

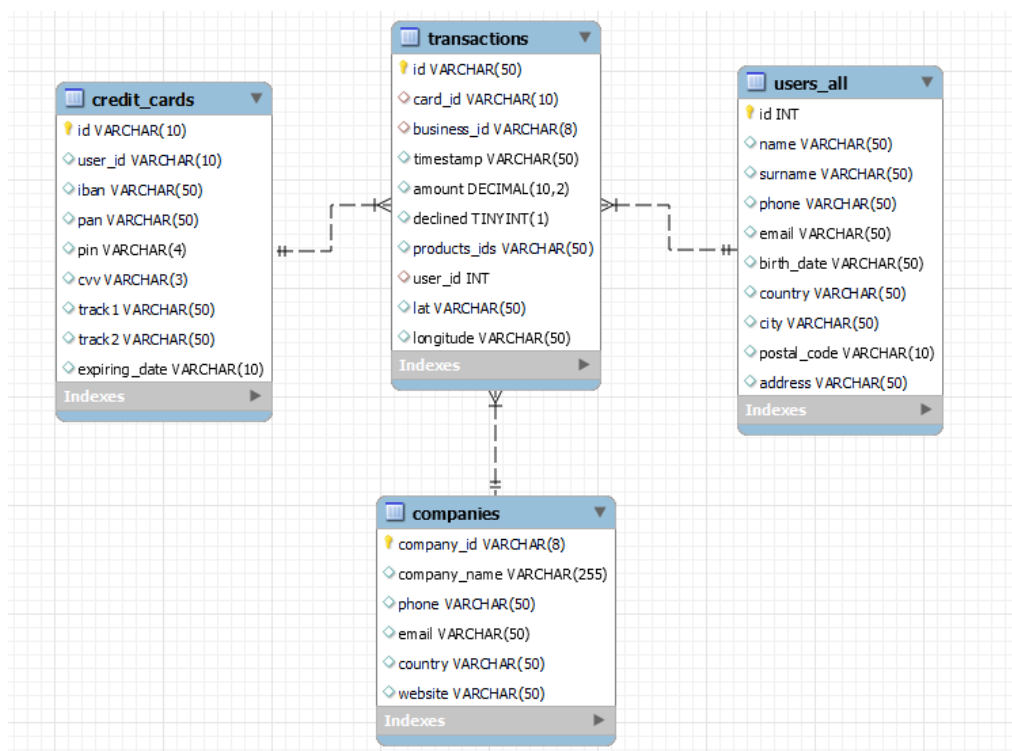
Nivel 1

La **base de datos** creada se llama **“TransactionsT4”**. Inicialmente tendrá 4 tablas:

- Companies
- Credit_cards
- Users_all
- Transactions

Las tablas siguen un esquema de tipo estrella, con la tabla **Transactions como tabla de hechos** y las otras como tabla de dimensiones. Todas estas tienen una **relación de 1 a N con la tabla central**, ya que cada compañía, usuario y tarjeta de crédito puede realizar muchas transacciones.

DB TransactionsT4



Para una explicación de la creación de la base de datos y de las tablas ver la sección ANEXO al final del documento.

Ejercicio 1

Los usuarios con más de 30 transacciones son los que tienen los siguientes ID:

- 92
- 267
- 272
- 275

The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL code:

```
64
65  #Nivell 1
66  ##Exercici 1: subconsulta que mostri tots els usuaris amb més de 30 transaccions utilitzant almenys 2 taules.
67  • SELECT id
68    FROM users_all
69  WHERE id IN (
70      SELECT user_id
71      FROM transactions
72      GROUP BY user_id
73      HAVING COUNT(transactions.id) >= 30);
74
```

The results pane shows a table with the following data:

id
92
267
272
275

The output pane shows the following messages:

#	Time	Action	Message
101	12:41:51	SELECT * FROM users_all	275 row(s) returned
102	12:43:41	SELECT id FROM users_all WHERE id IN (SELECT user_id FROM transactions GR...	4 row(s) returned

Ejercicio 2

La media de la suma de transacciones por IBAN de las tarjetas de crédito de la compañía Donec Ltd. es de 407.43.

