# Tarea S4.01. Creación de Bases de Datos

**Nivel 1**

**Descarga los archivos CSV, estúdialos 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:**

A continuación mostramos la base de datos con las tablas creadas y el diagrama con esquema de estrella. Para crear la base de datos simplemente usamos CREATE DATABASE con el nombre de la base de datos y luego escribimos USE y el mismo nombre de la base de datos con la que vamos a operar.

```
#NIVEL 1
#Crear Base de datos
CREATE DATABASE transactions_db;
USE transactions_db;
```

Para crear las tablas siempre empezaremos usando la cláusula CREATE TABLE IF NOT EXISTS y el nombre de la tabla. Empezamos con las tablas que contengan solamente claves primarias. La primera será users. Escribimos los nombres de las variables y le asignamos un tipo de dato. Para datos de carácter alfanumérico que pueden variar en longitud o espacio, usamos VARCHAR( ) y asignamos un número que representa este espacio o longitud que ocupan los datos. Asegurémonos que el número es suficientemente largo ya que si el número de caracteres de una fila supera esta cifra, nos dará error. Si queremos evitar ocupar mucho espacio, también podemos ajustar a una cifra que sabemos no se superará en ninguna fila. La primera variable suele ser una clave primaria id. Si sabemos que hay variables que todas las filas tienen la misma longitud, usamos CHAR(longitud de caracteres). Nos aseguramos siempre de acabar cada tabla con “ );” .

```sql
# Tabla users
CREATE TABLE IF NOT EXISTS users (
    id VARCHAR(3) PRIMARY KEY,
    name VARCHAR(50),
    surname VARCHAR(20),
    phone VARCHAR(20),
    email VARCHAR(50),
    birth_date VARCHAR(20),
    country VARCHAR(20),
    city VARCHAR(30),
    postal_code VARCHAR(25),
    address VARCHAR(50)
);
```

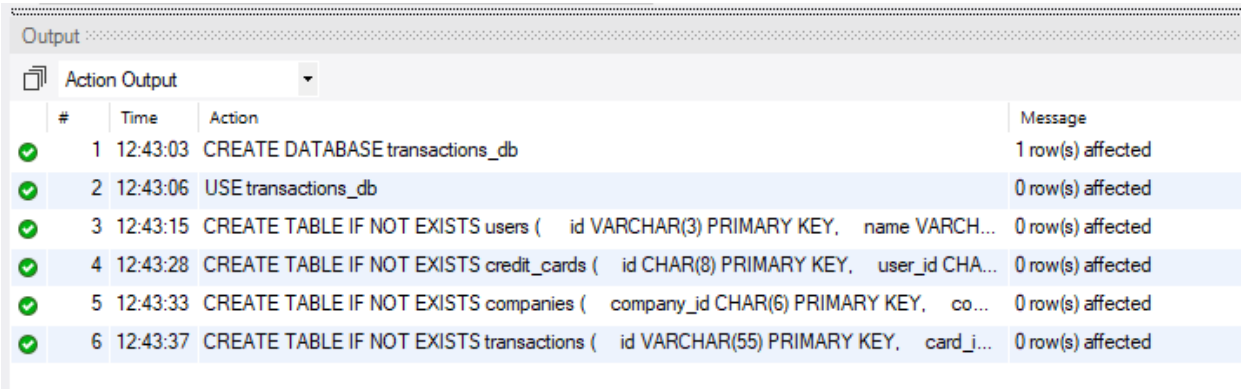
La siguiente tabla, credit_cards, contiene una clave foránea al final. La variable user_id hacer referencia a la id de la tabla users, de modo que usamos FOREIGN KEY (user_id) REFERENCES users(id) para que queden relacionadas ambas tablas.

```sql
# Tabla credit_cards
CREATE TABLE IF NOT EXISTS credit_cards (
    id CHAR(8) PRIMARY KEY,
    user_id CHAR(3),
    iban VARCHAR(34),
    pan VARCHAR(19),
    pin CHAR(4),
    cvv CHAR(3),
    track1 VARCHAR(50),
    track2 VARCHAR(50),
    expiring_date VARCHAR(10),
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

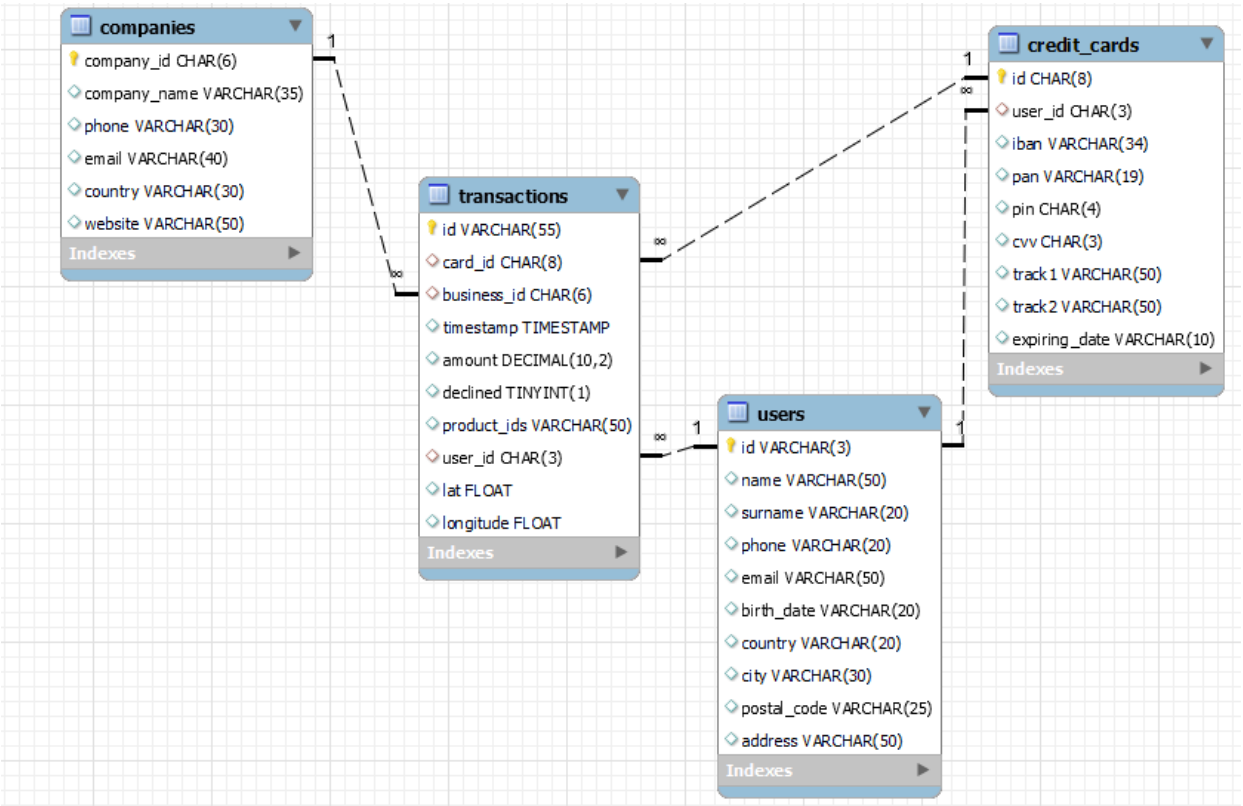
```sql
# Tabla companies
CREATE TABLE IF NOT EXISTS companies (
    company_id CHAR(6) PRIMARY KEY,
    company_name VARCHAR(35),
    phone VARCHAR(30),
    email VARCHAR(40),
    country VARCHAR(30),
    website VARCHAR(50)
);
```

En la tabla transactions, incluimos 3 claves foráneas, las variables id de cada una de las 3 tablas creadas anteriormente. Así todas estarán relacionadas y formarán el esquema de estrella requerido. Veremos que algunas variables contienen tipos de dato distintos como timestamp para representar fechas y horas, la variable declined será booleana con 1s y 0s para representar si la transacción está declinada o no y variables de coordenadas como lat y longitud con float o amount con DECIMAL para representar valores con decimales.

