_driver.execute_lambda(lambda x: update_documents(x, gov_id, name))
```

Note

In this example, we recommend having an index on the `GovId` field to optimize performance. Without an index on `GovId`, statements can have more latency and can also lead to more OCC conflict exceptions.

Deleting documents

The following code example uses native data types.

```
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 x: delete_documents(x, gov_id))
```

The following code example uses Ion data types.

```
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 x: delete_documents(x, gov_id))
```

Note

In this example, we recommend having an index on the `GovId` field to optimize performance. Without an index on `GovId`, statements can have more latency and can also lead to more OCC conflict exceptions.

Retry logic

The driver's `execute_lambda` method has a built-in retry mechanism that retries the transaction if a retryable exception occurs (such as timeouts or OCC conflicts).

3.x

The maximum number of retry attempts and the backoff strategy are configurable.

The default retry limit is `4`, and the default backoff strategy is Exponential Backoff and Jitter with a base of `10` milliseconds. You can set the retry configuration per driver instance and also per transaction by using an instance of pyqldb.config.retry_config.RetryConfig.

The following code example specifies retry logic with a custom retry limit and a custom backoff strategy for an instance of the driver.

```
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)
```

The following code example specifies retry logic with a custom retry limit and a custom backoff strategy for a particular lambda execution. This configuration for `execute_lambda` overrides the retry logic that is set for the driver instance.

```
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

The maximum number of retry attempts is configurable. You can configure the retry limit by setting the `retry_limit` property when initializing `PooledQldbDriver`.

The default retry limit is `4`.

Implementing uniqueness constraints

QLDB does not currently support unique indexes. But it's easy to implement this behavior in your application.

Suppose that you want to implement a uniqueness constraint on the `GovId` field in the `Person` table. To do this, you can write a transaction that does the following:

1. Assert that the table has no existing documents with a specified `GovId`.
2. Insert the document if the assertion passes.

If a competing transaction concurrently passes the assertion, only one of the transactions will commit successfully. The other transaction will fail with an OCC conflict exception.

The following code example shows how to implement this uniqueness constraint logic.

```
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 x: insert_documents(x, gov_id, document))
```

Note

In this example, we recommend having an index on the `GovId` field to optimize performance. Without an index on `GovId`, statements can have more latency and can also lead to more OCC conflict exceptions.

Amazon QLDB Python tutorial

In this tutorial, you use the Amazon QLDB driver with the AWS SDK for Python (Boto3) to create a QLDB ledger and populate it with sample data. The driver enables your application to interact with QLDB using the transactional data API. The AWS SDK for Python (Boto3) enables interaction with the QLDB control plane API.

The sample ledger that you create in this scenario is a department of motor vehicles (DMV) database that tracks the complete historical information about vehicle registrations. The following sections explain how to add vehicle registrations, modify them, and view the history of changes to those registrations. This guide also shows you how to verify a registration document cryptographically, and it concludes by cleaning up resources and deleting the sample ledger.

As you work through this tutorial, you can refer to QLDB low-level client in the *AWS SDK for Python (Boto3) API Reference* for control plane operations. For transactional data operations, you can refer to the QLDB Driver for Python API Reference.

Note

Where applicable, some tutorial steps have different commands or code examples for each supported major version of the QLDB driver for Python.

Topics

Installing the Amazon QLDB Python sample application

This section describes how to install and run the provided Amazon QLDB sample application for the step-by-step Python tutorial. The use case for this sample application is a department of motor vehicles (DMV) database that tracks the complete historical information about vehicle registrations.

The DMV sample application for Python is open-sourced in the GitHub repository aws-samples/amazon-qldb-dmv-sample-python.

Prerequisites

Before you get started, make sure that you complete the Prerequisites (p. 245). This includes signing up for AWS, getting an AWS access key for development, and installing Python.

Installation

To install the sample application

1. Enter the following `pip` command to clone and install the sample application from GitHub.

3.x

```
pip install git+https://github.com/aws-samples/amazon-qldb-dmv-sample-
python.git@v2.0.0
```

2.x

```
pip install git+https://github.com/aws-samples/amazon-qldb-dmv-sample-
python.git@v1.0.0
```

The sample application packages the complete source code from this tutorial and its dependencies, including the Python driver and the AWS SDK for Python (Boto3).

2. Before you start running code at the command line, switch your current working directory to the location where the `pyqldbsamples` package is installed. Enter the following command.

```
cd $(python -c "import pyqldbsamples; print(pyqldbsamples.__path__[0])")
```

3. Proceed to Step 1: Create a new ledger (p. 255) to start the tutorial and create a ledger.

Step 1: Create a new ledger

In this step, you create a new Amazon QLDB ledger named `vehicle-registration`.

To create a new ledger

1. Review the following file (`constants.py`), which contains constant values that are used by all of the other programs in this tutorial.

```
# 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. Use the following program (`create_ledger.py`) to create a ledger named `vehicle-registration`.

```
# 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)


if __name__ == '__main__':
    """
    Create a ledger and wait for it to be active.
    """
    try:
        create_ledger(Constants.LEDGER_NAME)
        wait_for_active(Constants.LEDGER_NAME)
    except Exception as e:
        logger.exception('Unable to create the ledger!')
        raise e
```

Note

- In the `create_ledger` call, you must specify a ledger name and a permissions mode. The only permissions mode that is currently supported for a ledger is `ALLOW_ALL`.
- When you create a ledger, *deletion protection* is enabled by default. This is a feature in QLDB that prevents ledgers from being deleted by any user. You have the option of disabling deletion protection on ledger creation using the QLDB API or the AWS Command Line Interface (AWS CLI).
- Optionally, you can also specify tags to attach to your ledger.

3. To run the program, enter the following command.

```
python create_ledger.py
```

To verify your connection to the new ledger, proceed to Step 2: Test connectivity to the ledger (p. 257).

Step 2: Test connectivity to the ledger

In this step, you verify that you can connect to the `vehicle-registration` ledger in Amazon QLDB using the transactional data API endpoint.

To test connectivity to the ledger

1. Use the following program (`connect_to_ledger.py`) to create a data session connection to the `vehicle-registration` ledger.

   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


if __name__ == '__main__':
    """
    Connect to a given ledger using default settings.
    """
    try:
        with create_qldb_driver() as driver:
            logger.info('Listing table names ')
            for table in driver.list_tables():
                logger.info(table)
    except ClientError:
        logger.exception('Unable to list tables.')
```

Note

- To run data transactions on your ledger, you must create a QLDB driver object to connect to a specific ledger. This is a different client object than the `qldb_client` object that you used in the previous step to create the ledger. That previous client is only used to run the control plane API operations listed in the Amazon QLDB API reference (p. 500).
- You must specify a ledger name when you create this driver object.

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

- To run data transactions on your ledger, you must create a QLDB driver object to connect to a specific ledger. This is a different client object than the `qldb_client` object that you used in the previous step to create the ledger. That previous client is only used to run the control plane API operations listed in the Amazon QLDB API reference (p. 500).
- First, create a pooled QLDB driver object. You must specify a ledger name when you create this driver.
- Then, you can create sessions from this pooled driver object.

2. To run the program, enter the following command.

```
python connect_to_ledger.py
```

To create tables in the `vehicle-registration` ledger, proceed to Step 3: Create tables, indexes, and sample data (p. 261).

Step 3: Create tables, indexes, and sample data

When your Amazon QLDB ledger is active and accepts connections, you can start creating tables for data about vehicles, their owners, and their registration information. After creating the tables and indexes, you can load them with data.

In this step, you create four tables in the `vehicle-registration` ledger:

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

You also create the following indexes.

Table name	Field
`VehicleRegistration`	`VIN`
`VehicleRegistration`	`LicensePlateNumber`
`Vehicle`	`VIN`
`Person`	`GovId`
`DriversLicense`	`LicenseNumber`
`DriversLicense`	`PersonId`

When inserting sample data, you first insert documents into the `Person` table. Then, you use the system-assigned `id` from each `Person` document to populate the corresponding fields in the appropriate `VehicleRegistration` and `DriversLicense` documents.

Tip
As a best practice, use a document's system-assigned `id` as a foreign key. While you can define fields that are intended to be unique identifiers (for example, a vehicle's VIN), the true unique identifier of a document is its `id`. This field is included in the document's metadata, which you can query in the *committed view* (the system-defined view of a table).
For more information about views in QLDB, see Core concepts (p. 7). To learn more about metadata, see Querying document metadata (p. 323).

To create tables and indexes

1. Use the following program (`create_table.py`) to create the previously mentioned tables.

   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(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_driver() as qldb_driver:
            qldb_driver.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.')
```

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
This program demonstrates how to use the `execute_lambda` function. In this example, you run multiple `CREATE TABLE` PartiQL statements with a single lambda expression. This execute function implicitly starts a transaction, runs all of the statements in the lambda, and then auto-commits the transaction.

2. To run the program, enter the following command.

```
python create_table.py
```

3. Use the following program (`create_index.py`) to create indexes on the tables, as previously described.

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(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_driver() as driver:
            driver.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.')
```

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. To run the program, enter the following command.

```
python create_index.py
```

To load data into the tables

1. Review the following file (`sample_data.py`), which represents the sample data that you insert into the `vehicle-registration` tables. This file also imports from the `amazon.ion` package to provide helper functions that convert, parse, and print Amazon Ion (p. 487) data.

```
# 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,
 value)
    cursor = transaction_executor.execute_statement(query)
    list_of_ids = map(lambda table: table.get('id'), cursor)
    return list_of_ids


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.stream_cursor.StreamCursor`
    :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.
        logger.info(dumps(row, binary=False, indent='  ', omit_version_marker=True))
        result_counter += 1
    return result_counter
```

Note

The `get_document_ids` function runs a query that returns system-assigned document IDs from a table. To learn more, see Using the BY clause to query document ID (p. 325).

2. Use the following program (`insert_document.py`) to insert the sample data into your tables.

   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(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_driver() 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.
            driver.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.')
```

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

- This program demonstrates how to call the `execute_statement` function with parameterized values. You can pass data parameters in addition to the PartiQL statement that you want to run. Use a question mark (`?`) as a variable placeholder in your statement string.
- If an `INSERT` statement succeeds, it returns the `id` of each inserted document.

3. To run the program, enter the following command.

```
python insert_document.py
```

Next, you can use `SELECT` statements to read data from the tables in the `vehicle-registration` ledger. Proceed to Step 4: Query the tables in a ledger (p. 276).

Step 4: Query the tables in a ledger

After creating tables in an Amazon QLDB ledger and loading them with data, you can run queries to review the vehicle registration data that you just inserted. QLDB uses PartiQL (p. 426) as its query language and Amazon Ion (p. 487) as its document-oriented data model.

PartiQL is an open source, SQL-compatible query language that has been extended to work with Ion. With PartiQL, you can insert, query, and manage your data with familiar SQL operators. Amazon Ion is a superset of JSON. Ion is an open source, document-based data format that gives you the flexibility of storing and processing structured, semistructured, and nested data.

In this step, you use `SELECT` statements to read data from the tables in the `vehicle-registration` ledger.

To query the tables

1. Use the following program (`scan_table.py`) to scan all documents in all tables in your ledger.

   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 print_result
from pyqldbsamples.connect_to_ledger import create_qldb_driver

logger = getLogger(__name__)
basicConfig(level=INFO)


def scan_table(transaction_executor, table_name):
    """
    Scan for all the documents in a 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 name of the table to operate on.

    :rtype: :py:class:`pyqldb.cursor.stream_cursor.StreamCursor`
    :return: Cursor on the result set of a statement query.
    """
    logger.info('Scanning {}...'.format(table_name))
    query = 'SELECT * FROM {}'.format(table_name)
    return transaction_executor.execute_statement(query)


if __name__ == '__main__':
    """
    Scan for all the documents in a table.
    """
    try:
        with create_qldb_driver() as driver:
            # Scan all the tables and print their documents.
            tables = driver.list_tables()
            for table in tables:
                cursor = driver.execute_lambda(
                    lambda executor: scan_table(executor, table))
                logger.info('Scan successful!')
                print_result(cursor)
    except Exception:
        logger.exception('Unable to scan tables.')
```

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 print_result
from pyqldbsamples.connect_to_ledger import create_qldb_session

logger = getLogger(__name__)
basicConfig(level=INFO)


def scan_table(transaction_executor, table_name):
    """
    Scan for all the documents in a 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 name of the table to operate on.

    :rtype: :py:class:`pyqldb.cursor.stream_cursor.StreamCursor`
    :return: Cursor on the result set of a statement query.
    """
    logger.info('Scanning {}...'.format(table_name))
    query = 'SELECT * FROM {}'.format(table_name)
    return transaction_executor.execute_statement(query)


if __name__ == '__main__':
    """
    Scan for all the documents in a table.
    """
    try:
        with create_qldb_session() as session:
            # Scan all the tables and print their documents.
            tables = session.list_tables()
            for table in tables:
                cursor = session.execute_lambda(
                    lambda executor: scan_table(executor, table),
                    retry_indicator=lambda retry_attempt: logger.info('Retrying due
 to OCC conflict...'))
                logger.info('Scan successful!')
                print_result(cursor)
    except Exception:
        logger.exception('Unable to scan tables.')
```

2. To run the program, enter the following command.

```
python scan_table.py
```

3. Use the following program (`find_vehicles.py`) to query all vehicles registered under a person in your ledger.

   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(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_driver() as driver:
            # Find all vehicles registered under a person.
            gov_id = SampleData.PERSON[0]['GovId']
            driver.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.')
```

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
First, this program queries the `Person` table for the document with `GovId LEWISR261LL` to get its `id` metadata field.
Then, it uses this document `id` as a foreign key to query the `VehicleRegistration` table by `PrimaryOwner.PersonId`. It also joins `VehicleRegistration` with the `Vehicle` table on the `VIN` field.

4. To run the program, enter the following command.

```
python find_vehicles.py
```

To learn about modifying documents in the tables in the `vehicle-registration` ledger, see Step 5: Modify documents in a ledger (p. 282).

Step 5: Modify documents in a ledger

Now that you have data to work with, you can start making changes to documents in the `vehicle-registration` ledger in Amazon QLDB. In this step, the following code examples demonstrate how to run data manipulation language (DML) statements. These statements update the primary owner of one vehicle and add a secondary owner to another vehicle.

To modify documents

1. Use the following program (`transfer_vehicle_ownership.py`) to update the primary owner of the vehicle with VIN `1N4AL11D75C109151` in your ledger.

   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(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_driver() as driver:
            driver.execute_lambda(lambda executor:
 validate_and_update_registration(executor, vehicle_vin,

  previous_owner, new_owner))
    except Exception:
        logger.exception('Error updating VehicleRegistration.')
```

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. To run the program, enter the following command.

```
python transfer_vehicle_ownership.py
```

3. Use the following program (`add_secondary_owner.py`) to add a secondary owner to the vehicle with VIN `KM8SRDHF6EU074761` in your ledger.

   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(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_driver() as driver:
            driver.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.')
```

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. To run the program, enter the following command.

```
python add_secondary_owner.py
```

To review these changes in the `vehicle-registration` ledger, see Step 6: View the revision history for a document (p. 292).

Step 6: View the revision history for a document

After modifying registration data for a vehicle in the previous step, you can query the history of all its registered owners and any other updated fields. In this step, you query the revision history of a document in the `VehicleRegistration` table in your `vehicle-registration` ledger.

To view the revision history

1. Use the following program (`query_history.py`) to query the revision history of the `VehicleRegistration` document with VIN `1N4AL11D75C109151`.

   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(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_driver() as driver:
            vin = SampleData.VEHICLE_REGISTRATION[0]['VIN']
            driver.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.')
```

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

- You can view the revision history of a document by querying the built-in History function (p. 326) in the following syntax.

```
SELECT * FROM history( table [, `start-time`, `end-time`] ) AS h
WHERE h.metadata.id = 'id'
```

- Best practice is to qualify a history query with a document `id`. This helps to avoid inefficient queries.
- The `start-time` and `end-time` are both optional. They are Amazon Ion literal values that can be denoted with backticks (`` `...` ``). To learn more, see Querying Ion with PartiQL (p. 444).

2. To run the program, enter the following command.

```
python query_history.py
```

To cryptographically verify a document revision in the `vehicle-registration` ledger, proceed to Step 7: Verify a document in a ledger (p. 296).

Step 7: Verify a document in a ledger

With Amazon QLDB, you can efficiently verify the integrity of a document in your ledger's journal by using cryptographic hashing with SHA-256. To learn more about how verification and cryptographic hashing work in QLDB, see Data verification in Amazon QLDB (p. 338).

In this step, you verify a document revision in the `VehicleRegistration` table in your `vehicle-registration` ledger. First, you request a digest, which is returned as an output file and acts as a signature of your ledger's entire change history. Then, you request a proof for the revision relative to that digest. Using this proof, the integrity of your revision is verified if all validation checks pass.

To verify a document revision

1. Review the following `.py` files, which represent QLDB objects that are required for verification and a utility module with helper functions to convert QLDB response types to strings.

   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. Use two `.py` programs (`get_digest.py` and `get_revision.py`) to perform the following steps:

   - Request a new digest from the `vehicle-registration` ledger.
   - Request a proof for each revision of the document with VIN `1N4AL11D75C109151` from the `VehicleRegistration` table.
   - Verify the revisions using the returned digest and proof by recalculating the digest.

   The `get_digest.py` program contains the following code.

```
# 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


if __name__ == '__main__':
    """
    This is an example for retrieving the digest of a particular ledger.
    """
    try:
        digest = get_digest_result(Constants.LEDGER_NAME)
    except Exception as e:
        logger.exception('Unable to get a ledger digest!')
        raise e
```

Note
Use the `get_digest_result` function to request a digest that covers the current *tip* of the journal in your ledger. The tip of the journal refers to the latest committed block as of the time that QLDB receives your request.

The `get_revision.py` program contains the following code.

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 = ?'
    parameters = [convert_object_to_ion(vin)]
    cursor = driver.execute_lambda( lambda txn: txn.execute_statement(query,
 parameters))
    return cursor
```

```
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))
```

```
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_driver() as driver:
            verify_registration(driver, Constants.LEDGER_NAME, vin)
    except Exception:
        logger.exception('Unable to verify revision.')
```

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
After the `get_revision` function returns a proof for the specified document revision, this program uses a client-side API to verify that revision.

3. To run the program, enter the following command.

```
python get_revision.py
```

If you no longer need to use the `vehicle-registration` ledger, proceed to Step 8 (optional): Clean up resources (p. 308).

Step 8 (optional): Clean up resources

You can continue using the `vehicle-registration` ledger. However, if you no longer need it, you should delete it.

To delete the ledger

1. Use the following program (`delete_ledger.py`) to delete your `vehicle-registration` ledger and all of its contents.

```
# 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))


if __name__ == '__main__':
    """
    Delete a ledger.
    """
    try:
        set_deletion_protection(Constants.LEDGER_NAME, False)
        delete_ledger(Constants.LEDGER_NAME)
        wait_for_deleted(Constants.LEDGER_NAME)
    except Exception as e:
        logger.exception('Unable to delete the ledger.')
        raise e
```

Note

If deletion protection is enabled for your ledger, you must first disable it before you can delete the ledger using the QLDB API.

The `delete_ledger.py` file also has a dependency on the following program (`describe_ledger.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 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


if __name__ == '__main__':
    """
    Describe a QLDB ledger.
    """
    try:
        describe_ledger(Constants.LEDGER_NAME)
    except Exception:
        logger.exception('Unable to describe a ledger.')
```

2. To run the program, enter the following command.

```
python delete_ledger.py
```

Amazon QLDB driver recommendations

This section describes best practices for configuring and using the Amazon QLDB driver for any supported language. The code examples provided are specifically for Java.

These recommendations apply for most typical use cases, but one size doesn't fit all. Use the following recommendations as you see fit for your application.

Topics

Configuring the QldbDriver object

The `QldbDriver` object manages connections to your ledger by maintaining a pool of *sessions* that are reused across transactions. A session (p. 335) represents a single connection to the ledger. QLDB supports one actively executing transaction per session.

Important

For older driver versions, the session pooling functionality is still in the `PooledQldbDriver` object instead of `QldbDriver`. If you're using one of the following versions, replace any mentions of `QldbDriver` with `PooledQldbDriver` for the rest of this topic.

Driver	Version
Java	`1.1.0` or earlier
.NET	`0.1.0-beta`
Node.js	`1.0.0-rc.1` or earlier
Python	`2.0.2` or earlier

The `PooledQldbDriver` object is deprecated in the latest version of the drivers. We recommend that you upgrade to the latest version and convert any instances of `PooledQldbDriver` to `QldbDriver`.

Configure QldbDriver as a global object

To optimize the use of drivers and sessions, ensure that only one global instance of the driver exists in your application instance. For example in Java, you can use *dependency injection* frameworks such as Spring, Google Guice, or Dagger. The following code example shows how to configure `QldbDriver` as a singleton.

```
@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();
}
```

Configure the retry attempts

The driver automatically retries transactions when common transient exceptions (such as `SocketTimeoutException` or `NoHttpResponseException`) occur. To set the maximum number of retry attempts, you can use the `maxRetries` parameter of the `transactionRetryPolicy` configuration object when creating an instance of `QldbDriver`. (For older driver versions as listed in the previous section, use the `retryLimit` parameter of `PooledQldbDriver`.)

The default value of `maxRetries` is `4`.

Client-side errors such as `InvalidParameterException` can't be retried. When they occur, the transaction is aborted, the session is returned to the pool, and the exception is thrown to the driver's client.

Configure the maximum number of concurrent sessions and transactions

The maximum number of ledger sessions that are used by an instance of `QldbDriver` to execute transactions is defined by its `maxConcurrentTransactions` parameter. (For older driver versions as listed in the previous section, this is defined by the `poolLimit` parameter of `PooledQldbDriver`.)

This limit must be greater than zero and less than or equal to the maximum number of open HTTP connections that the session client allows, as defined by the specific AWS SDK. For example in Java, the maximum number of connections is set in the ClientConfiguration object.

The default value of `maxConcurrentTransactions` is the maximum connection setting of your AWS SDK.

When you configure the `QldbDriver` in your application, take the following scaling considerations:

- Your pool should always have at least as many sessions as the number of concurrently executing transactions that you plan to have.
- In a multi-threaded model where a supervisor thread delegates to worker threads, the driver should have at least as many sessions as the number of worker threads. Otherwise, at peak load, threads will be waiting in line for an available session.
- The service limit of concurrent active sessions per ledger is defined in Quotas and limits in Amazon QLDB (p. 587). Ensure that you don't configure more than this limit of concurrent sessions to be used for a single ledger across all clients.

Retrying exceptions

When retrying exceptions that occur in QLDB, consider the following recommendations.

Retrying OccConflictException

Optimistic concurrency control (OCC) conflict exceptions occur when the data that the transaction is accessing has changed since the start of the transaction. QLDB throws this exception while trying to commit the transaction. The driver retries the transaction up to as many times as `maxRetries` is configured.

For more information about OCC and best practices for using indexes to limit OCC conflicts, see Amazon QLDB concurrency model (p. 332).

Retrying other exceptions outside of QldbDriver

To retry a transaction outside of the driver when custom, application-defined exceptions are thrown during execution, you must wrap the transaction. For example in Java, the following code shows how to use the Reslience4J library to retry a transaction in QLDB.

```
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

Retrying a transaction outside of the QLDB driver has a multiplier effect. For example, if `QldbDriver` is configured to retry three times, and the custom retry logic also retries three times, the same transaction can be retried up to nine times.

Making transactions idempotent

As a best practice, make your write transactions idempotent to avoid any unexpected side effects in the case of retries. A transaction is *idempotent* if it can be executed multiple times and produce identical results each time.

To learn more, see Amazon QLDB concurrency model (p. 334).

Optimizing performance

To optimize performance when you execute transactions using the driver, take the following considerations:

- The `execute` method always makes a minimum of three `SendCommand` API calls to QLDB, including the following commands:
  1. `StartTransaction`
  2. `ExecuteStatement`

     This command is executed for each PartiQL statement that you run in the `execute` method block.
  3. `CommitTransaction`

  Consider the total number of API calls that are made when you calculate the overall workload of your application.
- In general, we recommend starting with a single-threaded writer and optimizing transactions by batching multiple statements within a single transaction. Maximize the quotas on transaction size, document size, and number of documents per transaction, as defined in Quotas and limits in Amazon QLDB (p. 587).
- If batching is not sufficient for large transaction loads, you can try multi-threading by adding additional writers. However, you should carefully consider your application requirements for document and transaction sequencing and the additional complexity that this introduces.

Running multiple statements per transaction

As described in the previous section (p. 313), you can run multiple statements per transaction to optimize performance of your application. In the following Java example, you create a table and then create an index on that table within a transaction. You do this by passing a lambda expression to the `execute` method.

```
public static void main(final String... args) {
    ConnectToLedger.getDriver().execute(txn -> {
        final Result createTableResult = txn.execute("CREATE TABLE Vehicle");
        final Result createIndexResult = txn.execute("CREATE INDEX ON Vehicle (VIN)");
    });
    log.info("Vehicle table created successfully with index on VIN.");
}
```

The driver's `execute` method implicitly starts a session and a transaction in that session. Each statement that you run in the lambda expression is wrapped in the transaction. After all of the statements run, the driver auto-commits the transaction. If any statement fails after the automatic retry limit is exhausted, the transaction is aborted.

Propagate exceptions in a transaction

When running multiple statements per transaction, we generally don't recommend that you catch and swallow exceptions within the transaction.

For example in Java, the following program catches any instance of `RuntimeException`, logs the error, and continues. This code example is considered bad practice because the transaction succeeds even when the `UPDATE` statement fails. So, the client might assume that the update succeeded when it didn't.

Warning
Do not use this code example. It's provided to show an anti-pattern example that is considered bad practice.

```
// 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.");
}
```

Propagate the exception instead. If any part of the transaction fails, let the `execute` method abort the transaction so that the client can handle the exception accordingly.

Common errors from the Amazon QLDB driver

This section describes runtime errors that can be thrown by the Amazon QLDB driver when calling the *QLDB Session* API.

The following is a list of common exceptions that are returned by the driver. Each exception includes the specific error message, followed by a short description and suggestions for possible solutions.

InvalidSessionException

Message: Transaction `transactionId` has expired

A transaction exceeded its maximum lifetime. A transaction can run for up to 30 seconds before being committed. After this limit, any work done on the transaction is rejected, and Amazon QLDB discards the session. This limit protects the client from leaking sessions by starting transactions and not committing or aborting them.

If this is a common exception in your application, it is likely that transactions are simply taking too long to execute. If transaction execution is taking longer than 30 seconds, optimize your statements to speed up the transactions. Examples of statement optimization include executing fewer statements per transaction and tuning your table indexes.

InvalidSessionException

Message: Exceeded the session idle time limit: Session `sessionId`

QLDB has discarded the session because it remained idle past the idle time limit. QLDB defines an idle session as a session that is not in an active transaction. A session can be idle for up to 60 seconds, after which QLDB discards the session to protect the client from leaking sessions.

If you encounter this exception, we recommend that you acquire a new session and retry the transaction. We also recommend using the latest version of the `QldbDriver` object and its execute method. This driver manages the session pool and its health on the application's behalf.

InvalidSessionException

Message: Session `sessionId` has expired

QLDB discarded the session because it exceeded its maximum total lifetime. QLDB discards sessions after approximately 15 minutes, regardless of idleness. Sessions can be lost or impaired for a number of reasons, such as hardware failure, network failure, or application restarts. So, QLDB enforces a maximum lifetime on sessions to ensure that client software is resilient to session failure.

If you encounter this exception, we recommend that you acquire a new session and retry the transaction. We also recommend using the latest version of the `QldbDriver` object and its execute method. This driver object manages the session pool and its health on the application's behalf.

InvalidSessionException

Message: No such session

The client tried to transact with QLDB using a session that doesn't exist. Assuming that the client is using a session that previously existed, the session might no longer exist because of one of the following:

- If a session is involved in an internal server failure (that is, an error with HTTP response code 500), QLDB might choose to discard the session completely, rather than allow the customer to transact with a session of uncertain state. Then, any retry attempts on that session fail with this error.
- Sessions that expire or exceed the idle time limit are eventually forgotten by QLDB. Then, any attempts to continue using the session result in this error, rather than the initial `InvalidSessionException`.

If you encounter this exception, we recommend that you acquire a new session and retry the transaction. We also recommend using the latest version of the `QldbDriver` object and its execute method. This driver object manages the session pool and its health on the application's behalf.

RateExceededException

Message: The rate was exceeded

QLDB throttled a client based on the caller's identity. QLDB enforces throttling on a per-Region, per-account basis using a token bucket throttling algorithm. QLDB does this to help the performance of the service, and to ensure fair usage for all QLDB customers. For example, trying to acquire a large number of concurrent sessions using the `StartSessionRequest` operation might lead to throttling.

To maintain your application health and mitigate further throttling, you can retry on this exception using Exponential Backoff and Jitter. If your application is consistently getting throttled by QLDB for `StartSessionRequest` calls, we recommend using the latest version of the `QldbDriver`. This driver object maintains a pool of sessions that are reused across transactions, which can help to reduce the number of `StartSessionRequest` calls that your application makes. To request an increase in API throttling limits, contact the AWS Support Center.

LimitExceededException

Message: Exceeded the session limit

A ledger exceeded its quota (also known as a *limit*) on the number of active sessions. This quota is defined in Quotas and limits in Amazon QLDB (p. 587). A ledger's active session count is eventually consistent, and ledgers consistently running near the quota might periodically see this exception.

To maintain your application's health, we recommend retrying on this exception. To avoid this exception, ensure that you have not configured more than 1,500 concurrent sessions to be used for a single ledger across all clients. For example, you can use the maxConcurrentTransactions method of the Amazon QLDB driver for Java to configure the maximum number of available sessions in a driver instance.

QldbClientException

Message: A streamed result is only valid when the parent transaction is open

The transaction is closed, and it can't be used to retrieve the results from QLDB. A transaction closes when it's either committed or aborted.

This exception occurs when the client is working directly with the `Transaction` object, and it's trying to retrieve results from QLDB after committing or aborting a transaction. To mitigate this issue, the client must read the data before closing the transaction.

Working with data and history in Amazon QLDB

The following sections provide basic examples of *create, read, update, and delete* (CRUD) statements that you can manually run using the *Query editor* on the QLDB console (p. 18). This guide also walks you through the process of how QLDB handles your data as you make changes in your ledger.

QLDB supports the PartiQL query language.

For code examples that show how to programmatically run similar statements using the QLDB driver, see the tutorials in Getting started with the driver (p. 40).

Topics

Creating tables with indexes and inserting documents

After creating an Amazon QLDB ledger, your first step is to create a table with a basic CREATE TABLE (p. 449) statement. The fundamental data model in QLDB is described as follows:

1. Ledgers consist of user-defined tables. A table can be thought of as a *bag*, which is an unordered collection in PartiQL.
2. Tables consist of QLDB documents (p. 427), which are datasets in Amazon Ion (p. 487) `struct` format.

Topics

Creating tables and indexes

Tables have simple names with no namespaces. QLDB supports open content and does not enforce schema, so you don't define attributes or data types when creating tables.

```
CREATE TABLE VehicleRegistration
```

```
CREATE TABLE Vehicle
```

A `CREATE TABLE` statement returns the system-assigned ID of the new table. All QLDB-assigned IDs (p. 330) are universally unique identifiers (UUID) that are each represented in a Base62-encoded string.

You can also create indexes on tables to help speed up queries.

> **Note**
> In QLDB, indexes are optional. They can improve query performance for seek operations. However, note the following:
>
> - Indexes can only be created on empty tables.
> - Indexes can only be created on a single top-level field. Composite, nested, unique, and function-based indexes are currently not supported.
> - Indexes can't be dropped after they are created.
> - Query performance is improved only when you use an equality predicate; for example, `fieldName = 123456789`.
>
>   QLDB does not currently honor inequalities in query predicates. One implication of this is that range filtered scans are not implemented.
> - Names of indexed fields can have a maximum of 128 characters.

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

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

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

Inserting documents

Then you can insert documents into your tables. QLDB documents are stored in Amazon Ion format. The following PartiQL INSERT (p. 455) statements include a subset of the vehicle registration sample data used in Getting started with the Amazon QLDB console (p. 25).

```
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 syntax and semantics

- Field names are enclosed in single quotation marks (`'...'`).
- String values are also enclosed in single quotation marks (`'...'`).
- Timestamps are enclosed in backticks (`` `...` ``). Backticks can be used to denote any Ion literals.
- Integers and decimals are literal values that don't need to be denoted.

For more details on PartiQL's syntax and semantics, see Querying Ion with PartiQL (p. 444).

An `INSERT` statement creates the initial revision of a document with a version number of zero. To uniquely identify each document, QLDB assigns a *document ID* as part of the metadata. Insert statements return the ID of each document that is inserted.

Important
Because QLDB does not enforce schema, you can insert the same document into a table multiple times. Each insert statement commits a separate document entry to the journal, and QLDB assigns each document a unique ID.

To learn how to query the documents you inserted into your table, proceed to Querying your data (p. 319).

Querying your data

The *user view* returns the latest non-deleted revision of your data only. This is the default view in Amazon QLDB. This means that no special qualifiers are needed when you want to query only your data.

For details on the syntax and parameters of the following query examples, see SELECT (p. 457) in the *Amazon QLDB PartiQL reference*.

Topics

Basic queries

Basic `SELECT` queries return the documents that you inserted into the table. The following queries show results for the vehicle registration documents previously inserted in Creating tables with indexes and inserting documents (p. 317).

Tip
PartiQL is SQL compatible, so it supports table scans such as this example. But we don't recommend running statements without a `WHERE` predicate clause for production use. This can cause transaction timeouts on large tables.
The best practice is to run statements that filter on a document `id` or an indexed field. For more information, see Using indexes to limit OCC conflicts (p. 332) in the *Concurrency model* section.

```
SELECT * FROM VehicleRegistration
```

```
{
    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" }]
    }
},
{
    VIN: "KM8SRDHF6EU074761",
    LicensePlateNumber: "CA762X",
    State: "WA",
    City: "Kent",
    PendingPenaltyTicketAmount: 130.75,
    ValidFrom: 2017-09-14T,
    ValidTo: 2020-06-25T,
    Owners: {
        PrimaryOwner: { PersonId: "IN7MvYtUjkp1GMZu0F6CG9" },
        SecondaryOwners: []
    }
}
```

```
SELECT * FROM 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"
}
```

Important
In PartiQL, you use single quotation marks to specify strings in data manipulation language (DML) or query statements. But the QLDB console returns query results in Amazon Ion format, so you see strings enclosed in double quotation marks.
This syntax allows the PartiQL query language to maintain SQL compatibility, and the Amazon Ion data format to maintain JSON compatibility.

Projections and filters

You can do projections (targeted `SELECT`) and filters (`WHERE` clauses). The following query returns a subset of document fields from the `VehicleRegistration` table. It filters for vehicles with the following criteria:

- **String filter** – It is registered in Seattle.
- **Decimal filter** – It has a pending penalty ticket amount less than `100.0`.
- **Date filter** – It has a registration date that is valid on or after September 4, 2019.

```
SELECT r.VIN, r.PendingPenaltyTicketAmount, r.Owners
FROM VehicleRegistration AS r
WHERE 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" }]
    }
}
```

Joins

You can also write inner join queries. The following example shows an implicit inner join query that returns all registration documents along with attributes of the registered vehicles.

```
SELECT * FROM VehicleRegistration AS r, Vehicle AS v WHERE r.VIN = v.VIN
```

```
{
    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" }]
    },
    Type: "Sedan",
    Year: 2011,
    Make: "Audi",
    Model: "A5",
    Color: "Silver"
},
{
    VIN: "KM8SRDHF6EU074761",
    LicensePlateNumber: "CA762X",
    State: "WA",
    City: "Kent",
    PendingPenaltyTicketAmount: 130.75,
    ValidFrom: 2017-09-14T,
    ValidTo: 2020-06-25T,
    Owners: {
        PrimaryOwner: { PersonId: "IN7MvYtUjkp1GMZu0F6CG9" },
        SecondaryOwners: []
    },
    Type: "Sedan",
    Year: 2015,
    Make: "Tesla",
    Model: "Model S",
    Color: "Blue"
}
```

Nested data

PartiQL has Amazon Ion extensions that enable you to query nested data in documents. The following example shows a correlated subquery that flattens nested data. The `@` character is technically optional here. But it explicitly indicates that you want the `Owners` structure within `VehicleRegistration`, not a different collection named `Owners` (if one existed).

```
SELECT
    r.VIN,
    o.SecondaryOwners
FROM
    VehicleRegistration AS r, @r.Owners AS o
```

```
{
    VIN: "1N4AL11D75C109151",
    SecondaryOwners: [{ PersonId: "5Ufgdlnj06gF5CWcOIu64s" }]
},
{
    VIN: "KM8SRDHF6EU074761",
    SecondaryOwners: []
}
```

The following shows a subquery in the `SELECT` list that projects nested data, in additional to an inner join.

```
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
```

```
{
    Make: "Audi",
    Model: "A5",
    PrimaryOwner: ["294jJ3YUoH1IEEm8GSabOs"]
},
{
    Make: "Tesla",
    Model: "Model S",
    PrimaryOwner: ["IN7MvYtUjkp1GMZu0F6CG9"]
}
```

To learn how to query your document metadata, proceed to Querying document metadata (p. 323).

Querying document metadata

An `INSERT` statement creates the initial revision of a document with a version number of zero. To uniquely identify each document, Amazon QLDB assigns a *document ID* as part of the metadata.

In addition to the document ID and version number, QLDB stores other system-generated metadata for each document in a table. This metadata includes transaction information, journal attributes, and the document's hash value.

All QLDB-assigned IDs are universally unique identifiers (UUID) that are each represented in a Base62-encoded string. For more information, see Unique IDs in Amazon QLDB (p. 330).

Committed view

You can access this metadata by querying the *committed view*. This view returns documents from the system-defined table that directly corresponds to your user table. It includes the latest committed, non-deleted revision of both your data and the QLDB-generated metadata. To query this view, add the prefix `_ql_committed_` to the table name in your query. (The prefix `_ql_` is reserved in QLDB for system objects.)

```
SELECT * FROM _ql_committed_VehicleRegistration
```

Using the data previously inserted in Creating tables with indexes and inserting documents (p. 317), the output of this query shows the system contents of each non-deleted document's latest revision. The system document has metadata nested in the `metadata` field, and your user data nested in the `data` field.

