Building improved Data Integrity and Data Exchange Systems with Blockchain and Ledger Technologies

Amazon Web Services

Presented to: ATIEC 2019

By: Lana Kalashnyk, Principal Blockchain Architect lkalash@amazon.com

Date: ATIEC 2019
Let’s discuss

• Blockchain vs a Ledger Technology
• Amazon Managed Blockchain
• Amazon Quantum Ledger Database
• Customer Success Stories
Blockchain vs a Ledger
How do we think about blockchain?

Blockchain: healthcare's next frontier, or so much hype?
June 25, 2018

Blockchain Logistics – Changing the World or Just Marketing Hype?
15. July 2018 | Articles

Blockchain will be the killer app for supply chain management in 2018
The distributed ledger technology that underpins cryptocurrencies is now poised to disrupt supply chain management – especially in the global shipping industry.

Blockchain is this year’s buzzword – but can it outlive the hype?
The open-source ledger behind bitcoin is touted as revolutionary for everything from banking to health, but the jury is still out.
Need for a ledger with centralized trust

• DMV
  • Track vehicle title history

• Healthcare
  • Verify and track hospital equipment inventory

• Manufacturers
  • Track distribution of a recalled product

• HR & Payroll
  • Track changes to an individual’s profile

• LEDGERS WITH CENTRALIZED TRUST

• TRANSACTIONS WITH DECENTRALIZED TRUST
# Challenges customers face

<table>
<thead>
<tr>
<th>Building ledgers with traditional databases</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Resource intensive</td>
</tr>
<tr>
<td>• Difficult to manage and scale</td>
</tr>
<tr>
<td>• Error prone and incomplete</td>
</tr>
<tr>
<td>• Impossible to verify</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Blockchain approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Designed for a different purpose</td>
</tr>
<tr>
<td>• Adds unnecessary complexity</td>
</tr>
</tbody>
</table>
Need for running transactions with decentralized trust

- Financial institutions
  - Peer-to-peer payments

- Mortgage lenders
  - Process syndicated loans

- Supply Chain
  - Transact with suppliers and distributors

- Retail
  - Streamline customer rewards
Complexity of Multi-Party Businesses

- Multi-party businesses could achieve better outcomes by sharing information, but need:
  - A way to independently verify transactions
  - Single, current and accurate view of data with tamper-proof history of transactions

- To this end, organizations with multi-party business use:
  - Central authorities to securely and fairly share data and
  - Employ costly escrow process for asset transfers.
Blockchain

**What is it**
Linked transaction data in encrypted, redundant databases, or ledgers, hosted across the Internet

**What it does**
Makes online transactions across enterprises more secure and trustworthy
Lowers cost by eliminating the need for traditional intermediaries

**How**
Encrypted, redundant data prevents destruction or falsification of data in any single ledger

**Who**
No single entity controls the data, further reducing risk
Perspectives

- Business
- Technical
- Legal
Typical Distributed Application Stack

Applications

APIs
- Membership
- Cryptographic
- Events
- Transactions

Consensus

Smart Contracts

Distributed Ledger
Permissioned or Permissionless

• **Permissioned**
  - Users enrolled before transactions
  - Identifiable users
  - Trace transactions to users

• **Permissionless**
  - Anonymous
  - Anyone can perform transactions
  - Commonly restricted to operations on own data
Blockchain components: “smart contracts”

- Rules embedded in app
- Verified execution of code
- Conditional operators
- Application writes to ledger
- Contract can interact with components outside of the blockchain network (off-chain)
Consensus Algorithms

• Proof of Work
• Proof of Stake
• Proof of Authority
• Proof of Elapsed Time (PoET)
• Endorsement, Ordering, Validation (PBFT)
Blockchain Components: Sample Distributed Ledger

Block1 Header
- PrevHash
- ...
- Merkle Root

<Transactions>

Block2 Header
- PrevHash
- ...
- Merkle Root

<Transactions>

Block3 Header
- PrevHash
- ...
- Merkle Root

<Transactions>
Challenges with existing blockchain solutions

<table>
<thead>
<tr>
<th>Setup is hard</th>
<th>Hard to scale</th>
<th>Complicated to manage</th>
<th>Expensive</th>
</tr>
</thead>
</table>

Amazon Managed Blockchain
What is Amazon Managed Blockchain?

