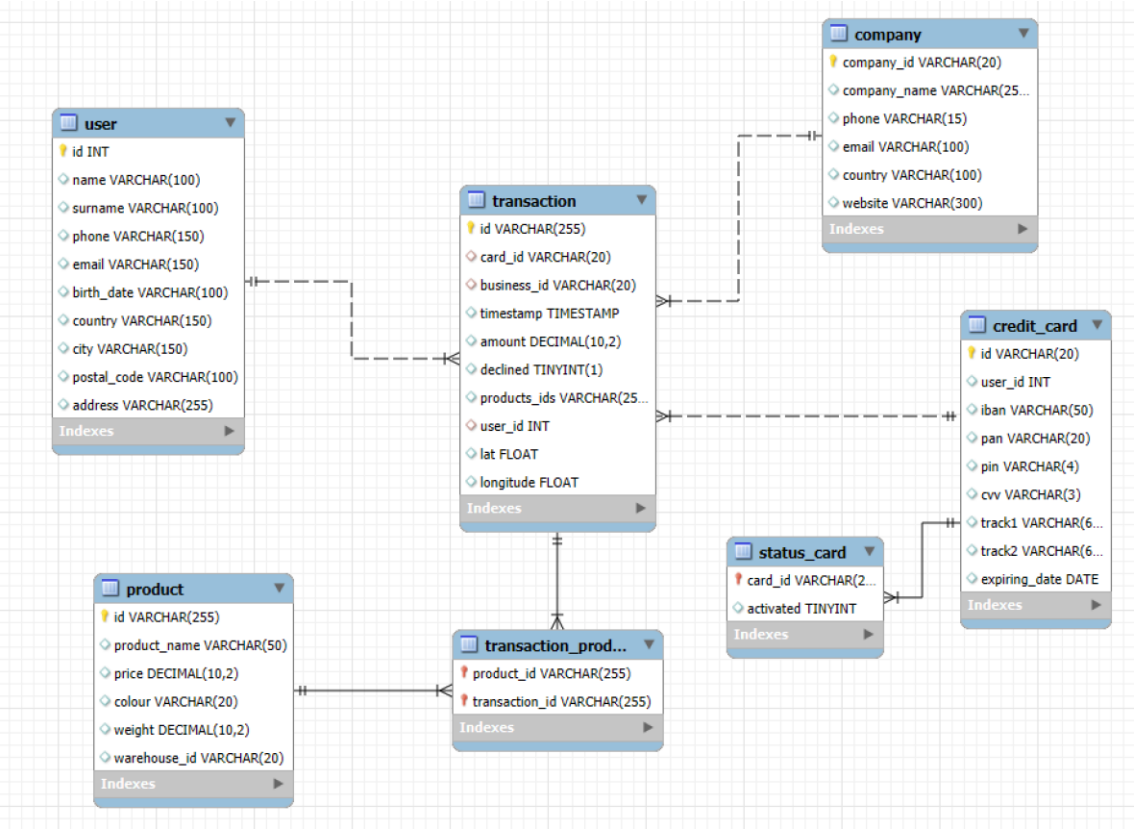
Partint d'alguns arxius CSV dissenyaràs i crearàs la teva base de dades.

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:



En primer lugar, tenemos 7 archivos CSV a partir de los cuales diseñaremos y construiremos la base de datos (transactionsCSV). De estos 7 archivos, haremos 5 tablas (COMPANY, USER, PRODUCT, CREDIT_CARD, TRANSACTION) ya que los CSV users_ca, users_usa, users_uk todos corresponden a la misma tabla y se respetan los id para que sean consecutivos y no se solapan.

Creamos primero la tabla company, seguido de user, credit_card, product y por último la tabla transaction, que tiene como Foreign Keys las Primary Key de card_id, business_id y user_id.

Se ha seguido el esquema estrella, es decir, transaction está en el centro y de ella salen las relaciones con las demás tablas. Todas son 1 to Many (1:N) EXCEPTO la relación de producto con transaction que es (N:M) y se ha generado la tabla intermedia transaction_product para poder realizar la relación. Esta tabla tiene como Primary Key la tupla con la columna correspondiente a product_id y transaction_id.

A la hora de hacer el fetching de los datos del CSV primero leemos casi todos los campos como VARCHAR y posteriormente hacemos los cambios necesarios. En nuestro caso a la columna

Price le quitamos el símbolo del $ y lo cambiamos a decimal. Para expiring_date y birth_date las tratamos como DATE. Para realizar los insert into de la tabla transaction_product, utilizamos la función FIND_IN_SET para iterar sobre los distintos products_ids y asignar a cada producto_id con su transaction_id y REPLACE para quitar el espacio entre la coma y el id en la lista de ids de productos.

```sql
-- SPRINT 4
CREATE DATABASE IF NOT EXISTS transactionsCSV;
USE transactionsCSV;

-- id,user_id,iban,pan,pin,cvv,track1,track2,expiring_date
CREATE TABLE IF NOT EXISTS credit_card (
        id VARCHAR(20) PRIMARY KEY,
        user_id INT,
        iban VARCHAR(50),
        pan VARCHAR(20),
        pin VARCHAR(4),
        cvv VARCHAR(3),
        track1 VARCHAR(60),
        track2 VARCHAR(60),
        expiring_date VARCHAR(20)
    );

    -- company_id,company_name,phone,email,country,website
    CREATE TABLE IF NOT EXISTS company(
        company_id VARCHAR(20) PRIMARY KEY,
        company_name VARCHAR(255),
        phone VARCHAR(15),
        email VARCHAR(100),
        country VARCHAR(100),
        website VARCHAR(300)
    );

```

```sql
-- id,name,surname,phone,email,birth_date,country,city,postal_code,address
CREATE TABLE IF NOT EXISTS user (
        id INT PRIMARY KEY,
        name VARCHAR(100),
        surname VARCHAR(100),
        phone VARCHAR(150),
        email VARCHAR(150),
        birth_date VARCHAR(100),
        country VARCHAR(150),
        city VARCHAR(150),
        postal_code VARCHAR(100),
        address VARCHAR(255)
    );

    -- id,product_name,price,colour,weight,warehouse_id
    CREATE TABLE IF NOT EXISTS product (
        id INT AUTO_INCREMENT PRIMARY KEY,
        product_name VARCHAR(50),
        price VARCHAR(30),
        colour VARCHAR(20),
        weight decimal(10,2),
        warehouse_id VARCHAR(20)
    );

-- id;card_id;business_id;timestamp;amount;declined;product_ids;user_id;lat;longitude
    CREATE TABLE IF NOT EXISTS transaction (
        id VARCHAR(255) PRIMARY KEY,
        card_id VARCHAR(20),
        business_id VARCHAR(20),
        timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        amount decimal(10,2),
        declined tinyint,
        products_ids VARCHAR(255),
        user_id INT,
        lat float,
        longitude float,
        FOREIGN KEY (card_id) REFERENCES credit_card(id),
        FOREIGN KEY (business_id) REFERENCES company(company_id),
        FOREIGN KEY (user_id) REFERENCES user(id)
    );
```

Action Output

#	Time	Action	Message
1	13:57:02	CREATE DATABASE IF NOT EXISTS transactionsCSV	1 row(s) affected
2	13:57:02	USE transactionsCSV	0 row(s) affected
3	13:57:02	CREATE TABLE IF NOT EXISTS credit_card (id VARCHAR(20) PRIMARY KEY, user_id INT, ...	0 row(s) affected
4	13:57:02	CREATE TABLE IF NOT EXISTS company(company_id VARCHAR(20) PRIMARY KEY, com...	0 row(s) affected
5	13:57:02	CREATE TABLE IF NOT EXISTS user (id INT PRIMARY KEY, name VARCHAR(100), su...	0 row(s) affected
6	13:57:02	CREATE TABLE IF NOT EXISTS product (id INT AUTO_INCREMENT PRIMARY KEY, produ...	0 row(s) affected
7	13:57:02	CREATE TABLE IF NOT EXISTS transaction (id VARCHAR(255) PRIMARY KEY, card_id VA...	0 row(s) affected

```
70  SET GLOBAL local_infile=1;
71
72  LOAD DATA LOCAL INFILE 'C:/Users/juanc/Documents/SPRINT4/companies.csv' INTO TABLE transactionscsv.company
73  FIELDS TERMINATED BY ','
74  ENCLOSED BY ''
75  LINES TERMINATED BY '\r\n'
76  IGNORE 1 LINES;
77
78  LOAD DATA LOCAL INFILE 'C:/Users/juanc/Documents/SPRINT4/credit_cards.csv' INTO TABLE transactionscsv.credit_card
79  FIELDS TERMINATED BY ','
80  ENCLOSED BY '"'
81  LINES TERMINATED BY '\n'
82  IGNORE 1 LINES;
83
84  LOAD DATA LOCAL INFILE 'C:/Users/juanc/Documents/SPRINT4/users_ca.csv' INTO TABLE transactionscsv.user
85  FIELDS TERMINATED BY ','
86  ENCLOSED BY '"'
87  LINES TERMINATED BY '\r\n'
88  IGNORE 1 LINES;
89
90  LOAD DATA LOCAL INFILE 'C:/Users/juanc/Documents/SPRINT4/users_uk.csv' INTO TABLE transactionscsv.user
91  FIELDS TERMINATED BY ','
92  ENCLOSED BY '"'
93  LINES TERMINATED BY '\r\n'
94  IGNORE 1 LINES;
95
96  LOAD DATA LOCAL INFILE 'C:/Users/juanc/Documents/SPRINT4/users_usa.csv' INTO TABLE transactionscsv.user
97  FIELDS TERMINATED BY ','
98  ENCLOSED BY '"'
99  LINES TERMINATED BY '\r\n'
100 IGNORE 1 LINES;
101
102 LOAD DATA LOCAL INFILE 'C:/Users/juanc/Documents/SPRINT4/products.csv' INTO TABLE transactionscsv.product
103 FIELDS TERMINATED BY ','
104 ENCLOSED BY ''
105 LINES TERMINATED BY '\n'
106 IGNORE 1 LINES;
107
108 LOAD DATA LOCAL INFILE 'C:/Users/juanc/Documents/SPRINT4/transactions.csv' INTO TABLE transactionscsv.transaction
109 FIELDS TERMINATED BY ';'
110 ENCLOSED BY ''
111 LINES TERMINATED BY '\n'
112 IGNORE 1 LINES;
113
114 SET GLOBAL local_infile=0;
```

#	Time	Action	Message
1	14:02:07	SET GLOBAL local_infile=1	0 row(s) affected
2	14:02:07	LOAD DATA LOCAL INFILE 'C:/Users/juanc/Documents/SPRINT4/companies.csv' INTO TABLE trans...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0
3	14:02:07	LOAD DATA LOCAL INFILE 'C:/Users/juanc/Documents/SPRINT4/credit_cards.csv' INTO TABLE tran...	275 row(s) affected Records: 275 Deleted: 0 Skipped: 0 Warnings: 0
4	14:02:07	LOAD DATA LOCAL INFILE 'C:/Users/juanc/Documents/SPRINT4/users_ca.csv' INTO TABLE transac...	75 row(s) affected Records: 75 Deleted: 0 Skipped: 0 Warnings: 0
5	14:02:07	LOAD DATA LOCAL INFILE 'C:/Users/juanc/Documents/SPRINT4/users_uk.csv' INTO TABLE transac...	50 row(s) affected Records: 50 Deleted: 0 Skipped: 0 Warnings: 0
6	14:02:07	LOAD DATA LOCAL INFILE 'C:/Users/juanc/Documents/SPRINT4/users_usa.csv' INTO TABLE transa...	150 row(s) affected Records: 150 Deleted: 0 Skipped: 0 Warnings: 0
7	14:02:07	LOAD DATA LOCAL INFILE 'C:/Users/juanc/Documents/SPRINT4/products.csv' INTO TABLE transac...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0
8	14:02:07	LOAD DATA LOCAL INFILE 'C:/Users/juanc/Documents/SPRINT4/transactions.csv' INTO TABLE tran...	587 row(s) affected Records: 587 Deleted: 0 Skipped: 0 Warnings: 0
9	14:02:07	SET GLOBAL local_infile=0	0 row(s) affected

```sql
-- Eliminamos el $ que se encuentra en la primera posición del string de price
SET SQL_SAFE_UPDATES = 0;
UPDATE product
SET price= SUBSTR(price,2);

-- Cambiamos el tipo de la columna price de varchar a decimal
ALTER TABLE product
MODIFY COLUMN price decimal(10,2);
-- cerrar safe updates

-- Cambiamos la fecha de VARCHAR a DATE modificando el tipo de la columna birth_date
UPDATE user
SET birth_date = STR_TO_DATE(birth_date, '%b %d, %Y');

-- Cambiamos la fecha de caducidad de VARCHAR a DATE modificando el tipo de la columna birth_date
UPDATE credit_card
SET expiring_date = STR_TO_DATE(expiring_date, '%m/%d/%y');

ALTER TABLE credit_card MODIFY COLUMN expiring_date DATE;
```

Output

Action Output

#	Time	Action	Message
1	14:03:23	SET SQL_SAFE_UPDATES = 0	0 row(s) affected
2	14:03:23	UPDATE product SET price= SUBSTR(price,2)	100 row(s) affected Rows matched: 100 Changed: 100 Warnings: 0
3	14:03:23	ALTER TABLE product MODIFY COLUMN price decimal(10,2)	100 row(s) affected Records: 100 Duplicates: 0 Warnings: 0
4	14:03:24	UPDATE user SET birth_date = STR_TO_DATE(birth_date, '%b %d, %Y')	275 row(s) affected Rows matched: 275 Changed: 275 Warnings: 0
5	14:03:24	UPDATE credit_card SET expiring_date = STR_TO_DATE(expiring_date, '%m/%d/%y')	275 row(s) affected Rows matched: 275 Changed: 275 Warnings: 0
6	14:03:24	ALTER TABLE credit_card MODIFY COLUMN expiring_date DATE	275 row(s) affected Records: 275 Duplicates: 0 Warnings: 0

```sql
CREATE TABLE IF NOT EXISTS transaction_product (
    product_id INT NOT NULL,
    transaction_id VARCHAR(255) NOT NULL,
    PRIMARY KEY (product_id, transaction_id),
    FOREIGN KEY (product_id) REFERENCES product(id),
    FOREIGN KEY (transaction_id) REFERENCES transaction(id)
);

-- EXPLICACION teniendo en cuenta no declined, utilizamos FIND_IN_SET como condicion para asociar cada product_id con su transaction_id
INSERT INTO transaction_product (product_id, transaction_id)
SELECT product.id as product_id, transaction.id as transaction_id
FROM product
JOIN transaction on FIND_IN_SET(product.id, REPLACE (products_ids, " ", ""))>0
WHERE declined=0;
```

Output

Action Output

#	Time	Action	Message
1	14:06:32	CREATE TABLE IF NOT EXISTS transaction_product (product_id INT NOT NULL, transaction_id VARC...	0 row(s) affected
2	14:06:32	INSERT INTO transaction_product (product_id, transaction_id) SELECT product.id as product_id, transa...	1236 row(s) affected Records: 1236 Duplicates: 0 Warnings: 0

Cuando realizamos los insert into de los transaction_product (tabla intermedia) solo añadimos aquellos que hayan sido no declinados con el filtro: WHERE declined=0. En este caso se añaden 1236 filas.

- Exercici 1

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

```
152     -- Ejercicio 1 nivel 1 contando transacciones declined como no declined
153 •   SELECT user.id,name,surname,phone,email,birth_date,country,city,postal_code,address, count(user.id) as contador_transacciones
154     FROM user
155     JOIN transaction ON user_id=user.id
156     GROUP BY user.id, name, surname, phone, email, birth_date, country,city, postal_code, address
157     HAVING contador_transacciones > 30;
158
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	name	surname	phone	email	birth_date	country	city	postal_code	address	contador_transacciones
92	Lynn	Riddle	1-387-885-4057	vitae.aliquet@outlook.edu	1984-09-21	United States	Bozeman	61871	P.O. Box 712, 7907 Est St.	39
267	Ocean	Nelson	079-481-2745	aenean@yahoo.com	1991-12-26	Canada	Charlottetown	85X 3P4	Ap #732-8357 Pede, Rd.	52
272	Hedwig	Gilbert	064-204-8788	sem.eget@icloud.edu	1991-04-16	Canada	Tuktoyaktuk	Q4C 3G7	P.O. Box 496, 5145 Sapien Road	76
275	Kenyon	Hartman	082-871-7248	convallis.ante.lectus@yahoo.com	1982-08-03	Canada	Richmond	R8H 2K2	8564 Facilisi. St.	48

Result 47 | Read

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	12:08:54	SELECT user.id,name,surname,phone,email,birth_date,country,city,postal_code,address, count(user.id) ...	4 row(s) returned	0.000 sec / 0.000

Aquí realizamos el JOIN de las tablas user y transaction con el mismo user_id. En este caso escogemos las columnas para mostrar id, nombre, apellido, teléfono, email, fecha de nacimiento, país, ciudad, código postal y dirección que luego utilizaremos para agrupar; y creamos un contador de los user_id con la función COUNT que nos dará la cantidad de veces que cada usuario ha realizado una transacción. Por último, tal y como nos dice el enunciado filtramos con la clausula HAVING aquellos que tengan estrictamente más de 30 transacciones. En este caso, estamos teniendo en cuenta todo tipo de transacciones (declined={0,1)} y nos devuelve 4 usuarios.

```
157     -- Ejercicio 1 nivel 1 contando solo transacciones no declined
158 •   SELECT user.id,name,surname,phone,email,birth_date,country,city,postal_code,address, count(user.id) as contador_transacciones
159     FROM user
160     JOIN transaction ON user_id=user.id
161     WHERE declined = 0
162     GROUP BY user.id, name, surname, phone, email, birth_date, country,city, postal_code, address
163     HAVING contador_transacciones > 30;
164
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	name	surname	phone	email	birth_date	country	city	postal_code	address	contador_transacciones
92	Lynn	Riddle	1-387-885-4057	vitae.aliquet@outlook.edu	1984-09-21	United States	Bozeman	61871	P.O. Box 712, 7907 Est St.	39
267	Ocean	Nelson	079-481-2745	aenean@yahoo.com	1991-12-26	Canada	Charlottetown	85X 3P4	Ap #732-8357 Pede, Rd.	39
272	Hedwig	Gilbert	064-204-8788	sem.eget@icloud.edu	1991-04-16	Canada	Tuktoyaktuk	Q4C 3G7	P.O. Box 496, 5145 Sapien Road	38

Result 54

Output

Action Output

#	Time	Action	Message	Duration /
1	13:31:53	SELECT user.id,name,surname,phone,email,birth_date,country,city,postal_code,address, count(user.id) ...	3 row(s) returned	0.015 sec

En este segundo caso solo tenemos en cuenta aquellos que las transacciones hayan sido NO declined. Aquí obtenemos solo 3 usuarios con más de 30 transacciones.

- Exercici 2

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

```
165     -- Ejercicio 2 nivel 2
166     -- Seleccionamos el iban, nombre de la compañia y la media de las transacciones de la empresa Donec Ltd
167     -- haciendo JOIN de las tablas transaction, company y credit_card, teniendo en cuenta declined y no declined
168 •   SELECT iban, company_name, round(AVG(amount),2) as mitjana
169     FROM transaction
170     JOIN company on business_id = company.company_id
171     JOIN credit_card on card_id=credit_card.id
172     WHERE company_name='Donec Ltd'
173     GROUP BY iban, company_name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

iban	company_name	mitjana
PT87806228135092429456346	Donec Ltd	203.72

Result 56

Output

Action Output

#	Time	Action	Message
1	13:35:33	SELECT iban, company_name, round(AVG(amount),2) as mitjana FROM transaction JOIN company on ...	1 row(s) returned

En este ejercicio, realizamos el JOIN de tres tablas: transaction con company cuando coincide el company_id y transaction con credit_card realizando la identificación de tablas con la columna cerdit_card_id. Utilizamos la tabla transaction como base para las uniones ya que tenemos un esquema de estrella.

Por último, filtramos por el nombre de compañía ‘Donec Ltd’, agrupamos por iban y company_name que son justo las dos columnas que hemos pedido en el SELECT y creamos la columna sintética mitjana que nos da la media de amount de las transacciones para cada empresa con la función AVG y ROUND para redondear a 2 decimales, ya que es un precio. En este caso, obtenemos que la media para ‘Donec Ltd’ es de 203,72 teniendo en cuenta tanto declined como no.

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?

Aquí la nueva tabla será una vista que llamaremos status_card y que contendrá únicamente las columnas card_id y card_status (activated ó desactivated).