```sql
# Tabla transactions
CREATE TABLE IF NOT EXISTS transactions (
    id VARCHAR(55) PRIMARY KEY,
    card_id CHAR(8),
    business_id CHAR(6),
    timestamp TIMESTAMP,
    amount DECIMAL(10, 2),
    declined BOOLEAN,
    product_ids VARCHAR(50),
    user_id CHAR(3),
    lat FLOAT,
    longitude FLOAT,
    FOREIGN KEY (card_id) REFERENCES credit_cards(id),
    FOREIGN KEY (business_id) REFERENCES companies(company_id),
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

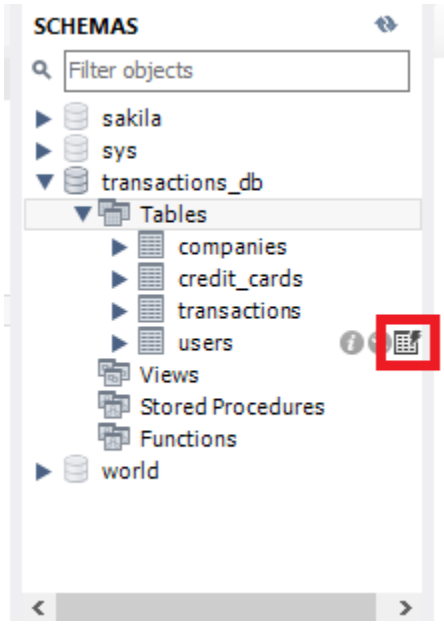


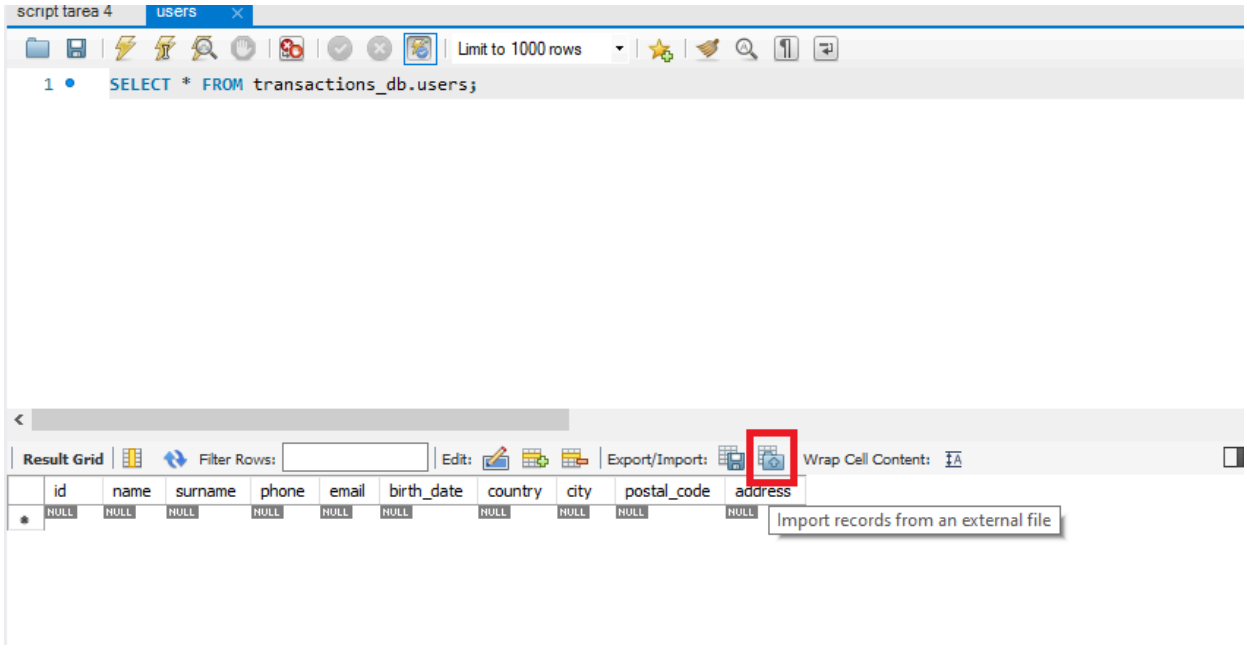

Una vez ejecutamos cada tabla y nos aseguramos que están correctamente creadas, vamos a database, reverse engineer y hacemos click en next, seleccionamos la base de datos transactions_db y seguimos haciendo click en next/finish hasta que nos aparezca el diagrama. Veremos que las relaciones son de N:1 entre transactions y el resto de tablas. Lo que significa que un usuario, una tarjeta de crédito o una compañía pueden tener varias transacciones. Entre users y credit_cards también hay una relación de 1:N, un usuario puede tener varias tarjetas de crédito.



