NoSQL & Amazon DynamoDB

DSCI 551

Wensheng Wu

Roadmap

- NoSQL 
- Amazon DynamoDB

Relational databases

- Mature & stable
  - Suitable for mission-critical applications, e.g., banking
- Feature-rich versatile query language: SQL
- ACID properties
  - In particular, strong consistency

ACID

- Atomicity: Either all or none of operations in the transaction should be executed
- Consistency: After transaction completes, the database is in a consistent state
- Isolation: allow concurrent execution of multiple transactions that do not interfere with each other
- Durability: can recover from failure

Strong consistency

- Traditionally, a database transaction needs to satisfy ACID properties
  - 'C' in ACID for strong consistency
- Consider a balance-transfer transaction
  - $500 from account A to account B
  - After transfer, the total balance remains the same
  - & users do not get to see the inconsistent state (e.g., debit $500 from A, not yet credit B)

Challenges

- Internet-scale systems & applications
  - E-commerce systems (e.g., Amazon)
  - Social media apps (e.g., Facebook, LinkedIn)
- Big data
  - Often unstructured or semi-structured
- New workloads
  - Write/update-heavy
  - Demand high availability
  - Can tolerate weak consistency

Challenges

- Internet-scale systems & applications
  - E-commerce systems (e.g., Amazon)
  - Social media apps (e.g., Facebook, LinkedIn)
- Big data
  - Often unstructured or semi-structured
- New workloads
  - Write/update-heavy
  - Demand high availability
  - Can tolerate weak consistency

Eventual consistency

- If no new updates are made to the object, eventually all accesses to the object will return the last updated value.

- A form of weak consistency
  - Allow users to see the inconsistency state
  - Needed to achieve high availability (HA)

Inconsistency window

- Time between update acknowledged to user and eventual consistency achieved
  - i.e., updates propagated to all replicas

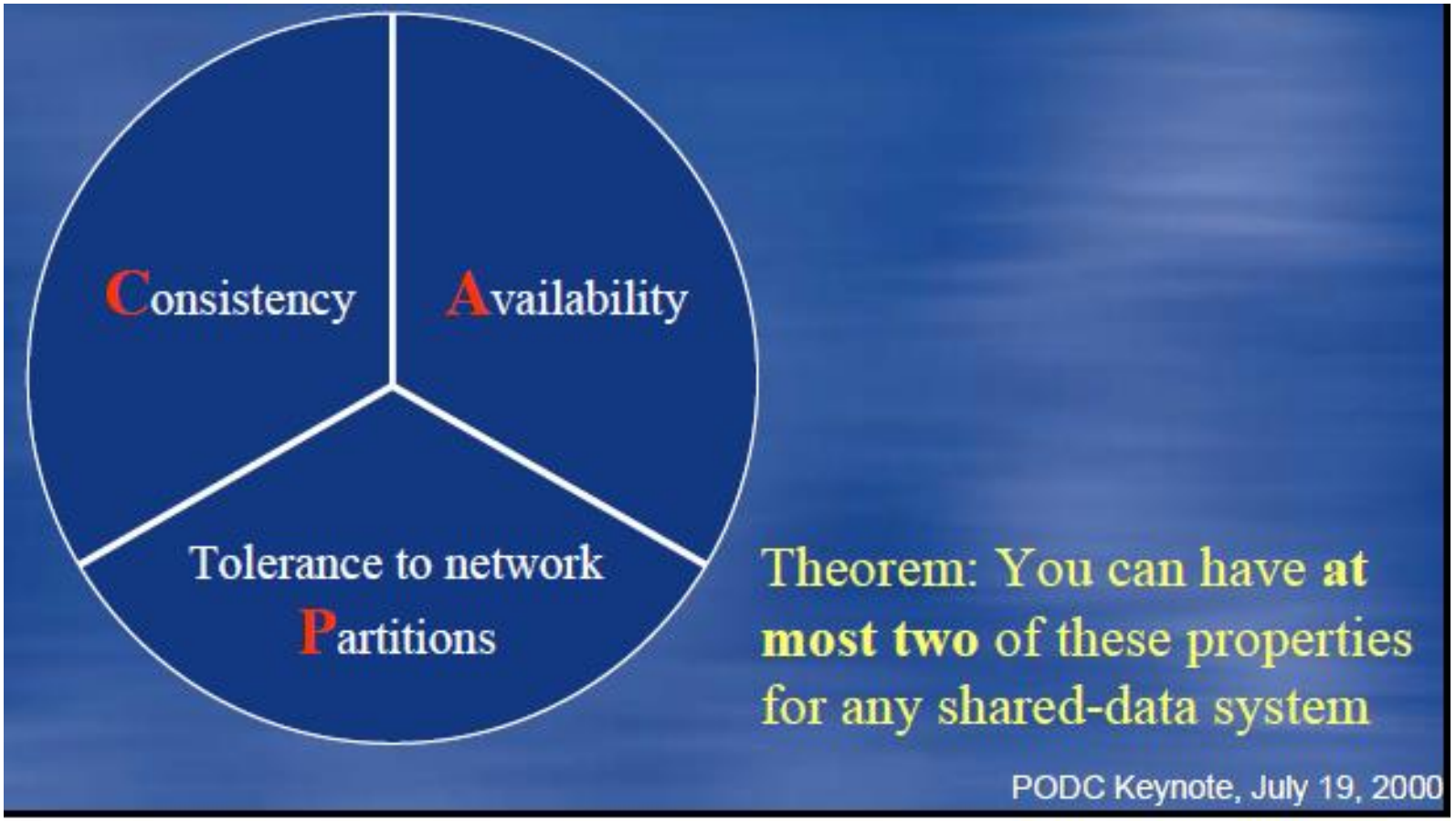
- Length of window determined by:
  - Communication delay
  - Load on the system
  - Number of replicas

Example

- DNS (domain name system) implements eventual consistency
  - E.g., DNS resolves www.usc.edu to 128.125.253.146

- Permissible for some DNS servers to have old data
  - As long as updates eventually propagated to them

