19AD501 – BIG DATA ANALYTICS

UNIT V

Introduction to NoSQL: Types of NoSQL Databases-Key-value store, Document store, Column family, Graph store, CAP theorem – CAP Theorem NoSQL databases, MongoDB: RDBMS Vs MongoDB – Mongo DB Database Model – Data Types, Sharding –Types of sharding, Introduction to Hive – Hive Architecture – Hive Query Language (HQL).

Introduction to NoSQL

The term NoSQL was first coined by Carlo Strozzi in 1998 to name his light weight, open source, non relational database that did not expose the standard SQL interface. The term was reintroduced by Eric Evans in early 2009.

NoSQL stands for Not Only SQL. These are non relational, open source, distributed databases. A NoSQL database provides a mechanism for storage and retrieval of data that employs less constrained consistency models than traditional relational database.

- NoSQL databases are non relational
- Distributed
- No Support for ACID properties
- No fixed table schema

Types of NoSQL Databases

- Key Value Stores
- Document Stores
- Column Family
- Graph Databases

Key Value

Key/value stores contain data (the value) that can be simply accessed by a given identifier.
It is a schema-less model in which values (or sets of values, or even more complex entity objects) are associated with distinct character strings called keys.

In a key/value store, there is no stored structure of how to use the data; the client that reads and writes to a key/value store needs to maintain and utilize the logic of how to meaningfully extract the useful elements from the key and the value.

The key value store does not impose any constraints about data typing or data structure—the value associated with the key is the value.

The core operations performed on a keyvalue store include:
• Get(key), which returns the value associated with the provided key.
• Put(key, value), which associates the value with the key.
• Multi-get(key1, key2,.., keyN), which returns the list of values associated with the list of keys.
• Delete(key), which removes the entry for the key from the data store.

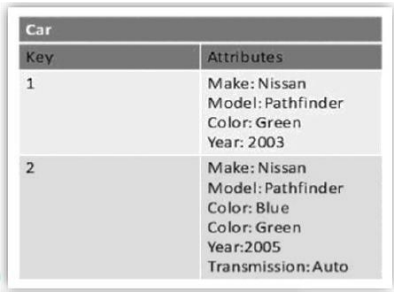
Keys can be hashed using a hash function that maps the key to a particular location (sometimes called a “bucket”) in the table.

The simplicity of the representation allows massive amounts of indexed data values to be appended to the same keyvalue table, which can then be sharded, or distributed across the storage nodes.

Drawbacks of Key Value Store
One is that the model will not inherently provide any kind of traditional database capabilities (such as atomicity of transactions, or consistency when multiple transactions are executed simultaneously)—those capabilities must be provided by the application itself.

Another is that as the model grows, maintaining unique values as keys may become more difficult, requiring the introduction of some complexity in generating character strings that will remain unique among a myriad of key.

Sample



Car	
Key	Attributes
1	Make: Nissan Model: Pathfinder Color: Green Year: 2003
2	Make: Nissan Model: Pathfinder Color: Blue Color: Green Year:2005 Transmission: Auto

Key	Value
First Name	Simmonds
Last Name	David

Example Databases – Redis, Riak, Dynamo etc.

Document Store
A document store is similar to a key value store in that stored objects are associated (and therefore accessed via) character string keys. The difference is that the values being stored, which are referred to as "documents," provide some structure and encoding of the managed data.

There are different common encodings, including XML (Extensible Markup Language), JSON (Java Script Object Notation), BSON (which is a binary encoding of JSON objects), or other means of serializing data.

Document stores are useful when the value of the key/value pair is a file and the file itself is self-describing.

One of the differences between a keyvalue store and a document store is that while the former requires the use of a key to retrieve data, the latter often provides a means (either through a programming API or using a query language) for querying the data based on the contents.

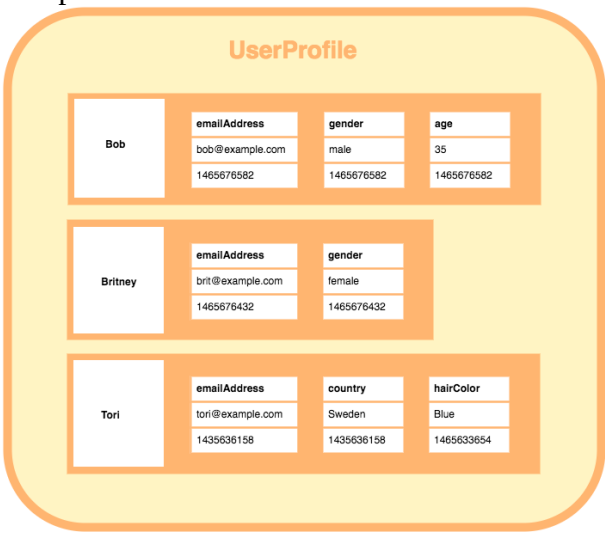
Sample document in document database
{
"Book Name": "Fundamentals of Business Analytics"

“Publisher”: “Wiley India”
“Year of Publication”: “2011”
}

Example Databases : MongoDB, Apache CouchDB, Couchbase etc.