Para obtener este resultado, primero obtenemos la suma de las transacciones agrupadas por Id de compañía e IBAN en una tabla temporal y, luego, calculamos el promedio de esas sumas para la compañía Donec.

SQL Studio interface showing a query in the editor and its results in the Result Grid.

Query (lines 81-93):

```

81  ##N1. Exercici 2: Mostra la mitjana de la suma de transaccions per IBAN de les targetes de crèdit en la compar
82  • WITH tt AS
83  (
84    (SELECT business_id, sum(amount) AS suma_trans
85     FROM transactions AS t
86     JOIN credit_cards AS cc
87     ON t.card_id=cc.id
88     GROUP BY business_id, iban)
89    SELECT AVG(suma_trans)
90    FROM tt
91    WHERE business_id IN (
92      SELECT company_id
93      FROM companies
94      WHERE company_name LIKE 'donec ltd%');

```

Result Grid:

AVG(suma_trans)
407.430000

Action Output:

#	Time	Action	Message
128	13:15:03	WITH tt AS (SELECT business_id, sum(amount) AS suma_trans FROM transactions A...	1 row(s) returned

Otra opción de hacerlo, también con tabla temporal, pero sin subquery y sólo joins sería la siguiente:

SQL Studio interface showing an alternative query using joins and its results in the Result Grid.

Query (lines 94-106):

```

94
95  ##otra opción con tabla temporal, pero sin subquery y solo joins
96  • WITH tt AS(
97    SELECT business_id, sum(amount) AS suma_trans
98    FROM transactions AS t
99    JOIN credit_cards AS cc
100    ON t.card_id=cc.id
101    JOIN companies AS c
102    ON c.company_id = t.business_id
103    WHERE company_name LIKE 'Donec Ltd%'
104    GROUP BY t.business_id, cc.iban)
105    SELECT AVG(suma_trans)
106    FROM tt;

```

Result Grid:

AVG(suma_trans)
407.430000

Action Output:

#	Time	Action	Message
133	13:24:40	WITH tt AS (SELECT business_id, sum(amount) AS suma_trans FROM transactions A...	1 row(s) returned

Nivel 2

Antes de crear la tabla, cambiaremos el nombre del campo timestamp a fecha_hora para evitar errores y luego le daremos formato de tipo datetime para poder ordenar las transacciones por fecha.

```

106 • ALTER TABLE transactions CHANGE timestamp fecha_hora VARCHAR(50);
107 • SET SQL_SAFE_UPDATES = 1;
108
109 #damos formato correcto de tipo datetime al campo fecha_hora para poder ordenar por este valor
110 • SET SQL_SAFE_UPDATES = 0;
111 • UPDATE transactions
112   SET fecha_hora = STR_TO_DATE(fecha_hora, '%d/%m/%Y %H:%i');
113 • SET SQL_SAFE_UPDATES = 1;
114
115 • DESCRIBE transactions;
116 • ALTER TABLE transactions MODIFY fecha_hora DATETIME;
117 • DESCRIBE transactions;

```

Field	Type	Null	Key	Default	Extra
id	varchar(50)	NO	PRI		
card_id	varchar(10)	YES	MUL		
business_id	varchar(8)	YES	MUL		
fecha_hora	datetime	YES			
amount	decimal(10,2)	YES			
declined	tinyint(1)	YES			

Result 11 x

Output

#	Time	Action	Message
25	10:47:33	ALTER TABLE transactions MODIFY fecha_hora DATETIME	587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0
26	10:47:48	DESCRIBE transactions	10 row(s) returned

Creamos la tabla para poder hacer la consulta. La función row_number nos permite asignar un número a cada transacción, partition by nos permite hacer una partición del conteo y recomenzarlo por cada tarjeta y con order by podemos ordenar las transacciones por fecha realizada.

```

122 • CREATE TABLE card_status AS (
123   WITH rn_table AS (
124     SELECT t.card_id, t.fecha_hora, t.declined,
125            ROW_NUMBER () OVER (PARTITION BY card_id ORDER BY fecha_hora DESC) AS rn
126   FROM transactions AS t)
127   SELECT *,
128          CASE
129            WHEN declined = 1 AND rn < 4 AND
130              (SELECT SUM(declined) FROM rn_table AS inner_table WHERE inner_table.card_id = rn_table.card_id) >= 3
131            THEN 'Rechazada'
132            ELSE 'Aceptada'
133          END AS Status
134   FROM rn_table);