CAP theorem


Consistency
Availability
Tolerance to network
Partitions
Theorem: You can have **at most two** of these properties for any shared-data system
PODC Keynote, July 19, 2000

Explanation

Strong consistency

Consistency	*Availability*	*Partition tolerance*
Every read receives the most recent write or an error	Every request receives a (non-error) response – without guarantee that it contains the most recent write	The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

Consequence

- A distributed system needs to tolerate partitioning
  - In other words, property P is required

- Thus, when the network is partitioned, we need to choose between availability and (strong) consistency

  $\Rightarrow$ viability of eventual consistency model

Consequence

- Consider update made to an object O
- User A in LA may see the updated O right away
- But user B in NYC may see the old value of O
  - At least for a while

Eventual consistency model

- Acceptable to many applications
  - E.g., social media, cloud data storage, e-commerce
- Examples:
  - Amazon S3
  - Amazon DynamoDB (backbone of Amazon e-commerce and Web services)

NoSQL databases

- NoSQL: Not only SQL

- Key features
  - Flexible (non-relational) data model
  - Can be easily scaled out (horizontal scalability)
  - Data replicated over multiple servers
  - Weaker consistency model
  - High availability

Scale out vs. scale up

- Scale up (vertical scaling)
  - Beefing up a computer system
  - E.g., adding more CPUs, RAMs, and storage
- Scale out (horizontal scaling)
  - Adding more (commodity) computers
  - Moving some data to new computers

Types of NoSQL databases

- Key-value stores
  - Redis

```
127.0.0.1:6379> set usc 'hello world'
OK
127.0.0.1:6379> get usc
"hello world"
```

- Document stores
  - Firebase: entire database is a JSON value
  - MongoDB: database -> collections/tables -> JSON docs
  - DynamoDB: database -> tables -> rows -> key-value pairs

- Wide column stores
  - Database -> tables -> rows & columns
  - Different rows may have different columns
  - E.g., Apache Cassandra & HBase

Roadmap

- NoSQL
- Amazon DynamoDB 

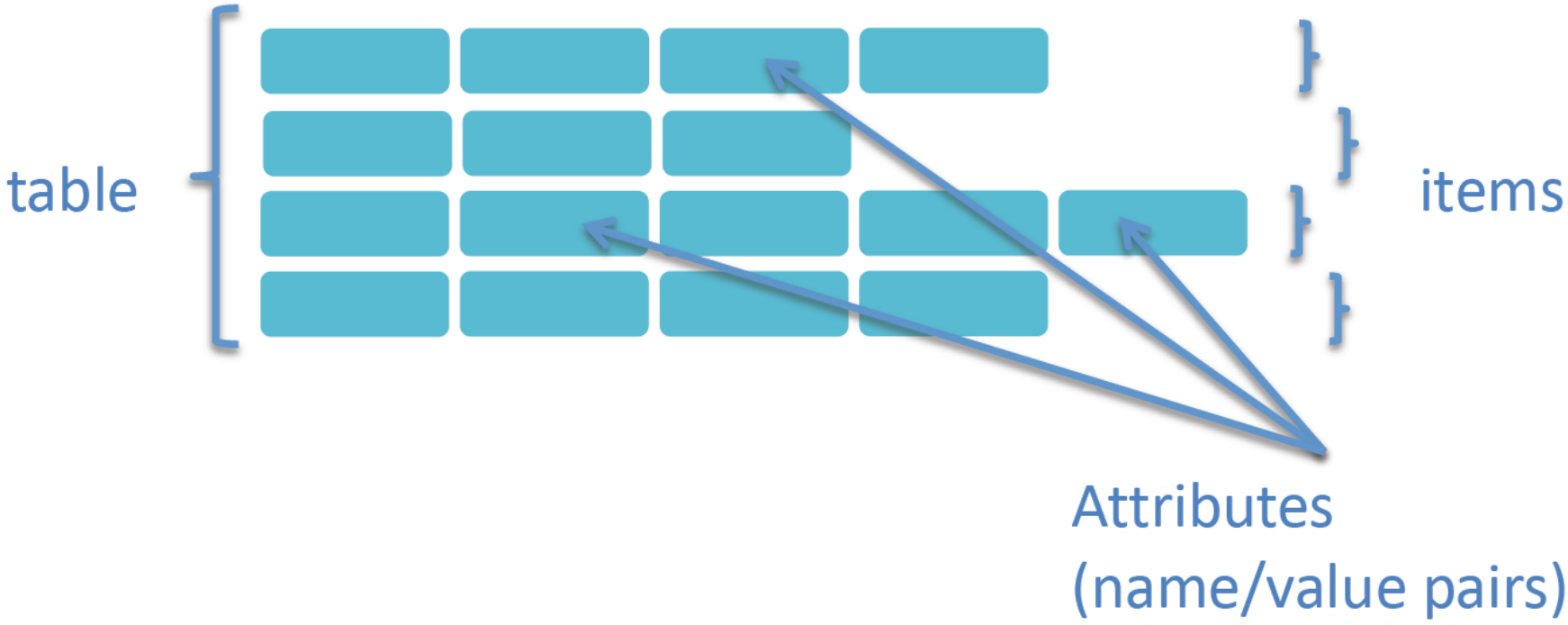
Amazon DynamoDB

- Schema-less: no predefined schema
  - Other than primary key
- Database contains a list of tables, e.g., music
- A consists of a set of items/rows
  - E.g., a set of music CDs
- Each item contains a set of attributes
  - E.g., artist, title, year of CD

Items

- Similar to rows in relational databases
- But different rows may have different set of attributes
- Max size of an item: 400K
- No concept of columns in DynamoDB

