DS684
Cloud Computing
Week 05

Regarding Labs and Assignments

- Class participation means more than Zoom attendance. You must actively participate in the discussion and labs, and answer questions.

- Must hit Submit button, otherwise no grade
- If you need extension in time, must send written request (***email***). Otherwise no grade and no makeup. Requests sent over Zoom chat do not count.
- For any technical difficulty (installation, Azure access, etc), you must send written explanation (***email***) before the deadline. Otherwise no grade and no makeup.

Agenda

- Challenge to the Relational Database
  - The Big Data Challenge
- NoSQL Introduction
  - JSON Processing in Relational Database
  - Azure Cosmos DB for Mongo DB Lab
- Distributed Computing

Data Management before 2000: RDBMS

1950’s: Early data management practices

1970’s: Relational model invented

1990’s: RDBMS takes over

- One size fits all: All relational, only relational, nothing but relational.
- An ecosystem of data management tools formed around RDBMS

Advantages of Relational Database

- Built on top of a solid foundation of relational algebra
- A generic model that can fit into many different scenarios
- A simple but universal operating tool - Structured Query Language (SQL)
- Full ecosystem with established vendors and community
- Well understood. Large user base.

SQL at the Center of Data Management

Ecosystem surrounding RDBMS:

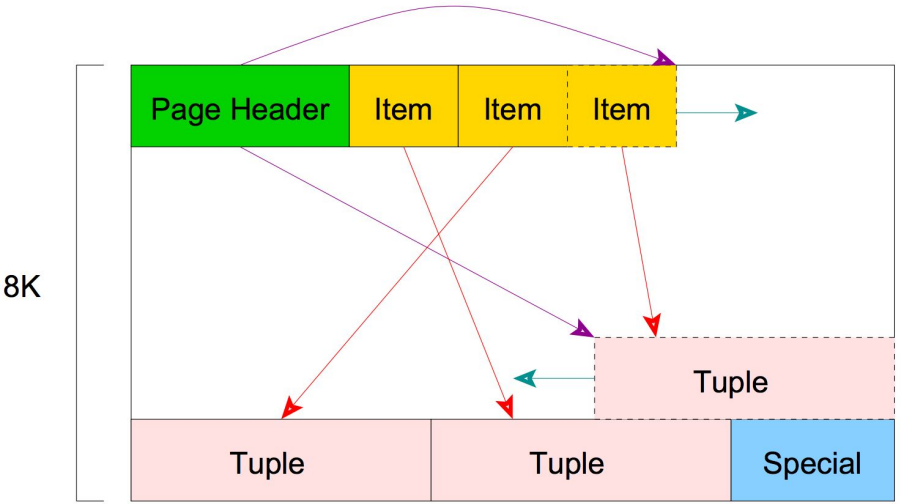
- Front end application (2-tier, 3-tier, 4-tier)
- Extraction, Transformation, and Loading (ETL)
- Business Intelligence (BI)

In its core, it is all about RDBMS: Interchangeable with Data Management

- Built to optimize relational operations, SQL as main language
- Individual server. Dedicated hardware

