Prague AWS User Group

Databases on AWS: Purpose-built DBs for all your applications needs

Vladimir Simek, Enterprise Solutions Architect
Prague AWS User Group
13th February 2019

AWS re:Invent

aws

How are you using your databases?

Two fundamental options

“Lift and shift” existing apps to the cloud

Quickly build new apps in the cloud

“Lift and shift” existing apps to the cloud

“Lift and shift” existing apps to the cloud

Quickly build new apps in the cloud

Amazon Relational Database Service (RDS)

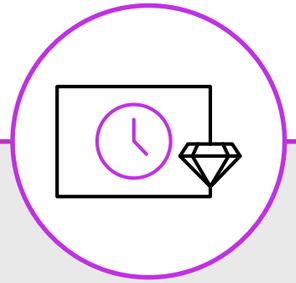
Managed relational database service with a choice of six popular database engines

Amazon Aurora · MySQL · PostgreSQL · MariaDB · Microsoft SQL Server · ORACLE

Easy to administer

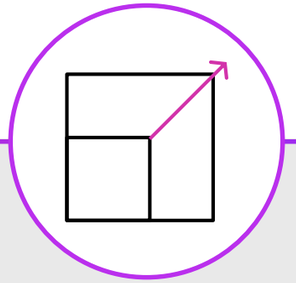
No need for infrastructure provisioning, installing, and maintaining DB software

Available and durable



Automatic Multi-AZ data replication; automated backup, snapshots, failover

Highly scalable



Scale database compute and storage with a few clicks with no app downtime

Fast and secure

SSD storage and guaranteed provisioned I/O; data encryption at rest and in transit

Amazon Aurora

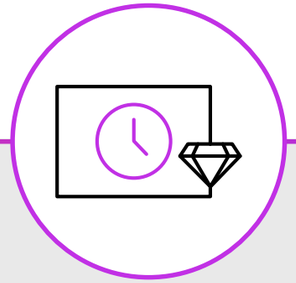
MySQL and PostgreSQL-compatible relational database built for the cloud

Performance and availability of commercial-grade databases at 1/10th the cost

Performance and scalability

5x throughput of standard MySQL and 3x of standard PostgreSQL; scale-out up to 15 read replicas

Availability and durability



Fault-tolerant, self-healing storage; six copies of data across three Availability Zones; continuous backup to Amazon S3

Highly secure

Network isolation, encryption at rest/transit

Fully managed

Managed by RDS: No hardware provisioning, software patching, setup, configuration, or backups

AWS Database Migration Service (AWS DMS)

MIGRATING DATABASES TO AWS

110,000+
databases migrated

Migrate between on-premises and AWS

Migrate between databases

Automated schema conversion

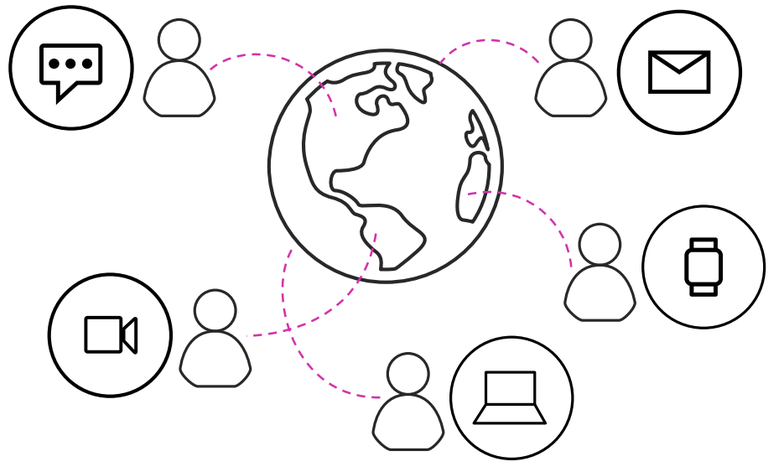
Data replication for
zero-downtime migration

Quickly build new apps in the cloud

“Lift and shift” existing apps to the cloud

Quickly build new apps in the cloud

Modern apps create new requirements



Ride hailing | Media streaming | Social media | Dating

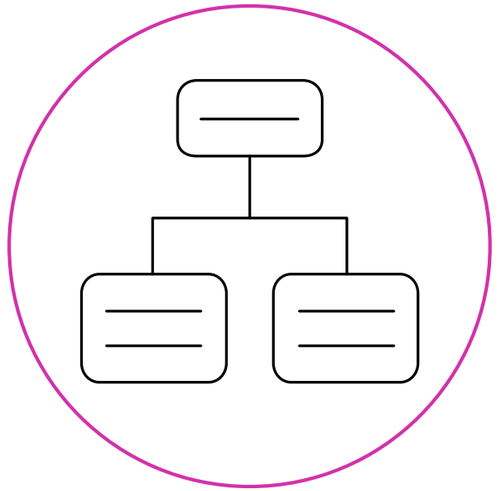
Users:	1 million+
Data volume:	TB–PB–EB
Locality:	Global
Performance:	Milliseconds–microseconds
Request rate:	Millions
Access:	Web, mobile, IoT, devices
Scale:	Up-down, Out-in
Economics:	Pay for what you use
Developer access:	No assembly required

One size doesn’t fit all

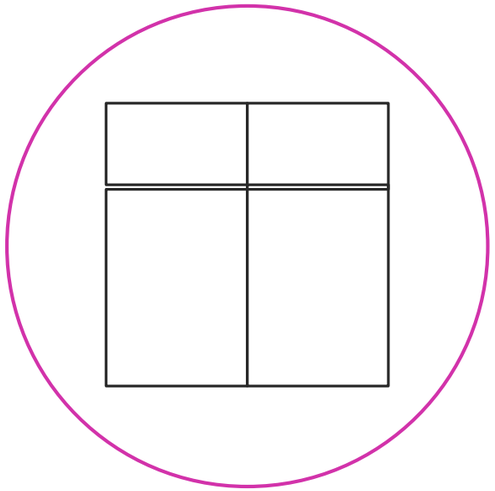
Common data categories and use cases

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

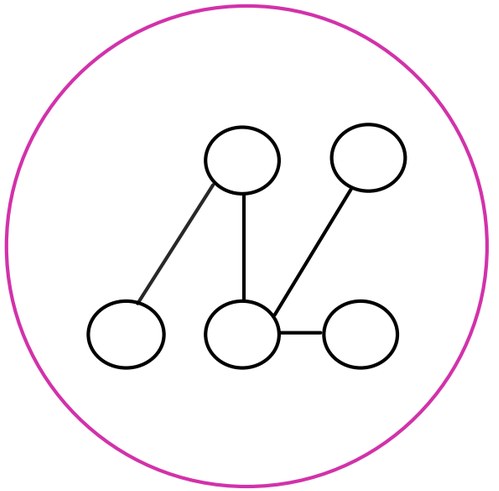
Let’s take a closer look at…



Relational



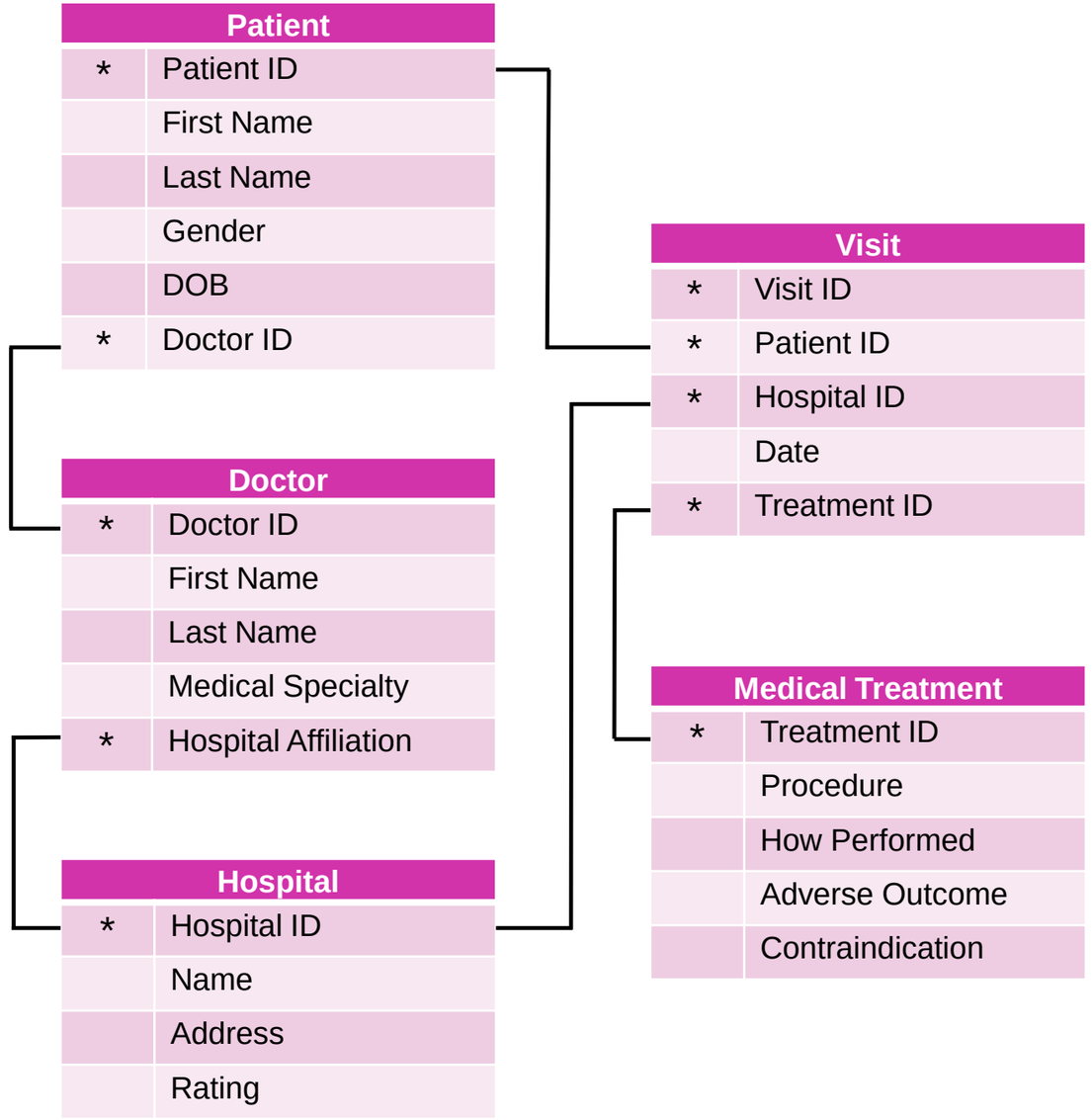
Key-value



Graph

Relational data

- Divide data among tables
- Highly structured
- Relationships established via keys enforced by the system
- Data accuracy and consistency