DynamoDB table structure



Primary key

- Each item is uniquely identified by a primary key

- Primary key consists of
  - partition key
  - (optional) sort key

Partition key

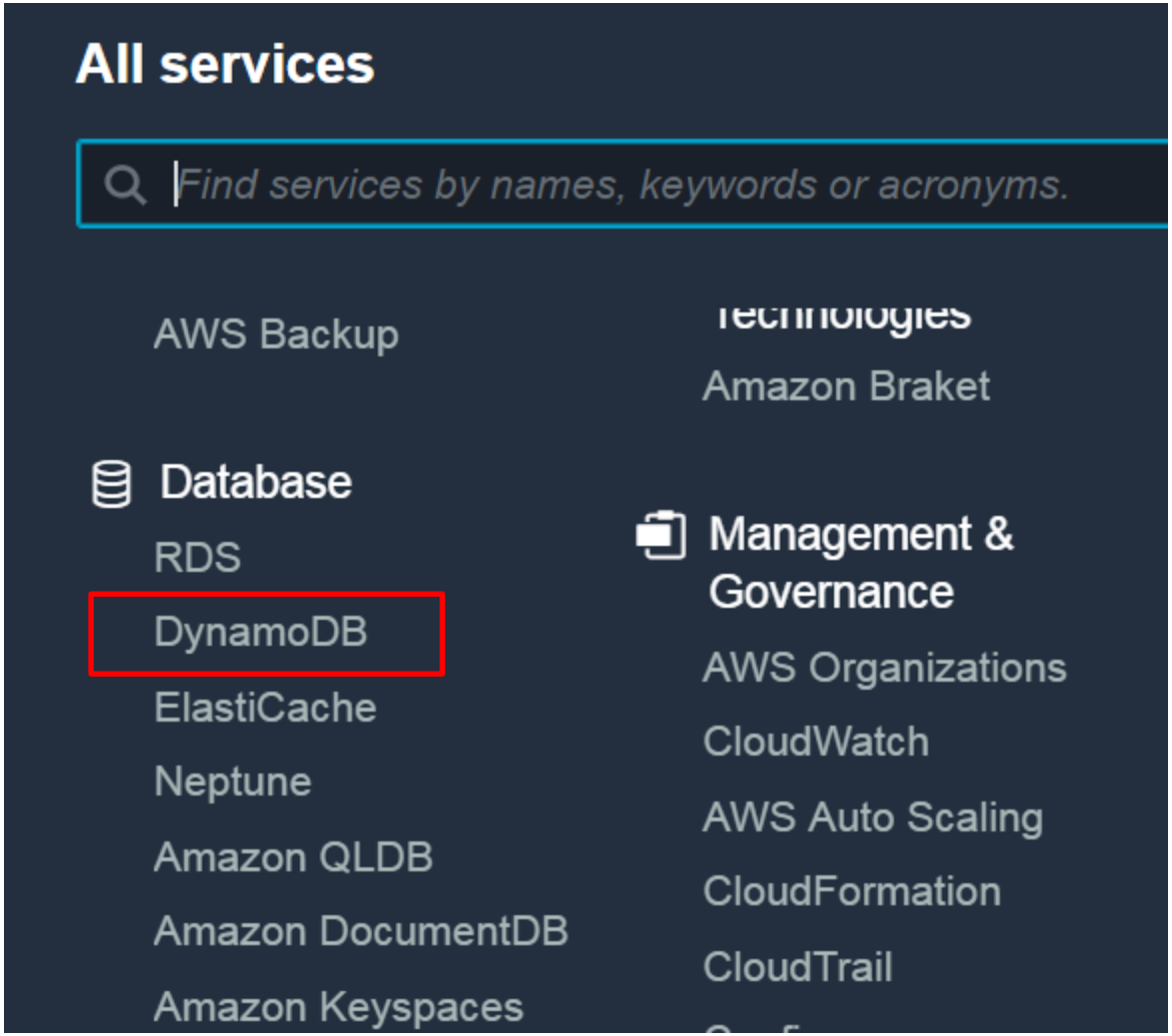
- Partition key
  - Partition (by hashing) the data across hosts for scalability & availability
- Pick an attribute with wide range of values & evenly distributed patterns for partition key
  - E.g., user ID
- E.g., artist name
  - Hash function may put "Rod Stewart" and "Maria Kelly" in the same partition

Sort key

- Allow searching within a partition
- E.g., year
  - So primary key = artist + year

Possible multiple items with the same artist but different years

- This allows search all CDs by a specific artist and produced in certain years


All services
Find services by names, keywords or acronyms.
AWS Backup
Amazon Braket
Database
RDS
DynamoDB
ElastiCache
Neptune
Amazon QLDB
Amazon DocumentDB
Amazon Keyspaces
Management & Governance
AWS Organizations
CloudWatch
AWS Auto Scaling
CloudFormation
CloudTrail

