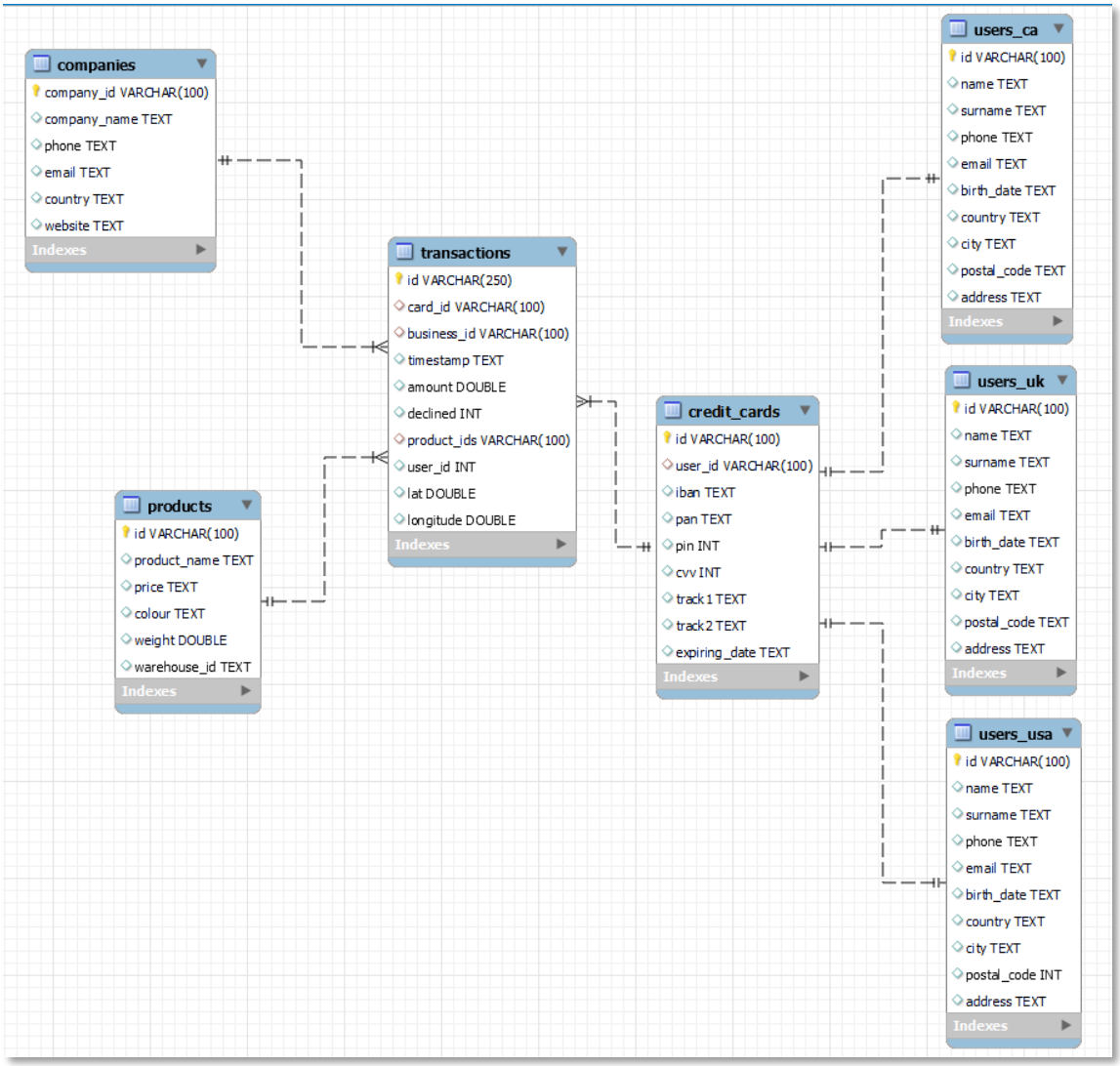
SPRINT 4

SPRINT 4, NIVEL 1, EJERCICIO 1 – Estudio de la base de datos previo

Se pide en el enunciado la búsqueda de usuarios con más de 30 transacciones, si bien inicialmente vamos a realizar algunos ejercicios y formateo de datos previos

```
# Elaboració d'un dataframe a partir de CSV.

# Importació, formateig i Relacions Entitats

Show databases;

use sprint4_db;

show tables;

# Utilización el Wizard para la creación e importación de las tablas desde un CSV

# Manipulación de las tablas y compatibilidad de los campos PK, FK - Es necesario garantizar la

# compatibilidad de formato entre tablas, si bien detectamos una incidencia en PRODUCTS

# que resolveremos más tarde, por ello utilizamos el comando SET FOREIGN_KEY_CHECKS = 0

# Creamos índices en tabla transactions

SHOW CREATE TABLE transactions;

ALTER TABLE transactions

ADD INDEX (card_id),

ADD INDEX (business_id),

ADD INDEX (product_ids),

ADD INDEX (user_id);

ALTER TABLE credit_cards

ADD INDEX (user_id);

ALTER TABLE credit_cards

ADD foreign key (user_id) REFERENCES users_ca(id);

ALTER TABLE credit_cards

ADD foreign key (user_id) REFERENCES users_uk(id);

ALTER TABLE credit_cards

ADD foreign key (user_id) REFERENCES users_usa(id);

ALTER TABLE transactions

ADD FOREIGN KEY (business_id) REFERENCES companies(company_id);

ALTER TABLE transactions
```

```
ADD FOREIGN KEY(card_id) REFERENCES credit_cards(id);

SET FOREIGN_KEY_CHECKS = 0;

ALTER TABLE transactions

ADD FOREIGN KEY(product_ids) REFERENCES products(id);

# Creamos una vista VIEW combinada con UNION de las tablas de usuarios

# Hago este paso previo porque intuio que me va a ser útil en el futuro

CREATE VIEW usuarios_all_countries AS

