

SPRINT 4 – Data Analytics

A continuación, se presentan los resultados del Sprint #4 de la especialización Data Analytics de IT Academy, realizados por Rossemary Castellanos entregado el día 14/10/2025. Se realizaron los 3 niveles del Sprint.

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

Los archivos .csv son los siguiente:

- "american_users.csv"
- "companies.csv"
- "credit_cards.csv"
- "european_users.csv"
- "products.csv"
- "transactions.csv"

NIVEL 1

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:

Lo primero a realizar es un análisis del tipo de datos presentes en los archivos de manera de poder definir las tablas a crear y tipo de relación entre ellas, para ello se visualizaron en Excel aprovechando la fácil analogía con el formato CSV.

Resalta el hecho al ver los nombres de los archivos, que hay parecidos con el tipo de base de datos con la que se trabajó en los sprints anteriores: *transactions.csv*, *american_users.csv*, *companies.csv*, *credit_cards.csv*, *european_users.csv*, *products.csv*. Esta observación permite inferir que probablemente se trata de una plataforma de ventas digitales, por lo cual se tomara como nombre de la base de datos **marketplace**.

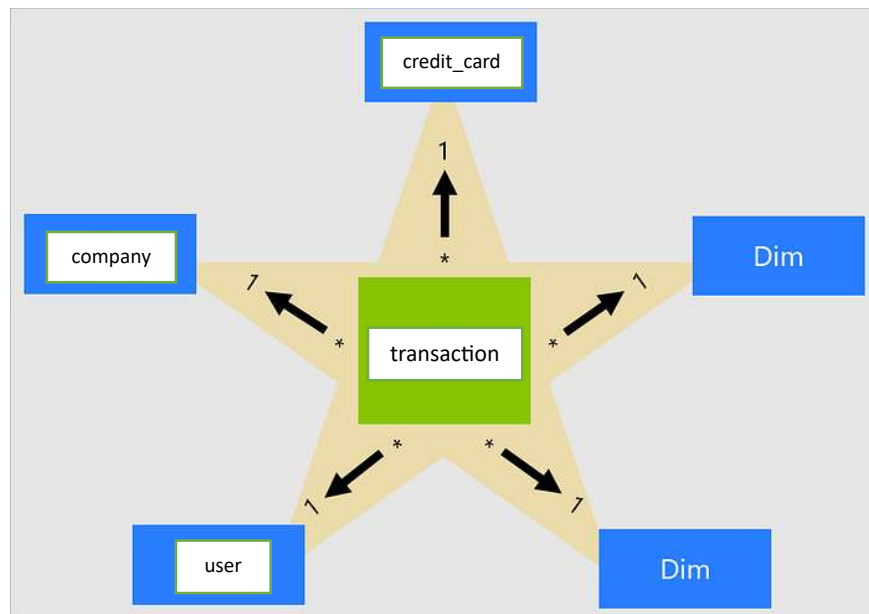
Basados en estas similitudes si analizamos la basa de datos **transactions** de los Sprint 2 y 3, la tabla central era la tabla **transaction**, esto se asocia a una tabla de hechos la cual se rodea de otras tablas (tablas de dimensiones), que proporcionan el contexto para analizar, permitiendo filtrar, agrupar y explorar los datos.

Por tal motivo nos enfocamos primero en visualizar las columnas en particular del archivo *trasactions*, donde se observan campos como *id*, *card_id*, *business_id*, *product_ids* y *user_id*; que se relacionan precisamente con los nombres del resto de archivos .csv.

De esta manera se establece que la tabla de hechos seria la formada por el archivo *trasactions* y en torno a la misma se diseñan y estructuran el resto de las tablas de dimensiones derivadas de forma general por los archivos *credit_cards*, *companies*, *users* tanto americanos como europeos y finalmente *products*.

Lo descrito antes permite diseñar la base de datos en un esquema de estrella:

- Tabla **transaction**, tabla de hecho proveniente de *transactions.csv*.
- Tabla **credit_card**, tabla de dimensión proveniente de *credit_card.csv*.
- Tabla **company**, tabla de dimensión proveniente de *companies.csv*.
- Tabla **user**, tabla de dimensión proveniente de *american_users.csv*, *european_users.csv*.



Para comenzar primero se procede a crear la base de datos **marketplace** con el comando CREATE DATABASE. Luego se crea la estructura de la tabla de hecho **transaction**, basados en los formatos usados en sprint anteriores como punto de inicio. A continuación, se listan los campos de la tabla:

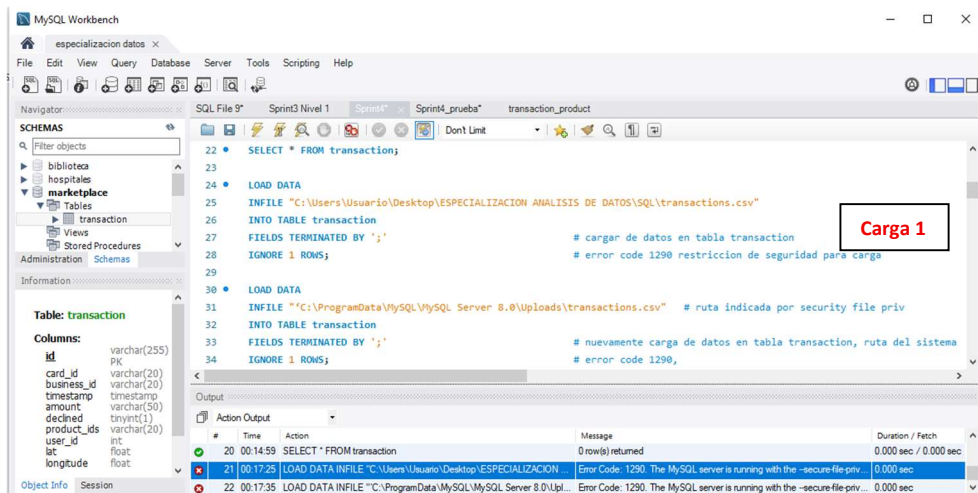
- *id*: identificación de la transacción como clave primaria y única,
- *credit_card_id*: número identificador de la tarjeta de crédito utilizada,
- *business_id*: identificador de la compañía que vende,
- *timestamp*: el momento en que se realiza la operación (fecha y hora),
- *amount*: el importe de la venta,
- *declined*: verificación de proceder o no con la venta,
- *product_ids*: números identificadores de los productos vendidos,
- *user_id*: identificador de usuario,
- *lat, longitude*: ubicación referida como latitud y longitud respectivamente.

```

1  # SPRINT 4
2
3  # Nivel 1.
4  # Descarga los archivos CSV, estudiales y diseña una base de datos con un esquema de estrella que contenga,
5  # al menos 4 tablas de las que puedas realizar las siguientes consultas:
6
7  # Se visualizaron los archivos csv y los campos que contenian, inferiendo que se trata de una plataforma de ventas digitales
8  • CREATE DATABASE marketplace;
9
10 • CREATE TABLE transaction (
11     id VARCHAR(255) PRIMARY KEY,
12     card_id VARCHAR(20),
13     business_id VARCHAR(20),
14     timestamp TIMESTAMP,
15     amount VARCHAR(50),
16     declined BOOLEAN,
17     product_ids VARCHAR(20),
18     user_id INTEGER,
19     lat FLOAT,
20     longitude FLOAT);
  
```

#	Time	Action	Message	Duration / Fetch
14	12:23:49	CREATE DATABASE marketplace	1 row(s) affected	0.000 sec
15	12:30:02	CREATE TABLE transaction (# el archivo transactions genera la tabla de hecho...	0 row(s) affected	0.046 sec

Ahora procedemos a llenar la tabla **transaction** con los datos del archivo *transactions.csv*, para lo cual se usa el comando **LOAD DATA INFILE**, se identifica la ruta donde se ubica el archivo, la tabla donde serán cargados los datos, así como el tipo de delimitador entre registros (en este caso el delimitador es “;”).

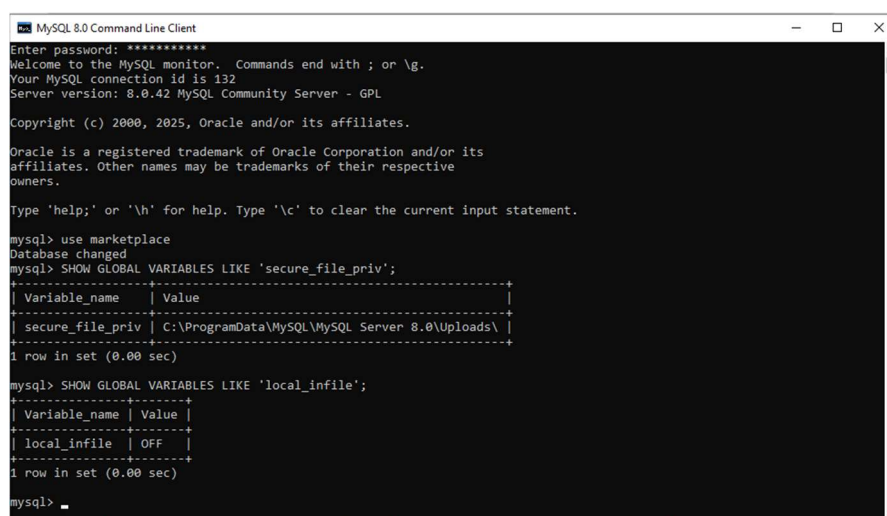


Al ejecutar la sentencia de carga identificada en la imagen como carga 1, se genera el *error code 1290*, asociado a la opción de configuración *--secure-file-priv*, parámetro de seguridad que controla el directorio desde el cual MySQL cargar y descargar archivos, restringiendo la ruta para interactuar con archivos a una definida como medida de protección.

Primero debemos identificar la ruta definida por el sistema para cargar y descargar archivo a través de un terminal mysql donde:

- Se introduce el *password* para MySQL,
- ingresa a la base de datos **marketplace** con el comando **USE**,
- ejecuta el comando **SHOW GLOBAL VARIABLES LIKE 'secure_file_priv'**, para identificar el directorio permitido por el sistema para cargar y descargar archivos.

La terminal muestra la ruta permitida para interactuar con archivos, se puede utilizar este directorio y copiar aquí los archivos o la otra opción es modificar el directorio donde se desea que apunten la ubicación de los archivos.



Para cambiar la ruta se accede como administrador para editar el archivo *my.ini* que contiene los datos de configuración de MySQL, se modifica lo relacionado a la ruta en *--secure-file-priv*.

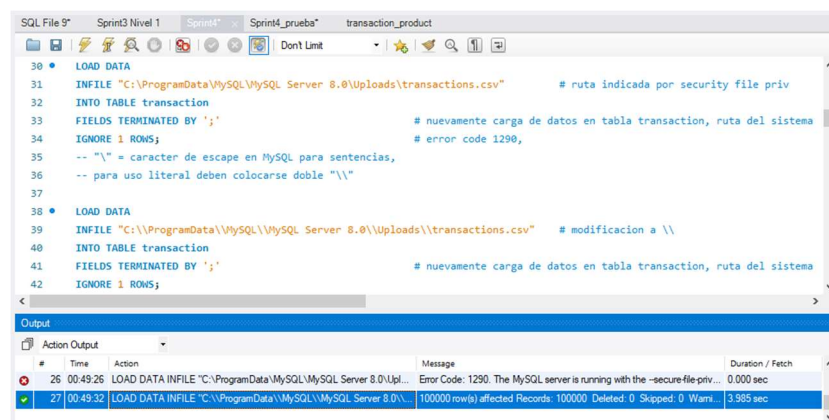
Existe otra variable de seguridad que se debe verificar '*local_infile*', esta controla que MySQL permita o rechace carga de datos locales desde los clientes, se ejecuta el comando *SHOW GLOBAL VARIABLES LIKE 'local_infile'*, para identificar su status el cual cambia según la versión de MySQL que se tenga instalada. En OFF esta deshabilitada y en ON habilitada. Se debe activar en caso de estar apagada modificando el archivo *my.ini*, (insertando *LOCAL INFILE = 1*).

Para este Sprint se decidió mover los archivos al directorio establecido que indica la terminal: '*C:\ProgramData\MySQL\MySQL Server 8.0\Uploads\archivo.csv*'. Esto con la finalidad de intentar no alterar la configuración de seguridad que establece el sistema en la medida de lo posible.

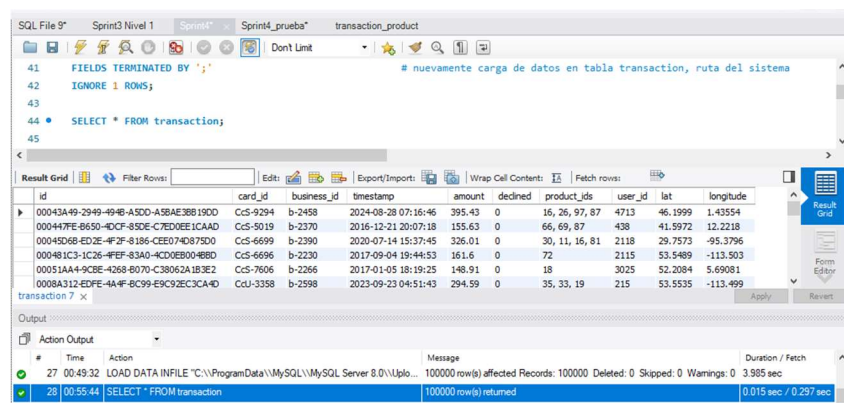
Se procedió a modificar la ruta en la sentencia, generando nuevamente error como se aprecia en la imagen anexa, según la documentación "La barra invertida es el carácter de escape de MySQL dentro de las cadenas en las sentencias SQL... para especificar una barra invertida literal, se deben especificar dos barras invertidas para que el valor se interprete como una sola".

Los valores predeterminados para el comando *LOAD DATA* hace que al leer las entradas interprete los caracteres precedidos por el carácter de escape (**), como secuencias de escape.

Se aplican los cambios a la ruta del directorio duplicando las barras invertidas, "*C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\transactions.csv*", realizándose la carga de forma exitosa como lo indica la imagen.



A continuación, se presenta la imagen de la tabla **transaction** con los datos provenientes del archivo *transactions.csv*.

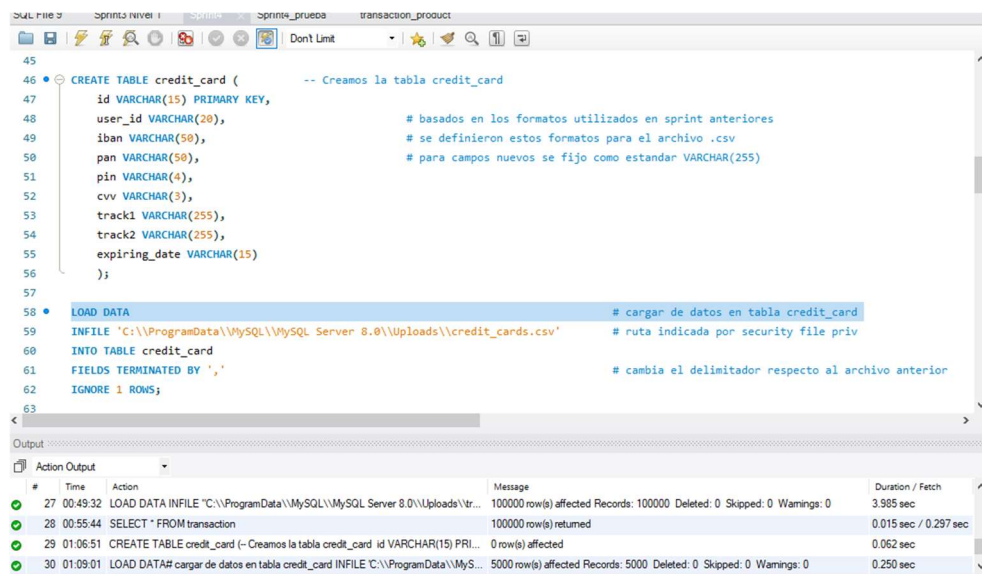


Para las siguientes tablas y archivos se realizó de forma similar a lo visto anteriormente, se diseñaron las estructuras de las tablas a partir de los archivos .csv que serían cargados a posterior.

Se inicio con la primera tabla de dimensiones, la tabla **credit_card**, que contendrá los datos de las tarjetas usadas para pagar las transacciones, se listan a continuación los campos diseñados a partir del archivo *credit_cards.csv*:

- *id*: número identificador del código de la tarjeta que sería la clave primaria y única, debemos revisar que el formato coincida con *credit_card_id* de la tabla **transaction**,
- *user_id*: número identificador del usuario,
- *iban*: alfanumérico, corresponde a la cuenta iban asociado a la tarjeta,
- *pan*: serial identificador único de la tarjeta,
- *pin*: código clave de la tarjeta,
- *cvv*: código de seguridad de la tarjeta,
- *track1*, *track2*: campos desconocidos se asociaron a formato VARCHAR (255),
- *expiring_date*: fecha de caducidad de la tarjeta.

En la imagen siguiente se muestran las sentencias para la creación y carga de datos de la tabla **credit_card** del archivo *credit_cards.csv*. Se identifico la Primary Key como *id*, verificando su similitud con el campo *card_id* de la tabla de hechos **transaction**. Para los formatos como ya se mencionó se basó en los usados en sprint anteriores y para campos desconocidos se utilizó VARCHAR (255). En la carga de datos el delimitador cambio para este archivo csv (",").



```
45
46 CREATE TABLE credit_card (          -- Creamos la tabla credit_card
47     id VARCHAR(15) PRIMARY KEY,
48     user_id VARCHAR(20),
49     iban VARCHAR(50),                 # basados en los formatos utilizados en sprint anteriores
50     pan VARCHAR(50),                 # se definieron estos formatos para el archivo .csv
51     pin VARCHAR(4),                  # para campos nuevos se fijo como estandar VARCHAR(255)
52     cvv VARCHAR(3),
53     track1 VARCHAR(255),
54     track2 VARCHAR(255),
55     expiring_date VARCHAR(15)
56 );
57
58 LOAD DATA                           # cargar de datos en tabla credit_card
59 INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\credit_cards.csv' # ruta indicada por security file priv
60 INTO TABLE credit_card
61 FIELDS TERMINATED BY ','             # cambia el delimitador respecto al archivo anterior
62 IGNORE 1 ROWS;
63
```

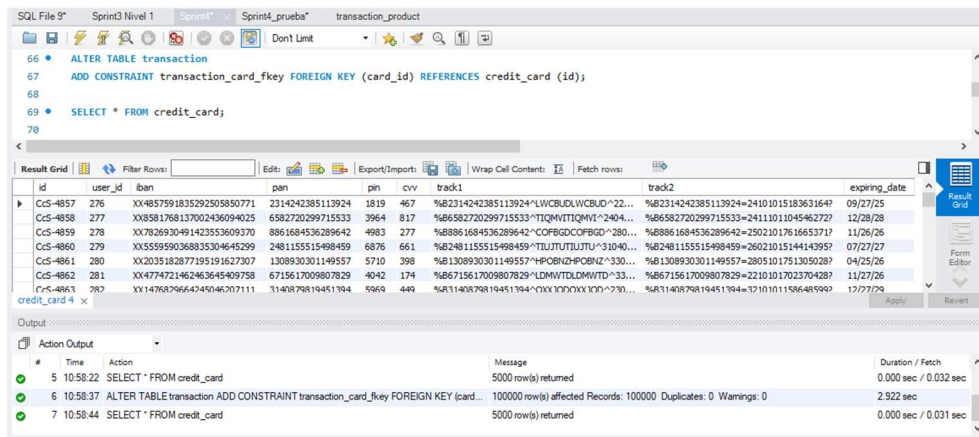
#	Time	Action	Message	Duration / Fetch
27	00:49:32	LOAD DATA INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\tr...	100000 row(s) affected Records: 100000 Deleted: 0 Skipped: 0 Warnings: 0	3.985 sec
28	00:55:44	SELECT * FROM transaction	100000 row(s) returned	0.015 sec / 0.297 sec
29	01:06:51	CREATE TABLE credit_card (- Creamos la tabla credit_card id VARCHAR(15) PRI...	0 row(s) affected	0.062 sec
30	01:09:01	LOAD DATA cargar de datos en tabla credit_card INFILE 'C:\\ProgramData\\MyS...	5000 row(s) affected Records: 5000 Deleted: 0 Skipped: 0 Warnings: 0	0.250 sec

Siendo esta la primera tabla de dimensiones generada, se debe establecer la relación con la tabla de hechos **transaction**.

La relación entre ambas tablas es de 1: N, uno a muchos, donde con una tarjeta (*credit_card_id*) puedo realizar varios pedidos o transacciones, pero una transacción es gestionada por una única tarjeta.

Se establece el vínculo entre las tablas (**credit_card** y **transaction**) a través de la sentencia conformada por los comandos ALTER TABLE, ADD CONSTRAINT, FOREIGN KEY y REFERENCES. De esta forma se crea una restricción de clave foránea (*transaction_card_fkey*) para enlazar las dos tablas, garantizando que los valores de una columna en una de las tablas existan como clave

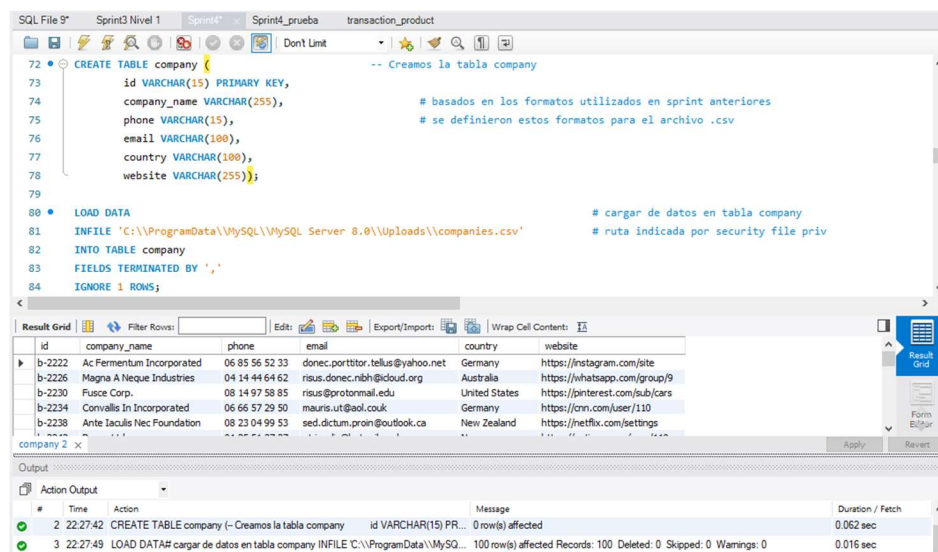
principal en la otra tabla, específicamente, como se asocia a la recién definida Foreign Key (**transaction.credit_card_id**) a la Primary Key (**credit_card.id**) de otra tabla. La imagen siguiente muestra la ejecución de la sentencia.



La siguiente tabla de dimensiones a crear es la tabla **company** que contendrá los datos de las empresas que venden u ofrecen productos, donde los campos definidos por medio del archivo **companies.csv** son:

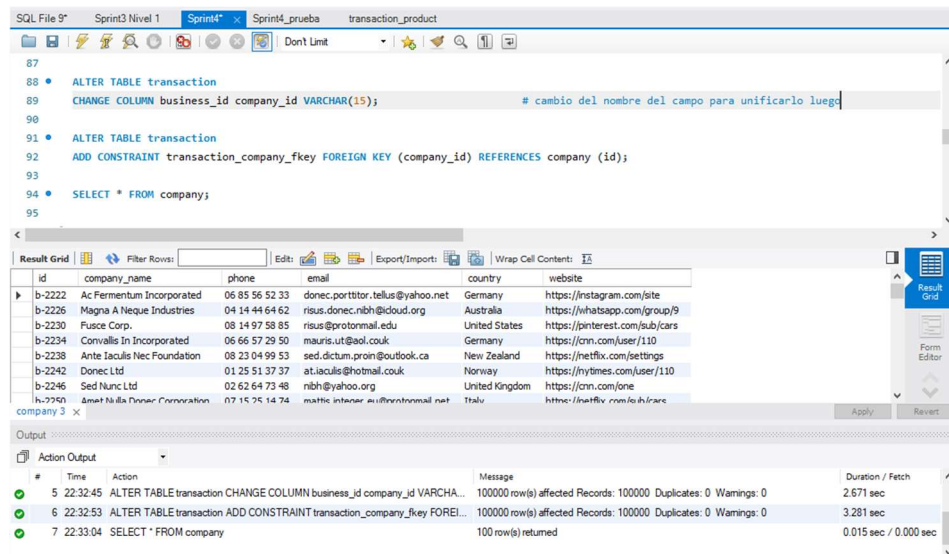
- **id**: número identificador del código de la empresa como clave primaria y única,
- **company_name**: nombre de la empresa,
- **phone**, **email**: referencias de contacto como teléfono y correo electrónico,
- **country**: información del país de ubicación, y
- **website**: su página web.

Al igual que la tabla anterior se creó la estructura de la tabla **company** y se cargaron los datos para llenar la tabla. Se identificó la Primary Key como **id**, verificando su similitud con un campo en la tabla de hechos **transaction**. En este caso hay coincidencia con el campo **business_id**, así se puede definir que la relación entre ambas tablas es de 1: N, uno a muchos, donde una empresa puede generar diversas ventas, pero una transacción producto de una única empresa.



Lo primero es unificar el nombre del campo que será el vínculo entre ambas tablas, cambiando con el comando ALTER y RENAME de **business_id** a **company_id**.

En la sentencia ADD CONSTRAINT añade la restricción (*transaction_company_fkey*), FOREIGN KEY especifica la columna que actúa como foránea en la tabla de hechos (*transaction.company_id*), y REFERENCES indica la tabla y columna principal con la que se está relacionando (*company.id*).



Ahora se creará la última tabla de dimensiones relacionada a los datos de los usuarios. En este caso se tienen 2 archivos de usuarios separados en región geográfica, *american_users.csv* y *european_users.csv*, por lo que se pueden realizar estructurar de diversas maneras. Analizando primero ambos archivos se observa que el nombre de los campos es igual en los 2 archivos:

- *id*: número identificador del código del usuario, asignado como Primary Key, se debe verificar que el formato coincida con *user_id* de la tabla **transaction**.
- *name*: nombre del usuario,
- *surname*: apellido del usuario,
- *phone*: teléfono del usuario,
- *email*: correo electrónico del usuario,
- *birth_date*: fecha de nacimiento del usuario,
- *country*: país donde se localiza o reside el usuario,
- *city*: ciudad donde se localiza o reside el usuario,
- *postal_code*: código postal del lugar donde se localiza o reside el usuario y
- *address*: dirección del sitio donde se localiza o reside el usuario.

Lo segundo es definir si se establecen 2 tablas por separado o se unifican en una única tabla **user**, esto último será lo que se realice, debido a que se tienen campos coincidentes, la numeración del identificador *id* es diferente en ambas tablas por lo que no se tendría problemas al momento de unirlos. Otro factor que se puede considerar es crear una columna adicional que identifique el continente, por si se desea realizar a futuro en la base de datos búsquedas o análisis por área geográfica.

Se utilizar el mismo procedimiento que antes, se crea la tabla **user** que tendrá los datos de todos los usuarios, considerando los formatos de los campos usados en sprint anteriores. Después se generan 2 sentencias para cargar los archivos *american_users.csv* y *european_users.csv*. En la imagen anexa se muestra el resultado.

```

95
96 CREATE TABLE user (           -- Creamos la tabla user
97     id INTEGER PRIMARY KEY,
98     name VARCHAR(100),          # basados en los formatos utilizados en sprint anteriores
99     surname VARCHAR(100),       # se definieron estos formatos para el archivo .csv
100    phone VARCHAR(150),
101    email VARCHAR(150),
102    birth_date VARCHAR(100),
103    country VARCHAR(150),
104    city VARCHAR(150),
105    postal_code VARCHAR(100),
106    address VARCHAR(255)
107 );
108
109 LOAD DATA                      # cargar de datos en tabla user
110 INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\european_users.csv' # archivo usuarios europeos
111 INTO TABLE user               # ruta indicada por security file priv
112 FIELDS TERMINATED BY ','
113 IGNORE 1 ROWS;
114
--

```

#	Time	Action	Message	Duration / Fetch
17	01:22:09	CREATE TABLE user (- Creamos la tabla user id INTEGER PRIMARY KEY, name...	0 row(s) affected	0.047 sec
18	01:22:17	LOAD DATA# cargar de datos en tabla user INFILE 'C:\\ProgramData\\MySQL\\...	Error Code: 1262. Row 1 was truncated: it contained more data than there were inp...	0.000 sec

La carga de datos arroja el error code 1262, asociado a la incongruencia entre numero de columnas entre la tabla y el archivo.csv. Analizando en detalle el archivo .csv se observa algunos campos con registros entre comillas (""), debido a que dentro de los datos se tiene comas (,). Como el delimitar del archivo es una coma (,) también genera conflicto.

El comando LOAD INFILE considera como clausula además de definir los delimitadores (FIELDS TERMINATED BY), establecer si los valores esta entre comillas u otro símbolo (ENCLOSED BY). En este caso no son todos los valores, la documentación indica el uso de OPTIONALLY ENCLOSED BY para considerar algunos registros y no todos. En la imagen anexa se presenta la ejecución de las sentencias de carga exitosa.

```

114 # error 1262 diferencia de campos entre archivo y tabla
115 -- algunos registros tiene campos entre comillas (") por uso de coma (,) dentro de los datos
116
117 LOAD DATA                      # nueva carga de datos en tabla user
118 INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\european_users.csv' # archivo usuarios europeos
119 INTO TABLE user               # ruta indicada por security file priv
120 FIELDS TERMINATED BY ','
121 OPTIONALLY ENCLOSED BY '"'      # comando para considerar algunos campos entre comillas "" no todos
122 IGNORE 1 ROWS;
123
124 LOAD DATA                      # cargar de datos en tabla user
125 INFILE 'C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\american_users.csv' # archivo usuarios americanos
126 INTO TABLE user               # ruta indicada por security file priv
127 FIELDS TERMINATED BY ','
128 OPTIONALLY ENCLOSED BY '"'
129 IGNORE 1 ROWS;
130
131 SELECT * FROM user;
132
--

```

#	Time	Action	Message	Duration / Fetch
18	01:22:17	LOAD DATA# cargar de datos en tabla user INFILE 'C:\\ProgramData\\MySQL\\...	Error Code: 1262. Row 1 was truncated: it contained more data than there were inp...	0.000 sec
19	01:28:34	LOAD DATA# nueva carga de datos en tabla user INFILE 'C:\\ProgramData\\MySQL\\...	3990 row(s) affected Records: 3990 Deleted: 0 Skipped: 0 Warnings: 0	0.172 sec
20	01:28:48	LOAD DATA# cargar de datos en tabla user INFILE 'C:\\ProgramData\\MySQL\\...	1010 row(s) affected Records: 1010 Deleted: 0 Skipped: 0 Warnings: 0	0.047 sec

Ahora se debe establecer la relación entre la tabla **transaction** y **user**, definida de 1: N, uno a muchos, ya que un usuario puede realizar varias transacciones, pero una transacción es ejecutada por un único usuario. A continuación, se muestra la imagen donde presentan los datos cargados en la tabla **user** y la aplicación de la sentencia para realizar el vínculo de ambas tablas. Como con las tablas anteriores se crea la restricción (*transaction_user_fk*) y se determina la Foreign Key (**transaction.user_id**) y la Primary Key referenciada (**user.id**).

The screenshot shows a SQL IDE with a query window containing the following SQL commands:

```

131 SELECT * FROM user;
132
133 ALTER TABLE transaction
134 ADD CONSTRAINT transaction_user_fkey FOREIGN KEY (user_id) REFERENCES user (id);
135
136 SELECT * FROM user;
137

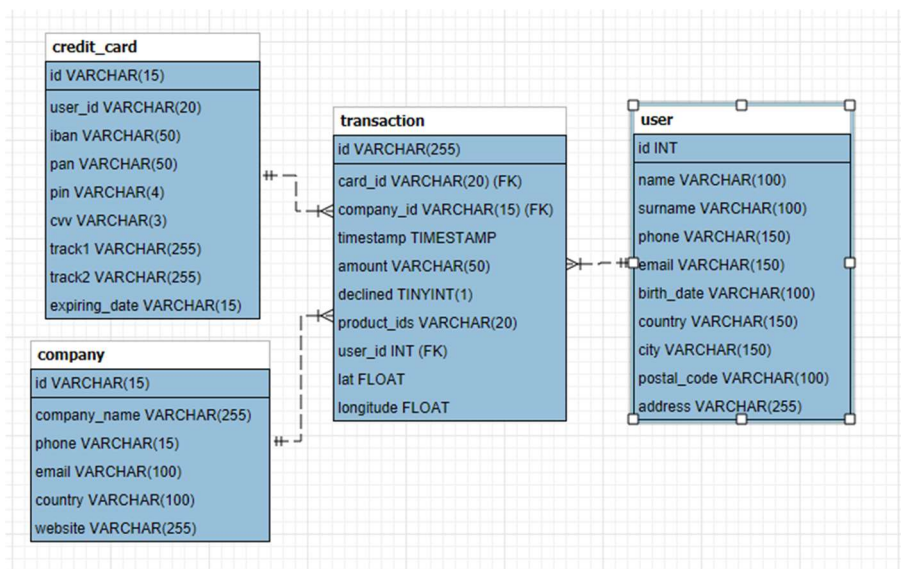
```

Below the query window is the 'Result Grid' showing a table with 10 columns: id, name, surname, phone, email, birth_date, country, city, postal_code, and address. The table contains 10 rows of user data.

At the bottom, the 'Output' window shows the execution log:

#	Time	Action	Message	Duration / Fetch
21	01:29:51	SELECT * FROM user	5000 row(s) returned	0.000 sec / 0.015 sec
22	01:32:52	ALTER TABLE transaction ADD CONSTRAINT transaction_user_fkey FOREIGN K...	100000 row(s) affected Records: 100000 Duplicates: 0 Warnings: 0	4.297 sec
23	01:32:59	SELECT * FROM user	5000 row(s) returned	0.000 sec / 0.016 sec

Ya con las tablas cargadas en la base de datos **marketplace** y las relaciones entre las tablas definidas se muestra el esquema finalizado.

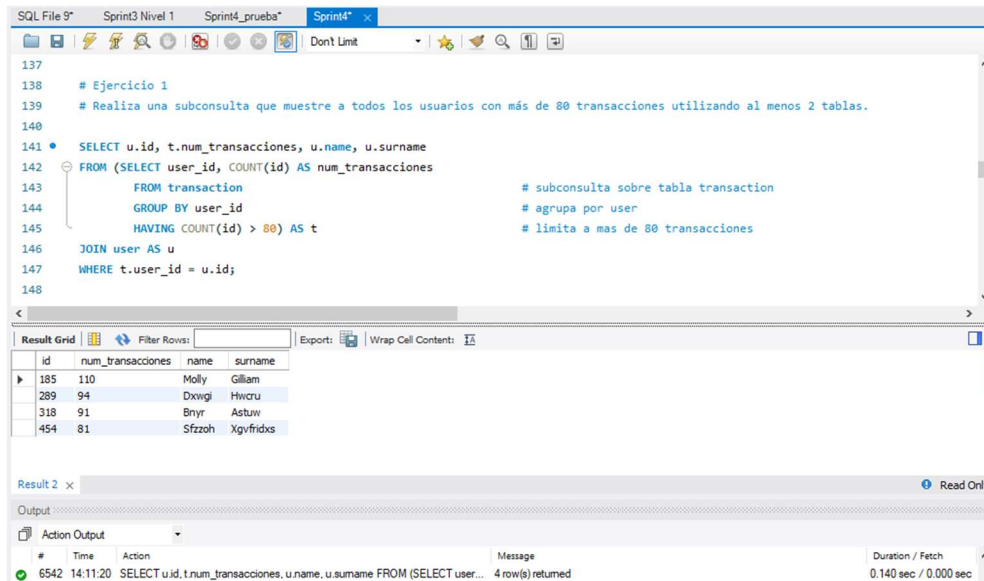


Ejercicio 1

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

El enunciado pide una subconsulta, sin embargo, si se puede evitar el uso de subconsulta y aplicar un JOIN es lo más acertado. De igual forma como la búsqueda necesita trabajar con las tablas **transaction** y **user** se debe aplicar el comando JOIN. Como se pide aplicar una subconsulta se hace con los datos requeridos de la tabla **transaction** donde se agrupa por usuarios y se aplica filtro de restricción a > 80 transacciones.

Se presenta en la imagen la consulta y resultados, 4 usuarios con más de 80 transacciones.



```
137
138 # Ejercicio 1
139 # Realiza una subconsulta que muestre a todos los usuarios con más de 80 transacciones utilizando al menos 2 tablas.
140
141 • SELECT u.id, t.num_transacciones, u.name, u.surname
142     FROM (SELECT user_id, COUNT(id) AS num_transacciones
143           FROM transaction
144           GROUP BY user_id
145           HAVING COUNT(id) > 80) AS t
146     JOIN user AS u
147     WHERE t.user_id = u.id;
148
```

id	num_transacciones	name	surname
185	110	Molly	Gillam
289	94	Dxwgl	Hwrcu
318	91	Bnyr	Astuw
454	81	Sfzzh	Xgvfricks

Result 2 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
6542	14:11:20	SELECT u.id, t.num_transacciones, u.name, u.surname FROM (SELECT user...	4 row(s) returned	0.140 sec / 0.000 sec

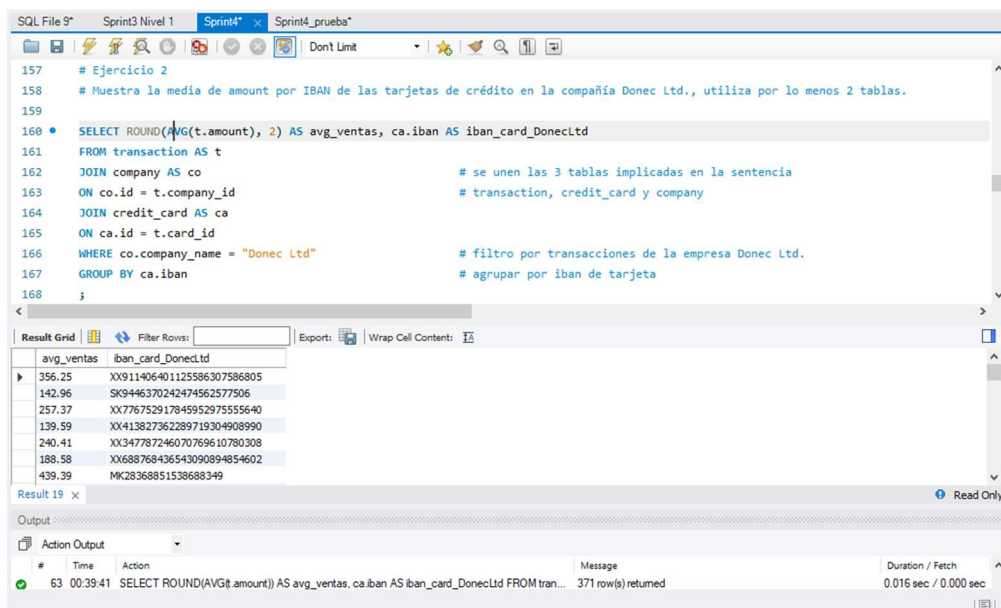
Ejercicio 2

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

Para esta consulta se requiere unir 3 tablas:

- **company** para filtrar por la empresa “Donec Ltd.”,
- **credit_card** para obtener los *iban* de las tarjetas y agrupar por este campo, y
- **transaction** para calcular las ventas promedio (*avg_ventas*) y vincular las otras tablas.

En la imagen anexa se presenta la consulta y los resultados, siendo 371 tarjetas con transacciones en la empresa Donec Ltd.



```
157
158 # Ejercicio 2
159 # Muestra la media de amount por IBAN de las tarjetas de crédito en la compañía Donec Ltd., utiliza por lo menos 2 tablas.
160
161 • SELECT ROUND(AVG(t.amount), 2) AS avg_ventas, ca.iban AS iban_card_DonecLtd
162     FROM transaction AS t
163     JOIN company AS co
164     ON co.id = t.company_id
165     JOIN credit_card AS ca
166     ON ca.id = t.card_id
167     WHERE co.company_name = "Donec Ltd"
168     GROUP BY ca.iban
169 ;
```

avg_ventas	iban_card_DonecLtd
356.25	XX911406401125586307586805
142.96	SK9446370242474562577506
257.37	XX776752917845952975555640
139.59	XX413827362289719304908990
240.41	XX347782746070769610780308
188.58	XX688768436543090894854602
439.39	MK28368851538688349

Result 19 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
63	00:39:41	SELECT ROUND(AVG(t.amount)) AS avg_ventas, ca.iban AS iban_card_DonecLtd FROM tran...	371 row(s) returned	0.016 sec / 0.000 sec

NIVEL 2

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

Ejercicio 1

¿Cuántas tarjetas están activas?

Para crear la tabla primero se definen que campos están involucrados *card_id*, *id*, *declined*, *timestamp*; todos de la tabla **transaction**. Ahora se requiere un comando que clasifique con una condición si son activas o inactivas (CASE WHEN). Pero se necesita establecer bien la condición ya que incluyen el ordenar cronológicamente cada transacción por tarjeta. Para estas búsquedas la función Ventana es adecuada (OVER).

```

170 # Nivel 2 - Nivel 2 - Nivel 2 - Nivel 2 - Nivel 2 - Nivel 2 - Nivel 2 - Nivel 2 - Nivel 2
171
172 # Crea una nueva tabla que refleje el estado de las tarjetas de crédito basado en:
173 # si las tres últimas transacciones han sido declinadas entonces es inactivo, si al menos una no es rechazada entonces es activo.
174 # Partiendo de esta tabla responde:
175
176 -- Se requieren 2 cosas para crear la nueva tabla:
177 # 1. Clasificar las tarjetas en activas e inactivas con un condicional, se usa CASE WHEN THEN
178 # 2. Definir la condición que permite separar las tarjetas, se usa la función ventana OVER, RANK, ROW_NUMBER
179
180 SELECT id, timestamp, card_id, declined, COUNT(declined)
181 OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS orden # aquí se agrupa con PARTITION BY, y ordena ORDER BY
182 FROM transaction;
  
```

id	timestamp	card_id	declined	orden
B0D988A9-5189-4ED8-9E4C-64B81D654F5B	2024-10-25 13:11:54	Cs-4857	0	1
B954A0B3-9314-4615-ACB8-831E0245D8D1	2024-10-07 16:43:17	Cs-4857	0	2
2C537AD9-D80E-402A-A79F-7D155138F47C	2024-07-27 10:50:49	Cs-4857	0	3
40A0B056-3438-40A9-8F5B-8CDE5660047D	2024-02-08 14:55:10	Cs-4857	1	4
FD857386-28F9-49DE-8F32-E2458CDE1752	2024-01-15 20:18:34	Cs-4857	0	5
8B8CF288-420E-4EAE-83C0-F222CB7F5863	2022-09-21 22:52:27	Cs-4857	0	6

Se inicia como se observa en la imagen aplicando la función ventana, esta permite usar funciones de agregación manteniendo el detalle de cada fila, lo cual es ideal para la clasificar datos. En la sentencia se usa el comando OVER para agrupar las tarjetas por transacciones declinadas o no (PARTITION BY), y ordenando por fecha (ORDER BY); así se puede identificar las ultimas transacciones y saber si fueron rechazadas o no. Esto ayudara a definir el condicional para crear la tabla.

```

187 # función ventana en select (para confirmar el orden de uso de tarjeta) y en condición del CASE para definir que separar
188 SELECT card_id, declined, COUNT(declined)
189 OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS orden,
190 CASE
191   WHEN declined = 1 AND COUNT(declined)
192   OVER (partition by card_id order by timestamp DESC) IN (1, 2, 3) THEN "inactiva" # condiciones para inactivo
193   ELSE "activa" # condiciones para activo
194 END AS status
195 FROM transaction;
  
```

card_id	declined	orden	status
Cs-4898	0	49	activa
Cs-4898	0	50	activa
Cs-4898	0	51	activa
Cs-4898	0	52	activa
Cs-4898	0	53	activa
Cs-4898	0	54	activa
Cs-4899	1	1	inactiva
Cs-4899	1	2	inactiva
Cs-4899	1	3	inactiva
Cs-4899	0	4	activa
Cs-4899	0	5	activa
Cs-4899	0	6	activa

Ahora se procede como se ve en la imagen anterior a armar la tabla, introduciendo el comando CASE para definir las condiciones de tarjeta activa y tarjeta inactiva. Aquí se indica la función ventana que detalla la condición que se necesita cumplir.

A continuación, como se aprecia en la imagen siguiente se intenta agrupar por tarjetas, esto genera el error code 1055, asociado a incompatibilidad con el modo sql “only_full_group_by”, para solucionar se debe limitar los elementos del select o desactivar el modo. Se intenta limitar los campos del select, obteniendo el mismo resultado error code 1055.

```

185 SELECT card_id, declined, COUNT(declined)
186 OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS orden,
187 CASE
188   WHEN declined = 1 AND COUNT(declined)
189     OVER (partition by card_id order by timestamp DESC) IN (1, 2, 3) THEN "inactiva" # condiciones para inactivo
190   ELSE "activa" # condiciones para activo
191 END AS status
192 FROM transaction
193 GROUP BY card_id;
194
195 # al usar GROUP BY genera error 1055, indica no compatible con sql_mode = only_full_group_by
196 # se soluciona limitando el select o desactivando este modo
197 # incluso abajo limitando el select a solo el campo que agrupa y la funcion agregada
198
199 SELECT card_id, COUNT(declined)
200 OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS orden,
201 CASE
202   WHEN declined = 1 AND COUNT(declined)
203     OVER (partition by card_id order by timestamp DESC) IN (1, 2, 3) THEN "inactiva" # condiciones para inactivo
204   ELSE "activa" # condiciones para activo
205 END AS status
206 FROM transaction
207 GROUP BY card_id;
  
```

Output:

#	Time	Action	Message	Duration / Fetch
23	11:27:27	SELECT card_id, declined, COUNT(declined) OVER (PARTITION BY card...	Error Code: 1055. Expression #2 of SELECT list is not in GROUP BY clause a...	0.000 sec
24	11:28:54	SELECT card_id, COUNT(declined) OVER (PARTITION BY card_id ORDE...	Error Code: 1055. Expression #2 of SELECT list is not in GROUP BY clause a...	0.000 sec

Por tal motivo se decide desactivar temporalmente el modo “only_full_group_by” para generar la tabla **card_status**. Se identifico en documentación los comandos para activar y desactivar el modo. Al desactivarlo se pudo ejecutar la sentencia para crear la tabla, adicionalmente se limito a presenta la tarjeta y el status, ya que el uso del COUNT (*declined*) en función ventana era solo para poder realizar la condición de clasificación en el CASE.

```

207 # Se procede a desactivar el modo ONLY_FULL_GROUP_BY para aplicar la consulta que genera la tabla
208 SET sql_mode=(SELECT REPLACE(@@sql_mode,'ONLY_FULL_GROUP_BY','')); # DESACTIVAR
209
210 CREATE TABLE card_status AS # se crea una tabla que permita revisar a futuro la consulta
211 SELECT card_id, #COUNT(declined) OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS orden,
212 CASE
213   WHEN declined = 1 AND COUNT(declined)
214     OVER (partition by card_id order by timestamp DESC) IN (1, 2, 3) THEN "inactiva" # condiciones para inactivo
215   ELSE "activa" # condiciones para activo
216 END AS status
217 FROM transaction
218 GROUP BY card_id;
  
```

Result Grid:

card_id	status
CS-4893	activa
CS-4894	activa
CS-4895	inactiva
CS-4896	inactiva
CS-4897	activa

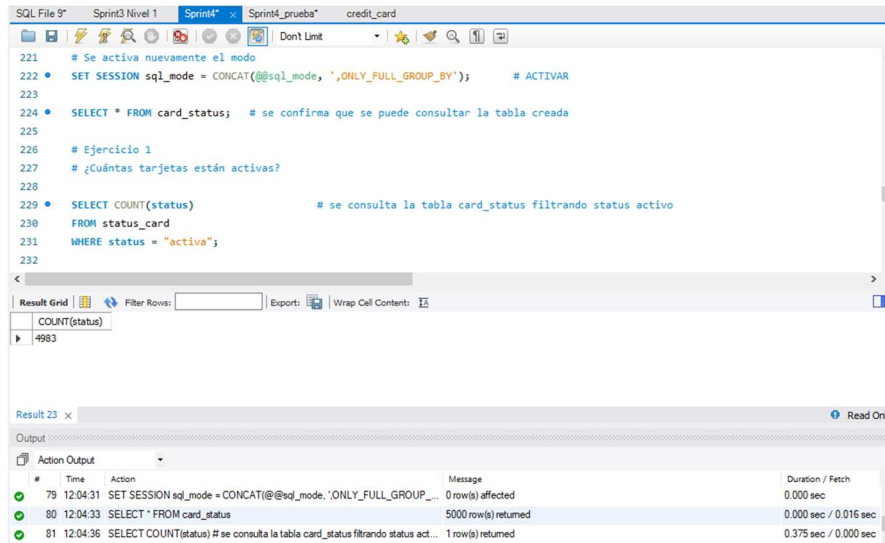
card_status 16 x

Output:

#	Time	Action	Message	Duration / Fetch
68	11:57:02	SET sql_mode=(SELECT REPLACE(@@sql_mode,'ONLY_FULL_GROUP_B...	0 row(s) affected	0.000 sec
69	11:57:08	CREATE TABLE card_status AS# se crea una tabla que permita revisar a fut...	5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0	0.516 sec
70	11:57:21	SELECT * FROM card_status	5000 row(s) returned	0.000 sec / 0.016 sec

En la imagen siguiente se presenta la activación de modo sql “only_full_group_by” y la confirmación de consultar la nueva tabla. Se intento crear una vista (VIEW) en lugar de una tabla, pero al activar nuevamente el modo sql, generaba error al realizar cualquier consulta.

Finalmente, en la imagen se muestra el resultado de la consulta solicitada en el ejercicio, acerca del numero de tarjetas activas.

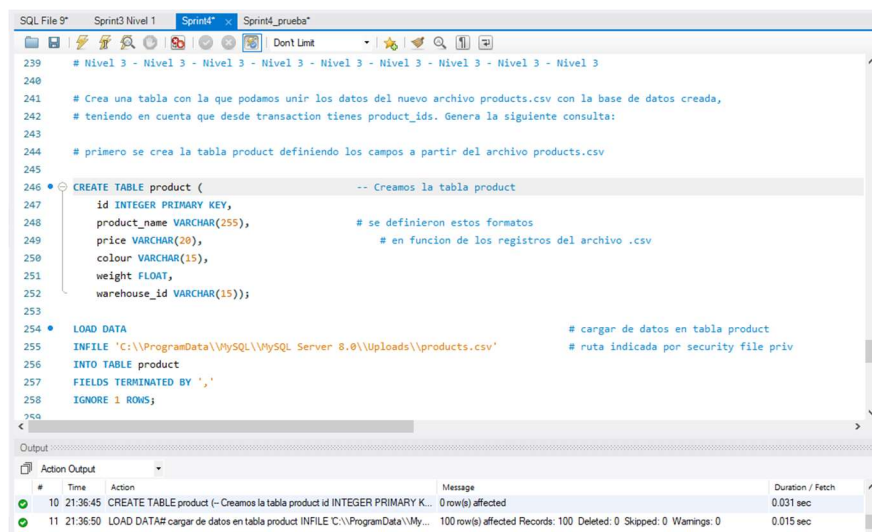


NIVEL 3

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:

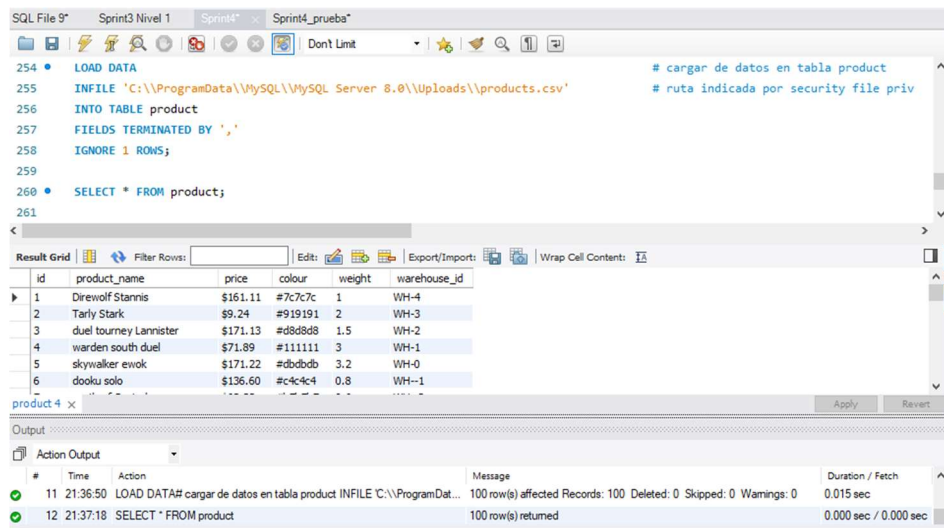
Primero se crea la estructura de la tabla **product** basados en el archivo *products.csv* que contiene los datos, donde se identifica:

- *id*: número identificador del código del producto, asignado como Primary Key,
- *product_name*: nombre del producto,
- *price*: precio del producto, en dólar (\$),
- *colour*: color de producto, en formato hexadecimal,
- *weight*: peso del producto,
- *warehouse_id*: se supone el almacén donde se localiza el producto,

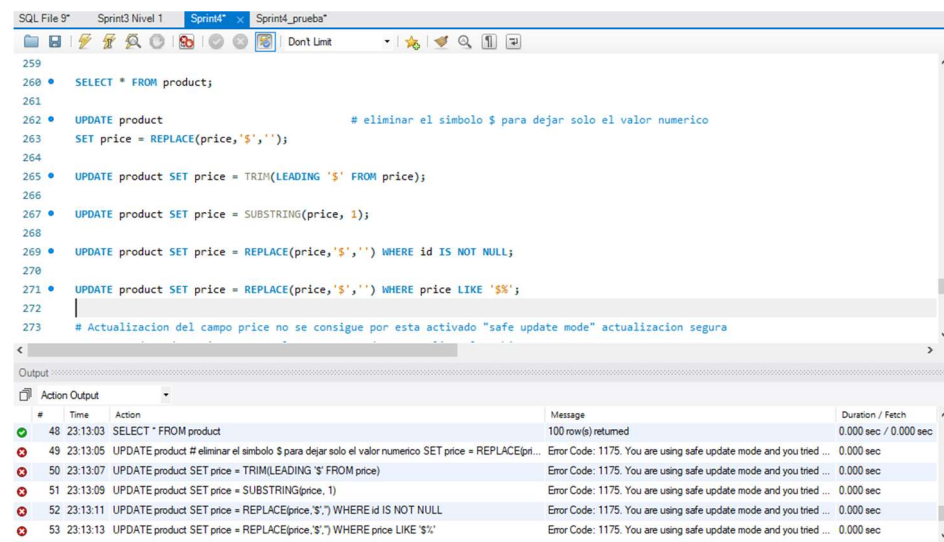


Los formatos en los campos se determinan del tipo de registros vistos en el archivo .csv y la longitud aproximada de los mismos. En la imagen se aprecia las sentencias ejecutadas para crear y cargar datos en la tabla **product**.

A continuación, se anexa como quedan los datos cargados en la tabla, se identifica que para la columna precio los registros tiene delante el símbolo dólar (\$), considerando que es un campo de métricas con el que realizaran cálculos a futuro, se debe retirar el símbolo para dejar solo el valor numérico.

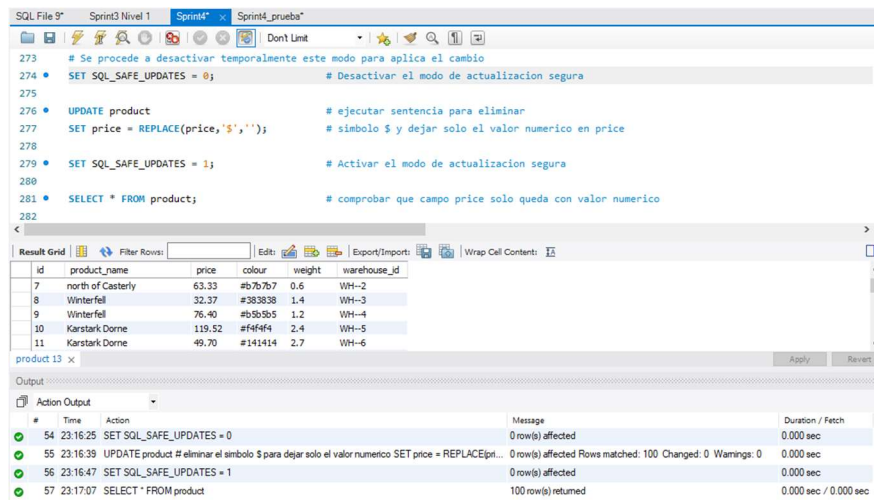


Para retirar el símbolo se utiliza el comando UPDATE y REPLACE para sustituir el símbolo \$ por un espacio vacío. En la imagen siguiente se muestra el error 1175, generado al intentar modificar la columna, indica que el modo de actualización segura (safe update mode) está activo. Este modo evita la eliminación o actualización accidental de datos. Para solucionar se puede modificar la consulta para incluir una clave o desactivar temporalmente el modo seguro.



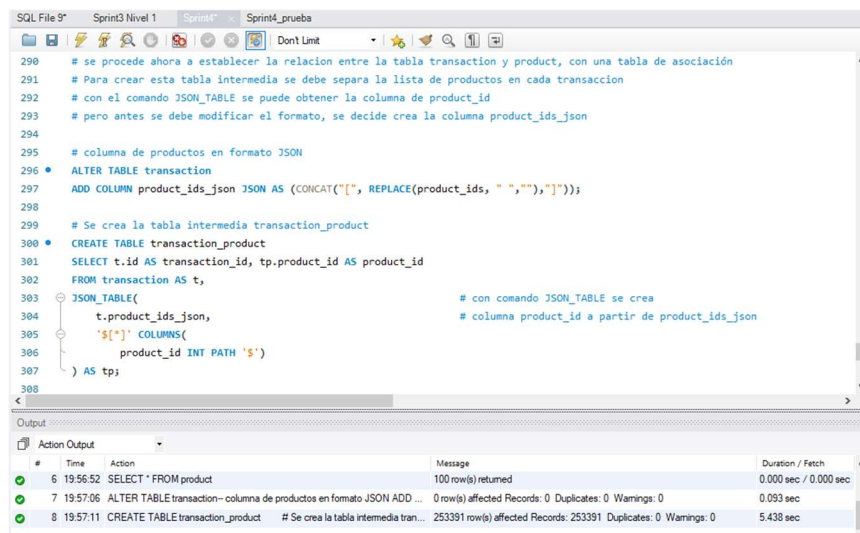
Con el fin de evitar alterar la seguridad se usaron diferentes comandos para intentar el cambio en la columna *price* (UPDATE con WHERE, TRIM y SUBSTRING), obteniendo el mismo error 1175.

Se decide desactivar el modo seguro aplicar el cambio y luego activarlo nuevamente. Como se observa en la imagen siguiente se puede modificar la columna *price* en la tabla **product**.



Ahora se debe realizar la relación entre la tabla de hecho **transaction** y la tabla de dimensión recién generada **product**. En este caso la relación es de N:M, de muchos a muchos, un conjunto de productos puede estar involucrados en diferentes transacciones. Para definir el vínculo se debe crear una tabla intermedia o de asociación, donde estén presentes las 2 Foreign Key representando a las Primary Key de ambas tablas.

Para la tabla **transaction** se tiene el campo *product_ids*, donde se listan los productos vendidos en cada transacción. Para crear la tabla intermedia es necesario separar estas listas de productos. Para esto se usará el comando **JSON_TABLE** que permite separa los elementos, pero necesita como entrada un archivo json, para esto se creará una columna adicional.



En la imagen anterior se aprecia la creación de la columna *product_ids_json* en formato json (línea 296), a partir de *product_ids* e insertando los corchetes ([]) propios de los archivos json.

Luego se crea la tabla intermedia de asociación **transaction_product** con los campos **transaction.id** y la nueva columna *product_id*, generada dentro de la sentencia a partir de **JSON_TABLE** usando la columna creada previamente en formato json.

Ya con la tabla de unión **transaction_product** creada, se generan los vínculos de esta con las tablas **transaction** y **product**. Se define las restricciones (CONSTRAIN) y Foreign key, junto con las referencias a cada Primary key. La sentencia se aplica para cada tabla por separado. De esta manera se asocia:

- **transaction.id = transaction_product.transaction_id** y
- **product.id = transaction_product.product_id**.

En la imagen siguiente se muestra las sentencias para generar las relaciones y la tabla de asociación creada.

```

311 # Relacion entre la tabla transaction_product a traves de restricción y Foreign key con tabla producto y transaction
312
313 • ALTER TABLE transaction_product
314   ADD CONSTRAINT product_fkey FOREIGN KEY (product_id) REFERENCES product (id); -- referencia a traves de product.id
315
316 • ALTER TABLE transaction_product
317   ADD CONSTRAINT transaction_fkey FOREIGN KEY (transaction_id) REFERENCES transaction (id); -- referencia a traves de product.id
318
319 • SELECT * FROM transaction_product;
320

```

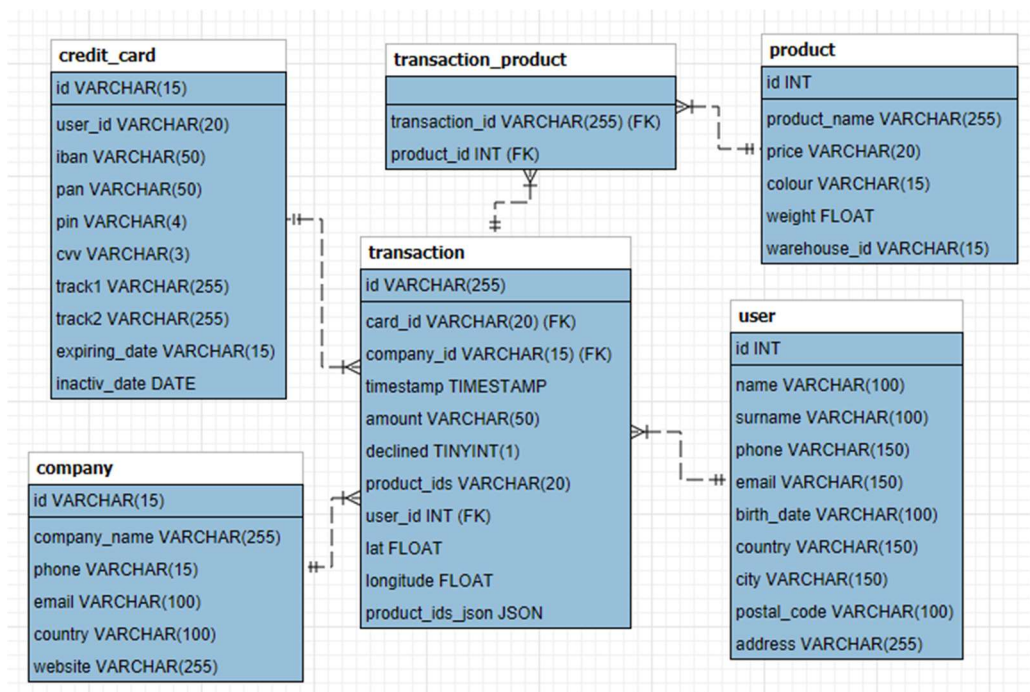
transaction_id	product_id
00043A49-2949-494B-A5DD-A5BAE388190D	16
00043A49-2949-494B-A5DD-A5BAE388190D	26
00043A49-2949-494B-A5DD-A5BAE388190D	97
00043A49-2949-494B-A5DD-A5BAE388190D	87
000447FE-8650-4DCF-8SDE-C7ED0EE1CAAD	66
000447FE-8650-4DCF-8SDE-C7ED0EE1CAAD	69
000447FE-8650-4DCF-8SDE-C7ED0EE1CAAD	87
00045D6B-ED2E-4F2F-8186-CEE074D875D0	30

transaction_product 6 x

Output

#	Time	Action	Message	Duration / Fetch
13	20:13:56	ALTER TABLE transaction_product ADD CONSTRAINT product_fkey FOREI...	253391 row(s) affected Records: 253391 Duplicates: 0 Warnings: 0	2.968 sec
14	20:14:04	ALTER TABLE transaction_product ADD CONSTRAINT transaction_fkey FO...	253391 row(s) affected Records: 253391 Duplicates: 0 Warnings: 0	5.266 sec
15	20:15:03	SELECT * FROM transaction_product	253391 row(s) returned	0.000 sec / 0.391 sec

Como resultado final se presenta el diagrama de la base de datos **marketplace**.



Ejercicio 1

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

Para saber cuántas veces se han vendido los productos se realiza la consulta con un COUNT de productos sobre las transacciones realizadas y agrupando por productos, se adiciono el nombre del producto con un JOIN a la tabla **product**. En la imagen se presenta la consulta con el resultado para cada producto.

The screenshot shows a SQL IDE window with a query editor and a results grid. The query is as follows:

```
321 # Ejercicio 1
322 # Necesitamos conocer el número de veces que se ha vendido cada producto.
323
324 # Ahora con una columna de product_id separada y relacionada con transaction_id se puede calcular la cantidad de ventas
325
326 • SELECT tp.product_id, p.product_name, COUNT(tp.product_id) AS cantidades_vendidas
327 FROM transaction_product AS tp
328 JOIN product AS p
329 ON p.id = tp.product_id
330 GROUP BY tp.product_id;
```

The results grid shows the following data:

product_id	product_name	cantidades_vendidas
1	Direwolf Stannis	2468
2	Tarly Stark	2573
3	duel tourney Lannister	2528
4	warden south duel	2584
5	skywalker evok	2548
6	dooku solo	2492
7	north of Casterly	2563

The output pane shows the execution of the query, indicating that 100 rows were returned.