

## - NIVELL 1

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

```
1 # Creamos la tabla transactions
2 CREATE TABLE IF NOT EXISTS transactions (
3     id VARCHAR(255) PRIMARY KEY,
4     card_id VARCHAR(20),
5     business_id VARCHAR(10),
6     timestamp DATETIME,
7     amount DECIMAL(10,2),
8     declined TINYINT,
9     product_ids VARCHAR(255),
10    user_id INT,
11    lat FLOAT,
12    longitude FLOAT
13 );
14
15
16
17
18
```

Output

#	Time	Action	Message	Duration / Fetch
1	11:42:04	CREATE TABLE IF NOT EXISTS transactions ( id VARCHAR(255) PRIMARY K...	0 row(s) affected	0.032 sec

Resum del realitzat a l'exercici:

- creem les taules a mysql assegurant-nos d'ajustar el tipus de variables a les quals es troben als arxius que contenen les dades, creant també les primary keys per poder després relacionar les taules entre elles.
- afegim les dades presents als csv a les taules preparades, assegurant-nos de que el format i les separacions i salts de línia són els correctes.
- afegim les foreign keys a la taula de fets transactions i les relacionem amb les primary keys de les taules de dimensions

Després de mirar les dades de les taules presentades a l'enunciat, ens trobem amb dades referents a transaccions, productes, usuaris i empreses, totes relacionades entre sí a través d'una taula a la que anomenem taula de fets.

Aquesta taula de fets correspon a la taula transactions. Aquesta, similarment a la taula transaction d'exercicis anteriors compta amb una clau primària anomenada id. Aquest id és un codi de lletres, números i caràcters especials que entenem que és genera cada vegada que un usuari fa una transacció (compra algun producte).

Per tant, ara crearem la taula a mysql de manera que puguem carregar sense problema les dades presents al csv.

```
15 • LOAD DATA infile "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/transactions.csv"
16 INTO TABLE transactions
17 FIELDS TERMINATED BY ';'
18 ENCLOSED BY '"'
19 LINES TERMINATED BY '\n'
20 IGNORE 1 ROWS;
21
22
23
24
25
26
27
28
29
30
31
```

Output

#	Time	Action	Message	Duration / Fetch
1	10:48:47	CREATE TABLE IF NOT EXISTS transactions ( id VARCHAR(255) PRIMARY ...	0 row(s) affected	0.078 sec
2	10:49:45	LOAD DATA infile "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/transaction...	587 row(s) affected Records: 587 Deleted: 0 Skipped: 0 Warnings: 0	0.078 sec

Ara que hem creat la taula transactions a la nostra database, hem d'omplir-la amb les dades que es troben als csv. Això ho fem mitjançant la funció LOAD DATA INFILE i la ruta de l'ordinador on ho tenim guardat INTO la taula que hem creat "transactions".

Li donem les següents comandes en referència a com ha de tractar les dades al posar-les a la taula. Quan obrim l'arxiu veiem les dades separades en columnes com si fos un arxiu excel (xlsx), però es tracta d'un csv. Això és perquè en realitat, les dades estan separades per punt i coma (;). Això ho podem veure si obrim 'arxiu csv mb un editor de text com el bloc de notes.

- fields terminated by: ";" li diem que l'arxiu, tot i ser un csv (que hauria de estar separat per comes) està separat per punt i coma (;).
- enclosed by " " : aquest és un camp opcional, ja que de fet en aquest cas cada camp no està separat per (") però sql ho interpreta correctament.
- lines terminated by "\n" és la manera estàndard de fi de línia a windows. Bàsicament, després de llegir cada línia saltarà a la següent.
- ignore 1 rows: això fa que ignori la primera fila, que és la que té els noms de tots els camps. Això ho fem perquè no llegeixi la primera fila del csv com a dades per analitzar.

```
21 • CREATE TABLE IF NOT EXISTS products (  
22     id INT PRIMARY KEY,  
23     product_name VARCHAR (255),  
24     price VARCHAR (10),  
25     colour VARCHAR (10),  
26     weight FLOAT,  
27     warehouse_id VARCHAR (10)  
28 );  
29  
30 • LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/products.csv"  
31 INTO TABLE products  
32 FIELDS TERMINATED BY ","  
33 ENCLOSED BY '"'  
34 LINES TERMINATED BY "\n"  
35 IGNORE 1 ROWS;  
36  
37
```

