**Session 9 : HBase**

**Assignment 1**

**1.What is NoSQL data base?**

NoSQL is an approach to databases that represents a shift away from traditional relational database management systems (RDBMS). To define NoSQL, it is helpful to start by describing SQL, which is a query language used by RDBMS. Relational databases rely on tables, columns, rows, or schemas to organize and retrieve data. In contrast, NoSQL databases do not rely on these structures and use more flexible data models. NoSQL can mean “not SQL” or “not only SQL.” As RDBMS have increasingly failed to meet the performance, scalability, and flexibility needs that next-generation, data-intensive applications require, NoSQL databases have been adopted by mainstream enterprises. NoSQL is particularly useful for storing unstructured data, which is growing far more rapidly than structured data and does not fit the relational schemas of RDBMS. Common types of unstructured data include: user and session data; chat, messaging, and log data; time series data such as IoT and device data; and large objects such as video and images.

BENEFITS OF NOSQL

NoSQL databases offer enterprises important advantages over traditional RDBMS, including:

• Scalability: NoSQL databases use a horizontal scale-out methodology that makes it easy to add or reduce capacity quickly and non-disruptively with commodity hardware. This eliminates the tremendous cost and complexity of manual sharding that is necessary when attempting to scale RDBMS.

• Performance: By simply adding commodity resources, enterprises can increase performance with NoSQL databases. This enables organizations to continue to deliver reliably fast user experiences with a predictable return on investment for adding resources—again, without the overhead associated with manual sharding.

• High Availability: NoSQL databases are generally designed to ensure high availability and avoid the complexity that comes with a typical RDBMS architecture that relies on primary and secondary nodes. Some “distributed” NoSQL databases use a masterless architecture that automatically distributes data equally among multiple resources so that the application remains available for both read and write operations even when one node fails.

• Global Availability: By automatically replicating data across multiple servers, data centers, or cloud resources, distributed NoSQL databases can minimize latency and ensure a consistent application experience wherever users are located. An added benefit is a significantly reduced database management burden from manual RDBMS configuration, freeing operations teams to focus on other business priorities.

• Flexible Data Modeling: NoSQL offers the ability to implement flexible and fluid data models. Application developers can leverage the data types and query options that are the most natural fit to the specific application use case rather than those that fit the database schema. The result is a simpler interaction between the application and the database and faster, more agile development.

**2. How does data get stored in NoSQL database?**

TYPES OF NOSQL DATABASES

Several different varieties of NoSQL databases have been created to support specific needs and use cases. These fall into four main categories:

• Key-value data stores: Key-value NoSQL databases emphasize simplicity and are very useful in accelerating an application to support high-speed read and write processing of non-transactional data. Stored values can be any type of binary object (text, video, JSON document, etc.) and are accessed via a key. The application has complete control over what is stored in the value, making this the most flexible NoSQL model. Data is partitioned and replicated across a cluster to get scalability and availability. For this reason, key value stores often do not support transactions. However, they are highly effective at scaling applications that deal with high-velocity, non-transactional data. E.g. Cassandra

• Document stores: Document databases typically store self-describing JSON, XML, and BSON documents. They are similar to key-value stores, but in this case, a value is a single document that stores all data related to a specific key. Popular fields in the document can be indexed to provide fast retrieval without knowing the key. Each document can have the same or a different structure. E.g. MongoDB

• Wide-column stores: Wide-column NoSQL databases store data in tables with rows and columns similar to RDBMS, but names and formats of columns can vary from row to row across the table. Wide-column databases group columns of related data together. A query can retrieve related data in a single operation because only the columns associated with the query are retrieved. In an RDBMS, the data would be in different rows stored in different places on disk, requiring multiple disk operations for retrieval. E.g.

• Graph stores: A graph database uses graph structures to store, map, and query relationships. They provide index-free adjacency, so that adjacent elements are linked together without using an index.

Multi-modal databases leverage some combination of the four types described above and therefore can support a wider range of applications.

**3. What is a column family in HBase?**

In the HBase data model columns are grouped into *column families*, which must be defined up front during table creation. Column families are stored together on disk, which is why HBase is referred to as a column-oriented data store.

|  |  |
| --- | --- |
| Row Key | Column Family: {Column Qualifier:Version:Value} |
| 00001 | CustomerName: {‘FN’: 1383859182496:‘John’, ‘LN’: 1383859182858:‘Smith’, ‘MN’: 1383859183001:’Timothy’, ‘MN’: 1383859182915:’T’} ContactInfo: {‘EA’: 1383859183030:‘John.Smith@xyz.com’, ’SA’: 1383859183073:’1 Hadoop Lane, NY 11111’} |
| 00002 | CustomerName: {‘FN’: 1383859183103:‘Jane’, ‘LN’: 1383859183163:‘Doe’, ContactInfo: { ’SA’: 1383859185577:’7 HBase Ave, CA 22222’} |

The table shows two column families: CustomerName and ContactInfo. When creating a table in HBase, the developer or administrator is required to define one or more column families using printable characters.

Generally, column families remain fixed throughout the lifetime of an HBase table but new column families can be added by using administrative commands. The official recommendation for the number of column families per table is three or less.

In addition, you should store data with similar access patterns in the same column family — you wouldn’t want a customer’s middle name stored in a separate column family from the first or last name because you generally access all name data at the same time.

**4. How many maximum number of columns can be added to HBase table?**

There is no hard limit to number of columns in HBase , we can have more than 1 million columns but usually three column families are recommended ( not more than three).

**5. Why columns are not defined at the time of table creation in HBase?**

Column families must be declared up front at schema definition time whereas columns do not need to be defined at schema time but can be conjured on the fly while the table is up and running.

Physically, all column family members are stored together on the filesystem. Because tunings and storage specifications are done at the column family level, it is advised that all column family members have the same general access pattern and size characteristics.

**6. How does data get managed in HBase?**

Hbase commands can be broadly divided into five categories. The commands operate in a similar way to those in relational databases. Security commands are used to GRANT, REVOKE and show USER\_PERMISSION. Cluster replication commands are used to manage a cluster. Some cluster management activities are: ADD\_PEER, REMOVE\_PEER, DISABLE\_PEER and STOP\_REPLICATION. Data manipulation commands include COUNT, DELETE, DELETEALL and SCAN. Some table management commands are: ALTER, CREATE, DESCRIBE, DROP, and DROPALL. Some general commands are VERSION and STATUS. This is a very small list of commands available for managing data in Hbase. Complete documentation is available online for reference.

The Hbase data model is different from the model provided by relational databases. Hbase is referred to by many terms like a key-value store, column oriented database and versioned map of maps which are correct. The easiest way of visualizing a Hbase data model is a table that has rows and tables. This is the only similarity shared by Hbase model and the relational model.

Data in Hbase is organized into tables. Any characters that are legal in file paths are used to name tables. Tables are further organized into rows that store data. Each row is identified by a unique row key which does not belong to any data type but is stored as a bytearray. Column families are further used to group data in rows. Column families define the physical structure of data so they are defined upfront and their modification is difficult. Each row in a table has same column families. Data in a column family is addressed using a column qualifier. It is not necessary to specify column qualifiers in advance and there is no consistency requirement between rows. No data types are specified for column qualifiers, as such they are just stored as bytearrays. A unique combination of row key, column family and column qualifier forms a cell. Data contained in a cell is referred to as cell value. There is no concept of data type when referring to cell values and they are stored as bytearrays. Versioning happens to cell values using a timestamp of when the cell was written.

Tables in Hbase have several properties that need to be understood for one to come up with an effective data model. Indexing and sorting only happens on the row key. The concept of data types is absent and everything is stored as bytearray. Only row level atomicity is enforced so multi row transactions are not supported.

**7. What happens internally when new data gets inserted into HBase table?**

Bulkload Tool runs the map reduce job behind the scenes and loads the data into HBase tables. These tools internally generates the HBase internal file format HFile, which allows us to import the data into a live HBase cluster.

You can insert data into Hbase using the add() method of the Put class. You can save it using the put() method of the HTable class. These classes belong to the org.apache.hadoop.hbase.client package. Below given are the steps to create data in a Table of HBase.

Step 1:Instantiate the Configuration Class

The Configuration class adds HBase configuration files to its object. You can create a configuration object using the create() method of the HbaseConfiguration class as shown below.

Configuration conf = HbaseConfiguration.create();

Step 2:Instantiate the HTable Class

You have a class called HTable, an implementation of Table in HBase. This class is used to communicate with a single HBase table. While instantiating this class, it accepts configuration object and table name as parameters. You can instantiate HTable class as shown below.

HTable hTable = new HTable(conf, tableName);

Step 3: Instantiate the PutClass

To insert data into an HBase table, the add() method and its variants are used. This method belongs to Put, therefore instantiate the put class. This class requires the row name you want to insert the data into, in string format. You can instantiate the Put class as shown below.

Put p = new Put(Bytes.toBytes("row1"));

Step 4: Insert Data

The add() method of Put class is used to insert data. It requires 3 byte arrays representing column family, column qualifier (column name), and the value to be inserted, respectively. Insert data into the HBase table using the add() method as shown below.

p.add(Bytes.toBytes("coloumn family "), Bytes.toBytes("column

name"),Bytes.toBytes("value"));

Step 5: Save the Data in Table

After inserting the required rows, save the changes by adding the put instance to the put() method of HTable class as shown below.

hTable.put(p);

Step 6: Close the HTable Instance

After creating data in the HBase Table, close the HTable instance using the close() method as shown below.

hTable.close();

Given below is the complete program to create data in HBase Table.

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.hbase.HBaseConfiguration;

import org.apache.hadoop.hbase.client.HTable;

import org.apache.hadoop.hbase.client.Put;

import org.apache.hadoop.hbase.util.Bytes;

public class InsertData{

public static void main(String[] args) throws IOException {

// Instantiating Configuration class

Configuration config = HBaseConfiguration.create();

// Instantiating HTable class

HTable hTable = new HTable(config, "emp");

// Instantiating Put class

// accepts a row name.

Put p = new Put(Bytes.toBytes("row1"));

// adding values using add() method

// accepts column family name, qualifier/row name ,value

p.add(Bytes.toBytes("personal"),

Bytes.toBytes("name"),Bytes.toBytes("raju"));

p.add(Bytes.toBytes("personal"),

Bytes.toBytes("city"),Bytes.toBytes("hyderabad"));

p.add(Bytes.toBytes("professional"),Bytes.toBytes("designation"),

Bytes.toBytes("manager"));

p.add(Bytes.toBytes("professional"),Bytes.toBytes("salary"),

Bytes.toBytes("50000"));

// Saving the put Instance to the HTable.

hTable.put(p);

System.out.println("data inserted");

// closing HTable

hTable.close();

}

}

Compile and execute the above program as shown below.

$javac InsertData.java

$java InsertData

The following should be the output:

data inserted