Sprint 4

Nivell 1

Descàrrega els arxius CSV, estudia'ls i dissenya una base de dades amb un esquema d'estrella que contingui, almenys 4 taules de les quals puguis realitzar les següents consultes:

Lo primero que tenemos que hacer es estudiar el contenido de las tablas y diseñar el esquema que utilizaremos para crear la base de datos y poder responder a las consultas.

Una vez analizado el contenido de los csv's, utilizamos app diagrams para crear el esquema que utilizaremos para nuesta database.

El esquema que crearemos es de tipo estrella y quedaría de la siguiente forma.

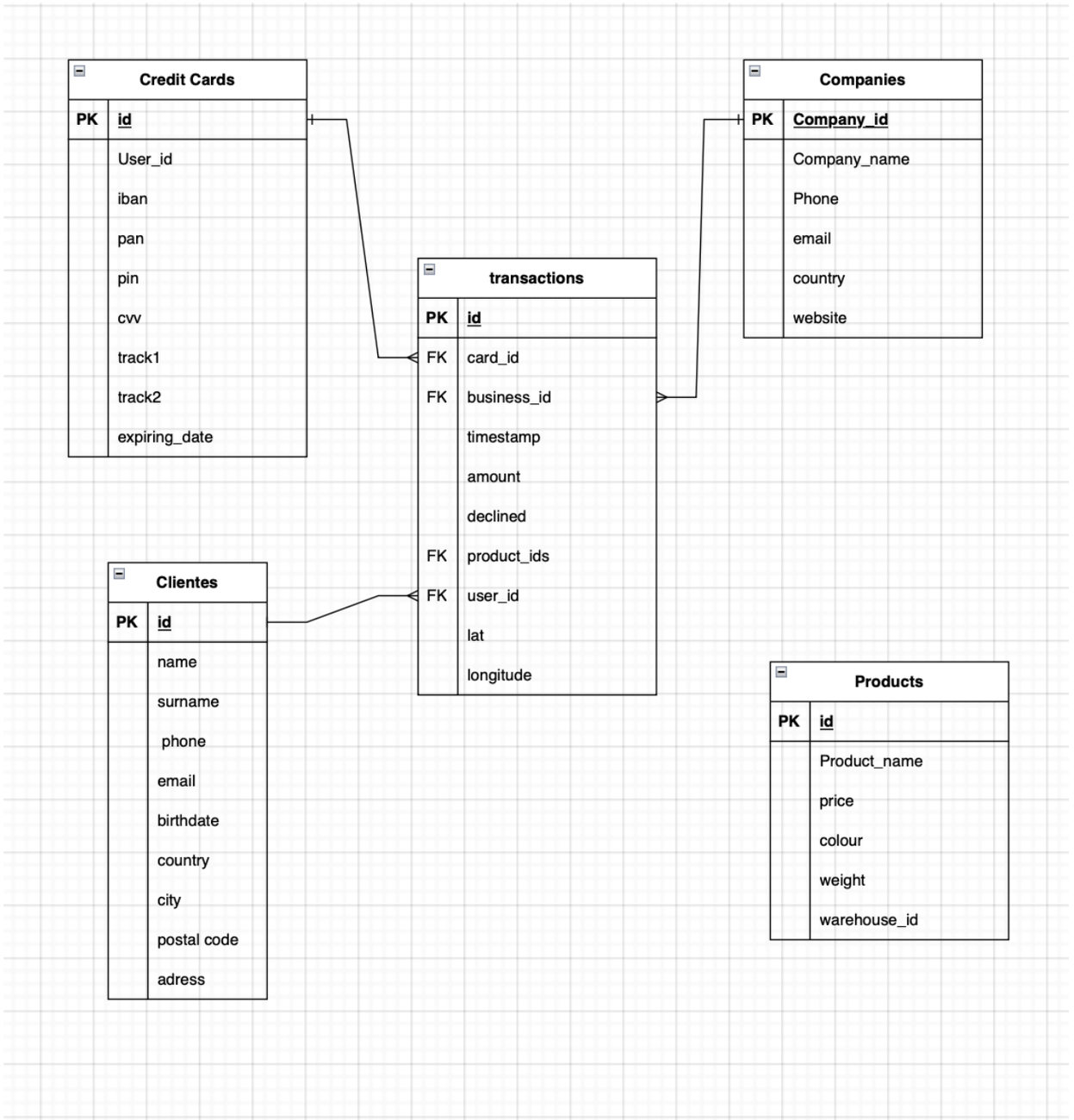
De momento, no crearemos la unión entre las tablas products y transactions ya que como vemos tienen una relación N-N y esto nos daría error al intentar generar la llave foranea de la tabla transactions.

