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)
);
```

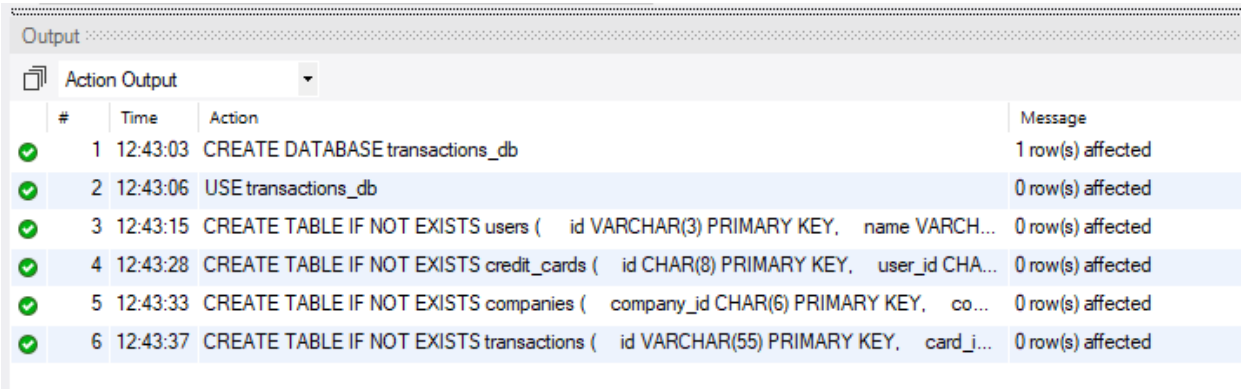
La siguiente tabla, credit_cards, contiene una clave foránea al final. La variable user_id hacer referencia a la id de la tabla users, de modo que usamos FOREIGN KEY (user_id) REFERENCES users(id) para que queden relacionadas ambas tablas.

```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)
);
```

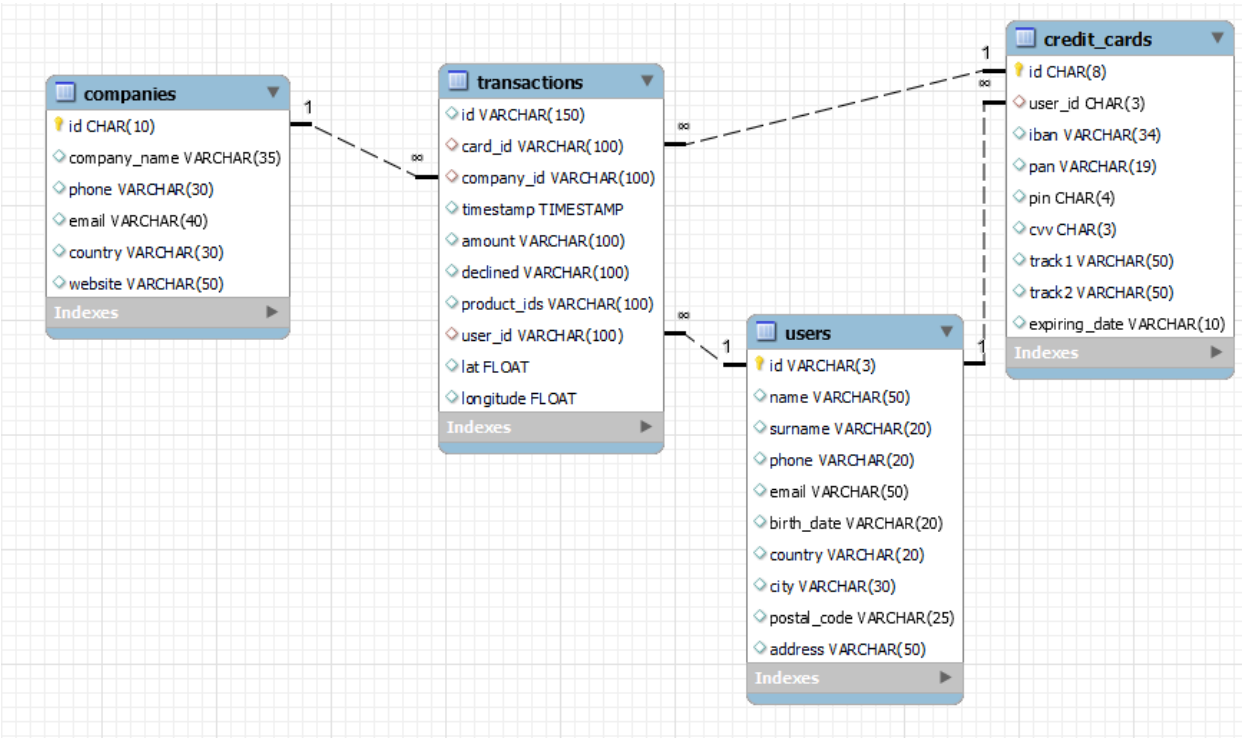
```sql
# Tabla companies
CREATE TABLE IF NOT EXISTS companies (
    company_id CHAR(6) PRIMARY KEY,
    company_name VARCHAR(35),
    phone VARCHAR(30),
    email VARCHAR(40),
    country VARCHAR(30),
    website VARCHAR(50)
);
```

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(150),
    card_id VARCHAR(100),
    company_id VARCHAR(100),
    timestamp TIMESTAMP,
    amount VARCHAR(100),
    declined VARCHAR(100),
    product_ids VARCHAR(100),
    user_id VARCHAR(100),
    lat FLOAT,
    longitude FLOAT,
    FOREIGN KEY (card_id) REFERENCES credit_cards(id),
    FOREIGN KEY (company_id) REFERENCES companies(id),