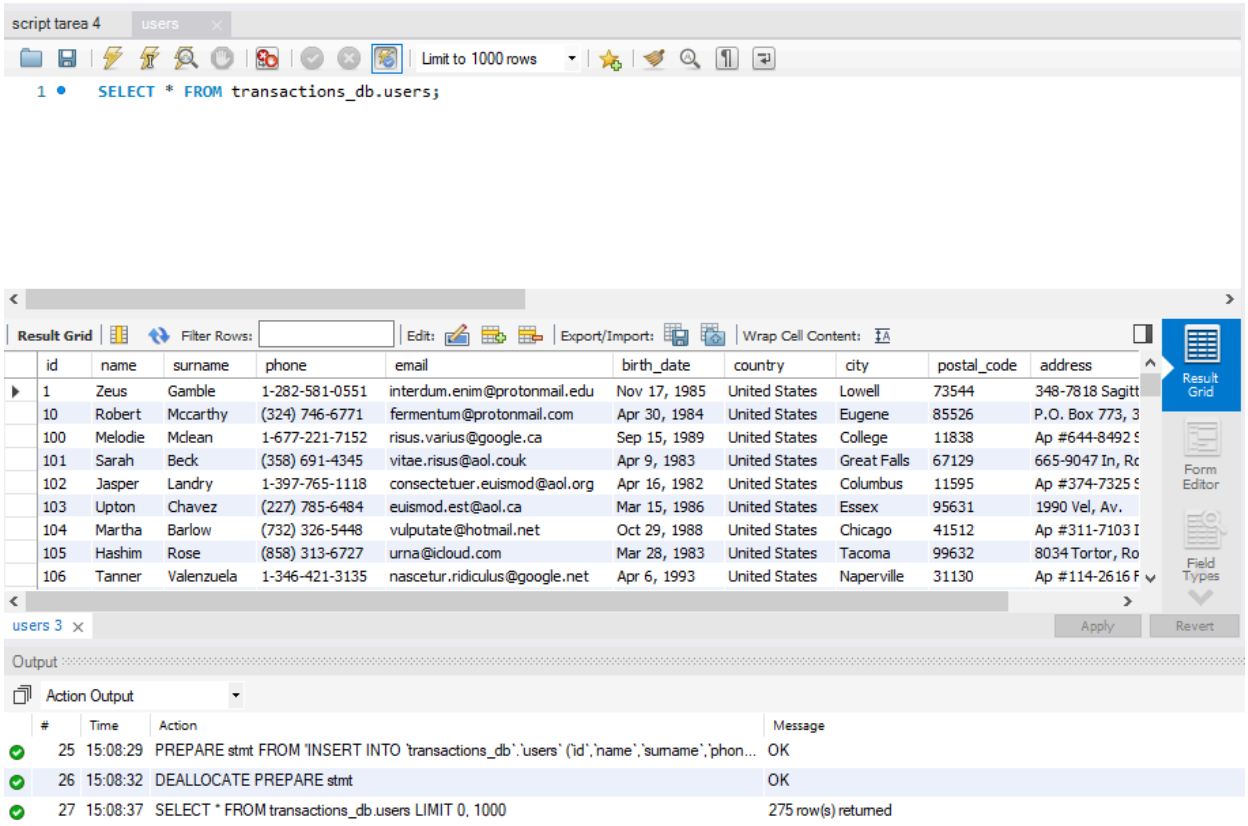

Una vez tenemos la base de datos, las tablas y el esquema, importamos los datos de los archivos csv disponibles en el enunciado. Si hacemos click en la tabla que aparece a la derecha de cada

tabla creada en los Schemas con las distintas bases de datos y click en import records from an external file, podemos coger la ubicación del archivo csv con los datos e importarlos en la tabla correspondiente. Hay que prestar atención en la parte donde seleccionamos la tabla donde se introducirán los datos, si nos equivocamos de nombre de tabla, se importarán en el lugar equivocado.



Aquí vemos cómo se han importado los valores para la tabla users. Al tener 3 archivos csv, comprobamos con excel que el número total de filas que contiene la tabla en Workbench es el mismo que el total de filas de los 3 archivos csv de 'users_usa', 'users_uk' y 'users_ca'.