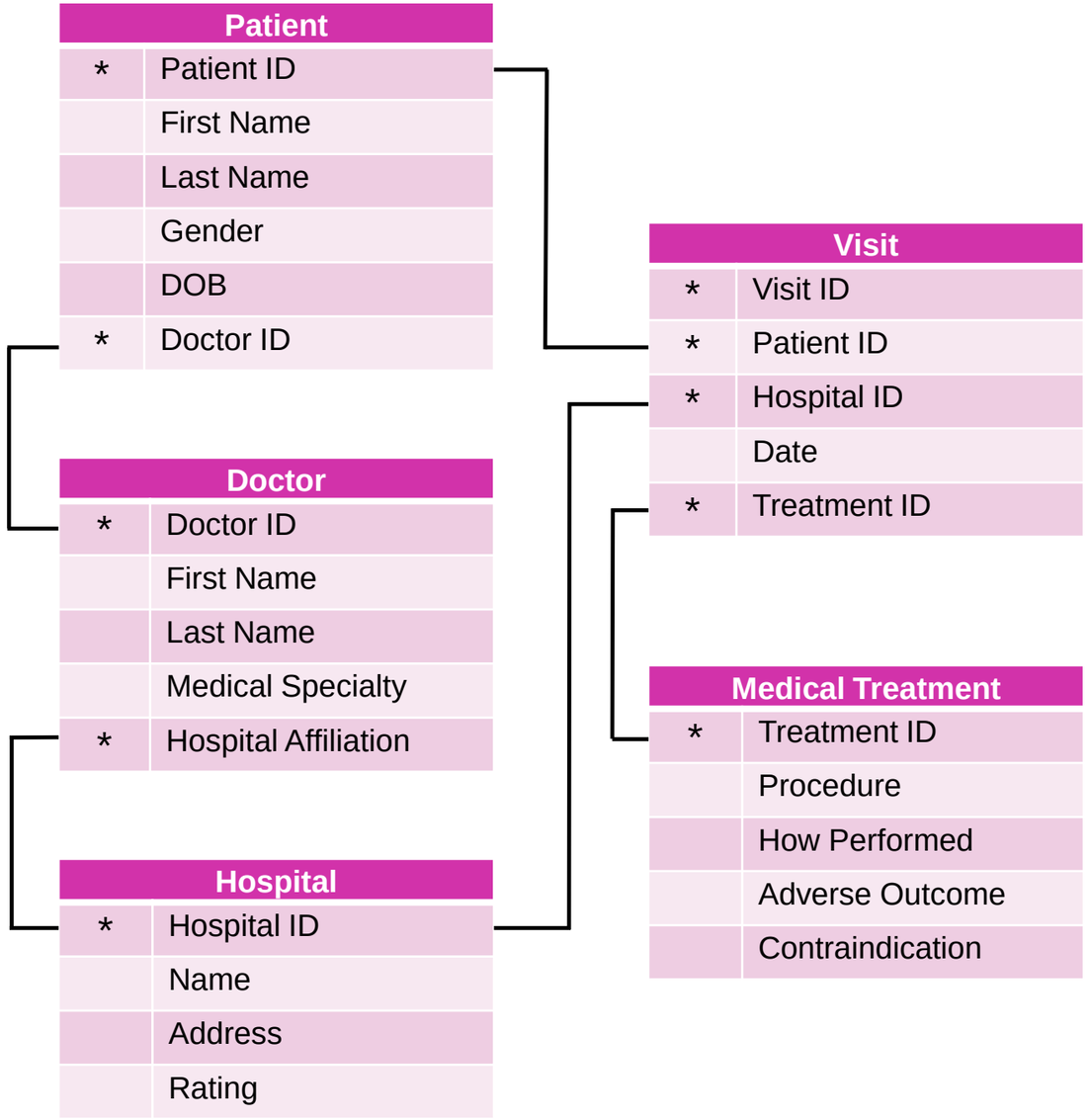


Relational use case

// Number of patients with visits for each doctor affiliated with hospital

// Doctors affiliated with Mercy hospital and visited last week

```
SELECT
SELECT    d.first_name, d.last_name
            d.first_name, d.last_name, count(*)
FROM
FROM      doctor as d,
            visit as v,
          hospital as h,
            hospital as h,
            doctor as d
WHERE
WHERE     d.hospital = h.hospital_id
AND       h.name = 'Mercy'
            h.hospital_id = d.hospital_id
            AND h.hospital_id = d.hospital
            AND v.t_date > date_trunc('week',
            CURRENT_TIMESTAMP - interval '1 week')
GROUP BY
            d.first_name, d.last_name;
```



Key-value data

- Simple key value pairs
- Partitioned by keys
- Resilient to failure
- High throughput, low-latency reads and writes
- Consistent performance at scale

```
PUT {
  TableName:"Gamers",
   Item: {
   "GamerTag":"Hammer57",
   "Level":21,
   "Points":4050,
   "Score":483610,
   "Plays":1722
    } }
```

```
GET {
  TableName:"Gamers",
  Key: {
    "GamerTag":"Hammer57“,
    “ProjectionExpression“:”Points”
     } }
```

Gamers				
Primary Key	**Attributes**			
GamerTag	**Level**	**Points**	**High Score**	**Plays**
Hammer57	21	4050	483610	1722
FluffyDuffy	5	1123	10863	43
Lol777313	14	3075	380500	1307
Jam22Jam	20	3986	478658	1694
ButterZZ_55	7	1530	12547	66
…	…	…	…	…

Key-value use case

// Status of Hammer57

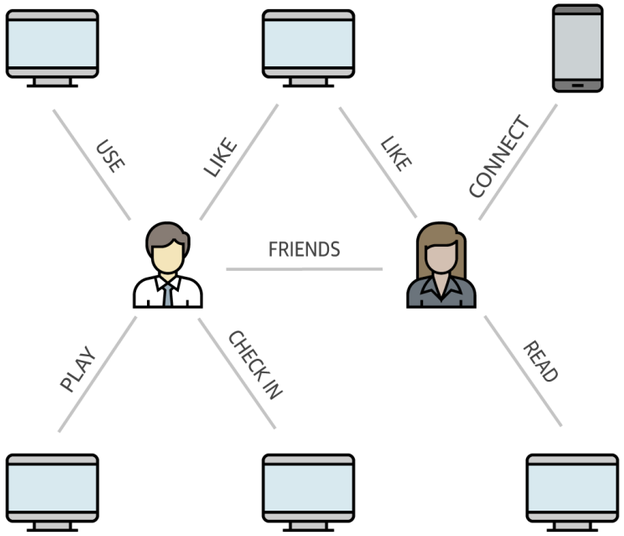
```
GET {
  TableName:"Gamers",
  Key: {
    "GamerTag":"Hammer57",
    "Type":"Status" } }
```

// Return all Hammer57

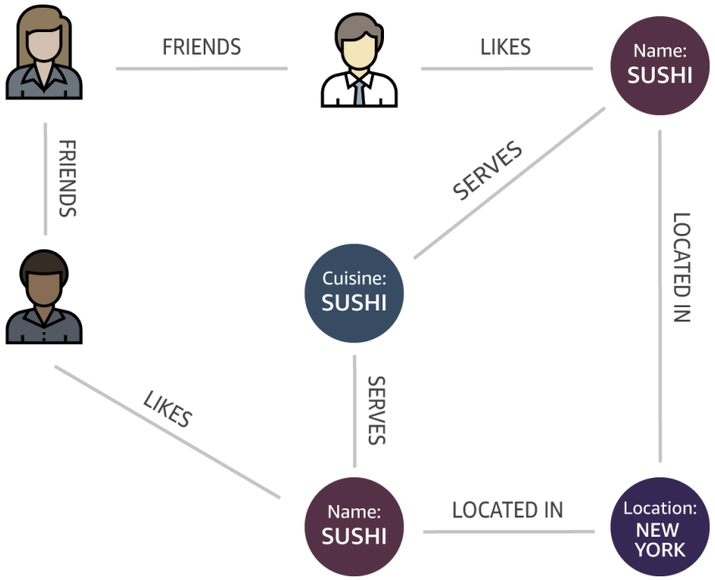
```
QUERY {
  TableName:“Gamers”,
  KeyConditionExpression:"GamerTag = :a",
  ExpressionAttributeValues: {
      ":a”:”Hammer57” } }
```

Gamers				
Primary Key		**Attributes**		
Gamer Tag	**Type**			
Hammer57	Rank	Level	Points	Tier
		87	4050	Elite
	Status	Health	Progress	
		90	30	
	Weapon	Class	Damage	Range
		Taser	87%	50
FluffyDuffy	Rank	Level	Points	Tier
		5	1072	Trainee
	Status	Health	Progress	
		37	8	

