Hive

1.Difference between Data warehouse and database?

Database:

Any collection of data organized for storage, accessibility, and retrieval.

There are different types of databases, but the term usually applies to an OLTP application database, which we’ll focus on throughout this table. Other types of databases include OLAP (used for data warehouses), XML, CSV files, flat text, and even Excel spreadsheets. We’ve actually found that many healthcare organizations use Excel spreadsheets to perform analytics (a solution that is not scalable).

Data Warehouse:

A type of database that integrates copies of transaction data from disparate source systems and provisions them for analytical use.

A data warehouse is an OLAP database. An OLAP database layers on top of OLTPs or other databases to perform analytics.

Not all OLAPs are created equal. They differ according to how the data is modelled.

Database:

OLTP - online transactional processing

Acid properties are fullfilled

Data stored in normalized format

Traditional data

WFS ,LFS

Write many read many

Works on single machines

Works slow

Datawarehouse:

OLAP - online analysis processing

Acid properties are not fullfiled

Bigdata

Write once read many

Works fast

2.Difference between Data warehouse and data mart?

Data Warehouse:

Data is of multiple subject/category/dimension. Eg: all stores,products,orders

Contains details info

Data size is large

Integrates data from various resources

DW provides data to dimension model

Data Mart:

Data is of one subject/category/dimension. Focuses on one dimension.

Holds summarized data

Comparatively its small

Integrates data from various resources for a given subject/category/dimension

DM has dimension - Data Modelling (star n snowflake schema)

A data warehouse serves as a centralized repository for all organizational data, while data marts offer tailored views of data for specific business areas or user groups within the organization.

3.Difference between OLTP vs OLAP

• OLTP: Optimized for managing and processing day-to-day transactional data generated by business operations, such as sales transactions, order processing, and inventory management.

• Uses a normalized data model to minimize redundancy and ensure data integrity. Focuses on efficiently adding, updating, and deleting individual records.

• Supports real-time transaction processing and ensures high availability, reliability, and concurrency for operational tasks.

Involves simple, read/write operations on individual records or small sets of records. Prioritizes fast response times and low latency.

Involves complex analytical queries, aggregations, and data mining operations across multiple dimensions and hierarchies. Emphasizes query performance and scalability.

Characterized by short, transactional queries with frequent data modifications and high concurrency requirements.

EX: E-commerce transactions, ATM withdrawals, online banking activities.

• OLAP:

• Designed for analysing and querying large volumes of historical data to support decision-making, trend analysis, and business intelligence activities.

• Utilizes a denormalized or dimensional data model (e.g., star schema) optimized for complex queries and aggregations across multiple dimensions.

• Enables complex ad-hoc queries, reporting, and data analysis across large datasets to derive insights and support strategic decision-making.

Ex: Business reporting, data mining, trend analysis, customer segmentation.

4. Why hive metadata is stored in SQL?

Hive metadata is stored in SQL-based databases for several reasons:

Compatibility: SQL-based databases are widely used in enterprise environments for managing structured data. By storing Hive metadata in SQL databases such as MySQL, PostgreSQL, or Oracle, Hive integrates seamlessly with existing data management systems and tools.

Scalability: SQL databases are designed to handle large volumes of structured data efficiently. By leveraging the scalability and performance capabilities of SQL databases, Hive can manage metadata for large-scale data warehouses and analytics platforms.

Concurrency: SQL databases provide robust support for concurrent access and transaction management. Storing Hive metadata in a SQL database enables multiple users to interact with Hive concurrently while ensuring data consistency and integrity.

Reliability: SQL databases offer features such as data replication, backups, and recovery mechanisms, which enhance the reliability and availability of Hive metadata. In case of failures or disasters, these features help ensure that metadata can be restored and accessed without data loss.

Security: SQL databases provide advanced security features such as role-based access control (RBAC), encryption, and auditing. Storing Hive metadata in a SQL database allows organizations to enforce security policies and access controls on metadata access, ensuring data protection and compliance with regulatory requirements.

