HIVE

Herramienta que pertenece al ecosistema de Hadoop, gestiona enormes datasets almacenados bajo HDFS de Hadoop y realiza consultas sobre los mismos.

Sistema de almacén de datos que facilita el manejo sencillo de datos, consultas AD-HOC, y análisis de grandes conjuntos de datos almacenados en sistemas de ficheros compatibles con Hadoop. Dota de estructura los datos y realiza consultas sobre los mismos con SQL (HiveQL).

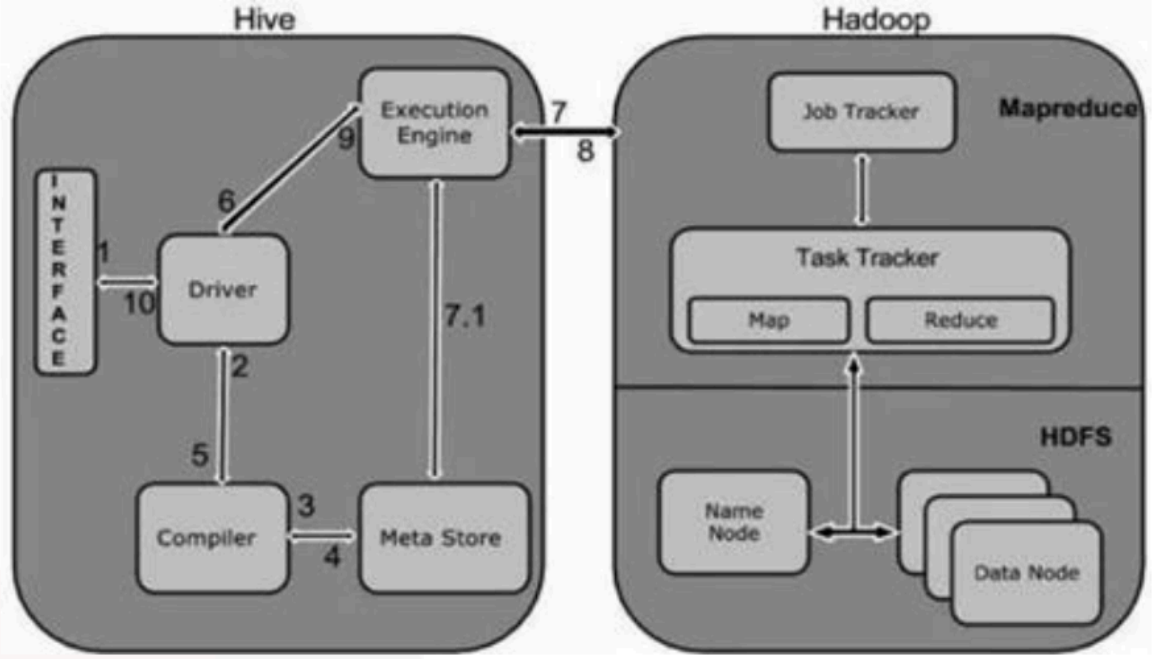
Las consultas expresadas en HQL, son en sí trabajos MapReduce que se ejecutan en Hadoop, también permite a los programadores de MapReduce incluir sus propios mappers y reducers cuando no es necesario usar HQL.

- **Características**
  - HiveQL es un lenguaje tipo SQL para consultas sobre grandes volúmenes de datos en hadoop.
  - Las consultas HQL se ejecutan sobre el modelo MapReduce.
  - Hive usa el metastore para almacenar metadatos y esquemas de datos.
  - Hive se encarga de la traducción de las consultas a tareas MapReduce.
  - Permite crear tus propios mappers y reducers.
  - HQL no permite la inserción, actualización, o borrado de datos a nivel de registro.
  - No dota de transaccionalidad a sus consultas.
  - Latencia superior que en BBDD relacionales.
  - Schema on read vs schema on write. Hive no garantiza que se cumpla estos esquemas, aunque intenta garantizar el esquema en las lecturas
  - Permite crear tablas e insertar datos que están almacenados en el sistema de archivos de Hadoop.

- **Niveles de granularidad (nivel de detalle de los datos)**
  - *BBDD*: espacio de nombres que agrupa tablas y otras unidades de datos.
  - *Tablas*: unidades de datos homogéneas que comparten el mismo esquema.
  - *Particiones*: cada tabla puede tener una o más claves de particionado que determinan cómo se almacenan los datos. Optimizan consultas.
  - *Buckets (Clusters)* : los datos en cada partición pueden dividirse en bucket basados en el valor de una función de dispersión sobre alguna columna de la tabla. Optimizan joins.