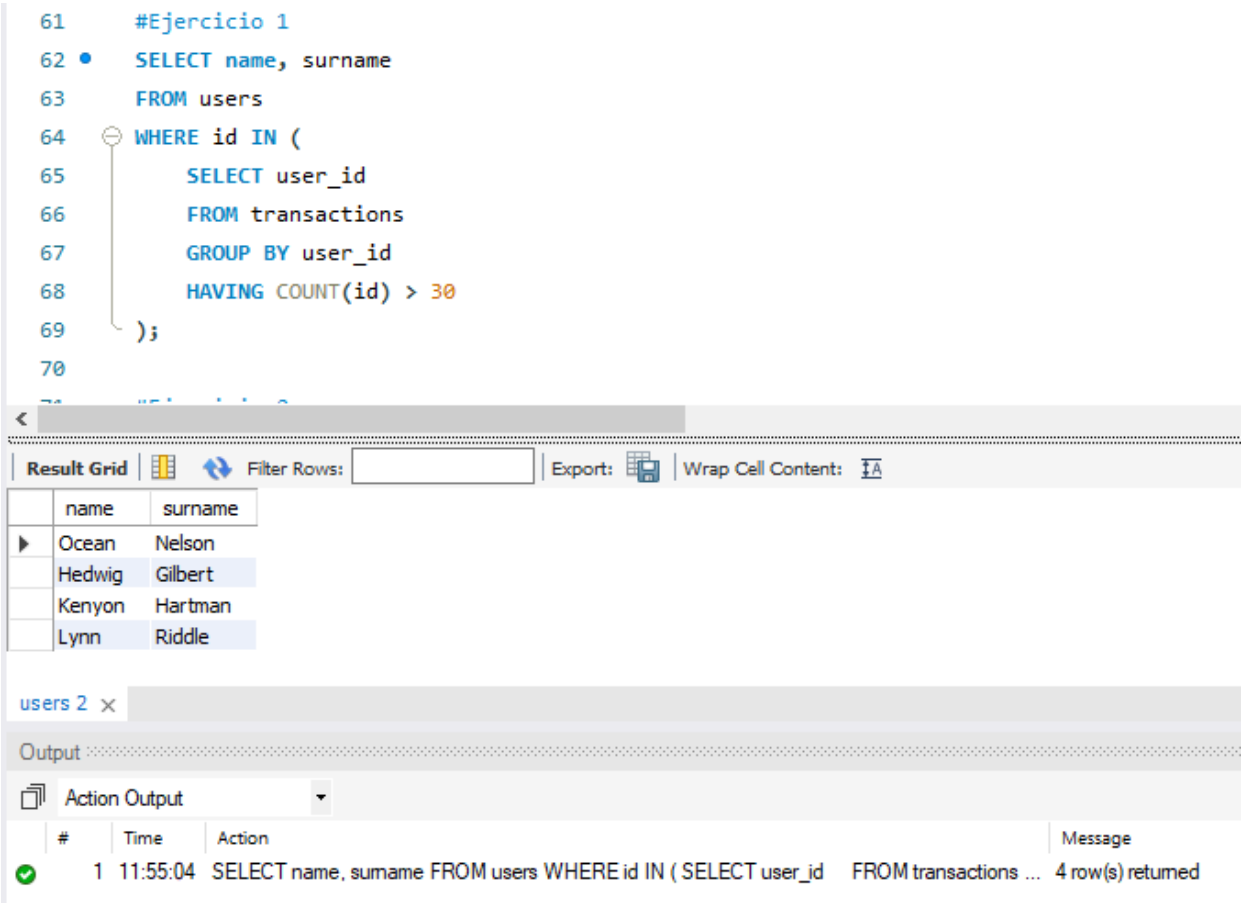



Hacemos exactamente lo mismo para el resto de tablas con sus respectivos datos importados de los archivos csv, proporcionados por el enunciado.

**Ejercicio 1**

**Realiza una subconsulta que muestre a todos los usuarios con más de 30 transacciones utilizando al menos 2 tablas.**

Para realizar dicha consulta, seleccionamos el nombre y apellido de la tabla usuarios con el criterio WHERE id IN con la subconsulta que nos seleccione la id del usuario de la tabla transactions agrupado por id de usuario y usando la cláusula HAVING, contamos por id con COUNT(id) el número de transacciones que han realizado. Si queremos seleccionar solamente aquellos usuarios que hayan realizado más de 30, ponemos > 30 al final.




**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.**

Para este ejercicio seleccionamos variables usando alias ya que utilizaremos JOINS para unir la tabla transactions con credit_cards y companies por medio de las ids de éstas. Seleccionamos de la tabla credit cards (alias cc) la variable iban, para calcular el importe promedio, usamos ROUND(AVG(t.amount, 2) para obtener el cálculo redondeado a 2 decimales, lo renombramos todo con el alias avg_amount. Ponemos en la cláusula FROM la variable con las claves foráneas transactions y hacemos el doble JOIN de credit_cards y transactions por medio de cc.id = t.card_id y companies con transactions por medio de t.business_id = c.company_id. Usamos el criterio WHERE con el nombre de la empresa 'Donec Ltd' para ver la media de importe de las tarjetas de crédito de dicha compañía. Agrupamos por la variable de credit_cards iban.

```sql
#Ejercicio 2
SELECT cc.iban, ROUND(AVG(t.amount), 2) AS avg_amount
FROM transactions t
JOIN credit_cards cc ON cc.id = t.card_id
JOIN companies c ON t.business_id = c.company_id
WHERE c.company_name = 'Donec Ltd'
GROUP BY cc.iban;
```

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

| iban | avg_amount |
|---|---|
| PT87806228135092429456346 | 203.72 |

Result 4

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 12:09:52 | SELECT cc.iban, ROUND(AVG(t.amount), 2) AS avg_amount FROM transactions t JOIN cred... | 1 row(s) returned |

**Nivel 2**

**Crea una nueva tabla que refleje el estado de las tarjetas de crédito basado en si las últimas tres transacciones fueron declinadas y genera la siguiente consulta:**

Tal como nos piden, creamos una tabla llamada card_status con una variable card_id que figure como clave primaria, con referencia a la variable id de la tabla credit_cards y otra variable status que nos represente el estado de la tarjeta de crédito (activada o desactivada), según el criterio establecido en el enunciado.

```sql
#NIVEL 2
#Ejercicio 1
CREATE TABLE IF NOT EXISTS card_status (
    card_id CHAR(8) PRIMARY KEY,
    status VARCHAR(20),
    FOREIGN KEY (card_id) REFERENCES credit_cards(id)
);
```

