Sprint 4

NIVEL 1 🜟

EXTRA | Creación de la base de datos "business_db" (La tabla products se introduce en el lvl3)

Se crea la base de datos y se selecciona para usar.

```
-- Creamos la database
create database if not exists business_db default character set utf8mb4;
use business_db;
```

- Tabla "companies":
 - Se crea una tabla temporal para alojar los datos del .csv "companies", ya que vienen en una única celda como bloque de texto separado por comas.

o Se introducen los datos en la tabla

```
11 -- introducimos los datos raw de companies
12 • load data
13 infile "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\companies.csv"
14 into table companies_raw;
```

 Se crea la tabla definitiva de "companies". Datos en Varchar para soportar cualquier problema que pueda surgir de formato.

Se separan los datos del archivo .csv y se alojan en sus correspondientes columnas.
 Para esto se usa el método substring_index() y se utiliza la coma (",") como argumento separador.

```
28
           -- Separamos la raw data en las columnas de la tabla definitiva
29 •
      insert into companies (company_id, company_name, phone, email, country, website)
30
      select
          substring_index(companies_raw_data, ",", 1) as company_id,
32
         substring_index(substring_index(companies_raw_data, ",", 2), ",", -1) as company_name,
33
          substring_index(substring_index(companies_raw_data, ",", 3), ",", -1) as phone,
           substring_index(substring_index(companies_raw_data, ",", 4), ",", -1) as email,
34
          substring_index(substring_index(companies_raw_data, ",", 5), ",", -1) as country,
35
           substring_index(substring_index(companies_raw_data, ",", 6), ",", -1) as website
     from companies raw;
```

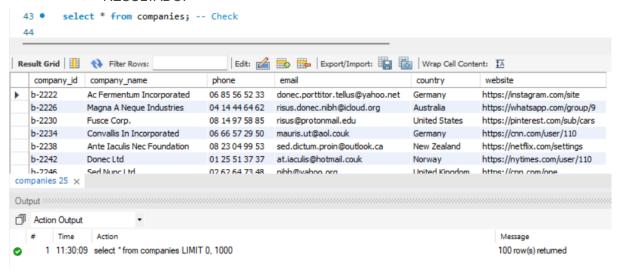
Se eliminan las cabeceras.

```
40 • delete from companies
41 where company_id = "company_id"; -- Eliminamos cabeceras
```

Se elimina la tabla temporal.

```
45 -- Eliminamos la tabla temporal
46 drop table companies_raw;
```

o RESULTADO:



- Tabla "credit cards":
 - Se crea una tabla temporal para alojar los datos "raw" del .csv "credit_cards".

Se introducen los datos en la tabla.

Se crea la tabla definitiva de "credit_cards".

```
65
         id varchar(15) primary key,
66
         user_id varchar(15),
         iban varchar(50),
67
         pan varchar(100),
68
69
         pin varchar(4),
70
         cvv varchar(4),
71
         track1 varchar(255),
72
         track2 varchar(255),
73
         expiring_date varchar(20)
74
         );
```

 Se separan los datos del archivo .csv y se alojan en sus correspondientes columnas, utilizando el método substring_index().

```
-- Separamos la raw data en las columnas de la tabla definitiva
       insert into credit_cards (id, user_id, iban, pan, pin, cvv, track1, track2, expiring_date)
78
       select
           substring_index(cc_raw_data, ",", 1) as id,
           substring_index(substring_index(cc_raw_data, ",", 2), ",", -1) as user_id,
80
           substring_index(substring_index(cc_raw_data, ",", 3), ",", -1) as iban,
81
82
          substring index(substring index(cc raw data, ",", 4), ",", -1) as pan,
83
           substring_index(substring_index(cc_raw_data, ",", 5), ",", -1) as pin,
           substring_index(substring_index(cc_raw_data, ",", 6), ",", -1) as cvv,
           substring_index(substring_index(cc_raw_data, ",", 7), ",", -1) as track1,
85
           substring_index(substring_index(cc_raw_data, ",", 8), ",", -1) as track2,
86
           substring_index(substring_index(cc_raw_data, ",", 9), ",", -1) as expiring_date
87
88
       from credit card raw;
```

o Se eliminan cabeceras.

```
90     -- Limpiamos la data
91     delete from credit_cards
92     where id = "id"; -- Eliminamos cabeceras
```