Column Family
Each storage block has data from only one column. The column is lowest/smallest instance of data. It is a tuple that contains a name, a value and a timestamp

Sample



ColumnFamily: Authors

Key	Value
“Eric Long”	Columns: Name / Value: “email” / “eric (at) long.com”; “country” / “United Kingdom”; “registeredSince” / “01/01/2002”
“John Steward”	Columns: Name / Value: “email” / “john.steward (at) somedomain.com”; “country” / “Australia”; “registeredSince” / “01/01/2009”
“Ronald Mathies”	Columns: Name / Value: “email” / “ronald (at) sodeso.nl”; “country” / “Netherlands, The”; “registeredSince” / “01/01/2010”

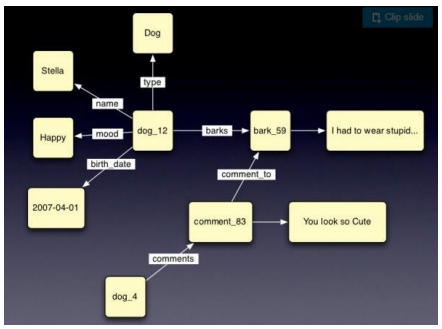
Example – Cassandra, HBase

Graph Database
Graph databases provide a model of representing individual entities and numerous kinds of relationships that connect those entities.

More precisely, it employs the graph abstraction for representing connectivity, consisting of a collection of vertices (which are also referred to as nodes or points) that represent the modeled entities, connected by edges (which are also referred to as links, connections, or relationships) that capture the way that two entities are related.

Graph analytics performed on graph data stores are somewhat different than more frequently used querying and reporting.

Sample



Example – Neo4j, Infinite Graph, HyperGraphDB

Why NoSQL

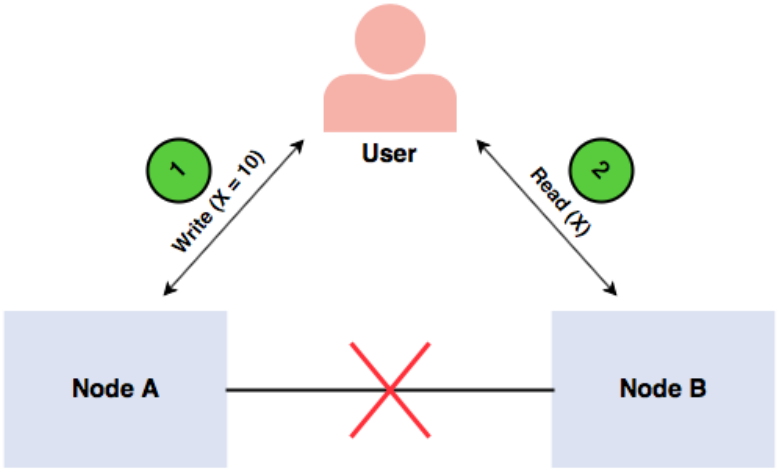
- It has scaleout architecture instead of monolithic architecture of relational databases.
- It can support structured, semi-structured and unstructured data
- Dynamic schema
- Auto Sharding
- Replication

Advantages of NoSQL

- Can easily scale up and down
- Does not require a pre-defined schema
- Cheap, Easy to implement
- Relaxes the data consistency requirements
- Data can be replicated to multiple nodes and can be partitioned

CAP Theorem

The CAP theorem, or Brewer's theorem, is a fundamental theorem within the field of system design. The CAP theorem states that a distributed system can only provide two of three properties simultaneously: **consistency, availability, and partition tolerance**. The theorem formalizes the tradeoff between consistency and availability when there's a partition.



