UD 5 - Apache Hadoop - Hive

Info

Apache Hive puede ejecutarse en clusters tradicionales de Hadoop o en entornos en la nube. Inicialmente, Hive se utilizaba principalmente para **ETL** y el procesamiento por lotes. Aunque sigue soportando estos casos de uso, ha evolucionado para soportar también casos de uso de data warehousing como la elaboración de informes, consultas interactivas e inteligencia empresarial. Esta evolución se ha llevado a cabo mediante la adopción de diferentes técnicas comunes de almacenamiento de datos y adaptando esas técnicas al ecosistema Hadoop. *Hive no está diseñado para cargas de trabajo de procesamiento de transacciones en línea (OLTP). Se utiliza mejor para tareas tradicionales de almacenamiento de datos*

Figura 5.1_Hive: Logo Hive. (Fuente: hive.apache.org)

Introducción

Apache Hive es sin lugar a dudas uno de los componentes más utilizados en el ecosistema Hadoop porque permite que usuarios no técnicos puedan acceder a los datos almacenados en el clúster, así como la integración de herramientas de terceros, como pueden ser herramientas de visualización de datos, con los datos de HDFS.

Y no sólo eso, además ofrece un lenguaje de consultas muy rico, potente y sencillo para los programadores, sustituyendo a MapReduce, que como has visto, requiere el desarrollo de programas complejos en lenguaje Java.

Hive es una tecnología distribuida diseñada y construida sobre **Hadoop**. Permite hacer consultas y analizar grandes cantidades de datos almacenados en **HDFS**, en la escala de *petabytes*. Tiene un lenguaje de consulta llamado **HiveQL** o **HQL** que internamente transforma las consultas SQL en trabajos **MapReduce** que ejecutan en Hadoop. El lenguaje de consulta HQL es un dialecto de *SQL*, que no sigue el estándar *ANSI SQL*, sin embargo es muy similar.

El proyecto comenzó en el 2008 y fue desarrollado por Facebook para hacer que Hadoop se comportara de una manera más parecida a un data warehouse tradicional.

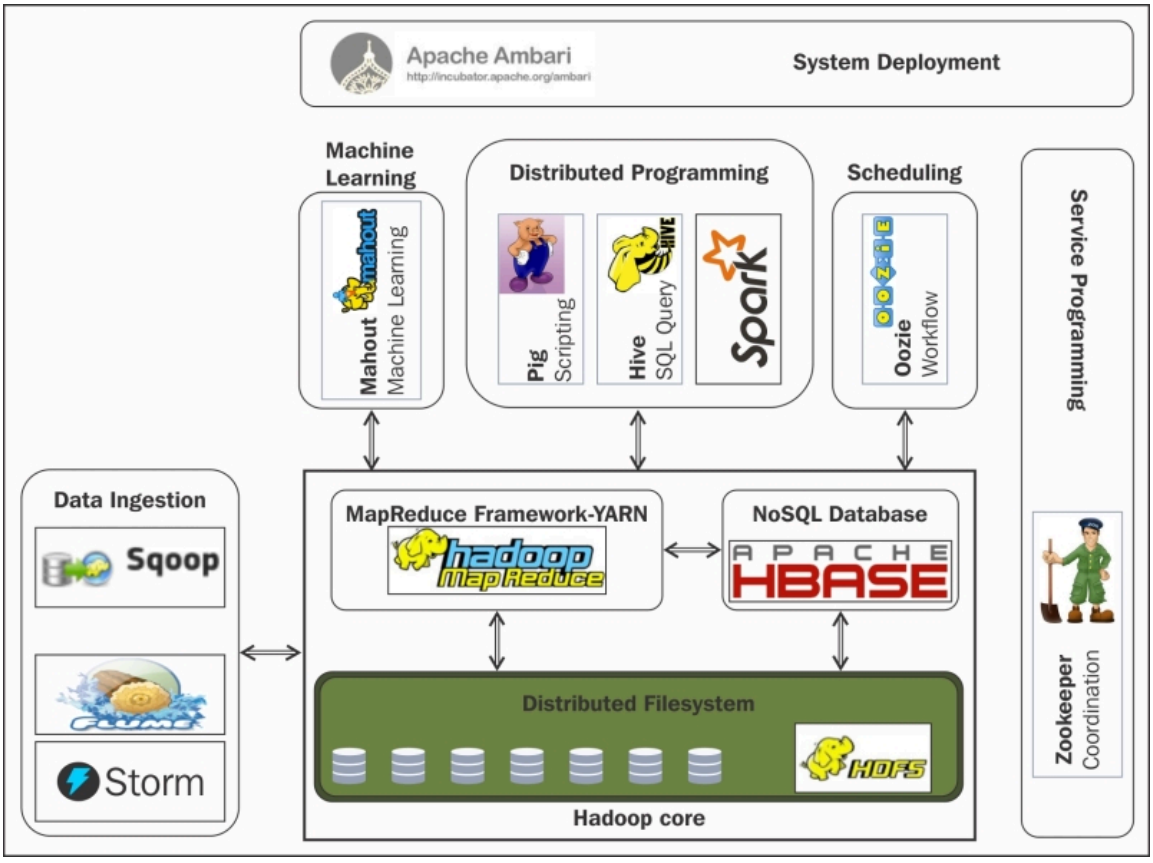

Figura 5.2_Hive: Ecosistema Hadoop. (Fuente: Cloudera)

Los datos gestionados por Hive son datos estructurados almacenados en HDFS. Así, optimiza de forma automática el plan de ejecución y usa particionado de tablas en determinadas consultas. También soporta diferentes formatos de ficheros, codificaciones y fuentes de datos como HBase.

Hive está diseñado para maximizar la escalabilidad (escalar con más máquinas agregadas dinámicamente al clúster de Hadoop), el rendimiento, la extensibilidad, la tolerancia a fallos y el acoplamiento flexible con sus tareas de entrada.

Conceptos Generales

La tecnología Hadoop es altamente escalable, sin embargo presenta limitaciones a la hora de explotar datos en HDFS:

- **Dificultad de uso**: La API de Java es complicada de usar y se necesita experiencia específica para tratar con diferentes formatos de ficheros y sistemas de almacenamiento. Necesita programación de procesos MapReduce para manipular datos, que requiere altos conocimientos de programación.
- **Orientado a operaciones Batch**: No soporta operaciones de acceso aleatorio y no está optimizado para gestionar ficheros pequeños.
- **Falta de integración con herramientas de gestión o explotación de datos**. La mayor parte de las herramientas que se utilizan en las empresas para analizar o visualizar datos, como Excel, las herramientas de Business Intelligence o de visualización, como PowerBI, Tableau, Microstrategy o Cognos, no pueden acceder a los datos de HDFS porque no permiten desarrollar programas MapReduce, ya que el único lenguaje que permiten utilizar para consulta de datos es SQL.
- **Encarecimiento de las soluciones software**. MapReduce requiere programar soluciones para las que es difícil reutilizar código, por ejemplo, si queremos obtener una lista de jugadores ordenada por minutos de juego, y posteriormente queremos una lista ordenada de jugadores por pases realizados, apenas podremos reutilizar el código de la primera para realizar la segunda. Además, MapReduce presenta dificultades para industrializar el desarrollo, es decir, estandarizarlo, automatizar los despliegues, etc.

Ante estas dificultades, un grupo de ingenieros de Facebook desarrolló Hive como una herramienta que permite simplificar las tarea de analítica con ficheros de HDFS, **utilizando un lenguaje similar a SQL para su acceso**.

Hive, y el lenguaje de consultas que ofrece, denominado **HQL**, se ha convertido en un estándar de facto del mundo Big Data.

Una consulta típica en Hive se ejecuta en varios datanodes en paralelo, con trabajos MapReduce asociados. Estas operaciones son de tipo batch, por lo que la latencia es más alta

que en otros tipos de bases de datos. Además, hay que considerar el retardo producido por la inicialización de los trabajos, sobre todo en el caso de consultar pequeños datasets.

Hive fue incluido como proyecto Apache, lanzando su primera versión estable en 2010.

Características y funcionalidades

- Permite definir una estructura relacional (tablas, campos, etc.) sobre la información "en bruto" de HDFS.
- Ofrece un **lenguaje de consultas, denominado HQL**, de sintaxis muy similar a SQL incluyendo muchas de las evoluciones de SQL para analítica: SQL-2003, SQL-2011 y SQL-2016. Esto permite que los analistas de negocio, o en general, personas sin mucho conocimiento técnico o de programación, puedan acceder a los datos de HDFS y consultarlos con un lenguaje muy rico y estándar, que ofrece operaciones para hacer cálculos, agregaciones, filtrado, etc.
- Interfaces estándar con **herramientas de terceros mediante JDBC/ODBC**
- Utiliza **motores de procesamiento de Hadoop estándar** para la ejecución de consultas: MapReduce, Spark o Tez, realizando la traducción automática entre el lenguaje de consultas (HQL) y el código de los programas a ejecutar.
- **Mecanismos de seguridad** en el acceso y modificación de la información: Hive permite definir permisos a nivel de tabla o de operación (consulta, modificación, borrado, ...) con una sintaxis similar a la de las bases de datos relacionales.
- Lee ficheros con **diferentes formatos**: texto plano, ORC, HBase, Parquet y otros.
- Extensible mediante **código a medida (UDF)**: en el caso de que el lenguaje HQL no ofrezca una funcionalidad que se requiere para una consulta, se puede implementar una función a medida, denominada UDF, que consiste en un código Java que se añade a la consulta y que permite realizar operaciones no estándar, como podría ser realizar una transformación de textos o eliminar palabras u ofuscar cantidades en una consulta.
- **Orientado a consultas** aunque también ofrece operativa para creación, modificación o borrado de registros: aunque Hive tiene operaciones de inserción, borrado o modificación de registros, no está optimizado para ello, sino para consultas, especialmente para consultas complejas sobre un volumen de datos elevado.

Arquitectura

La arquitectura de Hive dispone de los siguientes componentes:

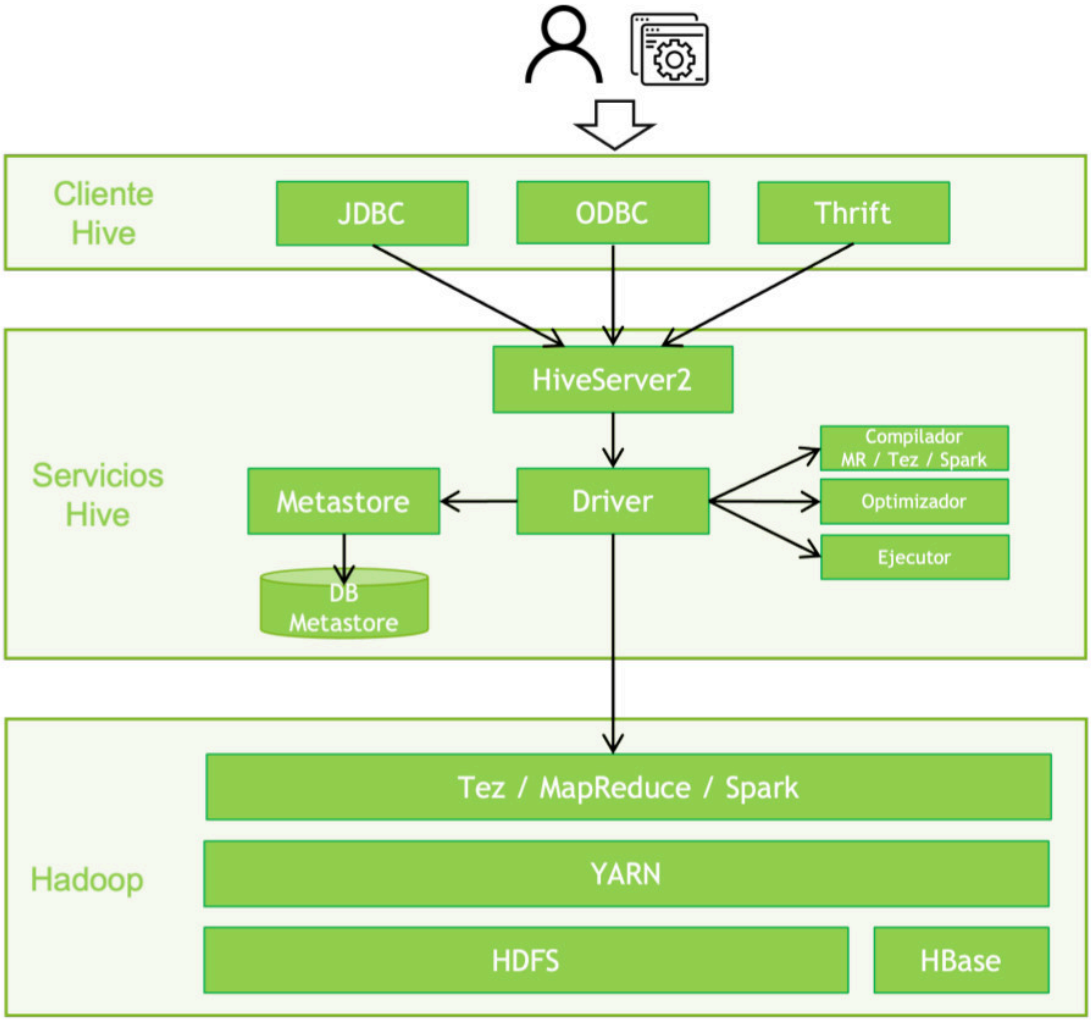

Figura 5.3_Hive: Arquitectura Hive. (Fuente: Ministerio de Educación)

Cliente

Hive permite escribir aplicaciones en varios lenguajes, incluidos Java, Python y C++. Dispone de tres tipos de cliente:

- **Thrift Server**: Apache Thrift es un protocolo e implementación para publicar y consumir servicios binarios similar a RPC que puede ser utilizado por cualquier lenguaje de programación. Hive es capaz de ofrecer este protocolo, por lo que aplicaciones que utilicen Thrift pueden utilizar este cliente para invocar a Hive.
- **Cliente JDBC**: Hive permite que las aplicaciones Java se conecten mediante el controlador JDBC. El controlador JDBC usa Thrift para comunicarse con el servidor Hive.
- **Cliente ODBC**: el controlador ODBC de Hive permite que las aplicaciones basadas en el protocolo ODBC se conecten a Hive. La mayoría de aplicaciones que se integran con Hive utilizan este cliente (Excel, herramientas de Business Intelligente, etc.). Al igual que el controlador JDBC, el controlador ODBC usa Thrift para comunicarse con el servidor Hive.

Los clientes se ejecutarán en la máquina que invoca a Hive, incluyendo el paquete de software de cliente de la distribución de Hive.

Servicios Hive

Estos servicios se suelen ejecutar en un nodo frontera, donde se instala todo el software que recibe las peticiones de los clientes, las traduce a trabajos **MapReduce, Tez o Spark**, las ejecuta en el clúster Hadoop, y devuelve los resultados al cliente.

> **Tip**
>
> **Recuerda**: El clúster Hadoop ejecuta los trabajos resultantes de las consultas como cualquier otra tarea YARN, es decir, asignándole una prioridad, un ApplicationMaster, etc. Por lo tanto, el desempeño de las consultas Hive dependerá en gran medida de la capacidad de ejecución del clúster Hadoop.