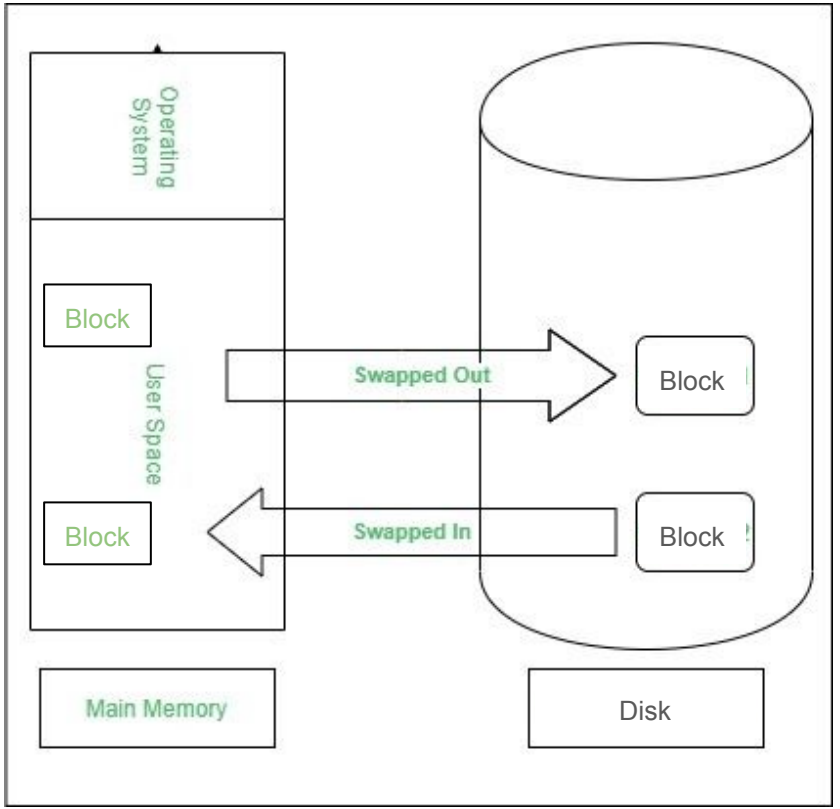
Data Storage of RDBMS

- Data is in disk block format: Row oriented or Column oriented
  - Row oriented: One tuple at a time
  - Column oriented: One data trunk at a time
- Memory operation for INSERT and UPDATE
  - Data block locks: Use transaction to ensure ACID



Data Swap

- Data is stored in block. Updating one record requires the reading and writing of the entire block
- What if two uses are operating at the same block at the same time?
  - Whoever writes last win (or is it a win?)



ACID

- Atomicity - each statement in a transaction (to read, write, update or delete data) is treated as a single unit.
- Consistency - transactions only make changes to tables in predefined, predictable ways.
- Isolation - concurrent transactions don't interfere with or affect one another
- Durability - changes to your data made by successfully executed transactions will be saved

Relational databases implement a series of mechanisms to ensure ACID, which ensures the data integrity, but hurts the overall performance.

Data Processing in RDBMS

- Use join to combine data: CPU (comparison) and memory intensive
  - Schema must be carefully structured. Objects must be pre-defined.
  - Use index and query plan: fixed optimization path
- Memory buffer for frequently accessed data

Challenges From Within

- Rigid schema/format
  - Difficult to expand: Fixed schema
  - Not optimal for sparse data: social networking tags
- Costly to maintain complex relationships:
  - Join is expensive
- ACID control: double edged sword
  - Transaction conflicts and locks
- Server based
  - Difficult to scale
  - Single point of failure.

Agenda

- Challenge to the Relational Database
  - The Big Data Challenge
- NoSQL Introduction
  - JSON Processing in Relational Database
  - Azure Cosmos DB for Mongo DB Lab
- Distributed Computing

Challenges From Outside: Big data

Big Data: The “Big” Challenge

- Introduced by the wide adoption of Internet (Google/Facebook)
- Makes the issues in relational database more apparent

Characteristics of Big Data:

- Three Vs: Volume, Variety, Velocity

Volume Challenge

Exponential growth of data

Phase 1: Enterprise data

Phase 2: Business - Consumer Interaction data

- Website search
- Online shopping

Phase 3: Consumer/device Generated data

- Social media
- IoT/mobile devices/sensors

Volume Challenge in Processing

- A single petabyte contains enough high-definition video to play 24/7 for 3.4 years --- Facebook 4 PB daily.
- Limited Processing Power
  - Laptop: 16GB RAM, 1TB Disk
  - Server: 1TB RAM, 10TB Disk
  - Teradata specialized server
    - Each node: 512GB RAM, > 5TB Disk
    - 16 node cluster: 8TB RAM, > 80TB Disk (up to 30PB)
    - Achievable but costly: $15K per month per node. $240K per cluster.