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 2 | 13:22:54 | CREATE TABLE IF NOT EXISTS card_status ( card_id CHAR(8) PRIMARY KEY, status V... | 0 row(s) affected |

Una vez tenemos la tabla creada, haremos la introducción de los datos. Para ello usamos INSERT INTO card_status (card_id, status) y en lugar de escribir VALUES como hemos hecho en anteriores ocasiones, haremos una consulta seleccionando card_id, usaremos CASE WHEN SUM(declined) = COUNT(*) THEN 'not activated' para determinar que si la suma de todas las transacciones declinadas que cogemos iguala el número total de filas contadas en la selección, la tarjeta está desactivada. Si hay un solo 0, estará activada (ELSE activated'). Finalizamos con END AS status para que se introduzcan los datos correctamente en dicha columna. Para saber de donde coger estos datos, en FROM usaremos una subconsulta seleccionando card_id, declined, timestamp y una columna nueva que deberemos crear para poder coger las 3 o menos de 3 transacciones más recientes de cada card_id. Creí que usando ORDER BY timestamp y LIMIT 3 podría establecer el criterio que nos dan como requisito para saber el estado de cada tarjeta. Pero haciendo esto solo cogemos las 3 primeras filas. El resto de card_ids no se seleccionan. Consultando online vi que se puede establecer dicho criterio si creamos una columna que enumere las transacciones para cada card_id. La cláusula es ROW_NUMBER( ) OVER(PARTITION BY card_id ORDER BY timestamp DESC) AS row_num. Esto nos añadirá dicha columna con un índice que se auto incrementará para cada card_id, ordenado según la fecha de más reciente a menos. Las otras columnas seleccionadas las cogemos de la tabla transactions. Cerramos paréntesis de la subconsulta y le añadimos un alias AS LastThreeTransactions. En este caso, si no añadimos alias a la subconsulta nos da error. Añadimos debajo el criterio WHERE row_num <= 3 para que se quede con las 3 o menos de 3 transacciones más recientes de la subconsulta y GROUP BY card_id. Vemos cómo queda la tabla card_status.

```
87    #Insertar valores de status según las tres últimas transacciones
88 •  INSERT INTO card_status (card_id, status)
89    SELECT card_id, CASE WHEN SUM(declined) = COUNT(*) THEN 'not activated' ELSE 'activated' END AS status
90    FROM (SELECT card_id, declined, timestamp,
91                 ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS row_num
92          FROM transactions) AS LastThreeTransactions
93    WHERE row_num <= 3
94    GROUP BY card_id;
```

```
Output
Action Output
#  Time      Action                                                                  Message
1  14:24:10  INSERT INTO card_status (card_id, status) SELECT card_id, CASE WHEN SUM(declined) = ...  275 row(s) affected Records: 275  Duplicates: 0  Warnings: 0
```

Hacemos una consulta para ver si los datos están bien introducidos

```
96 •  SELECT *
97    FROM card_status;
98
```

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

| card_id | status |
|---|---|
| CcU-2938 | activated |
| CcU-2945 | activated |
| CcU-2952 | activated |
| CcU-2959 | activated |
| CcU-2966 | activated |

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 2 | 14:24:14 | SELECT * FROM card_status LIMIT 0, 1000 | 275 row(s) returned |

**Ejercicio 1**

**¿Cuántas tarjetas están activas?**

Para responder esta pregunta, simplemente hacemos SELECT COUNT(*) AS active_cards de la tabla card_status WHERE status = 'activated' y nos contará el total de tarjetas activas.

```
99    #Número de tarjetas activas
100 • SELECT COUNT(*) AS active_cards
101   FROM card_status
102   WHERE status = 'activated';
103
```

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

| active_cards |
|---|
| 275 |

Result 3 | Read Only

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 14:57:40 | SELECT COUNT(*) AS active_cards FROM card_status WHERE status = 'activated' LIMIT 0,... | 1 row(s) returned |

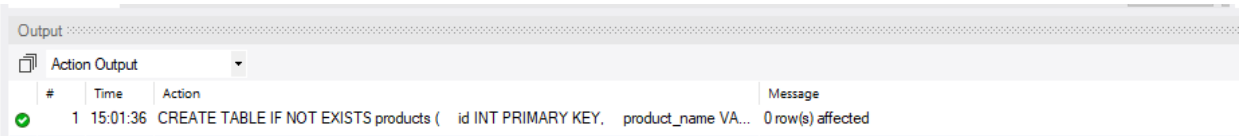
Según la consulta, todas están activadas, no hay ninguna con sus últimas 3 o menos de 3 transacciones declinadas.