Se cambia el tipo de data en la columna "id".

```
94 • alter table credit_cards
95 modify user_id smallint; -- Cambiamos el tipo de user id para poder ordenar por el
```

- Para poder trabajar con las "expiring_dates" se han de poder transformar en formato date. Este es el proceso seguido:
 - Se crea una nueva columna con el formato deseado (date).

```
97 • alter table credit_cards
98 add column expiring_date_format date;
```

 Se transforman las fechas con el método str_to_date() y se introducen en la nueva columna.

```
99 • update credit_cards
100 set expiring_date_format = str_to_date(expiring_date, "%m/%d/%y");
```

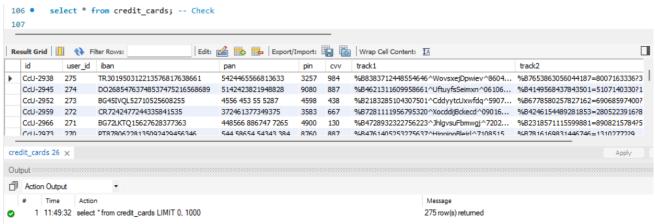
Se elimina la primera columna (la de los datos sin formatear).

```
101 • alter table credit_cards
102 drop column expiring_date;
```

■ Se renombra la nueva columna con el nombre de la primera.

Se elimina la tabla temporal.

• RESULTADO:



- Tabla "users":
 - Se crea una tabla temporal para alojar los datos "raw" del .csv "credit cards".

```
116 -- creamos una tabla "temporal" para los datos raw de users
117 • ⊖ create table if not exists users_raw (
118 users_raw_data text
119 );
```

Se introducen los datos de los archivos .csv "users usa", "users uk" y "users ca".

```
-- Introducimos los datos raw de users
        infile "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\users_usa.csv"
123
124
        into table users_raw;
125
126 • load data
127
        infile "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\users_uk.csv"
        into table users raw;
128
129
130 •
       load data
        infile "C:\\ProgramData\\MySQL\MySQL Server 8.0\\Uploads\\users ca.csv"
131
      into table users_raw;
132
```

Se crea la tabla definitiva de "users".

```
-- Creamos la tabla users
136
137 • ⊝ create table if not exists users (
138
           id varchar(50),
139
            name varchar(100),
140
           surname varchar(100),
141
           phone varchar(20),
142
            email varchar(150),
143
            birth date varchar(100),
144
            country varchar(100),
145
            city varchar(150).
146
            postal_code varchar(100),
            address varchar(255)
147
```

 Se separan los datos del archivo .csv y se alojan en sus correspondientes columnas, utilizando el método substring_index(). En este caso, las fechas vienen separadas por ",", por lo que hay que filtrarlas por otro separador ("").

```
150
             -- Separamos la raw data en las columnas de la tabla definitiva
151 •
        insert into users (id, name, surname, phone, email, birth_date, country, city, postal_code, address)
152
        select
            substring_index(users_raw_data, ",", 1) as id,
            substring_index(substring_index(users_raw_data, ",", 2), ",", -1) as name,
155
            substring_index(substring_index(users_raw_data, ",", 3), ",", -1) as surname,
156
            substring_index(substring_index(users_raw_data, ",", 4), ",", -1) as phone,
157
           substring_index(substring_index(users_raw_data, ",", 5), ",", -1) as email,
           substring_index(substring_index(users_raw_data, '"', 2), '"', -1) as birth_date,
158
           substring_index(substring_index(users_raw_data, ",", 8), ",", -1) as country,
159
            substring_index(substring_index(users_raw_data, ",", 9), ",", -1) as city,
            substring_index(substring_index(users_raw_data, ",", 10), ",", -1) as postal_code,
            substring_index(substring_index(users_raw_data, ",", 11), ",", -1) as address
        from users_raw;
```