Así que comenzaremos creado la database solo con las 4 primeras tablas y dejaremos la tabla products para más adelante.

En este diagrama definimos las PK y las FK de nuestras tablas y podemos observar su contanido y sus relaciones entre ellas.

Observamos que la tabla transactions es nuestra tabla de Hechos y que el resto de las tablas son las tablas de Dimensión teniendo entre ellas una relación de 1 a N.

Una vez creado el esquema, damos paso a crear la Database para poder crear las tablas, generar los enlaces entre ellas.



Utilizamos el comando CREATE DATABASE para crear la database.

```
17    -- creamos el database
18    CREATE DATABASE sprint4;
```

```
25399    19:34:16    CREATE DATABASE sprint4                    1 row(s) affected
```

Lo siguiente que tenemos que hacer es crear las tablas y cargar los datos de los archivos CSV

```
21    -- creamos la tabla transaction
22
23    CREATE TABLE transactions (
24    id VARCHAR(50) not null,
25    card_id VARCHAR(50),
26    business_id VARCHAR(50),
27    timestamp timestamp ON UPDATE CURRENT_TIMESTAMP,
28    amount FLOAT,
29    declined BOOL,
30    product_ids VARCHAR(20),
31    user_id VARCHAR(20),
32    lat FLOAT,
33    longitude FLOAT,
34    primary key(id));
```

```
4    12:33:23    CREATE TABLE transactions ( id VARCHAR(50) not null, card_id VARCHAR(50), business_...    0 row(s) affected
```

Utilizando el comando CREATE TABLE generamos la tabla transactions y definimos sus columnas, así como el tipode dato que van a contener. También definiremos cuál será la Primary KEY y que sabemos que no puede contener NULL,

Lo siguiente que haremos será cargar los datos a la tabla transactions que acabamos de crear.

```
38    LOAD DATA LOCAL INFILE '/Users/gabrielpsantana/Documents/IT_ACADEMY/Sprint\ 4/transactions.csv'
39    INTO TABLE transactions
40    FIELDS TERMINATED BY ';'
41    IGNORE 1 ROWS;
```

```
8   12:52:59   LOAD DATA LOCAL INFILE '/Users/gabrielpsantana/Documents/IT_ACADEMY/Sprint\ 4/tra...   100000 row(s) affected Records: 100000  Deleted: 0  Skipped: (
```

Para crear esta tabla hemos utilizado la función FIELDS TERMINATED BY para delimitar los campos de los datos y también hemos ignorado la primera fila para los encabezados de las columnas utilizando la función IGNORE.

