Apunts CE_5075 3.1

lloc:	Institut d'Ensenyaments a Distància de les Illes Balears	Imprès per:	Carlos Sanchez Recio
Curs:	Big data aplicat	Data:	dimarts, 26 de novembre 2024, 09:02
Llibre:	Apunts CE_5075 3.1		

Taula de continguts

1. Introducció

En aquest lliurament veurem què entenem per un sistema OLAP, centrat en l'analítica de les dades. En el nucli d'aquests sistemes trobam els anomenats magatzems de dades (o *data warehouses*). També introduirem el concepte de llac de dades (o *data lake*).

I dedicarem la major part del lliurament a Apache Hive, una eina de l'ecosistema Hadoop, que serveix com a infraestructura per a definir un *data warehouse* a partir de dades emmagatzemades en HDFS. Hive ens permetrà accedir a les dades mitjançant un llenguatge molt semblant a SQL, anomenat HiveQL, amb la qual cosa eliminam la complexitat de desenvolupar programes MapReduce per consultar o transformar les dades.

En el següent vídeo pots veure un resum del que tractarem en aquest lliurament.

Vídeo: *Resum del Lliurament 3*

2. Sistemes OLTP i OLAP

La majoria d'organitzacions tenen sistemes informàtics que generen una gran quantitat de dades diàriament, derivades de les operacions pròpies de l'organització. Aquestes dades normalment s'emmagatzemen en bases de dades relacionals, i més recentment, en bases de dades NoSQL. Aquests sistemes són transaccionals, orientats a gestionar un gran nombre d'operacions d'actualització, inserció i esborrat. També se'n fan operacions de consulta, tot i que normalment són senzilles, amb poc temps de processament. Aquests sistemes reben el nom d'**OLTP**, ***On-Line Transaction Processing*** (processament de transaccions en línia).

Per altra banda, ja sabem que les organitzacions tenen un interès en extreure informació rellevant a partir de totes aquestes dades gestionades per aquests sistemes OLTP. En general, les organitzacions volen obtenir-ne informació estratègica i tàctica que els serveixi com a suport per a la presa de decisions. És el que anomenam ***business intelligence*** (intel·ligència empresarial o dels negocis). Tot i que en xerrarem amb més profunditat en el lliurament 6 del mòdul de Sistemes de big data, veiem ara una definició de *business intelligence*:

DEFINICIÓ

Intel·ligència empresarial, intel·ligència de negoci o BI (en anglès *business intelligence*) són el conjunt d'estratègies i eines enfocades a l'administració i creació de coneixement mitjançant l'anàlisi de dades existents a una organització o empresa.

Font: Wikipedia

Apareixen així, els **sistemes d'ajuda a la presa de decisions** o ***decission support systems*** (DSS), on l'objectiu no és gestionar les tasques diàries (l'operativa) de l'organització, sinó oferir les eines necessàries per finalment generar uns informes que permetin als directius prendre decisions.

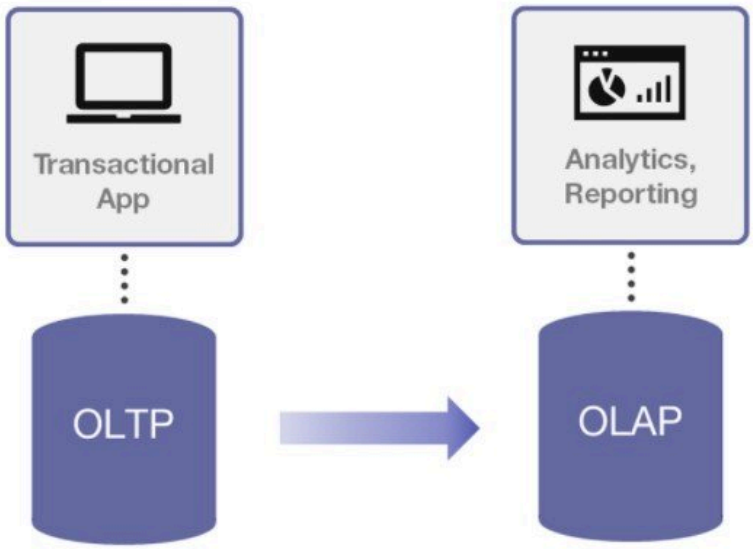
Aquests sistemes també reben el nom d'**OLAP**, ***On-Line Analytical Processing***, ja que l'èmfasi no està en les transaccions sinó en l'anàlisi de la informació. Així doncs, aquí no tenim operacions d'inserció, modificació i esborrat: les dades són estàtiques, una còpia de part de les dades transaccionals, després d'aplicar diverses transformacions. En canvi, tenim unes consultes molt més complexes que les típiques dels sistemes OLTP i que requereixen més temps de processament.

Els sistemes OLAP no treballen directament amb les base de dades operacionals (relacionals o NoSQL), sinó que prèviament se'n fa una selecció de les dades d'operativa considerades rellevants, que són guardades en els anomenats ***data warehouses*** (DW) o **magatzems de dades**. Aquesta és una de les definicions més referenciada de magatzem de dades:

DEFINICIÓ

Un magatzem de dades (o *data warehouse*) és una col·lecció de dades orientades a temes, integrades, variants en el temps i no volàtils, que donen suport als sistemes de presa de decisions de la direcció.

William H. Inmon. *Building the Data Warehouse*, 3rd Edition. John Wiley & Sons, Inc., EUA, 2002.



Imatge: *Sistemes OLTP i OLAP. Font: pc-solucion.es*

Vegem a continuació les principals diferències entre els sistemes OLTP i OLAP:

	OLTP	**OLAP**
Objectiu	Gestió de l'operativa de l'organització	Anàlisi de dades per a la presa de decisions
Usuaris	Molts	Pocs
Treballa amb	Transaccions predefinides (curtes)	Consultes específiques complexes (llargues)
Accés	Lectura i principalment escriptura	Principalment lectura
Dades	Detallades. Numèriques i alfanumèriques	Agregades. Principalment numèriques
Temporalitat	Principalment dades actuals	Principalment dades històriques
Actualitzacions	Contínues	Es fan de manera periòdica. Entre una actualització periòdica i altra, les dades són estables.

Taula: *Diferències principals entre els sistemes OLTP i els OLAP*

3. Arquitectura d'un sistema OLAP

Ja hem comentat en l'apartat anterior que les dades d'un magatzem de dades, i per extensió d'un sistema OLAP, normalment provenen de les bases de dades operacionals de l'organització. Els processos d'extracció, transformació i càrrega (**ETL** - *Extraction, Transformation and Load*) són els que extreuen, netegen, filtren, validen i carreguen les dades en el magatzem de dades. Tot i que en aquest lliurament no hi entrarem, les eines ETL (les que ens permeten fer aquests processos ETL d'una manera adequada) són fonamentals per al bon funcionament d'un sistema OLAP.

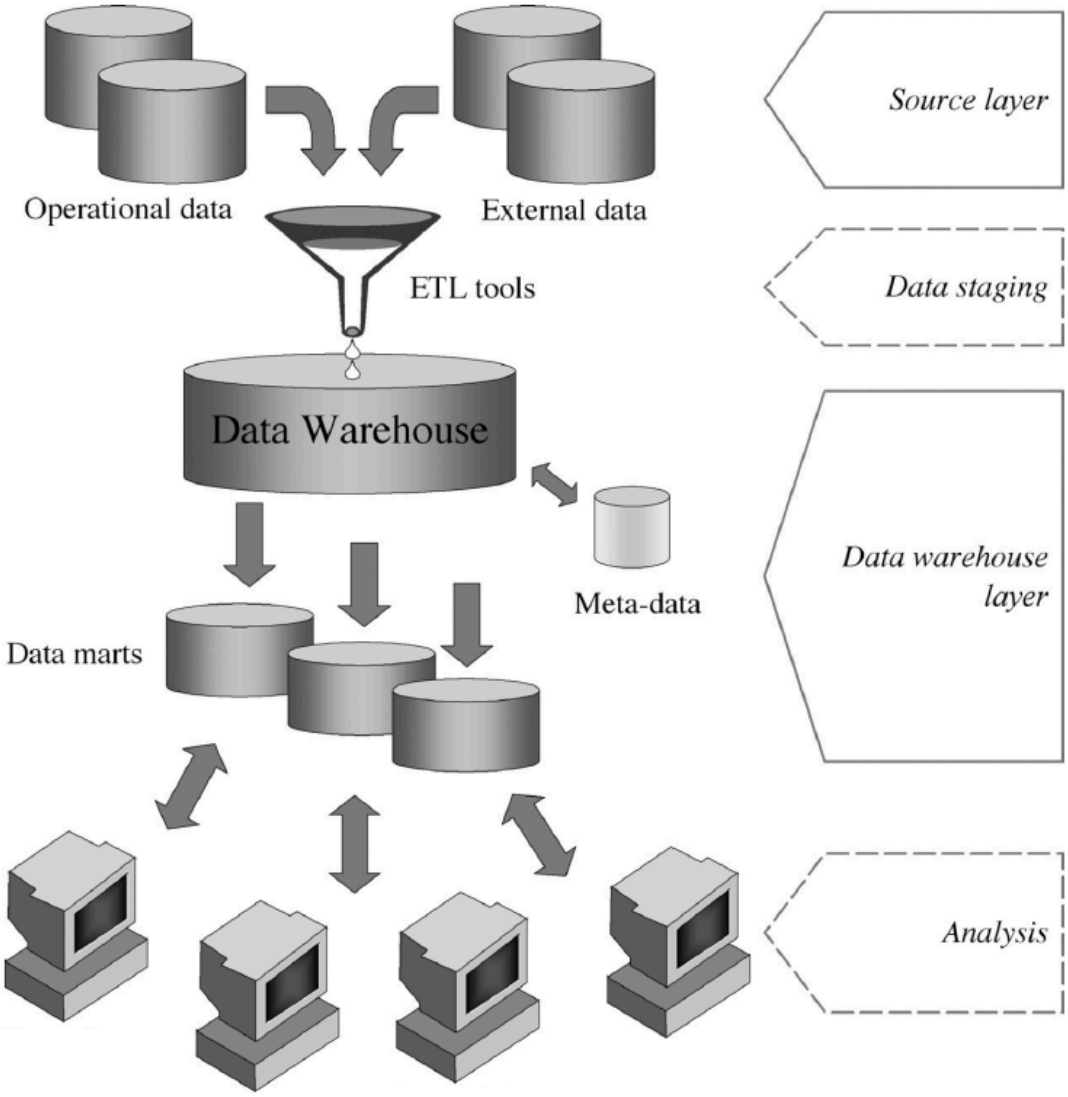
Així, en un sistema OLAP podem distingir quatre elements principals:

- Les fonts de dades, que normalment són les bases de dades operacionals de l'organització, tot i que també poden ser dades externes.
- Les eines ETL per fer la importació.
- El magatzem de dades, on s'emmagatzemen de manera centralitza les dades considerades rellevants per a prendre decisions. Pot estructurar-se en diversos *data marts* (vegeu definició a continuació).
- Eines d'anàlisi i visualització, que permeten mostrar als directius els informes d'una manera entenedora perquè puguin prendre les decisions adequades.

DEFINICIÓ

Un ***data mart*** és un subconjunt o agregació de les dades emmagatzemades en un magatzem de dades primari que inclou informació rellevant sobre una àrea específica del negoci.

Tenint en consideració aquests elements, la següent imatge mostra l'arquitectura típica d'un sistema OLAP.



Imatge: *Arquitectura típica d'un sistema OLAP. Font: Universitat de Castella-La Manxa*

4. Magatzems i llacs de dades

Ja hem comentat què són els **magatzems de dades** o ***data warehouses***. Ara introduirem també el concepte de **llac de dades** o ***data lake***.

Un *data warehouse* i un *data lake* són dos tipus diferents de sistemes de gestió de dades, que s'utilitzen per emmagatzemar i gestionar grans quantitats de dades. Les diferències entre els dos les trobam principalment en la seva finalitat, estructura i disseny.

Un *data warehouse* és un sistema de gestió de bases de dades dissenyat per gestionar dades estructurades d'una organització, que es volen utilitzar per a l'anàlisi de dades i la presa de decisions. Els *data warehouses* s'utilitzen per integrar dades de diverses fonts de dades de l'organització, que es volen transformar en informació útil per als usuaris de negoci, com ara analistes de dades i executius. Els *data warehouses* solen estar basats en una estructura de dades en estrella o de floc de neu, que s'organitzen al voltant d'una taula central de fets que es connecta a altres taules de dimensions. Ja vàrem comentar que un aspecte molt important dels *data warehouses* són els procesos d'ETL (extracció, transformació i càrrega - *extraction, transformation and load*) a través dels quals s'extreuen, netegen, filtren, validen i carreguen les dades en el magatzem de dades, a partir de les bases de dades operacionals de l'organització.

D'altra banda, un *data lake* és un dipòsit centralitzat de dades, que s'utilitza per emmagatzemar **dades estructurades, semiestructurades i no estructurades** de tot tipus, en el seu format original. Els *data lakes* són dissenyats per emmagatzemar dades **sense cap transformació prèvia**, i s'utilitzen per a l'anàlisi de dades a gran escala, amb eines d'analítica de dades com les de l'ecosistema Hadoop i Spark. Els *data lakes* no tenen una estructura predefinida, i les dades s'organitzen en funció de les necessitats dels usuaris.

Les diferències entre tots dos són les següents:

- Estructura: els *data warehouses* tenen una estructura predefinida, sovint basada en una taula de fets i taules de dimensions, mentre que els *data lakes* no tenen cap estructura predefinida, i les dades s'emmagatzemen tal com es recullen.
- Transformació de dades: els *data warehouses* requereixen que les dades s'hagin netejat, estructurat i transformades abans d'emmagatzemar-les, mentre que els *data lakes* s'emmagatzemen sense cap transformació prèvia.
- Finalitat: els *data warehouses* estan dissenyats per a l'anàlisi de dades empresarials i la presa de decisions, mentre que els *data lakes* estan dissenyats per a l'anàlisi de dades a gran escala i la ciència de dades.
- Eines i tecnologies: els *data warehouses* s'utilitzen amb eines d'ETL i tecnologies de base de dades tradicionals, mentre que els *data lakes* s'utilitzen amb tecnologies de Big Data com ara Hadoop o Spark, així com bases de dades NoSQL.

En resum, els *data warehouses* estan dissenyats per emmagatzemar i gestionar dades estructurades, netejades i transformades, mentre que els *data lakes* estan dissenyats per emmagatzemar dades de tot tipus, sense cap transformació prèvia, i s'utilitzen per a l'anàlisi de dades.

5. Apache Hive

Apache Hive es una infraestructura de codi obert per a *data warehousing*, que crea una capa que permet accedir a les dades massives emmagatzemades en un clúster Hadoop (principalment HDFS i HBase) mitjançant un llenguatge molt semblant a SQL, anomenat HiveQL. És a dir, Hive ens permet definir un *data warehouse* a partir de dades emmagatzemades en HDFS, i emprant una sintaxi com la de SQL, accedir a les dades per poder fer consultes interactives, elaborar informes, i demés funcionalitats de *business intelligence*.

Hem d'aclarir que les dades no estan emmagatzemades en Hive, ho estan en HDFS (o també pot emprar-se HBase, que veurem en el lliurament 4 de Sistemes de big data). Hive ens ofereix un nivell d'abstracció més elevat que ens proporciona la possibilitat d'accedir-hi d'una manera més senzilla i estructurada. Podríem dir que Hive converteix un *data lake* (un conjunt de dades emmagatzemades en un clúster Hadoop) en un *data warehouse* (un repositori de dades estructurades per fer informes i consultes interactives, en aquest cas fent servir un dialecte de SQL). Tot això en un entorn distribuït.

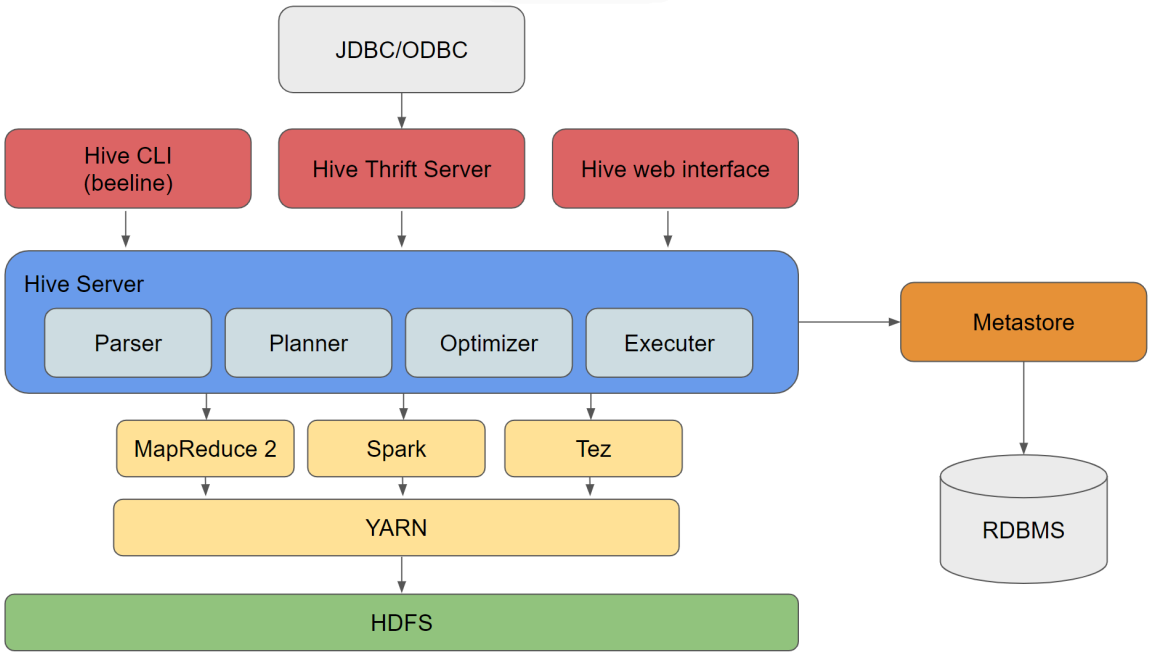
AMPLIACIÓ

Hive vol dir colmena i per això, el logo de Hive és l'elefant de Hadoop amb un cos d'abella amb franges grogues i negres.

Hive va ser desenvolupat originalment per Facebook en 2007 i posteriorment va passar a ser un projecte de codi obert d'Apache. Hive és utilitzat per grans companyies com Neflix o LinkedIn, a més de Facebook. Amazon també ha implementat una variant de Hive per a la seva plataforma de big data, Amazon Elastic MapReduce. I també veurem en el darrer lliurament com Hive es fa servir en Databricks, una plataforma unificada de dades massives en el nigul.

5.1. Arquitectura

La següent imatge mostra l'arquitectura de Hive:



Imatge: *Arquitectura de Hive*

En primer lloc, podem veure que hi ha tres tipus de **clients de Hive**: una interfície de línia d'ordres (CLI - *Command Line Interface*), que en les versions més modernes s'anomena beeline (podeu entrar-hi executant l'ordre *beeline* des de la consola); un client web; i un client Thrift que permet que aplicacions externes en facin ús, mitjançant queries JDBC/ODBC. Donarem més detalls sobre el client CLI i el client web en l'apartat següent.

AMPLIACIÓ

Apache Thrift és un llenguatge de definició d'interfícies i protocol de comunicació binari utilitzat per definir i crear serveis per a nombrosos llenguatges de programació.

Font: Wikipedia

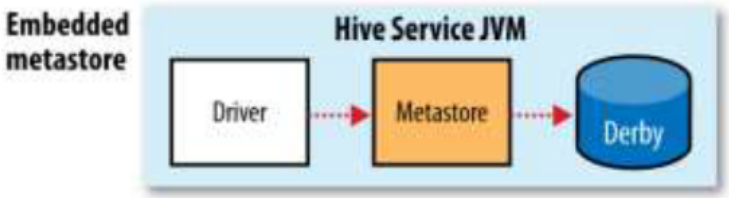
Les peticions d'aquest clients arriben al **servidor Hive** o **driver Hive**, que parseja la query, planifica i genera els treballs MapReduce corresponents i finalment els llença. Aquests treballs MapReduce (també es pot emprar Spark o Tez) finalment accedeixen a HDFS per recuperar les dades.

Un altre component fonamental de l'arquitectura Hive és el **metastore**. El metastore és el catàleg del sistema, emmagatzema les metadades necessàries per gestionar el magatzem de dades. És a dir, el metastore guarda la informació necessària sobre com les taules lògiques (i les seves columnes i particions) que configuren el *data warehouse* (i sobre les que podrem fer queries amb HiveQL) estan físicament emmagatzemades en directoris i arxius d'HDFS en un cluster Hadoop.

Per tant, quan una query arriba al servidor Hive, aquest primer de tot accedeix al metastore per obtenir l'estructura de la taula i la seva ubicació física, i després, amb aquesta informació genera i executa els treballs MapReduce per accedir a les dades.

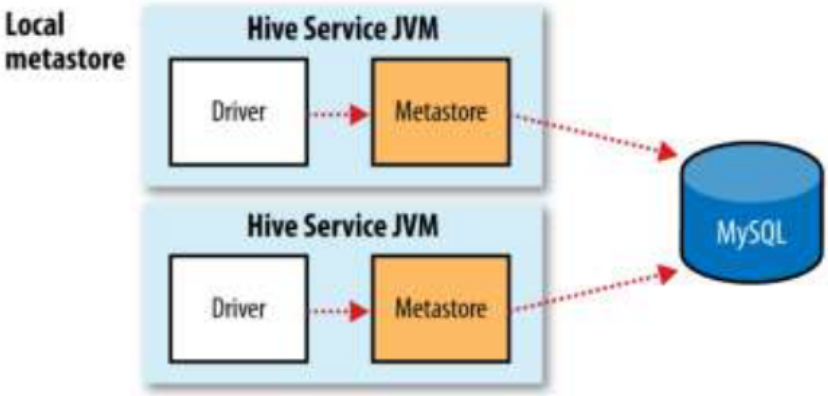
El metastore és un procés que s'executa fora de Hadoop i que sol fer servir una base de dades relacional (per exemple, MySQL) per emmagatzemar la informació del catàleg. Hi ha tres formes diferents d'organitzar aquest metastore.