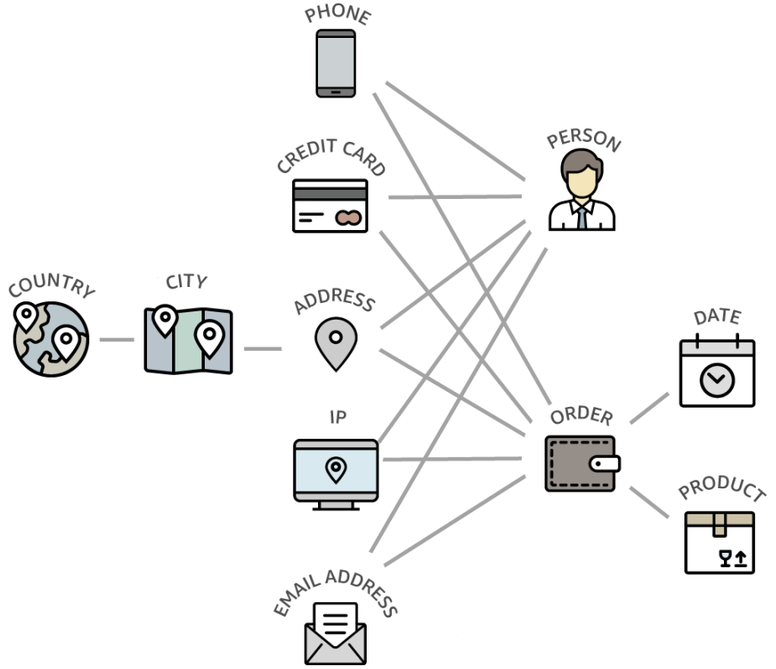
Relationships enable new applications



Social networks



Restaurant recommendations



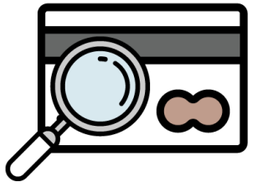
Retail fraud detection

Use cases for highly connected data

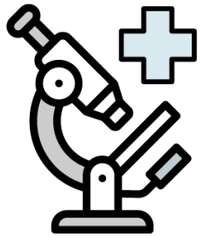
Social networking

Recommendations

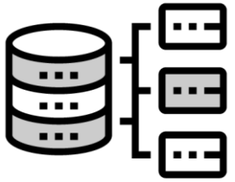
Knowledge graphs



Fraud detection



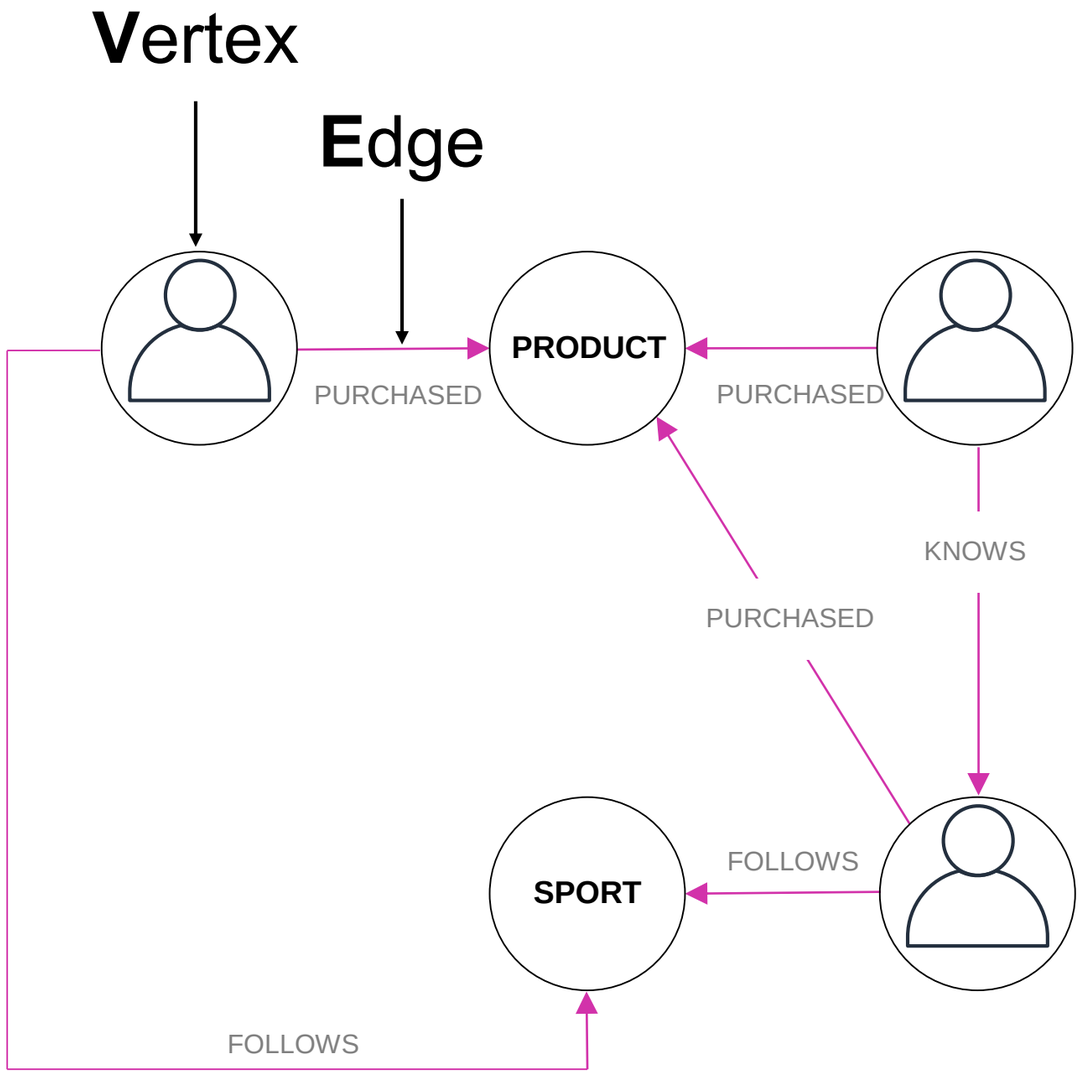
Life Sciences



Network & IT operations

Graph data

- Relationships are first-class objects
- **V**ertices connected by **E**dges



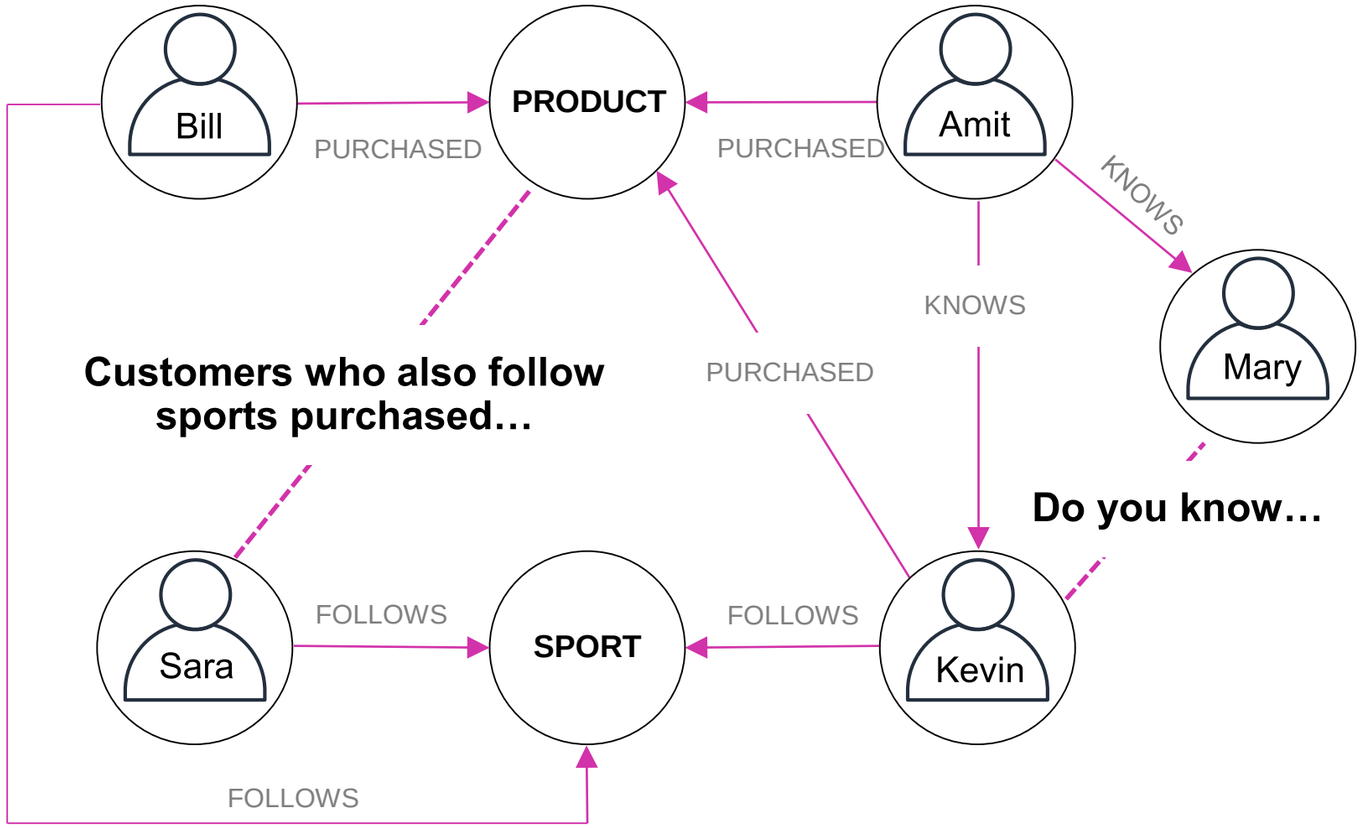
Graph use case

// Product recommendation to a user

```
gremlin> g.V().has('name','sara').as('customer').out('follows').in('follows').out('purchased')
where(neq('customer')).dedup().by('name').properties('name')
```

// Identify a friend in common and make a recommendation

```
gremlin> g.V().has('name','mary').as('start').
both('knows').both('knows').
where(neq('start')).
dedup().by('name').properties('name')
```