Ahora haremos lo mismo con el resto de tablas.

```
43    -- creamos tabla clientes
44
45    CREATE TABLE clientes (
46    id VARCHAR(20) not null,
47    name CHAR(50),
48    surname CHAR(50),
49    phone VARCHAR(50),
50    email VARCHAR(250),
51    birthday VARCHAR(50),
52    country CHAR(50),
53    city CHAR(50),
54    postal_code VARCHAR(20),
55    adress VARCHAR(250),
56    primary key(id));
```

```
21   13:17:57   CREATE TABLE clientes ( id VARCHAR(50) not null, name CHAR(50), surname CHAR(50),...   0 row(s) affected
```

```sql
-- cargamos información tabla clientes

LOAD DATA LOCAL INFILE  '/Users/gabrielpsantana/Documents/IT_ACADEMY/Sprint\ 4/american_users.csv'
INTO TABLE clientes
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
IGNORE 1 ROWS;

-- cargamos el otro documento de clientes a la tabla

LOAD DATA LOCAL INFILE  '/Users/gabrielpsantana/Documents/IT_ACADEMY/Sprint\ 4/european_users.csv '
INTO TABLE clientes
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
IGNORE 1 ROWS;
```

```
34  13:37:07  LOAD DATA LOCAL INFILE  '/Users/gabrielpsantana/Documents/IT_ACADEMY/Sprint\ 4/am...  1010 row(s) affected Records: 1010  Deleted: 0  Skipped: 0  Wa
36  13:41:47  LOAD DATA LOCAL INFILE  '/Users/gabrielpsantana/Documents/IT_ACADEMY/Sprint\ 4/eu...  3990 row(s) affected Records: 3990  Deleted: 0  Skipped: 0  W
```

```sql
-- creamos la tabla credit_cards

CREATE TABLE credit_cards (
id VARCHAR(50) not null,
user_id VARCHAR(20),
iban VARCHAR(50),
pan VARCHAR(20),
pin CHAR(6),
cvv CHAR(6),
track1 VARCHAR(250),
rtack2 VARCHAR(250),
expiring_date VARCHAR(20),
primary key(id));
```

```
37  13:44:03  CREATE TABLE credit_cards ( id VARCHAR(50) not null, user_id VARCHAR(20), iban VAR...  0 row(s) affected
```

```sql
-- cargamos la información a credit__cards

LOAD DATA LOCAL INFILE  '/Users/gabrielpsantana/Documents/IT_ACADEMY/Sprint\ 4/credit_cards.csv'
INTO TABLE credit_cards
FIELDS TERMINATED BY ','
IGNORE 1 ROWS;
```

```
46  19:11:22  LOAD DATA LOCAL INFILE '/Users/gabrielpsantana/Documents/IT_ACADEMY/Sprint\ 4/cre...  5000 row(s) affected Records: 5000  Deleted: 0  Skipped: 0  W
```

```sql
-- creamos la tabla companies

CREATE TABLE companies (
company_id VARCHAR(50) not null,
company_name VARCHAR(250),
phone VARCHAR(50),
email VARCHAR(250),
country CHAR(50),
website VARCHAR(250),
primary key(company_id));
```

```
48  19:14:11  CREATE TABLE companies ( company_id VARCHAR(50) not null, company_name VARCHA...  0 row(s) affected
```

```sql
-- cargamos la información a tabla companies

LOAD DATA LOCAL INFILE  '/Users/gabrielpsantana/Documents/IT_ACADEMY/Sprint\ 4/companies.csv'
INTO TABLE companies
FIELDS TERMINATED BY ','
IGNORE 1 ROWS;
```

```
49  19:16:22  LOAD DATA LOCAL INFILE '/Users/gabrielpsantana/Documents/IT_ACADEMY/Sprint\ 4/co...  100 row(s) affected Records: 100  Deleted: 0  Skipped: 0  Warn
```

```sql
-- creamos la tabla products

CREATE TABLE products (
id VARCHAR(50) not null,
product_name VARCHAR(250),
price CHAR(50),
colour VARCHAR(50),
weight FLOAT(50),
warehouse_id VARCHAR(50),
primary key(id));
```

```
50  19:19:31  CREATE TABLE products ( id VARCHAR(50) not null, product_name VARCHAR(250), price...  0 row(s) affected
```

```sql
-- cargamos la información a tabla products

LOAD DATA LOCAL INFILE  '/Users/gabrielpsantana/Documents/IT_ACADEMY/Sprint\ 4/products.csv'
INTO TABLE products
FIELDS TERMINATED BY ','
IGNORE 1 ROWS;
```

```
57  19:39:37  LOAD DATA LOCAL INFILE  '/Users/gabrielpsantana/Documents/IT_ACADEMY/Sprint\ 4/pr...  100 row(s) affected Records: 100  Deleted: 0  Skipped: 0  Warn
```

Ahora ya hemos creado todas las tablas de nuestra Database. Como reflexión a éste ejercicio, hay que remarcar la importancia de elegir correctamente el tipo de dato de cada columna y hay que prestar especial atención al tipo de dato de las PK y las FK ya que si no son el mismo tipo de dato obtendremos errores a la hora vincularlas o también podemos tener problemas al intentar cargar la información de los documentos.

