

SPRINT 4

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:

Ejercicio 1

Realiza una subconsulta que muestre a todos los usuarios con más de 80 transacciones utilizando al menos 2 tablas.

Respuesta:

Paso 1: Se crea la Base de datos 'comercial'

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'SCHEMAS' tree is expanded, showing the 'comercial' database. The main pane displays the following SQL script:

```
1 --- Se crea la base de datos
2 CREATE DATABASE IF NOT EXISTS comercial;
3 USE comercial;
4
```

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

#	Time	Action	Message	Duration / Fetch
1	09:45:56	CREATE DATABASE IF NOT EXISTS comercial	1 row(s) affected	0.016 sec
2	09:46:00	USE comercial	0 row(s) affected	0.015 sec

Paso 2: Se crean las tablas 'american_users', 'european_users', 'credit_cards', 'companies' y 'transactions'.

The screenshot shows the SQL Server Enterprise Manager interface. The main pane displays the following SQL script for creating the 'american_users' table:

```
5 --- Se crea la tabla 'american_users'
6 CREATE TABLE IF NOT EXISTS american_users (
7     id INT PRIMARY KEY,
8     name VARCHAR(100),
9     surname VARCHAR(100),
10    phone VARCHAR(150),
11    email VARCHAR(150),
12    birth_date VARCHAR(100),
13    country VARCHAR(150),
14    city VARCHAR(150),
15    postal_code VARCHAR(100),
16    address VARCHAR(255)
17 );
18
```

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

#	Time	Action	Message	Duration / Fetch
1	09:48:00	CREATE TABLE IF NOT EXISTS american_users (id INT P...	0 row(s) affected	0.047 sec

```

18  --- Se crea la tabla 'european_users'
19  ● CREATE TABLE IF NOT EXISTS european_users (
20      id INT PRIMARY KEY,
21      name VARCHAR(100),
22      surname VARCHAR(100),
23      phone VARCHAR(150),
24      email VARCHAR(150),
25      birth_date VARCHAR(100),
26      country VARCHAR(150),
27      city VARCHAR(150),
28      postal_code VARCHAR(100),
29      address VARCHAR(255)
30  );

```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 1	09:48:00	CREATE TABLE IF NOT EXISTS american_users (id INT P...	0 row(s) affected	0.047 sec
✓ 2	09:48:57	CREATE TABLE IF NOT EXISTS european_users (id INT ...	0 row(s) affected	0.047 sec

```

31  --- Se crea la tabla 'credit_cards'
32  ● CREATE TABLE IF NOT EXISTS credit_cards (
33      id VARCHAR(15) PRIMARY KEY,
34      user_id VARCHAR(15),
35      iban VARCHAR(50),
36      pan VARCHAR(50),
37      pin VARCHAR(4),
38      cvv VARCHAR(3),
39      track1 VARCHAR(150),
40      track2 VARCHAR(150),
41      expiring_date VARCHAR(15)
42  );

```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 1	09:50:26	CREATE TABLE IF NOT EXISTS credit_cards (id VARCHA...	0 row(s) affected	0.047 sec

```

43  --- Se crea la tabla 'companies'
44  ● CREATE TABLE IF NOT EXISTS companies (
45      company_id VARCHAR(20) PRIMARY KEY,
46      company_name VARCHAR(255),
47      phone VARCHAR(15),
48      email VARCHAR(100),
49      country VARCHAR(100),
50      website VARCHAR(255)
51  );

```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 1	09:50:26	CREATE TABLE IF NOT EXISTS credit_cards (id VARCHA...	0 row(s) affected	0.047 sec
✓ 2	09:51:22	CREATE TABLE IF NOT EXISTS companies (company_id ...	0 row(s) affected	0.062 sec

```

52  --- Se crea la tabla 'transactions'
53  CREATE TABLE IF NOT EXISTS transactions (
54      id VARCHAR(255) PRIMARY KEY,
55      card_id VARCHAR(15),
56      business_id VARCHAR(20),
57      timestamp TIMESTAMP,
58      amount DECIMAL(10,2),
59      declined TINYINT(1),
60      product_ids VARCHAR(255),
61      user_id INT,
62      lat FLOAT,
63      longitude FLOAT
64  );

```

Output

#	Time	Action	Message	Duration / Fetch
1	13:01:38	CREATE TABLE IF NOT EXISTS transactions (id VARC...	0 row(s) affected, 1 warning(s): 1681 Integer display widt...	0.047 sec

Paso 3: Se cargan los datos de las tablas 'american_users', 'european_users', 'credit_cards', 'companies' y 'transactions'

```

65  --- Se carga los datos de la tabla 'american_users'
66  LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\american_users.csv'
67  INTO TABLE american_users
68  FIELDS TERMINATED BY ','
69  ENCLOSED BY '"'
70  LINES TERMINATED BY '\n'
71  IGNORE 1 ROWS;
72

```

Output

#	Time	Action	Message	Duration / Fetch
1	12:50:16	LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL...	1010 row(s) affected Records: 1010 Deleted: 0 Skippe...	0.094 sec

```

79  --- Se carga los datos de la tabla 'european_users'
80  LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\european_users.csv'
81  INTO TABLE european_users
82  FIELDS TERMINATED BY ','
83  ENCLOSED BY '"'
84  LINES TERMINATED BY '\n'
85  IGNORE 1 ROWS;
86

```

Output

#	Time	Action	Message	Duration / Fetch
1	12:51:43	LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL...	3990 row(s) affected Records: 3990 Deleted: 0 Skippe...	0.266 sec

```

87  --- Se carga los datos de la tabla 'credit_cards'
88  LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\credit_cards.csv'
89  INTO TABLE credit_cards
90  FIELDS TERMINATED BY ','
91  ENCLOSED BY '"'
92  LINES TERMINATED BY '\n'
93  IGNORE 1 ROWS;