Variety Challenges

- Not all data has schema
  - Natural language
  - Image/Audio/Video
- Not all data has fixed schema
  - Human interactions: Like, Comment
  - Hashtag: sparse and not even

Data Schema Types

- Structured (predefined, fixed schema): relational database, CSV
- Unstructured (no schema): video, audio, images
- Semi-Structured (predefined, flexible schema): JSON

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "phoneNumbers": [
    {
      "type": "home", "number": "212 555-1234"
    }
  ],
  "children": [],
  "spouse": null
}
```

Velocity Challenges: Fast Delivery

Today’s applications must deliver contents in a much faster, even near-real time manner.

- Fraud detection
- Sensor: real time flood monitoring
- News: US Presidential Election

Agenda

- Challenge to the Relational Database
  - The Big Data Challenge
- NoSQL Introduction
  - JSON Processing in Relational Database
  - Azure Cosmos DB for Mongo DB Lab
- Distributed Computing

Current Trends of Data Management

- New platform
  - Open source RDBMS: MySQL, PostgreSQL
  - Cloud Computing: infrastructure, maintenance
- New Types of data management systems
  - NoSQL Databases
  - Data Lake
- Distributed computing
  - Apache Hadoop/Spark

NoSQL Introduction

A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases (“NoSQL”, Wiki)

They are built for specific data scenarios that relational databases have difficulties in dealing with.

NoSQL: Originally it means NO SQL

- Lately: Not Only SQL

Two Common NoSQL Scenarios

- Schemaless / Relaxed Schema: Flexible, Unstructured data handling
  - Column family
  - Non relational: Data may be structured, but schema is not enforced
  - Map, Video, Natural Language (hashtag, social interaction)
  - Graphic relationship
