Análisis de Datos con Apache Hive

Sitio oficial:

https://hive.apache.org/

Tutorial:

https://www.tutorialspoint.com/hive/index.htm

Tutorial Hive View 2.0 con Ambari:

https://docs.cloudera.com/HDPDocuments/Ambari-2.6.2.0/bk_ambari-views/content/ch_using_hive_view.html

- Introducción a Apache Hive
- Despliegue y configuración de Hive
- Objetos Hive y el metastore Hive
- Programación básica en HiveQL

Hive es quizás la herramienta más utilizada en el ecosistema Hadoop (seguida de cerca por Spark). Hive ha proliferado debido a su facilidad de uso y su interfaz de programación familiar, y a menudo puede ser la mejor manera de lograr rápidamente la adopción y la productividad en la plataforma Hadoop

Presentación de Hive

El proyecto Apache Hive comenzó en Facebook en 2010 para proporcionar una interfaz de alto nivel a Hadoop MapReduce. En lugar de crear un nuevo lenguaje, como se hizo con PigLatin, el proyecto Hive se propuso poner una abstracción similar a SQL sobre MapReduce.

El proyecto Hive introdujo un nuevo lenguaje llamado HiveQL (o Hive Query Language), que implementa un subconjunto de SQL-92, una especificación estándar acordada internacionalmente para el lenguaje SQL, con algunas extensiones. Al igual que en el caso de Pig, la

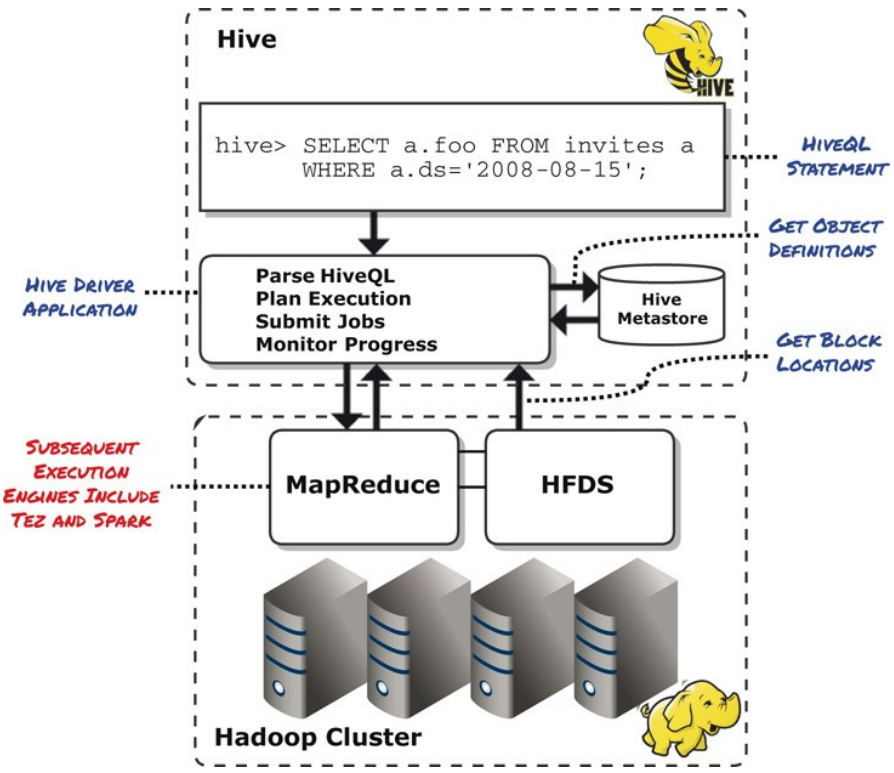

Figura 1. Descripción de alto nivel de Hive.

motivación para crear Hive fue que había pocos analistas con conocimientos de programación Java MapReduce. Sin embargo, Facebook reconoció que muchos analistas dominaban SQL. Además, SQL es el lenguaje común para las herramientas de BI, de visualización y de elaboración de informes, que suelen utilizar ODBC/JDBC como interfaz estándar.

En la implementación original de Hive, HiveQL sería analizado por el cliente de Hive y mapeado a una secuencia de operaciones Java MapReduce, que luego serían enviadas como trabajos en el clúster Hadoop. El progreso se monitoriza y los resultados se devuelven al cliente (o se escriben en la ubicación deseada en HDFS). La figura 1 ofrece una descripción a alto nivel de cómo Hive procesa los datos en HDFS.