Example

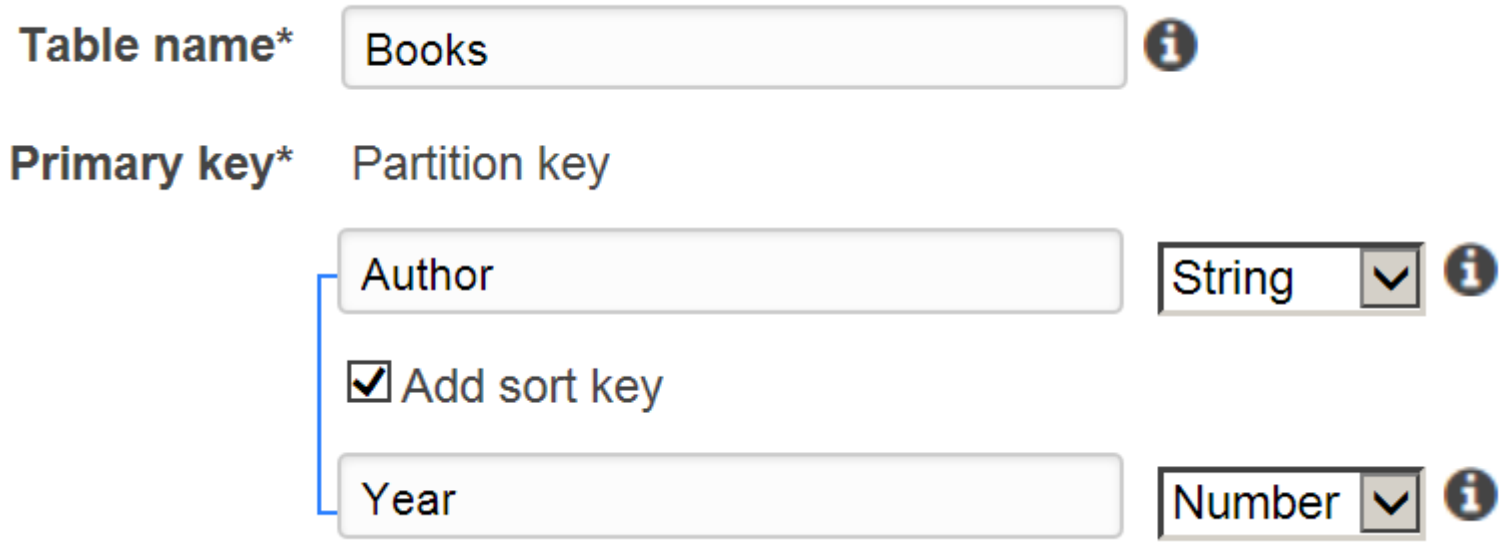
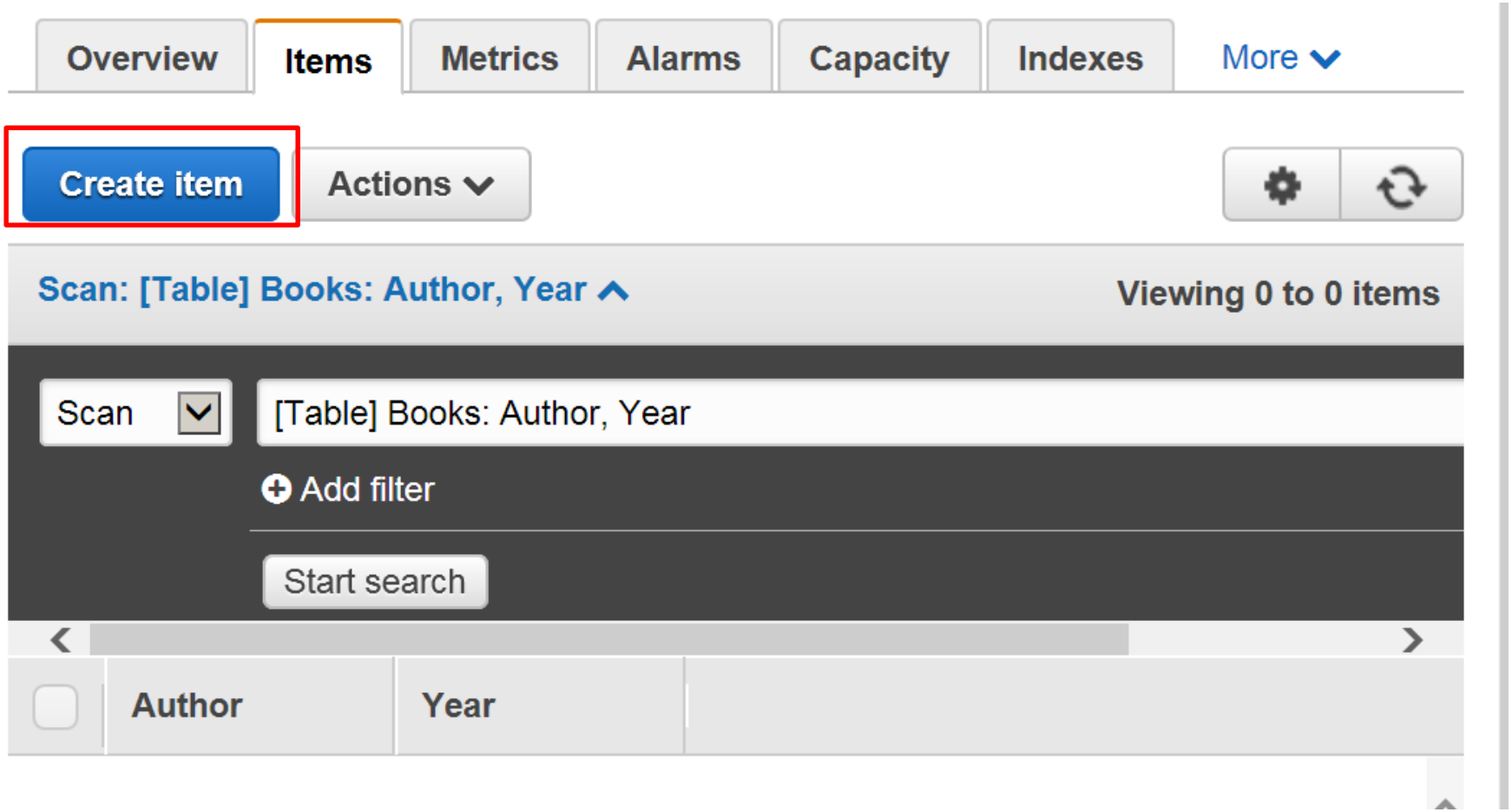
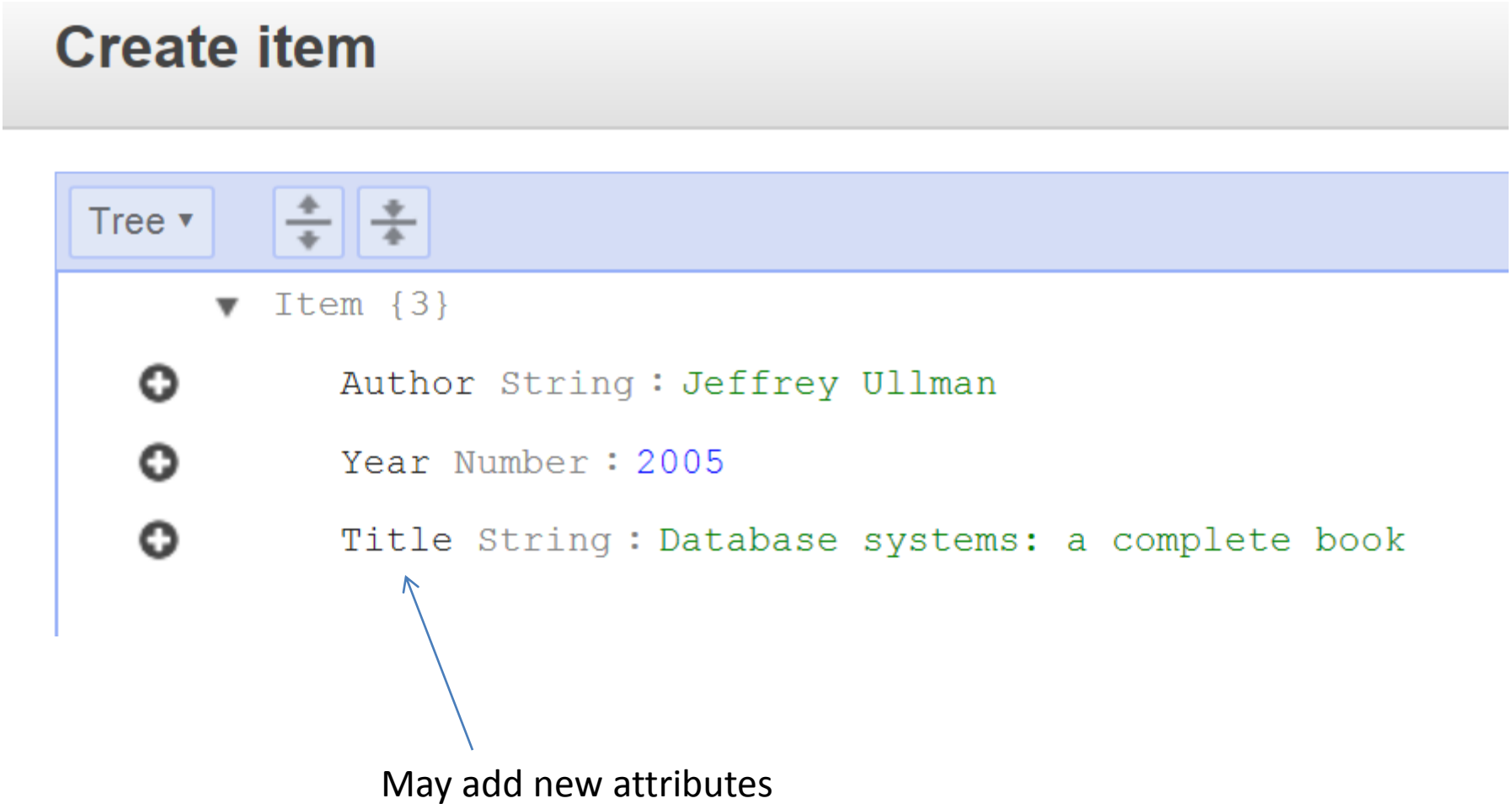


