NoSQL Databases and Apache HBase

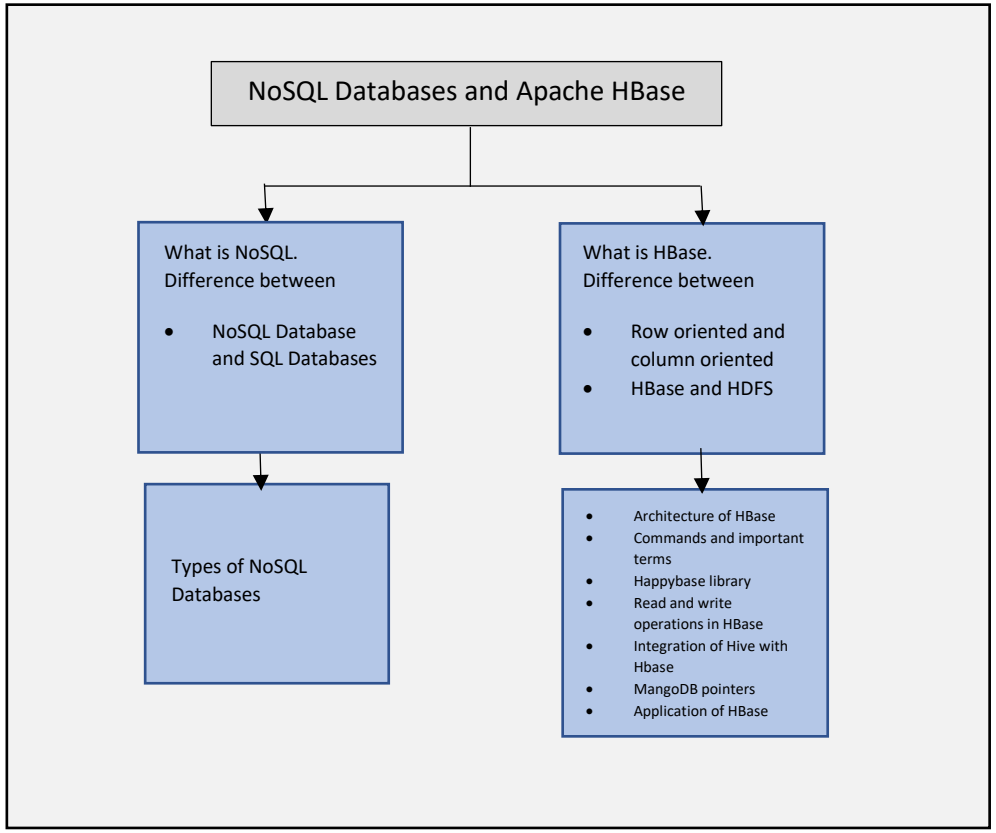
NoSQL Database is a non-relational Data Management System, that does not require a fixed schema. HBase is an essential part of our Hadoop ecosystem. It can store massive amounts of data from terabytes to petabytes.

As a part of NoSQL Databases and Apache HBase, you covered:

- Introduction to NoSQL
- Types of NoSQL
- Introduction to HBase
- HBase Architecture and commands

Common Interview Questions:

1. Mention the difference between HBase and Relational Database?
2. Explain why to use HBase?
3. HBase support syntax structure like SQL yes or No?
4. Mention what are the key components of HBase?
5. Explain what is the row key?
6. What are the pros and cons of a graph database under NoSQL databases?
7. List the different kinds of NoSQL data stores?
8. List some of the features of NoSQL?
9. Explain NoSQL Databases