5. Which SQL is the default database for hive?

\*\* Default storage to store metadata in hive : Derby. (Admin can configure any other DB)

The default SQL database for Apache Hive is Apache Derby, which is an open-source relational database written in Java. Apache Derby is lightweight, easy to set up, and suitable for development and testing purposes. However, for production environments, organizations often use more robust SQL databases such as MySQL, PostgreSQL, or Oracle as the backend storage

for Hive metadata.

6. What is managed table?

A managed table in Apache Hive is a table where Hive manages both the metadata and the data files associated with the table. When you create a managed table in Hive, Hive is responsible for managing the lifecycle of the table, including storing metadata in the metastore and managing the underlying data files on Hadoop Distributed File System (HDFS) or another storage system.

7. What is external table?

External tables is not managed by hive. Location of data is not default hive. It can be any location that user provides while creating a table.

If the data is used

An external table in Apache Hive is a table where Hive manages the metadata, but the data files associated with the table are stored externally, outside of the default Hive warehouse directory. When you create an external table in Hive, you specify the location of the data files, which can be on the Hadoop Distributed File System (HDFS), a different file system, or an external storage system such as Amazon S3 or Azure Data Lake Storage.

8. When do we use external table?

External tables are useful for scenarios where data needs to be shared or accessed across multiple systems, or when users want to maintain control over the data files outside of Hive. They provide flexibility and interoperability, enabling integration with other data processing tools and systems.

When data needs to be shared across multiple systems or processed by other tools outside of Hive, When data already exists in a specific location or format, external tables can be used to access and query that data without moving or copying it into the Hive warehouse directory.

When working with temporary or transient data that does not need to be persisted in Hive, external tables can be used to access and query the data without the need for permanent storage within Hive.

external tables are used when there is a need to access data stored externally, share data across systems, maintain data independence, or work with temporary data without moving it into the Hive warehouse directory.

9. Diff between managed and external table?

Managed Table:

Hive manages both metadata and data files.

Data files are stored within the default Hive warehouse directory.

Hive controls the lifecycle of data files, including creation, deletion, and clean up.

Typically used for permanent data storage within Hive.

Suited for scenarios where data is managed entirely by Hive and doesn't need to be accessed or manipulated externally.

External Table:

Hive manages only metadata, while data files are stored externally.

Users specify the location of data files, which can be on HDFS, another file system, or an external storage system.

Users have control over the lifecycle of data files, including creation, deletion, and maintenance.

Suitable for scenarios involving data sharing, existing data, or temporary data, where data needs to be accessed externally or shared across systems.

Provides flexibility and data independence, allowing users to manipulate data files separately from Hive.

10. What happens if you don’t provide location to external table?

If you don't provide a location to an external table in Apache Hive, the table creation will fail because the location of the data files must be specified for external tables. Unlike managed tables, where Hive manages both metadata and data files internally within the default Hive warehouse directory, external tables require users to explicitly specify the location of the data files.

Without specifying a location for the external table, Hive won't know where to find the data files associated with the table, and therefore won't be able to create the table successfully. Hive will likely return an error indicating that a location for the external table data files must be provided.

11. Performance optimization in hive?

Partitioning: Dividing data into partitions to reduce data scanned during queries.

Bucketing: Distributing data evenly across files to improve query efficiency.

Optimized File Formats: Using formats like ORC or Parquet for efficient storage and query processing.

Compression: Employing compression techniques to reduce disk I/O and storage overhead.

Vectorization: Enabling vectorized query execution for faster processing.

Caching and Materialized Views: Utilizing caching and materialized views for frequently accessed data.

Query Tuning: Analysing and optimizing query execution plans and syntax.

Hardware and Cluster Configuration: Optimizing hardware resources and cluster settings for efficient processing.

12. Explain partition table. Give example

A partitioned table in Apache Hive is a table that is logically divided into multiple partitions based on the values of one or more partition columns. Partitioning allows data to be organized and stored in a directory structure based on partition columns, which can significantly improve query performance by reducing the amount of data scanned during query execution.

Example: Suppose we have a dataset of sales transactions with the following columns:

transaction\_id: Unique identifier for each transaction.

transaction\_date: Date of the transaction.

product\_id: Identifier for the product purchased.

amount: Amount of the transaction.

We want to create a partitioned table to store this data, partitioned by transaction\_date to improve query performance.