SELECT * FROM users_ca

UNION

SELECT * FROM users_uk

UNION

SELECT * FROM users_usa;

# realizamos una comprobación (en principio innecesaria) de que un usuario no esté en varios países.

SELECT id, count(id) FROM usuarios_all_countries

GROUP BY id

HAVING count(id)>1;

# realizamos otra comprobación adicional mediante la cual descubrimos que

# los usuarios 1 y 10  están dados de alta en la tabla users_usa, sin embargo

# jamás han realizado una transacción, por lo que debermos JOIN bajo esta premisa

SELECT *  FROM usuarios_all_countries

ORDER BY id ASC;

SELECT *  FROM transactions

#WHERE user_id =1

ORDER BY user_id ASC;
```


companies
company_id VARCHAR(100)
company_name TEXT
phone TEXT
email TEXT
country TEXT
website TEXT
Indexes
transactions
id VARCHAR(250)
card_id VARCHAR(100)
business_id VARCHAR(100)
timestamp TEXT
amount DOUBLE
declined INT
product_ids VARCHAR(100)
user_id INT
lat DOUBLE
longitude DOUBLE
Indexes
products
id VARCHAR(100)
product_name TEXT
price TEXT
colour TEXT
weight DOUBLE
warehouse_id TEXT
Indexes
credit_cards
id VARCHAR(100)
user_id VARCHAR(100)
iban TEXT
pan TEXT
pin INT
cvv INT
track1 TEXT
track2 TEXT
expiring_date TEXT
Indexes
users_ca
id VARCHAR(100)
name TEXT
surname TEXT
phone TEXT
email TEXT
birth_date TEXT
country TEXT
city TEXT
postal_code TEXT
address TEXT
Indexes
users_uk
id VARCHAR(100)
name TEXT
surname TEXT
phone TEXT
email TEXT
birth_date TEXT
country TEXT
city TEXT
postal_code TEXT
address TEXT
Indexes
users_usa
id VARCHAR(100)
name TEXT
surname TEXT
phone TEXT
email TEXT
birth_date TEXT
country TEXT
city TEXT
postal_code INT
address TEXT
Indexes