Se eliminan las cabeceras.

```
167 • delete from users
168 where name = "name"; -- Eliminamos cabeceras
```

Se cambia el tipo de data de "id" y se setea como primary key.

```
170 • alter table users

171 modify id smallint primary key; -- Seteamos el tipo correcto para el id
```

- Para poder trabajar con las "birth_dates" se han de poder transformar en formato date. Este es el proceso seguido:
 - Se crea una nueva columna con el formato deseado (date).

```
173 • alter table users
174 add column birth_date_format date;
```

 Se transforman las fechas con el método str_to_date() y se introducen en la nueva columna.

```
175 • update users
176 set birth_date_format = str_to_date(birth_date, "%b %d, %Y");
```

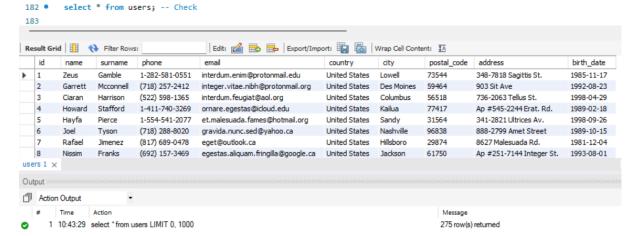
Se elimina la primera columna (la de los datos sin formatear).

```
177 • alter table users
178 drop column birth_date;
```

Se renombra la nueva columna con el nombre de la primera.

Se elimina la tabla temporal

RESULTADO:



- Tabla "transactions":
 - En este caso los datos están bien distribuidos en celdas en el propio documento .csv, por lo que no es necesario alojarlos primero en una tabla temporal. Se crea la tabla definitiva.

```
191
            -- Creamos la tabla transactions
192 • ⊖ create table if not exists transactions (
          id varchar(255),
           card_id varchar(15),
194
            business_id varchar(15),
195
            timestamp varchar(255),
            amount varchar(100),
197
            declined varchar(15),
198
            product_ids varchar(255),
            user id varchar(50),
200
            lat varchar(100),
201
202
            longitude varchar(100) null
203
```

Se introduce la data. Los archivos .csv en español separan sus celdas con ";". Como el método "load data infile", asume que por defecto están separadas con "," se debe especificar que no es así. 205 -- Insetamos la data 206 load data infile "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\transactions.csv" into table transactions 208 209 fields terminated by ";"; -- por defecto lee "," Se eliminan cabeceras. 212 • delete from transactions where id = "id"; -- Eliminamos cabeceras 213 Se setea la Primary Key. 216 alter table transactions modify id varchar(255) primary key; -- Seteamos la PK Se formatean las fechas como "timestamp". 219 • alter table transactions 220 modify timestamp timestamp; -- Formateamos fechas Se formatean los amount como "decimal" (2 decimales). 222 • alter table transactions modify amount decimal(10, 2); -- Formateamos amounts 223 Se formatea "declined" como "boolean". 225 • alter table transactions 226 modify declined boolean; -- Formateamos declined Se formatea "user_id" como "smallint" para poder ordenar por dicho campo y que coincida con la pk de la tabla users. 228 alter table transactions

```
229 modify user_id smallint; -- Formateamos user_id para que coincida con la tabla users
```

Se formatea "lat" como "float".

```
231 • alter table transactions
232 modify lat float; -- Formateamos lat
```

Se formatea "longitude" como "float".

```
234 • alter table transactions
235 modify longitude float; -- Formateamos longitude
```

Se añade la constraint con "credit_cards" (Foreign Key).

