BIG DATA ANALYTICS

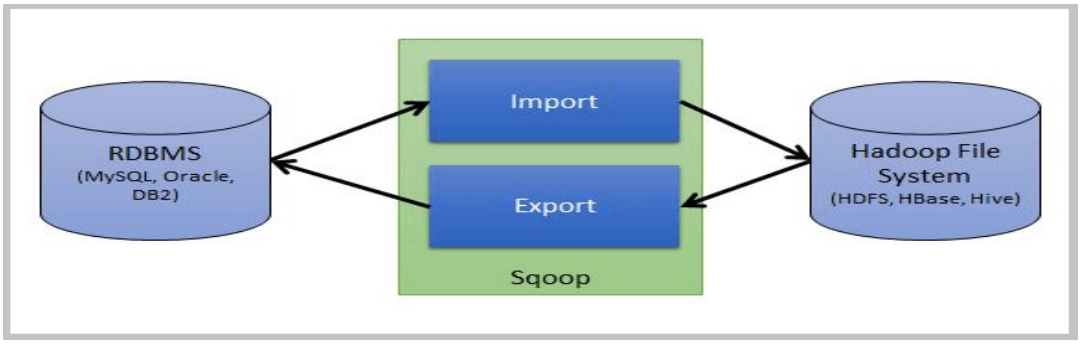
UNIT-III

SQOOP: Introduction to SQOOP, SQOOP Imports: From Database to HDFS/Hive, SQOOP Exports:From HDFS/Hive to Database, Incremental imports

NoSQL & HBase: Overview, HBase Architecture, CRUD Operations

Introduction to SQOOP :

- When Big Data storages and analyzers such as MapReduce, Hive, HBase, Cassandra, Pig, etc. of the Hadoop ecosystem came into picture, they required a tool to interact with the relational database servers for importing and exporting the Big Data residing in them. The **problems** administrators encountered using **Hadoop included**:
  - Maintaining data consistency
  - Ensuring efficient utilization of resources
  - Loading bulk data to Hadoop was not possible
  - Loading data using scripts was slow
- The solution was **SQOOP.** Using Sqoop in Hadoop helped to overcome all the challenges of the traditional approach and it could load **bulk data from RDBMS to Hadoop with ease.**
- Here, **Sqoop** occupies a place in the Hadoop ecosystem to provide feasible interaction between relational database server and Hadoop's HDFS.
- **Sqoop** is a tool used to transfer bulk data between Relational Database Management systems to Hadoop and vice versa. Two operations performed in Sqoop are **Import & Export.**



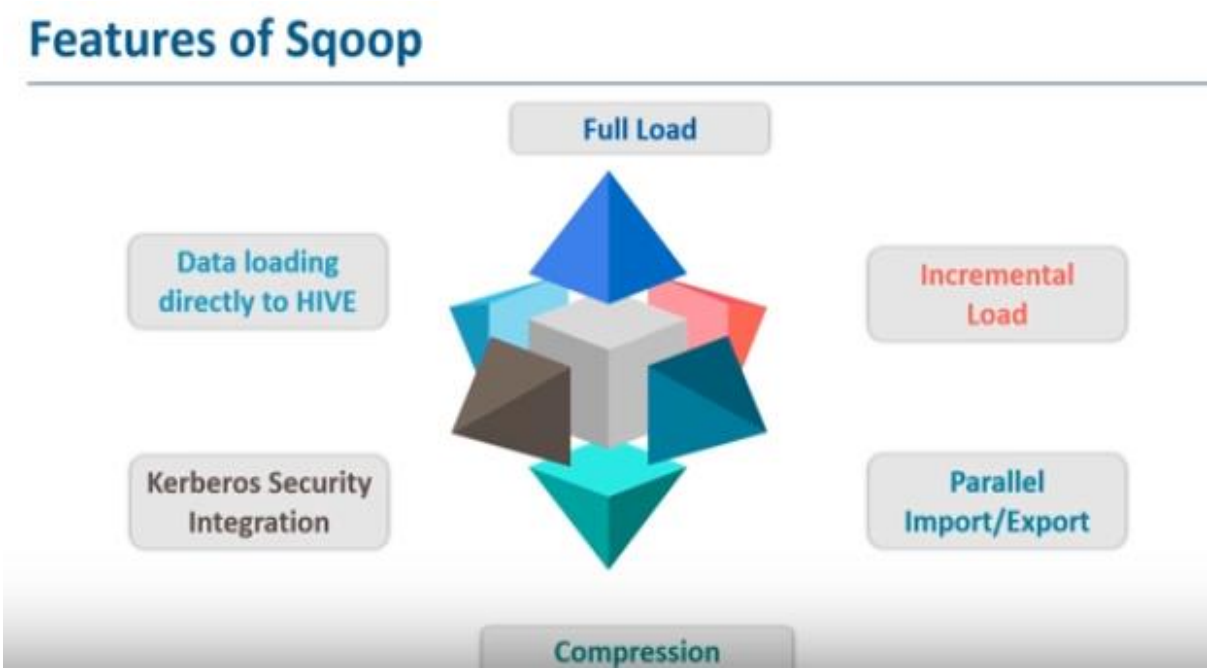
SQL+HADOOP = SQOOP

(First two letters in SQL & Last three letters in Hadoop form **SQOOP**).

- Sqoop ("SQL-to-Hadoop") is a straightforward command-line tool. It offers the following capabilities:

- Generally, helps to Import individual tables or entire databases to files in HDFS
- Also can Generate Java classes to allow you to interact with your imported data
- Moreover, it offers the ability to import from SQL databases straight into your **Hive** data warehouse.

SQOOP FEATURES:



Sqoop has several features, which makes it helpful in the Big Data world:

1. **Parallel Import/Export:** Sqoop uses the YARN framework to import and export data. This provides fault tolerance on top of parallelism.
2. **Import Results of an SQL Query:** Sqoop enables us to import the results returned from an SQL query into HDFS.
3. **Connectors For All Major RDBMS Databases:** Sqoop provides connectors for multiple RDBMSs, such as the MySQL and Microsoft SQL servers.

4. **Kerberos Security Integration :**Sqoop supports the Kerberos computer network authentication protocol, which enables nodes communication over an insecure network to authenticate users securely.

5. **Provides Full and Incremental Load :**

Incremental Load: Moreover, we can load parts of table whenever it is updated. Since Sqoop offers the facility of the incremental load.

Full Load: It is one of the important features of sqoop, in which we can load the whole table by a single command in Sqoop. Also, by using a single command we can load all the tables from a database.

6.**Load data directly into HIVE/HBase:**

Basically, for analysis, we can load data directly into Apache Hive. Also, can dump your data in HBase, which is a NoSQL database.

7. **Compression:** By using deflate(gzip) algorithm with –compress argument, We can compress your data. Moreover, it is also possible by specifying –compression-codec argument. In addition, we can also load compressed table in Apache Hive.