```

Field	Type	Null	Key	Default	Extra
card_id	varchar(10)	YES			
fecha_hora	datetime	YES			
declined	tinyint(1)	YES			
rn	bigint unsigned	YES	0		
Status	varchar(9)	NO			

Result 65 x

Output

#	Time	Action	Message
147	16:02:25	CREATE TABLE card_status AS (WITH m_table AS (SELECT t.card_id, t.fecha_hor...	587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0

Por último, añadimos una columna “Status” que analiza cada caso (CASE WHEN) y asigna el status “Rechazado” sólo si la suma de las últimas tres transacciones es igual o mayor que 3, es decir si las últimas 3 transacciones han sido declinadas para el mismo número de card_id.

Luego corroboramos la creación de la tabla:

SQL Editor:

```

136 #Corroboramos la creación de la tabla
137 • DESCRIBE card_status;
138
139 • SELECT * FROM card_status;
140

```

Result Grid:

card_id	fecha_hora	declined	rn	Status
CcU-2938	2022-03-12 09:23:00	0	1	Aceptada
CcU-2938	2022-03-09 20:53:00	0	2	Aceptada
CcU-2938	2022-02-24 11:01:00	0	3	Aceptada
CcU-2938	2021-10-24 01:29:00	0	4	Aceptada
CcU-2938	2021-10-17 03:52:00	0	5	Aceptada
CcU-2938	2021-09-28 02:24:00	0	6	Aceptada
CcU-2938	2021-09-24 08:33:00	0	7	Aceptada
CcU-2938	2021-09-18 00:31:00	0	8	Aceptada
CcU-2938	2021-09-15 05:23:00	0	9	Aceptada
CcU-2938	2021-08-28 23:42:00	0	10	Aceptada
CcU-2938	2021-07-27 17:14:00	0	11	Aceptada
CcU-2938	2021-07-25 12:34:00	0	12	Aceptada
CcU-2938	2021-07-18 00:30:00	0	13	Aceptada

card_status 66 x

Output:

#	Time	Action	Message
148	16:03:08	DESCRIBE card_status	5 row(s) returned
149	16:06:29	SELECT * FROM card_status	587 row(s) returned

Como todas las transacciones por tarjeta salen aceptadas, editaremos dos transacciones para que se cumpla el caso de rechazo, para ver si las analiza correctamente.

SQL Editor:

```

141
142 #-----PRUEBA PARA CORROBORAR QUE CASE FUNCIONA-----
143 #Cambiaremos a declined los datos de una tarjeta para que cumpla las condiciones de rechazo y verificar CASE
144 • DROP TABLE card_status;
145
146 #averiguamos el id de las últimas tres transacciones de la compañía con card_id CcU-2938 para cambiarlas a dec
147 • SELECT * FROM transactions
148 WHERE card_id LIKE '%2938'
149 ORDER BY fecha_hora DESC;
150
151 #Hacemos que las últimas tres transacciones sean declinadas
152 • SET SQL_SAFE_UPDATES = 0;
153 • UPDATE transactions
154 SET declined = 1

```

Result Grid:

card_id	fecha_hora	declined	rn	Status
CcU-2938	2022-03-12 09:23:00	0	1	Aceptada
CcU-2938	2022-03-09 20:53:00	0	2	Aceptada
CcU-2938	2022-02-24 11:01:00	0	3	Aceptada
CcU-2938	2021-10-24 01:29:00	0	4	Aceptada
CcU-2938	2021-10-17 03:52:00	0	5	Aceptada
CcU-2938	2021-09-28 02:24:00	0	6	Aceptada
CcU-2938	2021-09-24 08:33:00	0	7	Aceptada

card_status 67 x

Comprobamos con las transacciones editadas que el código funciona:

```

177 CREATE TABLE card_status AS (
178 WITH rn_table AS (
179     SELECT t.card_id, t.fecha_hora, t.declined,
180     ROW_NUMBER () OVER (PARTITION BY card_id ORDER BY fecha_hora DESC) AS rn
181 FROM transactions AS t)
182 SELECT *,
183 CASE
184     WHEN declined = 1 AND rn < 4 AND
185     (SELECT SUM(declined) FROM rn_table AS inner_table WHERE inner_table.card_id = rn_table.card_id) >= 3
186 THEN 'Rechazada'
187 ELSE 'Aceptada'
188 END AS Status
189 FROM rn_table

