

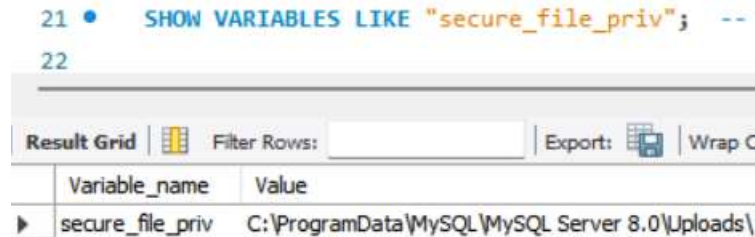
Sprint 4: Modelado SQL

Radostin Pavlov

NIVEL 1

Descarga los archivos CSV, estudialos y diseña una base de datos con un esquema de estrella que contenga, al menos 4 tablas de las que puedas realizar las siguientes consultas...

Empezamos creando las tablas e importando los datos desde los archivos csv. Para poder importar los datos de los archivos, hay que comprobar en qué carpeta hay que colocarlos para poder leerlos:



Después se crea la base de datos y todas las tablas. Las longitudes de los campos se ajustan aproximadamente a las de los datos, dejando cierto margen donde procede para nuevos datos. Se asegura que los tipos de las claves principales sean iguales que los de las claves foráneas para poder unir las tablas después de crearlas. Las claves foráneas se especifican tras crear las tablas para evitar errores por no poder establecer las referencias si las tablas correspondientes aun no se han creado.

```
create database ventas;
```

```
use ventas;
```

```
CREATE TABLE companies (  
    company_id VARCHAR(10) PRIMARY KEY,  
    company_name VARCHAR(40),  
    phone VARCHAR(20),  
    email VARCHAR(50),  
    country VARCHAR(60),  
    website VARCHAR(50)  
);
```

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/companies.csv'  
INTO TABLE companies  
FIELDS TERMINATED BY ','  
LINES TERMINATED BY '\n'  
IGNORE 1 ROWS;
```

```
CREATE TABLE credit_cards (  
    id VARCHAR(10) PRIMARY KEY,  
    user_id INT,  
    iban VARCHAR(40),  
    pan VARCHAR(30),  
    pin VARCHAR(10),
```

```
    cvv VARCHAR(5),
    track1 VARCHAR(50),
    track2 VARCHAR(50),
    expiring_date VARCHAR(10)
);
```

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/credit_cards.csv'
INTO TABLE credit_cards
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

```
CREATE TABLE products (
    id INT PRIMARY KEY,
    product_name VARCHAR(40),
    price VARCHAR(10),
    colour VARCHAR(10),
    weight VARCHAR(5),
    warehouse_id VARCHAR(10)
);
```

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/products.csv'
INTO TABLE products
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

```
CREATE TABLE transactions (
    id VARCHAR(45) PRIMARY KEY,
    card_id VARCHAR(10),
    business_id VARCHAR(10),
    timestamp TIMESTAMP,
    amount DECIMAL(10, 2),
    declined TINYINT,
    product_ids VARCHAR(15),
    user_id INT,
    lat DECIMAL(15, 11),
    longitude DECIMAL(15, 11)
);
```

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/transactions.csv'
INTO TABLE transactions
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

```
CREATE TABLE users (
```

```

id INT PRIMARY KEY,
name TINYTEXT,
surname TINYTEXT,
phone VARCHAR(20),
email VARCHAR(40),
birth_date VARCHAR(15),
country TINYTEXT,
city TINYTEXT,
postal_code VARCHAR(11),
address VARCHAR(40)
);

```

Al tratar de importar los datos correspondientes a la tabla users se encuentran errores, al parecer porque sql no lee bien los separadores. Para solventarlo, las comas usadas como separadores se sustituyen por semicolons, empleando el siguiente código de Python:

```

import pandas as pd
input_file = 'users_usa.csv'
output_file = 'users_usa-2.csv'
df = pd.read_csv(input_file, sep=',', quotechar='"')
df.to_csv(output_file, sep=';', index=False)

```

Esto se hace con cada uno de los tres archivos que van a la tabla usuarios. En los nombres de los archivos generados se incluye la extensión -2 para distinguirlos de los originales. A continuación se insertan los datos de los tres archivos en la misma tabla users ya que lo que varía básicamente es el país que aparece en el campo 'country'.

```

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users_ca-2.csv'

```

```

INTO TABLE users
FIELDS TERMINATED BY ';'
-- ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

```

```

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users_uk-2.csv'

```

```

-- El archivo original daba problemas, de modo que fue transformado con python
-- sustituyendo el delimitador coma por semicolon y eliminando comillas
INTO TABLE users
FIELDS TERMINATED BY ';'
-- ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

```

```

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users_usa-2.csv'

```

```

-- El archivo original daba problemas, de modo que fue transformado con python
-- sustituyendo el delimitador coma por semicolon y eliminando comillas
INTO TABLE users
FIELDS TERMINATED BY ';'
-- ENCLOSED BY '"'
LINES TERMINATED BY '\n'

```

IGNORE 1 ROWS;

A continuación se genera una tabla llamada products_transactions conteniendo las claves transaction_id y product_id de las tablas products y transactions para romper la relación n:n entre ellas. (Esto corresponde al Nivel 3, pero lo hacemos aquí y así ya completamos el modelo). La tabla se genera mediante Python (Se podría hacer por SQL pero es más complicado). El código usado se puede ver en el archivo 'id-s-to-table.ipynb' incluido en el repositorio. Básicamente consiste en crear con `str.split()` una tabla con una columna por cada valor de la lista del campo product_ids, insertar los transaction_id en la tabla con `insert()` y pivotar los datos de los product id-s colocándolos en una sola columna con `melt()`. El dataframe se exporta como 'products_transactions.csv' que luego se importa en la tabla products_transactions creada.

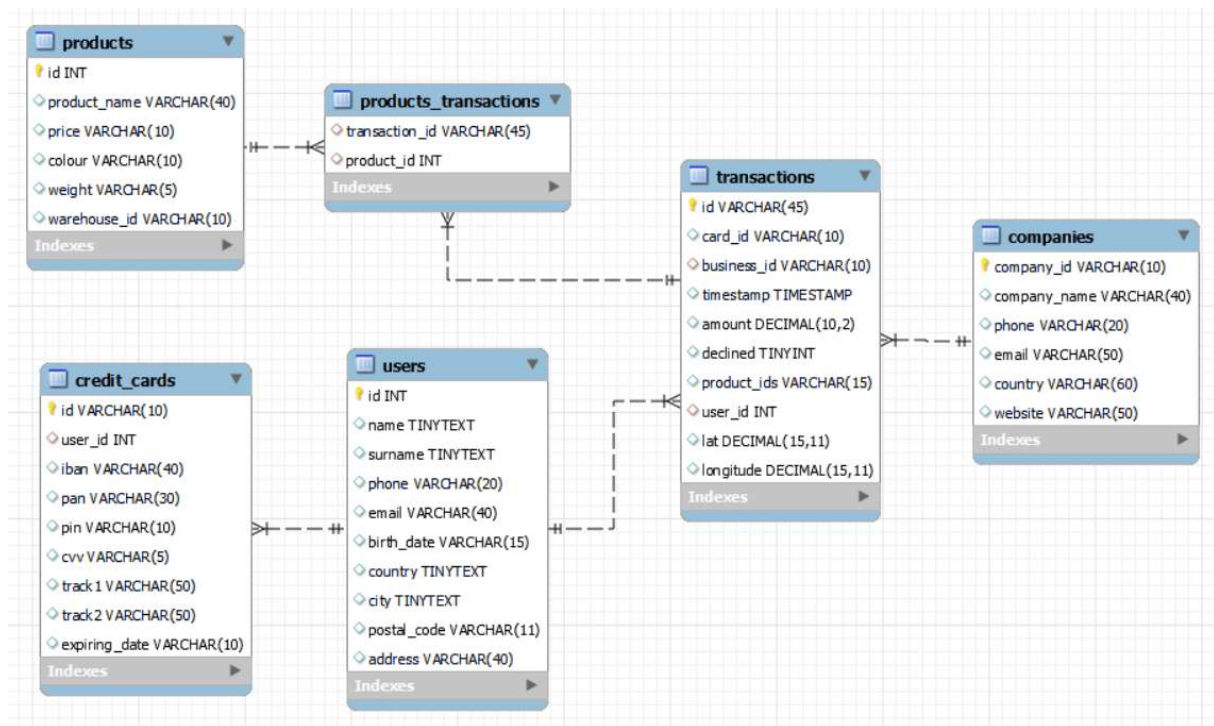
```
CREATE TABLE products_transactions (  
  transaction_id VARCHAR(45),  
  product_id INT  
);
```

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server  
8.0/Uploads/products_transactions.csv'  
INTO TABLE products_transactions  
FIELDS TERMINATED BY ','  
LINES TERMINATED BY '\n'  
IGNORE 1 ROWS;
```

A continuación se crean las claves foráneas:

```
ALTER TABLE credit_cards ADD FOREIGN KEY(user_id) REFERENCES users(id);  
ALTER TABLE transactions ADD FOREIGN KEY (user_id) REFERENCES users(id);  
ALTER TABLE transactions ADD FOREIGN KEY (business_id) REFERENCES  
companies(company_id);  
ALTER TABLE products_transactions ADD FOREIGN KEY (transaction_id) REFERENCES  
transactions(id);  
ALTER TABLE products_transactions ADD FOREIGN KEY (product_id) REFERENCES  
products(id);
```

Una vez creadas las tablas y sus relaciones se genera el modelo con la función Reverse Engineering de MySQL Woorkbench:



Se comprueba que todas las relaciones son correctas y se procede a realizar las consultas solicitadas.

```

159 -- Ejercicio 1: Realiza una subconsulta que muestre a todos los usuarios con más de 30 transacciones utilizando al menos 2 tablas.
160 • SELECT
161     name as Nombre, surname as Apellido, COUNT(transactions.id) as Transacciones
162 FROM
163     transactions
164 JOIN
165     users ON users.id = transactions.user_id
166 GROUP BY user_id
167 HAVING Transacciones > 30
168 ORDER BY Transacciones DESC;
  
```

Nombre	Apellido	Transacciones
Hedwig	Gilbert	76
Ocean	Nelson	52
Kenyon	Hartman	48
Lynn	Riddle	39

```

171 -- Ejercicio 2: Muestra el promedio de la suma de transacciones por IBAN de las tarjetas de crédito en la compañía Donec Ltd. utilizando al menos 2 tablas.
172
173 • SELECT
174     company_name AS Compañía,
175     IBAN,
176     round(AVG(amount), 2) AS Promedio,
177     COUNT(transactions.id) AS Transacciones
178 FROM
179     transactions
180 JOIN
181     credit_cards ON credit_cards.id = transactions.card_id
182 JOIN
183     companies ON companies.company_id = transactions.business_id
184 WHERE
185     company_name = 'Donec Ltd'
186 GROUP BY iban
187 ORDER BY Transacciones DESC;
  
```

Compañía	IBAN	Promedio	Transacciones
Donec Ltd	PT87806228135092429456346	203.72	2

NIVEL 2

- Crea una tabla nueva que refleje el estado de las tarjetas de crédito en base a si
- las últimas tres transacciones fueron declinadas y genera la siguiente consulta:
- Ejercicio 1: Cuántas tarjetas están activas

Para resolver este ejercicio se han creado dos queries preliminares que luego se han combinado en una sola:

```
196 -- Query preliminar 1: devuelve una view con las últimas 3 transacciones
197 • create view ultimas_compras as (
198   SELECT t.card_id, t.timestamp, t.declined
199   FROM transactions t
200   WHERE (
201     SELECT COUNT(*)
202     FROM transactions t2
203     WHERE t2.card_id = t.card_id AND t2.timestamp > t.timestamp
204   ) < 3
205   ORDER BY t.card_id, t.timestamp DESC);
206
207 -- Query preliminar 2: muestra el estado de las tarjetas como 'Inactiva' si la suma de los últimos tres valores de declined es 3
208 • select card_id, sum(declined), count(card_id), case
209   when sum(declined) = 3 then 'Inactiva'
210   else 'Activa'
211   end as Estado
212   from ultimas_compras
213   group by card_id
214   order by sum(declined) desc;
```

```
216 -- Esta query combina las dos anteriores para generar la tabla deseada
217 • CREATE VIEW estado_tarjetas AS
218   SELECT
219     card_id,
220     SUM(declined) AS total_declined,
221     COUNT(card_id) AS total_transactions,
222     CASE
223       WHEN SUM(declined) = 3 THEN 'Inactiva'
224       ELSE 'Activa'
225     END AS Estado
226   FROM (
227     SELECT
228       t.card_id,
229       t.timestamp,
230       t.declined
231     FROM transactions t
232     WHERE (
233       SELECT COUNT(*)
234       FROM transactions t2
235       WHERE t2.card_id = t.card_id AND t2.timestamp > t.timestamp
236     ) < 3
237   ) AS ult_compras
238   GROUP BY card_id;
```

```
240 • SELECT card_id as Tarjeta, total_declined as 'Num. rechazos ult. 3 trans.', Estado FROM ventas.estado_tarjetas;
241
```

Result Grid	Filter Rows:	Exports:	Wrap Cell Contents:
Tarjeta	Num. rechazos ult. 3 trans.	Estado	
CdU-2987	1	Activa	
CdU-3743	0	Activa	
CdU-3225	1	Activa	
CdU-3071	1	Activa	
CdU-4359	0	Activa	
CdU-3141	1	Activa	
CdU-3309	1	Activa	
CdU-3435	1	Activa	
CdU-3701	0	Activa	
CdU-3155	1	Activa	
CdU-4366	0	Activa	
CdU-3407	1	Activa	
CdU-4520	0	Activa	
CdU-3197	1	Activa	

```
240 -- Ejercicio 1: Cuantas tarjetas están activas
241 • select count(*) from estado_tarjetas where Estado = 'Activa';
```

Result Grid	Filter Rows:	Exports:	Wrap Cell Content:
count(*)			
275			

Se comprueba que este número corresponde a la totalidad de las tarjetas registradas:


```
245 • select count(id) from credit_cards;
246
```

Result Grid	Filter Rows:	Exports:
count(id)		
275		


```

247 # NIVEL 3
248 -- Crea una tabla con la que podamos unir los datos del nuevo archivo products.csv con la base de datos creada,
249 -- teniendo en cuenta que desde transaction tienes product_ids. Genera la siguiente consulta: (Se ha hecho antes)
250 -- Ejercicio 1: Necesitamos conocer el número de veces que se ha vendido cada producto.
251 • SELECT
252     product_name AS Producto, COUNT(transactions.id) AS Ventas
253 FROM
254     products
255     JOIN
256     products_transactions ON products_transactions.product_id = products.id
257     JOIN
258     transactions ON transactions.id = products_transactions.transaction_id
259 GROUP BY product_name
260 ORDER BY Ventas DESC;

```

Result Grid |   Filter Rows: | Export:  | Wrap Cell Content: 

Producto	Ventas
Direwolf Stannis	106
skywalker ewok	100
Winterfell	68
riverlands north	68
Direwolf riverlands the	66
duel	65
Tarly Stark	65
Tully	62
jinn Winterfell	61
skywalker ewok sith	61
palpatine chewbacca	60
kingsblood Littlefinger...	58
duel tourney	57

+