AWS re:Invent

DAT352

Building applications on Amazon QLDB

Sid Sanyal
Principal TPM, Amazon QLDB
Amazon Web Services

Brett Hensley
Solution Architect, State & Local
Amazon Web Services

AWS re:Invent

aws

Agenda

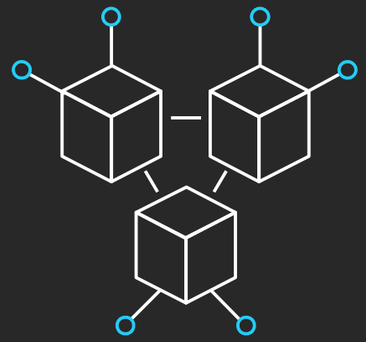
- Review
  - Amazon Quantum Ledger Database (QLDB) fundamentals
  - Amazon Ion & PartiQL fundamentals
  - Data modeling – “Vehicle Registration” example
  - Developing with Java – QLDB driver and Amazon Ion libraries
- Activity
  - Set up AWS Cloud9 IDE environment
  - Coding Exercise (Java tutorial application)
  - *….Verify documents (bonus)*

What is blockchain?

Ledgers

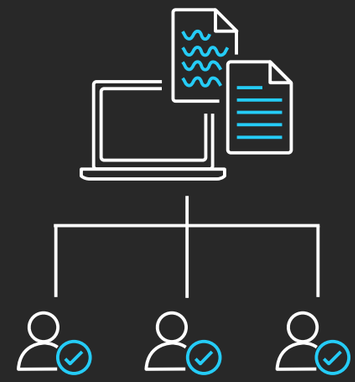
Immutable, append-only, cryptographically verifiable

Decentralization



Distributed trust and data replication

Consensus algorithms

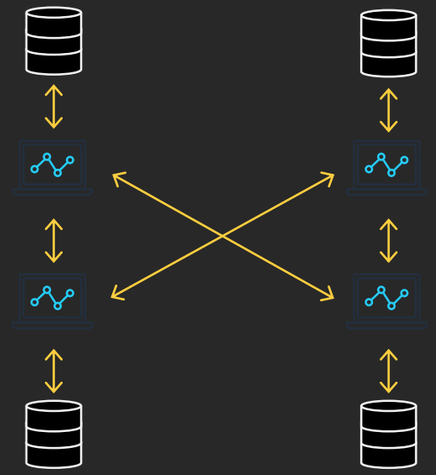


No intermediaries in decision process, support for smart contracts

Centralized (Amazon QLDB) vs. decentralized (blockchain)

Centralized

- Owned by a single, trusted authority
- Addresses core need of an immutable and verifiable transactional log
- Fast: doesn’t require consent from members to commit transactions



Decentralized

- No single owner of the ledger; Joint ownership by multiple parties
- Addresses core need of enabling multiple parties to transact transparently and with trust with each other
- Removes intermediaries when a group of members needs to transact; can make business processes more efficient

AWS's portfolio of purpose-built databases

Relational	Key-value	Document	In-memory	Graph	Time-series	Ledger
						
Amazon RDS	Amazon DynamoDB	Amazon DocumentDB	Amazon ElastiCache	Amazon Neptune	Amazon Timestream	Amazon QLDB

Recap of Amazon QLDB fundamentals

Sid Sanyal
Principal TPM, Amazon QLDB
Amazon Web Services

AWS re:Invent

aws

Traditional database architecture: The log

- Typically an internal implementation
- Difficult, or impossible, to directly access
- Used for replicating data

Amazon QLDB: the journal is the database

C

H

Cars (user view)

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

Document Revision History (system function)

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*				

Journal

J

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

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

H (T_1)