```

Output

#	Time	Action	Message	Duration / Fetch
1	12:53:21	LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL...	5000 row(s) affected Records: 5000 Deleted: 0 Skippe...	0.359 sec

```

95 --- Se carga los datos de la tabla 'companies'
96 • LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\companies.csv'
97 INTO TABLE companies
98 FIELDS TERMINATED BY ','
99 ENCLOSED BY '"'
100 LINES TERMINATED BY '\\n'
101 IGNORE 1 ROWS;

```

Output				
Action Output				
#	Time	Action	Message	Duration / Fetch
1	12:55:57	LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL...	100 row(s) affected Records: 100 Deleted: 0 Skipped: ...	0.047 sec

```

103 --- Se carga los datos de la tabla 'transactions'
104 • LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\transactions.csv'
105 INTO TABLE transactions
106 FIELDS TERMINATED BY ';'
107 ENCLOSED BY '"'
108 LINES TERMINATED BY '\\n'
109 IGNORE 1 ROWS;

```

Output				
Action Output				
#	Time	Action	Message	Duration / Fetch
1	13:04:50	LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL...	100000 row(s) affected Records: 100000 Deleted: 0 Sk...	2.641 sec

Paso 4: Se comparan las tablas 'american_users' y 'european_users', con el objetivo de juntarlas luego en una sola tabla.

```

111 --- Se comparan los 'id' de las tablas 'american_users' y 'european_users'
112 --- Encontrar IDs que existen en ambas tablas
113 • SELECT au.id
114 FROM american_users au
115 INNER JOIN european_users eu ON au.id = eu.id;
116

```

Result Grid				
id				
Result 5 x				
Read Only				
Output				
Action Output				
#	Time	Action	Message	Duration / Fetch
1	13:51:10	SELECT au.id FROM american_users au INNER JOIN ...	0 row(s) returned	0.016 sec / 0.000 sec

Al consultar la existencia de id's duplicados entre ambas tablas y no tener resultados, se comprueba que no hay conflictos (id duplicados) entre ambas tablas de usuarios (american y european). Por lo que se pueden juntar ambas tablas en una sola.

Paso 5: Se crea una tabla única de 'users', con la misma estructura que tienen las tablas 'american_users' y 'european_users'.

```

117 --- Se crea tabla unica 'users' que combina las tablas 'american_users' y 'european_users'
118 --- Se agrega una columna 'region' para identificar los registros de cada tabla.
119 CREATE TABLE IF NOT EXISTS users (
120     id INT PRIMARY KEY,
121     name VARCHAR(100),
122     surname VARCHAR(100),
123     phone VARCHAR(150),
124     email VARCHAR(150),
125     birth_date VARCHAR(100),
126     country VARCHAR(150),
127     city VARCHAR(150),
128     postal_code VARCHAR(100),
129     address VARCHAR(255),
130     region VARCHAR(150)
131 );

```

#	Time	Action	Message	Duration / Fetch
1	14:04:06	CREATE TABLE IF NOT EXISTS users (id INT PRIMAR...	0 row(s) affected	0.032 sec

Paso 6: Se fusionan los datos de las tablas 'american_users' y 'european_users' en la nueva tabla de 'users'.

```

132 --- Se fusionan los datos de las tablas 'american_users' y 'european_users' en la nueva tabla 'users'
133 INSERT INTO users (id, name, surname, phone, email, birth_date, country, city, postal_code, address, region)
134 SELECT id, name, surname, phone, email, birth_date, country, city, postal_code, address, 'AMERICAN' as region
135 FROM american_users
136 UNION ALL
137 SELECT id, name, surname, phone, email, birth_date, country, city, postal_code, address, 'EUROPEAN' as region
138 FROM european_users;
139
140 SELECT *
141 FROM users;

```

#	Time	Action	Message	Duration / Fetch
1	14:07:41	INSERT INTO users (id, name, surname, phone, email, birth_date, co...	5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0	0.360 sec
2	14:08:26	SELECT * FROM users	5000 row(s) returned	0.000 sec / 0.031 sec

Se utiliza UNION ALL, con el cual se juntan todos los datos de ambas tablas incluidos duplicados, pero en este caso se ha comprobado en el paso anterior que No existen duplicados. Por lo que utilizar UNION ALL no presentaría problema alguno con los datos en la nueva tabla 'users'.

Paso 7: Se eliminan las tablas 'american_users' y 'european_users', porque ya no las necesitamos.

```
144 • DROP TABLE IF EXISTS american_users;
145 • DROP TABLE IF EXISTS european_users;
146
147 ---
```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 1	20:28:23	DROP TABLE IF EXISTS american_users	0 row(s) affected	0.047 sec
✓ 2	20:28:27	DROP TABLE IF EXISTS european_users	0 row(s) affected	0.031 sec

Paso 8: Se crea las FK en la tabla de 'transactions', para vincular las tablas en la base de datos.