```
{
    blockAddress:{
        strandId:"JdxjkR9bSYB5jMHWcI464T",
        sequenceNo:14
    },
    hash:{{wCsmM6qD4STxz0WYmE+47nZvWtcCz9D6zNtCiM5GoWg=}},
    data:{
        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" }]
        }
    },
    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,
        ValidFrom: 2017-09-14T,
        ValidTo: 2020-06-25T,
        Owners: {
            PrimaryOwner: { PersonId: "IN7MvYtUjkp1GMZu0F6CG9" },
            SecondaryOwners: []
        }
    },
    metadata:{
        id:"JOzfB3lWqGU727mpPeWyxg",
        version:0,
        txTime:2019-06-05T20:53:321d-3Z,
        txId:"HgXAkLjAtV0HQ4lNYdzX60"
    }
}
```

Committed view fields

- `blockAddress`—The location of the block in your ledger's journal where the document revision was committed. An address, which can be used for cryptographic verification, has the following two fields.
  - `strandId`—The unique ID of the journal strand that contains the block.
  - `sequenceNo`—An index number that specifies the location of the block within the strand.

    Note
    Both documents in this example have an identical `blockAddress` with the same `sequenceNo`. Because these documents were inserted within a single transaction (and in this case, in a single statement), they were committed in the same block.
- `hash`—The SHA-256 value that uniquely represents the document revision. The hash covers the `data` and `metadata` fields and can be used for cryptographic verification.
- `metadata`—The document's metadata attributes.
  - `id`—The system-assigned unique ID of the document.
  - `version`—The zero-based integer that increments with each document revision.
  - `txTime`—The timestamp when the document revision was committed to the journal.

- `txId`—The unique ID of the transaction that committed the document revision.

To learn how to query the document ID field in the default user view, proceed to Using the BY clause to query document ID (p. 325).

Using the BY clause to query document ID

While you can define fields that are intended to be unique identifiers (for example, a vehicle's VIN), the true unique identifier of a document is the `id` metadata field, as explained in Inserting documents (p. 318). For this reason, you can use the `id` field to create relationships between tables.

The document `id` field is directly accessible in the committed view only, but you can also project it in the default user view by using the `BY` clause. For an example, see the following query and its results.

```
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" }]
    }
}
```

In this query, `r_id` is a user-defined alias that is declared in the `FROM` clause, using the `BY` keyword. This `r_id` alias binds to the `id` metadata field for each document in the query's result set. You can use this alias in the `SELECT` clause and also in the `WHERE` clause of a query in the *user view*.

To access other metadata attributes, however, you must query the committed view.

To learn how to make changes to a document in your table, proceed to Updating and deleting documents (p. 325).

Updating and deleting documents

In Amazon QLDB, a *document revision* is a structure that represents a single iteration of a document's full dataset and includes both your data and system-generated metadata. Each revision is uniquely identified by a combination of the document ID and a zero-based version number.

The lifecycle of a document ends when it's deleted from a table. This means that no document with the same document ID can be created again in the same ledger.

Making document revisions

For example, the following statements insert a new vehicle registration, update the registration city, and then delete the registration. This results in three revisions of a document.

```
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 statements and other DML statements return the ID of each affected document. Before you continue, save this ID because you need it for the history function in the next section. You can also find the document ID with the following query.

```
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'
```

For information about the syntax and parameters of these DML statements, see UPDATE (p. 460) and DELETE (p. 450) in the *Amazon QLDB PartiQL reference*.

After you delete a document, you can no longer query it in the committed or user views. To learn how to query the revision history of this document using the built-in history function, proceed to Querying revision history (p. 326).

Querying revision history

Amazon QLDB stores the complete history of every document in a table. You can see all three revisions of the vehicle registration document you previously inserted, updated, and deleted in Updating and deleting documents (p. 325) by querying the built-in history function.

History function

The history function in QLDB is a PartiQL extension that returns revisions from the system-defined view of your table. So, it includes both your data and the associated metadata in the same schema as the committed view.

The history function has the following syntax.

```
SELECT * FROM history( table [, `start-time`, `end-time`] ) AS h
WHERE h.metadata.id = 'id'
```

Tip

- As a best practice, qualify a history query with a `WHERE` predicate clause that filters on both a date range (*start-time* and *end-time*) and a document ID (`metadata.id`).

  QLDB processes `SELECT` queries in transactions, which are subject to a transaction timeout limit. QLDB history is indexed by document ID, and you can't create additional history indexes at this time. History queries that include a start time and end time gain the benefit of date range qualification.
- The *start-time* and *end-time* are Ion timestamp literals that can be denoted with backticks (`` `...` ``). To learn more, see Querying Ion with PartiQL (p. 444).

The *table* parameter can be either a table name or a table ID. A table name is a PartiQL identifier that you can denote with double quotation marks or no quotation marks. A table ID is a string literal that must be enclosed in single quotation marks. To learn more, see Querying the history of dropped tables (p. 330).

The start and end times specify the time range during which any revisions were active. They don't specify the transaction time range during which revisions were committed to the journal. The start and end time parameters have the following behavior:

- The *start-time* and *end-time* are optional parameters that are both inclusive. They must be in ISO 8601 date and time format and in Coordinated Universal Time (UTC).
- The *start-time* must be less than or equal to *end-time* and can be any arbitrary date in the past.
- The *end-time* must be less than or equal to the current UTC date and time.
- If you specify a *start-time* but not an *end-time*, your query defaults the *end-time* to the current date and time. If you specify neither, your query returns the entire history.

To query the document's history, use the `id` that you previously saved in Updating and deleting documents (p. 325). For example, the following query returns the revision history for document ID `ADR2Ll1fGsU4Jr4EqTdnQF` between `2000T` and `2019-06-05T23:59:59Z`. Be sure to replace the `id`, start time, and end time with your own values as appropriate.

```
SELECT * FROM history(VehicleRegistration, `2000T`, `2019-06-05T23:59:59Z`) AS h
WHERE h.metadata.id = 'ADR2Ll1fGsU4Jr4EqTdnQF' --replace with your id
```

Your query results should look similar to the following.

```
{
    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"
    }
}
```

The output includes metadata attributes that provide details on when each item was modified, and by which transaction. From this data, you can see the following:

- The document is uniquely identified by its system-assigned `id`: `ADR2Ll1fGsU4Jr4EqTdnQF`. This is a UUID that is represented in a Base62-encoded string.
- An `INSERT` statement creates the initial revision of a document (version `0`).
- Each subsequent update creates a new revision with the same document `id` and an incremented version number.
- The `txId` field indicates the transaction that committed each revision, and `txTime` shows when each was committed.
- A `DELETE` statement creates a new, but final revision of a document. This final revision has metadata only.

To learn how to query the system catalog, proceed to Querying the system catalog (p. 329).

Querying the system catalog

Each table that you create in an Amazon QLDB ledger has a system-assigned unique ID. You can find a table's ID, its list of indexes, and other metadata by querying the system catalog table `information_schema.user_tables`.

All QLDB-assigned IDs are universally unique identifiers (UUID) that are each represented in a Base62-encoded string. For more information, see Unique IDs in Amazon QLDB (p. 330).

The following example shows the results of a query that returns metadata attributes of the `VehicleRegistration` table.

```
SELECT * FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

```
{
    tableId: "5PLf9SXwndd63lPaSIa0O6",
    name: "VehicleRegistration",
    indexes: [{ expr: "[VIN]" }, { expr: "[LicensePlateNumber]" }],
    status: "ACTIVE"
}
```

Table metadata fields

- `tableId`—The unique ID of the table.
- `name`—The table name.
- `indexes`—The list of indexes on the table.
- `status`—The table's current status (`ACTIVE` or `INACTIVE`). A table becomes `INACTIVE` when you `DROP` it.

To learn how to manage tables using the `DROP TABLE` and `UNDROP TABLE` statements, proceed to Managing tables (p. 329).

Managing tables

This section describes how to manage tables using the `DROP TABLE` and `UNDROP TABLE` statements in Amazon QLDB. The quotas for the number of active tables and total tables that you can create are defined in Quotas and limits in Amazon QLDB (p. 587).

Topics

Dropping tables

To drop a table, use a basic DROP TABLE (p. 451) statement. When you drop a table in QLDB, you are just inactivating it.

For example, the following statement inactivates the `VehicleRegistration` table.

```
DROP TABLE VehicleRegistration
```

A `DROP TABLE` statement returns the unique ID of the table. The status of `VehicleRegistration` should now be `INACTIVE` in the system catalog table information_schema.user_tables (p. 329).

```
SELECT status FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

Querying the history of dropped tables

In addition to a table name, you can also query the QLDB History function (p. 326) with a table ID as the first input parameter. This enables you to query the history of dropped tables. After a table is dropped, you can no longer query its history with the table name.

First, find the table ID by querying the system catalog table. For example, the following query returns the `tableId` of the `VehicleRegistration` table.

```
SELECT tableId FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

Then, you can use this ID to run the same history query from the previous section. The following is an example that queries the history of document ID `ADR2Ll1fGsU4Jr4EqTdnQF` from table ID `5PLf9SXwndd63lPaSIa0O6`. The table ID is a string literal that must be enclosed in single quotation marks.

```
--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'
```

Undropping tables

After you drop a table in QLDB, you can use the UNDROP TABLE (p. 462) statement to reactivate it.

First, find the table ID from `information_schema.user_tables`. For example, the following query returns the `tableId` of the `VehicleRegistration` table. The status should be `INACTIVE`.

```
SELECT tableId FROM information_schema.user_tables
WHERE name = 'VehicleRegistration'
```

Then, use this ID to reactivate the table. The following is an example that undrops table ID `5PLf9SXwndd63lPaSIa0O6`. The table ID is a unique identifier that should be enclosed in double quotation marks.

```
UNDROP TABLE "5PLf9SXwndd63lPaSIa0O6"
```

The status of `VehicleRegistration` should now be `ACTIVE`.

Unique IDs in Amazon QLDB

This section describes the properties and usage guidelines of system-assigned unique identifiers in Amazon QLDB. It also provides some examples of QLDB unique IDs.

Topics

Properties

All QLDB-assigned IDs are universally unique identifiers (UUID). Each ID has the following properties:

- 128-bit UUID number.
- Represented in Base62-encoded text.
- Fixed length alphanumeric string of 22 characters (for example: `3Qv67yjXEwB9SjmvkuG6Cp`).

Usage

When using QLDB unique IDs in your application, note the following guidelines:

Do

- Treat the ID as a string.

Do not

- Try to decode the string.
- Ascribe semantic meaning to the string (such as deriving a time component).
- Sort the strings in a semantic order.

Examples

The following attributes are some examples of unique IDs in QLDB:

- Document ID
- Index ID
- Table ID
- Strand ID
- Transaction ID

Amazon QLDB concurrency model

Amazon QLDB is intended to address the needs of high-performance online transaction processing (OLTP) workloads. QLDB supports SQL-like query capabilities, and delivers full ACID transactions. In addition, QLDB data items are documents, delivering schema flexibility and intuitive data modeling. With a journal at the core, QLDB makes it easy to access the complete and verifiable history of all changes to your data, and to stream coherent transactions to other data services as needed.

Topics

Optimistic concurrency control

In QLDB, concurrency control is implemented using *optimistic concurrency control* (OCC). OCC operates on the principle that multiple transactions can frequently complete without interfering with each other.

Using OCC, transactions in QLDB don't acquire locks on database resources and operate with full serializable isolation. QLDB executes concurrent transactions in a serial manner, such that it produces the same effect as if those transactions were executed serially.

Before committing, each transaction performs a validation check to ensure that no other committed transaction has modified the snapshot of data that it's reading. If this check reveals conflicting modifications, or the state of the data snapshot changes, the committing transaction is rejected. However, the transaction can be restarted.

When a transaction writes to QLDB, the validation checks of the OCC model are implemented by QLDB itself. If a transaction can't be written to the journal due to a failure in the verification phase of OCC, QLDB returns an `OccConflictException` to the application layer. The application software is responsible for ensuring that the transaction is restarted. The application should abort the rejected transaction and then retry the whole transaction from the start.

For a code example that shows how to catch and handle OCC conflicts, see Handling OCC conflict exceptions (p. 335).

Using indexes to limit OCC conflicts

As a best practice, we recommend that you run statements with a `WHERE` predicate clause that filters on a document `id` or an indexed field. QLDB requires an index to efficiently look up a document. Without an index or a document `id`, QLDB needs to run a *table scan* when reading documents. This can cause more query latency and also increases the chances of an OCC conflict.

For example, consider a table named `Vehicle` that has an index on the `VIN` field only. It contains the following documents.

VIN	Make	Model	Color		
"1N4AL11D75C109151"	"Audi"	"A5"	"Silver"		
"KM8SRDHF6EU074761"	"Tesla"	"Model S"	"Blue"		
"3HGGK5G53FM761765"	"Ducati"	"Monster 1200"	"Yellow"		
"1HVBBAANXWH544237"	"Ford"	"F 150"	"Black"		
"1C4RJFAG0FC625797"	"Mercedes"	"CLK 350"	"White"		

Two concurrent users named Alice and Bob are working with the same table in a ledger. They want to update two different documents, as follows.

Alice:

```
UPDATE Vehicle AS v
SET v.Color = 'Blue'
WHERE v.VIN = '1N4AL11D75C109151'
```

Bob:

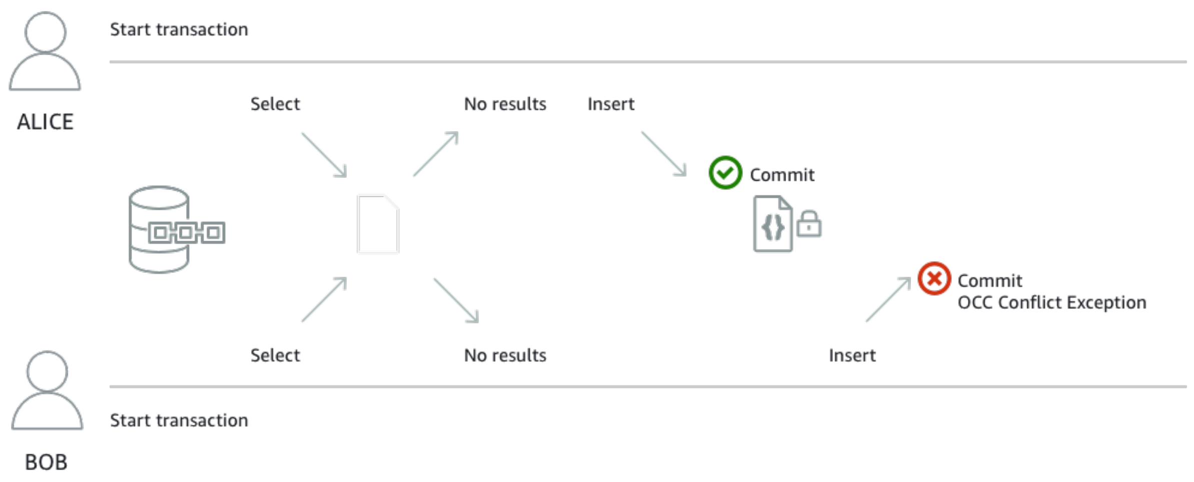
```
UPDATE Vehicle AS v
SET v.Color = 'Red'
WHERE v.Make = 'Tesla' AND v.Model = 'Model S'
```

Suppose that Alice and Bob start their transactions at the same time. Alice's `UPDATE` statement does an indexed lookup on the `VIN` field, so it only needs to read that one document. Alice finishes and successfully commits her transaction first.

Bob's statement filters on non-indexed fields, so it does a table scan and encounters an `OccConflictException`. This is because Alice's committed transaction modified the data snapshot that Bob's statement is reading, which includes every document in the table—not only the document that he is updating.

Insertion OCC conflicts

OCC conflicts can include documents that are newly inserted—not only documents that previously existed. Consider the following diagram, in which two concurrent users (Alice and Bob) are working with the same table in a ledger. They both want to insert a new document only under the condition that a predicate value does not yet exist.



In this example, both Alice and Bob execute the following `SELECT` and `INSERT` statements within a single transaction. Their application executes the `INSERT` statement only if the `SELECT` statement returns no results.

```
SELECT * FROM Vehicle v WHERE v.VIN = 'ABCDE12345EXAMPLE'
```

```
INSERT INTO Vehicle VALUE
{
    'VIN' : 'ABCDE12345EXAMPLE',
    'Type' : 'Wagon',
    'Year' : 2019,
    'Make' : 'Subaru',
    'Model' : 'Outback',
    'Color' : 'Gray'
}
```

Suppose that Alice and Bob start their transactions at the same time. Both of their `SELECT` queries return no existing document with a `VIN` of `ABCDE12345EXAMPLE`. So, their applications proceed with the `INSERT` statement.

Alice finishes and successfully commits her transaction first. Then, Bob tries to commit his transaction, but QLDB rejects it and throws an `OccConflictException`. This is because Alice's committed transaction modified the result set of Bob's `SELECT` query, and OCC detects this conflict before committing Bob's transaction.

Bob can then retry his whole transaction from the start. But his next `SELECT` query will return the document that Alice inserted, so Bob's application won't execute the `INSERT`.

Making transactions idempotent

The insert transaction in the previous section (p. 333) is also an example of an *idempotent* transaction. In other words, executing the same transaction multiple times produces identical results. If Bob executes the `INSERT` without first checking if the document already exists, the table might end up with duplicate documents that have the same `VIN`.

Consider other retry scenarios in addition to OCC conflicts. For example, it's possible that QLDB successfully commits a transaction on the server side, but the client times out while waiting for a response. As a best practice, make your write transactions idempotent to avoid any unexpected side effects in the case of retries.

Concurrent sessions management

If you have experience using a relational database management system (RDBMS), you might be familiar with concurrent connection limits. QLDB doesn't have the same concept of a traditional RDBMS connection because transactions are executed with HTTP request and response messages.

In QLDB, the analogous concept is an *active session*. A session manages information about your data transaction requests to a ledger. An active session is one that is actively executing a transaction—or a session that recently executed a transaction such that QLDB anticipates it will execute another transaction immediately. QLDB supports one actively executing transaction per session.

The limit of concurrent active sessions per ledger is defined in Quotas and limits in Amazon QLDB (p. 587). After this limit is reached, any session that tries to start a transaction will result in an error.

For best practices for using the QLDB driver in your application to configure a session pool, see Configuring the QldbDriver object (p. 311) in *Amazon QLDB driver recommendations*.

Handling OCC conflict exceptions

As described in Amazon QLDB concurrency model (p. 332), Amazon QLDB performs validation checks when trying to write transactions to the journal. If this verification phase fails, QLDB returns an *optimistic concurrency control (OCC)* conflict exception to the application layer. The application can then retry transactions that were rejected.

Note
When an OCC conflict exception occurs, the QLDB driver's `execute` method automatically retries the transaction up to a configurable number of attempts. If the transaction still fails after the limit is reached, the driver throws the exception to the client.

This section provides a Java code example that produces an `OccConflictException`. It demonstrates how to handle this conflict by catching and logging the exception.

Prerequisites

Before running the demo, make sure that you complete the following tasks if you haven't already done so:

1. Follow the steps in Installing the Amazon QLDB Java sample application (p. 45) to set up your Java environment and install the sample application.
2. Do steps 1–3 in the Java tutorial (p. 44) to create a ledger named `vehicle-registration` and load it with sample data.

Running the demo

Compile and run the following program (`OCCConflictDemo.java`), which simulates an OCC conflict exception.

Warning
This code example is for demonstration purposes only. We don't recommend running a transaction inside another transaction that accesses the same data.

```
/*
 * 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.IonValue;
import java.io.IOException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import software.amazon.awssdk.services.qldbsession.model.OccConflictException;
import software.amazon.qldb.QldbDriver;
import software.amazon.qldb.RetryPolicy;
import software.amazon.qldb.tutorial.model.SampleData;

/**
 * Demonstrates how to handle OCC conflicts, where two users try to execute and commit
 changes to the same document.
 * When an OCC conflict occurs on execute or commit, it is implicitly handled by restarting
 the transaction.
 * In this example, two sessions on the same ledger try to access the registration city for
 the same Vehicle Id.
 *
 * 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 OccConflictDemo {
    public static final Logger log = LoggerFactory.getLogger(OccConflictDemo.class);

    private OccConflictDemo() { }

    public static void main(final String... args) throws IOException {
        final String vehicleId1 = SampleData.REGISTRATIONS.get(0).getVin();
        final IonValue ionVin = Constants.MAPPER.writeValueAsIonValue(vehicleId1);

        QldbDriver driver = QldbDriver.builder()
                                              .ledger(Constants.LEDGER_NAME)
                                              .transactionRetryPolicy(RetryPolicy.none())

 .sessionClientBuilder(ConnectToLedger.getAmazonQldbSessionClientBuilder())
                                              .build();

        /*
         * Warning: Running a transaction inside another transaction that accesses the same
 document
         * is not recommended at all.
         *
         * The below code will always have an Optimistic Concurrency Control problem
 because the inner
         * transaction modified the same document that the outer transaction accessed when
 it started
         * the transaction.
         */
        try {
            log.info("Starting outer transaction");
            driver.execute(txn -> {
                txn.execute("UPDATE VehicleRegistration AS r SET r.City = 'Tukwila' WHERE
 r.VIN = ?",
                            ionVin);
                txn.execute("SELECT * FROM VehicleRegistration AS r WHERE r.VIN = ?",
                            ionVin);

                driver.execute(txn2 -> {
                    log.info("Starting inner transaction");
                    txn2.execute("UPDATE VehicleRegistration AS r SET r.City = 'Tukwila'
 WHERE r.VIN = ?",
                                 ionVin);
                    txn2.execute("SELECT * FROM VehicleRegistration AS r WHERE r.VIN = ?",
                                 ionVin);
                });
                log.info("Inner transaction succeeded and the Vehicle with VIN {} was
 updated", vehicleId1);
            });
        } catch (OccConflictException e) {
            log.error("Optimistic Concurrency Control Exception. One of the threads tried
 to commit the transaction that used"
                      + "the same "
                      + "thread already.");
        }
    }
}
```

Data verification in Amazon QLDB

With Amazon QLDB, you can trust that the history of changes to your application data is accurate. QLDB uses an immutable transactional log, known as a *journal*, for data storage. The journal tracks every change to your committed data and maintains a complete and verifiable history of changes over time.

QLDB uses the SHA-256 hash function with a Merkle tree–based model to generate a cryptographic representation of your journal, known as a *digest*. The digest acts as a unique signature of your data's entire change history as of a point in time. It enables you to look back and verify the integrity of your document revisions relative to that signature.

Topics

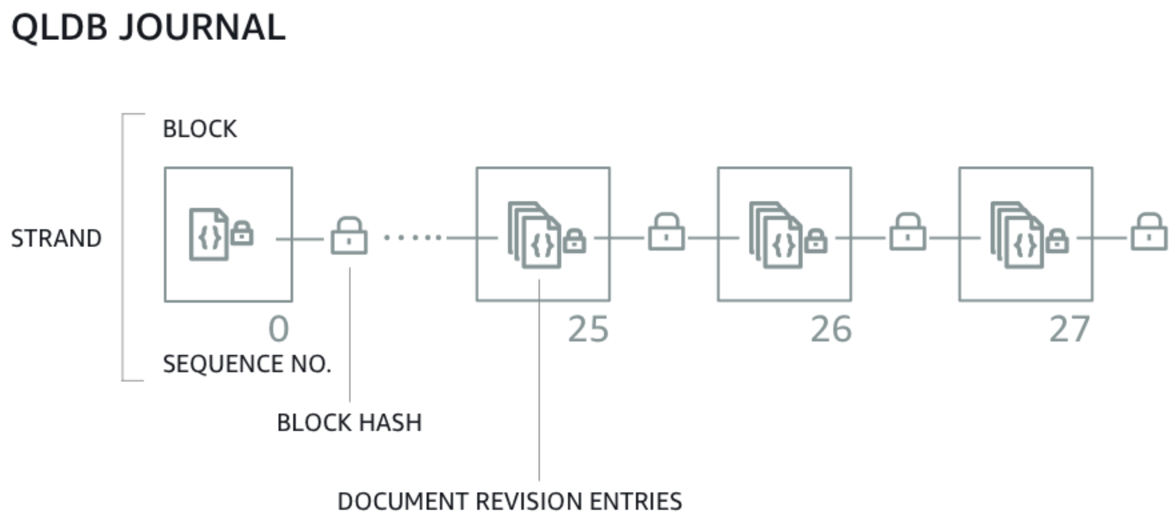
What kind of data can you verify in QLDB?

In QLDB, each ledger has exactly one journal. A journal can have multiple *strands*, which are partitions of the journal.

Note
QLDB currently supports journals with a single strand only.

A *block* is an object that is committed to the journal strand during a transaction. This block contains *entry* objects, which represent the document revisions that resulted from the transaction. You can verify either an individual revision or an entire journal block in QLDB.

The following diagram illustrates this journal structure.



The diagram shows that transactions are committed to the journal as blocks that contain document revision entries. It also shows that each block is hash-chained to subsequent blocks and has a sequence number to specify its address within the strand.

What does data integrity mean?

Data integrity in QLDB means that your ledger's journal is in fact immutable. In other words, your data (specifically, each document revision) is in a state where the following are true:

1. It exists at the same location in your journal where it was first written.
2. It hasn't been altered in any way since it was written.

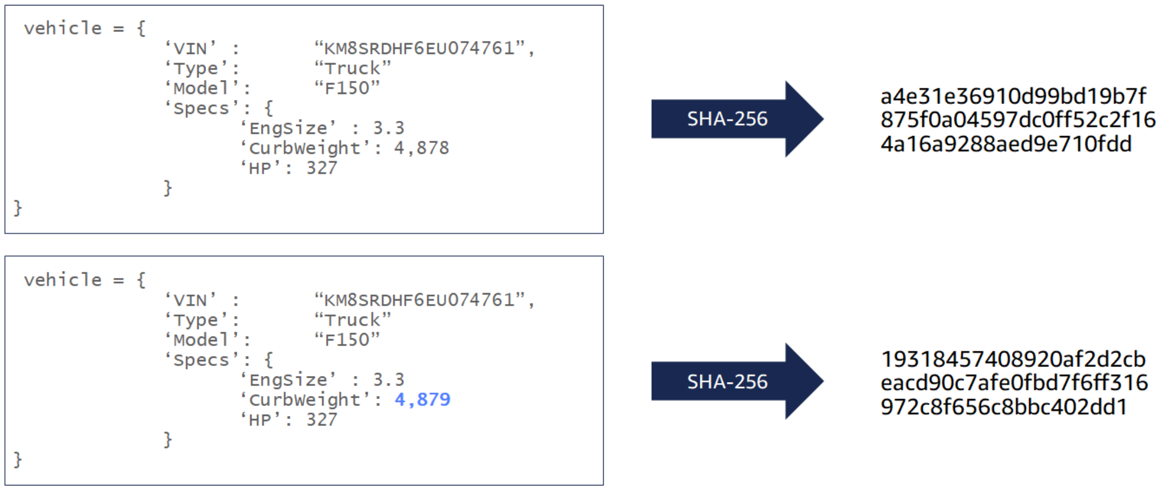
How does verification work?

To understand how verification works in Amazon QLDB, you can break down the concept into four basic components.

- Hashing (p. 339)
- Digest (p. 340)
- Merkle tree (p. 341)
- Proof (p. 341)

Hashing

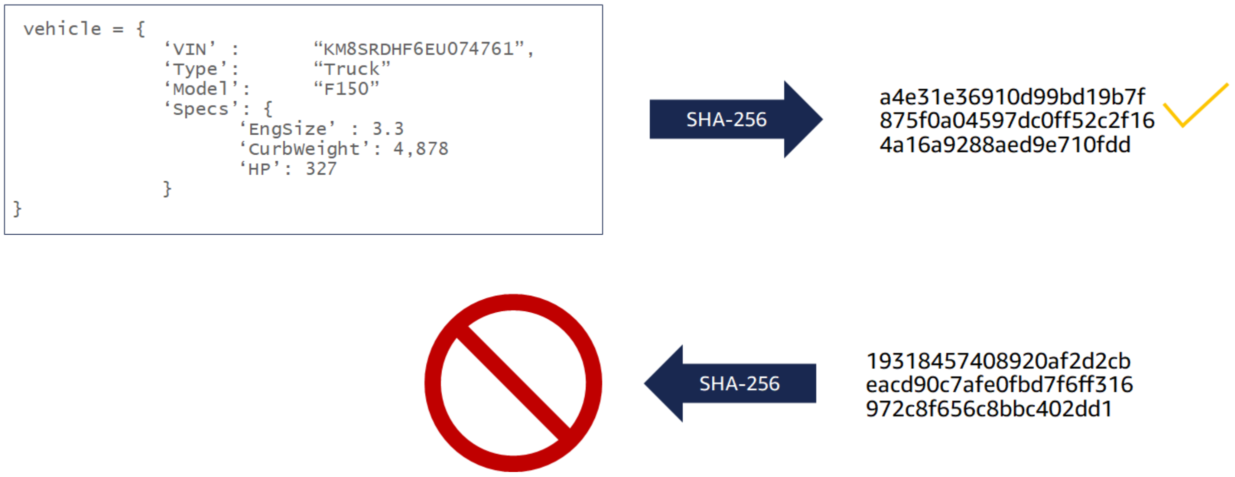
QLDB uses the SHA-256 cryptographic hash function to create 256-bit hash values. A hash acts as a unique, fixed-length signature of any arbitrary amount of input data. If you change any part of the input —even a single character or bit—then the output hash changes completely.



```
vehicle = {
              'VIN' :       "KM8SRDHF6EU074761",
              'Type':       "Truck"
              'Model':      "F150"
              'Specs': {
                     'EngSize' : 3.3
                     'CurbWeight': 4,878
                     'HP': 327
              }
}
```

```
vehicle = {
              'VIN' :       "KM8SRDHF6EU074761",
              'Type':       "Truck"
              'Model':      "F150"
              'Specs': {
                     'EngSize' : 3.3
                     'CurbWeight': 4,879
                     'HP': 327
              }
}
```

The SHA-256 hash function is one-way, meaning that it's not mathematically feasible to compute the input when given an output.



```
vehicle = {
              'VIN' :       "KM8SRDHF6EU074761",
              'Type':       "Truck"
              'Model':      "F150"
              'Specs': {
                     'EngSize' : 3.3
                     'CurbWeight': 4,878
                     'HP': 327
              }
}
```

The following data inputs are hashed in QLDB for verification purposes:

- Document revisions
- PartiQL statements
- Revision entries
- Journal blocks

Digest

A *digest* is a cryptographic representation of your ledger's entire journal at a point in time. A journal is append-only, and journal blocks are sequenced and hash-chained similar to blockchains.

QLDB enables you to generate a digest as a secure output file. Then, you can use that digest to verify the integrity of document revisions that were committed at a prior point in time. If you recalculate hashes by starting with a revision and ending with the digest, you prove that your data has not been altered in between.

Merkle tree

As the size of your ledger grows, it becomes increasingly inefficient to recalculate the journal's full hash chain for verification. QLDB uses a Merkle tree model to address this inefficiency.

A *Merkle tree* is a tree data structure in which each leaf node represents a hash of a data block. Each non-leaf node is a hash of its child nodes. Commonly used in blockchains, a Merkle tree enables efficient verification of large datasets with an audit proof mechanism. For more information about Merkle trees, see the Merkle tree Wikipedia page. To learn more about Merkle audit proofs and for an example use case, see How Log Proofs Work on the Certificate Transparency site.

The QLDB implementation of the Merkle tree is constructed from a journal's full hash chain. In this model, the leaf nodes are the set of all individual document revision hashes. The root node represents the digest of the entire journal as of a point in time.

Using a Merkle audit proof, you can verify a revision by checking only a small subset of your ledger's revision history. You do this by traversing the tree from a given leaf node (revision) to its root (digest). Along this traversal path, you recursively hash sibling pairs of nodes to compute their parent hash until you end with the digest. This traversal has a time complexity of `log(n)` nodes in the tree.

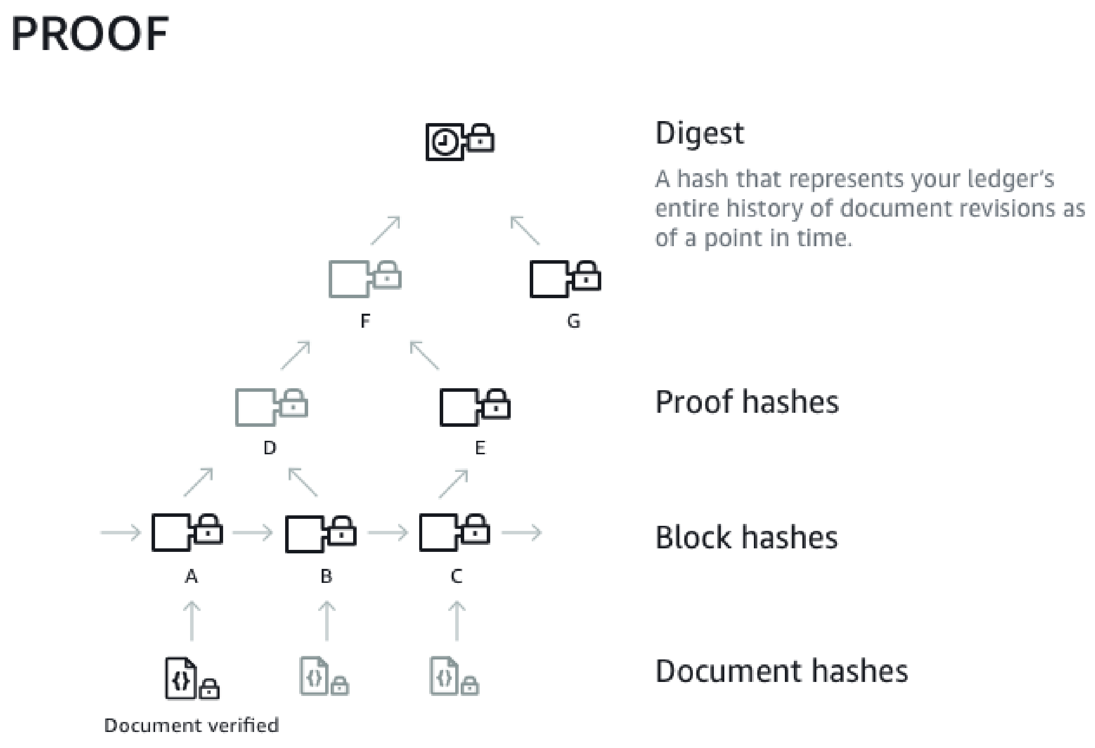
Proof

A *proof* is the ordered list of node hashes that QLDB returns for a given digest and document revision. It consists of the hashes that are required by a Merkle tree model to chain the given leaf node hash (a revision) to the root hash (the digest).

Changing any committed data between a revision and a digest breaks your journal's hash chain and makes it impossible to generate a proof.

Verification example

The following diagram illustrates the Amazon QLDB hash tree model. It shows a set of block hashes that rolls up to the top root node, which represents the digest of a journal strand. In a ledger with a single-strand journal, this root node is also the digest of the entire ledger.


PROOF
Digest
A hash that represents your ledger's entire history of document revisions as of a point in time.
F
G
Proof hashes
D
E
Block hashes
A
B
C
Document hashes
Document verified

Suppose that node **A** is the block that contains the document revision whose hash you want to verify. The following nodes represent the ordered list of hashes that QLDB provides in your proof: **B**, **E**, **G**. These hashes are required to recalculate the digest from hash **A**.

To recalculate the digest, do the following:

1. Start with hash **A** and concatenate it with hash **B**. Then, hash the result to compute **D**.
2. Use **D** and **E** to compute **F**.
3. Use **F** and **G** to compute the digest.

The verification is successful if your recalculated digest matches the expected value. Given a revision hash and a digest, it's not feasible to reverse engineer the hashes in a proof. Therefore, this exercise proves that your revision was indeed written in this journal location relative to the digest.

What is the verification process in QLDB?

Before you can verify data, you must request a digest from your ledger and save it for later. Any document revision that is committed before the latest block covered by the digest is eligible for verification against that digest.

Then, you request a proof from Amazon QLDB for an eligible revision that you want to verify. Using this proof, you call a client-side API to recalculate the digest, starting with your revision hash. As long as the previously saved digest is *known and trusted outside of QLDB*, the integrity of your document is proven if your recalculated digest hash matches the saved digest hash.

Important

- What you are specifically proving is that the document revision was not altered between the time that you saved this digest and when you run the verification. As a best practice, the digest is saved as soon as the document revision that you want to verify is written to the journal.
- We recommend that you request a digest and save it in a secure place at regular intervals. Determine the frequency at which you save digests based on how often you commit revisions in your ledger.

For step-by-step guides on how to request a digest from your ledger and then verify your data, see the following:

- Step 1: Requesting a digest in QLDB (p. 343)
- Step 2: Verifying your data in QLDB (p. 344)

Step 1: Requesting a digest in QLDB

Amazon QLDB provides an API to request a digest that covers the current *tip* of the journal in your ledger. The tip of the journal refers to the latest committed block as of the time that QLDB receives your request.

Topics

- AWS Management Console (p. 343)
- QLDB API (p. 344)

AWS Management Console

Follow these steps to request a digest using the QLDB console.

To request a digest (console)

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Ledgers**.
3. In the list of ledgers, select the ledger name for which you want to request a digest.
4. Choose **Get digest**. The **Get digest** dialog box displays the following digest details:
   - **Digest**—The SHA-256 hash value of the digest that you requested.
   - **Digest tip address**—The latest block location in the journal covered by the digest that you requested. An address has the following two fields:
     - `strandId`—The unique ID of the journal strand that contains the block.
     - `sequenceNo`—The index number that specifies the location of the block within the strand.
   - **Ledger**—The ledger name for which you requested a digest.
   - **Date**—The timestamp when you requested the digest.
5. Review the digest information. Then choose **Save**. You can keep the default file name, or enter a new name.

   Note
   You might notice that your digest hash and tip address values change even when you don't modify any data in your ledger. This is because the console retrieves the ledger's system

catalog each time that you run a query in the *Query editor*. This is a read transaction that gets committed to the journal and causes the latest block address to change.

This step saves a plaintext file with contents in Amazon Ion (p. 487) format. The file has a file name extension of `.ion.txt` and contains all the digest information that was listed on the preceding dialog box. The following is an example of a digest file's contents. The order of the fields can vary depending on your browser.

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

6. Save this file where you can access it in the future. Later, you can use this file as a fingerprint to verify a document revision against.

   Important
   The document revision that you verify later must be covered by the digest that you saved. That is, the sequence number of the document's address must be less than or equal to the sequence number of the **Digest tip address**.

QLDB API

You can also request a digest from your ledger by using the Amazon QLDB API with an AWS SDK or the AWS Command Line Interface (AWS CLI). The QLDB API provides the following action for use by application programs:

- GetDigest – Returns the digest of a ledger at the latest committed block in the journal. The response includes a 256-bit hash value and a block address.

For information about requesting a digest using the AWS CLI, see the get-digest command in the *AWS CLI Command Reference*.

Sample application

For Java code examples, see the GitHub repository aws-samples/amazon-qldb-dmv-sample-java. For instructions on how to download and install this sample application, see Installing the Amazon QLDB Java sample application (p. 45). Before requesting a digest, make sure that you follow Steps 1–3 in the Java tutorial (p. 44) to create a sample ledger and load it with sample data.

The tutorial code in class GetDigest provides an example of requesting a digest from the `vehicle-registration` sample ledger.

To verify a document revision using the digest that you saved, proceed to Step 2: Verifying your data in QLDB (p. 344).

Step 2: Verifying your data in QLDB

Amazon QLDB provides an API to request a proof for a specified document ID and its associated block. You must also provide the tip address of a digest that you previously saved, as described in Step 1: Requesting a digest in QLDB (p. 343).

Then, you can use the proof returned by QLDB to verify the document revision against the saved digest, using a client-side API. This gives you control over the algorithm that you use to verify your data.

Topics

- AWS Management Console (p. 345)
- QLDB API (p. 346)

AWS Management Console

This section describes the steps to verify a document revision against a previously saved digest using the Amazon QLDB console.

Before you start, make sure that you follow the steps in Step 1: Requesting a digest in QLDB (p. 343). Verification requires a previously saved digest that covers the revision that you want to verify.

To verify a document revision (console)

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. First, query your ledger for the `id` and `blockAddress` of the revision that you want to verify. These fields are included in the document's metadata, which you can query in the *committed view*.

   The document `id` is a system-assigned unique ID string. The `blockAddress` is an Ion structure that specifies the block location where the revision was committed.

   In the navigation pane, choose **Query editor**.
3. Choose the ledger name in which you want to verify a revision.
4. In the query editor window, enter a `SELECT` statement in the following syntax, and then choose **Run**.

   ```
   SELECT metadata.id, blockAddress FROM _ql_committed_table
   WHERE criteria
   ```

   For example, the following query returns a document from the `vehicle-registration` sample ledger created in Getting started with the Amazon QLDB console (p. 25).

   ```
   SELECT r.metadata.id, r.blockAddress FROM _ql_committed_VehicleRegistration AS r
   WHERE r.data.VIN = 'KM8SRDHF6EU074761'
   ```
5. Copy and save the `id` and `blockAddress` values that your query returns. Be sure to omit the double quotes for the `id` field. In Amazon Ion (p. 487), string data types are delimited with double quotes. For example, you must copy only the alphanumeric text in the following snippet.

   `"LtMNJYNjSwzBLgf7sLifrG"`
6. Now that you have a document revision selected, you can start the process of verifying it.

   In the navigation pane, choose **Verification**.
7. On the **Verify document** form, under **Specify the document that you want to verify**, enter the following input parameters:

   - **Ledger**—The ledger in which you want to verify a revision.
   - **Block address**—The `blockAddress` value returned by your query in *Step 4*.
   - **Document ID**—The `id` value returned by your query in *Step 4*.
8. Under **Specify the digest to use for verification**, select the digest that you previously saved by choosing **Choose digest**. If the file is valid, this auto-populates all the digest fields on your console. Or, you can manually copy and paste the following values directly from your digest file:

   - **Digest**—The `digest` value from your digest file.

- **Digest tip address**—The `digestTipAddress` value from your digest file.

9. Review your document and digest input parameters, and then choose **Verify**.

   The console automates two steps for you:

   a. Request a proof from QLDB for your specified document.
   b. Use the proof returned by QLDB to call a client-side API, which verifies your document revision against the provided digest. To examine this verification algorithm, see the following section QLDB API (p. 346) to download the example code.

   The console displays the results of your request in the **Verification results** card. For more information, see Verification results (p. 346).

QLDB API

You can also verify a document revision by using the Amazon QLDB API with an AWS SDK or the AWS CLI. The QLDB API provides the following actions for use by application programs:

- `GetDigest` – Returns the digest of a ledger at the latest committed block in the journal. The response includes a 256-bit hash value and a block address.
- `GetBlock` – Returns a block object at a specified address in a journal. Also returns a proof of the specified block for verification if `DigestTipAddress` is provided.
- `GetRevision` – Returns a revision data object for a specified document ID and block address. Also returns a proof of the specified revision for verification if `DigestTipAddress` is provided.

For complete descriptions of these API operations, see the Amazon QLDB API reference (p. 500).

For information about verifying data using the AWS CLI, see the AWS CLI Command Reference.

Sample application

For Java code examples, see the GitHub repository aws-samples/amazon-qldb-dmv-sample-java. For instructions on how to download and install this sample application, see Installing the Amazon QLDB Java sample application (p. 45). Before doing a verification, make sure that you follow Steps 1–3 in the Java tutorial (p. 44) to create a sample ledger and load it with sample data.

The tutorial code in class GetRevision provides an example of requesting a proof for a document revision and then verifying that revision. This class runs the following steps:

1. Requests a new digest from the sample ledger `vehicle-registration`.
2. Requests a proof for a sample document revision from the `VehicleRegistration` table in the `vehicle-registration` ledger.
3. Verifies the sample revision using the returned digest and proof.

Verification results

This section describes the results returned by an Amazon QLDB data verification request on the AWS Management Console. For detailed steps on how to submit a verification request, see Step 2: Verifying your data in QLDB (p. 344).

On the **Verification** page of the QLDB console, the results of your request are displayed in the **Verification results** card. The **Proof** tab shows the contents of the proof returned by QLDB for your specified document revision and digest. It includes the following details:

- **Revision hash**—The SHA-256 value that uniquely represents the document revision that you are verifying.
- **Proof hashes**—The ordered list of hashes provided by QLDB that are used to recalculate the specified digest. The console starts with the **Revision hash** and sequentially combines it with each proof hash until it ends with a recalculated digest.

  The list is collapsed by default, so you can expand it to reveal the hash values. Optionally, you can try the hash calculations yourself by following the steps as described in Using a proof to recalculate your digest (p. 347).
- **Digest calculated**—The hash that resulted from the series of **Hash calculations** that were done on the **Revision hash**. If this value matches your previously saved **Digest**, the verification is successful.

The **Block** tab shows the contents of the block that contains the revision you are verifying. It includes the following details:

- **Transaction ID**—The unique ID of the transaction that committed this block.
- **Transaction time**—The timestamp when this block was committed to the strand.
- **Block hash**—The SHA-256 value that uniquely represents this block and all of its contents.
- **Block address**—The location in your ledger's journal where this block was committed. An address has the following two fields:
  - **Strand ID**—The unique ID of the journal strand that contains this block.
  - **Sequence number**—The index number that specifies the location of this block within the strand.
- **Statements**—The PartiQL statements that were executed to commit entries in this block.

  Note
  If you run parameterized statements programmatically, they are recorded in your journal blocks with bind parameters instead of the literal data. For example, you might see the following statement in a journal block, where the question mark (`?`) is a variable placeholder for the document contents.