Table name*
Books
Primary key*
Partition key
Author
String
Add sort key
Year
Number

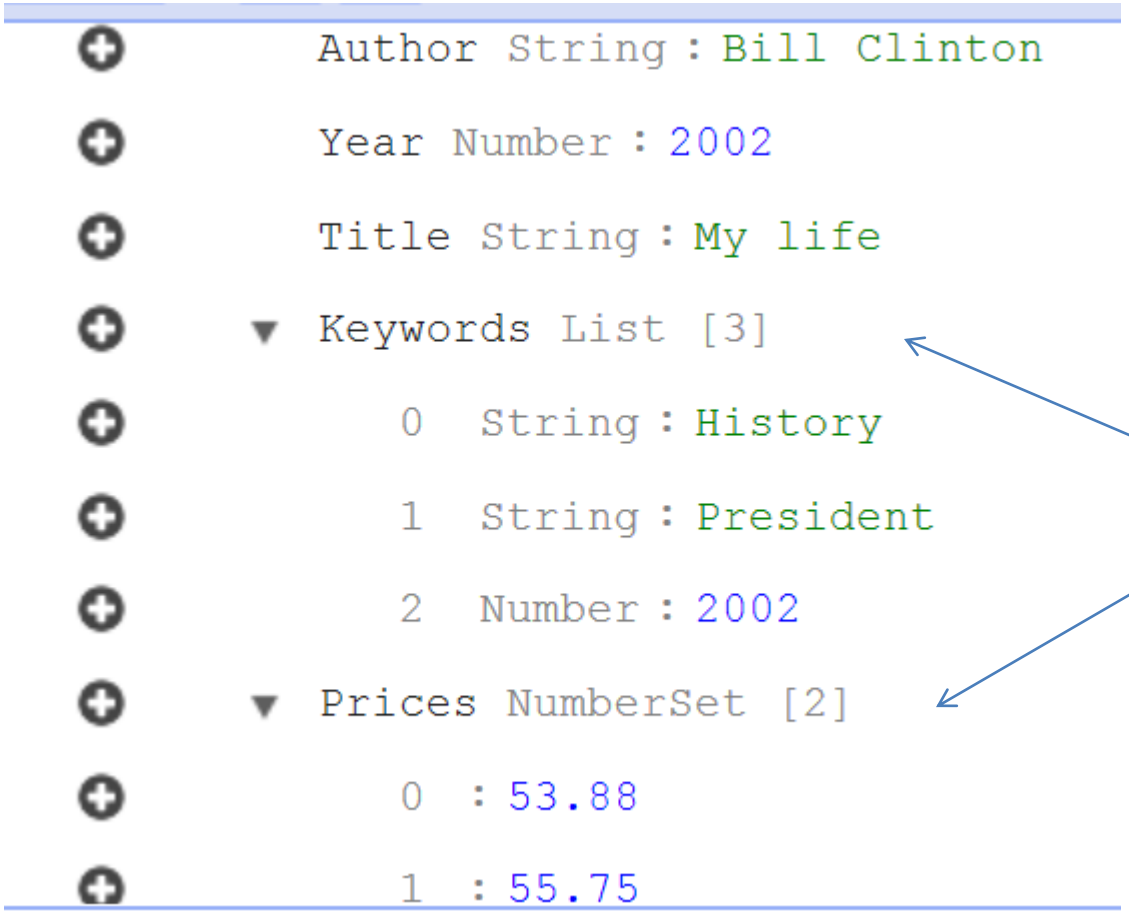
Example (may vary in new version)