Output

#	Time	Action	Message	Duration / Fetch
1	11:11:08	CREATE TABLE IF NOT EXISTS products (id INT PRIMARY KEY, product_n...	0 row(s) affected	0.063 sec
2	11:11:18	LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/prod...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0	0.000 sec

fem el mateix procediment per la taula products.

```
40 • CREATE TABLE IF NOT EXISTS credit_cards (  
41     id VARCHAR (20) PRIMARY KEY,  
42     user_id INT,  
43     iban VARCHAR (50),  
44     pan VARCHAR (30),  
45     pin VARCHAR(4),  
46     cvv VARCHAR(4),  
47     track1 TEXT,  
48     track2 TEXT,  
49     expiring_date VARCHAR (10)  
50 );  
51  
52 • LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/credit_cards.csv"  
53 INTO TABLE credit_cards  
54 FIELDS TERMINATED BY ","  
55 ENCLOSED BY '"'  
56 LINES TERMINATED BY "\n"  
57 IGNORE 1 ROWS;  
58
```

Output

#	Time	Action	Message	Duration / Fetch
1	11:44:42	CREATE TABLE IF NOT EXISTS credit_cards (id VARCHAR (20) PRIMARY KEY...	0 row(s) affected	0.078 sec
2	11:44:42	LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/credit...	275 row(s) affected Records: 275 Deleted: 0 Skipped: 0 Warnings: 0	0.125 sec

el mateix per la taula credit\_cards.

```
59 • CREATE TABLE IF NOT EXISTS companies (  
60     company_id VARCHAR (10) PRIMARY KEY,  
61     company_name VARCHAR (50),  
62     phone VARCHAR (20),  
63     email VARCHAR (50),  
64     country VARCHAR (20),  
65     website VARCHAR (50)  
66 );  
67 • LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/companies.csv"  
68 INTO TABLE companies  
69 FIELDS TERMINATED BY ","  
70 ENCLOSED BY ''''  
71 LINES TERMINATED BY "\n"  
72 IGNORE 1 ROWS;  
73  
74  
75
```

Output

#	Time	Action	Message	Duration / Fetch
1	11:41:47	CREATE TABLE IF NOT EXISTS companies ( company_id VARCHAR (10) PRIMA...	0 row(s) affected	0.047 sec
2	11:41:47	LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/compa...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0	0.015 sec

el mateix per la taula companies.

```
74 • CREATE TABLE IF NOT EXISTS users_all (  
75     id INT PRIMARY KEY,  
76     name VARCHAR (20),  
77     surname VARCHAR (20),  
78     phone VARCHAR (20),  
79     email VARCHAR (50),  
80     birth_date VARCHAR (20),  
81     country VARCHAR (20),  
82     city VARCHAR (50),  
83     postal_code VARCHAR (10),  
84     address VARCHAR (255)  
85 );  
86 • LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users_uk.csv"  
87 INTO TABLE users_all  
88 FIELDS TERMINATED BY ","  
89 ENCLOSED BY ''''  
90 LINES TERMINATED BY "\r\n"  
91 IGNORE 1 ROWS;  
92
```

Output

#	Time	Action	Message	Duration / Fetch
1	12:47:57	CREATE TABLE IF NOT EXISTS users_all (id INT PRIMARY KEY, name VA...	0 row(s) affected	0.063 sec
2	12:47:57	LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/user...	50 row(s) affected Records: 50 Deleted: 0 Skipped: 0 Warnings: 0	0.031 sec

Ara, en el cas de users, teniem tres csv diferents que contenen informació d'usuaris del Canadà, Estats Units i Regne Unit. Tots tres arxius contenen els mateixos camps, per tant el que hem fet des d'un principi és crear una taula que es diu users\_all que es compondrà d'una UNION dels tres arxius, de manera que tindrem tots els usuaris seguits, independentment del país del qual siguin.

Un canvi que hem realitzat en aquest cas és que hem afegit la condició lines terminated by "\r\n". En aquest cas, al tenir dades d'adreces geogràfiques algunes separades per comes i algunes no, al llegir l'arxiu csv, sql considerava algunes línies que tenien més camps dels que s'havia establert a la taula.

La condició \r torna el cursor al principi de la fila i amb \n fa el salt de línia.

```
93 • LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users_ca.csv"
94 INTO TABLE users_all
95 FIELDS TERMINATED BY ","
96 ENCLOSED BY '"'
97 LINES TERMINATED BY "\r\n"
98 IGNORE 1 ROWS;
99
100 • LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users_usa.csv"
101 INTO TABLE users_all
102 FIELDS TERMINATED BY ","
103 ENCLOSED BY '"'
104 LINES TERMINATED BY "\r\n"
105 IGNORE 1 ROWS;
```

Output

#	Time	Action	Message	Duration / Fetch
3	12:48:23	LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/us...	75 row(s) affected Records: 75 Deleted: 0 Skipped: 0 Warnings: 0	0.016 sec
4	12:48:42	LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/us...	150 row(s) affected Records: 150 Deleted: 0 Skipped: 0 Warnings: 0	0.031 sec