  ```
  INSERT INTO Vehicle ?
  ```
- **Document entries**—The document revisions that were committed in this block.

If your request failed to verify the document revision, see Common errors for verification (p. 352) for information about possible causes.

Using a proof to recalculate your digest

After QLDB returns a proof for your document verification request, you have the option of trying the hash calculations yourself. This section describes the steps to recalculate your digest using the proof that is provided.

First, pair your **Revision hash** with the first hash in the **Proof hashes** list. Then, do the following steps.

1. Sort the two hashes. Compare the hashes by their *signed* byte values in little-endian order.
2. Concatenate the two hashes in sorted order.
3. Hash the concatenated pair with an SHA-256 hash generator.
4. Pair your new hash with the next hash in the proof and repeat steps 1–3. After you process the last proof hash, your new hash is your recalculated digest.

If your recalculated digest matches your previously saved digest, your document is successfully verified.

The following Java code is an example that demonstrates these verification steps. See the `verify` method for the starting point. For the QLDB implementation of the hash comparison and sorting logic, see the `hashComparator` method. This class is part of the sample application from the Getting started with the driver (p. 40) tutorial for 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 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;
    }
}
```

Common errors for verification

This section describes runtime errors that are thrown by Amazon QLDB for verification requests.

The following is a list of common exceptions returned by the service. Each exception includes the specific error message, followed by the API actions that can throw it, a short description, and suggestions for possible solutions.

IllegalArgumentException

Message: The provided Ion value is not valid and cannot be parsed.

API: `GetDigest, GetBlock, GetRevision`

Make sure that you provide a valid Amazon Ion (p. 487) value before retrying your request.

IllegalArgumentException

Message: The provided block address is not valid.

API: `GetDigest, GetBlock, GetRevision`

Make sure that you provide a valid block address before retrying your request. A block address is an Amazon Ion structure that has two fields: `strandId` and `sequenceNo`.

For example: `{strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:14}`

IllegalArgumentException

Message: The sequence number of the provided digest tip address is beyond the strand's latest committed record.

API: `GetDigest, GetBlock, GetRevision`

The digest tip address that you provide must have a sequence number less than or equal to the sequence number of the journal strand's latest committed record. Before retrying your request, make sure that you provide a digest tip address with a valid sequence number.

IllegalArgumentException

Message: The Strand ID of the provided block address is not valid.

API: `GetDigest, GetBlock, GetRevision`

The block address that you provide must have a strand ID that matches the journal's strand ID. Before retrying your request, make sure that you provide a block address with a valid strand ID.

IllegalArgumentException

Message: The sequence number of the provided block address is beyond the strand's latest committed record.

API: `GetBlock, GetRevision`

The block address that you provide must have a sequence number less than or equal to the sequence number of the strand's latest committed record. Before retrying your request, make sure that you provide a block address with a valid sequence number.

IllegalArgumentException

Message: The Strand ID of the provided block address must match the Strand ID of the provided digest tip address.

API: `GetBlock, GetRevision`

You can only verify a document revision or block if it exists in the same journal strand as the digest that you provide.

IllegalArgumentException

Message: The sequence number of the provided block address must not be greater than the sequence number of the provided digest tip address.

API: `GetBlock, GetRevision`

You can only verify a document revision or block if it's covered by the digest that you provide. This means that it was committed to the journal before the digest tip address.

IllegalArgumentException

Message: The provided Document ID was not found in the block at the specified block address.

API: `GetRevision`

The document ID that you provide must exist in the block address that you provide. Before retrying your request, make sure that these two parameters are consistent.

Exporting journal data from Amazon QLDB

Amazon QLDB uses an immutable transactional log, known as a *journal*, for data storage. The journal tracks every change to your committed data and maintains a complete and verifiable history of changes over time.

You can access the contents of the journal in your ledger for various purposes including analytics, auditing, data retention, verification, and exporting to other systems. The following sections describe how to export journal blocks (p. 9) from your ledger into an Amazon Simple Storage Service (Amazon S3) bucket in your AWS account. A journal export job writes your data in Amazon S3 as objects in Amazon Ion format (p. 487) (`.ion`).

For information about Amazon S3, see the Amazon Simple Storage Service Developer Guide or the Amazon Simple Storage Service Console User Guide.

Topics

Requesting a journal export in QLDB

Amazon QLDB provides an API to request an export of your journal blocks for a specified date and time range and a specified Amazon Simple Storage Service (Amazon S3) bucket destination.

Topics

AWS Management Console

Follow these steps to submit a journal export request in QLDB using the QLDB console.

To request an export (console)

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Export**.
3. Choose **Create export job**.
4. On the **Create export job** page, enter the following export settings:

   - **Ledger**—The ledger whose journal blocks you want to export.

- **Start date and time**—The inclusive start timestamp in Coordinated Universal Time (UTC) of the range of journal blocks to export. This timestamp must be earlier than the **End date and time**. If you provide a start timestamp that is earlier than the ledger's `CreationDateTime`, QLDB defaults it to the ledger's `CreationDateTime`.
- **End date and time**—The exclusive end timestamp (UTC) of the range of journal blocks to export. This date and time cannot be in the future.
- **Destination for journal blocks**—The Amazon S3 bucket and prefix name in which your export job writes the data objects. Use the following Amazon S3 URI format.

  ```
  s3://AWSDOC-EXAMPLE-BUCKET/prefix/
  ```

  You must specify an S3 bucket name and an optional prefix name for the output objects. The following is an example.

  ```
  s3://AWSDOC-EXAMPLE-BUCKET/journalExport/
  ```

  The bucket name and prefix must both comply with the Amazon S3 naming rules and conventions. For more information about bucket naming, see Bucket restrictions and limitations in the *Amazon S3 Developer Guide*. For more information about key name prefixes, see Object key and metadata.

  Note
  Cross-region exports are not supported. The specified Amazon S3 bucket must be in the same AWS Region as your ledger.
- **S3 Encryption Configuration**—The encryption settings that are used by your export job to write data in an Amazon S3 bucket. To learn more about server-side encryption options in Amazon S3, see Protecting data using server-side encryption in the *Amazon S3 Developer Guide*.
  - **Bucket default encryption**—Use the default encryption settings of the specified Amazon S3 bucket.
  - **AES-256**—Use server-side encryption with Amazon S3 managed keys (SSE-S3).
  - **AWS-KMS**—Use server-side encryption with AWS KMS managed keys (SSE-KMS).

    If you choose this type along with the **Customer managed CMK** option, you must also specify a symmetric customer master key (CMK) in the following Amazon Resource Name (ARN) format.

    ```
    arn:aws:kms:aws-region:account-id:key/key-id
    ```
- **Permission**—The IAM role that grants QLDB write permissions in your Amazon S3 bucket. If applicable, the IAM role must also grant QLDB permissions to use your CMK in AWS KMS. To learn how to create this role, see Export permissions (p. 361).

5. When the settings are as you want them, choose **Create export job**.

   The amount of time it takes for your export job to finish varies depending on the data size. If your request submission is successful, the console returns to the main **Export** page and lists your export jobs with their current status.
6. You can see your export objects on the Amazon S3 console.

   Open the Amazon S3 console at https://console.aws.amazon.com/s3/.

   To learn more about the format of these output objects, see Journal export output in QLDB (p. 357).

Note
Export jobs expire seven days after they complete. For more information, see Export job expiration (p. 357).

QLDB API

You can also request a journal export by using the Amazon QLDB API with an AWS SDK or the AWS Command Line Interface (AWS CLI). The QLDB API provides the following actions for use by application programs:

- `ExportJournalToS3` – Exports journal contents within a date and time range from a given ledger into a specified Amazon S3 bucket. The data is written as objects in Amazon Ion format.
- `DescribeJournalS3Export` – Returns detailed information about a journal export job. The output includes its current status, creation time, and the parameters of your original export request.
- `ListJournalS3Exports` – Returns a list of journal export job descriptions for all ledgers that are associated with the current AWS account and Region. The output of each export job description includes the same details that are returned by `DescribeJournalS3Export`.
- `ListJournalS3ExportsForLedger` – Returns a list of journal export job descriptions for a given ledger. The output of each export job description includes the same details that are returned by `DescribeJournalS3Export`.

For complete descriptions of these API operations, see the Amazon QLDB API reference (p. 500).

For information about exporting journal data using the AWS CLI, see the AWS CLI Command Reference.

Sample application

For Java code examples, see the GitHub repository aws-samples/amazon-qldb-dmv-sample-java. For instructions on how to download and install this sample application, see Installing the Amazon QLDB Java sample application (p. 45). Before requesting an export, make sure that you follow Steps 1–3 in the Java tutorial (p. 44) to create a sample ledger and load it with sample data.

The tutorial code in the following classes provide examples of creating an export, checking the status of an export, and processing the output of an export.

Class	Description
ExportJournal	Exports journal blocks from the `vehicle-registration` sample ledger for a timestamp range of 10 minutes ago until now. Writes the output objects in a specified S3 bucket, or creates a unique bucket if one isn't provided.
DescribeJournalExport	Describes a journal export job for a specified `exportId` in the `vehicle-registration` sample ledger.
ListJournalExports	Returns a list of journal export job descriptions for the `vehicle-registration` sample ledger.
ValidateQldbHashChain	Validates the hash chain of the `vehicle-registration` sample ledger using a given `exportId`. If not provided, requests a new export to use for hash chain validation.

Export job expiration

Journal export jobs expire seven days after they complete. This expiration period is a hard limit and cannot be changed. This means that after an export job expires, you can no longer use the QLDB console or the following API actions to retrieve metadata about the job:

- `DescribeJournalS3Export`
- `ListJournalS3Exports`
- `ListJournalS3ExportsForLedger`

However, this expiration has no impact on the exported data itself. All of the metadata is preserved in the manifest files that are written by your exports. This expiration is designed to provide a smoother experience for the API actions that list journal export jobs. QLDB removes old export jobs to ensure that you only see recent exports without having to parse multiple pages of jobs.

Journal export output in QLDB

An Amazon QLDB journal export job writes two manifest files in addition to the Amazon Ion data objects containing your journal blocks. These are all saved in the Amazon Simple Storage Service (Amazon S3) bucket that you provided in your export request (p. 354). The following sections describe the format and contents of each output object.

Topics

Manifest files

Amazon QLDB creates two manifest files in the provided S3 bucket for each export request. The *initial manifest* file is created as soon as you submit the export request. The *final manifest* file is written after the export is complete. You can use these files to check the status of your export jobs in Amazon S3.

Initial manifest

The initial manifest indicates that your export job has started. It contains the input parameters that you passed to the request. In addition to the Amazon S3 destination and the start and end time parameters for the export, this file also contains an `exportId`. The `exportId` is a unique ID that QLDB assigns to each export job.

The file-naming convention is as follows.

```
s3://AWSDOC-EXAMPLE-BUCKET/prefix/exportId.started.manifest
```

The following is an example of an initial manifest file and its contents.

```
s3://AWSDOC-EXAMPLE-BUCKET/journalExport/8UyXulxccYLAsbN1aon7e4.started.manifest
```

```
{
  ledgerName:"my-ledger",
  exportId:"8UyXulxccYLAsbN1aon7e4",
  inclusiveStartTime:2019-04-15T00:00:00.000Z,
  exclusiveEndTime:2019-04-15T22:00:00.000Z,
  bucket:"AWSDOC-EXAMPLE-BUCKET",
  prefix:"journalExport",
  objectEncryptionType:"NO_ENCRYPTION"
}
```

Final manifest

The final manifest indicates that your export job for a particular journal *strand* has completed. The export job writes a separate final manifest file for each strand.

Note
In Amazon QLDB, a strand is a partition of your ledger's journal. QLDB currently supports journals with a single strand only.

The final manifest includes an ordered list of data object keys that were written during the export. The file naming convention is as follows.

```
s3://AWSDOC-EXAMPLE-BUCKET/prefix/exportId.strandId.completed.manifest
```

The `strandId` is a unique ID that QLDB assigns to the strand. The following is an example of a final manifest file and its contents.

```
s3://AWSDOC-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"
  ]
}
```

Ion data objects

Amazon QLDB writes journal data objects in the text representation of the Amazon Ion format (`.ion`) in the provided Amazon S3 bucket.

Data object names

A journal export job writes these data objects with the following naming convention.

```
s3://AWSDOC-EXAMPLE-BUCKET/prefix/yyyy/mm/dd/hh/strandId.startSn-endSn.ion
```

- The output data of each export job is broken up into chunks.
- `yyyy/mm/dd/hh`—The date and time when you submitted the export request. Objects that are exported within the same hour are grouped under the same Amazon S3 prefix.
- `strandId`—The unique ID of the particular strand that contains the journal block being exported.

- `startSn-endSn`—The sequence number range included in the object. Sequence number specifies the location of a block within a strand.

For example, suppose that you specify the following path.

```
s3://AWSDOC-EXAMPLE-BUCKET/journalExport/
```

Your export job creates an Amazon S3 data object that looks similar to the following.

```
s3://AWSDOC-EXAMPLE-BUCKET/journalExport/2019/04/15/22/JdxjkR9bSYB5jMHWcI464T.1-5.ion
```

Data object contents

Each Ion data object contains journal block objects with the following format.

```
{
  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
    }
  ]
}
```

A *block* is an object that is committed to the journal during a transaction. A block contains transaction metadata along with entries that represent the document revisions that were committed in the transaction and the PartiQL (p. 426) statements that committed them.

The following is an example of a block with sample data. For information about the fields in a block object, see Journal contents in Amazon QLDB (p. 9).

> **Note**
> This block example is for informational purposes only. The hashes shown are not real calculated hash values.

```
{
  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"
      }
    }
  ]
}
```

In the `revisions` field, some revision objects might only contain a `hash` value and no other attributes. These are internal-only system revisions that don't contain user data. An export job includes these revisions in their respective blocks because the hashes of these revisions are part of the journal's full hash chain. The full hash chain is required for cryptographic verification.

Journal export permissions in QLDB

Before submitting a journal export request in Amazon QLDB, you must provide QLDB with write permissions in your specified Amazon Simple Storage Service (Amazon S3) bucket. If you choose AWS Key Management Service (AWS KMS) as the Amazon S3 object encryption type for your export job, you must also provide QLDB with permissions to use your specified symmetric customer master key (CMK). QLDB does not support asymmetric CMKs. For more information, see Using symmetric and asymmetric keys in the *AWS Key Management Service Developer Guide*.

To provide your export job with the necessary permissions, you can make QLDB assume an IAM role with the appropriate permissions policies. An IAM role is an entity within your AWS account that has specific permissions. You can use an IAM role in your account to grant an AWS service permissions to access your account's resources.

In this example, you create a role that allows QLDB to write objects into an Amazon S3 bucket on your behalf. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

If you are exporting a QLDB journal in your AWS account for the first time, you must first create an IAM role with the appropriate policies by doing the following. Otherwise, you can choose a role that you previously created.

Topics

Create a permissions policy

Complete the following steps to create a permissions policy for a QLDB journal export job. This example shows an Amazon S3 bucket policy that grants QLDB permissions to write objects into your specified bucket. If applicable, the example also shows a key policy that allows QLDB to use your symmetric customer master key (CMK) in AWS KMS.

For more information about Amazon S3 bucket policies, see Using bucket policies and user policies in the *Amazon Simple Storage Service Developer Guide*. To learn more about AWS KMS key policies, see Using key policies in AWS KMS in the *AWS Key Management Service Developer Guide*.

Note
Your Amazon S3 bucket and CMK must both be in the same AWS Region as your ledger.

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation column on the left, choose **Policies**.

   If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.
3. At the top of the page, choose **Create policy**.
4. Choose the **JSON** tab.
5. Enter a JSON policy document.
   - If you're using AWS KMS as the Amazon S3 object encryption type and providing your own CMK, use the following example policy document. To use this policy, replace *`AWSDOC-EXAMPLE-BUCKET`*, *`aws-region`*, *`account-id`*, and *`key-id`* in the example with your own information.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBJournalExportS3Permission",
            "Action": [
                "s3:PutObjectAcl",
                "s3:PutObject"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::AWSDOC-EXAMPLE-BUCKET/*"
        },
        {
            "Sid": "QLDBJournalExportKMSPermission",
            "Action": [ "kms:GenerateDatakey" ],
            "Effect": "Allow",
            "Resource": "arn:aws:kms:aws-region:account-id:key/key-id"
        }
    ]
}
```

   - For other encryption types, use the following example policy document. To use this policy, replace *`AWSDOC-EXAMPLE-BUCKET`* in the example with your own Amazon S3 bucket name.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBJournalExportS3Permission",
            "Action": [
                "s3:PutObjectAcl",
                "s3:PutObject"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::AWSDOC-EXAMPLE-BUCKET/*"
        }
    ]
}
```

6. Choose **Review policy**.

   Note

   You can switch between the **Visual editor** and **JSON** tabs at any time. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring in the *IAM User Guide*.

7. On the **Review policy** page, enter a **Name** and an optional **Description** for the policy that you are creating. Review the policy **Summary** to see the permissions that are granted by your policy. Then choose **Create policy** to save your work.

Create an IAM role

After creating a permissions policy for your QLDB journal export job, you can then create an IAM role and attach your policy to it.

To create the service role for an QLDB (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. Choose the **AWS service** role type, and then choose QLDB.
4. Choose the QLDB use case. Then choose **Next: Permissions**.
5. Select the box next to the policy that you created in the previous steps.
6. (Optional) Set a permissions boundary. This is an advanced feature that is available for service roles, but not service-linked roles.

   Expand the **Set permissions boundary** section and choose **Use a permissions boundary to control the maximum role permissions**. IAM includes a list of the AWS managed and customer managed policies in your account. Select the policy to use for the permissions boundary or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see step 4 in the procedure Creating IAM policies in the *IAM User Guide*. After you create the policy, close that tab and return to your original tab to select the policy to use for the permissions boundary.
7. Choose **Next: Tags**.
8. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM Entities in the *IAM User Guide*.
9. Choose **Next: Review**.
10. If possible, enter a role name or role name suffix to help you identify the purpose of this role. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot create roles named both **`PRODROLE`** and **`prodrole`**. Because various entities might reference the role, you cannot edit the name of the role after it has been created.
11. (Optional) For **Role description**, enter a description for the new role.
12. Review the role and then choose **Create role**.

The following JSON document is an example of a trust policy that allows QLDB to assume an IAM role with specific permissions attached to it.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "qldb.amazonaws.com"
            },
            "Action": [ "sts:AssumeRole" ]
        }
    ]
}
```

After creating your IAM role, return to the QLDB console and refresh the **Create export job** page so that it can find your new role.

Common errors for journal export

This section describes runtime errors that are thrown by Amazon QLDB for journal export requests.

The following is a list of common exceptions returned by the service. Each exception includes the specific error message, followed by a short description and suggestions for possible solutions.

IllegalArgumentException

Message: QLDB encountered an error validating S3 configuration: *`errorCode errorMessage`*

A possible cause for this error is that the provided Amazon Simple Storage Service (Amazon S3) bucket doesn't exist in Amazon S3. Or, QLDB doesn't have enough permissions to write objects into your specified Amazon S3 bucket.

Verify that the S3 bucket name that you provide in your export job request is correct. For more information about bucket naming, see Bucket restrictions and limitations in the *Amazon Simple Storage Service Developer Guide*.

Also, verify that you define a policy for your specified bucket that grants `PutObject` and `PutObjectAcl` permissions to the QLDB service (`qldb.amazonaws.com`). To learn more, see Export permissions (p. 361).

IllegalArgumentException

Message: Unexpected response from Amazon S3 while validating the S3 configuration. Response from S3: *`errorCode errorMessage`*

The attempt to write journal export data into the provided S3 bucket failed with the provided Amazon S3 error response. For more information about possible causes, see Troubleshooting Amazon S3 in the *Amazon Simple Storage Service Developer Guide*.

IllegalArgumentException

Message: Amazon S3 bucket prefix must not exceed 128 characters

The prefix provided in the journal export request contains more than 128 characters.

IllegalArgumentException

Message: Start date must not be greater than end date

Both `InclusiveStartTime` and `ExclusiveEndTime` must be in ISO 8601 date and time format and in Coordinated Universal Time (UTC).

IllegalArgumentException

Message: End date cannot be in the future

Both `InclusiveStartTime` and `ExclusiveEndTime` must be in `ISO 8601` date and time format and in UTC.

IllegalArgumentException

Message: The supplied object encryption setting (S3EncryptionConfiguration) is not compatible with an AWS Key Management Service (KMS) key

You provided a `KMSKeyArn` with an `ObjectEncryptionType` of either `NO_ENCRYPTION` or `SSE_S3`. You can only provide a customer AWS KMS key for an object encryption type of `SSE_KMS`. To learn more about server-side encryption options in Amazon S3, see Protecting data using server-side encryption in the *Amazon S3 Developer Guide*.

LimitExceededException

Message: Exceeded the limit of 2 concurrently running Journal export jobs

QLDB enforces a default limit of two concurrent journal export jobs.

Streaming journal data from Amazon QLDB

Amazon QLDB uses an immutable transactional log, known as a *journal*, for data storage. The journal tracks every change to your committed data and maintains a complete and verifiable history of changes over time.

You can create a *stream* in QLDB that captures every document revision that is committed to your journal and delivers this data to Amazon Kinesis Data Streams in near-real time. A QLDB stream is a continuous flow of data from your ledger's journal to a Kinesis data stream resource.

Then, you use the Kinesis streaming platform or the *Kinesis Client Library* to consume your stream, process the data records, and analyze the data contents. A QLDB stream writes your data to Kinesis Data Streams in three types of records: *control*, *block summary*, and *revision details*. For more information, see QLDB stream records in Kinesis (p. 374).

Topics

Common use cases

Streaming lets you use QLDB as a single, verifiable source of truth while integrating your journal data with other services. The following are some of the common use cases supported by QLDB journal streams:

- **Event-driven architecture** – Build applications in an event-driven architectural style with decoupled components. For example, a bank can use AWS Lambda functions to implement a notification system that alerts customers when their account balance drops below a threshold. In such a system, the account balances are maintained in a QLDB ledger, and any balance changes are recorded in the journal. The AWS Lambda function can trigger the notification logic upon consuming a balance update event that is committed to the journal and sent to a Kinesis data stream.
- **Real-time analytics** – Build Kinesis consumer applications that run real-time analytics on event data. With this capability, you can gain insights in near-real time and respond quickly to a changing business environment. For example, an ecommerce website can analyze product sales data and stop advertisements for a discounted product as soon as sales reach a limit.
- **Historical analytics** – Take advantage of the journal-oriented architecture of Amazon QLDB by replaying historical event data. You can choose to start a QLDB stream as of any point in time in the past, in which all revisions since that time are delivered to Kinesis Data Streams. Using this feature, you can build Kinesis consumer applications that run analytics jobs on historical data. For example, an ecommerce website can run ad hoc analytics to generate past sales metrics that were not previously captured.

- **Replication to purpose-built databases** – Connect QLDB ledgers to other purpose-built data stores using QLDB journal streams. For example, use the Kinesis streaming data platform to integrate with Amazon Elasticsearch Service, which can provide full text search capabilities for QLDB documents. You can also build custom Kinesis consumer applications to replicate your journal data to other purpose-built databases that provide different materialized views. For example, replicate to Amazon Aurora for relational data or to Amazon Neptune for graph-based data.

Consuming your stream

Use Kinesis Data Streams to continuously consume, process, and analyze large streams of data records. In addition to Kinesis Data Streams, the Kinesis streaming data platform includes Amazon Kinesis Data Firehose and Amazon Kinesis Data Analytics. You can use this platform to send data records directly to services such as Amazon Elasticsearch Service (Amazon ES), Amazon Redshift, Amazon Simple Storage Service (Amazon S3), or Splunk. For more information, see Kinesis Data Streams consumers in the *Amazon Kinesis Data Streams Developer Guide*.

You can also use the Kinesis Client Library (KCL) to build a stream consumer application to process data records in a custom way. The KCL simplifies coding by providing useful abstractions above the low-level Kinesis Data Streams API. To learn more about the KCL, see Developing consumers using the Kinesis Client Library in the *Amazon Kinesis Data Streams Developer Guide*.

Delivery guarantee

QLDB streams provide an *at-least-once* delivery guarantee. Each data record (p. 374) that is produced by a QLDB stream is delivered to Kinesis Data Streams at least once. The same records can appear in a Kinesis data stream multiple times. So you must have deduplication logic in the consumer application layer if your use case requires it.

There are also no ordering guarantees. In some circumstances, QLDB blocks and revisions can be produced in a Kinesis data stream out of order. For more information, see Handling duplicate and out-of-order records (p. 377).

Getting started with streams

The following is a high-level overview of the steps that are required to get started with streaming journal data to Kinesis Data Streams:

1. Create a Kinesis Data Streams resource. For instructions, see Creating and updating data streams in the *Amazon Kinesis Data Streams Developer Guide*.
2. Create an IAM role that enables QLDB to assume write permissions for the Kinesis data stream. For instructions, see Stream permissions in QLDB (p. 378).
3. Create a QLDB journal stream. For instructions, see Creating and managing streams in QLDB (p. 367).
4. Consume the Kinesis data stream, as described in the previous section Consuming your stream (p. 367). For code examples that show how to use the Kinesis Client Library or AWS Lambda, see Developing with streams in QLDB (p. 372).

Creating and managing streams in QLDB

Amazon QLDB provides API actions to create and manage a stream of journal data from your ledger to Amazon Kinesis Data Streams. The QLDB stream captures every document revision that is committed to your journal and sends it to a Kinesis data stream.

You can use the AWS Management Console, the QLDB API, or the AWS Command Line Interface (AWS CLI) to create a journal stream. In addition, you can also use an AWS CloudFormation template to create streams. For more information, see the AWS::QLDB::Stream resource in the *AWS CloudFormation User Guide*.

Topics

Stream parameters

To create a QLDB journal stream, you must provide the following configuration parameters:

Ledger name

The QLDB ledger whose journal data you want to stream to Kinesis Data Streams.

Stream name

The name that you want to assign to the QLDB journal stream. User-defined names can help identify and indicate the purpose of a stream.

Your stream name must be unique among other *active* streams for a given ledger. Stream names have the same naming constraints as ledger names, as defined in Quotas and limits in Amazon QLDB (p. 588).

In addition to the stream name, QLDB assigns a *stream ID* to each QLDB stream that you create. The stream ID is unique among all streams for a given ledger, regardless of their status.

Start date and time

The date and time from which to start streaming journal data. This value can be any date and time in the past but cannot be in the future.

End date and time

(Optional) The date and time that specifies when the stream ends.

If you create an indefinite stream with no end time, you must manually cancel it to end the stream. You can also cancel an active, finite stream that has not yet reached its specified end date and time.

Destination Kinesis data stream

The Kinesis Data Streams target resource to which your stream writes the data records. To learn how to create a Kinesis data stream, see Creating and updating data streams in the *Amazon Kinesis Data Streams Developer Guide*.

Note
Cross-Region and cross-account streams are not supported. The specified Kinesis data stream must be in the same AWS Region and account as your ledger.

IAM role

The IAM role that enables QLDB to assume write permissions to your Kinesis data stream. Before creating a QLDB journal stream, you must have this role created. To learn how to create it, see Stream permissions (p. 378).

Stream ARN

Every QLDB journal stream is a subresource of a ledger and is uniquely identified by an Amazon Resource Name (ARN). The following is an example ARN of a QLDB stream with a stream ID of `IiPT4brpZCqCq3f4MTHbYy` for a ledger named `exampleLedger`.

```
arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/IiPT4brpZCqCq3f4MTHbYy
```

The following section describes how to create and cancel a QLDB stream using the AWS Management Console.

AWS Management Console

Follow these steps to create or cancel a QLDB stream using the QLDB console.

To create a stream (console)

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Streams**.
3. Choose **Create QLDB stream**.
4. On the **Create QLDB stream** page, enter the following settings:

   - **Stream name** – The name that you want to assign to the QLDB stream.
   - **Ledger** – The ledger whose journal data you want to stream.
   - **Start date and time** – The inclusive timestamp in Coordinated Universal Time (UTC) from which to start streaming journal data. This timestamp defaults to the current date and time. It cannot be in the future and must be earlier than the **End date and time**.
   - **End date and time** – (Optional) The exclusive timestamp (UTC) that specifies when the stream ends. If you keep this parameter blank, the stream runs indefinitely until you cancel it.
   - **Destination stream** – The Kinesis Data Streams target resource to which your stream writes the data records. Use the following ARN format.

     ```
     arn:aws:kinesis:aws-region:account-id:stream/kinesis-stream-name
     ```

     The following is an example.

     ```
     arn:aws:kinesis:us-east-1:123456789012:stream/stream-for-qldb
     ```

     Cross-Region and cross-account streams are not supported. The specified Kinesis data stream must be in the same AWS Region and account as your ledger.
   - **Enable record aggregation in Kinesis Data Streams** – Enables QLDB to publish multiple data records in a single Kinesis Data Streams record. This option allows you to increase the number of records sent per API call. To learn more, see KPL key concepts.
   - **IAM role** – The IAM role that grants QLDB write permissions to your Kinesis data stream. To learn how to create this role, see Stream permissions (p. 378).
   - **Tags** – (Optional) Add metadata to the stream by attaching tags as key-value pairs. You can add tags to your stream to help organize and identify them. For more information, see Tagging Amazon QLDB resources (p. 387).

     Choose **Add tag**, and then enter any key-value pairs as appropriate.
5. When the settings are as you want them, choose **Create QLDB stream**.

If your request submission is successful, the console returns to the main **Streams** page and lists your QLDB streams with their current status.

6. After your stream is active, use Kinesis to process your stream data with a consumer application.

   Open the Kinesis Data Streams console at https://console.aws.amazon.com/kinesis/.

For information about the format of the stream data records, see QLDB stream records in Kinesis (p. 374).

To learn how to handle streams that result in an error, see Handling impaired streams (p. 371).

To cancel a stream (console)

You can't restart a QLDB stream after you cancel it. To resume delivery of your data to Kinesis Data Streams, you can create a new QLDB stream.

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Streams**.
3. In the list of QLDB streams, select the active stream that you want to cancel.
4. Choose **Cancel stream**. Confirm this by entering **`cancel stream`** in the box provided.

For information about using the QLDB API with an AWS SDK to create and manage journal streams, see Developing with streams in QLDB (p. 372).

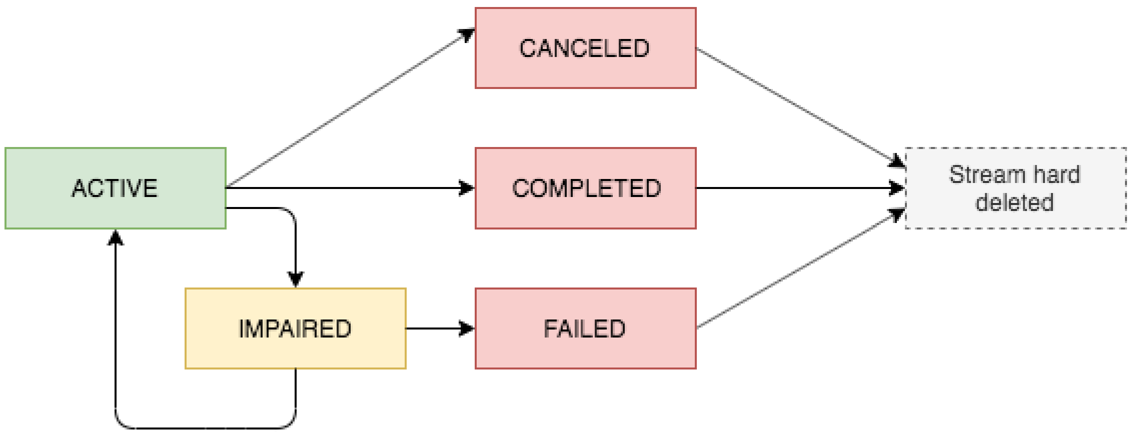
Stream states

The status of a QLDB stream can be one of the following:

- `ACTIVE` – Currently streaming or waiting to stream data (for an indefinite stream with no end time).
- `COMPLETED` – Successfully completed streaming all journal blocks within the specified time range. This is a terminal state.
- `CANCELED` – Ended by a user request before the specified end time and is no longer actively streaming data. This is a terminal state.
- `IMPAIRED` – Unable to write records to Kinesis due to an error that requires your action. This is a recoverable, non-terminal state.

  If you resolve the error within one hour, the stream automatically moves to `ACTIVE` state. If the error remains unresolved after one hour, the stream automatically moves to `FAILED` state.
- `FAILED` – Unable to write records to Kinesis due to an error and is in an unrecoverable, terminal state.

The following diagram illustrates how a QLDB stream resource can transition between states.



Expiration for terminal streams

Stream resources that are in a terminal state (`CANCELED`, `COMPLETED`, and `FAILED`) are subject to a 7-day retention period. They are automatically hard-deleted after this limit expires.

After a terminal stream is deleted, you can no longer use the QLDB console or the QLDB API to describe or list the stream resource.

Handling impaired streams

If your stream encounters an error, it moves to `IMPAIRED` state first. QLDB continues to retry `IMPAIRED` streams for up to one hour.

If you resolve the error within one hour, the stream automatically moves to `ACTIVE` state. If the error remains unresolved after one hour, the stream automatically moves to `FAILED` state.

An impaired or failed stream can have one of the following error causes:

- `KINESIS_STREAM_NOT_FOUND` – The destination Kinesis Data Streams resource doesn't exist. Verify that the Kinesis data stream that you provided in your QLDB stream request is correct. Then, go to Kinesis and create the data stream that you specified.
- `IAM_PERMISSION_REVOKED` – QLDB doesn't have enough permissions to write data records to your specified Kinesis data stream. Verify that you define a policy for your specified Kinesis data stream that grants the QLDB service (`qldb.amazonaws.com`) permissions to the following actions:
  - `kinesis:PutRecord`
  - `kinesis:PutRecords`
  - `kinesis:DescribeStream`
  - `kinesis:ListShards`

Monitoring impaired streams

If a stream becomes impaired, the QLDB console displays a banner that shows details about the stream and the error that it encountered. You can also use the `DescribeJournalKinesisStream` API action to get a stream's status and the underlying error cause.

In addition, you can use Amazon CloudWatch to create an alarm that monitors the `IsImpaired` metric of a stream. For information about monitoring QLDB metrics with CloudWatch, see Amazon QLDB dimensions and metrics (p. 406).

Developing with streams in QLDB

This section summarizes the API actions that you can use with an AWS SDK or the AWS CLI to create and manage journal streams in Amazon QLDB. It also describes the sample applications that demonstrate these operations and use the Kinesis Client Library (KCL) or AWS Lambda to implement a stream consumer.

The KCL enables you to build consumer applications and simplifies coding by providing useful abstractions above the low-level Kinesis Data Streams API. To learn more about the KCL, see Developing consumers using the Kinesis Client Library in the *Amazon Kinesis Data Streams Developer Guide*.

Topics

QLDB journal stream APIs

The QLDB API provides the following journal stream actions for use by application programs:

- `StreamJournalToKinesis` – Creates a journal stream for a given QLDB ledger. The stream captures every document revision that is committed to the ledger's journal and delivers the data to a specified Kinesis Data Streams resource.
- `DescribeJournalKinesisStream` – Returns detailed information about a given QLDB journal stream. The output includes the ARN, stream name, current status, creation time, and the parameters of your original stream creation request.
- `ListJournalKinesisStreamsForLedger` – Returns a list of all QLDB journal stream descriptors for a given ledger. The output of each stream descriptor includes the same details that are returned by `DescribeJournalKinesisStream`.
- `CancelJournalKinesisStream` – Ends a given QLDB journal stream. Before a stream can be canceled, its current status must be `ACTIVE`.

  You can't restart a stream after you cancel it. To resume delivery of your data to Kinesis Data Streams, you can create a new QLDB stream.

For complete descriptions of these API operations, see the Amazon QLDB API reference (p. 500).

For information about creating and managing journal streams using the AWS CLI, see the AWS CLI Command Reference.

Sample applications

QLDB provides sample applications that demonstrate various operations using journal streams. These applications are open sourced on the AWS Samples GitHub site.

Topics

Basic operations (Java)

For a Java code example that demonstrates basic operations for QLDB journal streams, see the GitHub repository aws-samples/amazon-qldb-dmv-sample-java. For instructions on how to download and install this sample application, see Installing the Amazon QLDB Java sample application (p. 45).

> **Note**
> After you install the application, don't proceed to *Step 1* of the *Getting started with the driver* tutorial to create a ledger. This sample application for streaming creates the `vehicle-registration` ledger for you.

This application packages the complete source code from the Java tutorial (p. 44) and its dependencies, including the following modules:

- AWS SDK for Java – To create and delete both QLDB and Kinesis Data Streams resources, including ledgers, QLDB journal streams, and Kinesis data streams.
- Amazon QLDB driver for Java (p. 40) – To execute data transactions on a ledger using PartiQL statements, including creating tables and inserting documents.
- Kinesis Client Library – To consume and process data from a Kinesis data stream.

Running the code

The StreamJournal class contains tutorial code that demonstrates the following operations:

1. Create a ledger named `vehicle-registration`, create tables, and load them with sample data.

   > **Note**
   > Before running this code, make sure that you don't already have an active ledger named `vehicle-registration`.
2. Create a Kinesis data stream, an IAM role that enables QLDB to assume write permissions for the Kinesis data stream, and a QLDB journal stream.
3. Use the KCL to start a stream reader that processes the Kinesis data stream and logs each QLDB data record.
4. Use the stream data to validate the hash chain of the `vehicle-registration` sample ledger.
5. Clean up all resources by stopping the stream reader, canceling the QLDB journal stream, deleting the ledger, and deleting the Kinesis data stream.

To run the `StreamJournal` tutorial code, enter the following Gradle command from your project root directory.