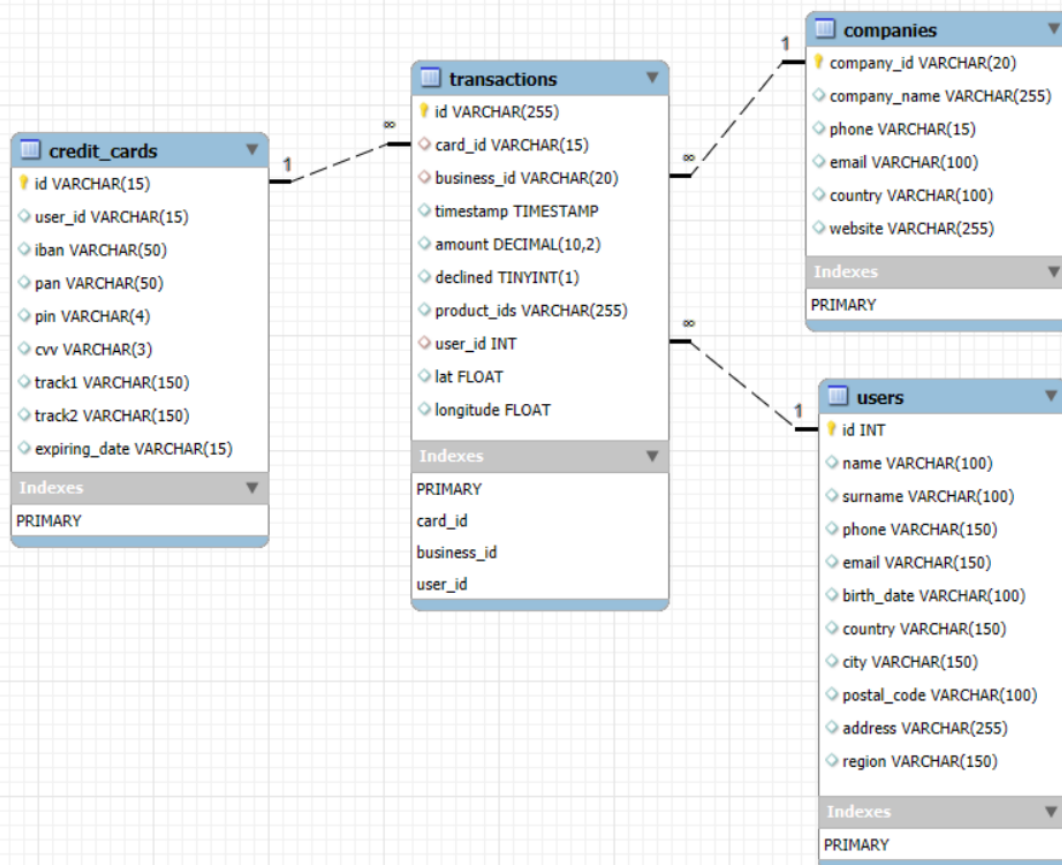
```
148 • ALTER TABLE transactions
149   ADD FOREIGN KEY (card_id) REFERENCES credit_cards (id),
150   ADD FOREIGN KEY (business_id) REFERENCES companies (company_id),
151   ADD FOREIGN KEY (user_id) REFERENCES users (id);
152
```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 1	20:53:16	ALTER TABLE transactions ADD FOREIGN KEY (card_id)...	100000 row(s) affected Records: 100000 Duplicates: 0 ...	3.875 sec

Paso 9: Se ha diseñado el siguiente esquema de estrella, donde la tabla de hechos es la tabla de 'transacciones' y las demás tablas son de 'dimensiones', tal como se ve en la siguiente imagen:



La relación de las tablas 'credit_card', 'companies' y 'users' es de 1 a muchos con respecto a la tabla 'transacciones'.

Se relacionan mediante lo siguiente:

- 'credit_cards' (id) = 'transactions' (card_id)
- 'companies' (company_id) = 'transactions' (business_id)
- 'users' (id) = 'transactions' (user_id)

Respuesta Ejercicio1: Se realiza una subconsulta que muestre a todos los usuarios con más de 80 transacciones utilizando al menos 2 tablas.

```
159 • SELECT u.id, u.name, u.surname, total_transactions
160 FROM users u, (SELECT DISTINCT t.user_id, COUNT(t.id) AS total_transactions
161 FROM transactions t
162 GROUP BY t.user_id
163 HAVING total_transactions > 80) ut
164 WHERE ut.user_id = u.id;
```

Result Grid

id	name	surname	total_transaction
185	Molly	Gilliam	110
289	Dxwgi	Hwcru	94
318	Bnyr	Astuw	91
454	Sfzzoh	Xgvfridxs	81

Result 22 x Read Only

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	23:07:39	SELECT u.id, u.name, u.surname, total_transactions FR...	4 row(s) returned	0.062 sec / 0.000 sec

Para este caso, se utilizan las tablas 'transactions' y 'users'. Y se ha identificado todos los usuarios (id) que tienen un total de transacciones por encima de 80.

Se ha considerado todas las transacciones registradas, que incluyen declinadas y no declinadas, ya que luego se puede profundizar en los usuarios con transacciones declinadas y usuarios con transacciones no declinadas, según el objetivo de análisis del departamento que solicita la información.

Por ejemplo: podría determinarse por un lado a los usuarios 'importantes' (alto volumen y amount de transacciones) para una campaña de fidelización de clientes, etc. Y por otro lado, se puede determinar los usuarios con comportamiento sospechoso de transacciones (alto volumen de transacciones declinadas, alto volumen y/o amount en diferentes ciudades en un tiempo muy corto, etc.) para identificar posibles peligros de fraude, fallas de tarjetas que solucionar de un lote determinado, etc.