Objetos Hive y el metastore Hive

Hive implementa una abstracción tabular de los objetos en HDFS, presentando los directorios y todos los archivos que contienen como una tabla en su modelo de programación. Al igual que en una base de datos relacional convencional, las tablas tienen columnas predefinidas con tipos de datos designados creados mediante comandos del lenguaje de definición de datos (DDL) de SQL. A continuación, se puede acceder a los datos de HDFS mediante sentencias del lenguaje de manipulación de datos (DML) de SQL, al igual que en un sistema de gestión de bases de datos normal.

Sin embargo, aquí termina la similitud, ya que Hadoop es una plataforma de lectura de esquemas, respaldada por un sistema de archivos inmutable, HDFS. Existen las siguientes diferencias clave entre Hive y una plataforma de base de datos convencional:

- La operación UPDATE no está (realmente) soportada

Aunque se ha introducido UPDATE en el dialecto HiveQL, HDFS sigue siendo un sistema de archivos inmutable.

- No hay transacciones, no hay *journaling*, no hay *rollbacks*, en definitiva, no hay aislamiento de transacciones.
- No hay integridad referencial, no hay claves primarias, no hay claves foráneas
- Los datos con un formato incorrecto (por ejemplo, datos mal escritos o registros mal formados) son simplemente representados al cliente como NULL

La asignación de las tablas a sus ubicaciones de directorio en HDFS y las columnas y sus definiciones se mantienen en el *metastore* de Hive. El *metastore* es una base de datos relacional (algo irónico) en la que escribe y lee el cliente Hive. Las definiciones de los objetos también incluyen los formatos de entrada y salida de los archivos representados por los objetos de la tabla (por ejemplo, CSVInputFormat, etc.) y SerDes (abreviatura de *Serialization/Deserialization functions*), que indican a Hive cómo extraer los registros y los campos de los archivos.

La figura 2 muestra un ejemplo de alto nivel de las interacciones entre Hive y el *metastore*.

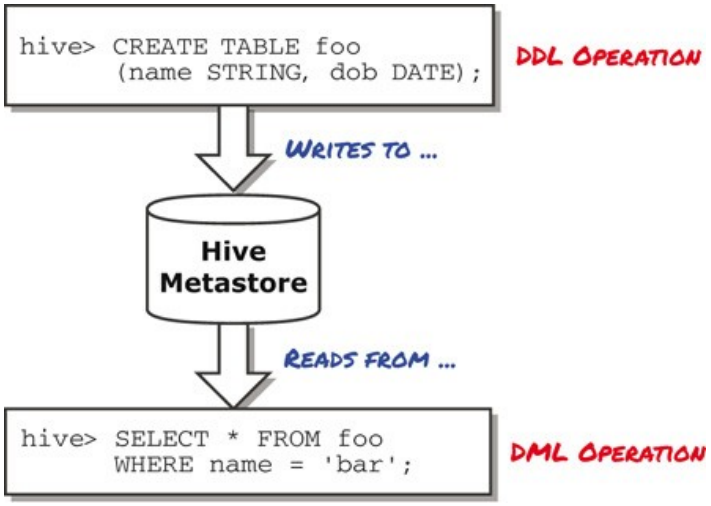

Figura 2. Interacción con el metastore de Hive

El *metastore* puede ser una base de datos Derby incrustada (la predeterminada) o una base de datos local o remota (como MySQL o PostgreSQL). En la mayoría de los casos, se implementa una base de datos compartida, que permite a los desarrolladores y analistas compartir definiciones de objetos. En las imágenes de abajo un *metastore* Hive en mysql:.

(En Hortonworks podemos acceder a Mysql con: **`mysql -u root -p hortonworks1`***)*

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| druid              |
| hive               |
| movielens          |
| mysql              |
| performance_schema |
| pruebaspark        |
| ranger             |
| superset           |
| sys                |
+--------------------+
```

```
mysql> use hive;
Database changed
mysql> show tables;
+---------------------------+
| Tables_in_hive            |
+---------------------------+
| AUX_TABLE                 |
| BUCKETING_COLS            |
| CDS                       |
| COLUMNS_V2                |
| COMPACTION_QUEUE          |
```

Hive CLI, HiveServer2, Beeswax y Beeline

Hive proporciona una interfaz de línea de comandos (CLI) de cliente que acepta y analiza los comandos de entrada HiveQL. Se trata de un método habitual para realizar consultas ad hoc. La figura 3 muestra la CLI de Hive.