```
./gradlew run -Dtutorial=streams.StreamJournal
```

Integration with Amazon ES (Python)

For a Python sample application that demonstrates how to integrate a QLDB stream with Amazon Elasticsearch Service (Amazon ES), see the GitHub repository aws-samples/amazon-qldb-streaming-amazon-elasticsearch-sample-python. This application uses an AWS Lambda function to implement a Kinesis Data Streams consumer.

To clone the repository, enter the following `git` command.

```
git clone https://github.com/aws-samples/amazon-qldb-streaming-amazon-elasticsearch-sample-
python.git
```

To run the sample application, see the README on GitHub for instructions.

Integration with Amazon SNS and Amazon SQS (Python)

For a Python sample application that demonstrates how to integrate a QLDB stream with Amazon Simple Notification Service (Amazon SNS), see the GitHub repository aws-samples/amazon-qldb-streams-dmv-sample-lambda-python.

This application uses an AWS Lambda function to implement a Kinesis Data Streams consumer. It sends messages to an Amazon SNS topic, which has an Amazon Simple Queue Service (Amazon SQS) queue subscribed to it.

To clone the repository, enter the following `git` command.

```
git clone https://github.com/aws-samples/amazon-qldb-streams-dmv-sample-lambda-python.git
```

To run the sample application, see the README on GitHub for instructions.

QLDB stream records in Kinesis

An Amazon QLDB stream writes three types of data records to a given Amazon Kinesis Data Streams resource: *control*, *block summary*, and *revision details*. All three record types are written in the *binary representation* of the Amazon Ion format (p. 487).

Control records indicate the start and completion of your QLDB streams. Whenever a revision is committed to your journal, a QLDB stream writes all of the associated journal block data in block summary and revision details records.

The three record types are polymorphic. They all consist of a common top-level record that contains the QLDB stream ARN, the record type, and the record payload. This top-level record has the following format.

```
{
  qldbStreamArn: string,
  recordType: string,
  payload: {
    //control | block summary | revision details record
  }
}
```

The `recordType` field can have one of three values:

- `CONTROL`
- `BLOCK_SUMMARY`
- `REVISION_DETAILS`

The following sections describe the format and contents of each individual payload record.

> **Note**
> QLDB writes all stream records to Kinesis Data Streams in the binary representation of Amazon Ion. The following examples are provided in the text representation of Ion to illustrate the record contents in a readable format.

Topics

Control records

A QLDB stream writes *control* records to indicate its start and completion events. The following are examples of control records with sample data for each `controlRecordType`:

- `CREATED` – The first record that a QLDB stream writes to Kinesis to indicate that your newly created stream is active.

```
{
  qldbStreamArn:"arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/
IiPT4brpZCqCq3f4MTHbYy",
  recordType:"CONTROL",
  payload:{
    controlRecordType:"CREATED"
  }
}
```

- `COMPLETED` – The last record that a QLDB stream writes to Kinesis to indicate that your stream has reached the specified end date and time. This record is not written if you cancel the stream.

```
{
  qldbStreamArn:"arn:aws:qldb:us-east-1:123456789012:stream/exampleLedger/
IiPT4brpZCqCq3f4MTHbYy",
  recordType:"CONTROL",
  payload:{
    controlRecordType:"COMPLETED"
  }
}
```

Block summary records

A *block summary* record represents a journal block in which your document revisions are committed. A block (p. 9) is an object that is committed to your QLDB journal during a transaction.

The payload of a block summary record contains the block address, timestamp, and other metadata of the transaction that committed the block. It also includes summary attributes of the revisions in the block and the PartiQL statements that committed them. The following is an example of a block summary record with sample data.

Note
This block summary example is for informational purposes only. The hashes shown are not real calculated hash values.

```
{
  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"
      }
    ]
  }
}
```

In the `revisionSummaries` field, some revisions might not have a `documentId`. These are internal-only system revisions that don't contain user data. A QLDB stream includes these revisions in their respective block summary records because the hashes of these revisions are part of the journal's full hash chain. The full hash chain is required for cryptographic verification.

Only the revisions that do have a document ID are published in separate revision details records, as described in the following section.

Revision details records

A *revision details* record represents a document revision that is committed to your journal. The payload contains all of the attributes from the committed view (p. 323) of the revision, along with the associated table name and table ID. The following is an example of a revision record with sample data.

```
{
  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"
      }
    }
  }
}
```

Handling duplicate and out-of-order records

QLDB streams can publish duplicate and out-of-order records to Kinesis Data Streams. So, a consumer application might need to implement its own logic to identify and handle such scenarios. The block summary and revision details records include fields that you can use for this purpose. Combined with the features of downstream services, these fields can indicate both a unique identity and a strict order for the records.

For example, consider a stream that integrates QLDB with an Elasticsearch index to provide full text search capabilities over documents. In this use case, you need to avoid indexing stale (out-of-order) revisions of a document. To enforce ordering and deduplication, you can use the document ID and external version fields in Elasticsearch, along with the document ID and version fields in a revision details record.

For an example of deduplication logic in a sample application that integrates QLDB with Amazon Elasticsearch Service, see the GitHub repository aws-samples/amazon-qldb-streaming-amazon-elasticsearch-sample-python.

Stream permissions in QLDB

Before creating an Amazon QLDB stream, you must provide QLDB with write permissions to your specified Amazon Kinesis Data Streams resource. You can do this by making QLDB assume an IAM role with the appropriate permissions policies.

An IAM role is an entity within your AWS account that has specific permissions. You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. In this example, you create a role that allows QLDB to write data records to a Kinesis data stream on your behalf. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

If you are streaming a QLDB journal in your AWS account for the first time, you must first create an IAM role with the appropriate policies by doing the following. Otherwise, you can choose a role that you previously created.

Topics

Create a permissions policy

Complete the following steps to create a permissions policy for a QLDB stream. This example shows a Kinesis Data Streams policy that grants QLDB permissions to write data records to your specified Kinesis data stream.

For more information about Kinesis Data Streams policies, see Controlling access to Amazon Kinesis Data Streams resources using IAM in the *Amazon Kinesis Data Streams Developer Guide*.

Note
Your Kinesis data stream must be in the same AWS Region and account as your QLDB ledger.

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane on the left, choose **Policies**.

   If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.
3. At the top of the page, choose **Create policy**.
4. Choose the **JSON** tab.
5. Enter a JSON policy document.

   The following is an example policy document. To use this policy, replace *aws-region*, *account-id*, and *kinesis-stream-name* in the example with your own information.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QLDBStreamKinesisPermissions",
            "Action": [ "kinesis:PutRecord*", "kinesis:DescribeStream",
 "kinesis:ListShards" ],
            "Effect": "Allow",
            "Resource": "arn:aws:kinesis:aws-region:account-id:stream/kinesis-stream-
name"
```

```
        }
    ]
}
```

6. Choose **Review policy**.

   Note
   You can switch between the **Visual editor** and **JSON** tabs at any time. However, if you make changes or choose **Review policy** in the **Visual editor** tab, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring in the *IAM User Guide*.
7. On the **Review policy** page, enter a **Name** and an optional **Description** for the policy that you are creating. Review the policy **Summary** to see the permissions that are granted by your policy. Then choose **Create policy** to save your work.

Create an IAM role

After creating a permissions policy for your QLDB stream, you can then create an IAM role and attach your policy to it.

To create the service role for an QLDB (IAM console)

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. Choose the **AWS service** role type, and then choose QLDB.
4. Choose the QLDB use case. Then choose **Next: Permissions**.
5. Select the box next to the policy that you created in the previous steps.
6. (Optional) Set a permissions boundary. This is an advanced feature that is available for service roles, but not service-linked roles.

   Expand the **Set permissions boundary** section and choose **Use a permissions boundary to control the maximum role permissions**. IAM includes a list of the AWS managed and customer managed policies in your account. Select the policy to use for the permissions boundary or choose **Create policy** to open a new browser tab and create a new policy from scratch. For more information, see step 4 in the procedure Creating IAM policies in the *IAM User Guide*. After you create the policy, close that tab and return to your original tab to select the policy to use for the permissions boundary.
7. Choose **Next: Tags**.
8. (Optional) Add metadata to the user by attaching tags as key-value pairs. For more information about using tags in IAM, see Tagging IAM Entities in the *IAM User Guide*.
9. Choose **Next: Review**.
10. If possible, enter a role name or role name suffix to help you identify the purpose of this role. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot create roles named both **`PRODROLE`** and **`prodrole`**. Because various entities might reference the role, you cannot edit the name of the role after it has been created.
11. (Optional) For **Role description**, enter a description for the new role.
12. Review the role and then choose **Create role**.

The following JSON document is an example of a trust policy that allows QLDB to assume an IAM role with specific permissions attached to it.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "qldb.amazonaws.com"
            },
            "Action": [ "sts:AssumeRole" ]
        }
    ]
}
```

After creating your IAM role, return to the QLDB console and refresh the **Create QLDB stream** page so that it can find your new role.

Common errors for journal streams in QLDB

This section describes runtime errors that are thrown by Amazon QLDB for journal stream requests.

The following is a list of common exceptions returned by the service. Each exception includes the specific error message, followed by a short description and suggestions for possible solutions.

IllegalArgumentException

Message: QLDB encountered an error validating Kinesis Data Streams: Response from Kinesis: *`errorCode errorMessage`*

A possible cause for this error is that the provided Amazon Kinesis Data Streams resource doesn't exist. Or, QLDB doesn't have enough permissions to write data records to your specified Kinesis data stream.

Verify that the Kinesis data stream that you provide in your stream request is correct. For more information, see Creating and updating data streams in the *Amazon Kinesis Data Streams Developer Guide*.

Also, verify that you define a policy for your specified Kinesis data stream that grants the QLDB service (`qldb.amazonaws.com`) permissions to the following actions. For more information, see Stream permissions (p. 378).

- `kinesis:PutRecord`
- `kinesis:PutRecords`
- `kinesis:DescribeStream`
- `kinesis:ListShards`

IllegalArgumentException

Message: Unexpected response from Kinesis Data Streams while validating the Kinesis configuration. Response from Kinesis: *`errorCode errorMessage`*

The attempt to write data records to the provided Kinesis data stream failed with the provided Kinesis error response. For more information about possible causes, see Troubleshooting Amazon Kinesis Data Streams producers in the *Amazon Kinesis Data Streams Developer Guide*.

IllegalArgumentException

Message: Start date must not be greater than end date.

Both `InclusiveStartTime` and `ExclusiveEndTime` must be in ISO 8601 date and time format and in Coordinated Universal Time (UTC).

IllegalArgumentException

Message: Start date cannot be in the future.

Both `InclusiveStartTime` and `ExclusiveEndTime` must be in `ISO 8601` date and time format and in UTC.

LimitExceededException

Message: Exceeded the limit of 5 concurrently running Journal streams to Kinesis Data Streams

QLDB enforces a default limit of five concurrent journal streams.

Ledger management in Amazon QLDB

This section describes how to use the QLDB API and the AWS Command Line Interface (AWS CLI) to do ledger management operations in Amazon QLDB.

You can also perform these same tasks using the AWS Management Console. For more information, see Accessing Amazon QLDB using the console (p. 18).

Topics

Basic operations for Amazon QLDB ledgers

You can use the QLDB API or the AWS Command Line Interface (AWS CLI) to create, update, and delete ledgers in Amazon QLDB. You can also list all the ledgers in your account, or get information about a specific ledger.

The following are the common steps for ledger operations using the AWS SDK for Java and the AWS CLI. For full Java code examples that demonstrate these operations, see the GitHub repository aws-samples/amazon-qldb-dmv-sample-java. For instructions on how to download and install this complete sample application, see Installing the Amazon QLDB Java sample application (p. 45). Or, follow the step-by-step instructions in the Getting started with the driver (p. 40) tutorial for Java.

Topics

Creating a ledger

Use the `CreateLedger` operation to create a ledger in your AWS account. You must provide the following information:

- **Ledger name** — The name of the ledger that you want to create in your account. The name must be unique among all of your ledgers in the current AWS Region.

  Naming constraints for ledger names are defined in Quotas and limits in Amazon QLDB (p. 588).
- **Permissions mode** — The permissions mode to assign to the ledger. The only permissions mode that is currently supported for a ledger is `ALLOW_ALL`.
- **Deletion protection** — (Optional) The flag that prevents a ledger from being deleted by any user. If not provided on ledger creation, this feature is enabled (`true`) by default.

If deletion protection is enabled, you must first disable it before you can delete the ledger using the QLDB API or the AWS CLI. You can disable it by using the `UpdateLedger` operation to set the flag to `false`.

- **Tags** — (Optional) Add metadata to the ledger by attaching tags as key-value pairs. You can add tags to your ledger to help organize and identify them. For more information, see Tagging Amazon QLDB resources (p. 387).

The ledger is not ready for use until QLDB creates it and sets its status to `ACTIVE`.

Using AWS CloudFormation

In addition to the following examples, you can also use an AWS CloudFormation template to create ledgers. For more information, see the AWS::QLDB::Ledger resource in the *AWS CloudFormation User Guide*.

Creating a ledger (Java)

To create a ledger using the AWS SDK for Java

1. Create an instance of the `AmazonQLDB` class.
2. Create an instance of the `CreateLedgerRequest` class to provide the request information.

   You must provide the ledger name and a permissions mode.
3. Execute the `createLedger` method by providing the request object as a parameter.

The `createLedger` request returns a `CreateLedgerResult` object that has information about the ledger. See the next section for an example of using the `DescribeLedger` operation to check your ledger's status after you create it.

The following examples demonstrate the preceding steps.

Example

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
CreateLedgerRequest request = new CreateLedgerRequest()
        .withName(ledgerName)
        .withPermissionsMode(PermissionsMode.ALLOW_ALL);
CreateLedgerResult result = client.createLedger(request);
```

Note
Deletion protection is enabled by default if you don't specify it.

Example — Disable deletion protection and attach specified tags

```
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.ALLOW_ALL)
        .withDeletionProtection(false)
        .withTags(tags);
CreateLedgerResult result = client.createLedger(request);
```

Creating a ledger (AWS CLI)

Create a new ledger named `vehicle-registration`.

Example

```
aws qldb create-ledger --name vehicle-registration --permissions-mode ALLOW_ALL
```

> **Note**
> Deletion protection is enabled by default if you don't specify it.

Or, create a new ledger named `vehicle-registration` with deletion protection disabled and with specified tags.

Example

```
aws qldb create-ledger --name vehicle-registration --no-deletion-protection --permissions-
mode ALLOW_ALL --tags IsTest=true,Domain=Test
```

Describing a ledger

Use the `DescribeLedger` operation to view details about a ledger. You must provide the ledger name. The output from `DescribeLedger` is in the same format as that from `CreateLedger`. It includes the following information:

- **Ledger name** — The name of the ledger that you want to describe.
- **ARN** — The Amazon Resource Name (ARN) for the ledger in the following format.

  ```
  arn:aws::qldb:aws-region:account-id:ledger/ledger-name
  ```
- **Deletion protection** — The flag that indicates whether the deletion protection feature is enabled.
- **Creation date and time** — The date and time, in epoch time format, when the ledger was created.
- **State** — The current status of the ledger. This can be one of the following values:
  - `CREATING`
  - `ACTIVE`
  - `DELETING`

> **Note**
> After you create a QLDB ledger, it becomes ready for use when its status changes from `CREATING` to `ACTIVE`.

Describing a ledger (Java)

To describe a ledger using the AWS SDK for Java

1. Create an instance of the `AmazonQLDB` class. Or, you can use the same instance of the `AmazonQLDB` client that you instantiated for the `CreateLedger` request.
2. Create an instance of the `DescribeLedgerRequest` class and provide the ledger name that you want to describe.
3. Execute the `describeLedger` method by providing the request object as a parameter.
4. The `describeLedger` request returns a `DescribeLedgerResult` object that has current information about the ledger.

The following code example demonstrates the preceding steps. You can call the `describeLedger` method of the client to get ledger information at any time.

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",
        result.getName(),
        result.getArn(),
        result.getState(),
        result.getCreationDateTime(),
        result.getDeletionProtection());
```

Describing a ledger (AWS CLI)

Describe the `vehicle-registration` ledger that you just created.

Example

```
aws qldb describe-ledger --name vehicle-registration
```

Updating a ledger

The `UpdateLedger` operation currently enables you to change the `DeletionProtection` flag only for an existing ledger. The output from `UpdateLedger` is in the same format as that from `CreateLedger`.

Updating a ledger (Java)

To update a ledger using the AWS SDK for Java

1. Create an instance of the `AmazonQLDB` class.
2. Create an instance of the `UpdateLedgerRequest` class to provide the request information.

   You must provide the ledger name and a new Boolean value for deletion protection.
3. Execute the `updateLedger` method by providing the request object as a parameter.

The following code example demonstrates the preceding steps. The `updateLedger` request returns an `UpdateLedgerResult` object that has updated information about the ledger.

Example

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
UpdateLedgerRequest request = new UpdateLedgerRequest()
        .withName(ledgerName)
        .withDeletionProtection(false);
UpdateLedgerResult result = client.updateLedger(request);
```

Updating a ledger (AWS CLI)

If your `vehicle-registration` ledger has deletion protection enabled, you must first disable it before you can delete it.

Example

```
aws qldb update-ledger --name vehicle-registration --no-deletion-protection
```

Deleting a ledger

Use the `DeleteLedger` operation to delete a ledger and all of its contents. Deleting a ledger is an unrecoverable operation.

If deletion protection is enabled for your ledger, you must first disable it before you can delete the ledger using the QLDB API or the AWS CLI. When you use the QLDB console to delete a ledger, the console disables deletion protection for you.

When you issue a `DeleteLedger` request, the ledger's state changes from `ACTIVE` to `DELETING`. It might take a while to delete the ledger, depending on the amount of storage it uses. When the `DeleteLedger` operation concludes, the ledger no longer exists in QLDB.

Deleting a ledger (Java)

To delete a ledger using the AWS SDK for Java

1. Create an instance of the `AmazonQLDB` class.
2. Create an instance of the `DeleteLedgerRequest` class and provide the ledger name that you want to delete.
3. Execute the `deleteLedger` method by providing the request object as a parameter.

The following code example demonstrates the preceding steps.

Example

```
AmazonQLDB client = AmazonQLDBClientBuilder.standard().build();
DeleteLedgerRequest request = new DeleteLedgerRequest().withName(ledgerName);
DeleteLedgerResult result = client.deleteLedger(request);
```

Deleting a ledger (AWS CLI)

Delete your `vehicle-registration` ledger.

Example

```
aws qldb delete-ledger --name vehicle-registration
```

Listing ledgers

The `ListLedgers` operation returns summary information of all QLDB ledgers for the current AWS account and Region.

Listing ledgers (Java)

To list ledgers in your account using the AWS SDK for Java

1. Create an instance of the `AmazonQLDB` class.
2. Create an instance of the `ListLedgersRequest` class.

If you received a value for `NextToken` in the response from a previous `ListLedgers` call, you must provide that value in this request. This enables you to get the next page of results.

3. Execute the `listLedgers` method by providing the request object as a parameter.
4. The `listLedgers` request returns a `ListLedgersResult` object. This object has a list of `LedgerSummary` objects and a pagination token that indicates whether there are more results available:

   - If `NextToken` is empty, the last page of results has been processed and there are no more results.
   - If `NextToken` is not empty, there are more results available. To retrieve the next page of results, use the value of `NextToken` in a subsequent `ListLedgers` call.

The following code example demonstrates the preceding steps.

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);
```

Listing ledgers (AWS CLI)

List all ledgers in the current AWS account and Region.

Example

```
aws qldb list-ledgers
```

Tagging Amazon QLDB resources

A *tag* is a custom attribute label that you assign or that AWS assigns to an AWS resource. Each tag has two parts:

- A *tag key* (for example, `CostCenter`, `Environment`, or `Project`). Tag keys are case sensitive.
- An optional field known as a *tag value* (for example, `111122223333` or `Production`). Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case sensitive.

Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you could assign the same tag to an Amazon QLDB ledger that you assign to an Amazon Simple Storage Service (Amazon S3) bucket.
- Track your AWS costs. You activate these tags on the AWS Billing and Cost Management dashboard. AWS uses the tags to categorize your costs and deliver a monthly cost allocation report to you. For more information, see Using cost allocation tags in the AWS Billing and Cost Management User Guide.

- Control access to your AWS resources with AWS Identity and Access Management (IAM). For information, see Authorization based on QLDB tags (p. 397) in this developer guide and Control access using IAM tags in the *IAM User Guide*.

For tips on using tags, see the AWS Tagging Strategies post on the *AWS Answers* blog.

The following sections provide more information about tags for Amazon QLDB.

Topics

Supported resources in Amazon QLDB

The following resources in Amazon QLDB support tagging:

- ledger
- journal stream

For information about adding and managing tags, see Managing tags (p. 388).

Tag naming and usage conventions

The following basic naming and usage conventions apply to using tags with Amazon QLDB resources:

- Each resource can have a maximum of 50 tags.
- For each resource, each tag key must be unique, and each tag key can have only one value.
- The maximum tag key length is 128 Unicode characters in UTF-8.
- The maximum tag value length is 256 Unicode characters in UTF-8.
- Allowed characters are letters, numbers, spaces representable in UTF-8, and the following characters: ***. : + = @ _ / -*** (hyphen).
- Tag keys and values are case sensitive. As a best practice, decide on a strategy for capitalizing tags, and consistently implement that strategy across all resource types. For example, decide whether to use `Costcenter`, `costcenter`, or `CostCenter`, and use the same convention for all tags. Avoid using similar tags with inconsistent case treatment.
- The `aws:` prefix is reserved for AWS use. You can't edit or delete a tag's key or value when the tag has a tag key with the `aws:` prefix. Tags with this prefix do not count against your tags per resource limit.

Managing tags

Tags are made up of the `Key` and `Value` properties on a resource. You can use the Amazon QLDB console, the AWS CLI, or the QLDB API to add, edit, or delete the values for these properties. You can also use the AWS Resource Groups Tag Editor to manage tags.

For information about working with tags, see the following API actions:

- ListTagsForResource in the *Amazon QLDB API Reference*
- TagResource in the *Amazon QLDB API Reference*
- UntagResource in the *Amazon QLDB API Reference*

To use the QLDB tagging panel (console)

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Ledgers**.
3. In the list of **Ledgers**, choose the ledger name whose tags you want to manage.
4. On the ledger details page, locate the **Tags** card and choose **Manage tags**.
5. On the **Manage tags** page, you can add, edit, or remove any tags as appropriate for your ledger. When the tag keys and values are as you want them, choose **Save**.

Security in Amazon QLDB

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to Amazon QLDB, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using QLDB. The following topics show you how to configure QLDB to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your QLDB resources.

Topics

Data protection in Amazon QLDB

Amazon QLDB conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM), so that each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with QLDB or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into QLDB or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

Data encryption in Amazon QLDB

By default, all Amazon QLDB data in transit and at rest is encrypted. Data in transit is encrypted using TLS, and data at rest is encrypted using AWS managed keys. QLDB does not currently support customer managed CMKs (customer master keys).

Identity and access management in Amazon QLDB

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use QLDB resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in QLDB.

Service user – If you use the QLDB service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more QLDB features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in QLDB, see Troubleshooting Amazon QLDB identity and access (p. 401).

Service administrator – If you're in charge of QLDB resources at your company, you probably have full access to QLDB. It's your job to determine which QLDB features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with QLDB, see How Amazon QLDB works with IAM (p. 395).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to QLDB. To view example QLDB identity-based policies that you can use in IAM, see Amazon QLDB identity-based policy examples (p. 397).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see The IAM Console and Sign-in Page in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don’t use AWS tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using Multi-Factor Authentication (MFA) in AWS in the *IAM User Guide*.

AWS Account Root User

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing Access Keys for IAM Users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to Create an IAM User (Instead of a Role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM Roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.

- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.
- **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances in the *IAM User Guide*.

To learn whether to use IAM roles, see When to Create an IAM Role (Instead of a User) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON Policies in the *IAM User Guide*.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM Policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing Between Managed Policies and Inline Policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

QLDB does not support resource-based policies.

Access control lists

Access control lists (ACLs) are a type of policy that controls which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format. Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access Control List (ACL) Overview in the *Amazon Simple Storage Service Developer Guide*.

QLDB does not support ACLs.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions Boundaries for IAM Entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs Work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session Policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy Evaluation Logic in the *IAM User Guide*.

How Amazon QLDB works with IAM

Before you use AWS Identity and Access Management (IAM) to manage access to QLDB, you should understand what IAM features are available to use with QLDB. To get a high-level view of how QLDB and other AWS services work with IAM, see AWS services that work with IAM in the *IAM User Guide*.

Topics

QLDB identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources in addition to the conditions under which actions are allowed or denied. QLDB supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Actions

The `Action` element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in QLDB use a *`qldb:`* prefix before the action. For example, to grant someone permission to describe a QLDB ledger with the QLDB `DescribeLedger` API operation, you include the `qldb:DescribeLedger` action in their policy. Policy statements must include either an `Action` element or a `NotAction` element. QLDB defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows.

```
"Action": [
      "qldb:CreateLedger",
      "qldb:UpdateLedger"
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action.

```
"Action": "qldb:Describe*"
```

To interact with the QLDB transactional data API by executing PartiQL (p. 426) statements on a ledger, you must grant permission to the `SendCommand` action as follows.

```
"Action": "qldb:SendCommand"
```

To see a list of QLDB actions, see Actions defined by Amazon QLDB in the *IAM User Guide*.

Resources

The `Resource` element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. You specify a resource using an ARN or using the wildcard (*) to indicate that the statement applies to all resources.

QLDB defines the following resource types and their ARNs.

Resource Type	ARN
ledger	`arn:${Partition}:qldb:${Region}:${Account}:ledger/${LedgerName}`
stream	`arn:${Partition}:qldb:${Region}:${Account}:stream/${LedgerName}/${StreamId}`

For more information about the format of ARNs, see Amazon Resource Names (ARNs).

For example, to specify the `exampleLedger` resource in your statement, use the following ARN.

```
"Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger"
```

To specify all ledgers that belong to a specific account, use the wildcard (*).

```
"Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/*"
```

Some QLDB actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

To specify multiple resources in a single statement, separate the ARNs with commas.

```
"Resource": [
      "arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger1",
      "arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger2"
```

To see a list of QLDB resource types and their ARNs, see Resources defined by Amazon QLDB in the *IAM User Guide*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon QLDB.

Condition keys

QLDB does not provide any service-specific condition keys, but it does support using some global condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

Examples

To view examples of QLDB identity-based policies, see Amazon QLDB identity-based policy examples (p. 397).

QLDB resource-based policies

QLDB does not support resource-based policies.

Authorization based on QLDB tags

You can attach tags to QLDB resources or pass tags in a request to QLDB. To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, or `aws:TagKeys` condition keys. For more information about tagging QLDB resources, see Tagging Amazon QLDB resources (p. 387).

To view an example of an identity-based policy for limiting access to a resource based on the tags on that resource, see Updating QLDB ledgers based on tags (p. 401).

QLDB IAM roles

An IAM role is an entity within your AWS account that has specific permissions.

Using temporary credentials with QLDB

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS Security Token Service (AWS STS) API operations such as AssumeRole or GetFederationToken.

QLDB supports using temporary credentials.

Service-linked roles

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but cannot edit the permissions for service-linked roles.

QLDB does not support service-linked roles.

Service roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

QLDB supports service roles for the `ExportJournalToS3` and `StreamJournalToKinesis` API operations, as described in the following section.

Choosing an IAM role in QLDB

When you export or stream journal blocks in QLDB, you must choose a role to allow QLDB to write objects to the given destination on your behalf. If you have previously created a service role, then QLDB provides you with a list of roles to choose from. Make sure that you choose a role that allows access to write into your specified Amazon S3 bucket for an export, or to your specified Amazon Kinesis Data Streams resource for a stream. For more information, see Journal export permissions in QLDB (p. 361) or Stream permissions in QLDB (p. 378).

Amazon QLDB identity-based policy examples

By default, IAM users and roles don't have permission to create or modify QLDB resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating policies on the JSON tab in the *IAM User Guide*.

Topics

Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete QLDB resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get Started Using AWS Managed Policies** – To start using QLDB quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get Started Using Permissions With AWS Managed Policies in the *IAM User Guide*.
- **Grant Least Privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant Least Privilege in the *IAM User Guide*.
- **Enable MFA for Sensitive Operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using Multi-Factor Authentication (MFA) in AWS in the *IAM User Guide*.
- **Use Policy Conditions for Extra Security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON Policy Elements: Condition in the *IAM User Guide*.

Using the QLDB console

To access the Amazon QLDB console and use all of its features, you must have full access with read and write permissions to the QLDB resources in your AWS account. In addition to these QLDB permissions, the console requires permissions from the *Database Query Metadata Service* (`dbqms`). This service provides your recent and saved queries for the *Query editor* on the QLDB console. For more information, see DBQMS API reference (p. 585).

If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy. To ensure that those entities can still use the QLDB console, also attach the following AWS managed policy to the entities, as described in AWS managed (predefined) policies for Amazon QLDB (p. 399).

```
AmazonQLDBConsoleFullAccess
```

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform. You can also use one of the AWS managed policies described in the following section.

AWS managed (predefined) policies for Amazon QLDB

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant necessary permissions for common use cases so that you can avoid having to investigate what permissions are needed. For more information, see AWS managed policies and Adding permissions to a user in the *IAM User Guide*.

The following AWS managed policies, which you can attach to users in your account, are specific to QLDB and are grouped by use case scenario:

- **AmazonQLDBReadOnly** — Grants read-only access to all QLDB resources.
- **AmazonQLDBFullAccess** — Grants full access to all QLDB resources by using the QLDB API or the AWS CLI.
- **AmazonQLDBConsoleFullAccess** — Grants full access to all QLDB resources by using the AWS Management Console.

Note
You can review these permissions policies by signing in to the IAM console and searching for specific policies there.

You can also create your own custom IAM policies to allow permissions for QLDB actions and resources. You can attach these custom policies to the IAM users or groups that require those permissions.

Executing data transactions

To interact with the QLDB transactional data API by executing PartiQL (p. 426) statements on a ledger, you must grant permission to the `SendCommand` action. The following JSON document is an example of a policy that grants permission to only the `SendCommand` action on the ledger `exampleLedger`. You can also use one of the AWS managed policies, as described in the previous section (p. 399), to grant full access to all QLDB resources.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "QLDBDataTransactionPermission",
         "Effect": "Allow",
         "Action": "qldb:SendCommand",
         "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger"
      }
   ]
}
```

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "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": "*"
        }
    ]
}
```

Exporting a journal to an Amazon S3 bucket

In this example, you want to grant an IAM user in your AWS account access to write into one of your Amazon S3 buckets, *AWSDOC-EXAMPLE-BUCKET*. This is required for a QLDB journal export job.

In addition to granting the `s3:PutObject` permission to the user, the policy also grants `s3:PutObjectAcl` for the ability to set the access control list (ACL) permissions for an object. For an example walkthrough that grants permissions to users and tests them using the console, see An example walkthrough: Using user policies to control access to your bucket.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "QLDBJournalExportS3Permission",
         "Effect": "Allow",
         "Action": [
            "s3: PutObject",
            "s3: PutObjectAcl"
         ],
         "Resource": "arn:aws:s3:::AWSDOC-EXAMPLE-BUCKET/*"
      }
   ]
}
```

Then, you attach this permissions policy to an IAM role that QLDB can assume to access your Amazon S3 bucket. The following JSON document is an example of a trust policy that allows QLDB to assume an IAM role with specific permissions attached to it.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "qldb.amazonaws.com"
            },
            "Action": [ "sts:AssumeRole" ]
        }
    ]
}
```

Updating QLDB ledgers based on tags

You can use conditions in your identity-based policy to control access to QLDB resources based on tags. This example shows how you might create a policy that allows updating a ledger. However, permission is granted only if the ledger tag `Owner` has the value of that user's user name. This policy also grants the permissions necessary to complete this action on the console.

```
{
    "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}"}
            }
        }
    ]
}
```

You can attach this policy to the IAM users in your account. If a user named `richard-roe` tries to view a QLDB ledger, the ledger must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise he is denied access. The condition tag key `Owner` matches both `Owner` and `owner` because condition key names are not case sensitive. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

Troubleshooting Amazon QLDB identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with QLDB and IAM.

Topics

I Am not authorized to perform an action in QLDB

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a ledger but does not have `qldb:DescribeLedger` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 qldb:DescribeLedger on resource: exampleLedger
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `exampleLedger` resource using the `qldb:DescribeLedger` action.

I Am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to QLDB.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in QLDB. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing Access Keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access QLDB

To allow others to access QLDB, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in QLDB.

To get started right away, see Creating Your First IAM Delegated User and Group in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my QLDB resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether QLDB supports these features, see How Amazon QLDB works with IAM (p. 395).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing Access to an IAM User in Another AWS Account That You Own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing Access to AWS Accounts Owned by Third Parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing Access to Externally Authenticated Users (Identity Federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the *IAM User Guide*.

Logging and monitoring in Amazon QLDB

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon QLDB and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. However, before you start monitoring QLDB, you should create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?
- How often will you monitor these resources?
- What monitoring tools will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something goes wrong?

The next step is to establish a baseline for normal QLDB performance in your environment, by measuring performance at various times and under different load conditions. As you monitor QLDB, store historical monitoring data so that you can compare it with current performance data, identify normal performance patterns and performance anomalies, and devise methods to address issues.

To establish a baseline you should, at a minimum, monitor the following items:

- Read and write I/Os and storage, so that you can track your ledger's consumption patterns for billing purposes.
- Command latency, so that you can track your ledger's performance when executing data operations.
- Exceptions, so that you can determine whether any requests resulted in an error.

Topics

Monitoring tools

AWS provides various tools that you can use to monitor Amazon QLDB. You can configure some of these tools to do the monitoring for you, while some of the tools require manual intervention. We recommend that you automate monitoring tasks as much as possible.

Topics

Automated monitoring tools

You can use the following automated monitoring tools to watch QLDB and report when something is wrong:

- **Amazon CloudWatch Alarms** – Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Amazon EC2 Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods. For more information, see Monitoring with Amazon CloudWatch (p. 405).
- **Amazon CloudWatch Logs** – Monitor, store, and access your log files from AWS CloudTrail or other sources. For more information, see Monitoring log files in the *Amazon CloudWatch User Guide*.
- **Amazon CloudWatch Events** – Match events and route them to one or more target functions or streams to make changes, capture state information, and take corrective action. For more information, see What is Amazon CloudWatch Events in the *Amazon CloudWatch User Guide*.
- **AWS CloudTrail Log Monitoring** – Share log files between accounts, monitor CloudTrail log files in real time by sending them to CloudWatch Logs, write log processing applications in Java, and validate that your log files have not changed after delivery by CloudTrail. For more information, see Working with CloudTrail log files in the *AWS CloudTrail User Guide*.

Manual monitoring tools

Another important part of monitoring QLDB involves manually monitoring those items that the CloudWatch alarms don't cover. The QLDB, CloudWatch, Trusted Advisor, and other AWS console dashboards provide an at-a-glance view of the state of your AWS environment. We recommend that you also check the log files on Amazon QLDB.

