What is HIVE?

Apache Hive is a data warehouse system built on top of Apache hadoop. It provides an SQL-like interface known as **HiveQL**. This stores the large dataset in hadoop HDFS or in a different storage such as AWS S3.

Hive is a Structured Query Language.

Why was HIVE needed?

So, in Hadoop to process data you develop code to run it in Map Reduce. This code is written in Java, and to process the SQL queries can be difficult in this case. This is why HIVE came into place. Hive works under Hadoop.

You use HIVE for the SQL, you write the program in SQL and then run it, HiveQL will automatically convert the HiveQL queries into MapReduce and run them.

This is useful because for SQL you can simply run SQL commands and run the programs. However, if you use MapReduce only then the issue will be that it runs in Java. You need to manually write a lot of items in the program to ensure they run the SQL queries.

Other Information:

Hive uses **Schema-on-read** instead of **Schema-on-write**. Schema on write is the traditional way how a schema has always been created. A **schema** is the table the way it is described in a database, *such as an employee table with employee_name, employee_age*, this is a schema. The way I just mentioned it, this is the traditional way and I have to define the columns initially before creating the schema in the database. I have to mention that I want to create a table called employee and it will have columns with details of id, name, age, department, salary, and etc. Once this is done then I can populate it with information.

However, in **schema-on-read** it is a bit different. Here you can add whatever you want to and then define a schema and once that is done you can run the program via Hive and it will query the program and automatically convert it into MapReduce format to execute it easily without any other extra thing from our side to do.

Components of HIVE:

1. Hive server is the main component
2. Meta store

Hive Server: This converts your query to run jobs (runs your query). We require multiple Hive server for **high-availability**

Meta Store: Stores all the data of the structured query. Stores query in RDBMS.
The data is stored in a third party RDBMS in Hive.

Types of tables in Hive:

1. Managed table (curd) (ACID) transactional (always needs to be stored in a specific location)
2. External table (worm) (NON ACID) non transactional (can be stored anywhere in hive which is why it is called external table)

Managed Table:
In hive we have managed tables that are ACID. These are tables owned, managed, and stored within Hive only. The files are stored in a specific location within Hive and that will always be its storage location. Usually, the storage path is like this: ‘ */user/hive/warehouse/employees/* ’ when we delete the metadata the table is also deleted. These types of tables are good for small and personal work. These tables can be updated, deleted and read as you like.

External Table:

In Hive, the external tables are non-ACID. These types of tables are good when shared between other systems too such as Apache, Spark. These tables are stored outside Hive like this ‘ *LOCATION '/data/employees/';* ’. When Hive deletes the table, only the metadata is deleted but the files remain there. This is for safety since other systems are using the same file too, this saves the file from being deleted by mistake.

The reason why Hive uses the write once read many times principle in External tables is because, Hive only directs towards where the table is already created to apply the query on. This table is not good for transactional processes. External tables are good for batch processings, to run queries on huge chunks of data that are stored in **HDFS**, Hadoop distributed file system.

What is ACID and Non ACID:

ACID is a type of rule of format that is used in databases, mostly SQL, postgreSQL. In ACID we ensure data consistency and reliability is very critical. The data has to be always correct, cannot be altered, cannot be seen by others. Such as the banking system, it has to be 100% correct. The data has **atomicity**, the execution can happen or not happen, there is never something half happening. This is very important when it comes to banking, you can send someone £100 or not send them. It is not like you send someone £100 and for some technical issue, they get £50. This can never be the case. Also the transaction will either happen or not happen. There is no partiality in atomicity. Same way it also has the **consistency** where the data is always consistent. £100 was sent, that will always remain £100. What is chosen or said will only be the desired choice or action. It cannot be altered. The transactions are **isolated** from each other and at the end the data is **durable** meaning that it cannot be changed or altered. In ACID principle, it is ensured that strict rules are applied to ensure data is consistent and correct.

In **Non-Acid**, this is used for huge amounts of data in Big Data, or for Analytical purposes. This is where consistency or accuracy is not restricting the queries to be performed. These are ideal for NoSQL. These are highly scalable, highly available.

ACID tables are good for e-commerce shops for purchase, and banking systems to ensure safety, consistency and avoid any errors. Whereas Non-ACID tables are best of big data, Hive, Hadoop or Analytics.

Partitioning

Partitioning in Hive is like separating your data into separate folders based on a specific column value like year, region, or country

For instance if you have data of student scores from 2023 and 2024. You can partition them into separate folders, this way we now have two folders with student scores. One from 2023 and the other from 2024. If I want to query through the year 2024, it will not go through 2023. This helps in large datasets for faster query. If the scores of the student from 2023 and 2024 were in the same folder the query had to go through the whole data to separate 2024 and then proceed with the query. This can be time consuming in large data sets. Thus partitioning is useful here. Hive will query through the year 2024 only and will provide the required results much faster this way.

It is best to use partitioning when you have got frequent search queries for a specific column, such as Date, Country, Region. Partitioning helps with faster querying on specific columns

Bucketing

Bucketing is like splitting your data into evenly sized groups. This is helpful in joins.

When We Join Tables Without Bucketing: Suppose you have two tables:
Table 1: Contains student records (student_id, name)
Table 2: Contains student scores (student_id, score)

Without bucketing, Hive will scan both tables entirely to find matching records for the student_id. If the tables are huge (e.g., millions of rows). Bucketing distributes data evenly into buckets based on a common column (e.g., student_id).

For example, if both Table 1 and Table 2 are bucketed by student_id into 4 buckets, Hive will know that:
Bucket 1 in Table 1 corresponds to Bucket 1 in Table 2,
Bucket 2 in Table 1 corresponds to Bucket 2 in Table 2,
And so on.

This means that when Hive performs the join, it can join matching buckets from both tables, instead of scanning all data.

Sources:

1. *https://docs.cloudera.com/runtime/7.3.1/using-hiveql/topics/hive_hive_3_tables.html*
2. *https://www.youtube.com/watch?v=RFS4JJFlzGc*