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```



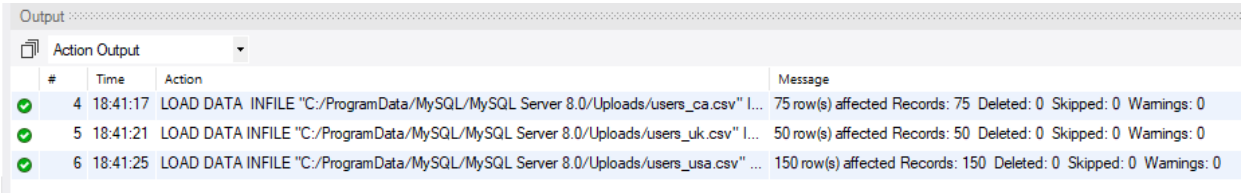
Una vez ejecutamos cada tabla y nos aseguramos que están correctamente creadas, vamos a database, reverse engineer y hacemos click en next, seleccionamos la base de datos transactions_db y seguimos haciendo click en next/finish hasta que nos aparezca el diagrama. Veremos que las relaciones son de N:1 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.

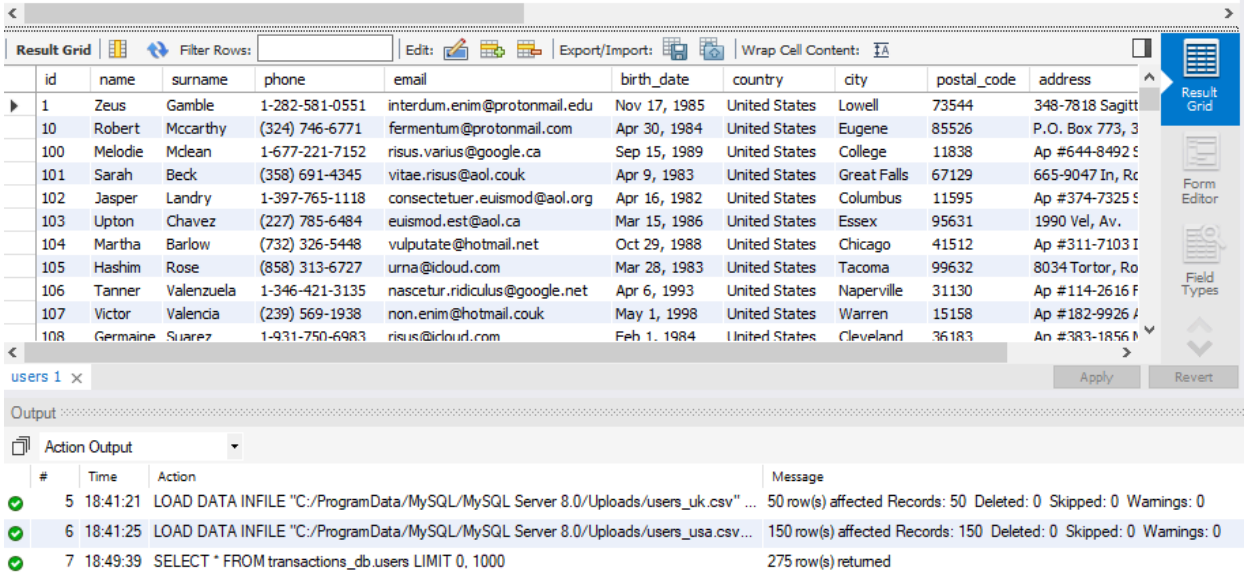
Para ello usaremos un bloque de código similar en todos los archivos. LOAD DATA INFILE con la ubicación permitida para cargar archivos a MySQL y el nombre del archivo a importar. INTO TABLE con el nombre de la tabla que queramos rellenar, FIELDS TERMINATED BY ‘,’ para indicar si lo valores de cada columna están separados por coma o punto y coma, ENCLOSED BY ‘ “ ’ para indicar si hay términos entre comillas, LINES TERMINATED BY con el código que indica fin o nueva línea y por último IGNORE 1 ROWS para ignorar la primera fila con los títulos de cada columna, solamente queremos los datos.