Example


Create item
Tree
Item {3}
Author String : Jeffrey Ullman
Year Number : 2005
Title String : Database systems: a complete book
May add new attributes

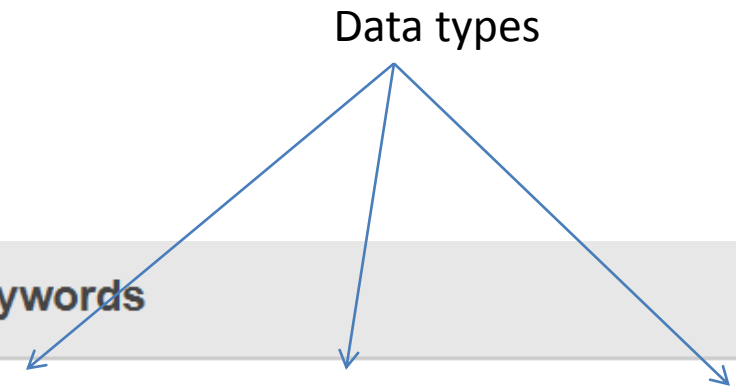
Example



Value can be a list or a set

List: ordered, heterogeneous
Set: unordered, homogeneous

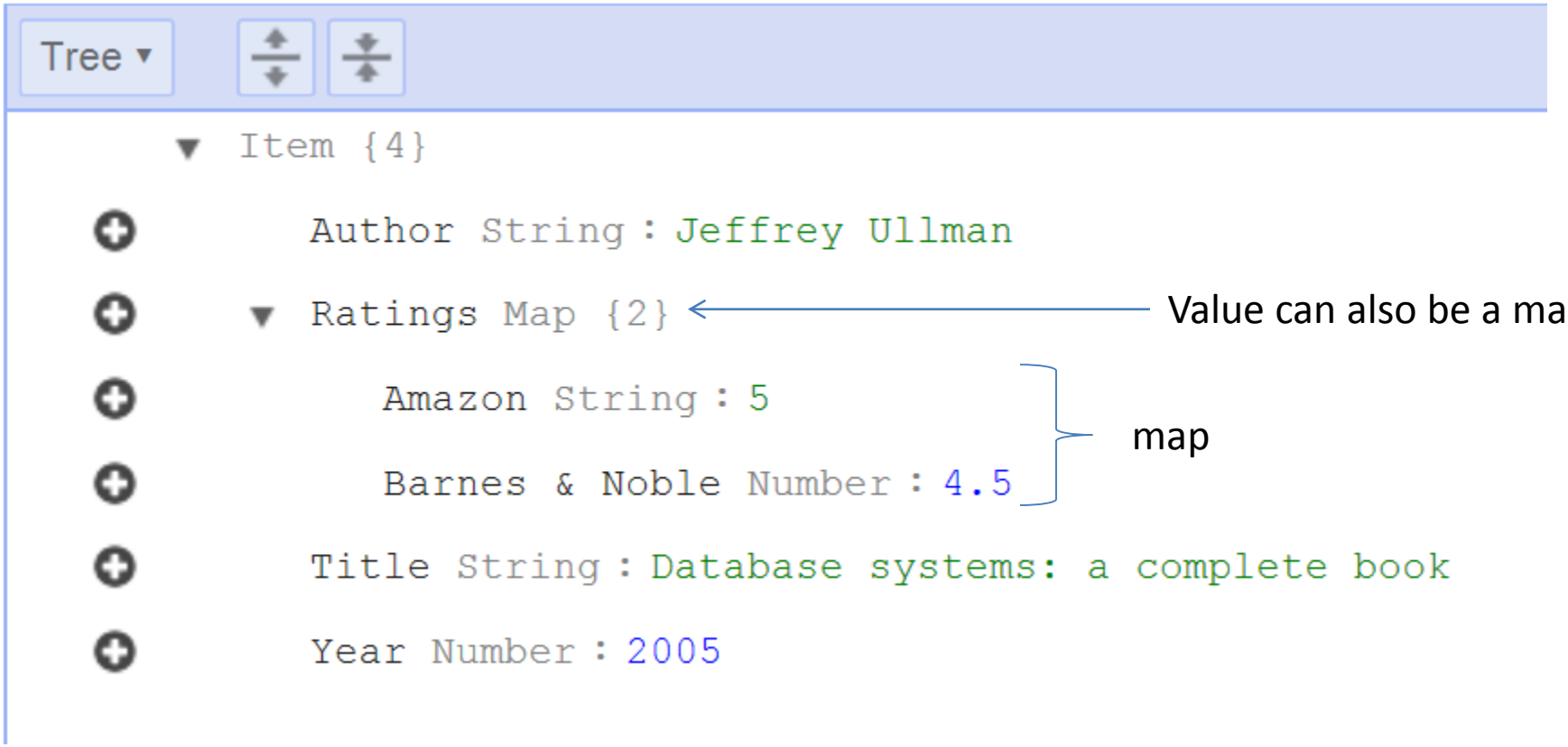
Example



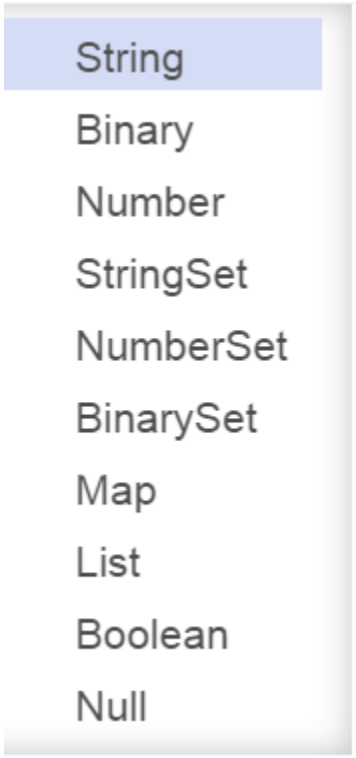
	Title	Keywords	Prices
	My life	[{ "S" : "History" }, { "S" : "President" }, { "N" : "2002" }]	{ 53.88, 55.75 }
	Databas…		

