Tarea S4.01. **Creación de Base de Datos**

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

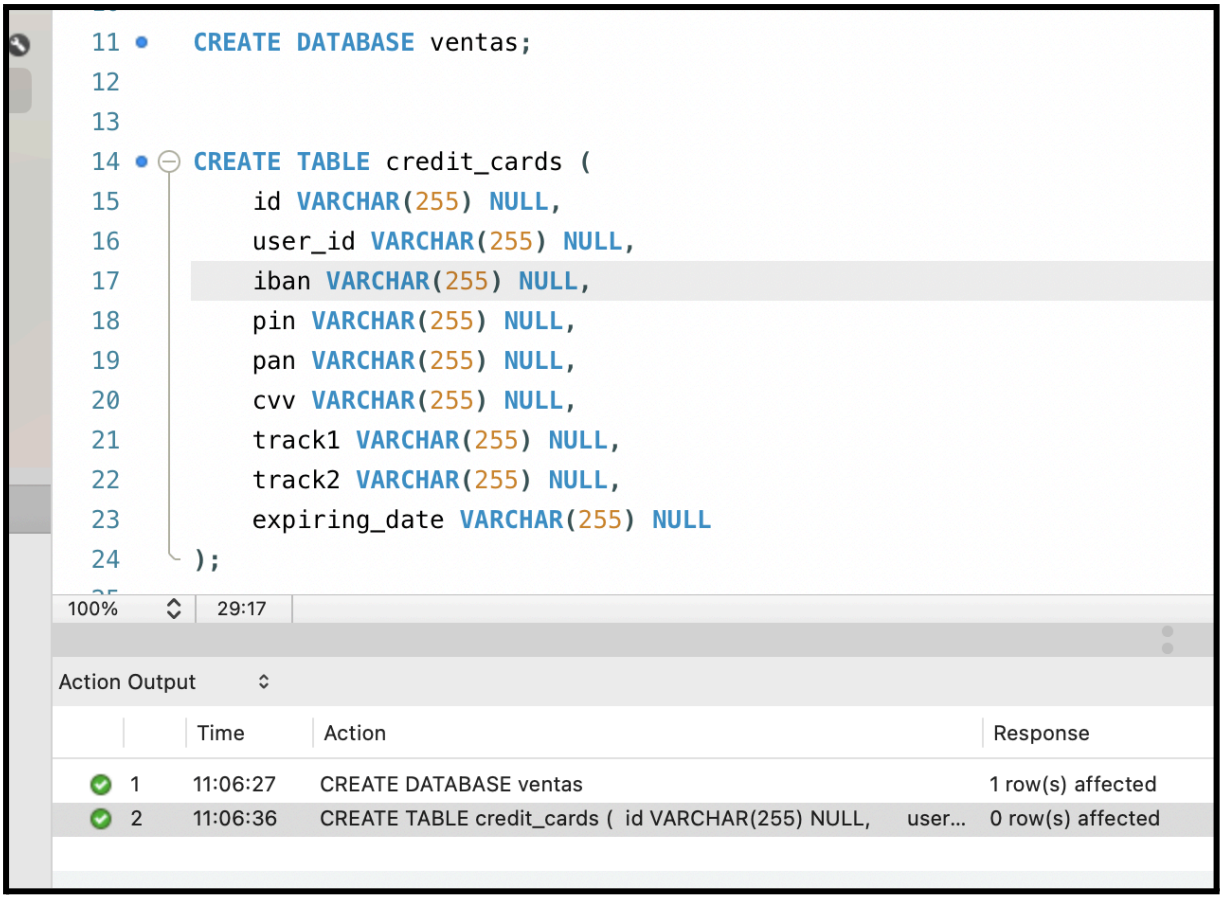
Nivel 1

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

Creación de la base de datos y tablas iniciales:

Se define la base de datos **ventas** y las tablas principales: `credit_cards`, `transactions`, `companies`, `american_users` y `european_users`.

