Exercici 1

He descargado los archivos csv y he importado todos menos el de productos que ese nos lo piden más adelante. He creado las tablas de nuevo ya que me estaba dando bastantes errores a la hora de insertar las filas adaptandolas a las nuevas columnas del csv.

```sql
CREATE TABLE credit_card (
    id VARCHAR(20) PRIMARY KEY,
    user_id INT,
    iban VARCHAR(34),
    pan VARCHAR(20),
    pin INT,
    cvv INT,
    track1 VARCHAR(255),
    track2 VARCHAR(255),
    expiring_date DATE
);

CREATE TABLE user (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    surname VARCHAR(100),
    phone VARCHAR(50),
    email VARCHAR(150),
    birth_date VARCHAR(10),
    country VARCHAR(100),
    city VARCHAR(100),
    postal_code VARCHAR(20),
    address VARCHAR(255)
);
```

En el .sql esta el codigo completo de la creación de tablas , pero como es parecido a la Tarea 3 solo pongo esta parte en el informe.

He insertado el csv mediante scripts tipo:

LOAD DATA LOCAL INFILE 'C:\\Users\\jordi\\Downloads\\transactions.csv'
INTO TABLE transaction
FIELDS TERMINATED BY ';'
ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(id, card_id, business_id, `timestamp`, amount, declined, product_ids, user_id, lat, longitude);

Ya que el wizzard de inserción me iba excesivamente lento (unos 3s por fila).

Una vez ya estaba todo insertado procedo ha hacer la query mediante un HAVING COUNT para contabilizar la cantidad de usuarios con más de 80 transacciones.

```sql
-- NIVELL 1
-- ECERCICI 1


SELECT u.*
FROM user u
WHERE u.id IN (
    SELECT t.user_id
    FROM transaction t
    GROUP BY t.user_id
    HAVING COUNT(*) > 80
```

Exercici 2

```sql
SELECT
    cc.iban,
    AVG(t.amount) AS average_amount
FROM
    transaction t
JOIN
    credit_card cc ON t.card_id = cc.id
JOIN
    company 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	average_amount
XX9114064011255863075868805	356.246667
SK9446370242474562577506	142.960000
XX7767529178459529755556 40	257.370000
XX4138273622897193049089 90	139.590000
XX3477872460707696107803 08	240.410000
XX6887684365430908948546 02	188.580000
MK2836885153868349	439.390000

Result 16

Output

Action Output

#	Time	Action	Message
22	10:18:44	SELECT u.* FROM user u WHERE u.id IN (SELECT t.user_id FROM transaction t GROUP BY t.user_i...	4 row(s) returned
23	10:26:07	SELECT cc.iban, AVG(t.amount) AS average_amount FROM transaction t JOIN credit_card cc ON...	371 row(s) returned

Se seleccionan solo las transacciones de “Donec Ltd”, se enlazan con las tarjetas de crédito para obtener el IBAN de cada una, y se calcula la media de los importes por cada IBAN. Finalmente, se agrupan todos estos resultados por IBAN para ver claramente cuál es el promedio de gastos por cada tarjeta.

Nivell 2

Exercici 1

Primero creamos una nueva tabla para guardar el card_id (identificador único) y un booleano para apuntar si las 3 últimas transacciones han sido declinadas o no

```
123     -- NIVELL 2
124     -- EXERCICI 1
125
126 •   CREATE TABLE credit_card_status (
127         card_id VARCHAR(50) PRIMARY KEY,
128         last_three_declined BOOLEAN NOT NULL
129     );
130
```

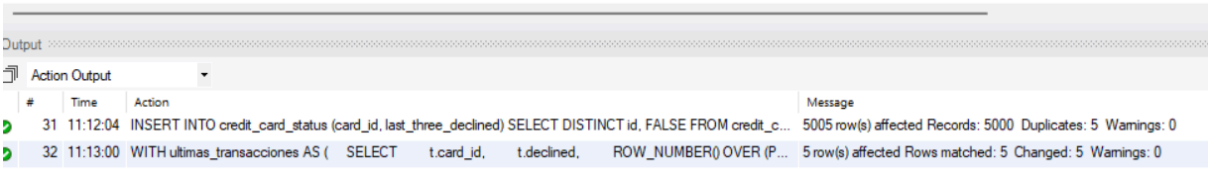
Ahora que hemos creado la tabla, lo que he hecho ha sido meter todas las filas de credit_card a la nuevo tabla y poniendo por default last_three_declined=False.

