★☆☆ 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:

1) creamos la **BBDD** sprint4

```
6
7 •  CREATE DATABASE sprint4;
8
9
10
```

Output

Action Output

#	Time	Action	Message
1	14:23:27	CREATE DATABASE sprint4	1 row(s) affected

2) creamos tablas american_users, european_users, companies, credit_cards y transactions. También relacionamos transactions con credit_card y companies. No relacionamos aun las tablas de users ya que planeamos unirlas.

```
11 •  DROP TABLE IF EXISTS american_users;
12 •  CREATE TABLE IF NOT EXISTS american_users (
13        id INT PRIMARY KEY,
14        name VARCHAR(30),
15        surname VARCHAR(30),
16        phone VARCHAR(30),
17        email VARCHAR(50),
18        birth_date VARCHAR(20),
19        country VARCHAR(30),
20        city VARCHAR(30),
21        postal_code VARCHAR(20),
22        address VARCHAR(50)
23    );
24
25 •  DROP TABLE IF EXISTS european_users;
26 •  CREATE TABLE IF NOT EXISTS european_users (
27        id INT PRIMARY KEY,
28        name VARCHAR(30),
29        surname VARCHAR(30),
30        phone VARCHAR(30),
31        email VARCHAR(50),
```

Output

Action Output

#	Time	Action	Message
9	12:57:52	DROP TABLE IF EXISTS transactions	0 row(s) affected, 1 warning(s): 1051 Unknown table 'sprint4.transactions'
10	12:57:52	CREATE TABLE IF NOT EXISTS transactions (id VARCHAR(50) PRIMARY KEY, card_id VARCHAR(20),...	0 row(s) affected

3) cargamos cada tabla con los archivos csv, teniendo en cuenta como estan separados y comprendidos los registros e ignorando la fila 1 que comprende los nombres de las columnas

```
81  LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/american_users.csv'
82  INTO TABLE american_users
83  FIELDS TERMINATED BY ','
84  ENCLOSED BY '"'
85  IGNORE 1 ROWS;
86
87  LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/companies.csv'
88  INTO TABLE companies
89  FIELDS TERMINATED BY ','
90  ENCLOSED BY '"'
91  IGNORE 1 ROWS;
92
93  LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/credit_cards.csv'
94  INTO TABLE credit_card
95  FIELDS TERMINATED BY ','
96  ENCLOSED BY '"'
97  IGNORE 1 ROWS;
98
99  LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/european_users.csv'
100 INTO TABLE european_users
101 FIELDS TERMINATED BY ','
102 ENCLOSED BY '"'
```

Output

Action Output

#	Time	Action	Message
3	13:05:39	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/credit_cards.csv' INTO TABLE c...	5000 row(s) affected Records: 5000 Deleted: 0 Skipped: 0 Warnings: 0
4	13:05:39	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/european_users.csv' INTO TAB...	3990 row(s) affected Records: 3990 Deleted: 0 Skipped: 0 Warnings: 0

4) verificamos si hay ids duplicados en las tablas american_users y european_users ya que nuestro objetivo es unirlas en una sola llamada users para asi relacionarla a transactions.

```
113 SELECT COUNT(*) AS total_repetidos
114 FROM american_users a
115 JOIN european_users e ON a.id = e.id;
116
```

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

total_repetidos
0

Result 1

Output

Action Output

#	Time	Action	Message
1	17:03:25	SELECT COUNT(*) AS total_repetidos FROM american_users a JOIN european_users e ON a.id = e.id	1 row(s) returned

5) agregamos una columna en american_users y en european_users llamada 'region' que identifique si el país corresponde a America o Europa en caso de que en un futuro tenga alguna consulta relacionada a cual continente pertenecen.

```
120  ALTER TABLE american_users
121  ADD region VARCHAR(20);
122
123  UPDATE american_users
124  SET region = 'America'
125  WHERE id > 0;
126
127  ALTER TABLE european_users
128  ADD region VARCHAR(20);
129
130  UPDATE european_users
131  SET region = 'Europa'
132  WHERE id != '';
133
```

Output

Action Output

#	Time	Action	Message
1	17:05:52	ALTER TABLE american_users ADD region VARCHAR(20)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
2	17:05:52	UPDATE american_users SET region = 'America' WHERE id > 0	1010 row(s) affected Rows matched: 1010 Changed: 1010 Warnings: 0
3	17:05:52	ALTER TABLE european_users ADD region VARCHAR(20)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
4	17:05:52	UPDATE european_users SET region = 'Europa' WHERE id != ''	3990 row(s) affected Rows matched: 3990 Changed: 3990 Warnings: 0

6) unimos tablas american.users y european_users. Usamos UNION ALL ya que anteriormente comprobamos que no habían duplicados.

```
136  DROP TABLE IF EXISTS users;
137  CREATE TABLE IF NOT EXISTS users AS
138  SELECT * FROM american_users
139  UNION ALL
140  SELECT * FROM european_users;
141
```

Output

Action Output

#	Time	Action	Message
1	17:05:52	ALTER TABLE american_users ADD region VARCHAR(20)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
2	17:05:52	UPDATE american_users SET region = 'America' WHERE id > 0	1010 row(s) affected Rows matched: 1010 Changed: 1010 Warnings: 0
3	17:05:52	ALTER TABLE european_users ADD region VARCHAR(20)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
4	17:05:52	UPDATE european_users SET region = 'Europa' WHERE id != ''	3990 row(s) affected Rows matched: 3990 Changed: 3990 Warnings: 0
5	17:07:42	DROP TABLE IF EXISTS users	0 row(s) affected, 1 warning(s): 1051 Unknown table 'sprint4.users'
6	17:07:42	CREATE TABLE IF NOT EXISTS users AS SELECT * FROM american_users UNION ALL SELECT * FROM european_users	5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0

7) agregamos primary key a users.

```
143
144  ALTER TABLE users
145  ADD PRIMARY KEY (id);
146
```

Output

Action Output

#	Time	Action	Message
1	17:09:00	ALTER TABLE users ADD PRIMARY KEY (id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

8) relacionamos tabla users con transactions.

```
149 •  ALTER TABLE transactions
150    ADD CONSTRAINT users_transaction
151    FOREIGN KEY (user_id)
152    REFERENCES users(id);
153
```

Output

Action Output

#	Time	Action	Message
1	17:09:44	ALTER TABLE transactions ADD CONSTRAINT users_transaction FOREIGN KEY (user_id) REFERENCES users(id)	100000 row(s) affected Records: 100000 Duplicates: 0 Warnings: 0

9) cambiamos users.birth_date a DATE.

```
155
156 •  UPDATE users
157    SET birth_date = STR_TO_DATE(birth_date,'%b %d, %Y')
158    WHERE id > 0;
159
```

Output

Action Output

#	Time	Action	Message
1	17:10:31	UPDATE users SET birth_date = STR_TO_DATE(birth_date,'%b %d, %Y') WHERE id > 0	5000 row(s) affected Rows matched: 5000 Changed: 5000 Warnings: 0

10) cambiamos credit_card.expiring_date a DATE.

```
162 •  UPDATE credit_card
163    SET expiring_date = STR_TO_DATE(expiring_date, '%m/%d/%y')
164    WHERE id != '';
165
```

Output

Action Output

#	Time	Action	Message
1	17:11:10	UPDATE credit_card SET expiring_date = STR_TO_DATE(expiring_date, '%m/%d/%y') WHERE id != ''	5000 row(s) affected Rows matched: 5000 Changed: 5000 Warnings: 0

11) eliminamos american_users y european_users ya que tenemos todo en users.

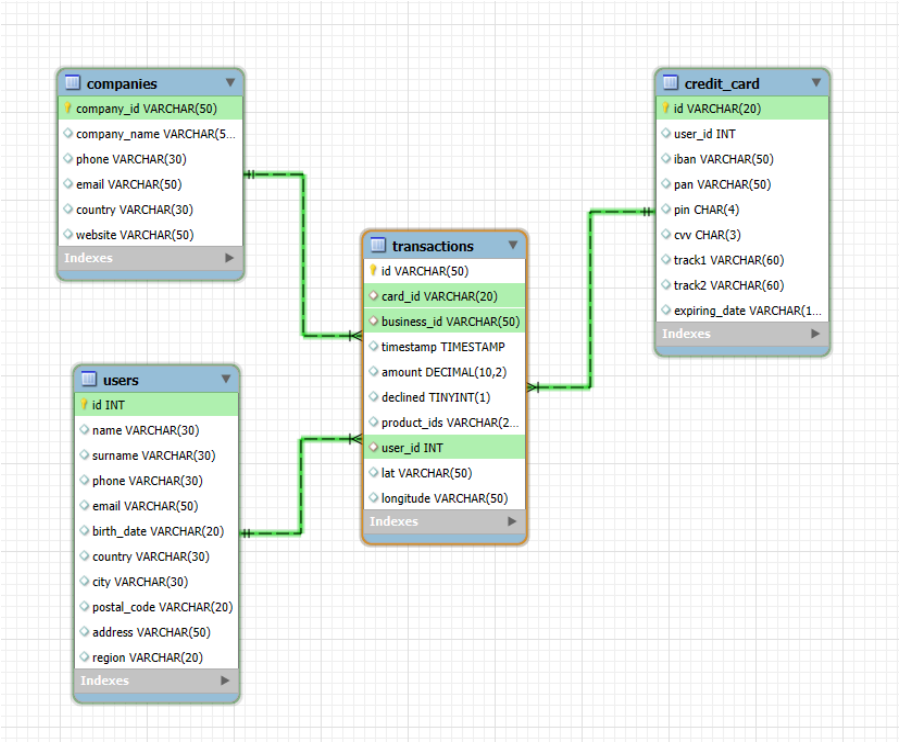
```
168 •  DROP TABLE IF EXISTS american_users, european_users;
169
```

Output

Action Output

#	Time	Action	Message
1	17:12:13	DROP TABLE IF EXISTS american_users, european_users	0 row(s) affected

De momento, este es el diagrama final :



Ejercicio 1

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

```
173 •  SELECT u.name, u.surname, COUNT(t.id) AS cantidad_transacciones
174    FROM transactions t
175    JOIN users u ON t.user_id = u.id
176    WHERE t.declined = 0 AND EXISTS (SELECT 1
177                                     FROM transactions t
178                                     WHERE t.user_id = u.id
179                                     AND t.declined = 0
180                                     GROUP BY t.user_id
181                                     HAVING COUNT(t.id) > 80)
182    GROUP BY u.id,u.name, u.surname;
183
```

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

name	surname	cantidad_transacciones
Bnyr	Astuw	86
Dxwgi	Hwcru	91
Molly	Gilliam	107

Result 3

Output

Action Output

#	Time	Action	Message
1	17:14:54	SELECT u.name, u.surname, COUNT(t.id) AS cantidad_transacciones FROM transactions t JOIN users u ON t.user_id = u.id WHERE t.decline...	3 row(s) returned

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.

```
187 •  SELECT cc.iban, ROUND(AVG(t.amount),2) AS media
188    FROM transactions t
189    JOIN credit_card cc ON t.card_id = cc.id
190    JOIN companies c ON c.company_id = t.business_id
191    WHERE c.company_name = 'Donec Ltd'
192    GROUP BY cc.iban;
193
```

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

iban	media
XX9114064011255863075868805	356.25
SK9446370242474562577506	142.96
XX7767529178459529755556 40	257.37
XX4138273622897193049089 90	139.59
XX3477872460707696107803 08	240.41
XX6887684365430908948546 02	188.58
MK28368851538688349	439.39

Result 4

Output

Action Output

#	Time	Action	Message
1	17:18:13	SELECT cc.iban, ROUND(AVG(t.amount),2) AS media FROM transactions t JOIN credit_card cc ON t.card_id = cc.id JOIN companies c ON c....	371 row(s) returned

★★☆ Nivel 2

Crea una nueva tabla que refleje el estado de las tarjetas de crédito basado en si las tres últimas transacciones han sido declinadas entonces es inactivo, si al menos una no es rechazada entonces es activo . Partiendo de esta tabla responde:

1) *creamos una tabla llamada estado_tarjetas para que en un futuro nos refleje si el estado de las tarjetas es 'Activo' o 'Inactivo'*

```
200 •  DROP TABLE IF EXISTS estado_tarjetas;
201 •  CREATE TABLE IF NOT EXISTS estado_tarjetas (
202        card_id VARCHAR(20) PRIMARY KEY,
203        estado VARCHAR(20)
204    );
205
```

Output

Action Output

#	Time	Action	Message
1	17:23:02	DROP TABLE IF EXISTS estado_tarjetas	0 row(s) affected, 1 warning(s): 1051 Unknown table 'sprint4.estado_tarjetas'
2	17:23:02	CREATE TABLE IF NOT EXISTS estado_tarjetas (card_id VARCHAR(20) PRIMARY KEY, estado VARCHAR(20))	0 row(s) affected

2) *cargamos la columna card_id de la tabla estado_tarjetas con todos los registros de la columna id de credit_card.*

```
208 •  INSERT INTO estado_tarjetas (card_id)
209    SELECT id FROM credit_card;
210
```

Output

Action Output

#	Time	Action	Message
1	17:24:21	INSERT INTO estado_tarjetas (card_id) SELECT id FROM credit_card	5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0

3) *hacemos un update en la tabla estado_tarjetas et. Para ello creamos una tabla temporal ultimos3. Primero filtramos las 3 últimas transacciones por cada card_id en una subconsulta interna que cuente cuantas transacciones existen para la misma tarjeta (WHERE t2.card_id = t1.card_id) y para cada transacción cuantas posteriores existen (AND t2.timestamp > t1.timestamp) con la condición que solo nos devuelva las últimas tres (< 3). Utilizando CASE WHEN creamos una columna temporal llamada nuevo_estado donde si la suma de las 3 transacciones es igual a 3 (o en otras palabras que todas son declinadas) será 'Inactiva' y si es de otra manera será 'Activa'. Y por último esta información resultante se agrega a la columna et.estado.*

```
UPDATE estado_tarjetas et
JOIN (
    SELECT t1.card_id,
           CASE
               WHEN SUM(t1.declined) = 3 THEN 'Inactiva'
               ELSE 'Activa'
           END AS nuevo_estado
    FROM transactions t1
    WHERE (
        SELECT COUNT(*)
        FROM transactions t2
        WHERE t2.card_id = t1.card_id
          AND t2.timestamp > t1.timestamp
    ) < 3
    GROUP BY t1.card_id
    ) AS ultimos3
ON et.card_id = ultimos3.card_id
SET et.estado = ultimos3.nuevo_estado
WHERE et.card_id != '';
```

Output

Action Output

#	Time	Action	Message
1	18:00:27	UPDATE estado_tarjetas et JOIN (SELECT t1.card_id, CASE WHEN SUM(t1.declined) = 3 THEN 'Inactiva' ELSE...	5000 row(s) affected Rows matched: 5000 Changed: 5000 Warnings: 0

4) *relacionamos estado_tarjetas con transactions.*

```
ALTER TABLE transactions
ADD CONSTRAINT et_transactions
FOREIGN KEY (card_id)
REFERENCES estado_tarjetas(card_id);
```

Output

Action Output

#	Time	Action	Message
1	18:00:27	UPDATE estado_tarjetas et JOIN (SELECT t1.card_id, CASE WHEN SUM(t1.declined) = 3 THEN 'Inactiva' ELSE...	5000 row(s) affected Rows matched: 5000 Changed: 5000 Warnings: 0
2	18:01:27	ALTER TABLE transactions ADD CONSTRAINT et_transactions FOREIGN KEY (card_id) REFERENCES estado_tarjetas(card_id)	100000 row(s) affected Records: 100000 Duplicates: 0 Warnings: 0

Ejercicio 1

¿Cuántas tarjetas están activas?

```
SELECT COUNT(*)
FROM estado_tarjetas
WHERE estado = 'Activa';
```

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

COUNT(*)
4995

Result 5

Output

Action Output

#	Time	Action	Message
1	18:02:02	SELECT COUNT(*) FROM estado_tarjetas WHERE estado = 'Activa'	1 row(s) returned

★★★ 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. Genera la siguiente consulta:

1) *creamos la tabla products.*

```
254 •  DROP TABLE IF EXISTS products;
255 •  CREATE TABLE IF NOT EXISTS products (
256        id VARCHAR(20) PRIMARY KEY,
257        product_name VARCHAR(30),
258        price VARCHAR(10),
259        colour VARCHAR(30),
260        weight VARCHAR(10),
261        warehouse_id VARCHAR(10)
262    );
263
```

Output

Action Output

#	Time	Action	Message
1	18:04:16	DROP TABLE IF EXISTS products	0 row(s) affected, 1 warning(s): 1051 Unknown table 'sprint4.products'
2	18:04:16	CREATE TABLE IF NOT EXISTS products (id VARCHAR(20) PRIMARY KEY, product_name VARCHAR(30), price VARCHAR(10), colo...	0 row(s) affected

2) *cargamos tabla products con el archivo csv proporcionado.*

```
266 •  LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/products.csv'
267    INTO TABLE products
268    FIELDS TERMINATED BY ','
269    ENCLOSED BY '"'
270    IGNORE 1 ROWS;
271
272    -- 3) renombramos la tabla transactions para en un futuro borrar la original sin modificar
```

Output

Action Output

#	Time	Action	Message
1	18:05:14	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/products.csv' INTO TABLE products FIELDS TERMINATED BY ',' ...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0

3) *renombramos la tabla transactions para en un futuro borrar la original que esta sin modificar.*

```
274 •  RENAME TABLE transactions TO transactions_backup;
```

Output

Action Output

#	Time	Action	Message
1	18:06:01	RENAME TABLE transactions TO transactions_backup	0 row(s) affected

4) *creamos una nueva tabla transactions para en un futuro sea la definitiva. Como modificamos la tabla transactions, ahora tenemos varios transaction id repetidos para cada product_id nos daba un error de duplicate entry y la manera de solucionarlo fue utilizando una clave primaria compuesta. También aprovechando la creación de la tabla relacione las tablas transactions y products con la foreign key.*

```sql
DROP TABLE IF EXISTS transactions;
CREATE TABLE IF NOT EXISTS transactions (
    id VARCHAR(50),
    card_id VARCHAR(20),
    business_id VARCHAR(50),
    timestamp TIMESTAMP,
    amount DECIMAL(10, 2),
    declined BOOLEAN,
    product_id VARCHAR(20),
    user_id INT,
    lat VARCHAR(50),
    longitude VARCHAR(50),
    PRIMARY KEY (id, product_id), -- para solucionar el error duplicate entry hacemos clave primaria compuesta (id, product_id)
    FOREIGN KEY (card_id) REFERENCES credit_card(id),
    FOREIGN KEY (business_id) REFERENCES companies(company_id),
    FOREIGN KEY (product_id) REFERENCES products(id) -- aqui se relaciona products con transactions
);
```

Output

Action Output

#	Time	Action	Message
1	18:07:10	DROP TABLE IF EXISTS transactions	0 row(s) affected, 1 warning(s): 1051 Unknown table 'sprint4.transactions'
2	18:07:10	CREATE TABLE IF NOT EXISTS transactions (id VARCHAR(50), card_id VARCHAR(20), business_id VARCHAR(50), timestamp TIMEST...	0 row(s) affected

5) *cargamos la tabla con todos los registros de transactions_backup tal cual están, menos product_id que la obtendremos utilizando JSON para desglosar cada product_id en columnas separadas.*

```sql
INSERT INTO transactions (
    id, card_id, business_id, timestamp, amount, declined, product_id, user_id, lat, longitude
)
SELECT
    t.id,
    t.card_id,
    t.business_id,
    t.timestamp,
    t.amount,
    t.declined,
    CAST(j.value AS UNSIGNED) AS product_id,
    t.user_id,
    t.lat,
    t.longitude
FROM transactions_backup t
JOIN JSON_TABLE(
    CONCAT(
        '["',
        REPLACE(t.product_ids, ',', '","'),
        '"]'
    ),
    '$[*]' COLUMNS (value VARCHAR(50) PATH '$')
) AS j;
```

Output

Action Output

#	Time	Action	Message
1	18:10:56	INSERT INTO transactions (id, card_id, business_id, timestamp, amount, declined, product_id, user_id, lat, longitude) SELECT t.id, t.ca...	253391 row(s) affected Records: 253391 Duplicates: 0 Warnings: 0

6) *eliminamos la tabla transactions_backup.*

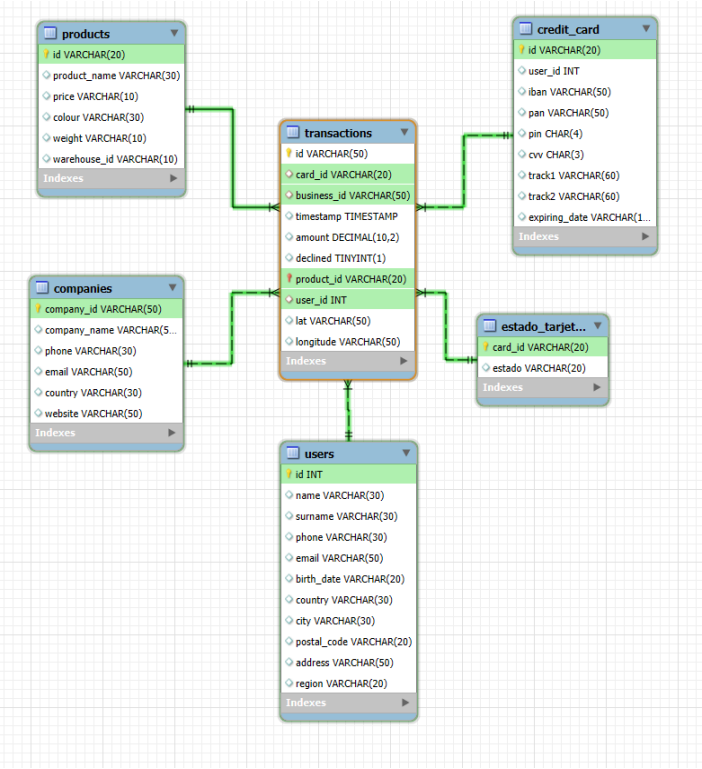
```
325 •   DROP TABLE IF EXISTS transactions_backup;
326
```

Output

Action Output

#	Time	Action	Message
1	18:15:34	DROP TABLE IF EXISTS transactions_backup	0 row(s) affected

Este es el diagrama de como se veria todo nuestro esquema final:



Ejercicio 1

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

```
330 •   SELECT product_id, COUNT(*) AS veces_vendidos
331     FROM transactions t
332     GROUP BY product_id;
333
```

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

product_id	veces_vendidos
1	2468
10	2571
100	2517
11	2573
12	2558
13	2560

Result 6

Output

Action Output

#	Time	Action	Message
1	18:17:41	SELECT product_id, COUNT(*) AS veces_vendidos FROM transactions t GROUP BY product_id	100 row(s) returned