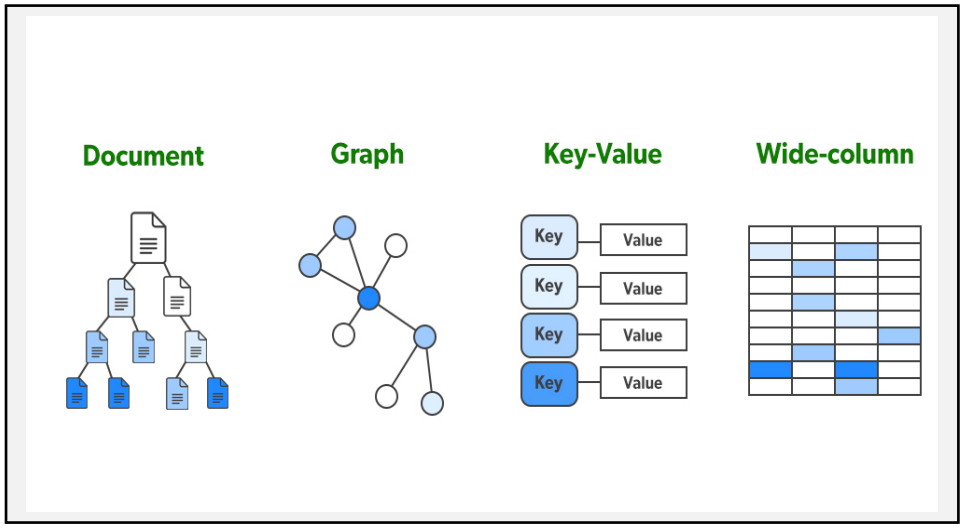


NoSQL Databases and Apache HBase

NoSQL Databases:

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.

Types of NoSQL Databases:



SQL and NoSQL Databases:

Based on	SQL	NoSQL
Data Storage Model	Tables with fixed rows and columns	Document: JSON documents, Key-value: key-value pairs, Wide-column: tables with rows and dynamic columns, Graph: nodes and edges
Schemas	Rigid	Flexible
Scaling	Vertical (scale-up with a larger server)	Vertical (scale-up with a larger server)
Vertical	Supported	Most do not support multi-record ACID transactions
Joins	Typically required	Typically, not required
Examples	Oracle, MySQL, Microsoft SQL Server, and PostgreSQL	Document: MongoDB and CouchDB, Key-value: Redis and DynamoDB, Wide-column: Cassandra and HBase, Graph: Neo4j and Amazon Neptune

NoSQL Databases and Apache HBase

HBase:

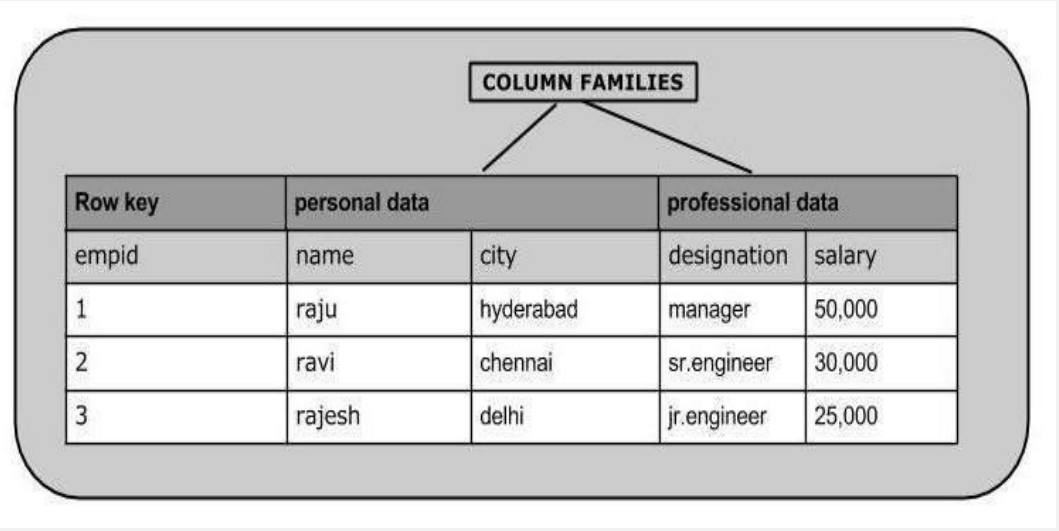
HBase is an open source, distributed database developed by Apache software foundation written in Java. HBase is an essential part of our Hadoop ecosystem. HBase runs on top of HDFS (Hadoop Distributed File System). It can store massive amounts of data from terabytes to petabytes. It is column oriented and horizontally scalable.