- The QLDB dashboard shows the following:
  - Read and write I/Os
  - Journal and indexed storage
  - Command latency
  - Exceptions
- The CloudWatch home page shows the following:
  - Current alarms and status

- Graphs of alarms and resources
- Service health status

In addition, you can use CloudWatch to do the following:

- Create customized dashboards to monitor the services you care about
- Graph metric data to troubleshoot issues and discover trends
- Search and browse all your AWS resource metrics
- Create and edit alarms to be notified of problems

Monitoring with Amazon CloudWatch

You can monitor Amazon QLDB using CloudWatch, which collects and processes raw data from Amazon QLDB into readable, near-real-time metrics. It records these statistics for two weeks so that you can access historical information and gain a better perspective on how your web application or service is performing. By default, QLDB metric data is automatically sent to CloudWatch in 1 or 15-minute periods. For more information, see What are Amazon CloudWatch, Amazon CloudWatch Events, and Amazon CloudWatch Logs? in the *Amazon CloudWatch User Guide*.

Topics

How do I use QLDB metrics?

The metrics reported by QLDB provide information that you can analyze in different ways. The following list shows some common uses for the metrics. These are suggestions to get you started, not a comprehensive list.

- You can monitor `JournalStorage` and `IndexedStorage` over a specified time period, to track how much disk space your ledger is consuming.
- You can monitor `ReadIOs` and `WriteIOs` over a specified time period, to track how many requests your ledger is processing.
- You can monitor `CommandLatency` to track your ledger's performance for data operations and analyze the types of commands that result in the most latency.

Amazon QLDB metrics and dimensions

When you interact with Amazon QLDB, it sends the following metrics and dimensions to CloudWatch. Storage metrics are reported every 15 minutes, and all other metrics are aggregated and reported every minute. You can use the following procedures to view the metrics for QLDB.

To view metrics using the CloudWatch console

Metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace.

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. If necessary, change the Region. On the navigation bar, choose the Region where your AWS resources reside. For more information, see Regions and endpoints.

3. In the navigation pane, choose **Metrics**.
4. Under the **All metrics** tab, choose **QLDB**.

To view metrics using the AWS CLI

- At a command prompt, use the following command.

```
aws cloudwatch list-metrics --namespace "AWS/QLDB"
```

CloudWatch displays the following metrics for QLDB.

Amazon QLDB dimensions and metrics

The metrics and dimensions that Amazon QLDB sends to Amazon CloudWatch are listed here.

QLDB metrics

Metric	Description
`JournalStorage`	The total amount of disk space used by the ledger's journal, reported in 15-minute intervals. The journal contains the complete, immutable, and verifiable history of all the changes to your data. Units: `Bytes` Dimensions: `LedgerName`
`IndexedStorage`	The total amount of disk space used by the ledger's tables, indexes, and indexed history, reported in 15-minute intervals. Indexed storage consists of ledger data that is optimized for high-performance queries. Units: `Bytes` Dimensions: `LedgerName`
`ReadIOs`	The number of disk read I/O operations, reported in one-minute intervals. This captures all types of read operations, including data transactions, verification API requests, journal exports, and journal streams. Units: `Count` Dimensions: `LedgerName`
`WriteIOs`	The number of disk write I/O operations, reported in one-minute intervals. Units: `Count` Dimensions: `LedgerName`
`CommandLatency`	The amount of time taken for data operations, reported in one-minute intervals. Units: `Milliseconds`

Metric	Description
	Dimensions: `LedgerName, CommandType`
`IsImpaired`	The flag that indicates if a journal stream to Kinesis Data Streams is impaired, reported in one-minute intervals. A value of `1` indicates that the stream is in impaired state, and `0` indicates otherwise. Units: `Boolean` (0 or 1) Dimensions: `StreamId`
`OccConflictExceptions`	The number of requests to QLDB that generate an `OccConflictException`. For information about optimistic concurrency control (OCC), see Amazon QLDB concurrency model (p. 332). Units: `Count`
`Session4xxExceptions`	The number of requests to QLDB that generate an HTTP 4xx error. Units: `Count`
`Session5xxExceptions`	The number of requests to QLDB that generate an HTTP 5xx error. Units: `Count`
`SessionRateExceededExceptions`	The number of requests to QLDB that generate a `SessionRateExceededException`. Units: `Count`

Dimensions for QLDB metrics

The metrics for QLDB are qualified by the values for the account, ledger name, stream ID, or command type. You can use the CloudWatch console to retrieve QLDB data along any of the dimensions in the following table.

Dimension	Description
`LedgerName`	This dimension limits the data to a specific ledger. This value can be any ledger name in the current AWS Region and the current AWS account.
`StreamId`	This dimension limits the data to a specific journal stream. This value can be any stream ID for a ledger in the current AWS Region and the current AWS account.
`CommandType`	This dimension limits the data to one of the following QLDB data API commands: • `AbortTransaction` • `CommitTransaction` • `EndSession` • `ExecuteStatement`

Dimension	Description
	• `FetchPage` • `StartSession` • `StartTransaction`

Creating CloudWatch alarms to monitor Amazon QLDB

You can create an Amazon CloudWatch alarm that sends an Amazon Simple Notification Service (Amazon SNS) message when the alarm changes state. An alarm watches a single metric over a time period that you specify. It performs one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon SNS topic or Auto Scaling policy.

Alarms invoke actions for sustained state changes only. CloudWatch alarms do not invoke actions simply because they are in a particular state. The state must have changed and been maintained for a specified number of periods.

For more information about creating CloudWatch alarms, see Using Amazon CloudWatch alarms in the *Amazon CloudWatch User Guide*.

Automating Amazon QLDB with CloudWatch Events

Amazon CloudWatch Events enables you to automate your AWS services and respond automatically to system events such as application availability issues or resource changes. Events from AWS services are delivered to CloudWatch Events in near-real time. You can write simple rules to indicate which events are of interest to you, and what automated actions to take when an event matches a rule. The actions that can be automatically triggered include the following:

- Invoking an AWS Lambda function
- Invoking Amazon EC2 Run Command
- Relaying the event to Amazon Kinesis Data Streams
- Activating an AWS Step Functions state machine
- Notifying an Amazon SNS topic or an Amazon SQS queue

Amazon QLDB reports an event to CloudWatch Events whenever the state of a resource in your AWS account changes. QLDB resources include the following:

- ledger

The following is an example of an event that QLDB reported, in which a ledger's state changed to `DELETING`.

```
{
    "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-2",
    "resources" : ["arn:aws:qldb:us-east-2:123456789012:ledger/exampleLedger"],
    "detail" : {
        "ledgerName" : "exampleLedger",
        "state" : "DELETING"
    }
}
```

Some examples of using CloudWatch Events with QLDB include the following:

- Activating a Lambda function whenever a new ledger is initially created in `CREATING` state and eventually becomes `ACTIVE`.
- Notifying an Amazon SNS topic when the state of your ledger changes to `DELETING`.

For more information, see the Amazon CloudWatch Events User Guide.

Logging Amazon QLDB API calls with AWS CloudTrail

Amazon QLDB is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in QLDB. CloudTrail captures all control plane API calls for QLDB as events. The calls that are captured include calls from the QLDB console and code calls to the QLDB API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon Simple Storage Service (Amazon S3) bucket, including events for QLDB. If you don't configure a trail, you can still view the most recent events on the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to QLDB, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, including how to configure and enable it, see the AWS CloudTrail User Guide.

QLDB information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported activity occurs in QLDB, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail event history.

For an ongoing record of events in your AWS account, including events for QLDB, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs.

For more information, see the following topics in the *AWS CloudTrail User Guide*:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple Regions
- Receiving CloudTrail log files from multiple accounts

All QLDB control plane and non-transactional data plane actions are logged by CloudTrail and are documented in the Amazon QLDB API reference. For example, calls to the `CreateLedger`, `DescribeLedger`, and `DeleteLedger` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials

- Whether the request was made with temporary security credentials for a role or federated user
- Whether the request was made by another AWS service

For more information, see the CloudTrail userIdentity element.

Understanding QLDB log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates these actions:

- `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"
    }
  ]
}
```

Compliance validation for Amazon QLDB

Third-party auditors assess the security and compliance of Amazon QLDB as part of multiple AWS compliance programs. These currently include HIPAA, ISO, and PCI.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using QLDB is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- Security and Compliance Quick Start Guides – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Config – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in Amazon QLDB

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Storage durability

QLDB *journal storage* features synchronous replication to multiple Availability Zones on transaction commits. This ensures that even a full Availability Zone failure of journal storage would not compromise data integrity or the ability to maintain an active service. Additionally, the QLDB journal features asynchronous archives to fault-tolerant storage. This feature supports disaster recovery in the highly unlikely event of simultaneous storage failure for multiple Availability Zones.

QLDB *indexed storage* is backed by replication to multiple Availability Zones. This ensures that even a full Availability Zone failure of indexed storage would not compromise data integrity or the ability to maintain an active service.

Data durability features

In addition to the AWS global infrastructure, QLDB offers the following features to help support your data resiliency and backup needs.

QLDB service features

On-demand journal export

QLDB provides an on-demand journal export feature. Access the contents of your journal by exporting journal blocks from your ledger into an Amazon Simple Storage Service (Amazon S3) bucket. You can use this data for various purposes such as data retention, analytics, and auditing. For more information, see Exporting journal data from Amazon QLDB (p. 354).

Backup and restore

Automated restore for exports is not supported at this time. Export provides a basic ability to create additional data redundancy at your defined frequency. However, a restore scenario is application dependent, where the exported records are presumably written back to a new ledger by leveraging the same or similar ingestion method.

QLDB does not provide a dedicated backup and related restore feature at this time.

Journal streams

QLDB also provides a continuous journal stream capability. You can integrate QLDB journal streams with the Amazon Kinesis streaming platform to process real-time journal data. For more information, see Streaming journal data from Amazon QLDB (p. 366).

QLDB design feature

QLDB is designed to be resilient against logical corruption. The QLDB journal is immutable, ensuring that all committed transactions are persisted to the journal. In addition, every committed document change is recorded, as this enables *point-in-time* visibility for any unintended changes to ledger data.

QLDB does not provide an automated recovery feature for logical corruption scenarios at this time.

Infrastructure security in Amazon QLDB

As a managed service, Amazon QLDB is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access QLDB through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

You can also use a *virtual private cloud* (VPC) endpoint for QLDB. Interface VPC endpoints enable your Amazon VPC resources to use their private IP addresses to access QLDB with no exposure to the public internet. For more information, see Using Amazon QLDB with interface VPC endpoints (p. 423).

Using Amazon QLDB with interface VPC endpoints

You can use an interface VPC endpoint to keep traffic between your Amazon Virtual Private Cloud (Amazon VPC) and Amazon QLDB from leaving the AWS network. Interface VPC endpoints don't require an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Interface VPC endpoints are powered by AWS PrivateLink. AWS PrivateLink enables private communication between AWS services using an elastic network interface with private IPs in your Amazon VPC. For more information, see What is Amazon VPC? and Interface VPC endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

Topics

Using interface VPC endpoints for QLDB

To get started, you don't need to change the settings for your QLDB ledgers. Simply create an interface VPC endpoint to enable your QLDB traffic from and to your Amazon VPC resources to start flowing through the interface VPC endpoint. For more information, see Creating an interface endpoint.

For example, consider a QLDB ledger application that runs on an Amazon Elastic Compute Cloud (Amazon EC2) instance in a VPC. If you enable a QLDB interface VPC endpoint, calls between the ledger application and QLDB flow through the private interface VPC endpoint instead of public endpoints.

> **Note**
> QLDB currently supports interface VPC endpoints for the *QLDB Session* transactional data API only. This API includes only the SendCommand action.

Controlling access to VPC endpoints for QLDB

You can use VPC endpoint policies to control access by attaching a policy to a VPC endpoint. Or, you can use additional fields in a policy that is attached to an IAM user, group, or role to allow access only from a specified VPC endpoint. When used together, VPC endpoint policies and IAM policies can restrict access to specific QLDB actions on specified ledgers to a specified VPC endpoint.

The following are example endpoint policies for accessing QLDB ledgers.

- **VPC policy example: Read-only access** — You can attach the following sample policy to a VPC endpoint. (For more information, see Controlling access to Amazon VPC resources in the *Amazon VPC User Guide*). This policy restricts actions to only listing and describing QLDB ledgers through the VPC endpoint that it's attached to.

```
{
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "qldb:List*",
        "qldb:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- **VPC policy example: Restrict access to a specific QLDB ledger** — You can attach the following sample policy to a VPC endpoint. This policy restricts access to a specific ledger resource through the VPC endpoint that it's attached to.

```
{
  "Statement": [
    {
      "Sid": "AccessToSpecificQLDBLedger",
      "Principal": "*",
      "Action": "qldb:*",
      "Effect": "Allow",
      "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger"
    }
  ]
}
```

- **IAM policy example: Restrict access to a specific QLDB ledger from a specific VPC endpoint only** — You can attach the following sample policy to an IAM user, role, or group. The policy allows access to a specified QLDB ledger only from a specified VPC endpoint.

  Important
  This policy example specifies only the `SendCommand` action because it's the only QLDB action that currently supports interface VPC endpoints.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AccessFromSpecificEndpoint",
         "Action": "qldb:SendCommand",
         "Effect": "Deny",
         "Resource": "arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger",
         "Condition": { "StringNotEquals" : { "aws:sourceVpce": "vpce-1a2b3c4d" } }
      }
   ]
}
```

Amazon QLDB PartiQL reference

Amazon QLDB supports a *subset* of the PartiQL query language. The following sections describe the QLDB implementation of PartiQL.

> **Note**
> QLDB does not support all PartiQL operations.
> This reference provides basic syntax and usage examples of PartiQL statements that you manually run on the QLDB console or the QLDB shell. For code examples that show how to programmatically run similar statements using the QLDB driver, see the tutorials in Getting started with the driver (p. 40).

Topics

What is PartiQL?

PartiQL provides SQL-compatible query access across multiple data stores containing structured data, semistructured data, and nested data. It is widely used within Amazon and is now available as part of many AWS services, including QLDB.

For the PartiQL specification and a tutorial on the core query language, see the PartiQL documentation.

PartiQL extends SQL-92 to support documents in the Amazon Ion data format. For information about Amazon Ion, see the Amazon Ion data format reference (p. 487).

PartiQL in Amazon QLDB

To run PartiQL queries in QLDB, you can do one of the following:

- Use the *Query editor* on the AWS Management Console for QLDB.
- Use the command line QLDB shell.
- Run them programmatically using a provided QLDB driver.

For information about using these methods to access QLDB, see Accessing Amazon QLDB (p. 15).

Data types

Amazon QLDB stores documents in Amazon Ion (p. 487) format. Amazon Ion is a data serialization format (both in text form and in binary-encoded form) that is a superset of JSON. The following table lists the Ion data types that you can use in QLDB documents.

Data type	Description
`null`	A generic null value
`bool`	Boolean values
`int`	Signed integers of arbitrary size
`decimal`	Decimal-encoded real numbers of arbitrary precision
`float`	Binary-encoded floating point numbers (64-bit IEEE)
`timestamp`	Date/time/timezone moments of arbitrary precision
`string`	Unicode text literals
`symbol`	Unicode symbolic atoms (identifiers)
`blob`	Binary data of user-defined encoding
`clob`	Text data of user-defined encoding
`struct`	Unordered collections of name-value pairs
`list`	Ordered heterogeneous collections of values

See the Ion specification document on the Amazon GitHub site for a full list of Ion core data types with complete descriptions and value formatting details.

QLDB documents

Amazon QLDB stores data records as documents, which are just Amazon Ion (p. 487) `struct` objects that are inserted into a table. For the Ion specification, see the Amazon Ion GitHub site.

Topics

Ion document structure

Like JSON, QLDB documents are composed of name-value pairs in the following structure.

```
{
  name1: value1,
  name2: value2,
  name3: value3,
  ...
  nameN: valueN
}
```

The names are symbol tokens, and the values are unrestricted. Each name-value pair is called a *field*. The value of a field can be any of the Ion Data types (p. 427), including container types: nested structures, lists, and lists of structures.

Also like JSON, a `struct` is denoted with curly braces (`{...}`), and a `list` is denoted with square brackets (`[...]`). The following example is a document from the sample data in

This tutorial guides you through steps to create your first Amazon QLDB ledger and populate it with tables and sample data. The sample ledger you create in this scenario is a database for a department of motor vehicles (DMV) application that tracks the complete historical information about vehicle registrations.

The history of an asset is a common use case for QLDB because it involves a variety of scenarios and operations that highlight the usefulness of a ledger database. QLDB enables you to track and verify the full history of changes in an easily accessible and flexible database.

As you work through this tutorial, the following sections explain how to add vehicle registrations, modify them, and view the history of changes to those registrations. This guide also shows you how to cryptographically verify a registration document, and concludes by cleaning up resources and deleting the sample ledger.

Each step in the tutorial has instructions for using the AWS Management Console.

Topics

Tutorial prerequisites

Before you start this tutorial, you must follow the AWS setup instructions in Accessing Amazon QLDB (p. 15). These steps include signing up for AWS and creating an AWS Identity and Access Management (IAM) user with QLDB access.

Step 1: Create a new ledger

In this step, you create a new Amazon QLDB ledger named `vehicle-registration`. Then, you confirm that the status of the ledger is **Active**. You can also verify any tags that you added to the ledger.

When you create a ledger, *deletion protection* is enabled by default. Deletion protection is a feature in QLDB that prevents ledgers from being deleted by any user. You can disable deletion protection when you create a ledger using the QLDB API or the AWS Command Line Interface (AWS CLI).

To create a new ledger

1. Sign in to the AWS Management Console, and open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Getting started**.
3. On the **Create your first ledger** card, choose **Create Ledger**.
4. On the **Create Ledger** page, do the following:
   - **Ledger information**—The **Ledger name** should be pre-populated with **`vehicle-registration`**.
   - **Tags**—(Optional) Add metadata to the ledger by attaching tags as key-value pairs. You can add tags to your ledger to help organize and identify them. For more information, see Tagging Amazon QLDB resources (p. 387).
     Choose **Add tag**, and then enter any key-value pairs as appropriate.
5. When the settings are as you want them, choose **Create ledger**.

   Note
   You can access your QLDB ledger when its status becomes **Active**. This can take several minutes.
6. In the list of **Ledgers**, locate `vehicle-registration` and confirm that the ledger's status is **Active**.
7. (Optional) Choose the `vehicle-registration` ledger name. On the **vehicle-registration** ledger details page, confirm that any tags that you added to the ledger appear on the **Tags** card. You can also edit your ledger tags using this console page.

To create tables in the `vehicle-registration` ledger, proceed to Step 2: Create tables, indexes, and sample data (p. 26).

Step 2: Create tables, indexes, and sample data

When your Amazon QLDB ledger is active, you can start creating tables for data about vehicles, their owners, and their registration information. After creating the tables and indexes, you can load them with data.

In this step, you create four tables in the `vehicle-registration` ledger:

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

You also create the following indexes.

`VehicleRegistration`	`VIN`

`VehicleRegistration`	`LicensePlateNumber`
`Vehicle`	`VIN`
`Person`	`GovId`
`DriversLicense`	`LicenseNumber`
`DriversLicense`	`PersonId`
Table name	**Field**

You can use the QLDB console to automatically create these tables with indexes and load them with sample data. Or, you can use the **Query editor** on the console to manually run each PartiQL (p. 426) statement step-by-step.

Automatic option

To create tables, indexes, and sample data

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Getting started**.
3. Under **Automatic option** on the **Sample application data** card, choose `vehicle-registration` in the list of ledgers.
4. Choose **Load sample data**.
   If the operation finishes successfully, the console displays the message **Sample data loaded**.
   This script runs all statements in a single transaction. If any part of the transaction fails, every statement is rolled back, and an appropriate error message is displayed. You can retry the operation after addressing any issues.

   Note

   - One possible cause for a transaction failure is attempting to create duplicate tables. Your request to load sample data will fail if any of the following table names already exist in your ledger: `VehicleRegistration`, `Vehicle`, `Person`, and `DriversLicense`.
     Instead, try loading this sample data in an empty ledger.
   - This script runs parameterized `INSERT` statements. So, these PartiQL statements are recorded in your journal blocks with bind parameters instead of the literal data. For example, you might see the following statement in a journal block, where the question mark (`?`) is a variable placeholder for the document contents.

     ```
     INSERT INTO Vehicle ?
     ```

Manual option

You insert documents into `VehicleRegistration` with an empty `PrimaryOwner` field, and into `DriversLicense` with an empty `PersonId` field. Later, you populate these fields with the system-assigned document `id` from the `Person` table.

> **Tip**
> As a best practice, use this document `id` metadata field as a foreign key. For more information, see Querying document metadata (p. 323).

To create tables, indexes, and sample data

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Query editor**.
3. Choose the `vehicle-registration` ledger.
4. Start by creating four tables. QLDB supports open content and does not enforce schema, so you don't specify attributes or data types.
   In the query editor window, enter the following statement, and then choose **Run**. To run the statement, you can also use the keyboard shortcut **Ctrl**+**Enter** for Windows, or **Cmd**+**Return** for macOS. For more keyboard shortcuts, see Accessing Amazon QLDB using the console (p. 18).

   ```
   CREATE TABLE VehicleRegistration
   ```

   Repeat this step for each of the following.

   ```
   CREATE TABLE Vehicle
   ```

   ```
   CREATE TABLE Person
   ```

   ```
   CREATE TABLE DriversLicense
   ```
5. Next, create indexes that help speed up queries against each table.
   In the query editor window, enter the following statement, and then choose **Run**.

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

   Repeat this step for the following.

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

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

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

   ```
   CREATE INDEX ON DriversLicense (LicenseNumber)
   ```

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

6. After creating your indexes, you can start loading data into your tables. In this step, insert documents into the `Person` table with personal information about owners of the vehicles that the ledger is tracking.
   In the query editor window, enter the following statement, and then choose **Run**.

```
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. Then, populate the `DriversLicense` table with documents that include driver's license information for each vehicle owner.
   In the query editor window, enter the following statement, and then choose **Run**.

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

8. Now, populate the `VehicleRegistration` table with vehicle registration documents. These documents include a nested `Owners` structure that stores the primary and secondary owners.
   In the query editor window, enter the following statement, and then choose **Run**.

```
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. Lastly, populate the `Vehicle` table with documents describing the vehicles that are registered in your ledger.
   In the query editor window, enter the following statement, and then choose **Run**.

```
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'
} >>
```

Next, you can use `SELECT` statements to read data from the tables in the `vehicle-registration` ledger. Proceed to Step 3: Query the tables in a ledger (p. 32).

Step 3: Query the tables in a ledger

After creating tables in an Amazon QLDB ledger and loading them with data, you can run queries to review the vehicle registration data that you just inserted. QLDB uses PartiQL as its query language and Amazon Ion as its document-oriented data model.

PartiQL is an open source, SQL-compatible query language that has been extended to work with Ion. With PartiQL, you can insert, query, and manage your data with familiar SQL operators. Amazon Ion is a superset of JSON. Ion is an open source, document-based data format that gives you the flexibility of storing and processing structured, semistructured, and nested data.

In this step, you use `SELECT` statements to read data from the tables in the `vehicle-registration` ledger.

To query the tables

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Query editor**.
3. Choose the `vehicle-registration` ledger.
4. In the query editor window, enter the following statement to query the `Vehicle` table for a particular vehicle identification number (VIN) that you added to the ledger, and then choose **Run**.
   To run the statement, you can also use the keyboard shortcut **Ctrl**+**Enter** for Windows, or **Cmd**+**Return** for macOS. For more keyboard shortcuts, see Accessing Amazon QLDB using the console (p. 18).

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

5. You can write inner join queries. This query example joins `Vehicle` with `VehicleRegistration` and returns registration information along with attributes of the registered vehicle for a specified `VIN`.
   Enter the following statement, and then choose **Run**.

```
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
```

You can also join the `Person` and `DriversLicense` tables to see attributes related to the drivers who were added to the ledger.
Repeat this step for the following.

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

To learn about modifying documents in the tables in the `vehicle-registration` ledger, see Step 4: Modify documents in a ledger (p. 33).

Step 4: Modify documents in a ledger

Now that you have data to work with, you can start making changes to documents in the `vehicle-registration` ledger in Amazon QLDB. For example, consider the Audi A5 with VIN `1N4AL11D75C109151`. This car is initially owned by a driver named Raul Lewis in Seattle, WA.

Suppose that Raul sells the car to a resident in Everett, WA named Brent Logan. Then, Brent and Alexis Pena decide to get married. Brent wants to add Alexis as a secondary owner on the registration. In this step, the following data manipulation language (DML) statements demonstrate how to make the appropriate changes in your ledger to reflect these events.

Tip
As a best practice, use a document's system-assigned `id` as a foreign key. While you can define fields that are intended to be unique identifiers (for example, a vehicle's VIN), the true unique identifier of a document is its `id`. This field is included in the document's metadata, which you can query in the *committed view* (the system-defined view of a table).
For more information about views in QLDB, see Core concepts (p. 7). To learn more about metadata, see Querying document metadata (p. 323).

To modify documents

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Query editor**.
3. Choose the `vehicle-registration` ledger.

   Note
   If you set up your ledger using the console's automatic **Load sample data** feature, skip ahead to *Step 6*.
4. If you manually ran `INSERT` statements to load the sample data, continue with these steps.
   To initially register Raul as this vehicle's owner, start by finding his system-assigned document `id` in the `Person` table. This field is included in the document's metadata, which you can query in the system-defined view of the table, called the *committed view*.
   In the query editor window, enter the following statement, and then choose **Run**.

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

The prefix `_ql_committed_` is a reserved prefix signifying that you want to query the committed view of the `Person` table. In this view, your data is nested in the `data` field, and metadata is nested in the `metadata` field.

5. Now, use this `id` in an `UPDATE` statement to modify the appropriate document in the `VehicleRegistration` table. Enter the following statement, and then choose **Run**.

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

Confirm that you modified the `Owners` field by issuing this statement.

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

6. To transfer the vehicle's ownership to Brent in the city of Everett, first find his `id` from the `Person` table with the following statement.

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

Next, use this `id` to update the `PrimaryOwner` and the `City` in the `VehicleRegistration` table.

```
UPDATE VehicleRegistration AS r
SET r.Owners.PrimaryOwner.PersonId = 'IN7MvYtUjkp1GMZu0F6CG9', --replace with
 your id
r.City = 'Everett'
WHERE r.VIN = '1N4AL11D75C109151'
```

Confirm that you modified the `PrimaryOwner` and `City` fields by issuing this statement.

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

7. To add Alexis as a secondary owner of the car, find her `Person id`.

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

Then, insert this `id` into the `SecondaryOwners` list with the following FROM-INSERT (p. 452) DML statement.

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

Confirm that you modified `SecondaryOwners` by issuing this statement.

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

To review these changes in the `vehicle-registration` ledger, see Step 5: View the revision history for a document (p. 34).

Step 5: View the revision history for a document

After modifying registration data for the car with VIN `1N4AL11D75C109151`, you can query the history of all its registered owners and any other updated fields. You can see all revisions of a document that you inserted, updated, and deleted by querying the built-in History function (p. 326).

The history function returns revisions from the *committed view* of your table, which includes both your application data and the associated metadata. The metadata shows exactly when each revision was made, in what order, and which transaction committed them.

In this step, you query the revision history of a document in the `VehicleRegistration` table in the `vehicle-registration` ledger.

To view the revision history

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Query editor**.
3. Choose the `vehicle-registration` ledger.
4. To query the history of a document, start by finding its unique `id`. In addition to querying the committed view, another way of getting a document `id` is to use the `BY` keyword in the table's default user view. To learn more, see Using the BY clause to query document ID (p. 325).
   In the query editor window, enter the following statement, and then choose **Run**.

   ```
   SELECT r_id FROM VehicleRegistration AS r BY r_id
   WHERE r.VIN = '1N4AL11D75C109151'
   ```
5. Next, you can use this `id` value to query the history function in the following syntax.

   ```
   SELECT * FROM history( table [, `start-time`, `end-time`] ) AS h
   WHERE h.metadata.id = 'id'
   ```
   - Best practice is to qualify a history query with a document `id`. This helps to avoid inefficient queries.
   - The `start-time` and `end-time` are optional parameters that are both inclusive. They must be in ISO 8601 date and time format and in Universal Coordinated Time (UTC).
   - The `start-time` and `end-time` are Amazon Ion literal values that can be denoted with backticks (`` `...` ``). To learn more, see Querying Ion with PartiQL (p. 444).

The history function returns documents with the same schema as the committed view. For example, enter the following statement, and then choose **Run**. Be sure to replace the `id` value with your own document ID as appropriate.

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

This first history query projects your modified vehicle registration data. The output includes all revisions of the document and should look similar to the following.

VIN	City	Owners
"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
The history query might not always return document revisions in sequential order.

Review the output and confirm that the changes reflect what you did in Step 4: Modify documents in a ledger (p. 33).

6. Then, you can inspect the document metadata of each revision. Enter the following statement, and then choose **Run**. Again, be sure to replace the `id` value with your own document ID as appropriate.

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

The output should look similar to the following.

versio	id	txTime	txId
0	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T19:20:360d-3Z	"BMoVdWuPxJg3k466Iz4i75"
1	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T21:40:199d-3Z	"KWByxe842Xw8DNHcvARPOt"
2	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T21:44:432d-3Z	"BKwDOJRwbHpFvmAyJ2Kdh9"

versio	id	txTime	txId
3	"ADR2LQq48kB9neZDupQrMm"	2019-05-23T21:49:254d-3Z	"9ZEiZd7vCmJ6RAvOvTZ4YA"

These metadata fields provide details on when each item was modified, and by which transaction. From this data, you can infer the following:

- The document is uniquely identified by its system-assigned `id`: `ADR2LQq48kB9neZDupQrMm`. This is a universally unique identifier (UUID) that is represented in a Base62-encoded string.
- The `txTime` shows that the initial revision of the document (version `0`) was created at `2019-05-23T19:20:360d-3Z`.
- Each subsequent transaction creates a new revision with the same document `id`, an incremented version number, and an updated `txId` and `txTime`.

To cryptographically verify a document revision in the `vehicle-registration` ledger, proceed to Step 6: Verify a document in a ledger (p. 36).

Step 6: Verify a document in a ledger

With Amazon QLDB, you can efficiently verify the integrity of a document in your ledger's journal by using cryptographic hashing with SHA-256. In this example, Alexis and Brent decide to upgrade to a new model by trading in the vehicle with VIN `1N4AL11D75C109151` at a car dealership. The dealership starts the process by verifying the vehicle's ownership with the registration office.

To learn more about how verification and cryptographic hashing work in QLDB, see Data verification in Amazon QLDB (p. 338).

In this step, you verify a document revision in the `vehicle-registration` ledger. First, you request a digest, which is returned as an output file and acts as a signature of your ledger's entire change history. Then, you request a proof for the revision relative to that digest. Using this proof, the integrity of your revision is verified if all validation checks pass.

To request a digest

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Ledgers**.
3. In the list of ledgers, select `vehicle-registration`.
4. Choose **Get digest**. The **Get digest** dialog box displays the following digest details:
   - **Digest**—The SHA-256 hash value of the digest that you requested.
   - **Digest tip address**—The latest block (p. 338) location in the journal covered by the digest that you requested. An address has the following two fields:
     - `strandId`—The unique ID of the journal strand that contains the block.
     - `sequenceNo`—The index number that specifies the location of the block within the strand.
   - **Ledger**—The ledger name for which you requested a digest.
   - **Date**—The timestamp when you requested the digest.
5. Review the digest information. Then choose **Save**. You can keep the default file name, or enter a new name.

This step saves a plaintext file with contents in Amazon Ion (p. 487) format. The file has a file name extension of `.ion.txt` and contains all the digest information that was

listed on the preceding dialog box. The following is an example of a digest file's contents. The order of the fields can vary depending on your browser.

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

6. Save this file where you can access it later. In the following steps, you use this file as a fingerprint to verify a document revision against.

After you have a ledger digest saved, you can start the process of verifying a document revision against that digest.

Note
In a real scenario for verification, you use a digest that was previously saved rather than doing the two tasks consecutively. As a best practice, the digest is saved as soon as the document revision that you want to verify is written to the journal.

To verify a document revision

1. First, query your ledger for the `id` and `blockAddress` of the document revision that you want to verify. These fields are included in the document's metadata, which you can query in the committed view.
   The document `id` is a system-assigned unique ID string. The `blockAddress` is an Ion structure that specifies the block location where the revision was committed.
   In the navigation pane of the QLDB console, choose **Query editor**.
2. Choose the `vehicle-registration` ledger.
3. In the query editor window, enter the following statement, and then choose **Run**.

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

4. Copy and save the `id` and `blockAddress` values that your query returns. Be sure to omit the double quotes for the `id` field. In Amazon Ion, string data types are delimited with double quotes.
5. Now that you have a document revision selected, you can start the process of verifying it. In the navigation pane, choose **Verification**.
6. On the **Verify document** form, under **Specify the document that you want to verify**, enter the following input parameters:
   - **Ledger**—Choose `vehicle-registration`.
   - **Block address**—The `blockAddress` value returned by your query in *step 3*.
   - **Document ID**—The `id` value returned by your query in *step 3*.
7. Under **Specify the digest to use for verification**, select the digest that you previously saved by choosing **Choose digest**. If the file is valid, this auto-populates all the digest

fields on your console. Or, you can manually copy and paste the following values directly from your digest file:

- **Digest**—The `digest` value from your digest file.
- **Digest tip address**—The `digestTipAddress` value from your digest file.

8. Review your document and digest input parameters, and then choose **Verify**.

   The console automates two steps for you:

   a. Request a proof from QLDB for your specified document.
   b. Use the proof returned by QLDB to call a client-side API, which verifies your document revision against the provided digest.

   The console displays the results of your request in the **Verification results** card. For more information, see Verification results (p. 346).
9. To test the verification logic, repeat steps 6–8 under **To verify a document revision**, but change a single character in the **Digest** input string. This should cause your **Verify** request to fail with an appropriate error message.

If you no longer need to use the `vehicle-registration` ledger, proceed to Step 7 (optional): Clean up resources (p. 38).

Step 7 (optional): Clean up resources

You can continue using the `vehicle-registration` ledger. However, if you no longer need it, you should delete it.

If deletion protection is enabled for your ledger, you must first disable it before you can delete the ledger using the QLDB API or the AWS Command Line Interface (AWS CLI). The QLDB console disables deletion protection for you when you use it to delete a ledger.

To delete the ledger

1. Open the Amazon QLDB console at https://console.aws.amazon.com/qldb.
2. In the navigation pane, choose **Ledgers**.
3. In the list of ledgers, select `vehicle-registration`.
4. Choose **Delete ledger**. Confirm this by entering **`vehicle-registration`** in the field provided.
5. If deletion protection is enabled for this ledger, you must also choose the option to **Override deletion protection**.

To learn more about working with ledgers in QLDB, see Getting started with Amazon QLDB: Next steps (p. 39).

Getting started with Amazon QLDB: Next steps

For more information about using Amazon QLDB, see the following topics:

- Working with data and history in Amazon QLDB (p. 317)

- Getting started with the Amazon QLDB driver (p. 40)
- Amazon QLDB concurrency model (p. 332)
- Data verification in Amazon QLDB (p. 338)
- Exporting journal data from Amazon QLDB (p. 354)
- Amazon QLDB PartiQL reference (p. 426)

(p.) that contains values of various types.

```
{
    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
In Ion, double quotation marks denote string values, and bare symbols represent field names. But in PartiQL, single quotation marks denote both strings and field names.
This difference in syntax allows the PartiQL query language to maintain SQL compatibility, and the Amazon Ion data format to maintain JSON compatibility. For details on PartiQL's syntax and semantics when using the QLDB console to query Ion data, see Querying Ion with PartiQL (p. 444).

PartiQL-Ion type mapping

In QLDB, PartiQL extends SQL's type system to cover the Ion data model. This mapping is described as follows:

- SQL scalar types are covered by their Ion counterparts. For example:
  - `CHAR` and `VARCHAR` are Unicode sequences that map to the Ion `string` type.
  - `NUMBER` maps to the Ion `decimal` type.
- Ion's `struct` type is equivalent to an SQL tuple, which traditionally represents a table *row*.
  - However, with open content and without schema, queries that rely on the ordered nature of an SQL tuple are not supported (such as the output order of `SELECT *`).
- In addition to `NULL`, PartiQL has a `MISSING` type. This is a specialization of `NULL` and indicates the lack of a field. This type is necessary because Ion `struct` fields might be sparse.

Document ID

QLDB assigns a *document ID* to each document that you insert into a table. All QLDB-assigned IDs are universally unique identifiers (UUID) that are each represented in a Base62-encoded string (for example, `3Qv67yjXEwB9SjmvkuG6Cp`). For more information, see Unique IDs in Amazon QLDB (p. 330).

Each document *revision* is uniquely identified by a combination of the document ID and a zero-based version number.

The document ID and version fields are included in the document's metadata, which you can query in the *committed view* (the system-defined view of a table). For more information about views in QLDB, see Core concepts (p. 7). To learn more about metadata, see Querying document metadata (p. 323).

Querying Ion with PartiQL

When you use the AWS Management Console to query data in Amazon QLDB, you write statements in PartiQL, but results are shown in Amazon Ion. PartiQL is intended to be SQL-compatible, whereas Ion is an extension of JSON. This leads to syntactic differences with how you notate data in your queries, compared to how the QLDB console presents your query results.

This section describes PartiQL semantics for running manual statements using the *Query editor* on the console or the QLDB shell command line interface.

Tip
When you run PartiQL queries programmatically, the best practice is to use parameterized statements. You can use a question mark (`?`) as a bind variable placeholder in your statements to avoid these syntax rules. This is also more secure and performant. To learn more, see Step 3: Create tables, indexes, and sample data (p. 63) in the *Getting started with the driver* Java tutorial.

Topics

Syntax and semantics

When using the QLDB console or the QLDB shell to query Ion data, the following are PartiQL's fundamental query syntax and semantics:

Case sensitivity

All QLDB system object names—including field names, table names, and ledger names—are case sensitive.

String values

In Ion, double quotation marks (`"..."`) denote a string.

In PartiQL, single quotation marks (`'...'`) denote a string.

Symbols and identifiers

In Ion, single quotation marks (`'...'`) denote a symbol. A subset of symbols in Ion called *identifiers* are represented in bare text without any quotes.

In PartiQL, double quotation marks (`"..."`) denote a quoted PartiQL identifier. Unquoted text represents a regular PartiQL identifier, such as a table name.

Ion literals

Any Ion literals can be denoted with backticks (`` `...` ``) in a PartiQL statement.

Field names

Ion field names are case-sensitive symbols. PartiQL enables you to denote field names with single quotation marks in a DML statement. This is a shorthand alternative to using PartiQL's `cast` function to define a symbol. It is also more intuitive than using backticks to denote a literal Ion symbol.

Literals

Literals of the PartiQL query language correspond to the Ion data types, as follows:

Scalars

Follow the SQL syntax when applicable, as described in PartiQL-Ion type mapping (p. 443) section. For example:

- `5`
- `'foo'`
- `null`

Structs

Also known as tuples or objects in many formats and other data models.

Denoted by curly braces (`{...}`) with `struct` elements separated by commas.

- `{ 'id' : 3, 'arr': [1, 2] }`

Lists

Also known as arrays.

Denoted by square brackets (`[...]`) with list elements separated by commas.

- `[ 1, 'foo' ]`

Bags

Unordered collections in PartiQL.

Denoted by double angle brackets (`<< ... >>`) with bag elements separated by commas. In QLDB, a table can be thought of as a bag. However, a bag cannot be nested within documents in a table.