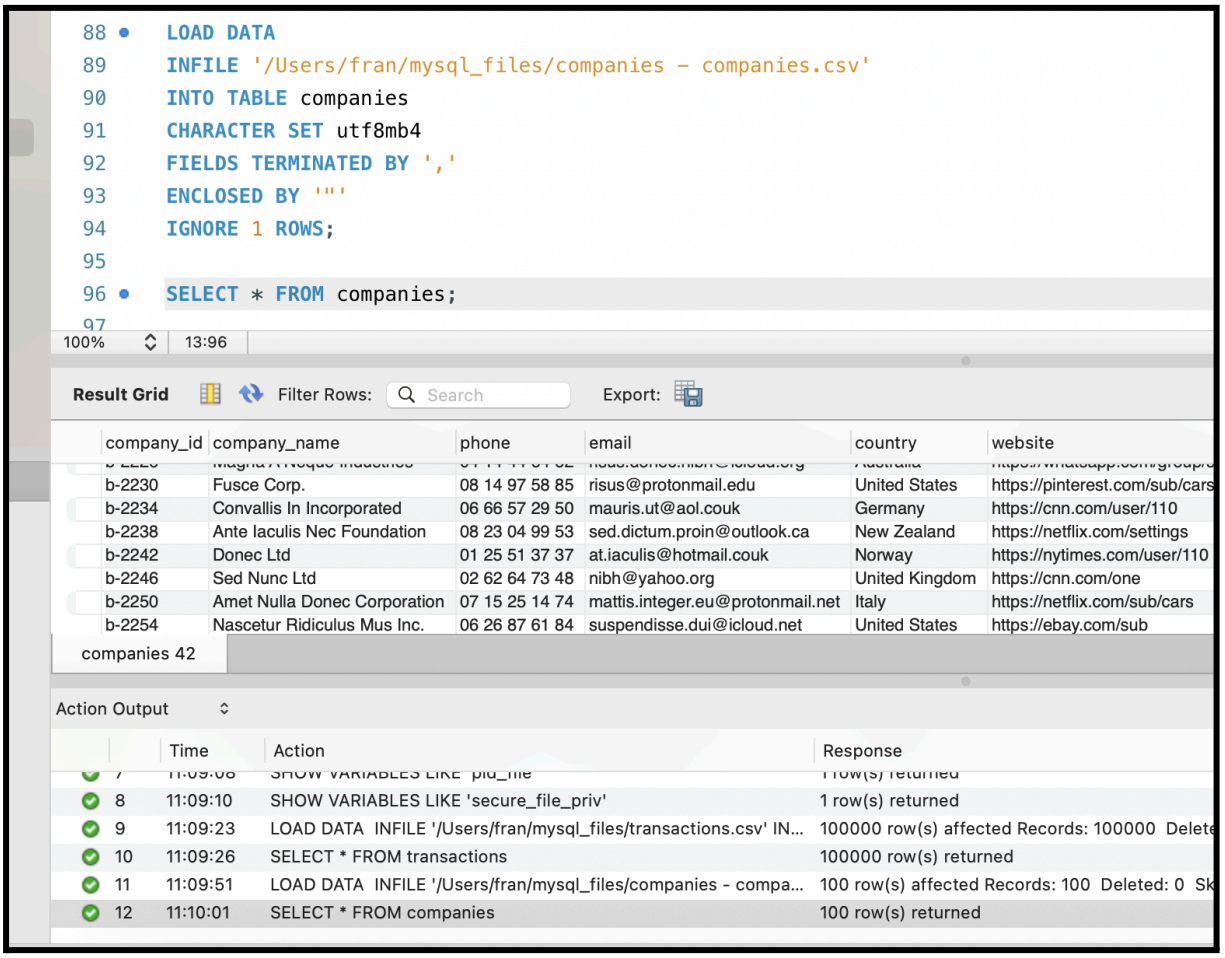
Cada tabla contiene campos textuales (`VARCHAR`) para permitir la carga inicial de datos desde los archivos CSV sin errores de formato ya que todavía no sabemos qué uso necesitaremos darle.


