

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

事务

DML操作遵循ACID模型，支持事务

****原子性 (Atomicity) ****：事务是数据库操作的基本单位，它要么完全执行，要么完全不执行。如果事务中的任何操作失败，整个事务将被回滚到初始状态，以确保数据库的一致性。

****一致性 (Consistency) ****：事务执行后，数据库从一个一致的状态转换到另一个一致的状态。这意味着事务必须遵循数据库的完整性约束，以确保数据的有效性和一致性。

****隔离性 (Isolation) ****：多个事务可以并发执行，但其效果不能相互干扰。隔离性确保在并发事务执行时，每个事务都感觉像是在独立执行，避免了由并发执行引起的数据不一致性问题。

****持久性 (Durability) ****：一旦事务提交，其结果应该永久保存在数据库中，即使系统发生故障或崩溃也不会丢失。数据库系统通过将事务的更改写入持久存储（如磁盘）来保证持久性。

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)

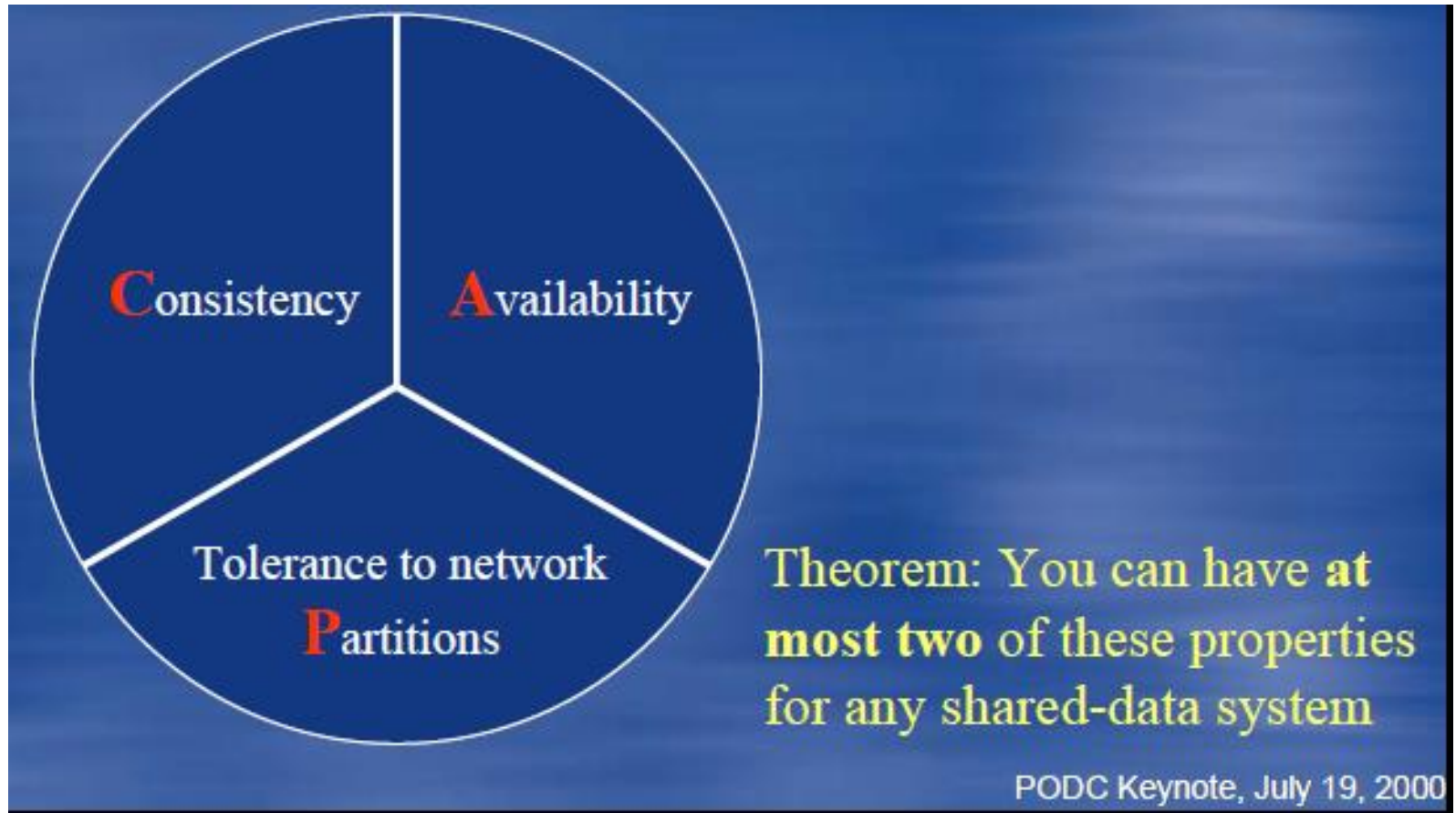
Inconsistency window

- 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



Explanation

Strong consistency



<i>Consistency</i>	<u><i>Availability</i></u>	<u><i>Partition tolerance</i></u>