- `<< 1, 'foo' >>`

Example

The following is an example of the syntax for an `INSERT` statement with various Ion types.

```
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': '1nmeDdLo3AhGswBtyM1eYh' }
        ]
    },
    'ValidFromDate' : `2017-09-14T`, --Ion timestamp literal with day precision
    'ValidToDate' : `2020-06-25T`
}
```

Backtick notation

PartiQL fully covers all Ion data types, so you can write any statement without using backticks. But there are cases where this Ion literal syntax can make your statements clearer and more concise.

For example, to insert a document with Ion timestamp and symbol values, you can write the following statement using purely PartiQL syntax only.

```
INSERT INTO myTable VALUE
{
    'myTimestamp': to_timestamp('2019-09-04T'),
    'mySymbol': cast('foo' as symbol)
}
```

This is fairly verbose, so instead, you can use backticks to simplify your statement.

```
INSERT INTO myTable VALUE
{
    'myTimestamp': `2019-09-04T`,
    'mySymbol': `foo`
}
```

You can also enclose the entire structure in backticks to save a few more keystrokes.

```
INSERT INTO myTable VALUE
`{
    myTimestamp: 2019-09-04T,
    mySymbol: foo
}`
```

Important
Strings and symbols are different classes in PartiQL. This means that even if they have the same text, they are not equal. For example, the following PartiQL expressions evaluate to different Ion values.

```
'foo'
```

```
`foo`
```

Path navigation

When writing data manipulation language (DML) or query statements, you can access fields within nested structures using path steps. PartiQL supports dot notation for accessing field names of a parent structure. The following example accesses the `Model` field of a parent `Vehicle`.

```
Vehicle.Model
```

To access a specific element of a list, you can use the square brackets operator to denote a zero-based ordinal number. The following example accesses the element of `SecondaryOwners` with an ordinal number of `2`. In other words, this is the third element of the list.

```
SecondaryOwners[2]
```

Aliasing

QLDB supports open content and schema. So, when you are accessing particular fields in a statement, the best way to ensure that you get the results that you expect is to use aliases. For example, if you don't specify an explicit alias, the system generates an implicit one for your `FROM` sources.

```
SELECT VIN FROM Vehicle
--is rewritten to
SELECT Vehicle.VIN FROM Vehicle AS Vehicle
```

But the results are unpredictable for field name conflicts. If another field named `VIN` exists in a nested structure within the documents, the `VIN` values returned by this query might surprise you. As a best practice, write the following statement instead. This query declares `v` as an alias that ranges over the `Vehicle` table. The `AS` keyword is optional.

```
SELECT v.VIN FROM Vehicle [ AS ] v
```

Aliasing is particularly useful when pathing into nested collections within a document. For example, the following statement declares `o` as an alias that ranges over the collection `VehicleRegistration.Owners`.

```
SELECT o.SecondaryOwners
FROM VehicleRegistration AS r, @r.Owners AS o
```

The `@` character is technically optional here. But it explicitly indicates that you want the `Owners` structure within `VehicleRegistration`, not a different collection named `Owners` (if one existed).

PartiQL specification

For more information about the PartiQL query language, see the PartiQL Specification.

PartiQL statements

PartiQL extends SQL-92 to support documents in the Amazon Ion data format. Amazon QLDB supports the following PartiQL statements.

> **Note**
> QLDB does not support all PartiQL statements.
> This reference provides basic syntax and usage examples of PartiQL statements that you manually run on the QLDB console or the QLDB shell. For code examples that show how to run similar statements using a supported programming language, see the tutorials in Getting started with the driver (p. 40).

DDL statements (data definition language)

Data definition language (DDL) is the set of PartiQL statements that you use to manage database objects, such as tables and indexes. You use DDL to create and drop these objects.

DML statements (data manipulation language)

Data manipulation language (DML) is the set of PartiQL statements that you use to manage data in QLDB tables. You use DML statements to add, modify, or delete data in a table.

The following DML and query language statements are supported:

CREATE INDEX

Use the `CREATE INDEX` statement to create an index for a document field on a table in your Amazon QLDB ledger.

In QLDB, indexes are optional. They can improve query performance for seek operations. However, note the following:

- Indexes can only be created on empty tables.
- Indexes can only be created on a single top-level field. Composite, nested, unique, and function-based indexes are currently not supported.
- Indexes can't be dropped after they are created.
- Query performance is improved only when you use an equality predicate; for example, `fieldName = 123456789`.

  QLDB does not currently honor inequalities in query predicates. One implication of this is that range filtered scans are not implemented.
- Names of indexed fields can have a maximum of 128 characters.

> **Warning**
> QLDB requires an index to efficiently look up a document. Without an indexed field in the `WHERE` predicate clause, QLDB needs to do a table scan when reading documents. This can cause more query latency and can also lead to more concurrency conflicts.
> The best practice is to run queries that filter on a document `id` or an indexed field. For more information, see Using indexes to limit OCC conflicts (p. 332) in the *Concurrency model* section.

Topics

Syntax

```
CREATE INDEX ON table (field)
```

Parameters

table

The table where you want to create the index. The table must already exist.

field

The document field name for which to create the index. The field must be a top-level attribute.

Return value

`tableId` – The ID of the table on which you created an index.

Examples

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

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

Running programmatically using the driver

To learn how to programmatically run this statement using the QLDB driver for a supported language, see the tutorials in Getting started with the driver (p. 40). For example, see Step 3: Create tables, indexes, and sample data (p. 63) in the Java tutorial.

CREATE TABLE

Use the `CREATE TABLE` statement to create a new table in your Amazon QLDB ledger.

Tables have simple names with no namespaces. QLDB supports open content and does not enforce schema, so you don't define attributes or data types when creating tables.

Topics

Syntax

```
CREATE TABLE table
```

Parameters

table

The name of the table to create. The following are the naming constraints:

- Must only contain 1–128 alphanumeric characters or underscores.
- The first character must be a letter or an underscore.
- The remaining characters can be any combination of alphanumeric characters and underscores.
- The table names are case sensitive.
- The name must not be a QLDB PartiQL reserved word (p. 483).

Return value

`tableId` – The ID of the table that you created.

Examples

```
CREATE TABLE VehicleRegistration
```

Running programmatically using the driver

To learn how to programmatically run this statement using the QLDB driver for a supported language, see the tutorials in Getting started with the driver (p. 40). For example, see Step 3: Create tables, indexes, and sample data (p. 63) in the Java tutorial.

DELETE

Use the `DELETE` statement to delete an existing document from your Amazon QLDB ledger.

Topics

Syntax

```
DELETE FROM table [ AS table_alias ] [ BY id_alias ]
[ WHERE condition ]
```

Parameters

table

The user table containing the data to be deleted. DML statements are only supported in the default user view (p. 319). Each statement can only be executed on a single table.

AS *table_alias*

(Optional) A user-defined alias that ranges over a table to be deleted from. The `AS` keyword is optional.

BY *`id_alias`*

(Optional) A user-defined alias that binds to the `id` metadata field of each document in the result set. The alias must be declared in the `FROM` clause using the `BY` keyword. This is useful when you want to filter on the document ID (p. 323) while querying the default user view. For more information, see Using the BY clause to query document ID (p. 325).

`condition`

The selection criteria for the documents to be deleted.

Note
If you omit the `WHERE` clause, then all of the documents in the table are deleted.

Return value

`documentId` – The ID of each document that you deleted.

Examples

```
DELETE FROM VehicleRegistration AS r
WHERE r.VIN = '1HVBBAANXWH544237'
```

DROP TABLE

Use the `DROP TABLE` statement to inactivate an existing table in your Amazon QLDB ledger. You can use the UNDROP TABLE (p. 462) statement to reactivate it.

Topics

Syntax

```
DROP TABLE table
```

Parameters

`table`

The name of the table to drop. The table must already exist and have a status of `ACTIVE`.

Return value

`tableId` – The ID of the table that you dropped.

Examples

```
DROP TABLE VehicleRegistration
```

FROM

A statement that starts with `FROM` is an Amazon Ion extension that enables you to insert and remove specific elements within a document in Amazon QLDB. You can also use this statement to update existing elements in a document, similar to the UPDATE (p. 460) statement.

Topics

Syntax

Insert a new element within an existing document. To insert a new top-level document into a table, you must use INSERT (p. 455).

```
FROM table [ AS table_alias ] [ BY id_alias ]
[ WHERE condition ]
INSERT INTO element VALUE data [ AT key_name ]
```

Remove an existing element within a document, or remove an entire top-level document. The latter is semantically the same as the traditional DELETE (p. 450) syntax.

```
FROM table [ AS table_alias ] [ BY id_alias ]
[ WHERE condition ]
REMOVE element
```

Update one or more elements within a document. If an element does not exist, it is inserted. This is semantically the same as the traditional UPDATE (p. 460) syntax.

```
FROM table [ AS table_alias ] [ BY id_alias ]
[ WHERE condition ]
SET element = data [, element = data, ... ]
```

Parameters

table

The user table containing the data to be modified. DML statements are only supported in the default user view (p. 319). Each statement can only be executed on a single table.

In this clause, you can also include one or more collections that are nested within the specified table. For more details, see Nested collections (p. 453).

AS *table_alias*

(Optional) A user-defined alias that ranges over a table to be modified. All table aliases that are used in the `SET`, `REMOVE`, `INSERT INTO`, or `WHERE` clause must be declared in the `FROM` clause. The `AS` keyword is optional.

BY *`id_alias`*

(Optional) A user-defined alias that binds to the `id` metadata field of each document in the result set. The alias must be declared in the `FROM` clause using the `BY` keyword. This is useful when you want to filter on the document ID (p. 323) while querying the default user view. For more information, see Using the BY clause to query document ID (p. 325).

`condition`

The selection criteria for the documents to be modified.

Note
If you omit the `WHERE` clause, then all of the documents in the table are modified.

`element`

A document element to be created or modified.

`data`

A new value for the element.

AT *`key_name`*

A key name to be added within the documents to be modified. You must specify the corresponding `VALUE` along with the key name. This is required for inserting a new value `AT` a specific position within a document.

Nested collections

While you can execute a DML statement on a single table only, you can specify nested collections within documents in that table as additional sources. Each alias that you declare for a nested collection can be used in the `WHERE` clause and the `SET`, `INSERT`, or `REMOVE` clause.

For example, the `FROM` sources of the following statement include both the `VehicleRegistration` table and the nested `Owners.SecondaryOwners` structure.

```
FROM VehicleRegistration r, @r.Owners.SecondaryOwners o
WHERE r.VIN = '1N4AL11D75C109151' AND o.PersonId = 'abc123'
SET o.PersonId = 'def456'
```

This example updates the specific element of the `SecondaryOwners` list that has a `PersonId` of `'abc123'` within the `VehicleRegistration` document that has a `VIN` of `'1N4AL11D75C109151'`. This expression enables you to specify an element of a list by its value rather than its index.

Return value

`documentId` – The ID of each document that you updated or deleted.

Examples

Modify an element within a document. If the element does not exist, it is inserted.

```
FROM Vehicle AS v
WHERE v.Color = 'Silver'
SET v.Color = 'Shiny Gray'
```

Modify or insert an element and filter on the system-assigned document `id` metadata field.

```
FROM Vehicle AS v BY v_id
WHERE v_id = 'documentId'
SET v.Color = 'Shiny Gray'
```

Modify the `PersonId` field of the *first* element in the `Owners.SecondaryOwners` list within a document.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
SET r.Owners.SecondaryOwners[0].PersonId = 'abc123'
```

Remove an existing element within a document.

```
FROM Person AS p
WHERE p.FirstName = 'Rosemarie' AND p.LastName = 'Holloway'
REMOVE p.Address
```

Remove a whole document from a table.

```
FROM Person AS p
WHERE p.FirstName = 'Rosemarie' AND p.LastName = 'Holloway'
REMOVE p
```

Remove the *first* element of the `Owners.SecondaryOwners` list within a document in the `VehicleRegistration` table.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
REMOVE r.Owners.SecondaryOwners[0]
```

Add `{'Mileage':26500}` as a top-level name-value pair within a document in the `Vehicle` table.

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

Append `{'PersonId':'abc123'}` as a name-value pair in the `Owners.SecondaryOwners` field of a document in the `VehicleRegistration` table. Note that `Owners.SecondaryOwners` must already exist and must be a list data type for this statement to be valid. Otherwise, the keyword `AT` is required in the `INSERT INTO` clause.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners VALUE { 'PersonId' : 'abc123' }
```

Insert `{'PersonId':'abc123'}` as the *first* element in the existing `Owners.SecondaryOwners` list within a document.

```
FROM VehicleRegistration AS r
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners VALUE {'PersonId' : 'abc123'} AT 0
```

Append multiple name-value pairs to the existing `Owners.SecondaryOwners` list within a document.

```
FROM VehicleRegistration AS r
```

```
WHERE r.VIN = '1N4AL11D75C109151'
INSERT INTO r.Owners.SecondaryOwners << {'PersonId' : 'abc123'}, {'PersonId' : 'def456'} >>
```

Running programmatically using the driver

To learn how to programmatically run this statement using the QLDB driver for a supported language, see the tutorials in Getting started with the driver (p. 40). For example, see Step 5: Modify documents in a ledger (p. 115) in the Java tutorial.

INSERT

Use the `INSERT` statement to add one or more Amazon Ion documents to a table in Amazon QLDB.

Topics

Syntax

Insert a single document.

```
INSERT INTO table VALUE doc
```

Insert multiple documents.

```
INSERT INTO table << doc, doc, ... >>
```

Parameters

table

The user table where you want to insert the data. The table must already exist. DML statements are only supported in the default user view (p. 319).

doc

A valid QLDB document (p. 427). You must specify at least one document. Multiple documents must be separated by commas.

The document must be denoted with curly braces (`{...}`).

Each field name in the document is a case-sensitive Ion symbol that can be denoted with *single* quotation marks (`'...'`) in PartiQL.

String values are also denoted with *single* quotation marks (`'...'`) in PartiQL.

Any Ion literals can be denoted with backticks (`` `...` ``).

Note

Double angle brackets (`<< ... >>`) denote an unordered collection (known as a *bag* in PartiQL) and are required only if you want to insert multiple documents.

Return value

`documentId` – The ID of each document that you inserted.

Examples

Insert a single document.

```
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': '1nmeDdLo3AhGswBtyM1eYh' }
        ]
    },
    'ValidFromDate' : `2017-09-14T`, --Ion timestamp literal with day precision
    'ValidToDate' : `2020-06-25T`
}
```

Insert multiple documents (that is, a bag of documents).

```
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'
}
>>
```

Running programmatically using the driver

To learn how to programmatically run this statement using the QLDB driver for a supported language, see the tutorials in Getting started with the driver (p. 40). For example, see Step 3: Create tables, indexes, and sample data (p. 63) in the Java tutorial.

SELECT

Use the `SELECT` statement to retrieve data from one or more tables in Amazon QLDB.

Topics

Syntax

```
SELECT [ VALUE ] expression [ AS field_alias ] [, ...]
FROM source [ AS source_alias ] [ AT idx_alias ] [ BY id_alias ] [, source, ...]
[ WHERE condition ]
```

Parameters

VALUE

A qualifier for your expression that makes the query return the raw data type value, rather than the value being wrapped in a tuple structure.

expression

A projection formed from the `*` wildcard or a projection list of one or more document fields from the result set. An expression can consist of calls to PartiQL functions (p. 462) or fields that are modified by PartiQL operators (p. 481).

AS *field_alias*

(Optional) A temporary, user-defined alias for the field that is used in the final result set. The `AS` keyword is optional.

If you don't specify an alias for an expression that isn't a simple field name, the result set applies a default name to that field.

source

A source to be queried. The only sources currently supported are table names, inner joins (p. 458) between tables, nested `SELECT` queries (subject to Nested query limitations (p. 458)), and history function (p. 326) calls for a table.

You must specify at least one source. Multiple sources must be separated by commas.

AS *source_alias*

(Optional) A user-defined alias that ranges over a source to be queried. All source aliases that are used in the `SELECT` OR `WHERE` clause must be declared in the `FROM` clause. The `AS` keyword is optional.

AT *idx_alias*

(Optional) A user-defined alias that binds to the index number of an element within a list structure. In other words, this specifies the position of an element within a list. The alias must be declared in the `FROM` clause using the `AT` keyword.

BY *id_alias*

(Optional) A user-defined alias that binds to the `id` metadata field of each document in the result set. The alias must be declared in the `FROM` clause using the `BY` keyword. This is useful when you want to project or filter on the document ID (p. 323) while querying the default user view. For more information, see Using the BY clause to query document ID (p. 325).

condition

The selection criteria and join criteria (if applicable) for the query.

Note
If you omit the `WHERE` clause, then all of the documents in the table are retrieved.

Joins

Only inner joins are currently supported. You can write inner join queries using the explicit `INNER JOIN` clause, as follows. In this syntax, `JOIN` must be paired with `ON`, and the `INNER` keyword is optional.

```
SELECT expression
FROM table1 AS t1 [ INNER ] JOIN table2 AS t2
ON t1.element = t2.element
```

Or, you can write inner joins using the implicit syntax, as follows.

```
SELECT expression
FROM table1 AS t1, table2 AS t2
WHERE t1.element = t2.element
```

Nested query limitations

You can write nested queries (subqueries) within `SELECT` expressions and within `FROM` sources. The main restriction is that only the outermost query can access the global database environment. For example, suppose that you have a ledger with tables `VehicleRegistration` and `Person`. The following nested query is not valid because the inner `SELECT` tries to access `Person`.

```
SELECT r.VIN,
    (SELECT p.PersonId FROM Person AS p WHERE p.PersonId = r.Owners.PrimaryOwner.PersonId)
 AS PrimaryOwner
FROM VehicleRegistration AS r
```

Whereas the following nested query is valid.

```
SELECT r.VIN, (SELECT o.PrimaryOwner.PersonId FROM @r.Owners AS o) AS PrimaryOwner
FROM VehicleRegistration AS r
```

Examples

The following query shows a basic `SELECT` all wildcard with no filter criteria.

```
SELECT * FROM Vehicle
```

Tip
PartiQL is SQL compatible, so it supports table scans such as this example. But we don't recommend running statements without a `WHERE` predicate clause for production use. This can cause transaction timeouts on large tables.

The best practice is to run statements that filter on a document `id` or an indexed field. For more information, see Using indexes to limit OCC conflicts (p. 332) in the *Concurrency model* section.

The following shows `SELECT` projections with a string filter.

```
SELECT FirstName, LastName, Address
FROM Person
WHERE Address LIKE '%Seattle%'
```

The following shows a correlated subquery that flattens nested data. Note that the `@` character is technically optional here. But it explicitly indicates that you want the `Owners` structure within `VehicleRegistration`, not a different collection named `Owners` (if one existed).

```
SELECT
    r.VIN,
    o.SecondaryOwners
FROM
    VehicleRegistration AS r, @r.Owners AS o
```

The following shows a subquery in the `SELECT` list that projects nested data, and an implicit inner join.

```
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
```

The following shows an explicit inner join.

```
SELECT
    v.Make,
    v.Model,
    r.Owners
FROM
    VehicleRegistration AS r JOIN Vehicle AS v
ON
    r.VIN = v.VIN
```

The following shows a projection of the document `id` metadata field, using the `BY` clause.

```
SELECT
    r_id,
    r.VIN
FROM
    VehicleRegistration AS r BY r_id
WHERE
    r_id = 'documentId'
```

The following uses the `BY` clause to join the `DriversLicense` and `Person` tables on their `PersonId` and document `id` fields respectively.

```
SELECT * FROM DriversLicense AS d INNER JOIN Person AS p BY pid
ON d.PersonId = pid
WHERE pid = 'documentId'
```

The following uses the Committed view (p. 323) to join the DriversLicense and Person tables on their PersonId and document id fields respectively.

```
SELECT * FROM DriversLicense AS d INNER JOIN _ql_committed_Person AS cp
ON d.PersonId = cp.metadata.id
WHERE cp.metadata.id = 'documentId'
```

The following returns the PersonId and index number of each person in the Owners.SecondaryOwners list for a document in table VehicleRegistration.

```
SELECT s.PersonId, owner_idx
FROM VehicleRegistration AS r, @r.Owners.SecondaryOwners AS s AT owner_idx
WHERE r.VIN = 'KM8SRDHF6EU074761'
```

Running programmatically using the driver

To learn how to programmatically run this statement using the QLDB driver for a supported language, see the tutorials in Getting started with the driver (p. 40). For example, see Step 4: Query the tables in a ledger (p. 108) in the Java tutorial.

UPDATE

Use the UPDATE statement to modify the value of one or more elements within a document in Amazon QLDB. If an element does not exist, it is inserted.

Topics

Syntax

```
UPDATE table [ AS table_alias ] [ BY id_alias ]
SET element = data [, element = data, ... ]
[ WHERE condition ]
```

Parameters

table

The user table containing the data to be modified. DML statements are only supported in the default user view (p. 319). Each statement can only be executed on a single table.

AS *table_alias*

(Optional) A user-defined alias that ranges over a table to be updated. The AS keyword is optional.

BY *id_alias*

(Optional) A user-defined alias that binds to the id metadata field of each document in the result set. The alias must be declared in the UPDATE clause using the BY keyword. This is useful when you want to filter on the document ID (p. 323) while querying the default user view. For more information, see Using the BY clause to query document ID (p. 325).

element

A document element to be created or modified.

data

A new value for the element.

condition

The selection criteria for the documents to be modified.

Note
If you omit the `WHERE` clause, then all of the documents in the table are modified.

Return value

`documentId` – The ID of each document that you updated.

Examples

Update a field in a document. If the field does not exist, it is inserted.

```
UPDATE Person AS p
SET p.LicenseNumber = 'HOLLOR123ZZ'
WHERE p.FirstName = 'Rosemarie' AND p.LastName = 'Holloway'
```

Filter on the system-assigned document `id` metadata field.

```
UPDATE Person AS p BY pid
SET p.LicenseNumber = 'HOLLOR123ZZ'
WHERE pid = 'documentId'
```

Overwrite an entire document.

```
UPDATE Person AS p
SET p = {
    'FirstName' : 'Rosemarie',
    'LastName' : 'Holloway',
    'DOB' : `1977-06-18T`,
    'GovId' : 'LEWISR261LL',
    'GovIdType' : 'Driver License',
    'Address' : '4637 Melrose Street, Ellensburg, WA, 98926'
}
WHERE p.FirstName = 'Rosemarie' AND p.LastName = 'Holloway'
```

Modify the `PersonId` field of the *first* element in the `Owners.SecondaryOwners` list within a document.

```
UPDATE VehicleRegistration AS r
SET r.Owners.SecondaryOwners[0].PersonId = 'abc123'
WHERE r.VIN = '1N4AL11D75C109151'
```

Running programmatically using the driver

To learn how to programmatically run this statement using the QLDB driver for a supported language, see the tutorials in Getting started with the driver (p. 40). For example, see Step 5: Modify documents in a ledger (p. 115) in the Java tutorial.

UNDROP TABLE

Use the `UNDROP TABLE` statement to reactivate a table that you previously dropped (that is, inactivated) in your Amazon QLDB ledger.

Topics

Syntax

```
UNDROP TABLE "tableId"
```

Parameters

`tableId`

The unique ID of the table to reactivate, denoted by double quotation marks.

The table must have been previously dropped, meaning that it exists in the system catalog table (p. 329) `information_schema.user_tables` and has a status of `INACTIVE`. There must also be no active, existing table with the same name.

Return value

`tableId` – The ID of the table that you reactivated.

Examples

```
UNDROP TABLE "5PLf9SXwndd63lPaSIa0O6"
```

PartiQL functions

PartiQL in Amazon QLDB supports the following built-in variants of SQL standard functions.

Note
Any SQL functions that are not included in this list are not currently supported in QLDB.

Unknown type (null and missing) propagation

Unless otherwise stated, all of these functions propagate null and missing argument values. *Propagation* of `NULL` or `MISSING` is defined as returning `NULL` if any function argument is either `NULL` or `MISSING`. The following are examples of this propagation.

```
CHAR_LENGTH(null)    -- null
CHAR_LENGTH(missing) -- null (also returns null)
```

Aggregate functions

- AVG function (p. 464)
- COUNT function (p. 468)
- MAX function (p. 472)
- MIN function (p. 473)
- SIZE function (p. 474)
- SUM function (p. 476)

Conditional functions

- COALESCE function (p. 467)
- EXISTS function (p. 470)
- NULLIF function (p. 473)

Date and time functions

- DATE_ADD function (p. 468)
- DATE_DIFF function (p. 469)
- EXTRACT function (p. 471)
- UTCNOW function (p. 480)

Scalar functions

- TXID function (p. 479)

String functions

- CHAR_LENGTH function (p. 467)
- CHARACTER_LENGTH function (p. 467)
- LOWER function (p. 472)
- SUBSTRING function (p. 475)
- TRIM function (p. 478)
- UPPER function (p. 479)

Data type formatting functions

- CAST function (p. 464)
- TO_STRING function (p. 476)
- TO_TIMESTAMP function (p. 477)

AVG function

Returns the average (arithmetic mean) of the input expression values. The `AVG` function works with numeric values and ignores null or missing values.

Syntax

```
AVG ( expression )
```

Arguments

`expression`

The field name or expression of a numeric data type that the function operates on.

Data types

Supported argument types:

- `int`
- `decimal`
- `float`

Return type: `decimal`

Examples

```
SELECT AVG(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r  -- 147.19
SELECT AVG(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >>         -- 2.
```

CAST function

Evaluates a given expression to a value and converts the value to a specified target data type. If the conversion cannot be made, the function returns an error.

Syntax

```
CAST ( expression AS type )
```

Arguments

`expression`

The field name or expression that evaluates to a value that the function converts. Converting null values returns nulls. This parameter can be any of the supported Data types (p. 427).

`type`

The name of the target data type for conversion. This parameter can be one of the supported Data types (p. 427).

Return type

The data type that is specified by the *type* argument.

Examples

The following examples show the propagation of unknown types (NULL or MISSING).

```
CAST(null    AS null)    -- null
CAST(missing AS null)    -- null
CAST(missing AS missing) -- null
CAST(null    AS missing) -- null
CAST(null    AS boolean) -- null (null AS any data type name results in null)
CAST(missing AS boolean) -- null (missing AS any data type name results in null)
```

Any value that is not an unknown type cannot be cast to NULL or MISSING.

```
CAST(true AS null)    -- error
CAST(true AS missing) -- error
CAST(1    AS null)    -- error
CAST(1    AS missing) -- error
```

The following examples show casting AS boolean.

```
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'`)
```

The following examples show casting AS integer.

```
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
```

The following examples show casting AS float.

```
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
```

The following examples show casting `AS decimal`.

```
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
```

The following examples show casting `AS timestamp`.

```
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
```

The following examples show casting `AS symbol`.

```
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'
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'
```

The following examples show casting `AS string`.

```
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"
```

The following examples show casting `AS struct`.

```
CAST(`{ a: 1 }` AS struct) -- {a:1}
CAST(true       AS struct) -- err
```

The following examples show casting `AS list`.

```
CAST(`[1, 2, 3]`        AS list) -- [1,2,3]
CAST(<<'a', { 'b':2 }>> AS list) -- ["a",{'b':2}]
CAST({ 'b':2 }          AS list) -- error
```

CHAR_LENGTH function

Returns the number of characters in the specified string, where *character* is defined as a single unicode code point.

Syntax

```
CHAR_LENGTH ( string )
```

`CHAR_LENGTH` is a synonym of CHARACTER_LENGTH function (p. 467).

Arguments

`string`

The field name or expression of data type `string` that the function evaluates.

Return type

`int`

Examples

```
CHAR_LENGTH('')         -- 0
CHAR_LENGTH('abcdefg') -- 7
CHAR_LENGTH('e#')      -- 2 (because 'e#' is two code points: the letter 'e' and combining
 character U+032B)
```

CHARACTER_LENGTH function

Synonym of the `CHAR_LENGTH` function.

See CHAR_LENGTH function (p. 467).

COALESCE function

Given a list of one or more arguments, evaluates the arguments in order from left to right and returns the first value that is not an unknown type (`NULL` or `MISSING`). If all argument types are unknown, the result is `NULL`.

The `COALESCE` function does not propagate `NULL` and `MISSING`.

Syntax

```
COALESCE ( expression [, expression, ... ] )
```

Arguments

`expression`

The list of one or more field names or expressions that the function evaluates. Each argument can be any of the supported Data types (p. 427).

Return type

Any supported data type. The return type is either `NULL` or the same as the type of the first expression that evaluates to a non-null and non-missing value.

Examples

```
COALESCE(missing, missing) -- null
COALESCE(1, null)          -- 1
COALESCE(null, null, 1)    -- 1
COALESCE(null, 'string')   -- 'string'
COALESCE(missing, 1)       -- 1
```

COUNT function

Returns the number of documents that are defined by the expression.

The `COUNT` function has two variations. `COUNT ( * )` counts all the documents in the target table whether or not they include null or missing values. `COUNT ( expression )` computes the number of documents with non-null values in a specific, existing field or expression.

Syntax

```
COUNT ( * | expression )
```

Arguments

`expression`

The field name or expression that the function operates on. This parameter can be any of the supported Data types (p. 427).

Return type

`int`

Examples

```
SELECT COUNT(*) FROM VehicleRegistration r                          -- 5
SELECT COUNT(r.VIN) FROM VehicleRegistration r WHERE r.City = 'Tacoma'   -- 1
```

DATE_ADD function

Increments a given timestamp value by a specified interval.

Syntax

```
DATEADD( datepart, interval, timestamp )
```

Arguments

datepart

The date part that the function operates on. This parameter can be one of the following:

- `year`
- `month`
- `day`
- `hour`
- `minute`
- `second`

interval

The integer that specifies the interval to add to the given *timestamp*. A negative integer subtracts the interval.

timestamp

The field name or expression of data type `timestamp` that the function increments.

Return type

`timestamp`

Examples

```
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
```

DATE_DIFF function

Returns the difference between the specified date parts of two given timestamps.

Syntax

```
DATE_DIFF( datepart, timestamp1, timestamp2 )
```

Arguments

datepart

The date part that the function operates on. This parameter can be one of the following:

- `year`
- `month`
- `day`

- `hour`
- `minute`
- `second`

`timestamp1`, *`timestamp2`*

The two field names or expressions of data type `timestamp` that the function compares. If *`timestamp2`* is later than *`timestamp1`*, the result is positive. If *`timestamp2`* is earlier than *`timestamp1`*, the result is negative.

Return type

`int`

Examples

```
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, `2010-01-01T23:00T`, `2010-01-02T01:00T`) -- 0 (must be at least 24 hours
 apart to evaluate as a 1 day difference)
```

EXISTS function

Given a value, returns `TRUE` if the value is a non-empty collection. Otherwise, returns `FALSE`. If the input to `EXISTS` is not a container, the result is `FALSE`.

The `EXISTS` function does not propagate `NULL` and `MISSING`.

Syntax

```
EXISTS ( value )
```

Arguments

`value`

The field name or expression that the function evaluates. This parameter can be any of the supported Data types (p. 427).

Return type

`bool`

Examples

```
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(`<<>>`)      -- false (empty bag)
EXISTS(`<<null>>`)  -- true (non-empty bag)
EXISTS(1)           -- false
EXISTS(`2017T`)     -- false
EXISTS(null)        -- false
EXISTS(missing)     -- false
```

EXTRACT function

Returns the integer value of a specified date part from a given timestamp.

Syntax

```
EXTRACT ( datepart FROM timestamp )
```

Arguments

`datepart`

The date part that the function extracts. This parameter can be one of the following:

- `year`
- `month`
- `day`
- `hour`
- `minute`
- `second`
- `timezone_hour`
- `timezone_minute`

`timestamp`

The field name or expression of data type `timestamp` that the function extracts from. If this parameter is an unknown type (`NULL` or `MISSING`), the function returns `NULL`.

Return type

`int`

Examples

```
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
```

LOWER function

Converts all uppercase characters to lowercase characters in a given string.

Syntax

```
LOWER ( string )
```

Arguments

`string`

The field name or expression of data type `string` that the function converts.

Return type

`string`

Examples

```
LOWER('AbCdEfG!@#$')  -- 'abcdefg!@#$'
```

MAX function

Returns the maximum value in a set of documents.

Syntax

```
MAX ( expression )
```

Arguments

`expression`

The field name or expression of a numeric data type that the function operates on.

Data types

Supported argument types:

- `int`
- `decimal`
- `float`

Supported return types:

- `int`
- `decimal`
- `float`

Examples

```
SELECT MAX(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r  -- 442.30
```

MIN function

Syntax

```
MIN ( expression )
```

Returns the minimum value in a set of documents.

Arguments

`expression`

The field name or expression of a numeric data type that the function operates on.

Data types

Supported argument types:

- `int`
- `decimal`
- `float`

Supported return types:

- `int`
- `decimal`
- `float`

Examples

```
SELECT MIN(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r  -- 30.45
```

NULLIF function

Given two expressions, returns `NULL` if the two expressions evaluate to the same value. Otherwise, returns the result of evaluating the first expression.

The `NULLIF` function does not propagate `NULL` and `MISSING`.

Syntax

```
NULLIF ( expression1, expression2 )
```

Arguments

expression1, *expression2*

The two field names or expressions that the function compares. These parameters can be any of the supported Data types (p. 427).

Return type

Any supported data type. The return type is either `NULL` or the same as the type of the first expression.

Examples

```
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
```

SIZE function

Returns the number of elements in a given container data type (list, structure, or bag).

Syntax

```
SIZE ( container )
```

Arguments

container

The container field name or expression that the function operates on.

Data types

Supported argument types:

- list
- structure
- bag

Return type: `int`

If the input to `SIZE` is not a container, the function throws an error.

Examples

```
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
```

SUBSTRING function

Returns a substring from a given string. The substring starts from the specified start index and ends at the last character of the string, or at the specified length.

Syntax

```
SUBSTRING ( string, start-index [, length ] )
```

Arguments

`string`

The field name or expression of data type `string` from which to extract a substring.

`start-index`

The start position within the *`string`* from which to begin the extraction. This number can be negative.

The first character of *`string`* has an index of 1.

`length`

(Optional) The number of characters (code points) to extract from the *`string`*, starting at *`start-index`* and ending at (*`start-index`* + *`length`*) - 1. In other words, the length of the substring. This number cannot be negative.

If this parameter is not provided, the function proceeds until the end of the *`string`*.

Return type

`string`

Examples

```
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)       -- ''
```

SUM function

Returns the sum of the input field or expression values. The `SUM` function works with numeric values and ignores null or missing values.

Syntax

```
SUM ( expression )
```

Arguments

`expression`

The field name or expression of a numeric data type that the function operates on.

Data types

Supported argument types:

- `int`
- `decimal`
- `float`

Supported return types:

- `int` – For integer arguments
- `decimal` – For decimal or floating point arguments

Examples

```
SELECT SUM(r.PendingPenaltyTicketAmount) FROM VehicleRegistration r  -- 735.95
SELECT SUM(a) FROM << { 'a' : 1 }, { 'a': 2 }, { 'a': 3 } >>         -- 6
```

TO_STRING function

Returns a string representation of a given timestamp in the specified format pattern.

Syntax

```
TO_STRING ( timestamp, 'format' )
```

Arguments

`timestamp`

The field name or expression of data type `timestamp` that the function converts to a string.

format

The string literal that specifies the format pattern of the result, in terms of its date parts. For valid formats, see Timestamp format strings (p. 480).

Return type

`string`

Examples

```
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"
```

TO_TIMESTAMP function

Given a string that represents a timestamp, converts the string to a `timestamp` data type. This is the inverse operation of `TO_STRING`.

Syntax

```
TO_TIMESTAMP ( string [, 'format' ] )
```

Arguments

string

The field name or expression of data type `string` that the function converts to a timestamp.

format

(Optional) The string literal that defines the format pattern of the input *string*, in terms of its date parts. For valid formats, see Timestamp format strings (p. 480).

If this argument is omitted, the function assumes that the *string* is in the format of a standard Ion timestamp. This is the recommended way to parse an Ion timestamp using this function.

Zero padding is optional when using a single-character format symbol (such as `y`, `M`, `d`, `H`, `h`, `m`, `s`) but is required for their zero-padded variants (such as `yyyy`, `MM`, `dd`, `HH`, `hh`, `mm`, `ss`).

Special treatment is given to two-digit years (format symbol `yy`). 1900 is added to values greater than or equal to 70, and 2000 is added to values less than 70.

Month names and AM or PM indicators are case insensitive.

Return type

`timestamp`

Examples

```
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('Febrary 2016', 'MMMM yyyy')     -- `2016-02T`
```

TRIM function

Trims a given string by removing the leading and trailing blank spaces or a specified *set* of characters.

Syntax

```
TRIM ( [ LEADING | TRAILING | BOTH [ characters ] FROM ] string)
```

Arguments

LEADING

(Optional) Indicates that the blank spaces or specified characters are to be removed from the beginning of `string`. If not specified, the default behavior is `BOTH`.

TRAILING

(Optional) Indicates that the blank spaces or specified characters are to be removed from the end of `string`. If not specified, the default behavior is `BOTH`.

BOTH

(Optional) Indicates that both the leading and trailing blank spaces or specified characters are to be removed from the beginning and end of `string`.

`characters`

(Optional) The *set* of characters to remove, specified as a `string`.

If this parameter is not provided, blank spaces are removed.

`string`

The field name or expression of data type `string` that the functions trims.

Return type

`string`

Examples

```
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'
```

TXID function

Returns the unique transaction ID of the current statement that you are executing. This is the value that is assigned to a document's `txId` metadata field when the current transaction is committed to the journal.

Syntax

```
TXID()
```

Arguments

None

Return type

`string`

Examples

```
SELECT TXID() FROM << 0 >>  -- "L7S9iJqcn9W2M4qOEn27ay"
```

UPPER function

Converts all lowercase characters to uppercase characters in a given string.

Syntax

```
UPPER ( string )
```

Arguments

`string`

The field name or expression of data type `string` that the function converts.

Return type

`string`

Examples

```
UPPER('AbCdEfG!@#$')  -- 'ABCDEFG!@#$'
```

UTCNOW function

Returns the current time in Coordinated Universal Time (UTC) as a `timestamp`.

Syntax

```
UTCNOW()
```

Arguments

None

Return type

`timestamp`

Examples

```
SELECT UTCNOW() FROM << 0 >>  -- 2019-12-27T20:12:16.999Z
```

Timestamp format strings

This section provides reference information for timestamp format strings.

Timestamp format strings apply to the `TO_STRING` and `TO_TIMESTAMP` functions. These strings can contain date part separators (such as '-', '/', or ':') and the following format symbols.

Format	Example	Description
`yy`	70	Two-digit year
`y`	1970	Four-digit year
`yyyy`	1970	Zero-padded four-digit year
`M`	1	Month integer
`MM`	01	Zero-padded month integer
`MMM`	Jan	Abbreviated month name
`MMMM`	January	Full month name
`d`	2	Day of the month (1–31)
`dd`	02	Zero-padded day of the month (01–31)
`a`	AM or PM	Meridian indicator (for 12-hour clock)
`h`	3	Hour (12-hour clock, 1–12)
`hh`	03	Zero-padded hour (12-hour clock, 01–12)
`H`	3	Hour (24-hour clock, 0–23)

Format	Example	Description
`HH`	03	Zero-padded hour (24-hour clock, 00–23)
`m`	4	Minutes (0–59)
`mm`	04	Zero-padded minutes (00–59)
`s`	5	Seconds (0–59)
`ss`	05	Zero-padded seconds (00–59)
`S`	0	Fraction of a second (precision: 0.1, range: 0.0–0.9)
`SS`	06	Fraction of a second (precision: 0.01, range: 0.0–0.99)
`SSS`	060	Fraction of a second (precision: 0.001, range: 0.0–0.999)
`X`	+07 or Z	Offset from UTC in hours, or "Z" if the offset is 0
`XX`	+0700 or Z	Offset from UTC in hours and minutes, or "Z" if the offset is 0
`XXX`	+07:00 or Z	Offset from UTC in hours and minutes, or "Z" if the offset is 0
`x`	+07	Offset from UTC in hours
`xx`	+0700	Offset from UTC in hours and minutes
`xxx`	+07:00	Offset from UTC in hours and minutes

PartiQL operators

PartiQL in Amazon QLDB supports the following SQL standard operators.

Note
Any SQL operators that are not included in this list are not currently supported in QLDB.

Arithmetic operators

Operator	Description
`+`	Add
`–`	Subtract
`*`	Multiply
`/`	Divide

Operator	Description
`%`	Modulo

Comparison operators

Operator	Description
`=`	Equal to
`>`	Greater than
`<`	Less than
`>=`	Greater than or equal to
`<=`	Less than or equal to
`<>`	Not equal to

Logical operators

Operator	Description
`AND`	`TRUE` if all the conditions separated by `AND` are `TRUE`
`BETWEEN`	`TRUE` if the operand is within the range of comparisons
`IN`	`TRUE` if the operand is equal to one of a list of expressions
`IS`	`TRUE` if the operand is a given data type, including `NULL` or `MISSING`
`LIKE`	`TRUE` if the operand matches a pattern
`NOT`	Reverses the value of a given Boolean expression
`OR`	`TRUE` if any of the conditions separated by `OR` is `TRUE`

String operators

Operator	Description
`||`	Concatenates two strings on either side of the `||` operator and returns the concatenated string. If one or both strings is `NULL`, the result of the concatenation is null.

Reserved words

The following is a list of PartiQL reserved words in Amazon QLDB. You can use a reserved word as a quoted identifier with double quotation marks (for example, `"user"`).

Important
The keywords in this list are all considered reserved because PartiQL is backwards compatible with SQL-92. However, QLDB only supports a subset of these reserved words. For the list of SQL keywords that QLDB currently supports, see the following sections:

- PartiQL functions (p. 462)
- PartiQL operators (p. 481)
- PartiQL statements (p. 447)

```
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 Ion data format reference

Amazon QLDB uses a data notation model that unifies Amazon Ion with a subset of PartiQL (p. 426) types. This section provides a reference overview of the Ion document data format, separate from its integration with PartiQL.