1. Metastore incrustat: el metastore i el servidor Hive s'executen sobre la mateixa màquina virtual Java i utilitzen la base de dades incrustada Derby, que està disponible a totes les JVM. Aquest model només permet una única sessió a la vegada.



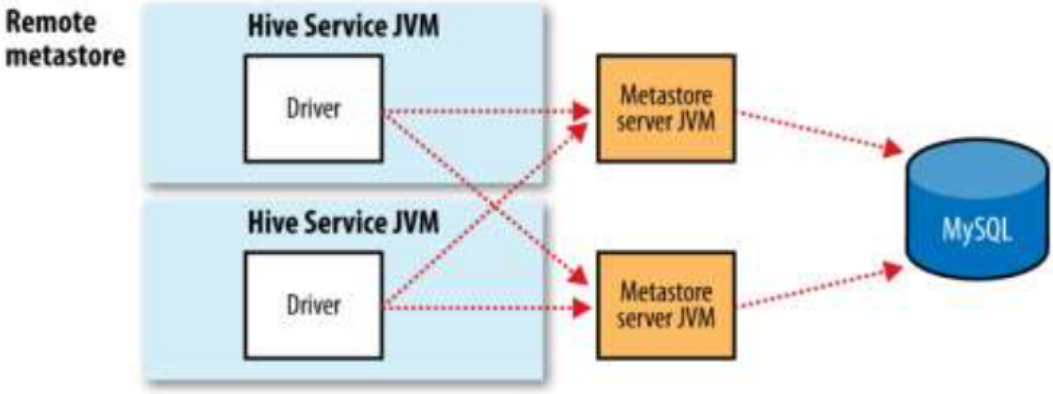
Imatge: *Metastore incrustat*

2. Metastore local: molts usuaris poden tenir moltes sessions actives a la vegada i en lloc d'emprar Derby, es fa servir qualsevol SGBD compatible amb JDBC, com MySQL, PostgreSQL o Oracle. El metastore i el servidor Hive s'executen sobre la mateixa màquina virtual Java.



Imatge: *Metastore local*

3. Metastore remot: el metastore s'executa en una JVM diferent del servidor Hive.



Imatge: *Metastore remot*

5.2. Clients Hive

Hem vist en l'apartat anterior que tenim tres tipus de clients en Hive. Un de línia d'ordres, un web i un de Thrift per integrar amb aplicacions. Ara veurem una mica més sobre els dos primers.

Client de línia d'ordres: beeline

Hive proporciona dos clients d'interfície de línia d'ordres (CLI). Al més antic s'accedeix mitjançant l'ordre *hive*. No obstant això, aquest client està *deprecated* i no es recomana el seu ús. És per això que ens centrarem en l'altre, **beeline**. Per entrar-hi des de la màquina virtual de Cloudera, executam l'odre:

```
beeline
```

I una vegada en el client, executam l'ordre següent per connectar-nos a la instància local del servidor Hive:

```
!connect jdbc:hive2://
```

Ens demanarà l'usuari i contrasenya, que per defecte són respectivament *hive* i *cloudera*.

També podem executar beeline connectant-nos directament a la instància local del servidor Hive, així:

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

A continuació podem executar queries HiveQL (veurem els detalls en els següents apartats), d'una manera semblant a com ho feim amb un terminal de SQL de MySQL, PostgreSQL o Oracle. També podem executar ordres específiques de beeline, que sempre comencen per ! . Abans ja hem vist l'ordre !connect, per connectar al servidor de Hive. Algunes de les ordres més utilitzades són:

- *!help* per mostrar la llista d'ordres de beeline i la seva explicació
- *!quit* per sortir de beeline
- *!run* per executar un script HiveQL. Per exemple: *!run /home/cloudera/hive/fitxer.hql*
- *!verbose* per activar el mode verbose, que permet mostrar més detalls quan s'executa una query

També podem executar beeline en mode batch. Amb el paràmetre -f feim que s'executi un fitxer amb sentències HiveQL:

```
beeline -u jdbc:hive2:// -f /home/cloudera/hive/fitxer.hql
```

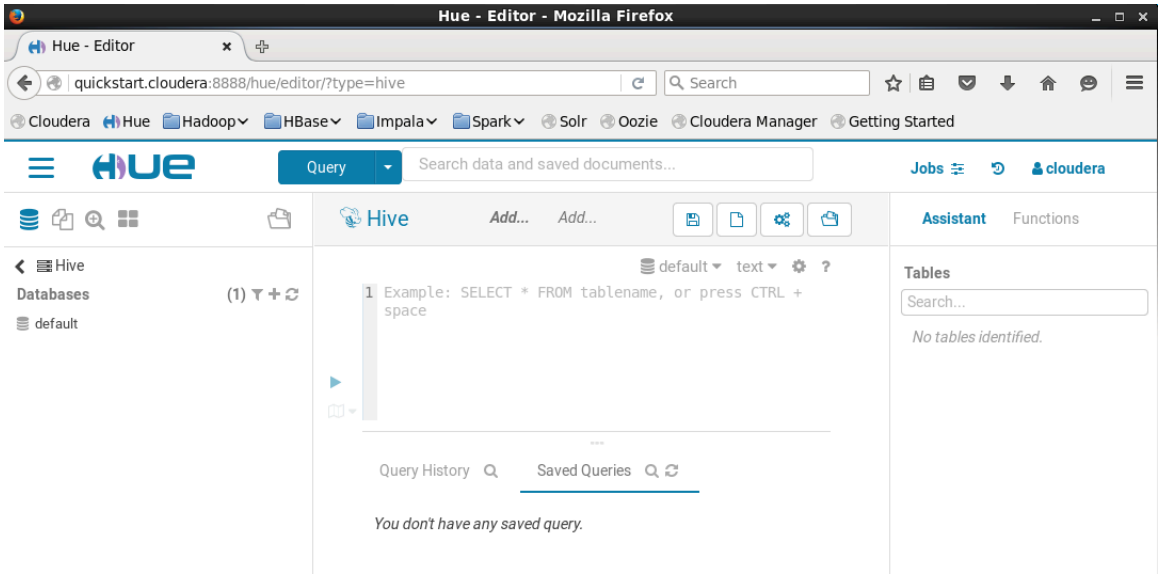
Amb el paràmetre -e podem executar directament una query HiveQL:

```
beeline -u jdbc:hive2:// -e 'SELECT * FROM nom_taula'
```

Client web: HUE

En el cas de les plataformes Cloudera (com la màquina virtual que estam emprant en el curs), el client web està integrat dins HUE (Hadoop User Experience), una eina, desenvolupada originalment per Cloudera i actualment de codi obert, que dona una interfície web per consultar bases de dades i magatzems de dades en un entorn Hadoop.

Per entrar a HUE amb la màquina virtual de Cloudera, trobarem un marcador dins del navegador Firefox. Podem emprar l'usuari *cloudera* amb contrasenya *cloudera*. Per defecte, HUE treballa amb l'editor d'Impala. Per canviar-ho a Hive, hem de fer clic sobre el desplegable que hi ha al botó Query i a l'opció Editor, seleccionar Hive.



Imatge: *HUE*

Una vegada configurat l'editor de Hive ja podem escriure les nostres queries en HiveQL.

Podem veure que HUE, a més de proporcionar un editor per a Hive i per a Impala, també en té per a Pig, Spark, MapReduce i Sqoop, entre d'altres.

5.3. HiveQL DDL

HiveQL, el llenguatge de consultes de Hive, proporciona també un DDL (*Data Definition Language*) per construir i mantenir les bases de dades i les taules, que també té moltes similituds amb el DDL de SQL.

Crear una base de dades

Per crear una nova base de dades podem emprar

```
CREATE DATABASE nom_bd;
```

O també, si volem evitar possibles errors si la base de dades ja existia:

```
CREATE DATABASE IF NOT EXISTS nom_bd;
```

ALERTA

Les sentències en HiveQL sempre acaben en punt i coma (;).

Per veure les bases de dades que tenim disponibles, podem executar:

```
show databases;
```

I per canviar quina és la base de dades activa, ho feim amb:

```
use nom_bd;
```

Esborrar una base de dades

De la mateixa manera, per esborrar una base de dades podem executar:

```
DROP DATABASE nom_bd;
```

O també, si volem evitar possibles errors si la base de dades no existia:

```
CREATE DATABASE IF EXISTS nom_bd;
```

IMPORTANT

Aquestes dues sentències poden suposar esborrar dades en HDFS.

Crear una taula

Per crear una taula tenim una sentència CREATE TABLE, mitjançant la qual definim les columnes que formen la taula, d'una manera semblant a la de SQL. D'altra banda, HiveQL utilitza una sèrie de clàusules per definir característiques de l'emmagatzematge en HDFS. La sintaxi general seria aquesta (hi ha altres clàusules que no veurem):

```
CREATE TABLE nom_taula( nom_columna1 TIPUS_DADES, nom_columna2 TIPUS_DADES, ...)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY 'caràcter_delimitador'
LINES TERMINATED BY 'caràcter_delimitador'
STOREDAS {TEXTFILE|SEQUENCEFILE|…}
LOCATION 'path_HDFS';
```

Veim que la primera línia, on definim el nom de la taula i les seves columnes és igual a SQL. A continuació especificam com s'emmagatzemen les dades ens els fitxers HDFS. Amb *ROW FORMAT* especificam com es guarden les files i les seves columnes. Hi ha dues maneres, *DELIMITED* (separats per caràcters delimitadors) o *SERDE* (SerDe, serialitzador/deserialitzador). En general, en aquest lliurament emprarem *DELIMITED*. Si empram *DELIMITED*, per defecte les files acaben amb el caràcter salt de línia ('\n') i les columnes amb tabulador ('\t') o coma (','). Si volem emprar altres caràcters ho podem especificar amb *FIELDS TERMINATED BY* i *LINES TERMINATED BY*.

Quan cream una taula, es crea un directori en HDFS. El directori HDFS de la base de dades per defecte (*default*) és */user/hive/warehouse*. Si cream una taula *nom_taula*, es crearà un subdirectori */user/hive/warehouse/nom_taula*, on s'aniran escrivint els seus fitxers de dades. Per altra banda, quan es crea una base de dades, també es crea un directori per a ella. Així, si per exemple cream una base de dades *nom_bd*, es crea el directori */user/hive/warehouse/nom_bd.db*. D'aquesta manera, quan es creï una taula dins aquesta base de dades, es crearà un subdirectori dins */user/hive/warehouse/nom_bd.db*. Per exemple: */user/hive/warehouse/nom_bd.db/nom_taula*.

Amb la clàusula *LOCATION* podem especificar un altre path HDFS per a la taula.

Per últim, amb *STOREDAS* podem especificar el tipus d'arxius que s'emprarà. Els valors permesos són:

- *TEXTFILE* (per defecte): un fitxer de text, és el tipus més bàsic. Utilitza caràcters delimitadors, és llegible per un humà i per qualsevol llenguatge de programació. El problema és que consumeix més espai (emmagatzemar un número mitjançant una cadena de caràcters requereix més bytes que amb la seva representació binària). I és difícil representar dades binàries.
- *SEQUENCEFILE*: emmagatzema parelles clau-valor en un format binari. És més eficient que el text i no és llegible per un humà.
- *AVRO*: fitxers Avro, sobre el qual ja hem parlat en lliuraments anteriors. Permet un emmagatzematge eficient, que permet incorporar l'esquema de les dades dins del fitxer. Pot ser llegit per molts llenguatges i és un format àmpliament utilitzat en l'ecosistema Hadoop (i també fora d'ell). Es considera que és la millor opció per a un emmagatzematge de propòsit general en Hadoop
- PARQUET: fitxers Parquet, un format columnar desenvolupat per Cloudera i Twitter i se li considera l'opció més eficient per afegir múltiples registres simultàniament
- *RCFILE*: un altre format columnar suportat per Hadoop (millor emprar Parquet)
- *ORC*: un altre format columnar suportat per Hadoop (millor emprar Parquet)
- *JSONFILE*: fitxers JSON
- *INPUTFORMAT classe_Java_format_entrada OUTPUTFORMAT classe_Java_format_sortida*: podem especificar les nostres pròpies classes Java per donar format d'entrada i sortida

En general, per als exemples d'aquest lliurament emprarem *TEXTFILE*, que és l'opció més senzilla i, a més, ens permet llegir el contingut.

Una vegada hem creat una taula, podem veure la seva estructura amb describe. Per exemple, la següent imatge mostra el resultat d'un describre d'una taula test, que s'ha creat mitjançant aquesta sentència:

```
CREATE TABLE test(id int, nom string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

```
0: jdbc:hive2://> describe test;
OK
+-----------+------------+----------+--+
| col_name  | data_type  | comment  |
+-----------+------------+----------+--+
| id        | int        |          |
| nom       | string     |          |
+-----------+------------+----------+--+
2 rows selected (0.832 seconds)
```

Imatge: *describe d'una taula test*

Primera fila de dades

Quan cream una taula i posteriorment volem carregar-hi un fitxer CSV (veurem com en l'apartat 3.5), podem configurar la taula perquè no tengui en compte la capçalera del fitxer (la primera fila). Per fer-ho, hem de configurar la propietat *skip.header.line.coun*t, donant-li el valor *1* (si la capçalera ocupa una fila). En la sentència *CREATE TABLE*, afegim *TBLPROPERTIES ("skip.header.line.count"="1")*. Quedaria així:

```
CREATE TABLE test(id int, nom string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
TBLPROPERTIES ("skip.header.line.count"="1");
```

També podem modificar la taula si ja l'havíem creada, mitjançant un ALTER TABLE:

```
ALTER TABLE test SET TBLPROPERTIES ("skip.header.line.count"="1");
```

CSV Serde

Hi ha situacions que amb el ROW FORMAT DELIMITED no podem importar correctament un fitxer CSV. El cas més habitual és si algun dels camps del CSV conté comes (i el text del camp està entre comentes) i s'ha emprat la coma també com a separador de camps. En aquests casos podem emprar un altre format de fila, CSV Serde, disponible des de la versió 0.14 de Hive. Ho escriuríem així:

```
CREATE TABLE test(id int, nom string)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde';
```

Per defecte, el format SERDE fa servir la coma com a separador entre camps, les cometes dobles per emmarcar un text que conté algun separador (la coma) i la barra invertida com a caràcter d'escapament. Si ho volem canviar, podem especificar-ho de la següent manera, on hem fixat el tabulador com a separador de camps, la comenta simple en lloc de la cometa doble per a emmarcar un text i dues barres invertides com a caràcter d'escapament:

```
CREATE TABLE test(id int, nom string)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
   "separatorChar" = "\t",
   "quoteChar"     = "'",
   "escapeChar"    = "\\"
)
```

No obstant això, un problema del format Serde és que totes les columnes de la taula passen a ser de tipus String, independentment de com ho hàgim definit en la sentència CREATE TABLE. El motiu és que els valors de cada columna són el resultat d'una des-serialització, que dona com a resultat una cadena de caràcters.

Podeu trobar més detalls a la documentació de Hive.

Taules internes i externes

En Hive tenim dos tipus de taules. A més de les taules que hem vist fins ara, anomenades internes, també en tenim les anomenades externes. En aquest cas, la taula es gestiona externament a Hive (les dades estan fora del directori HDFS de Hive), de manera que el que tenim és només una espècie d'enllaç (unes metadades al catàleg). D'aquesta manera, quan esborram una taula externa, no esborram les dades, només les metadades al catàleg. Per definir una taula com a externa, s'afegeix el prefix EXTERNAL en la seva creació:

```
CREATE EXTERNAL TABLE ...
```

Esborrar una taula

Podem esborrar una taula existent amb

```
DROP TABLE nom_taula;
```

AMPLIACIÓ

Podeu trobar una informació molt més completa sobre el DDL en la documentació oficial de Hive: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL

5.4. Tipus de dades en HiveQL

HiveQL proporciona uns tipus de dades primitius molt semblants als de SQL:

- Nombres enters: INT/TINYINT/SMALLINT/BIGINT
- Nombres amb decimals: FLOAT/DOUBLE
- Caràcters i cadenes de caràcters: STRING/CHAR/VARCHAR
- Dates i hores: DATE/TIMESTAMP
- Booleans: BOOLEAN

Però, a més, ofereix uns tipus de dades complexos:

- Structs. Per exemple: {a INT; b STRING}
- Maps. Per exemple: M['grup']
- Arrays. Per exemple: ['a', 'b', 'c']

També permet a l'usuari definir els seus propis tipus, amb estructures amb atributs (semblants als Structs).

Podeu trobar més detalls sobre el tipus de dades a la documentació de Hive.

5.5. Càrrega de dades

Una vegada hem creat una taula, podem carregar les dades des d'un path HDFS mitjançant la sentència LOAD:

```
LOAD DATA INPATH '/user/cloudera/proves/dades' INTO TABLE test;
```

En realitat això el que fa és moure les dades des d'un path de HDFS al path corresponent a la nostra taula (ja creada). Tot i que no és el més recomanable (millor emprar una sentència LOAD o bé utilitzar una eina d'ETL), també podem fer la importació de les dades directament executant una ordre d'HDFS per moure els arxius d'un path a un l'altre:

```
hdfs dfs –mv /user/cloudera/proves/dades /user/hive/warehouse/test
```

La sentència LOAD també ens permet carregar dades des d'un fitxer local (que no estigui en HDFS), afegint la paraula LOCAL:

```
LOAD DATA LOCAL INPATH '/home/cloudera/dades.csv' INTO TABLE test;
```

Per altra banda, existeixen eines d'ETL que ens permeten importar dades estructurades des d'una base de dades relacional. Una d'elles és **Sqoop**, disponible en la nostra màquina virtual de Cloudera. En concret, podem importar les dades a una taula Hive, emprant l'opció *-hive-import*. Per exemple:

```
sqoop import
 --connect jdbc:mysql://localhost/bd
 --username usuari
 --password contrasenya
 --table taula
 --hive-import
```

Les dades primer s'importaran des de la taula (o podria ser el resultat d'una query) de la base de dades relacional a un directori HDFS. A continuació Sqoop generarà una sentència de HiveQL amb l'operació de *CREATE TABLE* per definir l'estructura de les dades en el metastore. Per últim, Sqoop també generarà una sentència de tipus *LOAD DATA INPATH* per moure les dades al directori del *warehouse* corresponent.

Per defecte, la nova taula de Hive es crea dins la base de dades *default*. Podem emprar l'opció *--hive-table*, que ens permet especificar el nom de la taula en Hive, així com en quina base de dades volem que es creï. En el nom de la taula, podem especificar com a prefix el nom de la base de dades, separat per un punt. Per exemple, la següent opció fa que es creï la taula *taula1* en la base de dades *proves*:

```
--hive-table proves.taula1
```

Tot i que no apareix a la documentació, també es pot utilitzar l'opció *--hive-database* per especificar el nom de la base de dades:

```
--hive-database proves
```

En qualsevol cas, si la taula de Hive ja existia, podem utilitzar l'opció *--hive-overwrite* per indicar que la taula existent ha de ser reemplaçada. Alternativament, també podem emprar l'opció *--create-hive-table*, que donarà un error si la taula ja existeix (no la sobre-escriurà).

AMPLIACIÓ

Desafortunadament, tot i ser una eina molt utilitzada, Sqoop va deixar de mantenir-se en l'any 2021. En la terminologia d'Apache, va passar a l'àtic (*attic*), l'espai on es guarden els projectes que ja no estan actius.

Podeu consultar més informació a la guia d'usuari del projecte Scoop.

En concret, a l'apartat 7.2.13, hi trobareu més detalls sobre la importació en Hive, emprant Sqoop.

En la resta dels apunts, anam a fer feina amb una base de dades de MySQL, que ara carregarem dins del nostre magatzem de dades Hive. En concret, es tracta de la base de dades **retail_db**, una de les bases de dades d'exemple de MySQL, que ja està carregada en la nostra instància de MySQL en la màquina virtual de Cloudera.

Per obrir una sessió de Mysql, ho farem amb l'usuari **root** i contasenya **cloudera**.

```
mysql –u root –p
```

Si miram les bases de dades que tenim carregades, veurem que una d'elles és *retail_db*. Anam a fer feina amb aquesta base de dades.

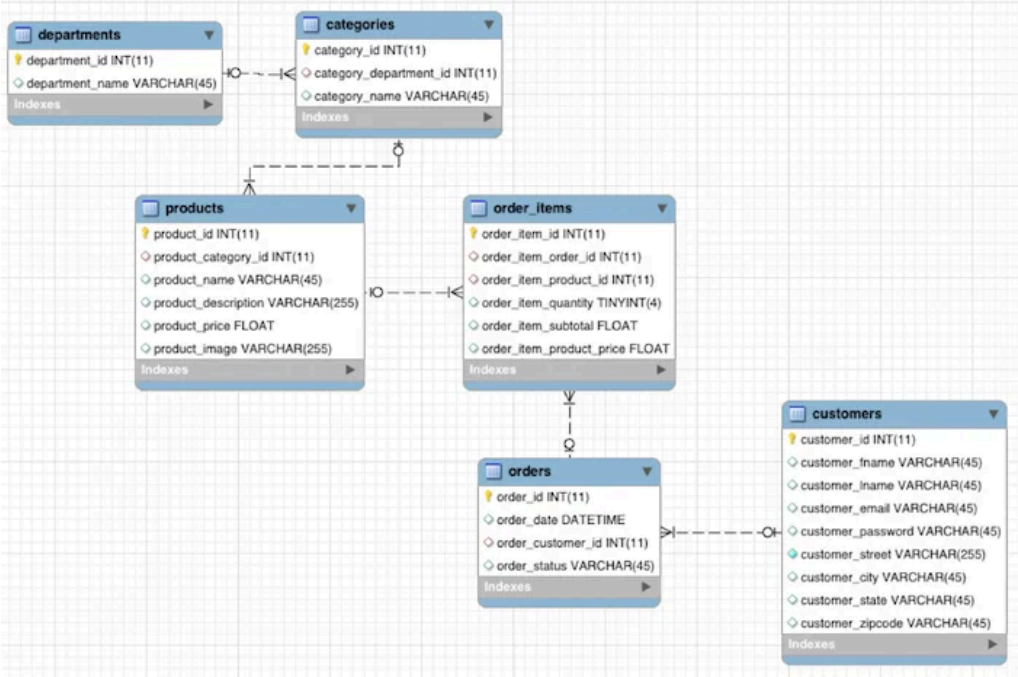
```
[cloudera@quickstart ~]$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 81
Server version: 5.1.73 Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| cm                 |
| firehose           |
| hue                |
| metastore          |
| mysql              |
| nav                |
| navms              |
| oozie              |
| retail_db          |
| rman               |
| sentry             |
+--------------------+
12 rows in set (0.00 sec)
```

Imatge: *Comprovació de les bases de dades MySQL*

Anam ara a crear un usuari (*cloudera* amb password *cloudera*) per poder connectar-nos-hi, en lloc de fer servir l'usuari *root*. Li donarem permisos sobre la base de dades retail_db:

```
CREATE USER 'cloudera'@'%' IDENTIFIED BY 'cloudera';
GRANT ALL PRIVILEGES ON retail_db.* TO 'cloudera'@'%' WITH GRANT OPTION;
FLUSH PRIVILEGES;
```

Aquesta base de dades té 6 taules per representar les vendes d'una tenda. Vegem-ne l'esquema:



Imatge: *Esquema de la base de dades retail_db*

Abans de carregar les dades a Hive, hi crearem una nova base de dades anomenada *proves* (dins Hive, ja sigui amb el client *beeline* o el de HUE):

```
CREATE DATABASE proves;
```

En concret, només treballarem amb les taules *products* i *categories*. Anem a importar-les en Hive, fent servir Sqoop:

```
sqoop import
 --connect jdbc:mysql://localhost/retail_db
 --username cloudera
 --password cloudera
 --table products
 --hive-import
 --hive-table proves.products
sqoop import
 --connect jdbc:mysql://localhost/retail_db
 --username cloudera
 --password cloudera
 --table categories
 --hive-import
 --hive-table proves.categories
```

5.6. Consultes en HiveQL

La sentència SELECT en HiveQL és pràcticament idèntica a la de SQL. Aquesta és la sintaxi general:

```
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]
```

La següent consulta ens recupera els productes amb un preu major que 1.000:

```
SELECT *
FROM products
WHERE product_price>1000;
```

Aquesta ens diu el preu màxim:

```
SELECT max(product_price)
FROM products;
```

I aquesta ens dona el número de productes per a cada codi de categoria::

```
SELECT product_category_id, count(product_category_id)
FROM products
GROUP BY product_category_id;
```

Com podem veure aquestes queries HiveQL són exactament iguals que en SQL.

AMPLIACIÓ Podeu trobar més informació sobre la sentència SELECT a la documentació de Hive.

També podem definir subqueries i joins, de la mateixa manera que en SQL. Per exemple, la següent query ens mostra el nom de producte i de categoria dels productes amb un preu superior a 1.000:

```
SELECT product_name, category_name
FROM products INNER JOIN categories ON product_category_id=category_id
WHERE product_price>1000;
```

AMPLIACIÓ Podeu trobar més detalls a la secció de joins i a la de subqueries de la documentació de Hive.

HiveQL incorpora també un bon conjunt d'operadors i funcions per utilitzar en les consultes. Podeu trobar una referència a la documentació de Hive. Però, a més, Hive permet definir a l'usuari noves funcions. Són les anomenades UDF (User Defined Functions) i UDAF (User Defined Aggregation Functions). El codi d'aquestes funcions es pot escriure en diversos llenguatges com ara Java o Python. No entrarem en més detalls en aquest curs, però val la pena saber que aquest és un mecanisme molt utilitzat per a fer transformacions de dades en Hive.

5.7. Insercions en HiveQL

HiveQL permet inserir valors en una taula, amb una sintaxi idèntica a la de SQL. Per exemple, seguint amb les taules products i categories de l'apartat anterior, podríem afegir una nova categoria 'Snow':

```
INSERT INTO categories VALUES (99,6,'Snow');
```

També podem inserir a una taula (ja existent) totes les files que ens retorna una query. Per exemple, volem inserir a la taula *categories_fitness*, les categories del departament 2:

```
INSERT TABLE categories_fitness2
 SELECT * FROM categories where category_department_id=2;
```

Això afegirà les files a la taula, sense esborrar les dades que pogués contenir la taula prèviament. Si, en canvi, volem que les dades prèvies s'esborrin, hem d'afegir la paraula *OVERWRITE*:

```
INSERT OVERWRITE TABLE categories_fitness
 SELECT * FROM categories where category_department_id=2;
```

5.8. Particionament i bucketing

Atès que les taules en un entorn de Big Data són molt grans, el seu particionament és un factor molt important en Hive.

Per defecte, totes les dades d'una taula s'emmagatzemen en un únic directori HDFS. Però podem especificar que la taula s'estructuri en diverses particions, de manera que cada una d'elles s'emmagatzemarà en un subdirectori diferent. Això permet que les queries s'executin de manera més ràpida, ja que no haurà d'accedir a les dades de tota la taula, només de la partició (o particions) necessàries.

Per tant, les particions són fragments horitzontals de dades que permeten que conjunts molt grans de dades es puguin separar en parts més manejables. Per crear una partició agafam de base una columna (o vàries). Cada possible valor de la columna (o columnes) seleccionada donarà lloc a una partició diferent: totes les files que tenguin un valor determinat de la columna (o columnes) de particionament, aniran a la mateixa partició.

El particionament es pot fer de manera estàtica o dinàmica.

Partició estàtica

En el particionament estàtic, afegim una columna de particionament, que no està dins les dades. Quan carreguem les dades, totes les files que carreguem de cop han d'anar a la mateixa partició. Normalment la columna de particionament és de tipus data i recull la data en que es fa la càrrega de les dades. D'aquesta manera, les dades que se carreguen avui aniran totes a una partició, les de demà a una altra, i així successivament.

Ho entendrem millor amb un exemple. Tenim una taula amb les vendes d'una tenda, que anam actualitzant cada vespre. El que farem és especificar que la columna de partició serà *data*, de tipus *date*.

```
CREATE TABLE vendes(
  id INT,
  nom_producte STRING,
  quantitat INT,
  preu_unitat FLOAT,
  departament STRING)
PARTITIONED BY(data DATE)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

Per fer el bolcat diari, tenim les dades de les vendes del darrer dia a un fitxer local */home/cloudera/vendes_darrer_dia.csv*. El contingut d'aquest fitxer per al darrer dia és aquest:

```
1,Cadira Bang,8,55.95,Saló
2,Làmpara Bing,10,20.05,Il·luminació
3,Taula Bong,4,199.99,Saló
```

Fixau-vos que en aquestes dades no hi ha la data. Quan feim la càrrega, hem d'especificar manualment la data de la partició:

```
LOAD DATA LOCAL INPATH '/home/cloudera/vendes_darrer_dia.csv' INTO TABLE vendes
PARTITION(data = '2023-02-22');
```

Les tres files que estam carregant conformaran la partició per al dia 2023-02-22. Es crearà, per tant, un subdirectori HDFS on aniran aquestes dades.