Airbnb uses different databases based on the purpose

User search history: **Amazon DynamoDB**

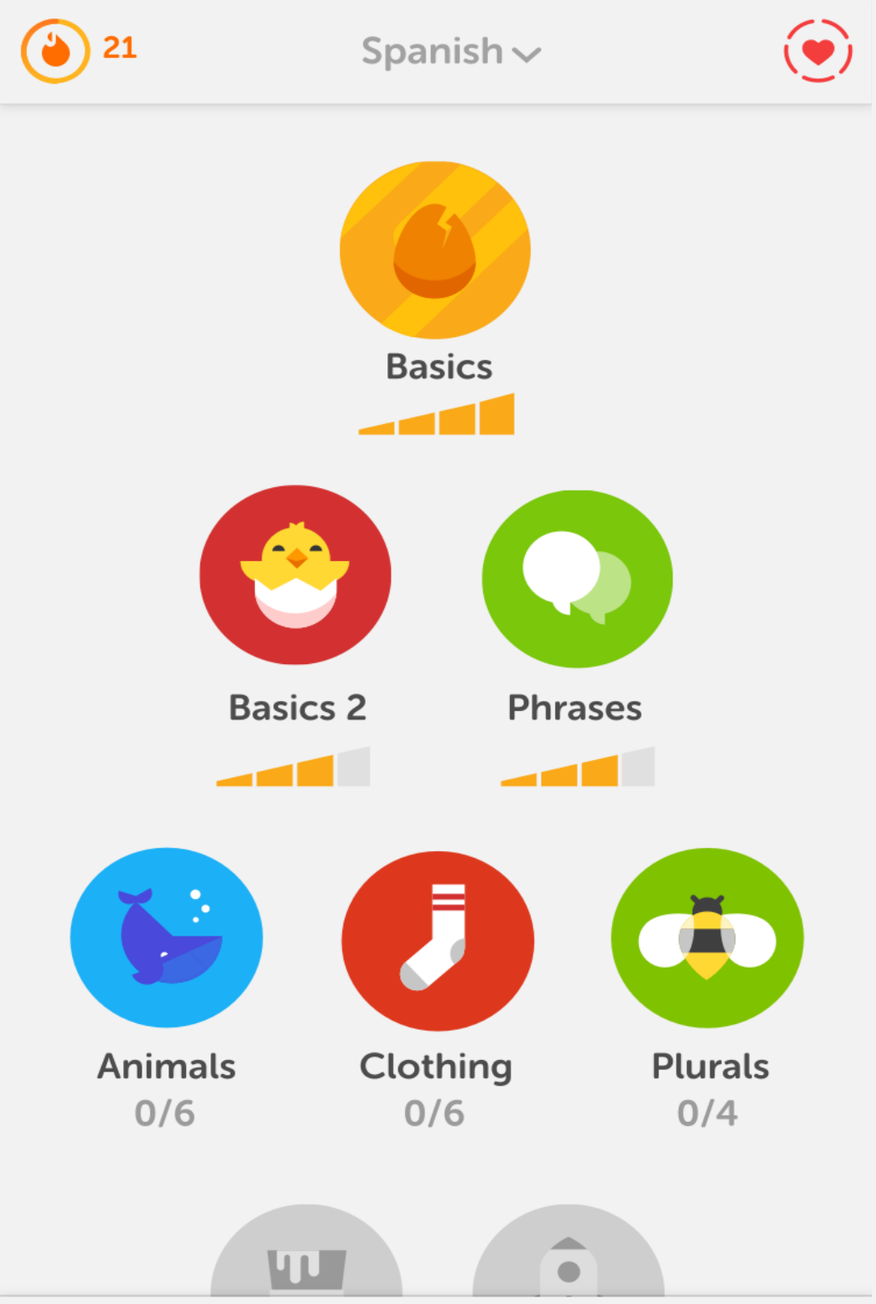
- Massive data volume
- Need quick lookups for personalized search

Session state: **Amazon ElastiCache**

- In-memory store for submillisecond site rendering

Relational data: **Amazon RDS**

- Referential integrity
- Primary transactional database



duolingo

300M total users
7B exercises per month

CHALLENGE

Wanted to enable anyone to learn a language for free.

SOLUTION

Purpose-built databases from AWS:

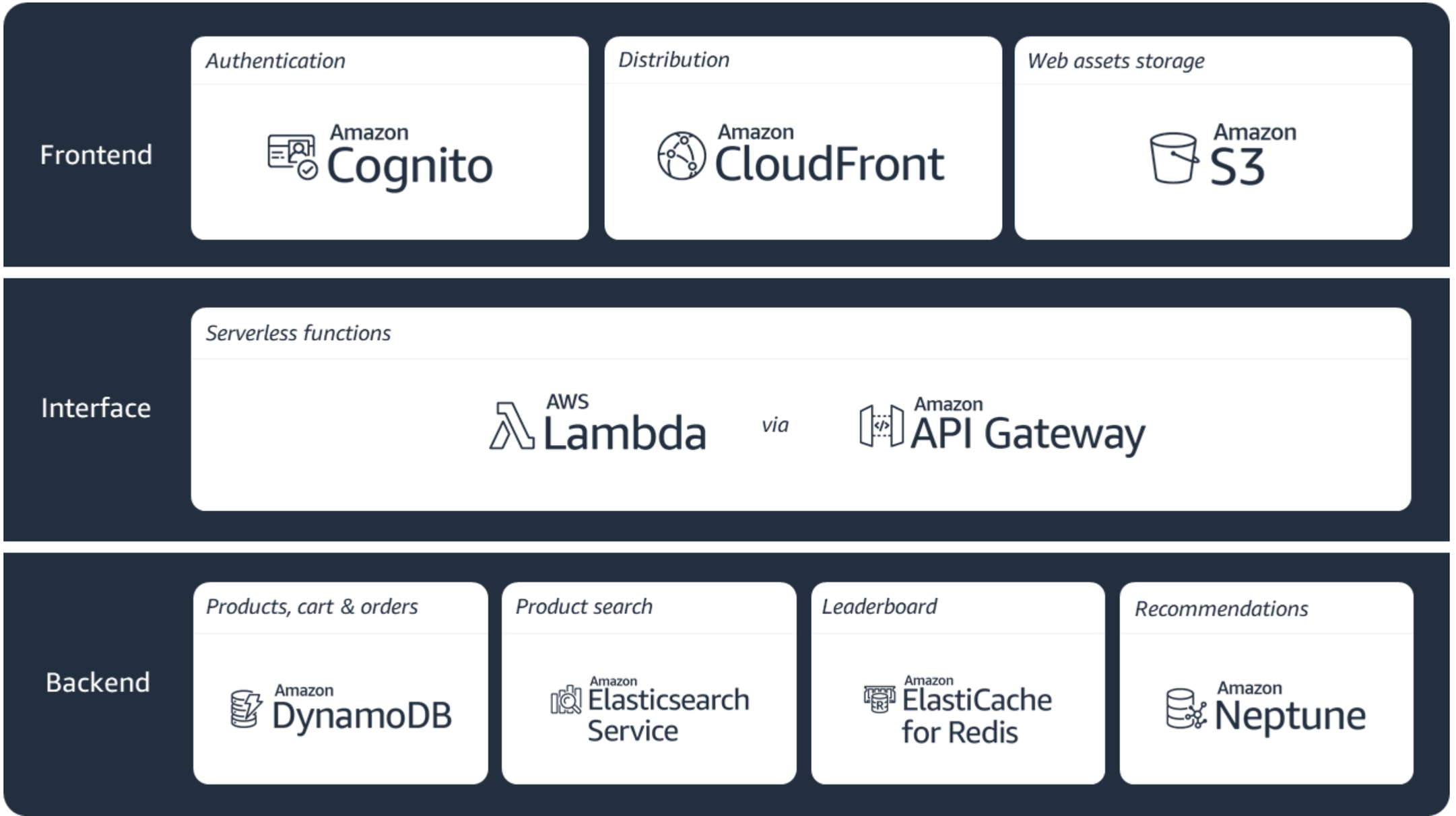
- **DynamoDB**: 31B items tracking which language exercises completed
- **Aurora**: primary transactional database for user data
- **ElastiCache**: instant access to common words and phrases

Result:

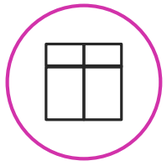
More people learning a language on Duolingo than entire US school system

Demo

Demo App Architecture



Retail demo application

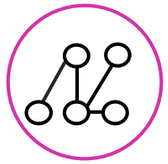


Key-value

High throughput, Low-latency reads and writes, endless scale

Shopping cart, user profile

DynamoDB

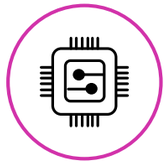


Graph

Quickly and easily create and navigate relationships between data

Product recommendation

Amazon Neptune

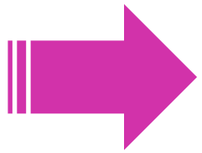


In-memory

Query by key with microsecond latency

Product leaderboard

ElastiCache



Search

Indexing and searching semistructured logs and data

Product search

Amazon Elasticsearch Service

Demo application:

1. Available today

2. On GitHub: /aws-samples/aws-bookstore-demo-app

3. One click CloudFormation deployment

aws re:Invent

aws

Ledger database

AWS re:Invent

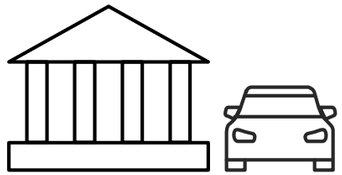
aws

Common customer use cases

Ledgers with centralized control

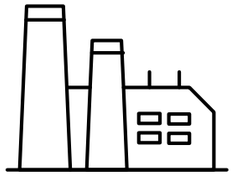
Healthcare

Verify and track hospital equipment inventory



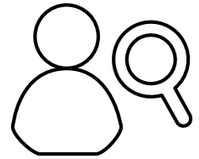
Government

Track vehicle title history



Manufacturers

Track distribution of a recalled product



HR & Payroll

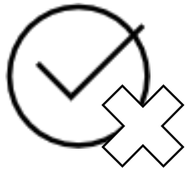
Track changes to an individual's profile

Challenges with building ledgers

RDBMS - audit tables

Hard to build

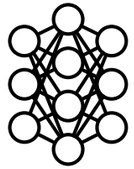
Custom audit functionality using triggers or stored procedures