Amazon Managed Blockchain is a fully managed service that makes it easy to create and manage scalable blockchain networks using popular open source frameworks:

Hyperledger Fabric and Ethereum
Amazon Managed Blockchain

- Fully managed
  Create a blockchain network in minutes

- Open-source variety
  Support for two frameworks

- Decentralized
  Democratically govern the network

- Reliable & scalable
  Backed with Amazon QLDB technology

- Low cost
  Only pay for resources used

- Integrated
  Send data to Amazon QLDB for secure analytics
How Amazon Managed Blockchain works

1. Create a network
   - Choose an open source blockchain framework

2. Invite members
   - Invite other AWS accounts to join the network

3. Add nodes
   - Create peer nodes that store a copy of the distributed ledger

4. Deploy applications
   - Create and deploy decentralized applications
Who ”owns” the network?

- Networks are decentralized and can remain active even after the initial creator leaves
- Inviting members to join
  - Preview: network creator can invite
  - GA: members vote on who to invite and remove
- Network-wide settings
  - GA: members can vote on network-wide settings and configure the actual voting rules (e.g., majority rules or one member decides)
- Each member pays for their resources
- Amazon Managed Blockchain manages shared components like the ordering service and networking settings
Amazon Managed Blockchain

Fabric Network managed by Amazon Managed Blockchain

Hyperledger Fabric Ordering Service (Orderer)

Bank A
- Fabric Certificate Authority
- Peer Node

Bank B
- Fabric Certificate Authority
- Peer Node

RMO
- Fabric Certificate Authority
- Peer Node

Central Bank
- Fabric Certificate Authority
- Peer Node

Amazon VPC
PrivateLink

VPC Bank A
Buyer/Seller
- VPC Endpoint
- Fabric Client Node

VPC Bank B
Buyer/Seller
- VPC Endpoint
- Fabric Client Node

VPC RMO
Arbitrator
- VPC Endpoint
- Fabric Client Node

VPC Central
Observer
- VPC Endpoint
- Fabric Client Node

Amazon VPC
PrivateLink

Aviation Information World - Forecasting the Future
Transaction flow with Hyperledger Fabric

1. Submitting-client
2. Transaction simulation
3. Endorsement signature
4. Broadcast endorsement
5. Verify policy
6. Transaction delivery to peers
Augmented Hyperledger Fabric

Ordering service

Core component of a Fabric network to guarantee delivery and order of transactions

Production grade networks using open source will utilize Apache Kafka for this component

Managed Blockchain uses Amazon QLDB technology, increasing durability and reliability

Certificate authority

Open source uses a “soft” HSM

Managed Blockchain uses AWS Key Management Service (AWS KMS) to secure the Certificate Authority service
Channels and private data for access control

Channels allow isolation of transactions among specific members in the network.

Create or update a channel with a configuration transaction (configtx).

Private data enables sub-channel access control.
Endorsement policies allow chaincode to specify which members (or how many) need to validate a transaction before submitting.

Endorsed transactions then get submitted to the ordering service and assembled into blocks.
Problem
Nestlé is committed to bringing transparency into the origin and quality of the ingredients used in their products, and wants their customers to have visibility into the end-to-end supply chain for their single origin coffee.

Solution
AWS Professional Services built supply-chain asset-tracking smart contracts to track single origin coffee from farm to customer on Amazon Managed Blockchain network, and exposed the contracts via a RESTful API. Nestlé’s mobile app consumes the API to capture events as the coffee moves through the supply chain.

Impact
Nestlé and their customers can now track the high quality single origin coffee from farmer to customer. Nestlé now has a platform they can expand to and trace the provenance of other products from their brand portfolio.
Sony Music Entertainment Japan - Music Rights

Sony Music Entertainment Japan (SMEJ) is committed to helping musicians and artists by removing undifferentiated heavy lifting such as filing and processing content rights and allow artists to focus more time on producing their work.

Solution

Using SMEJ’ system on AWS, participants will be able to share and verify information such as date and time of creation, and the author’s details and automatically verify the rights generation of any piece of written work.

Outcomes

The system is expected to improve productivity while maintaining proper rights processing, creating an environment where new generations of creators can launch hit content.
Legal & General picked Amazon Managed Blockchain for their global Pension Risk Transfer (PRT) ecosystem. This is a single ecosystem capable of driving every stage of the PRT reinsurance value chain including pricing, claims handling, financial reporting and collateral, utilising data dynamically stored on the blockchain.

