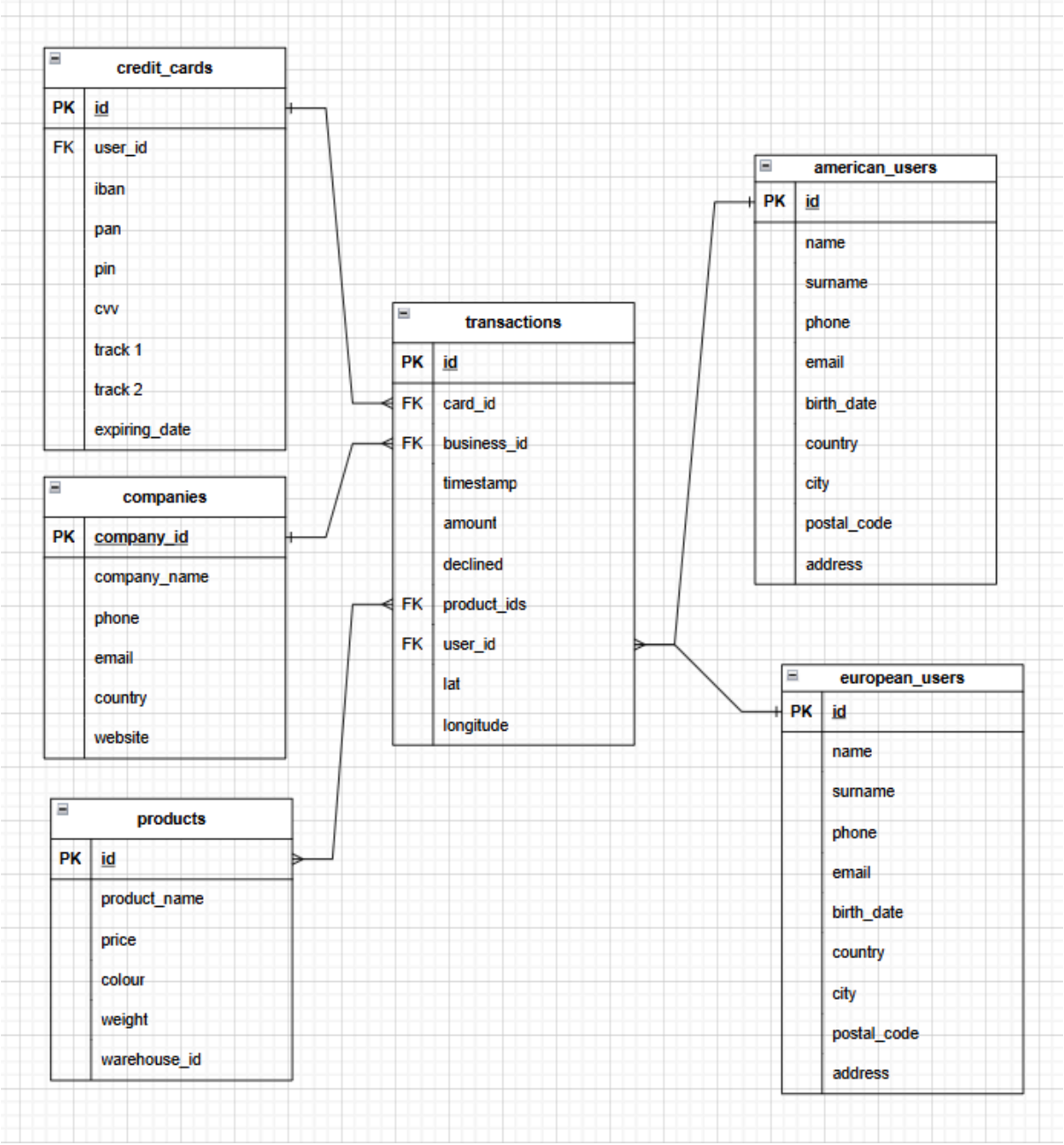
Nivel 1

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



La base de datos está compuesta por seis tablas que almacenan información sobre las transacciones realizadas por clientes de América y Europa, incluyendo las empresas, los productos comprados y las tarjetas de crédito utilizadas.

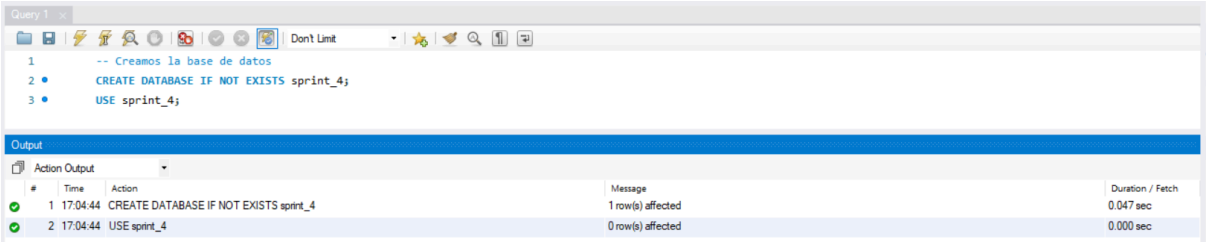
La información de los usuarios estaba inicialmente dividida en dos tablas según la región: american_users y european_users. Ambas compartían la misma estructura y contenían

datos personales (nombre, apellido, fecha de nacimiento) y datos de contacto (teléfono, email, país, ciudad, código postal y dirección). Para simplificar el esquema y facilitar las consultas generales, se unificaron en una sola tabla users, añadiendo una columna region con los valores “american” o “european”.

La tabla de dimensión products incluye: id, nombre, precio, color, peso e id del almacén.
La tabla de dimensión companies contiene: id, nombre, teléfono, email, país y página web.
La tabla de dimensión credit_cards contiene: id, id de usuario, IBAN, PAN, PIN, CVV, track1, track2 y fecha de caducidad.

La tabla de hechos transactions conecta todas las dimensiones y almacena la información de cada transacción: id, timestamp, amount, indicador de si fue declinada, coordenadas (latitud y longitud), y claves foráneas: card_id, business_id, product_ids y user_id.

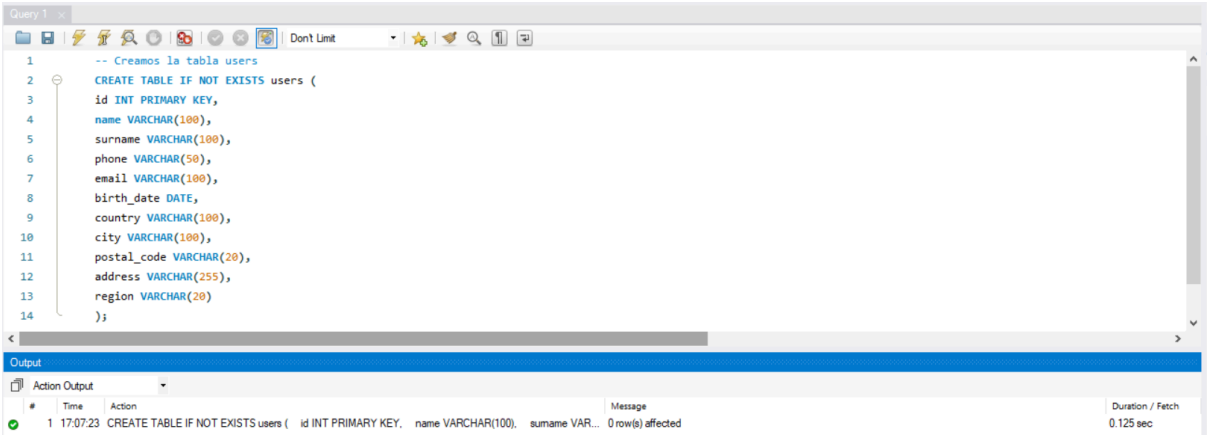
Primero, se crea la **base de datos**:



A continuación, se crean las tablas correspondientes. Las tablas american_users y european_users comparten la misma estructura y sus IDs no se repiten, lo que sugiere que originalmente pertenecían a una sola tabla. Mantenerlas separadas complicaría las consultas generales sobre transactions, ya que habría que unir información de ambas tablas según la región.