**NIVEL 3**

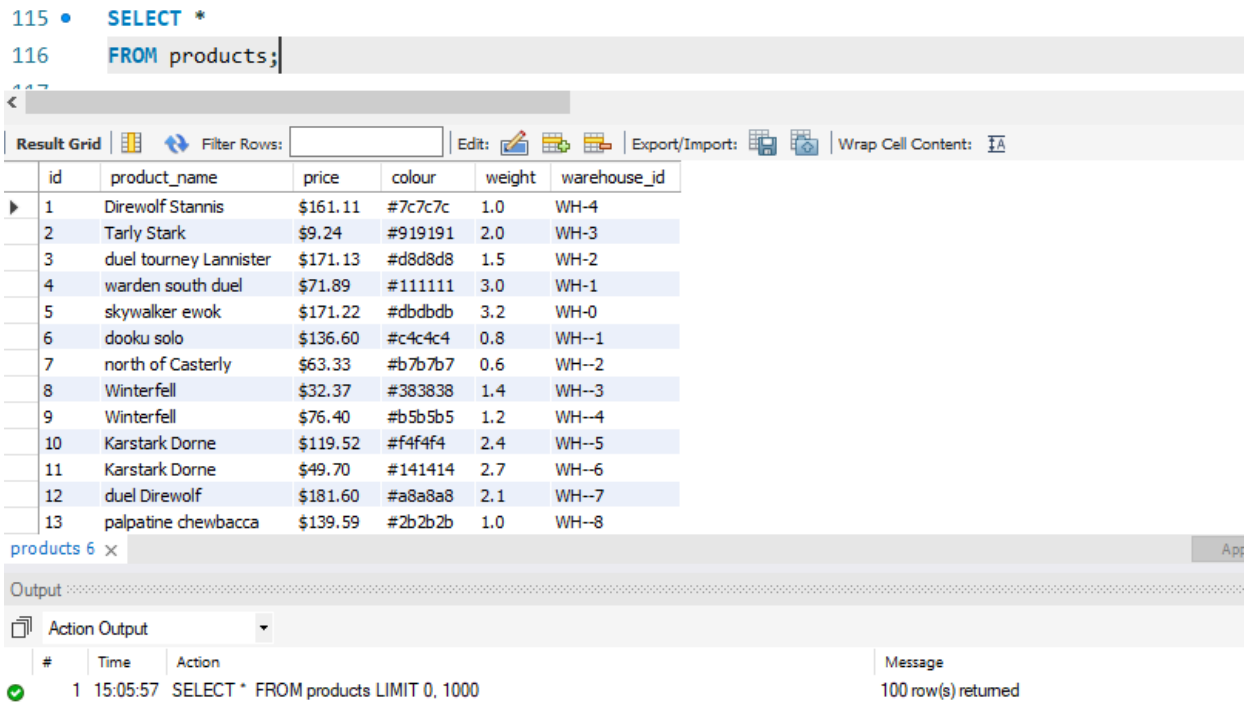
**Crea una tabla con la que podamos unir los datos del nuevo archivo products.csv con la base de datos creada, teniendo en cuenta que desde transaction tienes product_ids. Genera la siguiente consulta:**

Hacemos el mismo código que usamos para las tablas anteriores. Añadimos las columnas y nos aseguramos de incluir el tipo de dato adecuado para que no hayan errores al importar los datos del archivo csv. Id será la primary key.

```
106 • CREATE TABLE IF NOT EXISTS products (
107         id INT PRIMARY KEY,
108         product_name VARCHAR(50),
109         price VARCHAR(20),
110         colour CHAR(7),
111         weight DECIMAL(4, 1),
112         warehouse_id VARCHAR(10)
113     );
114
```




Importamos los datos y visualizamos la tabla para ver que todo está correcto.



```
115 •  SELECT *
116    FROM products;
```

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

| id | product_name | price | colour | weight | warehouse_id |
|---|---|---|---|---|---|
| 1 | Direwolf Stannis | $161.11 | #7c7c7c | 1.0 | WH-4 |
| 2 | Tarly Stark | $9.24 | #919191 | 2.0 | WH-3 |
| 3 | duel tourney Lannister | $171.13 | #d8d8d8 | 1.5 | WH-2 |
| 4 | warden south duel | $71.89 | #111111 | 3.0 | WH-1 |
| 5 | skywalker ewok | $171.22 | #dbdbdb | 3.2 | WH-0 |
| 6 | dooku solo | $136.60 | #c4c4c4 | 0.8 | WH--1 |
| 7 | north of Casterly | $63.33 | #b7b7b7 | 0.6 | WH--2 |
| 8 | Winterfell | $32.37 | #383838 | 1.4 | WH--3 |
| 9 | Winterfell | $76.40 | #b5b5b5 | 1.2 | WH--4 |
| 10 | Karstark Dorne | $119.52 | #f4f4f4 | 2.4 | WH--5 |
| 11 | Karstark Dorne | $49.70 | #141414 | 2.7 | WH--6 |
| 12 | duel Direwolf | $181.60 | #a8a8a8 | 2.1 | WH--7 |
| 13 | palpatine chewbacca | $139.59 | #2b2b2b | 1.0 | WH--8 |

products 6 ×

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 15:05:57 | SELECT * FROM products LIMIT 0, 1000 | 100 row(s) returned |

Una vez tenemos la tabla products, debemos relacionarla con transactions. Si nos fijamos en la columna product_ids de transactions, hay varios valores separados por comas en una misma fila o registro. Esto nos puede traer problemas si queremos hacer la relación con products o hacer consultas que impliquen ambas tablas. De modo que hay que separar los valores y hacer que aparezca un solo product_id por fila. Debemos separar y duplicar el resto de datos de cada fila o registro. Ignoro por completo la forma de hacerlo, así que consulté online y vi que se puede hacer de forma manual en excel usando fórmulas con cláusulas como SPLIT y FLATTEN. Split separa los valores de product_ids y los coloca en distintas columnas, flatten los convierte en filas. Sería cuestión de añadir el resto de valores manualmente copiando y pegando. También se puede usar SQL pero la query es demasiado larga o complicada en mi opinión. Sabiendo que vamos a manejar Python y librerías como pandas, he pensado que la solución más rápida y fácil de entender en cuanto a código, sería ésta: Creamos un notebook en Jupyter, importamos el archivo en pandas, hacemos la transformación y generamos un nuevo archivo limpio para poder importar en MySQL. Aquí está el código:

In [1]:

```python
import pandas as pd

# Cargar el archivo  desde la ubicación especificada
file_path = r'C:\Users\Joan Salas Dalmau\Downloads\transactions.xlsx'

# Leer el archivo en un DataFrame
df = pd.read_excel(file_path)

# la columna 'product_ids' contiene valores separados por comas
# Separamos los valores de 'product_ids' en listas
df['product_ids'] = df['product_ids'].str.split(',')

# Usamos explode para crear nuevas filas por cada elemento de la lista en 'product_ids'
df = df.explode('product_ids')

# Eliminar filas con valores nulos (NaN)
df = df.dropna(subset=['product_ids'])

# Convertimos los valores de 'product_ids' a int
df['product_ids'] = df['product_ids'].astype(int)

# Guardamos el nuevo DataFrame en un archivo Excel limpio
cleaned_file_path = r'C:\Users\Joan Salas Dalmau\Downloads\transactions_cleaned.xlsx'
df.to_excel(cleaned_file_path, index=False)

# Mostrar el DataFrame resultante
df
```