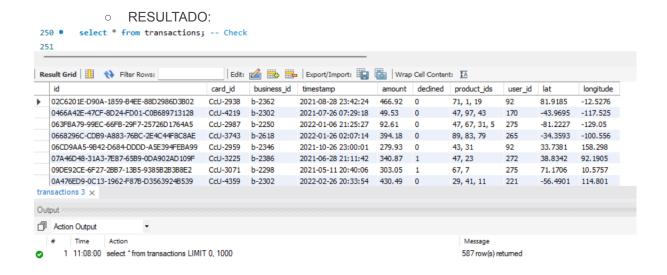
```
238 • alter table transactions
239 add constraint cards_constraint
240 foreign key (card_id) references credit_cards(id); -- Unimos con credit_cards
```

Se añade la constraint con "companies" (Foreign Key).

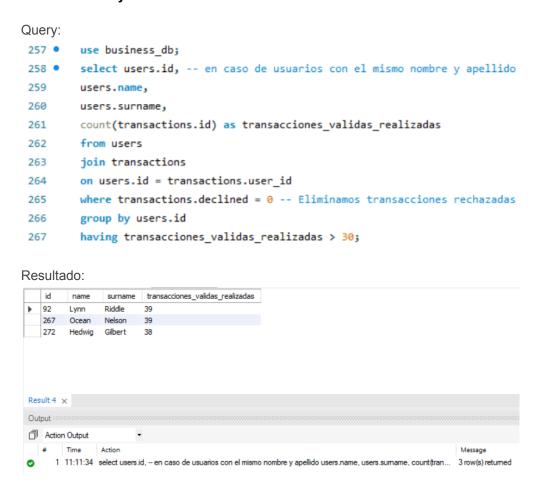
```
242 • alter table transactions
243 add constraint companies_constraint
244 foreign key (business_id) references companies(company_id); -- Unimos con companies
```

Se añade la constraint con "users" (Foreign Key).

```
246 • alter table transactions
247 add constraint users_constraint
248 foreign key (user_id) references users(id); -- Unimos con users
```



Ejercicio 1 | Realitza una subconsulta que mostri tots els usuaris amb més de 30 transaccions utilitzant almenys 2 taules.



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

Query:

```
270 • use business_db;
271 • select credit_cards.iban,
272    round(avg(transactions.amount), 2) as media_de_amount
273    from credit_cards
274    join transactions
275    on credit_cards.id = transactions.card_id
276    join companies
277    on transactions.business_id = companies.company_id
278    where companies.company_name = "Donec Ltd" and transactions.declined = 0 -- Eliminamos transacciones rechazadas
279    group by credit_cards.id;
```

Resultado:





EXTRA | 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:

• Se crea la tabla. Contiene 2 columnas: "card_id", que actúa como conector con la tabla "credit_cards"; "tarjeta_estado", que indica si la tarjeta en cuestión está operativa o no.

```
286 -- Creamos la tabla

287 • use business_db;

288 • ⊖ create table if not exists cards_status (

289 card_id varchar(15) primary key,

290 tarjeta_estado varchar(15),

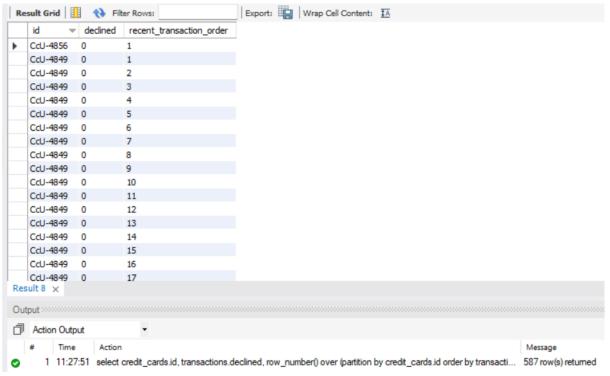
291 foreign key (card_id) references credit_cards(id)

292 );
```