SPRINT 4, NIVEL 1, EJERCICIO 1

Usuarios con más de 30 transacciones,

Este código SQL realiza una consulta para encontrar usuarios que hayan realizado más de 30 transacciones. Aquí está una breve explicación de lo que hace cada parte del código:

1. **SELECT**: Selecciona las columnas que queremos mostrar en el resultado de la consulta. En este caso, se seleccionan el id del usuario (`u.id`), su nombre (`u.name`), su apellido (`u.surname`), y la cuenta de transacciones realizadas por ese usuario (`count(user_id)`), la cual es renombrada como `contador`.

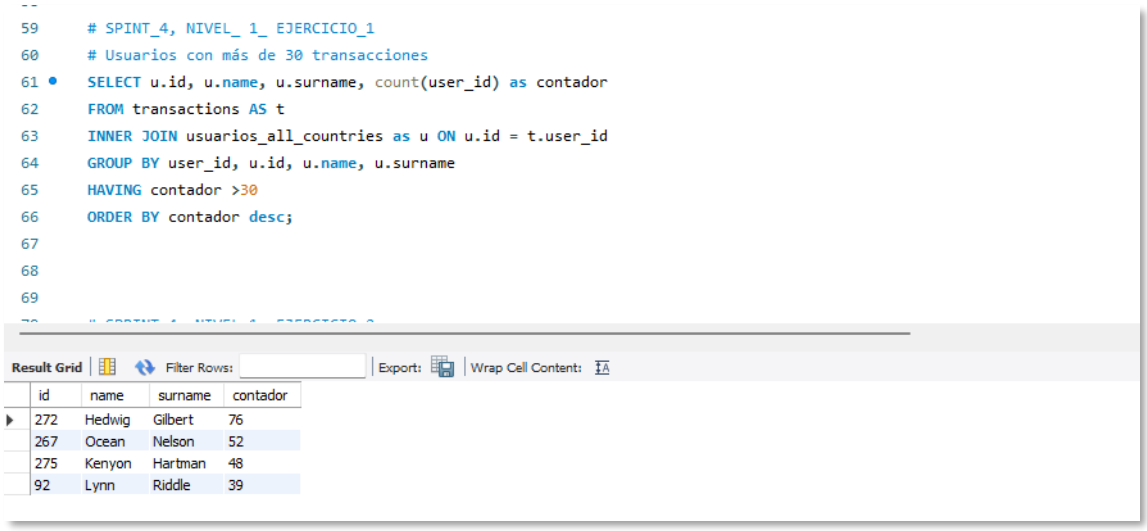
2. **FROM**: Especifica las tablas de donde se obtendrán los datos. `transactions` y `usuarios_all_countries` son las tablas utilizadas en esta consulta. La tabla `transactions` se referencia como `t` y la tabla `usuarios_all_countries` se referencia como `u`.

3. **INNER JOIN**: Une las tablas `transactions` y `usuarios_all_countries` basándose en la condición de que el `id` del usuario en la tabla de usuarios coincida con el `user_id` en la tabla de transacciones.

4. **GROUP BY**: Agrupa los resultados basándose en el `user_id`, `u.id`, `u.name` y `u.surname`. Esto significa que cada fila del resultado representará a un usuario distinto.

5. **HAVING**: Filtra los grupos resultantes, restringiendo el resultado a aquellos grupos donde el contador (es decir, el número de transacciones) es mayor que 30.

6. **ORDER BY**: Ordena el resultado final en orden descendente según el contador (número de transacciones), lo que significa que los usuarios con más transacciones aparecerán primero en el resultado.



```
59    # SPINT_4, NIVEL_ 1_ EJERCICIO_1
60    # Usuarios con más de 30 transacciones
61    SELECT u.id, u.name, u.surname, count(user_id) as contador
62    FROM transactions AS t
63    INNER JOIN usuarios_all_countries as u ON u.id = t.user_id
64    GROUP BY user_id, u.id, u.name, u.surname
65    HAVING contador >30
66    ORDER BY contador desc;
```