CREATE TABLE sales (

transaction\_id INT,

product\_id INT,

amount DECIMAL(10, 2)

)

PARTITIONED BY (transaction\_date DATE)

STORED AS ORC;

the sales table is partitioned by the transaction\_date column. When data is inserted into this table, Hive automatically creates directories in the table's location based on the distinct values of transaction\_date, and data files are stored within these directories according to the partitioning scheme.

if we insert sales data for transactions on different dates:

INSERT INTO TABLE sales PARTITION (transaction\_date='2022-01-01')

VALUES (1, 101, 50.00), (2, 102, 75.00);

INSERT INTO TABLE sales PARTITION (transaction\_date='2022-01-02')

VALUES (3, 103, 100.00), (4, 104, 120.00);

Hive will create directories like the following:

sales/

└── transaction\_date=2022-01-01

└── <data files>

└── transaction\_date=2022-01-02

└── <data files>

Partitioning the sales table by transaction\_date allows Hive to skip reading partitions that are not relevant to a query, which can significantly improve query performance when filtering by date ranges.

13. Explain bucket table. Give example

A bucketed table in Apache Hive is a type of table where data is organized into a fixed number of buckets based on the hash value of one or more columns. Bucketing can improve query performance by reducing data skew and facilitating more efficient data retrieval and join operations.

EX: Suppose we have a dataset of user profiles with the following columns:

user\_id: Unique identifier for each user.

username: Username of the user.

age: Age of the user.

country: Country of the user.

We want to create a bucketed table to store this data, bucketed by the user\_id column to evenly distribute data across buckets.

CREATE TABLE user\_profiles (

user\_id INT,

username STRING,

age INT,

country STRING

)

CLUSTERED BY (user\_id) INTO 4 BUCKETS

STORED AS ORC;

In this example, the user\_profiles table is bucketed by the user\_id column into 4 buckets. When data is inserted into this table, Hive hashes the user\_id values and distributes the data evenly across the specified number of buckets.

For example, if we insert user profile data:

INSERT INTO TABLE user\_profiles VALUES

(1, 'john\_doe', 30, 'USA'),

(2, 'jane\_smith', 25, 'Canada'),

(3, 'alice\_wong', 35, 'UK'),

(4, 'bob\_jones', 40, 'Australia');

Hive will distribute the data across the 4 buckets based on the hash value of the user\_id, ensuring even distribution of data.

Bucketing can improve query performance, especially for join operations, as Hive can perform more efficient joins by reading data from corresponding buckets rather than scanning the entire dataset. It's important to choose an appropriate number of buckets based on the size of the dataset and the desired level of data distribution.

14. Diff between partition and bucketed table.

Partitioned Table:

Data is divided into logical partitions based on the values of one or more partition columns.

Each partition is stored as a separate directory within the table's location, with data files stored inside these directories.

Partitioning improves query performance by allowing Hive to skip reading partitions that are not relevant to the query, reducing the amount of data scanned.

Partition columns are part of the table schema and used to define the partitioning scheme.

Bucketed Table:

Data is divided into a fixed number of buckets based on the hash value of one or more columns.

Data is distributed evenly across the specified number of buckets, with each bucket containing a subset of the data.

Bucketing improves query performance, especially for join operations, by facilitating more efficient data retrieval from corresponding buckets.

Bucketing columns are specified at table creation and used to determine the bucketing scheme.

In summary, partitioning divides data into logical partitions based on column values, while bucketing divides data into fixed-sized buckets based on hash values. Partitioning improves query performance by skipping irrelevant partitions, while bucketing improves performance by facilitating efficient data retrieval for joins.

15. How is data distributed among buckets?

Data is distributed among buckets in a bucketed table based on the hash value of the specified column(s). When data is inserted into a bucketed table, Hive calculates the hash value of the bucketing column(s) for each record. The hash value determines which bucket the record will be assigned to.

Hive uses a hash function to evenly distribute records across the specified number of buckets. This ensures that data is evenly distributed among the buckets, minimizing data skew and facilitating efficient query processing.

Overall, data distribution among buckets is determined by the hash values of the bucketing column(s), ensuring even distribution and balanced query execution.