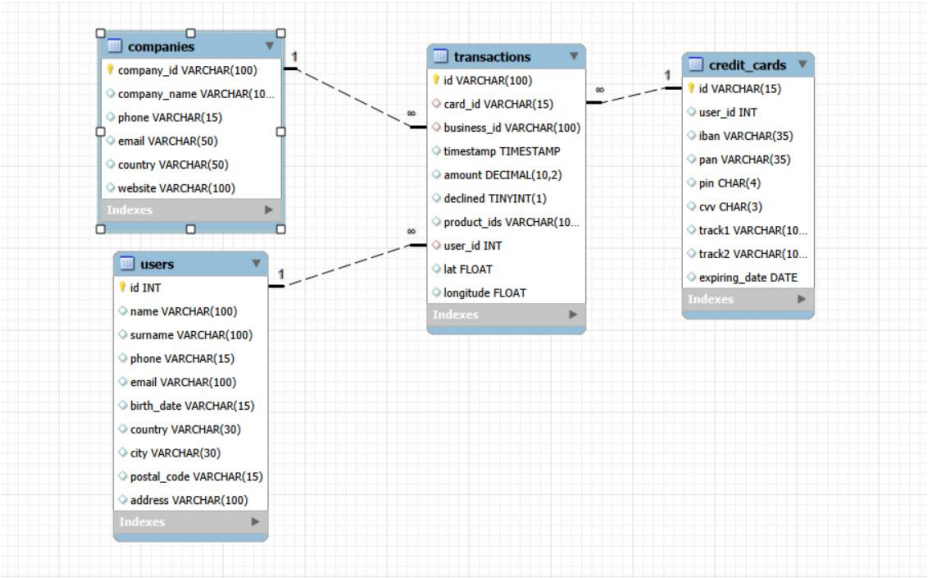
Nivel 1

Descarga los archivos CSV, estudiales 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:



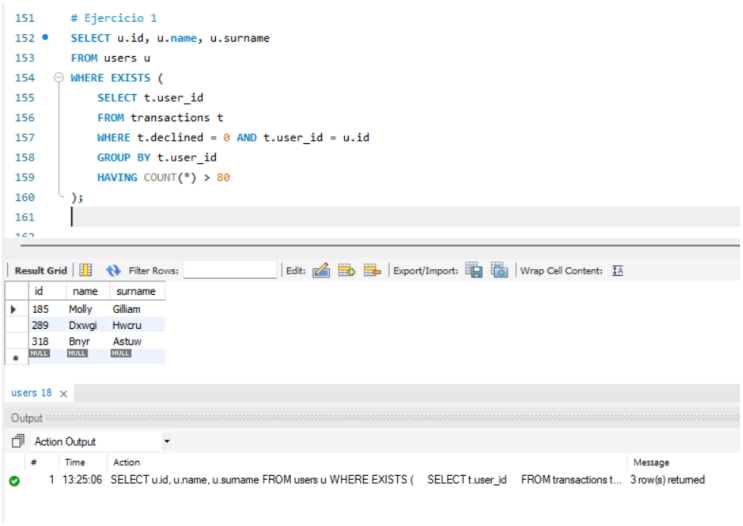
Nota: Se diseño un modelo de estrella en el que la tabla transactions actúa como tabla de hechos, ya que tiene información que se va a analizar.

La tabla users se creó a partir de american_users y european_users, para unir la información en una sola tabla, se aplicó una operación de UNION ALL, posteriormente se eliminaron las tablas originales que ya no eran necesarias.

Las tablas users, credit_cards y companies funcionan como tablas de dimensiones, dando contexto a cada transacción. Las relaciones se establecen mediante llaves foráneas busisnes_id, user_id y card_id en la tabla transactions.