Los servicios principales son los siguientes:

- **HiveServer**: recibe las peticiones de los clientes e inicia su resolución. Permite peticiones concurrentes y desde interfaces variados como JDBC, ODBC o Thrift.
- **Driver**: este componente gobierna toda la ejecución de la petición, utilizando el resto de servicios como Metastore, el compilador, etc.
- **Metastore**: contiene el registro de todos los metadatos de Hive y otros componentes que requieren aplicar un modelo sobre datos existentes en HDFS o HBase. Por ejemplo, almacena la información de las tablas definidas, qué campos tienen, con qué ficheros se corresponden, etc. Permite que aplicaciones distintas a Hive puedan usar la definición de tablas definida en Hive, o al revés.
- **Compilador**: analiza la consulta. Realiza análisis semánticos y verificación de tipos en los diferentes bloques de consulta y expresiones de consulta utilizando los metadatos almacenados en metastore y genera un plan de ejecución. El plan de ejecución creado por el compilador es el DAG (Gráfico acíclico dirigido), donde cada etapa es un trabajo de mapeo/reducción, operación en HDFS, una operación de metadatos. Es decir, este componente realiza la traducción de HQL a un programa MapReduce, Tez o Spark.
- **Optimizador**: realiza las operaciones de transformación en el plan de ejecución y divide la tarea para mejorar la eficiencia y la escalabilidad.
- **Ejecutor**: ejecuta el plan de ejecución creado por el compilador y mejorado por el optimizador en el orden de sus dependencias usando Hadoop.

Diseño completo de la arquitectura

Si quieres saber con más detalle cual es el diseño de la arquitectura de Hive, accede a este recurso de la documentación oficial

HQL

Es un lenguaje con una sintaxis similar a SQL. En las primeras versiones de Hive, HQL no incluía muchas de las sentencias o funcionalidades de SQL, pero con el tiempo ha ido ganando en funcionalidad hasta llegar a ser un lenguaje prácticamente idéntico a SQL.

En primer lugar, es preciso conocer qué modelo conceptual utiliza Hive para tratar con los datos:

Figura 5.4_Hive: Modelo conceptual HQL - Hive. (Fuente: Ministerio de Educación)

Hive tiene diferentes niveles de entidades:

- El concepto general es una **tabla**, que contiene información de una misma entidad, por ejemplo, la tabla ventas, empleados o tiendas.
- La tabla tiene **campos**, que es cada una de las propiedades que pueden tener todos los elementos de una tabla.
- Las tablas contienen registros o filas, que serían cada una de las ventas que hay en la tabla. Las tablas se pueden dividir en **particiones**. Una partición es una división vertical de la tabla:

Partition

Imagina que la tabla de ventas tiene 50.000.000.000 registros porque incluye las ventas de todas las tiendas de todos los países. La mayor parte de las consultas que se realizan son para obtener diferentes métricas de una tienda (ingresos, número de ventas, ticket medio, etc.). Sin particionar, significa que para cada consulta en la que se quiera calcular una métrica, es necesario leer todos los registros de la tabla. Particionar permite dividir la tabla, por ejemplo, teniendo una partición por cada tienda y año, es decir, tendríamos una partición para la tienda 1 y el año 2022, otra para la tienda 1 y año 2021, otra para la tienda 2 y año 2022, etc. Cada partición se va a almacenar en ficheros o directorios distintos. Ahora, si queremos hacer una consulta para obtener el total de ventas de la tienda 1 en el año 2022, sólo hay que leer los ficheros asociados a esa partición, en lugar de todos los ficheros de la tabla.

Por lo tanto, las **particiones** es un mecanismo para hacer eficientes las operaciones de Hive.

- Las **tablas** se agrupan en bases de datos, teniendo, por ejemplo, una base de datos de ventas, otra base de datos de recursos humanos, etc. La división en bases de datos permite ajustar los permisos así como no tener una multitud de tablas gestionadas por Hive que es difícil de manejar.

En cuanto a los tipos de datos que pueden contener las columnas, Hive permite los siguientes:

- **Enteros**: TINYINT, SMALLINT, INT/INTEGER, BIGINT
- **Decimales**: FLOAT, DOUBLE, DECIMAL, NUMERIC.
- **Fecha/hora**: TIMESTAMP, DATE, INTERVAL.
- **Cadenas**: STRING, VARCHAR, CHAR.
- **Otros**: BOOLEAN, BINARY.
- **Compuestos**:
  - **ARRAY**: agrupa elementos del mismo tipo, es decir, cada campo almacena una lista de elementos, por ejemplo, en una tabla de ventas, la lista de artículos comprados: ["toalla", "camisa", "pantalón].
  - **MAP**: define parejas de clave-valor, por ejemplo, para la tabla de ventas podría haber un campo con los artículos vendidos con su referencia: {1121: "toalla", 3324: "camisa"}
  - **STRUCT**: define estructuras con propiedades, por ejemplo, para cada venta: {"IP origen": "127.0.0.1", "Tiempo de compra": 13123, "Tipo de cliente": "VIP"}.

Finalmente, para poder realizar estas consultas, o para definir la tabla, HQL utiliza 2 tipos de consultas:

- **DDL (Data Definition Language)**: permite crear, consultar o modificar estructuras de datos (tablas, bases de datos, etc.)

- **DML (Data Manipulation Language)**: permite alta, baja, modificación y consulta de datos.

Sentencias DDL

Son las sentencias con la que se ve o manipula la definición de las bases de datos, tablas, particiones, etc.

En este apartado vamos a realizar un pequeño resumen de las sentencias DDL HQL. Tienes la información completa en la documentación oficial: Hive Data Definition Language

Bases de Datos

Sentencias para crear o modificar bases de datos

- Creación de una base de datos: `CREATE DATABASE my_database`
- Visualización de las bases de datos existentes: `SHOW DATABASES`
- Uso de una base de datos para las consultas que se van a lanzar: `USE my_database`
- Borrado de una base de datos: `DROP DATABASE my_database`
- Modificación de las propiedades de una base de datos: `ALTER DATABASE my_database SET …`
- Visualización de detalle de una base de datos: `DESCRIBE DATABASE my_database`

Tablas

- Creación de una table: `CREATE TABLE`. Ejemplo para crear una tabla de inmuebles en la que existe un identificador, una dirección, la provincia, superficie y precio a partir de un fichero de texto en el que los campos están separador por tabuladores:

```
CREATE TABLE inmuebles (
id INT,
direccion STRING,
provincia STRING,
superficie INT
precio DOUBLE)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;
```

- Visualización de las tablas que hay en una base de datos: `SHOW TABLES`
- Borrado: `DROP TABLE inmuebles`
- Modificación: `ALTER TABLE`. Ejemplo para añadir un campo país a la tabla anterior de inmuebles:

```
ALTER TABLE inmuebles
ADD COLUMNS (pais STRING COMMENT 'Nombre del país')
```

Ejemplo para renombrar la tabla:

```
ALTER TABLE inmuebles RENAME TO casas
```

- Visualización del detalle de una tabla, es decir, para conocer los campos que tiene, formato, etc: `DESCRIBE [EXTENDED|FORMATED] inmuebles`

Sentencias DML

Son las sentencias con las que se puede ver o modificar los datos de las tablas.

En este apartado vamos a realizar un pequeño resumen de las sentencias DML HQL. Tienes la información completa en la documentación oficial: Hive Data Manipulation Language

Consulta

La sentencia `SELECT` se utiliza para consultar los datos de una o varias tablas, aplicando filtros, haciendo agregaciones, uniendo tablas, etc.

Es una sentencia muy potente ya que permite consultar los datos y responder a preguntas complejas sobre ellos. Su sintaxis es muy parecida a la sentencia *SELECT de SQL*, por lo que es recomendable que revises esta sentencia de SQL para poder entender bien todo lo que se puede hacer para el caso de Hive.

La sintaxis de SELECT es la siguiente:

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[ORDER BY col_list]
[CLUSTER BY col_list
| [DISTRIBUTE BY col_list] [SORT BY col_list]
]
[LIMIT [offset,] rows]
```

Para entender su sintaxis, vemos algunos ejemplos:

- Consulta para obtener la dirección de todos los inmuebles de Madrid ordenados por superficie:

```
SELECT id, direccion FROM inmuebles
WHERE provincia = 'Madrid'
ORDER BY superficie
```

- Consulta para obtener cuántos inmuebles de un tamaño mayor a 100 metros hay para cada provincia:

```
SELECT provincia, count(*) FROM inmuebles
WHERE superficie > 100
GROUP BY provincia
```

La sentencia SELECT es muy rica, ofreciendo una gran cantidad de posibilidades, por ejemplo:

- Subconsultas (`SELECT FROM SELECT`): permite anidar varias consultas, tomando una de ellas los datos de la otra.
- JOIN de tablas: permite unir dos tablas para procesarlas, por ejemplo, si hay una tabla de ventas y una tabla de tiendas, se podrían unir las dos tablas para obtener un listado de la dirección de la tienda junto con el total de sus ventas mensuales.

Carga de datos en una tabla

Para cargar datos en una tabla existen tres posibilidades:

- Carga de datos en la tabla desde un fichero existente en **HDFS**. *Para este caso, se utiliza la sentencia a LOAD DATA, por ejemplo, para cargar la tabla inmuebles con los datos de un fichero que está en el disco local:*

```
LOAD DATA LOCAL INPATH './data/inmuebles.csv'
OVERWRITE INTO TABLE inmuebles
```

- Carga de datos en una tabla del resultado de una consulta. *Por ejemplo, para insertar en la tabla inmuebles el resultado de una consulta a una tabla del Catastro:*

```
INSERT OVERWRITE TABLE inmuebles SELECT a.direccion, a.provincia,
a.superficie FROM catastro a;
```

- Inserción de registros fila a fila mediante la sentencia INSERT, por ejemplo:

```
INSERT INTO TABLE inmuebles
VALUES (‘Calle Sol 23, 'Madrid', 'Madrid’, 95, 234452)
```

Modificación o borrado de registros

Si se desea modificar los valores de algunas columnas de los registros de una tabla, se utiliza la sentencia `UPDATE`.

Por ejemplo, para modificar el campo provincia, cambiando el literal "La Coruña" por "A Coruña", se utilizaría la siguiente sentencia:

```
UPDATE inmuebles SET provincia = 'A Coruña' WHERE provincia = 'La Coruña'
```

Para borrar los registros de una tabla que cumplan una condición, se utiliza la sentencia `DELETE`, a la que se puede añadir condiciones de los registros a borrar.

Por ejemplo, para borrar todos los registros de los inmuebles de la provincia de Madrid con una superficie mayor a 500 metros cuadrados, se utilizaría la siguiente sentencia:

```
DELETE FROM inmuebles WHERE provincia = 'Madrid' AND superficie > 500
```

Instalación

> **Success**
>
> Vamos a explicar paso a paso como instalar y configurar Apache Hive. *Nos basaremos sobre nuestra instalación de cluster Hadoop. Lo instalaremos en el nodo master*. Si no tienes configurado el cluster, revisa la documentación correspondiente

Toda la información sobre la instalación puedes encontrarla en la documentación oficial.

1. Prerequisitos:

- Tener instalado y correctamente configurado **Apache Hadoop Common**. *(Si no es así accede a la documentación facilitada en este recurso sobre **Apache Hadoop**)*
- Java 8
- Usar como engine Tez en lugar de Map Reduce (Recomendado por la documentación oficial)

2. Descargar la versión estable más reciente desde CDN de Apache. En nuestro caso, la versión 4.0.1

```
wget https://dlcdn.apache.org/hive/hive-4.0.1/apache-hive-4.0.1-bin.tar.gz
```

3. Una vez descargado, desempaquetamos el archivo descargado con el comando tar y entra dentro de la carpeta:

```
tar -xzvf apache-hive-4.0.1-bin.tar.gz
```

4. Movemos el directorio descomprimido al lugar donde vamos a tener instalado Hive. En nuestro caso, dentro del directorio `$HADOOP_HOME`

```
mv apache-hive-4.0.1-bin $HADOOP_HOME/hive-4.0.1
```

5. Creamos las variables de entorno necesarias. Para ello abrimos el archivo `~/.bashrc` y añadimos al final el siguiente código y ejecuta el comando `source ~/.bashrc`

.bashrc

```
export HIVE_HOME=$HADOOP_HOME/hive-4.0.1
export PATH=$HIVE_HOME/bin:$PATH
export HIVE_CONF_DIR=$HIVE_HOME/conf
```

Warning

Apache Hive (versión 4.0.1) no es compatible con Java 11 o superior, sólo con Java 8. Si instalaste Hadoop con Java 11 o superior, tenemos que instalar la versión 8 y cambiar la variable de entorno `$JAVA_HOME` en el archivo de configuración de Hadoop `hadoop-env.sh` y sustituirla por la ubicación de Java 8 en el sistema.

```
sudo apt-get install openjdk-8-jdk
// Observa la ruta de instalaciones de Java
hadoop@master:~$ update-alternatives --list java
/usr/lib/jvm/java-11-openjdk-amd64/bin/java
/usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
```

hadoop-env.sh

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

No olvides reiniciar Hadoop

6. Recuerda que en `$HIVE_HOME/bin`, que ya están configuradas en $PATH, tenemos las siguientes herramientas:

- `hive`: Herramienta cliente (deprecated)
- `beeline`: Herramienta cliente usada para hiveserve2
- `hiveserver2`: Nos permite arrancar el servidor de Hive
- `schematool`: Nos permite trabajar contra la base de datos de metadatos (Metastore)

7. Comprobamos que hemos instalado correctamente Hive

```
hive --version
```

Nos debe salir la versión de hive

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hadoop-3.4.1/hive-4.0.1/lib/log4j-slf4j-impl-2.18.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop-3.4.1/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
```

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hadoop-3.4.1/hive-4.0.1/lib/log4j-slf4j-impl-2.18.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop-3.4.1/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive 4.0.1
Git git://MacBook-Pro.local/Users/zdeng/workspace/github/hive -r 3af4517eb8cfd9407ad34ed78a0b48b57dfaa264
Compiled by zdeng on Thu Sep 26 10:27:41 CST 2024
From source with checksum 45419248b246139ede889cd9bbd1c2ea
```

8. Descargar la versión estable más reciente de TEZ desde CDN de Apache. En nuestro caso, la versión 0.14.1

```
wget https://dlcdn.apache.org/tez/0.10.4/apache-tez-0.10.4-bin.tar.gz
```

9. Una vez descargado, desempaquetamos el archivo descargado con el comando tar y entra dentro de la carpeta:

```
tar -xzvf apache-tez-0.10.4-bin.tar.gz
```

10. Cambiamos los nombres:

```
mv apache-tez-0.10.4-bin tez-0.10.4
mv apache-tez-0.10.4-bin.tar.gz tez-0.10.4-bin.tar.gz
```

11. Copiamos los archivos en HDFS (recuerda tener el servicio levantado):

```
hdfs dfs -mkdir -p /apps/tez
hdfs dfs -put tez-0.10.4-bin.tar.gz /apps/tez
hdfs dfs -put tez-0.10.4 /apps/tez
hdfs dfs -ls /apps/tez
```

12. Movemos el directorio descomprimido al lugar donde vamos a tener instalado Tez en nuestro sistema de ficheros local. En nuestro caso, dentro del directorio `$HADOOP_HOME`

```
mv tez-0.10.4 $HADOOP_HOME/tez-0.10.4
```

13. Creamos las variables de entorno necesarias. Para ello abrimos el archivo `~/.bashrc` y añadimos al final el siguiente código y ejecuta el comando `source ~/.bashrc`

.bashrc

```
export TEZ_HOME=$HADOOP_HOME/tez-0.10.4
```

```
export PATH=$TEZ_HOME/bin:$PATH
export TEZ_CONF_DIR=$TEZ_HOME/conf
```

Configuración

1. Para configurar Hive, lo primero que tenemos que hacer es acceder a los "templates" de configuración que nos proporciona Apache Hive. Accedemos a directorio `$HIVE_HOME/conf` y ejecutamos los siguientes comandos

```
cp hive-default.xml.template hive-site.xml
cp hive-env.sh.template hive-env.sh
cp hive-exec-log4j2.properties.template hive-exec-log4j2.properties
cp hive-log4j2.properties.template hive-log4j2.properties
```

2. Modificamos el fichero `hive-env.sh` para definir la variable de entorno con las rutas de Hadoop y la configuración de Hive.