HDFS and HBase:

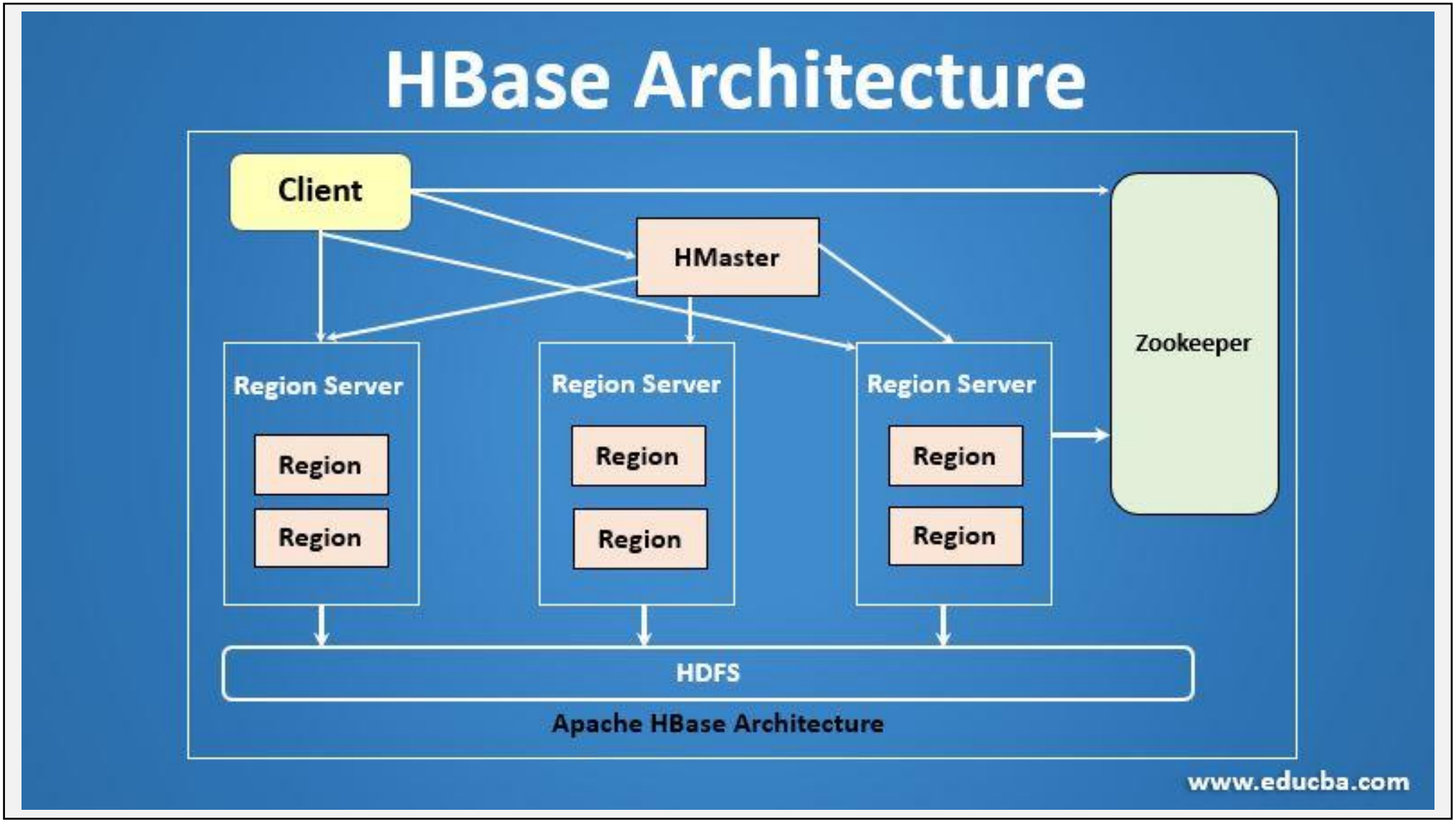
HDFS	HBase
It provides high latency batch processing; no concept of batch processing.	It provides low latency access to single rows from billions of records (Random access).
It provides only sequential access of data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

Row-oriented and Column-Oriented Databases:

Row-Oriented Database	Column-Oriented Database
It is suitable for Online Transaction Process (OLTP).	It is suitable for Online Analytical Processing (OLAP).
Such databases are designed for small number of rows and columns.	Column-oriented databases are designed for huge tables.



HBase Architecture:



NoSQL Databases and Apache HBase

Important Terminologies:

MasterServer: Assigns regions to the region servers and takes the help of Apache ZooKeeper for this task. Handles load balancing of the regions across region servers. It unloads the busy servers and shifts the regions to less occupied servers. Is responsible for schema changes and other metadata operations such as creation of tables and column families.