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

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

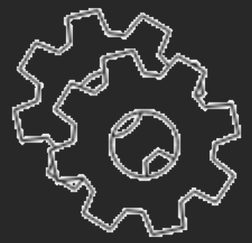
H (T_2)

```
DELETE cars
ID:1

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

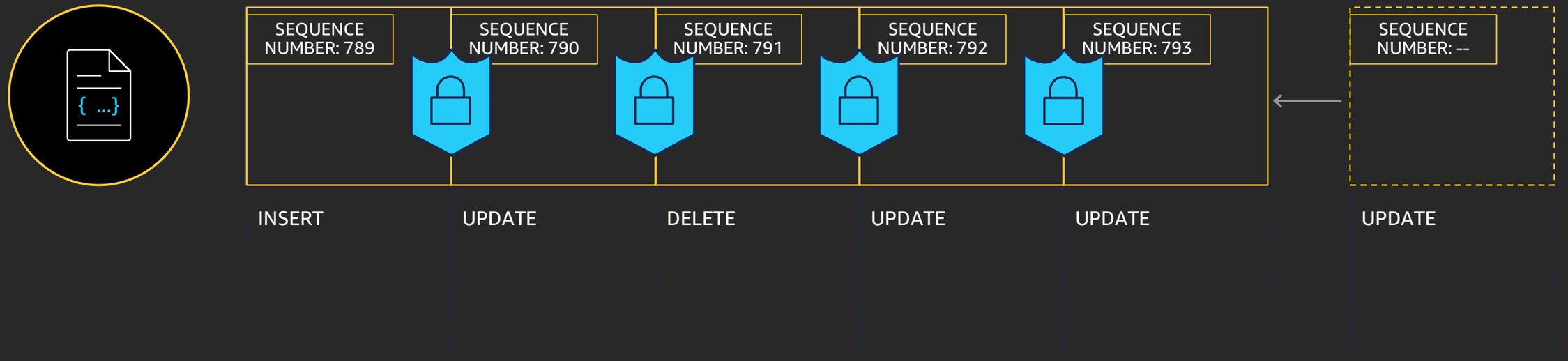
H (T_3)

Event Processor (streaming)



Immutable Journal

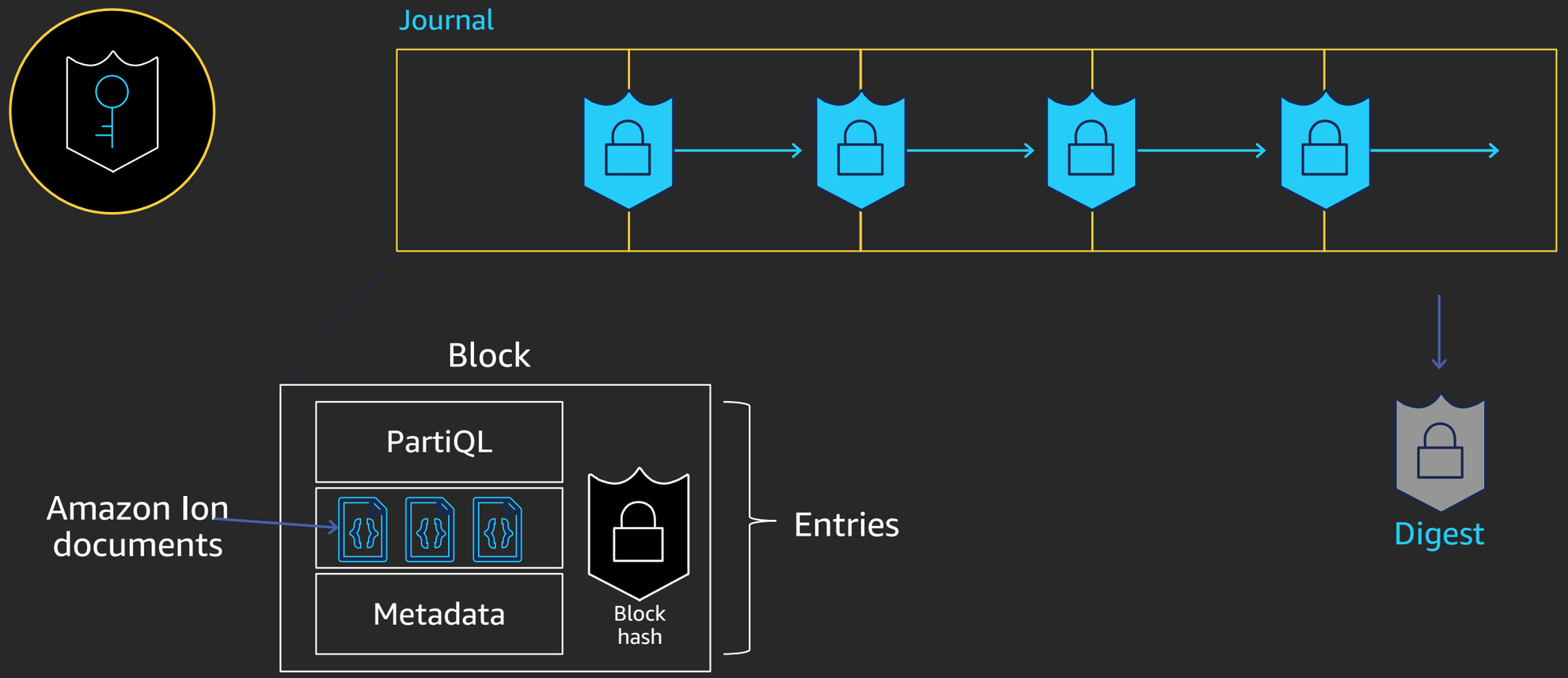
Records cannot be altered



- The journal is append-only and sequenced
- There is no API or other method to alter committed data
- All operations, including deletes, are written to the journal

Cryptographic verification

Hash chaining using SHA-256



Transactions (ACID)

HIGHEST TO LOWEST

Isolation level	Potential issues
Serializable	-
Snapshot Isolation	Potential write skew
Repeatable read	Phantom reads
Read committed	Phantom reads/non-repeatable reads
Read uncommitted	Phantom reads/non-repeatable reads/dirty reads

- Amazon QLDB supports the highest level of isolation
- There is no other mode for Amazon QLDB
- There is no risk that you'll see phantom reads, write skew, dirty reads, or other issues

Easy to use - Amazon Ion & PartiQL

Amazon Ion

```
/* Ion supports comments.
This is a “Vehicle” document */
{
    ‘VIN’ : ‘KM8SRDHF6EU074761’ ,
    ‘MfgDate’:      `2017-03-01T` ,
    ‘Type’:    ‘Truck’ ,
    ‘Mfgr’:    ‘Ford’ ,
    ‘Model’:   ‘F150’
    ‘Color”:  ‘Black’ ,
    ‘Specs’: {
          ‘EngSize’ : 3.3 ,//decimal
          ‘CurbWeight’: 4878 , //int
          ‘HP’: 327  //int
          ‘BatterySize’ : null.int ,
    }
}
```

PartiQL

```
INSERT INTO cars
     { 'Manufacturer':'Tesla',
       'Model':'Model S',
       'Year': 2012,
       'VIN': 123456789,
       'Owner':'Traci Russell'
     }

UPDATE cars SET owner = 'Ronnie
Nash'

WHERE VIN = 123456789

SELECT * FROM cars
```

Amazon QLDB summary

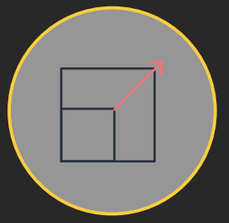
Immutable

Append-only, sequenced

Cryptographically verifiable

Hash-chaining provides data integrity

Highly scalable



Serverless, highly available

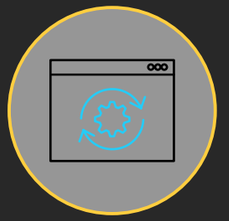
Easy to use

Familiar SQL operators

ACID transactions

Fully serializable isolation

Journal first



Journal is the database

Recap of PartiQL & Amazon Ion fundamentals

Kevin Yung
Senior Application Architect, Professional Services
Amazon Web Services

AWS re:Invent

aws

PartiQL: Fundamental design tenets

1. SQL compatibility
2. First-class nested data
3. Minimal extensions
4. Format independence
5. Data store independence

Basic SQL query

Input data (Ion format):

```
{
  VIN: "1N4AL11D75C109151",
  LicensePlateNumber: "LEWISR261LL",
  State: "WA",
  City: "Seattle",
  PendingPenaltyTicketAmount: 90.25,
  ValidFromDate: 2017-08-21T,
  ValidToDate: 2020-05-11T,
  Owners: {
    PrimaryOwner: {
      PersonId: "C6XZswBSmTS0myD5MW34B9"
    },
    SecondaryOwners: [
      {PersonId:"5Ufgdlnj06gF5CWcOIu64s"},
      {PersonId:"HrBDqUrWTHD3lN12CJ605q"}
    ]
  }
}
```

Query:

```
SELECT
  s.VIN AS VIN
FROM
  VehicleRegistration AS s
WHERE
  s.City = 'Seattle'
```

Output:

```
{
  VIN: "1N4AL11D75C109151"
}
```

Querying nested collections

Input data (Ion format):

```
{
  VIN: "1N4AL11D75C109151",
  LicensePlateNumber: "LEWISR261LL",
  State: "WA",
  City: "Seattle",
  PendingPenaltyTicketAmount: 90.25,
  ValidFromDate: 2017-08-21T,
  ValidToDate: 2020-05-11T,
  Owners: {
    PrimaryOwner: {
      PersonId: "C6XZswBSmTS0myD5MW34B9"
    },
    SecondaryOwners: [
      {PersonId:"5Ufgdlnj06gF5CWcOIu64s"},
      {PersonId:"HrBDqUrWTHD3lN12CJ605q"}
    ]
  }
}
```

Query:

```
SELECT
  s.VIN AS VIN,
  p.PersonId as PersonId
FROM
  VehicleRegistration AS s,
  s.Owners AS o,
  o.SecondaryOwners as p
WHERE
  p.PersonId = '5Ufgdlnj06gF5CWcOIu64s'
```

Output:

```
{
  VIN: "1N4AL11D75C109151",
  PersonId: "5Ufgdlnj06gF5CWcOIu64s"
}
```

Supported PartiQL statements in QLDB

- CREATE INDEX
- CREATE TABLE
- DELETE
- DROP TABLE
- FROM
- INSERT
- SELECT
- UPDATE

Supported PartiQL functions and operations in QLDB

CAST

CHAR_LENGTH

CHARACTER_LENGTH

COALESCE

EXTRACT

TO_STRING

LOWER

SIZE

NULLIF

UTCNOW

TO_TIMESTAMP

TRIM

DATE_ADD

LIKE

IN

UPPER

LOWER

INNER

COUNT

SUM

AVG

MAX

MIN

JOIN

DATE_DIFF

SUBSTRING

<

>

BETWEEN

EXISTS

Amazon Ion features

1. Value based
2. Supports primitive and collection types
3. Flexible and extensible
4. Supports hierarchy
5. Human readable
6. Binary format for efficiency

Ion in Amazon QLDB

1. SQL scalar types are covered by their Ion counterparts
2. Ion's struct type is equivalent to a SQL tuple
3. Support for timestamp, decimal, Blob (not available in JSON)

Primitive types

bool, int, float, decimal, timestamp, string, symbol, blob, clob, null

Collection types

struct, list

JSON vs. Amazon Ion

JSON document

```
{
      “VIN” : “KM8SRDHF6EU074761”,
      “MfgDate” : “2017-03-01”,
      “Type : “Truck”,
      “Mfgr” : “Ford”,
      “Model” : “F150”,
      “Color” : “Black”,
      “Specs” : {
            “EngSize” : 3.3,
            “CurbWeight”: 4878,
            “HP”: 327,
            “BatterySize”: null
      }
}
```

Amazon Ion document