```
export HIVE_HOME=$HADOOP_HOME/hive-4.0.1
export HIVE_CONF_DIR=$HIVE_HOME/conf
```

3. Además, debemos usar los siguientes comandos HDFS para crear `/tmp` y `/user/hive/warehouse` *(también conocido como hive.metastore.warehouse.dir)* y configurarlos `chmod g+w` antes de poder crear una tabla en Hive. ***Recuerda levantar HDFS en tu cluster para poder realizar este paso***

```
hdfs dfs -rm -r /tmp
hdfs dfs -mkdir /tmp
hdfs dfs -mkdir -p /user/hive/warehouse
hdfs dfs -chmod g+w /tmp
hdfs dfs -chmod g+w /user/hive/warehouse
```

4. Modificamos el archivo de configuración `hive-site.xml` y añadimos lo siguiente (dentro de las etiquetas `<configuration></configuration>`). Es **importante** que coloques esta configuración al principio del documento, ya que a lo largo del mismo hace referencia al valor de esta configuración, y si no la hemos cambiado al principio, entonces obtiene el valor por defecto, y esto no nos interesa.

hive-site.xml

```
  <property>
      <name>system:java.io.tmpdir</name>
      <value>/tmp/hive/java</value>
  </property>
  <property>
      <name>system:user.name</name>
      <value>${user.name}</value>
  </property>
```

5. Para configurar Tez, lo primero que tenemos que hacer es acceder a los "templates" de configuración que nos proporciona Apache Hive. Accedemos a directorio `$TEZ_HOME/conf` y ejecutamos los siguientes comandos

```
cp tez-default-template.xml tez-site.xml
```

6. Copiamos las librerías jar tez para que hadoop tenga acceso a ellas desde el classpath

```
cp $HOME_TEZ/tez-0.10.4/*.jar $HADOOP_HOME/share/hadoop/common/lib/
```

7. Abrimos el archivo de configuración `tez-site.xml` y modificamos el siguiente parámetro, para que se quede como se indica abajo.

tez-site.xml

```
   <property>
     <name>tez.lib.uris</name>
     <value>hdfs://cluster-bda:9000/apps/tez/tez-0.10.4-
bin.tar.gz,hdfs://cluster-bda:9000/apps/tez/tez-0.10.4/lib,hdfs://cluster-
bda:9000/apps/tez/tez-0.10.4</value>
     <type>string</type>
   </property>
```

Sustituye `cluster-bda` por el nombre de tu cluster hadoop.

Seguridad y acceso de usuarios

Hive y Hadoop tienen seguridad en cuanto acceso de usuarios. Si no tenemos bien configurado nos saldrá el siguiente error:

```
Error: Could not open client transport with JDBC Uri: jdbc:hive2://cluster-bda
open new session: java.lang.RuntimeException:
org.apache.hadoop.ipc.RemoteException(org.apache.hadoop.security.authorize.Authori
User: hadoop is not allowed to impersonate hadoop (state=08S01,code=0)
```

Y es que no tenemos configurado ningún acceso de usuario. Esto hay que configurarlo. Tenemos 3 opciones (nosotros usaremos *Proxy User*):

1. **Anonymous**: Cambiamos la configuración en hive-site.xml de la propiedad `hive.server2.enable.doAs` a `false`, para deshabilitar el "user impersonation". En este caso, cualquier conexión la va a realizar como usuario `anonimous`

hive-site.xml

```
    <property>
        <name>hive.server2.enable.doAs</name>
        <value>false</value>
        <description>Setting this property to true will have HiveServer2
execute Hive operations as the user making the calls to it.</description>
    </property>
```

2. **Proxy User**: Establecer un usuario cliente para cada conexión. Para ello tenemos que hacer uso de la característica Proxy User: superusuarios que actúan en nombre de otros usuarios. Como nos indica la documentación oficial: Un superusuario puede enviar trabajos o acceder a hdfs en nombre de otro usuario.

Example

Por ejemplo: Un superusuario con el nombre de usuario "super" quiere enviar un trabajo y acceder a hdfs en nombre de un usuario joe. El superusuario tiene credenciales de kerberos pero el usuario joe no tiene ninguna. Las tareas deben ejecutarse como usuario joe y cualquier acceso a archivos en namenode debe realizarse como usuario joe. Se requiere que el usuario joe pueda conectarse al nodo de nombre o al rastreador de trabajos en una conexión autenticada con las credenciales de kerberos de super. En otras palabras, super se hace pasar por el usuario joe.

Algunos productos como Apache Oozie necesitan esto.

Para ello tenemos que realizar las siguientes configuraciones, donde debes sustituir a `<usuario>` por el usuario que quieras que haga pasar por otro usuario y `<grupo>` por el grupo que quieres dar permiso para que cualquier usuario de ese grupo tenga este permiso.

core-site.xml

```
    <property>
        <name>hadoop.proxyuser.<usuario>.hosts</name>
        <value>*</value>
    </property>
    <property>
        <name>hadoop.proxyuser.<grupo>.groups</name>
        <value>*</value>
    </property>
```

En mi caso, lo voy a hacer con mi usuario:grupo `hadoop:hadoop` que son los que estamos usando en esta documentación

core-site.xml

```
    <property>
        <name>hadoop.proxyuser.hadoop.hosts</name>
        <value>*</value>
    </property>
    <property>
        <name>hadoop.proxyuser.hadoop.groups</name>
        <value>*</value>
    </property>
```

Puedes estudiar más opciones de configuraciones(varios usuarios, ips, ...) en la documentación oficial enlazada anteriormente.

3. **Authentication/Security Configuration**: HiveServer2 es compatible con conexión anónima con y sin SASL, Kerberos (GSSAPI), paso a través de LDAP, autenticación personalizada conectable y módulos de autenticación conectables (PAM, compatible con Hive 0.13 en adelante). Para establecer estas configuraciones de autentificación, sigue las instrucciones de la documentación oficial

Seguiremos la configuración del cluster teniendo en cuenta esto y usando la opción ***Proxy Users***

8. Configuración de **Proxy User**: Establecemos un usuario cliente para cada conexión. Para ello tenemos que hacer uso de la característica Proxy User: superusuarios que actúan en nombre de otros usuarios. Como nos indica la documentación oficial: Un superusuario puede enviar trabajos o acceder a hdfs en nombre de otro usuario.

core-site.xml

```
    <property>
        <name>hadoop.proxyuser.<usuario>.hosts</name>
        <value>*</value>
    </property>
    <property>
        <name>hadoop.proxyuser.<grupo>.groups</name>
        <value>*</value>
    </property>
```

En mi caso, lo voy a hacer con mi usuario:grupo `hadoop:hadoop` que son los que estamos usando en esta documentación

core-site.xml

```
    <property>
        <name>hadoop.proxyuser.hadoop.hosts</name>
        <value>*</value>
    </property>
    <property>
        <name>hadoop.proxyuser.hadoop.groups</name>
        <value>*</value>
    </property>
```

9. Añadimos Tez a la configuración de Hive. Para ello modificamos los siguiente valores en `hive-site.xml`.

hive-site.xml

```
  <property>
      <name>hive.tez.container.size</name>
      <value>1024</value>
  </property>

  <property>
      <name>hive.execution.engine</name>
      <value>tez</value>
  </property>
```

Y añadimos los siguientes:

hive-site.xml

```
   <property>
       <name>tez.lib.uris</name>
       <value>hdfs://cluster-bda:9000/apps/tez/tez-0.10.4-
bin.tar.gz,hdfs://cluster-bda:9000/apps/tez/tez-0.10.4/lib,hdfs://cluster-
bda:9000/apps/tez/tez-0.10.4</value>
   </property>

   <property>
       <name>tez.configuration</name>
       <value>/opt/hadoop-3.4.1/tez-0.10.4/conf/tez-site.xml</value>
   </property>

   <property>
       <name>tez.use.cluster.hadoop-libs</name>
       <value>true</value>
   </property>
```

10. Reiniciamos los servicios hadoop

```
stop-all.sh
start-dfs.sh
start-yarn.sh
```

11. Ahora debemos usar "*The Hive Schema Tool*" con el comando `schematool`. La distribución de Hive ahora incluye una herramienta *offline* para la manipulación de esquemas de **metastore** de Hive. Esta herramienta se puede utilizar para inicializar el esquema de **metastore** para la versión actual de Hive. También puede gestionar la actualización del esquema de una versión anterior a la actual. `schematool` determina los scripts SQL necesarios para inicializar o actualizar el esquema y luego ejecuta esos scripts en la base de datos back-end. La información de conexión de **metastore** DB como la URL de JDBC, el controlador JDBC y las credenciales de DB se extraen de la configuración de Hive. Puede proporcionar credenciales de base de datos alternativas si es necesario. (configurando en `hive-site.xml`).

```
schematool -help
```

```
hadoop@master:~/hadoop-3.4.1/hive-4.0.1/conf$ schematool -help
usage: schemaTool
 -alterCatalog <arg>               Alter a catalog, requires
                                   --catalogLocation and/or
                                   --catalogDescription parameter as
well
 -catalogDescription <arg>         Description of new catalog
 -catalogLocation <arg>            Location of new catalog, required
when
                                   adding a catalog
 -createCatalog <arg>              Create a catalog, requires
                                   --catalogLocation parameter as well
 -createLogsTable <arg>            Create table for Hive
                                   warehouse/compute logs
 -createUser                       Create the Hive user, set hiveUser to
                                   the db admin user and the hive
                                   password to the db admin password
with
                                   this
 -dbOpts <databaseOpts>            Backend DB specific options
 -dbType <databaseType>            Metastore database type
 -driver <driver>                  driver name for connection
 -dropAllDatabases                 Drop all Hive databases (with
                                   CASCADE). This will remove all
managed
                                   data!
 -dryRun                           list SQL scripts (no execute)
 -fromCatalog <arg>                Catalog a moving database or table is
                                   coming from.  This is required if you
                                   are moving a database or table.
 -fromDatabase <arg>               Database a moving table is coming
                                   from.  This is required if you are
                                   moving a table.
 -help                             print this message
 -hiveDb <arg>                     Hive database (for use with
```

```
                                        createUser)
   -hivePassword <arg>                  Hive password (for use with
                                        createUser)
   -hiveUser <arg>                      Hive user (for use with createUser)
   -ifNotExists                         If passed then it is not an error to
                                        create an existing catalog
   -info                                Show config and schema details
   -initOrUpgradeSchema                 Initialize or upgrade schema to
latest
                                        version
   -initSchema                          Schema initialization
   -initSchemaTo <initTo>               Schema initialization to a version
   -mergeCatalog <arg>                  Merge databases from a catalog into
                                        other, Argument is the source catalog
                                        name Requires --toCatalog to indicate
                                        the destination catalog
   -metaDbType <metaDatabaseType>       Used only if upgrading the system
                                        catalog for hive
   -moveDatabase <arg>                  Move a database between catalogs.
                                        Argument is the database name.
                                        Requires --fromCatalog and --
toCatalog
                                        parameters as well
   -moveTable <arg>                     Move a table to a different database.
                                        Argument is the table name. Requires
                                        --fromCatalog, --toCatalog,
                                        --fromDatabase, and --toDatabase
                                        parameters as well.
   -passWord <password>                 Override config file password
   -retentionPeriod <arg>               Specify logs table retention period
   -servers <serverList>                a comma-separated list of servers
used
                                        in location validation in the format
                                        of scheme://authority (e.g.
                                        hdfs://localhost:8000)
   -toCatalog <arg>                     Catalog a moving database or table is
                                        going to.  This is required if you
are
                                        moving a database or table.
   -toDatabase <arg>                    Database a moving table is going to.
                                        This is required if you are moving a
                                        table.
   -upgradeSchema                       Schema upgrade
   -upgradeSchemaFrom <upgradeFrom>     Schema upgrade from a version
   -url <url>                           connection url to the database
   -userName <user>                     Override config file user name
   -validate                            Validate the database
   -verbose                             only print SQL statements
   -yes                                 Don't ask for confirmation when using
                                        -dropAllDatabases.
```

Si te da algún tipo de error, fuerza a indicar también el -dbType: `schematool -help -dbType derby` *(Parece un bug de la nueva CLI en esta versión reciente)*

12. Por tanto, debemos ejecutar el siguiente comando `schematool` como paso de inicialización. Para ello usaríamos el siguiente comando:

```
schematool -dbType <db type> -initSchema
```

donde `db type` puede ser: `derby|mysql|postgress|oracle|mssql`

13. Debemos indicar al **metastore** el tipo de schema de base de datos a usar. Nosotros usaremos la opción recomendada en la documentación oficial. Crearemos el metastore en el directorio $HIVE_HOME para tenerlo localizado.