```
21    #inserción de datos de los 3 archivos csv para users
22 •  LOAD DATA  INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users_ca.csv"
23    INTO TABLE users
24    FIELDS TERMINATED BY ','
25    ENCLOSED BY '"'
26    LINES TERMINATED BY '\r\n'
27    IGNORE 1 ROWS;
```



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’.

```
1 •  SELECT * FROM transactions_db.users;
```



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

id	name	surname	phone	email	birth_date	country	city	postal_code	address
1	Zeus	Gamble	1-282-581-0551	interdum.enim@protonmail.edu	Nov 17, 1985	United States	Lowell	73544	348-7818 Sagitt
10	Robert	Mccarthy	(324) 746-6771	fermentum@protonmail.com	Apr 30, 1984	United States	Eugene	85526	P.O. Box 773, 3
100	Melodie	Mclean	1-677-221-7152	risus.varius@google.ca	Sep 15, 1989	United States	College	11838	Ap #644-8492 S
101	Sarah	Beck	(358) 691-4345	vitae.risus@aol.couk	Apr 9, 1983	United States	Great Falls	67129	665-9047 In, Rc
102	Jasper	Landry	1-397-765-1118	consectetuer.euismod@aol.org	Apr 16, 1982	United States	Columbus	11595	Ap #374-7325 S
103	Upton	Chavez	(227) 785-6484	euismod.est@aol.ca	Mar 15, 1986	United States	Essex	95631	1990 Vel, Av.
104	Martha	Barlow	(732) 326-5448	vulputate@hotmail.net	Oct 29, 1988	United States	Chicago	41512	Ap #311-7103 I
105	Hashim	Rose	(858) 313-6727	urna@icloud.com	Mar 28, 1983	United States	Tacoma	99632	8034 Tortor, Ro