```
/* Ion supports comments. */
{
      VIN : “KM8SRDHF6EU074761”,
      MfgDate : 2017-03-01T, // (timestamp)
      Type : “Truck”,
      Mfgr : “Ford”,
      Model : “F150”,
      Color : “Black”,
      Specs: {
            EngSize : 3.3     // (decimal)
            CurbWeight : 4878 // (int)
            HP : 327
            BatterySize : null.int
      }
}
```

https://github.com/amzn/ion-java

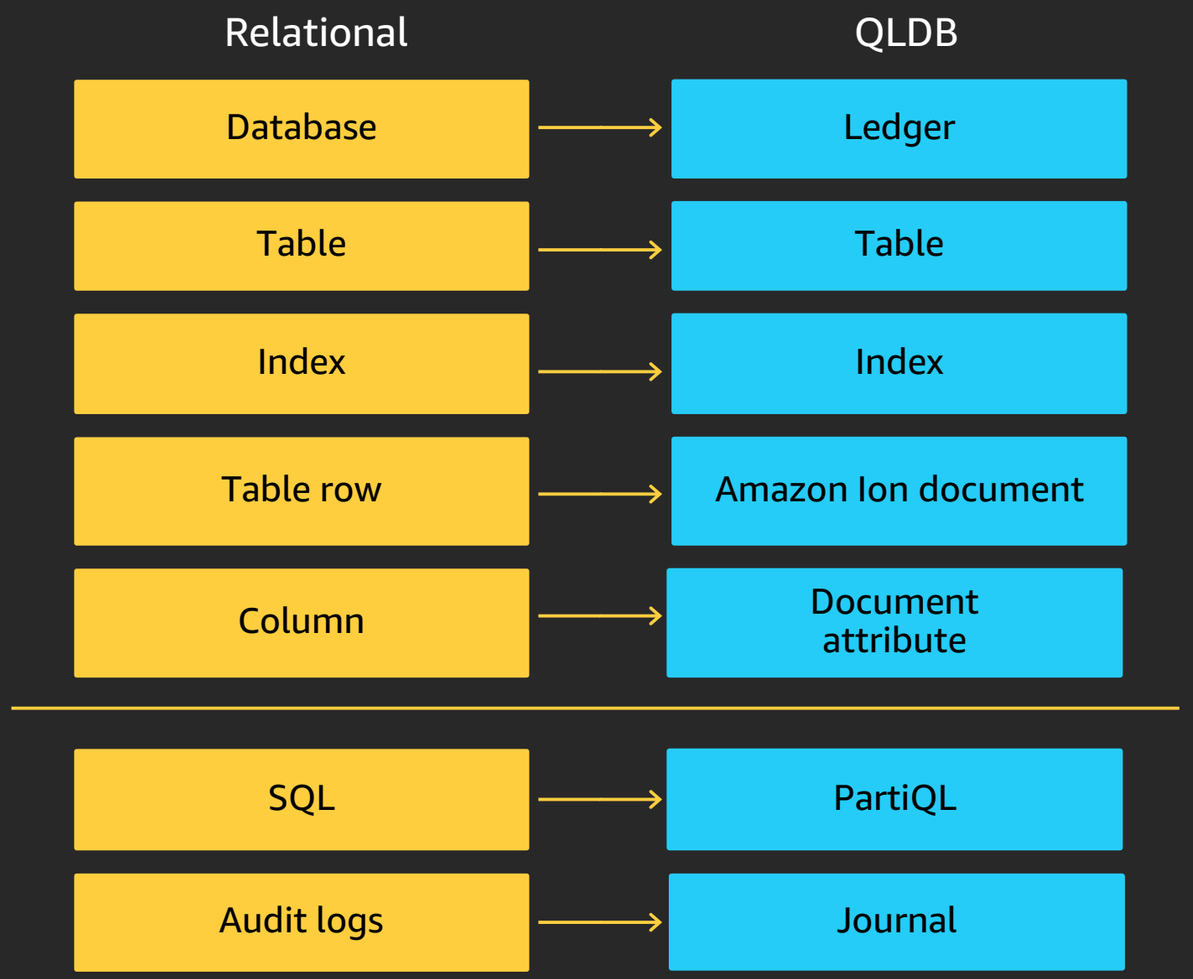
Data modeling & developing with Amazon QLDB

Sid Sanyal
Principal TPM, QLDB
Amazon Web Services

AWS re:Invent

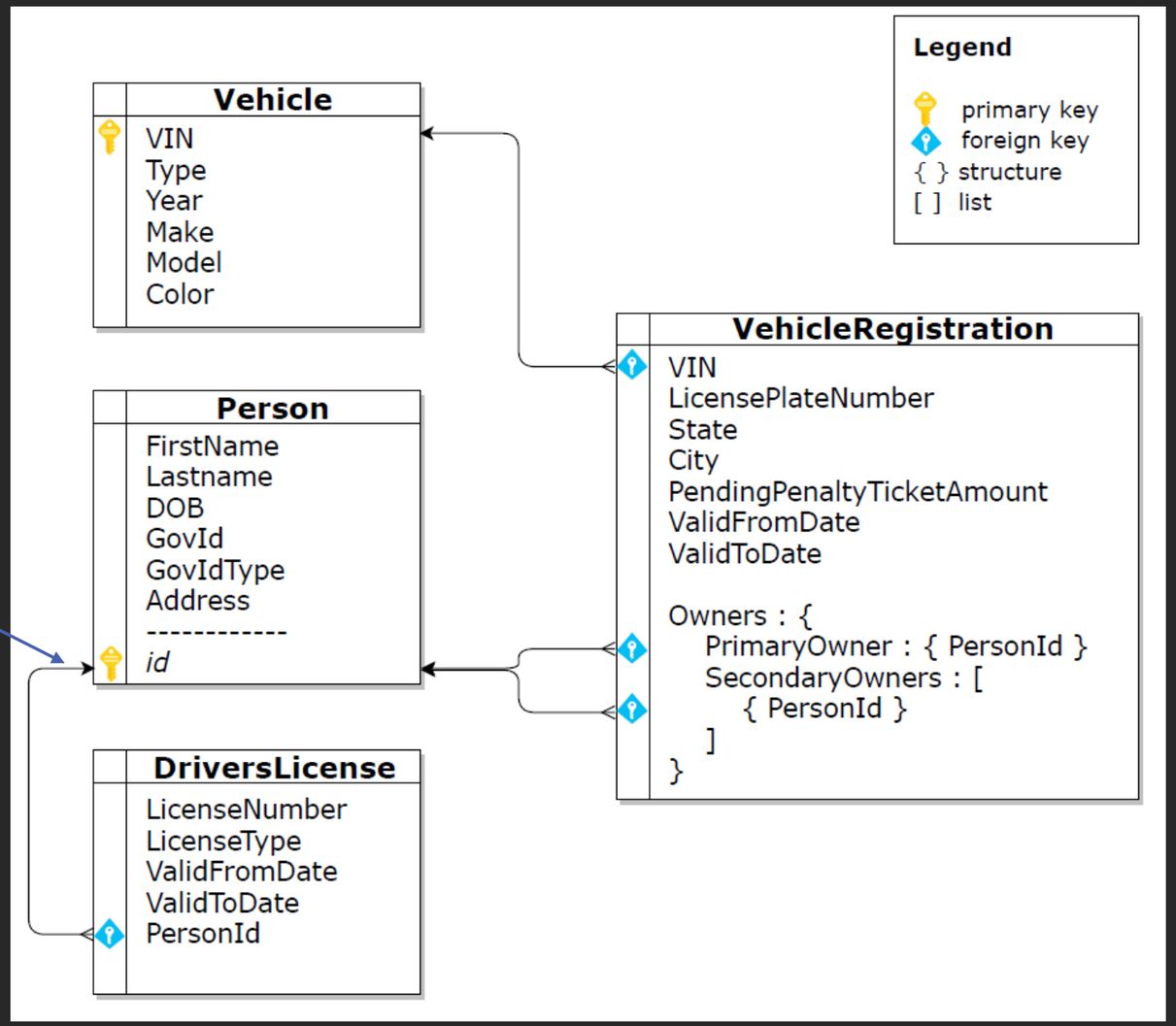
aws

Mapping constructs between relational database management system (RDBMS) & Amazon QLDB



Logical data model for “vehicle registration” example

“document ID” is a unique identifier assigned to each document



Physical data model - Tables

Create tables

CREATE TABLE
Person

CREATE TABLE
DriversLicense

CREATE TABLE
DriversLicense

CREATE TABLE
Vehicle

CREATE TABLE
VehicleRegistration

Physical data model - Indexes

Create indexes

CREATE INDEX ON
Person
(GovId)

CREATE INDEX ON
DriversLicense
(LicenseNumber)

CREATE INDEX ON
DriversLicense
(PersonId)

CREATE INDEX ON
Vehicle
(VIN)

CREATE INDEX ON
VehicleRegistration
(LicensePlateNumber)

Inserting documents

Person

Flexible document model leveraging Amazon Ion

```
INSERT INTO Person
{
  'FirstName' : 'Raul’,
  'LastName' : 'Lewis’,
  'DOB' : `1963-08-19T`,
  'GovId' : 'LEWISR261LL’,
  'GovIdType' : 'Driver License’,
}
```

Inserting nested documents