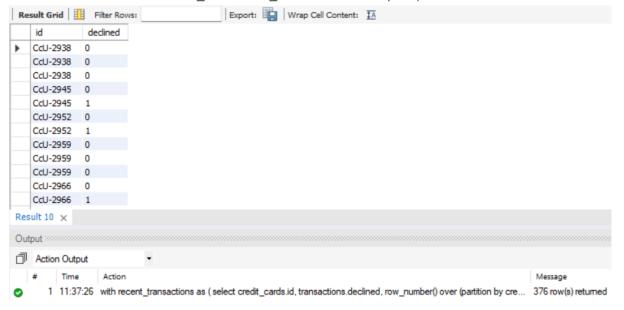
Se crean los datos y se introducen en la tabla.

```
295 • use business_db;
       insert into cards_status (card_id, tarjeta_estado)
297
       select
298
           data_estado_tarjeta.card_id,
           data_estado_tarjeta.tarjeta_estado
299
300
301 ⊝
               (with recent transactions as (
302
                   select credit_cards.id,
                   transactions.declined,
304
                   row_number() over (partition by credit_cards.id order by transactions.timestamp desc) as recent_transaction_order
305
306
                  join transactions
307
                   on credit_cards.id = transactions.card_id
308
                   ),
309
                three_latest_transactions as (
310
                   select recent_transactions.id,
311
                   recent_transactions.declined
312
                   from recent transactions
                    where recent_transactions.recent_transaction_order < 4
313
314
               valid_check as (
315
316
                   select three_latest_transactions.id,
317
                    count(case when three_latest_transactions.declined = 0 then 1 end) as transacciones_validas,
318
                    count(case when three_latest_transactions.declined = 1 then 1 end) as transacciones_no_validas
319
                   from three_latest_transactions
320
                   group by three_latest_transactions.id
321
322
                select valid_check.id as card_id,
               case when valid_check.transacciones_no_validas = 3 then "CANCELADA" else "OPERATIVA" end as tarjeta_estado
323
324
                from valid_check
325
                ) as data_estado_tarjeta;
```

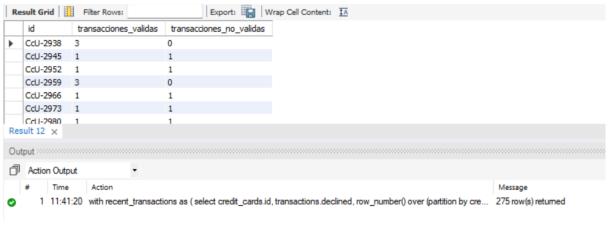
- Contenido de la subquery a introducir
 - "recent_transactions": Selecciona los id de las tarjetas, si las transacciones asociadas a ellas han sido rechazadas o no y el orden en el dichas transacciones se han realizado en el tiempo por cada tarjeta.



"three_latest transactions" selecciona, por cada id de tarjeta, los 3 primeros "recent_transaction_order" de la tabla "recent_transactions" (es decir, donde "recent_transaction_order" sea menor que 4).



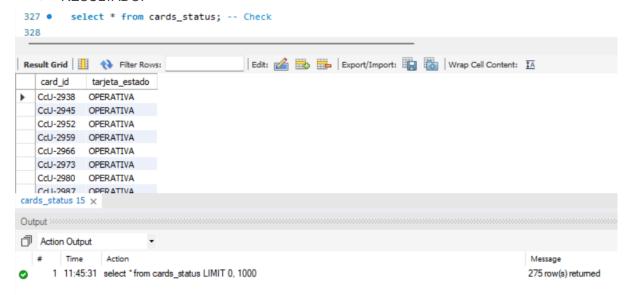
• "valid_check" agrupa cada id de tarjeta y cuenta cuántas transacciones han sido rechazadas y cuántas no (de las últimas 3).



Finalmente se establece que tarjetas están operativas y cuáles no (Aquellas cuyas transacciones NO válidas sean 3).



• RESULTADO:



Ejercicio 1 | Quantes targetes estan actives?