The distributed system acts as a plain register with the value of variable X. There's a network failure that results in a network partition between the two nodes in the system. An end-user performs a write request, and then a read request. Let's examine a case where a different node of the system processes each request. In this case, our system has two options:

- It can fail at one of the requests, breaking the system's availability
- It can execute both requests, returning a stale value from the read request and breaking the system's consistency

The system can't process both requests successfully while also ensuring that the read returns the latest value written by the write. This is because the results of the write operation can't be

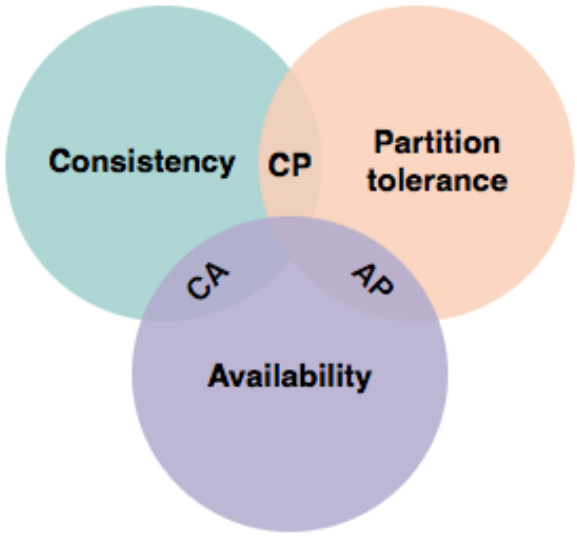
propagated from node A to node B because of the network partition.

Consistency
In a consistent system, all nodes see the same data simultaneously. If we perform a read operation on a consistent system, it should return the value of the most recent write operation. The read should cause all nodes to return the same data. All users see the same data at the same time, regardless of the node they connect to. When data is written to a single node, it is then replicated across the other nodes in the system.

Availability
When availability is present in a distributed system, it means that the system remains operational all of the time. Every request will get a response regardless of the individual state of the nodes. This means that the system will operate even if there are multiple nodes down. Unlike a consistent system, there's no guarantee that the response will be the most recent write operation.

Partition tolerance
When a distributed system encounters a partition, it means that there's a break in communication between nodes. If a system is partition-tolerant, the system does not fail, regardless of whether messages are dropped or delayed between nodes within the system. To have partition tolerance, the system must replicate records across combinations of nodes and networks.

CAP theorem NoSQL databases



NoSQL databases are great for distributed networks. They allow for horizontal scaling, and they can quickly scale across multiple nodes. When deciding which NoSQL database to use, it's important to keep the CAP theorem in mind. NoSQL databases can be classified based on the two CAP features they support

Consistency and Availability databases
CA databases enable consistency and availability across all nodes. Unfortunately, CA databases can't deliver fault tolerance. In any distributed system, partitions are bound to happen, which means this type of database isn't a very practical choice. Some relational

databases, such as **PostgreSQL**, **MySQL** etc allow for consistency and availability.

Consistency and Partition tolerance databases
CP databases enable consistency and partition tolerance, but not availability. When a partition occurs, the system has to turn off inconsistent nodes until the partition can be fixed.
Example - MongoDB, HBase,Redis, BigTable

Availability Partition tolerance databases
AP databases enable availability and partition tolerance, but not consistency. In the event of a partition, all nodes are available, but they're not all updated. For example, if a user tries to access data from a bad node, they won't receive the most up-to-date version of the data.
Example-Riak, Cassandra, CouchDB, DynamoDB

MongoDB

MongoDB is cross platform, open source, non-relational, distributed, NoSQL and document oriented data store. It provides, high performance, high availability, and easy scalability. MongoDB works on concept of **collection and document**.

Database
Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection
Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document
Document is analogous to row/record/tuple in RDBMS table. A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

RDBMS Vs MongoDB

- **RDBMS** has a typical **schema design** that shows number of tables and the relationship between these tables whereas **MongoDB** is document-oriented. There is no concept of **schema or relationship**.

- Complex transactions are not supported in MongoDB because complex join operations are not available.

- MongoDB allows a highly flexible and scalable document structure. For example, one data document of a collection in MongoDB can have two fields whereas the other document in the same collection can have four.

- MongoDB is faster as compared to RDBMS due to efficient indexing and storage techniques.

- There are a few terms that are related in both databases. What's called Table in RDBMS is called a Collection in MongoDB.

- Similarly, a Tuple is called a Document and A Column is called a Field.