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

id	name	surname	contador
272	Hedwig	Gilbert	76
267	Ocean	Nelson	52
275	Kenyon	Hartman	48
92	Lynn	Riddle	39

SPRINT 4, NIVEL 1, EJERCICIO 2

El ejercicio nos solicita la media de transacciones para una empresa llamada Donec, si bien como se verá, realizo antes una serie de comprobaciones:

Comprobación de tarjetas duplicadas: Verifica si hay tarjetas duplicadas en la tabla credit_cards, mostrando el IBAN y contando cuántas veces aparece cada uno. Esto es importante para garantizar la integridad de los datos y asegurarse de que el recuento coincida con la cantidad esperada de transacciones.

Verificación de la empresa Donec: Se asegura de que la empresa Donec esté representada correctamente en la tabla companies. Si hay más de una entrada para Donec o si alguna de ellas no coincide exactamente con "Donec Ltd", podría haber un problema en los datos.

Cálculo de la media de transacciones por IBAN de Donec Ltd: Finalmente, calcula la media de las transacciones por IBAN para la empresa Donec Ltd, mostrando el nombre de la empresa, el IBAN y la media de las transacciones redondeada a dos decimales. Esto proporciona información sobre el comportamiento de las transacciones para esta empresa específica.

```
--
-- SPRINT_4, NIVEL 1_ EJERCICIO_2
-- Media por IBAN de Donec LTD

-- Primero hacemos una comprobación rutinaria para asegurarnos de que no hay tarjetas duplicadas
-- y que el recuento coincide con las 275 transacciones existentes
SELECT iban, COUNT(iban) AS contador
FROM credit_cards
GROUP BY iban
HAVING contador > 1;

-- Luego, verificamos si hay más de una entrada para Donec y si la especificación coincide con Donec Ltd
SELECT company_id, company_name
FROM companies
WHERE company_name LIKE 'Donec%';

-- Finalmente, calculamos la media de las transacciones por IBAN para Donec Ltd
SELECT c.company_name, cd.iban, ROUND(AVG(t.amount), 2) AS media_trans
FROM transactions AS t
JOIN companies AS c ON c.company_id = t.business_id
JOIN credit_cards AS cd ON cd.id = t.card_id
WHERE c.company_name = 'Donec Ltd'
GROUP BY c.company_name, cd.iban;