```

card_id	fecha_hora	declined	rn	Status
CcU-2938	2022-03-12 09:23:00	1	1	Rechazada
CcU-2938	2022-03-09 20:53:00	1	2	Rechazada
CcU-2938	2022-02-24 11:01:00	1	3	Rechazada
CcU-2938	2021-10-24 01:29:00	0	4	Aceptada
CcU-2938	2021-10-17 03:52:00	0	5	Aceptada

card_status 60

Output

#	Time	Action	Message
132	15:44:21	CREATE TABLE card_status AS (WITH m_table AS (SELECT t.card_id, t.fecha_hora, t.declined, ROW_NUMBER () OVER (PARTITION BY card_id ORDER BY fecha_hora DESC) AS rn FROM transactions AS t) SELECT *, CASE WHEN declined = 1 AND rn < 4 AND (SELECT SUM(declined) FROM m_table AS inner_table WHERE inner_table.card_id = m_table.card_id) >= 3 THEN 'Rechazada' ELSE 'Aceptada' END AS Status FROM m_table)	587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0
133	15:44:48	SELECT * FROM card_status	587 row(s) returned

Volvemos a colocar la tabla Transactions con sus datos originales y creamos la tabla Card_status sabiendo que funciona correctamente, cambiamos la nomenclatura de case a Activa/Inactiva en lugar de Aceptada/ Rechazada:

```

195 #volvemos a dejar los datos de la tabla transaccion como estaban originalmente, eliminamos y creamos de nuevo
196 DROP TABLE card_status;
197
198 UPDATE transactions
199 SET declined = 0
200 WHERE id IN ('AD85A78A-8829-5746-93A0-8B7A792EBC18', 'F1A598A2-86C5-50A9-F1CE-FB1D69866C39', '55166D02-D74C-6A
201
202 SET SQL_SAFE_UPDATES = 1;
203
204 #Creamos la tabla card_status sabiendo que funciona correctamente:
205 CREATE TABLE card_status AS (
206 WITH rn_table AS (
207     SELECT t.card_id, t.fecha_hora, t.declined,
208     ROW_NUMBER () OVER (PARTITION BY card_id ORDER BY fecha_hora DESC) AS rn
209 FROM transactions AS t)
210 SELECT *,
211 CASE

```

Output

#	Time	Action	Message
134	15:48:33	DROP TABLE card_status	0 row(s) affected
135	15:49:05	UPDATE transactions SET declined = 0 WHERE id IN (AD85A78A-8829-5746-93A0-8B7A792EBC18, F1A598A2-86C5-50A9-F1CE-FB1D69866C39, 55166D02-D74C-6A)	3 row(s) affected Rows matched: 3 Changed: 3 Warnings: 0
136	15:49:08	SET SQL_SAFE_UPDATES = 1	0 row(s) affected
137	15:49:54	CREATE TABLE card_status AS (WITH m_table AS (SELECT t.card_id, t.fecha_hora, t.declined, ROW_NUMBER () OVER (PARTITION BY card_id ORDER BY fecha_hora DESC) AS rn FROM transactions AS t) SELECT *, CASE WHEN declined = 1 AND rn < 4 AND (SELECT SUM(declined) FROM m_table AS inner_table WHERE inner_table.card_id = m_table.card_id) >= 3 THEN 'Rechazada' ELSE 'Aceptada' END AS Status FROM m_table)	587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0