Query:

Resultado:





EXTRA | 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:

- Tabla "products":
 - Se crea una tabla temporal para alojar los datos "raw" del .csv "products".

```
341 -- Creamos una tabla de productos raw
342 • use business_db;
343 • ⊖ create table if not exists products_raw(
344 products_raw_data text
345 );
```

Se introducen los datos en la tabla.

```
347 -- Introducimos la data raw
348 • load data
349 infile "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\products.csv"
350 into table products_raw;
```

Se crea la tabla definitiva de "products".

```
352 -- Creamos la tabla productos
353 • use business_db;
354 • ⊖ create table if not exists products(
355 id varchar(5),
356 product_name varchar(255),
357 price varchar(20),
358 colour varchar(50),
359 weight varchar(50),
360 warehouse_id varchar(100)
361
```

Se separan los datos del archivo .csv y se alojan en sus correspondientes columnas, utilizando el método substring_index().

```
-- introducimos la data raw en la tabla definitva
364 •
       insert into products (id, product_name, price, colour, weight, warehouse_id)
365
366
         substring index(products raw data, ",", 1) as id,
367
         substring_index(substring_index(products_raw_data, ",", 2), ",", -1) as product_name,
           substring_index(substring_index(products_raw_data, ",", 3), ",", -1) as price,
           substring_index(substring_index(products_raw_data, ",", 4), ",", -1) as colour,
369
370
         substring_index(substring_index(products_raw_data, ",", 5), ",", -1) as weight,
           substring_index(substring_index(products_raw_data, ",", 6), ",", -1) as warehouse_id
     from products_raw;
372
```

Se eliminan cabeceras.

- Se elimina el símbolo de la columna "price" para poder transformar los datos y operar con ellos.
 - Se crea una nueva columna.

Se elimina el símbolo y se introducen los datos de la primera columna en la nueva.

■ Eliminamos la primera columna

• Se formatea la columna "id" y se establece como Primary Key:

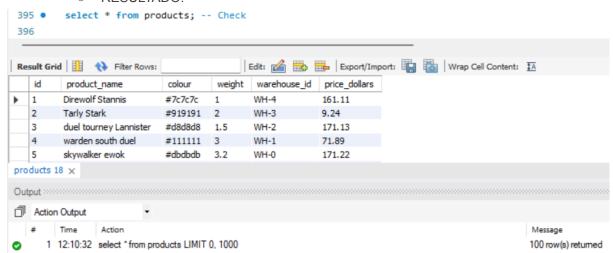
```
388 -- ID (seteamos pk)
389 • alter table products
390 modify id smallint primary key;
```

 Se formatea "weight" como float (No se formatea como decimal, porque no se sabe si en un futuro se será más preciso con los gramos, por lo que se deja abierta la posibilidad a colocar los gramos necesarios con los decimales que hagan falta):

Se elimina la tabla temporal

```
397 -- Eliminamos la tabla temporal398 • drop table products_raw;
```

o RESULTADO:



 Para poder operar con los "product_id" de la tabla transactions hay que separarlos. Para ello se crea una nueva tabla que contiene el id de la transacción y múltiples columnas con los "product_id" asociados a cada pedido. Esta tabla nunca se unirá con nada, ya que es un paso intermedio, pero necesario.

Tabla "productos comprados"

Se crea la tabla.

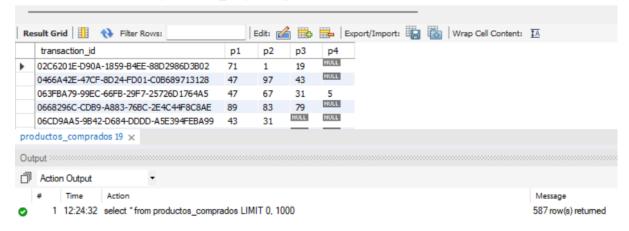
*Previamente se ha comprobado la cantidad máxima de productos en cada pedido. Si en un futuro se crean transacciones con más cantidad (y se sigue usando este método) habría que editar la tabla para añadir las columnas "px" que sean necesarias.