- MongoDB provides a default '_id' (if not provided explicitly) which is a 12-byte hexadecimal number that assures the uniqueness of every document. It is similar to the Primary key in RDBMS.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)
Database Server and Client	
mysqld/Oracle	mongod
mysql/sqlplus	mongo

Why use MongoDB – Features of MongoDB
Document Oriented: MongoDB stores the main subject in the minimal number of documents and not by breaking it up into multiple relational structures like RDBMS.

Indexing: Without indexing, a database would have to scan every document of a collection to select those that match the query which would be inefficient. So, for efficient searching Indexing is a must and MongoDB uses it to process huge volumes of data in very less time.

Scalability: MongoDB scales horizontally using sharding (partitioning data across various servers). Data is partitioned into data chunks using the shard key, and these data chunks are evenly distributed across shards that reside across many physical servers.

Replication and High Availability: MongoDB increases the data availability with multiple copies of data on different servers. By providing redundancy, it protects the database from hardware failures. If one server goes down, the data can be retrieved easily from other active servers which also had the data stored on them.

Aggregation: Aggregation operations process data records and return the computed results. It

is similar to the GROUPBY clause in SQL. A few aggregation expressions are sum, avg, min, max, etc

MongoDB Database Model

Data modeling is the process of defining how data is stored and what relationships exist between different entities in our data.

Data modeling aims to visually represent the relationship between different entities in data.

MongoDB provides two types of data models:

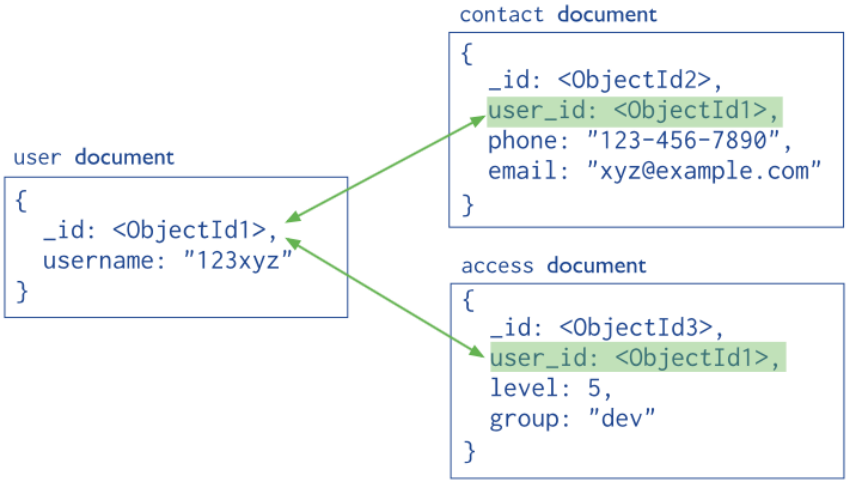
- Embedded data model and
- Normalized data model or Referenced Data model

Embedded Data Model

In this model, you can have (embed) all the related data in a single document, it is also known as **de-normalized data model**. It is applied when two data sets contain a relationship. Hence an embedded data model sets relationships between data elements, keeping documents in a single document structure.

Normalized Data Model or Referenced Data model

In this model, you can refer the sub documents in the original document, using references. Object references are used to model relationships between data elements/documents. This model reduces duplication of data.



MongoDB - Data Types and Sharding

Data Types

MongoDB supports many datatypes. Some of them are

- **String** − This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- **Integer** − This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** − This type is used to store a boolean (true/ false) value.
- **Double** − This type is used to store floating point values.
- **Min/ Max keys** − This type is used to compare a value against the lowest and highest BSON elements.
- **Arrays** − This type is used to store arrays or list or multiple values into one key.
- **Timestamp** − ctimestamp. This can be handy for recording when a document has been modified or added.
- **Object** − This datatype is used for embedded documents.
- **Null** − This type is used to store a Null value.
- **Symbol** − This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
- **Date** − This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- **Object ID** − This datatype is used to store the document's ID.
- **Binary data** − This datatype is used to store binary data.
- **Code** − This datatype is used to store JavaScript code into the document.
- **Regular expression** − This datatype is used to store regular expression.

Sharding

Sharding is a database architecture pattern related to horizontal partitioning or horizontal scaling. It divides large datasets and distributed over multiple servers or shards. Each shard is independent database and collectively they would constitute a logical database.

Database sharding is a type of horizontal partitioning that splits large databases into smaller components, which are faster and easier to manage.
A shard is an individual partition that exists on separate database server instance to spread load.