```
INSERT INTO VehicleRegistration
{
    'VIN' : '1N4AL11D75C109151', // index field
    'LicensePlateNumber' : 'LEWISR261LL',
    'State' : 'WA',
    'City' : 'Seattle',
    'PendingPenaltyTicketAmount' : 90.25,
    'ValidFromDate' : `2017-08-21T`,
    'ValidToDate' : `2020-05-11T`,
    'Owners' : {
        'PrimaryOwner' : {'PersonId': '294jJ3YUoH1IEEm8GSabOs'},
        'SecondaryOwners' : [ ]
    }
}
```

Nested document structure enables optimal queries and data access for a particular vehicle registration record

Capturing the "document ID" corresponding to the owner (Person table) provides uniqueness

Modifying documents

Scenario:

Add a secondary owner to the vehicle registration

Step1: Get the document ID (unique)

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

>> Returns document ID ‘5Ufgdlnj06gF5CWcOIu64s’

Use the FROM-INSERT clause to modify, remove, or insert elements within an Ion document

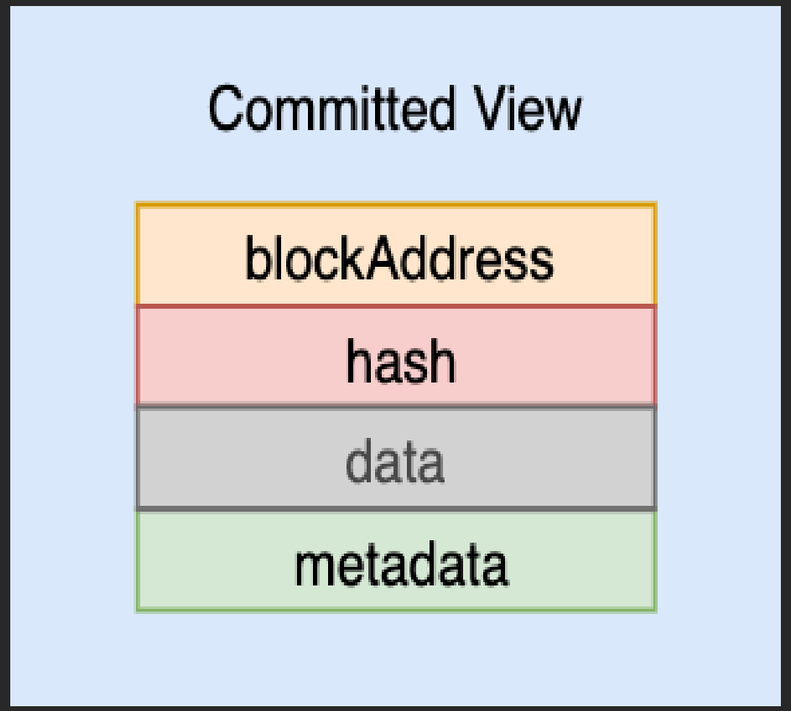
Step2: Insert this document ID into the SecondaryOwners list

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

Updated Ion document – Default View

```
{
  'VIN' : '1N4AL11D75C109151’,
  'LicensePlateNumber' : 'LEWISR261LL’,
  'State' : 'WA’,
  'City' : 'Seattle’,
  'PendingPenaltyTicketAmount' : 90.25,
  'ValidFromDate' : `2017-08-21T`,
  'ValidToDate' : `2020-05-11T`,
  'Owners' : {
    'PrimaryOwner' : {
       'PersonId': ' 'IN7MvYtUjkp1GMZu0F6CG9’
     },
    'SecondaryOwners' : [ {'PersonId’: '5UfgdlnjO6gF5CWcOIu64s’} ]
  }
}
```

Updated Ion document - Committed View