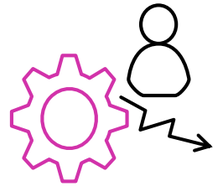
Impossible to verify

No way to verify changes made to data by sys admins

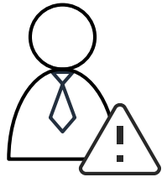
Blockchain



Adds unnecessary complexity

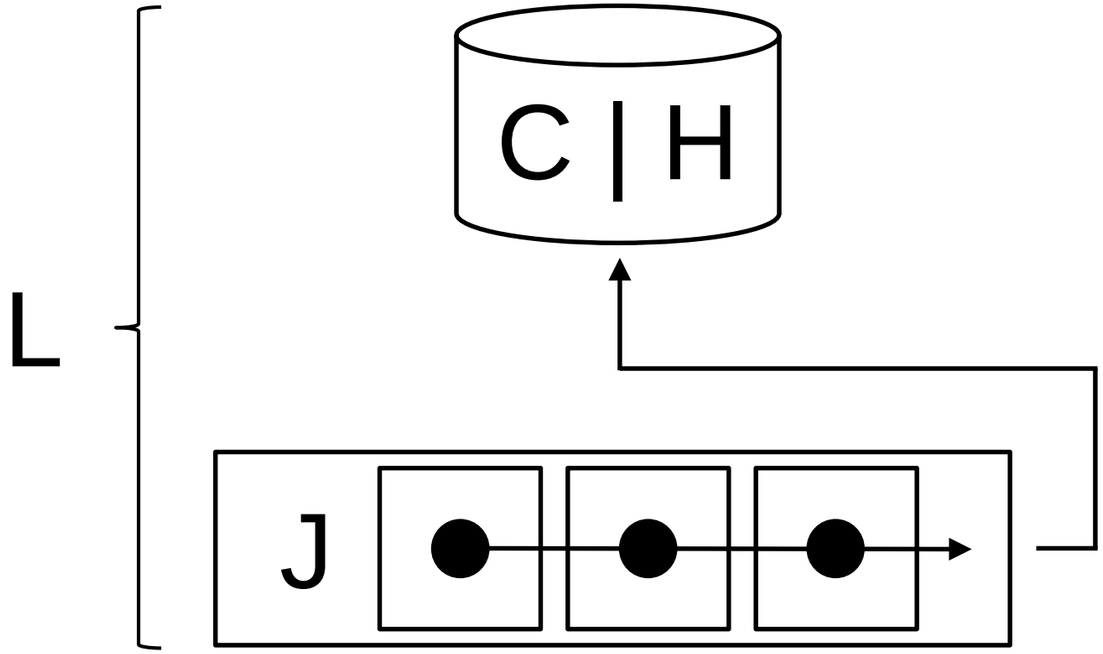


Hard to use and slow



Difficult to maintain

Ledger database concepts



L — Ledger database

J — Journal

C | H — Current | History

Ledger comprises { Current | History, Journal }

Journal determines → Current | History

How it works

How it works

C

current.cars

ID	Manufacturer	Model	Year	VIN	Owner

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner

J

How it works

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

C

current.cars

ID	Manufacturer	Model	Year	VIN	Owner

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner

J

How it works

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

C

current.cars

ID	Manufacturer	Model	Year	VIN	Owner

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner

J

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

Metadata: {
Date:07/16/2012
}

How it works

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

C

current.cars

ID	Manufacturer	Model	Year	VIN	Owner

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner

J

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

Metadata: {
Date:07/16/2012
}

H (T_1)

How it works

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

C

current.cars

ID	Manufacturer	Model	Year	VIN	Owner
1	Tesla	Model S	2012	123456789	Traci Russell

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner

J

INSERT cars
H (T_1)
ID:1
Manufacturer: Tesla
Model: Model S
Year: 2012
VIN: 123456789
Owner: Traci Russell

Metadata: {
Date:07/16/2012
}

How it works

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

C

current.cars

ID	Manufacturer	Model	Year	VIN	Owner
1	Tesla	Model S	2012	123456789	Traci Russell

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner
1	1	07/16/2012	Tesla	Model S	2012	123456789	Traci Russell

J

INSERT cars
H (T_1)
ID:1
Manufacturer: Tesla
Model: Model S
Year: 2012
VIN: 123456789
Owner: Traci Russell

Metadata: {
Date:07/16/2012
}

How it works

C

current.cars

ID	Manufacturer	Model	Year	VIN	Owner
1	Tesla	Model S	2012	123456789	Traci Russell

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner
1	1	07/16/2012	Tesla	Model S	2012	123456789	Traci Russell

J

INSERT cars
H (T_1)
ID:1
Manufacturer: Tesla
Model: Model S
Year: 2012
VIN: 123456789
Owner: Traci Russell

Metadata: {
Date:07/16/2012
}

How it works

```
FROM cars WHERE VIN = '123456789' UPDATE owner = 'Ronnie Nash'
```

C

current.cars

ID	Manufacturer	Model	Year	VIN	Owner
1	Tesla	Model S	2012	123456789	Traci Russell

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner
1	1	07/16/2012	Tesla	Model S	2012	123456789	Traci Russell

J

INSERT cars
H (T_1)
ID:1
Manufacturer: Tesla
Model: Model S
Year: 2012
VIN: 123456789
Owner: Traci Russell

Metadata: {
Date:07/16/2012
}

How it works

```
FROM cars WHERE VIN = '123456789' UPDATE owner = 'Ronnie Nash'
```

C

current.cars

ID	Manufacturer	Model	Year	VIN	Owner
1	Tesla	Model S	2012	123456789	Traci Russell

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner
1	1	07/16/2012	Tesla	Model S	2012	123456789	Traci Russell

J

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

Metadata: {
Date:07/16/2012
}

H (T_1)

UPDATE cars
ID:1
Owner: Ronnie Nash

Metadata: {
Date:08/03/2013
}

H (T_2)

How it works

```
FROM cars WHERE VIN = '123456789' UPDATE owner = 'Ronnie Nash'
```

C

current.cars

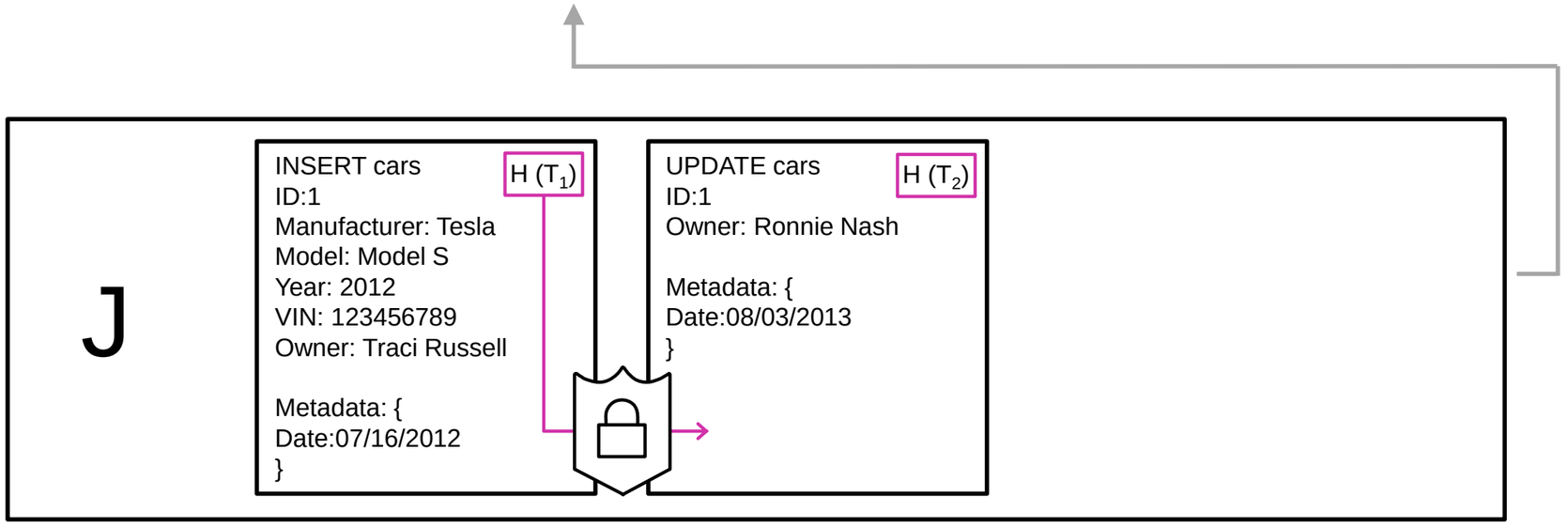
ID	Manufacturer	Model	Year	VIN	Owner
1	Tesla	Model S	2012	123456789	Traci Russell

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner
1	1	07/16/2012	Tesla	Model S	2012	123456789	Traci Russell

J



How it works

```
FROM cars WHERE VIN = '123456789' UPDATE owner = 'Ronnie Nash'
```

C

current.cars

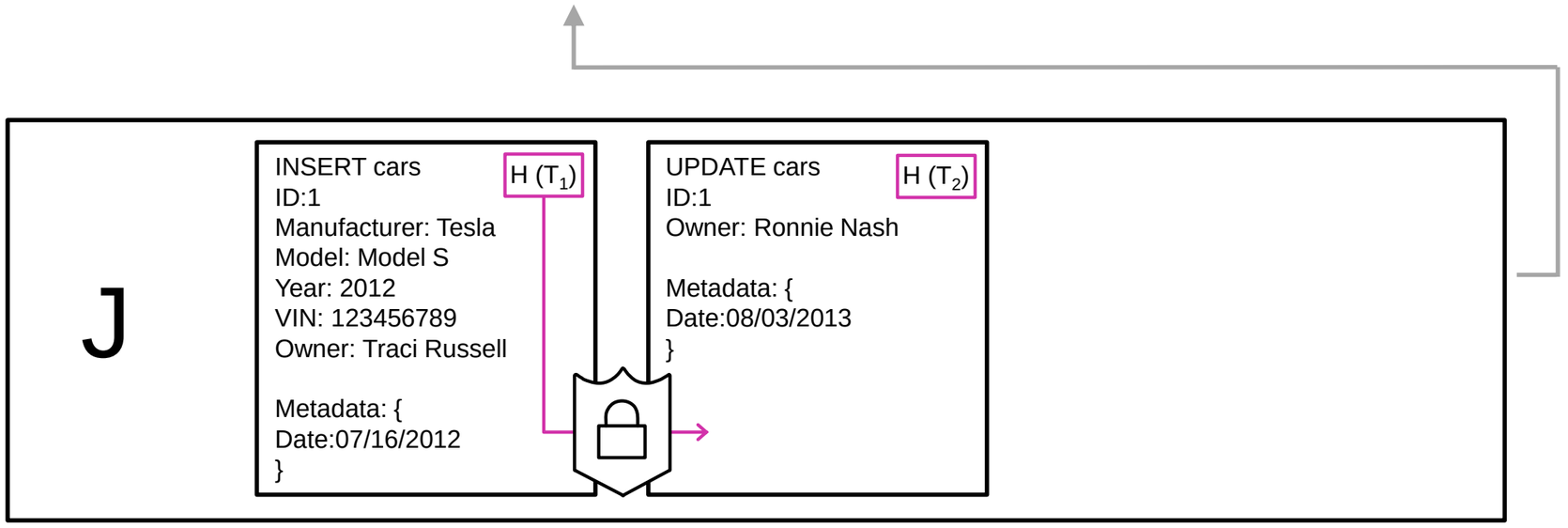
ID	Manufacturer	Model	Year	VIN	Owner
1	Tesla	Model S	2012	123456789	Ronnie Nash