Region: Regions are nothing but tables that are split up and spread across the region servers.

Zookeeper: Zookeeper is an open-source project that provides services like maintaining configuration information, naming, providing distributed synchronization, etc. Zookeeper has ephemeral nodes representing different region servers. Master servers use these nodes to discover available servers.

General Commands in HBase:

Command	Summary
Status **hbase(main):009:0> status**	This command returns the status of the system including the details of the servers running on the system
Version **hbase(main):010:0> version**	This command returns the version of HBase used in your system.
Table_help **hbase(main):02:0> table_help**	This command guides you what and how to use table-referenced commands
Whoami **hbase(main):008:0> whoami**	This command returns the user details of HBase.

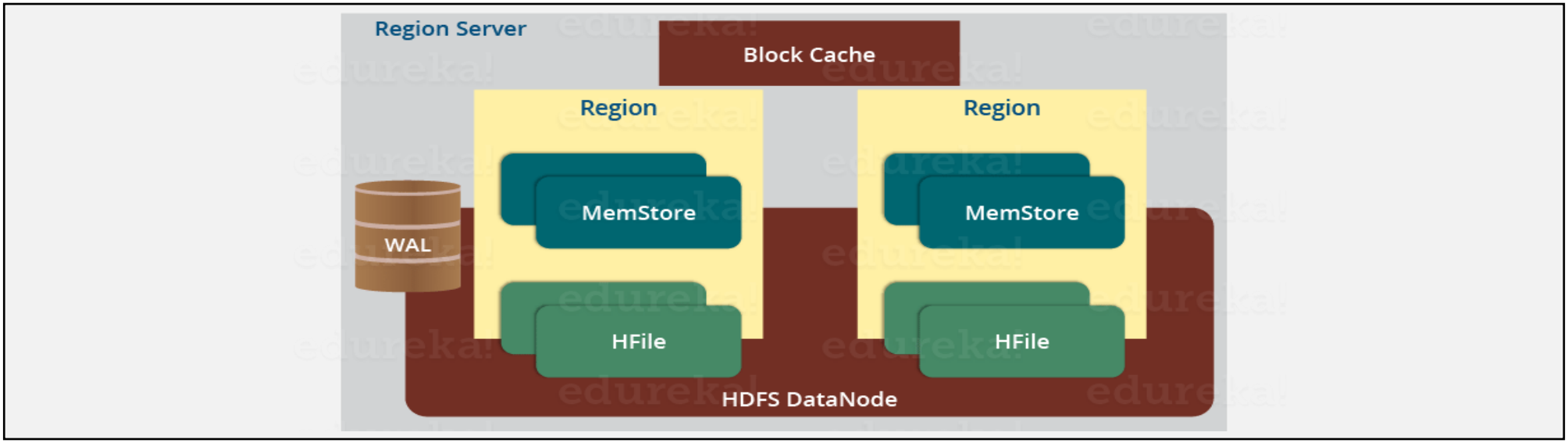
Region Server Components:

WAL: Write Ahead Log (WAL) is a file attached to every Region Server inside the distributed environment. The WAL stores the new data that hasn't been persisted or committed to the permanent storage. It is used in case of failure to recover the data sets.

Block Cache: The Block Cache resides in the top of Region Server. It stores the frequently read data in the memory. If the data in BlockCache is least recently used, then that data is removed from BlockCache.

MemStore: It is the write cache. It stores all the incoming data before committing it to the disk or permanent memory. There is one MemStore for each column family in a region. As you can see in the image, there are multiple MemStores for a region because each region contains multiple column families. The data is sorted in lexicographical order before committing it to the disk.

HFile: HFile is stored on HDFS. Thus it stores the actual cells on the disk. MemStore commits the data to HFile when the size of MemStore exceeds.



Read and Write Operations in HBase:

Steps for Read Operation:

Step 1: The client gets the information about region servers from META Table.

Step 2: The Client gets the information about row keys and the META table location.

Step 3: After getting information from META table Client communicates to Region Servers

Step 4: The client will retrieve the rows from the corresponding region servers.

Steps for Write operation:

Step 1: Each data is first written into Write Ahead Log (Wal). WAL is used as a backup or recovery process.

Step 2: After the data is written to WAL, Data is written to Memstore.