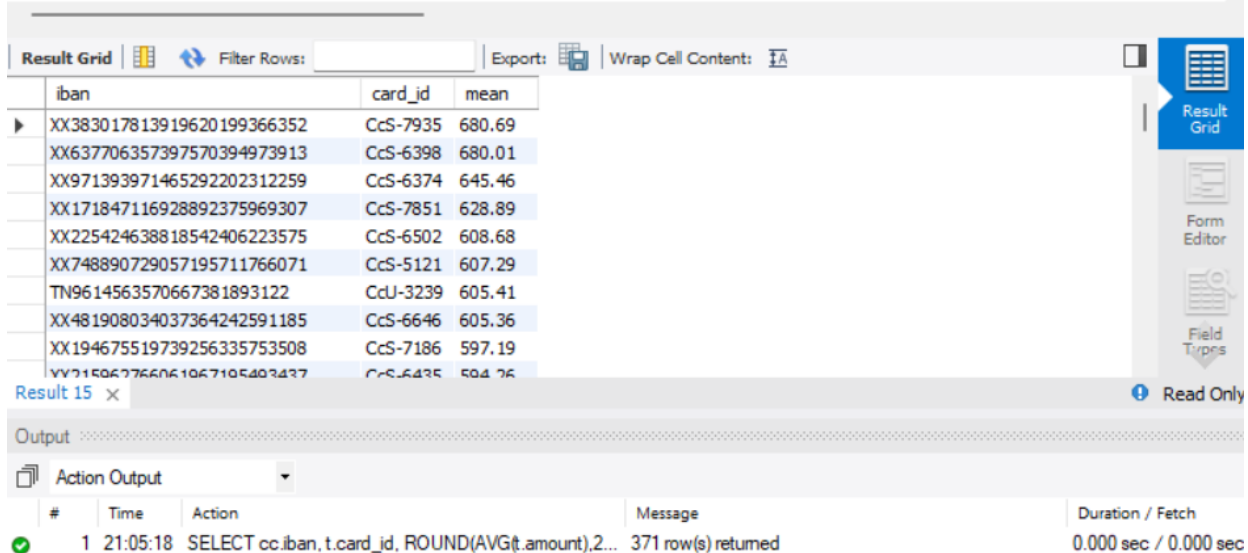
Se precisa, que se ha realizado una 'subconsulta' porque así lo pide el ejercicio.

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.

Respuesta:

```
167 • SELECT cc.iban, t.card_id, ROUND(AVG(t.amount),2) AS mean
168 FROM transactions t
169 JOIN credit_cards cc ON cc.id = t.card_id
170 JOIN companies c ON c.company_id = t.business_id AND c.company_name = 'Donec Ltd'
171 GROUP BY t.card_id, t.business_id
172 ORDER BY mean DESC;
173
```



The screenshot shows a database query tool interface. At the top, the SQL query is displayed. Below it, the 'Result Grid' shows the results of the query. The grid has three columns: 'iban', 'card_id', and 'mean'. The results are ordered by 'mean' in descending order. The first row shows an IBAN of XX383017813919620199366352, card_id CcS-7935, and a mean of 680.69. The last row shows an IBAN of YY715067766061067105403437, card_id CcS-6435, and a mean of 504.76. The interface also includes a 'Filter Rows' section, an 'Export' button, and a 'Wrap Cell Content' option. On the right side, there are buttons for 'Result Grid', 'Form Editor', and 'Field Types'. At the bottom, the 'Output' section shows the execution details: 'Action Output', '1 21:05:18 SELECT cc.iban, t.card_id, ROUND(AVG(t.amount),2)... 371 row(s) returned', and 'Duration / Fetch 0.000 sec / 0.000 sec'.

iban	card_id	mean
XX383017813919620199366352	CcS-7935	680.69
XX637706357397570394973913	CcS-6398	680.01
XX971393971465292202312259	CcS-6374	645.46
XX171847116928892375969307	CcS-7851	628.89
XX225424638818542406223575	CcS-6502	608.68
XX748890729057195711766071	CcS-5121	607.29
TN9614563570667381893122	CcU-3239	605.41
XX481908034037364242591185	CcS-6646	605.36
XX194675519739256335753508	CcS-7186	597.19
YY715067766061067105403437	CcS-6435	504.76

En este caso se ha utilizado las tablas 'credit_cards', 'transactions' y 'companies'.

Se ha palicado 2 JOIN de 'transactions' con 'credit_cards' y con companies'.

De la tabla 'transactions' y 'credit_card' se ha obtenido las columnas iban, card_id y mean (la media de amount).

Para la media de 'amount', se ha utilizado la agrupación por 'card_id' y 'business_id', debido a que así se podía obtener la media por tarjeta y por empresa (esto ya que los id de tarjeta de crédito se repetían en diferentes empresas con diferentes amount).

Finalmente, con el JOIN con 'companies' se filtró por el id de la empresa DONEC Ltd. Y se ordenó la información final por 'mean' (media) de mayor a menor.

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:

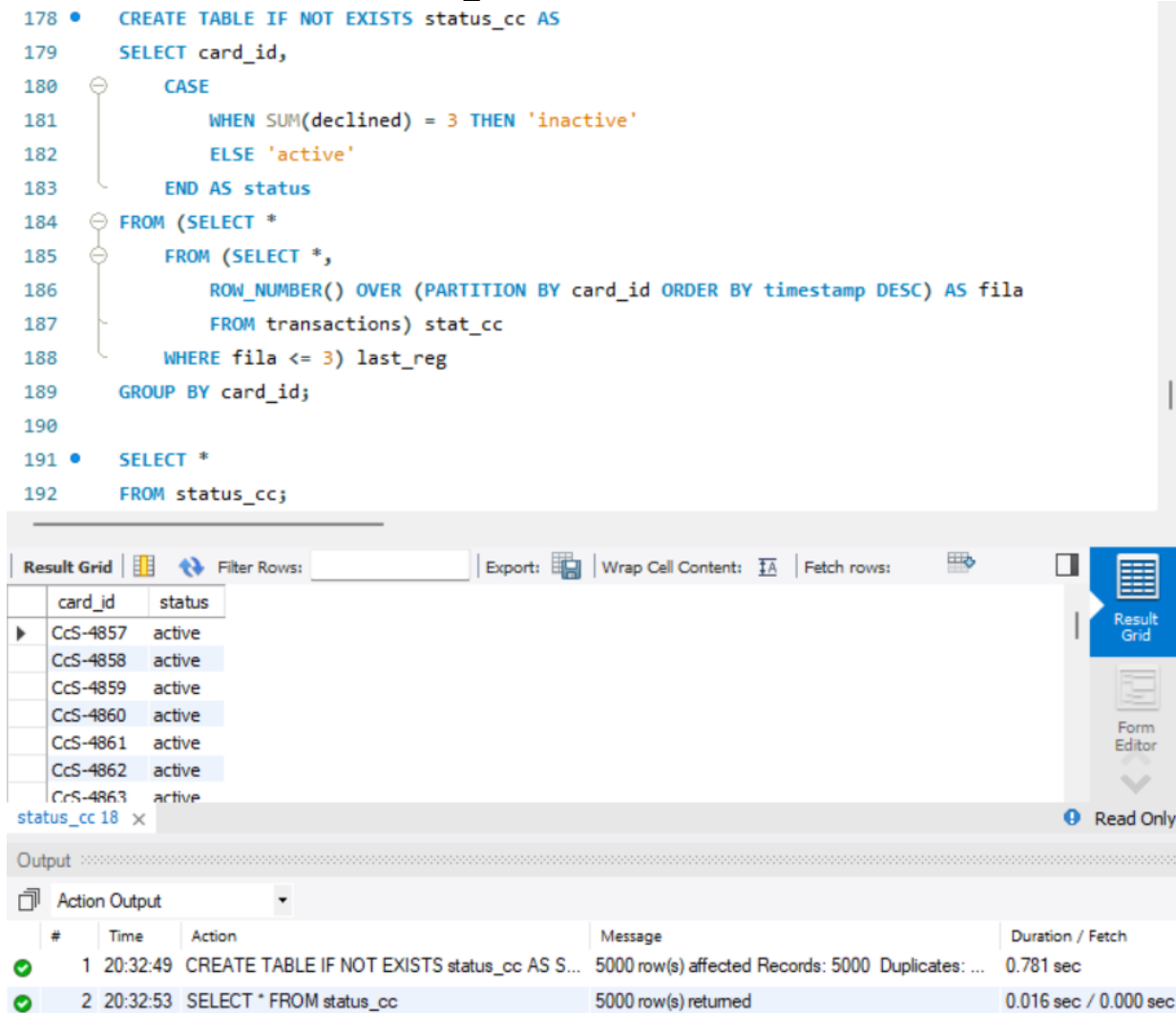
Ejercicio 1

¿Cuántas tarjetas están activas?

Respuesta:

Paso 1: Se ha creado la tabla 'status_cc'

```
178 • CREATE TABLE IF NOT EXISTS status_cc AS
179     SELECT card_id,
180            CASE
181                WHEN SUM(declined) = 3 THEN 'inactive'
182                ELSE 'active'
183            END AS status
184     FROM (SELECT *
185           FROM (SELECT *,
186                 ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS fila
187                FROM transactions) stat_cc
188           WHERE fila <= 3) last_reg
189     GROUP BY card_id;
190
191 • SELECT *
192     FROM status_cc;
```



card_id	status
CcS-4857	active
CcS-4858	active
CcS-4859	active
CcS-4860	active
CcS-4861	active
CcS-4862	active
CcS-4863	active

#	Time	Action	Message	Duration / Fetch
1	20:32:49	CREATE TABLE IF NOT EXISTS status_cc AS S...	5000 row(s) affected Records: 5000 Duplicates: ...	0.781 sec
2	20:32:53	SELECT * FROM status_cc	5000 row(s) returned	0.016 sec / 0.000 sec

Se ha creado una tabla 'status_cc' con 2 columnas: card_id VARCHAR(15) y status VARCHAR(8).

En la columna 'status' aparece el mensaje 'active' o 'inactive', según la condición de tener las ultimas 3 transacciones declinadas o no, de cada tarjeta.

Se aplica 'ROW_NUMBER()' Para obtener todas las transacciones ordenadas de más reciente a más antigua y por tarjeta.

Luego se limita a obtener las 3 filas más recientes de cada tarjeta y finalmente se realiza la consulta 'CASE...' con la condición a cumplir para aparecer como 'active' o 'inactive'.

Paso 2: Se crea la PK y la FK para vincular la nueva tabla en la base de datos.

```
218 --- Se crea la PK de la tabla 'status_cc'.
219 • ALTER TABLE status_cc
220 ADD PRIMARY KEY (card_id);
221
222 --- Se crea la FK de la tabla 'status_cc'.
223 • ALTER TABLE status_cc
224 ADD FOREIGN KEY (card_id) REFERENCES credit_cards (id),
225 ADD UNIQUE (card_id);
```