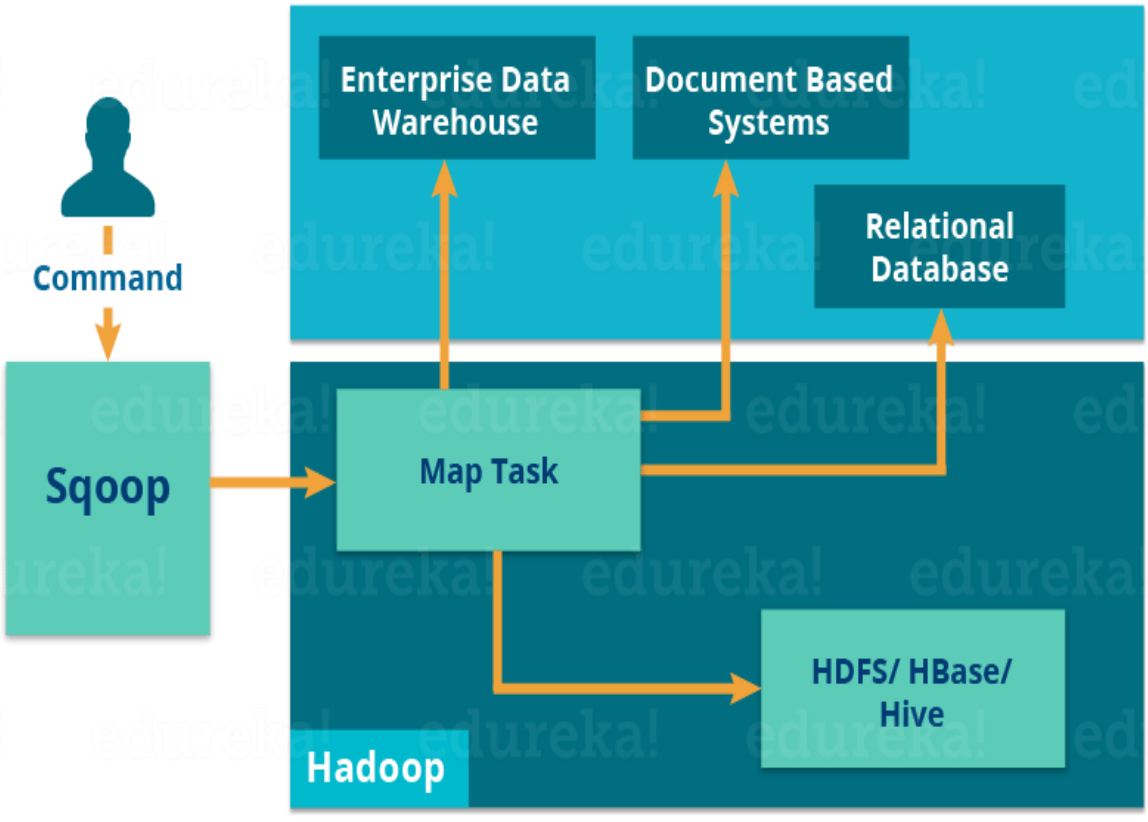
Advantages of Sqoop :

- With the help of Sqoop, we can perform transfer operations of data with a variety of structured data stores like Oracle etc.
- Sqoop helps us to perform Extract, transform, and load (ETL) operations in a very fast and cost-effective manner.
- With the help of Sqoop, we can perform parallel processing of data which leads to fasten the overall process.
- Sqoop uses the MapReduce mechanism for its operations which also supports fault tolerance.

Disadvantages of Sqoop :

- The failure occurs during the implementation of operation needed a special solution to handle the problem.
- The Sqoop uses JDBC (Java™ database connectivity)connection to establish a connection with the relational database management system which is an inefficient way.
- The performance of Sqoop export operation depends upon hardware configuration relational database management system.

Sqoop Architecture:



Procedure:

- The client submits the import/ export command to import or export data

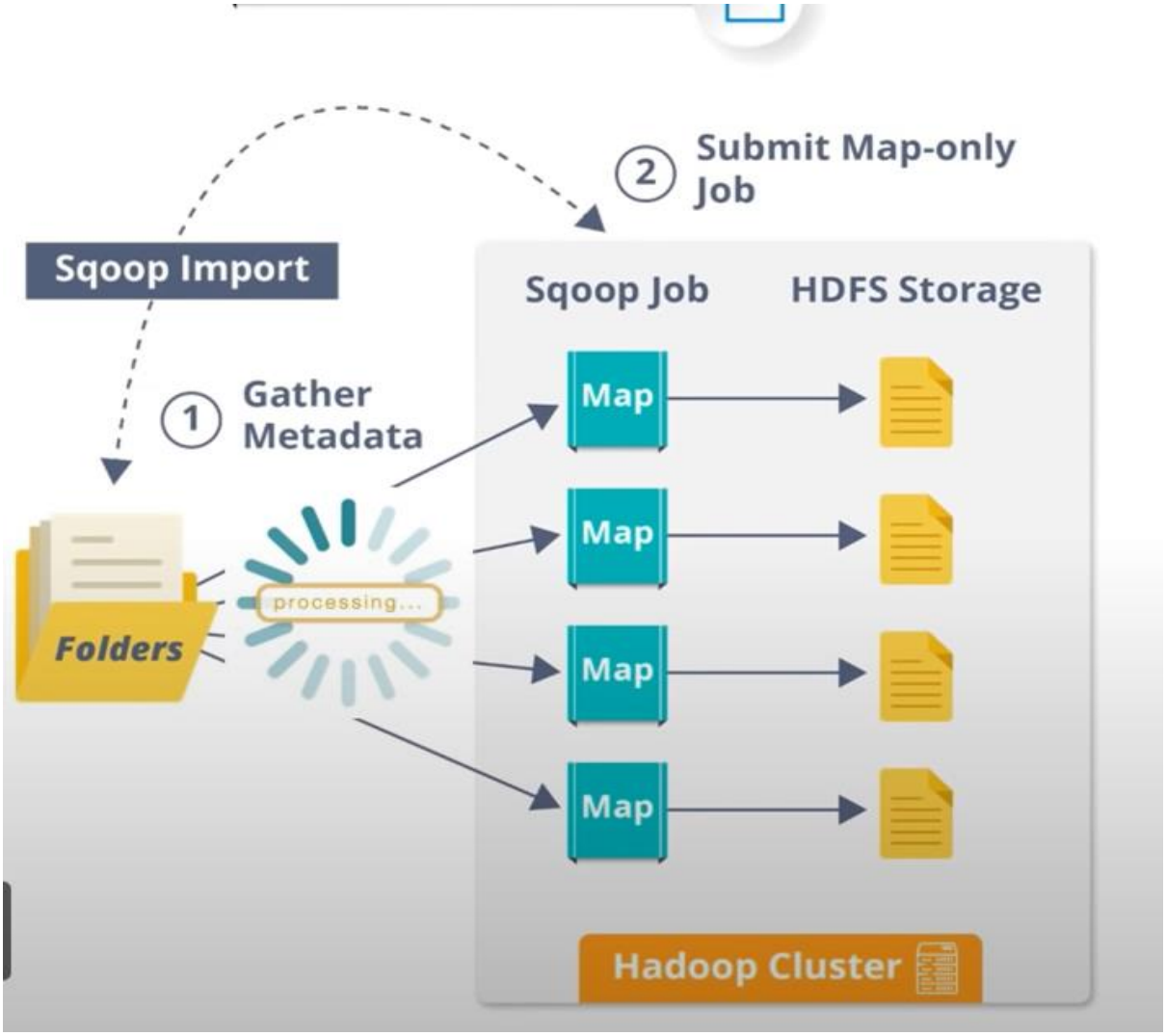
- Sqoop fetches data from different databases. Here, we have an enterprise data warehouse, document-based systems, and a relational database. We have a connector for each of these, connectors help to work with a range of accessible databases.
- Multiple mappers perform map tasks to load the data on to HDFS.
- Similarly, numerous map tasks will export the data from HDFS on to RDBMS using the Sqoop export command.

SQOOP IMPORT:

- Basically, a tool which imports individual tables from RDBMS to HDFS is what we call **Sqoop import tool.**
- The import tool imports individual tables from RDBMS to HDFS.
- However, in HDFS we treat each row in a table as a record.
- Each row in a table is treated as a record in HDFS.
- All records are stored as text data in text files or as binary data in Sequence files.
- It can import all tables, a single table, or a portion of a table into HDFS.
- In addition, in case of aggregations, we require reducing phase.
- However, Sqoop does not perform any aggregations it just imports and exports the data. Also, on the basis of the number defined by the user, map job launch multiple mappers.
- In addition, each mapper task will be assigned with a part of data to be imported for Sqoop import.
- Also, to get high-performance Sqoop distributes the input data among the mappers equally.
- Afterwards, by using JDBC each mapper creates the connection with the database. Also fetches the part of data assigned by Sqoop. Moreover, it writes it into HDFS or Hive or HBase on the basis of arguments provided in the CLI.

Working:

- Sqoop import command helps in implementation of the operation.
- In this example, a company's data is present in the RDBMS. All this metadata is sent to the Sqoop import.
- Scoop then performs an introspection of the database to gather metadata (primary key information).
- It then submits a map-only job.
- Sqoop divides the input dataset into splits and uses individual map tasks to push the splits to HDFS.
- Thus data is imported from RDBMS to Hadoop.



Syntax

The following syntax is used to **import data into HDFS.**

```
$ sqoopimport(generic-args)(import-args)
$ sqoop-import(generic-args)(import-args)
```

Example

Let us take an example of three tables named as **emp**, **emp_add**, and **emp_contact**, which are in a database called userdb in a MySQL database server.