- Not ACID compliant: Trade consistency for performance, scalability, and/or availability
  - eCommerce: availability is more important
- Might be both

Common Types of NoSQL

- Key-value pair: AWS DynamoDB, Redis
- Wide column store: HBase, GCP Big Table, Cassandra
- Document Store: MongoDB, GCP Firestore (new generation for Datastore)
- Graph database: Neo4j
- And more …
  - Specialized DBMS such as time series, blockchain, etc.

Key Value NoSQL

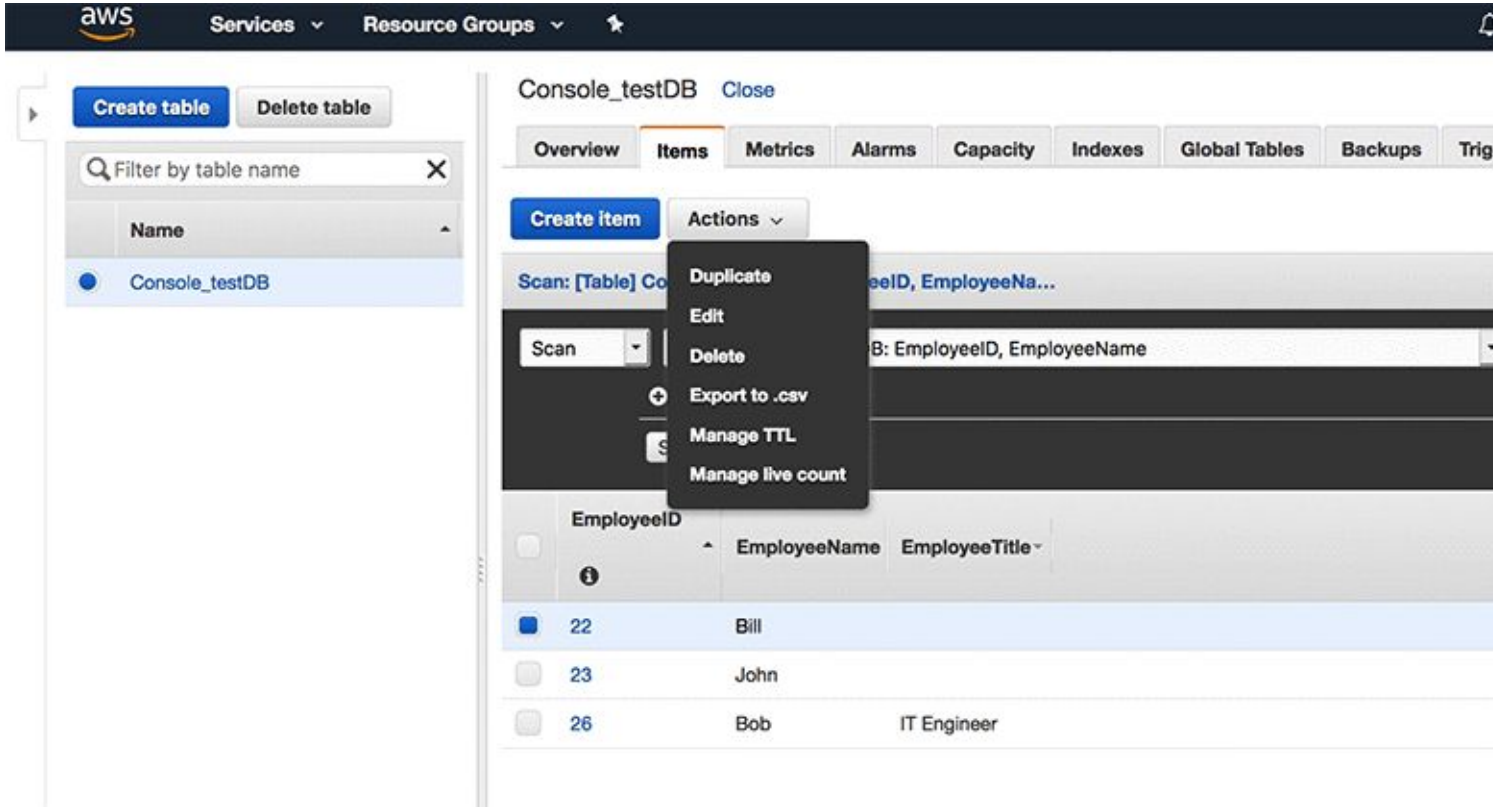
Key Value NoSQL

- Stores data as key-value pairs
- Key must be unique in the table

Characteristics

- No complex query
- No ACID
- Fast
- Scalable

Key Value NoSQL: AWS DynamoDB



Wide Column NoSQL

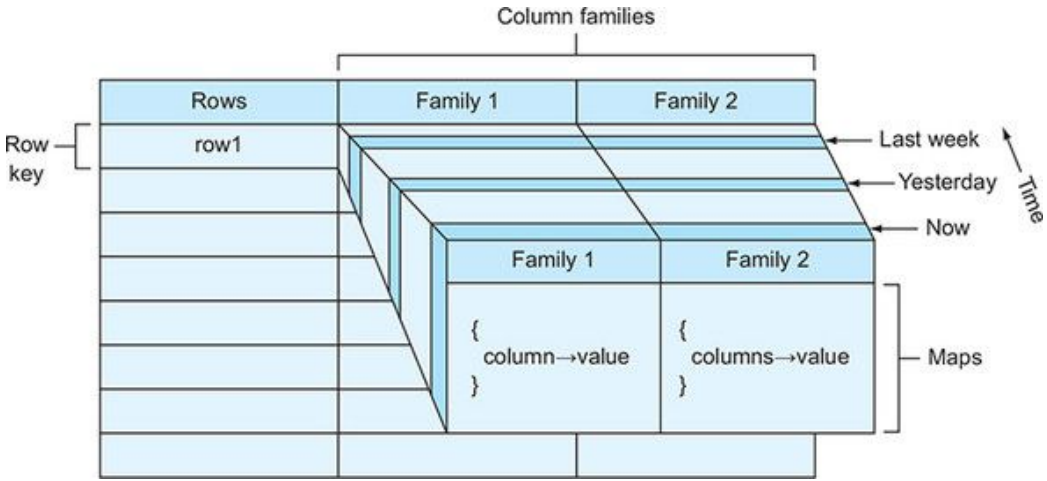
Wide Column NoSQL:

- The names and format of each column can vary from row to row
- A natural extension of key-value database: key - multi value
- Usually aims at high performance/high availability reading

Example: Google BigTable (the database that is behind Google search and Google Map)

BigTable Schema

- Hashed key
- Sparse column storage
- How to determine row granularity is a major design consideration, both for performance and storage



Document Store NoSQL

Document Store NoSQL:

- Stores, retrieves and manages document-oriented information, also known as semi-structured data.
- Usually, the semi-structured data is in JSON format. Each element is called a document.

Example: MongoDB

Graph DB NoSQL

Graph DB NoSQL:

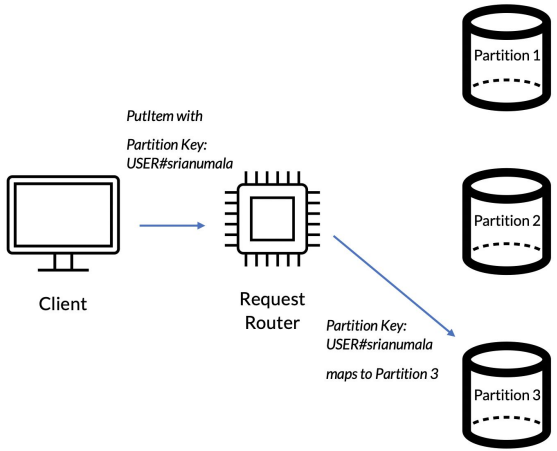
- Uses graph structures for semantic queries with nodes, edges, and properties to represent and store data
- Handles complex relationship

Example: Neo4j

Which One to Pick, and How?

Partition Tolerance

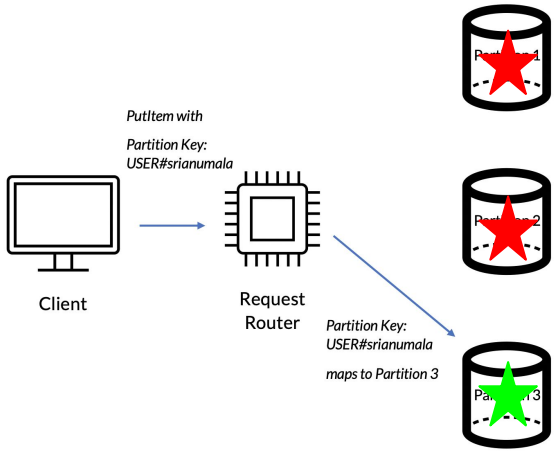
- Some databases choose to be distributed (multiple nodes, multiple locations). Some others don't.