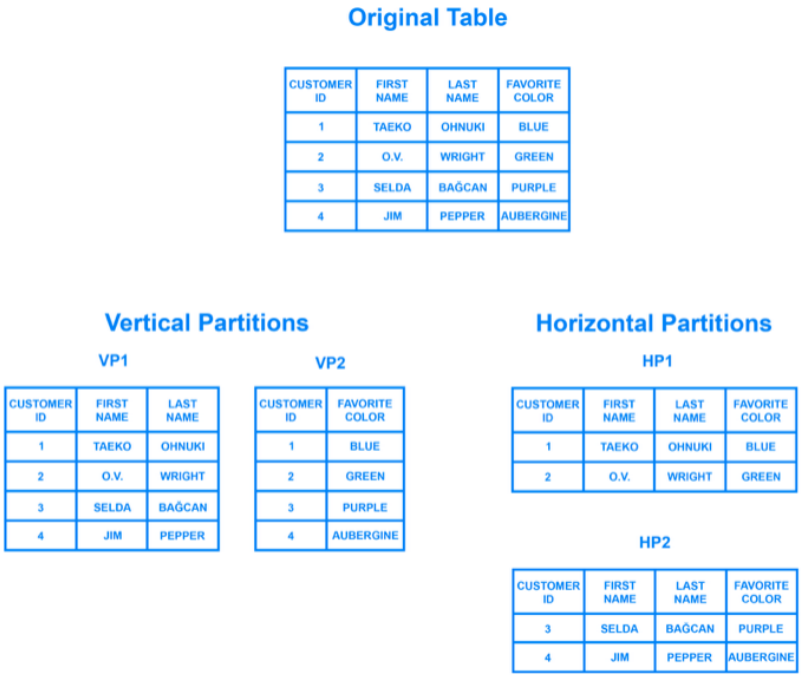
Auto sharding or data sharding is needed when a dataset is too big to be stored in a single database.

Why Sharding?
As both the database size and number of transactions increase, so does the response time for querying the database. Costs associated with maintaining a huge database can also skyrocket due to the number and quality of computers you need to manage your workload.

Data shards, on the other hand, have fewer hardware and software requirements and can be

managed on less expensive servers.