The three tables and their data are as follows.

emp:

id	name	deg	salary	dept
1201	gopal	manager	50,000	TP
1202	manisha	Proof reader	50,000	TP
1203	khalil	php dev	30,000	AC
1204	prasanth	php dev	30,000	AC
1204	kranthi	admin	20,000	TP

emp_add:

id	hno	street	city
1201	288A	vgiri	jublee
1202	108I	aoc	sec-bad
1203	144Z	pgutta	hyd
1204	78B	old city	sec-bad
1205	720X	hitec	sec-bad

emp_contact:

id	phno	email
1201	2356742	gopal@tp.com
1202	1661663	manisha@tp.com
1203	8887776	khalil@ac.com
1204	9988774	prasanth@ac.com
1205	1231231	kranthi@tp.com

Importing a Table

Sqoop tool ‘import’ is used to import table data from the table to the Hadoop file system as a text file or a binary file.

The following command is used to import the **emp** table from MySQL database server to HDFS.

```
$ sqoop import \
--connect jdbc:mysql://localhost/userdb \
--username root \
--table emp --m 1
```

To verify the imported data in HDFS, use the following command.

```
$ $HADOOP_HOME/bin/hadoop fs -cat /emp/part-m-*
```

It shows you the **emp** table data and fields are separated with comma (,).

```
1201, gopal,    manager, 50000, TP
1202, manisha,  preader, 50000, TP
1203, kalil,    php dev, 30000, AC
1204, prasanth, php dev, 30000, AC
1205, kranthi,  admin,   20000, TP
```

Importing into Target Directory

- We can specify the target directory while importing table data into HDFS using the Sqoop import tool.
- Following is the syntax to specify the target directory as option to the Sqoop import command.

```
--target-dir <new or exist directory in HDFS>
```

The following command is used to import **emp_add** table data into '/queryresult' directory.

```
$ sqoop import \
--connect jdbc:mysql://localhost/userdb \
--username root \
--table emp_add \
--m 1 \
--target-dir /queryresult
```

The following command is used to verify the imported data in /queryresult directory form **emp_add** table.

```
$ $HADOOP_HOME/bin/hadoop fs -cat /queryresult/part-m-*
```

It will show you the emp_add table data with comma (,) separated fields.

```
1201, 288A, vgiri,   jublee
1202, 108I, aoc,     sec-bad
1203, 144Z, pgutta,  hyd
1204, 78B,  oldcity, sec-bad
1205, 720C, hitech,  sec-bad
```

Import Subset of Table Data

- We can import a subset of a table using the 'where' clause in Sqoop import tool. It executes the corresponding SQL query in the respective database server and stores the result in a target directory in HDFS.

The syntax for where clause is as follows.

```
--where <condition>
```

- The following command is used to import a subset of **emp_add** table data. The subset query is to retrieve the employee id and address, who lives in Secunderabad city.

```
$ sqoop import \
--connect jdbc:mysql://localhost/userdb \
--username root \
--table emp_add \
--m 1 \
--where “city =’sec-bad’” \
--target-dir /wherequery
```

The following command is used to verify the imported data in /wherequery directory from the **emp_add** table.

```
$ $HADOOP_HOME/bin/hadoop fs -cat /wherequery/part-m-*
```

It will show you the **emp_add** table data with comma (,) separated fields.

```
1202, 108I, aoc,    sec-bad
1204, 78B,  oldcity, sec-bad
1205, 720C, hitech,  sec-bad
```

Table 1. **Sqoop Import – Common arguments**

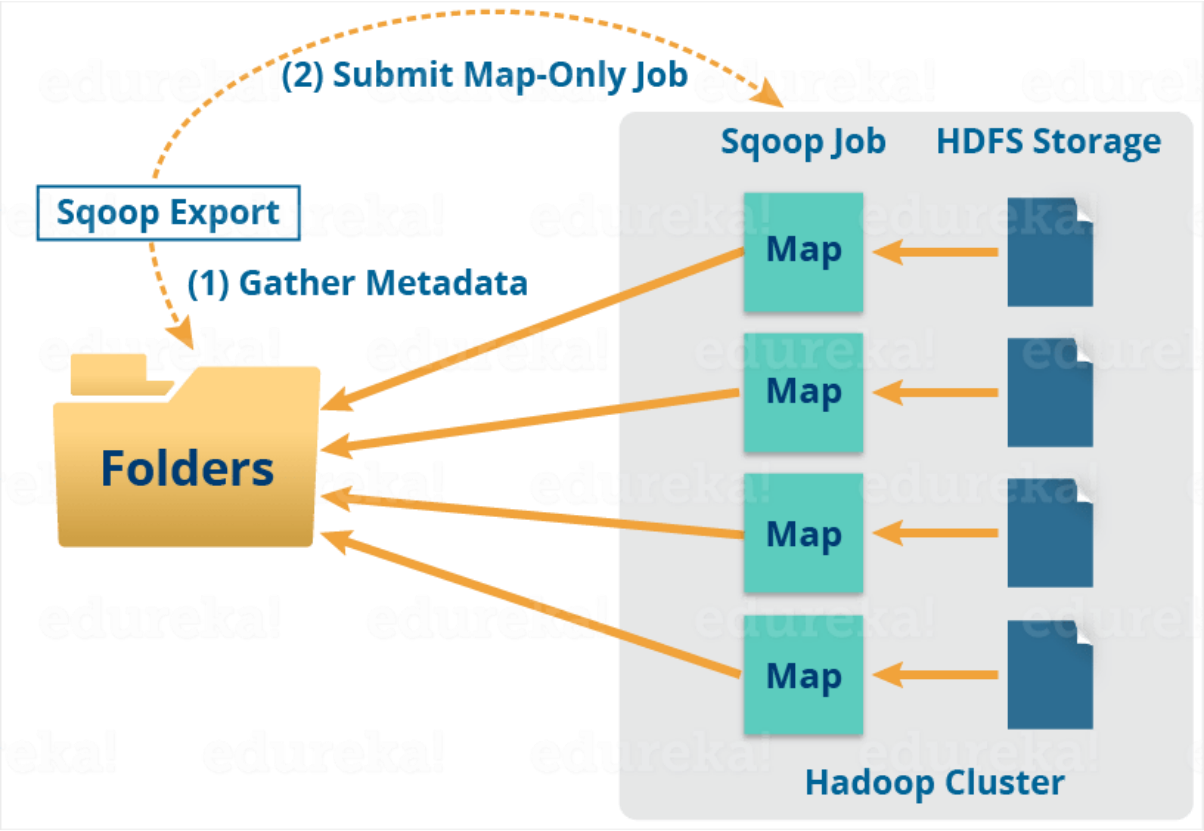
Argument	Description
–connect <jdbc-uri>	Specify JDBC connect string
–connection-manager <class-name>	Specify connection manager class to use
–driver <class-name>	Manually specify JDBC driver class to use
–hadoop-mapped-home <dir>	Override $HADOOP_MAPRED_HOME
–help	Print usage instructions
–password-file	Set path for a file containing the authentication password
-P	Read password from console
–password <password>	Set authentication password
–username <username>	Set authentication username
–verbose	Print more information while working
–connection-param-file <filename>	Optional properties file that provides connection parameters
–relaxed-isolation	Set connection transaction isolation to read uncommitted for the mappers.

SQOOP EXPORT: After **Sqoop Import**, there is a tool which exports a set of files from **HDFS** back to RDBMS, that tool is what we call an Export Tool in **Apache Sqoop**.

Sqoop Export Syntax

$ sqoop export (generic-args) (export-args)
$ sqoop-export (generic-args) (export-args)

- A tool which exports a set of files from HDFS back to an RDBMS is a Sqoop Export tool.
- Moreover, there are files which behave as input to Sqoop which also contain records.
- Those files what we call as rows in the table. Moreover, the job is mapped into map tasks, while we submit our job, that brings the chunk of data from HDFS.
- Then we export these chunks to a structured data destination.
- Likewise, we receive the whole data at the destination by combining all these exported chunks of data.
- However, in most of the cases, it is an RDBMS (MYSQL/Oracle/SQL Server).
- The data which will be exported is processed into records before operation is completed.
- The export of data is done with two steps, first is to examine the database for metadata and second step involves migration of data.
- Sqoop then divides the input dataset into splits and uses individual map tasks to push the splits to RDBMS.