Girololi - S04* x

```

188 #Creamos la tabla card_status sabiendo que funciona correctamente:
189 CREATE TABLE card_status AS (
190   WITH rn_table AS (
191     SELECT t.card_id, t.fecha_hora, t.declined,
192     ROW_NUMBER () OVER (PARTITION BY card_id ORDER BY fecha_hora DESC) AS rn
193   FROM transactions AS t)
194   SELECT *,
195   CASE
196     WHEN declined = 1 AND rn < 4 AND
197     (SELECT SUM(declined) FROM rn_table AS inner_table WHERE inner_table.card_id = rn_table.card_id) >= 3
198   THEN 'Inactiva'
199   ELSE 'Activa'
200   END AS Status
201 FROM rn_table);

```

Result Grid

card_id	fecha_hora	declined	rn	Status
CcU-2938	2022-03-12 09:23:00	0	1	Activa
CcU-2938	2022-03-09 20:53:00	0	2	Activa
CcU-2938	2022-02-24 11:01:00	0	3	Activa
CcU-2938	2021-10-24 01:29:00	0	4	Activa
CcU-2938	2021-10-17 03:52:00	0	5	Activa

card_status 4 x

Output

Action Output

#	Time	Action	Message
5	10:11:08	SELECT COUNT(distinct card_id) FROM card_status WHERE status = 'Activa'	1 row(s) returned

Ejercicio 1

Hay 275 tarjetas activas, es decir todas las tarjetas. Ya que de las 587 transacciones realizadas, 87 fueron declinadas, pero cada operación declinada ha sido con diferentes tarjetas. Con lo que ninguna tarjeta ha tenido las últimas tres transacciones declinadas como para considerarse rechazada.

Girololi - S04* x

```

204
205 #-----
206
207 ##N2: Ejercici 1: Quantes targetes estan actives?
208
209 SELECT COUNT(distinct card_id)
210 FROM card_status
211 WHERE status = 'Activa';
212

```

Result Grid

COUNT(distinct card_id)
275

Result 5 x

Output

Action Output

#	Time	Action	Message
5	10:11:08	SELECT COUNT(distinct card_id) FROM card_status WHERE status = 'Activa'	1 row(s) returned
6	10:12:19	SELECT * FROM card_status	587 row(s) returned

Nivel 3

Primero creamos la tabla Products e importamos los datos:

The screenshot shows the Giori application interface. The top part is a SQL editor with a blue header bar containing the text "Giori - S04" and "Giori - S03". The editor contains the following SQL code:

```

213
214 ##Nivel 3
215
216 CREATE TABLE products (
217     id VARCHAR(50) PRIMARY KEY,
218     product_name VARCHAR(50),
219     price VARCHAR(50),
220     colour VARCHAR(50),
221     weight VARCHAR(50),
222     warehouse_id VARCHAR(50));
223
224 SELECT * FROM products;
225

```

Below the editor is a toolbar with icons for "Result Grid", "Filter Rows", "Edit", "Export/Import", and "Wrap Cell Content". The "Result Grid" is active, showing a table with the following data:

id	product_name	price	colour	weight	warehouse_id
1	Direwolf Stannis	\$161.11	#7c7c7c	1	WH-4
10	Karstark Dorne	\$119.52	#f4f4f4	2.4	WH--5
100	south duel	\$40.43	#6d6d6d	3	WH--95
11	Karstark Dorne	\$49.70	#141414	2.7	WH--6
12	duel Direwolf	\$181.60	#a8a8a8	2.1	WH--7

Below the results grid is a section labeled "products 7" with a close button. Below that is an "Output" section with a dropdown menu set to "Action Output". The output shows the following actions:

#	Time	Action	Message
15	10:45:15	DEALLOCATE PREPARE stmt	OK
16	10:45:32	SELECT * FROM products	100 row(s) returned

Luego creamos la tabla puente “Products_per_transactions” entre Products y Transactions para romper la relación N-N entre ambas.