Com hem comentat, fem un load de les dades dels 3 arxius csv a una mateixa taula (users\_all).

```
108 • ALTER TABLE transactions
109 ADD CONSTRAINT fk_transactions_companies
110 FOREIGN KEY (business_id)
111 REFERENCES companies(company_id);
112
113 • ALTER TABLE transactions
114 ADD CONSTRAINT fk_transactions_credit_cards
115 FOREIGN KEY (card_id)
116 REFERENCES credit_cards(id);
117
118 • ALTER TABLE transactions
119 ADD CONSTRAINT fk_transactions_users
120 FOREIGN KEY (user_id)
121 REFERENCES users_all(id);
122
123 • ALTER TABLE transactions
124 ADD CONSTRAINT fk_transactions_products
125 FOREIGN KEY (product_ids)
126 REFERENCES products(id);
```

Output

#	Time	Action	Message	Duration / Fetch
3	11:47:59	ALTER TABLE transactions ADD CONSTRAINT fk_transactions_users FOREIGN...	587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0	0.375 sec

Output

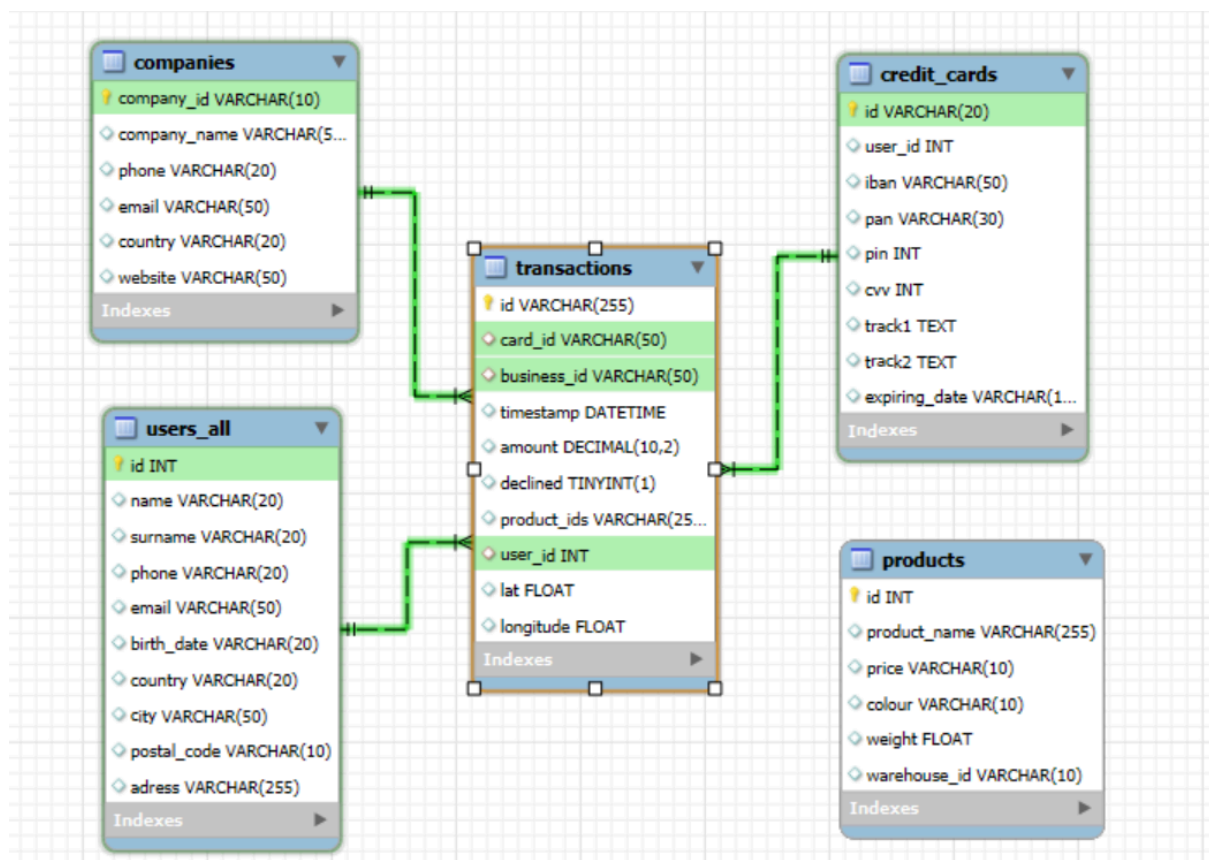
#	Time	Action	Message	Duration / Fetch
3	11:47:59	ALTER TABLE transactions ADD CONSTRAINT fk_transactions_users FOREIGN...	587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0	0.375 sec
4	11:47:59	ALTER TABLE transactions ADD CONSTRAINT fk_transactions_products FORE...	Error Code: 3780. Referencing column 'product_ids' and referenced column 'id' in ...	0.000 sec

Ara que tenim totes les taules creades i plenes, podem establir relacions entre elles. Com hem comentat al principi, la taula de fets serà la taula central, que anirà omplint-se cada vegada que es fa una transacció.

D'aquesta manera, aquesta taula es relacionarà amb les taules de dimensions a través de foreign keys.

- 1) Entre transactions i companies existeix una relació amb la primary key de companies (company\_id). Aquesta es relaciona amb el camp business\_id de transactions.
- 2) Entre transactions i credit\_cards existeix una relació amb la primary key de credit\_cards(id). Aquesta es relaciona amb el camp card\_id de transactions.
- 3) Entre transactions i users\_all existeix una relació amb la primary key de users\_all (id). Aquesta es relaciona amb el camp user\_id de transactions.
- 4) Entre transactions i products hauria d'existir una relació amb la primary key de products (product\_id). Aquesta hauria de relacionar-se amb el camp product\_ids de transactions. El problema és que no es pot establir aquesta relació perquè les dades no corresponen en format. Les dades de product\_ids de transactions inclouen més d'un product\_id separat per comes i estan en format VARCHAR. És a dir, per una transacció podem tenir més d'un product\_id comprat. Això suposa un problema quan volem relacionar-ho amb la taula product amb el camp id. El tipus de dada que trobem al camp id és un valor INT únic. Més endavant veurem com resoldre aquest problema.

Per tant, una vegada establertes les relacions entre les taules ens queda aquest esquema:



## - EXERCICI 1 - NIVEL 1

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

```
128 # Exercici 1
129 • SELECT ua.id, ua.name, ua.surname
130 FROM users_all ua
131 JOIN (
132     SELECT user_id
133     FROM transactions
134     GROUP BY user_id
135     HAVING COUNT(*) > 30
136 ) AS usuarios_moltes_transaccions
137 ON ua.id = usuarios_moltes_transaccions.user_id;
138
139
140
141
```



id	name	surname
92	Lynn	Riddle
275	Kenyon	Hartman
272	Hedwig	Gilbert
267	Ocean	Nelson

Result 7 x

Output

Action Output

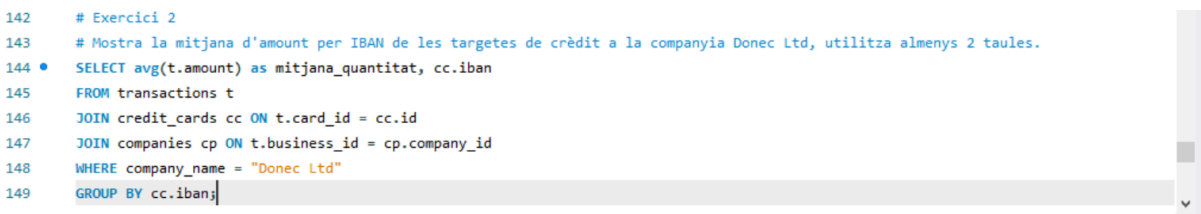
#	Time	Action	Message	Duration / Fetch
1	18:20:41	SELECT ua.id, ua.name, ua.surname FROM users_all ua JOIN ( SELECT user_id FROM transactions GROUP BY user_id HAVING CO... 4 row(s) returned		0.016 sec / 0.000 sec

En aquest exercici, seleccionarem els camps id, noms i cognoms de la llista users\_all. mitjançant SELECT. Després, dins la subconsulta, selecciona els user\_id de la taula transactions, que hagin realitzat un nombre superior a 30 transaccions, agrupant-los per user\_id, filtrant per HAVING COUNT(\*) > 30. La consulta principal farà una JOIN de la taula users\_all amb el resultat de la subconsulta mitjançant ua.id = user\_id

## - EXERCICI 2

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

```
142 # Exercici 2
143 # Mostra la mitjana d'import per IBAN de les targetes de crèdit a la companyia Donec Ltd, utilitza almenys 2 taules.
144 • SELECT avg(t.amount) as mitjana_quantitat, cc.iban
145 FROM transactions t
146 JOIN credit_cards cc ON t.card_id = cc.id
147 JOIN companies cp ON t.business_id = cp.company_id
148 WHERE company_name = "Donec Ltd"
149 GROUP BY cc.iban;
```



mitjana_quantitat	iban
203.715000	PT87806228135092429456346

Result 37 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	12:05:47	SELECT avg(t.amount) as mitjana_quantitat, cc.iban FROM transactions t JOIN ... 1 row(s) returned		0.016 sec / 0.000 sec

Per aquest exercici, extraïem la quantitat mitjana d'amount mitjançant la funció d'agregació *average* (arrodonint a dos decimals per fer-ho més llegible) i agafem també el camp *iban* de *credit\_cards*. Per agafar els camps d'aquestes dues taules necessitem fer una join per la clau forana de *transactions* (*card\_id*) amb *credit\_cards*. Per poder filtrar pel nom de l'empresa que ens diu l'enunciat, haurem de fer també una join amb la taula *companies*. D'aquesta manera tindrem accés al camp *company\_name* i filtrarem per "Donec Ltd". Finalment, agruparem pel camp *iban* per tenir la quantitat mitjana per aquest.

## NIVELL 2

Crea una nova taula que reflecteix l'estat de les targetes de crèdit basat en si les últimes tres transaccions van ser declinados y genera la següent consulta:

```
153 • CREATE TABLE estado_tarjetas AS
154 SELECT
155     card_id,
156     CASE
157         WHEN SUM(CASE WHEN declined = 1 THEN 1 ELSE 0 END) = 3 THEN 0
158         ELSE 1
159     END AS tarjeta_activa
160 FROM (
161     SELECT card_id, declined,
162            ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS orden
163     FROM transactions) AS ultimas_transacciones
164 WHERE orden <= 3
165 GROUP BY card_id;
```

Output

#	Time	Action	Message	Duration / Fetch
1	10:15:30	CREATE TABLE estado_tarjetas AS SELECT card_id, CASE WHEN SUM(CA...	275 row(s) affected Records: 275 Duplicates: 0 Warnings: 0	0.219 sec

Primer de tot, crearem la taula la qual anomenarem *estado\_tarjetas*. Aquesta taula tindrà solament dos camps. Un per llistar totes les targetes diferents que tenen els clients anomenat *card\_id* i l'altre per saber si la tarjeta es troba activa o no. Aquest últim s'anomena *tarjeta\_activa* i tindrà un valor binari (1 i 0) segons si la targeta es considera activa o no.

El que vol determinar aquesta taula és si la targeta es troba activa en funció de les 3 últimes transaccions realitzades.

Per fer-ho, es parteix d'una subconsulta que utilitza la funció *PARTITION BY card\_id* per agrupar les transaccions segons la *card\_id* (tractem les transaccions de cada targeta com un conjunt independent).

La funció *ROW\_NUMBER()*, assigna un número d'ordre consecutiu a cada transacció de cada targeta. Per poder-les ordenar, tenim la funció *ORDER BY timestamp desc* fent que l'ordenació per timestamp funcioni agrupant cada transacció realitzada per targeta de més recent a més antiga. Aquesta numeració, llavors, ordena les transaccions de manera descendent, de manera que la transacció més recent se li assigna el valor 1 per cada