106	Tanner	Valenzuela	1-346-421-3135	nascetur.ridiculus@google.net	Apr 6, 1993	United States	Naperville	31130	Ap #114-2616 F
107	Victor	Valencia	(239) 569-1938	non.enim@hotmail.couk	May 1, 1998	United States	Warren	15158	Ap #182-9926 A
108	Germaine	Suarez	1-931-750-6983	risus@icloud.com	Feb 1, 1984	United States	Cleveland	36183	Ap #383-1856 M

Result Grid | Form Editor | Field Types

users 1 | Apply | Revert

Output

Action Output

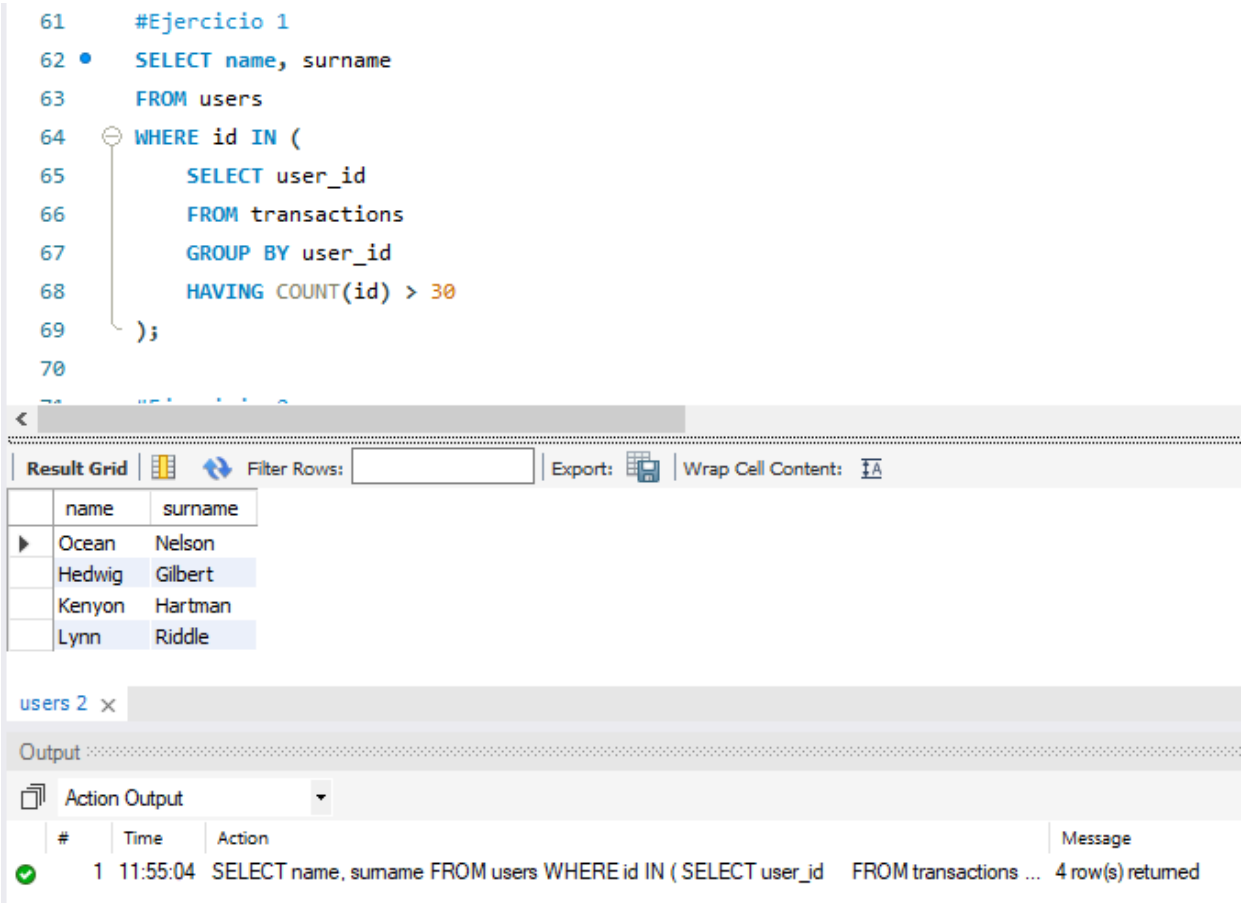
#	Time	Action	Message
5	18:41:21	LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users_uk.csv" ...	50 row(s) affected Records: 50 Deleted: 0 Skipped: 0 Warnings: 0
6	18:41:25	LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users_usa.csv...	150 row(s) affected Records: 150 Deleted: 0 Skipped: 0 Warnings: 0
7	18:49:39	SELECT * FROM transactions_db.users LIMIT 0, 1000	275 row(s) returned

Hacemos exactamente lo mismo para el resto de tablas con sus respectivos datos importados de los archivos csv, proporcionados por el enunciado. Para las demás tablas podemos prescindir de LINES TERMINATED BY. A tener en cuenta también que antes y después de escribir el código con la carga de datos para transactions, al incorporar claves foráneas, debemos usar SET FOREIGN_KEY_CHECKS = 0 antes y SET FOREIGN_KEY_CHECKS = 1 después para evitar errores en el proceso de carga.

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.