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner
1	1	07/16/2012	Tesla	Model S	2012	123456789	Traci Russell

J



How it works

```
FROM cars WHERE VIN = '123456789' UPDATE owner = 'Ronnie Nash'
```

C

current.cars

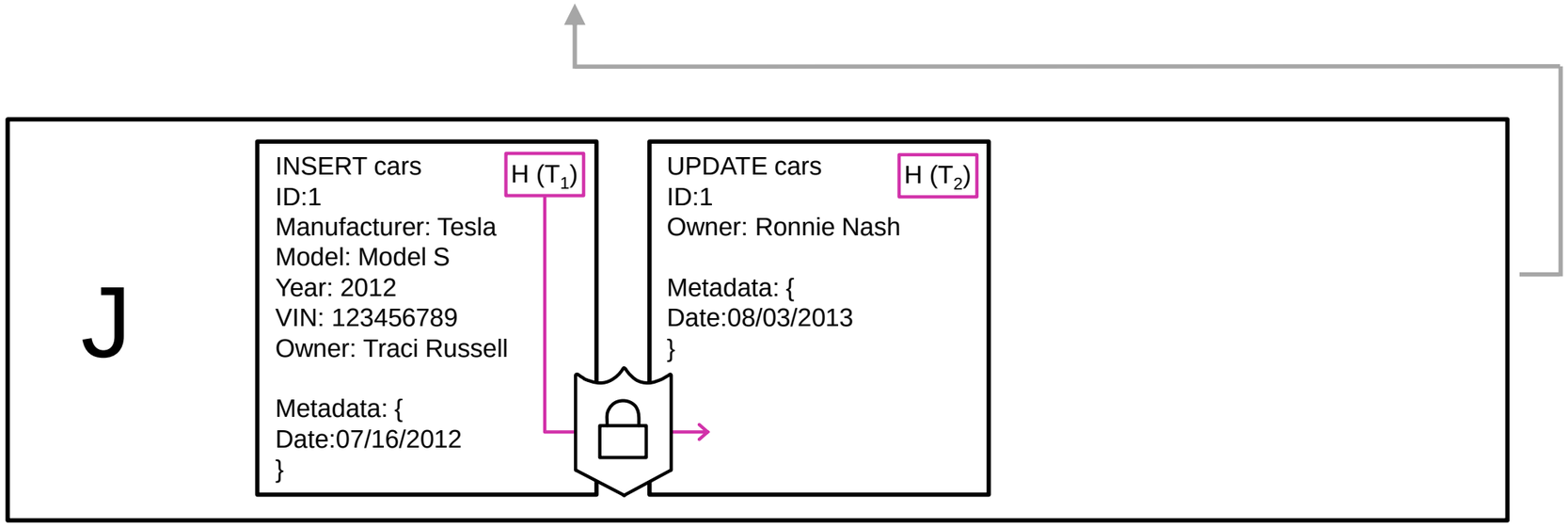
ID	Manufacturer	Model	Year	VIN	Owner
1	Tesla	Model S	2012	123456789	Ronnie Nash

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner
1	1	07/16/2012	Tesla	Model S	2012	123456789	Traci Russell
1	2	08/03/2013	Tesla	Model S	2012	123456789	Ronnie Nash

J



How it works

C

current.cars

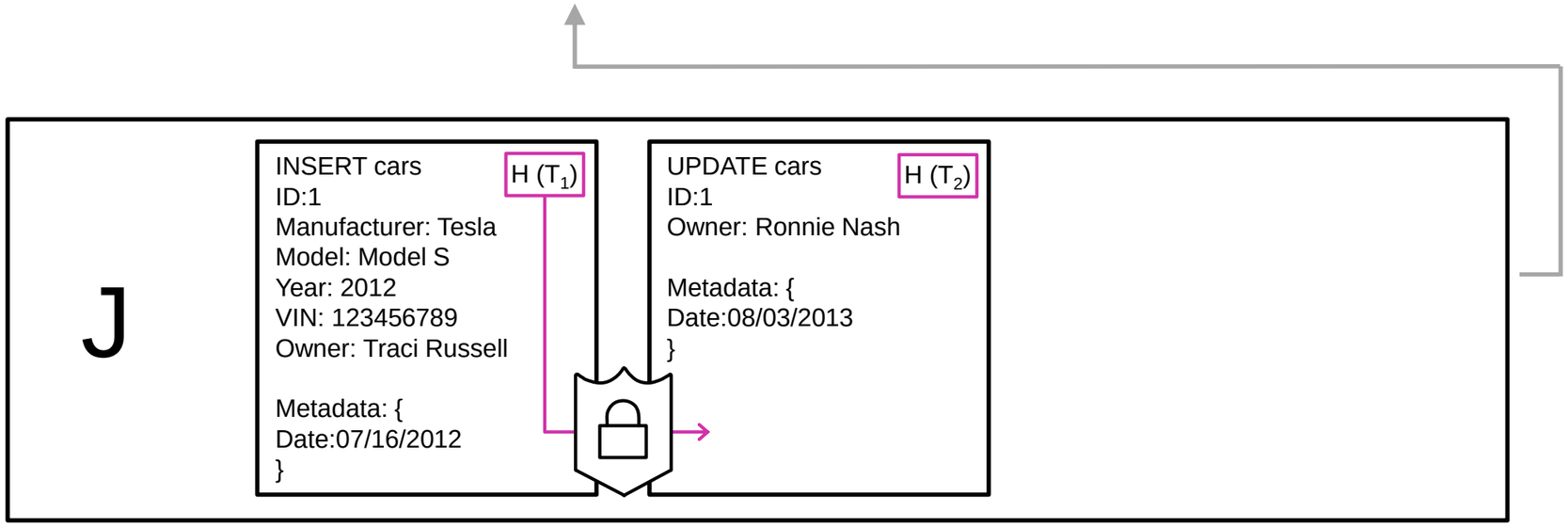
ID	Manufacturer	Model	Year	VIN	Owner
1	Tesla	Model S	2012	123456789	Ronnie Nash

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner
1	1	07/16/2012	Tesla	Model S	2012	123456789	Traci Russell
1	2	08/03/2013	Tesla	Model S	2012	123456789	Ronnie Nash

J



How it works

```
FROM cars WHERE VIN = '123456789' UPDATE owner = 'Elmer Hubbard'
```

C

current.cars

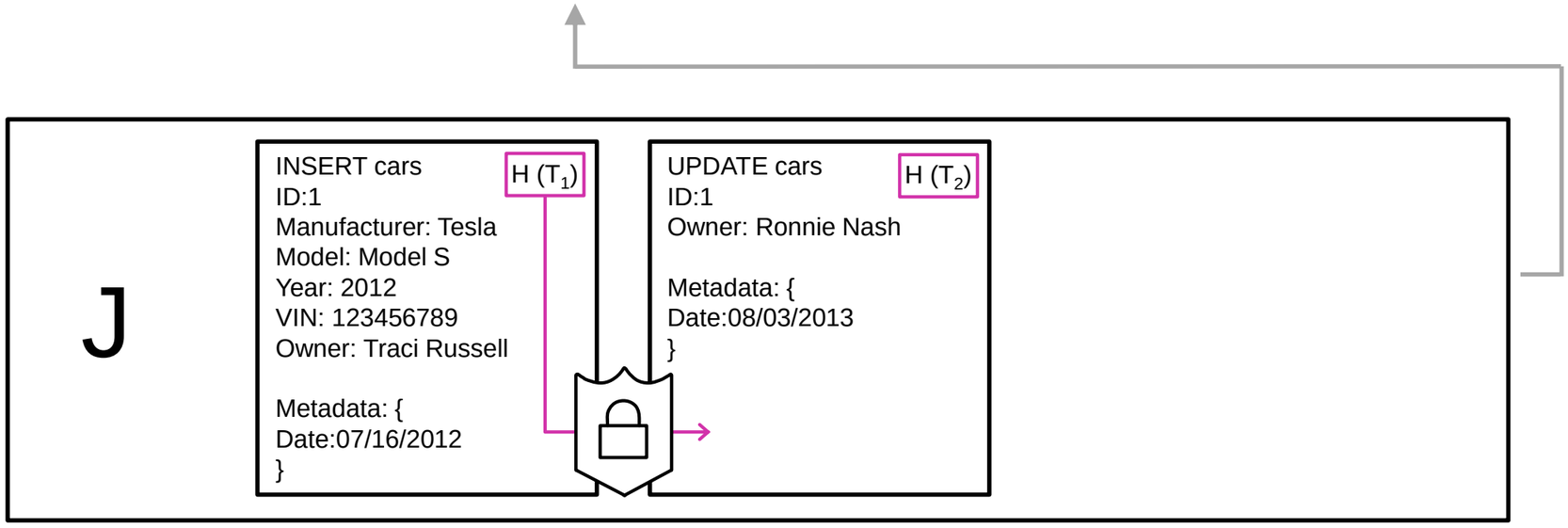
ID	Manufacturer	Model	Year	VIN	Owner
1	Tesla	Model S	2012	123456789	Ronnie Nash

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner
1	1	07/16/2012	Tesla	Model S	2012	123456789	Traci Russell
1	2	08/03/2013	Tesla	Model S	2012	123456789	Ronnie Nash

J



How it works

```
FROM cars WHERE VIN = '123456789' UPDATE owner = 'Elmer Hubbard'
```