The screenshot shows the SQL Server Enterprise Manager interface. The top pane displays a script with the following SQL commands:

```

226
227
228 #Luego creamos la siguiente tabla puente para evitar la relación N-N entre las tablas Products y Transactions
229 • CREATE TABLE products_per_transactions (
230     id_transaction VARCHAR(50),
231     id_product VARCHAR(50));
232
233 #Comprobamos la creación de la tabla
234 • SELECT * FROM products_per_transactions;
235
236

```

The bottom pane shows the 'Result Grid' with the execution results of the 'CREATE TABLE' statement. The grid has two columns: 'id_transaction' and 'id_product'. The results show '0 row(s) affected'.

id_transaction	id_product

Esta tabla puente se compone de dos campos:

- uno es id_transaccion que contiene los id de cada transacción realizada y actúa como FK con la tabla Transacciones, con la que tiene una relación de N-1.
- Otro es id_product que contiene los id de cada producto y actúa como FK con la tabla Products, con la que tiene una relación de N-1.

Añadiremos las FK luego de insertar los datos en la tabla para facilitar la inserción.

A continuación, insertamos los datos en la tabla puente a partir de las columnas que ya existen en la tabla Transacciones.

The screenshot shows a SQL IDE window with a query editor and a result grid. The query is as follows:

```

238 • INSERT INTO products_per_transactions (id_transaction, id_product)
239 SELECT t.id AS id_transaction,
240        SUBSTRING_INDEX(SUBSTRING_INDEX(t.products_ids, ',', numbers.n), ',', -1) AS id_product #aqui separamos
241 FROM transactions t
242 JOIN (
243     SELECT ROW_NUMBER() OVER () AS n #aqui generamos una secuencia de números para cada fila de la tabla t
244     FROM transactions
245     CROSS JOIN (SELECT 0 AS n UNION ALL SELECT 1 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) AS numbers
246     #aqui hacemos una cross join con una tabla de 10 filas para asegurarnos que habrá sitio para al menos 10
247 ) AS numbers
248 ON numbers.n <= LENGTH(t.products_ids) - LENGTH(REPLACE(t.products_ids, ',', '')) + 1; #aqui calculamos la cantidad de productos
249
250 #Comprobamos la inserción correcta de los datos
251 • SELECT * FROM products_per_transactions;
  
```

The result grid shows the following data:

id_transaction	id_product
FE96CE47-BD59-381C-4E18-E3CA3D44E8FF	3
FE809ED4-2DB6-55AC-C915-929516E4646B	23
FD9CBCCD-8E1E-8DA1-4606-7E3A6F3A5A65	37
FD89D51B-AE8D-77DC-E450-B8083FBD3187	3
FD2E8957-414B-BEEC-E9AD-59AA7A8A6290	83

The status bar at the bottom indicates: 1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0

Para ello, seguiremos los siguientes pasos:

- 1) Separamos cada producto del campo products_ids con la función **SUBSTRING_INDEX**, la aplicamos dos veces porque sino nos quedaría el primer producto sin separar
- 2) Hacemos una **JOIN** con una secuencia de números generada con la función **ROW_NUMBER** para cada fila de la tabla transactions
- 3) Hacemos una **CROSS JOIN** (es decir un producto cartesiano) con una tabla ficticia de 10 números, para asegurar que al menos habrá 10 filas disponibles para productos por cada transacción
- 4) La JOIN la hacemos teniendo en cuenta el cálculo de la cantidad de productos que hay en cada campo, para ello restamos la función **LENGTH**

la longitud de la cadena con comas menos la sin comas, para saber la cantidad de comas que hay y le sumamos uno.

Al comprobar la creación de la tabla encontramos que el campo id_product tiene espacios en blanco, los quitamos con la función TRIM:

```

250
251 #Comprobamos la inserción correcta de los datos
252 • SELECT * FROM products_per_transactions;
253
254 #descubrimos que hay espacios en blanco en el campo, los eliminamos con TRIM
255 • select length(id_product)
256   from products_per_transactions;
257
258 #eliminamos los espacios en blanco con la función TRIM
259 • SET SQL_SAFE_UPDATES = 0;
260 • UPDATE products_per_transactions
261   SET id_product = TRIM(id_product);
262 • SET SQL_SAFE_UPDATES = 1;
263
264
265
266
267
268
269