Solution

With Amazon Managed Blockchain, Legal and General is able to create a solution that addresses not only the greater speeds at which risks are transacted but also drives transparency and security in an increasingly interconnected market. This platform replaces multiple processes and systems traditionally used to support each function, with the added security of blockchain technology.

Impact

Legal & General’ platform enables the Group to provide excellent service to customers in multiple markets at lower costs, redefining the way long term life reinsurance business is sold and managed. All Legal & General Reinsurance clients will eventually be supported on this platform.
Amazon Managed Blockchain Customers

Guardian
healthdirect
Liberty Mutual Insurance
GE Aviation
CHANGE HEALTHCARE
workday
PHILIPS
DTCC
verizon

Aviation Information World - Forecasting the Future
Amazon Quantum Ledger Database
# Purpose-built databases at AWS

<table>
<thead>
<tr>
<th><strong>Relational</strong></th>
<th><strong>Key-value</strong></th>
<th><strong>Document</strong></th>
<th><strong>In-memory</strong></th>
<th><strong>Graph</strong></th>
<th><strong>Time-series</strong></th>
<th><strong>Ledger</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Referential integrity, ACID transactions, schema-on-write</td>
<td>High throughput, low-latency reads and writes, endless scale</td>
<td>Store documents and quickly access querying on any attribute</td>
<td>Query by key with microsecond latency</td>
<td>Quickly and easily create and navigate relationships between data</td>
<td>Collect, store, and process data sequenced by time</td>
<td>Complete, immutable, and verifiable history of all changes to application data</td>
</tr>
<tr>
<td>Lift and shift, ERP, CRM, finance</td>
<td>Real-time bidding, shopping cart, social, product catalog, customer preferences</td>
<td>Content management, personalization, mobile</td>
<td>Leaderboards, real-time analytics, caching</td>
<td>Fraud detection, social networking, recommendation engine</td>
<td>IoT applications, event tracking</td>
<td>Systems of record, supply chain, healthcare, registrations, financial</td>
</tr>
</tbody>
</table>
Traditional database architecture: the log

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

Application data
Credit & debit transactions, insurance claim history, supply chain asset tracking, vehicle records, and more.

Amazon Quantum Ledger Database

Current state and indexed history
Stores the current value and historical state of your data, for example, the current value of a bank account, and its history.

Journal
Append-only, immutable journal stores a sequenced, cryptographically verifiable entry of each change made. Changes are chained together as blocks. For example, credits & debits.

Access complete history of changes to your data.

Cryptographically verify the change history of your data.
Amazon QLDB: the journal is the database

- QLDB’s journal has structural similarity to a database log
- All writes go to the journal—the journal determines state
- Journal handles concurrency, sequencing, cryptographic verifiability, and availability
- Accessible history of all transactions, document versions, document metadata

<table>
<thead>
<tr>
<th>ID</th>
<th>Manufacturer</th>
<th>Model</th>
<th>Year</th>
<th>VIN</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tesla</td>
<td>Model S</td>
<td>2012</td>
<td>123456789</td>
<td>Robert Dennison</td>
</tr>
</tbody>
</table>

### history() #function to query document history

<table>
<thead>
<tr>
<th>blockAddress</th>
<th>hash</th>
<th>data</th>
<th>metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>(strandId: &quot;pbmngzF2V7FHj4uE1R10&quot;, sequenceNo: 78)</td>
<td>[XXkYx5WElY8PRgup1X6xQ6UE2PEBa8nc0XxVGM8c=]</td>
<td>(Manufacturer: &quot;Tesla&quot;, Model: &quot;Model S&quot;, Year: &quot;2012&quot;, VIN: &quot;123456789&quot;, Owner: &quot;Traci Russell&quot;)</td>
<td>(id: &quot;5PLf8cUXL&quot;, version: 0, txTime: 2019-06-28, txId: &quot;3mOCdWAbtY6vGpFjUD6Gf&quot;)</td>
</tr>
<tr>
<td>(strandId: &quot;60bpz7k1B4B11uw0khe8&quot;, sequenceNo: 11)</td>
<td>[ii2h58wRChk/1ZRp4RLvG9DR2SNDa32r/UV2ctS1le=]</td>
<td>(Manufacturer: &quot;Tesla&quot;, Model: &quot;Model S&quot;, Year: &quot;2012&quot;, VIN: &quot;123456789&quot;, Owner: &quot;Traci Russell&quot;)</td>
<td>(id: &quot;Kwo6aQwJ4D1o1y0KgRx&quot;, version: 1, txTime: 2019-07-04T20:22:07Z, txId: &quot;68Fqpx97Mts4s6Ed339gMf&quot;)</td>
</tr>
<tr>
<td>(strandId: &quot;60bpz7k1B4B11uw0khe8&quot;, sequenceNo: 13)</td>
<td>[UdPrq7T7HfikK958Rv8Pg0ico5pFtDE5mOQ6Gr=]</td>
<td>(Manufacturer: &quot;Tesla&quot;, Model: &quot;Model S&quot;, Year: &quot;2012&quot;, VIN: &quot;123456789&quot;, Owner: &quot;Traci Russell&quot;)</td>
<td>(id: &quot;Kwo6aQwJ4D1o1y0KgRx&quot;, version: 2, txTime: 2019-07-04T20:45:768Z, txId: &quot;23knh4IuvHlXdW36kFls&quot;)</td>
</tr>
</tbody>
</table>