C

current.cars

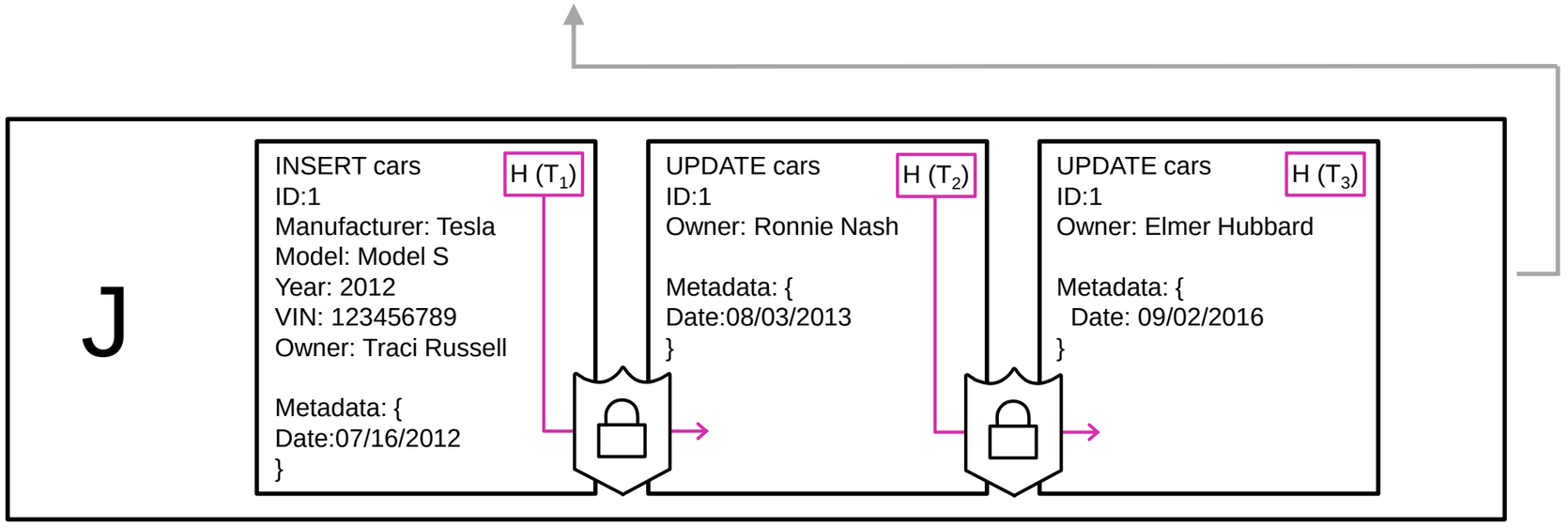
ID	Manufacturer	Model	Year	VIN	Owner
1	Tesla	Model S	2012	123456789	Ronnie Nash

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner
1	1	07/16/2012	Tesla	Model S	2012	123456789	Traci Russell
1	2	08/03/2013	Tesla	Model S	2012	123456789	Ronnie Nash

J



How it works

```
FROM cars WHERE VIN = '123456789' UPDATE owner = 'Elmer Hubbard'
```

C

current.cars

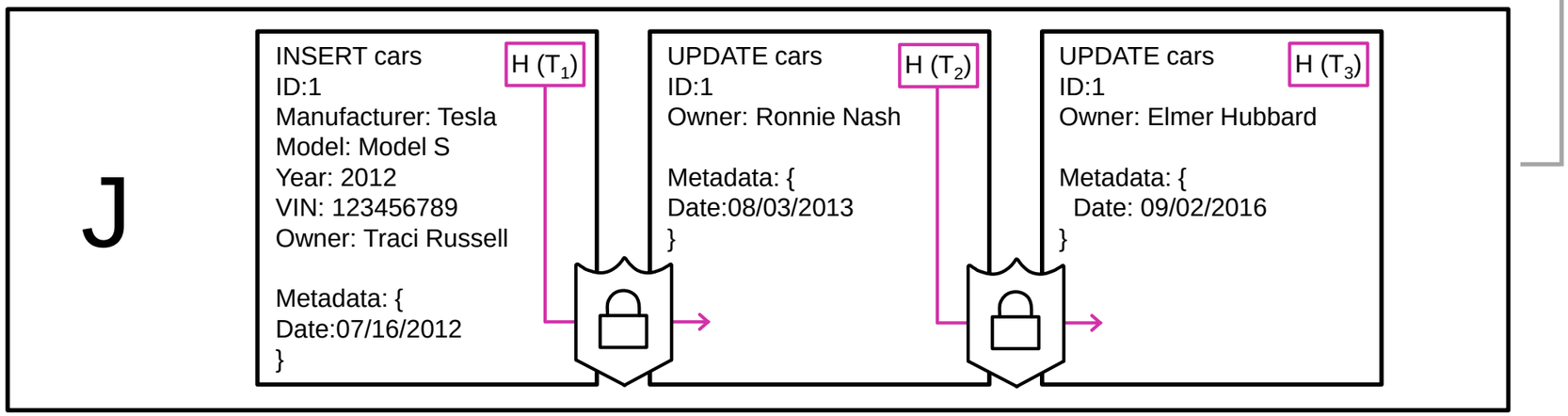
ID	Manufacturer	Model	Year	VIN	Owner
1	Tesla	Model S	2012	123456789	Elmer Hubbard

H

history.cars

ID	Version	Start	Manufacturer	Model	Year	VIN	Owner
1	1	07/16/2012	Tesla	Model S	2012	123456789	Traci Russell
1	2	08/03/2013	Tesla	Model S	2012	123456789	Ronnie Nash
1	3	09/02/2016	Tesla	Model S	2012	123456789	Elmer Hubbard

J



Amazon Quantum Ledger Database (QLDB) (Preview)

NEW!

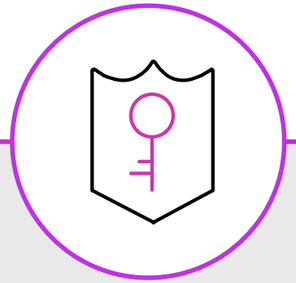
Fully managed ledger database

Track and verify history of all changes made to your application's data

Immutable

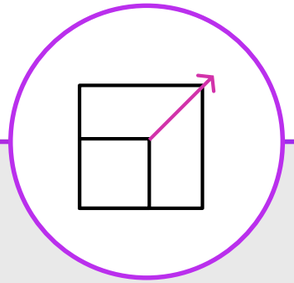
Maintains a sequenced record of all changes to your data, which cannot be deleted or modified; you have the ability to query and analyze the full history

Cryptographically verifiable



Uses cryptography to generate a secure output file of your data's history

Highly scalable



Executes 2–3X as many transactions than ledgers in common blockchain frameworks

Easy to use

Easy to use, letting you use familiar database capabilities like SQL APIs for querying the data

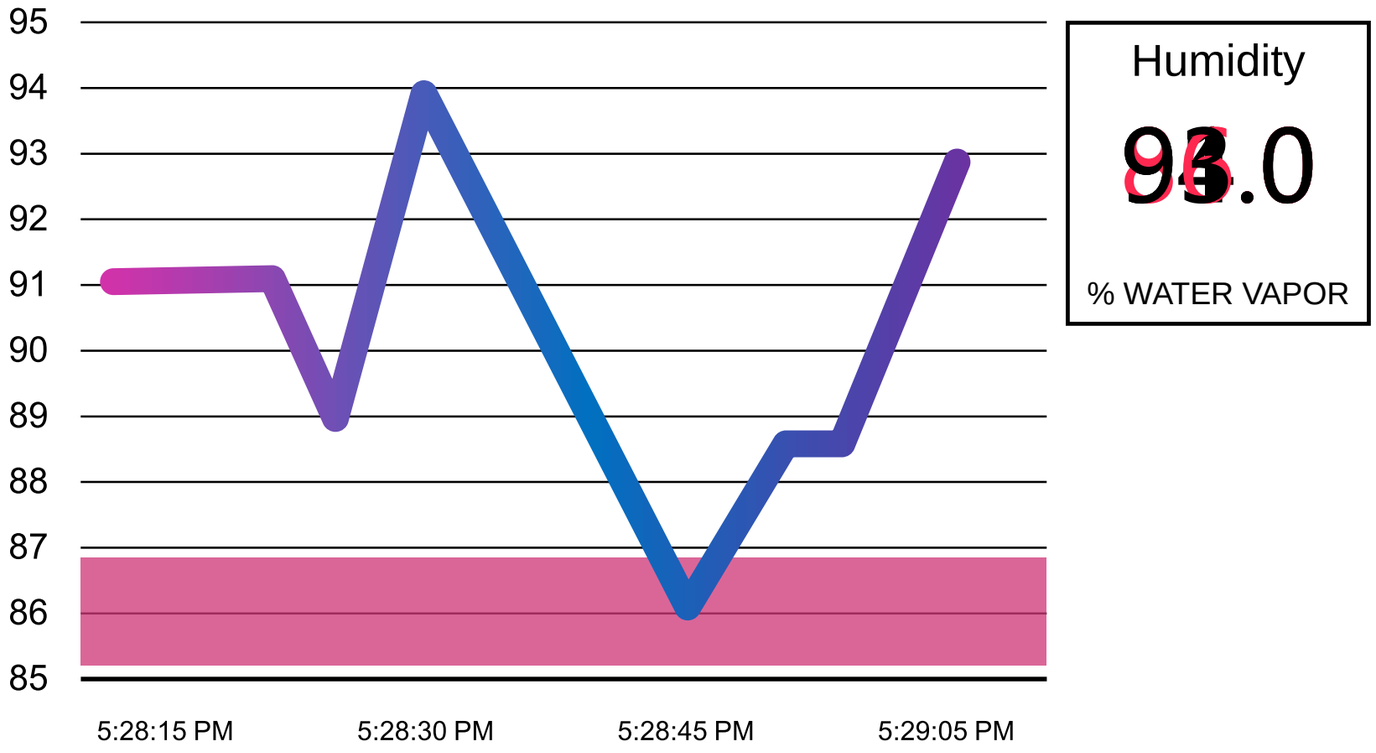
Time-series database

Time-series data

What is time-series data?

What is special about a time-series database?

Time-series use case

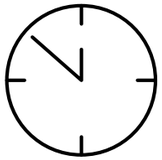


1. Application events
2. IoT Sensor Readings
3. DevOps data

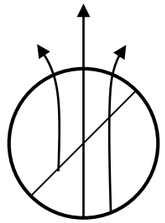
Building with time-series data is challenging

Relational databases

Unnatural for time-series data

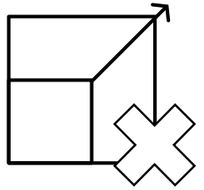


Inefficient time-series data processing

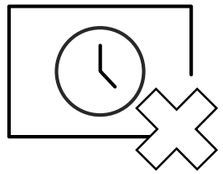


Rigid schema inflexible for fast moving time-series data

Existing time-series databases



Difficult to scale



Difficult to maintain high availability

Limited data lifecycle management

Amazon Timestream (sign up for the prev NEW!