Out[1]:

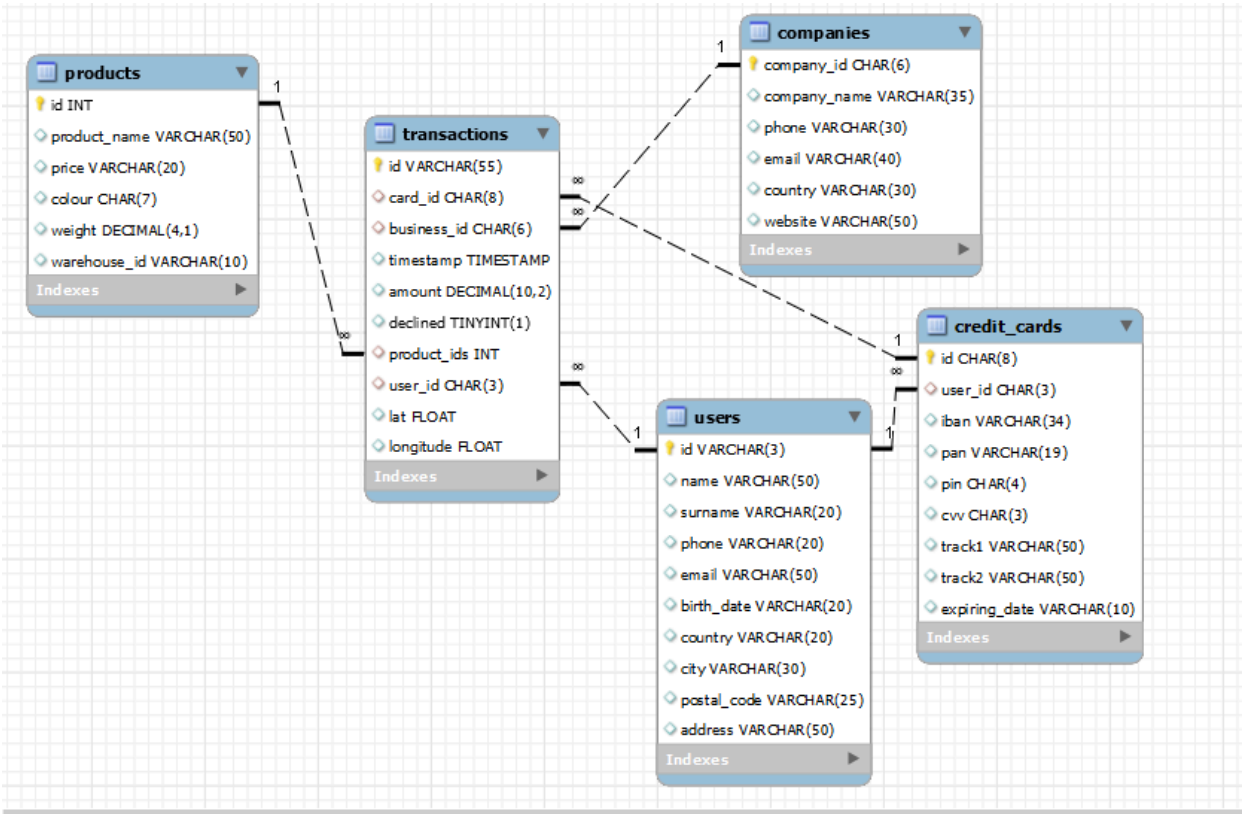
| | id | card_id | business_id | timestamp | amount | declined | product_ids | user_id | lat | longitude |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 7DC26247-20EC-53FE-E555-B6C2E55CA5D5 | CcU-2945 | b-2226 | 2022-02-04 15:52:56 | 312.50 | 0 | 71 | 275 | 58.936718 | -76.817110 |
| 1 | 7DC26247-20EC-53FE-E555-B6C2E55CA5D5 | CcU-2945 | b-2226 | 2022-02-04 15:52:56 | 312.50 | 0 | 41 | 275 | 58.936718 | -76.817110 |
| 2 | 72997E96-DC2C-A4D7-7C24-66C302F8AE5A | CcU-2952 | b-2230 | 2022-01-30 15:16:36 | 239.87 | 0 | 97 | 275 | 43.358406 | -17.657968 |
| 2 | 72997E96-DC2C-A4D7-7C24-66C302F8AE5A | CcU-2952 | b-2230 | 2022-01-30 15:16:36 | 239.87 | 0 | 41 | 275 | 43.358406 | -17.657968 |
| 2 | 72997E96-DC2C-A4D7-7C24-66C302F8AE5A | CcU-2952 | b-2230 | 2022-01-30 15:16:36 | 239.87 | 0 | 3 | 275 | 43.358406 | -17.657968 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 585 | A4D0D84F-4622-BB83-E6B6-51E545D4A217 | CcU-3533 | b-2562 | 2021-05-14 16:59:27 | 395.81 | 1 | 71 | 267 | -24.640383 | -69.876969 |
| 585 | A4D0D84F-4622-BB83-E6B6-51E545D4A217 | CcU-3533 | b-2562 | 2021-05-14 16:59:27 | 395.81 | 1 | 43 | 267 | -24.640383 | -69.876969 |
| 585 | A4D0D84F-4622-BB83-E6B6-51E545D4A217 | CcU-3533 | b-2562 | 2021-05-14 16:59:27 | 395.81 | 1 | 2 | 267 | -24.640383 | -69.876969 |
| 586 | 9FBB3D61-D3C2-E5BB-4BC3-6CC83C718D34 | CcU-3540 | b-2566 | 2021-04-19 21:33:41 | 490.19 | 1 | 53 | 267 | 75.211672 | -171.044656 |
| 586 | 9FBB3D61-D3C2-E5BB-4BC3-6CC83C718D34 | CcU-3540 | b-2566 | 2021-04-19 21:33:41 | 490.19 | 1 | 17 | 267 | 75.211672 | -171.044656 |