Lo siguiente que tenemos que hacer es vincular las tablas, para ello utilizaremos la función ALTER TABLE para añadir el CONSTRAINT y las Foreign Keys

```sql
-- ahora creamos las foreign keys

ALTER TABLE transactions
ADD CONSTRAINT fk_card_id
FOREIGN KEY (card_id)
REFERENCES credit_cards(id);
```

```
63  19:53:09  ALTER TABLE transactions ADD CONSTRAINT fk_card_id FOREIGN KEY (card_id)  REFER...  100000 row(s) affected Records: 100000  Duplicates: 0  Warnir
```

```sql
ALTER TABLE transactions
ADD CONSTRAINT fk_business_id
FOREIGN KEY (business_id)
REFERENCES companies(company_id);
```

```
64  19:53:52  ALTER TABLE transactions ADD CONSTRAINT fk_business_id FOREIGN KEY (business_id...  100000 row(s) affected Records: 100000  Duplicates: 0  Warnir
```

```sql
ALTER TABLE transactions
ADD CONSTRAINT fk_user_id
FOREIGN KEY (user_id)
REFERENCES clientes(id);
```

```
71  19:59:10  ALTER TABLE transactions ADD CONSTRAINT fk_user_id FOREIGN KEY (user_id)  REFER...  100000 row(s) affected Records: 100000  Duplicates: 0  Warnir
```

* Como mencionamos anteriormente, de momento no vincularemos la tabla Products con la tabla Transactions, ya que no tienen una relación directa y tenemos que hacer modificaciones para poder crear ese vínculo.

Ahora que ya tenemos el resto de tablas de dimensiones vinculadas con la tabla de hechos, podemos realizar la primera consulta

Exercici 1

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

```sql
-- Exercici 1
-- Realitza una subconsulta que mostri tots els usuaris amb més de 80 transaccions utilitzant almenys 2 taules.

SELECT *
FROM clientes as c
HAVING c.id IN (
                SELECT t.user_id
                FROM transactions as t
                GROUP BY t.user_id
                HAVING COUNT(DISTINCT t.id)>80)
;
```

id	name	surname	phone	email	birthday	country	city	postal_code	adress
185	Molly	Gilliam	0800 120 8023	donec@outlook.couk	Dec 21, 1993	United Ki…	London	EC1A 1BB	P.O. Box 202, 5…
289	Dxwgi	Hwcru	+98-309-8797	dxwgi.hwcru@example.com	Aug 20, 1976	Germany	Stuttgart	70173	82 Hwcru Street
318	Bnyr	Astuw	+33-120-9644	bnyr.astuw@example.com	May 3, 1974	Italy	Genoa	16100	53 Astuw Street
454	Sfzzoh	Xgvfridxs	+58-495-3945	sfzzoh.xgvfridxs@example.com	Aug 28, 1962	Poland	Gdansk	80-001	52 Xgvfridxs St…
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

59 17:53:42 SELECT * FROM clientes as c HAVING c.id IN (SEL... 4 row(s) returned 0.101 sec / 0.000013...

Para este ejercicio, primero hacemos la subconsulta en la que seleccionamos los user_id de la tabla transactions que cumplan con la condición de que tengan más de 80 transacciones y para filtrarlos, utilizamos un HAVING COUNT DISTINCT de los id de transactions. Y luego utilizamos ésta subconsulta para filtrar los resultados de la tabla clientes de la que hacemos una selección de todos los campos y utilizamos de nuevo el HAVING para filtrar la información.