Example

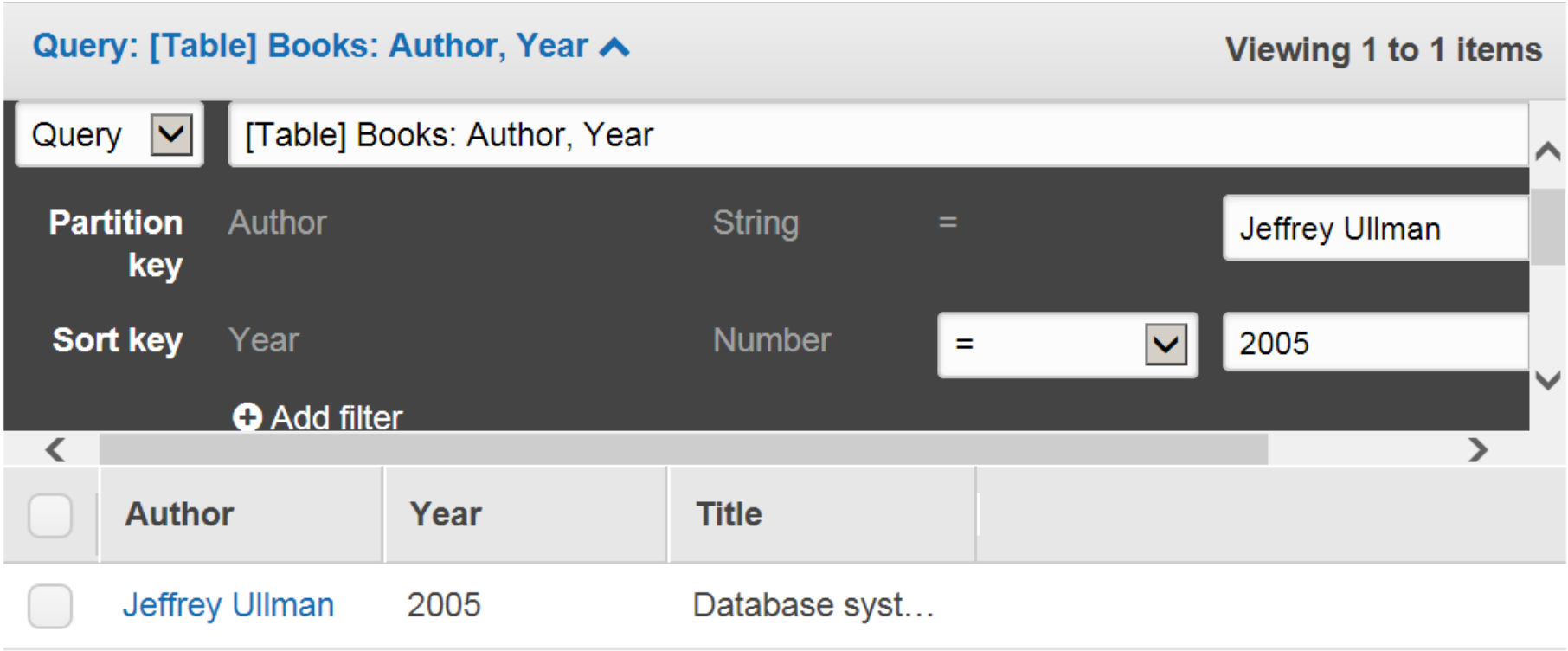
Map: contains a list of key-value pairs



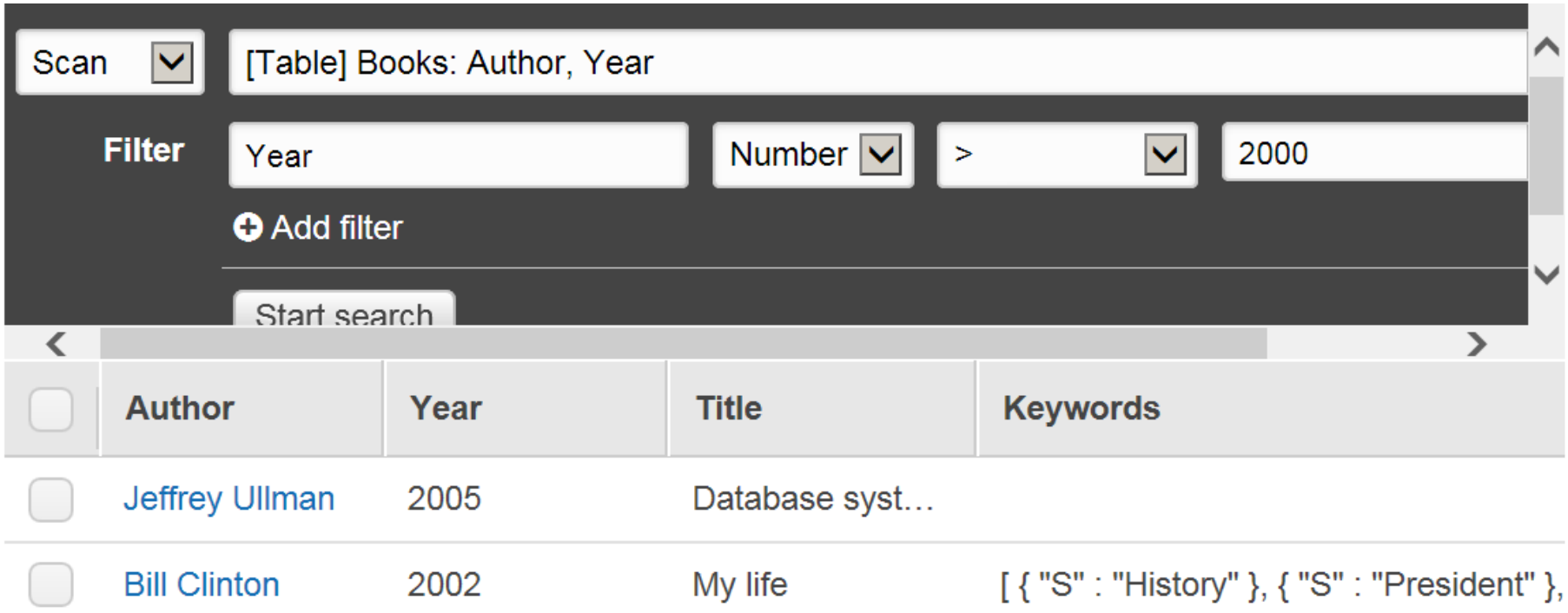
Available data types for values



Query


Query: [Table] Books: Author, Year
Viewing 1 to 1 items
Query
[Table] Books: Author, Year
Partition key
Author
String
=
Jeffrey Ullman
Sort key
Year
Number
=
2005
Add filter
Author
Year
Title
Jeffrey Ullman
2005
Database syst…

Scan


Scan
[Table] Books: Author, Year
Filter
Year
Number
>
2000
Add filter
Start search
Author
Year
Title
Keywords
Jeffrey Ullman
2005
Database syst…
Bill Clinton
2002
My life
[{ "S" : "History" }, { "S" : "President" },

PartiQL

- Insert:
  - insert into books value {'author': 'trump1', 'year': 2021}