CREATE DATABASE ventas;
CREATE TABLE credit_cards (
id VARCHAR(255) NULL,
user_id VARCHAR(255) NULL,
iban VARCHAR(255) NULL,
pin VARCHAR(255) NULL,
pan VARCHAR(255) NULL,
cvv VARCHAR(255) NULL,
track1 VARCHAR(255) NULL,
track2 VARCHAR(255) NULL,
expiring_date VARCHAR(255) NULL
);
100%
29:17
Action Output
Time
Action
Response
1
11:06:27
CREATE DATABASE ventas
1 row(s) affected
2
11:06:36
CREATE TABLE credit_cards (id VARCHAR(255) NULL, user...
0 row(s) affected

Carga de datos desde archivos CSV:

Se utiliza el comando **LOAD DATA INFILE** para importar la información desde los archivos locales, configurando el tipo de codificación (`utf8mb4`), el delimitador y las comillas de texto.

Se omite la primera fila con **IGNORE 1 ROWS**, ya que contiene los encabezados.



```
88 •  LOAD DATA
89    INFILE '/Users/fran/mysql_files/companies - companies.csv'
90    INTO TABLE companies
91    CHARACTER SET utf8mb4
92    FIELDS TERMINATED BY ','
93    ENCLOSED BY '"'
94    IGNORE 1 ROWS;
95
96 •  SELECT * FROM companies;
97
```

100% 13:96

Result Grid Filter Rows: Search Export:

company_id	company_name	phone	email	country	website
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

companies 42

Action Output

	Time	Action	Response
8	11:09:10	SHOW VARIABLES LIKE 'secure_file_priv'	1 row(s) returned
9	11:09:23	LOAD DATA INFILE '/Users/fran/mysql_files/transactions.csv' IN...	100000 row(s) affected Records: 100000 Delete
10	11:09:26	SELECT * FROM transactions	100000 row(s) returned
11	11:09:51	LOAD DATA INFILE '/Users/fran/mysql_files/companies - compa...	100 row(s) affected Records: 100 Deleted: 0 Sk
12	11:10:01	SELECT * FROM companies	100 row(s) returned

```sql
LOAD DATA
INFILE "/Users/fran/mysql_files/european_users.csv"
INTO TABLE european_users
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

SELECT * FROM european_users;
```

100% 23:106

Result Grid Filter Rows: Search Export: Fetch rows:

id	name	surname	phone	email	birth_date	country	city
153	Keegan	Pugh	(016977) 3851	sodales.nisi@aol.org	Jul 27, 1994	United Kingdom	London
154	Cooper	Bullock	(021) 2521 6627	et@outlook.net	Nov 2, 1986	United Kingdom	Manchester
155	Joshua	Russell	055 4409 5286	justo.nec.ante@outlook.edu	Jan 23, 1984	United Kingdom	Manchester
156	Remedios	Case	055 3114 1566	mollis.phasellus.libero@aol.com	Oct 9, 1994	United Kingdom	Liverpool
157	Philip	Carey	0800 640 6251	phasellus@yahoo.net	Oct 10, 1992	United Kingdom	Manchester
158	Fatima	Dyer	0800 1111	adipiscing@google.org	Dec 24, 1988	United Kingdom	Birmingham
159	Kylynn	Acevedo	056 5727 9602	dignissim.lacus.aliquam@google.org	May 10, 2000	United Kingdom	Liverpool

european_users 43

Action Output

	Time	Action	Response
10	11:09:26	SELECT * FROM transactions	100000 row(s) returned
11	11:09:51	LOAD DATA INFILE '/Users/fran/mysql_files/companies - compa...	100 row(s) affected Records: 100 Dele
12	11:10:01	SELECT * FROM companies	100 row(s) returned
13	11:10:20	LOAD DATA INFILE "/Users/fran/mysql_files/european_users.cs...	3990 row(s) affected Records: 3990
14	11:10:21	SELECT * FROM european_users	3990 row(s) returned

```sql
LOAD DATA
INFILE '/Users/fran/mysql_files/american_users - american_users.csv'
INTO TABLE american_users
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

SELECT * FROM american_users;
```