```

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

company_name	iban	media_trans
Donec Ltd	PT87806228135092429456346	203.72

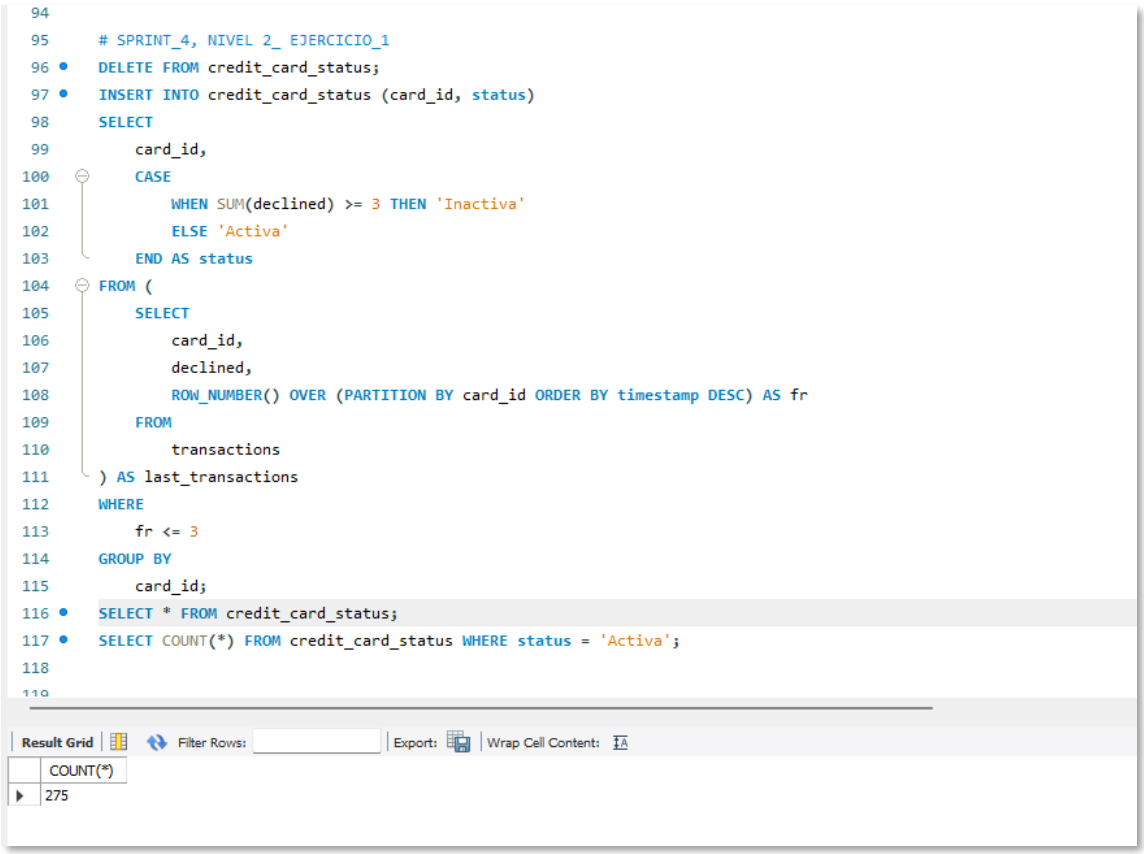
SPRINT 4, NIVEL 2, EJERCICIO1

Se quiere saber cuantas tarjetas están activas mediante la creación de una tabla que refleje operaciones declinadas, las tres últimas.

Empezamos por eliminar los registros de la tabla creada, el motivo es que me he dado cuenta de que cómo era lógico, iba acumulando nuevas líneas abajo cada vez que ejecutábamos el código, por eso este paso es imprescindible. Luego ya el código sigue con su trabajo e inserta nuevos registros en la tabla credit_card_status: Para cada tarjeta de crédito, determina su estado (activo o inactivo) según la cantidad de transacciones rechazadas (declined) registradas en las tres transacciones más recientes. Si la suma de las transacciones rechazadas es igual o mayor a 3, el estado de la tarjeta se establece como "Inactiva"; de lo contrario, se establece como "Activa".

Muestra todos los registros de la tabla credit_card_status: Esto muestra el estado actual de todas las tarjetas de crédito en la tabla.

Cuenta cuántas tarjetas están activas en la tabla credit_card_status: Esto proporciona el número total de tarjetas de crédito que se consideran activas en la tabla credit_card_status.



```
94
95      # SPRINT_4, NIVEL 2_ EJERCICIO_1
96      DELETE FROM credit_card_status;
97      INSERT INTO credit_card_status (card_id, status)
98      SELECT
99          card_id,
100         CASE
101             WHEN SUM(declined) >= 3 THEN 'Inactiva'
102             ELSE 'Activa'
103         END AS status
104     FROM (
105         SELECT
106             card_id,
107             declined,
108             ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS fr
109         FROM
110             transactions
111     ) AS last_transactions
112     WHERE
113         fr <= 3
114     GROUP BY
115         card_id;
116     SELECT * FROM credit_card_status;
117     SELECT COUNT(*) FROM credit_card_status WHERE status = 'Activa';
118
```