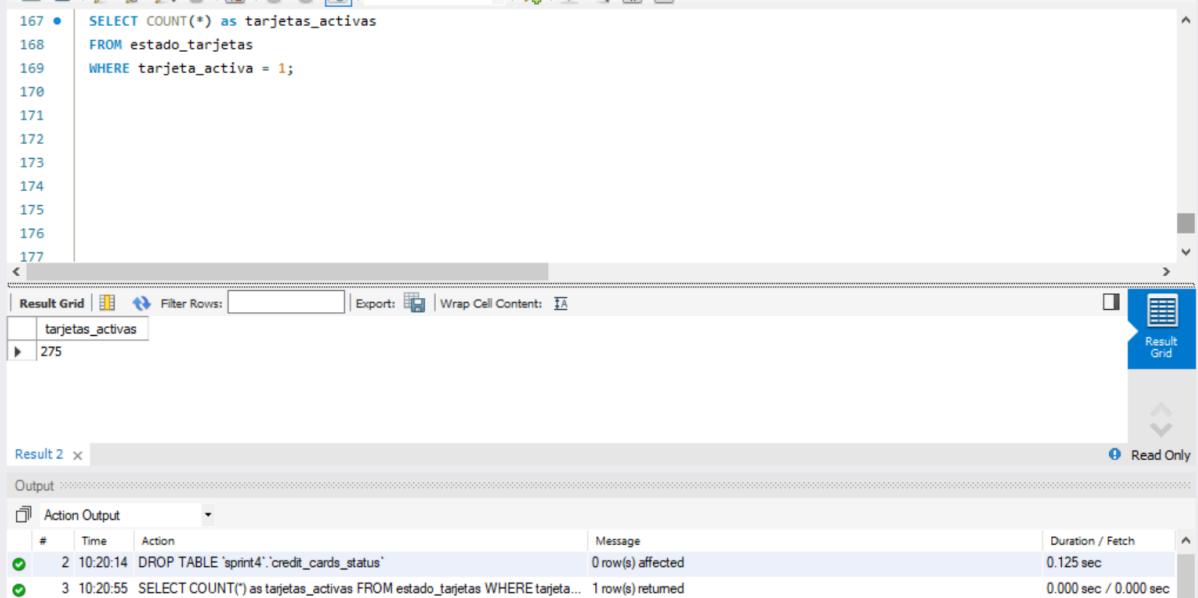


targeta. Per últim, la condició WHERE orden <= 3 agafa només les 3 últimes transaccions, agrupant per *card\_id*.

Una vegada tenim aquest conjunt filtrat, s'executa la query de fora: farem un SELECT del *card\_id* de la taula transactions i dins d'aquest SELECT li posarem una condició CASE WHEN. Aquesta condició fa una funció d'agregació SUM per comptar el nombre de transaccions que han estat declinades (declined = 1). Si les últimes 3 transaccions han estat declinades (suma = 3), aquesta es considera com targeta no activa (tarjeta\_activa = 0). La funció ELSE 1 significa que en qualsevol altre cas, que no sigui l'anterior, consideri la targeta com a activa (tarjeta\_activa = 1).

## - EXERCICI 1

Quantes targetes están actives?



The screenshot shows a SQL IDE interface. The top pane contains a SQL query:

```
167 SELECT COUNT(*) as tarjetas_activas
168 FROM estado_tarjetas
169 WHERE tarjeta_activa = 1;
170
171
172
173
174
175
176
177
```

The bottom pane shows the results of the query. The 'Result Grid' displays a single row with the value 275 for the column 'tarjetas\_activas'. The 'Output' pane shows the execution log:

#	Time	Action	Message	Duration / Fetch
2	10:20:14	DROP TABLE 'sprint4'.credit_cards_status'	0 row(s) affected	0.125 sec
3	10:20:55	SELECT COUNT(*) as tarjetas_activas FROM estado_tarjetas WHERE tarjeta...	1 row(s) returned	0.000 sec / 0.000 sec

Una vegada hem creat la taula *tarjetas\_activas*, si volem conèixer quantes són actives, simplement podem comptar el nombre de files a la taula, filtrant per *tarjeta\_activa* = 1 (condició explicada a l'exercici anterior). Ens surten 275 files, que corresponen al nombre total de targetes existents. Per tant, totes les targetes existents es consideren actives.

### - 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:

```
170 CREATE TABLE transaction_product (  
171     transaction_id VARCHAR(255),  
172     product_id INT,  
173     PRIMARY KEY (transaction_id, product_id),  
174     FOREIGN KEY (transaction_id) REFERENCES transactions(id),  
175     FOREIGN KEY (product_id) REFERENCES products(id)  
176 ) ;  
177  
178  
179  
180  
181  
182  
183  
184  
185
```

Output

#	Time	Action	Message	Duration / Fetch
1	11:06:34	CREATE TABLE transaction_product ( transaction_id VARCHAR(255), prod...	0 row(s) affected	0.079 sec

Com hem observat, a l'hora de crear les taules i establir les relacions entre elles, la taula transactions i products tenen problemes per relacionar-se. La taula transactions inclou files en les que trobem més d'un *product\_id* separat per comes, mentre que la taula products és una taula de dimensions en la que només tenim un *product\_id* per fila. Al tractar-se de diferents tipus de dades no les podem relacionar correctament.

El que farem en aquest exercici és fer una taula intermitja, que pugui separar els *product\_id* d'una mateixa transacció, de manera que ens quedi un *product\_id* per fila, assignat al id de la transacció que li correspongui. D'això se'n diu normalitzar les nostres dades.

Per fer-ho, primerament hem de crear la taula. Aquesta taula només tindrà dos camps: *transaction\_id*, farà referència al id de les transaccions realitzades que trobem a la taula transactions i, el segon, *product\_id*, referent al *product\_id* comprat per cada transacció. En aquest cas, ambdós camps faran de primary key, ja que relacionaran dues taules diferents cadascun.

Tot seguit, establim les relacions. Primer, una foreign key a la taula transactions que relacionarà l'id de les transaccions amb l'id de la nostra taula. S'ha d'esmentar que aquesta relació serà de molts a 1 en direcció a la taula transactions, perquè tindrem valors de *transaction\_id* repetits a la nostra nova taula.

Per altra banda, establim una foreign key a la taula products, que relacionarà l'id dels productes amb l'id dels productes comprats de la nostra taula. Aquesta relació també serà de molts a 1 en direcció a products.