Fast, scalable, fully managed time-series database

1,000x faster and 1/10th the cost of relational databases

Collect data at the rate of millions of inserts per second (10M/second)

Trillions of daily events

Adaptive query processing engine maintains steady, predictable performance

Time-series analytics

Built-in functions for interpolation, smoothing, and approximation

Serverless

Automated setup, configuration, server provisioning, software patching

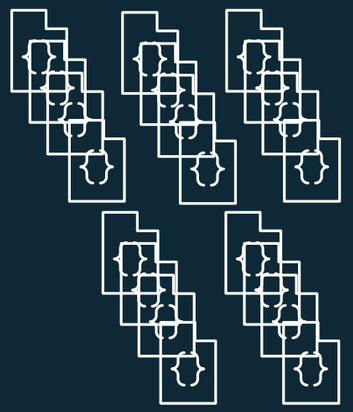
Document database

What is a document database?

Non-relational database that stores semi-structured data as JSON documents

Flexible schema – each document can contain different fields

Semi-structured data sets



Highly scalable

Modern applications

aws

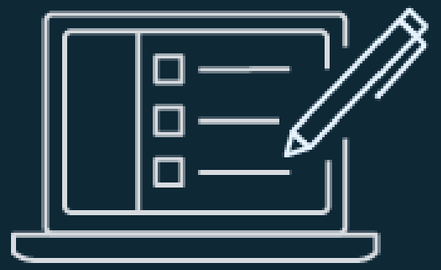
Why use a document database?

The JSON document model maps naturally to application data

Each document can have a different data structure and is independent of other documents

Index on any key in a document, and run ad hoc and aggregation queries across your data set

Use cases for document databases



Content Management

Mobile

Personalization

Catalog

Retail and Marketing

User profiles

aws

MongoDB – popular document database engine

Open-source

Over 40 million downloads

Easy for developers to get started

MongoDB API is rich and powerful

Easy to programmatically query

Rank			DBMS
Jan 2019	Dec 2018	Jan 2018	
1.	1.	1.	Oracle
2.	2.	2.	MySQL
3.	3.	3.	Microsoft SQL Server
4.	4.	4.	PostgreSQL
5.	5.	5.	MongoDB
6.	6.	6.	IBM Db2
7.	7.	↑ 9.	Redis
8.	8.	↑ 10.	Elasticsearch
9.	9.	↓ 7.	Microsoft Access
10.	10.	↑ 11.	SQLite

Source: http://db-engines.com

aws

Running MongoDB is difficult……

Expensive

Experts only

Time to market

What if you could create and scale MongoDB compatible database clusters in minutes?

Introducing:

Amazon DocumentDB

aws

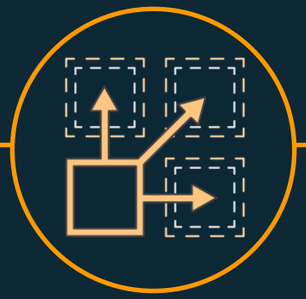
Fast

Fast, reliable, and fully-managed MongoDB-compatible database service

Fast

Millions of requests per second with millisecond latency; scale-out up to
15 read replicas

More throughput



Separation of storage and compute offloads replication, providing 2x the throughput of current MongoDB managed services

Automatic scaling

DocumentDB will automatically grow the size of your storage volume as your cluster storage needs grow.

Analytics

Scale-up instances in minutes for analytical queries and scale them down at the end of the day

Reliable

Fast, reliable, and fully-managed MongoDB-compatible database service

Automatic failure recovery

Failing instances are automatically detected and recovered; no cache warm-up needed

Automatic failover

Replicas are automatically promoted to primary

Point-in-time restore

Automated backups are stored in Amazon S3, designed for 99.999999999% durability

Durable

Data is replicated six-ways across three AZs

Fully-managed

Fast, reliable, and fully-managed MongoDB-compatible database service

Automatic patching

Up-to-date with the latest patches

Quick start

Provision production-ready clusters in minutes

Monitoring

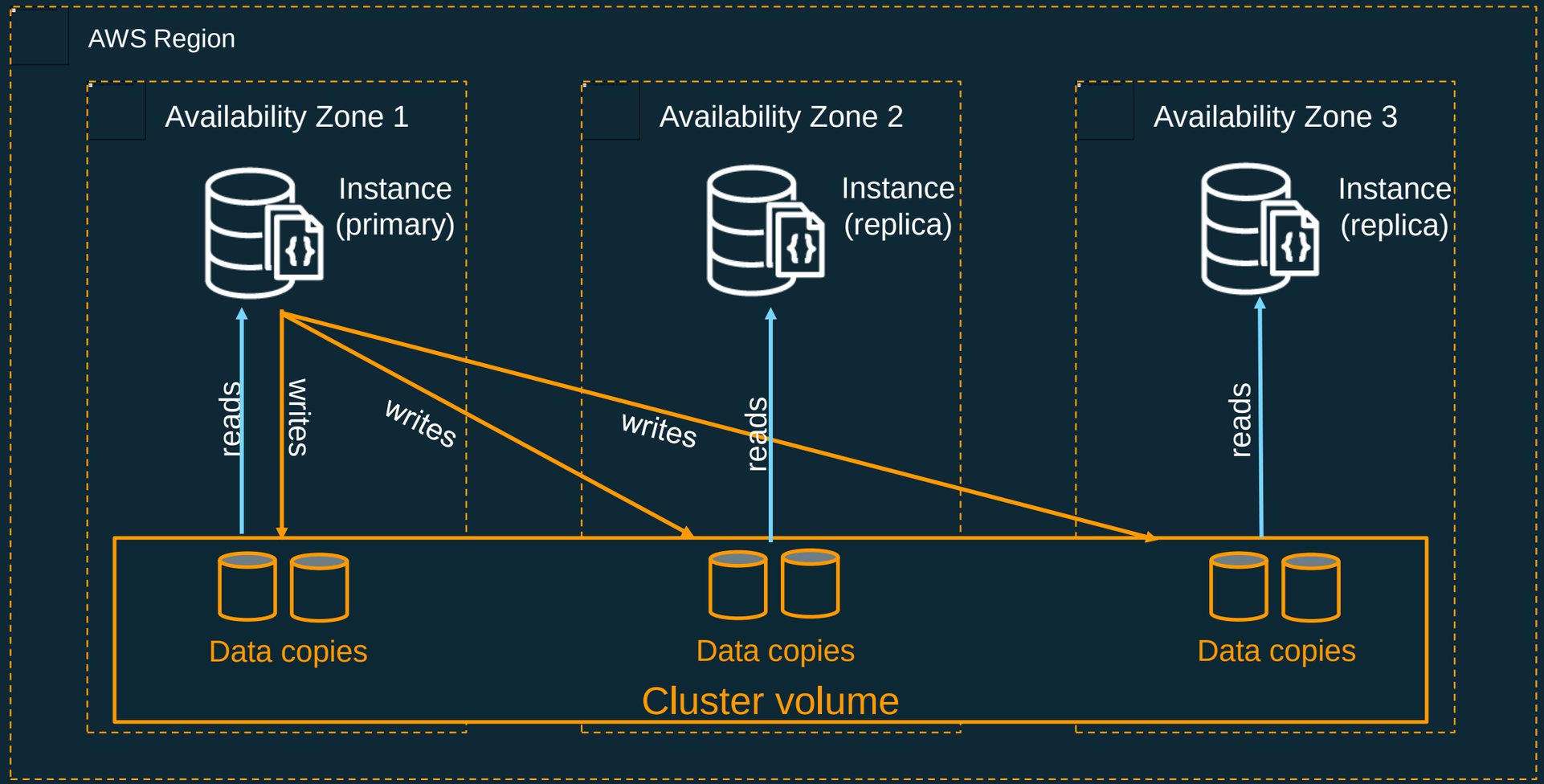
Over 20 key operational metrics for your clusters at no extra charge

Integrated

Deeply integrated with AWS services such as CloudFormation, CloudTrail, CloudWatch, DMS, IAM, VPC, and more.

DocumentDB Architecture

Separate compute and storage provide 2x throughput of current MongoDB managed services



MongoDB-compatible

Fast, reliable, fully-managed MongoDB-compatible database

MongoDB 3.6

Compatible with MongoDB Community Edition 3.6

Same drivers, tools

Use the same MongoDB drivers and tools with DocumentDB; as simple as changing an application connection string

Migration with DMS

Live migrations with DMS; free for 6-months

Replica sets

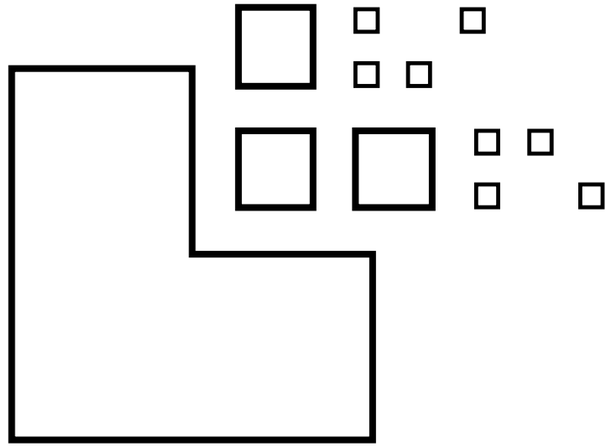
Read scaling is easy with automatic replica set configurations

Amazon DocumentDB availability

- US West (Oregon)
- US East (N. Virginia)
- US East (Ohio)
- EU (Ireland)

Summary

One size doesn’t fit all

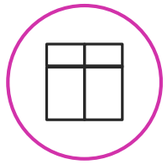


Purpose-built databases

The right tool for the right job

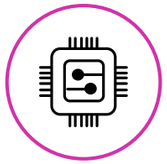
Purpose-built databases

Relational

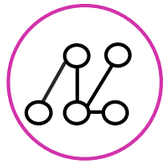


Key-value

Document



In-memory



Graph

Time-series

Ledger

Amazon RDS

DynamoDB

DocumentDB

ElastiCache

Neptune

Timestream

QLDB

Aurora

Community

Commercial

Redis Memcached

Q & A

AWS re:Invent

aws

Thank you

vladsim@amazon.com

AWS re:Invent

aws