The following command is used to export the table data (which is in **emp_data** file on HDFS) to the employee table in db database of Mysql database server.

```
$ sqoop export \
--connect jdbc:mysql://localhost/db \
--username root \
--table employee \
--export-dir /emp/emp_data
```

The following command is used to verify the table in mysql command line.

```
mysql>select * from employee;
```

If the given data is stored successfully, then you can find the following table of given employee data.

```
+------+-------------+------------+------------------+--------+
| Id   | Name        | Designation | Salary          | Dept   |
+------+-------------+------------+------------------+--------+
| 1201 | gopal       | manager    | 50000            | TP     |
| 1202 | manisha     | preader    | 50000            | TP     |
| 1203 | kalil       | php dev    | 30000            | AC     |
| 1204 | prasanth    | php dev    | 30000            | AC     |
| 1205 | kranthi     | admin      | 20000            | TP     |
| 1206 | satish p    | grp des    | 20000            | GR     |
+------+-------------+------------+------------------+--------+
```

INCREMENTAL IMPORTS:

Incremental import is a technique that imports only the newly added rows in a table. It is required to add 'incremental', 'check-column', and 'last-value' options to perform the incremental import.

Table 5. Sqoop Import – Incremental import arguments

Argument	Description
–check-column (col)	Specifies the column to be examined when determining which rows to import. (the column should not be of type CHAR/NCHAR/VARCHAR/VARNCHAR/ LONGVARCHAR/LONGNVARCHAR)
–incremental (mode)	Specifies how Sqoop determines which rows are new. Legal values for mode include append and lastmodified.
–last-value (value)	Specifies the maximum value of the check column from the previous import.

The following syntax is used for the incremental option in Sqoop import command.

```
--incremental <mode>
--check-column <column name>
--last value <last check column value>
```

- Incremental Imports are classified into two types:
  1. --Incremental Append
  2. --Incremental Lastmodified

General Syntax for Importing data:

```
$ sqoop import
--connect jdbc:mysql://localhost:3306/<database>
--username <username>
                --password <password>
--table <tablename>
--target-dir<targert-dir>
                --Incremental <mode>
                --check-column <id>
                --last-value <value>
```

--Incremental Append:

- It fetches the newly added rows from table and import to HDFS.
- It **doesn't fetch** the **updated rows** from the table.
- so these four properties we will have to consider for inserting new records

```
- -append \
- --check-column <primary key> \
- --incremental append \
- --last-value <Last Value of primary key which sqoop job has
  inserted in last run>
```

Eg: Let us assume the newly added data into **emp** table is as follows −

1206, satish p, grp des, 20000, GR

The following command is used to perform the incremental import in the **emp** table.

```
$ sqoop import \
--connect jdbc:mysql://localhost/userdb \
--username root \
--table emp \
--m 1 \
--incremental append \
--check-column id \
-last value 1205
```

The following command is used to verify the imported data from **emp** table to HDFS emp/ directory.

```
$ $HADOOP_HOME/bin/hadoop fs -cat /emp/part-m-*
```

It shows you the **emp** table data with comma (,) separated fields.

```
1201, gopal,    manager, 50000, TP
1202, manisha,  preader, 50000, TP
1203, kalil,    php dev, 30000, AC
1204, prasanth, php dev, 30000, AC
1205, kranthi,  admin,   20000, TP
1206, satish p, grp des, 20000, GR
```

--Incremental Lastmodified:

- It Fetches the modified or updated rows from table and import to HDFS.
- It doesn't import any new added record in the table.
- Taking the below illustration forward we will update the salary for employee id "1203" from "30000" to "60000", has updated (so lastmodified) recently in Emp table which we want to import in HDFS.
- **Emp id 1203 , Kalil, PhP dev, 60000, AC**
- eg: | Emp id | Name|Desg|Sal|Dept |

```
| 1203 | Kalil|Php dev|60000|AC |
```

Syntax:

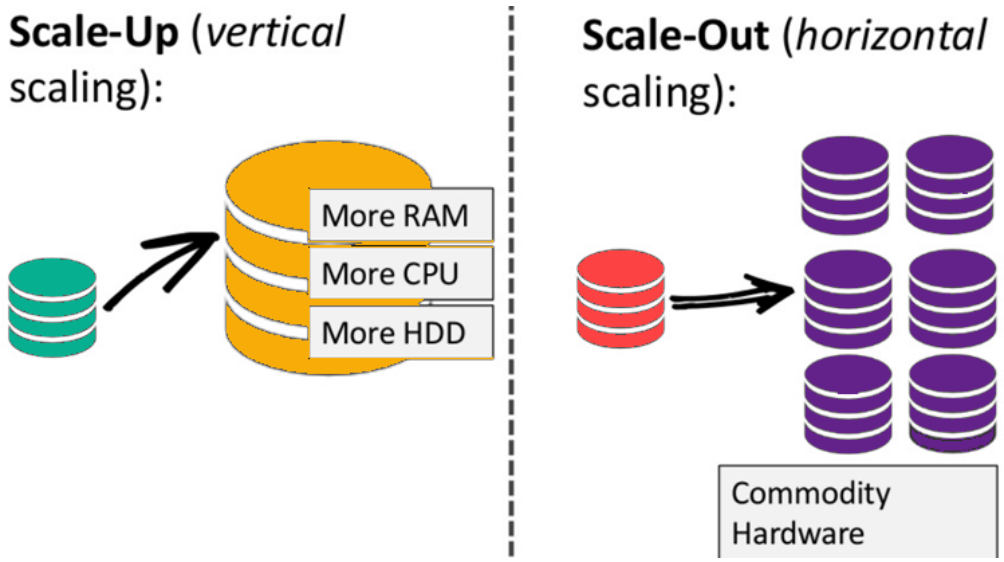
```
 sqoop import \
--connect jdbc:mysql://localhost:3306/db \
--username root -P \
--table customer \
--target-dir /user/hive/warehouse/<your db>/<table> \
  --append
--check-column custid \
--incremental lastmodified \
--last-value <value>
```

Introduction to NoSQL:

- NoSQL is a type of database management system (DBMS) that is designed to handle and store large volumes of unstructured and semi-structured data.
- **NoSQL** stands for Non-Relational Database Management System, or **NoSQL**database stands for "**Not Only SQL".**
- **NoSQL** Database is a non-relational Data Management System, that does not require a fixed schema.
- It avoids joins, and is easy to scale.
- The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs.
- NoSQL is used for Big data and real-time web apps. For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.
- Traditional RDBMS uses SQL syntax to store and retrieve data , Instead, a NoSQL database system encompasses a wide range of database technologies that can store semi- structured, unstructured data.

Why NoSQL?

- The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data. The system response time becomes slow when we use RDBMS for massive volumes of data.
- To resolve this problem, we could "**scale up"** our systems by upgrading our existing hardware(i.e, by increasing size of RAM, HDD, but this process is expensive.
- The alternative for this issue is to distribute database load on multiple hosts whenever theload increases. This method is known as "**scaling out."**



Brief History of NoSQL Databases:

- In the year 1998- **Carlo Strozzi** used the term NoSQL for his lightweight,open-source relational database
- In the year 2000- Graph database Neo4j is launched
- In the year 2004- Google BigTable is launched
- In the year 2005- CouchDB is launched
- In the year 2007- The research paper on Amazon Dynamo is released
- In the year 2008- Facebooks open sources the Cassandra project
- In the year 2009- The term NoSQL was reintroduced