100% 17:116

Result Grid Filter Rows: Search Export: Fetch rows:

id	name	surname	phone	email	birth_date	country	city	postal_code	address
3	Ciaran	Harrison	(522) 598-1365	interdum.feugiat@aol.org	Apr 29, 1998	United States	Houston	77001	736-2063 Tellus St.
4	Howard	Stafford	1-411-740-3269	ornare.egestas@icloud.edu	feb 18, 1989	United States	Phoenix	85001	Ap #545-2244 Erat. Rd.
5	Hayfa	Pierce	1-554-541-2077	et.malesuada.fames@hotmail.org	Sep 26, 1998	United States	Philadelphia	19101	341-2821 Ultrices Av.
6	Joel	Tyson	(718) 288-8020	gravida.nunc.sed@yahoo.ca	oct 15, 1989	United States	San Jose	95101	888-2799 Amet Street
7	Rafael	Jimenez	(817) 689-0478	eget@outlook.ca	Dec 4, 1981	United States	Chicago	60601	8627 Malesuada Rd.
8	Nissim	Franks	(692) 157-3469	egestas.aliquam.fringilla@google.ca	Aug 1, 1993	United States	New York	10001	Ap #251-7144 Integer St.
9	Mannix	Mcclain	(590) 883-2184	aliquam.nisl@outlook.com	Jan 24, 1987	United States	San Antonio	78201	647-3080 Lacus. St.

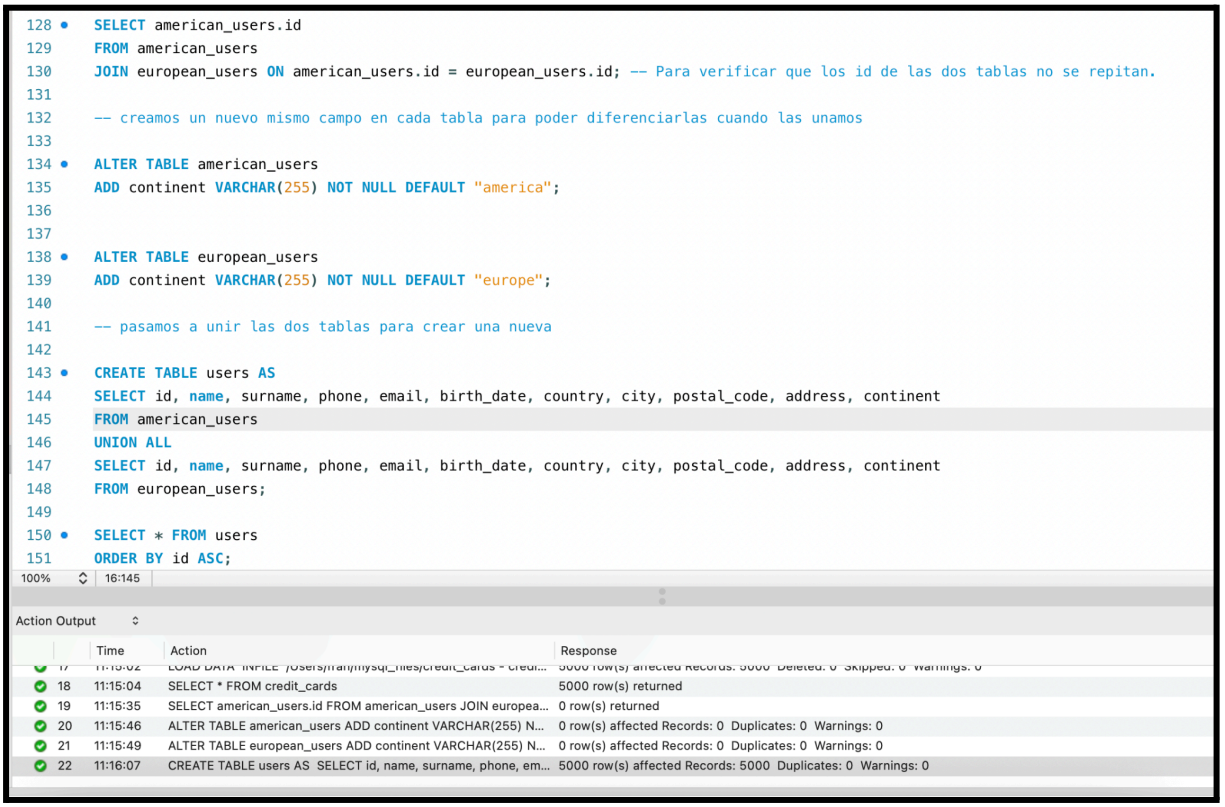
american_users 44

Action Output

	Time	Action	Response
12	11:10:01	SELECT * FROM companies	100 row(s) returned
13	11:10:20	LOAD DATA INFILE "/Users/fran/mysql_files/european_users.cs...	3990 row(s) affected Records: 3990 Deleted: 0 Skipped: 0 Warnings: 0
14	11:10:21	SELECT * FROM european_users	3990 row(s) returned
15	11:10:36	LOAD DATA INFILE '/Users/fran/mysql_files/american_users - a...	1010 row(s) affected Records: 1010 Deleted: 0 Skipped: 0 Warnings: 0
16	11:10:39	SELECT * FROM american_users	1010 row(s) returned

Unificación de tablas de usuarios:
Se ejecuta una consulta **JOIN** entre `american_users` y `european_users` para asegurar que los ID no se repiten antes de unirlas.

Se añade una columna **continent** a cada tabla para identificar su procedencia y luego se crea una tabla unificada **users** con **UNION ALL**, combinando los registros de ambos continentes.
Una vez verificada la correcta unión, se eliminan las tablas originales (`DROP TABLE`).



```sql
SELECT american_users.id
FROM american_users
JOIN european_users ON american_users.id = european_users.id; -- Para verificar que los id de las dos tablas no se repitan.

-- creamos un nuevo mismo campo en cada tabla para poder diferenciarlas cuando las unamos

ALTER TABLE american_users
ADD continent VARCHAR(255) NOT NULL DEFAULT "america";


ALTER TABLE european_users
ADD continent VARCHAR(255) NOT NULL DEFAULT "europe";

-- pasamos a unir las dos tablas para crear una nueva

CREATE TABLE users AS
SELECT id, name, surname, phone, email, birth_date, country, city, postal_code, address, continent
FROM american_users
UNION ALL
SELECT id, name, surname, phone, email, birth_date, country, city, postal_code, address, continent
FROM european_users;

SELECT * FROM users
ORDER BY id ASC;
```

100% 16:145

Action Output

	Time	Action	Response
18	11:15:04	SELECT * FROM credit_cards	5000 row(s) returned
19	11:15:35	SELECT american_users.id FROM american_users JOIN europea...	0 row(s) returned
20	11:15:46	ALTER TABLE american_users ADD continent VARCHAR(255) N...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
21	11:15:49	ALTER TABLE european_users ADD continent VARCHAR(255) N...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
22	11:16:07	CREATE TABLE users AS SELECT id, name, surname, phone, em...	5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0

Estandarización de tipos de datos y restricciones:

- En **companies**, se definen los tipos correspondientes de datos, se establece la clave primaria en `company_id`.
- En **users**, se ajustan tipos, se renombran columnas (`phone` → `personal_phone`, `email` → `personal_email`) porque se repiten los nombres con los de **companies** y así será más fácil distinguirlos. También se formatea la fecha de nacimiento.
  Para esto, se crea una columna auxiliar (**birth_date_clean**) donde se transforma el formato original con **STR_TO_DATE**, luego se reemplaza la columna vieja.

```sql
-- empezamos modificando los tipos de datos en las tablas

ALTER TABLE companies
MODIFY company_id VARCHAR(30) PRIMARY KEY,
MODIFY company_name VARCHAR(255) NOT NULL,
MODIFY phone VARCHAR(30),
MODIFY country VARCHAR(150);

ALTER TABLE users
MODIFY id INT PRIMARY KEY,
MODIFY name VARCHAR(100),
MODIFY surname VARCHAR(100),
CHANGE phone personal_phone VARCHAR(50),
CHANGE email personal_email VARCHAR(255),
MODIFY country VARCHAR(100),
MODIFY city VARCHAR(100),
MODIFY postal_code VARCHAR(20),
MODIFY address VARCHAR(255);
```

100% 27:175

Action Output

	Time	Action	Response
23	11:16:50	SELECT * FROM users ORDER BY id ASC	5000 row(s) returned
24	11:17:17	SELECT COUNT(*) FROM european_users	1 row(s) returned
25	11:17:22	DROP TABLE european_users	0 row(s) affected
26	11:19:28	ALTER TABLE companies MODIFY company_id VARCHAR(30) P...	100 row(s) affected Records: 100 Duplicates: 0 Warnings: 0
27	11:19:30	ALTER TABLE users MODIFY id INT PRIMARY KEY, MODIFY nam...	5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0

Ajustes en tablas de tarjetas y transacciones:

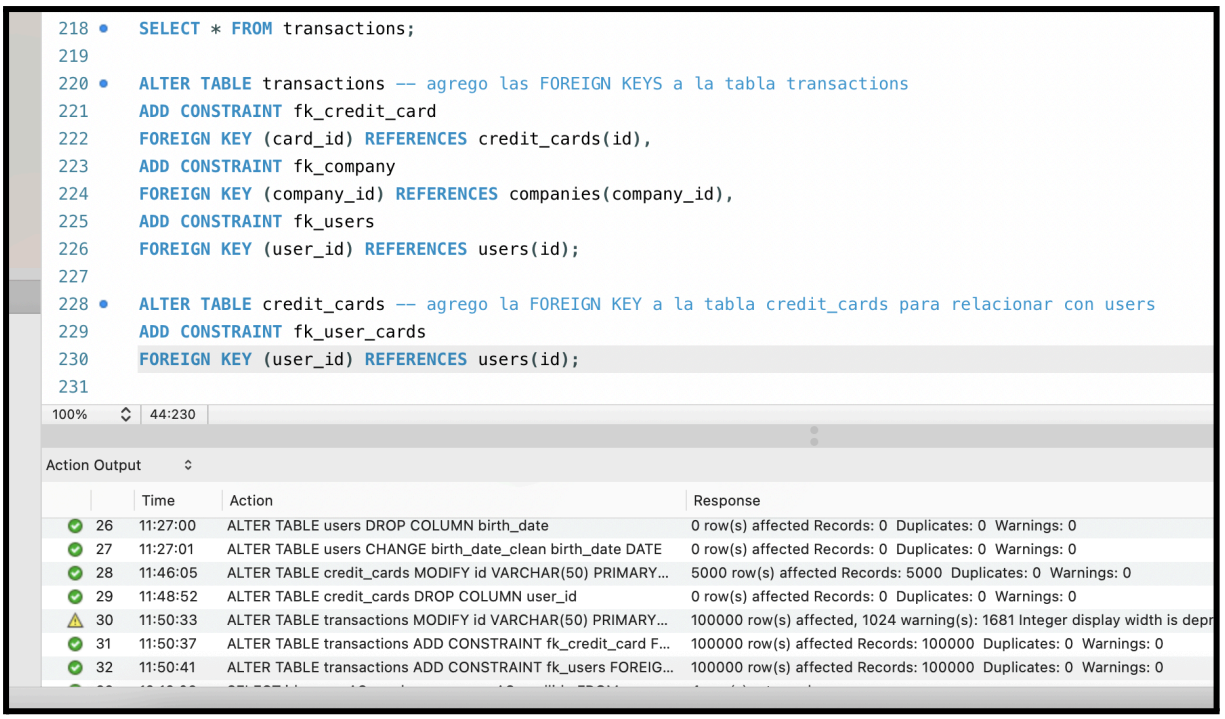
- En **credit_cards**, se normalizan los campos de longitud y se define `id` como clave primaria.
- En **transactions**, se tipifican correctamente los campos numéricos (`amount`, `lat`, `longitude`) y se reemplaza `business_id` por `company_id` para coincidir con la tabla **companies**.

```sql
ALTER TABLE credit_cards
MODIFY id VARCHAR(50) PRIMARY KEY,
MODIFY iban VARCHAR(150),
MODIFY pan VARCHAR(50),
MODIFY pin VARCHAR(50),
MODIFY cvv VARCHAR(10),
MODIFY expiring_date VARCHAR(30);
```