Ejercicio 1

Realiza una subconsulta que muestre a todos los usuarios con más de 80 transacciones utilizando al menos 2 tablas.



```
151    # Ejercicio 1
152    SELECT u.id, u.name, u.surname
153    FROM users u
154    WHERE EXISTS (
155        SELECT t.user_id
156        FROM transactions t
157        WHERE t.declined = 0 AND t.user_id = u.id
158        GROUP BY t.user_id
159        HAVING COUNT(*) > 80
160    );
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	name	surname
185	Molly	Gilliam
289	Dxwgi	Hwcru
318	Bnyr	Astuw
NULL	NULL	NULL

users 18

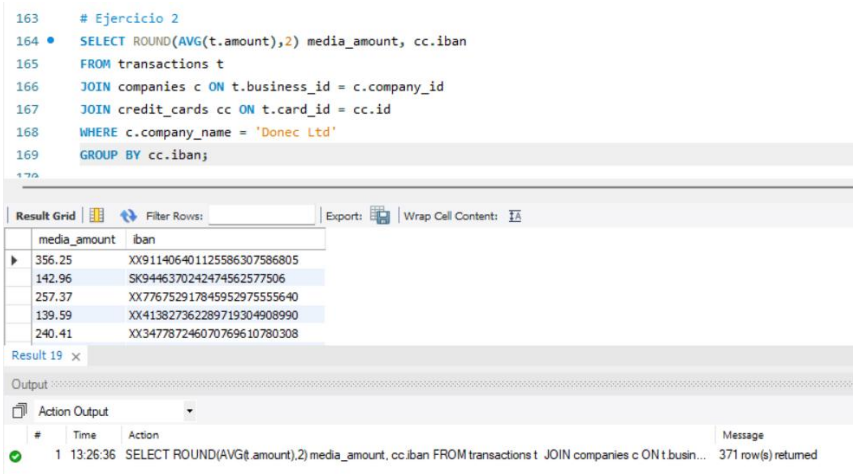
Output

Action Output

#	Time	Action	Message
1	13:25:06	SELECT u.id, u.name, u.surname FROM users u WHERE EXISTS (SELECT t.user_id FROM transactions t...	3 row(s) returned

Ejercicio 2

Muestra la media de amount por IBAN de las tarjetas de crédito en la compañía Donec Ltd., utiliza por lo menos 2 tablas.



```
163    # Ejercicio 2
164    SELECT ROUND(AVG(t.amount),2) media_amount, cc.iban
165    FROM transactions t
166    JOIN companies c ON t.business_id = c.company_id
167    JOIN credit_cards cc ON t.card_id = cc.id
168    WHERE c.company_name = 'Donec Ltd'
169    GROUP BY cc.iban;
```

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

media_amount	iban
356.25	XX9114064011255863075868O5
142.96	SK9446370242474562577506
257.37	XX776752917845952975555640
139.59	XX413827362289719304908990
240.41	XX347787246070769610780308

Result 19

Output

Action Output

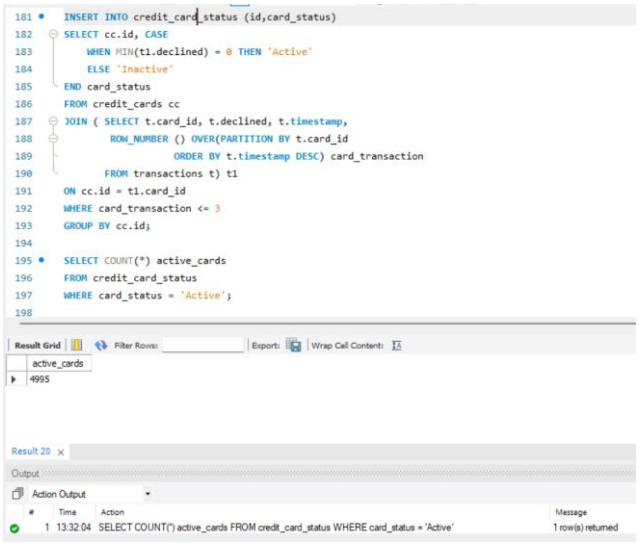
#	Time	Action	Message
1	13:26:36	SELECT ROUND(AVG(t.amount),2) media_amount, cc.iban FROM transactions t JOIN companies c ON t.busin...	371 row(s) returned

Nivel 2

Crea una nueva tabla que refleje el estado de las tarjetas de crédito basado en si las tres últimas transacciones han sido declinadas entonces es inactivo, si al menos una no es rechazada entonces es activo. Partiendo de esta tabla responde:

Ejercicio 1

¿Cuántas tarjetas están activas?



Nota: Se creó la tabla credit_card_status uniéndola al modelo mediante la llave foránea id, que hace referencia al card_id en transactions. Los datos se introdujeron a partir de una tabla calculada que determina el estado de cada tarjeta (activa o inactiva).

El cálculo se basa en las últimas 3 transacciones de cada tarjeta, se utilizó una función de ventana con ROW_NUMBER() para enumerar las transacciones por tarjeta según la fecha, PARTITION BY card_id para agrupar las transacciones por tarjeta, WHERE limita el análisis a las tres más recientes.

Finalmente, se aplicó MIN(declined) dentro de cada grupo donde indica que si el mínimo es 0, significa que la tarjeta esta activa y si el mínimo es 1, significa que la tarjeta esta inactiva.

Nivel 3

Crea una tabla con la que podamos unir los datos del nuevo archivo products.csv con la base de datos creada, teniendo en cuenta que desde transaction tienes product_ids. Genera la siguiente consulta:

```sql
-- Tabla products_transactions
CREATE TABLE IF NOT EXISTS products_transactions (
    product_id VARCHAR(100),
    transaction_id VARCHAR(100),
    PRIMARY KEY (product_id, transaction_id),
    FOREIGN KEY (product_id) REFERENCES products(id),
    FOREIGN KEY (transaction_id) REFERENCES transactions(id)
);

-- Datos products_transactions
INSERT INTO products_transactions (transaction_id, product_id)
SELECT t.id,
       TRIM(jt.product_id) AS product_id
FROM transactions AS t
JOIN JSON_TABLE(
       CONCAT('["', REPLACE(REPLACE(t.product_ids, ' ', ''), ',', '","'), '"]'),
       '$[*]' COLUMNS (
           product_id VARCHAR(100) PATH '$'
       )
) AS jt
JOIN products p
  ON p.id = TRIM(jt.product_id);
```

Output

Action Output

	#	Time	Action	Message
✔	39	14:28:58	CREATE TABLE IF NOT EXISTS products_transactions (product_id VARCHAR(100), transaction_id VA...	0 row(s) affected
✔	40	14:29:03	INSERT INTO products_transactions (transaction_id, product_id) SELECT t.id, TRIM(jt.product_id) AS pro...	253391 row(s) affected Records: 253391 Duplicates: 0 Warnings: 0

Nota: Se crea la tabla products. Para relacionarla con transactions se define una tabla intermedia llamada products_transactions para normalizar la relación muchos a muchos (una transacción puede incluir varios productos). En esta tabla se establece una clave primaria compuesta (product_id, transaction_id). Los datos se insertan transformando la lista de product_ids: con REPLACE y CONCAT se construye un array JSON válido que la función JSON_TABLE separa en filas individuales (un producto por transacción). Además, se aplica TRIM para eliminar espacios en blanco y asegurar que los identificadores coincidan correctamente con la tabla products.

Ejercicio 1

Necesitamos conocer el número de veces que se ha vendido cada producto.

```sql
# Ejercicio 1
SELECT product_id, COUNT(transaction_id) times_sold
FROM products_transactions
GROUP BY product_id;
```

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

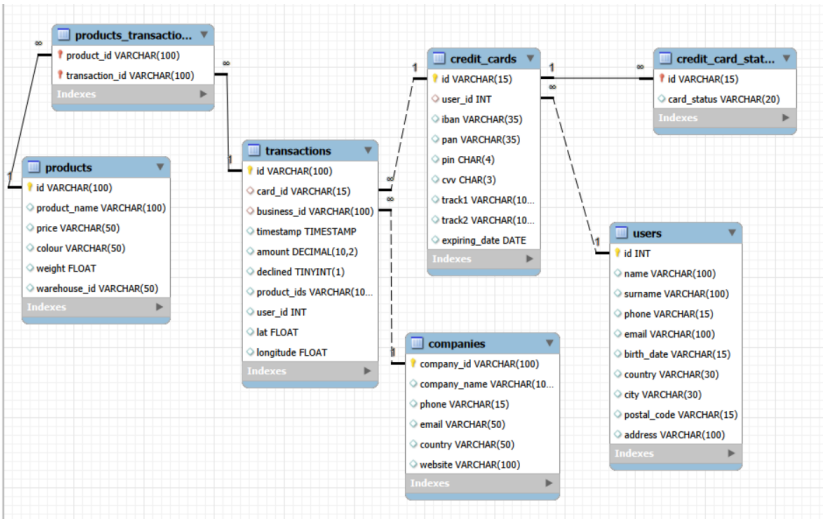
product_id	times_sold
1	2468
10	2571
100	2517
11	2573
12	2558

Result 22

Output

Action Output

	#	Time	Action	Message
✔	1	13:39:42	SELECT product_id, COUNT(transaction_id) times_sold FROM products_transactions GROUP BY product_id	100 row(s) returned



Nota: En la estructura final se implementa un modelo de copo de nieve, ya que las tablas de dimensiones están normalizadas y se complementan con otras tablas (por ejemplo, credit_card_status extiende la dimensión credit_cards). Se eliminó la llave foránea directa entre users y transactions, ya que la relación se establece indirectamente a través de la tabla credit_cards, dado que cada transacción se realiza con una tarjeta de crédito asociada a un usuario.