Step 3: Memstore is the intermediate store before the final commit to Hfile. The data is sorted before finally persisted in the disk. There is one Memstore per column family.

Step 4: When there is enough data in Memstore, the intermediate data is written into Hfile.

Step 5: Data is stored as key-value pair in Hfile.

HappyBase:

HappyBase is a developer-friendly Python library to interact with Apache HBase. HappyBase is designed for use in standard HBase setups, and offers application developers a Pythonic API to interact with HBase. HappyBase uses the Python Thrift library to connect to HBase using its Thrift gateway.

Application of HappyBase:

Pre requisites:

1) Linux virtual environment setup
2) HBase installed
3) A table created in HBase

Step 1: Install Python 3 and HappyBase package:

```
apk add python3

pip install happybase
```

Step 2: Start Trift server:

```
alpine-hbase:/home/downloads/hbase-2.2.3/bin# ./hbase-daemon.sh start thrift
running thrift, logging to /home/downloads/hbase-2.2.3/bin/../logs/hbase-root-th
rift-alpine-hbase.out
```

Step 3: Start HBase server:

```
alpine-hbase:/home/downloads/hbase-2.2.3/bin# ./start-hbase.sh
running master, logging to /home/downloads/hbase-2.2.3/bin/../logs/hbase-root-ma
ster-alpine-hbase.out
alpine-hbase:/home/downloads/hbase-2.2.3/bin# _
```

Step 4: Now we open an Interactive Python session and make a connection to HBase to see that everything is working fine using Happybase APIs:

```
alpine-hbase:/home/downloads/hbase-2.2.3/bin# python3
Python 3.8.1 (default, Dec 30 2019, 15:43:37)
[GCC 9.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import happybase as hb
>>> conn = hb.Connection('127.0.0.1',9090)
>>> conn.tables()
[b'employers']
>>> conn.table('employers').row('row1')
{b'empAnagrafical:name': b'andrea', b'empAnagrafical:surname': b'guidi'}
>>> conn.table('employers').row('row2')
{b'empAnagrafical:name': b'george', b'empAnagrafical:surname': b'clooney'}
```

NoSQL Databases and Apache HBase

Integration of HBase with Hive:

Example: Migrate the data from Hive to HBase table.

For setting up of HBase Integration with Hive, we mainly require a few jar files. The required jar files are:

```
zookeeper-*.jar           //This will be present in $HIVE_HOME/lib directory
hive-hbase-handler-*.jar  //This will be present in $HIVE_HOME/lib directory
guava-*.jar               //This will be present in $HIVE_HOME/lib director
hbase-*.jar files         //This will be present in $HBASE_HOME/lib directory
```

Step 1: Create Hive table

```
hive> CREATE TABLE hive_table(
    >   empno int,
    >   ename string,
    >   designation string,
    >   manager int,
    >   hire_date string,
    >   sal int,
    >   deptno int)
    > ROW FORMAT DELIMITED
    >   FIELDS TERMINATED BY ','
    >   LINES TERMINATED BY '\n'
    > STORED AS TEXTFILE;
OK
Time taken: 3.03 seconds
```

Step 2: Load data into Hive

```
hive> LOAD DATA LOCAL INPATH '/root/bdp/hbase/data/emp_data.csv' INTO TABLE hive_table;
Loading data to table db_bdp.hive_table
Table db_bdp.hive_table stats: [numFiles=1, totalSize=598]
OK
Time taken: 4.922 seconds
hive> select * from hive_table;
OK
7369    SMITH   CLERK   7902    12/17/1980      800     20
7499    ALLEN   SALESMAN        7698    2/20/1981       1600    30
7521    WARD    SALESMAN        7698    2/22/1981       1250    30
7566    TURNER  MANAGER 7839    4/2/1981        2975    20
7654    MARTIN  SALESMAN        7698    9/28/1981       1250    30
7698    MILLER  MANAGER 7839    5/1/1981        2850    30
7782    CLARK   MANAGER 7839    6/9/1981        2450    10
7788    SCOTT   ANALYST 7566    12/9/1982       3000    20
7839    KING    PRESIDENT       NULL    11/17/1981      5000    10
7844    TURNER  SALESMAN        7698    9/8/1981        1500    30
7876    ADAMS   CLERK   7788    1/12/1983       1100    20
7900    JAMES   CLERK   7698    12/3/1981       950     30
7902    FORD    ANALYST 7566    12/3/1981       3000    20
7934    MILLER  CLERK   7782    1/23/1982       1300    10
Time taken: 5.041 seconds, Fetched: 14 row(s)
hive>
```