100% 1:205

Action Output

	Time	Action	Response
22	11:26:35	ALTER TABLE users MODIFY id INT PRIMARY KEY, MODIFY nam...	5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0
23	11:26:38	ALTER TABLE users ADD birth_date_clean DATE	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
24	11:26:41	UPDATE users SET birth_date_clean = STR_TO_DATE(birth_dat...	5000 row(s) affected Rows matched: 5000 Changed: 5000 Warnings
25	11:26:44	SELECT id, birth_date, birth_date_clean FROM users	5000 row(s) returned
26	11:27:00	ALTER TABLE users DROP COLUMN birth_date	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
27	11:27:01	ALTER TABLE users CHANGE birth_date_clean birth_date DATE	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
28	11:46:05	ALTER TABLE credit_cards MODIFY id VARCHAR(50) PRIMARY...	5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0



```
218 •  SELECT * FROM transactions;
219
220 •  ALTER TABLE transactions -- agrego las FOREIGN KEYS a la tabla transactions
221    ADD CONSTRAINT fk_credit_card
222    FOREIGN KEY (card_id) REFERENCES credit_cards(id),
223    ADD CONSTRAINT fk_company
224    FOREIGN KEY (company_id) REFERENCES companies(company_id),
225    ADD CONSTRAINT fk_users
226    FOREIGN KEY (user_id) REFERENCES users(id);
227
228 •  ALTER TABLE credit_cards -- agrego la FOREIGN KEY a la tabla credit_cards para relacionar con users
229    ADD CONSTRAINT fk_user_cards
230    FOREIGN KEY (user_id) REFERENCES users(id);
231
```

100% 44:230

Action Output

	Time	Action	Response
26	11:27:00	ALTER TABLE users DROP COLUMN birth_date	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
27	11:27:01	ALTER TABLE users CHANGE birth_date_clean birth_date DATE	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
28	11:46:05	ALTER TABLE credit_cards MODIFY id VARCHAR(50) PRIMARY...	5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0
29	11:48:52	ALTER TABLE credit_cards DROP COLUMN user_id	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
30	11:50:33	ALTER TABLE transactions MODIFY id VARCHAR(50) PRIMARY...	100000 row(s) affected, 1024 warning(s): 1681 Integer display width is depr
31	11:50:37	ALTER TABLE transactions ADD CONSTRAINT fk_credit_card F...	100000 row(s) affected Records: 100000 Duplicates: 0 Warnings: 0
32	11:50:41	ALTER TABLE transactions ADD CONSTRAINT fk_users FOREIG...	100000 row(s) affected Records: 100000 Duplicates: 0 Warnings: 0