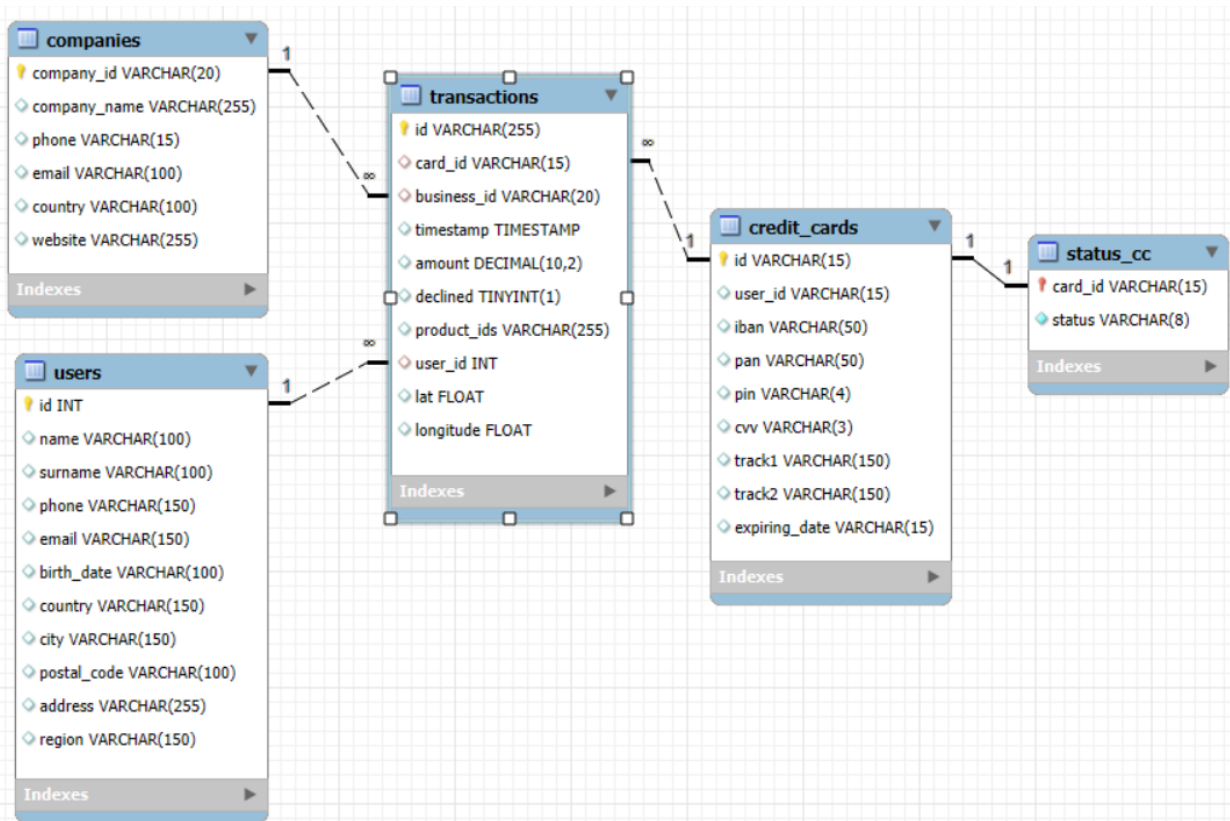
Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 1	22:30:45	ALTER TABLE status_cc ADD PRIMARY KEY (ca...	0 row(s) affected Records: 0 Duplicates: 0 Wamin...	0.234 sec
✓ 2	22:31:50	ALTER TABLE status_cc ADD FOREIGN KEY (ca...	5000 row(s) affected Records: 5000 Duplicates: 0 ...	0.266 sec

Como resultado, se vincula la tabla 'status_cc' con la tabla 'credit_cards', en una relación de 1 a 1, siendo: 'credit_cards'(id) = 'status_cc' (card_id).

--- Paso 3: se genera diagrama, para verificar las tablas relacionadas.



Se visualiza que se ha añadido la tabla 'status_cc', vinculada a la tabla 'credit_cards'. Con lo que, el modelo deja de ser un modelo de estrella.

Respuesta Ejercicio1: ¿Cuántas tarjetas están activas?.

```
230 • SELECT COUNT(card_id) AS num_creditcards, status
231 FROM status_cc
232 WHERE status = 'active';
233
```

Result Grid

num_creditcards	status
4995	active

Result 10 x Read Only

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	22:43:41	SELECT COUNT(card_id) AS num_creditcards, st...	1 row(s) returned	0.000 sec / 0.000 sec

Están activas 4995 tarjetas.

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:

Ejercicio 1

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

Respuesta:

Paso 1: Se ha creado la tabla 'products'

```
243 --- Se crea la tabla 'products'
244 • CREATE TABLE IF NOT EXISTS products (
245     id VARCHAR(250),
246     product_name VARCHAR(250),
247     price VARCHAR(150),
248     colour VARCHAR(150),
249     weight VARCHAR(150),
250     warehouse_id VARCHAR(150)
251 );
252
```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	10:22:57	CREATE TABLE IF NOT EXISTS products (id VA...	0 row(s) affected	0.031 sec

Paso 2: Se cargan los datos de la tabla 'products'

```

253 --- Se carga los datos de la tabla 'products'
254 • LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\products.csv'
255 INTO TABLE products
256 FIELDS TERMINATED BY ','
257 ENCLOSED BY '"'
258 LINES TERMINATED BY '\\n'
259 IGNORE 1 ROWS;
260

```

Output

#	Time	Action	Message	Duration / Fetch
1	10:22:57	CREATE TABLE IF NOT EXISTS products (id VA...	0 row(s) affected	0.031 sec
2	10:23:42	LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\...	100 row(s) affected Records: 100 Deleted: 0 Skip...	0.063 sec

Paso 3: Teniendo la tabla 'products' creada y con datos, se procede a verificar la columna 'product_ids' de la tabla 'transactions', con el fin de ver la mejor forma de crear una tabla intermedia (bridge) que conecte a ambas tablas.

Para ello, se decide 'normalizar' la tabla 'transactions', en específico se busca que los productos ingresados con comas en la columna 'product_ids' por cada transacción, se separen en un producto por cada fila, sin dejar de ser identificados con su id de transacción respectivo.

Por esto, se realiza una consulta en la que se busca conocer el largo de cadena en la columna 'product_ids' y el número de comas máximo (n).

```

265 • SELECT id, product_ids, length(product_ids) AS l1, length(REPLACE(product_ids,',')) AS l2, length(product_ids) - length(REPLACE(product_ids,',')) AS n
266 FROM transactions
267 ORDER BY length(product_ids) - length(REPLACE(product_ids,',')) DESC;
268

```

Result Grid

id	product_ids	l1	l2	n
F72AECAD-492C-4747-B464-7D02D747221D	94, 84, 87, 13, 52	18	14	4
F76A7150-F024-44DB-88FB-9D2ABF2AA8A9	52, 35, 46, 18, 14	18	14	4
F85C2DEB-4610-4511-8082-4FBF889158FD	84, 74, 34, 24, 98	18	14	4
F897778C-1449-4F6A-A976-84C45E15D171	73, 84, 6, 83, 60	17	13	4
F8F42F41-A9D4-4B0F-94C2-907B6D032B87	77, 70, 21, 28, 39	18	14	4
F8FB55BA-581B-4677-B87C-E04558BC1E6D	16, 42, 43, 46, 94	18	14	4
F9E80CC8-8E85-40B1-8AF0-C0EC60DBF940	51, 34, 2, 61, 59	17	13	4
FA5ED844-86DD-414A-AD09-9D929AA6FF44	33, 96, 10, 9, 16	17	13	4

Output

#	Time	Action	Message	Duration / Fetch
1	20:25:58	SELECT id, product_ids, length(product_ids) AS l1, length(REPLACE(product_ids,',')) AS l2, length...	100000 row(s) returned	0.109 sec / 0.047 sec

En esta consulta, se mide el largo original de la cadena (l1), el largo de la cadena quitando comas y espacios (l2) y luego se realiza una resta para saber cuántas comas existen (columna 'n'). Se ordena por la columna 'n', para saber el 'n' mayor.