Step 3: Create HBase-Hive Mapping table

```
hive> CREATE TABLE hbase_table_employee
    >   (
    >       empno       INT,
    >       ename       STRING,
    >       designation STRING,
    >       manager     INT,
    >       hire_date   STRING,
    >       sal         INT,
    >       deptno      INT
    >   )
    > STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
    > WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf:ename,
no")
    > TBLPROPERTIES ("hbase.table.name" = "employee_hbase");
OK
Time taken: 17.237 seconds
```

Step 4: Load data into HBase from Hive

```
hive> INSERT INTO TABLE hbase_table_employee SELECT * FROM hive_table;
Query ID = root_20171228155155_22b31bcb-d741-4126-8d05-c64602e26312
Total jobs = 1
Launching Job 1 out of 1
Tez session was closed. Reopening...
Session re-established.


Status: Running (Executing on YARN cluster with App id application_1514453878963_0009)

--------------------------------------------------------------------------------
        VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
--------------------------------------------------------------------------------
Map 1 ..........   SUCCEEDED      1          1        0        0       0       0
--------------------------------------------------------------------------------
VERTICES: 01/01  [==========================>>] 100%  ELAPSED TIME: 15.08 s
--------------------------------------------------------------------------------
OK
Time taken: 44.236 seconds
```

Step 5: Scan HBase Table

```
hbase(main):008:0> scan 'employee_hbase'
ROW                             COLUMN+CELL
 7369                           column=cf:deptno, timestamp=1514476352028, value=20
 7369                           column=cf:designation, timestamp=1514476352028, value=CLERK
 7369                           column=cf:ename, timestamp=1514476352028, value=SMITH
 7369                           column=cf:hire_date, timestamp=1514476352028, value=12/17/1980
 7369                           column=cf:manager, timestamp=1514476352028, value=7902
 7369                           column=cf:sal, timestamp=1514476352028, value=800
 7499                           column=cf:deptno, timestamp=1514476352028, value=30
 7499                           column=cf:designation, timestamp=1514476352028, value=SALESMAN
 7499                           column=cf:ename, timestamp=1514476352028, value=ALLEN
 7499                           column=cf:hire_date, timestamp=1514476352028, value=2/20/1981
 7499                           column=cf:manager, timestamp=1514476352028, value=7698
 7499                           column=cf:sal, timestamp=1514476352028, value=1600
 7521                           column=cf:deptno, timestamp=1514476352028, value=30
 7521                           column=cf:designation, timestamp=1514476352028, value=SALESMAN
 7521                           column=cf:ename, timestamp=1514476352028, value=WARD
 7521                           column=cf:hire_date, timestamp=1514476352028, value=2/22/1981
 7521                           column=cf:manager, timestamp=1514476352028, value=7698
 7521                           column=cf:sal, timestamp=1514476352028, value=1250
 7566                           column=cf:deptno, timestamp=1514476352028, value=20
 7566                           column=cf:designation, timestamp=1514476352028, value=MANAGER
 7566                           column=cf:ename, timestamp=1514476352028, value=TURNER
 7566                           column=cf:hire_date, timestamp=1514476352028, value=4/2/1981
 7566                           column=cf:manager, timestamp=1514476352028, value=7839
 7566                           column=cf:sal, timestamp=1514476352028, value=2975
 7654                           column=cf:deptno, timestamp=1514476352028, value=30
 7654                           column=cf:designation, timestamp=1514476352028, value=SALESMAN
```

NoSQL Databases and Apache HBase

MangoDB:

MongoDB is an open-source document database and leading NoSQL database. MongoDB is written in C++. A record in MongoDB is a document, which is a data structure composed of key value pairs similar to the structure of JSON objects.

Features of MangoDB:

1) Support ad hoc queries
2) Can index any field in a document
3) Duplication of data
4) Load balancing
5) Supports map reduce and aggregation tools.
6) Uses JavaScript instead of Procedures.
7) It is a schema-less database written in C++.
8) Provides high performance.
9) Stores files of any size easily without complicating your stack.

Application of HBase: