

## DESCRIPCIÓN

Partiendo de algunos archivos CSV diseñarás y crearás tu base de datos.

## NIVEL I

Descarga los archivos CSV, estúdialos 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:

Analizando los archivos CSV, pude diseñar una base de datos con un esquema estrella de la siguiente manera:

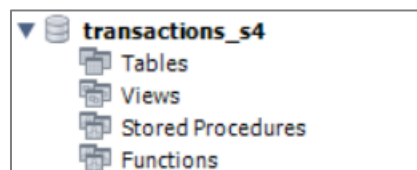
### PASO 1 = Creación de base de datos

```
• CREATE DATABASE IF NOT EXISTS transactions_S4;  
• USE transactions_S4;
```

✓	4	17:09:16	CREATE DATABASE IF NOT EXISTS transactions_S4	1 row(s) affected
✓	5	17:09:16	USE transactions_S4	0 row(s) affected

- `CREATE DATABASE IF NOT EXISTS transactions_S4`: declaración de que crea una base de datos llamada "Transaction\_S4".
- `USE transactions_S4`: en esta declaración indicamos que a partir de ahora queremos trabajar con la base de datos "Transaction\_S4".

Una vez ejecutadas ambas declaraciones, podemos corroborar que la base de datos ya fue creada y seleccionada:



### PASO 2: Creación de la estructura

En este paso voy a crear la estructura de las siguientes tablas:

- Tabla companies
- Tabla credit\_cards
- Tabla users
- Tabla products
- Tabla transactions

a) TABLA COMPANIES

```
8      -- Creación de la tabla companies
9      CREATE TABLE IF NOT EXISTS companies (
10         company_id VARCHAR(20) PRIMARY KEY,
11         company_name VARCHAR(255),
12         phone VARCHAR(15),
13         email VARCHAR(100),
14         country VARCHAR(100),
15         website VARCHAR(255)
16     );
```

6 11:53:06 CREATE TABLE IF NOT EXISTS companies ( company\_id VARCHAR(20) PRIMARY KEY, c... 0 row(s) affected

- **CREATE TABLE IF NOT EXISTS companies:** declaración que indica la creación de una tabla en la base de datos, con la condición de que dicha tabla no existe. Luego de este primer parte, coloqué el nombre de la nueva tabla a crear, en este caso es “companies”.
- **CREACION DE CAMPOS:** luego de la declaración, procedí a definir los campos que compondrán la tabla, teniendo en cuenta los datos que se encuentran en el archivo ‘companies.csv’. Los campos creados con sus especificaciones son las siguientes:
  - **Company\_id VARCHAR (20) PRIMARY KEY:** identificador único de la compañía, con la restricción PRIMARY KEY para asegurar la unicidad y evitar valores null. Es de tipo VARCHAR con una longitud de hasta 20 caracteres, permitiendo letras, números y caracteres especiales.
  - **Company\_name VARCHAR (255):** Campo que almacena el nombre de la compañía. Lo definí como un tipo de dato VARCHAR con una longitud de 255 caracteres que pueden contener letras, números y caracteres especiales.
  - **Phone VARCHAR (15):** corresponde al número de teléfono de la compañía. Lo definí como un tipo de dato VARCHAR con una longitud de 15 caracteres que pueden contener letras, números y caracteres especiales.
  - **Email VARCHAR (100):** corresponde a la dirección de correo electrónico de la compañía y lo definí como un tipo de dato VARCHAR con una longitud de 100 caracteres que pueden contener letras, números y caracteres especiales.
  - **Country VARCHAR (100):** País donde está ubicada la compañía y lo definí como un tipo de dato VARCHAR con una longitud de 100 caracteres que pueden contener letras, números y caracteres especiales.
  - **Website VARCHAR (255):** campo que contiene la página web de la compañía. Lo definí como VARCHAR con una longitud de 255 caracteres que pueden contener letras, números y caracteres especiales.

b) TABLA CREDIT\_CARDS

```
18  -- Creación de la tabla credit_cards
19  CREATE TABLE IF NOT EXISTS credit_cards (
20      id VARCHAR(20) PRIMARY KEY,
21      user_id INT,
22      iban VARCHAR(50),
23      pan VARCHAR (30),
24      pin VARCHAR (4),
25      cvv VARCHAR (4),
26      track1 VARCHAR (100),
27      track2 VARCHAR (100),
28      expiring_date VARCHAR (30)
29  );
```

7 11:53:31 CREATE TABLE IF NOT EXISTS credit\_cards (id VARCHAR(20) PRIMARY KEY, user\_id INT, iba... 0 row(s) affected

- **CREATE TABLE IF NOT EXISTS credit\_cards:** declaración que indica la creación de una tabla en la base de datos, con la condición de que dicha tabla no existe. Luego de este primer parte, coloque el nombre de la nueva tabla a crear, en este caso es “credit\_cards”.
- **CREACION DE CAMPOS:** luego de la declaración, procedí a definir los campos que compondrán la tabla, teniendo en cuenta la información proporcionada en el archivo “credit\_cards.csv”. Los campos creados con sus especificaciones son las siguientes:
  - **id VARCHAR (20) PRIMARY KEY:** identificador único para cada tarjeta de crédito, con la restricción PRIMARY KEY para asegurar la unicidad y evitar valores null. Es de tipo VARCHAR con una longitud de hasta 20 caracteres, permitiendo letras, números y caracteres especiales.
  - **User\_id INT:** identificador único del usuario y lo definí como un tipo de dato de un valor entero (INT).
  - **iban VARCHAR (50):** (International Bank Account Number). Campo que almacena el número de cuenta internacional asociada la tarjeta. Lo definí como un tipo de dato VARCHAR con una longitud de 50 caracteres que pueden contener letras, números y caracteres especiales.
  - **pan VARCHAR (30):** (Primary Account Number de la tarjeta). Corresponde al número principal de la cuenta. Lo definí como un tipo de dato VARCHAR con una longitud de 30 caracteres que pueden contener letras, números y caracteres especiales.
  - **pin VARCHAR (4):** corresponde al PIN de la tarjeta y lo definí como un tipo de dato VARCHAR con una longitud de 4 caracteres que pueden contener letras, números y caracteres especiales.
  - **cvv VARCHAR (4):** código de verificación de la tarjeta y lo definí como un tipo de dato VARCHAR con una longitud de 4 caracteres que pueden contener letras, números y caracteres especiales.
  - **Track1 VARCHAR (100):** campo que almacena datos de las pistas magnéticas de las tarjetas como, por ejemplo: numero de la tarjeta, nombre del titular de la tarjeta, etc. Lo definí como un tipo de dato VARCHAR con una longitud de 100 caracteres que pueden contener letras, números y caracteres especiales.
  - **Track2 VARCHAR (100):** campo que almacena datos de las pistas magnéticas de las tarjetas como, por ejemplo: un código de servicio, etc. Lo definí como un tipo de dato VARCHAR con una longitud de 100 caracteres que pueden contener letras, números y caracteres especiales.
  - **expiring\_date VARCHAR (30):** campo que contiene la fecha de expiración de la tarjeta. Temporalmente, lo definí como VARCHAR (30) para admitir los datos de fecha que no coinciden

con el formato DATE de SQL, con la intención de insertar inicialmente los datos y luego transformarlos y ajustar el tipo de dato a DATE cuando la estructura de los registros esté adaptada. (esto se realizará al final del script en la sección “Actualización De Formato Y Tipo De Datos Pendientes”).

c) TABLA USERS

```
31 -- Creación de la tabla users
32 CREATE TABLE IF NOT EXISTS users (
33     id INT PRIMARY KEY,
34     name VARCHAR(100),
35     surname VARCHAR(100),
36     phone VARCHAR(150),
37     email VARCHAR(150),
38     birth_date VARCHAR(100),
39     country VARCHAR(150),
40     city VARCHAR(150),
41     postal_code VARCHAR(100),
42     address VARCHAR(255)
43 );
```

8 11:53:48 CREATE TABLE IF NOT EXISTS users ( id INT PRIMARY KEY, name VARCHAR(100), ... 0 row(s) affected

- **CREATE TABLE IF NOT EXISTS users:** declaración que indica la creación de una tabla en la base de datos, con la condición de que dicha tabla no existe. Luego de este primer parte, coloque el nombre de la nueva tabla a crear, en este caso es “users”.
- **CREACION DE CAMPOS:** luego de la declaración, procedí a definir los campos que compondrán la tabla, teniendo en cuenta los datos que se encuentran en el archivo ‘users.csv’. Los campos creados con sus especificaciones son las siguientes:
  - **id INT PRIMARY KEY:** identificador único del usuario, con la restricción PRIMARY KEY para asegurar la unicidad y evitar valores null. Es de tipo de valor entero (INT).
  - **name VARCHAR (100):** Campo que almacena el nombre del usuario. Lo definí como un tipo de dato VARCHAR con una longitud de 100 caracteres que pueden contener letras, números y caracteres especiales.
  - **surname VARCHAR (100):** Campo que almacena el apellido del usuario. Lo definí como un tipo de dato VARCHAR con una longitud de 100 caracteres que pueden contener letras, números y caracteres especiales.
  - **Phone VARCHAR (150):** corresponde al número de teléfono del usuario. Lo definí como un tipo de dato VARCHAR con una longitud de 150 caracteres que pueden contener letras, números y caracteres especiales.
  - **Email VARCHAR (150):** corresponde a la dirección de correo electrónico del usuario y lo definí como un tipo de dato VARCHAR con una longitud de 150 caracteres que pueden contener letras, números y caracteres especiales.

- **Birth\_date VARCHAR (100):** campo que almacena la fecha de nacimiento del usuario. Lo definí como un tipo de dato VARCHAR con una longitud de 100 caracteres que pueden contener letras, números y caracteres especiales.
- **Country VARCHAR (150):** País donde está ubicado el usuario y lo definí como un tipo de dato VARCHAR con una longitud de 150 caracteres que pueden contener letras, números y caracteres especiales.
- **City VARCHAR (150):** ciudad donde está ubicado el usuario y lo definí como un tipo de dato VARCHAR con una longitud de 150 caracteres que pueden contener letras, números y caracteres especiales.
- **Postal\_code VARCHAR (100):** código postal de donde vive el usuario y lo definí como un tipo de dato VARCHAR con una longitud de 100 caracteres que pueden contener letras, números y caracteres especiales.
- **Address VARCHAR (255):** campo que contiene la dirección donde vive el usuario. Lo definí como VARCHAR con una longitud de 255 caracteres que pueden contener letras, números y caracteres especiales.

d) TABLA PRODUCTS

```
45      -- Creación de la tabla products
46 • CREATE TABLE IF NOT EXISTS products(
47     id INT PRIMARY KEY,
48     product_name VARCHAR(50),
49     price DECIMAL(10, 2),
50     colour VARCHAR(50),
51     weight DECIMAL(3, 2),
52     warehouse_id VARCHAR(30)
53 );
```

9 11:55:19 CREATE TABLE IF NOT EXISTS products( id INT PRIMARY KEY, product\_name VARCHAR(50), pri... 0 row(s) affected

- **CREATE TABLE IF NOT EXISTS products:** declaración que indica la creación de una tabla en la base de datos, con la condición de que dicha tabla no existe. Luego de esta primera parte, coloque el nombre de la nueva tabla a crear, en este caso es “products”.
- **CREACION DE CAMPOS:** luego de la declaración, procedí a definir los campos que compondrán la tabla, teniendo en cuenta los datos que se encuentran en el archivo ‘products.csv’. Los campos creados con sus especificaciones son las siguientes:
  - **id INT PRIMARY KEY:** identificador único del producto, con la restricción PRIMARY KEY para asegurar la unicidad y evitar valores null. Es de tipo de valor entero (INT).
  - **product\_name VARCHAR (50):** Campo que almacena el nombre del producto. Lo definí como un tipo de dato VARCHAR con una longitud de 50 caracteres que pueden contener letras, números y caracteres especiales.
  - **Price DECIMAL (10,2):** corresponde al precio del producto. Lo definí como un tipo de dato DECIMAL para definir un número decimal de manera precisa, en este caso sería 10 especifica el número total de dígitos que se pueden almacenar en esta columna y 2 define la cantidad de dígitos que se pueden almacenar después del punto decimal.

- **Colour VARCHAR (50):** campo que almacena los colores del producto y lo definí como un tipo de dato VARCHAR con una longitud de 50 caracteres que pueden contener letras, números y caracteres especiales.
- **Weight DECIMAL (3,2):** corresponde al peso del producto. Lo definí como un tipo de dato DECIMAL para definir un número decimal de manera precisa, en este caso sería 3 especifica el número total de dígitos que se pueden almacenar en esta columna y 2 define la cantidad de dígitos que se pueden almacenar después del punto decimal.
- **Warehouse\_id VARCHAR (30):** campo que contiene el centro de distribución del producto. Lo definí como VARCHAR con una longitud de 30 caracteres que pueden contener letras, números y caracteres especiales.

e) TABLA TRANSACTIONS

```
CREATE TABLE IF NOT EXISTS transactions (  
  id VARCHAR(255) PRIMARY KEY,  
  card_id VARCHAR(20),  
  business_id VARCHAR(20),  
  timestamp TIMESTAMP,  
  amount DECIMAL(10, 2),  
  declined BOOLEAN,  
  product_ids VARCHAR (200),  
  user_id INT,  
  lat FLOAT,  
  longitude FLOAT,  
  
  FOREIGN KEY (card_id) REFERENCES credit_cards(id),  
  FOREIGN KEY (business_id) REFERENCES companies(company_id),  
  FOREIGN KEY ( user_id ) REFERENCES users (id)  
);
```

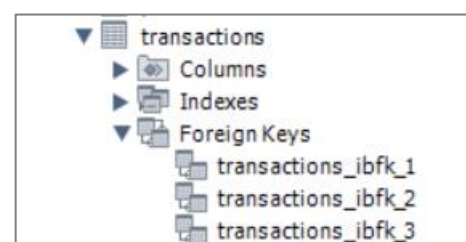
10 11:56:18 CREATE TABLE IF NOT EXISTS transactions ( id VARCHAR(255) PRIMARY KEY, card\_id VARCHA... 0 row(s) affected

- **CREATE TABLE IF NOT EXISTS transactions:** declaración que indica la creación de una tabla en la base de datos, con la condición de que dicha tabla no existe. Luego de este primer parte, coloque el nombre de la nueva tabla a crear, en este caso es “transactions”.
- **CREACION DE CAMPOS:** luego de la declaración, procedí a definir los campos que compondrán la tabla, teniendo en cuenta los datos que se encuentran en el archivo ‘transactions.csv’. Los campos creados con sus especificaciones son las siguientes:
  - **id VARCHAR (255) PRIMARY KEY:** identificador único de la transacción, con la restricción PRIMARY KEY para asegurar la unicidad y evitar valores null. Es de tipo VARCHAR con una longitud de hasta 255 caracteres, permitiendo letras, números y caracteres especiales.
  - **Card\_id VARCHAR (20):** Campo que almacena Identificador único de la tarjeta de crédito utilizada para la transacción. Lo definí como un tipo de dato VARCHAR con una longitud de 20 caracteres que pueden contener letras, números y caracteres especiales.

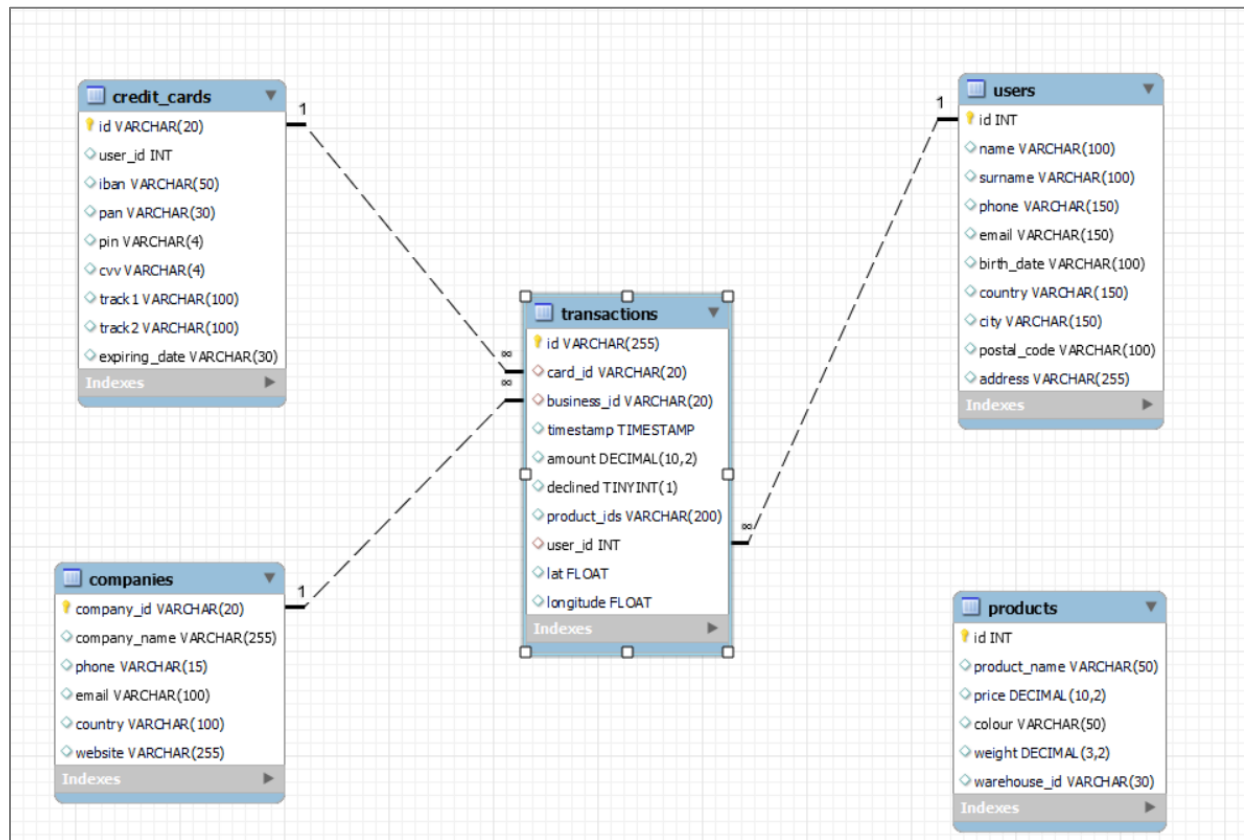


- **Business\_id VARCHAR (20):** identificador único de la compañía. Lo definí como un tipo de dato VARCHAR con una longitud de 20 caracteres que pueden contener letras, números y caracteres especiales.
  - **Timestamp TIMESTAMP:** campo que almacena fechas y horas de las transacciones. Le corresponde el tipo de dato TIMESTAMP.
  - **Amount DECIMAL (10,2):** corresponde al monto total de transacción. Lo definí como un tipo de dato DECIMAL para definir un número decimal de manera precisa, en este caso sería 10 especifica el número total de dígitos que se pueden almacenar en esta columna y 2 define la cantidad de dígitos que se pueden almacenar después del punto decimal.
  - **Declined BOOLEAN:** indicador booleano que muestra si la transacción fue aprobada (0) o rechazada (1).
  - **Products\_ids VARCHAR (200):** identificador único de los productos que participan en la transacción. Lo definí como un tipo de dato VARCHAR con una longitud de 200 caracteres que pueden contener letras, números y caracteres especiales, ya que los registros que contiene la columna se encuentran en forma de lista.
  - **User\_id INT:** identificador único del usuario. Es de tipo de valor entero (INT).
  - **Lat FLOAT:** Latitud. Ubicación de donde se realizó la transacción. El tipo de dato que almacena permite números que pueden tener decimales.
  - **Longitude FLOAT:** Longitud. Ubicación de donde se realizó la transacción. El tipo de dato que almacena permite números que pueden tener decimales.
- **CREACION DE LAS FOREIGN KEY :**
- **FOREIGN KEY (card\_id) REFERENCES credit\_cards (id):** función que establece que el campo card\_id de la tabla *transactions* debe coincidir con el campo id de la tabla *credit\_cards*.
  - **FOREIGN KEY (business\_id) REFERENCES companies(company\_id):** función que establece que el campo business\_id de la tabla *transactions* debe coincidir con el campo company\_id de la tabla *companies*.
  - **FOREIGN KEY ( user\_id ) REFERENCES users (id):** función que establece que el campo user\_id de la tabla *transactions* debe coincidir con el campo id de la tabla *users*.

A continuación, podemos ver que la creación de dichas tablas aparece en la base de datos “Transactions\_S4” y que en la tabla “Transactions” también se encuentran creadas sus FOREIGN KEY:



### PASO 3: MODELO RELACIONAL



Como podemos ver en la imagen, este modelo de datos corresponde a un modelo denominado “Estrella”. Este tipo de modelo organiza la estructura de la base de datos en dos (2) tipos principales de tablas: Tabla de Dimensiones y Tabla de hechos.

Las tablas companies (tabla de dimensión) y transactions (tabla de hechos) están interrelacionadas a través de una relación uno a muchos (1: N), donde la clave primaria company\_id de la tabla companies se utiliza como clave foránea business\_id en la tabla transactions. Esto significa que una única empresa puede realizar múltiples transacciones.

Lo mismo pasa con las tablas credit\_cards (tabla de dimensión) y transactions (tabla de hechos) que están interrelacionadas a través de una relación uno a muchos (1: N), donde la clave primaria id de la tabla credit\_cards se utiliza como clave foránea card\_id en la tabla transactions. Es decir, que una única tarjeta de crédito puede estar asociada con múltiples transacciones.

También las tablas users (tabla de dimensión) y transactions (tabla de hechos) que están interrelacionadas a través de una relación uno a muchos (1: N), donde la clave primaria id de la tabla users se utiliza como clave foránea user\_id en la tabla transactions. Es decir, que un usuario puede realizar múltiples transacciones.

Por otro lado, podemos ver que la tabla products (tabla de dimensión) no tiene una relación directa con la tabla transactions (tabla de hechos). Esto se debe a que la columna product\_ids en la tabla transactions contiene una lista de identificadores de productos en lugar de un único identificador, lo cual impide establecer una relación directa entre ambas tablas. Para resolver esta situación, habría que crear una tabla de hechos puente que registre cada combinación de transacción y producto. Esta tabla puente permitirá manejar la relación de muchos a muchos entre las transacciones y los productos, facilitando un modelo relacional eficiente.



Dado que esta solución se implementará en el nivel 3 del presente script, mantendré el modelo relacional de forma provisional en esta fase. La estructura final, que incluirá la tabla de hechos puente, la presentaré al culminar el sprint 4.

## PASO 4: CARGA DE DATOS

Antes de comenzar a detallar el proceso de carga de datos en cada tabla, es importante mencionar que al intentar realizar la primera carga de datos en la tabla `users`, se presentó el siguiente error: *Error Code: 1290. The MySQL server is running with the --secure-file-priv option so it cannot execute this statement.*

Para resolver este inconveniente, fue necesario realizar un procedimiento de configuración, que incluyó ajustes de permisos y la reubicación de los archivos CSV correspondientes a cada tabla en el directorio autorizado por MySQL. Este directorio seguro, designado por la opción `--secure-file-priv`, está ubicado en:

84	•	SHOW VARIABLES LIKE 'secure_file_priv';
Result Grid   Filter Rows:   Export:   Wrap Cell Content:		
	Variable_name	Value
	secure_file_priv	C:\ProgramData\MySQL\MySQL Server 8.0\Uploads\

✓	84	20:22:23	SHOW VARIABLES LIKE 'secure_file_priv'	1 row(s) returned
---	----	----------	--	-------------------

También fue necesario activar la opción `local_infile` para permitir la carga de datos en las tablas correspondientes. Cada uno de estos pasos se detalla en la sección inicial de la tabla `USERS`, donde explico el procedimiento completo.

## • TABLA USERS

Esta tabla se le debe insertar los siguientes archivos CSV:

1. `users_ca.csv`
2. `users_uk.csv`
3. `users_usa.csv`

### 1. `users_ca.csv`

Para cargar este archivo en la tabla, realicé la siguiente declaración:

88	•	LOAD DATA INFILE "C:\Users\van_m\OneDrive\Escritorio\FADD\Especializacion\DA_Especializacion_Ejercicios\S4_Ejercicios\users_ca.csv"
89		INTO TABLE users
90		FIELDS TERMINATED BY ','
91		ENCLOSED BY ''
92		LINES TERMINATED BY '\r\n'
93		IGNORE 1 ROWS;

✖	50	13:10:40	LOAD DATA INFILE "C:\Users\van_m\OneDrive\Escritorio\FADD\Especializacion\DA_Especializacion_Ejercicios\S4_Ejercicios\users_ca.csv" Error Code: 1290. The MySQL server is running with the --secure-file-priv option so it cannot execute thi...
---	----	----------	--

- **LOAD DATA INFILE** "C:\Users\van\_m\OneDrive\Escritorio\FADD\Especializacion\DA\_Especializacion\_Ejercicios\S4\_Ejercicios\users\_ca.csv": con este comando indico que debe cargar los datos de un archivo de texto que se encuentra en la ruta señalada.
- **INTO TABLE users**: declaración que indica en que tabla se insertarán los datos. En este caso es en la tabla `users`.

- **FIELDS TERMINATED BY ';' :** declaración que indica que los campos de los datos dentro del archivo CSV están separados con comas (,).
- **ENCLOSED BY '"':** este comando especifica que los valores de los campos dentro del archivo CSV se pueden encontrar entre comillas dobles (").
- **LINES TERMINATED BY '\r\n':** es un comando que indica como están terminadas las líneas en el archivo CSV. En este caso, \r\n es un formato de salto de línea utilizado por sistemas operativos Windows para marcar el fin de cada línea.
  - \r (carriage return): es un carácter de control, que indica que el curso de vuelta al principio de la línea sin cambiar línea.
  - \n (line feed): es un carácter de control que mueve el curso hacia la siguiente línea en el archivo.
- **IGNORE 1 ROWS:** declaración que indica que se debe ignorar la primera fila del archivo CSV, sería el encabezado (nombre de los campos).

Al ejecutar esta declaración, arrojo el siguiente error que me impidió subir los datos a la tabla seleccionada:

**Error Code: 1290. The MySQL server is running with the --secure-file-priv option so it cannot execute this statement**

Este error significa que se quiso cargar un archivo, pero el servidor MySQL está configurado con una opción de seguridad que restringe las ubicaciones desde las cuales se pueden cargar archivos. Por tal motivo, la solución que encontré para solucionar este error fue la siguiente:

a) Verificar la configuración de --secure-file-priv

```
84 • SHOW VARIABLES LIKE 'secure_file_priv';
85
86
```

Variable_name	Value
secure_file_priv	C:\ProgramData\MySQL\MySQL Server 8.0\Uploads\

51 13:14:36 SHOW VARIABLES LIKE 'secure\_file\_priv' 1 row(s) returned

Esta declaración muestra la ruta donde MySQL permite leer o escribir archivos. En este caso, la ruta es: C:\ProgramData\MySQL\MySQL Server 8.0\Uploads\.

Al encontrar esta ruta permitida, cambie el archivo users\_ca.csv a esa carpeta para que MySQL permita leer los archivos y así poder intentar nuevamente la declaración para cargar los datos a la tabla users.

```
91 • LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users_ca.csv"
92 INTO TABLE users
93 FIELDS TERMINATED BY ','
94 ENCLOSED BY '"'
95 LINES TERMINATED BY '\r\n'
96 IGNORE 1 ROWS;
```

55 13:20:03 LOAD DATA INFILE "C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/users\_ca.csv" INTO TA... Error Code: 1290. The MySQL server is running with the --secure-file-priv option so it cannot execute thi...

Al ejecutar nuevamente esta declaración, siguió el mismo error por lo que procedí al siguiente paso.

b) Verificación del servidor MySQL

```
86 • SHOW VARIABLES LIKE 'local_infile';
87
88
89
```

Variable_name	Value
local_infile	OFF

59 14:49:53 SHOW VARIABLES LIKE 'local\_infile' 1 row(s) returned

Con esta declaración, verifico si la variable 'local\_infile' está habilitada en el servidor MySQL, lo cual es necesario para utilizar el comando LOAD DATA LOCAL INFILE para la carga de datos desde archivos locales. Al ejecutar la consulta, observé que 'local\_infile' estaba desactivada en el servidor. Por lo tanto, procedí a activarla mediante el siguiente comando, permitiendo así la carga de datos desde archivos ubicados en el servidor local:

```
100 • set global local_infile = 1;
101
```

60 14:51:52 set global local\_infile = 1 0 row(s) affected

Este comando habilita la opción 'local\_infile' globalmente, permitiendo el uso de LOAD DATA INFILE en el servidor MySQL. Aquí, 1 es el equivalente de ON, lo que significa que, al ejecutar este comando, se activará la opción para cargar archivos desde mi ordenador local. A continuación, muestro el cambio de estado:

```
86 • SHOW VARIABLES LIKE 'local_infile';
87
88
89
```

Variable_name	Value
local_infile	ON

61 14:52:54 SHOW VARIABLES LIKE 'local\_infile' 1 row(s) returned

Una vez completada esta tarea, procedí a ejecutar nuevamente la instrucción LOAD DATA INFILE, ajustando las barras de las rutas de directorio para asegurar que MySQL pudiera interpretarlas correctamente. Detecté que había un error en la orientación de las barras, así que no solo corregí la dirección (de / a \ en entornos compatibles) sino que también dupliqué las barras invertidas (\\) para evitar que se interpretaran de forma incorrecta. Estos ajustes permitieron que el comando reconociera los directorios de manera precisa y evitara posibles errores en la ruta del archivo:

```
89 • LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\users_ca.csv"
90 INTO TABLE users
91 FIELDS TERMINATED BY ','
92 ENCLOSED BY '"'
93 LINES TERMINATED BY '\\r\\n'
94 IGNORE 1 ROWS;
```

80 15:46:57 LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\... 75 row(s) affected Records: 75 Deleted: 0 Skipped: 0 Warnings: 0

Como resultado, la ejecución fue exitosa, logrando cargar un total de 75 registros.

Para comprobar dicha carga, realicé la siguiente declaración:

```
97 • select * from users;
98
```

id	name	surname	phone	email	birth_date	country	city	postal_code	address
201	Iola	Powers	018-139-4717	ante.blandit@outlook.edu	Mar 20, 2000	Canada	Rigolet	V6T 6M7	154-5415 Auctor St.
202	Maxwell	Holden	045-402-7693	donec@hotmail.edu	Dec 2, 1986	Canada	Murdochville	S7E 6E0	Ap #880-6372 Ultrices. St.
203	Jarrold	Fields	010-741-8105	sit.amet@google.couk	Jan 6, 1982	Canada	Baddeck	K3X 6Z5	441-8969 Rhoncus Road
204	Emerson	Sharp	068-138-9383	ante.iaculis@outlook.ca	Oct 15, 1994	Canada	Maple Creek	Y2C 9E6	517-6759 Ut, Av.
205	Sonya	Mckee	041-151-9737	magna.phasellus.dolor@google.ca	May 7, 1983	Canada	Dieppe	E7S 4P8	Ap #916-8051 A St.
206	Harper	Hart	030-656-1670	fringilla.donec@outlook.net	Nov 17, 2000	Canada	Québec City	B4K 0J6	8588 Massa. Ave
207	Yvonne	Hatfield	003-854-1445	magna.et.ipsuam@google.edu	Sep 22, 1981	Canada	Rae-Edzo	20Y 8L2	Ap #636-8055 Egestas St.
208	Burke	Graham	064-568-4454	vel@yahoo.org	Feb 23, 1993	Canada	Annapolis Royal	S4Y 8V5	Ap #983-6042 Amet Street
209	Athena	Malone	027-280-8275	pellentesque.tincidunt@yahoo.ca	Dec 14, 1991	Canada	Cambridge Bay	93Z 5S5	Ap #388-8542 Est St.
210	Slade	Poole	084-771-1363	amet@icloud.com	Feb 16, 2001	Canada	Ottawa	A1S 9W6	601-6142 Etiam St.

83 20:07:26 select \* from users 75 row(s) returned

## 2. users\_uk.csv

Para cargar los datos de este archivo, realicé la siguiente declaración:

```
98 • LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\users_uk.csv"
99 INTO TABLE users
100 FIELDS TERMINATED BY ','
101 ENCLOSED BY '"'
102 LINES TERMINATED BY '\\r\\n'
103 IGNORE 1 ROWS;
```

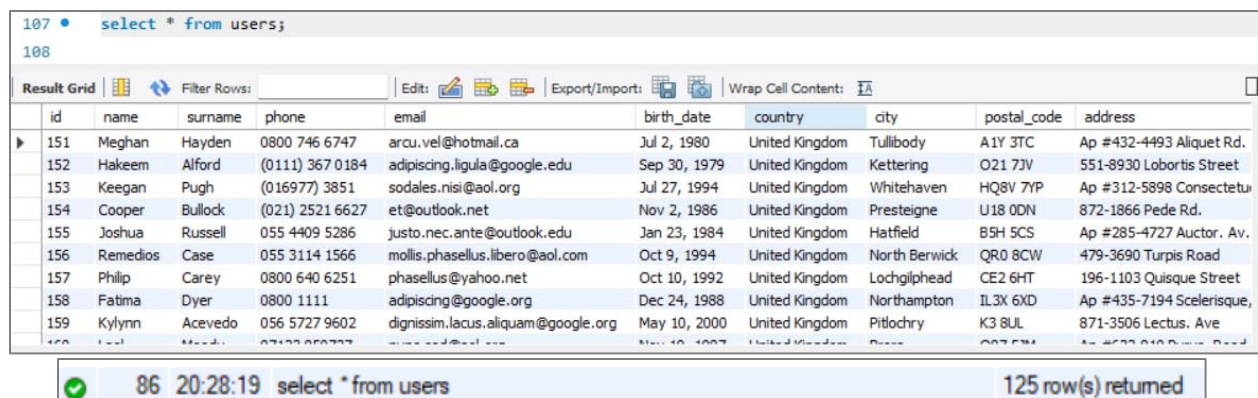
85 20:25:40 LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\... 50 row(s) affected Records: 50 Deleted: 0 Skipped: 0 Warnings: 0

- **LOAD DATA INFILE** "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\users\_uk.csv": con este comando indico que debe cargar los datos de un archivo de texto que se encuentra en la ruta señalada.
- **INTO TABLE users**: declaración que indica en que tabla se insertarán los datos. En este caso es en la tabla *users*.
- **FIELDS TERMINATED BY ','**: declaración que indica que los campos de los datos dentro del archivo CSV están separados con comas (,).
- **ENCLOSED BY '"'**: este comando especifica que los valores de los campos dentro del archivo CSV se pueden encontrar entre comillas dobles ("").

- **LINES TERMINATED BY '\r\n':** es un comando que indica como están terminadas las líneas en el archivo CSV. En este caso, \r\n es un formato de salto de línea utilizado por sistemas operativos Windows para marcar el fin de cada línea.  
 \r (carriage return): es un carácter de control, que indica que el curso de vuelta al principio de la línea sin cambiar línea.  
 \n (line feed): es un carácter de control que mueve el curso hacia la siguiente línea en el archivo.
- **IGNORE 1 ROWS:** declaración que indica que se debe ignorar la primera fila del archivo CSV, sería el encabezado (nombre de los campos).

Como resultado, la ejecución fue exitosa, logrando cargar un total de 50 registros.

Para comprobar dicha carga, realicé la siguiente declaración:

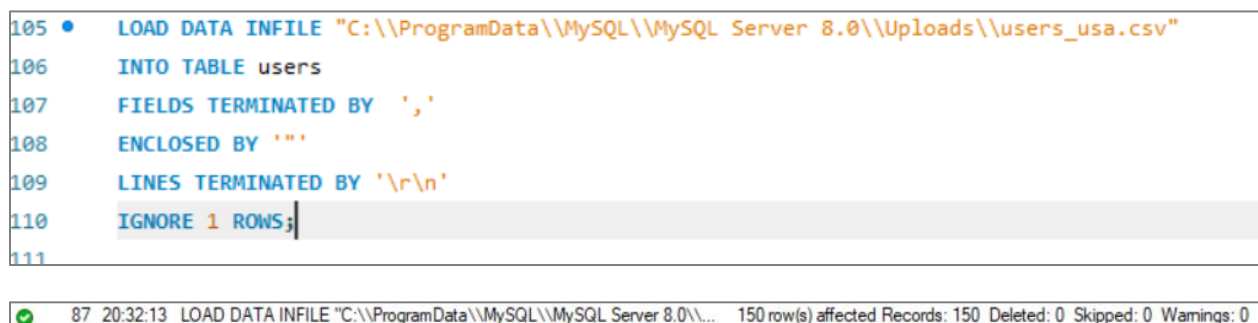


id	name	surname	phone	email	birth_date	country	city	postal_code	address
151	Meghan	Hayden	0800 746 6747	arcu.vel@hotmail.ca	Jul 2, 1980	United Kingdom	Tullibody	A1Y 3TC	Ap #432-4493 Aliquet Rd.
152	Hakeem	Alford	(0111) 367 0184	adipiscing.ligula@google.edu	Sep 30, 1979	United Kingdom	Kettering	O21 7JV	551-8930 Lobortis Street
153	Keegan	Pugh	(016977) 3851	sodales.nisi@aol.org	Jul 27, 1994	United Kingdom	Whitehaven	HQ8V 7YP	Ap #312-5898 Consectetur
154	Cooper	Bullock	(021) 2521 6627	et@outlook.net	Nov 2, 1986	United Kingdom	Presteigne	U18 0DN	872-1866 Pede Rd.
155	Joshua	Russell	055 4409 5286	justo.nec.ante@outlook.edu	Jan 23, 1984	United Kingdom	Hatfield	B5H 5CS	Ap #285-4727 Auctor. Av.
156	Remedios	Case	055 3114 1566	mollis.phasellus.libero@aol.com	Oct 9, 1994	United Kingdom	North Berwick	QR0 8CW	479-3690 Turpis Road
157	Philip	Carey	0800 640 6251	phasellus@yahoo.net	Oct 10, 1992	United Kingdom	Lochgilphead	CE2 6HT	196-1103 Quisque Street
158	Fatima	Dyer	0800 1111	adipiscing@google.org	Dec 24, 1988	United Kingdom	Northampton	IL3X 6XD	Ap #435-7194 Scelerisque
159	Kylynn	Acevedo	056 5727 9602	dignissim.lacus.aliquam@google.org	May 10, 2000	United Kingdom	Pitlochry	K3 8UL	871-3506 Lectus. Ave

Aquí podemos notar que la tabla `users` paso de tener 75 resultados, a tener ahora 125 resultados. Es decir, la carga fue exitosa.

### 3. users\_usa.csv

Para cargar los datos de este archivo, realicé la siguiente declaración:



```
105 • LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\users_usa.csv"
106 INTO TABLE users
107 FIELDS TERMINATED BY ','
108 ENCLOSED BY ''''
109 LINES TERMINATED BY '\\r\\n'
110 IGNORE 1 ROWS;
111
```

- **LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\users\_usa.csv":** con este comando indico que debe cargar los datos de un archivo de texto que se encuentra en la ruta señalada.
- **INTO TABLE users:** declaración que indica en que tabla se insertarán los datos. En este caso es en la tabla `users`.
- **FIELDS TERMINATED BY ',':** declaración que indica que los campos de los datos dentro del archivo CSV están separados con comas (,).
- **ENCLOSED BY ''':** este comando especifica que los valores de los campos dentro del archivo CSV se pueden encontrar entre comillas dobles (").

- **LINES TERMINATED BY '\r\n':** es un comando que indica como están terminadas las líneas en el archivo CSV. En este caso, \r\n es un formato de salto de línea utilizado por sistemas operativos Windows para marcar el fin de cada línea.  
    \r (carriage return): es un carácter de control, que indica que el curso de vuelta al principio de la línea sin cambiar línea.  
    \n (line feed): es un carácter de control que mueve el curso hacia la siguiente línea en el archivo.
- **IGNORE 1 ROWS:** declaración que indica que se debe ignorar la primera fila del archivo CSV, sería el encabezado (nombre de los campos).

Como resultado, la ejecución fue exitosa, logrando cargar un total de 150 registros.

Para comprobar dicha carga, realicé la siguiente declaración:

id	name	surname	phone	email	birth_date	country	city	postal_code	address
1	Zeus	Gamble	1-282-581-0551	interdum.enim@protonmail.edu	Nov 17, 1985	United States	Lowell	73544	348-7818 Sagittis St.
2	Garrett	Mcconnell	(718) 257-2412	integer.vitae.nibh@protonmail.org	Aug 23, 1992	United States	Desmomes	59464	903 Sit Ave
3	Claran	Harrison	(522) 598-1365	interdum.feugiat@aol.org	Apr 29, 1998	United States	Columbus	56518	736-2063 Tellus St.
4	Howard	Stafford	1-411-740-3269	ornare.egestas@icloud.edu	Feb 18, 1989	United States	Kailua	77417	Ap #545-2244 Erat. Rd.
5	Hayfa	Pierce	1-554-541-2077	et.malesuada.fames@hotmail.org	Sep 26, 1998	United States	Sandy	31564	341-2821 Ultrices Av.
6	Joel	Tyson	(718) 288-8020	gravidam.nunc.sed@yahoo.ca	Oct 15, 1989	United States	Nashville	96838	888-2799 Amet Street
7	Rafael	Jimenez	(817) 689-0478	eget@outlook.ca	Dec 4, 1981	United States	Hillsboro	29874	8627 Malesuada Rd.
8	Nissim	Franks	(692) 157-3469	egestas.aliquam.fringilla@google.ca	Aug 1, 1993	United States	Jackson	61750	Ap #251-7144 Integer St.
9	Mannix	Mcclain	(590) 883-2184	aliquam.nisl@outlook.com	Jan 24, 1987	United States	Richmond	35987	647-3080 Lacus. St.

Aquí podemos notar que la tabla users paso de tener 125 resultados, a tener ahora 275 resultados. Es decir, la carga fue exitosa y coincide con el total de usuarios detectados en el archivo csv.

## • TABLA COMPANIES

Para cargar los datos de este archivo, realicé la siguiente declaración:

```
117 • LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\companies.csv"
118 INTO TABLE companies
119 FIELDS TERMINATED BY ','
120 ENCLOSED BY ''''
121 LINES TERMINATED BY '\\r\\n'
122 IGNORE 1 ROWS;
```

91 20:47:14 LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\... 100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0

- **LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\companies.csv":** con este comando indico que debe cargar los datos de un archivo de texto que se encuentra en la ruta señalada.
- **INTO TABLE companies:** declaración que indica en que tabla se insertarán los datos. En este caso es en la tabla companies.
- **FIELDS TERMINATED BY ',':** declaración que indica que los campos de los datos dentro del archivo CSV están separados con comas (,).



- **ENCLOSED BY ''**: este comando especifica que los valores de los campos dentro del archivo CSV se pueden encontrar entre comillas dobles (").
- **LINES TERMINATED BY '\r\n'**: es un comando que indica como están terminadas las líneas en el archivo CSV. En este caso, \r\n es un formato de salto de línea utilizado por sistemas operativos Windows para marcar el fin de cada línea.
  - \r (carriage return): es un carácter de control, que indica que el curso de vuelta al principio de la línea sin cambiar línea.
  - \n (line feed): es un carácter de control que mueve el curso hacia la siguiente línea en el archivo.
- **IGNORE 1 ROWS**: declaración que indica que se debe ignorar la primera fila del archivo CSV, sería el encabezado (nombre de los campos).

Como resultado, la ejecución fue exitosa, logrando cargar un total de 100 registros.

Para comprobar dicha carga, realicé la siguiente declaración:

```
124 • select * from companies;
125
```

	company_id	company_name	phone	email	country	website
▶	b-2222	Ac Fermentum Incorporated	06 85 56 52 33	donec.porttitor.tellus@yahoo.net	Germany	https://instagram.com/site
	b-2226	Magna A Neque Industries	04 14 44 64 62	risus.donec.nibh@icloud.org	Australia	https://whatsapp.com/group/9
	b-2230	Fusce Corp.	08 14 97 58 85	risus@protonmail.edu	United States	https://pinterest.com/sub/cars
	b-2234	Convallis In Incorporated	06 66 57 29 50	mauris.ut@aol.couk	Germany	https://cnn.com/user/110
	b-2238	Ante Iaculis Nec Foundation	08 23 04 99 53	sed.dictum.proin@outlook.ca	New Zealand	https://netflix.com/settings
	b-2242	Donec Ltd	01 25 51 37 37	at.iaculis@hotmail.couk	Norway	https://nytimes.com/user/110
	b-2246	Sed Nunc Ltd	02 62 64 73 48	nibh@yahoo.org	United Kingdom	https://cnn.com/one
	b-2250	Amet Nulla Donec Corporation	07 15 25 14 74	mattis.integer.eu@protonmail.net	Italy	https://netflix.com/sub/cars
	b-2254	Nascetur Ridiculus Mus Inc.	06 26 87 61 84	suspendisse.dui@icloud.net	United States	https://ebay.com/sub
	b-2258	Vestibulum Lorem PC	02 02 87 33 40	aenean.massa.integer@aol.net	Belgium	https://pinterest.com/sub/cars
	b-2262	Condone Sed Nunc Ltd	02 04 28 33 07	quisque.quisque@protonmail.net	Sweden	https://netflix.com/site

92 20:52:21 select \* from companies 100 row(s) returned

## • TABLA CREDIT\_CARDS

Para cargar los datos de este archivo, realicé la siguiente declaración:

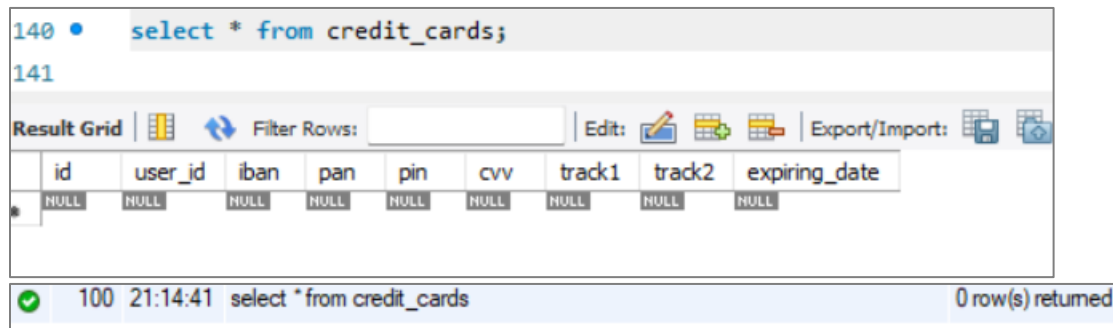
```
128 • LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\credit_cards.csv"
129 INTO TABLE credit_cards
130 FIELDS TERMINATED BY ','
131 ENCLOSED BY ''
132 LINES TERMINATED BY '\r\n'
133 IGNORE 1 ROWS;
```

94 21:02:52 LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\... 0 row(s) affected Records: 0 Deleted: 0 Skipped: 0 Warnings: 0

- **LOAD DATA INFILE "LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\credit\_cards.csv"**: con este comando indico que debe cargar los datos de un archivo de texto que se encuentra en la ruta señalada.
- **INTO TABLE credit\_cards**: declaración que indica en que tabla se insertarán los datos. En este caso es en la tabla companies.
- **FIELDS TERMINATED BY ','**: declaración que indica que los campos de los datos dentro del archivo CSV están separados con comas (,).

- **ENCLOSED BY ''**: este comando especifica que los valores de los campos dentro del archivo CSV se pueden encontrar entre comillas dobles (").
- **LINES TERMINATED BY '\r\n'**: es un comando que indica como están terminadas las líneas en el archivo CSV. En este caso, \r\n es un formato de salto de línea utilizado por sistemas operativos Windows para marcar el fin de cada línea.  
  \r (carriage return): es un carácter de control, que indica que el curso de vuelta al principio de la línea sin cambiar línea.  
  \n (line feed): es un carácter de control que mueve el curso hacia la siguiente línea en el archivo.
- **IGNORE 1 ROWS**: declaración que indica que se debe ignorar la primera fila del archivo CSV, sería el encabezado (nombre de los campos).

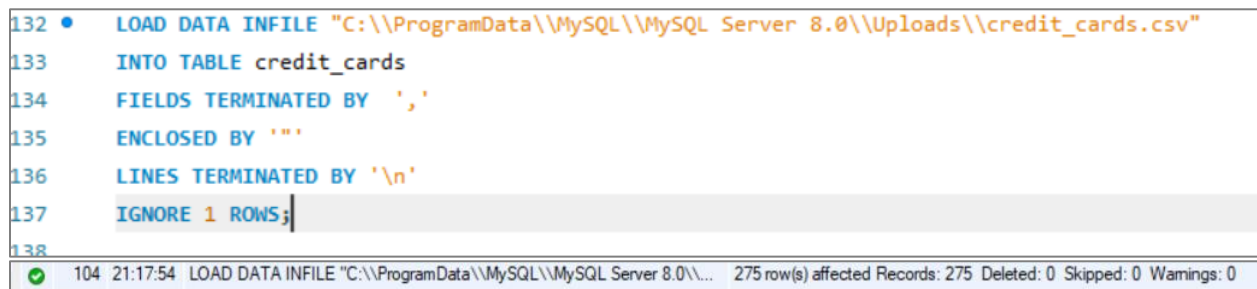
Como podemos ver esta ejecución funciona, pero no se inserto correctamente los datos por lo que al realizar la consulta, la misma presenta datos NULL.



The screenshot shows a SQL query execution interface. The query is `select * from credit_cards;`. The result is displayed in a table with the following columns: `id`, `user_id`, `iban`, `pan`, `pin`, `cvv`, `track1`, `track2`, and `expiring_date`. All values in the table are NULL. The status bar at the bottom indicates "0 row(s) returned".

id	user_id	iban	pan	pin	cvv	track1	track2	expiring_date
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Este problema se debió a una incompatibilidad en la declaración de **LINES TERMINATED BY '\r\n'**, ya que es probable que el archivo CSV se haya generado en un sistema distinto a Windows. En sistemas como Linux o macOS, el terminador de línea suele ser `\n` en lugar de `\r\n`. Por lo tanto, ajusto la instrucción para utilizar el terminador de línea `\n`, asegurando una correcta interpretación del archivo CSV al realizar la carga de datos:



The screenshot shows a SQL query execution interface. The query is `LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\credit_cards.csv" INTO TABLE credit_cards FIELDS TERMINATED BY ',' ENCLOSED BY '' LINES TERMINATED BY '\n' IGNORE 1 ROWS;`. The status bar at the bottom indicates "275 row(s) affected Records: 275 Deleted: 0 Skipped: 0 Warnings: 0".

id	user_id	iban	pan	pin	cvv	track1	track2	expiring_date
1	1	1234567890123456789012345678901234	1234 5678 9012 3456	1234 5678 9012 3456	1234 5678 9012 3456	1234 5678 9012 3456	1234 5678 9012 3456	12/31/2025

Como resultado, la ejecución fue exitosa, logrando cargar un total de 275 registros.

Para comprobar dicha carga, realicé la siguiente declaración:

```
140 • select * from credit_cards;
141
```

	id	user_id	iban	pan	pin	cvv	track1	track2
▶	CcU-2938	275	TR301950312213576817638661	5424465566813633	3257	984	%88383712448554646^WovsxejDpwiev^8604...	%87653863056044187=80071
	CcU-2945	274	DO26854763748537475216568689	5142423821948828	9080	887	%84621311609958661^UftuyfsSeimxn^06106...	%84149568437843501=51071
	CcU-2952	273	BG451VQL52710525608255	4556 453 55 5287	4598	438	%82183285104307501^CddytytcUxwfdq^5907...	%86778580257827162=69068
	CcU-2959	272	CR7242477244335841535	372461377349375	3583	667	%87281111956795320^XocddjBckecd^09016...	%84246154489281853=28052
	CcU-2966	271	BG72LKTQ15627628377363	448566 886747 7265	4900	130	%84728932322756223^JhlgsuFbmwgj^7202...	%82318571115599881=89082
	CcU-2973	270	PT87806228135092429456346	544 58654 54343 384	8760	887	%84761405253275637^HjnnipoBlejrl^7108515...	%87816169831446746=13102
	CcU-2980	269	DE39241881883086277136	402400 7145845969	5075	596	%87320483593870549^OokzqxHpsed^4901...	%82474313962214151=04122
	CcU-2987	268	GE89681434837748781813	3763 747687 76666	2298	797	%84750646345146674^PjmlyrfGwwtrf^83051...	%85441935173418615=41037
	CcU-2994	267	BH62714428368066765294	344283273252593	7545	595	%81583759784015674^GmqoyhtUtoqrm^2507...	%84141467473024349=65068

105 21:18:57 select \* from credit\_cards 275 row(s) returned

## TABLA PRODUCTS

Para cargar los datos de este archivo, realicé la siguiente declaración:

```
144 • LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\products.csv"
145 INTO TABLE products
146 FIELDS TERMINATED BY ','
147 ENCLOSED BY '"'
148 LINES TERMINATED BY '\r\n'
149 IGNORE 1 ROWS;
150
```

111 21:46:45 LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\products.csv" IN... 0 row(s) affected Records: 0 Deleted: 0 Skipped: 0 Warnings: 0

- **LOAD DATA INFILE** "LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\products.csv": con este comando indico que debe cargar los datos de un archivo de texto que se encuentra en la ruta señalada.
- **INTO TABLE products**: declaración que indica en que tabla se insertarán los datos. En este caso es en la tabla companies.
- **FIELDS TERMINATED BY ','**: declaración que indica que los campos de los datos dentro del archivo CSV están separados con comas (,).
- **ENCLOSED BY '"'**: este comando especifica que los valores de los campos dentro del archivo CSV se pueden encontrar entre comillas dobles ("").
- **LINES TERMINATED BY '\r\n'**: es un comando que indica como están terminadas las líneas en el archivo CSV. En este caso, \r\n es un formato de salto de línea utilizado por sistemas operativos Windows para marcar el fin de cada línea.
  - \r (carriage return): es un carácter de control, que indica que el curso de vuelta al principio de la línea sin cambiar línea.
  - \n (line feed): es un carácter de control que mueve el curso hacia la siguiente línea en el archivo.
- **IGNORE 1 ROWS**: declaración que indica que se debe ignorar la primera fila del archivo CSV, sería el encabezado (nombre de los campos).

Como podemos ver esta ejecución funciona, pero no se insertó correctamente los datos por lo que, al realizar la consulta, la misma presenta datos NULL.

```
152 • select * from products;
153
```

Result Grid

	id	product_name	price	colour	weight	warehouse_id
*	NULL	NULL	NULL	NULL	NULL	NULL

112 21:48:34 select \* from products 0 row(s) returned

Este problema se debió a una incompatibilidad en la declaración de LINES TERMINATED BY '\r\n', ya que es probable que el archivo CSV se haya generado en un sistema distinto a Windows. En sistemas como Linux o macOS, el terminador de línea suele ser \n en lugar de \r\n. Por lo tanto, ajusto la instrucción para utilizar el terminador de línea \n, asegurando una correcta interpretación del archivo CSV al realizar la carga de datos:

```
144 • LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\products.csv"
145 INTO TABLE products
146 FIELDS TERMINATED BY ','
147 ENCLOSED BY '"'
148 LINES TERMINATED BY '\n'
149 IGNORE 1 ROWS;
150
```

113 21:49:55 LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\products.csv" IN... Error Code: 1366. Incorrect decimal value: '\$161.11' for column 'price' at row 1

Al ejecutar esta instrucción, surgió un nuevo error debido a un valor incorrecto en la columna *price*, ya que los datos contienen el símbolo de moneda (\$), lo cual impide la inserción en un campo definido como DECIMAL (10,2). Para resolver esto, modifiqué temporalmente el tipo de la columna *price* a VARCHAR (100), permitiendo así la carga de datos en la tabla. Posteriormente, al final del sprint en la sección "Actualización De Formato Y Tipo De Datos Pendientes", realizaré las modificaciones necesarias para limpiar los símbolos de moneda y devolver la columna *price* a su tipo original DECIMAL (10,2), asegurando la integridad del tipo de datos:

```
ALTER TABLE products MODIFY price VARCHAR(100);
```

18 11:32:24 ALTER TABLE products MODIFY price VARCHAR(100) 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

A continuación, podemos ver que dicha modificación se realizó correctamente:

Table: products

Columns:

id	int PK
product_name	varchar(50)
price	varchar(100)
colour	varchar(50)
weight	decimal(3,2)
warehouse_id	varchar(30)

Terminadas estas modificaciones, procedí a ejecutar nuevamente la declaración LOAD DATA INFILE:

```
146 • LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\products.csv"
147 INTO TABLE products
148 FIELDS TERMINATED BY ','
149 ENCLOSED BY '"'
150 LINES TERMINATED BY '\\n'
151 IGNORE 1 ROWS;
```

19 11:34:33 LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\products.csv" INTO ... 100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0

Como resultado, la ejecución fue exitosa, logrando cargar un total de 100 registros.

Para comprobar dicha carga, realicé la siguiente declaración:

```
154 • select * from products;
155
```

	id	product_name	price	colour	weight	warehouse_id
	1	Direwolf Stannis	\$161.11	#7c7c7c	1.00	WH-4
	2	Tarly Stark	\$9.24	#919191	2.00	WH-3
	3	duel tourney Lannister	\$171.13	#d8d8d8	1.50	WH-2
	4	warden south duel	\$71.89	#111111	3.00	WH-1
	5	skywalker ewok	\$171.22	#bdbdbd	3.20	WH-0
	6	dooku solo	\$136.60	#c4c4c4	0.80	WH--1
	7	north of Casterly	\$63.33	#b7b7b7	0.60	WH--2
	8	Winterfell	\$32.37	#383838	1.40	WH--3
	9	Winterfell	\$76.40	#b5b5b5	1.20	WH--4
	10	Karstark Dorne	\$119.52	#f4f4f4	2.40	WH--5

20 11:35:16 select \* from products 100 row(s) returned

## • TABLA TRANSACTIONS

Para cargar los datos de este archivo, realicé la siguiente declaración:

```
146 • LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\transactions.csv"
147 INTO TABLE transactions
148 FIELDS TERMINATED BY ';'
149 ENCLOSED BY '"'
150 LINES TERMINATED BY '\\r\\n'
151 IGNORE 1 ROWS;
```

22 11:51:27 LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\transactions.csv" IN... 587 row(s) affected Records: 587 Deleted: 0 Skipped: 0 Warnings: 0

- **LOAD DATA INFILE** "LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\transactions.csv": con este comando indico que debe cargar los datos de un archivo de texto que se encuentra en la ruta señalada.
- **INTO TABLE transactions**: declaración que indica en que tabla se insertarán los datos. En este caso es en la tabla companies.
- **FIELDS TERMINATED BY ';'** : declaración que indica que los campos de los datos dentro del archivo CSV están separados con puntos y comas (;).
- **ENCLOSED BY '"'**: este comando especifica que los valores de los campos dentro del archivo CSV se pueden encontrar entre comillas dobles ("").
- **LINES TERMINATED BY '\r\n'**: es un comando que indica como están terminadas las líneas en el archivo CSV. En este caso, \r\n es un formato de salto de línea utilizado por sistemas operativos Windows para marcar el fin de cada línea.  
    \r (carriage return): es un carácter de control, que indica que el curso de vuelta al principio de la línea sin cambiar línea.  
    \n (line feed): es un carácter de control que mueve el curso hacia la siguiente línea en el archivo.
- **IGNORE 1 ROWS**: declaración que indica que se debe ignorar la primera fila del archivo CSV, sería el encabezado (nombre de los campos).

Como resultado, la ejecución fue exitosa, logrando cargar un total de 587 registros.

Para comprobar dicha carga, realicé la siguiente declaración:

id	card_id	business_id	timestamp	amount	declined	product_ids	user_id	lat	longitude
02C6201E-D90A-1859-B4EE-88D2986D3B02	CcU-2938	b-2362	2021-08-28 23:42:24	466.92	0	71, 1, 19	92	81.9185	-12.5276
0466A42E-47CF-8D24-FD01-C0B689713128	CcU-4219	b-2302	2021-07-26 07:29:18	49.53	0	47, 97, 43	170	-43.9695	-117.525
063FBA79-99EC-66FB-29F7-25726D1764A5	CcU-2987	b-2250	2022-01-06 21:25:27	92.61	0	47, 67, 31, 5	275	-81.2227	-129.05
0668296C-CD89-A883-76BC-2E4C4F8C8AE	CcU-3743	b-2618	2022-01-26 02:07:14	394.18	0	89, 83, 79	265	-34.3593	-100.556
06CD9AA5-9842-D684-DDDD-A5E394FEBA99	CcU-2959	b-2346	2021-10-26 23:00:01	279.93	0	43, 31	92	33.7381	158.298
07A46D48-31A3-7E87-65B9-0DA902AD109F	CcU-3225	b-2386	2021-06-28 21:11:42	340.87	1	47, 23	272	38.8342	92.1905
09DE92CE-6F27-2BB7-13B5-938582B388E2	CcU-3071	b-2298	2021-05-11 20:40:06	303.05	1	67, 7	275	71.1706	10.5757
0A476ED9-0C13-1962-F87B-D3563924B539	CcU-4359	b-2302	2022-02-26 20:33:54	430.49	0	29, 41, 11	221	-56.4901	114.801
08EB80B7-9D66-1707-CE4B-9DC7E71914B5	CcU-3141	b-2338	2022-03-04 14:54:35	288.81	1	19, 41, 29, 3	272	23.3264	-13.6037

## EJERCICIO 1

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

user_id	Cant_Transacciones
92	39
275	48
272	76
267	52



32 11:28:23 SELECT user\_id, count(amount) AS Cant\_Transacciones FROM transactions WHERE user\_id IN (sel... 4 row(s) returned

- **SELECT user\_id, count(amount) AS Cant\_Transacciones, declined:** aquí seleccione los campos que son pertinentes para la consulta seleccionada:
  - User\_id
  - Count(amount) AS Cant\_Transacciones=> este valor es el cálculo de la cantidad de las transacciones realizadas por cada usuario.
- **FROM transactions:** indicación de que tabla debe obtener los datos.
- **WHERE user\_id IN (select id FROM users):** añadí una subconsulta en la cláusula WHERE, donde indico que el campo user\_id debe coincidir con los identificadores de la tabla users.
- **GROUP BY 1:** agrupa el campo user\_id para poder hacer el cálculo de la cantidad de transacciones por usuario.
- **HAVING count(amount)>30:** aplica un filtro que limita los resultados de la función COUNT (amount). Esta condición asegura que solo se muestren aquellos resultados en los que el número total de registros es superior a 30.

## 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.**

A) Verificación de que existe la compañía Donec Ltd en la tabla companies

```
170 • SELECT* FROM companies
171 WHERE company_name = 'Donec Ltd';
172
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

	company_id	company_name	phone	email	country	website
▶	b-2242	Donec Ltd	01 25 51 37 37	at.iaculis@hotmail.couk	Norway	https://nytimes.com/user/110
*	NULL	NULL	NULL	NULL	NULL	NULL

85 13:32:25 SELECT\* FROM companies WHERE company\_name = 'Donec Ltd' 1 row(s) returned

- **SELECT \* FROM companies:** declaración para consultar todos los registros existentes en la tabla companies.
- **WHERE company\_name = 'Donec Ltd':** filtro que restringe la búsqueda en la columna company\_name. En este caso, indico que el company\_name tiene que ser igual a 'Donec Ltd'.

Esta declaración arroja como resultado que existe la empresa en la tabla companies.

B) Cálculo de la media.

```
172 -- Calculo de la media de las transacciones
173 • SELECT iban, ROUND(avg(amount),2) AS Media_amount
174 FROM transactions
175 JOIN companies
176 ON company_id= business_id
177 JOIN credit_cards
178 ON card_id=credit_cards.id
179 WHERE company_name='Donec Ltd'
180 GROUP BY 1;
181
```

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

iban	Media_amount
PT87806228135092429456346	203.72

✓ 39 11:43:31 SELECT iban, ROUND(avg(amount),2) AS Media\_amount FROM transactions JOIN companies ON co... 1 row(s) returned

- **SELECT iban, ROUND(avg(amount),2) AS Media\_amount:** aquí seleccione los campos que son pertinentes para la consulta seleccionada:
  - Iban
  - ROUND(avg(amount),2) AS Media\_amount=> apliqué la función AVG al campo amount para calcular el promedio de las ventas realizadas. Para asegurar que el resultado sea más legible, utilicé la función ROUND, limitando el promedio a dos decimales. Para facilitar la interpretación de la columna, renombre el campo con el título “Media\_amount” con la función AS.
- **FROM transactions JOIN companies ON company\_id=business\_id:** función que une la tabla transactions con la tabla companies a través de la clave foránea business\_id.
- **JOIN credit\_cards ON card\_id=credit\_cards.id:** función que une la tabla transactions con la tabla credit\_card a través de la clave foránea card\_id.
- **WHERE company\_name = 'Donec Ltd':** cláusula WHERE para enfocar los resultados sólo a transacciones de la compañía 'Donec Ltd'.
- **GROUP BY 1:** agrupa el campo iban para poder calcular la media del monto de las transacciones.

Esta declaración arroja como resultado que la media del monto de las transacciones aprobadas de la empresa Donec Ltd.es de 203,72 (incluye transacciones aprobadas y no aprobadas).

## 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:**

Teniendo en cuenta esta consigna, voy a considerar que el departamento de Riesgo y Seguridad me ha solicitado dicha tabla para supervisar patrones de declinación en transacciones, por lo que a continuación voy a crear una view denominada "Vista Riesgo y Seguridad":

```
189 • CREATE VIEW VistaRiesgoSeguridad AS
190   SELECT card_id, sum(declined) AS Cant_declined,
191   CASE WHEN sum(declined) >= 3 THEN "Bloqueadas"
192   ELSE "Activas" END AS Status_card
193 FROM (SELECT card_id, declined, timestamp
194 FROM (SELECT card_id, declined, timestamp, ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS Numeracion
195 FROM transactions
196 ) AS Tabla_numeracion
197 WHERE Numeracion <= 3
198 ORDER BY card_id, timestamp DESC) AS Aux_Limit_Transactions
199 GROUP BY 1
200 ORDER BY 1 ASC;
```

43 11:53:27 CREATE VIEW VistaRiesgoSeguridad AS SELECT card\_id, sum(declined) AS Cant\_declined, CASE ... 0 row(s) affected

- **CREATE VIEW VistaRiesgoSeguridad AS:** declaración que indica la creación de una vista. En este caso es una tabla virtual llamada "VistaRiesgoSeguridad" que guarda el resultado de una consulta.
- **SELECT card\_id, sum(declined) AS Cant\_declined,:** aquí seleccione los campos que compondrán dicha vista de acuerdo a los requerimientos del enunciado. Los campos seleccionados son:
  - Card\_id
  - sum(declined) AS Cant\_declined=> suma la cantidad de transacciones
- **CASE WHEN sum(declined) >= 3 THEN "Bloqueadas" ELSE "Activas" END AS Status\_card:** implementé la instrucción CASE, que permite evaluar condiciones de manera secuencial y devolver un valor basado en la primera condición que se cumple. En este caso definí las siguientes 2 condiciones según el número de transacciones rechazadas:
  - **WHEN sum(declined) >= 3 THEN "Bloqueadas":** si la tarjeta tiene 3 o mas transacciones rechazadas, el estado de la tarjeta será "Bloqueadas".
  - **ELSE "Activas":** Si tiene menos de 3 transacciones rechazadas, el estado será "Activas".

Al finalizar la estructura de la función CASE, incluí la palabra END para cerrar la declaración y renombré el resultado como "Status\_card" utilizando la cláusula AS, lo que facilita la interpretación de los datos.

- **FROM (SELECT card\_id, declined, timestamp FROM (SELECT card\_id, declined, timestamp, ROW\_NUMBER() OVER (PARTITION BY card\_id ORDER BY timestamp DESC) AS Numeracion FROM transactions) AS Tabla\_numeracion WHERE Numeracion <= 3 AND declined=1 ORDER BY card\_id, timestamp DESC) AS Aux\_Limit\_Transactions:**

Al momento de elegir la tabla para extraer los datos, lo que hice fue una subconsulta para obtener las ultimas 3 transacciones rechazadas para cada tarjeta. Esta subconsulta se desglosa de la siguiente manera:

- **SELECT card\_id, declined, timestamp:** aquí seleccione los campos que compondrán dicha consulta. Los campos seleccionados son:
  - Card\_id
  - Declined
  - Timestamp

- **FROM (SELECT card\_id, declined, timestamp, ROW\_NUMBER() OVER (PARTITION BY card\_id ORDER BY timestamp DESC) AS Numeracion FROM transactions) AS Tabla\_numeracion:** siguiendo con la lógica del enunciado, para obtener los datos precisos tuve que crear una tabla auxiliar a través de otra subconsulta. La misma se explica de la siguiente manera:
  - **SELECT card\_id, declined, timestamp, ROW\_NUMBER () OVER (PARTITION BY card\_id ORDER BY timestamp DESC) AS Numeracion:** seleccione los campos que compondrán dicha consulta. Los campos seleccionados son:
    - Card\_id
    - Declined
    - Timestamp
    - **ROW\_NUMBER() OVER (PARTITION BY card\_id ORDER BY timestamp DESC) AS Numeracion**=> esta función **ROW\_NUMER ( )** asigna un número único secuencial a cada transacción realizada con una tarjeta ordenándolas de la más reciente a la más antigua considerando el campo timestamp. La parte **PARTITION BY card\_id** asegura que el contador de las filas se reinicie para cada tarjeta, es decir, que no hace una enumeración global de las transacciones, sino que se hace por tarjeta. Para cada tarjeta, el contador empieza desde 1. Para la parte de **ORDER BY timestamp DESC** lo que hace es ordenar las transacciones de cada tarjeta en orden descendente (mas recientes a las más antiguas) a través del campo timestamp. Es decir, la transacción mas reciente va a tener el número 1, la que le sigue el número 2 y así sucesivamente. Luego, esta función la renombré "Numeracion" utilizando la cláusula AS, lo que facilita la interpretación de los datos.
  - **FROM transactions:** selecciona la tabla de donde van a surgir los datos. En este caso es de la tabla *transactions*.

A total esta subconsulta que generó una tabla adicional, la denominé "Tabla\_numeracion" para una mejor interpretación de los datos.

- **WHERE Numeracion <= 3:** cláusula WHERE para filtrar los resultados de las transacciones de cada tarjeta manteniendo la siguiente condición:
  - Que los resultados a sean de las ultimas 3 transacciones más recientes (numeración <=3)
- **ORDER BY card\_id, timestamp DESC) AS Aux\_Limit\_Transactions:** a su vez, aquí lo que se indica que se deben ordenar las tarjetas junto con sus fechas de manera descendente, es decir de mayor a menor.

Para un mejor entendimiento de esta subconsulta, la misma la renombre "Aux\_Limit\_Transactions" a través de la cláusula AS para una mejor comprensión de sus datos.

- **GROUP BY I:** agrupa el campo por card\_id, es decir que todas las transacciones de una misma tarjeta se agruparan en una sola fila.
- **ORDER BY I ASC:** ordena los datos por el campo card\_id de manera ascendente, es decir de menos a mayor.

A continuación, podremos ver que dicha vista se creó correctamente en la base de datos "Transactions\_s4":



Aquí podremos ver el resultado de la consulta a la view:

199 • `SELECT * FROM VistaRiesgoySeguridad;`

	card_id	Cant_declined	Status_card
▶	CcU-2938	0	Activas
	CcU-2945	1	Activas
	CcU-2952	1	Activas
	CcU-2959	0	Activas
	CcU-2966	1	Activas
	CcU-2973	1	Activas
	CcU-2980	1	Activas
	CcU-2987	1	Activas
	CcU-2994	0	Activas

151 09:38:03 `SELECT * FROM VistaRiesgoySeguridad` 275 row(s) returned

## EJERCICIO I

### ¿Cuántas tarjetas están activas?

Teniendo en cuenta el punto anterior, voy a calcular cuantas tarjetas se encuentran activas a través de la siguiente declaración:

206 • `SELECT count(*)`  
207 `FROM VistaRiesgoySeguridad`  
208 `WHERE Status_card = 'Activas';`  
209

	count(*)
▶	275

44 11:56:03 `SELECT count(*) FROM VistaRiesgoySeguridad WHERE Status_card = 'Activas'` 1 row(s) returned

- `SELECT count (*)`: función que cuenta el número total de filas. En este caso, al poner como parámetro (\*), estoy indicando que se cuenten todas las filas de la tabla que coinciden con las condiciones creadas mas adelante.
- `FROM VistaRiesgoySeguridad`: indico a través de la palabra FROM que los datos deben obtenerse de la tabla vista “VistaRiesgoySeguridad” creada en el punto anterior.
- `WHERE Status_card = 'Activas'`: con esta cláusula WHERE filtro los resultados de la vista para mostrar sólo aquellas que cumple con la condición que el campo Status\_card debe ser igual a 'Activas'.

El resultado de esta consulta muestra que hay 275 tarjetas activas.

### 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:**

Tal como se mencionó en el Nivel 1, vamos a crear una tabla puente que conectará la dimensión Products con la tabla de hechos Transactions. Esta tabla puente es necesaria porque la columna product\_ids en la tabla Transactions contiene una lista de identificadores de productos, lo que impide establecer una relación directa entre ambas tablas. Para solucionar este inconveniente, crearemos una tabla puente que registre cada combinación de transacción y producto, es decir, cada transacción estará asociada a un producto específico. Esta tabla puente facilitará el manejo de la relación muchos a muchos entre las transacciones y los productos, lo que permitirá representar de manera eficiente las relaciones entre estas dos entidades.

#### PASO 1: CREACION DE TABLA PUENTE O INTERMEDIA

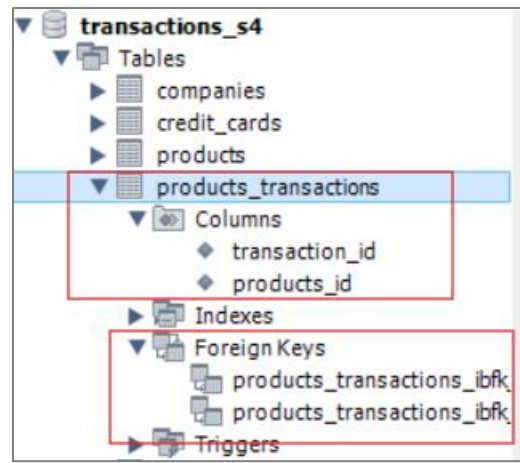
```
213 CREATE TABLE IF NOT EXISTS Products_Transactions (  
214     transaction_id VARCHAR (255),  
215     products_id INT,  
216     FOREIGN KEY (products_id) REFERENCES products (id),  
217     FOREIGN KEY (transaction_id) REFERENCES transactions (id)  
218 );
```

33 16:28:17 CREATE TABLE IF NOT EXISTS Products\_Transactions (transaction\_id VARCHAR (255), products\_id INT) 0 row(s) affected

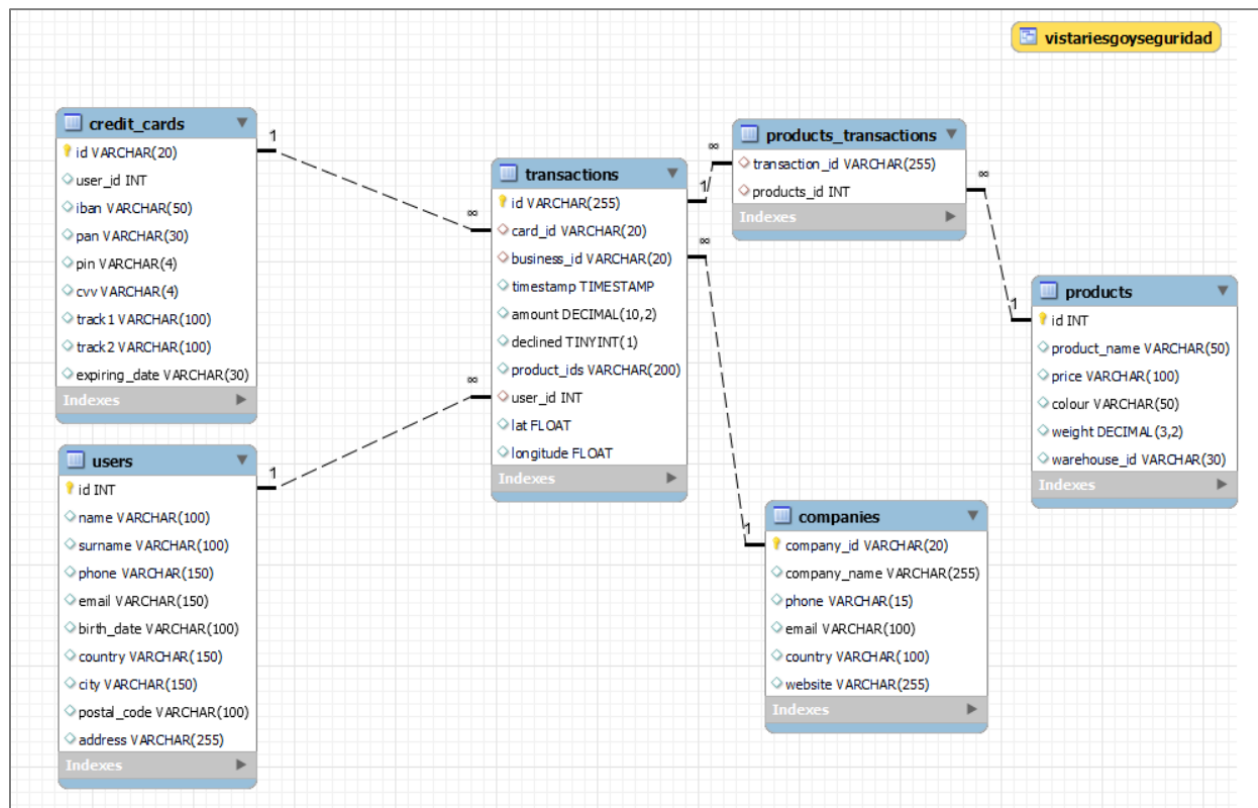
- **CREATE TABLE IF NOT EXISTS transactions:** declaración que indica la creación de una tabla en la base de datos, con la condición de que dicha tabla no existe. Luego de este primer parte, coloque el nombre de la nueva tabla a crear, en este caso es “Products\_Transactions”.
- **CREACION DE CAMPOS:** luego de la declaración, procedí a definir los campos que compondrán la tabla, teniendo en cuenta las primary key de las tablas Products y Transactions que se encuentra en la base de datos “Transactions\_s4”. Los campos creados con sus especificaciones son las siguientes:
  - **Transaction\_id VARCHAR (255):** identificador único de la transacción. Es de tipo VARCHAR con una longitud de hasta 255 caracteres, permitiendo letras, números y caracteres especiales.
  - **Products\_id INT:** identificador único de los productos. Es de tipo de valor entero (INT).
- **CREACION DE LAS FOREIGN KEY :**
  - **FOREIGN KEY (products\_id) REFERENCES products (id):** función que establece que el campo products\_id de la tabla Products\_Transactions debe coincidir con el campo id de la tabla products.
  - **FOREIGN KEY (transactions\_id) REFERENCES transactions (id):** función que establece que el campo transactions\_id de la tabla Products\_Transactions debe coincidir con el campo id de la tabla transactions.

A continuación, podemos ver que la creación de dicha tabla aparece en la base de datos “Transactions\_S4”:





## PASO 2: MODELO RELACIONAL



Aquí se observa cómo todas las tablas están correctamente relacionadas dentro del modelo relacional. La tabla **products\_transactions** actúa como una tabla puente, facilitando la conexión entre la tabla de hechos **transactions** y la tabla de dimensión **products**. Esta estructura permite gestionar de manera eficiente la relación muchos a muchos entre transacciones y productos, transformándola en relaciones uno a muchos a través de la tabla puente.

### PASO 3: CARGA DE DATOS

Para realizar dicha carga de datos, hice la siguiente declaración:

```
• INSERT INTO Products_Transactions (transaction_id, products_id)
  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;
```

121 15:37:09 INSERT INTO Products\_Transactions (transaction\_id, products\_id) SELECT transactions.id AS transa... 1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0

- **INSERT INTO Products\_Transactions (transaction\_id, products\_id):** declaración que se utiliza para insertar registros en una tabla. En este caso estamos indicando que queremos insertar datos a la tabla Products\_Transactions, en especial a las columnas transaction\_id y products\_id (parámetros).
- **SELECT transactions.id AS transaction\_id, products.id AS product\_id:** en esta parte de la consulta seleccione los campos de los registros que se quiere insertar en la nueva tabla. El orden de los valores debe ser el mismo que el orden de las columnas. En este caso los campos son:
  - **transactions.id AS transaction\_id:** identificador de las transacciones de la tabla transactions que lo renombre “transaction\_id” a través de la función AS para que coincida con la columna de la tabla Products\_Transactions.
  - **products.id AS product\_id:** identificador de los productos de la tabla products.id que lo renombre “product\_id” a través de la función AS para que coincida con la columna de la tabla Products\_Transactions.
- **FROM transactions:** indico a través de la palabra FROM que los datos deben obtenerse de la tabla vista “transactions” que sería la tabla principal.
- **JOIN products:** función que permite unir la tabla products.
- **ON FIND\_IN\_SET(products.id, REPLACE(transactions.product\_ids, ' ', '')) > 0:** esta parte de la consulta define la condición de unión entre las filas de la tabla transactions y las de la tabla products. En lugar de una relación directa basada en una clave primaria o foránea, se utiliza la función FIND\_IN\_SET. Esta función evalúa si el valor de products.id está presente en la lista delimitada por comas contenida en la columna transactions.product\_ids. Si encuentra una coincidencia, devuelve la posición del elemento dentro de la lista; de lo contrario, devuelve 0. Para garantizar un funcionamiento correcto, se aplica la función REPLACE a transactions.product\_ids, eliminando cualquier espacio en blanco presente en los valores. Esto es importante, ya que FIND\_IN\_SET no opera adecuadamente con listas que contienen espacios. Respecto a la condición >0, esta asegura que la consulta solo devuelve combinaciones donde el products.id se encuentra en transactions.product\_ids.

A continuación, comprobaremos que se insertaron correctamente los datos a la nueva tabla:

```
29 • SELECT * FROM Products_Transactions;
```

Result Grid | Filter Rows: | Export:

transaction_id	products_id
02C6201E-D90A-1859-B4EE-88D2986D3B02	71
02C6201E-D90A-1859-B4EE-88D2986D3B02	19
02C6201E-D90A-1859-B4EE-88D2986D3B02	1
0466A42E-47CF-8D24-FD01-C0B689713128	97
0466A42E-47CF-8D24-FD01-C0B689713128	47
0466A42E-47CF-8D24-FD01-C0B689713128	43

Products\_Transactions 23 x

122 15:38:38 SELECT \* FROM Products\_Transactions 1457 row(s) returned

## EJERCICIO I

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

```
235 • SELECT products_id, product_name, count(transaction_id) AS Cant_vendida, declined
236 FROM Products_Transactions
237 JOIN transactions
238 ON transaction_id=id
239 JOIN products
240 ON products_id=products.id
241 WHERE declined=0
242 GROUP BY 1,2,4
243 ORDER BY 1 ASC;
```

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

	products_id	product_name	Cant_vendida	declined
1	1	Direwolf Stannis	51	0
2	2	Tarly Stark	56	0
3	3	duel tourney Lannister	43	0
5	5	skywalker ewok	42	0
7	7	north of Casterly	44	0
11	11	Karstark Dorne	40	0
13	13	palpatine chewbacca	51	0
17	17	skywalker ewok sith	54	0
19	19	dooku solo	44	0

52 21:26:48 SELECT products\_id, product\_name, count(transaction\_id) AS Cant\_vendida, declined FROM Product... 26 row(s) returned

- SELECT products\_id, product\_name, count(transaction\_id) AS Cant\_vendida, declined: aquí seleccione los campos que son pertinentes para la consulta seleccionada:
- Products\_id
  - Producto\_name
  - Count(transaction\_id) AS Cant\_vendida=> este valor es el cálculo de la cantidad de las transacciones realizadas por cada producto.

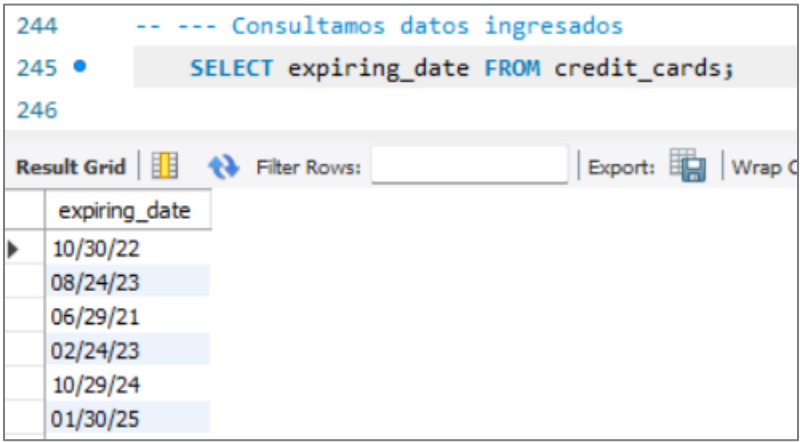
- Declined = > lo seleccione para corroborar que todas las consultas que aparezcan son aquellas transacciones aprobadas (0)
- `FROM Products_Transactions JOIN transactions ON transaction_id=id`: función que une la tabla Products\_Transactions con la tabla transactions a través de la clave foránea.
- `JOIN products ON products_id=products.id`: función que une la tabla Products\_Transactions con la tabla products a través de la clave foránea.
- `WHERE declined=0`: cláusula WHERE para enfocar los resultados sólo a transacciones aprobadas.
- `GROUP BY 1,2,4`: agrupa los campos por products\_id, producto\_name y declined para poder hacer el cálculo de la cantidad de transacciones por producto.
- `ORDER BY 1 ASC`: función que ordena los resultados por su products\_id de manera ascendente.

## ACTUALIZACIÓN DE FORMATO Y TIPO DE DATOS PENDIENTES

### 1) TABLA CREDIT CARDS

Al crear la tabla Credit\_cards al principio del sprint, comenté que temporalmente, definí el campo expiring\_date como VARCHAR (30) para admitir los datos de fecha que no coinciden con el formato DATE de SQL, con la intención de insertar inicialmente los datos y luego transformarlos y ajustar el tipo de dato a DATE cuando la estructura de los registros esté adaptada.

Antes de empezar con las modificaciones, mostraré como se encuentran los registros del campo expiring\_date:



expiring_date
10/30/22
08/24/23
06/29/21
02/24/23
10/29/24
01/30/25

128 19:20:22 SELECT expiring\_date FROM credit\_cards 275 row(s) returned

A continuación, procederé realizar los ajustes de dicho campo y registros:

- a) Conversión de los registros de la columna “expiring\_date”: en esta declaración lo que hago es estandarizar el formato de los registros que se encuentran en dicha columna, a un tipo de dato DATE. Para ello, realice las siguientes declaraciones:

```
247      -- ---- Convertir las fechas de strings a date
248 •    SET SQL_safe_updates = 0;
249
250 •    UPDATE credit_cards
251      SET expiring_date = STR_TO_DATE(expiring_date, '%m/%d/%y')
252      WHERE expiring_date IS NOT NULL;
253
254 •    SET SQL_safe_updates = 1;
255
```

✓	134	19:25:55	SET SQL_safe_updates = 0	0 row(s) affected
✓	135	19:25:59	UPDATE credit_cards SET expiring_date = STR_TO_DATE(expiring_date, '%m/%d/%y') WHERE expir...	275 row(s) affected Rows matched: 275 Changed: 275 Warnings: 0
✓	136	19:26:03	SET SQL_safe_updates = 1	0 row(s) affected

## PRIMER PASO

- **SET SQL\_safe\_updates = 0:** debido a que mi usuario tiene activado el modo seguro de actualizaciones, para poder realizar esta tarea, tuve que desactivar dicho modo a través de esta declaración. Donde establezco que la función SQL\_safe\_update es igual a 0, es decir lo desactivo.

## SEGUNDO PASO

- **UPDATE credit\_cards:** declaración que especifica que se desea modificar los registros existentes en la tabla *credit\_cards*.
- **SET expiring\_date = STR\_TO\_DATE(expiring\_date, '%m/%d/%y');** establece que la columna *expiring\_date* debe ser actualizada utilizando la función **STR\_TO\_DATE**, la cual transforma los valores de la columna a un tipo de dato **DATE**. Dentro de la función, se especifica que los valores de *expiring\_date* deben interpretarse con el siguiente formato de fecha:
  - %m: mes
  - %d: día
  - %y: año.
- **WHERE expiring\_date IS NOT NULL:** aplique la cláusula **WHERE** para limitar la actualización de los registros de la columna *expiring\_date* a aquellos que tienen un valor, es decir que no sean **NULL**.

En conclusión, esta declaración convierte los registros que tienen un formato de fecha como 10/30/2024 en el formato estándar **DATE**, resultando en 2024-10-30

## TERCER PASO

- **SET SQL\_safe\_updates = 1:** luego de haber realizado la actualización de los registros del campo *expiring\_date*, vuelvo a activar el modo seguro de actualizaciones en SQL. Es la misma declaración que el primer paso, pero esta vez la igualación de la función **SQL\_safe\_update** es igual a 1, es decir lo activo.

## COMPROBACION

Una vez realizado los pasos anteriores, lo que hice fue comprobar que efectivamente los registros del campo *expiring\_date* fueron actualizados, para ello hice la siguiente declaración:

```
256 • SELECT expiring_date FROM credit_cards;
```

expiring_date
2022-10-30
2023-08-24
2021-06-29
2023-02-24
2024-10-29
2025-01-30

✓ 139 19:27:25 SELECT expiring\_date FROM credit\_cards 275 row(s) returned

- b) Modificación del tipo de dato de la columna “expiring\_date”: una vez transformado los registros de la columna “expiring\_date” a un formato compatible con el tipo de dato DATE, lo que queda para finalizar con esta tarea es modificar la información del tipo de dato de dicha columna. Es decir, de cambiar VARCHAR (30) a DATE. Para ello, hice la siguiente declaración:

```
259 -- ----- Modificacion del tipo de dato de la columna 'expiring_date'
260 • ALTER TABLE credit_cards
261     MODIFY COLUMN expiring_date DATE;
262
```

✓ 141 19:28:29 ALTER TABLE credit\_cards MODIFY COLUMN expiring\_date DATE 275 row(s) affected Records: 275 Duplicates: 0 Warnings: 0

- **ALTER TABLE credit\_cards**: cláusula que indica que se va a realizar una modificación en la estructura de la tabla credit\_card.
- **MODIFY COLUMN expiring\_date DATE**: especifica que la modificación se centrará en la columna expiring\_date, transformándola en un tipo de dato DATE.

A continuación, comprobamos dicha modificación:

Table: credit_cards	
Columns:	
id	varchar(20) PK
user_id	int
iban	varchar(50)
pan	varchar(30)
pin	varchar(4)
cvv	varchar(4)
track1	varchar(100)
track2	varchar(100)
expiring_date	date



## 2) TABLA PRODUCTS

Al querer insertar los datos en la tabla products, surgió un error debido a que en la columna *price*, los datos a insertar contienen el símbolo de moneda (\$), lo cual impide la inserción en un campo definido como DECIMAL (10,2). Para resolver esto, había modificado temporalmente el tipo de la columna *price* a VARCHAR (100), permitiendo así la carga de datos en la tabla. Por lo tanto, a continuación, realizaré las modificaciones necesarias para limpiar los símbolos de moneda y devolver la columna *price* a su tipo original DECIMAL (10,2), asegurando la integridad del tipo de datos.

Antes de empezar con las modificaciones, mostraré como se encuentran los registros del campo *price*:

```
264 -- --- Consultamos datos ingresados
265 • SELECT price FROM products;
266
```

price
\$161.11
\$9.24
\$171.13
\$71.89
\$171.22
\$136.60
\$63.33

142 19:50:52 SELECT price FROM products 100 row(s) returned

A continuación, procederé realizar los ajustes de dicho campo y registros:

- a) Eliminación del símbolo de la moneda (\$) de los registros de la columna “*price*”: en esta declaración lo que hago es estandarizar el formato de los registros que se encuentran en dicha columna, a un tipo de dato numérico. Para ello, realice las siguientes declaraciones:

```
267 -- ---- Eliminación de los signos de moneda
268 • SET SQL_safe_updates = 0;
269
270 • UPDATE products
271     SET price = REPLACE (price, '$', '')
272     WHERE price IS NOT NULL;
273
274 • SET SQL_safe_updates = 1;
```

143 19:59:18 SET SQL_safe_updates = 0	0 row(s) affected
144 19:59:24 UPDATE products SET price = REPLACE (price, '\$', '') WHERE price IS NOT NULL	100 row(s) affected Rows matched: 100 Changed: 100 Warnings: 0
145 19:59:28 SET SQL_safe_updates = 1	0 row(s) affected

## PRIMER PASO

- `SET SQL_safe_updates = 0;` debido a que mi usuario tiene activado el modo seguro de actualizaciones, para poder realizar esta tarea, tuve que desactivar dicho modo a través de esta declaración. Donde establezco que la función `SQL_safe_update` es igual a 0, es decir lo desactivo.

## SEGUNDO PASO

- `UPDATE products;` declaración que especifica que se desea modificar los registros existentes en la tabla `products`.
- `SET price = REPLACE (price, '$', '');` establece que la columna `price` debe ser actualizada utilizando la función `REPLACE`, la cual busca el carácter `$` en la columna `price` y lo reemplaza con una cadena vacía, eliminando dicho signo moneda.
- `WHERE expiring_date IS NOT NULL;` aplique la cláusula `WHERE` para limitar la actualización de los registros de la columna `price` a aquellos que tienen un valor, es decir que no sean `NULL`.

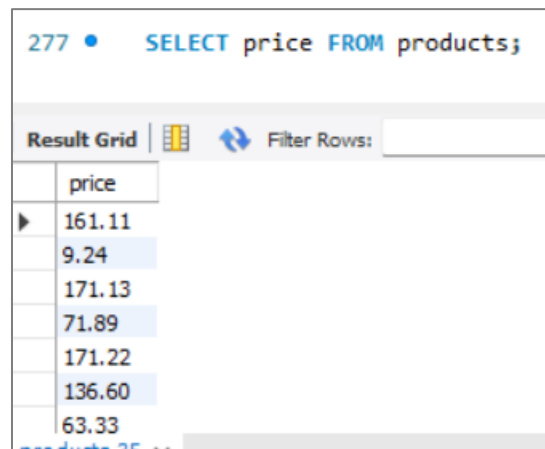
En conclusión, esta declaración convierte los registros que tienen un formato no numérico como `$161.11` en el formato estándar numérico, resultando en `161.11`

## TERCER PASO

- `SET SQL_safe_updates = 1;` luego de haber realizado la actualización de los registros del campo `price`, vuelvo a activar el modo seguro de actualizaciones en SQL. Es la misma declaración que el primer paso, pero esta vez la igualación de la función `SQL_safe_update` es igual a 1, es decir lo activo.

## COMPROBACION

Una vez realizado los pasos anteriores, lo que hice fue comprobar que efectivamente los registros del campo `price` fueron actualizados, para ello hice la siguiente declaración:



price
161.11
9.24
171.13
71.89
171.22
136.60
63.33

146 19:59:34 SELECT price FROM products 100 row(s) returned

- b) Modificación del tipo de dato de la columna “price”: una vez transformado los registros de la columna “price” a un formato compatible con el tipo de dato `DECIMAL`, lo que queda para finalizar con esta tarea es modificar la información del tipo de dato de dicha columna. Es decir, de cambiar `VARCHAR (100)` a `DECIMAL (10,2)`. Para ello, hice la siguiente declaración:

```
279 -- ----- Modificacion del tipo de dato de la columna 'price'
280 • ALTER TABLE products
281 MODIFY COLUMN price DECIMAL(10, 2);
```

147 20:08:43 ALTER TABLE products MODIFY COLUMN price DECIMAL(10, 2) 100 row(s) affected Records: 100 Duplicates: 0 Warnings: 0

- **ALTER TABLE products:** cláusula que indica que se va a realizar una modificación en la estructura de la tabla products.
- **MODIFY COLUMN price DECIMAL (10,2):** especifica que la modificación se centrará en la columna price, transformándola en un tipo de dato DECIMAL.

A continuación, comprobamos dicha modificación:

Table: products	
Columns:	
id	int PK
product_name	varchar(50)
price	decimal(10,2)
colour	varchar(50)
weight	decimal(3,2)
warehouse_id	varchar(30)