### User #standard user data, the default

<table>
<thead>
<tr>
<th>ID</th>
<th>Manufacturer</th>
<th>Model</th>
<th>Year</th>
<th>VIN</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tesla</td>
<td>Model S</td>
<td>2012</td>
<td>123456789</td>
<td>Robert Dennison</td>
</tr>
</tbody>
</table>
Amazon QLDB: the journal is the database

<table>
<thead>
<tr>
<th>ID</th>
<th>Manufacturer</th>
<th>Model</th>
<th>Year</th>
<th>VIN</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tesla</td>
<td>Model S</td>
<td>2012</td>
<td>123456789</td>
<td>Robert Dennison</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Version</th>
<th>Start</th>
<th>Manufacturer</th>
<th>Model</th>
<th>Year</th>
<th>VIN</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>7/16/2012</td>
<td>Tesla</td>
<td>Model S</td>
<td>2012</td>
<td>123456789</td>
<td>Traci Russell</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>8/03/2013</td>
<td>Tesla</td>
<td>Model S</td>
<td>2012</td>
<td>123456789</td>
<td>Ronnie Nash</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>9/02/2016</td>
<td>Tesla</td>
<td>Model S</td>
<td>2012</td>
<td>123456789</td>
<td>Robert Dennison</td>
</tr>
</tbody>
</table>
Easy to use (SQL)

• **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
Serverless, scalable, highly available

- Region
- Availability zone 1
- Journals
- Availability zone 2
- Journals
- Availability zone 3
- Journals

- Multi-AZ for high availability
- Multiple copies per AZ providing strong durability
Immutable

• 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
Amazon QLDB summary

- Immutable
  - Append-only, sequenced

- Cryptographically verifiable
  - Hash-chaining provide data integrity

- Highly scalable
  - Serverless, highly available

- Easy to use
  - Familiar SQL operators

- ACID Transactions
  - Fully serializable isolation

- Journal-first
  - The journal is the database
QLDB’s data model: Amazon Ion

data format

vehicle = {
  'VIN': "KM8SRDHF6EU074761",
  'MfgDate': '2017-03-01T00:00:00Z',
  'Type': "Truck",
  'Mfgr': "Ford",
  'Model': "F150",
  'Color': "Black",
  'Specs': {
    'EngSize': 3.3,
    'CurbWeight': 4878,
    'HP': 327,
    'BatterySize': null
  }
}

/* Ion supports comments. */

vehicle = {
  'VIN': "KM8SRDHF6EU074761",
  'MfgDate': 2017-03-01T00:00:00Z,
  'Type': "Truck",
  'Mfgr': "Ford",
  'Model': "F150",
  'Color': "Black",
  'Specs': {
    'EngSize': 3.3 (decimal)
    'CurbWeight': 4878 (int)
    'HP': 327 (int)
    'BatterySize': NULL.int
  }
}

https://github.com/amzn/ion-java
QLDB’s data model: PartiQL

SQL-compatible language

```javascript
vehicle = {
  'VIN': "KM8SRDHF6EU074761",
  'MfgDate': 2017-03-01T // timestamp
  'Type': "Truck"
  'Mfgr': "Ford"
  'Model': "F150"
  'Color': "Black"
  'Specs': {
    'EngSize': 3.3
    'CurbWeight': 4,878
    'HP': 327
    'BatterySize': NULL // null values
  }
}
```

```sql
SELECT VIN, Specs.EngSize, Specs.HP
FROM vehicles as v
WHERE v.type = 'Truck'
```

