

Sprint 4: Modelat SQL

Tarea S4.01. Creación de Base de Datos

NIVEL 1

Descarga los archivos CSV, estudialos 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:

Primero se crea la base de datos con la siguiente instrucción.

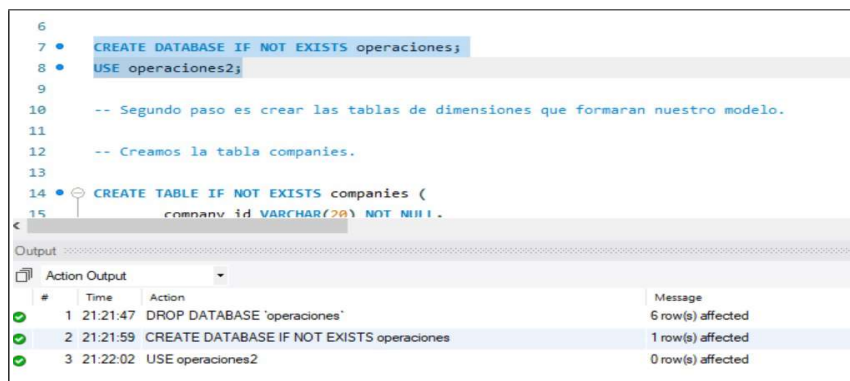


Imagen 1. Instrucción de MySql para crear un DATABASE.

Una vez creada la DATABASE, se crean las diferentes tablas, y se cargan los datos de cada una a partir de los distintos archivos csv.

La primera tabla a crear es **Companies**.

```
CREATE TABLE IF NOT EXISTS companies (
    company_id VARCHAR(20) NOT NULL,
    company_name VARCHAR(50) NOT NULL,
    phone VARCHAR(20),
    email VARCHAR(50),
    country VARCHAR(30),
    website VARCHAR(100)
);
```

Después de crear la tabla, se intentan cargar los datos. Surge error de permiso de acceso de MySQL (Error: **secure-file_priv**). Se debe encontrar la carpeta del sistema que tiene permiso el programa. Para encontrar esa carpeta y guardar los archivos ahí, se ejecuta la siguiente instrucción.

```
SHOW VARIABLES LIKE 'secure_file_priv';
```

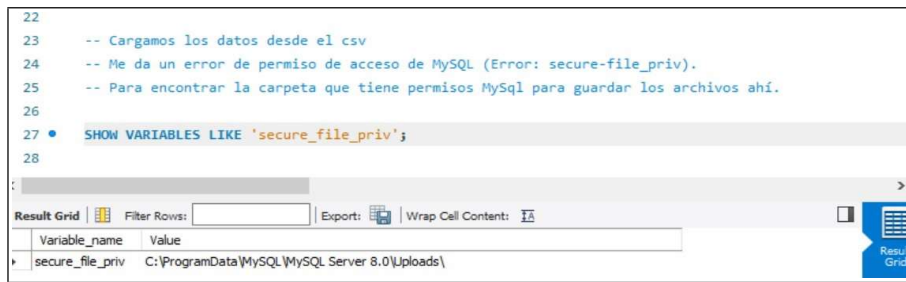


Imagen 2. Carpeta del sistema donde guardar los archivos a cargar en las tablas, que tiene acceso MySQL.

Se guardan los archivos csv en esa carpeta, y la carga de datos se hace desde esa ruta de carpeta. El código para cargar datos desde un archivo csv es el siguiente:

```

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/companies.csv'
INTO TABLE companies
FIELDS TERMINATED BY ','
ENCLOSED BY "
IGNORE 1 ROWS;

```

La instrucción dice cargar los datos desde el archivo **companies.csv** que se encuentra en dicha ruta en la carpeta companies. Este archivo tiene los campos separados por comas ','. Los campos de texto no van encerrados por ningún parámetro, y se pide ignorar la primera fila de todos los registros porque corresponde a los títulos de los diferentes campos o columnas.

En este punto, se pasa a crear la Primary Key, ya que no la hemos creado con la tabla, por si daba problemas al cargar datos.

```

ALTER TABLE companies MODIFY COLUMN company_id VARCHAR(20) NOT NULL PRIMARY
KEY;

```

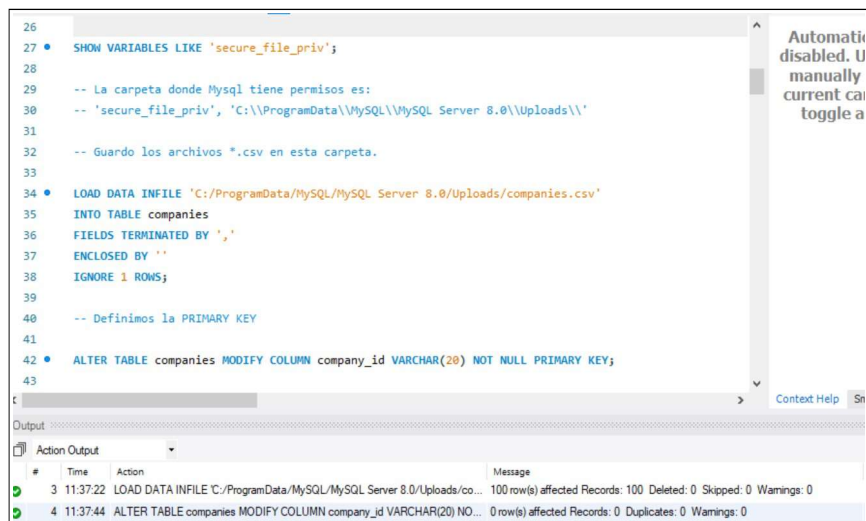


Imagen 3. Captura pantalla de la ejecución de la carga de datos, y creación de PRIMARY KEY en la tabla companies.

