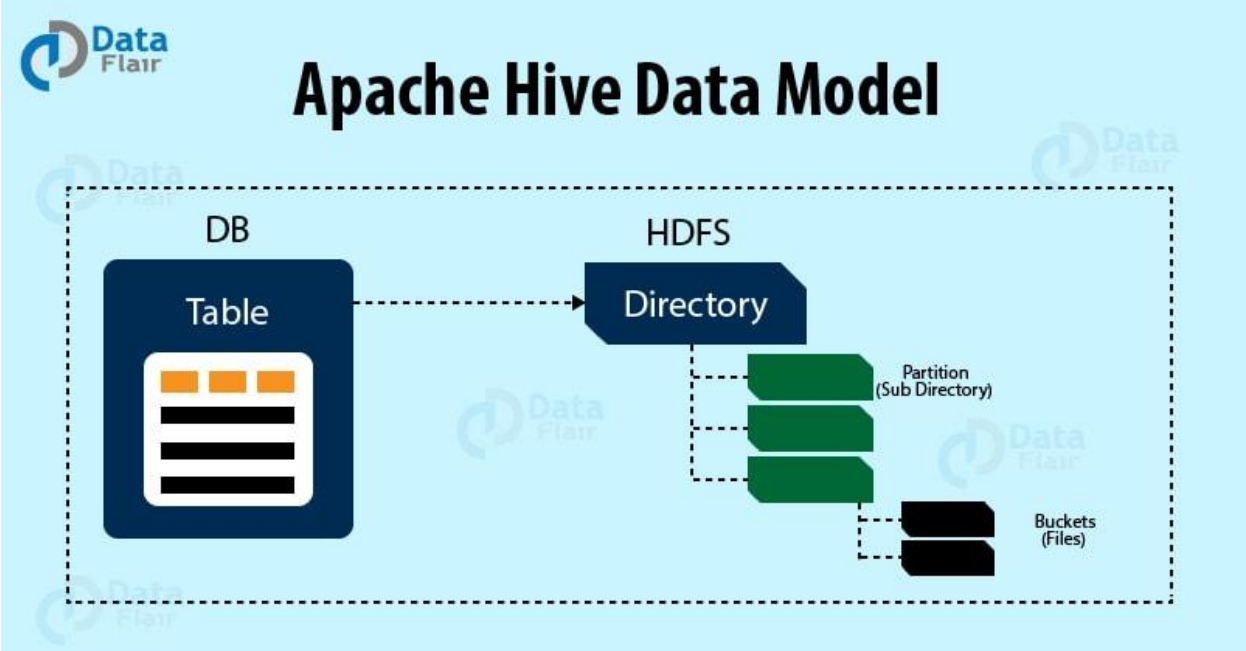
1. Hive Data Model



2. Introduction to Hive Data Model

Hive is an open source data warehouse system built on top of Hadoop Haused for querying and analyzing large datasets stored in Hadoop files. It process structured and semi-structured data in Hadoop. Data in Apache Hive can be categorized into:

- Table
- Partition
- Bucket

Let us now understand these data modeling considerations in Hive one by one-

2.1. Table

Apache Hive tables are the same as the tables present in a Relational Database. The table in Hive is logically made up of the data being stored. And the associated metadata describes the layout of the data in the table. We can perform *filter, project, join* and *union* operations on tables. In Hadoop data typically resides in **HDFS**, although it may reside in any Hadoop filesystem, including the local filesystem or **S3**. But Hive stores the metadata in a relational database and not in HDFS. Hive has two types of tables which are as follows:

- Managed Table
- External Table

In Hive when we create a table, Hive by default manage the data. It means that Hive moves the data into its warehouse directory. Alternatively, we can also create an external table, it tells Hive to refer to the data that is at an existing location outside the warehouse directory.

We can see the difference between the two table type in the **LOAD** and **DROP**semantics. Let us consider a Managed table first.

2.1.1. Managed Table

When we load data into a **Managed table**, Hive moves data into Hive warehouse directory.

For example:

```
1. CREATE TABLE managed_table (dummy STRING);
2. LOAD DATA INPATH '/user/tom/data.txt' INTO table managed_table;
```

This will move the file ***hdfs://user/tom/data.txt*** into Hive's warehouse directory for the managed_table table, which

is ***hdfs://user/hive/warehouse/managed_table***.

Further, if we drop the table using:

```
DROP TABLE managed_table
```

Then this will delete the table including its data and metadata. It bears repeating that since initial LOAD performed a move operation. And the DROP performed a delete operation. The data no longer exists anywhere. This is what it means for HIVE to manage the data.

2.1.2. External Table

The **external table** in Hive behaves differently. We can control the creation and deletion of the data. The location of the external data is specified at the table creation time:

```
1. CREATE EXTERNAL TABLE external_table (dummy STRING)
2. LOCATION '/user/tom/external_table';
3. LOAD DATA INPATH '/user/tom/data.txt' INTO TABLE external_table;
```

Now, with the EXTERNAL keyword, Apache Hive knows that it is not managing the data. So it does not move data to its warehouse directory. It does not even check whether the external location exists at the time it is defined. This very useful feature because it means we create the data lazily after creating the table.