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

COUNT(*)
275

SPRINT 4, NIVEL 3, EJERCICIO1

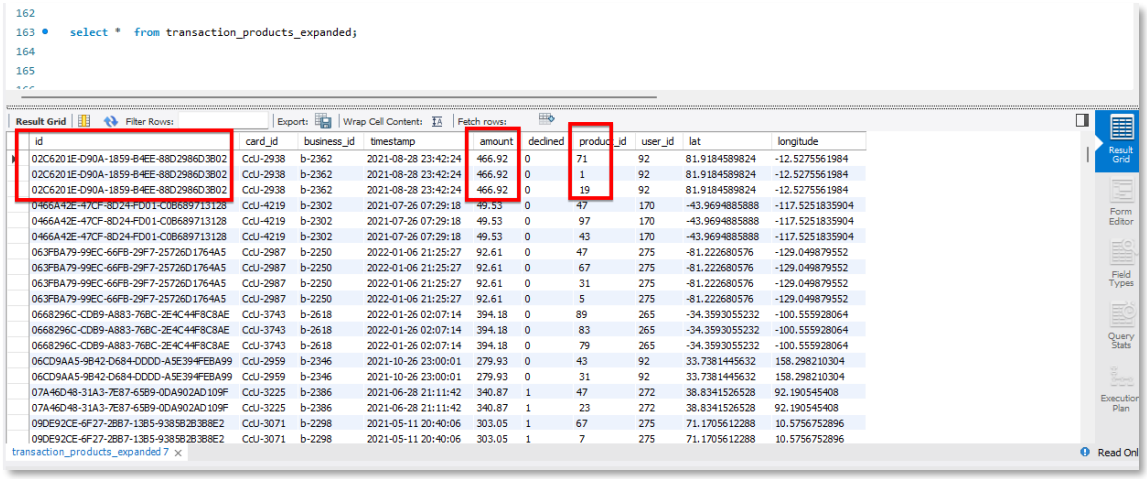
Queremos saber las ventas en unidades que se han realizado de cada producto.

EL problema es que la tabla transacciones refleja que en una misma transacción puede haber varios productos, los cuales son mostrados mediante una como como separador.

Podríamos intentar romper la cadena, PERO HE UTILIZADO UN MÉTODO QUE AUNQUE POCO ORTODOXO, CREO QUE ES BASTANTE ÚTIL, SE TRATA CREAR UNA TABLA NUEVA EN LA QUE MANTENIENDO EL RESTO DE VALORES DE LA FILA, SEGREGE LAS VENTAS DE PRODUCTOS EN VARIAS FILAS, POR EJEMPLO, SI UNA VENTA TIENE TRES PRODUCTOS, VA A CREAR 3 FILAS.

Luego usaré esta tabla de forma fácil para hacer la consulta.

Aquí muestro el resultado parcial de este método:



```
162
163 •   select *  from transaction_products_expanded;
164
165
```