- For the former, nodes must be distributed far enough to provide enough availability, or, to provide Partition Tolerance: even if there is a partition (break in network), the nodes are still functionable



Consistency and Availability

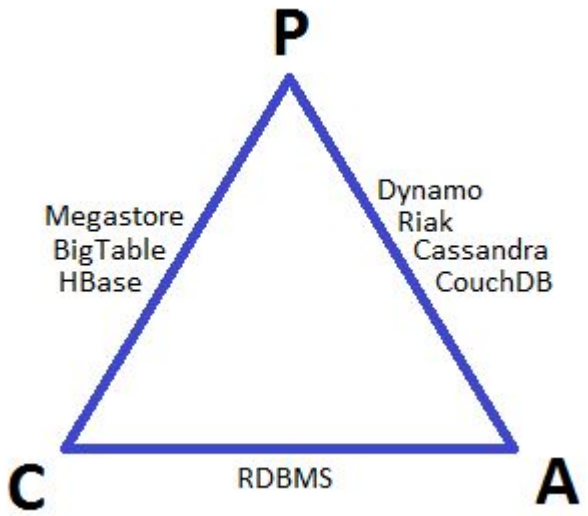
In a partition tolerance environment, when there is a write to a node, the other nodes must be updated. Between the initial write and the eventual consistent state, we must choose between:

- Consistency: No data until all nodes are consistent
- Availability: Provide data even as we know they may be inconsistent



CAP Triangle

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



Definition from Wikipedia

Back to Relational

Relational database is still the fundamental technology in data management:

- Simplicity: Two dimensional data. Easy to figure out.
  - Comparing to Graph DB
- Atomicity: Data operation is at its lowest level. No embedded processing.
  - Comparing to JSON processing
- Declarative: No need to worry about low level implementation
  - Comparing to MongoDB or MapReduce coding
  - A powerful but intuitive standardized tool: Everybody loves SQL.
- Matured: Nothing is unheard of.

Even Competitors Agree

From “NO SQL” to “Not Only SQL”:

- Many NoSQL systems still use relational model for the conceptual design
- SQL support are added to new tools: Spark, Cassandra, Flink, etc
- NoSQL databases have been successful in many areas. But no NoSQL database has been successful in *all* areas.

The Near-Future Prospect

- Relational Database will serve the majority, most common cases.
- Different NoSQL and Big Data technologies will each find their niche market
  - Fierce competition in each niche market
  - Each technology (even relational databases) will try to break into other markets