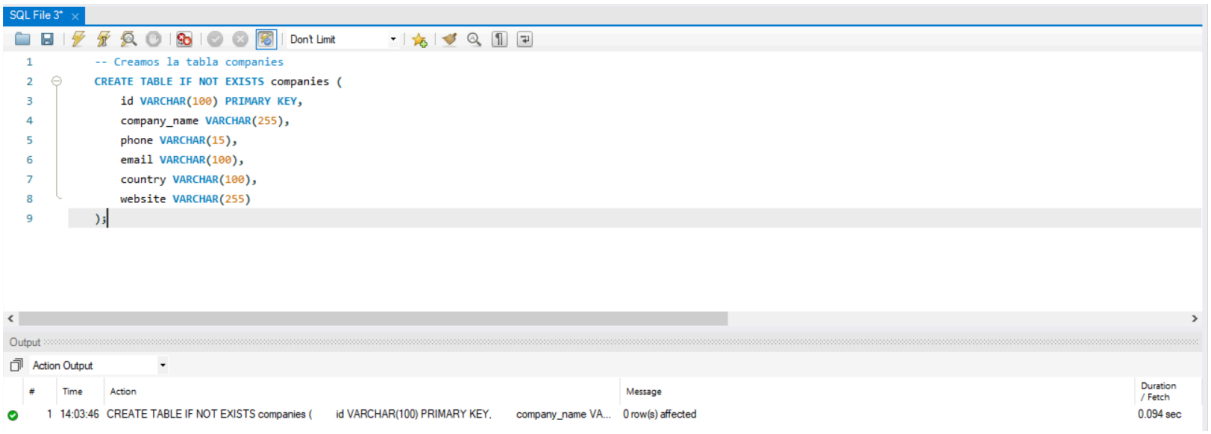
Por coherencia del modelo y para simplificar el esquema estrella, decidí unificar estas tablas en una sola tabla users e incorporar la columna region, con valores “european” y “american”.

Se procede a crear la tabla **users** con esta nueva columna para mantener la información de origen de cada usuario:

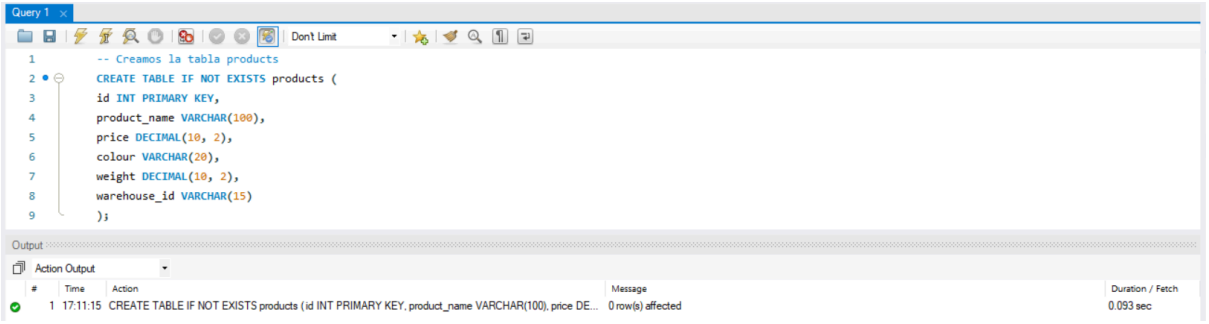


Seguimos con las demás tablas.

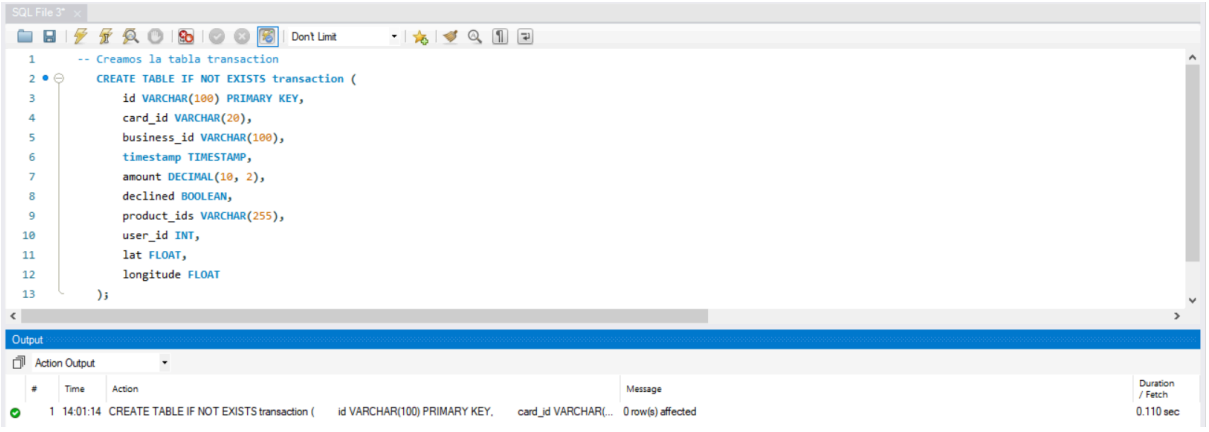
companies:



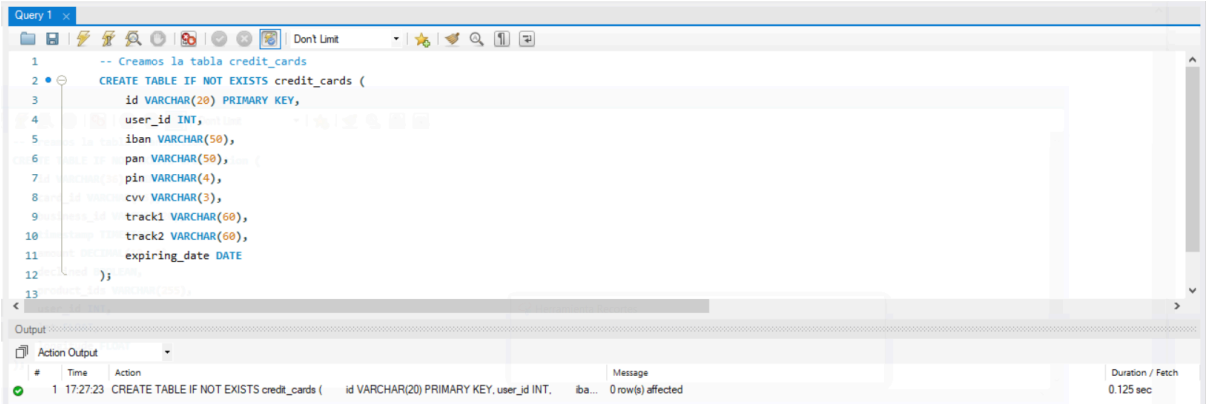
products:



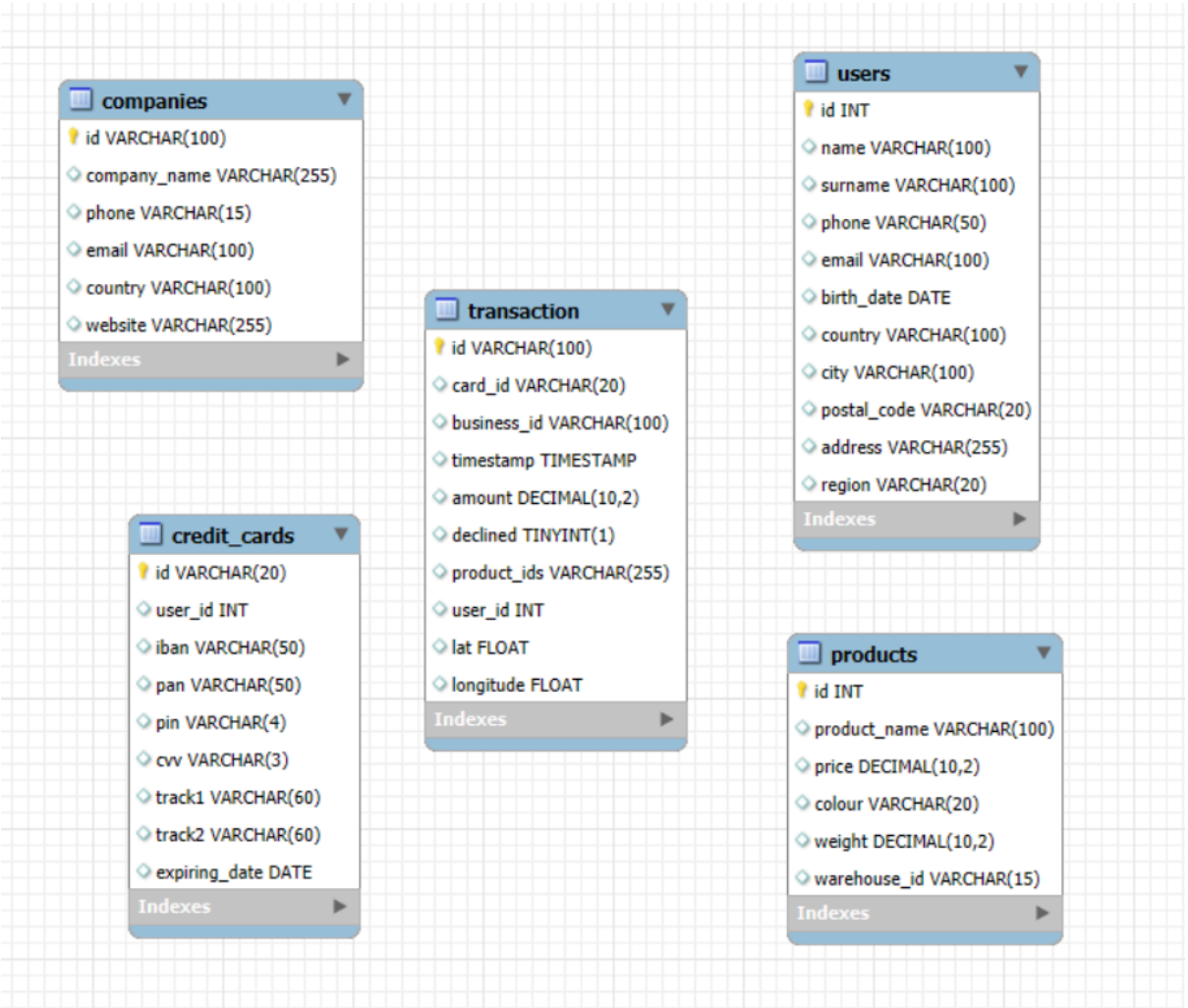
transactions:



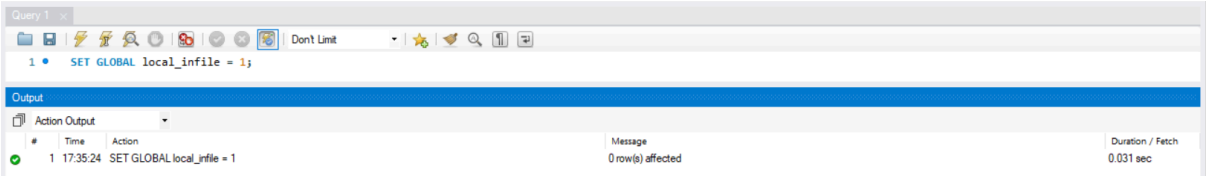
credit_cards:



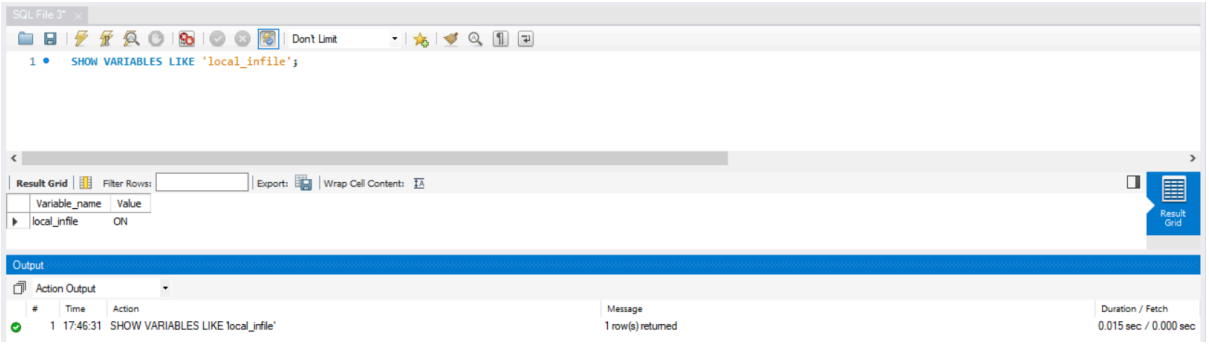
Tras crear todas las tablas revisamos el esquema general de la base de datos. De momento no hemos definido las relaciones mediante claves foráneas, ya que primero importaremos los datos desde los archivos CSV para evitar errores de integridad referencial.



Para poder cargar los datos de los archivos CSV hace falta activar los permisos:

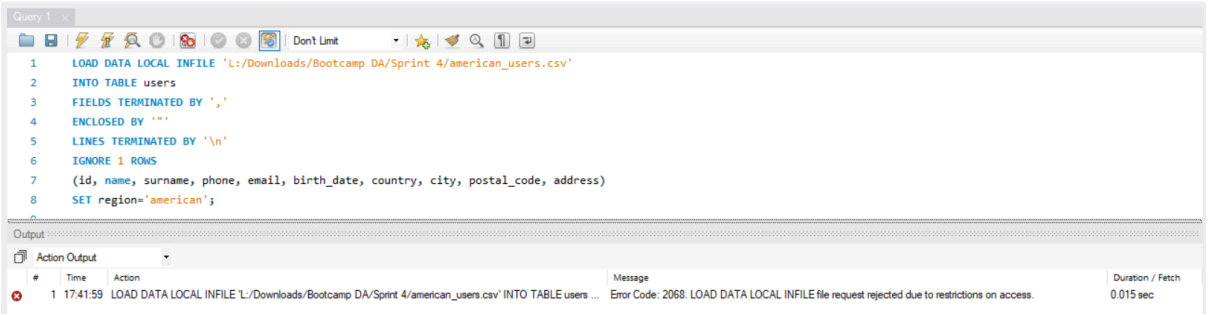


Lo comprobamos:

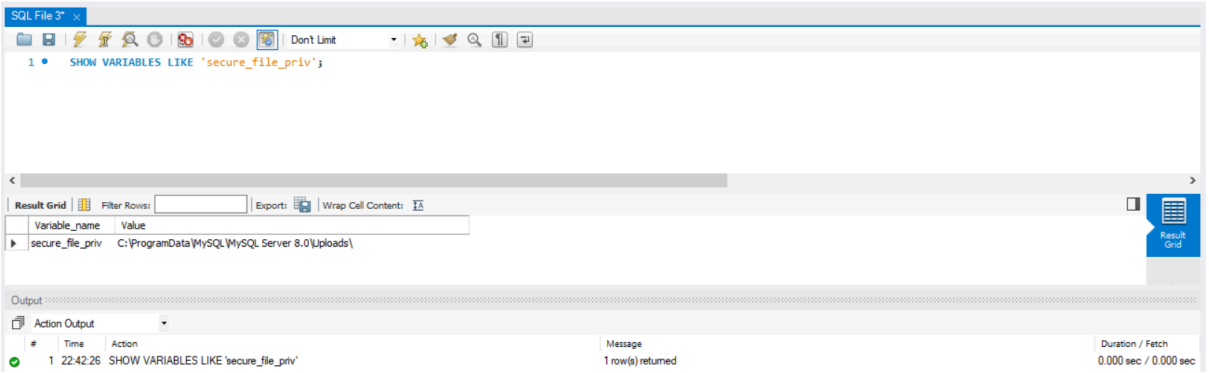


Empezamos cargando los datos de los dos archivos a la tabla users.
american:

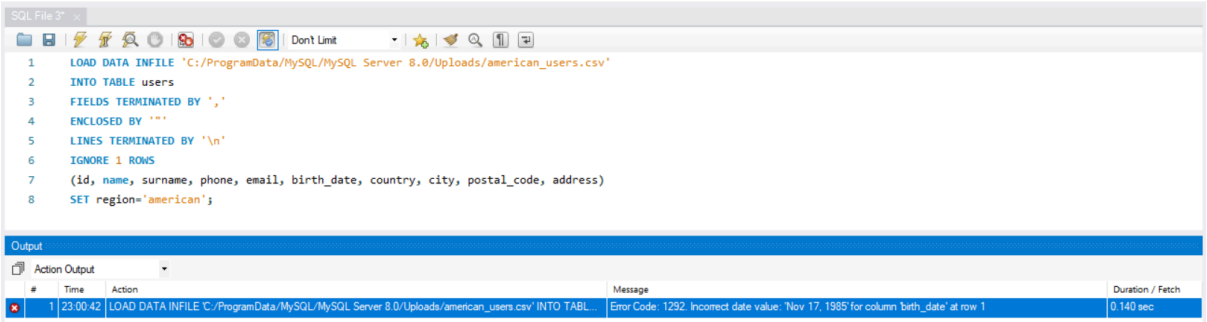
Apareció el error 2068: Aunque local_infile = ON, MySQL Workbench bloqueó el uso de archivo local.



