Sprint 4
Nivel 1

Descarga los archivos CSV, estudiales y diseña una base de datos con un esquema de estrella que contenga, al menos 4 tablas de las que puedas realizar las siguientes consultas.

Los archivos CSV que descargué fueron los siguientes y estos son los nombres de las columnas que están reflejados en la primera fila:

american_users:
id, name, surname, phone, email, birth_date, country, city, postal_code, address

european_users:
id, name, surname, phone, email, birth_date, country, city, postal_code, address

companies:
company_id, company_name, phone, email, country, website

credit_cards:
id, user_id, iban, pan , pin, cvv , track1, track2, expiring_date

products:
id, product_name, price, colour, weight, warehouse_id

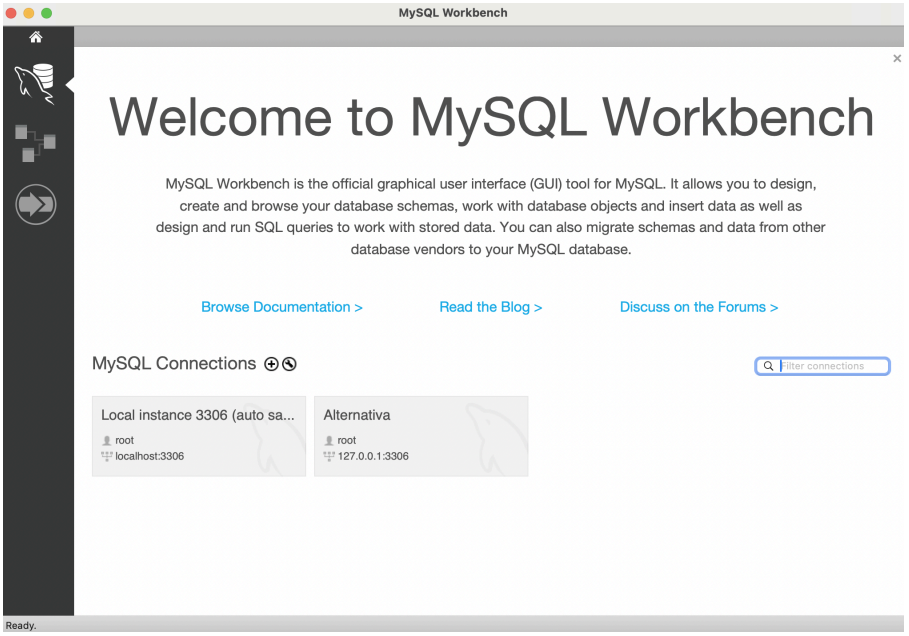
transactions:
id; card_id; business_id; timestamp; amount; declined; product_ids; user_id; lat; longitude

Configuración de Mysql para cargar los datos CSV.

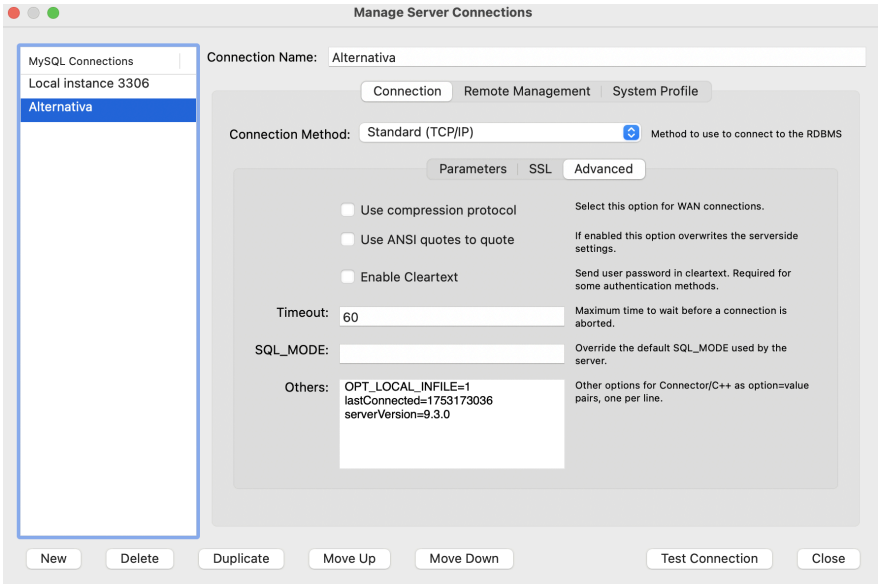
Antes de explicar como cree la base de datos, sus tablas, sus relaciones y como cargué los datos CSV en esas tablas, explico un paso fundamental en este proceso que es configurar el Mysql para poder cargar estos datos CSV.

Por medidas de seguridad en la configuración Mysql exige que el archivo donde están los datos que necesitas cargar estén en una carpeta especial privada, pero la ruta donde está la carpeta no la pude conseguir así que resolví creando una nueva conexión local en Mysql que permite cargar estos datos desde cualquier lugar del disco local.

Cree una nueva conexión llamada “Alternativa”



En la configuración en Advanced/Others puse el comando OPT_LOCAL_INFILE



Luego me conecto, creo una nueva query y ejecuto este comando cada vez que me conecte. Es importante ejecutar esto cada vez que se establezca la conexión porque por medidas de seguridad MySQL desactiva la opción que permite cagar los datos CSV desde cualquier parte del disco local.

```
SET GLOBAL local_infile = 1;
```

Creación de la base de datos y tablas.

En este paso cree la base de datos y la llamé '**business**'.

```
-- Creación de base de datos 'business'
CREATE DATABASE IF NOT EXISTS business;
USE business;
```

Basados en los datos CSV creo la tabla '**companies'** y cargo los datos en esa tabla, pido que ignore la primera fila que tiene el nombre de las columnas y le indico que los datos están separados por coma.

```
-- Creación de la tabla 'companies'
CREATE TABLE IF NOT EXISTS companies (
    company_id VARCHAR(15) PRIMARY KEY,
    company_name VARCHAR(255),
    phone VARCHAR(15),
    email VARCHAR(100),
    country VARCHAR(100),
    website VARCHAR(255)
);

-- Cargar los datos de la tabla 'companies'

LOAD DATA INFILE '/Users/luischicott/Documents/Especialización Datos/Bases de datos/DB Sprint 4/companies.csv'
INTO TABLE companies
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

Creo la tabla '**users'** y pongo los nombres de las columnas basados en los datos que tengo, hago una sola tabla para cargar los datos de los usuarios europeos y americanos.

```
-- Creación de la tabla 'users'
CREATE TABLE IF NOT EXISTS users (
    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)
);
```

Cargo los datos indicando la ruta donde está el archivo, le digo que ignore la primera fila que tiene el nombre de las columnas y le indico que los datos están separados por coma.

```
    -- Datos de los usuarios europeos
LOAD DATA LOCAL INFILE '/Users/luischicott/Documents/Especialización Datos/Bases de datos/DB Sprint 4/european_users.csv'
INTO TABLE users
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

-- Datos de los usuarios americanos

LOAD DATA LOCAL INFILE '/Users/luischicott/Documents/Especialización Datos/Bases de datos/DB Sprint 4/american_users.csv'
INTO TABLE users
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

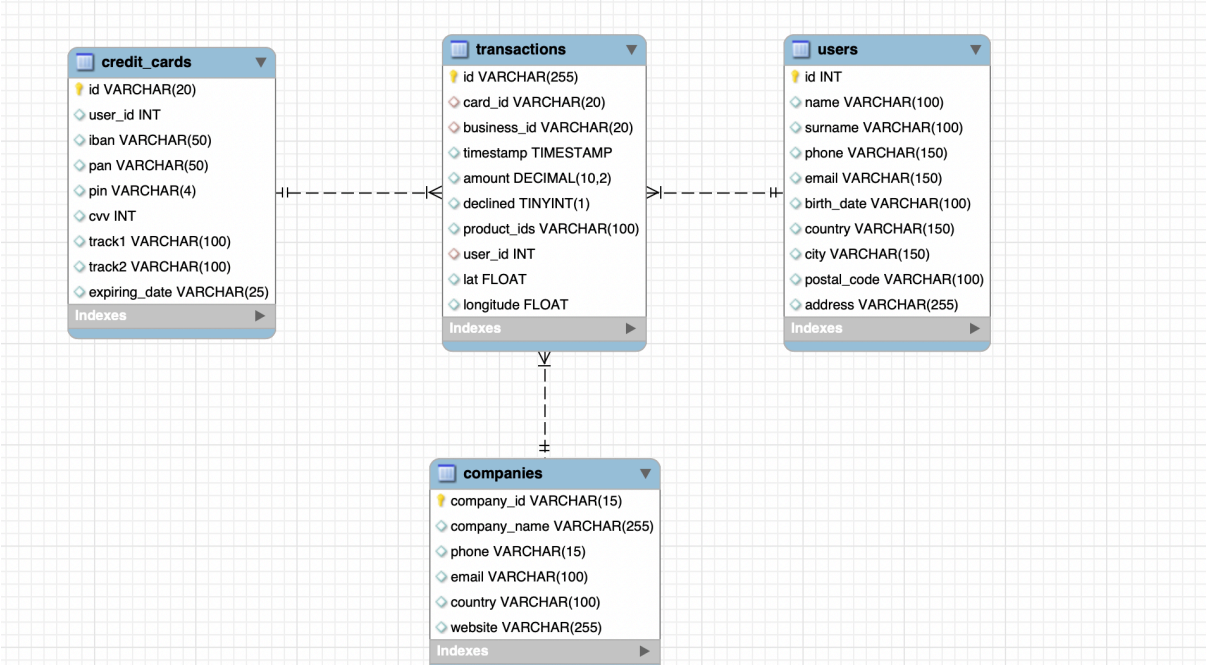
Creo la tabla **'credit_cards'** y cargo los datos indicando la ruta, pidiendo que ignore la primera fila y diciéndole que los datos están separados por coma.

```
-- Creación de la tabla 'credit_cards'
CREATE TABLE IF NOT EXISTS credit_cards (
    id VARCHAR(20),
    user_id INT,
    iban VARCHAR(50),
    pan VARCHAR(50),
    pin VARCHAR(4),
    cvv INT,
    track1 VARCHAR(100),
    track2 VARCHAR(100),
    expiring_date VARCHAR(25),
    PRIMARY KEY(id)
);

-- Cargar datos de la tabla 'creditcards'
LOAD DATA LOCAL INFILE '/Users/luischicott/Documents/Especialización Datos/Bases de datos/DB Sprint 4/credit_cards.csv'
INTO TABLE credit_cards
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

Creo la tabla **'transactions'** al final porque es la tabla de hechos en la estructura y desde su creación establezco con REFERENCE la relación con las otras 3 tablas a través de sus ids. Al revisar los datos veo que en la columna product_ids hay muchos ids separados por comas así que lo declaro como VARCHAR para que guarde todo y le indico que los datos están separados esta vez por punto y coma.

```
-- Creación de la tabla transactions
    CREATE TABLE IF NOT EXISTS transactions (
    id VARCHAR(255) PRIMARY KEY,
    card_id VARCHAR(20) REFERENCES credit_cards(id),
    business_id VARCHAR(20) REFERENCES companies(company_id),
    timestamp TIMESTAMP,
    amount DECIMAL(10, 2),
    declined BOOLEAN,
    product_ids VARCHAR(100),
    user_id INT REFERENCES users(id),
    lat FLOAT,
    longitude FLOAT
);

-- Cargar datos de la tabla 'transactions'. Datos separados por ';'
LOAD DATA LOCAL INFILE '/Users/luischicott/Documents/Especialización Datos/Bases de datos/DB Sprint 4/transactions.csv'
INTO TABLE transactions
FIELDS TERMINATED BY ';' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

La estructura de la base de datos es de modelo estrella donde la tabla de hechos es **transactions** y está rodeada de las tablas de dimensiones que son **credit_cards**, **companies** y **users.** La relación de la tabla de hechos hacia las tablas de dimensiones son de N:1 (muchos a uno) donde están unidas por los ids que son clave primaria en las tablas de dimensiones y esos ids son claves foráneas en la tabla de hechos.



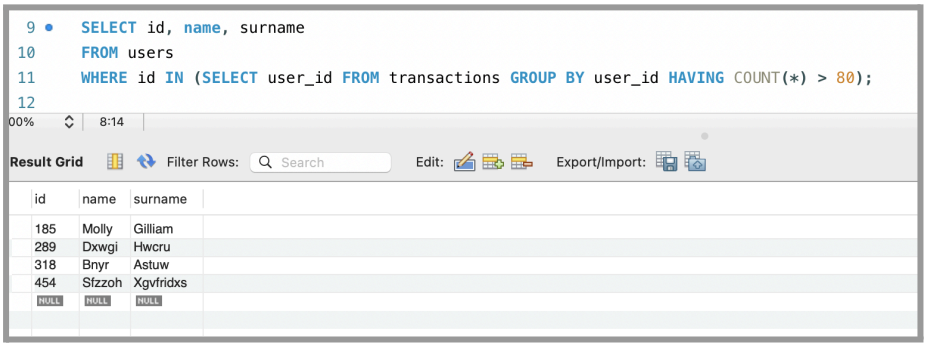
Sprint 4

Nivel 1

Ejercicio 1

Realiza una subconsulta que muestre a todos los usuarios con más de 80 transacciones utilizando al menos 2 tablas.

En la subconsulta tomé de la tabla **transactions** el id de los usuarios que tuvieran más de 80 transacciones y en la consulta principal mostré el id, nombre y apellido de la tabla **users** de esos usuarios que me devolvía la subconsulta.



```
9 •   SELECT id, name, surname
10    FROM users
11    WHERE id IN (SELECT user_id FROM transactions GROUP BY user_id HAVING COUNT(*) > 80);
12
```

00% 8:14

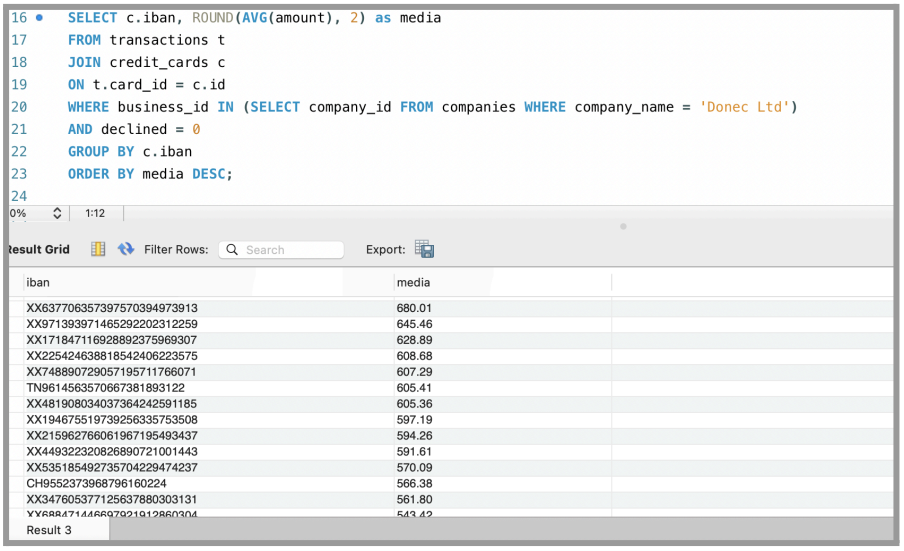
Result Grid | Filter Rows: Search | Edit: | Export/Import:

id	name	surname
185	Molly	Gilliam
289	Dxwgi	Hwcru
318	Bnyr	Astuw
454	Sfzzoh	Xgvfridxs
NULL	NULL	NULL

Sprint 4
Nivel 1
Ejercicio 2

Muestra la media de amount por IBAN de las tarjetas de crédito en la compañía Donec Ltd., utiliza por lo menos 2 tablas.

Hice una subconsulta de la tabla **companies** para tener el id de la empresa Donec Ltd, luego hice un JOIN de las tablas **transactions** y **credit_cards** para agrupar por IBAN y calcular la media del amount de las tarjetas de Donect Ltd poniendo también como condición que sean transacciones no declinadas y ordené el resultado por esa media de manera descendente.



```sql
SELECT c.iban, ROUND(AVG(amount), 2) as media
FROM transactions t
JOIN credit_cards c
ON t.card_id = c.id
WHERE business_id IN (SELECT company_id FROM companies WHERE company_name = 'Donec Ltd')
AND declined = 0
GROUP BY c.iban
ORDER BY media DESC;
```

Result Grid — Filter Rows: Search — Export:

iban	media
XX6377063573975703949739 13	680.01
XX9713939714652922023122 59	645.46
XX1718471169288923759693 07	628.89
XX2254246388185424062235 75	608.68
XX7488907290571957117660 71	607.29
TN9614563570667381893122	605.41
XX4819080340373642425911 85	605.36
XX1946755197392563357535 08	597.19
XX2159627660619671954934 37	594.26
XX4493223208268907210014 43	591.61
XX5351854927357042294742 37	570.09
CH9552373968796160224	566.38
XX3476053771256378803031 31	561.80
XX6884714469792191286030 4	543.42

Result 3

Sprint 4
Nivel 2

Crea una nueva tabla que refleje el estado de las tarjetas de crédito basado en si las últimas tres transacciones fueron declinadas.

Como la tabla va a tener los datos del resultado de una consulta puse el CREATE TABLE antes de la consulta y la encerré entre paréntesis y lo importante es ponerle alias a todas las columnas que den como resultado porque luego será el nombre de la columna en la tabla.

Creo dos tablas temporales que me van a servir sucesivamente para tener el resultado final.

En la **primera tabla** temporal *"UltimasTransacciones"* pido que agrupe/ordene con PARTITION BY las transacciones por los card_id y luego que ordene de la manera descendente desde las últimas transacciones y con ROW_NUMBER enumere en una nueva columna llamada *rank_transacciones* esas transacciones.

Ya con el resultado de esa tabla temporal puedo hacer la **segunda tabla** temporal "EstadoTarjetas" donde pongo como condición que tome solo como max las primeras 3 filas de cada *rank_transacciones*, que agrupe por card_id y que sume la columna declined y con un CASE pongo como condición que si el resultado es 1 es TRUE y sino es FALSE y ese resultado lo ponga en una columna llamada *transacciones_declinadas,* también pido que me cuente el número de transacciones y lo pongo en la columna *total_ultimas_transacciones*

Por último con ese resultado ya puedo hacer la **consulta final** que consiste que como resultado me dé 2 columnas una con el id de la tarjeta y otra una columna llamada active_card donde pondré el resultado de una condición que verifique que si esa tarjeta tiene 3 transacciones y las últimas 3 fueron declinadas entonces el resultado sea FALSE y sino es TRUE.

```sql
-- Creación de la tabla 'cards_status' y en ingresar los datos que resulten de esta consulta

CREATE TABLE IF NOT EXISTS cards_status AS (
WITH UltimasTransacciones AS (
    SELECT
        timestamp,
        card_id,
        declined,
        ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS rank_transacciones
    FROM
        transactions
), EstadoTarjetas AS (
    SELECT
        card_id,
        SUM(CASE WHEN declined = TRUE THEN 1 ELSE 0 END) AS transacciones_declinadas,
        COUNT(*) AS total_ultimas_transacciones
    FROM
        UltimasTransacciones
    WHERE
        rank_transacciones <= 3
    GROUP BY
        card_id
)
SELECT
    e.card_id AS card_id,
    CASE
        WHEN e.transacciones_declinadas = 3 AND e.total_ultimas_transacciones = 3 THEN FALSE
        ELSE TRUE
    END AS active_card
FROM
    EstadoTarjetas e
);
```

Sprint 4
Nivel 2
Ejercicio 1

¿Cuántas tarjetas están activas?

Con la tabla creada **card_status** que tiene 2 columnas una con el id de las tarjetas y otra llamada active_card que tiene un valor booleano, pido que me cuente la cantidad de tarjetas y como condición pongo que el valor de active_card sea “1” es decir TRUE.

```sql
-- Sprint 4, Nivel 2,  Ejercicio 1
-- ¿Cuántas tarjetas están activas?

SELECT COUNT(*) as cant_tarjetas_activas
FROM cards_status
WHERE active_card = 1;
```

100% 1:34

Result Grid | Filter Rows: Search | Export:

cant_tarjetas_activas
4995

Sprint 4
Nivel 3

Crea una tabla con la que podamos unir los datos del nuevo archivo products.csv con la base de datos creada, teniendo en cuenta que desde transactions tienes product_ids.

Primero creo la tabla ‘**products’** y pongo los nombres de las columnas basados en los datos que tengo y cargo los datos.

```sql
-- Creación de la tabla 'products'
    CREATE TABLE IF NOT EXISTS products (
        id INT PRIMARY KEY,
        product_name VARCHAR(255),
        price VARCHAR(20),
        colour VARCHAR(100),
        weight VARCHAR(100),
        warehouse_id VARCHAR(15)
    );

    -- Cargar los datos de la tabla 'products'

LOAD DATA LOCAL INFILE '/Users/luischicott/Documents/Especialización Datos/Bases de datos/DB Sprint 4/products.csv'
INTO TABLE products
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

Creo una tabla llamada **transactions_products** y los datos que va a tener son el resultado de una consulta que voy a hacer entre las tablas **transactions** y **products.**

Para hacer la consulta tengo que unir las tablas y solo lo puedo hacer por los ids, pero el id del producto en la tabla transactions está en una lista, es decir hay varios ids por fila en una sola columna. Por eso usé el FIND_IN_SET para que desde la tabla products vaya buscando por cada producto separado por una coma en cada fila y usé REMPLACE para eliminar los espacios en blaco de la lista.

El resultado es una tabla que tiene 2 columnas, por un lado el id de las transacciones y por otro lado el id del producto esa transacción esta vez separado.

```sql
-- Creación de la tabla 'transactions_products'.

CREATE TABLE IF NOT EXISTS transactions_products AS(
SELECT  transactions.id AS transaction_id, products.id AS product_id
    FROM transactions
    JOIN products
    ON   FIND_IN_SET((products.id),
    REPLACE(transactions.product_ids, ' ', '')) > 0
    );
```

```sql
175 •  SELECT * FROM transactions_products;
176
```

100% 1:176

Result Grid Filter Rows: Search Export: Fetch rows:

transaction_id	product_id
00043A49-2949-494B-A5DD-A5BAE3BB19DD	97
00043A49-2949-494B-A5DD-A5BAE3BB19DD	87
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	30
00045D6B-ED2E-4F2F-8186-CEE074D875D0	16
00045D6B-ED2E-4F2F-8186-CEE074D875D0	11
000481C3-1C26-4FEF-83A0-4CD0EB004BBD	72
00051AA4-9CBF-4268-B070-C38062A1B3F2	18

Sprint 4
Nivel 3
Ejercicio 1

Necesitamos conocer el número de veces que se ha vendido cada producto.

Tomé la tabla creada **trasantions_products** y agrupé por product_id conté la cantidad de transacciones para ver cuantas veces se vendió el producto y ordené de manera descendente para ver del producto que más se vendió al que menos.

```sql
-- Necesitamos conocer el número de veces que se ha vendido cada producto.

SELECT product_id, COUNT(transaction_id) AS cantidad
FROM transactions_products
GROUP BY product_id
ORDER BY cantidad DESC;
```

100% 24:40

Result Grid Filter Rows: Search Export:

product_id	cantidad
52	2654
29	2635
21	2609
16	2608
66	2601
87	2598
48	2597
33	2597
23	2593
68	2589