

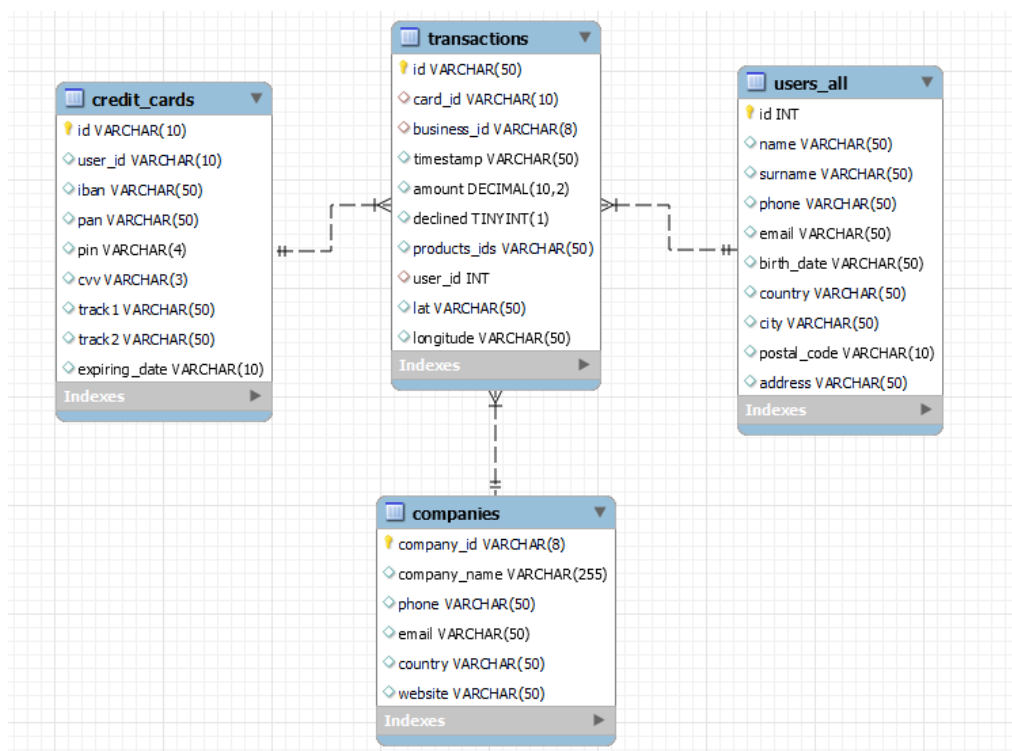
# Nivel 1

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 window with a query editor and a result grid. The query is as follows:

```

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 result grid shows 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 203.715 ya que ha realizado dos transacciones con la misma tarjeta.

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

```

1  ##N1. Exercici 2: Mostra la mitjana de la suma de transaccions per IBAN de les targetes de crèdit en la compa
2
3  • SELECT t.business_id, cc.iban, avg(t.amount) AS mitjana_trans
4    FROM transactions AS t
5   JOIN credit_cards AS cc
6     ON t.card_id=cc.id
7   JOIN companies AS c
8     ON c.company_id = t.business_id
9   WHERE company_name LIKE 'Donec Ltd%'
10  GROUP BY t.business_id, cc.iban;
11
12
13

```

The result grid shows the following data:

| business_id | iban                      | mitjana_trans |
|-------------|---------------------------|---------------|
| b-2242      | PT87806228135092429456346 | 203.715000    |

The output pane shows the following messages:

| #   | Time     | Action   | Message             |
|-----|----------|--|---------------------|
| 165 | 13:36:32 | SELECT * FROM TRANSACTIONS   | 587 row(s) returned |
| 166 | 13:37:16 | SELECT t.business_id, cc.iban, avg(t.amount) AS mitjana_trans FROM transactions A... | 1 row(s) returned   |

## Nivel 2

Antes de crear la tabla que refleje el estado de las tarjetas, cambiaremos el nombre del campo timestamp a fecha\_hora para evitar errores y 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 y **partition by** nos permite hacer una partición tanto de la enumeración, como de la suma, para separar por cada tarjeta. Con **order by** ordenamos las transacciones por fecha realizada.