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
 - ⇒ 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

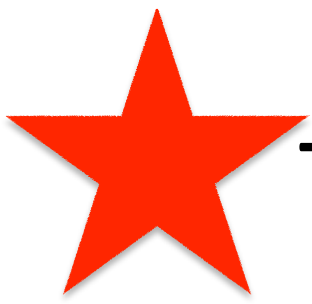
- Acceptable to many applications
 - E.g., social media, cloud data storage, e-commerce
- Examples:
 - Amazon S3
 - Amazon DynamoDB (backbone of Amazon e-commerce and Web services)

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



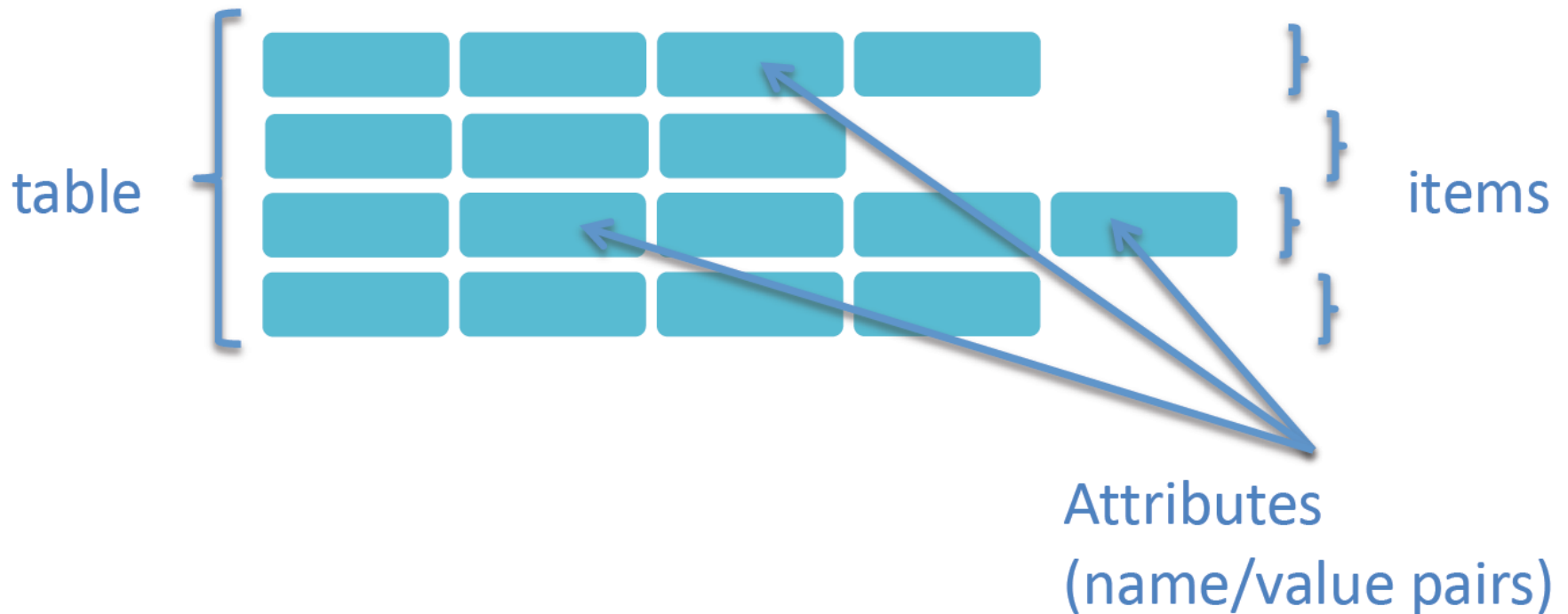
Amazon DynamoDB

- 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



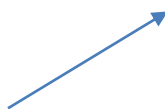
Primary key

- 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
 - This allows search all CDs by a specific artist and produced in certain years
- Possible multiple items with the same artist but different years
- 

All services

🔍 *Find services by names, keywords or acronyms.*

AWS Backup

technologies

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)

The screenshot displays a web interface for managing data items. At the top, a horizontal navigation bar contains tabs for 'Overview', 'Items' (which is selected and highlighted with an orange border), 'Metrics', 'Alarms', 'Capacity', and 'Indexes'. To the right of these tabs is a 'More' link with a downward arrow. Below the navigation bar, there is a row of controls. On the left, a blue button labeled 'Create item' is highlighted with a red rectangular border. Next to it is a grey button labeled 'Actions' with a downward arrow. To the right of these are two icons: a gear for settings and a circular arrow for refresh. Below this row is a header section with the text 'Scan: [Table] Books: Author, Year' followed by an upward arrow, and on the right, 'Viewing 0 to 0 items'. Underneath the header is a search area. It starts with a 'Scan' label and a dropdown arrow, followed by a text input field containing '[Table] Books: Author, Year'. Below the input field is a '+ Add filter' link. At the bottom of the search area is a 'Start search' button. Below the search area is a table with two visible columns: 'Author' and 'Year'. The 'Author' column has a checkbox in its header. The table is currently empty. At the bottom right of the table, there is a small upward arrow icon.

Overview Items Metrics Alarms Capacity Indexes More ▾

Create item Actions ▾

Scan: [Table] Books: Author, Year ⬆ Viewing 0 to 0 items

Scan ▾ [Table] Books: Author, Year



+ Add filter

Start search

<input type="checkbox"/>	Author	Year
--------------------------	--------	------

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

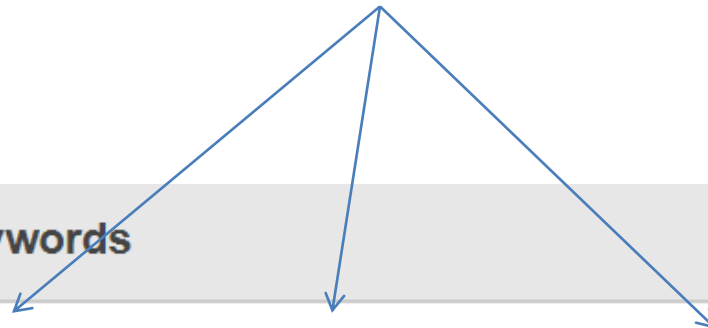
```
+ Author String : Bill Clinton
+ Year Number : 2002
+ Title String : My life
+ ▼ Keywords List [3]
+   0 String : History
+   1 String : President
+   2 Number : 2002
+ ▼ Prices NumberSet [2]
+   0 : 53.88
+   1 : 55.75
```

Value can be a list
or a set

List: ordered, heterogeneous
Set: unordered, homogeneous

Example

Data types



	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

Tree ▾

▼ Item {4}

- ⊕ Author String : Jeffrey Ullman
- ⊕ ▼ Ratings Map {2} ← Value can also be a map
 - ⊕ Amazon String : 5
 - ⊕ Barnes & Noble Number : 4.5
- ⊕ Title String : Database systems: a complete book
- ⊕ Year Number : 2005

map

Available data types for values

String

Binary

Number

StringSet

NumberSet

BinarySet

Map

List

Boolean

Null

Query

Query: [Table] Books: Author, Year ^Viewing 1 to 1 items

Query

Partition key

Author

String

=

Sort key

Year

Number

=


+ Add filter

<

>

<input type="checkbox"/>	Author	Year	Title	
<input type="checkbox"/>	Jeffrey Ullman	2005	Database syst...	


Scan


Scan 

[Table] Books: Author, Year


Filter

Year

Number 

> 

2000

 Add filter

Start search

<input type="checkbox"/>	Author	Year	Title	Keywords
<input type="checkbox"/>	Jeffrey Ullman	2005	Database syst...	
<input type="checkbox"/>	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

The image shows a web-based SQL query editor interface. On the left, a sidebar titled "Resources" contains a search bar labeled "Find tables" and a section titled "Tables (1)" with a list item "book1". The main area on the right is titled "Query 1" and contains a SQL query: `select * from books where year between 2019 and 2020`. Below the query editor are two buttons: "Run" and "Clear".

Resources

Find tables

Tables (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.htm>
!