Sharding Example

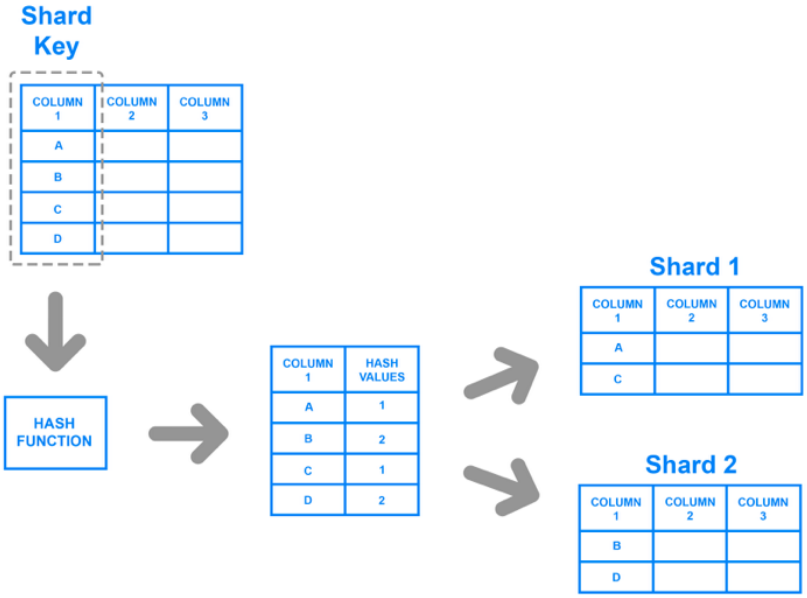


Sharding Architectures

Key Based Sharding

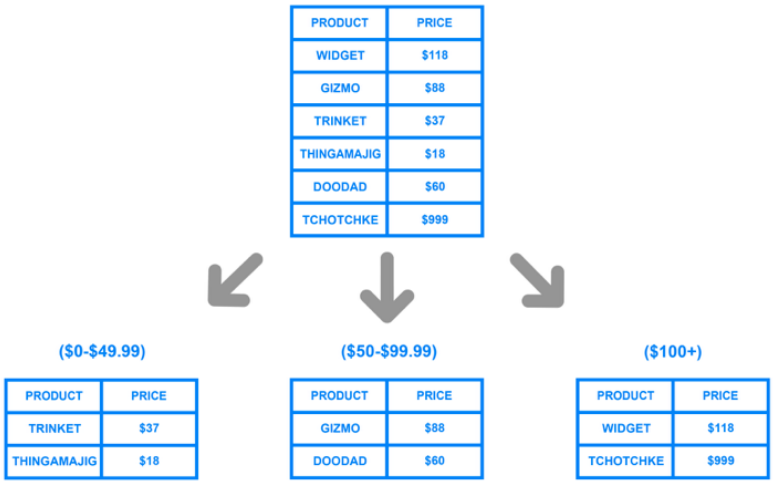
Key based sharding, also known as hash based sharding, involves using a value taken from newly written data — such as a customer's ID number, a client application's IP address, a ZIP code, etc. — and plugging it into a hash function to determine which shard the data should go to.

A hash function is a function that takes as input a piece of data (for example, a customer email) and outputs a discrete value, known as a hash value



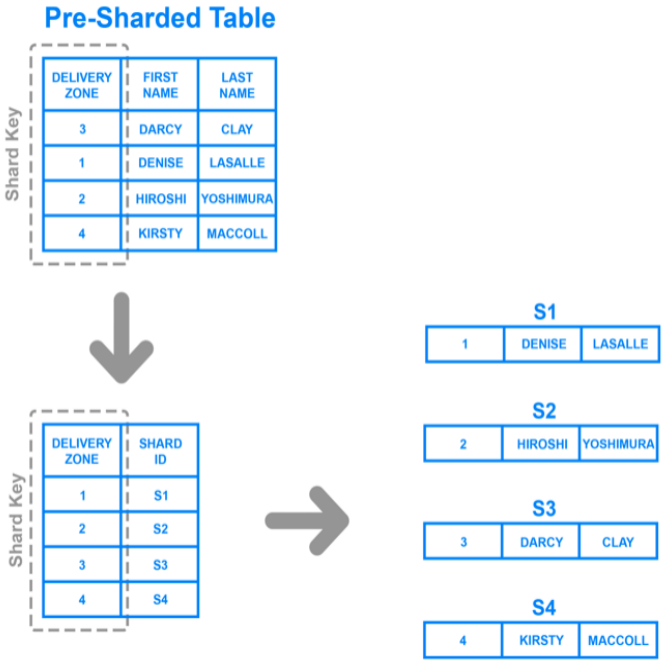
Range Based Sharding

- Range based sharding involves sharding data based on ranges of a given value.
- The main benefit of range based sharding is that it's relatively simple to implement.
- Every shard holds a different set of data but they all have an identical schema as one another, as well as the original database.



Directory Based Sharding

- To implement directory based sharding, one must create and maintain a lookup table that uses a shard key to keep track of which shard holds which data.
- The main appeal of directory based sharding is its flexibility. Range based sharding architectures limit you to specifying ranges of values, while key based ones limit you to using a fixed hash function which, as mentioned previously, can be exceedingly difficult to change later on.
- Directory based sharding, on the other hand, allows you to use whatever system or algorithm you want to assign data entries to shards, and it's relatively easy to dynamically add shards using this approach.



Introduction to Hive

Apache Hive is a distributed, fault-tolerant data warehouse system that enables analytics at a massive scale., and is used for analyzing structured and semi-structured data. It was developed by the Data Infrastructure Team at Facebook. Later, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive.

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.

Hive makes use of the following

- HDFS for storage
- MapReduce for execution
- Stores meta data in RDBMS

Hive runs SQL like queries called **HQL (Hive query language)** which gets internally converted to MapReduce jobs and then runs the job into hadoop cluster. What makes Hive unique is the ability to query large datasets, leveraging Apache Tez or MapReduce, with a SQL-like interface.

Hive was created to allow non-programmers familiar with SQL to work with petabytes of data, using a SQL-like interface called HiveQL.

The Hive system provides tools for extracting/ transforming/loading data (ETL) into a variety of different data formats.

Features of Hive

- It is similar to SQL, HQL is easy to code.
- Hive supports rich data types such as struct, list and maps.
- It stores schema in a database and processed data into HDFS.
- It is designed for OLAP.
- It provides SQL type language for querying called HiveQL or HQL.
- It is familiar, fast, scalable, and extensible.

Hive Data Units
Database – The namespace for tables
Tables – Set of records that have similar schema
Partitions – Logical separation of data based on classification of given information as per specific attributes.
Buckets (or Clusters) – Similar to partitions but uses hash function to segregate data and determines buckets or cluster into which the record should be placed.