```

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

| card_id  | fecha_hora          | declined | Status |
|----------|---------------------|----------|--------|
| CcU-2938 | 2022-03-12 09:23:00 | 0        | Activa |
| CcU-2938 | 2022-03-09 20:53:00 | 0        | Activa |
| CcU-2938 | 2022-02-24 11:01:00 | 0        | Activa |
| CcU-2938 | 2021-10-24 01:29:00 | 0        | Activa |
| CcU-2938 | 2021-10-17 03:52:00 | 0        | Activa |

card\_status 10 x

Output

| #  | Time     | Action   | Message  |
|----|----------|--|--|
| 22 | 11:07:18 | CREATE TABLE card_status AS ( SELECT t.card_id, t.fecha_hora, t.decline... | 587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0 |
| 23 | 11:07:20 | SELECT * FROM card_status  | 587 row(s) returned  |

Por último, añadimos una columna "Status" que analiza cada caso (**CASE WHEN**) y asigna el status "Inactiva" a las últimas tres transacciones (número de fila menor que 4,  $rn < 4$ ), sólo si la suma de las últimas tres transacciones es mayor o igual que 3, es decir si las últimas 3 transacciones han sido declinadas para el mismo número de card\_id.

**Como todas las transacciones por tarjeta salen Activas, editaremos dos transacciones para que se cumpla el caso Inactiva, para ver si las analiza correctamente.**

```

140
141 #-----PRUEBA PARA CORROBORAR QUE CASE FUNCIONA-----
142 #Cambiaremos a declined los datos de una tarjeta para que cumpla las condiciones de rechazo y verificar CASE
143 • DROP TABLE card_status;
144
145 #averiguamos el id de las últimas tres transacciones de la compañía con card_id CcU-2938 para cambiarlas a dec
146 • SELECT * FROM transactions
147 WHERE card_id LIKE '%2938'
148 ORDER BY fecha_hora DESC;
149
150 #Hacemos que las últimas tres transacciones sean declinadas
151 • SET SQL_SAFE_UPDATES = 0;
152 • UPDATE transactions
153 SET declined = 1
154 WHERE id IN ('AD85A78A-8829-5746-93A0-8B7A792EBC18', 'F1A598A2-86C5-50A9-F1CE-FB1D69866C39', '55166D02-D74C-6A
155

```

| #  | Time     | Action  | Message  |
|----|----------|---|--|
| 7  | 10:44:51 | SELECT * FROM card_status   | 587 row(s) returned  |
| 8  | 10:50:56 | DROP TABLE card_status  | 0 row(s) affected  |
| 9  | 10:51:00 | SELECT * FROM transactions WHERE card_id LIKE '%2938' ORDER BY fecha_hora...  | 24 row(s) returned   |
| 10 | 10:51:10 | SET SQL_SAFE_UPDATES = 0  | 0 row(s) affected  |
| 11 | 10:51:15 | UPDATE transactions SET declined = 1 WHERE id IN (AD85A78A-8829-5746-93A0-... | 3 row(s) affected Rows matched: 3 Changed: 3 Warnings: 0   |
| 12 | 10:52:20 | CREATE TABLE card_status AS (SELECT t.card_id, t.fecha_hora, t.decline...     | 587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0 |

Comprobamos con las transacciones editadas que el código funciona:

```

154 #Creamos la tabla de nuevo para corroborar que funciona correctamente:
155 • CREATE TABLE card_status AS (
156 SELECT t.card_id,
157 t.fecha_hora,
158 t.declined,
159 CASE
160 WHEN t.declined = 1 AND
161 ROW_NUMBER () OVER (PARTITION BY t.card_id ORDER BY t.fecha_hora DESC) < 4 AND
162 SUM(t.declined) OVER (PARTITION BY t.card_id) >= 3
163 THEN 'Inactiva'
164 ELSE 'Activa'
165 END AS Status
166 FROM transactions AS t);

```

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

| #  | Time     | Action  | Message  |
|----|----------|---|--|
| 28 | 11:10:55 | CREATE TABLE card_status AS (SELECT t.card_id, t.fecha_hora, t.decline... | 587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0 |
| 29 | 11:10:58 | 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.