Querying Ion with PartiQL in Amazon QLDB

For the syntax and semantics of querying Ion data with PartiQL in QLDB, see Querying Ion with PartiQL (p. 444) in the *Amazon QLDB PartiQL reference*.

For code examples that query and process Ion data in a QLDB ledger, see Ion code examples (p. 489).

Topics

What is Amazon Ion?

Ion is an open source, richly typed, self-describing, hierarchical data serialization format that was originally developed internally at Amazon. It is based on an abstract data model that lets you store both structured and unstructured data. It is a superset of JSON, meaning that any valid JSON document is also a valid Ion document. This guide assumes a baseline working knowledge of JSON. If you are not already familiar with JSON, see Introducing JSON for more information.

You can notate Ion documents interchangeably in either human-readable text form or binary-encoded form. Like JSON, the text form is easy to read and write, supporting rapid prototyping. The binary encoding is more compact and efficient to persist, transmit, and parse. An Ion processor can transcode between both formats to represent exactly the same set of data structures without any loss of data. This enables applications to optimize how it processes data for different use cases.

Note
The Ion data model is strictly value-based and does not support references. Thus, the data model can represent data hierarchies that can be nested to arbitrary depth, but not directed graphs.

Ion Specification

For a full list of Ion core data types with complete descriptions and value formatting details, see the Ion specification document on the Amazon GitHub site.

To streamline application development, Amazon Ion provides client libraries that process Ion data for you. For code examples of common use cases for processing Ion data, see the Amazon Ion Cookbook on GitHub.

JSON compatible

Similar to JSON, you compose Amazon Ion documents with a set of primitive data types and a set of recursively defined container types. Ion includes the following traditional JSON data types:

- `null`: A generic, untyped null (empty) value. Additionally, as described in the following section, Ion supports a distinct null type for each primitive type.
- `bool`: Boolean values.
- `string`: Unicode text literals.
- `list`: Ordered heterogeneous collections of values.
- `struct`: Unordered collections of name-value pairs. Like JSON, `struct` allows multiple values per name, but this is generally discouraged.

Extensions from JSON

Number types

Instead of the ambiguous JSON `number` type, Amazon Ion strictly defines numbers as one of the following types:

- `int`: Signed integers of arbitrary size.
- `decimal`: Decimal-encoded real numbers of arbitrary precision.
- `float`: Binary-encoded floating point numbers (64-bit IEEE).

When parsing documents, an Ion processor assigns number types as follows:

- `int`: Numbers with no exponent or decimal point (for example, `100200`).
- `decimal`: Numbers with a decimal point and no exponent (for example, `0.00001`, `200.0`).
- `float`: Numbers with an exponent, such as scientific notation or E-notation (for example, `2e0`, `3.1e-4`).

New data types

Amazon Ion adds the following data types:

- `timestamp`: Date/time/timezone moments of arbitrary precision.
- `symbol`: Unicode symbolic atoms (such as identifiers).
- `blob`: Binary data of user-defined encoding.
- `clob`: Text data of user-defined encoding.
- `sexp`: Ordered collections of values with application-defined semantics.

Null types

In addition to the generic null type defined by JSON, Amazon Ion supports a distinct null type for each primitive type. This indicates a lack of value while maintaining a strict data type.

```
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 text example

```
// 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 references

- ion-java
- ion-js
- ion-python

Ion code examples in QLDB

This section provides code examples that process Amazon Ion data by reading and writing document values in an Amazon QLDB ledger. The code examples use the QLDB driver to run PartiQL statements on the ledger. These examples are part of the sample application in the Getting started with the driver (p. 40) tutorials, and are open-sourced on the AWS Samples GitHub site.

For general code examples that show common use cases of processing Ion data, see the Amazon Ion Cookbook on GitHub.

Running the code

The tutorial code for each programming language does the following steps:

1. Connect to the `vehicle-registration` sample ledger.
2. Create a table named `IonTypes`.
3. Insert a document into the table with a single `Name` field.
4. For each supported Ion data type:
   a. Update the document's `Name` field with a literal value of the data type.
   b. Query the table to get the latest revision of the document.
   c. Validate that the value of `Name` retained its original data type properties by checking that it matches the expected type.
5. Delete the `IonTypes` table.

Note
Before you run this tutorial code, you must create a ledger named `vehicle-registration`.

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;
        }
    }
}
```

TypeScript

```
/*
 * 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.
 */
var 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.STRING);
            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.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(transaction_executor, 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 transaction_executor: :py:class:`pyqldb.execution.executor.Executor`
    :param transaction_executor: An Executor object allowing for execution of
 statements within a transaction.

    :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:`IonPyDict`

 /: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)
    transaction_executor.execute_statement(update_query, parameter)
    logger.info('Updated record.')

    search_query = 'SELECT VALUE Name FROM {}'.format(TABLE_NAME)
    cursor = transaction_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(transaction_executor, table_name):
    """
    Delete a 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 delete.
    :rtype: int
    :return: The number of changes to the database.
    """
    logger.info("Deleting '{}' table...".format(table_name))
    cursor = transaction_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'))

    driver.execute_lambda(lambda transaction_executor:
 create_table(transaction_executor, TABLE_NAME)
                          and insert_documents(transaction_executor, TABLE_NAME,
 [{'Name': 'val'}])
                          and update_record_and_verify_type(transaction_executor,
 python_bytes, IonPyBytes,
                                                            IonType.BLOB)
                          and update_record_and_verify_type(transaction_executor,
 python_bool, IonPyBool,
                                                            IonType.BOOL)
                          and update_record_and_verify_type(transaction_executor,
python_float, IonPyFloat,
                                                                  IonType.FLOAT)
                          and update_record_and_verify_type(transaction_executor,
python_decimal, IonPyDecimal,
                                                                  IonType.DECIMAL)
                          and update_record_and_verify_type(transaction_executor,
python_string, IonPyText,
                                                                  IonType.STRING)
                          and update_record_and_verify_type(transaction_executor,
python_int, IonPyInt,
                                                                  IonType.INT)
                          and update_record_and_verify_type(transaction_executor,
python_null, IonPyNull,
                                                                  IonType.NULL)
                          and update_record_and_verify_type(transaction_executor,
python_datetime,
                                                                  IonPyTimestamp,
IonType.TIMESTAMP)
                          and update_record_and_verify_type(transaction_executor,
python_list, IonPyList,
                                                                  IonType.LIST)
                          and update_record_and_verify_type(transaction_executor,
python_dict, IonPyDict,
                                                                  IonType.STRUCT)
                          and update_record_and_verify_type(transaction_executor,
ion_clob, IonPyBytes,
                                                                  IonType.CLOB)
                          and update_record_and_verify_type(transaction_executor,
ion_blob, IonPyBytes,
                                                                  IonType.BLOB)
                          and update_record_and_verify_type(transaction_executor,
ion_bool, IonPyBool,
                                                                  IonType.BOOL)
                          and update_record_and_verify_type(transaction_executor,
ion_decimal, IonPyDecimal,
                                                                  IonType.DECIMAL)
                          and update_record_and_verify_type(transaction_executor,
ion_float, IonPyFloat,
                                                                  IonType.FLOAT)
                          and update_record_and_verify_type(transaction_executor,
ion_int, IonPyInt,
                                                                  IonType.INT)
                          and update_record_and_verify_type(transaction_executor,
ion_list, IonPyList,
                                                                  IonType.LIST)
                          and update_record_and_verify_type(transaction_executor,
ion_null, IonPyNull,
                                                                  IonType.NULL)
                          and update_record_and_verify_type(transaction_executor,
ion_sexp, IonPyList,
                                                                  IonType.SEXP)
                          and update_record_and_verify_type(transaction_executor,
ion_string, IonPyText,
                                                                  IonType.STRING)
                          and update_record_and_verify_type(transaction_executor,
ion_struct, IonPyDict,
                                                                  IonType.STRUCT)
                          and update_record_and_verify_type(transaction_executor,
ion_symbol, IonPySymbol,
                                                                  IonType.SYMBOL)
                          and update_record_and_verify_type(transaction_executor,
ion_timestamp,
                                                                  IonPyTimestamp,
IonType.TIMESTAMP)
```

```
                           and update_record_and_verify_type(transaction_executor,
 ion_null_clob, IonPyNull,
                                                             IonType.CLOB)
                           and update_record_and_verify_type(transaction_executor,
 ion_null_blob, IonPyNull,
                                                             IonType.BLOB)
                           and update_record_and_verify_type(transaction_executor,
 ion_null_bool, IonPyNull,
                                                             IonType.BOOL)
                           and update_record_and_verify_type(transaction_executor,
 ion_null_decimal,
                                                             IonPyNull,
 IonType.DECIMAL)
                           and update_record_and_verify_type(transaction_executor,
 ion_null_float, IonPyNull,
                                                             IonType.FLOAT)
                           and update_record_and_verify_type(transaction_executor,
 ion_null_int, IonPyNull,
                                                             IonType.INT)
                           and update_record_and_verify_type(transaction_executor,
 ion_null_list, IonPyNull,
                                                             IonType.LIST)
                           and update_record_and_verify_type(transaction_executor,
 ion_null_sexp, IonPyNull,
                                                             IonType.SEXP)
                           and update_record_and_verify_type(transaction_executor,
 ion_null_string, IonPyNull,
                                                             IonType.STRING)
                           and update_record_and_verify_type(transaction_executor,
 ion_null_struct, IonPyNull,
                                                             IonType.STRUCT)
                           and update_record_and_verify_type(transaction_executor,
 ion_null_symbol, IonPyNull,
                                                             IonType.SYMBOL)
                           and update_record_and_verify_type(transaction_executor,
 ion_null_timestamp,
                                                             IonPyNull,
 IonType.TIMESTAMP)
                           and delete_table(transaction_executor, TABLE_NAME),
                           lambda retry_attempt: logger.info('Retrying due to OCC
 conflict...'))


if __name__ == '__main__':
    """
    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() as driver:
            insert_and_verify_ion_types(driver)
    except Exception:
        logger.exception('Error updating and validating Ion types.')
```

Amazon QLDB API reference

This section describes the low-level API operations for Amazon QLDB that are accessible via HTTP, the AWS Command Line Interface (AWS CLI), or an AWS SDK:

- *Amazon QLDB* – The QLDB control plane. These API actions are used only for managing ledgers and for non-transactional data operations. You can use these actions to create, delete, describe, list, and update ledgers. You can also verify a document cryptographically, and export or stream journal blocks.
- *Amazon QLDB Session* – The QLDB transactional data plane. You can use this API to execute data transactions on a ledger with PartiQL (p. 426) statements.

  Important
  Instead of interacting directly with the *QLDB Session* API, we recommend that you use the QLDB driver or the QLDB shell to execute data transactions on a ledger.

  - If you are working with an AWS SDK, use the QLDB driver. The driver provides a high-level abstraction layer above the *QLDB Session* data plane and manages `SendCommand` API calls for you. For information and a list of supported programming languages, see Getting started with the driver (p. 40).
  - If you are working with the AWS CLI, use the QLDB shell. The shell is a command line interface that uses the QLDB driver for Python to interact with a ledger. For information, see Using the Amazon QLDB shell (data plane only) (p. 20).

Topics

Actions

The following actions are supported by Amazon QLDB:

- CancelJournalKinesisStream (p. 502)
- CreateLedger (p. 504)
- DeleteLedger (p. 508)
- DescribeJournalKinesisStream (p. 510)
- DescribeJournalS3Export (p. 512)
- DescribeLedger (p. 514)
- ExportJournalToS3 (p. 517)
- GetBlock (p. 520)
- GetDigest (p. 523)
- GetRevision (p. 525)
- ListJournalKinesisStreamsForLedger (p. 528)
- ListJournalS3Exports (p. 531)
- ListJournalS3ExportsForLedger (p. 533)

The following actions are supported by Amazon QLDB Session:

Amazon QLDB

The following actions are supported by Amazon QLDB:

CancelJournalKinesisStream

Service: Amazon QLDB

Ends a given Amazon QLDB journal stream. Before a stream can be canceled, its current status must be `ACTIVE`.

You can't restart a stream after you cancel it. Canceled QLDB stream resources are subject to a 7-day retention period, so they are automatically deleted after this limit expires.

Request Syntax

```
DELETE /ledgers/name/journal-kinesis-streams/streamId HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

name (p. 502)

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

streamId (p. 502)

The UUID (represented in Base62-encoded text) of the QLDB journal stream to be canceled.

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "StreamId": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

StreamId (p. 502)

The UUID (Base62-encoded text) of the canceled QLDB journal stream.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

CreateLedger

Service: Amazon QLDB

Creates a new ledger in your AWS account in the current Region.

Request Syntax

```
POST /ledgers HTTP/1.1
Content-type: application/json

{
   "DeletionProtection": boolean,
   "Name": "string",
   "PermissionsMode": "string",
   "Tags": {
      "string" : "string"
   }
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

DeletionProtection (p. 504)

The flag that prevents a ledger from being deleted by any user. If not provided on ledger creation, this feature is enabled (`true`) by default.

If deletion protection is enabled, you must first disable it before you can delete the ledger using the QLDB API or the AWS Command Line Interface (AWS CLI). You can disable it by calling the `UpdateLedger` operation to set the flag to `false`. The QLDB console disables deletion protection for you when you use it to delete a ledger.

Type: Boolean

Required: No

Name (p. 504)

The name of the ledger that you want to create. The name must be unique among all of your ledgers in the current AWS Region.

Naming constraints for ledger names are defined in Quotas in Amazon QLDB in the *Amazon QLDB Developer Guide*.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

PermissionsMode (p. 504)

The permissions mode to assign to the ledger that you want to create.

Type: String

Valid Values: `ALLOW_ALL`

Required: Yes

Tags (p. 504)

The key-value pairs to add as tags to the ledger that you want to create. Tag keys are case sensitive. Tag values are case sensitive and can be null.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "Arn": "string",
   "CreationDateTime": number,
   "DeletionProtection": boolean,
   "Name": "string",
   "State": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Arn (p. 505)

The Amazon Resource Name (ARN) for the ledger.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

CreationDateTime (p. 505)

The date and time, in epoch time format, when the ledger was created. (Epoch time format is the number of seconds elapsed since 12:00:00 AM January 1, 1970 UTC.)

Type: Timestamp

DeletionProtection (p. 505)

The flag that prevents a ledger from being deleted by any user. If not provided on ledger creation, this feature is enabled (`true`) by default.

If deletion protection is enabled, you must first disable it before you can delete the ledger using the QLDB API or the AWS Command Line Interface (AWS CLI). You can disable it by calling the

`UpdateLedger` operation to set the flag to `false`. The QLDB console disables deletion protection for you when you use it to delete a ledger.

Type: Boolean

Name (p. 505)

The name of the ledger.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

State (p. 505)

The current status of the ledger.

Type: String

Valid Values: `CREATING | ACTIVE | DELETING | DELETED`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

LimitExceededException

You have reached the limit on the maximum number of resources allowed.

HTTP Status Code: 400

ResourceAlreadyExistsException

The specified resource already exists.

HTTP Status Code: 409

ResourceInUseException

The specified resource can't be modified at this time.

HTTP Status Code: 409

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript

- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteLedger

Service: Amazon QLDB

Deletes a ledger and all of its contents. This action is irreversible.

If deletion protection is enabled, you must first disable it before you can delete the ledger using the QLDB API or the AWS Command Line Interface (AWS CLI). You can disable it by calling the `UpdateLedger` operation to set the flag to `false`. The QLDB console disables deletion protection for you when you use it to delete a ledger.

Request Syntax

```
DELETE /ledgers/name HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

name (p. 508)

The name of the ledger that you want to delete.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceInUseException

The specified resource can't be modified at this time.

HTTP Status Code: 409

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeJournalKinesisStream

Service: Amazon QLDB

Returns detailed information about a given Amazon QLDB journal stream. The output includes the Amazon Resource Name (ARN), stream name, current status, creation time, and the parameters of the original stream creation request.

This action does not return any expired journal streams. For more information, see Expiration for terminal streams in the *Amazon QLDB Developer Guide*.

Request Syntax

```
GET /ledgers/name/journal-kinesis-streams/streamId HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

name (p. 510)

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

streamId (p. 510)

The UUID (represented in Base62-encoded text) of the QLDB journal stream to describe.

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

Request Body

The request does not have a request body.

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"
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Stream (p. 510)

Information about the QLDB journal stream returned by a `DescribeJournalS3Export` request.

Type: JournalKinesisStreamDescription (p. 557) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeJournalS3Export

Service: Amazon QLDB

Returns information about a journal export job, including the ledger name, export ID, creation time, current status, and the parameters of the original export creation request.

This action does not return any expired export jobs. For more information, see Export job expiration in the *Amazon QLDB Developer Guide*.

If the export job with the given `ExportId` doesn't exist, then throws `ResourceNotFoundException`.

If the ledger with the given `Name` doesn't exist, then throws `ResourceNotFoundException`.

Request Syntax

```
GET /ledgers/name/journal-s3-exports/exportId HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

exportId (p. 512)

The UUID (represented in Base62-encoded text) of the journal export job to describe.

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

name (p. 512)

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "ExportDescription": {
      "ExclusiveEndTime": number,
      "ExportCreationTime": number,
      "ExportId": "string",
      "InclusiveStartTime": number,
      "LedgerName": "string",
      "RoleArn": "string",
      "S3ExportConfiguration": {
         "Bucket": "string",
         "EncryptionConfiguration": {
            "KmsKeyArn": "string",
            "ObjectEncryptionType": "string"
         },
         "Prefix": "string"
      },
      "Status": "string"
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ExportDescription (p. 512)

Information about the journal export job returned by a `DescribeJournalS3Export` request.

Type: JournalS3ExportDescription (p. 560) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DescribeLedger

Service: Amazon QLDB

Returns information about a ledger, including its state and when it was created.

Request Syntax

```
GET /ledgers/name HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

name (p. 514)

The name of the ledger that you want to describe.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "Arn": "string",
   "CreationDateTime": number,
   "DeletionProtection": boolean,
   "Name": "string",
   "State": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Arn (p. 514)

The Amazon Resource Name (ARN) for the ledger.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

CreationDateTime (p. 514)

The date and time, in epoch time format, when the ledger was created. (Epoch time format is the number of seconds elapsed since 12:00:00 AM January 1, 1970 UTC.)

Type: Timestamp

DeletionProtection (p. 514)

The flag that prevents a ledger from being deleted by any user. If not provided on ledger creation, this feature is enabled (`true`) by default.

If deletion protection is enabled, you must first disable it before you can delete the ledger using the QLDB API or the AWS Command Line Interface (AWS CLI). You can disable it by calling the `UpdateLedger` operation to set the flag to `false`. The QLDB console disables deletion protection for you when you use it to delete a ledger.

Type: Boolean

Name (p. 514)

The name of the ledger.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

State (p. 514)

The current status of the ledger.

Type: String

Valid Values: `CREATING | ACTIVE | DELETING | DELETED`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python

- AWS SDK for Ruby V3

ExportJournalToS3

Service: Amazon QLDB

Exports journal contents within a date and time range from a ledger into a specified Amazon Simple Storage Service (Amazon S3) bucket. The data is written as files in Amazon Ion format.

If the ledger with the given `Name` doesn't exist, then throws `ResourceNotFoundException`.

If the ledger with the given `Name` is in `CREATING` status, then throws `ResourcePreconditionNotMetException`.

You can initiate up to two concurrent journal export requests for each ledger. Beyond this limit, journal export requests throw `LimitExceededException`.

Request Syntax

```
POST /ledgers/name/journal-s3-exports HTTP/1.1
Content-type: application/json

{
   "ExclusiveEndTime": number,
   "InclusiveStartTime": number,
   "RoleArn": "string",
   "S3ExportConfiguration": {
      "Bucket": "string",
      "EncryptionConfiguration": {
         "KmsKeyArn": "string",
         "ObjectEncryptionType": "string"
      },
      "Prefix": "string"
   }
}
```

URI Request Parameters

The request uses the following URI parameters.

name (p. 517)

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request accepts the following data in JSON format.

ExclusiveEndTime (p. 517)

The exclusive end date and time for the range of journal contents to export.

The `ExclusiveEndTime` must be in `ISO 8601` date and time format and in Universal Coordinated Time (UTC). For example: `2019-06-13T21:36:34Z`.

The `ExclusiveEndTime` must be less than or equal to the current UTC date and time.

Type: Timestamp

Required: Yes

InclusiveStartTime (p. 517)

The inclusive start date and time for the range of journal contents to export.

The `InclusiveStartTime` must be in `ISO 8601` date and time format and in Universal Coordinated Time (UTC). For example: `2019-06-13T21:36:34Z`.

The `InclusiveStartTime` must be before `ExclusiveEndTime`.

If you provide an `InclusiveStartTime` that is before the ledger's `CreationDateTime`, Amazon QLDB defaults it to the ledger's `CreationDateTime`.

Type: Timestamp

Required: Yes

RoleArn (p. 517)

The Amazon Resource Name (ARN) of the IAM role that grants QLDB permissions for a journal export job to do the following:

- Write objects into your Amazon Simple Storage Service (Amazon S3) bucket.
- (Optional) Use your customer master key (CMK) in AWS Key Management Service (AWS KMS) for server-side encryption of your exported data.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

S3ExportConfiguration (p. 517)

The configuration settings of the Amazon S3 bucket destination for your export request.

Type: S3ExportConfiguration (p. 565) object

Required: Yes

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "ExportId": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ExportId (p. 518)

The UUID (represented in Base62-encoded text) that QLDB assigns to each journal export job.

To describe your export request and check the status of the job, you can use `ExportId` to call `DescribeJournalS3Export`.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

GetBlock

Service: Amazon QLDB

Returns a block object at a specified address in a journal. Also returns a proof of the specified block for verification if `DigestTipAddress` is provided.

For information about the data contents in a block, see Journal contents in the *Amazon QLDB Developer Guide*.

If the specified ledger doesn't exist or is in `DELETING` status, then throws `ResourceNotFoundException`.

If the specified ledger is in `CREATING` status, then throws `ResourcePreconditionNotMetException`.

If no block exists with the specified address, then throws `InvalidParameterException`.

Request Syntax

```
POST /ledgers/name/block HTTP/1.1
Content-type: application/json

{
   "BlockAddress": {
      "IonText": "string"
   },
   "DigestTipAddress": {
      "IonText": "string"
   }
}
```

URI Request Parameters

The request uses the following URI parameters.

name (p. 520)

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request accepts the following data in JSON format.

BlockAddress (p. 520)

The location of the block that you want to request. An address is an Amazon Ion structure that has two fields: `strandId` and `sequenceNo`.

For example: `{strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:14}`.

Type: ValueHolder (p. 566) object

Required: Yes

DigestTipAddress (p. 520)

The latest block location covered by the digest for which to request a proof. An address is an Amazon Ion structure that has two fields: `strandId` and `sequenceNo`.

For example: `{strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:49}`.

Type: ValueHolder (p. 566) object

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "Block": {
      "IonText": "string"
   },
   "Proof": {
      "IonText": "string"
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Block (p. 521)

The block data object in Amazon Ion format.

Type: ValueHolder (p. 566) object

Proof (p. 521)

The proof object in Amazon Ion format returned by a `GetBlock` request. A proof contains the list of hash values required to recalculate the specified digest using a Merkle tree, starting with the specified block.

Type: ValueHolder (p. 566) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

GetDigest

Service: Amazon QLDB

Returns the digest of a ledger at the latest committed block in the journal. The response includes a 256-bit hash value and a block address.

Request Syntax

```
POST /ledgers/name/digest HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

name (p. 523)

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "Digest": blob,
   "DigestTipAddress": {
      "IonText": "string"
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Digest (p. 523)

The 256-bit hash value representing the digest returned by a `GetDigest` request.

Type: Base64-encoded binary data object

Length Constraints: Fixed length of 32.

DigestTipAddress (p. 523)

The latest block location covered by the digest that you requested. An address is an Amazon Ion structure that has two fields: `strandId` and `sequenceNo`.

Type: ValueHolder (p. 566) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

GetRevision

Service: Amazon QLDB

Returns a revision data object for a specified document ID and block address. Also returns a proof of the specified revision for verification if `DigestTipAddress` is provided.

Request Syntax

```
POST /ledgers/name/revision HTTP/1.1
Content-type: application/json

{
   "BlockAddress": {
      "IonText": "string"
   },
   "DigestTipAddress": {
      "IonText": "string"
   },
   "DocumentId": "string"
}
```

URI Request Parameters

The request uses the following URI parameters.

name (p. 525)

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request accepts the following data in JSON format.

BlockAddress (p. 525)

The block location of the document revision to be verified. An address is an Amazon Ion structure that has two fields: `strandId` and `sequenceNo`.

For example: `{strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:14}`.

Type: ValueHolder (p. 566) object

Required: Yes

DigestTipAddress (p. 525)

The latest block location covered by the digest for which to request a proof. An address is an Amazon Ion structure that has two fields: `strandId` and `sequenceNo`.

For example: `{strandId:"BlFTjlSXze9BIh1KOszcE3",sequenceNo:49}`.

Type: ValueHolder (p. 566) object

Required: No

DocumentId (p. 525)

The UUID (represented in Base62-encoded text) of the document to be verified.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "Proof": { 
      "IonText": "string"
   },
   "Revision": { 
      "IonText": "string"
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Proof (p. 526)

The proof object in Amazon Ion format returned by a `GetRevision` request. A proof contains the list of hash values that are required to recalculate the specified digest using a Merkle tree, starting with the specified document revision.

Type: ValueHolder (p. 566) object

Revision (p. 526)

The document revision data object in Amazon Ion format.

Type: ValueHolder (p. 566) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListJournalKinesisStreamsForLedger

Service: Amazon QLDB

Returns an array of all Amazon QLDB journal stream descriptors for a given ledger. The output of each stream descriptor includes the same details that are returned by `DescribeJournalKinesisStream`.

This action does not return any expired journal streams. For more information, see Expiration for terminal streams in the *Amazon QLDB Developer Guide*.

This action returns a maximum of `MaxResults` items. It is paginated so that you can retrieve all the items by calling `ListJournalKinesisStreamsForLedger` multiple times.

Request Syntax

```
GET /ledgers/name/journal-kinesis-streams?max_results=MaxResults&next_token=NextToken
 HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

name (p. 528)

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

MaxResults (p. 528)

The maximum number of results to return in a single `ListJournalKinesisStreamsForLedger` request. (The actual number of results returned might be fewer.)

Valid Range: Minimum value of 1. Maximum value of 100.

NextToken (p. 528)

A pagination token, indicating that you want to retrieve the next page of results. If you received a value for `NextToken` in the response from a previous `ListJournalKinesisStreamsForLedger` call, you should use that value as input here.

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Request Body

The request does not have a request body.

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"
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 528)

- If `NextToken` is empty, the last page of results has been processed and there are no more results to be retrieved.
- If `NextToken` is *not* empty, more results are available. To retrieve the next page of results, use the value of `NextToken` in a subsequent `ListJournalKinesisStreamsForLedger` call.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Streams (p. 528)

The array of QLDB journal stream descriptors that are associated with the given ledger.

Type: Array of JournalKinesisStreamDescription (p. 557) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListJournalS3Exports

Service: Amazon QLDB

Returns an array of journal export job descriptions for all ledgers that are associated with the current AWS account and Region.

This action returns a maximum of `MaxResults` items, and is paginated so that you can retrieve all the items by calling `ListJournalS3Exports` multiple times.

This action does not return any expired export jobs. For more information, see Export job expiration in the *Amazon QLDB Developer Guide*.

Request Syntax

```
GET /journal-s3-exports?max_results=MaxResults&next_token=NextToken HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

MaxResults (p. 531)

The maximum number of results to return in a single `ListJournalS3Exports` request. (The actual number of results returned might be fewer.)

Valid Range: Minimum value of 1. Maximum value of 100.

NextToken (p. 531)

A pagination token, indicating that you want to retrieve the next page of results. If you received a value for `NextToken` in the response from a previous `ListJournalS3Exports` call, then you should use that value as input here.

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "JournalS3Exports": [
      {
         "ExclusiveEndTime": number,
         "ExportCreationTime": number,
         "ExportId": "string",
         "InclusiveStartTime": number,
         "LedgerName": "string",
         "RoleArn": "string",
         "S3ExportConfiguration": {
            "Bucket": "string",
            "EncryptionConfiguration": {
               "KmsKeyArn": "string",
                "ObjectEncryptionType": "string"
            },
            "Prefix": "string"
         },
         "Status": "string"
      }
   ],
   "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

JournalS3Exports (p. 531)

The array of journal export job descriptions for all ledgers that are associated with the current AWS account and Region.

Type: Array of JournalS3ExportDescription (p. 560) objects

NextToken (p. 531)

- If `NextToken` is empty, then the last page of results has been processed and there are no more results to be retrieved.
- If `NextToken` is *not* empty, then there are more results available. To retrieve the next page of results, use the value of `NextToken` in a subsequent `ListJournalS3Exports` call.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListJournalS3ExportsForLedger

Service: Amazon QLDB

Returns an array of journal export job descriptions for a specified ledger.

This action returns a maximum of `MaxResults` items, and is paginated so that you can retrieve all the items by calling `ListJournalS3ExportsForLedger` multiple times.

This action does not return any expired export jobs. For more information, see Export job expiration in the *Amazon QLDB Developer Guide*.

Request Syntax

```
GET /ledgers/name/journal-s3-exports?max_results=MaxResults&next_token=NextToken HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

MaxResults (p. 533)

The maximum number of results to return in a single `ListJournalS3ExportsForLedger` request. (The actual number of results returned might be fewer.)

Valid Range: Minimum value of 1. Maximum value of 100.

name (p. 533)

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

NextToken (p. 533)

A pagination token, indicating that you want to retrieve the next page of results. If you received a value for `NextToken` in the response from a previous `ListJournalS3ExportsForLedger` call, then you should use that value as input here.

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "JournalS3Exports": [
      {
         "ExclusiveEndTime": number,
         "ExportCreationTime": number,
         "ExportId": "string",
         "InclusiveStartTime": number,
         "LedgerName": "string",
         "RoleArn": "string",
         "S3ExportConfiguration": {
            "Bucket": "string",
            "EncryptionConfiguration": {
               "KmsKeyArn": "string",
               "ObjectEncryptionType": "string"
            },
            "Prefix": "string"
         },
         "Status": "string"
      }
   ],
   "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

JournalS3Exports (p. 533)

The array of journal export job descriptions that are associated with the specified ledger.

Type: Array of JournalS3ExportDescription (p. 560) objects

NextToken (p. 533)

- If `NextToken` is empty, then the last page of results has been processed and there are no more results to be retrieved.
- If `NextToken` is *not* empty, then there are more results available. To retrieve the next page of results, use the value of `NextToken` in a subsequent `ListJournalS3ExportsForLedger` call.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3

- AWS SDK for Python
- AWS SDK for Ruby V3

ListLedgers

Service: Amazon QLDB

Returns an array of ledger summaries that are associated with the current AWS account and Region.

This action returns a maximum of 100 items and is paginated so that you can retrieve all the items by calling `ListLedgers` multiple times.

Request Syntax

```
GET /ledgers?max_results=MaxResults&next_token=NextToken HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

MaxResults (p. 536)

The maximum number of results to return in a single `ListLedgers` request. (The actual number of results returned might be fewer.)

Valid Range: Minimum value of 1. Maximum value of 100.

NextToken (p. 536)

A pagination token, indicating that you want to retrieve the next page of results. If you received a value for `NextToken` in the response from a previous `ListLedgers` call, then you should use that value as input here.

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "Ledgers": [
      {
         "CreationDateTime": number,
         "Name": "string",
         "State": "string"
      }
   ],
   "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Ledgers (p. 536)

The array of ledger summaries that are associated with the current AWS account and Region.

Type: Array of LedgerSummary (p. 563) objects

NextToken (p. 536)

A pagination token, indicating whether there are more results available:

- If `NextToken` is empty, then the last page of results has been processed and there are no more results to be retrieved.
- If `NextToken` is *not* empty, then there are more results available. To retrieve the next page of results, use the value of `NextToken` in a subsequent `ListLedgers` call.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListTagsForResource

Service: Amazon QLDB

Returns all tags for a specified Amazon QLDB resource.

Request Syntax

```
GET /tags/resourceArn HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

resourceArn (p. 538)

The Amazon Resource Name (ARN) for which to list the tags. For example:

`arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger`

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "Tags": {
      "string" : "string"
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Tags (p. 538)

The tags that are currently associated with the specified Amazon QLDB resource.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

StreamJournalToKinesis

Service: Amazon QLDB

Creates a journal stream for a given Amazon QLDB ledger. The stream captures every document revision that is committed to the ledger's journal and delivers the data to a specified Amazon Kinesis Data Streams resource.

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 Request Parameters

The request uses the following URI parameters.

name (p. 540)

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request accepts the following data in JSON format.

ExclusiveEndTime (p. 540)

The exclusive date and time that specifies when the stream ends. If you don't define this parameter, the stream runs indefinitely until you cancel it.

The `ExclusiveEndTime` must be in `ISO 8601` date and time format and in Universal Coordinated Time (UTC). For example: `2019-06-13T21:36:34Z`.

Type: Timestamp

Required: No

InclusiveStartTime (p. 540)

The inclusive start date and time from which to start streaming journal data. This parameter must be in `ISO 8601` date and time format and in Universal Coordinated Time (UTC). For example: `2019-06-13T21:36:34Z`.

The `InclusiveStartTime` cannot be in the future and must be before `ExclusiveEndTime`.

If you provide an `InclusiveStartTime` that is before the ledger's `CreationDateTime`, QLDB effectively defaults it to the ledger's `CreationDateTime`.

Type: Timestamp

Required: Yes

KinesisConfiguration (p. 540)

The configuration settings of the Kinesis Data Streams destination for your stream request.

Type: KinesisConfiguration (p. 562) object

Required: Yes

RoleArn (p. 540)

The Amazon Resource Name (ARN) of the IAM role that grants QLDB permissions for a journal stream to write data records to a Kinesis Data Streams resource.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

StreamName (p. 540)

The name that you want to assign to the QLDB journal stream. User-defined names can help identify and indicate the purpose of a stream.

Your stream name must be unique among other *active* streams for a given ledger. Stream names have the same naming constraints as ledger names, as defined in Quotas in Amazon QLDB in the *Amazon QLDB Developer Guide*.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Tags (p. 540)

The key-value pairs to add as tags to the stream that you want to create. Tag keys are case sensitive. Tag values are case sensitive and can be null.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

Response Syntax

```
HTTP/1.1 200
```

```
Content-type: application/json

{
   "StreamId": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

StreamId (p. 541)

The UUID (represented in Base62-encoded text) that QLDB assigns to each QLDB journal stream.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

ResourcePreconditionNotMetException

The operation failed because a condition wasn't satisfied in advance.

HTTP Status Code: 412

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

TagResource

Service: Amazon QLDB

Adds one or more tags to a specified Amazon QLDB resource.

A resource can have up to 50 tags. If you try to create more than 50 tags for a resource, your request fails and returns an error.

Request Syntax

```
POST /tags/resourceArn HTTP/1.1
Content-type: application/json

{
   "Tags": {
      "string" : "string"
   }
}
```

URI Request Parameters

The request uses the following URI parameters.

resourceArn (p. 544)

The Amazon Resource Name (ARN) to which you want to add the tags. For example:

`arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger`

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

Request Body

The request accepts the following data in JSON format.

Tags (p. 544)

The key-value pairs to add as tags to the specified QLDB resource. Tag keys are case sensitive. If you specify a key that already exists for the resource, your request fails and returns an error. Tag values are case sensitive and can be null.

Type: String to string map

Map Entries: Minimum number of 0 items. Maximum number of 200 items.

Key Length Constraints: Minimum length of 1. Maximum length of 128.

Value Length Constraints: Minimum length of 0. Maximum length of 256.