```
hadoop-01
[javen@hadoop-01 ~]$hive

Logging initialized using configuration in file:/etc/hive/conf.dis
t/hive-log4j.properties
hive> SHOW TABLES IN movielens;
OK
data
genre
info
item
occupation
user
Time taken: 0.678 seconds, Fetched: 6 row(s)
hive>
```

Figura 3. La interfaz de línea de comandos de Hive.

La CLI de Hive se utiliza cuando la aplicación cliente o controlador de Hive se despliega en la máquina local, incluyendo la conexión al *metastore*.

Para accesos remotos se utiliza HiveServer2. Puede actuar ahora como una aplicación de controlador multisesión para múltiples clientes. HiveServer2 proporciona una interfaz JDBC que puede ser utilizada por clientes externos, como herramientas de visualización, así como una CLI ligera llamada Beeline. También hay una interfaz basada en la web llamada Beeswax, que se utiliza dentro del proyecto HUE (Entorno de Usuario Hadoop).

Para conectarnos vía Beeline en Hortonworks hemos de utilizar el comando:

```
beeline -u jdbc:hive2://127.0.0.1:10000
```

Uso del comando Hive

Hive comenzó con *hiveserver1*. Sin embargo, esta versión del servidor Hive no era muy estable. A veces suspendía o bloqueaba la conexión del cliente silenciosamente. Desde la v0.11.0, Hive ha incluido un nuevo servidor llamado *hivesever2* para sustituir a *hiveserver1*. *hiveserver2* tiene un servidor mejorado diseñado para la concurrencia de múltiples clientes y una autenticación mejorada. También recomienda utilizar **beeline** como interfaz principal de línea de comandos de Hive en lugar del comando **hive**. La principal diferencia entre las dos versiones de servidores es la forma en que los clientes se conectan a ellos. **hive** es un cliente basado en Apache-Thrift, y **beeline** es un cliente JDBC. El comando *hive* se conecta directamente a los controladores Hive, por lo que necesitamos instalar la biblioteca Hive en el cliente. Sin embargo, *beeline* se conecta a hiveserver2 a través de conexiones JDBC sin necesidad de instalar las librerías Hive en el cliente. Esto significa que podemos ejecutar *beeline* remotamente desde fuera del clúster.

En las dos tablas siguientes se enumeran los comandos más utilizados en los distintos modos de comando teniendo en cuenta las diferentes preferencias del usuario:

Purpose	hiveserver2 - beeline	hiveserver1 - hive
Connect server	beeline -u <jdbc_url>	hive -h <hostname> -p <port>
Help	beeline -h	hive -H
Run query	beeline -e "hql query" beeline -f hql_query_file.hql	hive -e "hql query" hive -f hql_query_file.hql

Enter mode	`beeline`	`hive`
Connect server	`!connect <jdbc_url>`	N/A
List tables	`!table` `show tables; --also support`	`show tables;`
List columns	`!column table_name` `desc table_name;`	`desc table_name;`
Run query	`select * from table_name;`	`select * from table_name;`

Como se ve arriba, también puede ejecutar sentencias y scripts Hive en modo no interactivo, o por lotes, utilizando la opción -f o -e como se muestra en el Listado 1.

```
# run all statements in a file

$ hive -f MyHiveQuery.hql

# run an individual statement

$ hive -e "SELECT * FROM mytable"
```

Listado 1 Ejecución de consultas Hive en modo batch

Bases de datos y tablas Hive

Los objetos Hive se componen de bases de datos y tablas. Las bases de datos Hive se utilizan para la organización, la autorización y la gestión del espacio de nombres. Las tablas Hive existen en una base de datos Hive. La base de datos por defecto en Hive se llama **`default`**. El contexto de la base de datos puede cambiarse mediante la sentencia USE, como se muestra en el listado .2.

```
hive> USE bikeshare;
```

Listado 2. Cambio de base de datos Hive

También se puede hacer referencia a los objetos utilizando su identificador de objeto completo (que incluye el nombre de la base de datos y de la tabla) como se muestra en el Listado 3.

```
hive> SELECT * FROM bikeshare.trips;
```

Listado 3 Referencia a objetos en Hive

Autorización de Hive

Hive soporta la autorización básica, que determina los niveles de acceso a los objetos dentro de una instancia Hive. La autorización en Hive no está habilitada por defecto; para habilitar la autorización es necesario añadir los valores de configuración al archivo de configuración principal de Hive (**`hive-site.xml`**) como se muestra en el Listado 4.

```
...