```
{
    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"
    }
}
```

Query document revision history

SELECT * FROM history(<Table>, fromDate, toDate) AS h WHERE h.metadata.id = '<*doc Id*>'

Version 0

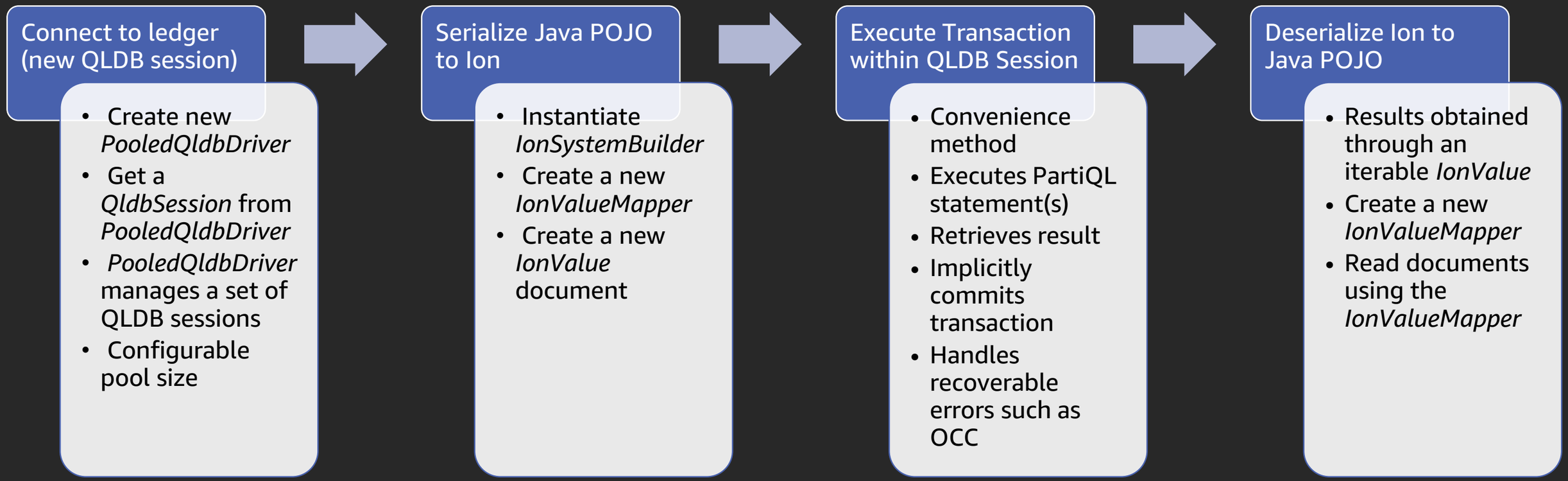
Version 1

Version 2

blockAddress	hash	data	metadata
{strandId:"Ad3A17nmNIGLFxMhY7DDTl",sequenceNo:0}	{{fwx9Sy3PABxQHoHoTnTM/GfOEtpJXm7AdZC2kwjaO8g=}}	{VIN:"1N4AL11D75C109151",LicensePlateNumber:"LEWISR261LL",State:"WA",City:"Seattle",PendingPenaltyTicketAmount:90.25,ValidFromDate:2017-08-21T,ValidToDate:2020-05-11T,Owners:{PrimaryOwner:{PersonId:"BqZVK31T1zOI2sz2vtgeJc"},SecondaryOwners:[]}}	{id:"6mW3XIkmw2Z6HMzhkdOy7S",**version:0**,txTime:2019-10-28T16:37:47.903Z,txId:"HLF4iYIVYNc5EGBH8EeZAN"}
{strandId:"Ad3A17nmNIGLFxMhY7DDTl",sequenceNo:13}	{{wpsGTBAq8qKXpeTRBgUbmYoDkSiAj6epXXQ8eB7tIDk=}}	{VIN:"1N4AL11D75C109151",LicensePlateNumber:"LEWISR261LL",State:"WA",PendingPenaltyTicketAmount:90.25,ValidFromDate:2017-08-21T,ValidToDate:2020-05-11T,Owners:{PrimaryOwner:{PersonId:"3Qv6BaVYn21497hc76kuKn"},SecondaryOwners:[]},City:"Everett"}	{id:"6mW3XIkmw2Z6HMzhkdOy7S",**version:1**,txTime:2019-10-28T16:48:26.456Z,txId:"J3hZ9aKfPRw88oogqevlWc"}
{strandId:"Ad3A17nmNIGLFxMhY7DDTl",sequenceNo:25}	{{QKS4W1VF2gcdlgs46koDJcFRORogmjSkYzrnN9b32XA=}}	{VIN:"1N4AL11D75C109151",LicensePlateNumber:"LEWISR261LL",State:"WA",PendingPenaltyTicketAmount:90.25,ValidFromDate:2017-08-21T,ValidToDate:2020-05-11T,Owners:{PrimaryOwner:{PersonId:"3Qv6BaVYn21497hc76kuKn"},SecondaryOwners:[{PersonId:"1yQgJbt7x8qJom1Nmtfgjy"}]},City:"Everett"}	{id:"6mW3XIkmw2Z6HMzhkdOy7S",**version:2**,txTime:2019-10-28T16:50:32.981Z,txId:"Gucx6nIzfNaDa4MgtPtxNi"}

Core concepts for building Java applications on QLDB

Use QLDB Java driver API to connect to QLDB ledger and execute PartiQL statements



Core concepts for building Java applications on QLDB

Serialize Java POJO to Ion

```
class Person {. . .}  // Create a serializable POJO class

Person p = new Person(. . .);

IonSystem ion = IonSystemBuilder.standard().build() //Entry point to Ion

IonObjectMapper mapper = new IonValueMapper(ion);   //Serialization Mapper

IonValue document = mapper.writeValueAsIonValue(p); //Serialize to Ion
```

Core concepts for building Java applications on QLDB

Execute transaction to insert documents using convenience method in QLDB driver (*PooledQldbDriver*)

```
try (QldbSession qldbSession = ConnectToLedger.createQldbSession()) {

       qldbSession.execute(txn -> {

          txn.execute(“INSERT INTO personTable ?”, Collections.singletonList(document));

          }

       }, (retryAttempt) -> log.info("Retrying due to ..."));  //Specify a useful message

} catch (Exception e) {

       log.error("Unable to . . .", e); //After max retries are exhausted

}
```

Core concepts for building Java applications on QLDB

Execute a PartiQL query statement

```
Result result = txn.execute(“SELECT * FROM personTable”);
```

Retrieve query results

```
List<IonStruct> documentList = new ArrayList<>();

result.iterator().forEachRemaining(row -> documentList.add((IonStruct) row));
```

Deserialize Ion to Java POJO

```
IonObjectMapper mapper = new IonValueMapper(ion);   //Serialization Mapper

for(IonStruct i : documentList) {

  Person p = mapper.readValue(i, Person.class);