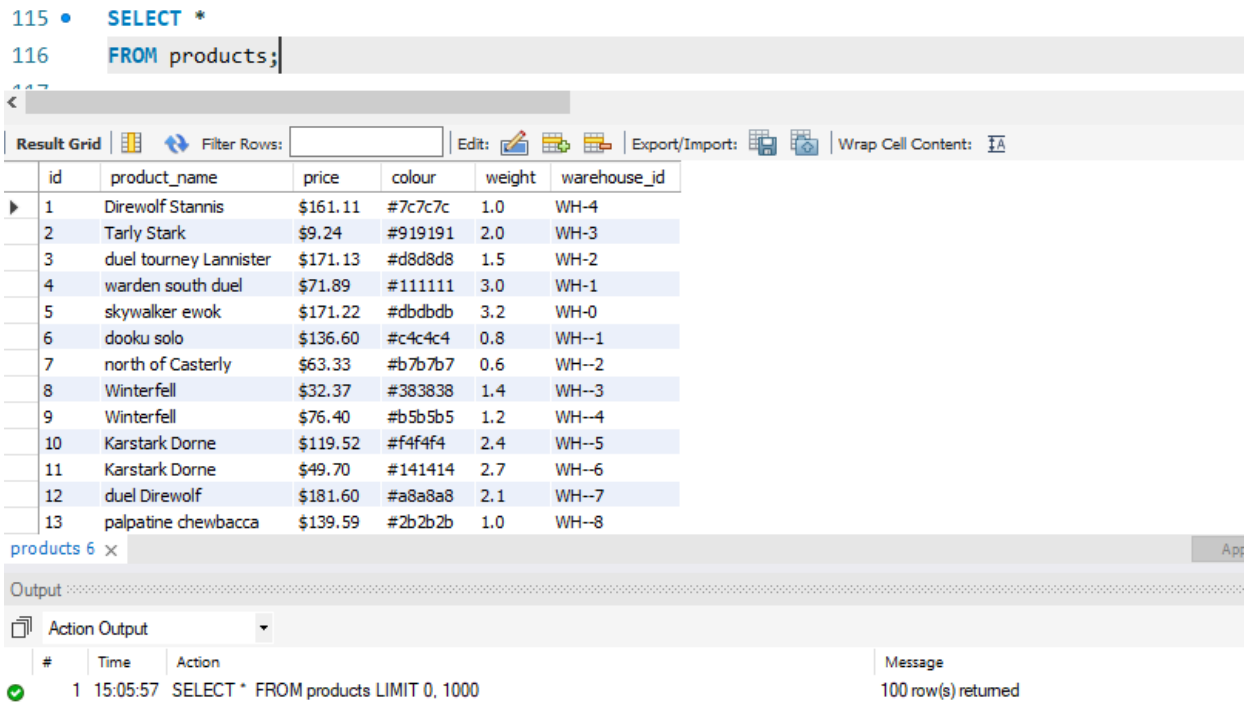
```
159    # Ejercicio 1
160 •  CREATE TABLE IF NOT EXISTS products (
161        id VARCHAR(100) PRIMARY KEY,
162        product_name VARCHAR(50),
163        price VARCHAR(20),
164        colour CHAR(7),
165        weight DECIMAL(4, 1),
166        warehouse_id VARCHAR(10)
167    );
168
169 •  LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/products.csv"
170    INTO TABLE products
171    FIELDS TERMINATED BY ','
172    ENCLOSED BY '"'
173    IGNORE 1 ROWS;
```

Output

Action Output

#	Time	Action	Message
15	18:50:47	SET FOREIGN_KEY_CHECKS = 1	0 row(s) affected
16	18:55:12	CREATE TABLE IF NOT EXISTS products (id VARCHAR(100) PRIMARY KEY, produ...	0 row(s) affected
17	18:55:17	LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/products.csv" ...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0

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. Para ello usaremos una CTE (Common Table Expression) que contenga la misma columna de product_ids en transactions pero con un solo valor por fila. El código es bastante difícil de interpretar, primero se crea la tabla transactions_products y luego introducimos valores en esta CTE haciendo la separación de product_ids por ‘,’ e introduciendo un solo valor por fila. Veamos como queda en MySQL y posteriormente describimos cada bloque de código:

```sql
#Crear un índice en las tabla de referencia si no existe
ALTER TABLE transactions ADD INDEX (id);

# Crear la tabla puente transaction_products
CREATE TABLE IF NOT EXISTS transaction_products (
    transaction_id VARCHAR(255),
    product_id VARCHAR(100),
    FOREIGN KEY (transaction_id) REFERENCES transactions(id),
    FOREIGN KEY (product_id) REFERENCES products(id)
);
```

Output

Action Output

#	Time	Action	Message
1	19:08:32	ALTER TABLE transactions ADD INDEX (id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
2	19:08:33	CREATE TABLE IF NOT EXISTS transaction_products (transaction_id VARCHAR(255), ...	0 row(s) affected

```sql