Ahora se crea la tabla credit_card, y se pasan a cargar los datos desde el archivo csv. En esta tabla creamos la PRIMARY KEY desde la misma instrucción de creación de tabla.

```
52 • CREATE TABLE IF NOT EXISTS credit_card (  
53     id VARCHAR(50) NOT NULL PRIMARY KEY,  
54     user_id INT NOT NULL,  
55     iban VARCHAR(50),  
56     pan VARCHAR(100),  
57     pin VARCHAR(8),  
58     cvv VARCHAR(8),  
59     track1 VARCHAR(150),  
60     track2 VARCHAR(150),  
61     expiring_date VARCHAR(12)  
62 );  
63  
64 • DESC credit_card;  
65 • SELECT * FROM credit_card;
```

Output

#	Time	Action	Message
10	21:50:29	DESC companies	6 row(s) returned
11	21:50:39	CREATE TABLE IF NOT EXISTS credit_card (id VARCHAR(50) NOT NULL P...	0 row(s) affected

Imagen 4. Creación de la tabla *credit_card*. En la parte inferior de la imagen, se ve que se ha creado correctamente.

```
68 -- Cargamos los datos del csv  
69 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/credit_cards.csv'  
70 INTO TABLE credit_card  
71 FIELDS TERMINATED BY ','  
72 ENCLOSED BY ''  
73 IGNORE 1 ROWS;  
74  
75 • SELECT * FROM credit_card;  
76  
77  
78  
79  
80
```

Output

#	Time	Action	Message
11	21:50:39	CREATE TABLE IF NOT EXISTS credit_card (id VARCHAR(50) NOT NULL P...	0 row(s) affected
12	21:57:05	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/cr...	275 row(s) affected Records: 275 Deleted: 0 Skipped: 0 Warnings: 0

Imagen. Captura de pantalla de la subida de datos en la tabla *credit_card* con código, y con éxito.

*** Este cambio lo realizo después de mostrar el esquema de estrella de mi base de datos. Una vez acabado el ejercicio, paso a cambiar los formatos de date de string a fecha.

```
UPDATE credit_card  
SET expiring_date = STR_TO_DATE(expiring_date_str, '%m/%d/%Y');
```

Después de crear las tablas `companies` y `credit_card`. Se crea la tabla **users**. Una vez creada, se cargan los datos de las tres tablas **users_usa.csv**, **users_ca.csv** y **users_uk**.

```

78 CREATE TABLE IF NOT EXISTS users (
79     id INT NOT NULL PRIMARY KEY,
80     name VARCHAR(20) NOT NULL,
81     surname VARCHAR(50) NOT NULL,
82     phone VARCHAR(20),
83     email VARCHAR(50),
84     birth_date VARCHAR(20),
85     country VARCHAR(50),
86     city VARCHAR(40),
87     postal_code VARCHAR(12),
88     address VARCHAR(120)
89 );
90

```

#	Time	Action	Message
12	21:57:05	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/cr...	275 row(s) affected Rec
13	22:00:17	SELECT * FROM credit_card LIMIT 0, 10000	275 row(s) returned
14	22:00:49	CREATE TABLE IF NOT EXISTS users (id INT NOT NULL PRIMARY KEY, ...	0 row(s) affected

Imagen5. Creación de la tabla users.

```

93 LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users_usa.csv'
94 INTO TABLE users
95 FIELDS TERMINATED BY ','
96 ENCLOSED BY '"'
97 LINES TERMINATED BY '\r\n';
98
99 -- Cargamos archivo users_uk.csv
100 LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users_uk.csv'
101 INTO TABLE users
102 FIELDS TERMINATED BY ','
103 ENCLOSED BY '"'
104 LINES TERMINATED BY '\r\n'
105 IGNORE 1 ROWS;
106
107 -- Cargamos archivo users_ca.csv
108 LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users_ca.csv'
109 INTO TABLE users
110 FIELDS TERMINATED BY ','
111 ENCLOSED BY '"'
112 LINES TERMINATED BY '\r\n'
113 IGNORE 1 ROWS;

```

Imagen.6 Carga de datos en la tabla users.

En la siguiente imagen, se puede ver que las queries relacionadas con la tabla users se han ejecutado con éxito.

✓	14	22:00:49	CREATE TABLE IF NOT EXISTS users (id INT NOT NULL PRIMARY KEY, ...	0 row(s) affected
✓	15	22:14:06	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/us...	150 row(s) affected Records: 150 Deleted: 0 Skipped: 0 Warnings: 0
✓	16	22:14:22	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/us...	50 row(s) affected Records: 50 Deleted: 0 Skipped: 0 Warnings: 0
✓	17	22:14:26	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/us...	75 row(s) affected Records: 75 Deleted: 0 Skipped: 0 Warnings: 0
✓	18	22:14:30	SELECT * FROM users LIMIT 0, 10000	275 row(s) returned

Imagen7. Mensaje de pantalla donde se puede ver que las instrucciones han sido exitosas.

*** Este cambio lo realizo después de mostrar el esquema de estrella de mi base de datos. Una vez acabado el ejercicio, paso a cambiar los formatos de date de string a fecha.

```
UPDATE users
SET birth_date =
CASE WHEN birth_date_str REGEXP '^[A-Za-z]{3} [0-9]{1,2}, [0-9]{4}$'
THEN STR_TO_DATE(birth_date_str, '%b %e, %Y')
ELSE STR_TO_DATE(birth_date_str, '%m/%d/%Y')
END;
```

La última tabla que se crea hasta este punto del ejercicio, es la tabla **transactions**. Como hasta ahora, se crea la tabla, y se suben los datos desde el archivo csv.

```
CREATE TABLE IF NOT EXISTS transactions (
    id VARCHAR(100) NOT NULL PRIMARY KEY,
    card_id VARCHAR(50) NOT NULL,
    bussiness_id VARCHAR(20) NOT NULL,
    dia_hora TIMESTAMP,
    amount DECIMAL(9,2),
    declined TINYINT,
    product_ids VARCHAR(100),
    user_id INT NOT NULL,
    lat DECIMAL(20, 16),
    longitude DECIMAL(20, 16)
);
```