Se introducen los datos de productos pedidos a la nueva tabla desde "transactions". Para ello se hace una comprobación que revisa cuántos productos hay en cada transacción (hasta un máximo de 4 de momento) y los mete en su columna correspondiente. Si hay menos del máximo de productos, se setean las columnas correspondientes como "null".

```
410
          -- Insertamos la data separada de la columna transactions.product_id
411
          use business_db;
412
          insert into productos_comprados (transaction_id, p1, p2, p3, p4)
413
              select id,
              substring_index(product_ids, ",", 1) as p1,
415
416
                   length(product_ids) - length(replace(product_ids, ",", "")) + 1 >= 2,
                   substring_index(substring_index(product_ids, ",", 2), ",", -1),
417
                  null
418
419
                  ) as p2,
420
              if (
                   length(product_ids) - length(replace(product_ids, ",", "")) + 1 >= 3,
421
422
                   substring_index(substring_index(product_ids, ",", 3), ",", -1),
423
                   null
424
                  ) as p3,
              if (
425
                   length(product_ids) - length(replace(product_ids, ",", "")) + 1 >= 4,
426
427
                  substring_index(substring_index(product_ids, ",", 4), ",", -1),
428
429
                   ) as p4
430
               from transactions;
```

• RESULTADO:

432 • select * from productos_comprados; -- Check



 Utilizando como base la tabla "productos_comprados" se crea la tabla "bought_products" que será la que se utilizará realmente en el esquema final (ya que será más sencillo operar con ella y más flexible). Esta tabla consta de dos únicas columnas: "transactions_id" y "product_id", que se utilizan como Foreign Keys con sus respectivas tablas. Tabla "bought products":

Se crea la tabla

 Se introducen los datos de "productos_comprados" en el formato deseado. Se utiliza union de cada cada "id+px".

Se limpian los registros nulos.

```
451 -- Limpiamos los nulls
452 • delete from bought_products
453 where product_id is null;
```

Se une la tabla con "transactions"

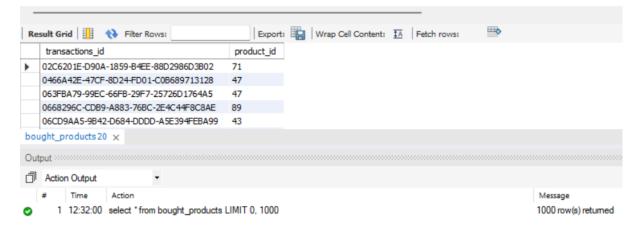
```
458 -- Union bought_products con transactions
459 • alter table bought_products
460 add constraint transaction_product_fk foreign key (transactions_id) references transactions(id);
```

Se une la tabla con "products"

```
462 -- Union con bought_products products
463 • alter table bought_products
464 add constraint bought_product_id_fk foreign key (product_id) references products(id);
```

o RESULTADO:

```
455 • select * from bought_products; -- Check
456
```



*Idealmente ahora se eliminarían la tabla "productos_comprados" y la columna transactions.product_id y se utlizaría esta nueva tabla ("bought_products") para introducir los datos de productos comprados directamente. No obstante, las voy a dejar por un hipotético caso donde sigan llegando registros de esta manera y haya que procesarlos.

Ejercicio 1 | Necessitem conèixer el nombre de vegades que s'ha venut cada producte.

Query:

```
466
       -- Requested query
467 • use business_db;
468 • select products.id,
469
     products.product_name,
470 count(bought_products.product_id) as cantidad_comprada
471 from products
472 join bought_products
      on products.id = bought_products.product_id
473
474
     join transactions
475
      on bought_products.transactions_id = transactions.id
476
     where transactions.declined = 0
477 group by products.id
478 order by cantidad_comprada desc;
```

Resultado:

