

PR_06.2

Configuración Previa Sugerida

Para la práctica, se necesitarán al menos las siguientes tres tablas:

 **u_data** Calificaciones (UserID, MovieID, Rating, Timestamp).

 **u_user** Usuarios (UserID, Age, Gender, Occupation, ZipCode).

 **u_item** Películas (MovieID, MovieTitle, ReleaseDate, Genres...).

Nota: Presten especial atención a cómo cargar los datos, ya que los archivos MovieLens 100k usan diferentes delimitadores (barra vertical | y tabulador \t).

Tablas necesarias PELICULAS, USUARIOS, VOTACIONES

VOTOS: u.data2.txt : user id , movie id , rating , timestamp. (AHORA SEPARADO POR COMAS)

USUARIOS: - u.user2.txt: user id , age , gender , occupation | (AHORA SEPARADO POR COMAS)

PELICULAS: - u.item: movie id | movie title | release date + 19 columnas (SEPARADA POR TAB)



Ejercicios de Apache HiveQL con MovieLens

I. Gestión de Tablas y Formato de Datos (CREATE TABLE, EXTERNAL, ROW FORMAT)

Objetivo: Practicar la definición de esquemas, delimitadores y tipos de tablas.

Carga de Datos Delimitados (Básico):

- Crea una **Tabla Externa** llamada `ml_ratings_ext` para el archivo `u.data`.
- Asegúrate de especificar que los campos están delimitados por el carácter **tabulador** (\t) y que las filas terminan en salto de línea.
- Define las cuatro columnas: `user_id`, `movie_id`, `rating` INT y `timestamp` BIGINT.

```
hive> CREATE EXTERNAL TABLE IF NOT EXISTS ml_ratings_ext (
>     user_id INT,
>     movie_id INT,
>     rating INT,
>     times_stamp BIGINT
> )
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ','
> STORED AS TEXTFILE;
```

```
hive> LOAD DATA INPATH '/user/maria_dev/movielens/u.data2'
>         OVERWRITE INTO TABLE ml_ratings_ext;
Loading data to table movielens.ml_ratings_ext
```

Definición de Esquema y Comentarios:

- Crea una **Tabla Gestionada** (Managed) llamada `ml_users_managed` para el archivo `u.user`.
- Especifica el delimitador **barra vertical** (|). Mis archivos están delimitados por(,)
- Añade un `COMMENT` a la tabla y un `COMMENT` a la columna `occupation` explicando su uso.

```
hive> CREATE TABLE IF NOT EXISTS ml_users_managed (
>     user_id INT,
>     age INT,
>     gender STRING,
>     occupation STRING COMMENT 'Tipo de ocupación del usuario'
> )
> COMMENT 'Tabla de usuarios del dataset MovieLens'
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY ','
> STORED AS TEXTFILE;
```

```
hive> LOAD DATA INPATH '/user/maria_dev/movielens/u.user2'
>         OVERWRITE INTO TABLE ml_users_managed;
```

El comando `COMMENT` sirve para añadir anotaciones las tablas o a los encabezados de los datos.

Simulación de Pérdida de Datos (DROP TABLE)

- Ejecuta `DROP TABLE ml_ratings_ext;`. ¿Se eliminaron los datos de HDFS?
Explica por qué.

```
hive> DROP TABLE ml_ratings_ext;
```

No borra los datos almacenados en HDFS porque es **EXTERNAL**.

- Ejecuta `DROP TABLE ml_users_managed;`. ¿Se eliminaron los datos de HDFS?

Explica por qué.

```
hive> DROP TABLE ml_users_managed;
```

Sí borra los datos del HDFS porque es **MANAGED**.

TABLES 5			TABLE > USUARIOS
Search		COLUMNS	DDL STORAGE INFORMATION DETAILS
AUTHORIZATION			
		COLUMN NAME	COLUMN TYPE
peliculas			
votos			
usuarios		user_id	int
ml_items_managed		age	int
ml_user_info		gender	string
		occupation	string

II. Tipos de Datos Complejos (ARRAY, MAP, STRUCT)

Objetivo: Practicar la definición de tipos complejos y el uso de las funciones `size()`, `map_keys()` y el acceso mediante `[]` y `..`.

Uso de **STRUCT** (Información Personal):

Crea una nueva tabla `ml_user_info` basada en `u_user`.

Combina las columnas `age` `INT` y `gender` `STRING` en una sola columna de tipo **STRUCT** llamada `personal_data`.

```
hive> CREATE TABLE ml_user_info AS
> SELECT
>     user_id,
>     STRUCT(age, gender) AS personal_data,
>     occupation
> FROM usuarios;
```

```

-----+-----+-----+-----+-----+-----+-----+-----+
 VERTICES   STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
-----+-----+-----+-----+-----+-----+-----+-----+
Map 1 ..... SUCCEEDED    1        1        0        0        0        0
-----+-----+-----+-----+-----+-----+-----+-----+
VERTICES: 01/01  [=====>] 100% ELAPSED TIME: 4.05 s
-----+-----+-----+-----+-----+-----+-----+-----+
Moving data to directory hdfs://sandbox-hdp.hortonworks.com:8020/apps/hive/warehouse/movielens.db/ml_user_info
Table movielens.ml_user_info stats: [numFiles=1, numRows=943, totalSize=16970, rawDataSize=16027]
OK
Time taken: 0.622 seconds

```

Escribe una consulta para seleccionar el campo `gender` de esta nueva columna y filtra por todas las mujeres (`'F'`).

```

hive> SELECT user_id, personal_data.gender
> FROM ml_user_info
> WHERE personal_data.gender = "F";

```

Uso de ARRAY (Géneros de Película):

- El archivo `u.item` tiene 19 columnas binarias para géneros. Simula que ya han sido procesadas en una columna de tipo **ARRAY** llamada `genres_list` (`ARRAYSTRING`).
-

```

hive> CREATE TABLE IF NOT EXISTS ml_items_managed (
>     movie_id INT,
>     title STRING,
>     release_date STRING,
>     video_release_date STRING,
>     imdb_url STRING,
>     unknown INT,
>     action INT,
>     adventure INT,
>     animation INT,
>     childrens INT,
>     comedy INT,
>     crime INT,
>     documentary INT,
>     drama INT,
>     fantasy INT,
>     film_noir INT,
>     horror INT,
>     musical INT,
>     mystery INT,
>     romance INT,
>     sci_fi INT,
>     thriller INT,
>     war INT,
>     western INT
> )
> ROW FORMAT DELIMITED
> FIELDS TERMINATED BY '\t'
> STORED AS TEXTFILE;
OK
Time taken: 0.716 seconds

```

Escribe una consulta que devuelva el **tamaño** (`size()`) de esta lista de géneros para cada película.

```

hive> SELECT movie_id, size(genres_list)
> FROM ml_items_managed;

```

- Muestra el **segundo** género (índice `1`) de la película con `movie_id 50`.

```

hive> SELECT genres_list[1]
> FROM ml_items_managed
> WHERE movie_id = 50;

```

III. Consultas Básicas y JOINS

Objetivo: Practicar `SELECT`, `WHERE`, `GROUP BY`, `JOIN` y `LEFT SEMI JOIN`.

💡 Agregación Básica:

- Calcula el **promedio** de todas las calificaciones (`rating`) que existen en la tabla `ml_ratings_ext`

```
hive> SELECT AVG(rating) FROM ml_ratings_ext;
```

```
-----  
 VERTICES      STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED  
-----  
Map 1 ..... SUCCEEDED    1        1        0        0        0        0  
Reducer 2 .... SUCCEEDED    1        1        0        0        0        0  
-----  
VERTICES: 02/02  [=====>>] 100%  ELAPSED TIME: 3.96 s  
-----  
OK  
3.52986
```

💡 Filtrado y Agrupación:

- Agrupa los usuarios por `gender` y `occupation` (de `ml_users_managed`). Calcula el número de
- usuarios que hay en cada combinación (e.g., '`M', 'programmer'`, '`F', 'student'`).
- Ordena el resultado de forma descendente por el conteo.

```
hive> SELECT gender, occupation, COUNT(*) AS total  
> FROM ml_users_managed  
> GROUP BY gender, occupation  
> ORDER BY total DESC;
```

	VERTICES	STATUS	TOTAL	COMPLETED	RUNNING	PENDING	FAILED	KILLED
Map 1	SUCCEEDED	1	1	0	0	0	0	0
Reducer 2	SUCCEEDED	1	1	0	0	0	0	0
Reducer 3	SUCCEEDED	1	1	0	0	0	0	0
VERTICES: 03/03 [=====>>] 100% ELAPSED TIME: 4.97 s								
OK								
M	student	136						
M	other	69						
M	educator	69						
M	engineer	65						
F	student	60						
M	programmer	60						
M	administrator	43						
F	administrator	36						
F	other	36						
M	executive	29						
F	librarian	29						
M	scientist	28						
M	technician	26						
M	writer	26						
F	educator	26						
M	librarian	22						
F	writer	19						
M	entertainment	16						
M	marketing	16						
M	artist	15						
F	artist	13						

?? INNER JOIN??

- Utiliza un **INNER JOIN** entre `ml_ratings_ext` y `ml_users_managed` sobre `user_id`.
- Calcula la **calificación promedio** por `occupation`.

```
hive> SELECT u.occupation, AVG(r.rating)
> FROM ml_ratings_ext r
> JOIN ml_users_managed u ON r.user_id = u.user_id
> GROUP BY u.occupation
> ORDER BY 2 DESC;
```

VERTICES	STATUS	TOTAL	COMPLETED	RUNNING	PENDING	FAILED	KILLED
Map 1	SUCCEEDED	1	1	0	0	0	0
Map 4	SUCCEEDED	1	1	0	0	0	0
Reducer 2	SUCCEEDED	1	1	0	0	0	0
Reducer 3	SUCCEEDED	1	1	0	0	0	0
VERTICES: 04/04 [=====>>] 100% ELAPSED TIME: 16.36 s							
OK							
administrator	3.6356464768017114						
artist	3.653379549393414						
doctor	3.688888888888889						
educator	3.6706206312221985						
engineer	3.541406727828746						
entertainment	3.4410501193317424						
executive	3.3491037320011756						
healthcare	2.896219686162625						
homemaker	3.301003344481605						
lawyer	3.7353159851301116						
librarian	3.560781338896264						
marketing	3.4856410256410255						
none	3.779134295227525						
other	3.5523773797242804						
programmer	3.5682604794257147						
retired	3.4667495338719703						
salesman	3.582943925233645						
scientist	3.611273080660836						
student	3.5151432345038027						
technician	3.5322304620650313						
writer	3.3757225433526012						

?? LEFT OUTER JOIN??

- Crea una tabla ficticia `unrated_users` que contenga solo 5 `user_id`s de la tabla `ml_users_managed` que *no* hayan calificado ninguna película.
- Realiza un `LEFT OUTER JOIN` desde `ml_users_managed` a `ml_ratings_ext`.
- Filtra el resultado para encontrar aquellos usuarios que **no tienen calificaciones** (donde el `rating` es `NULL` después del `join`).

```
hive> SELECT u.user_id
  > FROM ml_users_managed u
  > LEFT JOIN ml_ratings_ext r ON u.user_id = r.user_id
  > WHERE r.rating IS NULL;
Query ID = maria_dev_20251204084722_6e873f5b-1d5a-4144-b92a-626cc53f0f8f
Total jobs = 1
Launching Job 1 out of 1
```

```

Status: Running (Executing on YARN cluster with App id application_1764836683577_0001)

-----
      VERTICES    STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
-----
Map 1 ..... SUCCEEDED     1        1        0        0        0        0
Map 2 ..... SUCCEEDED     1        1        0        0        0        0
-----
VERTICES: 02/02  [=====>>] 100%  ELAPSED TIME: 5.18 s
-----
OK
Time taken: 7.774 seconds
-----
```

?? LEFT SEMI JOIN ??Simulación de IN/EXISTS??

- Usando `ml_ratings_ext` y `ml_users_managed`, lista todos los `user_id` que tienen una ocupación de 'programmer' y que han calificado al menos una película. Debe usarse `LEFT SEMI JOIN`, no un `INNER JOIN simple`.

```

hive> SELECT u.user_id
>   FROM ml_users_managed u
>   LEFT SEMI JOIN ml_ratings_ext r
>   ON u.user_id = r.user_id
>   WHERE u.occupation = 'programmer';
Query ID = maria_dev_20251204084801_bf982ef6-113b-4a8d-85a2-468a28d09b8b
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1764836683577_0001)
```

```

-----
      VERTICES    STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
-----
Map 1 ..... SUCCEEDED     1        1        0        0        0        0
Map 2 ..... SUCCEEDED     1        1        0        0        0        0
-----
VERTICES: 02/02  [=====>>] 100%  ELAPSED TIME: 5.67 s
-----
OK
17
29
45
53
55
58
82
102
114
119
134
141
144
160
177
182
200
215
219
222
263
279
283
285
292
300
-----
```