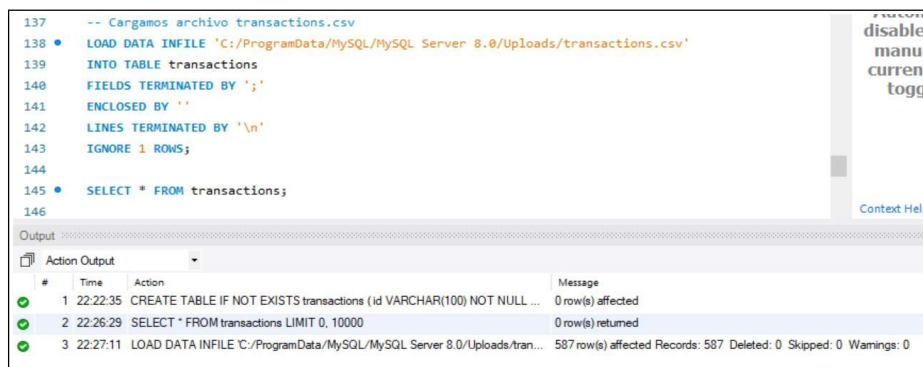


Imagen8. Carga de datos desde archivo csv en la tabla transactions.

Por último, se crean las relaciones entre las diferentes tablas de nuestra base de datos. Se crean las FOREIGN KEYS en la tabla transactions. Estas relacionarán la tabla transacciones con el resto de tablas.

```
ALTER TABLE transactions ADD
CONSTRAINT FOREIGN KEY (card_id) REFERENCES credit_card(id);
ALTER TABLE transactions ADD
CONSTRAINT FOREIGN KEY (user_id) REFERENCES users(id);
ALTER TABLE transactions ADD
CONSTRAINT FOREIGN KEY (bussiness_id) REFERENCES companies(company_id);
```

En este momento, el diagrama de estrella queda define nuestra base de datos es de esta manera. La tabla transacciones es la tabla central, y el resto son las tablas de dimensiones.

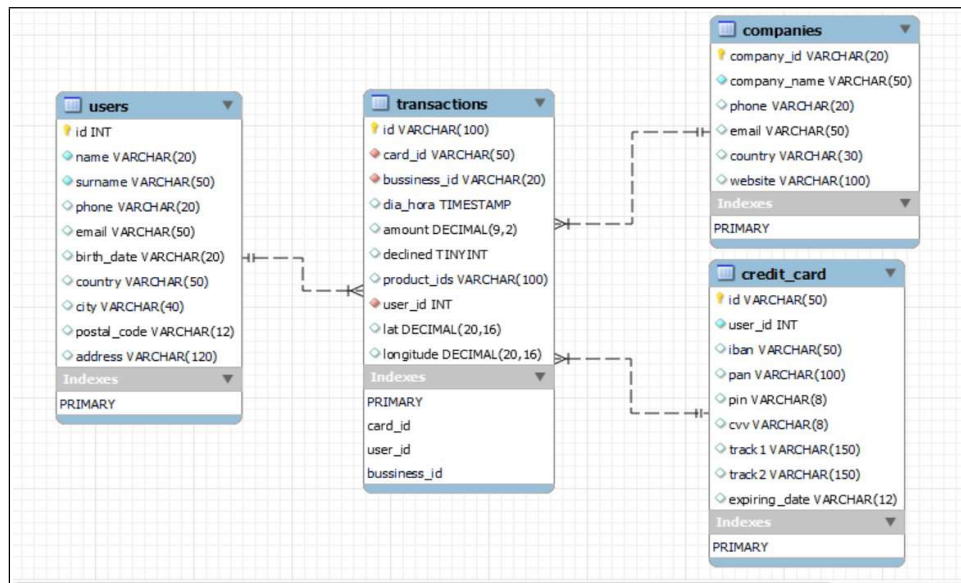


Imagen9. Diagrama en estrella de nuestra base de datos con cuatro tablas: transactions, companies, credit_card y users.

Ejercicio 1

Realiza una subconsulta que muestre a todos los usuarios con más de 30 transacciones utilizando al menos 2 meses.

Primero, se realiza con una join, y después con una subquery

Se seleccionan los campos:

- users.id,
- users.name, users.surname,
- count(transactions.id) as numTrans: se cuentan los ids de transacciones

Se realiza una join de las tablas users y transactions, que son las tablas donde se encuentran los datos que buscamos. Se realiza el conteo de los ids de transactions agrupados por el campo **user.id**.

Por último, le decimos la condición de filtraje que es que el número de ids sea mayor de 30.

```
SELECT users.id, users.name, users.surname, count(transactions.id) AS numTrans
FROM users
JOIN transactions
ON users.id = transactions.user_id
GROUP BY users.id
HAVING count(transactions.id)>30;
```

El resultado de la consulta es el siguiente. Son cuatro registros.

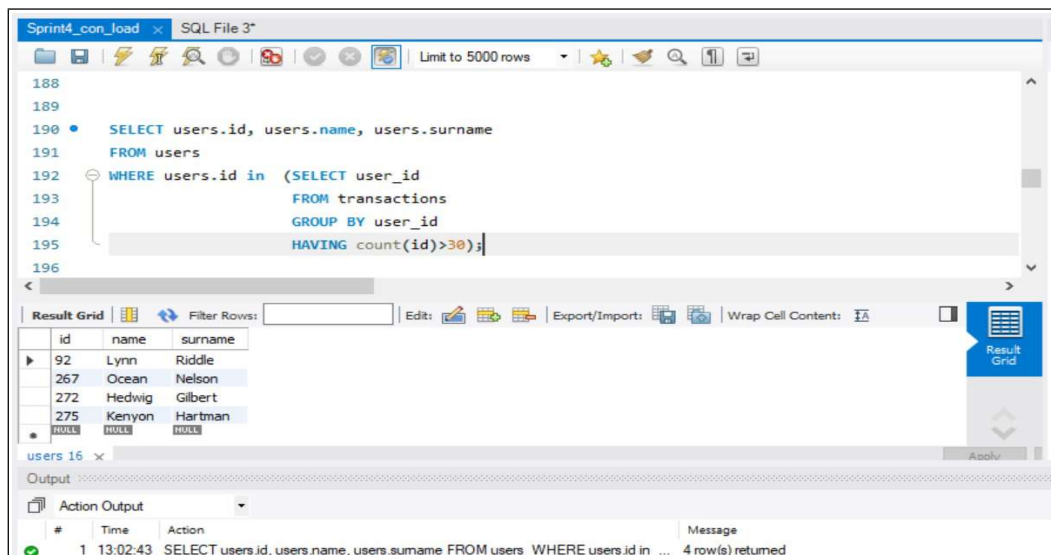


Imagen10. Resultado de la consulta con una subquery . Los registros de usuarios que cumplen que han realizado más de 30 transacciones.