Particionament dinàmic

En el particionament dinàmic, les particions es creen automàticament quan es carreguen les dades. Noves particions es poden crear dinàmicament segons el valor de la darrera columna. Les particions prèviament existents es poden sobreescriure emprant la clàusula OVERWRITE.

Per permetre el particionament dinàmic, hem d'establir aquestes propietats de Hive:

```
SET hive.exec.dynamic.partition=true;
SET hive.exec.dynamic.partition.mode=nonstrict;
```

Vegem un exemple. Tenim la taula *vendes* sense particionament:

```
CREATE TABLE vendes(
  id INT,
  nom_producte STRING,
  quantitat INT,
  preu_unitat FLOAT,
  departament STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

I hi carregam les dades d'un fitxer que conté totes les vendes:

```
LOAD DATA LOCAL INPATH '/home/cloudera/totes_vendes.csv' INTO TABLE vendes;
```

Ara cream una segona taula vendes_per_departament, que és la que volem particionar:

```
CREATE TABLE vendes_per_departament(
  id INT,
  nom_producte STRING,
  quantitat INT,
  preu_unitat FLOAT)
PARTITIONED BY (departament STRING)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

I per últim, carregam les dades:

```
INSERT OVERWRITE TABLE vendes_per_departament
 PARTITION(departament)
 SELECT * FROM vendes;
```

Per veure les particions existents sobre una taula particionada, podem fer-ho amb la sentència:

```
SHOW PARTITIONS vendes_per_departament;
```

El resultat és que tenim dues particions, una per al departament Il·luminació i una altra per al de Saló:

```
+--------------------------+--+
|         partition        |
+--------------------------+--+
| departament=Il·luminació  |
| departament=Saló          |
+--------------------------+--+
```

Bucketing

Particionar una taula és útil quan aquesta és molt gran i llegir-la sencera requereix molt de temps. També és convenient que la majoria de queries facin un filtre per la columna (o columnes) que hem emprat per particionar. D'aquesta manera, la query només haurà de recuperar dades d'una (o unes poques) de les particions. A més, la columna que emprem per particionar ha de tenir un número raonable de possibles valors: ha de donar origen a un número raonable de particions. Si té molts valors possibles, tendrem massa particions. Si en té molt pocs (per exemple, 2), el benefici serà mínim.

De vegades, però, no tenim cap columna que sigui adequada per fer el particionament. En aquests casos, podem emprar la tècnica de *bucketing*. El concepte de *bucket* (poal o galleda en català, *cubo* en castellà), també anomenat en Hive *cluster*, és similar al de partició, però no ho fem pels valors únics d'una columna. En canvi,

especificarem en quants *buckets* volem organitzar la taula i automàticament cada fila s'anirà col·locant a un d'ells. Per decidir en quin *bucket* ha d'anar cada fila, s'utilitza una funció de hash.

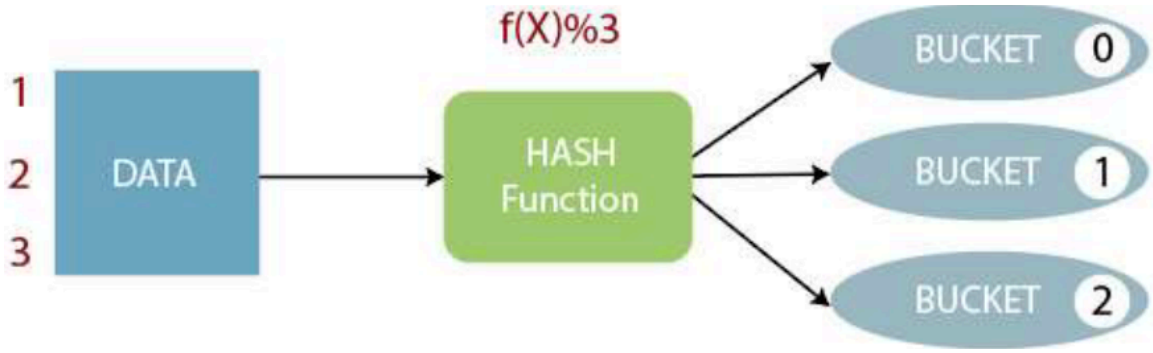
Per activar el bucketing hem de establir a *true* el valor de la propietat *hive.enforce.bucketing*:

```
SET hive.enforce.bucketing = true;
```

Vegem un exemple, amb una taula vendes (sense el camp departament), on especificam que treballi amb 3 buckets:

```
CREATE TABLE vendes(
  id INT,
  nom_producte STRING,
  quantitat INT,
  preu_unitat FLOAT)
CLUSTERED BY(id) INTO 3 BUCKETS
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

El que fa això és repartir cada fila, aplicant una funció de hash al seu id, a un dels tres *buckets* que hem definit. La funció de hash aquí és molt senzilla: com que el *id* és un número enter, simplement fa el mòdul 3 (residu de dividir per 3). El resultat d'aquesta operació li diu al servidor de Hive en quin *bucket* va cada fila.



Imatge: *Funció de hash per determinar el bucket d'una fila. Font: Universitat de Castella-La Manxa*

5.9. Eines relacionades amb Hive: Pig i Impala

Apache Pig

Apache Hive i Apache Pig són dos components molt usats de l'ecosistema Hadoop. A vegades es confonen ja que poden fer coses semblants. Totes dues tecnologies simplifiquen l'escriptura de programes MapReduce complexos, i estalvien als usuaris la corba d'aprenentatge que això suposa. Tots dos tenen un llenguate, HiveQL i Pig Latin, per escriure el codi de consulta a les dades. A partir d'aquest codi es generen els treballs MapReduce corresponents.

La diferència principal és que Apache Hive utilitza un llenguatge molt semblant a SQL, per la qual cosa és més fàcil d'aprendre per als professionals habituats a les bases de dades relacionals. És un llenguatge purament de consulta de dades, mentre que Pig Latin és un llenguatge de programació més complet i complex.

Així doncs, mentre que Pig s'utilitza principalment per a programar, Hive és més usat per analistes de dades en la creació d'informes.

Apache Impala

Apache Impala és una eina que veurem en detall en el pròxim lliurament i que també té molta relació amb Apache Hive. Impala és bàsicament un motor de SQL per fer processament paral·lel massiu (*Massive Parallel Processing*, MPP) sobre un clúster Hadoop. L'objectiu és, en conseqüència, bastant semblant: poder executar sentències SQL sobre dades que es troben en HDFS en un clúster Hadoop.

Hi ha, però, algunes diferències importants en la implementació i en la finalitat d'ambdues eines.

Pel que fa a la implementació, Hive està desenvolupat en Java, mentre que Impala ho està en C++, un llenguatge de més baix nivell i més eficient. A més, mentre que Hive està construït sobre MapReduce (el servidor Hive genera i llença treballs MapReduce), Impala no treballa amb MapReduce, cosa que també fa que sigui més ràpid. Així doncs, mentre que Impala és més ràpid, Hive funciona molt bé amb consultes llargues i complexes sobre conjunts de dades molt més grans.

Per tant, Hive i Impala són eines complementàries. Mentre que Impala està dissenyat per optimitzar la velocitat en un entorn de queries interactives, sense necessitat de moviment o transformació de les dades, Hive pretén executar queries complexes, amb transformacions i múltiples joins, en un entorn de processament *batch*, que sovint requereix de procesos ETL per preparar les dades.