Required: Yes

Response Syntax

```
HTTP/1.1 200
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UntagResource

Service: Amazon QLDB

Removes one or more tags from a specified Amazon QLDB resource. You can specify up to 50 tag keys to remove.

Request Syntax

```
DELETE /tags/resourceArn?tagKeys=TagKeys HTTP/1.1
```

URI Request Parameters

The request uses the following URI parameters.

resourceArn (p. 546)

The Amazon Resource Name (ARN) from which to remove the tags. For example:

`arn:aws:qldb:us-east-1:123456789012:ledger/exampleLedger`

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

TagKeys (p. 546)

The list of tag keys to remove.

Array Members: Minimum number of 0 items. Maximum number of 200 items.

Length Constraints: Minimum length of 1. Maximum length of 128.

Required: Yes

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UpdateLedger

Service: Amazon QLDB

Updates properties on a ledger.

Request Syntax

```
PATCH /ledgers/name HTTP/1.1
Content-type: application/json

{
   "DeletionProtection": boolean
}
```

URI Request Parameters

The request uses the following URI parameters.

name (p. 548)

The name of the ledger.

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

Request Body

The request accepts the following data in JSON format.

DeletionProtection (p. 548)

The flag that prevents a ledger from being deleted by any user. If not provided on ledger creation, this feature is enabled (`true`) by default.

If deletion protection is enabled, you must first disable it before you can delete the ledger using the QLDB API or the AWS Command Line Interface (AWS CLI). You can disable it by calling the `UpdateLedger` operation to set the flag to `false`. The QLDB console disables deletion protection for you when you use it to delete a ledger.

Type: Boolean

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
   "Arn": "string",
   "CreationDateTime": number,
   "DeletionProtection": boolean,
   "Name": "string",
   "State": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Arn (p. 548)

The Amazon Resource Name (ARN) for the ledger.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

CreationDateTime (p. 548)

The date and time, in epoch time format, when the ledger was created. (Epoch time format is the number of seconds elapsed since 12:00:00 AM January 1, 1970 UTC.)

Type: Timestamp

DeletionProtection (p. 548)

The flag that prevents a ledger from being deleted by any user. If not provided on ledger creation, this feature is enabled (`true`) by default.

If deletion protection is enabled, you must first disable it before you can delete the ledger using the QLDB API or the AWS Command Line Interface (AWS CLI). You can disable it by calling the `UpdateLedger` operation to set the flag to `false`. The QLDB console disables deletion protection for you when you use it to delete a ledger.

Type: Boolean

Name (p. 548)

The name of the ledger.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

State (p. 548)

The current status of the ledger.

Type: String

Valid Values: `CREATING | ACTIVE | DELETING | DELETED`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

InvalidParameterException

One or more parameters in the request aren't valid.

HTTP Status Code: 400

ResourceNotFoundException

The specified resource doesn't exist.

HTTP Status Code: 404

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

Amazon QLDB Session

The following actions are supported by Amazon QLDB Session:

- SendCommand (p. 551)

SendCommand

Service: Amazon QLDB Session

Sends a command to an Amazon QLDB ledger.

> **Note**
> Instead of interacting directly with this API, we recommend that you use the QLDB Driver or the QLDB Shell to execute data transactions on a ledger.
>
> - If you are working with an AWS SDK, use the QLDB Driver. The driver provides a high-level abstraction layer above this *QLDB Session* data plane and manages `SendCommand` API calls for you. For information and a list of supported programming languages, see Getting started with the driver in the *Amazon QLDB Developer Guide*.
> - If you are working with the AWS Command Line Interface (AWS CLI), use the QLDB Shell. The shell is a command line interface that uses the QLDB Driver to interact with a ledger. For information, see Accessing Amazon QLDB using the QLDB Shell.

Request Syntax

```
{
   "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": {
   }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 584).

The request accepts the following data in JSON format.

AbortTransaction (p. 551)

Command to abort the current transaction.

Type: AbortTransactionRequest (p. 567) object

Required: No

CommitTransaction (p. 551)

Command to commit the specified transaction.

Type: CommitTransactionRequest (p. 569) object

Required: No

EndSession (p. 551)

Command to end the current session.

Type: EndSessionRequest (p. 571) object

Required: No

ExecuteStatement (p. 551)

Command to execute a statement in the specified transaction.

Type: ExecuteStatementRequest (p. 573) object

Required: No

FetchPage (p. 551)

Command to fetch a page.

Type: FetchPageRequest (p. 575) object

Required: No

SessionToken (p. 551)

Specifies the session token for the current command. A session token is constant throughout the life of the session.

To obtain a session token, run the `StartSession` command. This `SessionToken` is required for every subsequent command that is issued during the current session.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Required: No

StartSession (p. 551)

Command to start a new session. A session token is obtained as part of the response.

Type: StartSessionRequest (p. 578) object

Required: No

StartTransaction (p. 551)

Command to start a new transaction.

Type: StartTransactionRequest (p. 580) object

Required: No

Response Syntax

```
{
   "AbortTransaction": {
   },
   "CommitTransaction": {
      "CommitDigest": blob,
      "TransactionId": "string"
   },
   "EndSession": {
   },
   "ExecuteStatement": {
      "FirstPage": {
         "NextPageToken": "string",
         "Values": [
            {
               "IonBinary": blob,
               "IonText": "string"
            }
         ]
      }
   },
   "FetchPage": {
      "Page": {
         "NextPageToken": "string",
         "Values": [
            {
               "IonBinary": blob,
               "IonText": "string"
            }
         ]
      }
   },
   "StartSession": {
      "SessionToken": "string"
   },
   "StartTransaction": {
      "TransactionId": "string"
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AbortTransaction (p. 553)

Contains the details of the aborted transaction.

Type: AbortTransactionResult (p. 568) object

CommitTransaction (p. 553)

Contains the details of the committed transaction.

Type: CommitTransactionResult (p. 570) object

EndSession (p. 553)

Contains the details of the ended session.

Type: EndSessionResult (p. 572) object

ExecuteStatement (p. 553)

Contains the details of the executed statement.

Type: ExecuteStatementResult (p. 574) object

FetchPage (p. 553)

Contains the details of the fetched page.

Type: FetchPageResult (p. 576) object

StartSession (p. 553)

Contains the details of the started session that includes a session token. This `SessionToken` is required for every subsequent command that is issued during the current session.

Type: StartSessionResult (p. 579) object

StartTransaction (p. 553)

Contains the details of the started transaction.

Type: StartTransactionResult (p. 581) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 582).

BadRequestException

Returned if the request is malformed or contains an error such as an invalid parameter value or a missing required parameter.

HTTP Status Code: 400

InvalidSessionException

Returned if the session doesn't exist anymore because it timed out or expired.

HTTP Status Code: 400

LimitExceededException

Returned if a resource limit such as number of active sessions is exceeded.

HTTP Status Code: 400

OccConflictException

Returned when a transaction cannot be written to the journal due to a failure in the verification phase of *optimistic concurrency control* (OCC).

HTTP Status Code: 400

RateExceededException

Returned when the rate of requests exceeds the allowed throughput.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

Data Types

The following data types are supported by Amazon QLDB:

- JournalKinesisStreamDescription (p. 557)
- JournalS3ExportDescription (p. 560)
- KinesisConfiguration (p. 562)
- LedgerSummary (p. 563)
- S3EncryptionConfiguration (p. 564)
- S3ExportConfiguration (p. 565)
- ValueHolder (p. 566)

The following data types are supported by Amazon QLDB Session:

- AbortTransactionRequest (p. 567)
- AbortTransactionResult (p. 568)
- CommitTransactionRequest (p. 569)
- CommitTransactionResult (p. 570)
- EndSessionRequest (p. 571)
- EndSessionResult (p. 572)
- ExecuteStatementRequest (p. 573)
- ExecuteStatementResult (p. 574)
- FetchPageRequest (p. 575)
- FetchPageResult (p. 576)
- Page (p. 577)
- StartSessionRequest (p. 578)
- StartSessionResult (p. 579)
- StartTransactionRequest (p. 580)
- StartTransactionResult (p. 581)
- ValueHolder (p. 582)

Amazon QLDB

The following data types are supported by Amazon QLDB:

- JournalKinesisStreamDescription (p. 557)

- JournalS3ExportDescription (p. 560)
- KinesisConfiguration (p. 562)
- LedgerSummary (p. 563)
- S3EncryptionConfiguration (p. 564)
- S3ExportConfiguration (p. 565)
- ValueHolder (p. 566)

JournalKinesisStreamDescription

Service: Amazon QLDB

Information about an Amazon QLDB journal stream, including the Amazon Resource Name (ARN), stream name, creation time, current status, and the parameters of the original stream creation request.

Contents

Arn

The Amazon Resource Name (ARN) of the QLDB journal stream.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: No

CreationTime

The date and time, in epoch time format, when the QLDB journal stream was created. (Epoch time format is the number of seconds elapsed since 12:00:00 AM January 1, 1970 UTC.)

Type: Timestamp

Required: No

ErrorCause

The error message that describes the reason that a stream has a status of `IMPAIRED` or `FAILED`. This is not applicable to streams that have other status values.

Type: String

Valid Values: `KINESIS_STREAM_NOT_FOUND | IAM_PERMISSION_REVOKED`

Required: No

ExclusiveEndTime

The exclusive date and time that specifies when the stream ends. If this parameter is blank, the stream runs indefinitely until you cancel it.

Type: Timestamp

Required: No

InclusiveStartTime

The inclusive start date and time from which to start streaming journal data.

Type: Timestamp

Required: No

KinesisConfiguration

The configuration settings of the Amazon Kinesis Data Streams destination for a QLDB journal stream.

Type: KinesisConfiguration (p. 562) object

Required: Yes

LedgerName

The name of the ledger.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

RoleArn

The Amazon Resource Name (ARN) of the IAM role that grants QLDB permissions for a journal stream to write data records to a Kinesis Data Streams resource.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

Status

The current state of the QLDB journal stream.

Type: String

Valid Values: `ACTIVE | COMPLETED | CANCELED | FAILED | IMPAIRED`

Required: Yes

StreamId

The UUID (represented in Base62-encoded text) of the QLDB journal stream.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

StreamName

The user-defined name of the QLDB journal stream.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go

- AWS SDK for Java
- AWS SDK for Ruby V3

JournalS3ExportDescription

Service: Amazon QLDB

Information about a journal export job, including the ledger name, export ID, creation time, current status, and the parameters of the original export creation request.

Contents

ExclusiveEndTime

The exclusive end date and time for the range of journal contents that are specified in the original export request.

Type: Timestamp

Required: Yes

ExportCreationTime

The date and time, in epoch time format, when the export job was created. (Epoch time format is the number of seconds elapsed since 12:00:00 AM January 1, 1970 UTC.)

Type: Timestamp

Required: Yes

ExportId

The UUID (represented in Base62-encoded text) of the journal export job.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

InclusiveStartTime

The inclusive start date and time for the range of journal contents that are specified in the original export request.

Type: Timestamp

Required: Yes

LedgerName

The name of the ledger.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

RoleArn

The Amazon Resource Name (ARN) of the IAM role that grants QLDB permissions for a journal export job to do the following:

- Write objects into your Amazon Simple Storage Service (Amazon S3) bucket.

- (Optional) Use your customer master key (CMK) in AWS Key Management Service (AWS KMS) for server-side encryption of your exported data.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

S3ExportConfiguration

The Amazon Simple Storage Service (Amazon S3) bucket location in which a journal export job writes the journal contents.

Type: S3ExportConfiguration (p. 565) object

Required: Yes

Status

The current state of the journal export job.

Type: String

Valid Values: `IN_PROGRESS | COMPLETED | CANCELLED`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

KinesisConfiguration

Service: Amazon QLDB

The configuration settings of the Amazon Kinesis Data Streams destination for an Amazon QLDB journal stream.

Contents

AggregationEnabled

Enables QLDB to publish multiple data records in a single Kinesis Data Streams record. To learn more, see KPL Key Concepts in the *Amazon Kinesis Data Streams Developer Guide*.

Type: Boolean

Required: No

StreamArn

The Amazon Resource Name (ARN) of the Kinesis Data Streams resource.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

LedgerSummary

Service: Amazon QLDB

Information about a ledger, including its name, state, and when it was created.

Contents

CreationDateTime

The date and time, in epoch time format, when the ledger was created. (Epoch time format is the number of seconds elapsed since 12:00:00 AM January 1, 1970 UTC.)

Type: Timestamp

Required: No

Name

The name of the ledger.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: No

State

The current status of the ledger.

Type: String

Valid Values: `CREATING | ACTIVE | DELETING | DELETED`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

S3EncryptionConfiguration

Service: Amazon QLDB

The encryption settings that are used by a journal export job to write data in an Amazon Simple Storage Service (Amazon S3) bucket.

Contents

KmsKeyArn

The Amazon Resource Name (ARN) for a symmetric customer master key (CMK) in AWS Key Management Service (AWS KMS). Amazon QLDB does not support asymmetric CMKs.

You must provide a `KmsKeyArn` if you specify `SSE_KMS` as the `ObjectEncryptionType`.

`KmsKeyArn` is not required if you specify `SSE_S3` as the `ObjectEncryptionType`.

Type: String

Length Constraints: Minimum length of 20. Maximum length of 1600.

Required: No

ObjectEncryptionType

The Amazon S3 object encryption type.

To learn more about server-side encryption options in Amazon S3, see Protecting Data Using Server-Side Encryption in the *Amazon S3 Developer Guide*.

Type: String

Valid Values: `SSE_KMS | SSE_S3 | NO_ENCRYPTION`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

S3ExportConfiguration

Service: Amazon QLDB

The Amazon Simple Storage Service (Amazon S3) bucket location in which a journal export job writes the journal contents.

Contents

Bucket

The Amazon S3 bucket name in which a journal export job writes the journal contents.

The bucket name must comply with the Amazon S3 bucket naming conventions. For more information, see Bucket Restrictions and Limitations in the *Amazon S3 Developer Guide*.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 255.

Pattern: `^[A-Za-z-0-9-_.]+$`

Required: Yes

EncryptionConfiguration

The encryption settings that are used by a journal export job to write data in an Amazon S3 bucket.

Type: S3EncryptionConfiguration (p. 564) object

Required: Yes

Prefix

The prefix for the Amazon S3 bucket in which a journal export job writes the journal contents.

The prefix must comply with Amazon S3 key naming rules and restrictions. For more information, see Object Key and Metadata in the *Amazon S3 Developer Guide*.

The following are examples of valid `Prefix` values:

- `JournalExports-ForMyLedger/Testing/`
- `JournalExports`
- `My:Tests/`

Type: String

Length Constraints: Minimum length of 0. Maximum length of 128.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

ValueHolder

Service: Amazon QLDB

A structure that can contain a value in multiple encoding formats.

Contents

IonText

An Amazon Ion plaintext value contained in a `ValueHolder` structure.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1048576.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

Amazon QLDB Session

The following data types are supported by Amazon QLDB Session:

- AbortTransactionRequest (p. 567)
- AbortTransactionResult (p. 568)
- CommitTransactionRequest (p. 569)
- CommitTransactionResult (p. 570)
- EndSessionRequest (p. 571)
- EndSessionResult (p. 572)
- ExecuteStatementRequest (p. 573)
- ExecuteStatementResult (p. 574)
- FetchPageRequest (p. 575)
- FetchPageResult (p. 576)
- Page (p. 577)
- StartSessionRequest (p. 578)
- StartSessionResult (p. 579)
- StartTransactionRequest (p. 580)
- StartTransactionResult (p. 581)
- ValueHolder (p. 582)

AbortTransactionRequest

Service: Amazon QLDB Session

Contains the details of the transaction to abort.

Contents

The members of this structure are context-dependent.

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

AbortTransactionResult

Service: Amazon QLDB Session

Contains the details of the aborted transaction.

Contents

The members of this structure are context-dependent.

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

CommitTransactionRequest

Service: Amazon QLDB Session

Contains the details of the transaction to commit.

Contents

CommitDigest

Specifies the commit digest for the transaction to commit. For every active transaction, the commit digest must be passed. QLDB validates `CommitDigest` and rejects the commit with an error if the digest computed on the client does not match the digest computed by QLDB.

The purpose of the `CommitDigest` parameter is to ensure that QLDB commits a transaction if and only if the server has processed the exact set of statements sent by the client, in the same order that client sent them, and with no duplicates.

Type: Base64-encoded binary data object

Required: Yes

TransactionId

Specifies the transaction ID of the transaction to commit.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

CommitTransactionResult

Service: Amazon QLDB Session

Contains the details of the committed transaction.

Contents

CommitDigest

The commit digest of the committed transaction.

Type: Base64-encoded binary data object

Required: No

TransactionId

The transaction ID of the committed transaction.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

EndSessionRequest

Service: Amazon QLDB Session

Specifies a request to end the session.

Contents

The members of this structure are context-dependent.

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

EndSessionResult

Service: Amazon QLDB Session

Contains the details of the ended session.

Contents

The members of this structure are context-dependent.

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

ExecuteStatementRequest

Service: Amazon QLDB Session

Specifies a request to execute a statement.

Contents

Parameters

Specifies the parameters for the parameterized statement in the request.

Type: Array of ValueHolder (p. 582) objects

Required: No

Statement

Specifies the statement of the request.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100000.

Required: Yes

TransactionId

Specifies the transaction ID of the request.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

ExecuteStatementResult

Service: Amazon QLDB Session

Contains the details of the executed statement.

Contents

FirstPage

Contains the details of the first fetched page.

Type: Page (p. 577) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

FetchPageRequest

Service: Amazon QLDB Session

Specifies the details of the page to be fetched.

Contents

NextPageToken

Specifies the next page token of the page to be fetched.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Required: Yes

TransactionId

Specifies the transaction ID of the page to be fetched.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

FetchPageResult

Service: Amazon QLDB Session

Contains the page that was fetched.

Contents

Page

Contains details of the fetched page.

Type: Page (p. 577) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

Page

Service: Amazon QLDB Session

Contains details of the fetched page.

Contents

NextPageToken

The token of the next page.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Required: No

Values

A structure that contains values in multiple encoding formats.

Type: Array of ValueHolder (p. 582) objects

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

StartSessionRequest

Service: Amazon QLDB Session

Specifies a request to start a new session.

Contents

LedgerName

The name of the ledger to start a new session against.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 32.

Pattern: `(?!^.*--)(?!^[0-9]+$)(?!^-)(?!.*-$)^[A-Za-z0-9-]+$`

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

StartSessionResult

Service: Amazon QLDB Session

Contains the details of the started session.

Contents

SessionToken

Session token of the started session. This `SessionToken` is required for every subsequent command that is issued during the current session.

Type: String

Length Constraints: Minimum length of 4. Maximum length of 1024.

Pattern: `^[A-Za-z-0-9+/=]+$`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

StartTransactionRequest

Service: Amazon QLDB Session

Specifies a request to start a transaction.

Contents

The members of this structure are context-dependent.

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

StartTransactionResult

Service: Amazon QLDB Session

Contains the details of the started transaction.

Contents

TransactionId

The transaction ID of the started transaction.

Type: String

Length Constraints: Fixed length of 22.

Pattern: `^[A-Za-z-0-9]+$`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

ValueHolder

Service: Amazon QLDB Session

A structure that can contain a value in multiple encoding formats.

Contents

IonBinary

An Amazon Ion binary value contained in a `ValueHolder` structure.

Type: Base64-encoded binary data object

Length Constraints: Minimum length of 1. Maximum length of 131072.

Required: No

IonText

An Amazon Ion plaintext value contained in a `ValueHolder` structure.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1048576.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

Common Errors

This section lists the errors common to the API actions of all AWS services. For errors specific to an API action for this service, see the topic for that API action.

AccessDeniedException

You do not have sufficient access to perform this action.

HTTP Status Code: 400

IncompleteSignature

The request signature does not conform to AWS standards.

HTTP Status Code: 400

InternalFailure

The request processing has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

InvalidAction

The action or operation requested is invalid. Verify that the action is typed correctly.

HTTP Status Code: 400

InvalidClientTokenId

The X.509 certificate or AWS access key ID provided does not exist in our records.

HTTP Status Code: 403

InvalidParameterCombination

Parameters that must not be used together were used together.

HTTP Status Code: 400

InvalidParameterValue

An invalid or out-of-range value was supplied for the input parameter.

HTTP Status Code: 400

InvalidQueryParameter

The AWS query string is malformed or does not adhere to AWS standards.

HTTP Status Code: 400

MalformedQueryString

The query string contains a syntax error.

HTTP Status Code: 404

MissingAction

The request is missing an action or a required parameter.

HTTP Status Code: 400

MissingAuthenticationToken

The request must contain either a valid (registered) AWS access key ID or X.509 certificate.

HTTP Status Code: 403

MissingParameter

A required parameter for the specified action is not supplied.

HTTP Status Code: 400

OptInRequired

The AWS access key ID needs a subscription for the service.

HTTP Status Code: 403

RequestExpired

The request reached the service more than 15 minutes after the date stamp on the request or more than 15 minutes after the request expiration date (such as for pre-signed URLs), or the date stamp on the request is more than 15 minutes in the future.

HTTP Status Code: 400

ServiceUnavailable

The request has failed due to a temporary failure of the server.

HTTP Status Code: 503

ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationError

The input fails to satisfy the constraints specified by an AWS service.

HTTP Status Code: 400

Common Parameters

The following list contains the parameters that all actions use for signing Signature Version 4 requests with a query string. Any action-specific parameters are listed in the topic for that action. For more information about Signature Version 4, see Signature Version 4 Signing Process in the *Amazon Web Services General Reference*.

Action

The action to be performed.

Type: string

Required: Yes

Version

The API version that the request is written for, expressed in the format YYYY-MM-DD.

Type: string

Required: Yes

X-Amz-Algorithm

The hash algorithm that you used to create the request signature.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Valid Values: `AWS4-HMAC-SHA256`

Required: Conditional

X-Amz-Credential

The credential scope value, which is a string that includes your access key, the date, the region you are targeting, the service you are requesting, and a termination string ("aws4_request"). The value is expressed in the following format: *access_key*/*YYYYMMDD*/*region*/*service*/aws4_request.

For more information, see Task 2: Create a String to Sign for Signature Version 4 in the *Amazon Web Services General Reference*.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-Date

The date that is used to create the signature. The format must be ISO 8601 basic format (YYYYMMDD'T'HHMMSS'Z'). For example, the following date time is a valid X-Amz-Date value: `20120325T120000Z`.

Condition: X-Amz-Date is optional for all requests; it can be used to override the date used for signing requests. If the Date header is specified in the ISO 8601 basic format, X-Amz-Date is not required. When X-Amz-Date is used, it always overrides the value of the Date header. For more information, see Handling Dates in Signature Version 4 in the *Amazon Web Services General Reference*.

Type: string

Required: Conditional

X-Amz-Security-Token

The temporary security token that was obtained through a call to AWS Security Token Service (AWS STS). For a list of services that support temporary security credentials from AWS Security Token Service, go to AWS Services That Work with IAM in the *IAM User Guide*.

Condition: If you're using temporary security credentials from the AWS Security Token Service, you must include the security token.

Type: string

Required: Conditional

X-Amz-Signature

Specifies the hex-encoded signature that was calculated from the string to sign and the derived signing key.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-SignedHeaders

Specifies all the HTTP headers that were included as part of the canonical request. For more information about specifying signed headers, see Task 1: Create a Canonical Request For Signature Version 4 in the *Amazon Web Services General Reference*.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

DBQMS API reference

The Database Query Metadata Service (`dbqms`) provides your recent and saved queries for the *Query editor* on the AWS Management Console for Amazon QLDB.

The following DBQMS actions are supported:

Topics

CreateFavoriteQuery

Save a new favorite query. Each IAM user can create up to 1000 saved queries. This limit is subject to change in the future.

CreateQueryHistory

Save a new query history entry.

DeleteFavoriteQueries

Allows to delete multiple saved queries.

DeleteQueryHistory

Delete query history entries.

DescribeFavoriteQueries

List of saved queries created by an IAM user in a given account.

DescribeQueryHistory

List query history entries.

GetQueryString

Retrieve full query text from a query ID.

UpdateFavoriteQuery

Update the query string, description, name, or expiration date.

UpdateQueryHistory

Update the status of query history.

Quotas and limits in Amazon QLDB

This section describes the current quotas, also referred to as limits, in Amazon QLDB.

Topics

Default quotas

QLDB has the following default quotas, as also listed at Amazon QLDB endpoints and quotas in the *AWS General Reference*. These quotas are per AWS account per Region. To request a quota increase for your account in a Region, use the Service Quotas console.

Open the Service Quotas console at https://console.aws.amazon.com/servicequotas/.

Resource	Default quota
The maximum number of active ledgers that you can create in this account in the current Region	5
The maximum number of active journal exports to Amazon S3 per ledger	2
The maximum number of active journal streams to Kinesis Data Streams per ledger	5

Fixed quotas

In addition to default quotas, QLDB has the following fixed quotas per ledger. These quotas cannot be increased:

Resource	Fixed quota
Number of concurrent active sessions (p. 335)	1500
Number of active tables	20
Number of total tables (active and inactive) **Note** In QLDB, dropped tables (p. 329) are considered *inactive* and count against this total quota.	40

Resource	Fixed quota
Number of indexes per table	5
Number of documents in a transaction	40
Document size	128 KB
Transaction size	4 MB
Expiration period for completed journal export jobs	7 days
Expiration period for terminal journal streams	7 days

Transaction size

The maximum size for a transaction in QLDB is 4 MB. The size of a transaction is calculated based on the sum of the following factors.

Deltas

The document changes that are generated by *all the statements* within the transaction. In a transaction that impacts several documents, the total delta size is the sum of each affected document's individual delta.

Metadata

The QLDB-generated transaction metadata that is associated with each affected document.

Indexes

If an index is defined on a table that is affected by the transaction, the associated index entry also generates a delta.

History

Because all document revisions are persisted in QLDB, all transactions also append to the history.

Inserts—Every document inserted into a table also has a copy inserted into its history table. For example, a newly inserted 100 KB document generates a *minimum* of 200 KB of deltas in a transaction. (This is a rough estimate that does not include metadata or indexes.)

Updates—Any document update, even for a single field, creates a new revision of the entire document in history, plus or minus the delta of the update. This means that a small update in a large document would still generate a large transaction delta. For example, adding 2 KB of data in an existing 100 KB document creates a new 102 KB revision in history. This adds up to at least 104 KB of total deltas in a transaction. (Again, this estimate does not include metadata or indexes.)

Deletes—Similar to updates, any delete transaction creates a new document revision in the history. However, the newly created `DELETE` revision is smaller than the original document because it has null user data and only contains metadata.

Naming constraints

The following table describes naming constraints within Amazon QLDB.

Ledger name Journal stream name	• Must only contain 1–32 alphanumeric characters or hyphens. • Must not be all numbers. • Can't contain two consecutive hyphens. • The first and last characters must be a letter or number. • The ledger names are case sensitive.
Table name	• Must only contain 1–128 alphanumeric characters or underscores. • The first character must be a letter or an underscore. • The remaining characters can be any combination of alphanumeric characters and underscores. • The table names are case sensitive. • Must not be a QLDB PartiQL reserved word (p. 483).

Amazon QLDB related information

The following related resources can help you as you work with this service.

Topics

Technical documentation

- **Amazon Ion** – Developer guides, user guides, and references for the Amazon Ion data format.
- **PartiQL** – A specification document and general tutorials for the PartiQL query language.
- **QLDB Workshops** – A collection of workshops that provides practical, hands-on examples of using QLDB to build system-of-record applications, including the following labs:
  - Learning QLDB fundamentals.
  - Working with Amazon Ion, and converting Ion to and from JSON (Java).
  - Using AWS Glue and Amazon Athena to enable QLDB data for a data lake.

GitHub repositories

Drivers

- **.NET driver** – The .NET implementation of the QLDB driver.
- **Go driver** – The Go implementation of the QLDB driver.
- **Java driver** – The Java implementation of the QLDB driver.
- **Node.js driver** – The Node.js implementation of the QLDB driver.
- **Python driver** – The Python implementation of the QLDB driver.

Sample applications

- **Java DMV application** – A tutorial application—based on a department of motor vehicles (DMV) use case—that demonstrates basic operations and best practices for using QLDB and the QLDB driver for Java.
- **Node.js DMV application** – A DMV-based tutorial application that demonstrates basic operations and best practices for using QLDB and the QLDB driver for Node.js.
- **Python DMV application** – A DMV-based tutorial application that demonstrates basic operations and best practices for using QLDB and the QLDB driver for Python.
- **Double entry application** – A Java sample application that demonstrates how to model a double entry financial ledger application using QLDB.

- **QLDB integration with AWS AppSync** – A Java and Python sample application that integrates the DMV ledger data with AWS AppSync to build a GraphQL interface to QLDB.
- **QLDB stream integration with Elasticsearch** – A Python sample application that implements an AWS Lambda function to stream data from QLDB to Amazon Elasticsearch Service (Amazon ES).
- **QLDB stream integration with Amazon SNS and Amazon SQS** – A Python sample application that implements an AWS Lambda function to stream data from QLDB to an Amazon SNS topic, which has a Amazon SQS queue subscribed to it.

Amazon Ion and PartiQL

- **Amazon Ion libraries** – Ion team supported libraries, tools, and documentation.
- **PartiQL implementation** – The implementation, specification, and tutorials for PartiQL.

AWS blog posts and articles

- **Building a GraphQL interface to Amazon QLDB with AWS AppSync: Part 1** – *(May 4, 2020)* Part one of a two-post series that explores a reusable approach for integrating QLDB and AWS AppSync to power a government use case example.
- **Building a GraphQL interface to Amazon QLDB with AWS AppSync: Part 2** – *(May 4, 2020)* Part two of a two-post series that discusses how to integrate QLDB and AWS AppSync to provide a versatile, GraphQL-powered API on top of a QLDB ledger.
- **MediSci's Platform Leverages Amazon QLDB to Bridge the Gap Between Healthcare and Life Science Product Development** – *(May 27, 2020)* An article that explains how MediSci is leveraging QLDB to bring together the frontline healthcare and life science product development industries to enable better patient outcomes.
- **How I streamed data from Amazon QLDB to DynamoDB using Nodejs in near-real time** – *(July 7, 2020)* An AWS Heroes post that describes how to stream data from QLDB to DynamoDB to support single-digit latency and infinitely scalable key-value inquiries.
- **Streaming data from Amazon QLDB to Elasticsearch** – *(August 19, 2020)* An AWS Heroes post that describes how to stream data from QLDB to Amazon Elasticsearch Service (Amazon ES) to support rich text search and downstream analytics, such as aggregation or metrics across records.

Media

AWS videos

- **AWS Tech Talk: A Customer's Perspective on Building an Event-Triggered System-of-Record Application with Amazon QLDB** – *(March 31, 2020; 29 minutes)* A tech talk by Matt Lewis—AWS Data Hero and Chief Architect of the Driver and Vehicle Licensing Agency (DVLA) in the UK—that explains DVLA's use case for QLDB and their application's event-driven architecture.
- **AWS re:Invent 2019: Why you need a ledger database: BMW, DVLA, & Sage discuss use cases** – *(December 5, 2019; 47 minutes)* A presentation by customers BMW, the UK government organization DVLA, and Sage that discusses why to use a ledger database and shares their use cases for QLDB.
- **AWS re:Invent 2019: Amazon QLDB: An engineer's deep dive on why this is a game changer** – *(December 5, 2019; 50 minutes)* A presentation by Andrew Certain (AWS Distinguished Engineer) that discusses QLDB's unique journal-first architecture along with its various innovations—including cryptographic hashing, Merkle trees, multiple Availability Zone replication, and PartiQL support.
- **Building Applications with Amazon QLDB, a First-of-Its-Kind Ledger Database - AWS Online Tech Talks** – *(November 19, 2019; 51 minutes)* A deep dive tech talk that describes the unique features of

QLDB and specifics about how to use the core functionality—including querying your data's complete history, cryptographically verifying documents, and designing a data model.

- **Building System of Record Applications with Amazon QLDB - AWS Online Tech Talks** – *(July 24, 2019; 43 minutes)* A tech talk that provides an introduction to QLDB, a demo of how QLDB works, and use cases for which you can use the service.
- **What is Amazon Quantum Ledger Database (Amazon QLDB)** – *(January 30, 2019; 6 minutes)* A brief overview and demonstration of QLDB.

Podcasts

- **Why are customers choosing Amazon QLDB?** – *(July 5, 2020; 33 minutes)* A discussion that explains the definition of a *ledger database*, how it's different from a blockchain, and how customers are using it today.

General AWS resources

- **Classes & Workshops** – Links to role-based and specialty courses as well as self-paced labs to help sharpen your AWS skills and gain practical experience.
- **AWS Developer Tools** – Links to developer tools, SDKs, IDE toolkits, and command line tools for developing and managing AWS applications.
- **AWS Whitepapers** – Links to a comprehensive list of technical AWS whitepapers, covering topics such as architecture, security, and economics and authored by AWS Solutions Architects or other technical experts.
- **AWS Support Center** – The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
- **AWS Support** – The primary web page for information about AWS Support, a one-on-one, fast-response support channel to help you build and run applications in the cloud.
- **Contact Us** – A central contact point for inquiries concerning AWS billing, account, events, abuse, and other issues.
- **AWS Site Terms** – Detailed information about our copyright and trademark; your account, license, and site access; and other topics.

Release history for Amazon QLDB

The following table describes important changes in each release of Amazon QLDB and the corresponding updates in the *Amazon QLDB Developer Guide*. For notification about updates to this documentation, you can subscribe to the RSS feed.

- **API version:** 2019-01-02
- **Latest documentation update:** August 28, 2020

update-history-change	update-history-description	update-history-date
Java driver v2.0 (p. 593)	The Amazon QLDB driver for Java version 2.0 is now generally available. For release notes, see the GitHub repository awslabs/amazon-qldb-driver-java.	August 28, 2020
Node.js driver v2.0 (p. 593)	The Amazon QLDB driver for Node.js version 2.0 is now generally available. For release notes, see the GitHub repository awslabs/amazon-qldb-driver-nodejs.	August 27, 2020
Related information (p. 593)	Added a new topic that contains links for Related information and additional resources to help you understand and work with Amazon QLDB.	August 24, 2020
Python driver v3.0 (p. 593)	The Amazon QLDB driver for Python version 3.0 is now generally available. For release notes, see the GitHub repository awslabs/amazon-qldb-driver-python.	August 20, 2020
Amazon QLDB driver for Go (p. 593)	A preview release of the Amazon QLDB driver for Go is now available. This driver enables you to use the AWS SDK for Go to interact with QLDB's transactional data API. For more information, see Amazon QLDB driver for Go (preview).	August 6, 2020
Cookbook reference for the Python driver (p. 593)	Added a Cookbook reference that shows code examples of common use cases for the Amazon QLDB driver for Python.	July 24, 2020
Unique IDs in Amazon QLDB (p. 593)	Added a new section that describes the properties and usage guidelines of Unique IDs in Amazon QLDB.	July 9, 2020

AWS CloudFormation resource for journal streams (p. 593)	Amazon QLDB now supports a resource for creating journal streams using an AWS CloudFormation template. For more information, see the AWS::QLDB::Stream resource in the *AWS CloudFormation User Guide*.	July 9, 2020
Cookbook reference for the .NET driver (p. 593)	Added a Cookbook reference that shows code examples of common use cases for the Amazon QLDB driver for .NET.	July 1, 2020
Amazon QLDB driver for .NET (p. 593)	The Amazon QLDB driver for .NET is now generally available. This driver is open-sourced on GitHub and enables you to use the AWS SDK for .NET to interact with QLDB's transactional data API. For release notes, see the GitHub repository awslabs/amazon-qldb-driver-dotnet.	June 26, 2020
Amazon QLDB driver for Node.js (p. 593)	The Amazon QLDB driver for Node.js is now generally available. This driver is open-sourced on GitHub and enables you to use the AWS SDK for JavaScript in Node.js to interact with QLDB's transactional data API. For release notes, see the awslabs/amazon-qldb-driver-nodejs GitHub repository.	June 5, 2020
Amazon QLDB journal streams (p. 593)	Amazon QLDB now enables you to create a *stream* that captures every document revision that is committed to your journal and delivers this data to Amazon Kinesis Data Streams in near-real time. For more information, see Streaming journal data from Amazon QLDB.	May 19, 2020
Amazon Ion code examples (p. 593)	Added code examples that query and process Amazon Ion data in an Amazon QLDB ledger by using the QLDB driver for Java, Node.js, and Python. For more information, see Ion code examples in Amazon QLDB.	May 12, 2020

Concurrency model and journal contents (p. 593)	Expanded the Amazon QLDB concurrency model topic to add information about using indexes to limit *optimistic concurrency control* (OCC) conflicts. Added a new section that describes Journal contents in Amazon QLDB.	May 4, 2020
Quick start guide for the Node.js driver (p. 593)	Added a Quick start guide for the Amazon QLDB driver for Node.js.	May 1, 2020
Amazon QLDB shell (p. 593)	The Amazon QLDB shell is now generally available. This shell is open-sourced on GitHub and provides a command line interface that enables you to execute PartiQL statements on ledger data. For release notes, see the awslabs/amazon-qldb-shell GitHub repository.	April 20, 2020
Compliance certifications (p. 593)	Amazon QLDB is now certified for AWS compliance programs, including HIPAA and ISO. For more information, see Compliance validation for Amazon QLDB.	April 3, 2020
Expanded driver recommendations (p. 593)	Expanded the Amazon QLDB driver recommendations section to make it apply to all supported programming languages.	April 2, 2020
Amazon QLDB shell (p. 593)	A preview release of the Amazon QLDB shell is now available. This shell provides a command line interface that enables you to execute PartiQL statements on ledger data. For more information, see Accessing Amazon QLDB using the QLDB shell (data plane only) (preview).	March 23, 2020
Java driver v1.1 (p. 593)	The Amazon QLDB driver for Java version 1.1 is now generally available. For release notes, see the awslabs/amazon-qldb-driver-java GitHub repository.	March 20, 2020

Amazon QLDB driver for .NET (p. 593)	Amazon QLDB now provides a preview release of the .NET driver. This driver enables you to use the AWS SDK for .NET to interact with QLDB's transactional data API. For more information, see Amazon QLDB driver for .NET (preview).	March 13, 2020
Amazon QLDB driver for Python (p. 593)	The Amazon QLDB driver for Python is now generally available. This driver is open-sourced on GitHub and enables you to use the AWS SDK for Python (Boto3) to interact with QLDB's transactional data API. For release notes, see the awslabs/amazon-qldb-driver-python GitHub repository.	March 11, 2020
PartiQL UNDROP TABLE statement and nested DML (p. 593)	PartiQL in Amazon QLDB now supports *data manipulation language* (DML) statements in which nested collections are specified as sources in the `FROM` clause. For more information, see the FROM statement in the *PartiQL reference*. QLDB PartiQL also supports the UNDROP TABLE statement. For more information, see Undropping tables.	February 26, 2020
Expanded intro topic (p. 593)	Expanded the introductory *What is Amazon QLDB?* topic to include the new sections Overview of Amazon QLDB and From relational to ledger.	January 24, 2020
PartiQL reference for supported functions (p. 593)	Expanded the *Amazon QLDB PartiQL reference* to include detailed usage information about the supported SQL functions. For more information, see PartiQL functions.	January 2, 2020
Driver recommendations and common errors (p. 593)	Added new sections that describe Driver recommendations for the Amazon QLDB driver for Java and Common errors for all driver languages.	January 2, 2020

New Regions launch (p. 593)	Amazon QLDB is now available in the Asia Pacific (Seoul), Asia Pacific (Singapore), Asia Pacific (Sydney), and Europe (Frankfurt) Regions. For a complete list of available Regions, see Amazon QLDB endpoints and quotas in the *Amazon Web Services General Reference*.	November 19, 2019
Amazon QLDB driver for Node.js (p. 593)	Amazon QLDB now provides a preview release of the Node.js driver. This driver enables you to use the AWS SDK for JavaScript in Node.js to interact with QLDB's transactional data API. For more information, see Amazon QLDB driver for Node.js (preview).	November 13, 2019
Amazon QLDB driver for Python (p. 593)	Amazon QLDB now provides a preview release of the Python driver. This driver enables you to use the AWS SDK for Python (Boto3) to interact with QLDB's transactional data API. For more information, see Amazon QLDB driver for Python (preview).	October 29, 2019
Public release (p. 593)	This is the initial public release of Amazon QLDB. This release includes the Developer Guide and the integrated ledger management API reference.	September 10, 2019