```
176     -- Creamos una vista status_card, utilizando RANK para obtener las ultimas transacciones de forma descendiente
177     -- filtramos por las 3 ultimas, agrupamos por card_id y comprobamos que el estado de declined suma 3 para saber si las 3 ultimas han sido declined
178     -- y asignamos "Desactivated" o "Active" a cada tarjeta si se cumple la condición
179 •   CREATE VIEW status_card AS
180     SELECT
181             card_id,
182             IF(SUM(declined)=3, "Desactivated", "Active") AS card_status
183     FROM (SELECT
184             card_id,
185             timestamp,
186             declined,
187             RANK() OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS ultimas
188             FROM transaction) AS inter
189     WHERE ultimas <= 3
190     GROUP BY card_id;
191
192     -- Realizamos la cuenta de las Active, en este caso todas las cards estan activas
193 •   SELECT COUNT("Active")
194     FROM status_card;
195
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

COUNT("Active")
275

Result 57

Output

Action Output

#	Time	Action	Message	Duration / Fet
1	13:35:33	SELECT iban, company_name, round(AVG(amount),2) as mitjana FROM transaction JOIN company on ...	1 row(s) returned	0.000 sec / 0.
2	13:38:27	SELECT COUNT("Active") FROM status_card LIMIT 0, 2000	1 row(s) returned	0.016 sec / 0.

Para realizar esta vista utilizamos una subquery y el uso de la función RANK(). En la subquery queremos tener las columnas card_id, timestamp, declined y últimas, que a partir de la función RANK nos asignará un índice para cada transacción hecha por una misma credit card (gracias al PARTITION BY) y ordenado de manera descendiente según la columna timestamp (de la más reciente a la menos). Toda esta información la cogemos de la tabla transaction. Por último, en el SELECT que nos servirá para introducir la información a la vista pedimos el card_id y asignamos Descativated (en caso contrario Active) si se cumple la condición de que haya exactamente 3 declined (SUM(declined)=3) dentro de las tres (o menos) últimas transacciones para cada credit card (esto lo miramos con el WHERE ultimas <=3). Finalmente, agrupamos por card_id.

Luego realizamos la query SELECT COUNT('Active') de la vista recién creada, para obtener cuantas hay activas. En este caso, hay 275 cards activas que son el total de todas las credit cards y por lo tanto no hay ninguna desactivada.

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 realizar este ejercicio, tendré en cuenta tanto si un producto aparece o no aparece en ninguna transacción, por lo tanto mostraremos los 100 productos. Es decir, se mostrarán también los productos que se han vendido 0 veces.

Realizamos la query que nos seleccionará las columnas product_id, el producto_name y la cantidad de veces que aparece en una transacción.

Utilizamos el LEFT JOIN entre las tablas product y transaction_product con la condición de relación que coincida el product_id. LEFT para tener todos los productos y utilizamos el contador COUNT de product_id de la tabla transaction_product para contar el numero de apariciones. Finalmente agrupamos por product.id. Vemos que nos aparecen los 100 products.

```
197     -- Ejercicio 2 nivel 3. Dir el nombre de cops que s'ha venut cada producte
198     -- Con todos los products id, incluso los productos de 0 veces
199
200 •   SELECT DISTINCT product.id, product_name, COUNT(product_id) AS veces
201     FROM product
202     LEFT JOIN transaction_product ON product_id=product.id
203     GROUP BY product.id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	product_name	veces
1	Direwolf Stannis	51
2	Tarly Stark	56
3	duel tourney Lannister	43
4	warden south duel	0
5	skywalker ewok	42
6	dooku solo	0
7	north of Casterly	44
8	Winterfell	0
9	Winterfell	0
10	Karstark Dorne	0
11	Karstark Dorne	40
12	duel Direwolf	0
13	palpatine chewbacca	51
14	Direwolf	0
15	Stannis warden	0
16	the duel warden	0
17	skywalker ewok sith	54
18	Karstark warden	0
19	dooku solo	44

Result 61

Output

Action Output

#	Time	Action	Message
1	13:44:52	SELECT DISTINCT product.id, product_name, COUNT(product_id) AS veces FROM product LEFT JOI...	100 row(s) returned