The Basic Lifecycle of Data

No matter what the DBMS is (relational, key-value, document, etc.), or what your processing tool is, the lifecycle of data is the same:

Create, **R**ead, **U**pdate and **D**elete (CRUD)

For any new technology, understand how it carries out CRUD.

- CREATE/INSERT/SELECT/UPDATE/DELETE/DROP in SQL
- use/insert/find/update/remove in MongoDB
- etc.

Agenda

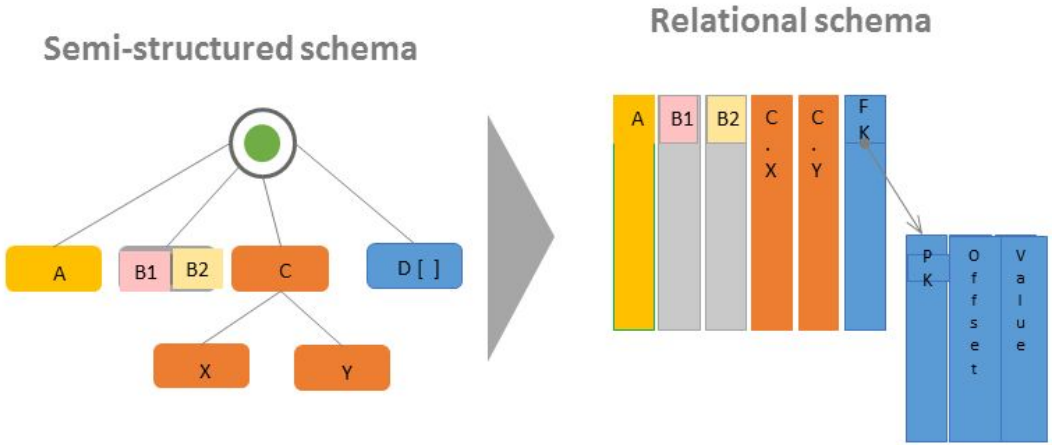
- Challenge to the Relational Database
  - The Big Data Challenge
- NoSQL Introduction
  - JSON Processing in Relational Database
  - Azure Cosmos DB for Mongo DB Lab
- Distributed Computing

JSON Processing in Relational DB

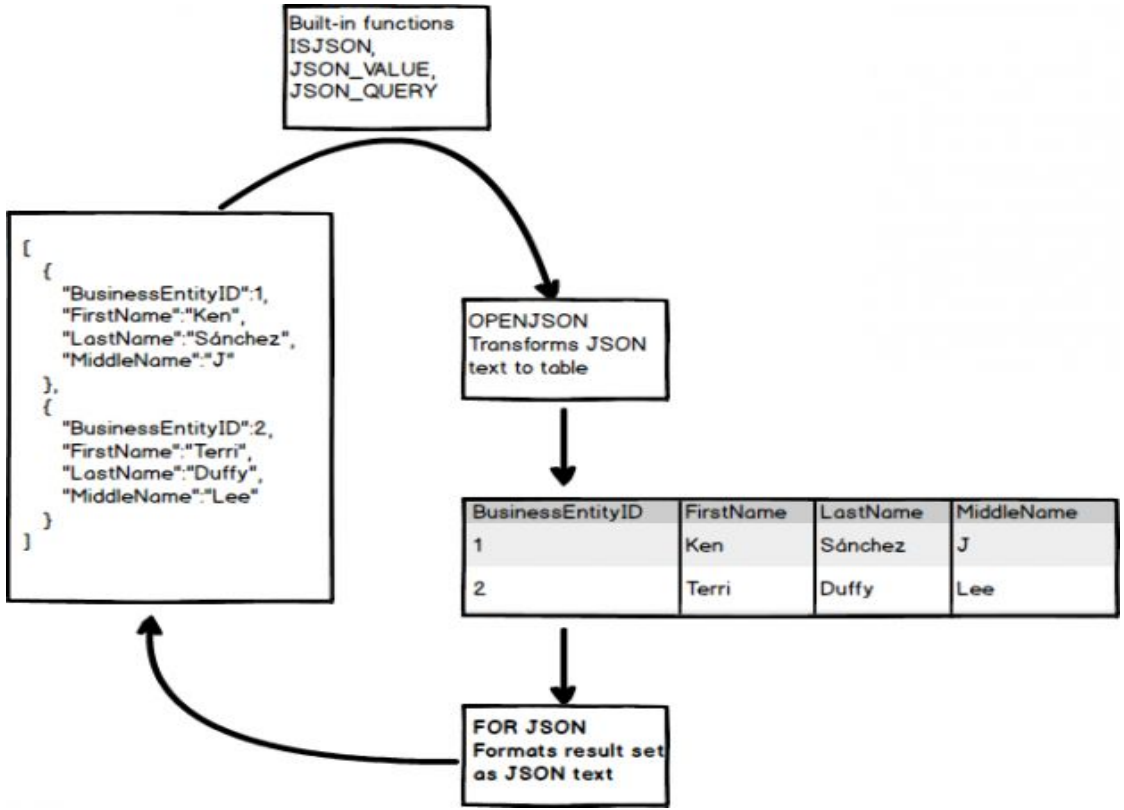
Flatten the schema

- Schemaless does not mean there is no schema
- Usually there is a core set of attributes. These attributes usually carries the identification information for the element. Put these attributes as columns in the relational database.
- Other columns can be defined using JSON related data type

JSON Processing in Relational DB



JSON Processing in Relational DB



Agenda

- Challenge to the Relational Database
  - The Big Data Challenge
- NoSQL Introduction
  - JSON Processing in Relational Database
  - Azure Cosmos DB for Mongo DB Lab
- Distributed Computing

Azure Cosmos DB for Mongo DB Lab

Create a MongoDB compatible Cosmos DB database

Use standard python to connect to the Cosmos DB and upload data

Agenda

- Challenge to the Relational Database
  - The Big Data Challenge
- NoSQL Introduction
  - JSON Processing in Relational Database
  - Azure Cosmos DB for Mongo DB Lab
- Distributed Computing

Current Trends of Data Management

- New platform
  - Open source RDBMS: MySQL, PostgreSQL
  - Cloud Computing: infrastructure, maintenance
- New Types of data management systems
  - NoSQL Databases