}
```

Core concepts for using the QLDB Driver API

- Convenient abstractions
  - Session pooling (*PooledQldbDriver)*
  - Handling implicit transaction commits
  - Automatic retries (configurable) for recoverable errors (such as OCC)
- A *QldbSession* represents a logical connection to your QLDB ledger
- Execute one or more transactions in a *QldbSession*
- Support for multiple PartiQL statements in a transaction
- Support for Java Lambda expressions

Amazon QLDB developer ecosystem

Drivers (Client SDKs)	Java: github.com/awslabs/amazon-qldb-driver-java
	Python: github.com/awslabs/amazon-qldb-driver-python
	NodeJS: github.com/awslabs/amazon-qldb-driver-nodejs
Sample tutorials	Java: github.com/aws-samples/amazon-qldb-dmv-sample-java/
	Python: github.com/aws-samples/amazon-qldb-dmv-sample-python/
	NodeJS: github.com/aws-samples/amazon-qldb-dmv-sample-nodejs/

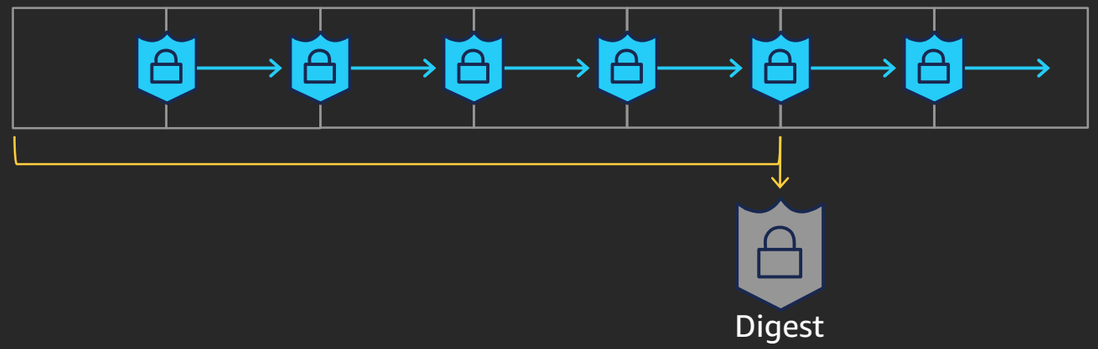
Verification with Amazon QLDB

Cryptographic verifiability

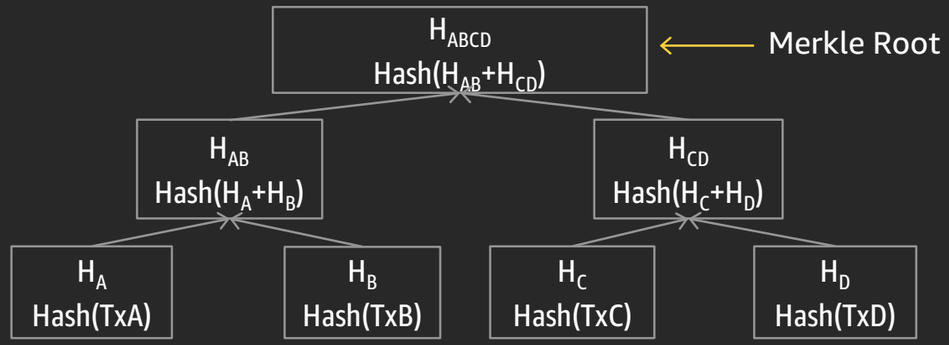
Four basic steps to seeing how QLDB's verifiability works

SHA256: Unique signature of a document

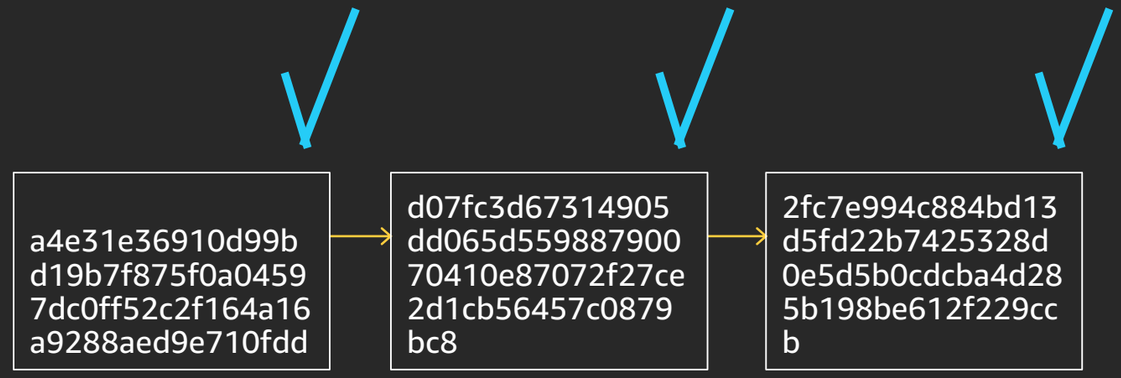
Digest: Periodic hash covering all history



Merkle Trees: Chaining past hashes together



Proof: A chain of hashes linking a document to its digest



Cryptographic verifiability: SHA-256

QLDB uses the SHA-256 algorithm to create unique, fixed-length outputs (hashes)

Change any part, even one character, and the output (hash) is different

```
vehicle = {
            ‘VIN’ :       “KM8SRDHF6EU074761”,
            ‘Type’:       “Truck”
            ‘Model’:      “F150”
            ‘Specs’: {
                  ‘EngSize’ : 3.3
                  ‘CurbWeight’: 4,878
                  ‘HP’: 327
            }
}
```

SHA-256

a4e31e36910d99bd19b7f
875f0a04597dc0ff52c2f16
4a16a9288aed9e710fdd

```
vehicle = {
            ‘VIN’ :       “KM8SRDHF6EU074761”,
            ‘Type’:       “Truck”
            ‘Model’:      “F150”
            ‘Specs’: {
                  ‘EngSize’ : 3.3
                  ‘CurbWeight’: 4,879
                  ‘HP’: 327
            }
}
```

SHA-256

1931845740892 0af2d2cb
eacd90c7afe0fbd7f6ff316
972c8f656c8bbc402dd1

Cryptographic verifiability: SHA-256

SHA-256 is one way. It is infeasible to compute the input given an output.