<property>

  <name>hive.security.authorization.enabled</name>

  <value>true</value>

</property>

<property>

  <name>hive.security.authorization.createtable.owner.grants</name>

  <value>ALL</value>

</property>

...
```

Listado 4 Configuración de la autorización de Hive

Una vez habilitado, se pueden asignar privilegios a nivel de objeto utilizando la sentencia GRANT. Esto es generalmente sinónimo de la semántica GRANT disponible en la mayoría de las plataformas de gestión de bases de datos relacionales. En el Listado 5 se muestra un ejemplo de asignación de privilegios a una tabla en Hive.

```
hive> GRANT SELECT ON TABLE trips TO USER javen;
```

Listado 5 La sentencia GRANT

Los privilegios específicos disponibles en Hive incluyen SELECT, CREATE, ALTER, DROP y otros. También existe un privilegio ALL que concede todas las acciones a un usuario concreto.

Los privilegios también pueden ser revocados, como se muestra en el Listado 6.

```
hive> REVOKE SELECT ON TABLE trips FROM USER javen;
```

Listado 6 Declaración REVOKE

También existe el comando SHOW GRANT, que puede utilizarse para mostrar todos los privilegios asignados a un usuario, grupo o rol específico en un objeto particular o en todos los objetos.

Precaución: La autorización de Hive por sí sola no es segura

La autorización de Hive se construye sobre la autorización de HDFS, que es débil en ausencia de una fuerte autenticación de clúster. Si la seguridad es una consideración de diseño primordial, debe utilizar la autenticación Kerberos, junto con la autorización Hive y la autorización HDFS.

Creación de objetos Hive

A través de sus tipos de datos y el lenguaje de definición de datos (DDL), Hive soporta la mayoría de los tipos de datos primitivos comunes, similares a los que se encuentran en la mayoría de los sistemas de bases de datos, así como varios tipos de datos complejos. Los tipos de datos simples utilizados en Hive se enumeran en la Tabla 1.

Datatype	Category	Description
TINYINT	Numeric	1-byte signed integer (–128 to 127)
SMALLINT	Numeric	2-byte signed integer (–32,768 to 32,767)
INT	Numeric	4-byte signed integer
BIGINT	Numeric	8-byte signed integer
FLOAT	Numeric	4-byte single precision floating point
DOUBLE	Numeric	8-byte double precision floating point
DECIMAL(p,s)	Numeric	User definable precision and scale
TIMESTAMP	Date/Time	Unix timestamp
DATE	Date/Time	Date (formatted as YYYY-MM-DD)
STRING	String	Character sequence (variable length)
VARCHAR	String	Character sequence (variable length)
CHAR(n)	String	Character sequence (fixed length)
BOOLEAN	Misc	True or False
BINARY	Misc	Raw binary data

Tabla 1 Tipos de datos simples de Hive

Hive tiene tres tipos de complejos principales: ARRAY, MAP y STRUCT. Estos tipos de datos se construyen sobre los tipos de datos primitivos. ARRAY y MAP son similares a los de Java. STRUCT es un tipo de registro que puede contener un conjunto de campos de cualquier tipo. Los tipos complejos permiten el anidamiento de tipos. Los detalles de los tipos complejos son los siguientes:

Complex type	Description	Example
ARRAY	This is a list of items of the same type, such as [val1, val2, and so on]. You can access the value using array_name[index], for example, fruit[0]="apple". Index starts from 0.	["apple","orange","mango"]
MAP	This is a set of key-value pairs, such as {key1, val1, key2, val2, and so on}. You can access the value using map_name[key] for example, fruit[1]="apple".	{1: "apple",2: "orange"}
STRUCT	This is a user-defined structure of any type of field, such as {val1, val2, val3, and so on}. By default, STRUCT field names will be col1, col2, and so on. You can access the value using structs_name.column_name, for example, fruit.col1=1.	{1, "apple"}

Creación de tablas en Hive

Sintaxis Básica y Estructura

a. Nombre y Definición de Columnas

Esta es la parte obligatoria donde se define el nombre de la tabla y todas sus columnas con sus tipos de datos (incluidos los tipos complejos como ARRAY, MAP, STRUCT).