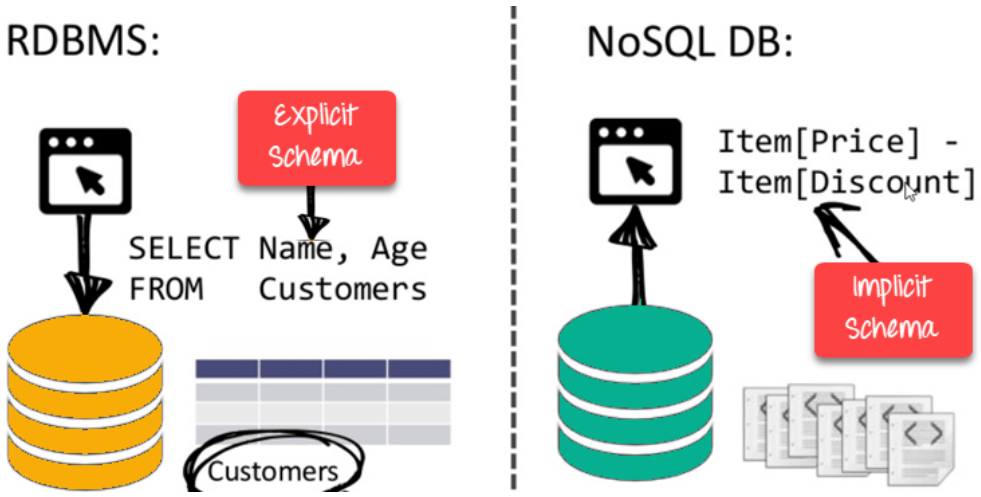
Features of NoSQL:

Non-relational

- NoSQL databases never follow the relational model
- Never provide tables with flat fixed-column records
- Doesn't require object-relational mapping and data normalization

Schema-free

- NoSQL databases are either schema-free or have relaxed schemas
- Do not require any sort of definition of the schema of the data
- Offers heterogeneous structures of data in the same domain



Simple API:

- Offers easy to use interfaces for storage and querying data provided
- APIs allow low-level data manipulation & selection methods
- Text-based protocols mostly used with HTTP REST with JSON.
- Web-enabled databases running as internet-facing services

Distributed:

- Multiple NoSQL databases can be executed in a distributed fashion
- Offers auto-scaling and fail-over capabilities
- Mostly no synchronous replication between distributed nodes Asynchronous Multi-Master Replication, peer-to-peer, HDFS Replication
- Shared Nothing Architecture. This enables less coordination and higher distribution.

Horizontal scalability:

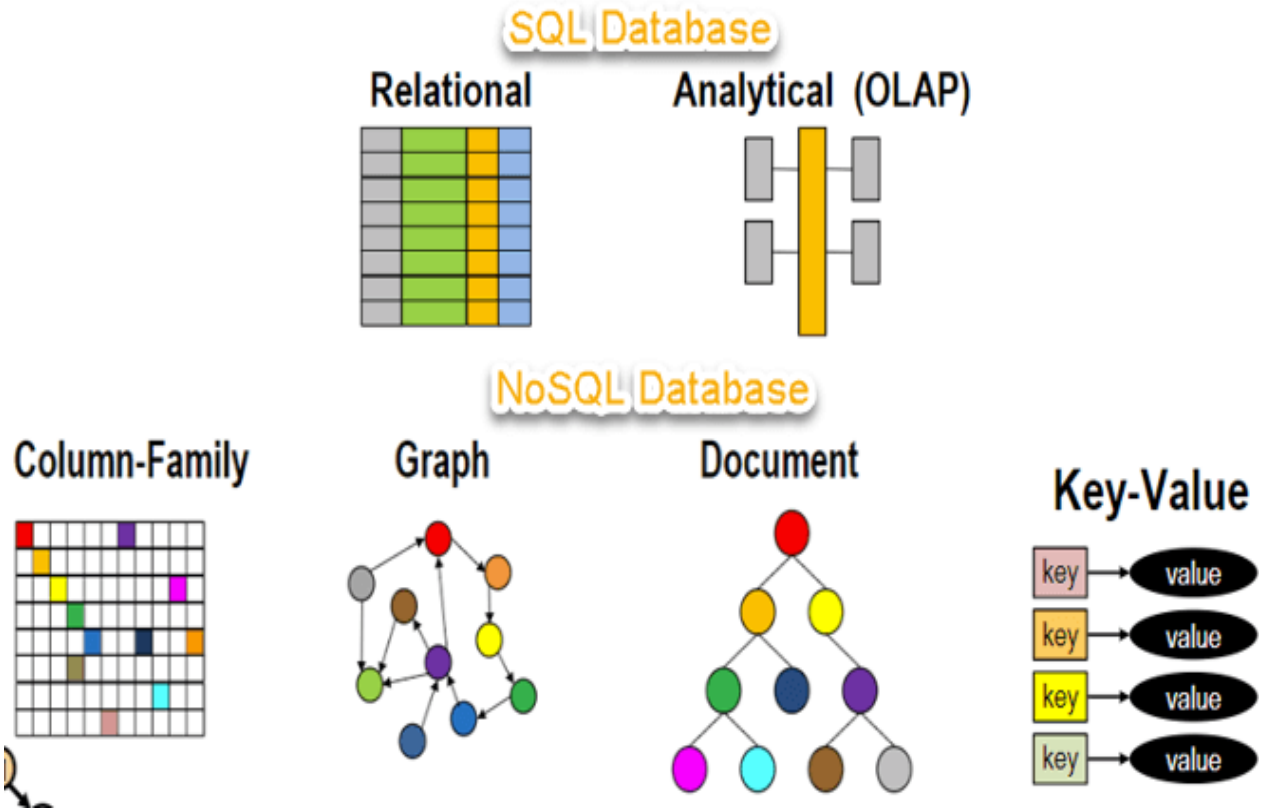
- NoSQL databases are designed to scale out by adding more nodesto a database cluster, making them well-suited for handling large amounts of data and high levels of traffic.

Types of NoSQL Databases:

NoSQL Databases are mainly categorized into four types: Key-value pair, Column-oriented, Graph-based and Document-oriented. Every category has its unique attributes and limitations. None of the above-specified database is better to solve all the problems. Users should select the database based on their product needs.

Types of NoSQL Databases:

- **Key-value Pair Based**
- **Column-oriented Graph**
- **Graphs based**
- **Document-oriented**



Key Value Pair Based:

- These databases store data as key-value pairs, and are optimized for simple and fast read/write operations.
- Key-value pair storage databases store data as a hash table where each key is unique, andthe value can be a JSON(java Script object Notation), string, etc.

For example, a key-value pair may contain a **key like "Website"** associated with a **value like"GeeksforGeeks"**.

Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

- It is one of the most basic NoSQL database example.
- This kind of NoSQL database is used as a collection, dictionaries, associative arrays, etc.
- Key value stores help the developer to store schema-less data. They work best for shopping cart contents.
- Redis, Dynamo, Riak are some NoSQL examples of key-value store DataBases. They areall based on Amazon's Dynamo paper.

Column-based:

- These databases store data as column families, which are sets of columns that are treated as a single entity. They are optimized for fast and efficient querying of large amounts of data .

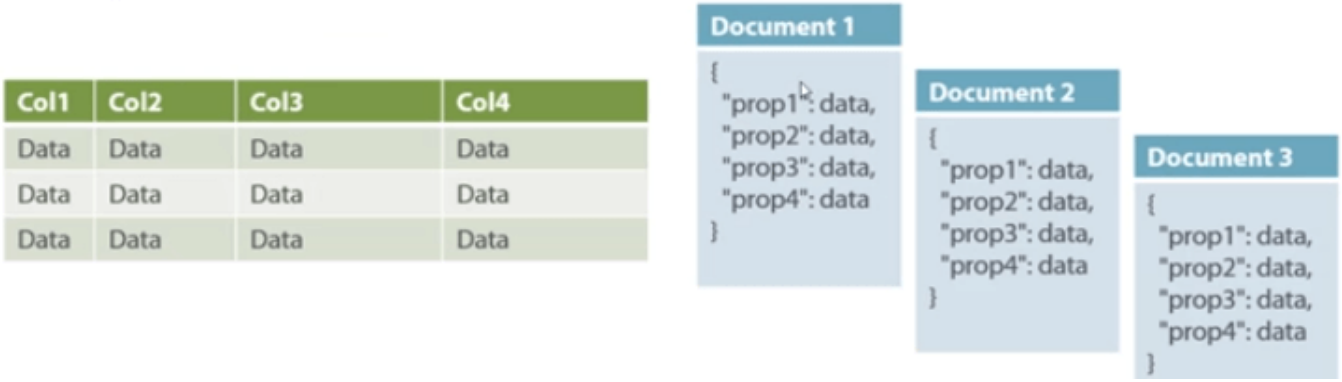
- They deliver high performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column.
- Column-based NoSQL databases are widely used to manage data warehouses, business intelligence, CRM, HBase, Cassandra, are NoSQL query examples of column based database.
-

ColumnFamily			
Row Key	Column Name		
	Key	Key	Key
	Value	Value	Value
	Column Name		
	Key	Key	Key
	Value	Value	Value

Column based NoSQL database

Document-Oriented:

- These databases store data as semi-structured documents, such as JSON or XML, and can be queried using document-oriented query languages.
- Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the valuepart is stored as a document. The document is stored in JSON(Java Script Object Notation) or XML(Extensible Markup Language) formats.