# Insertar los resultados descompuestos en la tabla transaction_product
INSERT INTO transaction_products (transaction_id, product_id)
WITH RECURSIVE numbers AS (
    SELECT 1 AS n
    UNION ALL
    SELECT n + 1 FROM numbers WHERE n < 5 -- Ajustar límite según valor máximo de product_ids por celda
)
SELECT t.id, TRIM(SUBSTRING_INDEX(SUBSTRING_INDEX(t.product_ids, ',', n.n), ',', -1)) AS product_id
FROM transactions t
JOIN numbers n
ON CHAR_LENGTH(t.product_ids) - CHAR_LENGTH(REPLACE(t.product_ids, ',', '')) >= n.n - 1;
```

Output

Action Output

#	Time	Action	Message
1	19:08:32	ALTER TABLE transactions ADD INDEX (id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
2	19:08:33	CREATE TABLE IF NOT EXISTS transaction_products (transaction_id VARCHAR(255), ...	0 row(s) affected
3	19:10:23	INSERT INTO transaction_products (transaction_id, product_id) WITH RECURSIVE numbers ...	1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0

Primero creamos la tabla transaction_products que hará de puente entre products y transactions. Añadimos como variables las ids de ambas tablas y las relacionamos con claves foráneas. A continuación procedemos a la inserción de los valores de esta tabla con INSERT INTO. Dentro de esta cláusula creamos una CTE (Common Table Expression) recursiva llamada numbers que genera una secuencia de números. Se puede interpretar como un for o while loop de Python.

SELECT 1 AS n inicia la secuencia con el número 1 que será el valor inicial de n. UNION ALL se usa para combinar el valor inicial con el siguiente valor en la secuencia generada recursivamente. SELECT n + 1 FROM numbers WHERE n < 5 añade 1 al valor de n generado anteriormente, hasta que n alcance el valor límite de 5. Este límite se puede ajustar para que

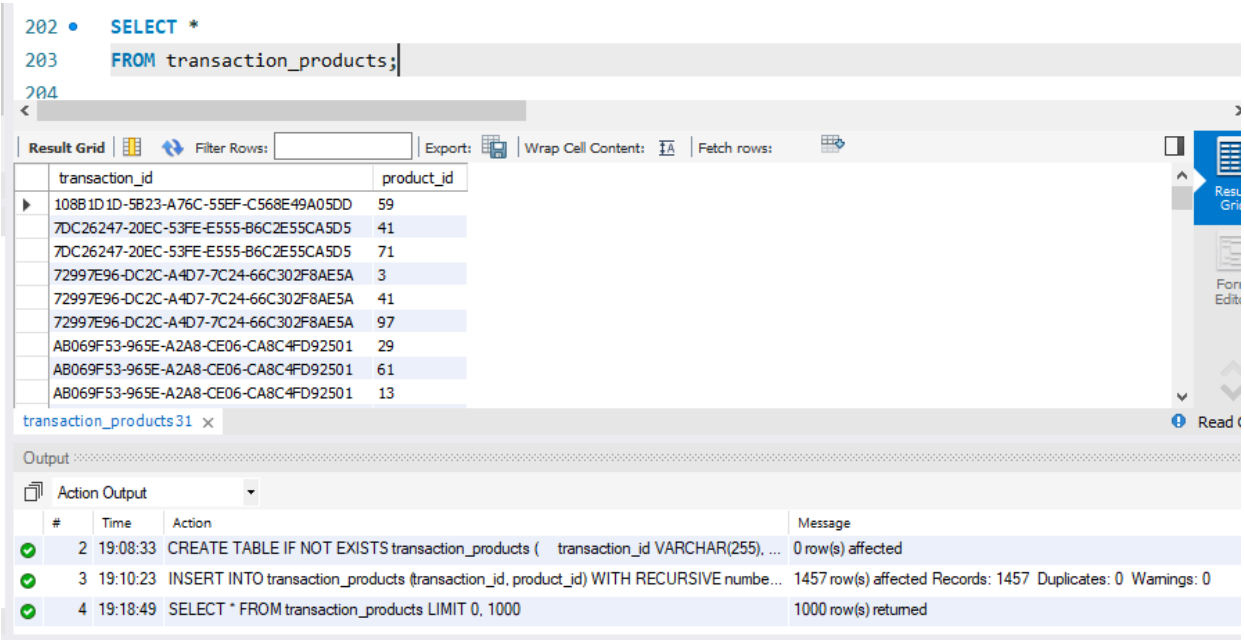
coincida con el número máximo de productos que esperas en la columna product_ids. Esta secuencia la usamos para generar índices que nos ayudarán a extraer cada elemento separado por comas en la columna product_ids.