```

Output

#	Time	Action	Message
76	13:58:40	SET SQL_SAFE_UPDATES = 0	0 row(s) affected
77	13:58:42	UPDATE products_per_transactions SET id_product = TRIM(id_product)	870 row(s) affected Rows matched: 1457 Changed: 870 Warnings: 0
78	13:58:52	SET SQL_SAFE_UPDATES = 1	0 row(s) affected

Ahora creamos las FK, añadimos un índice en la tabla products porque sino no nos deja crearla:

```

263
264
265 #Ahora añadimos las FK
266 • ALTER TABLE products_per_transactions
267   ADD CONSTRAINT FOREIGN KEY (id_transaction) REFERENCES transactions(id);
268
269 ### Creamos un índice porque sino da error al querer crear la FK
270 • CREATE INDEX idx_products_id ON products(id);
271 • ALTER TABLE products_per_transactions
272   ADD CONSTRAINT FOREIGN KEY (id_product) REFERENCES products (id);
273
274
275
276
277
278
279
280
281

```

Output

#	Time	Action	Message
80	14:01:39	CREATE INDEX idx_products_id ON products(id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
81	14:01:44	ALTER TABLE products_per_transactions ADD CONSTRAINT FOREIGN KEY (id_pr...	1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0

Ejercicio 1

Para obtener un listado de la cantidad de veces que se ha vendido cada producto usamos la siguiente consulta:

Como el enunciado especifica productos “vendidos” tendremos en cuenta que la transaccion no haya sido declinada (declined = 0).

Por ejemplo en el primer renglón podemos ver que el producto con id = 1 se ha vendido 51 veces.

The screenshot shows a database query editor with a SQL query and its results. The query is as follows:

```

272
273 #N3: E1: Necessitem conèixer el nombre de vegades que s'ha venut cada produ
274 • SELECT id_product, COUNT(distinct id_transaction)
275 FROM products_per_transactions ppt
276 JOIN transactions t
277 ON ppt.id_transaction=t.id
278 WHERE declined=0
279 GROUP BY id_product;
280

```

The results are displayed in a table with the following data:

id_product	COUNT(distinct id_transaction)
1	51
11	40
13	51
17	54
19	44
2	56
23	60
29	43
3	12

The bottom of the screenshot shows the output log with the following entries:

#	Time	Action	Message
84	14:11:24	SELECT id_product, COUNT(distinct id_transaction) FROM products_per_transaction...	Error Code: 1140. In aggregated query without GROUP BY, expressio
85	14:11:38	SELECT id_product, COUNT(distinct id_transaction) FROM products_per_transaction...	26 row(s) returned

ANEXO

Para crear la base de datos y el modelo realizamos los siguientes comandos:

1) Creamos la Base de datos y la tabla companies

The screenshot shows a SQL script in a text editor with the following content:

```

1  #Creamos base de datos
2  • CREATE DATABASE transactionsT4;
3  • USE transactionsT4;
4
5  #Creamos tablas
6  • CREATE TABLE companies (
7      company_id VARCHAR(8) PRIMARY KEY,
8      company_name VARCHAR (255),
9      phone VARCHAR (50),
10     email VARCHAR (50),
11     country VARCHAR (50),
12     website VARCHAR (50)
13 );

```

Below the script, the 'Output' window shows the execution results:

#	Time	Action	Message
1	10:11:25	CREATE DATABASE transactionsT4	1 row(s) affected
2	10:12:26	USE transactionsT4	0 row(s) affected
3	10:21:35	CREATE TABLE companies (company_id VARCHAR(8) PRIMARY KEY, company_na...	0 row(s) affected
4	10:31:05	CREATE TABLE credit_cards (id VARCHAR (10) PRIMARY KEY, user_id VARCHAR ...	0 row(s) affected

2) Creamos las tablas credit_Cards, users_all y transactions

The screenshot shows a SQL script in a text editor with the following content:

```

15
16 • CREATE TABLE credit_cards (
17     id VARCHAR (10) PRIMARY KEY,
18     user_id VARCHAR (10),
19     iban VARCHAR (50),
20     pan VARCHAR (50),
21     pin VARCHAR (4),
22     cvv VARCHAR (3),
23     track1 VARCHAR (50),
24     track2 VARCHAR (50),
25     expiring_date VARCHAR (10)
26 );
27

```

Below the script, the 'Output' window shows the execution results:

#	Time	Action	Message
1	10:11:25	CREATE DATABASE transactionsT4	1 row(s) affected
2	10:12:26	USE transactionsT4	0 row(s) affected
3	10:21:35	CREATE TABLE companies (company_id VARCHAR(8) PRIMARY KEY, company_na...	0 row(s) affected
4	10:31:05	CREATE TABLE credit_cards (id VARCHAR (10) PRIMARY KEY, user_id VARCHAR ...	0 row(s) affected
5	10:45:30	CREATE TABLE users_usa (id VARCHAR (10) PRIMARY KEY, name VARCHAR (50)...	0 row(s) affected

Giarioli - S03* Giarioli - S04

```

29 CREATE TABLE users_all (
30     id INTEGER PRIMARY KEY,
31     name VARCHAR (50),
32     surname VARCHAR (50),
33     phone VARCHAR (50),
34     email VARCHAR (50),
35     birth_date VARCHAR (50),
36     country VARCHAR (50),
37     city VARCHAR (50),
38     postal_code VARCHAR (10),
39     address VARCHAR (50)
40 );

```

Result Grid

id	card_id	business_id	timestamp	amount	declined	products_ids	user_id	lat
02C6201E-D90A-1859-B4EE-88D2986D3B02	CcU-2938	b-2362	28/08/2021 23:42	466.92	0	71, 1, 19	92	819.18
0466A42E-47CF-8D24-FD01-C0B689713128	CcU-4219	b-2302	26/07/2021 7:29	49.53	0	47, 97, 43	170	-439.6
063FBA79-99EC-66FB-29F7-25726D1764A5	CcU-2987	b-2250	06/01/2022 21:25	92.61	0	47, 67, 31, 5	275	-81.22
0668296C-CDB9-A883-76BC-2E4C4F8C8AE	CcU-3743	b-2618	26/01/2022 2:07	394.18	0	89, 83, 79	265	-343.5

transactions 5

Output

Action Output

#	Time	Action	Message
49	11:28:24	CREATE TABLE users_all (id INTEGER PRIMARY KEY, name VARCHAR (50), ...	0 row(s) affected
50	11:28:49	CREATE TABLE transactions (id VARCHAR (50) PRIMARY KEY, card_id VARCHAR...	0 row(s) affected

```

42 CREATE TABLE transactions (
43     id VARCHAR (50) PRIMARY KEY,
44     card_id VARCHAR (10),
45     business_id VARCHAR(8),
46     timestamp VARCHAR (50),
47     amount DECIMAL (10,2),
48     declined BOOLEAN,
49     products_ids VARCHAR (50),
50     user_id INT,
51     lat VARCHAR (50),
52     longitude VARCHAR (50),
53     FOREIGN KEY (card_id) REFERENCES credit_cards(id),
54     FOREIGN KEY (business_id) REFERENCES companies(company_id),

```

Result Grid

id	card_id	business_id	timestamp	amount	declined	products_ids	user_id	lat
02C6201E-D90A-1859-B4EE-88D2986D3B02	CcU-2938	b-2362	28/08/2021 23:42	466.92	0	71, 1, 19	92	819.18
0466A42E-47CF-8D24-FD01-C0B689713128	CcU-4219	b-2302	26/07/2021 7:29	49.53	0	47, 97, 43	170	-439.6
063FBA79-99EC-66FB-29F7-25726D1764A5	CcU-2987	b-2250	06/01/2022 21:25	92.61	0	47, 67, 31, 5	275	-81.22
0668296C-CDB9-A883-76BC-2E4C4F8C8AE	CcU-3743	b-2618	26/01/2022 2:07	394.18	0	89, 83, 79	265	-343.5

transactions 5

Output

Action Output

#	Time	Action	Message
50	11:28:49	CREATE TABLE transactions (id VARCHAR (50) PRIMARY KEY, card_id VARCHAR...	0 row(s) affected

3) **Insertamos los datos en cada tabla** usando la función "Table Data Import Wizard".

4) Y **verificamos la inserción** de los datos con el comando SELECT *