The screenshot shows a SQL Developer window with a script named 'Giaroli - S04\*'. The script contains the following SQL statements:

```

170
171 #volvemos a dejar los datos de la tabla transaccion como estaban originalmente, eliminamos y creamos de nuevo
172 • DROP TABLE card_status;
173
174 • UPDATE transactions
175   SET declined = 0
176   WHERE id IN ('AD85A78A-8829-5746-93A0-8B7A792EBC18', 'F1A598A2-86C5-50A9-F1CE-FB1D69866C39', '55166D02-D74C-6A
177
178 • SET SQL_SAFE_UPDATES = 1;
179
180 #-----ACABADA LA COMPROBACIÓN DEL CÓDIGO CASE-----
181
182 #Creamos la tabla card_status sabiendo que funciona correctamente:
183 • CREATE TABLE card_status AS (
184     SELECT t.card_id,
185            t.fecha_hora,
186            t.declined,
187            CASE
188              WHEN t.declined = 1 AND
189                ROW_NUMBER () OVER (PARTITION BY t.card_id ORDER BY t.fecha_hora DESC) < 4 AND
  
```

The Output window shows the execution results:

| #  | Time     | Action  | Message  |
|----|----------|---|--|
| 31 | 11:12:52 | UPDATE transactions SET declined = 0 WHERE id IN (AD85A78A-8829-5746-93A0-... | 3 row(s) affected Rows matched: 3 Changed: 3 Warnings: 0 |
| 32 | 11:12:55 | SET SQL_SAFE_UPDATES = 1  | 0 row(s) affected  |

Añadimos una FK para vincular las tablas card\_status con credit\_cards:

The screenshot shows a SQL Developer window with a script named 'Giaroli - S04\*'. The script contains the following SQL statements:

```

286
287
288
289
290
291
292 #-----ADICIONAL PARA MEJORAR LA BASE DE DATOS-----
293 #Añadimos un FK la tabla card_status para vincularla a credit_cards
294 • ALTER TABLE card_status
295   ADD CONSTRAINT FOREIGN KEY (card_id) REFERENCES credit_cards(id);
296
297
  
```

The Output window shows the execution results:

| # | Time     | Action  | Message  |
|---|----------|---|--|
| 1 | 13:44:08 | ALTER TABLE card_status ADD CONSTRAINT FOREIGN KEY (card_id) REFERENCE... | 587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0 |

## 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 inactiva.

The screenshot shows a SQL Developer window with a query titled "Exercici 1: Quantes targetes estan actives?". The query is:

```
SELECT COUNT(distinct card_id)
FROM card_status
WHERE status = 'Activa';
```

The result grid shows a single row with the value 275 for the COUNT(distinct card\_id).

The Output window shows the following actions:

| #  | Time     | Action   | Message  |
|----|----------|--|--|
| 32 | 11:12:55 | SET SQL_SAFE_UPDATES = 1   | 0 row(s) affected  |
| 33 | 11:14:16 | CREATE TABLE card_status AS ( SELECT t.card_id, t.fecha_hora, t.decline... | 587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0 |
| 34 | 11:14:19 | SELECT * FROM card_status  | 587 row(s) returned  |
| 35 | 11:14:48 | SELECT COUNT(distinct card_id) FROM card_status WHERE status = 'Activa'    | 1 row(s) returned  |

## Nivel 3

Primero creamos la tabla Products e importamos los datos:

The screenshot shows a SQL Developer window with a query titled "Nivel 3". The query is:

```
CREATE TABLE products (
  id VARCHAR(50) PRIMARY KEY,
  product_name VARCHAR(50),
  price VARCHAR(50),
  colour VARCHAR(50),
  weight VARCHAR(50),
  warehouse_id VARCHAR(50));
SELECT * FROM products;
```