Una vez identificado cuantas comas hay como máximo, en este caso 4; se puede determinar que el número máximo de productos en una transacción es de 5.

Paso 4: Se crea una nueva tabla 'products_related', para relacionar la tabla 'products' y la tabla 'transactions', mediante una consulta que tiene en cuenta que en la tabla 'transactions' la columna de id de productos tiene más de un valor.

```

271 • CREATE TABLE IF NOT EXISTS products_related AS
272 WITH prod_trans AS (
273     SELECT *
274     FROM (
275         SELECT id, product_ids, trim(SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', 1), ',', -1)) AS valor FROM transactions
276         UNION
277         SELECT id, product_ids, trim(SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', 2), ',', -1)) AS valor FROM transactions
278         UNION
279         SELECT id, product_ids, trim(SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', 3), ',', -1)) AS valor FROM transactions
280         UNION
281         SELECT id, product_ids, trim(SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', 4), ',', -1)) AS valor FROM transactions
282         UNION
283         SELECT id, product_ids, trim(SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', 5), ',', -1)) AS valor FROM transactions
284     ) a
285     ORDER BY length(product_ids) - length(REPLACE(product_ids, ',', '')) DESC, id
286 )
287 SELECT pt.id AS id_transaction, p.id AS id_product
288 FROM prod_trans pt
289 JOIN products p ON FIND_IN_SET(p.id, pt.valor);

```

Output

#	Time	Action	Message	Duration / Fetch
1	13:58:19	CREATE TABLE IF NOT EXISTS products_related AS WITH prod_trans AS (...)	253391 row(s) affected Records: 253391 Duplicates: 0 Warnings: 0	5.906 sec

Para esta tabla, se ha tomado el resultado de la consulta anterior, referida al largo de cadena, número de comas máximo y número de productos máximo, registrados en una fila de la columna 'product_id'. Siendo que se hallaron como máximo 4 comas, quiere decir que hay como máximo 5 productos en una fila.

Con esta información, se realizó una consulta que separa los productos de cada transacción, para que cada producto tenga una fila (se utiliza 5 instrucciones de 'substring_index') y 'union' para obtener todos los productos de cada transacción.

Luego se utilizó 'order by' para ordenar la aparición de productos, según transacción con más productos.

Esta consulta se guardó como un CTE y se hace un JOIN entre este CTE 'prod_trans' y la tabla 'products' que teníamos en .csv. Aquí aplicamos un FIND_IN_SET con el cual se comprueba que los id de la tabla productos y los id de la tabla transacciones coinciden.

Finalmente, se crea la tabla 'products related' con la consulta realizada (Se visualiza parte de la tabla). Se precisa, que esta consulta logra obtener que todos los productos de cada transacción tengan una fila, aunque en alguna fila de origen haya algún producto repetido.

```

291 • SELECT *
292 FROM products_related
293 ORDER BY id_transaction;

```

Result Grid

id_transaction	id_product
00043A49-2949-494B-A5DD-A5BAE38B19DD	87
00043A49-2949-494B-A5DD-A5BAE38B19DD	16
00043A49-2949-494B-A5DD-A5BAE38B19DD	97
00043A49-2949-494B-A5DD-A5BAE38B19DD	26
000447FE-B650-4DCF-85DE-C7ED0EE1CAAD	69
000447FE-B650-4DCF-85DE-C7ED0EE1CAAD	87
000447FE-B650-4DCF-85DE-C7ED0EE1CAAD	66
00045D68-ED2E-4F2F-8186-CEE074D875D0	30
00045D68-ED2E-4F2F-8186-CEE074D875D0	81
00045D68-ED2E-4F2F-8186-CEE074D875D0	16
00045D68-ED2E-4F2F-8186-CEE074D875D0	11
000481C3-1C26-4FEF-83A0-4CD0EB0048BD	72

products_related 22 x

Output

#	Time	Action	Message	Duration / Fetch
1	14:17:46	SELECT * FROM products_related ORDER BY id_tr...	253391 row(s) returned	0.437 sec / 0.079 sec

Paso 4: Se crea la PK para vincular la tabla 'products'.

```
349 • ALTER TABLE products
350   ADD PRIMARY KEY (id);
---
```

Output

#	Time	Action	Message	Duration / Fetch
1	18:01:36	ALTER TABLE products ADD PRIMARY KEY (id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.141 sec

Cuando se creó la tabla 'products' no se había asignado primary key, lo realizamos en este momento.

Paso 5: Se crea la PK y FK para vincular la nueva tabla 'products_related' en la base de datos.

