

## Exercici 1

Realitza una subconsulta que mostri tots els usuaris amb més de 80 transaccions utilitzant almenys 2 taules.

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.

```
1 ● CREATE TABLE credit_card (  
2     id VARCHAR(20) PRIMARY KEY,  
3     user_id INT,  
4     iban VARCHAR(34),  
5     pan VARCHAR(20),  
6     pin INT,  
7     cvv INT,  
8     track1 VARCHAR(255),  
9     track2 VARCHAR(255),  
10    expiring_date DATE  
11 );  
12  
13 ● CREATE TABLE user (  
14     id INT PRIMARY KEY,  
15     name VARCHAR(100),  
16     surname VARCHAR(100),  
17     phone VARCHAR(50),  
18     email VARCHAR(150),  
19     birth_date VARCHAR(10),  
20     country VARCHAR(100),  
21     city VARCHAR(100),  
22     postal_code VARCHAR(20),  
23     address VARCHAR(255)  
24 );
```

En el .sql esta el código 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 a hacer la query mediante un HAVING COUNT

para contabilizar la cantidad de usuarios con más de 80 transacciones.

```

92 -- NIVELL 1
93 -- ECERCICI 1
94
95
96 SELECT u.*
97 FROM user u
98 WHERE u.id IN (
99     SELECT t.user_id
100 FROM transaction t
101 GROUP BY t.user_id
102 HAVING COUNT(*) > 80
103 );
104
105
106

```

Result Grid

id	name	surname	phone	email	birth_date	country	city	postal_code	address
185	Molly	Gilliam	0800 120 8023	donec@outlook.co.uk	Dec 21, 1993	United Kingdom	London	EC1A 1BB	P.O. Box 202, 5638 MI Rd.
289	Dxwgi	Hwrcu	+98-309-8797	dxwgi.hwrcu@example.com	Aug 20, 1976	Germany	Stuttgart	70173	82 Hwrcu Street
318	Bnyr	Astuw	+33-120-9644	bnyr.astuw@example.com	May 3, 1974	Italy	Genoa	16100	53 Astuw Street
454	Sfzoh	Xgvfrdxs	+58-495-3945	sfzoh.xgvfrdxs@example.com	Aug 28, 1962	Poland	Gdansk	80-001	52 Xgvfrdxs Street

ser 15 x

Output

Action Output

#	Time	Action	Message
21	10:16:59	SELECT COUNT(*) FROM transaction LIMIT 0, 1000	1 row(s) returned
22	10:18:44	SELECT u.* FROM user u WHERE u.id IN ( SELECT t.user_id FROM transaction t GROUP BY t.user_id...	4 row(s) returned

## Exercici 2

Mostra la mitjana d'amount per IBAN de les targetes de crèdit a la companyia Donec Ltd, utilitza almenys 2 taules.

```

107
108 • SELECT
109     cc.iban,
110     AVG(t.amount) AS average_amount
111 FROM
112     transaction t
113 JOIN
114     credit_card cc ON t.card_id = cc.id
115 JOIN
116     company c ON t.business_id = c.company_id
117 WHERE
118     c.company_name = 'Donec Ltd'
119 GROUP BY
120     cc.iban;

```

Result Grid

iban	average_amount
XX911406401125586307586805	356.246667
SK9446370242474562577506	142.960000
XX776752917845952975555640	257.370000
XX413827362289719304908990	139.590000
XX347787246070769610780308	240.410000
XX688768436543090894854602	188.580000
MX728368851538688740	479.390000

Result 16 x

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_id...	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

Quantes targetes estan actives?

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.

```
L31      -- METEMOS TODAS LAS TARGETAS
L32
L33 • INSERT INTO credit_card_status (card_id, last_three_declined)
L34     SELECT DISTINCT id, FALSE
L35     FROM credit_card
L36     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

```

138 -- MODIFICAMOS EL BOOLEANO SEGUN LAS 3 ULTIMAS TRANSACCIONES:
139
140 WITH ultimas_transacciones AS (
141     SELECT
142         t.card_id,
143         t.declined,
144         ROW_NUMBER() OVER (PARTITION BY t.card_id ORDER BY t.timestamp DESC) AS rn
145     FROM transaction t
146 ),
147 filtradas AS (
148     SELECT card_id
149     FROM ultimas_transacciones
150     WHERE rn <= 3
151     GROUP BY card_id
152     HAVING SUM(declined) = 3
153 )
154 UPDATE credit_card_status
155 JOIN filtradas USING (card_id)
156 SET last_three_declined = TRUE;
157
158

```

Output

Action Output

#	Time	Action	Message
31	11:12:04	INSERT INTO credit_card_status (card_id, last_three_declined) SELECT DISTINCT id, FALSE FROM credit_c...	5005 row(s) affected Records: 5000 Duplicates: 5 Warnings: 0
32	11:13:00	WITH ultimas_transacciones AS ( SELECT t.card_id, t.declined, ROW_NUMBER() OVER (P...	5 row(s) affected Rows matched: 5 Changed: 5 Warnings: 0

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

Necessitem conèixer el nombre de vegades que s'ha venut cada producte.

```

163 • CREATE TABLE product (
164     id INT PRIMARY KEY,
165     product_name VARCHAR(255),
166     price DECIMAL(10, 2),
167     colour VARCHAR(20),
168     weight DECIMAL(10, 2),
169     warehouse_id VARCHAR(20)
170 );
171 • 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.

```

175 • CREATE TABLE transaction_product (
176     transaction_id VARCHAR(50), -- o el tipo que sea id en transaction
177     product_id INT,
178     PRIMARY KEY (transaction_id, product_id),
179     FOREIGN KEY (transaction_id) REFERENCES transaction(id),
180     FOREIGN KEY (product_id) REFERENCES product(id)
181 );
182
183

```

Output			
Action Output			
#	Time	Action	Message
57	11:38:22	SELECT id FROM product LIMIT 0, 1000	100 row(s) returned
58	11:45:42	CREATE TABLE transaction_product ( transaction_id VARCHAR(50), -- o el tipo que sea id en transaction	0 row(s) affected

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 línea en la tabla intermedia junto con el transaction\_id.

```

185 -- RELLENAMOS LA TABLA INTERMEDIA A PARTIR DE TRANSACTION
186 • INSERT INTO transaction_product (transaction_id, product_id)
187     SELECT
188         t.id,
189         CAST(TRIM(j.value) AS UNSIGNED) AS product_id
190     FROM
191         transaction t,
192         JSON_TABLE(
193             CONCAT('[', REPLACE(t.product_ids, ',', ', '), ']'),
194             "$[*]" COLUMNS(value VARCHAR(10) PATH "$")
195         ) j;

```

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

```

197 -- BUSCAMOS EN LA TABLA INTERMEDIA
198
199 • SELECT
200     product_id,
201     COUNT(*) AS nombre_vendes
202 FROM transaction_product
203 GROUP BY product_id
204 ORDER BY nombre_vendes DESC;
205
206

```

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.