Exercici 2

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

```sql
-- Exercici 2
-- Mostra la mitjana d'amount per IBAN de les targetes de crèdit a la companyia Donec Ltd, utilitza almenys 2 ta

SELECT cc.id, cc.iban, TRUNCATE(AVG(t.amount), 2) as media,c.company_name
FROM credit_cards as cc
JOIN transactions as t
ON cc.id=t.card_id
JOIN companies as c
ON t.business_id=c.company_id
WHERE company_name LIKE "Donec Ltd"
GROUP BY cc.id,cc.iban,c.company_name
ORDER BY media DESC;
```

id	iban	media	company_name
CcS-7935	XX3830178139196201993663 52	680.69	Donec Ltd
CcS-6398	XX6377063573975703949739 13	680.01	Donec Ltd
CcS-6374	XX9713939714652922023122 59	645.46	Donec Ltd
CcS-7851	XX1718471169288923759693 07	628.89	Donec Ltd
CcS-6502	XX2254246388185424062235 75	608.67	Donec Ltd
CcS-5121	XX7488907290571957117660 71	607.28	Donec Ltd
CcU-3239	TN9614563570667381893122	605.4	Donec Ltd
CcS-6646	XX4819080340373642425911 85	605.35	Donec Ltd
CcS-7186	XX1946755197392563357535 08	597.19	Donec Ltd

```
344  13:50:09  SELECT cc.id, cc.iban, TRUNCATE(AVG(t.amount), 2) as media,c.company_name FROM cr...  371 row(s) returned
```

Para esta consulta seleccionamos el id de la tarjeta, el iban, sacamos la media (AVG) de las transsacciones y el nombre de la compañía. Al utilizar diferentes tablas, debemos hacer un JOIN entre las tablas credit_cards y transactions y por último filtrar los resultados por el nombre de la compañía utilizando un WHERE y un LIKE. Por último, ordenaremos los datos a partir de la media en orden descendente.

Nivell 2

Crea una nova taula que reflecteixi l'estat de les targetes de crèdit basat en si les últimes tres transaccions van ser declinades i genera la següent consulta

Exercici 1

Quantes targetes estan actives?

Lo primero que tenemos que hacer para resolver ésta consulta es crear una tabla en la que reflejaremos el estado de las tarjetas. Para ello solo requeriremos de dos columnas, la columna del id de la tarjeta y otra columna que llamaremos estado en la que nos indica si la la tarjeta esta activa o bloqueada.

```
179    -- Crea una nova taula que reflecteixi l'estat de les targetes de crèdit basat en si les últime
180    -- transaccions van ser declinades i genera la següent consulta.
181
182    CREATE TABLE tarjetas_activas (
183    card_id VARCHAR(50) NOT NULL,
184    estado VARCHAR(50),
185    PRIMARY KEY (card_id));
```

```
20  12:46:48  CREATE TABLE tarjetas_activas ( card_id VARCHAR(50) NOT NULL, e...  0 row(s) affected  0.0067 sec
```

Para crear la tabla utilizamos la función CREATE TABLE, introducimos las columnas y el tipo de dato que contendrán y aprovecharemos para crear la Primary Key que será el id de la tarjeta y que como ya sabemos no puede contener datos nulos ni duplicados para que pueda servir como PK.

Lo siguiente que necesitamos es conocer cuáles son las últimas 3 transacciones de cada tarjeta.

Para ello, utilizamos la función row_number que cuenta las filas y nos permite diferenciar el campo del que realiza el conteo mediante la función (Partition By) y además nos permite ordenar el conteo utilizando otro campo. Que en éste caso, ordenaremos en función del Timestamp de forma descendiente para que las transacciones vayan de la mas reciente a la más antigua. De ésta forma sabemos cuáles son las tres últimas transacciones.

```
225    SELECT *,
226    row_number() OVER (partition by card_id ORDER BY timestamp DESC) as contador
227    FROM transactions;
```