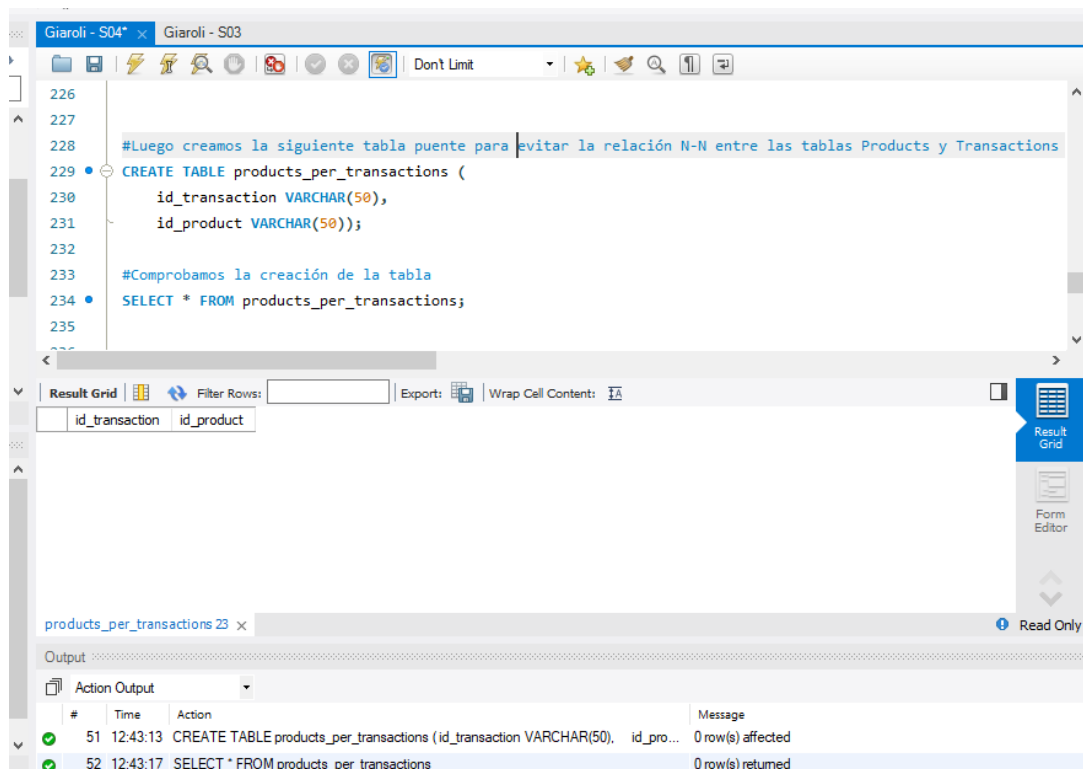
The result grid shows the following data:

| 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        |

The Output window 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.



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.

```

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 por transacción
249
250     #Comprobamos la inserción correcta de los datos
251 • SELECT * FROM products_per_transactions;
  
```

| id_transaction                       | id_product |
|--------------------------------------|------------|
| FE96CE47-8D59-381C-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         |

products\_per\_transactions 27 x

Output

Action Output

| #  | Time     | Action   | Message  |
|----|----------|--|--|
| 66 | 13:13:48 | INSERT INTO products_per_transactions (id_transaction, id_product) SELECT t.id AS... | 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) Haremos una **JOIN** con una tabla llamada numbers, creada por un producto cartesiano (es decir una **CROSS JOIN**) entre una secuencia de números generada con la función **ROW\_NUMBER** para cada fila de la tabla transactions y una tabla ficticia de 10 números, para asegurar que al menos habrá 10 filas disponibles para los productos de cada transacción
- 3) La JOIN la hacemos teniendo en cuenta el cálculo de la cantidad de productos que hay en cada registro del campo products\_ids, para ello restamos la función **LENGTH** para la longitud de la cadena con comas menos la sin comas, así sabremos la cantidad de comas que hay y le sumamos uno para tener el total de productos por registro.

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

```

| #  | 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 y añadimos un índice en la tabla products:

```

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

```

| #  | 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 tendremos en cuenta que la transacción no haya sido declined (declined = 0). Obtenemos el listado con la siguiente consulta:

The screenshot shows a database IDE with a SQL query in the editor and its result grid below it. 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 result grid shows 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                             |

The output pane shows the following messages:

| #  | 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, expresio |
| 85 | 14:11:38 | SELECT id_product, COUNT(distinct id_transaction) FROM products_per_transaction... | 26 row(s) returned   |

Para que el resultado pueda ser ordenado por id de producto, cambiaremos el formato de varchar a integer. Como es FK la desactivamos momentáneamente:

The screenshot shows a database IDE with a SQL script in the editor. The script is as follows:

```

255
256
257 #Para que el resultado nos salga ordenado por id de producto, cambiaremos el tipo de dato de varchar a int, co
258 • SHOW CREATE TABLE products_per_transactions;
259 • ALTER TABLE products_per_transactions
260 DROP FOREIGN KEY `products_per_transactions_ibfk_3`;
261
262 • ALTER TABLE products_per_transactions CHANGE id_product id_product INT;
263 • ALTER TABLE products CHANGE id id INT;
264
265 • ALTER TABLE products_per_transactions
266 ADD CONSTRAINT FOREIGN KEY (id_product) REFERENCES products(id);
267
268

```

The output pane shows the following messages:

| # | Time     | Action   | Message  |
|---|----------|--|--|
| 1 | 10:20:29 | SHOW CREATE TABLE products   | 1 row(s) returned  |
| 2 | 10:21:11 | SHOW CREATE TABLE products_per_transactions                                    | 1 row(s) returned  |
| 3 | 10:22:53 | ALTER TABLE products_per_transactions DROP FOREIGN KEY `products_per_transa... | 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0       |
| 4 | 10:26:12 | ALTER TABLE products_per_transactions CHANGE id_product id_product INT         | 1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0 |
| 5 | 10:26:15 | ALTER TABLE products CHANGE id id INT  | 100 row(s) affected Records: 100 Duplicates: 0 Warnings: 0   |
| 6 | 10:26:47 | ALTER TABLE products_per_transactions ADD CONSTRAINT FOREIGN KEY (id_prod...   | 1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0 |

Corremos de nuevo la misma consulta que ahora por defecto sale ordenada por id de producto:

268 #corremos nuevamente la consulta para obtener el listado de productos vendidos

269 • `SELECT id_product, COUNT(distinct id_transaction)`

270 `FROM products_per_transactions ppt`

271 `JOIN transactions t`

272 `ON ppt.id_transaction=t.id`

273 `WHERE t.declined = 0`

274 `GROUP BY id_product;`

275

| id_product | COUNT(distinct id_transaction) |
|------------|--------------------------------|
| 1          | 51                             |
| 2          | 56                             |
| 3          | 43                             |
| 5          | 42                             |
| 7          | 44                             |
| 11         | 40                             |
| 13         | 51                             |
| 17         | 54                             |
| 19         | 44                             |
| 23         | 60                             |

Result 3 x

Output

Action Output

| # | Time     | Action   | Message  |
|---|----------|--|--|
| 6 | 10:26:47 | ALTER TABLE products_per_transactions ADD CONSTRAINT FOREIGN KEY (id_pr...         | 1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0 |
| 7 | 10:31:14 | SELECT id_product, COUNT(distinct id_transaction) FROM products_per_transaction... | 26 row(s) returned   |

Para interpretar los resultados, podemos ver el primer renglón que el producto con id = 1 se ha vendido 51 veces. Si el producto no aparece en el listado es porque aún no se ha vendido ninguna vez.

Si quisiéramos obtener un listado tanto con los productos que se han vendido, como los que no se han vendido nunca, podemos hacer una subquery con la consulta anterior:

276 #si también quisiéramos que salgan los productos que no se han vendido la consulta sería la siguiente:

277 • `SELECT p.id AS id_producto, cant_transacc`

278 `FROM products p`

279 `LEFT JOIN ( SELECT id_product, COUNT(distinct id_transaction) AS cant_transacc`

280 `FROM products_per_transactions ppt`

281 `JOIN transactions t`

282 `ON ppt.id_transaction=t.id`

283 `WHERE t.declined = 0`

284 `GROUP BY id_product) cant_vendida`

285 `ON cant_vendida.id_product=p.id;`

286

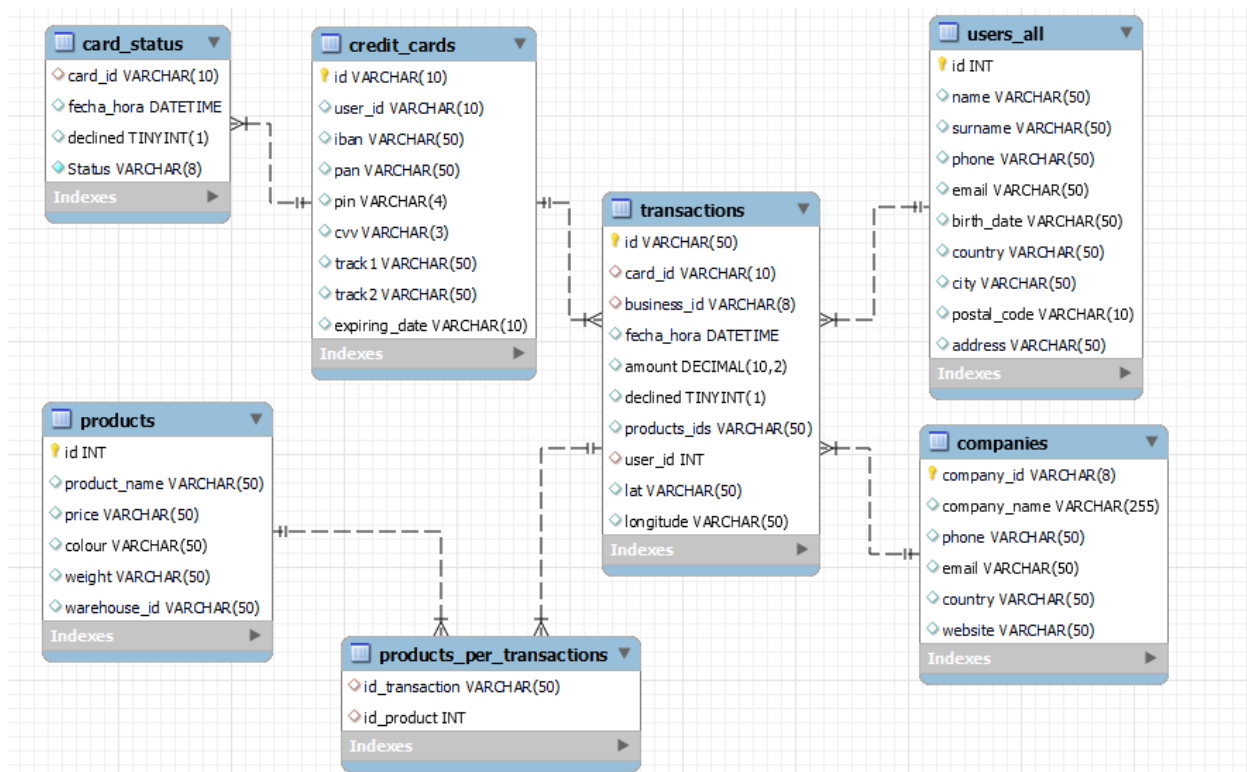
| id_producto | cant_transacc |
|-------------|---------------|
| 1           | 51            |
| 2           | 56            |
| 3           | 43            |
| 4           | NULL          |
| 5           | 42            |
| 6           | NULL          |
| 7           | 44            |

Result 7 x

Output

Action Output

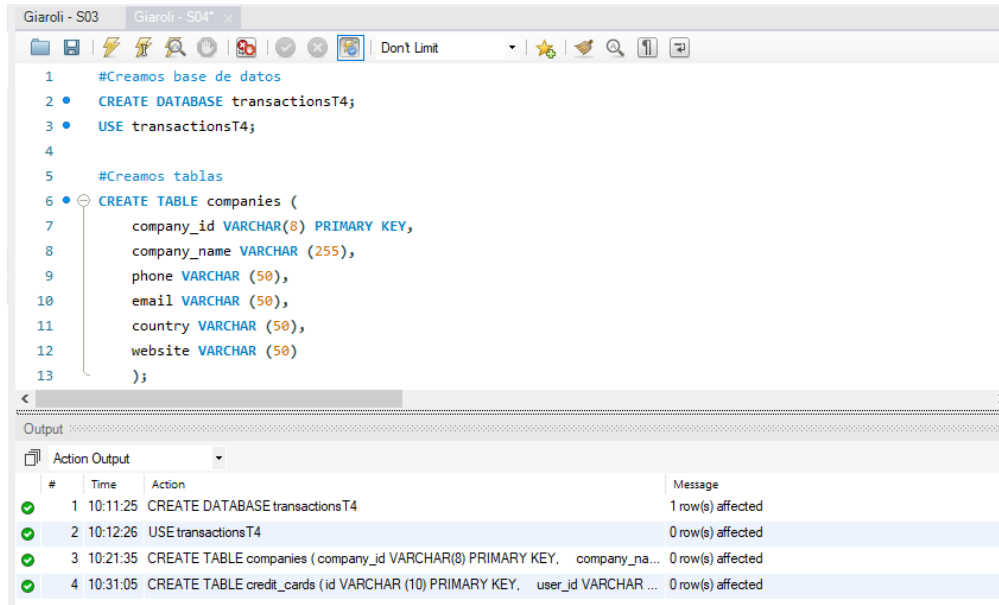
| #  | Time     | Action   | Message             |
|----|----------|--|---------------------|
| 12 | 10:46:37 | SELECT p.id, cant_transacc FROM products p LEFT JOIN ( SELECT id_product, CO...  | 100 row(s) returned |
| 13 | 10:47:01 | SELECT p.id AS id_producto, cant_transacc FROM products p LEFT JOIN ( SELECT ... | 100 row(s) returned |

**ESTRUCTURA DE LA BASE DE DATOS AL FINAL DEL SPRINT:**

## 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 IDE window with two tabs: 'Giaroli - S03' and 'Giaroli - S04\*'. The active tab 'Giaroli - S04\*' contains the following SQL code:

```

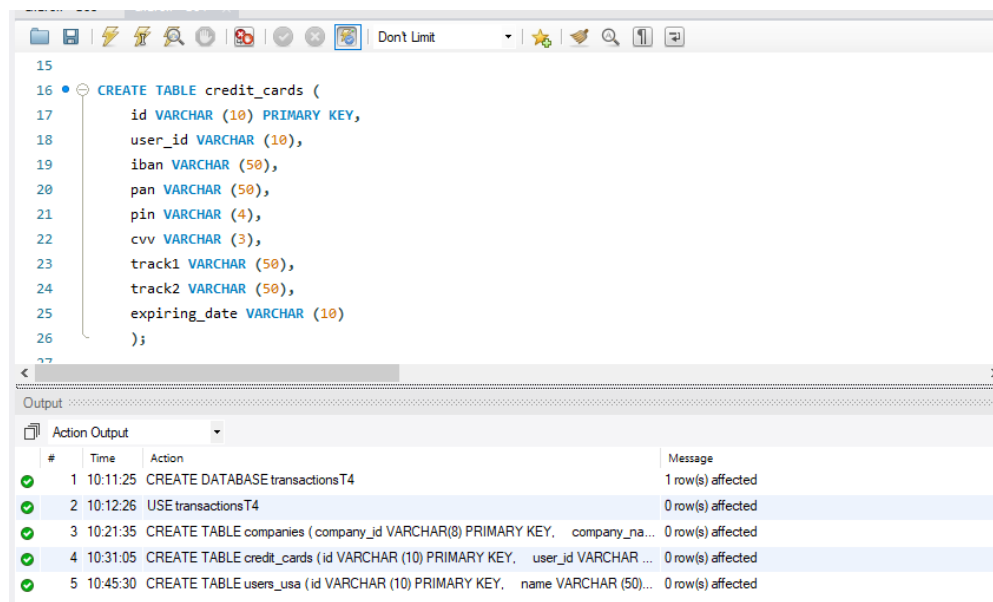
1  #Creamos base de datos
2  • CREATE DATABASE transactionsT4;
3  • USE transactionsT4;
4
5  #Creamos tablas
6  • CREATE TABLE companies (
7      company_id VARCHAR(8) PRIMARY KEY,
8      company_name VARCHAR (255),
9      phone VARCHAR (50),
10     email VARCHAR (50),
11     country VARCHAR (50),
12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

```

Below the code editor is the 'Output' panel, which shows the execution results of the SQL commands:

| # | 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 IDE window with the following SQL code in the active tab:

```

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 );

```

Below the code editor is the 'Output' panel, which shows the execution results of the SQL commands:

| # | 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 |

The screenshot shows a database management tool interface. The top pane displays the SQL command to create a table named `users_all`:

```

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 );

```

The bottom pane shows a result grid with 10 columns: `id`, `card_id`, `business_id`, `timestamp`, `amount`, `declined`, `products_ids`, `user_id`, and `lat`. The grid contains 5 rows of data:

| 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 |

Below the result grid, the 'Output' pane shows the execution log:

| #  | 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 |

The screenshot shows the same database management tool interface. The top pane displays the SQL command to create a table named `transactions`:

```

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),

```

The bottom pane shows the same result grid as the previous screenshot, containing 5 rows of transaction data.

The 'Output' pane shows the execution log:

| #  | 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 *`