- **Funcionamiento**
  1. Ejecutar la consulta: la interfaz de Hive envía una solicitud al controlador para que se ejecute.
  2. Obtener plan: el controlador se ayuda del compilador de consultas que analiza dicha consulta para verificar la sintaxis y el plan de consulta.
  3. Obtener dato: el compilador envía una solicitud de datos al Metastore.
  4. Enviar mensaje: el Metastore envía datos en respuesta al compilador.
  5. Enviar plan: el compilador verifica el registro y envía el plan al controlador, *hasta el momento , el análisis y compilación de una consulta está completo.*
  6. Ejecutar plan: el controlador envía el plan de ejecución al motor de ejecución.

7. Ejecutar tarea: internamente, el proceso de ejecución del Job es un MapReduce, el tiempo de ejecución envía el Job a JobTracker, que se encuentra en el NameNode y asigna este trabajo al TaskTracker, que se encuentra en el DataNode. Aquí la consulta ejecuta el Job MapReduce.
8. Durante la ejecución el motor puede realizar operaciones entre datos y MetaStore.
9. Obtención de resultados: el motor de ejecución recibe los resultados de DataNode.
10. Enviar resultados: el motor de ejecución envía estos valores resultantes al pilot.
11. Enviar resultados: el controlador envía los resultados a las interfaces de Hive.



- Bases de Datos

- **Optimización Sort By**
  La principal diferencia entre SORT BY y ORDER BY es que el primero ordena los datos dentro de cada reducer, por lo tanto, realiza una ordenación local, donde la salida de cada reducer se ordena por separado.

  Los datos se envían a N reducers. Antes de enviar los datos, se ordenan en función de la columna de ordenación, pero esto no garantiza que la salida final de todos los reducers sea ordenada. El orden depende del tipo de columna, lo que significa que si la columna es de tipo numérico, entonces el orden es numérico y si la columna es de tipo cadena, entonces el orden de ordenación es de tipo lexicográfico.

- **Distributed By con Sort By**
  Controla cómo se divide la salida del map entre los reducers, todos los datos que fluyen a través de un Job de MapReduce se organiza en pares clave-valor. Esta cláusula se utiliza para distribuir datos según una clave en particular, HIVE debe utilizar esta función internamente cuando convierte las consultas en Jobs MapReduce.

- **Cluster By**
  Se puede utilizar como alternativa para las cláusulas DISTRIBUTED BY y SORT BY en HQL, Hive usa las columnas de CLUSTER BY para distribuir las filas entre múltiples reducers. Se puede usar cuando los datos están sesgados o cuando se ordenan en varios UNIONs.

- **Explode**
  Una de las funciones usadas en HIVE avanzado es EXPLODE, esta función crea un registro por cada elemento en un array (es muy usado con vistas laterales). Con MAPS, cada K-V se convierte en un registro separado, y cada K y V se convierten en campos separados dentro de cada registro

- **RegexSerDe**
  A veces hay que analizar datos que no tienen un delimitador específico o no está claro, para estos casos usamos RegexSerDe, el cual mediante una expresión regular nos permite extraer la información, cuando los datos tienen un formato fijo y delimitado también podemos usar RegexSerDe hay que tener en cuenta que se leen en formato String por lo que es necesario castear determinados datos.

- **Partitioning y bucketing**
  El partitioning es una técnica de optimización basada en dividir los datos de una tabla en función del valor de sus columnas, en función del uso que le vaya a dar.

  El bucketing es otra forma de subdividir datos para ganar eficiencia

  Calcula un código Hash para valores insertados en las columnas "bucket".
  ● Divide los datos en base a esa columna.
  ● El código Hash es usado para asignar nuevos valores al bucket correspondiente.
  ● En bucketing, el número de participantes es fija. En partitioning no.
  ● En partitioning, todas las filas de la partición tienen la misma key.
  ● El objetivo del bucket es el de distribuir filas a lo largo de un número predefinido de buckets.
    - o Útil para Jobs donde se necesitan samples aleatorios de datos.
    - o El Join de dos tablas que han sido bucketizadas sobre la misma columna de Join, puede ser implementado más eficientemente como un Map Join.

  ● Es una opción muy válida para columnas donde el partitioning no es válido por el problema de "muchos valores con pocos registros".