- Select:
  - select * from books where instock = true

PartiQL

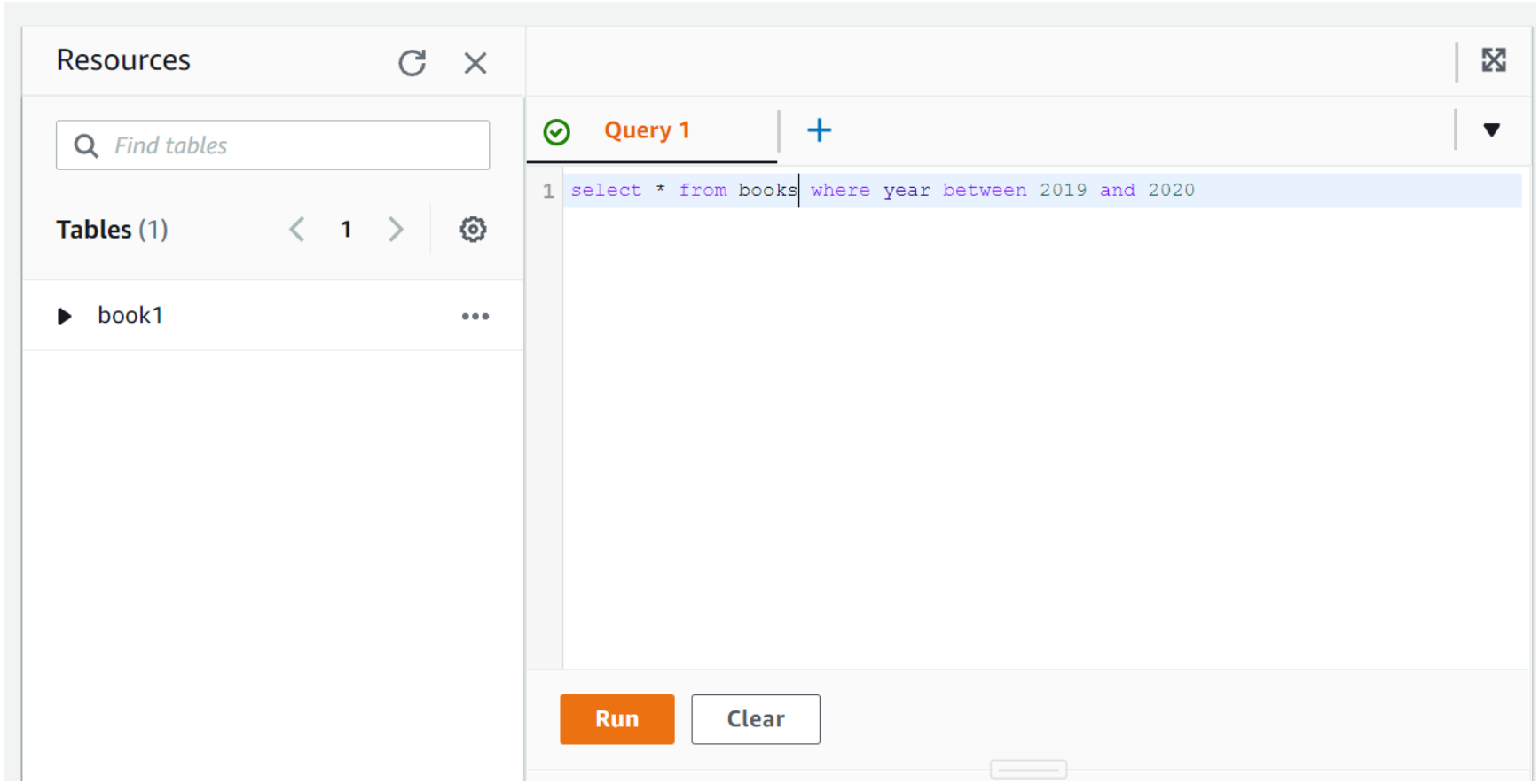
- Update:

```
update books
set title = 'the art of deal' // a new attribute
where author = 'trump' and year = 2021;
```

- Delete:

```
delete from books
where author = 'trump' and year = 2021
```

Example


Resources
Find tables
Tables (1)
1
book1
Query 1
1 select * from books where year between 2019 and 2020
Run
Clear

References

- PartiQL for DynamoDB:
  - https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-reference.html

- Working with Tables, Items, Queries, Scans, and Indexes
  - https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/WorkingWithDynamo.html