  - Data Lake
- Distributed computing
  - Apache Hadoop/Spark

Distributed Computing

System components are located on different networked computers, which communicate and coordinate their actions by passing messages to one another from any system. (Distributed Computing, Wikipedia)

Advantages

- Elasticity: Horizontally scalable
- Fault tolerant

Apache Hadoop Introduction

Apache Hadoop is an open source software framework for storage and large scale processing of data-sets on clusters of commodity hardware.

- Cluster: many computers (nodes) putting together, linked by Hadoop software
- Commodity hardware: regular, non-dedicated hardware

Hadoop Modules

1. Hadoop Common: contains libraries and utilities needed by other Hadoop modules
2. Hadoop Distributed File System (HDFS): a distributed file-system that stores data on the commodity machines, providing very high aggregate bandwidth across the cluster
3. Hadoop MapReduce: a programming framework for large scale data processing

HDFS: Data Storage. MapReduce: Data Analysis

Hadoop Distributed File System (HDFS)

HDFS stores large files (typically in the range of gigabytes to terabytes) across multiple machines (nodes).

- Split data into blocks
- Replicate the data blocks across multiple (default 3) hosts
  - Two on the same rack
  - One on a different rack

Data nodes can talk to each other to rebalance data, to move copies around, and to keep the replication of data high.

MapReduce

A processing technique and a program model for distributed computing.

- Two important tasks, Map and Reduce. Reduce is always performed after map.
- Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).
- Reduce takes the output from a map as an input and combines those data tuples into a smaller set of tuples.

Map Example

Algorithm:

```
A = [1, 2, 3]
map (item in A): A[item] = A[item] * 2
```

Sample Data:

```
1 => 1 * 2 => 2
2 => 2 * 2 => 4
3 => 3 * 2 => 6
```

Reduce Example

Algorithm:

```
A = [1, 2, 3, 4, 5, 6]
reduce (item in A): sum = A[item1] + A[item2]
```

Sample Data (parallel computing):

```
sum = 1 + 2 = 3   =>
sum = 3 + 4 = 7   =>   sum = 3 + 7 = 10 =>
sum = 5 + 6 = 11                        => sum = 10 + 11 = 21
```

Benefit of MapReduce

Both map and reduce can be distributed across multiple computing nodes

- Makes it easy to scale data processing over multiple computing nodes.
- There might be data shuffling among different nodes

Decomposing a data processing application into mappers and reducers is sometimes nontrivial, but...

- scaling the application is merely a configuration change: 10 -> 100 -> 1000

Spark

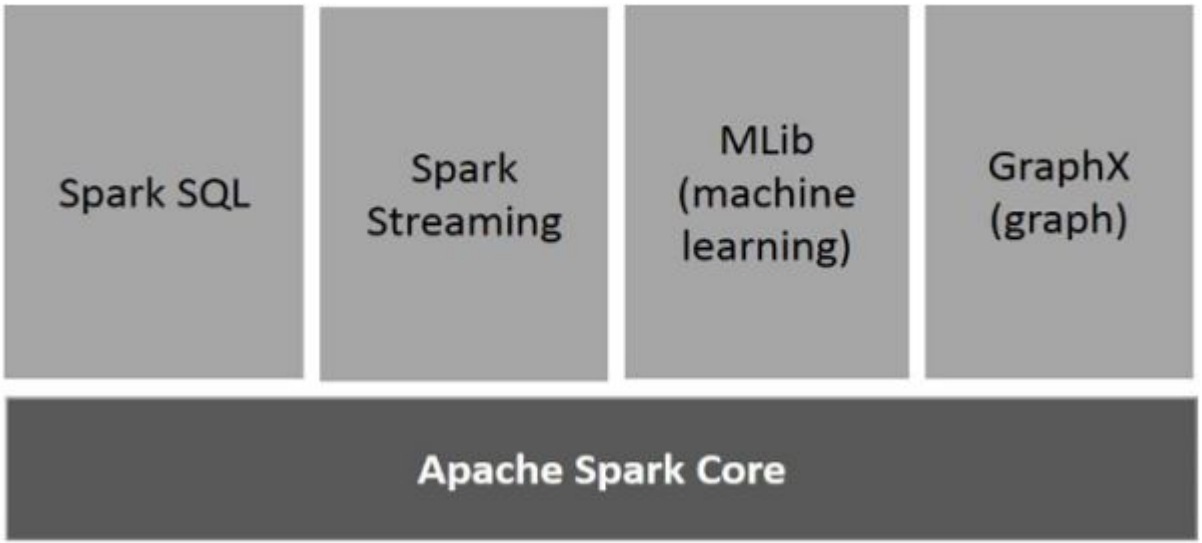
Apache Spark is based on Hadoop MapReduce.

The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application.

- Same MapReduce process. In-memory is the keyword

Spark extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing.

Spark


Spark SQL
Spark Streaming
MLib (machine learning)
GraphX (graph)
Apache Spark Core

RDD in Spark

RDD (Resilient Distributed Dataset) - RDD is the central unit of data in Apache Spark. It is a distributed collection of components across cluster nodes and can implement parallel operations.

Two operations

- Transformations - It creates a blueprint of what should happen the next time an Action is called. This is called Lazy Evaluation.
- Actions - It turns final result to driver program or corresponds it to the external data store.

RDD in Spark

RDD is an object container. It can contain any data. It is a Java collection and is very different from relational tables.

```
val dataSeq = Seq(("Java", 20000), ("Python", 100000), ("Scala", 3000))
val rdd=spark.sparkContext.parallelize(dataSeq)
```

It is hard for non Java programmers to understand and process.

Dataframe in Spark

A dataframe is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database. Thus, dataframe is easier to handle than RDD, especially with the introduction of Spark SQL.

```
df = spark.read
    .format("csv")
    .options(header='true')
    .load("/tmp/dataframe_sample.csv")
df.printSchema()
```

Dataframe in Spark

Dataframe has the same scalability as RDD. It is compatible with many big data processing technologies (Hadoop ecosystem) and ML tools (Pandas).

Since dataframe is relational, we can treat it as a 2 dimensional data table and use SQL to query it. This is called Spark SQL.

Spark vs MapReduce

Spark is widely used for ETL and Data Analysis.

MapReduce:

- Linear processing of huge data sets.
- Economical solution, if no immediate results are expected.

Spark:

- Fast data processing
- Data processing requiring compute power: Graph processing, Machine Learning, Joining

Assignment