```
vehicle = {
            ‘VIN’ :       “KM8SRDHF6EU074761”,
            ‘Type’:       “Truck”
            ‘Model’:      “F150”
            ‘Specs’: {
                  ‘EngSize’ : 3.3
                  ‘CurbWeight’: 4,878
                  ‘HP’: 327
            }
}
```

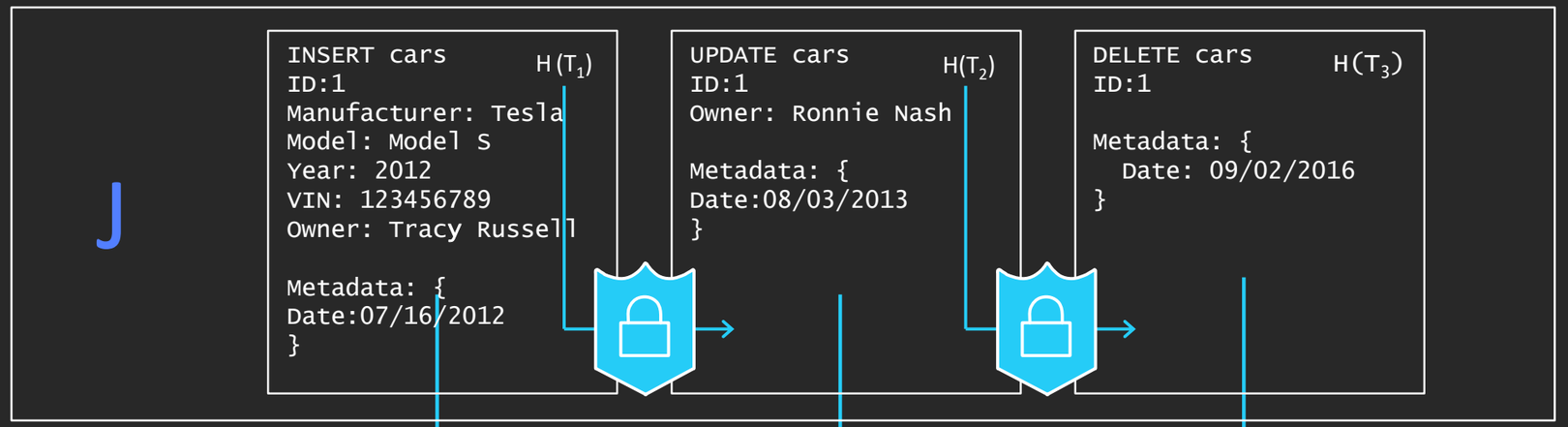
SHA-256

a4e31e36910d99bd19b7f
875f0a04597dc0ff52c2f16
4a16a9288aed9e710fdd

SHA-256

19318457408920af2d2cb
eacd90c7afe0fbd7f6ff316
972c8f656c8bbc402dd1

Changing committed data breaks the chain



$H(T_1)$ = ~~2526f16306c819d651af07593417Od2430d246d9ab98d975d28a83baded47ca7~~

$H(T_1)$ = 25d0b44e6e8878151646ffc1fea4eb85c3e4bf4baec212a9fcf67b6d5a81e01a

$H(T_2)$ = ~~86a90e4166453d9423b84d47dcbd97c0e3099b1a1f0d7cfca6c191d8fd8994ff~~

$H(T_2)$ = a90a9898c7e4b1aab19c705b554afd9e0bf6539bb0346df19be362ff63001098

$H(T_3)$ = ~~ae2d64e562ec754ec3194c744eec72c9fdafffc6b559e0414d0e75bf96ca92ad~~

$H(T_3)$ = c6268578a24dbe0c7cfba07bd967411a35462b8c875d42f1991faad02c0ac93c

Why does immutability and verifiability matter?

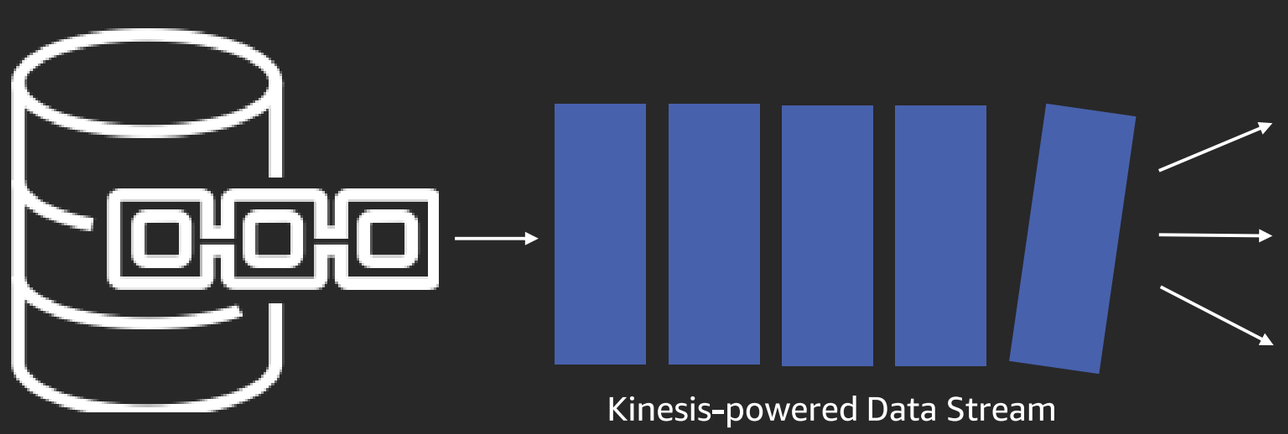
Reduce risk: Ensure safeguarding of critical system-of-record applications where data loss could be expensive

Improve data tracking: Helps you or any parties that have access to the system to quickly and accurately track data's entire lineage, improving efficiency in tracking the source of issues (e.g., manufacturing defects, maintain supply network data hygiene)

Auditability: Helps reduce downtime caused due to audit and compliance issues, saving hundreds of productivity hours for your team

Reduce implementation effort: Building immutability and verifiability in a traditional way is time consuming, complex, and expensive

Announcing private preview of QLDB streaming



Event processing (e.g., AWS Lambda triggers)

Real-time analytics (e.g., Amazon Athena, Amazon Kinesis Data Firehose)

Linkage to other purpose-built database (e.g., Amazon Elasticsearch Service, Amazon Neptune)

To get started, email: qldb-outbound@amazon.com

Hands-on lab

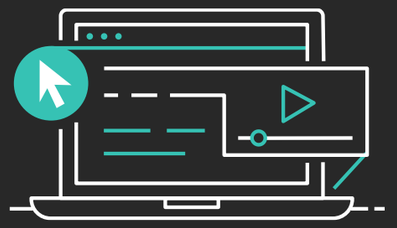
https://bit.ly/35XF9Ul

Meet us at our booth

- Come talk to our engineering and product teams and get answers to your questions
- Answer some trivia and win a QLDB shirt
- Venetian Expo Hall, AWS Village, Blockchain and Ledger Booth

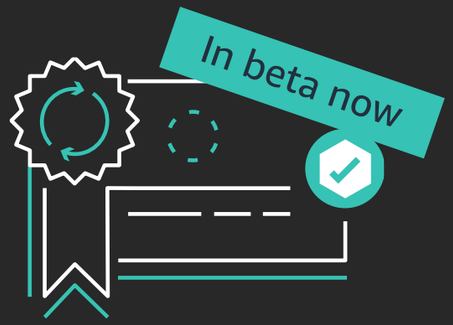
Learn databases with AWS Training and Certification

Resources created by the experts at AWS to help you build and validate database skills



25+ free digital training courses cover topics and services related to databases, including:

- Amazon Aurora
- Amazon Neptune
- Amazon DocumentDB
- Amazon DynamoDB
- Amazon ElastiCache
- Amazon Redshift
- Amazon RDS



Validate expertise with the new **AWS Certified Database - Specialty** beta exam

Visit aws.training

aws training and certification

Thank you!

AWS re:Invent

aws

Please complete the session survey in the mobile app.