La segunda forma de hacer la consulta, es con una subquery (*Imagen 10*). La subquery, situada en el WHERE representa una tabla donde aparecen los id de usuario que cumplen la condición de tener más de 30 transacciones y el número de transacciones.

```

SELECT users.id, users.name, users.surname
FROM users
WHERE users.id in (SELECT user_id
                  FROM transactions
                  GROUP BY user_id
                  HAVING count(id)>30);

```

Ejercicio 2

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

```

SELECT      companies.company_id,
            companies.company_name,
            credit_card.iban,
            truncate(avg(transactions.amount),2)
FROM credit_card
JOIN transactions
ON transactions.card_id = credit_card.id
JOIN companies
ON companies.company_id = transactions.business_id
WHERE company_name = 'Donec Ltd'
GROUP BY companies.company_id, companies.company_name, credit_card.iban;

```


The screenshot shows a SQL IDE window titled "Sprint4_con_load" and "SQL File 3". The query editor contains the following SQL code:

```

206 SELECT companies.company_id,
207        companies.company_name,
208        credit_card.iban,
209        truncate(avg(transactions.amount),2)
210 FROM credit_card
211 JOIN transactions
212 ON transactions.card_id = credit_card.id
213 JOIN companies
214 ON companies.company_id = transactions.bussiness_id
215 WHERE company_name = 'Donec Ltd'
216 GROUP BY companies.company_id, companies.company_name, credit_card.iban;
217
218 -- NIVEL 2
219 -- Crea una nueva tabla que refleje el estado de las tarjetas de credito basado en si las ultimas
220 -- tres transacciones fueron declinadas, y genera la siguiente consulta:

```

Below the query editor, the "Result Grid" shows the following data:

company_id	company_name	iban	truncate(avg(transactions.amount),2)
b-2242	Donec Ltd	PT87806228135092429456346	203.71

The "Output" pane shows the execution log:

#	Time	Action	Message
2	13:15:30	SELECT companies.company_id, companies.company_name, credit_card.iba...	1 row(s) returned
3	13:17:45	SELECT companies.company_id, companies.company_name, credit_card.iba...	1 row(s) returned

Imagen11. Resultado de la consulta del ejercicio 2 del nivel 1.

Se seleccionan los campos requeridos que cumplen la condición de **company_name = 'Donec Ltd'**. Como piden la media del amount, se aplica la función de agregación **average**. Todo agrupado por id de la company.

NIVEL 2

Crea una nueva tabla que refleje el estado de las tarjetas de crédito basado en si las últimas tres transacciones fueron declinadas y genera la siguiente consulta:

```
222
223 • CREATE TABLE estado_tarjeta
224 SELECT card_id, CASE WHEN sum(declined)<3 THEN 'ACTIVA'
225 ELSE 'NO ACTIVA'
226 END as estado_tarjeta
227 FROM (SELECT card_id, dia_hora, declined
228 FROM (SELECT card_id, dia_hora, declined,
229 ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY dia_hora DESC) as row_num
230 FROM transactions) as numtranstarjetas
231 WHERE row_num <= 3
232 ORDER BY card_id, dia_hora DESC) ordenfecha
233 GROUP BY card_id;
234
```

Imagen 12. Consulta que da solución al ejercicio.

En esta consulta, se seleccionan ordenan las fechas de manera descendiente con la función **ROW_NUMBER()**, agrupadas por la **card_id**. Primero aparecen las fechas de una tarjeta, después la siguiente, etc. La columna **row_number** numera de 1 a N los diferentes registros de cada tarjeta de crédito, ordenadas de manera descendiente.

card_id	dia_hora	declined	row_num
CcU-2938	2021-07-18 08:20:59	0	13
CcU-2938	2021-07-11 00:19:27	0	14
CcU-2938	2021-07-07 17:43:16	0	15
CcU-2938	2021-07-03 19:56:27	0	16
CcU-2938	2021-05-28 15:21:36	0	17
CcU-2938	2021-05-19 01:05:28	0	18
CcU-2938	2021-05-09 10:25:08	1	19
CcU-2938	2021-04-18 11:05:12	0	20
CcU-2938	2021-04-09 19:23:41	0	21
CcU-2938	2021-04-01 07:27:49	0	22
CcU-2938	2021-03-28 05:01:44	0	23
CcU-2938	2021-03-23 01:12:06	0	24
CcU-2945	2022-02-04 15:52:56	0	1
CcU-2945	2021-06-15 00:26:29	1	2
CcU-2952	2022-01-30 15:16:36	0	1
CcU-2952	2021-05-06 05:33:39	1	2
CcU-2959	2022-03-16 14:01:36	0	1
CcU-2959	2022-03-04 02:48:32	0	2

Imagen 13. Resultado de la aplicar la función ROW_COLUMN.

Después se seleccionan las tres fechas más recientes de cada tarjeta.

card_id	dia_hora	declined
CcU-2938	2022-03-12 09:23:10	0
CcU-2938	2022-03-09 20:53:59	0
CcU-2938	2022-02-24 11:01:42	0
CcU-2945	2022-02-04 15:52:56	0
CcU-2945	2021-06-15 00:26:29	1
CcU-2952	2022-01-30 15:16:36	2022-01-30 1
CcU-2952	2021-05-06 05:33:39	1
CcU-2959	2022-03-16 14:01:36	0
CcU-2959	2022-03-04 02:48:32	0

Imagen14. Resultado de la aplicar la condición siguiente: ROW_COLUMN>=3.

Por último, se suman los diferentes valores de declined de los tres registros de cada **car_id**. En este caso con la función **CASE WHEN**, si la suma es menor de tres, la tarjeta está **ACTIVA**, y sino está **INACTIVA**. El resultado del **SELECT** se agrupa **card_id** porque contiene una función de agregación **SUM()**.

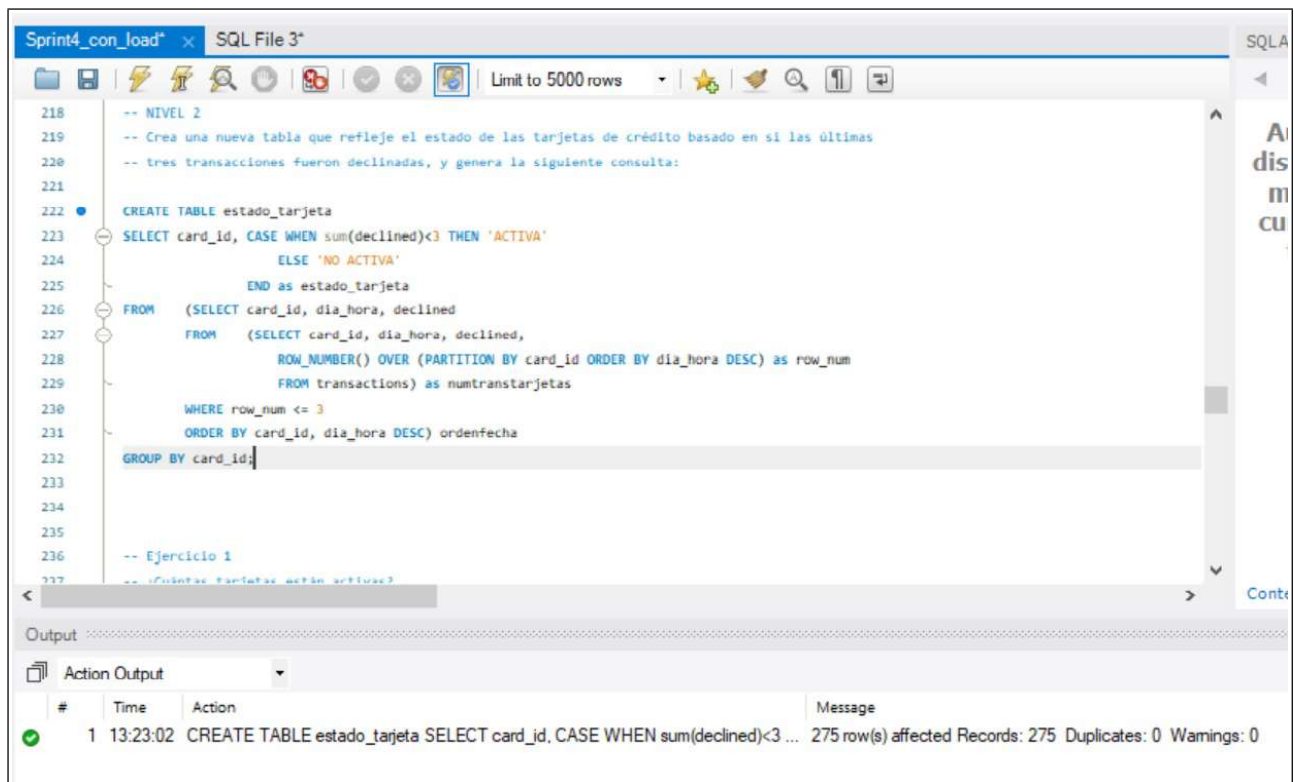


Imagen15. Resultado de la consulta del nivel 2.

Ejercicio 1

¿Cuántas tarjetas están activas?

Para encontrar el resultado, unicamente se añade una SELECT que englobe la consulta anterior, donde se solicita que se cuente el número de registros que tienen la condición ACTIVA en estado_tarjeta.

```

SELECT count(*) AS tarjetas_activas
FROM estado_tarjeta
WHERE estado_tarjeta = 'ACTIVA';

```

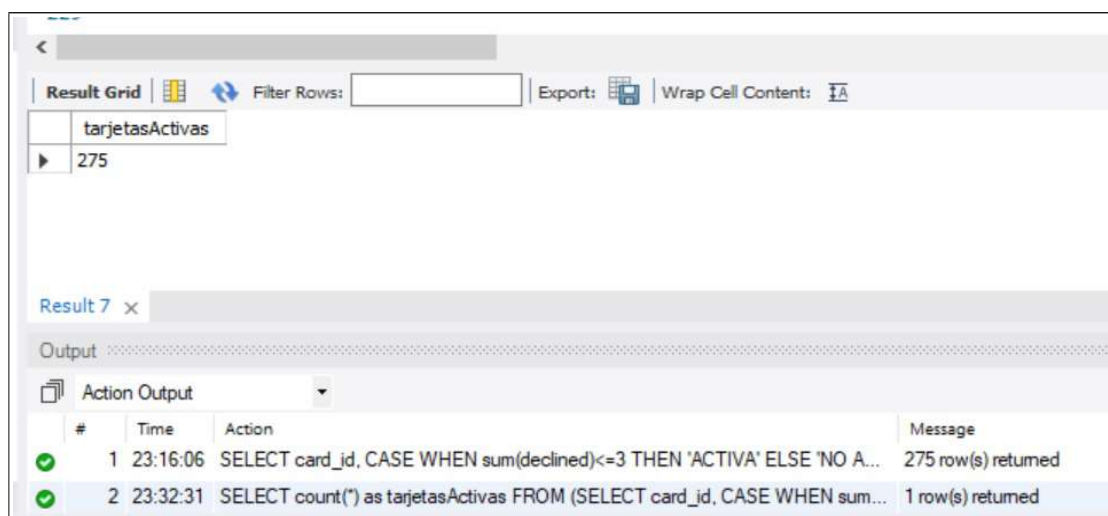


Imagen16. Resultado de la consulta.

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 transaction tienes product_ids. Genera la siguiente consulta:

Para acabar nuestra base de datos se añade la tabla de **productos**. Para añadirla, en el archivo csv, se modifica el formato del precio eliminando el símbolo de la moneda.

Se crea la tabla con sus correspondientes campos.

```
CREATE TABLE IF NOT EXISTS products (  
    id INT NOT NULL PRIMARY KEY,  
    product_name VARCHAR(50),  
    price DECIMAL(8,2),  
    colour VARCHAR(15),  
    weight DECIMAL(5,1),  
    warehouse_id VARCHAR(10)  
);
```

Una vez creada la tabla, se cargan los datos desde el archivo products.csv con el siguiente código.

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/products.csv'  
INTO TABLE products  
FIELDS TERMINATED BY ','  
ENCLOSED BY "  
LINES TERMINATED BY "\n"  
IGNORE 1 ROWS;
```

Ahora se busca la relación entre la tabla transacciones, y la tabla productos. La relación entre ambas tablas es de N:M. Para romper esta relación se debe crear una tabla nueva intermedia que nos sirva para relacionar ambas tablas. Esta nueva tabla se llamará **tiquets**. Tendrá las columnas transactions_id relacionada con los diferentes ids de productos que contiene cada transacción.

En la tabla transacciones, existe una columna o campo que contiene los diferentes códigos de productos que forman parte de una transacción. Se deben separar estos ids con su correspondiente id de transacción en diferentes registros en la nueva tabla tiquets.

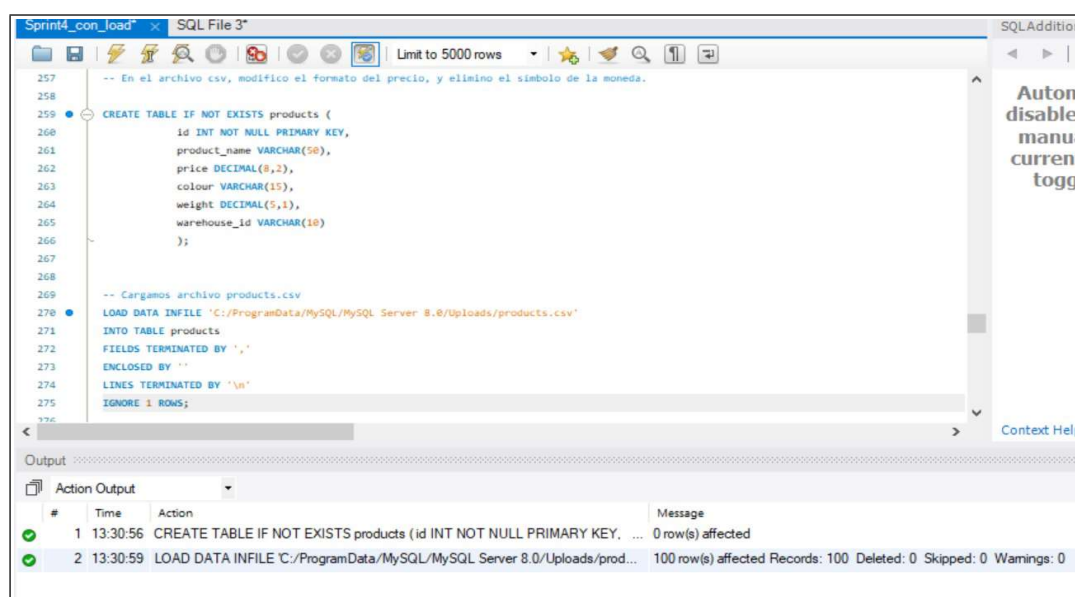


Imagen17. Pantallazo del código de creación de la tabla products y la carga de los datos desde el archivo csv, de manera satisfactoria.

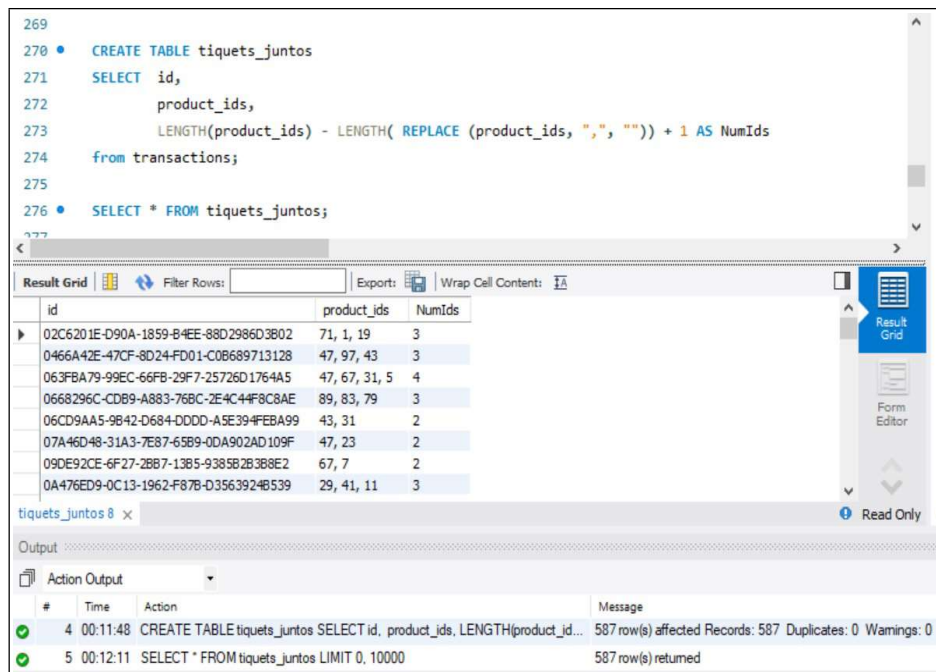
1) Primero creamos una tabla auxiliar intermedia.

Importamos las dos columnas que nos interesan, de la tabla transacciones. Creamos una tercera columna donde se vea el número de ids que contiene la columna dos. La tabla contiene tres columnas.

- id_transactions
- product_ids
- el número de ids de la columna 2.

```
CREATE TABLE tiquets_juntos
SELECT      id,
            product_ids,
            LENGTH(product_ids) - LENGTH( REPLACE (product_ids, ",", "")) + 1 AS NumIds
FROM transactions;
```

En este punto, se deben separar los diferentes ids, una fila por cada id. Después será la tabla que uniremos a productos y transacciones.



id	product_ids	NumIds
02C6201E-D90A-1859-84EE-88D2986D3B02	71, 1, 19	3
0466A42E-47CF-8D24-FD01-C0B689713128	47, 97, 43	3
063FBA79-99EC-66FB-29F7-25726D1764A5	47, 67, 31, 5	4
0668296C-CDB9-A883-76BC-2E4C44F8C8AE	89, 83, 79	3
06CD9AA5-9B42-D684-DDDD-ASE394FEBA99	43, 31	2
07A46D48-31A3-7E87-65B9-0DA902AD109F	47, 23	2
09DE92CE-6F27-2B87-13B5-9385B2B388E2	67, 7	2
0A476ED9-0C13-1962-F87B-D3563924B539	29, 41, 11	3

#	Time	Action	Message
4	00:11:48	CREATE TABLE tiquets_juntos SELECT id, product_ids, LENGTH(product_id...	587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0
5	00:12:11	SELECT * FROM tiquets_juntos LIMIT 0, 10000	587 row(s) returned

Imagen18. Captura de pantalla del contenido de la tabla tiquets_juntos.

```
CREATE TABLE tiquets
SELECT id,
       SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', n.digit), ',', -1) AS product
FROM tiquets_juntos
JOIN (
  SELECT 1 AS digit UNION ALL
  SELECT 2 UNION ALL
  SELECT 3 UNION ALL
  SELECT 4
) AS n
ON NumIds >= n.digit;
```

En la subconsulta, se indican las posiciones de cada valor de la lista de product_ids.

Con la función SUBSTRING_INDEX, se seleccionan cada valor según la posición indicada como digit.

Se repetirá el proceso para cada fila, separando todos los valores de product_ids, una fila por cada valor.

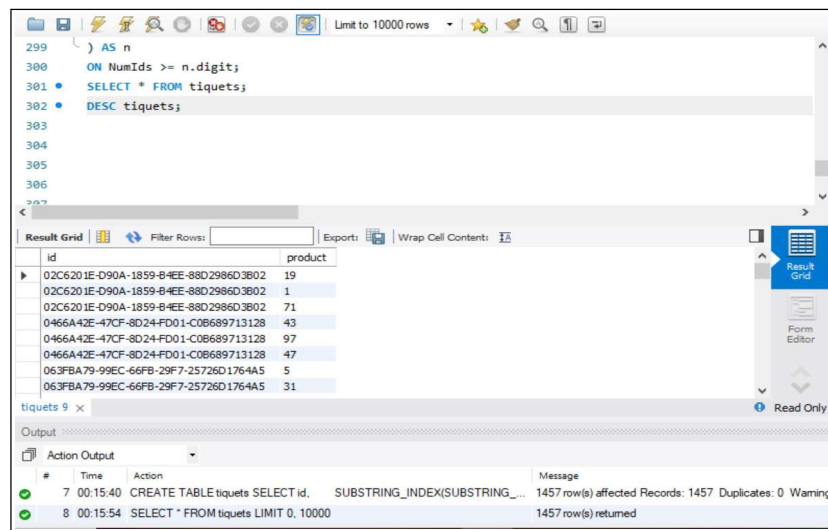


Imagen19. Captura de pantalla del contenido de la tabla tickets.

A continuación una segunda manera de obtener la tabla, pero menos dinámica.

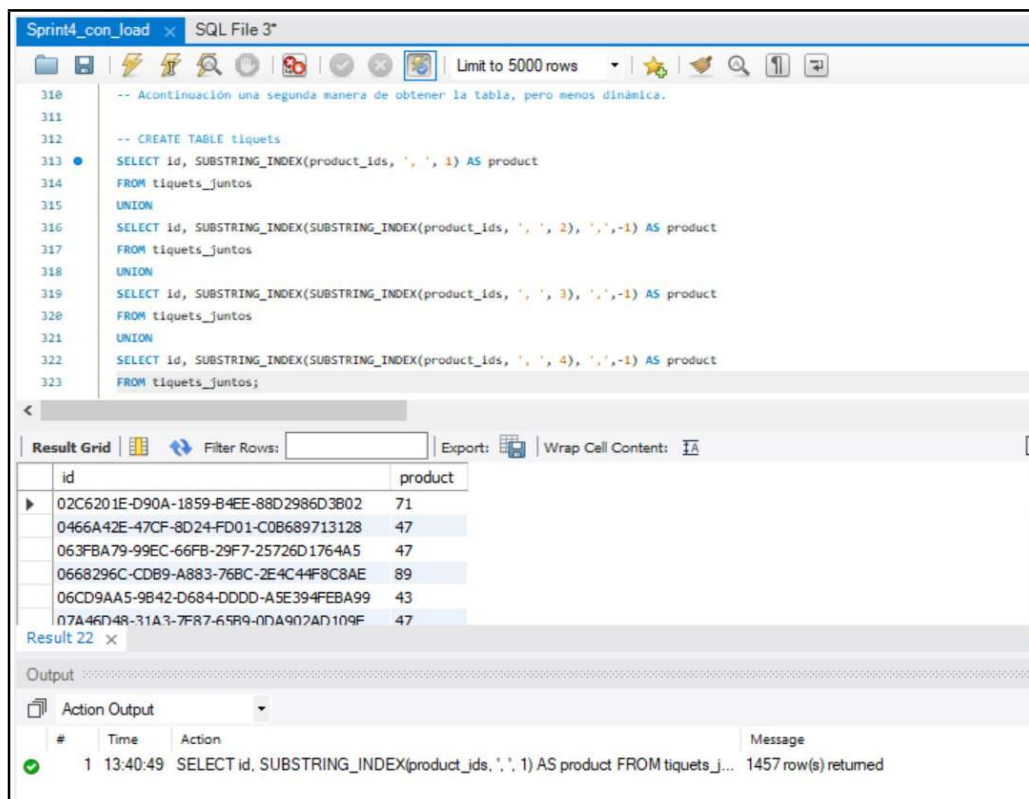


Imagen20. Captura de pantalla del contenido de la tabla tickets realizada desde la consulta menos dinámica.

Ahora lo que hacemos es modificar el tipo de dato de product, de VARCHAR a INTEGER.

`ALTER TABLE tickets MODIFY COLUMN product INT NOT NULL;`

Por último, creamos las FOREIGN KEYS en la tabla tickets para relacionar ésta con la tabla de transactions y products.