<table>
<thead>
<tr>
<th></th>
<th>Specs.EngSize</th>
<th>Specs.HP</th>
</tr>
</thead>
<tbody>
<tr>
<td>KM8SRDHF6EU074761</td>
<td>3.3</td>
<td>327</td>
</tr>
</tbody>
</table>

•https://partiql.org/
QLDB’s data model: e-commerce data model using Ion

Assume 3 tables

• Orders
• Customers
• Products

• CREATE TABLE Orders
• CREATE TABLE Customers
• CREATE TABLE Products
Ledger: Order-System

Nested document structure enables optimal queries and data access

```
INSERT INTO orders
{
  'order-id': 100056,
  'customer': {
    'document-id': 'some-value',
    'customer-id': 1000,
    'first-name': 'Mike',
    'last-name': 'Labib',
    'address': '126 Brampton Lane',
    'city': 'Chicago',
    'state': 'IL'
  },
  'order-date': '2019-04-30T',
  'order-details': {
    'item': {
      'document-id': 'some-value',
      'product-id': 346211,
      'product-description': '3 pair socks',
      'product-color': 'blue',
      'price': 15.00,
      'quantity': 2
    }
  }
}
```
Ledger: Order-System

Query

- SELECT o.order-details from orders o
- WHERE o.customer.customer-id = 1000
- AND o.order-id = 100056

Nested document query (customer within orders)

Result

- { item:
  - { 'product-id': 346211,
    'product-description': '3 pair socks',
    'product-color': 'blue',
    'price': 15.00,
    'quantity': 2
  }
}
Mapping constructs between RDBMS & QLDB

- **Relational**
  - Database
  - Table
  - Index
  - Table row
  - Column

- **QLDB**
  - Ledger
  - Table
  - Index
  - Amazon Ion Document
  - Document
  - Attribute

- **SQL**
- **QLDB SQL**

- **Audit Logs**
- **Journal**
Deeper look at 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 that links a document to its digest

\[
\begin{align*}
H_A & = \text{Hash(Tx}_A) \\
H_B & = \text{Hash(Tx}_B) \\
H_C & = \text{Hash(Tx}_C) \\
H_D & = \text{Hash(Tx}_D)
\end{align*}
\]

\[
\begin{align*}
\text{MERKLE ROOT} & = H_{\text{MERKLE ROOT}} \\
& = \text{Hash(H}_{A} + H_{B}) \\
& \vdots \\
& = \text{Hash(H}_{1} + H_{0})
\end{align*}
\]
How it works

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

• cars
<table>
<thead>
<tr>
<th>ID</th>
<th>Manufacturer</th>
<th>Model</th>
<th>Year</th>
<th>VIN</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• journal
  • INSERT cars $H(T_i)$
  • ID: 1
  • Manufacturer: Tesla
  • Model: Model S
  • Year: 2012
  • VIN: 123456789
  • Owner: Traci Russell
  • Metadata: {
    • Date: 07/16/2012
  }

• history()
<table>
<thead>
<tr>
<th>ID</th>
<th>Version</th>
<th>Start</th>
<th>Manufacturer</th>
<th>Model</th>
<th>Year</th>
<th>VIN</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• H
How it works

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

- INSERT INTO cars
  - ID: 1
  - Manufacturer: Tesla
  - Model: Model S
  - Year: 2012
  - VIN: 123456789
  - Owner: Traci Russell

- INSERT INTO history
  - ID: 1
  - Version: 1
  - Start: 07/16/2012
  - Manufacturer: Tesla
  - Model: Model S
  - Year: 2012
  - VIN: 123456789
  - Owner: Traci Russell

- INSERT INTO journal
  - ID: 1
  - Manufacturer: Tesla
  - Model: Model S
  - Year: 2012
  - VIN: 123456789
  - Owner: Traci Russell

- Metadata: {
  - Date: 07/16/2012
}
How it works

```sql
•UPDATE cars SET owner = 'Ronnie Nash' WHERE VIN = '123456789'
```

---

### cars

<table>
<thead>
<tr>
<th>ID</th>
<th>Manufacturer</th>
<th>Model</th>
<th>Year</th>
<th>VIN</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tesla</td>
<td>Model S</td>
<td>2012</td>
<td>123456789</td>
<td>Ronnie Nash</td>
</tr>
</tbody>
</table>