```
131     -- METEMOS TODAS LAS TARGETAS
132
133 •   INSERT INTO credit_card_status (card_id, last_three_declined)
134     SELECT DISTINCT id, FALSE
135     FROM credit_card
136     ON DUPLICATE KEY UPDATE last_three_declined = FALSE;
```

Una vez tenemos todos los registros de tarjetas de crédito con este Script vemos si las 3 últimas transacciones fueron aprobada y en caso positivo modificamos la fila en cuestión para poner a TRUE last_three_declined

```sql
-- MODIFICAMOS EL BOOLEANO SEGUN LAS 3 ULTIMAS TRANSACCIONES:

WITH ultimas_transacciones AS (
    SELECT
        t.card_id,
        t.declined,
        ROW_NUMBER() OVER (PARTITION BY t.card_id ORDER BY t.timestamp DESC) AS rn
    FROM transaction t
),
filtradas AS (
    SELECT card_id
    FROM ultimas_transacciones
    WHERE rn <= 3
    GROUP BY card_id
    HAVING SUM(declined) = 3
)
UPDATE credit_card_status
JOIN filtradas USING (card_id)
SET last_three_declined = TRUE;
```



Primero, se crea una especie de tabla temporal llamada "ultimas_transacciones" donde se seleccionan todas las transacciones y se asigna un número a cada una por tarjeta, ordenando desde la más reciente a la más antigua. Esto sirve para identificar las tres últimas transacciones de cada tarjeta. Después, se filtran estas transacciones para quedarse solo con las tres primeras de cada tarjeta y se agrupan por tarjeta, calculando la suma de los valores de "declined" (que indica si la transacción fue rechazada). Si la suma es tres, quiere decir que las tres últimas transacciones fueron todas declinadas.

Nivell 3

Exercici 1

Creamos la tabla products basandonos en los inputs del csv y importamos la información.

```sql
CREATE TABLE product (
    id INT PRIMARY KEY,
    product_name VARCHAR(255),
    price DECIMAL(10, 2),
    colour VARCHAR(20),
    weight DECIMAL(10, 2),
    warehouse_id VARCHAR(20)
);
ALTER TABLE product MODIFY price VARCHAR(20);
```

Como vemos que en transactions se referencian en product_ids a varios elementos a la vez, no podemos simplemente enlazarla con products así como sí nada. Para ello tenemos que crear una tabla intermedia donde se relacione de forma individual el transaction_id y el product_id.

```sql
CREATE TABLE transaction_product (
    transaction_id VARCHAR(50),  -- o el tipo que sea id en transaction
    product_id INT,
    PRIMARY KEY (transaction_id, product_id),
    FOREIGN KEY (transaction_id) REFERENCES transaction(id),
    FOREIGN KEY (product_id) REFERENCES product(id)
);
```

Ahora tenemos que rellenar esta tabla con la información que hay en transactions para normalizar los elementos de product_ids y poder relacionarlos correctamente con la tabla products . Como puede haber diferente número de elementos en product_ids , tenemos que hacer un algoritmo que tenga esto en cuenta y vaya añadiendo de forma continua cada elemento de product_ids en una linea en la tabla intermedia junto con el transaction_id.

```sql
-- RELLENAMOS LA TABLA INTERMEDIA A PARTIR DE TRANSACTION
INSERT INTO transaction_product (transaction_id, product_id)
SELECT
    t.id,
    CAST(TRIM(j.value) AS UNSIGNED) AS product_id
FROM
    transaction t,
    JSON_TABLE(
        CONCAT('[', REPLACE(t.product_ids, ',', ','), ']'),
        "$[*]" COLUMNS(value VARCHAR(10) PATH "$")
    ) j;
```

Una vez hecho esto ya tenemos la tabla intermedia rellenada y entrelazada con las tablas transaction y product , por lo tanto ya podemos hacer la query que nos decia el enunciado.

```sql
-- BUSCAMOS EN LA TABLA INTERMEDIA

SELECT
    product_id,
    COUNT(*) AS nombre_vendes
FROM transaction_product
GROUP BY product_id
ORDER BY nombre_vendes DESC;
```

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

product_id	nombre_vendes
52	2654
29	2635
21	2609
16	2608
66	2601

Result 28

Output

Action Output

#	Time	Action	Message
63	12:30:47	SELECT * FROM transaction_product WHERE transaction_id="CDDA7E40-544D-47BB-A4ED-671DD8A950D...	3 row(s) returned
64	12:45:06	SELECT product_id, COUNT(*) AS nombre_vendes FROM transaction_product GROUP BY product_id ...	100 row(s) returned

Hacemos la búsqueda de forma sencilla ya que simplemente es agrupar por product_id y contarlos. Sabemos que es correcto debido a que el csv de products tenia 100 elementos y justo nos ha devuelto 100 filas como se puede ver abajo a la derecha de la captura.