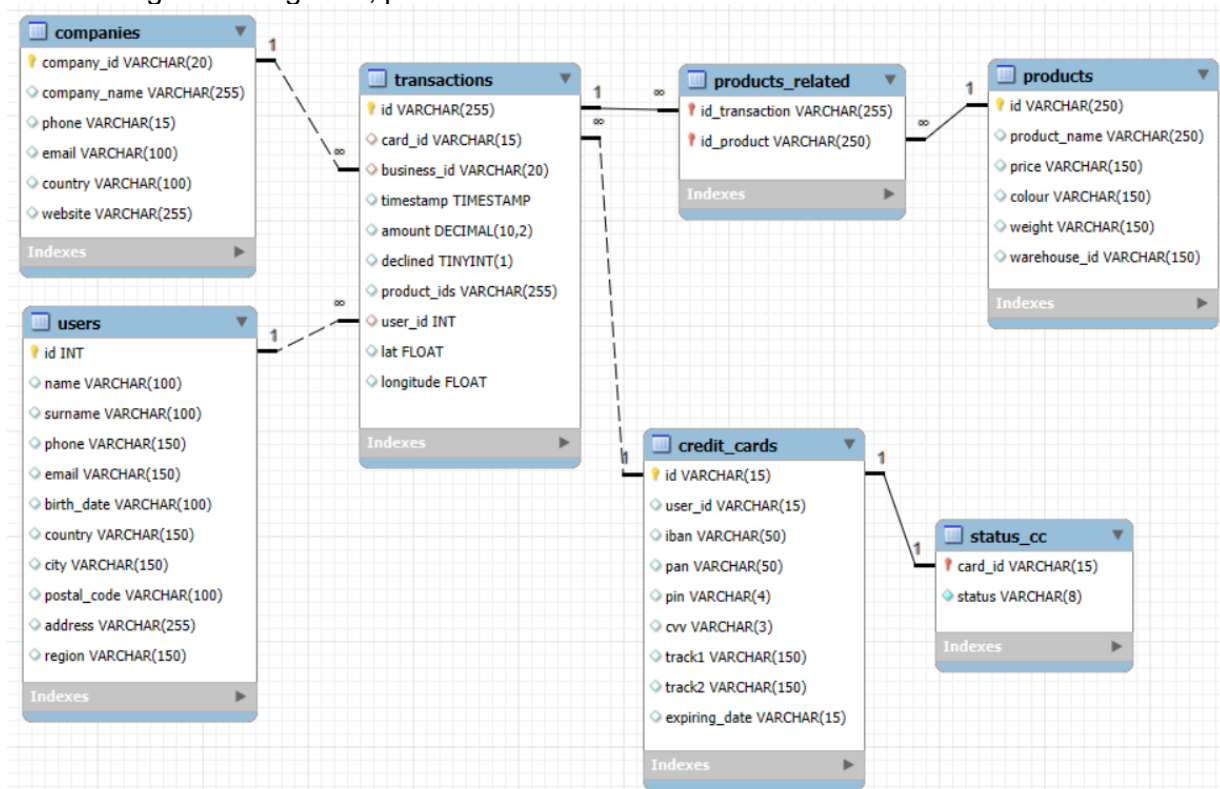
```
352 --- Se crea la PK y la FK para vincular la nueva tabla en la base de datos.
353 • ALTER TABLE products_related
354   ADD PRIMARY KEY (id_transaction, id_product);
355
356 • ALTER TABLE products_related
357   ADD FOREIGN KEY (id_transaction) REFERENCES transactions (id);
358
359 • ALTER TABLE products_related
360   ADD FOREIGN KEY (id_product) REFERENCES products (id);
---
```

Output

#	Time	Action	Message	Duration / Fetch
1	18:03:18	ALTER TABLE products_related ADD PRIMARY KEY (id...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	3.672 sec
2	18:03:38	ALTER TABLE products_related ADD FOREIGN KEY (id...	253391 row(s) affected Records: 253391 Duplicates: 0 ...	3.719 sec
3	18:03:51	ALTER TABLE products_related ADD FOREIGN KEY (id...	253391 row(s) affected Records: 253391 Duplicates: 0 ...	3.578 sec

Se crea las PK de la tabla 'products_related', al ser una tabla intermedia, ambas columnas corresponden con PK de otras tablas, por lo que deben mantener dicha característica. Asimismo, cada columna de la tabla 'products_related' debe asignarse también como FK para vincularla con su tabla correspondiente.

Paso 6: se genera diagrama, para verificar las tablas relacionadas.



Se visualiza diagrama modificado, con el añadido de las tablas 'products' y 'products_related'.

La tabla 'products_related' tiene dos columnas: id_transaction VARCHAR(255) y id_product VARCHAR(250).

La tabla 'products_related' es una tabla intermedia entre 'transactions' y 'products'.

Asimismo, la relación entre las tablas es de '1 a muchos', siendo muchos para 'products_related' y 1 para 'transactions' y 'products'.

Respuesta Ejercicio1: Necesitamos conocer el número de veces que se ha vendido cada producto.

The screenshot shows a database query interface. At the top, a SQL query is displayed in a text area:

```
364 • SELECT pr.id_product, COUNT(pr.id_transaction) AS quantity_products
365 FROM products_related pr
366 JOIN transactions t ON pr.id_transaction = t.id AND t.declined = 0
367 GROUP BY id_product;
368
```

Below the query, there is a toolbar with options like 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. The main area displays a table with the following data:

	id_product	quantity_products
▶	16	2602
	26	2532
	87	2591
	97	2520
	66	2590
	69	2453
	11	2559
	30	2463

On the right side, there are buttons for 'Result Grid' and 'Form Editor'. Below the table, there is a 'Result 2' tab and a 'Read Only' status. At the bottom, there is an 'Output' section with a dropdown menu set to 'Action Output'. The output shows a single action with the following details:

#	Time	Action	Message	Duration / Fetch
1	13:12:18	SELECT pr.id_product, COUNT(pr.id_transaction) AS ...	100 row(s) returned	1.110 sec / 0.000 sec

Se ha realizado la consulta a 'products_related' ya que ahí aparecen todos los productos de cada transacción, y se obtiene la cantidad de veces que aparece cada producto. A esto se le ha aplicado un JOIN con la tabla 'transactions' para filtrar que solo se consideren los productos de transacciones que no estén declinadas; por ser ventas efectivas, que es lo que solicitan en el ejercicio.

Este filtro es necesario, dado que la tabla 'products_related' contiene los productos de todas las transacciones declinadas y no declinadas de la tabla 'transactions'.

Revisión con ayuda de Ekaterina.