Hive Architecture

Hive contains the following parts

Hive Clients:
Apache Hive supports different types of client applications for performing queries on the

Hive. Hive supports application written in many languages like Java, C++, Python etc. using JDBC, Thrift and ODBC drivers. Hence one can always write hive client application written in a language of their choice. These clients can be categorized into three types:

Thrift Clients: As Hive server is based on Apache Thrift, it can serve the request from all those programming language that supports Thrift.

JDBC Clients: Hive allows Java applications to connect to it using the JDBC driver.

ODBC Clients: The Hive ODBC Driver allows applications that support the ODBC protocol to connect to Hive.

Hive Services

Hive Command Line Interface (CLI) – The most commonly used interface to interact with Hive. This is the default shell provided by the Hive where you can execute your Hive queries and commands directly.

Hive Web Interface – It is simple graphical user interface to interact with Hive and to execute query.

Hive Server – It is referred to as Apache **Thrift Server**. It accepts the request from different clients and provides it to Hive Driver.

Driver - It is responsible for receiving the queries submitted through the CLI, the web UI, Thrift, ODBC or JDBC interfaces by a client. Hive queries are sent to the driver for compilation, optimization and execution. Execution engine executes these tasks in the order of their dependencies, using Hadoop.

Metastore – The Metastore stores the information about the tables, partitions, the columns within the tables. Hive table definitions and mappings to the data are stored in Metastore. A metastore consists the following

Metastore service – offers interface to the hive

Databases – Stores data and definitions mapping to the data and others.

Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping

There are 3 ways of storing in Metastore: **Embedded Metastore, Local Metastore and Remote Metastore.** Mostly, Remote Metastore will be used in production mode

Embedded Metastore

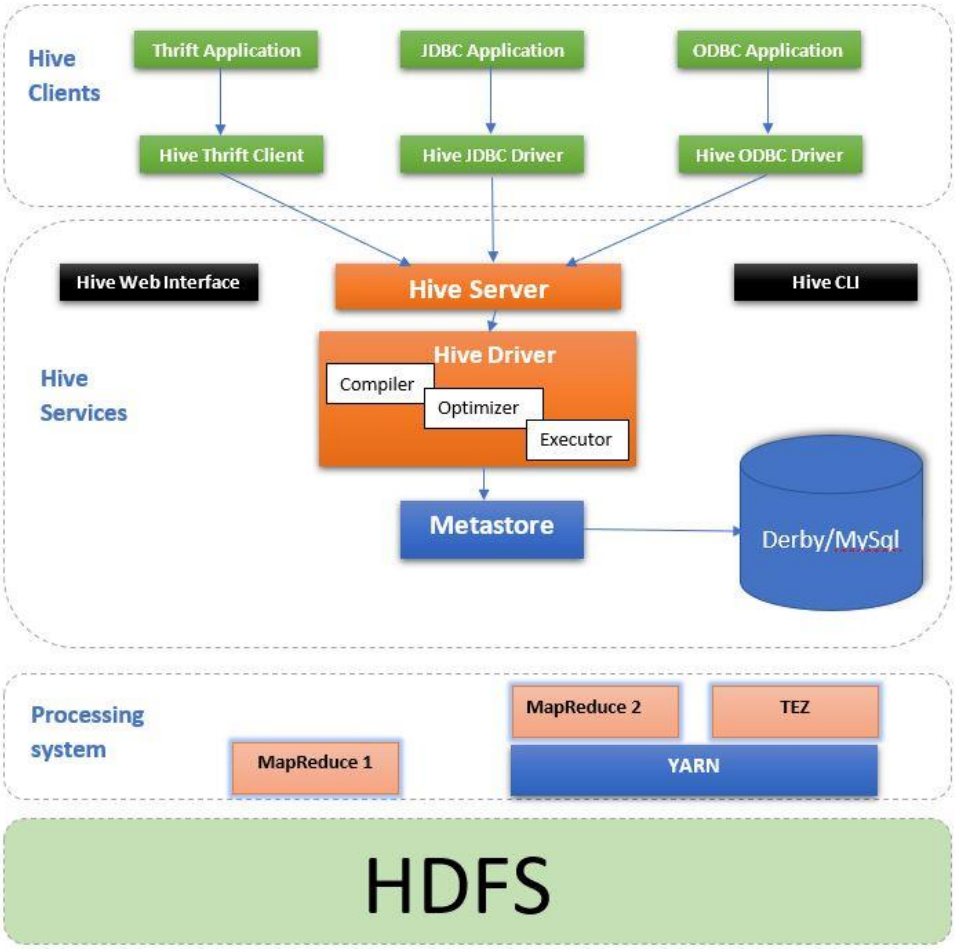
Both the metastore service and the Hive service runs in the same JVM by default using an embedded Derby Database instance where metadata is stored in the local disk. This is called embedded metastore configuration. In this case, only one user can connect to metastore database at a time.