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

id	card_id	business_id	timestamp	amount	declined	product_id	user_id	lat	longitude
02C6201E-D90A-1859-B4EE-88D2986D3B02	CcU-2938	b-2362	2021-08-28 23:42:24	466.92	0	71	92	81.9184589824	-12.5275561984
02C6201E-D90A-1859-B4EE-88D2986D3B02	CcU-2938	b-2362	2021-08-28 23:42:24	466.92	0	1	92	81.9184589824	-12.5275561984
02C6201E-D90A-1859-B4EE-88D2986D3B02	CcU-2938	b-2362	2021-08-28 23:42:24	466.92	0	19	92	81.9184589824	-12.5275561984
0466A42E-47CF-8D24-FD01-C0B689713128	CcU-4219	b-2302	2021-07-26 07:29:18	49.53	0	47	170	-43.9694885888	-117.5251835904
0466A42E-47CF-8D24-FD01-C0B689713128	CcU-4219	b-2302	2021-07-26 07:29:18	49.53	0	97	170	-43.9694885888	-117.5251835904
0466A42E-47CF-8D24-FD01-C0B689713128	CcU-4219	b-2302	2021-07-26 07:29:18	49.53	0	43	170	-43.9694885888	-117.5251835904
063FBA79-99EC-66FB-29F7-25726D1764A5	CcU-2987	b-2250	2022-01-06 21:25:27	92.61	0	47	275	-81.222680576	-129.049879552
063FBA79-99EC-66FB-29F7-25726D1764A5	CcU-2987	b-2250	2022-01-06 21:25:27	92.61	0	67	275	-81.222680576	-129.049879552
063FBA79-99EC-66FB-29F7-25726D1764A5	CcU-2987	b-2250	2022-01-06 21:25:27	92.61	0	31	275	-81.222680576	-129.049879552
063FBA79-99EC-66FB-29F7-25726D1764A5	CcU-2987	b-2250	2022-01-06 21:25:27	92.61	0	5	275	-81.222680576	-129.049879552
0668296C-CDB9-A883-76BC-2E4C44F8C8AE	CcU-3743	b-2618	2022-01-26 02:07:14	394.18	0	89	265	-34.3593055232	-100.555928064
0668296C-CDB9-A883-76BC-2E4C44F8C8AE	CcU-3743	b-2618	2022-01-26 02:07:14	394.18	0	83	265	-34.3593055232	-100.555928064
0668296C-CDB9-A883-76BC-2E4C44F8C8AE	CcU-3743	b-2618	2022-01-26 02:07:14	394.18	0	79	265	-34.3593055232	-100.555928064
06CD9AA5-9B42-D684-DDDD-A5E394FEBA99	CcU-2959	b-2346	2021-10-26 23:00:01	279.93	0	43	92	33.7381445632	158.298210304
06CD9AA5-9B42-D684-DDDD-A5E394FEBA99	CcU-2959	b-2346	2021-10-26 23:00:01	279.93	0	31	92	33.7381445632	158.298210304
07A46D48-31A3-7E87-65B9-0DA902AD109F	CcU-3225	b-2386	2021-06-28 21:11:42	340.87	1	47	272	38.8341526528	92.190545408
07A46D48-31A3-7E87-65B9-0DA902AD109F	CcU-3225	b-2386	2021-06-28 21:11:42	340.87	1	23	272	38.8341526528	92.190545408
09DE92CE-6F27-2BB7-13B5-9385B2B3B8E2	CcU-3071	b-2298	2021-05-11 20:40:06	303.05	1	67	275	71.1705612288	10.5756752896
09DE92CE-6F27-2BB7-13B5-9385B2B3B8E2	CcU-3071	b-2298	2021-05-11 20:40:06	303.05	1	7	275	71.1705612288	10.5756752896

transaction_products_expanded 7 ×

Result Grid | Form Editor | Field Types | Query Stats | Execution Plan | Read Onl

Aquí muestro el resto del código y su resultado.

El código hace un JOIN con productosentre transaction_products_expanded y la tabla products para obtener los nombres de los productos junto con sus IDs.

Agrupación y conteo: Luego, agrupa los resultados por product_id para contar cuántas veces cada producto fue vendido (unitats_venudes).

Ordenación: Finalmente, los resultados se ordenan alfabéticamente por product_name.

```
INSERT INTO transaction_products_expanded (id, card_id, business_id, timestamp, amount, declined, product_id, user_id, lat, longitude)
SELECT
    transactions.id,
    transactions.card_id,
    transactions.business_id,
    transactions.timestamp,
    transactions.amount,
    transactions.declined,
    SUBSTRING_INDEX(SUBSTRING_INDEX(transactions.product_ids, ',', numbers.n), ',', -1) as product_id,
    transactions.user_id,
    transactions.lat,
    transactions.longitude
FROM
  (SELECT 1 n UNION ALL SELECT 2 UNION ALL SELECT 3 UNION ALL SELECT 4 UNION ALL
   SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7 UNION ALL SELECT 8 UNION ALL
   SELECT 9 UNION ALL SELECT 10) numbers INNER JOIN transactions
  ON CHAR_LENGTH(transactions.product_ids)
     - CHAR_LENGTH(REPLACE(transactions.product_ids, ',', '')) >= numbers.n - 1
ORDER BY
  transactions.id,
  numbers.n;

select product_id, p.product_name, count(product_id) as unitats_venudes from transaction_products_expanded as te
join products as p on te.product_id  = p.id
group by product_id
order by p.product_name asc;
```

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

product_id	product_name	unitats_venudes
37	Direwolf Littlefinger	23
79	Direwolf riverlands the	31
59	Direwolf Stannis	20
1	Direwolf Stannis	22
19	dooku solo	20