Un cop creada la taula, només ens queda omplir-la amb les dades que li corresponen.

## - EXERCICI 1

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

L'objectiu principal que busca aquest codi és normalitzar els valors de la columna `product_ids` de la taula `transactions` (que es trobaven separats per comes) i inserir-los en la taula que hem creat (que serveix com taula intermèdia).

Els inserirem de manera que tindrem una fila per cada combinació de `transaction_id` - `product_id`, és a dir tindrem tantes files repetides de `transaction_id` com `product_ids` tingui aquella transacció.

```
180 • INSERT INTO transaction_product (transaction_id, product_id)
181     SELECT transaction_id, CAST(product_id AS UNSIGNED)
182 FROM (
183     WITH RECURSIVE numbers AS (
184         SELECT 0 AS n
185         UNION ALL
186         SELECT n + 1 FROM numbers WHERE n < 20
187     ),
188     cortar AS (
189         SELECT
190             t.id AS transaction_id,
191             TRIM(SUBSTRING_INDEX(SUBSTRING_INDEX(t.product_ids, ',', n + 1), ',', -1)) AS product_id
192         FROM transactions t
193         JOIN numbers
194             ON n <= CHAR_LENGTH(t.product_ids) - CHAR_LENGTH(REPLACE(t.product_ids, ',', ''))
195     )
196     SELECT * FROM cortar
197 ) AS resultado;
```

Output

#	Time	Action	Message	Duration / Fetch
1	12:14:10	INSERT INTO transaction_product (transaction_id, product_id) SELECT transacti...	1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0	0.094 sec

Primerament, li demanem que ens insereixi a la taula que hem creat anomenada `transaction_product` en els camps `transaction_id` i `product_id` respectivament, la següent informació:

- 1) Farem una subconsulta a través de FROM en el que generarem una taula CTE recursiva a la que anomenem `numbers`. Aquesta taula ens servirà per generar una seqüència de números del 0 fins al 20. Aquest límit s'estableix suposant que cap transacció conté més de 20 productes. Aquesta seqüència serveix per iterar sobre les posicions dels valors del camp `product_ids`, separat per comes per a després "trossejar" aquests en valors individuals.
- 2) La CTE "cortar" utilitza aquesta seqüència generada per aplicar iterativament la funció `SUBSTRTING_INDEX` (que mitjançant la join que fem més endavant), ens permet extreure cada identificador individual del producte d'una cadena separada per comes. Aquesta funció té dues utilitats: extreu i elimina els espais en blanc entre els productes separats per comes i la utilitza de manera iterativa per poder extreure cada valor de `product_id` abans separats per comes.
- 3) La JOIN amb la CTE `numbers` es fa mitjançant la següent condició: la part `CHAR_LENGTH(t.product_ids)` compta quants caràcters té la cadena de text

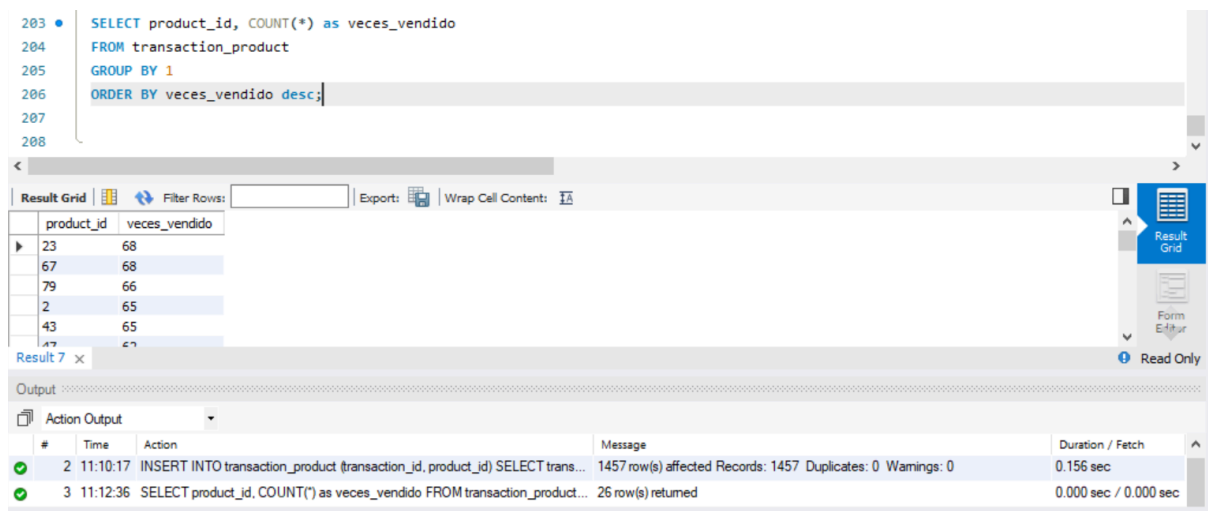
product\_ids. La part REPLACE(t.product\_ids, ',', ''), treu les comes i compta els caràcters restants. La diferència entre els dos operands ens diu quantes comes hi ha a la cadena de text i per tant, quants productes tenim.