id	card_id	business_id	timestamp	amount	declined	product_ids	user_id	lat	longitude	contador
BDD988A9-51B9-4BD8-9E4C-84…	CcS-4857	b-2610	2024-10-25 13:11:54	129.3	0	46, 48, 24	276	40.3093	-3.45906	1
B954A0B3-9314-4615-ACBB-B3…	CcS-4857	b-2558	2024-10-07 16:43:17	107.17	0	35	276	40.3055	-3.74459	2
2C537AD9-DB0E-402A-A75F-7D…	CcS-4857	b-2458	2024-07-27 10:50:49	365.39	0	19, 25, 14	276	40.0486	-3.65004	3
40AB0B56-343B-40A9-8F5B-8C…	CcS-4857	b-2550	2024-02-08 14:55:10	221.27	1	98, 56, 81	276	40.6607	-3.96141	4
FD857386-28F9-49DE-8F32-E24…	CcS-4857	b-2434	2024-01-15 20:18:34	204.63	0	48, 54, 23	276	40.9611	-3.96145	5
BBCF25B8-420E-4EAE-83C0-F2…	CcS-4857	b-2378	2022-09-21 22:52:27	273.9	0	28, 8, 73	276	40.5137	-4.23886	6
6BE556F4-9B3F-4C64-A4E2-23…	CcS-4857	b-2442	2022-05-20 08:36:18	639.02	0	14, 39, 65, 27	276	40.0317	-4.1694	7
CA32CD20-B2CC-4BA7-8B57-9…	CcS-4857	b-2442	2021-10-02 03:58:28	376.99	0	40, 59, 45, 87	276	40.7634	-4.19461	8
6EE02B25-00A0-4880-A606-3E2…	CcS-4857	b-2434	2021-09-30 07:08:18	154.24	0	91, 75, 31	276	40.8251	-3.30613	9
09B6BBB0-AAD2-4114-944C-6F…	CcS-4857	b-2466	2021-06-30 22:34:08	362.11	0	12, 7, 87, 96	276	40.8603	-3.80518	10

21 12:57:09 SELECT *, row_number() OVER (partition by card_id ORDER BY timest… 100000 row(s) returned 0.159 sec / 0.378 sec

Aquí podemos ver en la columna (contador) como nos muestra el conteo de las transacciones, ordenadas por el Timestamp de más reciente a más antigua.

Una vez que conocemos cuáles son las últimas transacciones de cada tarjeta, crearemos un CASE que nos relfeje el estado de la tarjeta y que cumpla con las siguientes condiciones. Que el conteo de los registros sea igual a 3, que la suma de Declined sea igual a 3 también y por último que el número del contador sea menor o igual a 3. Como estamos creando un CASE, podemos aprovechar para introducir los resultados directamente en la tabla que hemos creado anteriomente.

```sql
INSERT INTO tarjetas_activas (card_id,estado)
SELECT t.card_id,
CASE WHEN COUNT(*)=3 AND SUM(declined)=3 THEN "bloqueada"
ELSE "activa"
END as estado
FROM (SELECT *,
row_number() OVER (partition by card_id ORDER BY timestamp DESC) as contador
FROM transactions) as t
WHERE contador<=3
GROUP BY t.card_id;
```

```
30  13:46:38  INSERT INTO tarjetas_activas (card_id,estado) SELEC...  5000 row(s) affected Records: 5000  Duplicates: 0  Warnings: 0  0.606 sec
```

Utilizamos la función INSERT INTO para agregar los datos de nuestra función CASE en la que seleccionamos los card_id y nos genera la columna estado, que determinará si la tarjeta está bloqueada si se cumplen las tres condiciones mencionadas anteriormente y si no se cumlplen la tarjeta está activa.

Hacemos una selección de todos los registros de la tabla que hemos generado para comprobar si se han introducido los datos de forma correcta.

```sql
select * from tarjetas_activas;
```

card_id	estado
CcS-4868	activa
CcS-4869	activa
CcS-4870	bloqueada
CcS-4871	activa
CcS-4872	activa
CcS-4873	activa

```
31  13:48:53  select * from tarjetas_activas  5000 row(s) returned  0.0015 sec / 0.0024 s...
```

Y por último hacemos una selección manual de alguna tarjeta bloqueada para comprobar los datos y observamos que las 3 últimas transacciones han sido declinadas.

```sql
select *
FROM transactions
WHERE card_id LIKE "%4870"
ORDER BY timestamp DESC;
```

id	card_id	business_id	timestamp	amount	declined	product_ids	user_id	lat	longitude
E9F2F014-876D-4269-8226-FF8...	CcS-4870	b-2358	2024-04-01 21:30:17	264.26	1	24, 97, 22	289	50.6952	9.98768
367316BC-488F-430D-A8A7-6E...	CcS-4870	b-2518	2024-02-11 12:41:35	449.72	1	87, 4, 50, 25	289	50.9986	10.8164
80956D50-5F3E-44C0-8624-CC...	CcS-4870	b-2546	2023-09-08 14:00:06	103.4	1	60, 80, 96	289	51.2349	10.8797
004CC633-F31E-4DD5-86A1-57...	CcS-4870	b-2250	2023-03-13 02:38:09	211.19	0	65, 85, 37	289	50.813	10.2382
6D8E6E64-D128-4FBA-9D18-64...	CcS-4870	b-2374	2022-05-31 13:01:11	376.6	0	75, 10, 9, 5	289	51.1125	10.6326

```
32  13:53:11  select * FROM transactions WHERE card_id LIKE "%4...  94 row(s) returned  0.058 sec / 0.00003...
```

Exercici 1

Quantes targetes estan actives?

```
206     -- Quantes targetes estan actives?
207
208 •   SELECT * FROM tarjetas_activas
209     WHERE estado="bloqueada";
210
211 •   SELECT COUNT(card_id) as total_activas
212     FROM tarjetas_activas
213     WHERE estado ="activa";
```

	card_id	estado
▶	CcS-4870	bloqueada
	CcS-4899	bloqueada
	CcS-4998	bloqueada
	CcS-5035	bloqueada
	CcU-3568	bloqueada
	NULL	NULL

```
33  18:23:22  SELECT * FROM tarjetas_activas WHERE estado="blo...  5 row(s) returned  0.0017 sec / 0.00001...
```

	total_activas
▶	4995

```
34  18:24:37  SELECT COUNT(card_id) as total_activas FROM tarje...  1 row(s) returned  0.0020 sec / 0.00000
```

Para responder a la pregunta de cuántas tarjetas están activas he realizado dos consultas, una en la que nos muestra qué tarjetas están bloquedas y otra en la que hacemos un conteo de las tarjetas activas, haciendo el filtro con un condicional WHERE y su estado, utilizando la tabla que hemos creado anteriormente.

Nivell 3

Crea una taula amb la qual puguem unir les dades del nou arxiu products.csv amb la base de dades creada, tenint en compte que des de transaction tens product_ids. Genera la següent consulta:

Exercici 1

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

Para resolver ésta consulta, lo primero que tenemos que hacer es crear la tabla dónde colocaremos los datos, para ello utilizamos CREATE TABLE con las columnas de id y product_id y elegimos el tipo de variable.

```
236     -- Crea una taula amb la qual puguem unir les dades del nou arxiu products.csv amb la base de dades creada,
237     -- tenint en compte que des de transaction tens product_ids. Genera la següent consulta:
238
239     -- Exercici 1
240     -- Necessitem conèixer el nombre de vegades que s'ha venut cada producte.
241
242
243     CREATE TABLE transitionproduct (
244     id  VARCHAR (255),
245     product_id VARCHAR(50));
```

```
11  12:27:59  CREATE TABLE transitionproduct (  id  VARCHAR (25...  0 row(s) affected  0.0084 sec
```

Lo siguiente que tenemos que hacer es insertar en nuestra tabla los datos y para ello debemos separar los product_ids que están divididos por cómas, pero manteniendo la relación con los datos de los id de la transacción. Para ello utilizaremos una Common Table Expression (CTE) con una función RECURSIVE, que asignará una variable en función de la cantidad de product_ids que tengamos. Y luego utilizaremos una función SUBTRING_INDEX para separar los product_ids de forma individual.

```sql
INSERT INTO transitionproduct (id,product_id)
WITH RECURSIVE numero as (
    SELECT 1 as n
    UNION ALL
    SELECT n + 1 FROM numero WHERE n < 100
),
split_ids as (
    SELECT
        id,
        SUBSTRING_INDEX(SUBSTRING_INDEX(t.product_ids, ',', numero.n), ',', -1) as product_id
    FROM transactions as t
    JOIN numero ON numero.n <= CHAR_LENGTH(t.product_ids) - CHAR_LENGTH(REPLACE(t.product_ids, ',', '')) + 1
)
SELECT id,product_id
FROM split_ids;
```

```
12  12:28:04  INSERT INTO transitionproduct (id,product_id) WITH...  253391 row(s) affected Records: 253391  Duplicates: 0  Warnings: 0  3.381 sec
```

Aquí vemos como queda la consulta y comprobamos que los datos se han introducido correctamente en nuestra tabla.

```sql
SELECT * FROM transitionproduct;
```

id	product_id
00043A49-2949-494B-A5DD-A5BAE3BB19DD	26
00043A49-2949-494B-A5DD-A5BAE3BB19DD	16
000447FE-B650-4DCF-85DE-C7ED0EE1CAAD	87
000447FE-B650-4DCF-85DE-C7ED0EE1CAAD	69
000447FE-B650-4DCF-85DE-C7ED0EE1CAAD	66
00045D6B-ED2E-4F2F-8186-CEE074D875D0	81
00045D6B-ED2E-4F2F-8186-CEE074D875D0	16
00045D6B-ED2E-4F2F-8186-CEE074D875D0	11

```
39  10:36:14  SELECT * FROM transitionproduct  253391 row(s) returned  0.0011 sec / 0.262 sec
```

Ahora lo que tenemos que hacer es limpiar los espacios en blanco que se han generado en la columna product_id y para ello utilizamos la función TRIM.

```sql
UPDATE transitionproduct SET product_id = TRIM(product_id);
```

```
40  12:32:45  UPDATE transitionproduct SET product_id = TRIM(pr...  153391 row(s) affected Rows matched: 253391  Changed: 153391  Warnings: 0  1.705 sec
```

Y ahora con los datos limpios podemos crear las Foreign Keys para poder relacionar la tabla que hemos creado con la tabla Transactions y la tabla Products.

```
265 •   ALTER TABLE transitionproduct
266     ADD CONSTRAINT fk_transactions
267     FOREIGN KEY (id)
268     REFERENCES transactions(id);
```

```
14  12:30:31  ALTER TABLE transitionproduct ADD CONSTRAINT fk...  253391 row(s) affected Records: 253391  Duplicates: 0  Warnings: 0  2.197 sec
```

```
277 •   ALTER TABLE transitionproduct
278     ADD CONSTRAINT fk_products
279     FOREIGN KEY (product_id)
280     REFERENCES products(id);
```

```
41  12:39:22  ALTER TABLE transitionproduct ADD CONSTRAINT fk...  253391 row(s) affected Records: 253391  Duplicates: 0  Warnings: 0  1.556 sec
```

Una vez que tenemos nuestra tabla vinculada con el resto de tablas podemos resolver la consulta.

Exercici 1

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

```
278     -- Necessitem conèixer el nombre de vegades que s'ha venut cada producte.
279 •   SELECT p.id,p.product_name,COUNT(tp.product_id) AS ventas
280     FROM products p
281     JOIN transitionproduct AS tp
282     ON p.id=tp.product_id
283     group by tp.product_id
284     ORDER BY p.id;
```

id	product_name	ventas
1	Direwolf Stannis	2468
10	Karstark Dorne	2571
100	south duel	2517
11	Karstark Dorne	2573
12	duel Direwolf	2558
13	palpatine chewbacca	2560
14	Direwolf	2496
15	Stannis warden	2535

```
42  12:41:28  SELECT p.id,p.product_name,COUNT(tp.product_id)...  100 row(s) returned  0.567 sec / 0.00002
```

Para resolver la consulta hacemos la selección del product_id, el nombre del producto y hacemos un conteo del product_id de la tabla que hemos generado anteriormente y finalmente agrupamos por product_id para conocer la cantidad de veces que se ha vendido cada producto.

Aquí tenemos una imagen de como queda nuestra database con todas las tablas vinculadas.