```
ALTER TABLE tickets ADD
CONSTRAINT FOREIGN KEY (id) REFERENCES transactions(id);
ALTER TABLE tickets ADD
CONSTRAINT FOREIGN KEY (product) REFERENCES products(id);
```

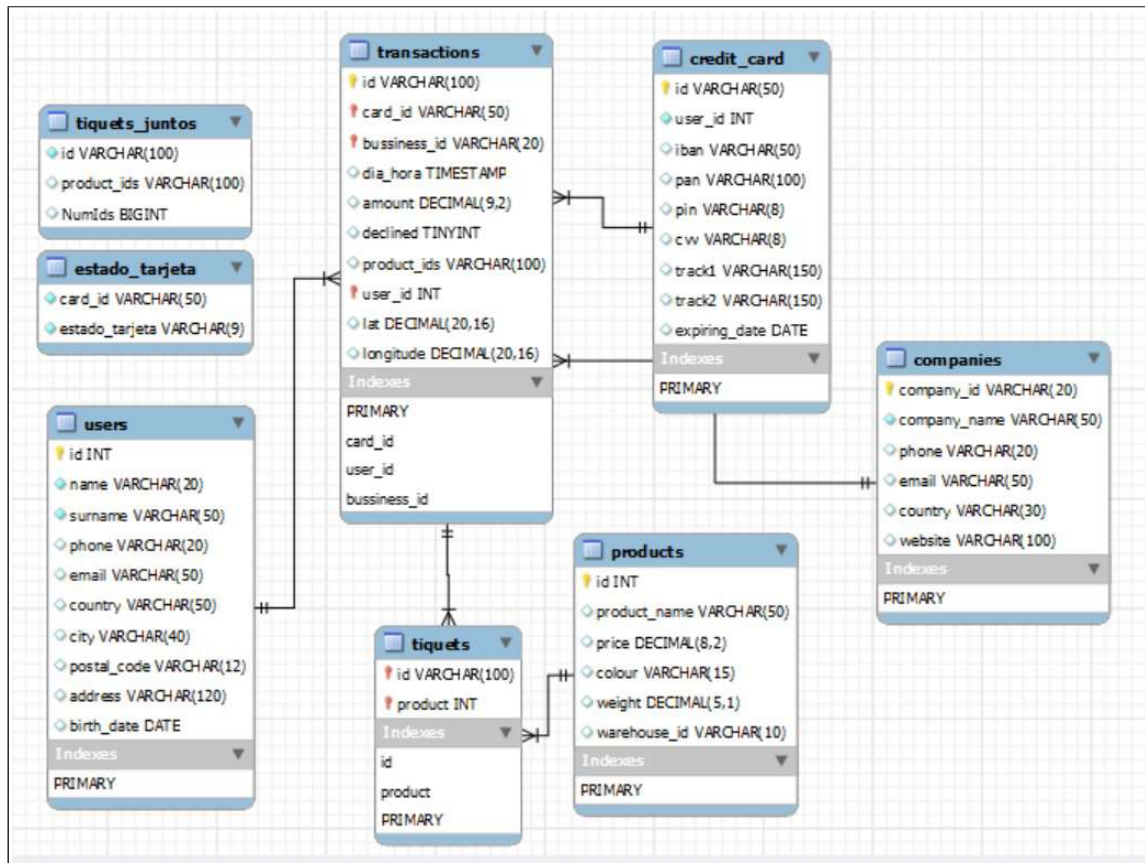


Imagen21. Diagrama final de nuestra base de datos *Operaciones*.

Ejercicio 1

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

```
SELECT tickets.product, products.product_name, count(tickets.product) AS num
FROM tickets
JOIN products
ON products.id = tickets.product
JOIN transactions
ON transactions.id = tickets.id
WHERE transactions.declined = 0
GROUP BY tickets.product, products.product_name
ORDER BY num DESC;
```

Para encontrar los productos vendidos, se seleccionan el campo product_name de la tabla products, el campo id de ticket y el conteo de los ids de producto, Como los datos a listar están en dos tablas diferentes, se hace un JOIN de las dos tablas para poder relacionar los datos. En este caso, como se piden las ventas, se debe filtrar por los declined efectivos, que son igual a cero. El

declined pertenece a la tabla transactions. Por ese motivo, se hace un JOIN con la tabla transactions. Por último, se debe agrupar la selección por id de tiquet y nombre de producto, al haber una función de agregación en la SELECT.

Para acabar se ordena por el campo conteo de id, que representa el número de veces que se ha vendido un producto, de manera descendiente

```
326 • SELECT tickets.product, products.product_name, count(tickets.product) AS num
327 FROM tickets
328 JOIN products
329 ON products.id = tickets.product
330 JOIN transactions
331 ON transactions.id = tickets.id
332 WHERE transactions.declined = 0
333 GROUP BY tickets.product, products.product_name
334 ORDER BY num DESC;
```

Result Grid

	product	product_name	num
▶	23	riverlands north	60
	67	Winterfell	59
	2	Tarly Stark	56
	43	duel	54
	17	skywalker ewok sith	54
	97	jinn Winterfell	53
	79	Direwolf riverlands the	52

Result 1 x

Output

Action Output

#	Time	Action	Message
✓ 1	00:22:42	SELECT tickets.product, products.product_name, count(tickets.product) AS nu...	26 row(s) returned

Imagen22. Resultado del ejercicio 1 del Nivel 3. El número de productos vendidos es de 26.