An important thing to notice is that when we *drop* an external table, Hive will leave the data untouched and only delete the metadata.

2.2. Partition

Apache Hive organizes tables into **partitions** for grouping same type of data together based on a column or partition key. Each table in the hive can have one or more partition keys to identify a particular partition. Using partition we can also make it faster to do queries on slices of the data.

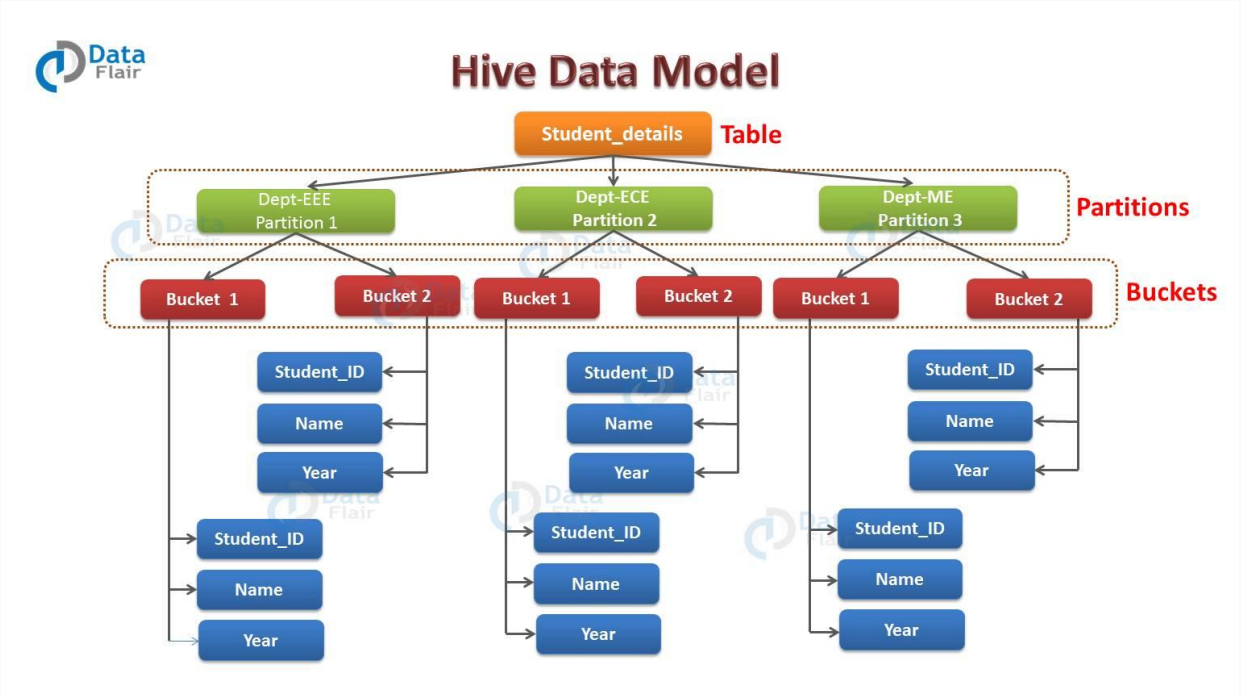
Command:

```
CREATE TABLE table_name (column1 data_type, column2 data_type)
PARTITIONED BY (partition1 data_type, partition2 data_type,….);
```

By example, we can understand partition. Consider you have a table student_details containing the student information of some engineering college like student_id, name, department, year, etc. Now, if you want to perform partitioning on the basis of department column. Then the information of all the students belonging to a particular department will

be stored together in that very partition. Physically, a partition in Hive is nothing but just a sub-directory in the table directory.

Suppose you have data for three departments in our student_details table – EEE, ECE and ME. Thus you will have three partitions in total for each of the departments as you can see in the diagram below. For each department, you will have all the data regarding that very department residing in a separate sub – directory under the table directory.



So for example, all the student data regarding EEE departments will be stored in user/hive/warehouse/student_details/dept.=EEE. So, the queries regarding EEE students would only have to look through the data present in the EEE partition.

Hence, from above example, you can say that partitioning is very useful. Because it reduces the query latency by scanning only **relevant** partitioned data instead of the whole data set. In real world implementations, you might be dealing with hundreds of TBs of data. So, you can imagine scanning this huge amount of data for some query where **95%** data scanned by you was un-relevant to your query.

2.3 Buckets

In Hive, Tables or partition are subdivided into **buckets** based on the hash function of a column in the table to give extra structure to the data that may be used for more efficient queries.

Command:

CREATE TABLE table_name PARTITIONED BY (partition1 data_type, partition2 data_type,….) CLUSTERED BY (column_name1, column_name2, …) SORTED BY (column_name [ASC|DESC], …)] INTO num_buckets BUCKETS;
Each bucket in Hive is just a file in the table directory (unpartitioned table) or the partition directory. So, you have chosen to divide the partitions into n buckets. Then you will have n files in each of your partition directories. Hence, from the above diagram, you can see, where you have bucketed each partition into 2 buckets. Therefore each partition, say EEE, will have two files where each of them will be storing the EEE student's data.