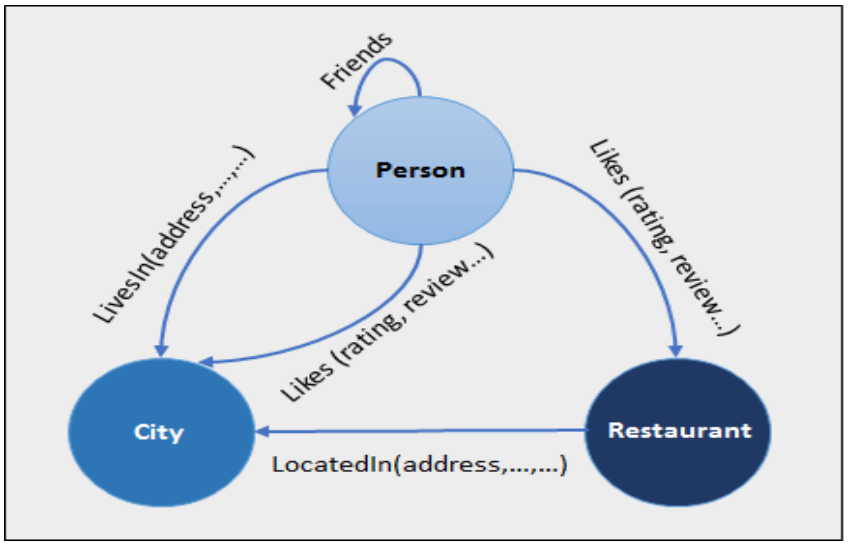
Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

Relational Vs. Document

- In this diagram on the left we have rows and columns, and in the right, we have a document database which has a similar structure to JSON.
- Now for the relational database, you have to know what columns you have and so on. However, for a document database, you have data store like JSON object.
- The document type is mostly used for blogging platforms, real-time analytics & e-commerce applications.
- It should not use for complex transactions which require multiple operations or queries against varying aggregate structures.
- Amazon SimpleDB, CouchDB, MongoDB, Riak, MongoDB, are popular Document originated DBMS systems.

Graph-Based:

- These databases store data as nodes and edges, and are designed to handle complex relationships between data.
- The entity is stored as a node with the relationship as edges.
- An edge gives a relationship between nodes. Every node and edge has a unique identifier.



- Compared to a relational database where tables are loosely connected, a Graph database is a multi-relational in nature.
- Traversing relationship is fast as they are already captured into the DB, and there is no need to calculate them.
- Graph base database mostly used for social networks, logistics, spatial

data.

- Neo4J, Infinite Graph, OrientDB, are some popular graph-based databases.

Advantages of NoSQL:

1. High scalability :
    - NoSQL databases use sharding(process of Storing large database across multiple machines) for horizontal scaling.
    - Partitioning of data and placing it on multiple machines in such a way that the order of the data is preserved is sharding.
    - Vertical scaling means adding more resources to the existing machine whereas horizontal scaling means adding more machines to handle the data.
    - Vertical scaling is not that easy to implement but horizontal scaling is easy to implement.
    - Examples of horizontal scaling databases are MongoDB, Cassandra, etc. NoSQL can handle a huge amount of data because of scalability, as the data grows NoSQL scale itself to handle that data in an efficient manner.
2. Flexibility:
    - NoSQL databases are designed to handle unstructured or semi-structured data, which means that they can accommodate dynamic changes to the data model.
    - This makes NoSQL databases a good fit for applications that need to handle changing data requirements.
3. High availability :
    - Auto replication feature in NoSQL databases makes it highly available because in case of any failure data replicates itself to the previous consistent state.
4. Performance:
    - NoSQL databases are designed to handle large amounts of data and traffic, which means that they can offer improved performance compared to traditional relational databases.
5. Cost-effectiveness:
    - NoSQL databases are often more cost-effective than traditional relational databases, as they are typically less complex and do not require expensive hardware or software.

Disadvantages of NoSQL:

NoSQL has the following disadvantages.

1. **Lack of standardization** : There are many different types of NoSQL databases, each with its own unique strengths and weaknesses. This lack of standardization can make it difficult to choose the right database for a specific application
2. **Lack of ACID compliance :** NoSQL databases are not fully ACID-compliant, which means that they do not guarantee the consistency, integrity, and durability of data. This can be a drawback for applications that require strong data consistency guarantees.
3. **Open-source :** NoSQL is open-source database. There is no reliable standard for NoSQL yet. In other words, two database systems are likely to be unequal.
4. **Lack of support for complex queries :** NoSQL databases are not designed to handle complex queries, which means that they are not a good fit for applications that require complex data analysis or reporting.
5. Management challenge :
    - The purpose of big data tools is to make the management of a large amount of data as simple as possible.
    - But it is not so easy. Data management in NoSQL is much more complex than in a relational database.
6. **Backup :** Backup is a great weak point for some NoSQL databases like MongoDB. MongoDB has no approach for the backup of data in a consistent manner.

Limitations of Hadoop

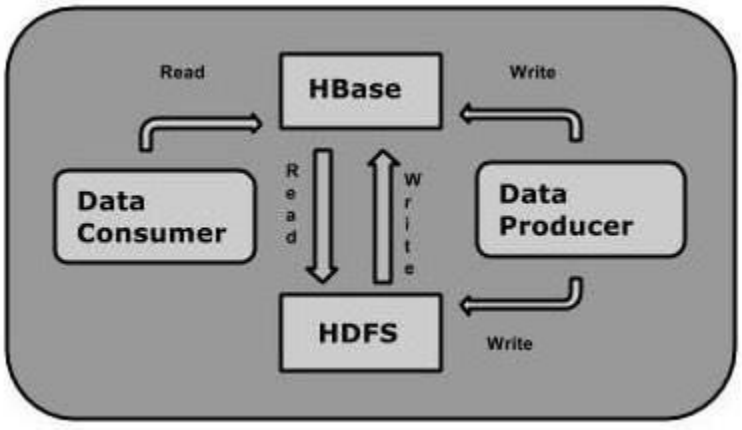
Hadoop can perform only batch processing, and data will be accessed only in a sequential manner. That means one has to search the entire dataset even for the simplest of jobs.

A huge dataset when processed results in another huge data set, which should also be processed sequentially. At this point, a new solution is needed to access any point of data in a single unit of **time (random access).**

Here comes **Hbase as solution.**

HBase:

- HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable.
- HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS).
- It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.
- One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.



Storage Mechanism in HBase

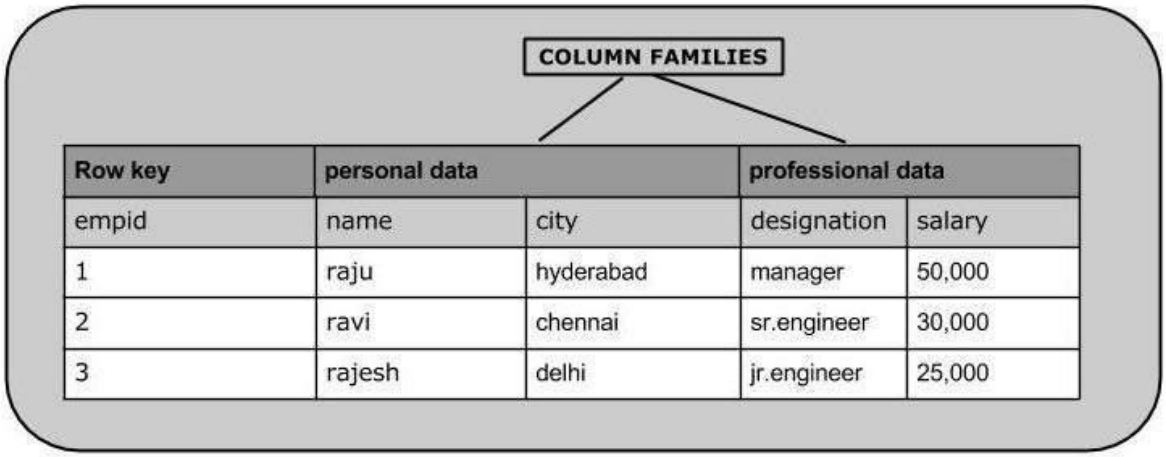
- HBase is a **column-oriented database** and the tables in it are sorted by row.
- The table schema defines only column families, which are the key value pairs.
- A table have multiple column families and each column family can have any number of columns.
- Subsequent column values are stored contiguously on the disk. Each cell value of the table has a timestamp. In short, in an HBase:

- Table is a collection of rows.
- Row is a collection of column families.
- Column family is a collection of columns.
- Column is a collection of key value pairs.

Given below is an example schema of table in HBase.

Rowid	Column Family			Column Family			Column Family			Column Family		
	col1	col2	col3	col1	col2	col3	col1	col2	col3	col1	col2	co
1												
2												
3												

The following image shows column families in a column-oriented database:



Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

FEATURES OF HBase:

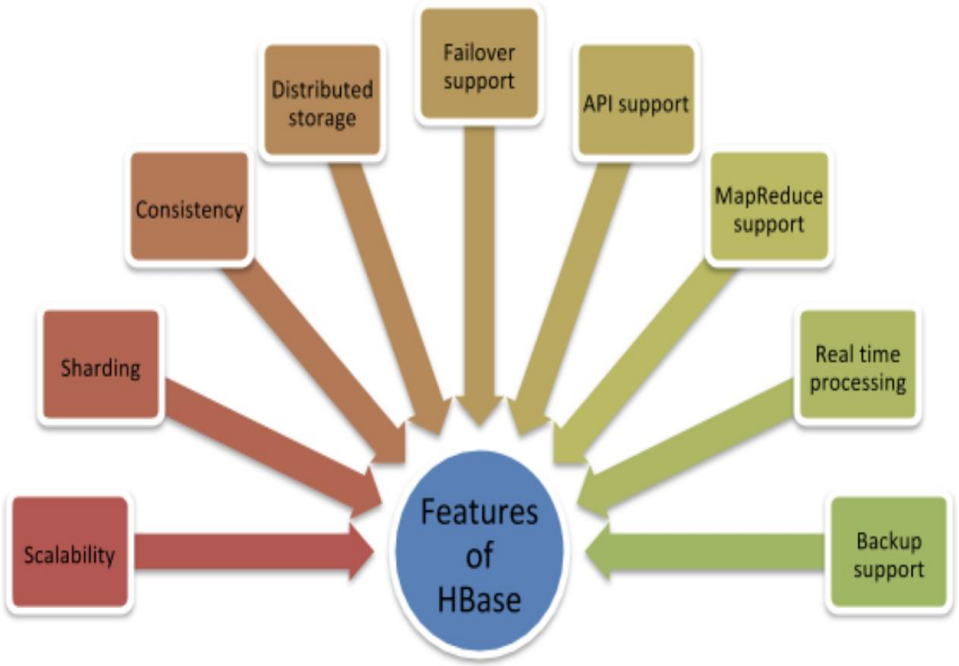


Fig – Showing important features offered by HBase

- **Scalability** – HBase supports scalability in both modular and linear format
- **Sharding** – Sharding of tables is supported by HBase. It is also configurable.
- **Consistency** – HBase also supports consistent read and write operations
- **Distributed storage** – Distributed storage like HDFS is supported by HBase
- **Failover support** – HBase supports automatic failover
- **API support** – Java APIs are supported by HBase
- **Backup support** – Backup support for Hadoop MapReduce jobs in Hbase tables is available in HBase.
- **MapReduce support** – MapReduce support is available for parallel processing of bulk data
- **Real-time processing** – HBase supports block cache and Bloom filters to make real-time processing easy.

Applications of HBase

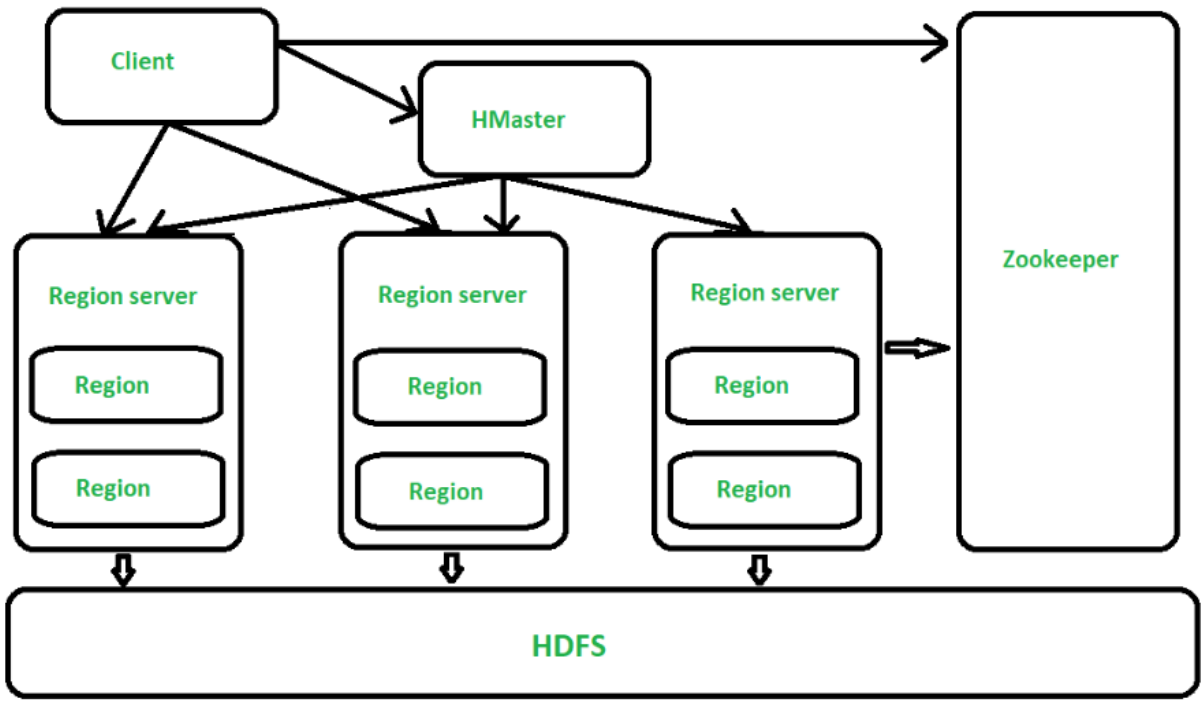
- It is used whenever there is a need to write heavy applications.
- HBase is used whenever we need to provide fast random access to available data.
- Companies such as Facebook, Twitter, Yahoo, and Adobe use HBase internally.

HBase History

Year	Event
Nov 2006	Google released the paper on BigTable.
Feb 2007	Initial HBase prototype was created as a Hadoop contribution.

Oct 2007	The first usable HBase along with Hadoop 0.15.0 was released.
Jan 2008	HBase became the sub project of Hadoop.
Oct 2008	HBase 0.18.1 was released.
Jan 2009	HBase 0.19.0 was released.
Sept 2009	HBase 0.20.0 was released.
May 2010	HBase became Apache top-level project.

HBase Architecture:



HBase architecture has different components:

- HMaster
- Region Server
- Zookeeper
- HDFS

HMaster –

- The implementation of Master Server in HBase is HMaster.
- It is a process in which regions are assigned to region server as well as DDL (create, delete table) operations.
- It monitor all Region Server instances present in the cluster.
- In a distributed environment, Master runs several background threads.
- HMaster has many features like controlling load balancing, failover etc.
- Is responsible for schema changes and other metadata operations such as creation of tables and column families.

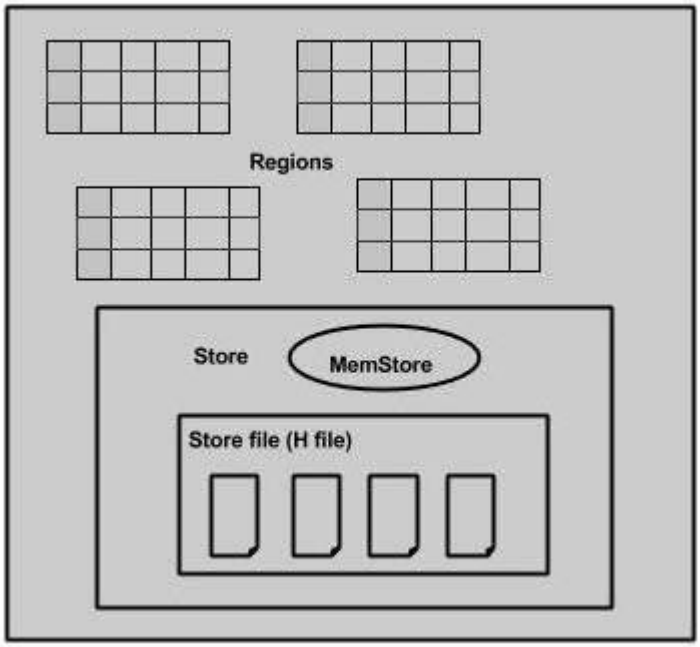
Regions:

- Regions are nothing but tables that are split up and spread across the region servers. **Regions** are the basic building elements of HBase cluster that consists of the distribution of tables and are comprised of Column families.

Region Server –

- HBase Tables are divided horizontally by row key range into Regions.
- Region servers Communicate with the client and handle data-related operations.
- Handle read and write requests for all the regions under it.
- Region Server runs on HDFS DataNode which is present in Hadoop cluster.

- Regions of Region Server are responsible for several things, like handling, managing, executing as well as reads and writes HBase operations on that set of regions. The default size of a region is 256 MB.
- When we take a deeper look into the region server, it contain regions and stores as shown below:



- The store contains **memory store and HFiles**.
- Memstore is just like a cache memory. Anything that is entered into the HBase is stored here initially. Later, the data is transferred and saved in Hfiles as blocks and the memstore is flushed.

Zookeeper –

- It is like a coordinator in HBase.
- It provides services like maintaining configuration information, naming, providing distributed synchronization, server failure notification etc.
- Clients communicate with region servers via zookeeper.

HDFS

Data is stored in HDFS

Advantages of HBase –

1. Can store large data sets
2. Database can be shared
3. Cost-effective from gigabytes to petabytes
4. High availability through failover and replication

Disadvantages of HBase –

1. No support SQL structure
2. No transaction support
3. Sorted only on key
4. Memory issues on the cluster

Features of HBase architecture :

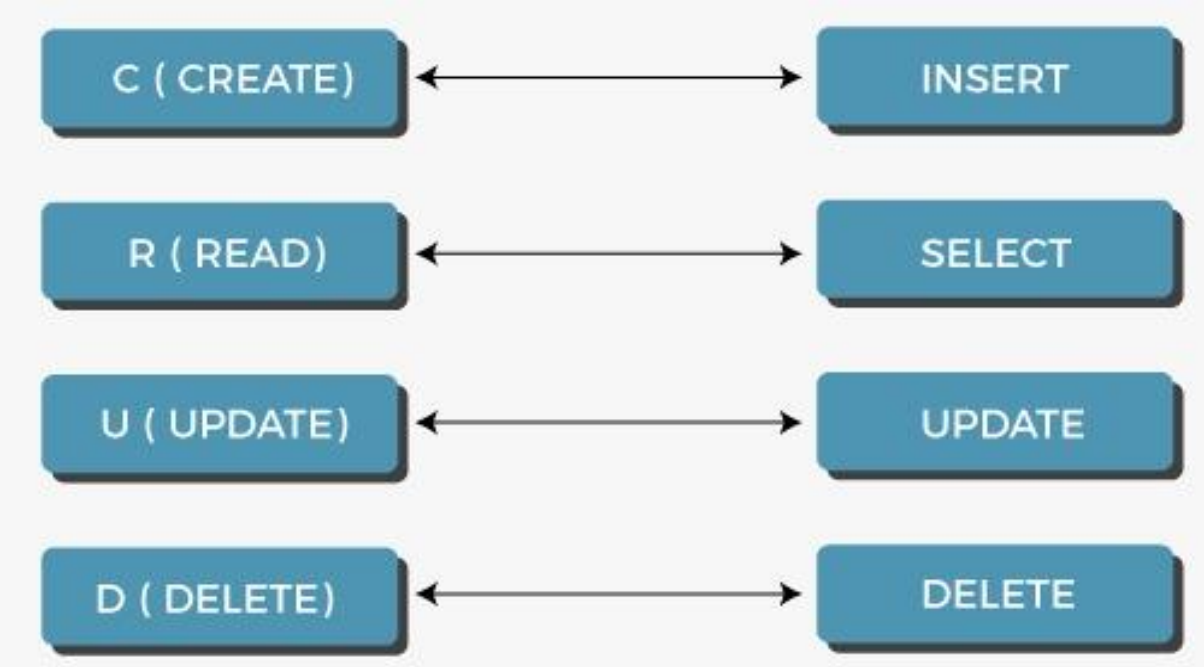
- **Distributed and Scalable:** HBase is designed to be distributed and scalable, which means it can handle large datasets and can scale out horizontally by adding more nodes to the cluster.
- **Column-oriented Storage:** HBase stores data in a column-oriented manner, which means data is organized by columns rather than rows. This allows for efficient data retrieval and aggregation.
- **Hadoop Integration:** HBase is built on top of Hadoop, which means it can leverage Hadoop's distributed file system (HDFS) for storage and MapReduce for data processing.
- **Consistency and Replication:** HBase provides strong consistency guarantees for read and write operations, and supports replication of data across multiple nodes for fault tolerance.

- **Built-in Caching:** HBase has a built-in caching mechanism that can cache frequently accessed data in memory, which can improve query performance.
- **Compression:** HBase supports compression of data, which can reduce storage requirements and improve query performance.
- **Flexible Schema:** HBase supports flexible schemas, which means the schema can be updated on the fly without requiring a database schema migration.

Note – HBase is extensively used for online analytical operations, like in banking applications such as real-time data updates in ATM machines, HBase can be used.

CRUD Operations:

- CRUD operations act as the foundation of any computer programming language or technology. So before taking a deeper dive into any programming language or technology, one must be proficient in working on its CRUD operations. This same rule applies to databases as well.



Create:

- In CRUD operations, 'C' is an acronym for create, which means to add or insert data into the SQL table. So, firstly we will create a table using CREATE command and then we will use the INSERT INTO command to insert rows in the created table.
- **Syntax for table creation:**

CREATE TABLE Table_Name (ColumnName1 Datatype, ColumnName2 Datatype,..., ColumnNameN Datatype)

Where

- Table_Name is the name that we want to assign to the table.
- Column_Name is the attributes under which we want to store data of the table.
- Datatype is assigned to each column. Datatype decides the type of data that will be stored in the respective column.

Read:

- In CRUD operations, 'R' is an acronym for read, which means *retrieving or fetching the data from the SQL table.*
- So, we will use the SELECT command to fetch the inserted records from the SQL table.
- We can retrieve all the records from a table using an asterisk (*) in a SELECT query.
- There is also an option of retrieving only those records which satisfy a particular condition by using the WHERE clause in a SELECT query.

Syntax to fetch all the records:

```
SELECT *FROM TableName;
```

Syntax to fetch records according to the condition:

```
SELECT *FROM TableName WHERE CONDITION;
```

Update:

- In CRUD operations, 'U' is an acronym for the update, which *means making updates to the records present in the SQL tables.*
- So, we will use the UPDATE command to make changes in the data present in tables.

Syntax:

UPDATE Table_Name SET ColumnName = Value WHERE CONDITION;

Delete:

- In CRUD operations, 'D' is an acronym for delete, which means removing or deleting the records from the SQL tables.
- We can delete all the rows from the SQL tables using the DELETE query.
- There is also an option to remove only the specific records that satisfy a particular condition by using the WHERE clause in a DELETE query.

Syntax to delete all the records:

DELETE FROM TableName;

Syntax to delete records according to the condition:

DELETE FROM TableName WHERE CONDITION;