### history()

<table>
<thead>
<tr>
<th>ID</th>
<th>Version</th>
<th>Start</th>
<th>Manufacturer</th>
<th>Model</th>
<th>Year</th>
<th>VIN</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>7/16/2012</td>
<td>Tesla</td>
<td>Model S</td>
<td>2012</td>
<td>123456789</td>
<td>Traci Russell</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>8/03/2013</td>
<td>Tesla</td>
<td>Model S</td>
<td>2012</td>
<td>123456789</td>
<td>Ronnie Nash</td>
</tr>
</tbody>
</table>

### journal

```sql
•INSERT cars
  ID: 1
  Manufacturer: Tesla
  Model: Model S
  Year: 2012
  VIN: 123456789
  Owner: Traci Russell
  Metadata: {
    Date: 07/16/2012
  }
```

```sql
•UPDATE cars
  ID: 1
  Owner: Ronnie Nash
  Metadata: {
    Date: 08/03/2013
  }
```
How it works

- **DELETE FROM cars** WHERE **VIN** = '123456789'

### cars

<table>
<thead>
<tr>
<th>ID</th>
<th>Manufacturer</th>
<th>Model</th>
<th>Year</th>
<th>VIN</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tesla</td>
<td>Model S</td>
<td>2012</td>
<td>123456789</td>
<td>Ronnie Nash</td>
</tr>
</tbody>
</table>

### history()

<table>
<thead>
<tr>
<th>ID</th>
<th>Version</th>
<th>Start</th>
<th>Manufacturer</th>
<th>Model</th>
<th>Year</th>
<th>VIN</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>7/16/2012</td>
<td>Tesla</td>
<td>Model S</td>
<td>2012</td>
<td>123456789</td>
<td>Traci Russell</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>8/03/2013</td>
<td>Tesla</td>
<td>Model S</td>
<td>2012</td>
<td>123456789</td>
<td>Ronnie Nash</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>9/02/2016</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### journal

- **INSERT cars**
  - **ID:** 1
  - **Manufacturer:** Tesla
  - **Model:** Model S
  - **Year:** 2012
  - **VIN:** 123456789
  - **Owner:** Traci Russell
  - **Metadata:**
    - **Date:** 07/16/2012

- **UPDATE cars**
  - **ID:** 1
  - **Owner:** Ronnie Nash
  - **Metadata:**
    - **Date:** 08/03/2013

- **DELETE cars**
  - **ID:** 1
  - **Metadata:**
    - **Date:** 09/02/2016

- **DELETE FROM cars** WHERE **VIN** = '123456789'
A digest is a hash value at a point in time

- $H(T_1) = 2526f16306c819d651af075934170d2430d246d9ab98d975d28a83baded47ca7$
- $H(T_2) = 86a90e4166453d9423b84d47dcbd97c0e3099b1a1f0d7cfca6c191d8fd8994ff$
- $H(T_3) = ae2d64e562ec754ec3194c744e6c72c9fdaafffc6b559e0414d0e75bf96ca92ad$
Changing committed data breaks the chain

- \( H(T_1) = 2926f103c8c8198d951af0793347bd248d246d9ab58d975d26a83baced47ca7 \)
- \( H(T_2) = 25d0b44e6e8878151646ffcf1fca4eb85c3e4bf4baec212a9f67b5d5a8e01a \)
- \( H(T_3) = a90a9898c7e4b1a19c705b554afd9e0bf6539bb0346df19be362ff63001098 \)

- INSERT cars
  - ID: 1
  - Manufacturer: Tesla
  - Model: Model S
  - Year: 2012
  - VIN: 123456789
  - Owner: Tracy Russell
  - Metadata: {Date: 07/16/2012}

- UPDATE cars
  - ID: 1
  - Owner: Ronnie Nash
  - Metadata: {Date: 08/03/2013}

- DELETE cars
  - ID: 1
  - Metadata: {Date: 09/02/2016}
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
Blockchain Success Stories on AWS
What’s next?

To learn more about our services

Amazon Managed Blockchain
• Amazon Managed Blockchain: https://aws.amazon.com/managed-blockchain

Amazon QLDB
• Amazon QLDB webpage: https://aws.amazon.com/qldb

Amazon Blockchain Partners
• APN Blockchain Partners Spotlight: https://aws.amazon.com/partners/spotlights/blockchain-partner-spotlight/