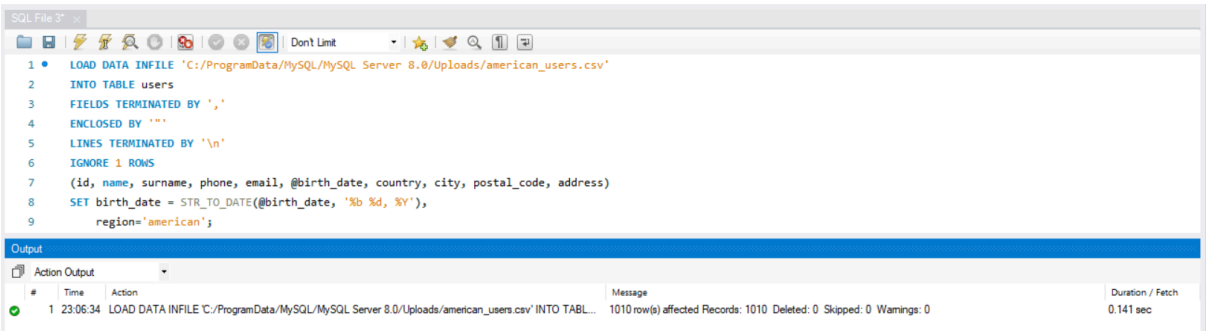
Revisé la variable secure_file_priv para conocer la carpeta permitida por MySQL:



Moví todos los CSV a esa carpeta (C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/). Reemplacé LOAD DATA LOCAL INFILE por LOAD DATA INFILE apuntando a la carpeta segura.



Formato original de la fecha: Nov 17, 1985 mientras que el tipo de columna en la tabla es DATE (MySQL requiere YYYY-MM-DD).



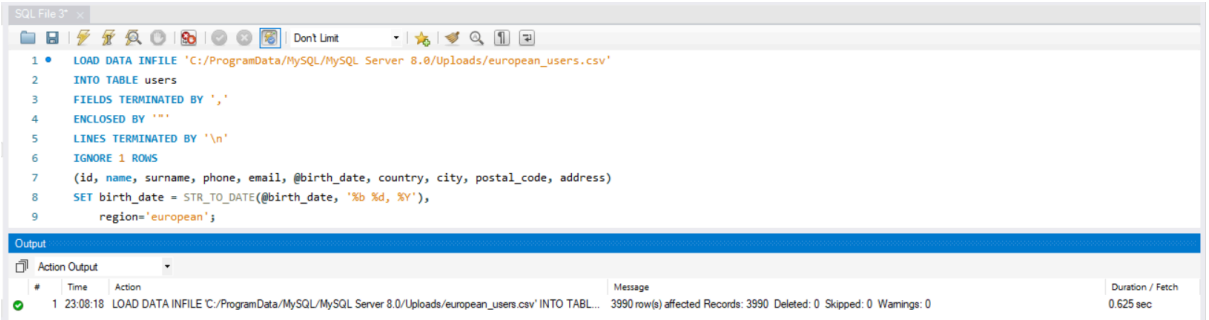
STR_TO_DATE() convierte automáticamente el texto al formato correcto durante la importación.
@birth_date es una variable temporal que almacena la cadena del CSV.

('%b %d, %Y') indica a MySQL cómo interpretar ese texto:

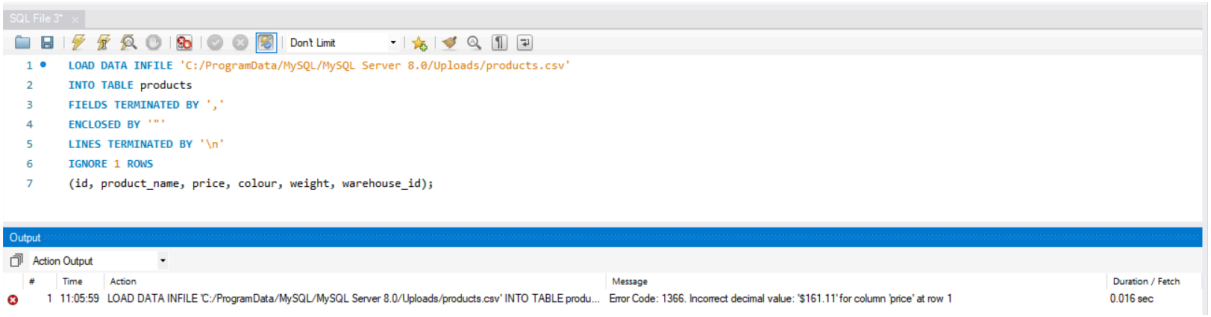
- %b → nombre abreviado del mes (Jan, Feb, Mar…)
- %d → día con dos dígitos
- %Y → año con cuatro dígitos

Esto permite cargar los datos sin modificar el archivo original.

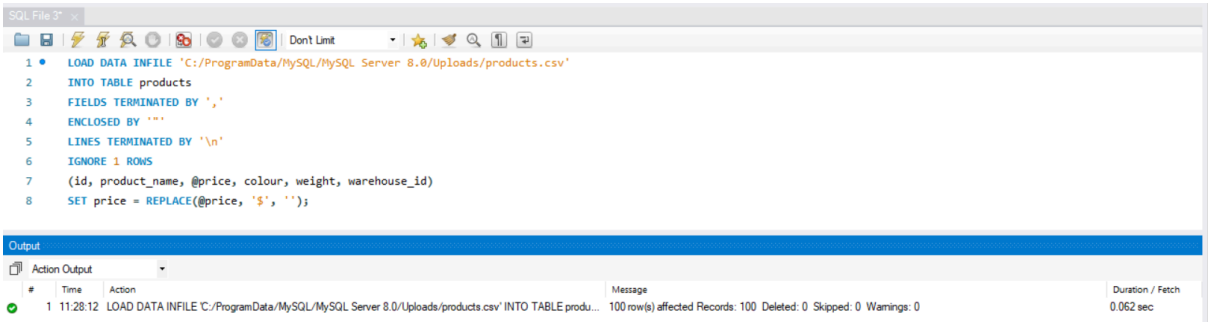
european:



products:



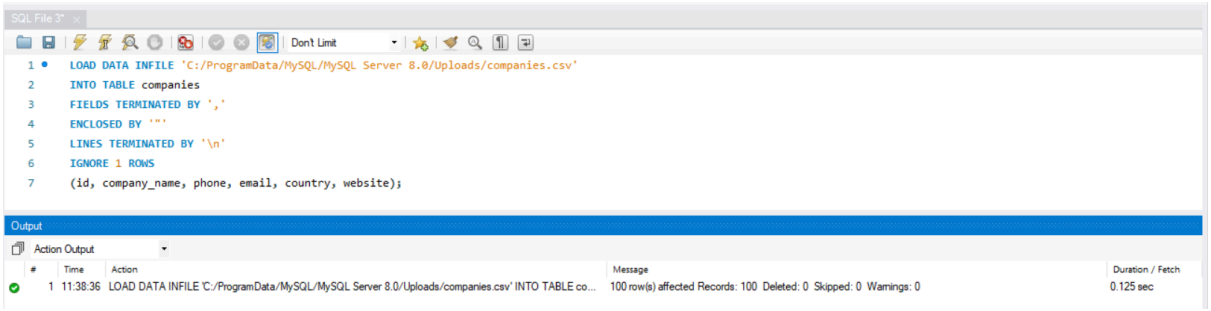
El error 1366 se debe a que MySQL no puede convertir directamente "$161.11" a un DECIMAL, es el tipo más apropiado porque permite realizar las operaciones aritméticas. Todos los productos están en la misma moneda, por eso podemos ignorar el símbolo y quedarnos solo con el número, no hay necesidad de almacenar la moneda como columna separada.



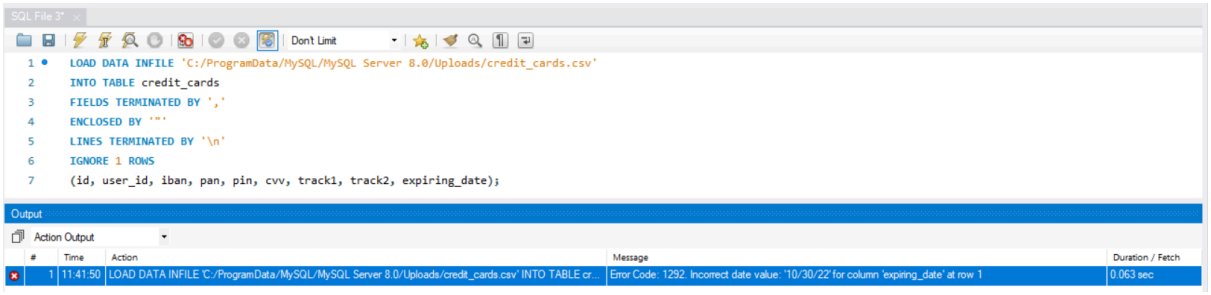
La solución es limpiar durante la importación con SET y REPLACE():

- Usamos una variable temporal (@price) que captura el valor del CSV y luego eliminamos el signo $ antes de insertarlo.
- REPLACE(@price, '$', '') elimina el $ antes de insertarlo en la columna price.

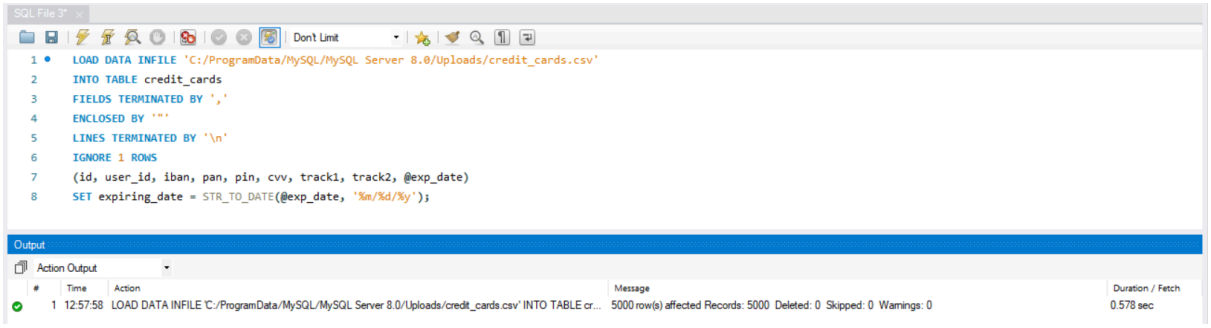
companies:



credit_cards:



El campo expiring_date de la tabla credit_cards se definió como DATE, ya que conceptualmente todas las fechas deberían almacenarse en este tipo de datos. Sin embargo, los datos del CSV vienen en formato MM/DD/YY, que no coincide con el formato que MySQL espera (YYYY-MM-DD). Por esta razón, al intentar importar directamente se produjo un error de tipo 1292 (Incorrect date value).



Usamos SET para convertir la fecha del CSV al formato que MySQL entiende (YYYY-MM-DD).

@exp_date:

- Se usa una variable temporal para capturar el valor tal cual viene del CSV.
- Esto evita que MySQL intente insertarlo directamente en la columna DATE y falle.

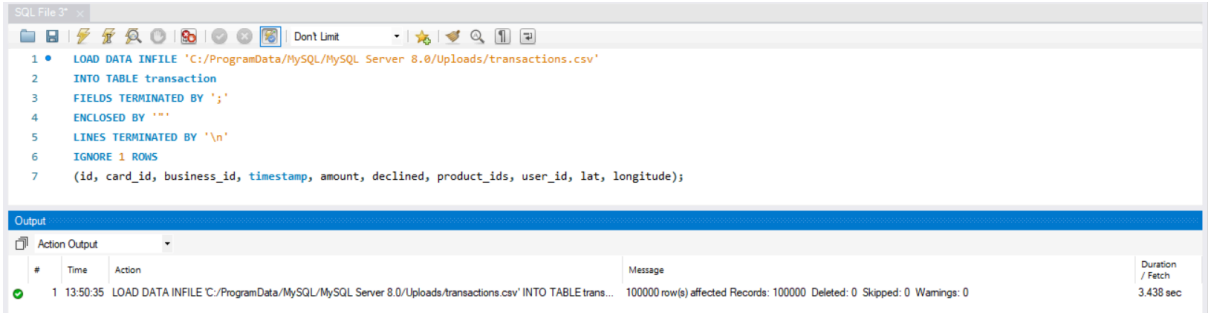
STR_TO_DATE(@exp_date, '%m/%d/%y'):

- Convierte la cadena '10/30/22' a un valor DATE '2022-10-30'.
- %m → mes, %d → día, %y → año de 2 dígitos.

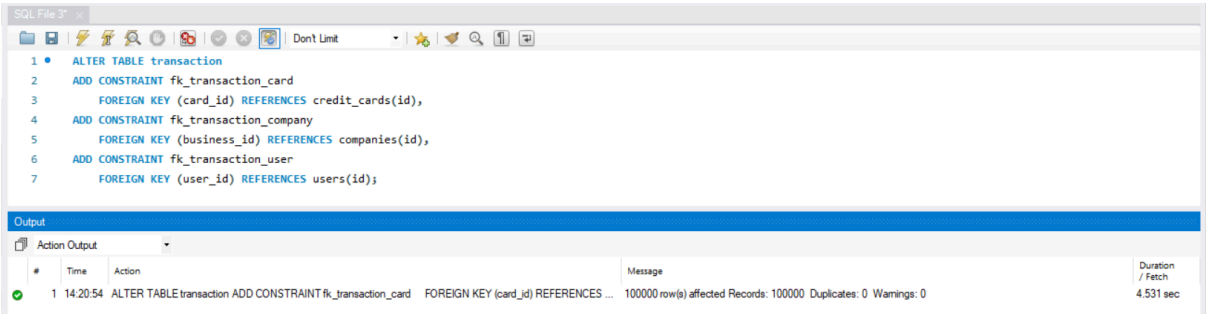
SET expiring_date = ...:

- Inserta el valor convertido en la columna expiring_date de tipo DATE.

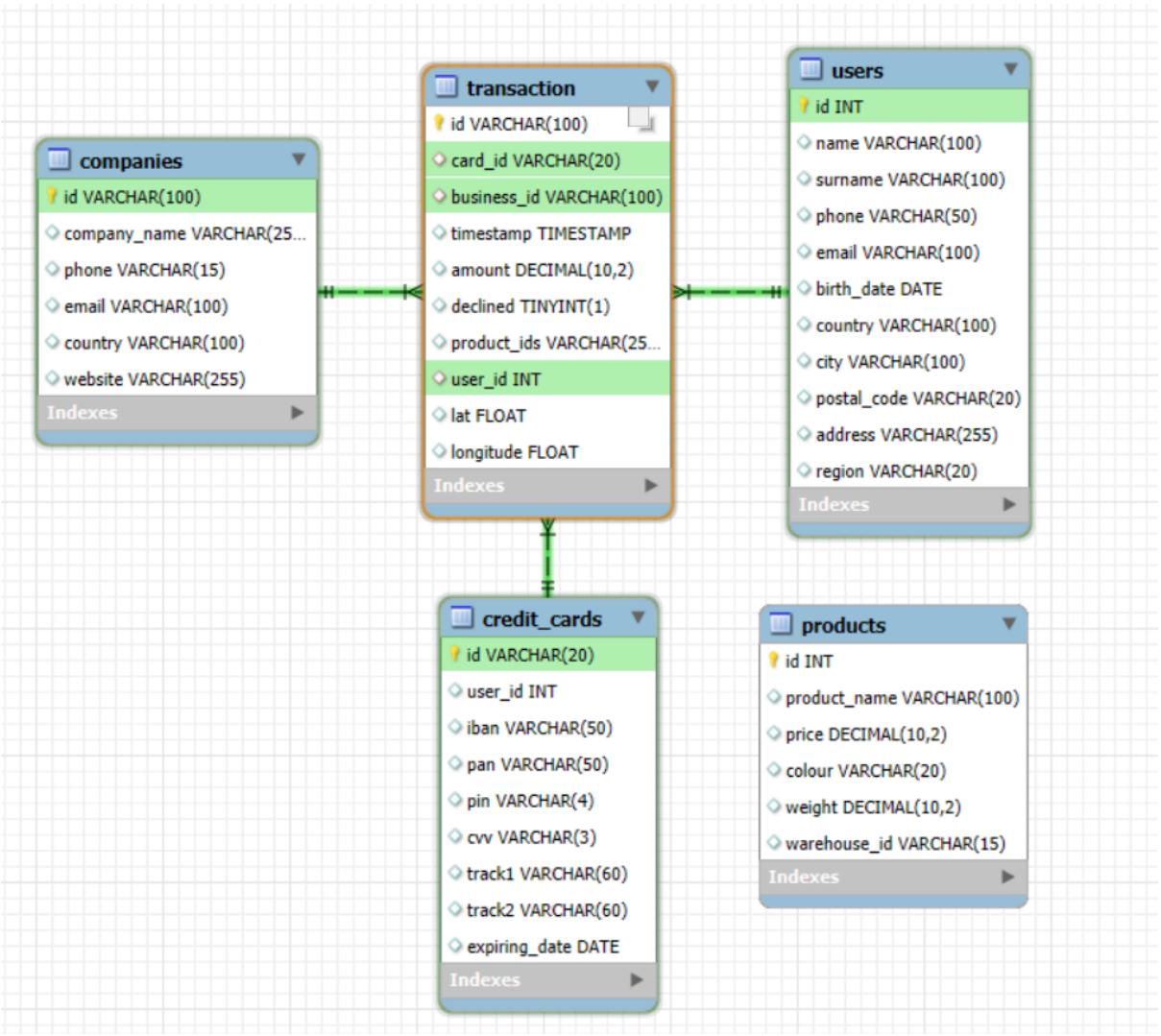
transactions:



A diferencia de las otras tablas la tabla transactions tiene como delimitador “;” y no “,”.
Añadimos las FK a la tabla transaction para conectarla con las demás tablas:

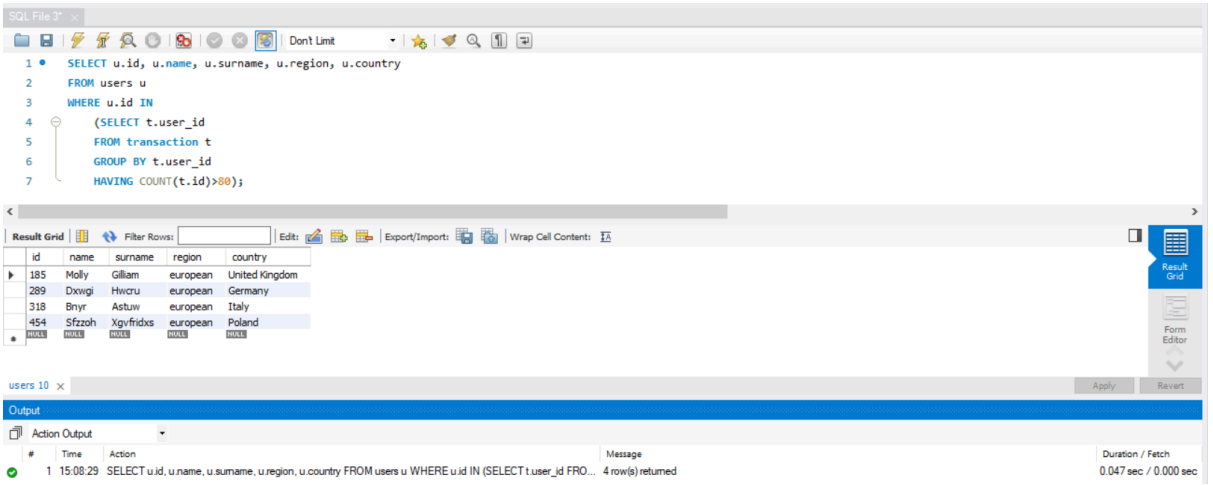


El esquema actualizado:



Ejercicio 1

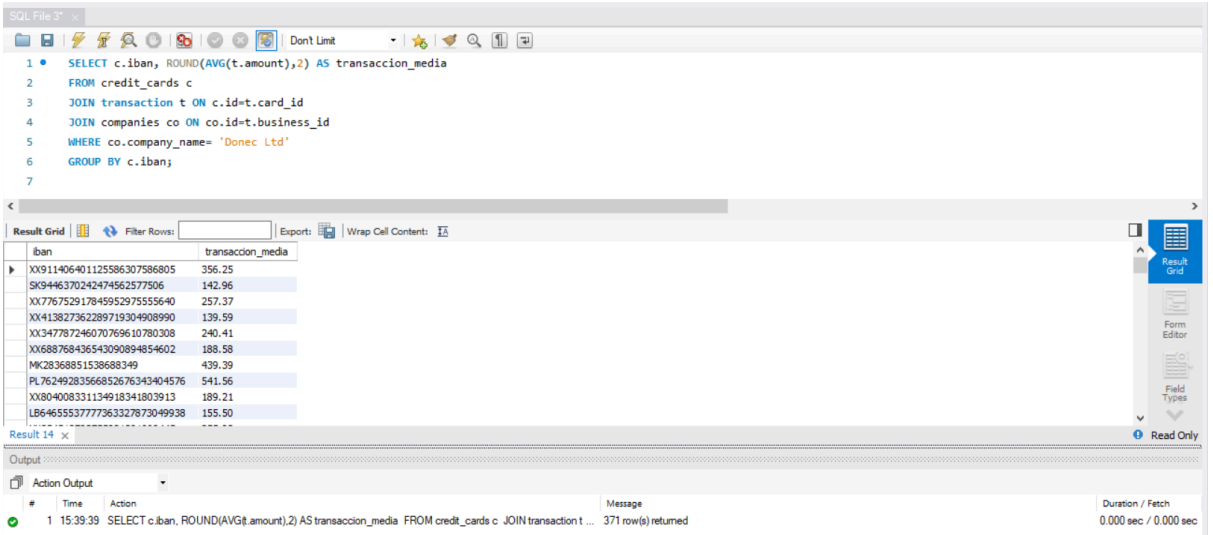
Realiza una subconsulta que muestre a todos los usuarios con más de 80 transacciones utilizando al menos 2 tablas.



Para realizar esta consulta necesitamos la tabla users para obtener la información relevante como, por ejemplo, el nombre, apellido, región y país. El enunciado nos pide sólo los usuarios que tienen más de 80 transacciones, por tanto en la consulta interna hacemos recuento de las transacciones y filtramos aquellos usuarios que cumplen con la condición.

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 este ejercicio usamos tres tablas: companies para filtrar la empresa Donec Ltd, credit_cards para obtener el IBAN de cada tarjeta, y transaction para calcular la media de los importes.
Unimos las tablas con JOIN, filtramos por la compañía y agrupamos por IBAN para obtener la media del amount.

Nivel 2

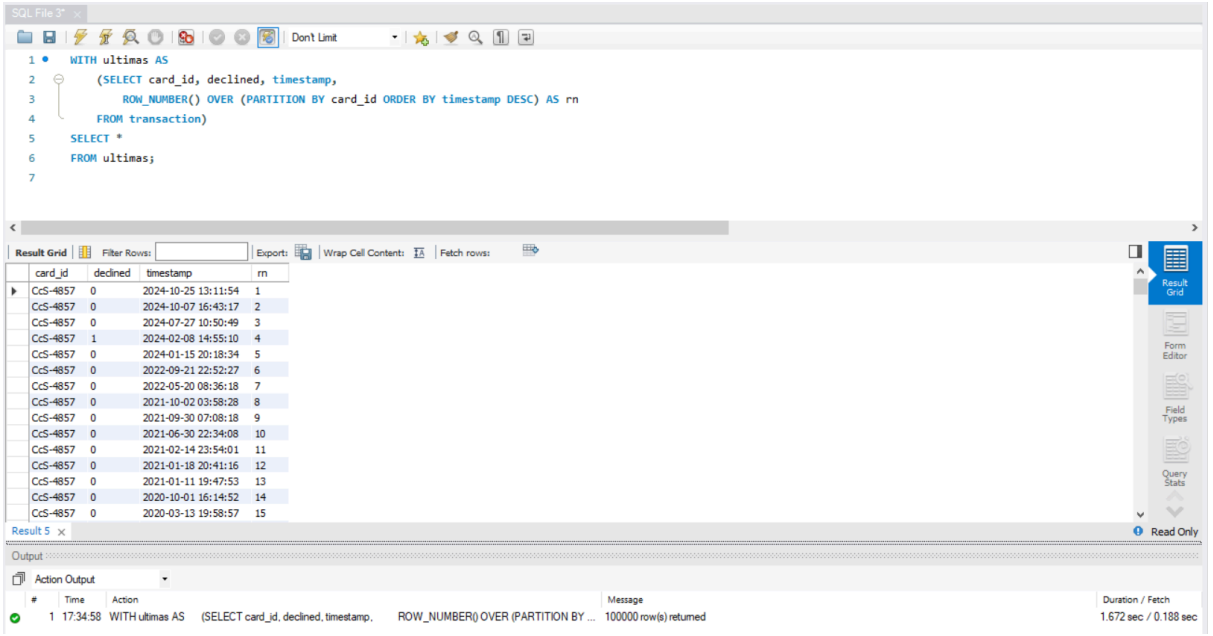
Crea una nueva tabla que refleje el estado de las tarjetas de crédito basado en *si las tres últimas transacciones han sido declinadas entonces es inactivo, si al menos una no es rechazada entonces es activo* . Partiendo de esta tabla responde:

Ejercicio 1

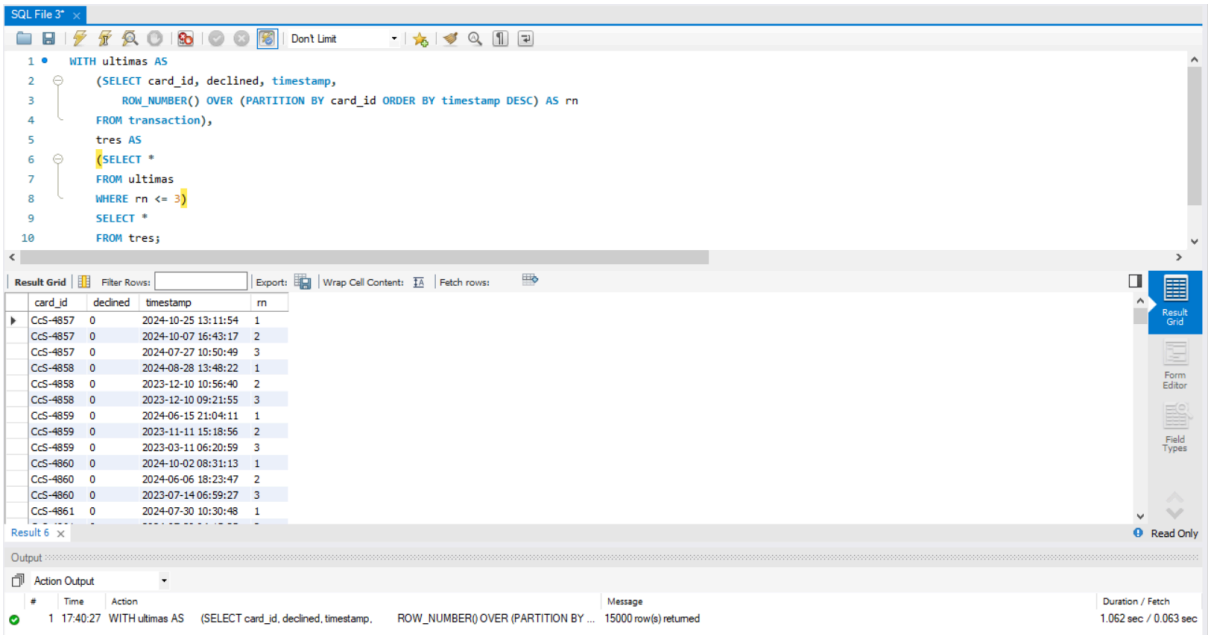
¿Cuántas tarjetas están activas?

Dividimos el problema en varios pasos.

1. Identificación de las transacciones más recientes
Primero creamos la CTE ultimas, donde organizamos todas las transacciones por tarjeta, ordenándolas de la más reciente a la más antigua.
Para ello utilizamos la función de ventana ROW_NUMBER(), particionada por card_id y ordenada por timestamp DESC.
Esto nos permite enumerar cada transacción dentro de su tarjeta y saber cuál es la 1ª, 2ª, 3ª, etc.

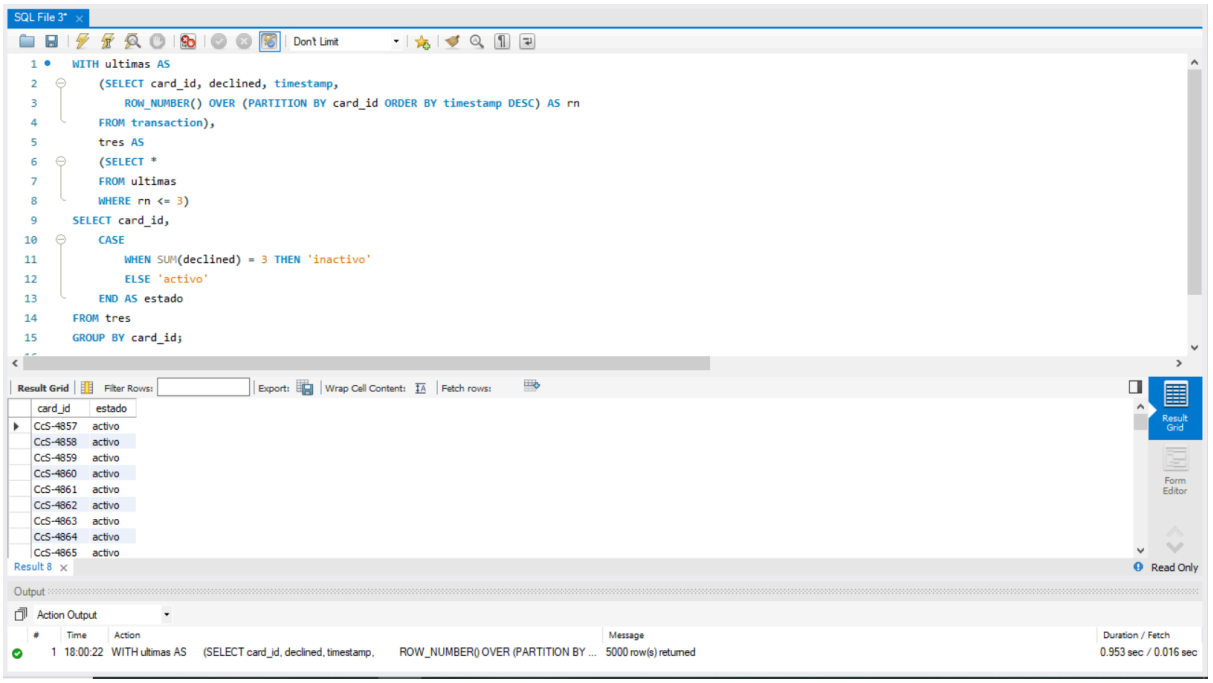


2. Selección de las tres últimas transacciones
Como el ejercicio pide evaluar el estado de la tarjeta basándose solo en las tres últimas transacciones, creamos la CTE tres, filtrando aquellas transacciones donde rn <= 3.
De este modo obtenemos exactamente las tres transacciones más recientes por tarjeta.

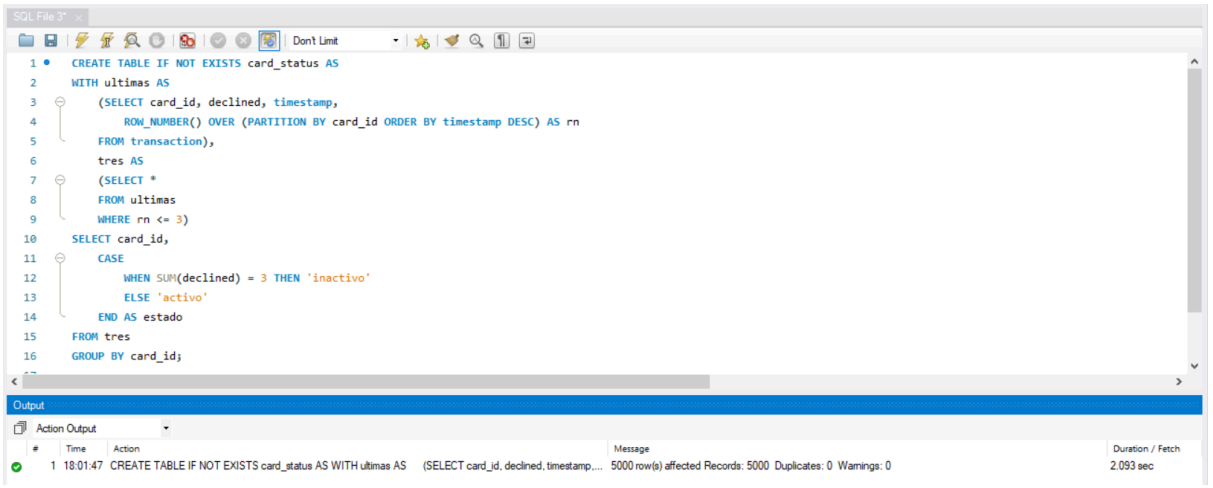


3. Determinación del estado (“activo” vs “inactivo”)
A partir de estas tres transacciones, calculamos el estado de cada tarjeta:

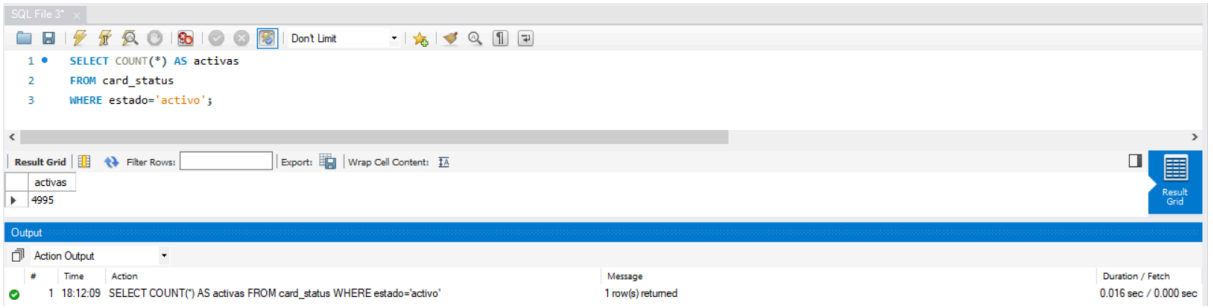
- si las 3 últimas transacciones fueron declinadas → inactivo
- si al menos una fue aceptada → activo



4. Creación de la nueva tabla solicitada (card_status)
Como el ejercicio indica explícitamente que debe crearse una nueva tabla, generamos la tabla card_status a partir de CTE para reflejar el estado final de cada tarjeta según sus transacciones más recientes.



5. Consulta final
Para averiguar cuántas tarjetas están activas, consultamos la tabla creada “card_status”:



Nivel 3

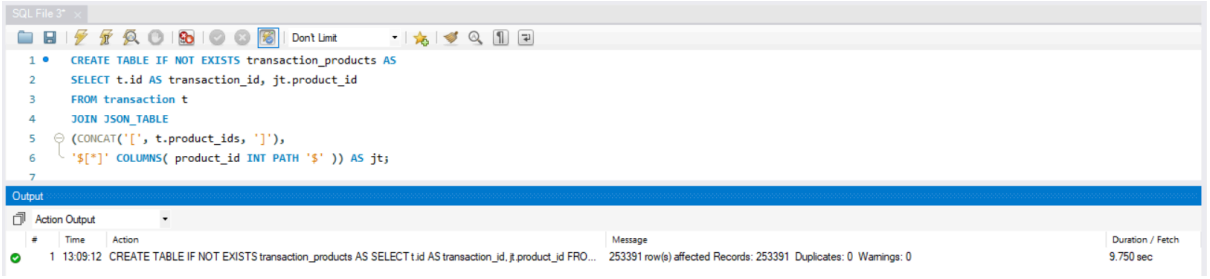
Crea una tabla con la que podamos unir los datos del nuevo archivo products.csv con la base de datos creada, teniendo en cuenta que desde transaction tienes product_ids. Genera la siguiente consulta:

Ejercicio 1

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

1. Creamos una tabla intermedia transaction_products para poder descomponer los product_ids de la tabla transaction.

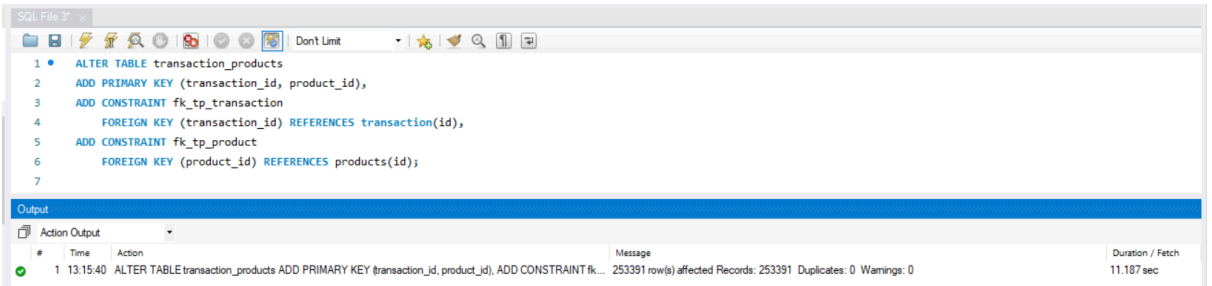
    a. Convertimos la columna product_ids a formato JSON válido, pasando de valores como 75, 73, 98 a [75, 73, 98] usando CONCAT, para que JSON_TABLE pueda interpretarlos como un array.
    b. JSON_TABLE genera una fila por cada producto dentro de cada transacción:
        - '$[*]' recorre todos los elementos del array.
        - product_id INT PATH '$' crea una columna con cada ID.
    c. Hacemos el JOIN correspondiente para crear la tabla que relaciona cada transacción con sus productos.



2. Definimos una llave primaria compuesta (transaction_id, product_id), ya que ni transaction_id ni product_id por sí solos son únicos. Cada fila representa un producto dentro de una transacción.

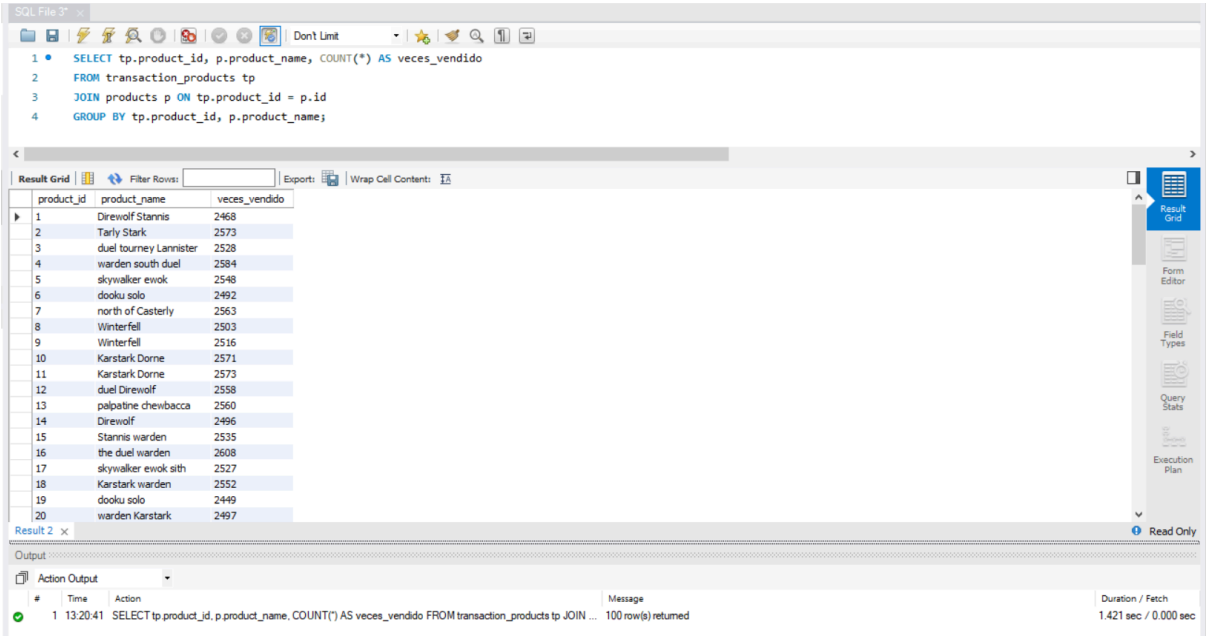
    Añadimos FK hacia las tablas originales:
    - transaction_id → transaction(id)
    - product_id → products(id)

Lo que asegura integridad referencial: cada fila de la tabla intermedia solo contenga transacciones y productos existentes.



3. Consulta final

Utilizamos la tabla products junto con la tabla intermedia, agrupamos por product_id y contamos cuántas veces aparece cada uno para obtener el número de ventas por producto.



```
SELECT tp.product_id, p.product_name, COUNT(*) AS veces_vendido
FROM transaction_products tp
JOIN products p ON tp.product_id = p.id
GROUP BY tp.product_id, p.product_name;
```

product_id	product_name	veces_vendido
1	Direwolf Stannis	2468
2	Tarly Stark	2573
3	duel tourney Lannister	2528
4	warden south duel	2584
5	skywalker ewok	2548
6	dooku solo	2492
7	north of Casterly	2563
8	Winterfell	2503
9	Winterfell	2516
10	Karstark Dorne	2571
11	Karstark Dorne	2573
12	duel Direwolf	2558
13	palpatine chewbacca	2560
14	Direwolf	2496
15	Stannis warden	2535
16	the duel warden	2608
17	skywalker ewok sith	2527
18	Karstark warden	2552
19	dooku solo	2449
20	warden Karstark	2497

La tabla de hechos central sigue siendo transaction.
transaction_products actúa como tabla intermedia para descomponer los productos por transacción.