```
cd $HIVE_HOME
schematool -dbType derby -initSchema
```

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hadoop-3.4.1/hive-4.0.1/lib/log4j-slf4j-impl-2.18.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop-3.4.1/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Initializing the schema to: 4.0.0
Metastore connection URL:  jdbc:derby:;databaseName=metastore_db;create=true
Metastore connection Driver :    org.apache.derby.jdbc.EmbeddedDriver
Metastore connection User:   APP
Starting metastore schema initialization to 4.0.0
Initialization script hive-schema-4.0.0.derby.sql
Initialization script completed
```

14. Si quieres conectar otro tipo de Base de Datos de las diferentes opciones, sigue las instrucciones de este recurso

15. Es **muy importante** que inicialices `hiveserver2` en el mismo directorio que has generado el esquema del metastore (observa que se ha creado un directorio llamado **`metastore_db`**), si no te dará error al levantar el servicio. En nuestro caso `cd $HIVE_HOME`. Inicializamos `hiveserver2`

```
hiveserver2
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hadoop-3.4.1/hive-4.0.1/lib/log4j-slf4j-impl-2.18.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop-3.4.1/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
SLF4J: Class path contains multiple SLF4J bindings.
```

```
 8  SLF4J: Found binding in [jar:file:/opt/hadoop-3.4.1/hive-4.0.1/lib/log4j-
slf4j-impl-2.18.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
 9  SLF4J: Found binding in [jar:file:/opt/hadoop-
3.4.1/share/hadoop/common/lib/slf4j-reload4j-
1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
10  SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an
explanation.
11  SLF4J: Actual binding is of type
[org.apache.logging.slf4j.Log4jLoggerFactory]
12  2025-01-07 18:06:57: Starting HiveServer2
13  SLF4J: Class path contains multiple SLF4J bindings.
14  SLF4J: Found binding in [jar:file:/opt/hadoop-3.4.1/hive-4.0.1/lib/log4j-
slf4j-impl-2.18.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
15  SLF4J: Found binding in [jar:file:/opt/hadoop-
3.4.1/share/hadoop/common/lib/slf4j-reload4j-
1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
16  SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an
explanation.
17  SLF4J: Actual binding is of type
[org.apache.logging.slf4j.Log4jLoggerFactory]
18  Hive Session ID = a5de2539-993d-4b84-a6ef-42338e7c49fc
```

16. Comprobamos que está levantado el puerto `10000`

```
1  netstat -putona | grep 10000
2  (Not all processes could be identified, non-owned process info
3   will not be shown, you would have to be root to see it all.)
4  tcp6       0      0 :::10000                :::*                    LISTEN
3232/java           off (0.00/0/0)
5  hadoop@master:~$
```

17. Si no se encuentra el puerto levantado, observa el log en `/tmp/{$user}/hive.log`, en mi caso `/tmp/hadoop/hive.log`
18. Ahora, ya tenemos el servicio levantado para la conexión en el puerto `10000` y la webUI de Hive en el `10002`. Y para conectarnos entramos a `beeline` y tendríamos que indicar la siguiente conexión

```
1  beeline
2  beeline> !connect jdbc:hive2://localhost:10000
```

En mi caso, en lugar de localhost: `cluster-bda` ya configurado en `core-site.xml`

```
1  beeline
2  beeline> !connect jdbc:hive2://cluster-bda:10000
```

19. A partir de esta configuración, ya podemos conectarnos al servidor indicando el usuario con el que nos queremos conectar. (Evidentemente, debemos ser superusuarios). Nos podemos conectar de 2 formas: *(Recuerda reiniciar hadoop si has cambiado la configuración)*. **Nosotros usaremos la configuración Proxy Users**

```
beeline -u jdbc:hive2://cluster-bda:10000/ -n hadoop
show databases;
```

```
hadoop@master:~/hadoop-3.4.1/hive-4.0.1/conf$ beeline -u jdbc:hive2://cluster-bda:10000/ -n hadoop
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hadoop-3.4.1/hive-4.0.1/lib/log4j-slf4j-impl-2.18.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop-3.4.1/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hadoop-3.4.1/hive-4.0.1/lib/log4j-slf4j-impl-2.18.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop-3.4.1/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Connecting to jdbc:hive2://cluster-bda:10000/
Connected to: Apache Hive (version 4.0.1)
Driver: Hive JDBC (version 4.0.1)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 4.0.1 by Apache Hive
0: jdbc:hive2://cluster-bda:10000/> show databases;
INFO  : Compiling command(queryId=hadoop_20250107182855_5cf05d45-95df-425e-83bf-cffb88d61315): show databases
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Created Hive schema: Schema(fieldSchemas:[FieldSchema(name:database_name, type:string, comment:from deserializer)], properties:null)
INFO  : Completed compiling command(queryId=hadoop_20250107182855_5cf05d45-95df-425e-83bf-cffb88d61315); Time taken: 1.602 seconds
INFO  : Concurrency mode is disabled, not creating a lock manager
INFO  : Executing command(queryId=hadoop_20250107182855_5cf05d45-95df-425e-83bf-cffb88d61315): show databases
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=hadoop_20250107182855_5cf05d45-95df-425e-83bf-cffb88d61315); Time taken: 0.155 seconds
+----------------+
| database_name  |
+----------------+
| default        |
+----------------+
1 row selected (2.618 seconds)
```

20. Ya podemos acceder a WebUI para monitorizar ***HiveServer2*** mediante la url `http://cluster-bda:10002/`

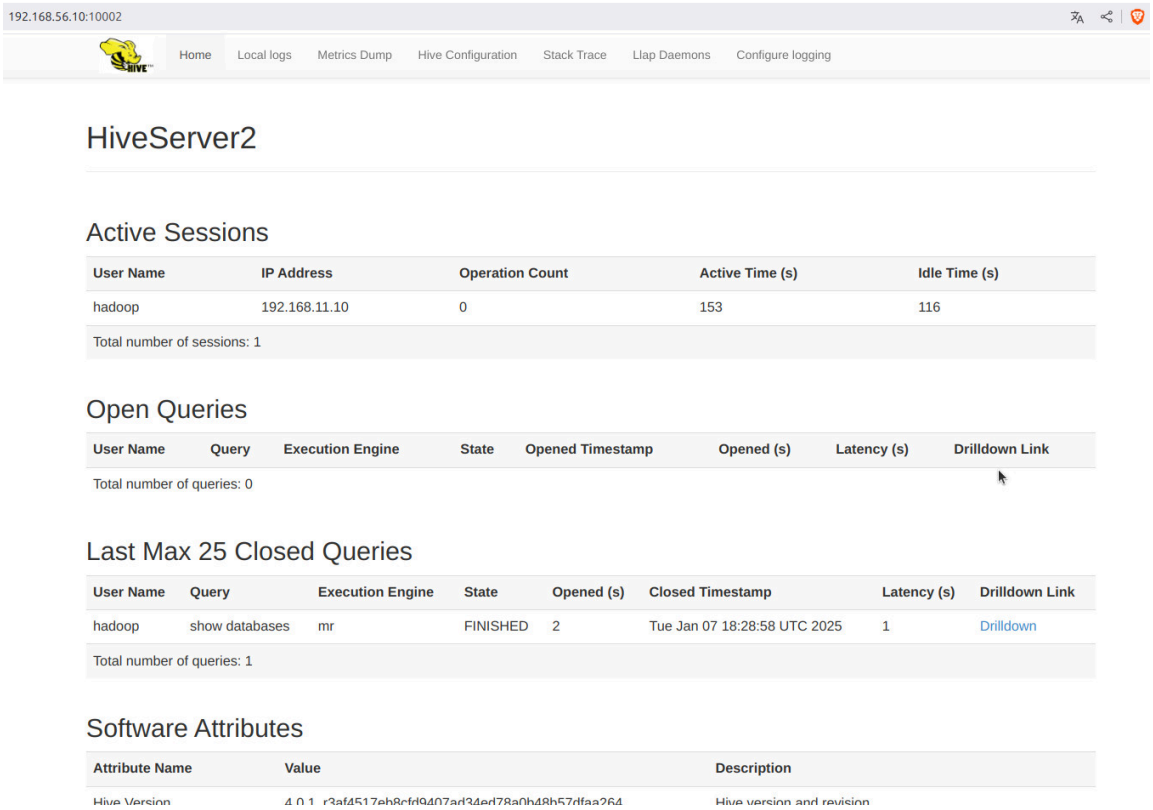

Figura 5.5_Hive: WebUI HiveServer2. (Fuente: Propia)

21. Podemos ver todas las opciones del CLI `beeline` ejecutando el comando `!help`

```
 0: jdbc:hive2://cluster-bda:10000/> !help
 !addlocaldriverjar  Add driver jar file in the beeline client side.
 !addlocaldrivername Add driver name that needs to be supported in the
beeline
                     client side.
 !all                Execute the specified SQL against all the current
connections
 !autocommit         Set autocommit mode on or off
 !batch              Start or execute a batch of statements
 !brief              Set verbose mode off
 !call               Execute a callable statement
 !close              Close the current connection to the database
 !closeall           Close all current open connections
 !columns            List all the columns for the specified table
 !commit             Commit the current transaction (if autocommit is off)
 !connect            Open a new connection to the database.
 !dbinfo             Give metadata information about the database
 !delimiter          Sets the query delimiter, defaults to ;
 !describe           Describe a table
 !dropall            Drop all tables in the current database
 !exportedkeys       List all the exported keys for the specified table
 !go                 Select the current connection
```

```
  !help                Print a summary of command usage
  !history             Display the command history
  !importedkeys        List all the imported keys for the specified table
  !indexes             List all the indexes for the specified table
  !isolation           Set the transaction isolation for this connection
  !list                List the current connections
  !manual              Display the BeeLine manual
  !metadata            Obtain metadata information
  !nativesql           Show the native SQL for the specified statement
  !nullemptystring     Set to true to get historic behavior of printing null
as
                       empty string. Default is false.
  !outputformat        Set the output format for displaying results

(table,vertical,csv2,dsv,tsv2,xmlattrs,xmlelements,json,jsonfile
                       and deprecated formats(csv, tsv))
  !primarykeys         List all the primary keys for the specified table
  !procedures          List all the procedures
  !properties          Connect to the database specified in the properties
file(s)
  !quit                Exits the program
  !reconnect           Reconnect to the database
  !record              Record all output to the specified file
  !rehash              Fetch table and column names for command completion
  !rollback            Roll back the current transaction (if autocommit is
off)
  !run                 Run a script from the specified file
  !save                Save the current variabes and aliases
  !scan                Scan for installed JDBC drivers
  !script              Start saving a script to a file
  !set                 Set a beeline variable
  !sh                  Execute a shell command
  !sql                 Execute a SQL command
  !tables              List all the tables in the database
  !typeinfo            Display the type map for the current connection
  !verbose             Set verbose mode on

 Comments, bug reports, and patches go to ???
 0: jdbc:hive2://cluster-bda:10000>
```

22. No quería terminar la configuración sin comentar que Hive tiene la posibilidad de usarse con un cliente en Python. Observa cómo en la documentación oficial

Modificando datos

> **Info**
>
> **ACID Transactions**: Conjunto de propiedades de transacciones de bases de datos destinadas a garantizar la validez de los datos a pesar de errores, fallas de energía y otros contratiempos.
>
> ***ACID**(**A**tomicity, **C**onsistency, **I**solation and **D**urability: Atomicidad, Consistencia, Aislamiento y Durabilidad)*
>
> **Recuerda tener correctamente configuraado HIVE para realizar estas consultas en todos los casos de uso que lo necesites**

HDFS no se diseñó pensando en las modificaciones de archivos, con lo que los cambios resultantes de las inserciones, modificaciones y borrados se almacenan en **archivos delta**. Por cada transacción, se crea un conjunto de archivo delta que altera la tabla (o partición). Los ficheros delta se fusionan periódicamente con los ficheros base de las tablas mediante jobs MapReduce que el metastore ejecuta en background.

Para poder modificar o borrar los datos, Hive necesita trabajar en un contexto transaccional, por tanto, tenemos que habilitar **ACID Transactions** para admitir consultas transaccionales, por lo que necesitamos activar las siguientes variables:

- hive.support.concurrency=true
- hive.exec.dynamic.partition.mode=nonstrict;
- hive.txn.manager=org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;

También hay que realizar otras modificaciones para la correcta ejecución de Hive si tienes un cluster y no un nodo pseudo-distribuido.

- hive.compactor.initiator.on=true
- hive.compactor.worker.threads = número positivo

Para ello, entra en `hive-site.xml` y cambia las configuraciones *(No olvides reiniciar el servicio después)*:

```
   <property>
     <name>hive.support.concurrency</name>
     <value>true</value>
     <description>
       Whether Hive supports concurrency control or not.
       A ZooKeeper instance must be up and running when using zookeeper
Hive lock manager
     </description>
   </property>

   <property>
     <name>hive.exec.dynamic.partition.mode</name>
     <value>nonstrict</value>
```

```
      <description>
        In strict mode, the user must specify at least one static partition
        in case the user accidentally overwrites all partitions.
        In nonstrict mode all partitions are allowed to be dynamic.
      </description>
    </property>

    <property>
      <name>hive.txn.manager</name>
      <value>org.apache.hadoop.hive.ql.lockmgr.DbTxnManager</value>
      <description>
        Set to org.apache.hadoop.hive.ql.lockmgr.DbTxnManager as part of
turning on Hive
        transactions, which also requires appropriate settings for
hive.compactor.initiator.on,
        hive.compactor.worker.threads, hive.support.concurrency (true),
        and hive.exec.dynamic.partition.mode (nonstrict).
        The default DummyTxnManager replicates pre-Hive-0.13 behavior and
provides
        no transactions.
      </description>
    </property>

    <property>
      <name>hive.compactor.initiator.on</name>
      <value>true</value>
      <description>
        Whether to run the initiator and cleaner threads on this metastore
instance or not.
        Set this to true on one instance of the Thrift metastore service as
part of turning
        on Hive transactions. For a complete list of parameters required
for turning on
        transactions, see hive.txn.manager.
      </description>
    </property>

    <property>
      <name>hive.compactor.worker.threads</name>
      <value>12</value>
      <description>
        How many compactor worker threads to run on this metastore
instance. Set this to a
        positive number on one or more instances of the Thrift metastore
service as part of
        turning on Hive transactions. For a complete list of parameters
required for turning
        on transactions, see hive.txn.manager.
        Worker threads spawn MapReduce jobs to do compactions. They do not
do the compactions
        themselves. Increasing the number of worker threads will decrease
the time it takes
        tables or partitions to be compacted once they are determined to
need compaction.
        It will also increase the background load on the Hadoop cluster as
more MapReduce jobs
        will be running in the background.
```

```
57      </description>
58    </property>
```

Una vez configurado, ya podemos crear las tablas con el **formato ORC(*Optimized Row Columnar*)** y organizar los datos mediante **buckets** (los explicaremos más adelante).

> **File Formats**
>
> **File Formats**: Recuerda que hay varios tipos de formatos diferentes que soporta Hive:
>
> - Text files
> - Sequence File (Para almacenar formato de imagen y/o binario)
> - Avro Files
> - ORC Files
> - Parquet
> - Compressed Data Storage
> - LZO Compression

Ahora, ya podemos crear tablas para que soporten operaciones transaccionales.

Ejemplos

Vamos a realizar algunos ejemplos de uso con Hive para familiarizarnos con la herramienta. Recuerda que puedes consultar todo en el Manual oficial del lenguaje HQL.

Los datos estarán en la carpeta `/opt/hive/` y en `/user/hive/warehouse` en HDFS

Ejemplo1 - Empezando con Hive

En primer lugar, vamos a crear una Base de datos para realizar el ejemplo. Si no lo hacemos, usaremos la Base de Datos `default`. *Si accedes con ProxyUser, recuerda indicar un usuario que tenga permisos en la carpeta donde almacenamos el Warehouse en HDFS*

Creación de la Base de Datos

```
1  /*CREATE DATABASE [IF NOT EXISTS] db_ejemplo1;*/
2  CREATE DATABASE db_ejemplo1;
3  use db_ejemplo1;
```

Sintaxis creación de tablas

La sintaxis de creación de tablas en HQL es:

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name
[(col_name data_type [column_constraint] [COMMENT col_comment], ...)]
[PARTITIONED BY (col_name data_type [COMMENT 'col_comment'], ...)]
[CLUSTERED BY (col_name, col_name,.......]
[COMMENT table_comment]
[ROW FORMAT row_format]
[FIELDS TERMINATED BY char]
[LINES TERMINATED BY char]
[LOCATION 'hdfs_path']
[STORED AS file_format]
```

Info

Tablas Temporales: Las tablas temporales de Hive son similares a las tablas temporales que existen en SQL Server o cualquier base de datos RDBMS. Como sugiere el nombre, estas tablas se crean temporalmente dentro de una sesión activa.

Por lo general, las tablas temporales se crean en tiempo de ejecución para almacenar los datos intermedios que se utilizan para realizar más procesamiento de datos. Una vez que se realiza el procesamiento, puede descartar explícitamente la tabla temporal o la finalización de la sesión eliminará estas tablas.

Para crearlas, recuerda añadir la palabra reservada `TEMPORARY` de la siguiente forma: `CREATE TEMPORARY TABLE`

La creación de tabla tiene un serie de clausulas optativas. Las más usuales son:

- `IF NOT EXISTS` – Puedes usar IF NOT EXISTS para evitar el error en caso de que la tabla ya exista
- `EXTERNAL` – Usado para crear una tabla externa
- `TEMPORARY` – Usado para crear una tabla temporal.
- `ROW FORMAT` – Especifica el formato de una fila.
- `FIELDS TERMINATED BY` – Por defecto Hive usa el campo separador `^A`. Para cargar un fichero con un separador de campo personalizado como coma, tuebria, tabulador,... usa esta opción.
- `PARTITION BY` – Usado para crear una partición de datos. Se usa para mejorar el rendimiento.
- `CLUSTERED BY` – Dividir los datos en un número específico de cubos.
- `LOCATION` – Tu puedes especificar la localización personalizada donde almacenar los datos en HDFS.

Vamos a crear la siguiente tabla

```
CREATE TABLE u_data (
  userid INT,
  movieid INT,
  rating INT,
  unixtime STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;
```

```
0: jdbc:hive2://cluster-bda:10000/> describe u_data;
INFO  : Compiling command(queryId=hadoop_20250107184534_b4e8fdf7-1b8d-4e7c-a50e-7831f7403406): describe u_data
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Created Hive schema: Schema(fieldSchemas:[FieldSchema(name:col_name, type:string, comment:from deserializer), FieldSchema(name:data_type, type:string, comment:from deserializer), FieldSchema(name:comment, type:string, comment:from deserializer)], properties:null)
INFO  : Completed compiling command(queryId=hadoop_20250107184534_b4e8fdf7-1b8d-4e7c-a50e-7831f7403406); Time taken: 0.352 seconds
INFO  : Operation DESCTABLE obtained 0 locks
INFO  : Executing command(queryId=hadoop_20250107184534_b4e8fdf7-1b8d-4e7c-a50e-7831f7403406): describe u_data
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=hadoop_20250107184534_b4e8fdf7-1b8d-4e7c-a50e-7831f7403406); Time taken: 0.099 seconds
+-----------+------------+----------+
| col_name  | data_type  | comment  |
+-----------+------------+----------+
| userid    | int        |          |
| movieid   | int        |          |
| rating    | int        |          |
| unixtime  | string     |          |
+-----------+------------+----------+
4 rows selected (0.738 seconds)
```

Carga de Datos

Directorio datos de origen

Vamos a crear un directorio dentro de $HADOOP_HOME llamado `hive`, que usaremos para guardar todos los datos fuente necesarios para realización de los ejemplos

```
mkdir -p $HADOOP_HOME/hive
```

Debes tener en cuenta que `beeline`, independientemente de donde sea llamada en la terminal(recuerda que la hemos incluido en el path), su acceso al sistema de ficheros local es el directorio $HIVE_HOME.

Luego, descargamos los archivos de datos de *MovieLens 100k* en la página de conjuntos de datos de GroupLens (el cuál tiene un archivo README.txt y un índice de archivos descomprimidos):

```
wget https://files.grouplens.org/datasets/movielens/ml-100k.zip
unzip ml-100k.zip
```

- Carga de datos desde el sistema de ficheros. **Hive toma como raiz del path de la variable de entorno $HIVE_HOME**.

```
LOAD DATA LOCAL INPATH '<path>/u.data'
OVERWRITE INTO TABLE u_data;
```

En nuestro caso:

```
LOAD DATA LOCAL INPATH '../hive/ml-100k/u.data'
OVERWRITE INTO TABLE u_data;
```

Ahora puedes ver los datos almacenados en HDFS

```
hdfs dfs -ls /user/hive/warehouse/db_ejemplo1.db/u_data
```

Contamos las filas de la tabla

```
SELECT COUNT(*) FROM u_data;
```

Mostramos las filas de la tabla

```
SELECT * FROM u_data;
```

Creando a partir de otra tabla

- Crear una tabla usando `LIKE` de una tabla ya existente.

```
CREATE TABLE u_data_2 LIKE u_data;
```

- Crear una tabla desde los resultados de una sentencia

```
CREATE TABLE u_data_rating_5 AS SELECT userid, movieid, rating, unixtime
FROM u_data WHERE rating = 5;
```

Contamos el número de filas:

```
SELECT COUNT(*) FROM u_data_rating_5;
```

Mostramos las filas de la tabla

```
SELECT * FROM u_data_rating_5;
```

Para finalizar, borramos la Base de datos (hay que borrar antes todas las tablas existentes)

Borrado de Base de Datos

```
DROP TABLE u_data;
DROP TABLE u_data_2;
DROP TABLE u_data_rating_5;
DROP DATABASE db_ejemplo1;
```

Acceder a Yarn WebUI y observa que se han ejecutado Jobs correspondientes de las consultas realizadas.

Ejemplo 2 - Uso de Hive con Logs Apache

Vamos a implementar una tabla con **Regex** con capacidad para recoger los logs de Apache. Para ello vamos a usar como datos de origen los recogidos en sus trace de "The Internet Traffic Archive". En concreto 2 meses de logs HTTP del servidor www de la NASA.

```
wget ftp://ita.ee.lbl.gov/traces/NASA_access_log_Jul95.gz
```

Descomprimimos el fichero

```
gzip -d NASA_access_log_Jul95.gz
```

Creando la Base de Datos

```
/*CREATE DATABASE [IF NOT EXISTS] db_ejemplo2;*/
CREATE DATABASE db_ejemplo2;
use db_ejemplo2;
```

Creamos la tabla para almacenar los logs de apache

```
 CREATE TABLE apache_logs(
   host STRING COMMENT 'Host',
   identity STRING COMMENT 'User Identity',
   usuario STRING COMMENT 'User identifier',
   tiempo STRING COMMENT 'Date time of access',
   request STRING COMMENT 'Http request',
   status STRING COMMENT 'Http status',
   size STRING COMMENT 'Http response size',
   referer STRING COMMENT 'Referrer url',
```

```
    agent STRING COMMENT 'Web client agent')
  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
  WITH SERDEPROPERTIES (
    "input.regex" = "([^ ]*) ([^ ]*) ([^ ]*) (-|\\[[^\\]]*\\]) ([^ \"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*)(?: ([^ \"]*|\"[^\"]*\") ([^ \"]*|\"[^\"]*\"))?",
    "output.format.string" = "%1$s %2$s %3$s %4$s %5$s %6$s %7$s %8$s %9$s"
  )
  STORED AS TEXTFILE;
```

Cargando los datos

```
LOAD DATA LOCAL INPATH '../hive/NASA_access_log_Jul95'
OVERWRITE INTO TABLE apache_logs;
```

Si es desde HDFS, sería:

```
LOAD DATA INPATH '/bda/hive/ejemplo2/NASA_access_log_Jul95'
OVERWRITE INTO TABLE apache_logs;
```

Con este ejemplo acabamos de ver como podemos hacer una operación de **ETL** sobre los datos de un servidor. Aquí podemos filtrar, cambiar, recoger y/o obviar los datos que nos demande nuestra situación real, obteniendo conocimientos de ellos para obtener información posterior en base a ellos.

Algunas consultas que podemos hacer de los datos de la tabla generada:

Select con sólo 10 registros

```
select host, request, size from apache_logs limit 10;
```

Media de tamaño de los logs de acceso

```
select avg(size) from apache_logs;
```

Encuentra el histograma del tamaño del request

```
SELECT inline(histogram_numeric(CAST(size AS INT), 10)) FROM apache_logs;
```

Partitions

Como ya hemos visto, para mejorar el rendimiento de Hive, podemos hacer uso de las particiones.

En este caso, si hemos visto que hay cerca de 1.900.000 registros en un sólo mes. Si tuvieramos todos los logs de todos los meses, sería una tabla muy grande. En este caso

podríamos hacer uso de las particiones.

Recordemos que la sintaxis es:

```
ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION partition_spec
[LOCATION 'location'][, PARTITION partition_spec [LOCATION 'location'], ...];

partition_spec:
  : (partition_column = partition_col_value, partition_column =
partition_col_value, ...)
```

Entraremos a explicar las particiones en detalle en el Ejemplo 6

Ejemplo 3 - Usando DML

Tomando como ejemplo la Base de datos del primer ejemplo, vamos a realizar una serie de sentencias que nos ayude a practicar son el lenguaje `HQL DML`

Creando las Bases de Datos

```
CREATE DATABASE db_ejemplo3;
use db_ejemplo3;
```

```
CREATE TABLE u_data (
  userid INT,
  movieid INT,
  rating INT,
  unixtime STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;
```

Carga de Datos

```
wget https://files.grouplens.org/datasets/movielens/ml-100k.zip
unzip ml-100k.zip
```

En esta ocasión vamos a cargar los datos desde HDFS (terminal)

```
hdfs dfs -mkdir -p /bda/hive/ejemplo3
hdfs dfs -copyFromLocal u.data /bda/hive/ejemplo3/
```

```
LOAD DATA INPATH '<path>/u.data'
OVERWRITE INTO TABLE u_data;
```

En nuestro caso (desde beeline):

```
LOAD DATA INPATH '/bda/hive/ejemplo3/u.data'
OVERWRITE INTO TABLE u_data;
```

Insertando datos

```
INSERT INTO TABLE u_data VALUES (488,706,5,891294474);
```

```
 NFO  : Semantic Analysis Completed (retrial = false)
 INFO  : Created Hive schema: Schema(fieldSchemas:[FieldSchema(name:_c0,
type:bigint, comment:null)], properties:null)
 INFO  : Completed compiling
command(queryId=hadoop_20250107192343_5765bd30-81c5-4097-acc7-8615c526ddd8);
Time taken: 0.231 seconds
 INFO  : Operation QUERY obtained 0 locks
 INFO  : Executing command(queryId=hadoop_20250107192343_5765bd30-81c5-
4097-acc7-8615c526ddd8): SELECT COUNT(*) FROM u_data
 ....
 ....
 +---------+
 |   _c0   |
 +---------+
 | 100001  |
 +---------+
 1 row selected (20,514 seconds)
```

Si necesitamos insertar datos en múltiples tablas a la vez lo haremos mediante el comando `from-insert`, ya que ofrece un mejor rendimiento al sólo necesitar un escaneado de los datos:

```
FROM fuente
INSERT OVERWRITE TABLE destino1
  SELECT col1, col2
INSERT OVERWRITE TABLE destino2
  SELECT col1, col3
```

Por ejemplo, vamos a crear un par de tablas con la misma estructura de datos, pero para almacenar las valoraciones por ratio de puntuacion:

```
CREATE TABLE u_data_ratio5 LIKE u_data;
CREATE TABLE u_data_ratio4 LIKE u_data;
```

```
FROM u_data
INSERT OVERWRITE TABLE u_data_ratio5
  SELECT userid, movieid, rating, unixtime WHERE rating = 5
INSERT OVERWRITE TABLE u_data_ratio4
  SELECT userid, movieid, rating, unixtime WHERE rating = 4;
```

Vamos a crear la tabla anterior permitiendo transacciones.

```
CREATE TABLE u_data2 (
  userid INT,
  movieid INT,
  rating INT,
  unixtime STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS ORC
TBLPROPERTIES ('transactional' = 'true');
```

Puedes observar que está habilitada la propiedad con el comando `DESCRIBE FORMATTED` y la propiedad de la tabla `transactional`:

```
0: jdbc:hive2://cluster-bda:10000> DESCRIBE FORMATTED u_data2;
INFO  : Compiling command(queryId=hadoop_20250107193841_fcca1966-c1d5-4b98-9188-6f27d9c65c58): DESCRIBE FORMATTED u_data2
INFO  : Semantic Analysis Completed (retrial = false)
INFO  : Created Hive schema: Schema(fieldSchemas:[FieldSchema(name:col_name, type:string, comment:from deserializer), FieldSchema(name:data_type, type:string, comment:from deserializer), FieldSchema(name:comment, type:string, comment:from deserializer)], properties:null)
INFO  : Completed compiling command(queryId=hadoop_20250107193841_fcca1966-c1d5-4b98-9188-6f27d9c65c58); Time taken: 0.078 seconds
INFO  : Operation DESCTABLE obtained 1 locks
INFO  : Executing command(queryId=hadoop_20250107193841_fcca1966-c1d5-4b98-9188-6f27d9c65c58): DESCRIBE FORMATTED u_data2
INFO  : Starting task [Stage-0:DDL] in serial mode
INFO  : Completed executing command(queryId=hadoop_20250107193841_fcca1966-c1d5-4b98-9188-6f27d9c65c58); Time taken: 0.03 seconds
+-------------------------------+----------------------------------------------------+----------+
|           col_name            |                     data_type                      |  comment |
+-------------------------------+----------------------------------------------------+----------+
| userid                        | int                                                |          |
| movieid                       | int                                                |          |
| rating                        | int                                                |          |
| unixtime                      | string                                             |          |
|                               | NULL                                               | NULL     |
| # Detailed Table Information  | NULL                                               | NULL     |
| Database:                     | db_ejemplo3                                        | NULL     |
| OwnerType:                    | USER                                               | NULL     |
| Owner:                        | hadoop                                             | NULL     |
```

```
22  | CreateTime:                   | Tue Jan 07 19:38:30 UTC 2025
| NULL                                                |
23  | LastAccessTime:               | UNKNOWN
| NULL                                                |
24  | Retention:                    | 0
| NULL                                                |
25  | Location:                     | hdfs://cluster-
bda:9000/user/hive/warehouse/db_ejemplo3.db/u_data2 | NULL
|
26  | Table Type:                   | MANAGED_TABLE
| NULL                                                |
27  | Table Parameters:             | NULL
| NULL                                                |
28  |                               | COLUMN_STATS_ACCURATE
| {\"BASIC_STATS\":\"true\",\"COLUMN_STATS\":
{\"movieid\":\"true\",\"rating\":\"true\",\"unixtime\":\"true\",\"userid\":\"t
|
29  |                               | bucketing_version
| 2                                                   |
30  |                               | numFiles
| 0                                                   |
31  |                               | numRows
| 0                                                   |
32  |                               | rawDataSize
| 0                                                   |
33  |                               | totalSize
| 0                                                   |
34  |                               | transactional
| true                                                |
35  |                               | transactional_properties
| default                                             |
36  |                               | transient_lastDdlTime
| 1736278710                                          |
37  |                               | NULL
| NULL                                                |
38  | # Storage Information         | NULL
| NULL                                                |
39  | SerDe Library:                | org.apache.hadoop.hive.ql.io.orc.OrcSer
| NULL                                                |
40  | InputFormat:                  |
org.apache.hadoop.hive.ql.io.orc.OrcInputFormat    | NULL
|
41  | OutputFormat:                 |
org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat   | NULL
|
42  | Compressed:                   | No
| NULL                                                |
43  | Num Buckets:                  | -1
| NULL                                                |
44  | Bucket Columns:               | []
| NULL                                                |
45  | Sort Columns:                 | []
| NULL                                                |
46  +-------------------------------+-----------------------------------------
-----+-----------------------------------------------------+
47  33 rows selected (0.371 seconds)
48  0: jdbc:hive2://cluster-bda:10000>
```

```
```

Cargamos datos desde la tabla u_data:

```
INSERT INTO u_data2 SELECT * FROM u_data;
```

Vamos a cambiar el "ratio" de la valoración de la película del registro introducido anteriormente

```
SELECT * FROM u_data2
WHERE userid=488 AND movieid=706;
```

```
+-----------------+-----------------+----------------+-----------------
-+
| u_data2.userid  | u_data2.movieid  | u_data2.rating  | u_data2.unixtime
|
+-----------------+-----------------+----------------+-----------------
-+
| 488             | 706              | 5               | 891294474
|
+-----------------+-----------------+----------------+-----------------
-+
```

```
UPDATE u_data2 SET rating=6 WHERE (userid=488 AND movieid=706);
```

```
+-----------------+-----------------+----------------+-----------------
-+
| u_data2.userid  | u_data2.movieid  | u_data2.rating  | u_data2.unixtime
|
+-----------------+-----------------+----------------+-----------------
-+
| 488             | 706              | 6               | 891294474
|
+-----------------+-----------------+----------------+-----------------
-
```

Vamos a borra este registro que hemos actualizado

```
DELETE FROM u_data2 WHERE userid=488 AND movieid=706;
```

Extrayendo datos insertados

Combinando los comandos de **HQL** y **HDFS** podemos extraer datos a ficheros remotos o locales:

```
# Añadiendo contenido local
beeline -u jdbc:hive2://cluster-bda:10000/ -n hadoop -e "use db_ejemplo3;
select * from u_data2 limit 10" >> prueba1
# Sobrescribiendo contenido local
```

```
beeline -u jdbc:hive2://cluster-bda:10000/ -n hadoop -e "use db_ejemplo3; select * from u_data2" > prueba2
# Añadiendo contenido HDFS
beeline -u jdbc:hive2://cluster-bda:10000/ -n hadoop -e "use db_ejemplo3; select * from u_data2" | hdfs dfs -appendToFile - /bda/hive/ejemplo3/prueba3
# Sobrescribiendo contenido
beeline -u jdbc:hive2://cluster-bda:10000/ -n hadoop -e "use db_ejemplo3; select * from u_data2" | hdfs dfs -put -f - /bda/hive/ejemplo3/prueba4
```

Extrayendo datos en CSV

Extrayendo datos en formato CSV dentro de HDFS

```
beeline -u jdbc:hive2://cluster-bda:10000/ -n hadoop -e "INSERT OVERWRITE DIRECTORY '/bda/hive/ejemplo3/salida.csv' ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' SELECT * FROM db_ejemplo3.u_data"
```

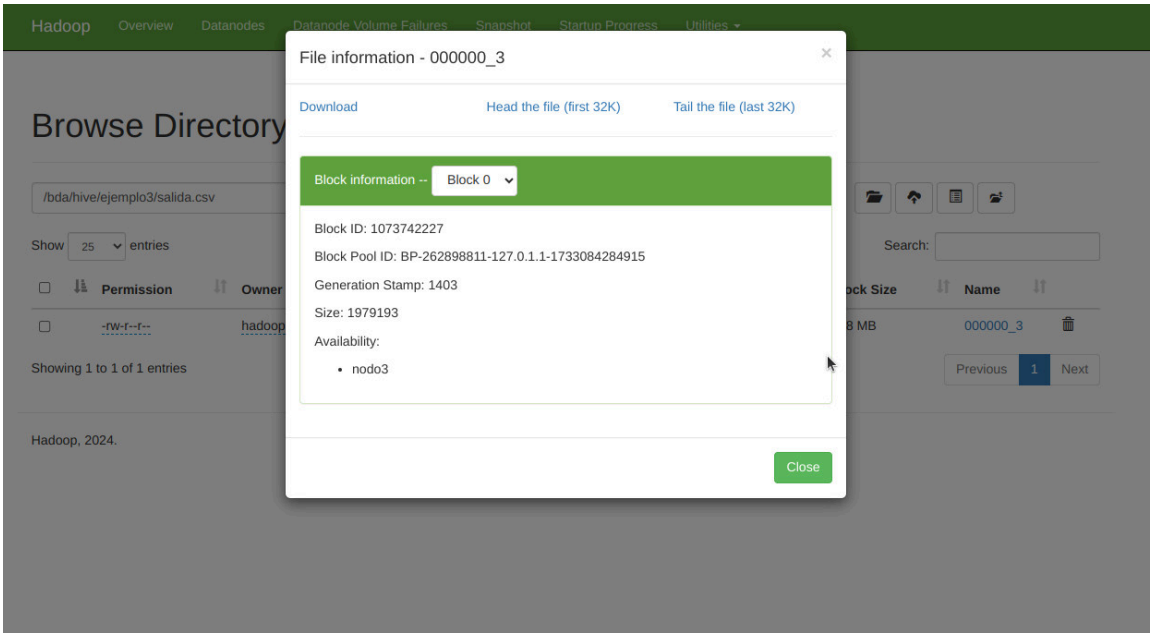

Figura 5.6_Hive: Exportando datos a CSV dentro de HDFS. (Fuente: Propia)

Podemos verlo también mostrandro el contenido desde terminal

```
hadoop@master:/opt/hadoop-3.4.1/hive$ hdfs dfs -cat -more /bda/hive/ejemplo3/salida.csv/000000_3
....
....
449,120,1,879959573
661,762,2,876037121
721,874,3,877137447
821,151,4,874792889
764,596,3,876243046
537,443,3,886031752
618,628,2,891308019
```

```
487,291,3,883445079
113,975,5,875936424
943,391,2,888640291
864,685,4,888891900
750,323,3,879445877
279,64,1,875308510
646,750,3,888528902
654,370,2,887863914
617,582,4,883789294
913,690,3,880824288
660,229,2,891406212
421,498,4,892241344
495,1091,4,888637503
806,421,4,882388897
676,538,4,892685437
721,262,3,877137285
913,209,2,881367150
378,78,3,880056976
880,476,3,880175444
716,204,5,879795543
276,1090,1,874795795
13,225,2,882399156
12,203,3,879959583
488,706,5,891294474
```

Extrayendo datos en formato CSV en Local

Crea un directorio con el nombre indicado (salida_csv en nuestro caso) al estilo HDFS donde exporta a un fichero CSV

```
beeline -u jdbc:hive2://cluster-bda:10000/ -n hadoop -e "INSERT OVERWRITE
LOCAL DIRECTORY '/home/hadoop/salida_csv' ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',' SELECT * FROM db_ejemplo3.u_data"
```

Comprueba la salida haciendo un `cat /home/hadoop/salida_csv/000000_0`

Si generara más de un fichero, puedes unirlos todos: `cat /home/hadoop/salida_csv/* > salida.csv`

Extrayendo datos en formato CSV con cabecera

De forma predeterminada, el beeline CLI genera los datos en un formato tabular; al cambiar el formato a CSV2, Hive beeline devuelve los resultados en formato CSV. Al canalizar esta salida a un archivo CSV, obtendremos un archivo CSV con un encabezado.

```
beeline -u jdbc:hive2://cluster-bda:10000/ -n hadoop --outputformat=csv2 -
e "SELECT * FROM db_ejemplo3.u_data" > salida.csv
```

Compruébalo con `head salida.csv`

Ejemplo 4 - Join

Antes de empezar con el ejemplo, debemos resaltar algunos puntos sobre **join** en Hive:

- Solo se permiten uniones de igualdad en *Join*
- Se pueden unir más de dos tablas en la misma consulta
- Las uniones LEFT, RIGHT, FULL OUTER existen para proporcionar más control sobre la cláusula ON para la que no hay coincidencia
- Las uniones no son conmutativas
- Las uniones son asociativas a la izquierda independientemente de si son uniones LEFT o RIGHT

Vamos a realizar un ejemplo sobre Hive Join

Para ello vamos a tomar como ejemplo la clasica Base de Datos de Tiger/Scott de empleados y departamentos.

Creamos la Base de Datos del ejemplos y las 2 tablas.

```
CREATE DATABASE db_ejemplo4;
use db_ejemplo4;
```

```
 CREATE TABLE emp (
   empno INT,
   ename STRING,
   job STRING,
   mgr INT,
   hiredate DATE,
   sal INT,
   comm INT,
   deptno INT)
 ROW FORMAT DELIMITED
 STORED AS ORC;
```

```
CREATE TABLE dep (
  deptno INT,
  dname STRING,
  loc STRING)
ROW FORMAT DELIMITED
STORED AS ORC;
```

```
 INSERT INTO TABLE dep VALUES (10, 'ACCOUNTING', 'NEW YORK');
 INSERT INTO TABLE dep VALUES (20, 'RESEARCH', 'DALLAS');
 INSERT INTO TABLE dep VALUES (30, 'SALES', 'CHICAGO');
 INSERT INTO TABLE dep VALUES (40, 'OPERATIONS', 'BOSTON');
 INSERT INTO TABLE emp VALUES(7369, 'SMITH', 'CLERK', 7902,'17-12-1980',
800, NULL, 20);
```

```
 6  INSERT INTO TABLE emp VALUES(7499, 'ALLEN', 'SALESMAN', 7698,'20-02-
1981',1600, 300, 30);
 7  INSERT INTO TABLE emp VALUES(7521, 'WARD', 'SALESMAN', 7698,'22-02-
1981',1250, 500, 30);
 8  INSERT INTO TABLE emp VALUES(7566, 'JONES', 'MANAGER', 7839,'02-04-1981',
2975, NULL, 20);
 9  INSERT INTO TABLE emp VALUES(7654, 'MARTIN', 'SALESMAN', 7698,'28-09-
1981',1250, 1400, 30);
10  INSERT INTO TABLE emp VALUES(7698, 'BLAKE', 'MANAGER', 7839,'01-05-1981',
2850, NULL, 30);
11  INSERT INTO TABLE emp VALUES(7782, 'CLARK', 'MANAGER', 7839,'09-06-1981',
2450, NULL, 10);
12  INSERT INTO TABLE emp VALUES(7788, 'SCOTT', 'ANALYST', 7566,'09-12-1982',
3000, NULL, 20);
13  INSERT INTO TABLE emp VALUES(7839, 'KING', 'PRESIDENT', NULL,'17-11-
1981', 5000, NULL, 10);
14  INSERT INTO TABLE emp VALUES(7844, 'TURNER', 'SALESMAN', 7698,'08-09-
1981', 1500, 0, 30);
15  INSERT INTO TABLE emp VALUES(7876, 'ADAMS', 'CLERK', 7788,'12-01-1983',
1100, NULL, 20);
16  INSERT INTO TABLE emp VALUES(7900, 'JAMES', 'CLERK', 7698,'03-12-1981',
950, NULL, 30);
17  INSERT INTO TABLE emp VALUES(7902, 'FORD', 'ANALYST', 7566,'03-12-1981',
3000, NULL, 20);
18  INSERT INTO TABLE emp VALUES(7934, 'MILLER', 'CLERK', 7782,'23-01-1982',
1300, NULL, 10);
19  INSERT INTO TABLE emp VALUES(8000, 'JAIME', 'TEACHER', 7788,'01-09-2004',
1100, NULL, NULL);
```

Inner Join

Si queremos relacionar los datos de ambas tablas, tenemos que hacer un join entre la clave ajena de emp (deptno) y la clave primaria de dep (deptno)

```
1  SELECT e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm,
d.deptno, d.dname, d.loc
2  FROM emp e JOIN dep d
3  ON(e.deptno=d.deptno);
```

La salida muestra los registros comunes presentes en la tabla al verificar la condición `ON`

```
1  +----------+----------+-----------+--------+-------------+--------+-----
----+-----------+-------------+-----------+
2  | e.empno  | e.ename  |   e.job   | e.mgr  | e.hiredate  | e.sal  |
e.comm  | d.deptno  |   d.dname   |   d.loc   |
3  +----------+----------+-----------+--------+-------------+--------+-----
----+-----------+-------------+-----------+
4  | 7499     | ALLEN    | SALESMAN  | 7698   | NULL        | 1600   | 300
| 30        | SALES       | CHICAGO   |
5  | 7521     | WARD     | SALESMAN  | 7698   | NULL        | 1250   | 500
| 30        | SALES       | CHICAGO   |
6  | 7900     | JAMES    | CLERK     | 7698   | NULL        | 950    | NULL
| 30        | SALES       | CHICAGO   |
```

```
 7   | 7902     | FORD     | ANALYST    | 7566   | NULL         | 3000   | NULL
| 20        | RESEARCH    | DALLAS    |
 8   | 7934     | MILLER   | CLERK      | 7782   | NULL         | 1300   | NULL
| 10        | ACCOUNTING  | NEW YORK  |
 9   | 7566     | JONES    | MANAGER    | 7839   | NULL         | 2975   | NULL
| 20        | RESEARCH    | DALLAS    |
10   | 7654     | MARTIN   | SALESMAN   | 7698   | NULL         | 1250   | 1400
| 30        | SALES       | CHICAGO   |
11   | 7698     | BLAKE    | MANAGER    | 7839   | NULL         | 2850   | NULL
| 30        | SALES       | CHICAGO   |
12   | 7782     | CLARK    | MANAGER    | 7839   | NULL         | 2450   | NULL
| 10        | ACCOUNTING  | NEW YORK  |
13   | 7788     | SCOTT    | ANALYST    | 7566   | NULL         | 3000   | NULL
| 20        | RESEARCH    | DALLAS    |
14   | 7839     | KING     | PRESIDENT  | NULL   | NULL         | 5000   | NULL
| 10        | ACCOUNTING  | NEW YORK  |
15   | 7844     | TURNER   | SALESMAN   | 7698   | NULL         | 1500   | 0
| 30        | SALES       | CHICAGO   |
16   | 7876     | ADAMS    | CLERK      | 7788   | NULL         | 1100   | NULL
| 20        | RESEARCH    | DALLAS    |
17   +----------+----------+------------+--------+--------------+--------+-----
----+-----------+-------------+-----------+
```

Left Outer Join

LEFT OUTER JOIN devuelve todas las filas de la tabla de la izquierda aunque no haya coincidencias en la tabla de la derecha.

Si la Cláusula `ON` coincide con cero registros en la tabla de la derecha, las uniones aún devuelven un registro en el resultado con NULL en cada columna de la tabla de la derecha

```
1   SELECT e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm,
d.deptno, d.dname, d.loc
2   FROM emp e LEFT OUTER JOIN dep d
3   ON(e.deptno=d.deptno);
```

```
 1   +----------+----------+------------+--------+--------------+--------+-----
----+-----------+-------------+-----------+
 2   | e.empno  | e.ename  |   e.job    | e.mgr  | e.hiredate   | e.sal  |
e.comm  | d.deptno  |   d.dname   |   d.loc   |
 3   +----------+----------+------------+--------+--------------+--------+-----
----+-----------+-------------+-----------+
 4   | 7499     | ALLEN    | SALESMAN   | 7698   | NULL         | 1600   | 300
| 30        | SALES       | CHICAGO   |
 5   | 7521     | WARD     | SALESMAN   | 7698   | NULL         | 1250   | 500
| 30        | SALES       | CHICAGO   |
 6   | 7900     | JAMES    | CLERK      | 7698   | NULL         | 950    | NULL
| 30        | SALES       | CHICAGO   |
 7   | 7902     | FORD     | ANALYST    | 7566   | NULL         | 3000   | NULL
| 20        | RESEARCH    | DALLAS    |
 8   | 7934     | MILLER   | CLERK      | 7782   | NULL         | 1300   | NULL
| 10        | ACCOUNTING  | NEW YORK  |
```

```
 9   | 8000      | JAIME     | TEACHER    | 7788   | NULL         | 1100   | NULL
| NULL       | NULL         | NULL       |
10   | 7566      | JONES     | MANAGER    | 7839   | NULL         | 2975   | NULL
| 20         | RESEARCH     | DALLAS     |
11   | 7654      | MARTIN    | SALESMAN   | 7698   | NULL         | 1250   | 1400
| 30         | SALES        | CHICAGO    |
12   | 7698      | BLAKE     | MANAGER    | 7839   | NULL         | 2850   | NULL
| 30         | SALES        | CHICAGO    |
13   | 7782      | CLARK     | MANAGER    | 7839   | NULL         | 2450   | NULL
| 10         | ACCOUNTING   | NEW YORK   |
14   | 7788      | SCOTT     | ANALYST    | 7566   | NULL         | 3000   | NULL
| 20         | RESEARCH     | DALLAS     |
15   | 7839      | KING      | PRESIDENT  | NULL   | NULL         | 5000   | NULL
| 10         | ACCOUNTING   | NEW YORK   |
16   | 7844      | TURNER    | SALESMAN   | 7698   | NULL         | 1500   | 0
| 30         | SALES        | CHICAGO    |
17   | 7876      | ADAMS     | CLERK      | 7788   | NULL         | 1100   | NULL
| 20         | RESEARCH     | DALLAS     |
18   +----------+----------+------------+--------+--------------+--------+-----
----+-----------+-------------+-----------+
```

Right Outer Join

RIGHT OUTER JOIN devuelve todas las filas de la tabla de la derecha aunque no haya coincidencias en la tabla de la izquierda.

Si la Cláusula `ON` coincide con cero registros en la tabla de la izquierda, las uniones aún devuelven un registro en el resultado con NULL en cada columna de la tabla de la izquierda

```
1   SELECT e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm,
d.deptno, d.dname, d.loc
2   FROM emp e RIGHT OUTER JOIN dep d
3   ON(e.deptno=d.deptno);
```

```
1   +----------+----------+------------+--------+--------------+--------+-----
----+-----------+-------------+-----------+
2   | e.empno  | e.ename  |   e.job    | e.mgr  | e.hiredate   | e.sal  |
e.comm  | d.deptno  |   d.dname   |   d.loc   |
3   +----------+----------+------------+--------+--------------+--------+-----
----+-----------+-------------+-----------+
4   | 7934      | MILLER    | CLERK      | 7782   | NULL         | 1300   | NULL
| 10         | ACCOUNTING   | NEW YORK   |
5   | 7782      | CLARK     | MANAGER    | 7839   | NULL         | 2450   | NULL
| 10         | ACCOUNTING   | NEW YORK   |
6   | 7839      | KING      | PRESIDENT  | NULL   | NULL         | 5000   | NULL
| 10         | ACCOUNTING   | NEW YORK   |
7   | 7902      | FORD      | ANALYST    | 7566   | NULL         | 3000   | NULL
| 20         | RESEARCH     | DALLAS     |
8   | 7566      | JONES     | MANAGER    | 7839   | NULL         | 2975   | NULL
| 20         | RESEARCH     | DALLAS     |
9   | 7788      | SCOTT     | ANALYST    | 7566   | NULL         | 3000   | NULL
| 20         | RESEARCH     | DALLAS     |
```

```
| 7876 | ADAMS | CLERK | 7788 | NULL | 1100 | NULL | 20 | RESEARCH | DALLAS |
| 7499 | ALLEN | SALESMAN | 7698 | NULL | 1600 | 300 | 30 | SALES | CHICAGO |
| 7521 | WARD | SALESMAN | 7698 | NULL | 1250 | 500 | 30 | SALES | CHICAGO |
| 7900 | JAMES | CLERK | 7698 | NULL | 950 | NULL | 30 | SALES | CHICAGO |
| 7654 | MARTIN | SALESMAN | 7698 | NULL | 1250 | 1400 | 30 | SALES | CHICAGO |
| 7698 | BLAKE | MANAGER | 7839 | NULL | 2850 | NULL | 30 | SALES | CHICAGO |
| 7844 | TURNER | SALESMAN | 7698 | NULL | 1500 | 0 | 30 | SALES | CHICAGO |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | 40 | OPERATIONS | BOSTON |
+----------+----------+-----------+--------+-------------+--------+-----
----+-----------+-------------+-----------+
```

Full Outer Join

FULL OUTER JOIN devuelve todas los registros comunes presentes en la tabla al verificar la condición `ON`

Pero además devuelve todos los registros de ambas tablas y completa los valores NULL para las columnas que faltan valores coincidentes en ambos lados.

```
SELECT e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm,
d.deptno, d.dname, d.loc
FROM emp e FULL OUTER JOIN dep d
ON(e.deptno=d.deptno);
```

```
+----------+----------+-----------+--------+-------------+--------+-----
----+-----------+-------------+-----------+
| e.empno | e.ename | e.job | e.mgr | e.hiredate | e.sal |
e.comm | d.deptno | d.dname | d.loc |
+----------+----------+-----------+--------+-------------+--------+-----
----+-----------+-------------+-----------+
| 8000 | JAIME | TEACHER | 7788 | NULL | 1100 | NULL
| NULL | NULL | NULL |
| 7782 | CLARK | MANAGER | 7839 | NULL | 2450 | NULL
| 10 | ACCOUNTING | NEW YORK |
| 7839 | KING | PRESIDENT | NULL | NULL | 5000 | NULL
| 10 | ACCOUNTING | NEW YORK |
| 7934 | MILLER | CLERK | 7782 | NULL | 1300 | NULL
| 10 | ACCOUNTING | NEW YORK |
| 7876 | ADAMS | CLERK | 7788 | NULL | 1100 | NULL
| 20 | RESEARCH | DALLAS |
| 7788 | SCOTT | ANALYST | 7566 | NULL | 3000 | NULL
| 20 | RESEARCH | DALLAS |
| 7566 | JONES | MANAGER | 7839 | NULL | 2975 | NULL
| 20 | RESEARCH | DALLAS |
```

```
| 7902 | FORD | ANALYST | 7566 | NULL | 3000 | NULL | 20 | RESEARCH | DALLAS |
| 7698 | BLAKE | MANAGER | 7839 | NULL | 2850 | NULL | 30 | SALES | CHICAGO |
| 7844 | TURNER | SALESMAN | 7698 | NULL | 1500 | 0 | 30 | SALES | CHICAGO |
| 7499 | ALLEN | SALESMAN | 7698 | NULL | 1600 | 300 | 30 | SALES | CHICAGO |
| 7900 | JAMES | CLERK | 7698 | NULL | 950 | NULL | 30 | SALES | CHICAGO |
| 7654 | MARTIN | SALESMAN | 7698 | NULL | 1250 | 1400 | 30 | SALES | CHICAGO |
| 7521 | WARD | SALESMAN | 7698 | NULL | 1250 | 500 | 30 | SALES | CHICAGO |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | 40 | OPERATIONS | BOSTON |
+----------+----------+-----------+--------+-------------+--------+-----
----+-----------+-------------+-----------+
```

Ejemplo 5 - Tabla externa

Hive permite crear tablas de dos tipos:

- **Tabla interna** o gestionada: Hive gestiona la estructura y el almacenamiento de los datos. Para ello, crea los datos en HDFS. Al borrar la tabla de Hive, se borra la información de HDFS.
- **Tabla externa**: Hive define la estructura de los datos en el metastore, pero los datos se mantienen permanentemente en HDFS. **Al borrar la tabla de Hive, no se eliminan los datos de HDFS**. Se emplea cuando compartimos datos almacenados en HDFS entre diferentes herramientas.

Anteriormente hemos usado tablas internas. Vamos a usar la tabla empleados del ejercicio anterior, pero en este caso vamos a definirla como tabla externa.

```
CREATE DATABASE db_ejemplo5;
use db_ejemplo5;
```

```
CREATE EXTERNAL TABLE emp_externa (
  empno INT,
  ename STRING,
  job STRING,
  mgr INT,
  hiredate DATE,
  sal INT,
  comm INT,
  deptno INT)
ROW FORMAT DELIMITED
STORED AS ORC
LOCATION "/bda/hive/ejemplo5/emp_externa";
```

```
INSERT INTO TABLE emp_externa VALUES(7499, 'ALLEN', 'SALESMAN', 7698,'20-02-1981',1600, 300, 30);
INSERT INTO TABLE emp_externa VALUES(7521, 'WARD', 'SALESMAN', 7698,'22-02-1981',1250, 500, 30);
INSERT INTO TABLE emp_externa VALUES(7566, 'JONES', 'MANAGER', 7839,'02-04-1981', 2975, NULL, 20);
INSERT INTO TABLE emp_externa VALUES(7654, 'MARTIN', 'SALESMAN', 7698,'28-09-1981',1250, 1400, 30);
INSERT INTO TABLE emp_externa VALUES(7698, 'BLAKE', 'MANAGER', 7839,'01-05-1981', 2850, NULL, 30);
INSERT INTO TABLE emp_externa VALUES(7782, 'CLARK', 'MANAGER', 7839,'09-06-1981', 2450, NULL, 10);
INSERT INTO TABLE emp_externa VALUES(7788, 'SCOTT', 'ANALYST', 7566,'09-12-1982', 3000, NULL, 20);
INSERT INTO TABLE emp_externa VALUES(7839, 'KING', 'PRESIDENT', NULL,'17-11-1981', 5000, NULL, 10);
INSERT INTO TABLE emp_externa VALUES(7844, 'TURNER', 'SALESMAN', 7698,'08-09-1981', 1500, 0, 30);
INSERT INTO TABLE emp_externa VALUES(7876, 'ADAMS', 'CLERK', 7788,'12-01-1983', 1100, NULL, 20);
INSERT INTO TABLE emp_externa VALUES(7900, 'JAMES', 'CLERK', 7698,'03-12-1981', 950, NULL, 30);
INSERT INTO TABLE emp_externa VALUES(7902, 'FORD', 'ANALYST', 7566,'03-12-1981', 3000, NULL, 20);
INSERT INTO TABLE emp_externa VALUES(7934, 'MILLER', 'CLERK', 7782,'23-01-1982', 1300, NULL, 10);
INSERT INTO TABLE emp_externa VALUES(8000, 'JAIME', 'TEACHER', 7788,'01-09-2004', 1100, NULL, NULL);
```

Observa en HDFS los datos almacenados en la ruta indicada en `LOCATION`

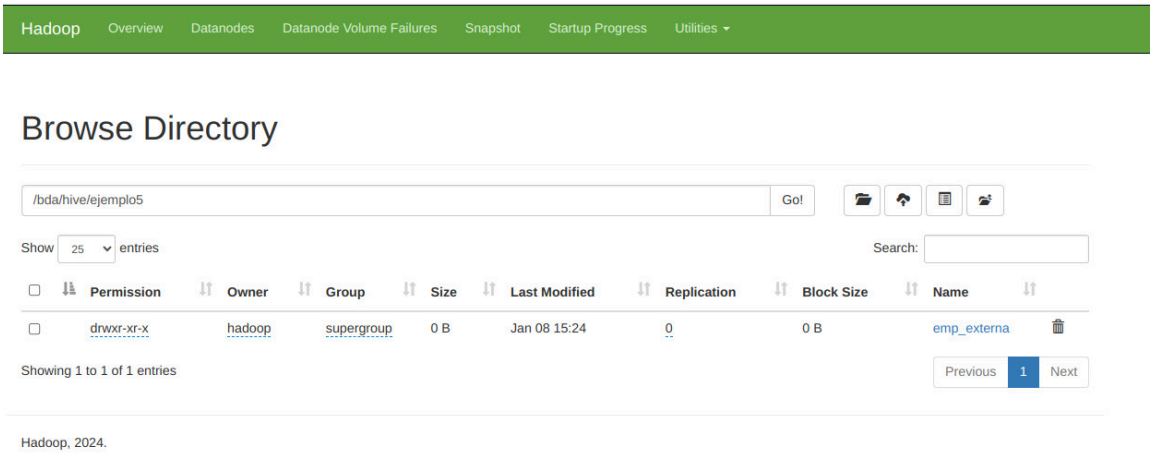

Figura 5.7_Hive: Tabla Externa. (Fuente: Propia)

Borra la tabla en Hive

```
DROP TABLE emp_externa;
```

Vuelve a comprobar la ruta en HDFS y observa que los datos no se han borrado

Ejemplo 6 - Partitions

Recuerda que **Hive Partitions** es una forma de organizar las tablas en particiones dividiendo las tablas en diferentes partes según las claves de partición.

La partición es útil cuando la tabla tiene una o más claves de partición. Las claves de partición son elementos básicos para determinar cómo se almacenan los datos en la tabla.

Vamos a realizar un ejemplo con unos datos obtenidos del IHMD(Instituto para la Métrica y Evaluación de la Salud)

Creamos la Base de datos y la tabla

```
CREATE DATABASE db_ejemplo6;
use db_ejemplo6;
```

```
CREATE TABLE estados (
  delta INT,
  fecha TIMESTAMP,
  fpsid INT,
  state STRING,
  forecastweek STRING,
  value INT,
  modelname STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

Descargamos los datos

```
wget
https://gist.githubusercontent.com/jaimerabasco/cb528c32b4c4092e6a0763d8b6bc25c0/
```

- Carga de datos desde el sistema de ficheros. **Hive toma como raiz del path del fichero $HIVE_HOME**. En nuestro caso:

```
LOAD DATA LOCAL INPATH '../hive/allstates.csv'
OVERWRITE INTO TABLE estados;
```

Particionado dinámico

Podemos establecer que los registros se añadan directamente para que sea HIVE quien crea las particiones teniendo en cuenta los valores de `state`

```
CREATE TABLE estados_part (
  delta INT,
  fecha TIMESTAMP,
  fpsid INT,
```

```
  forecastweek STRING,
  value INT,
  modelname STRING)
 PARTITIONED BY (state STRING)
 ROW FORMAT DELIMITED
 FIELDS TERMINATED BY ',';
```

```
 INSERT OVERWRITE TABLE estados_part PARTITION(state)
 SELECT delta, fecha, fpsid, forecastweek, value, modelname, state FROM
estados;
```

El campo state pasado en Select debe estar al final.

Mostramos las particiones creadas en la nueva tabla

```
 show partitions estados_part;
```

```
 +-----------------------------+
 |          partition          |
 +-----------------------------+
 | state=Alabama               |
 | state=Alaska                |
 | state=Arizona               |
 | state=Arkansas              |
 | state=California            |
 | state=Colorado              |
 | state=Connecticut           |
 | state=Delaware              |
 | state=District of Columbia  |
 | state=Florida               |
 | state=Georgia               |
 | state=Hawaii                |
 | state=Idaho                 |
 | state=Illinois              |
 | state=Indiana               |
 | state=Iowa                  |
 | state=Kansas                |
 | state=Kentucky              |
 | state=Louisiana             |
 | state=Maine                 |
 | state=Maryland              |
 | state=Massachusetts         |
 | state=Michigan              |
 | state=Minnesota             |
 | state=Mississippi           |
 | state=Missouri              |
 | state=Montana               |
 | state=Nebraska              |
 | state=Nevada                |
 | state=New Hampshire         |
 | state=New Jersey            |
 | state=New Mexico            |
 | state=New York              |
 | state=North Carolina        |
```

```
| state=North Dakota          |
| state=Ohio                  |
| state=Oklahoma              |
| state=Oregon                |
| state=Pennsylvania          |
| state=Rhode Island          |
| state=South Carolina        |
| state=South Dakota          |
| state=StateName             |
| state=Tennessee             |
| state=Texas                 |
| state=Utah                  |
| state=Vermont               |
| state=Virginia              |
| state=Washington            |
| state=West Virginia         |
| state=Wisconsin             |
| state=Wyoming               |
+-----------------------------+
```

bashObserva las particiones creadas también en HDFS

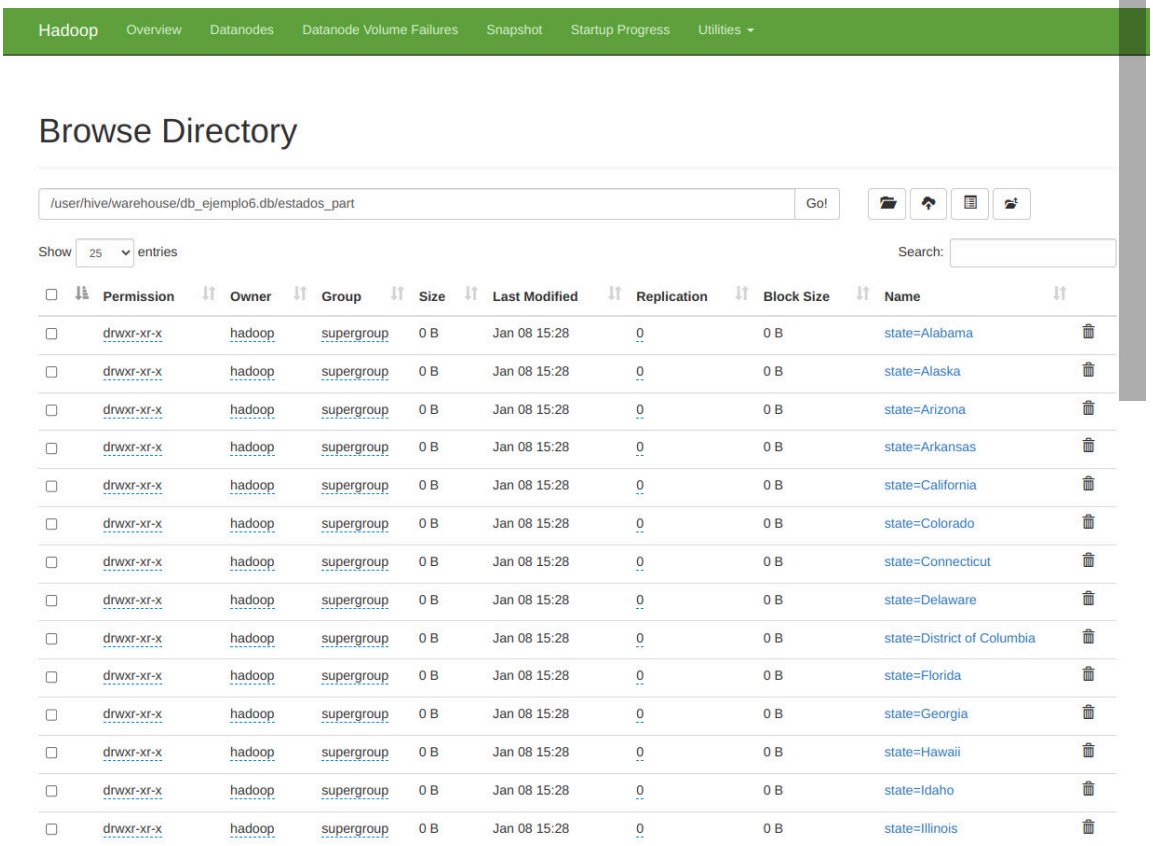

Figura 5.8_Hive: Partitions. (Fuente: Propia)

Creación de particiones

Imaginemos que queremos cargar particiones especificas de estados concretos

```
CREATE TABLE estados_part2 (
  delta INT,
  fecha TIMESTAMP,
  fpsid INT,
  forecastweek STRING,
  value INT,
  modelname STRING)
PARTITIONED BY (state STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

```
INSERT OVERWRITE TABLE estados_part2 PARTITION(state)
SELECT delta, fecha, fpsid, forecastweek, value, modelname, state FROM
estados
WHERE state='Utah';
```

```
show partitions estados_part2;
INFO  : OK
+-------------+
|  partition  |
+-------------+
| state=Utah  |
+-------------+
```

Creación de particiones de más de un campo

Imaginemos que queremos cargar particiones especificas de estados concretos

```
CREATE TABLE estados_part_varios_campos (
  delta INT,
  fecha TIMESTAMP,
  fpsid INT,
  forecastweek STRING,
  value INT)
PARTITIONED BY (state STRING, modelname STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

```
INSERT OVERWRITE TABLE estados_part_varios_campos PARTITION(state,
modelname)
SELECT delta, fecha, fpsid, forecastweek, value, state, modelname FROM
estados;
```

```
show partitions estados_part_varios_campos;
INFO  : OK
+-----------------------------------------------+
|                  partition                    |
+-----------------------------------------------+
| state=Alabama/modelname=CFH-IHME              |
| state=Alaska/modelname=CFH-IHME               |
| state=Arizona/modelname=CFH-IHME              |
```

```
 9  | state=Arkansas/modelname=CFH-IHME              |
10  | state=California/modelname=CFH-IHME            |
11  | state=Colorado/modelname=CFH-IHME              |
12  | state=Connecticut/modelname=CFH-IHME           |
13  | state=Delaware/modelname=CFH-IHME              |
14  | state=District of Columbia/modelname=CFH-IHME  |
15  | state=Florida/modelname=CFH-IHME               |
16  | state=Georgia/modelname=CFH-IHME               |
17  | state=Hawaii/modelname=CFH-IHME                |
18  | state=Idaho/modelname=CFH-IHME                 |
19  | state=Illinois/modelname=CFH-IHME              |
20  | state=Indiana/modelname=CFH-IHME               |
21  | state=Iowa/modelname=CFH-IHME                  |
22  | state=Kansas/modelname=CFH-IHME                |
23  | state=Kentucky/modelname=CFH-IHME              |
24  | state=Louisiana/modelname=CFH-IHME             |
25  | state=Maine/modelname=CFH-IHME                 |
26  | state=Maryland/modelname=CFH-IHME              |
27  | state=Massachusetts/modelname=CFH-IHME         |
28  | state=Michigan/modelname=CFH-IHME              |
29  | state=Minnesota/modelname=CFH-IHME             |
30  | state=Mississippi/modelname=CFH-IHME           |
31  | state=Missouri/modelname=CFH-IHME              |
32  | state=Montana/modelname=CFH-IHME               |
33  | state=Nebraska/modelname=CFH-IHME              |
34  | state=Nevada/modelname=CFH-IHME                |
35  | state=New Hampshire/modelname=CFH-IHME         |
36  | state=New Jersey/modelname=CFH-IHME            |
37  | state=New Mexico/modelname=CFH-IHME            |
38  | state=New York/modelname=CFH-IHME              |
39  | state=North Carolina/modelname=CFH-IHME        |
40  | state=North Dakota/modelname=CFH-IHME          |
41  | state=Ohio/modelname=CFH-IHME                  |
42  | state=Oklahoma/modelname=CFH-IHME              |
43  | state=Oregon/modelname=CFH-IHME                |
44  | state=Pennsylvania/modelname=CFH-IHME          |
45  | state=Rhode Island/modelname=CFH-IHME          |
46  | state=South Carolina/modelname=CFH-IHME        |
47  | state=South Dakota/modelname=CFH-IHME          |
48  | state=Tennessee/modelname=CFH-IHME             |
49  | state=Texas/modelname=CFH-IHME                 |
50  | state=Texas/modelname=ECMWF                    |
51  | state=Utah/modelname=CFH-IHME                  |
52  | state=Vermont/modelname=CFH-IHME               |
53  | state=Virginia/modelname=CFH-IHME              |
54  | state=Washington/modelname=CFH-IHME            |
55  | state=West Virginia/modelname=CFH-IHME         |
56  | state=Wisconsin/modelname=CFH-IHME             |
57  | state=Wyoming/modelname=CFH-IHME               |
58  +------------------------------------------------+
```

Esto es válido para todas las siguientes opciones de particiones

Creación de particiones desde los datos fuente

Imaginemos que queremos cargar particiones especificas directamente desde los archivos fuente. Usando como ejemplo `allstates.csv` vamos a crear particiones de `modelname`

```
CREATE TABLE estados_part_model (
  delta INT,
  fecha TIMESTAMP,
  fpsid INT,
  state STRING,
  forecastweek STRING,
  value INT)
PARTITIONED BY (modelname STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

```
LOAD DATA LOCAL INPATH '../hive/allstates.csv'
OVERWRITE INTO TABLE estados_part_model;
```

Tip

Estos casos sólo funcionan cuando el último campo es el de la partición

```
show partitions estados_part2;
INFO  : OK
+---------------------+
|      partition      |
+---------------------+
| modelname=CFH-IHME  |
| modelname=ECMWF     |
+---------------------+
```

Añadir una partición

Usando la tabla anterior `estados_part2` vamos a ver como se puede añadir una partición

```
ALTER TABLE estados_part2 ADD PARTITION (state='Alabama');
```

Tip

Estos casos sólo funcionan cuando el último campo es el de la partición

```
show partitions estados_part2;
INFO  : OK
+----------------+
|   partition    |
+----------------+
```

```
| state=Alabama  |
| state=Utah     |
+----------------+
```

Renombrar una partición

Usando la primera tabla `estados` vamos a ver como se puede añadir una partición

```
ALTER TABLE estados_part2 PARTITION (state='Alabama') RENAME TO PARTITION (state='Alaska');
```

```
show partitions estados_part2;
INFO  : OK
+---------------+
|   partition   |
+---------------+
| state=Alaska  |
| state=Utah    |
+---------------+
```

Eliminar una partición

Usando la primera tabla `estados` vamos a ver como se puede añadir una partición

```
ALTER TABLE estados_part2 DROP IF EXISTS PARTITION (state='Alaska');
```

```
show partitions estados_part2;
INFO  : OK
+-------------+
|  partition  |
+-------------+
| state=Utah  |
+-------------+
```

Reparar una tabla

Hive almacena una lista de particiones para cada tabla en su metastore. Sin embargo, si se agregan nuevas particiones directamente a HDFS (por ejemplo, usando el comando `hadoop fs -put`) o se eliminan de HDFS, el metastore (y, por lo tanto, Hive) no será consciente de estos cambios en la información de la partición a menos que ejecutemos `ALTER TABLE table_name ADD/DROP PARTITION` en cada una de las particiones recién agregadas o eliminadas, respectivamente.

Sin embargo, podemos ejecutar un comando de verificación de metastore con la opción de tabla de reparación:

```
MSCK [REPAIR] TABLE table_name [ADD/DROP/SYNC PARTITIONS];
```

Este comando actualizará los metadatos sobre las particiones en el almacén de metadatos de Hive para las particiones para las que aún no existen dichos metadatos.

Buckets

Buckets en Hive se utilizan para segregar datos de las particioones de las tablas de Hive en varios archivos o directorios. Se utilizan para que las consultas sean más eficientes.

Es decir, los datos presentes en esas particiones, se pueden dividir aún más en **buckets**. La división se realiza en base al *Hash* de columnas particulares que seleccionamos en la tabla.

Para indicar que nuestras tablas utilicen **buckets**, se usa la clausula `CLUSTERED BY` para indicar la columnas y el número de buckets(se recomienda que la cantidad de buckets sea potencia de 2).

Teniendo como base los datos y tablas anteriores, vamos a crear una tabla con `bucket`

```
CREATE TABLE estados_bucket (
  delta INT,
  fecha TIMESTAMP,
  fpsid INT,
  forecastweek STRING,
  value INT,
  modelname STRING)
PARTITIONED BY (state STRING)
CLUSTERED BY (fpsid) INTO 4 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

```
INSERT OVERWRITE TABLE estados_bucket PARTITION(state)
SELECT delta, fecha, fpsid, forecastweek, value, modelname, state FROM
estados;
```

Si observamos dentro de cualquier particion que se ha creado, vemos que esta está dividida en 4 bucket

```
hadoop@master:~/hive$ hdfs dfs -ls
/user/hive/warehouse/db_ejemplo6.db/estados_bucket/state=Alabama
Found 4 items
-rw-r--r--   1 hadoop supergroup          0 2025-01-08 14:34
/user/hive/warehouse/db_ejemplo6.db/estados_bucket/state=Alabama/000000_0
-rw-r--r--   1 hadoop supergroup       1923 2025-01-08 14:34
/user/hive/warehouse/db_ejemplo6.db/estados_bucket/state=Alabama/000001_0
-rw-r--r--   1 hadoop supergroup          0 2025-01-08 14:34
/user/hive/warehouse/db_ejemplo6.db/estados_bucket/state=Alabama/000002_0
-rw-r--r--   1 hadoop supergroup          0 2025-01-08 14:34
/user/hive/warehouse/db_ejemplo6.db/estados_bucket/state=Alabama/000003_0
hadoop@master:~/hive$
```