1248 rows × 10 columns

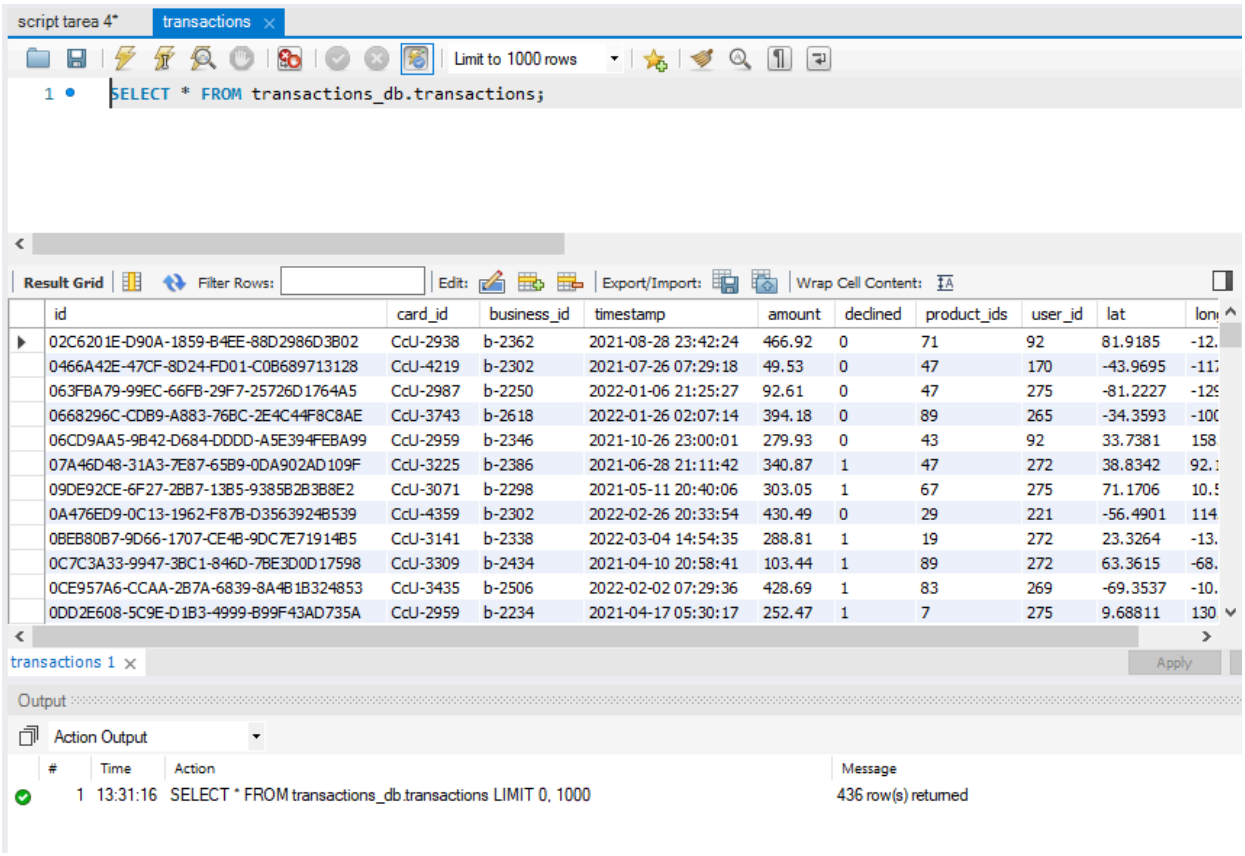
Ahora ya tenemos un solo valor en product_id con el resto de columnas duplicadas. Eliminamos los datos de la tabla transactions, añadimos la clave foránea en transactions e importamos el archivo limpiado.

```
118     #Eliminamos datos e importamos tabla con product_ids limpiada (un solo valor por fila)
119 •   TRUNCATE TABLE transactions;
120
121     #Añadimos restricción y clave foránea para relacionar la tabla transactions con products
122 •   ALTER TABLE transactions
123     ADD CONSTRAINT fk_product_ids
124     FOREIGN KEY (product_ids) REFERENCES products(id);
```

No incluyo screenshot de la ejecución del código en action output pero puedo confirmar que se ejecuta correctamente ya que si vuelvo a generar un diagrama con database > reverse engineer, aparecen todas las tablas relacionadas según las claves establecidas.




Veamos la tabla limpia con la columna de product_ids con un valor por registro.




**Ejercicio 1**

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

Para saber el número de ventas de cada producto haremos la siguiente consulta, seleccionamos de la tabla products con el alias ‘p’ product_name, usamos la cláusula COUNT(t.product_ids) AS sales de la tabla transactions para que nos dé el número de ventas de cada producto. Juntamos products y transactions por p.id = t.product_ids, usamos el criterio WHERE declined = 0 para coger las transacciones que se han hecho efectivas, agrupamos por nombre de producto y ordenamos por ventas de mayor a menor para ver los más vendidos primero.

```
126    # número de veces que se ha vendido cada producto
127    SELECT p.product_name, COUNT(t.product_ids) AS sales
128    FROM transactions t
129    JOIN products p ON p.id = t.product_ids
130    WHERE declined = 0
131    GROUP BY p.product_name
132    ORDER BY sales DESC;
```

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

| product_name | sales |
|---|---|
| skywalker ewok | 24 |
| riverlands north | 23 |
| Direwolf riverlands the | 23 |
| Direwolf Stannis | 21 |
| Winterfell Lannister | 21 |
| duel | 20 |
| skywalker ewok sith | 20 |

Result 5

Output

Action Output

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 13:18:56 | SELECT p.product_name, COUNT(t.product_ids) AS sales FROM transactions t JOIN produc... | 24 row(s) returned |