Brief Introduction to NoSQL

NoSQL is a category of relatively new technologies and products.

NoSQL Characteristics

- Non-Relational Database
- Big Data
- Distributed Storage & Processing
- Open Source
- Less expensive hardware
- Batch Processing
  - Google Map Reduce
- Interactive and Stream Processing
  - Apache Tez Framework
  - Apache Spark
  - Facebook Presto

NoSQL Characteristics - continued

- Denormalization at ingestion to speed up query
- Append instead of update to improve performance
- Schema-agnostic

Facebook Presto

- Open source distributed SQL query engine
- Run interactive analytic queries against data sources of all sizes ranging from gigabytes to petabytes
- Designed for interactive analytics

Three V's of Big Data

- Volume: Ranges from terabytes to petabytes of data
- Variety: Includes data from a wide range of sources and formats (e.g. web logs, social media interactions, transactions, etc)
- Velocity: data needs to be collected, stored, processed, and analyzed within relatively short windows – ranging from daily to real-time

NoSQL Databases

- **Key Value**
  - Dynamo, Riak, Basho
- **Columnar**
  - Google's Bigtable, Apache's HBase (part of Hadoop)
  - Column Family/Columns
- **Document**
  - MongoDB
  - JSON/XML
- **Graph and Triple Store**
  - Neo4j
- **Analytics and Data Warehousing**
  - Hive
  - Redshift (Amazon)
  - Presto (Facebook)
  - Airpal (Airbnb)

NoSQL Database Use Cases

- **Key-value stores**
  - Simple binary values, lists, maps, and strings
- **Columnar stores**
  - Related information values can be grouped in column families
- **Document stores**
  - Highly complex parent-child hierarchal structures
- **Triple and Graph stores**
  - A web of interrelated information

NoSQL Database Application

- **Key-value stores**
  - provide easy and fast storage of simple data through use of key
- **Columnar stores**
  - support very wide tables but not relationships between tables
- **Document stores**
  - keep JSON and/or XML hierarchical structures
- **Triple and graph stores**
  - store complex relationships

Key-value Store vs Columnar Store

Key-value store

Key	Timestamp	Value

Bigtable clone

Row Key	Column Family	Column Name	Timestamp	Value

NoSQL Databases Operations

- Memory Cache
- Distributed
- Proprietary Interface
- SQL-like Interface

Example of SQL-like Interface

- Presto
- Hive QL
- Pig
- Cassandra Query Language (CQL)
- Cosmos/Scope

Hadoop with Hive vs RDBMS

Hadoop with Hive	RDBMS
• Can handle petabytes of data and unstructured data.	• Most can handle terabytes of data and only structured data
• Opens source, flexible, fast and still evolving	• Most are proprietary and defined constraints
• Supports distributed architecture	• Support client server architecture
• Can run on commodity hardware	• Data intensive applications need high-end servers
• Cost efficient	• High cost to scale
• Some traditional data handling features are not available in Hive. For example, ACID principles are not available in Hive	• Provides traditional features such as transaction management and ACID principles for data reliability

Hive QL

```
Hive> CREATE DATABASE
Employee
```

```
Hive> CREATE DATABASE
IF NOT EXISTS Employee
```

Hive QL

This view retrieves data containing details about graduate courses in New York:

```
CREATE VIEW NY_graduate_courses AS
 SELECT *
 FROM Universities_courses_all
 JOIN course_List ON (course.id = course.id )
 WHERE state = 'NY'
```

Cosmos/Scope – INNER JOIN

```
// INNER JOIN
rs_inner = SELECT employees.DepID AS EmpDepId,
                  departments.DepID,  employees.EmpName,
                  departments.DepName
           FROM employees
           INNER JOIN departments
                  ON employees.DepID == departments.DepID;
```

Cosmos/Scope – LEFT OUTER JOIN

```
// LEFT OUTER JOIN
rs_left_outer = SELECT employees.DepID AS EmpDepId,
                       departments.DepID , employees.EmpName,
                       departments.DepName
                FROM employees
                LEFT OUTER JOIN departments
                       ON employees.DepID == departments.DepID;
```