SELECT t.id, TRIM(SUBSTRING_INDEX(SUBSTRING_INDEX(t.product_ids, ',', n.n), ',', -1)) AS product_id. La segunda selección extrae cada valor de product_ids de forma individual. Lo que hace esta parte es usar SUBSTRING_INDEX para obtener un producto específico basado en el número generado por la secuencia n. Primero extrae desde la columna product_ids una subcadena que contiene los primeros n productos separados por comas. Una vez se extraen los primeros n productos, selecciona sólo el último producto (usando -1). La cláusula TRIM es similar a .split( ) en Python, elimina cualquier espacio en blanco extra que pueda estar alrededor del producto extraído.

JOIN numbers n liga la tabla transactions con la CTE numbers, lo que permite iterar sobre las listas de productos. Esto crea una fila por cada producto en product_ids. ON CHAR_LENGTH(t.product_ids) - CHAR_LENGTH(REPLACE(t.product_ids, ',', '')) >= n.n - 1 es una condición que asegura que la secuencia n no exceda el número de productos en product_ids. Funciona calculando cuántas comas hay en la columna product_ids (indicando cuántos productos hay) y comparando ese valor con n:

CHAR_LENGTH(t.product_ids) Obtiene la longitud total de la cadena product_ids.
CHAR_LENGTH(REPLACE(t.product_ids, ',', '')) obtiene la longitud de la cadena product_ids eliminando todas las comas. La diferencia entre las dos expresiones anteriores es el número de comas, es decir, el número de productos menos uno. ‘n.n - 1’ se usa para comparar el índice actual n con el número de productos para asegurarse de no intentar acceder a un índice que no existe.

Aquí vemos el aspecto que tiene la CTE transaction_products:



Una vez tenemos creada la tabla puente con 1 solo product_id por fila, podemos hacer el último ejercicio.

Ejercicio 1

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

Para ello, seleccionaremos de la tabla puente product_id, el nombre del producto de la tabla products, usamos una función agregada para contar el número de transacciones realizadas según el product_id y usamos como alias total_sales. Hacemos uso de JOIN para unir products y transactions con la tabla puente transaction_products por medio de las ids. Usamos WHERE declined = 0 para coger transacciones aceptadas solamente (ventas) y agrupamos por product_id ya que un producto puede llamarse igual pero tener distintas características. Es importante agrupar por id para sacar el número de veces que se ha vendido un producto concreto. De lo contrario agrupamos por nombre y no obtenemos las cifras correctas. Finalmente, ordenamos de mayor a menor número de veces que se haya vendido un producto con total_sales DESC.

```sql
# Consultar el número de ventas por producto
SELECT tp.product_id, p.product_name, COUNT(t.id) AS total_sales
FROM transaction_products tp
JOIN products p ON tp.product_id = p.id
JOIN transactions t ON tp.transaction_id = t.id
WHERE t.declined = 0
GROUP BY tp.product_id
ORDER BY total_sales DESC;
```

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

product_id	product_name	total_sales
23	riverlands north	60
67	Winterfell	59
2	Tarly Stark	56
17	skywalker ewok sith	54
43	duel	54
97	jinn Winterfell	53
79	Direwolf riverlands the	52

Result 33 | Read Only

Output

Action Output

#	Time	Action	Message
4	19:18:49	SELECT * FROM transaction_products LIMIT 0, 1000	1000 row(s) returned
5	19:20:32	SELECT tp.product_id, p.product_name, COUNT(t.id) AS total_sales FROM transaction_pro...	26 row(s) returned
6	19:24:51	SELECT tp.product_id, p.product_name, COUNT(t.id) AS total_sales FROM transaction_pro...	26 row(s) returned