Local Metastore

This configuration allows us to have multiple Hive sessions i.e. Multiple users can use the metastore database at the same time. This is achieved by using any JDBC compliant database like MySQL which runs in a separate JVM or a different machine than that of the Hive service and metastore service which are running in the same JVM.

Remote Metastore

In the remote metastore configuration, the metastore service runs on its own separate JVM and not inthe Hive service JVM. Other processes communicate with the metastore server using Thrift Network APIs.



Processing framework and Resource Management: Internally, Hive uses Hadoop MapReduce framework as de facto engine to execute the queries.

Distributed Storage: As Hive is installed on top of Hadoop, it uses the underlying HDFS for the distributed storage.

Advantages of Hive

- Useful for people who aren't from a programming background as it eliminates the need to write complex MapReduce program.
- Extensible and scalable to cope up with the growing volume and variety of data, without affecting performance of the system.
- It is as an efficient ETL (Extract, Transform, Load) tool.
- Hive supports any client application written in Java, PHP, Python, C++ or Ruby by exposing its Thrift server.
- As the metadata information of Hive is stored in an RDBMS, it significantly reduces the time to perform semantic checks during query execution.

Limitations of Hive

- Hive has the following limitations and cannot be used under such circumstances:
- Not designed for online transaction processing.
- Provides acceptable latency for interactive data browsing.
- Does not offer real-time queries and row level updates.
- Latency for Hive queries is generally very high.

Hive Query Language (HQL)

Hive Query Language (HiveQL) is a query language in Apache Hive for processing and analyzing structured data. It separates users from the complexity of Map Reduce programming. It reuses common concepts from relational databases, such as tables, rows, columns, and schema, to ease learning. Hive provides a CLI for Hive query writing using Hive Query Language (HiveQL).

Hive query language provides basic SQL like operations. Tasks performed by HQL

- Create and manage tables and partitions
- Support various relational arithmetic and logical operators
- Evaluate functions
- Download the contents of a task to local directory

Hive uses derby database for single user metadata storage, and for multiple user Metadata or shared Metadata case, Hive uses MYSQL.

DDL (Data Definition Language) Statements

These statements are used to build and modify the tables and other objects in the database. The DDL commands are as follows

- Create / Drop / Alter Database
- Create / Drop / Truncate Table
- Alter Table / Partition / Column
- Create / Drop / Alter View
- Create / Drop / Alter Index
- Show
- Describe

DML (Data Manipulation Language) Statements

These statements are used to retrieve, store, modify, delete and update data in database. The DML commands are as follows

- Loading files into tables
- Inserting data into Hive tables from queries

Create Database Statement

Create Database is a statement used to create a database in Hive. A database in Hive is a namespace or a collection of tables. The syntax for this statement is as follows:

CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>

Example
hive> CREATE DATABASE [IF NOT EXISTS] userdb;
It creates userdb database

Show Database
To display a list of all databases
Hive> show databases;

Describe Databases
To describe a database
Hive> describe database student

Alter database
To alter the database properties

Hive> Alter database student set DBproperties('edited-by'="James");

Drop
To drop database
Hive> drop database student;

Tables
Hive provides two kinds of tables – Managed and External Table

Managed Table
Hive stores managed tables under the warehouse folder under Hive
The complete lifecycle of the table and data is managed by Hive.
When the internal table is dropped it drops the data aswell as meta data.

Create Table
To create table named student
Hive> CREATE TABLE STUDENT IF NOT EXISTS STUDENT(rollno INT, name STRING, gpa FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';

External Table
When the table dropped it retains the data in the underlying locations.
EXTERNAL keyword is used to create external table
LOCATION needs to be specified to store the dataset in that particular location.

Create Table
Hive> CREATE EXTERNAL TABLE STUDENT IF NOT EXISTS STUDENT (rollno INT, name STRING, gpa FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LOCATION '/STUDENT_INFO';

Querying table
To retrieve the student details from "EXT_STUDENT" table
Hive> SELECT * from EXT_STUDENT;