- 4) La query de fora assigna la funció CAST(product\_id AS UNSIGNED) que ens permet convertir el que abans llegia com una cadena de text a un nombre enter positiu i insereix els valors obtinguts a la taula transaction\_product.

Observem que tenim un resultat de 1457 files, és a dir, s'han comprat un total de 1457 productes.

## - EXERCICI 1

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



The screenshot shows a database interface with a SQL query editor and a results grid. The query is:

```
SELECT product_id, COUNT(*) as veces_vendido
FROM transaction_product
GROUP BY 1
ORDER BY veces_vendido desc;
```

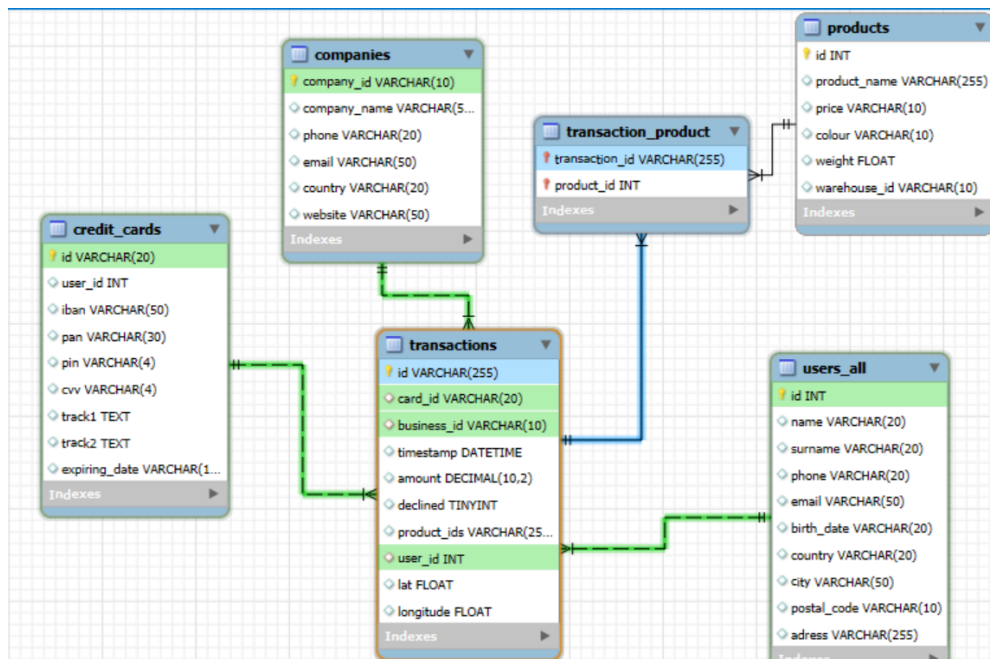
The results grid displays the following data:

product_id	veces_vendido
23	68
67	68
79	66
2	65
43	65
17	65

The output section shows the following actions:

#	Time	Action	Message	Duration / Fetch
2	11:10:17	INSERT INTO transaction_product (transaction_id, product_id) SELECT trans...	1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0	0.156 sec
3	11:12:36	SELECT product_id, COUNT(*) as veces_vendido FROM transaction_product...	26 row(s) returned	0.000 sec / 0.000 sec

Per conèixer quantes vegades s'ha venut cada producte hem de seleccionar l'id del producte i fer una funció d'agregació count per saber quantes files de producte tenim a la nova taula. Ho agruparem per *product\_id* i ordenarem per vegades venut.



L'esquema resultant final queda de la següent manera:

La taula `transaction_product` queda com a taula intermèdia al nostre esquema. Aquesta taula solucionarà el nostre problema inicial, al poder establir una relació entre `transactions` i `products`.

Aquesta taula, relaciona de *n* a 1 a través de `transaction_id` amb la taula `transactions` i de *n* a 1 mitjançant `product_id` amb la taula `products`. Per tant, tenim que les dues primary keys de la taula també fan de foreign keys.

Aquesta *bridge table* ens ha permès representar, de manera normalitzada, la relació de molts a molts que teníem entre les taules `transactions` i `products` (una transacció podia tenir molts productes i, de la mateixa manera, un producte podia estar en múltiples transaccions).