```
CREATE TABLE [IF NOT EXISTS] table_name (
  col_name1 data_type1 [COMMENT '...'],
  col_name2 data_type2 [COMMENT '...'],
  ...
)
```

Argumento	Descripción
IF NOT EXISTS	Opción para evitar un error si la tabla ya existe.

Argumento	Descripción
`col_name data_type`	Define el nombre y el **tipo primitivo o complejo** de la columna.
`COMMENT '...'`	Permite añadir una descripción a la tabla o a una columna individual.

b. EXTERNAL (Tipo de Tabla)

Esta cláusula define el tipo de gestión de la tabla, una de las distinciones más importantes en Hive:

Cláusula	Tipo de Tabla	Gestión y DROP
Por defecto (sin EXTERNAL)	**Managed** (Interna)	Cuando se ejecuta `DROP TABLE`, Hive elimina tanto los **metadatos** (del Metastore) como los **datos subyacentes** (de HDFS).
EXTERNAL	**External** (Externa)	Cuando se ejecuta `DROP TABLE`, Hive solo elimina los **metadatos** del Metastore. Los **datos en HDFS permanecen intactos**. Es ideal para datos compartidos o cuando la pérdida de datos es inaceptable.

Argumentos de Formato y Ubicación de Datos

Estas cláusulas le dicen a Hive **cómo** leer y escribir los datos en HDFS.

c. ROW FORMAT (Formato de Fila)

Especifica el **formato de serialización y deserialización (SerDe)** que Hive debe usar para leer y escribir los datos en los archivos de texto (o formatos similares).

- **`DELIMITED`**: Utilizado para archivos CSV o TSV. Permite definir los delimitadores:
    - `FIELDS TERMINATED BY` 'char' (separador de **columnas**, p. ej., **`,`** o `\t`).
    - `COLLECTION ITEMS TERMINATED BY` 'char' (separador para elementos de **`ARRAY`** o **`MAP`**).
    - `MAP KEYS TERMINATED BY` 'char' (separador entre clave y valor en un **MAP**).
    - `LINES TERMINATED BY` 'char' (separador de **líneas**, usualmente \n).
- **`SERDE 'clase.serde.completa'`**: Permite especificar un SerDe personalizado o avanzado, como org.apache.hadoop.hive.serde2.JsonSerde para JSON o SerDes binarios.

d. `STORED AS` (Formato de Almacenamiento)

Define el **formato de archivo físico** de los datos en HDFS. Esto impacta directamente en el rendimiento de lectura y la eficiencia del espacio de almacenamiento.

- `TEXTFILE`: El formato por defecto. Simplemente almacena datos como texto delimitado. Lento para consultas.
- `SEQUENCEFILE`: Formato binario de Hadoop, más rápido que `TE1XTFILE`.

- **ORC (Optimized Row Columnar)**: Formato columnar optimizado para Hive. Muy eficiente para almacenamiento y consultas. **Recomendado**.
- **PARQUET**: Formato columnar de código abierto, altamente compatible con Spark y otros motores de Big Data. **Recomendado**.

5. LOCATION (Ubicación en HDFS)

Especifica la ruta exacta en HDFS donde se encuentran o se almacenarán los datos de la tabla.

`LOCATION '/user/hive/warehouse/mi_directorio_personalizado/'`

Si se omite, Hive utiliza su ubicación por defecto dentro del Hive Warehouse (`/user/hive/warehouse/<nombre_tabla>`).

A continuación, se presenta un breve ejercicio para todos los tipos de datos de uso común.

Preparamos los datos de la siguiente manera:

```
$vi employee.txt
      Michael|Montreal,Toronto|Male,30|DB:80|Product:Developer,Lead
      Will|Montreal|Male,35|Perl:85|Product:Lead,Test:Lead
      Shelley|New York|Female,27|Python:80|Test:Lead,COE:Architect
      Lucy|Vancouver|Female,57|Sales:89,HR:94|Sales:Lead
```

En el Listado 7 se muestra una sentencia DDL típica de Hive utilizada para crear una tabla donde importaremos posteriormente los datos de arriba.

```
hive> CREATE TABLE employee (
      name STRING,
      work_place ARRAY<STRING>,
      gender_age STRUCT<gender:STRING,age:INT>,
      skills_score MAP<STRING,INT>,
      depart_title MAP<STRING,ARRAY<STRING>>
      )
      ROW FORMAT DELIMITED
      FIELDS TERMINATED BY '|'
      COLLECTION ITEMS TERMINATED BY ','
      MAP KEYS TERMINATED BY ':'
      STORED AS TEXTFILE;
```

Listado 7 Declaración CREATE TABLE
La sentencia CREATE TABLE del Listado 7 no incluye una directiva EXTERNAL, ni una directiva LOCATION. Como tal, la tabla es simplemente una referencia a un directorio vacío creado en el directorio interno de Hive (**`/user/hive/warehouse`**).

Hive proporciona una sentencia DESCRIBE que puede utilizarse para mostrar el esquema de las tablas Hive. En el Listado 8 se proporciona un ejemplo de sentencia DESCRIBE y su salida.

```
hive> DESCRIBE employee;
OK
name                    string
work_place              array<string>
```

```
gender_age              struct<gender:string,age:int>
skills_score            map<string,int>
depart_title            map<string,array<string>>
```

Listado 8 Sentencia DESCRIBE

También existe un comando alternativo DESCRIBE FORMATTED que proporciona información adicional, como la ubicación física de los datos en HDFS y otros metadatos sobre el objeto.

Precaución: Tablas internas frente a tablas externas en Hive

Cuando se crean tablas en Hive, la opción por defecto es crear una tabla "interna" de Hive. Los directorios de las tablas internas son gestionados por Hive, y una sentencia DROP TABLE para una tabla interna eliminará los archivos correspondientes de HDFS. Normalmente se recomienda utilizar tablas externas especificando la palabra clave EXTERNAL en la sentencia CREATE TABLE. Esto proporciona el esquema y la ubicación del objeto en HDFS, pero una operación DROP TABLE no elimina el directorio y los archivos.

Cargar datos en la tabla:

```
hive> LOAD DATA INPATH '/user/maria_dev/employee.txt'
      OVERWRITE INTO TABLE employee;
```

Varias consultas:

```
hive> select * from employee;

Michael ["Montreal","Toronto"]  {"gender":"Male","age":30}      {"DB":80}       {"Product":["Developer"]}

Will    ["Montreal"]    {"gender":"Male","age":35}      {"Perl":85}     {"Product":["Lead"],"Test":["Lead"]}

Shelley ["New York"]    {"gender":"Female","age":27}    {"Python":80}   {"Test":["Lead"],"COE":["Architect"]}

Lucy    ["Vancouver"]   {"gender":"Female","age":57}    {"Sales":89,"HR":94}    {"Sales":["Lead"]}

SELECT work_place FROM employee;
      +----------------------+
      |      work_place      |
      +----------------------+
      | [Montreal, Toronto]  |
      | [Montreal]           |
      | [New York]           |
      | [Vancouver]          |
      +----------------------+
      4 rows selected (27.231 seconds)

      > SELECT
      > work_place[0] as col_1, work_place[1] as col_2,
      > work_place[2] as col_3
      > FROM employee;
      +------------+----------+--------+
      |   col_1    |  col_2   | col_3  |
      +------------+----------+--------+
```

```
     | Montreal   | Toronto  |         |
     | Montreal   |          |         |
     | New York   |          |         |
     | Vancouver  |          |         |
     ------------+----------+---------+
     4 rows selected (24.689 seconds)
```

Consulta toda la estructura y cada atributo de la estructura en la tabla:

```
> SELECT gender_age FROM employee;
     +------------------+
     |    gender_age    |
     +------------------+
     | [Male, 30]       |
     | [Male, 35]       |
     | [Female, 27]     |
     | [Female, 57]     |
     +------------------+
     4 rows selected (28.91 seconds)


     > SELECT gender_age.gender, gender_age.age FROM employee;
     +------------+------+
     |   gender   | age  |
     +------------+------+
     | Male       | 30   |
     | Male       | 35   |
     | Female     | 27   |
     | Female     | 57   |
     +------------+------+
     4 rows selected (26.663 seconds)
```

Consultar todo el *map* y cada elemento del *map* en la tabla:

```
> SELECT skills_score FROM employee;
     +--------------------+
     |    skills_score    |
     +--------------------+
     | {DB=80}            |
     | {Perl=85}          |
     | {Python=80}        |
     | {Sales=89, HR=94}  |
     +--------------------+
     4 rows selected (32.659 seconds)


     > SELECT
     > name, skills_score['DB'] as DB, skills_score['Perl'] as Perl,
     > skills_score['Python'] as Python,
     > skills_score['Sales'] as Sales,
     > skills_score['HR'] as HR
     > FROM employee;
```

```
+----------+-----+------+--------+--------+-----+
|   name   | db  | perl | python | sales  | hr  |
+----------+-----+------+--------+--------+-----+
| Michael  | 80  |      |        |        |     |
| Will     |     | 85   |        |        |     |
| Shelley  |     |      | 80     |        |     |
| Lucy     |     |      |        | 89     | 94  |
+----------+-----+------+--------+--------+-----+
4 rows selected (24.669 seconds)
```

Consulta el tipo de compuesto en la tabla:

```
> SELECT depart_title FROM employee;
    +---------------------------------+
    |          depart_title           |
    +---------------------------------+
    | {Product=[Developer, Lead]}     |
    | {Test=[Lead], Product=[Lead]}   |
    | {Test=[Lead], COE=[Architect]}  |
    | {Sales=[Lead]}                  |
    +---------------------------------+
    4 rows selected (30.583 seconds)


    > SELECT
    > name, depart_title['Product'] as Product, depart_title['Test']
    as Test,
    > depart_title['COE'] as COE, depart_title['Sales'] as Sales
    > FROM employee;
    +--------+--------------------+---------+-------------+------+
    |   name |      product       |  test   |     coe     |sales |
    +--------+--------------------+---------+-------------+------+
    | Michael| [Developer, Lead]  |         |             |      |
    | Will   | [Lead]             | [Lead]  |             |      |
    | Shelley|                    | [Lead]  | [Architect] |      |
    | Lucy   |                    |         |             |[Lead]|
    +--------+--------------------+---------+-------------+------+
    4 rows selected (26.641 seconds)


    > SELECT
    > name, depart_title['Product'][0] as product_col0,
    > depart_title['Test'][0] as test_col0
    > FROM employee;
    +----------+---------------+------------+
    |   name   | product_col0  | test_col0  |
    +----------+---------------+------------+
    | Michael  | Developer     |            |
    | Will     | Lead          | Lead       |
    | Shelley  |               | Lead       |
    | Lucy     |               |            |
    +----------+---------------+------------+
    4 rows selected (26.659 seconds)
```

Formatos de entrada/salida en Hive

Hive utiliza un InputFormat y un SerDe (abreviatura de Serializer/Deserializer) para determinar cómo leer los archivos de entrada y extraer los registros para su procesamiento.

El InputFormat se especifica mediante la directiva STORED AS en la sentencia CREATE TABLE. En el Listado 7, se especificó STORED AS TEXTFILE, que a su vez corresponde al TextInputFormat. El TextInputFormat es el mismo InputFormat utilizado en nuestros ejemplos de MapReduce y se utiliza para todos los archivos de texto en Hive.

Del mismo modo, otras directivas STORED AS soportadas incluidas en la sintaxis DDL de Hive CREATE TABLE son las siguientes

- SEQUENCEFILE
- ORC
- PARQUET
- AVRO

Avro y SequenceFile son formatos de archivo preserializados y autodescriptivos, mientras que ORC y Parquet son formatos columnares optimizados.

Al igual que MapReduce y Pig, Hive utiliza un OutputFormat para especificar cómo escribir los datos como resultado de una consulta Hive. El OutputFormat por defecto de Hive es el HiveIgnoreKeyTextOutputFormat. Esta clase OutputFormat se utiliza para escribir los datos en archivos de texto en Hadoop. Hay disponibles OutputFormats alternativos para ORC, Parquet, Avro, SequenceFiles, etc.

Carga de datos en Hive

Los datos se pueden cargar en las tablas de una de las siguientes maneras:

- Simplemente copiando el/los archivo/s en el directorio HDFS para el objeto de la tabla.
- Utilizar el comando Hive LOAD DATA INPATH para los datos que existen en HDFS, que utiliza una función subyacente de movimiento de HDFS.
- Utilizar el comando Hive LOAD DATA LOCAL INPATH para los datos que no existen en HDFS, que utiliza una función subyacente **put** de HDFS.

Un ejemplo del comando LOAD DATA INPATH se proporciona en el Listado 9.

```
hive> LOAD DATA INPATH '/bikeshare/stations' INTO TABLE stations;
```

Listado 9 Declaración LOAD

Con cualquier operación de carga se puede añadir la directiva OVERWRITE para reemplazar el contenido del directorio existente con los nuevos datos que se deseen. Esto se muestra en el Listado 10.

```
hive> LOAD DATA INPATH '/bikeshare/stations'

> OVERWRITE INTO TABLE stations;
```

Listado 10 Opción OVERWRITE

Hive también admite la carga datos durante el proceso de creación de la tabla mediante una operación CREATE TABLE AS SELECT. En el Listado 11 se ofrece un ejemplo de ello.

```
hive> CREATE TABLE stations_copy AS

> SELECT * FROM stations;
```

Listado 11 CREAR TABLA COMO SELECT

Análisis de datos con Hive

El lenguaje de consulta de Hive (HiveQL) se basa en la especificación SQL-92, con algunas funciones adicionales específicas de Hive y algunas limitaciones debidas a las propiedades inmutables inherentes a HDFS.

Las palabras clave de Hive, como SELECT y CREATE, no distinguen entre mayúsculas y minúsculas, pero suelen ir en mayúsculas por convención (como ocurría con Pig). Las sentencias HiveQL pueden abarcar varias líneas y se terminan con un punto y coma. Los comentarios de una sola línea se soportan con el doble guión (--). La semántica típica de SQL, como las listas de columnas y las cláusulas WHERE, está totalmente soportada en Hive.

En el listado 12 se muestra un ejemplo de sentencia SELECT en Hive.

```
-- this is a comment

hive> SELECT name, lat, long

> FROM stations

> WHERE landmark = 'San Jose';
```

Listado 12 Sentencia SELECT

Uso de las funciones integradas de Hive

Hive incluye numerosas funciones incorporadas para realizar operaciones matemáticas comunes como ROUND, CEIL, FLOOR, RAND, etc., así como para manipular cadenas y fechas como SUBSTRING, LOWER, UPPER, RTRIM, LTRIM, CONCAT, TO_DATE, DAY, MONTH, YEAR, etc. Muchas de estas funciones son idénticas a sus homólogas disponibles en los dialectos SQL de los RDBMS más populares.

Sugerencia: Cómo escribir las UDF de Hive

Si no se encuentra una función que realice la operación que se necesita, siempre se puede escribir una propia. Las UDF de Hive pueden escribirse en Java. Encontrará más información sobre cómo escribir UDFs de Hive en https:// hive.apache.org.

En el listado 13 se muestra un ejemplo de invocación de una función integrada de Hive.

```
hive> SELECT LOWER(name) FROM stations;
```

Listado 13 Funciones integradas de Hive

También puede utilizar el comando DESCRIBE para mostrar la ayuda y el uso de la función, como se muestra en el Listado 14.

```
hive> DESCRIBE FUNCTION RAND;

OK

RAND([seed]) - Returns a pseudorandom number between 0 and 1

Time taken: 0.013 seconds, Fetched: 1 row(s)
```

Agrupación y agregación de datos con Hive

HiveQL proporciona una semántica para la agrupación y agregación de datos que resultaría familiar a la mayoría de los analistas o desarrolladores de SQL. En el listado 15 se ofrece un ejemplo de ello.

```
hive> SELECT landmark, COUNT(*) FROM stations

> GROUP BY landmark;

OK

Mountain View   7

Palo Alto       5

Redwood City    7

San Francisco   35

San Jose        16

Time taken: 19.443 seconds, Fetched: 5 row(s)
```

Listado 15 Agrupación y agregación de datos en Hive

Unir datos con Hive

Las uniones en Hive funcionan de forma sinónima con sus homólogos en dialectos SQL estándar, incluyendo la asignación de alias de tabla para desreferenciar columnas en un conjunto de datos unidos. En el listado 16 se muestra un ejemplo de operación JOIN.

```
hive> SELECT t.trip_id

> ,t.duration

> ,t.start_date

> ,s.name

> FROM

> stations s

> JOIN trips t ON s.station_id = t.start_terminal;
```

Listado 16 Declaración JOIN

Hive admite todos los tipos de unión comunes, INNER JOIN (por defecto), LEFT OUTER JOIN, RIGHT OUTER JOIN y FULL OUTER JOIN. Hive sólo admite condiciones de unión de igualdad; por ejemplo, una condición ON como `s.station_id <> t.start_terminal` no se admite en una sentencia Hive JOIN.

Hive también admite la operación CROSS JOIN, pero al igual que con el operador CROSS de Pig, hay que utilizar esta operación con precaución, ya que podría generar cantidades masivas de datos.

Además de los tipos típicos de join, Hive también soporta el LEFT SEMI JOIN. La operación LEFT SEMI JOIN es útil en los casos en los que normalmente se utiliza una cláusula IN con una subconsulta en algunos dialectos de SQL. Hive no admite el uso de cláusulas IN o EXISTS con subconsultas, por lo que el LEFT SEMI JOIN es una solución práctica. En el listado 17 se muestra un ejemplo.

Supongamos que tienes:

- `clientes`: lista de todos los clientes
- `pedidos`: lista de pedidos realizados

Quieres obtener solo los clientes que han realizado al menos un pedido (sin repetir clientes y sin traer datos del pedido).

Forma ineficiente (no recomendada):

```
SQL
SELECT DISTINCT c.cliente_id, c.nombre
FROM clientes c
LEFT JOIN pedidos p ON c.cliente_id = p.cliente_id
WHERE p.cliente_id IS NOT NULL;
```

Forma correcta y eficiente con SEMI JOIN:

```
SQL
SELECT c.cliente_id, c.nombre
FROM clientes c
LEFT SEMI JOIN pedidos p ON (c.cliente_id = p.cliente_id);
```

Listado 17 Declaración LEFT SEMI JOIN