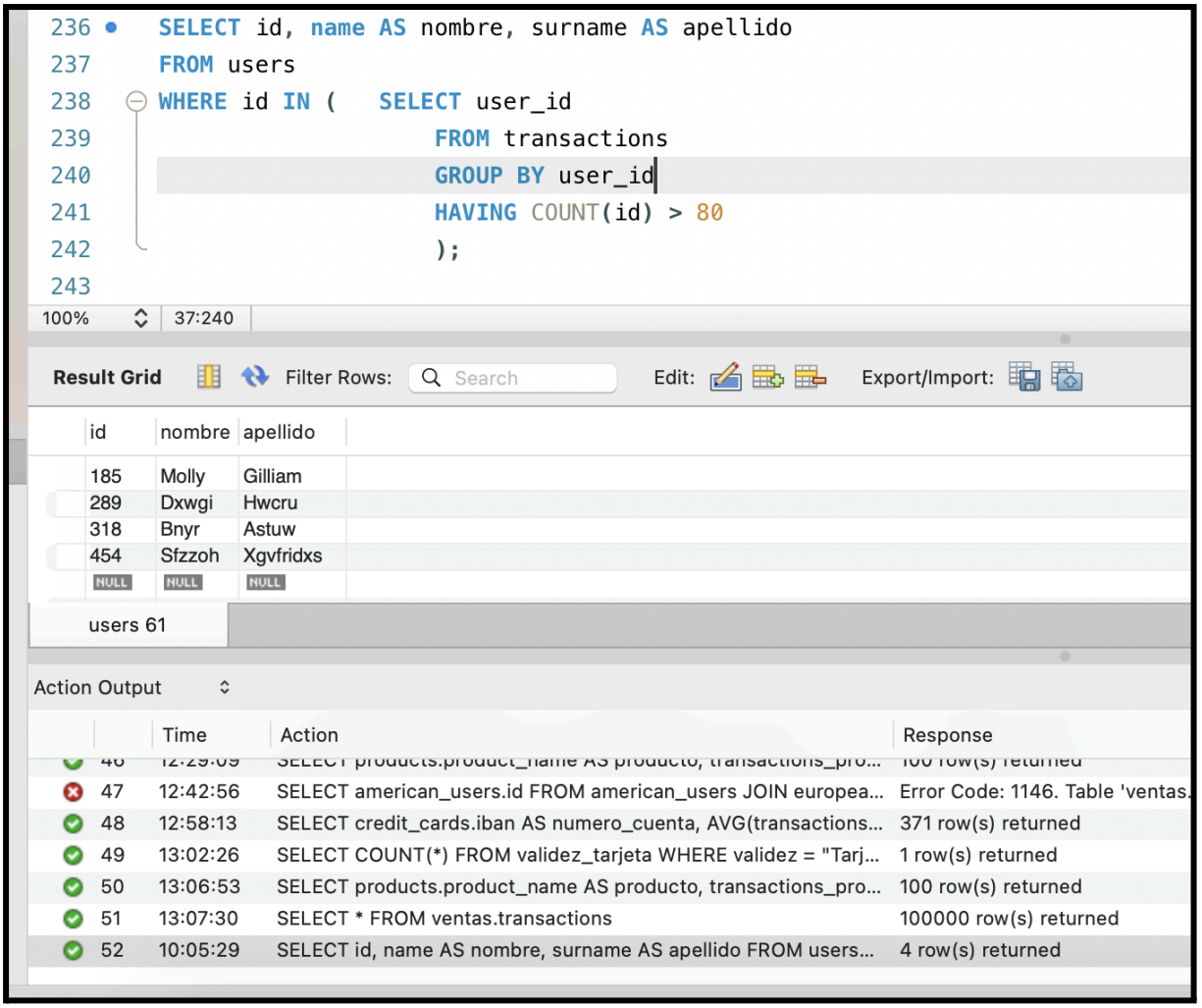
Creación de claves foráneas:
Se agregan relaciones entre las tablas principales:

- `transactions.card_id → credit_cards.id`
- `transactions.company_id → companies.company_id`
- `transactions.user_id → users.id`
- `credit_cards.user_id → users.id`

Estas restricciones aseguran la integridad referencial del modelo en estrella.

Ejercicio 1

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



En la consulta principal se seleccionan los campos **id**, **name** (renombrado como *nombre*) y **surname** (renombrado como *apellido*) de la tabla **users**.

En la condición **WHERE**, se utiliza una **subconsulta** para filtrar únicamente aquellos usuarios cuyo identificador (**id**) aparece en la tabla **transactions** con más de 80 registros asociados.

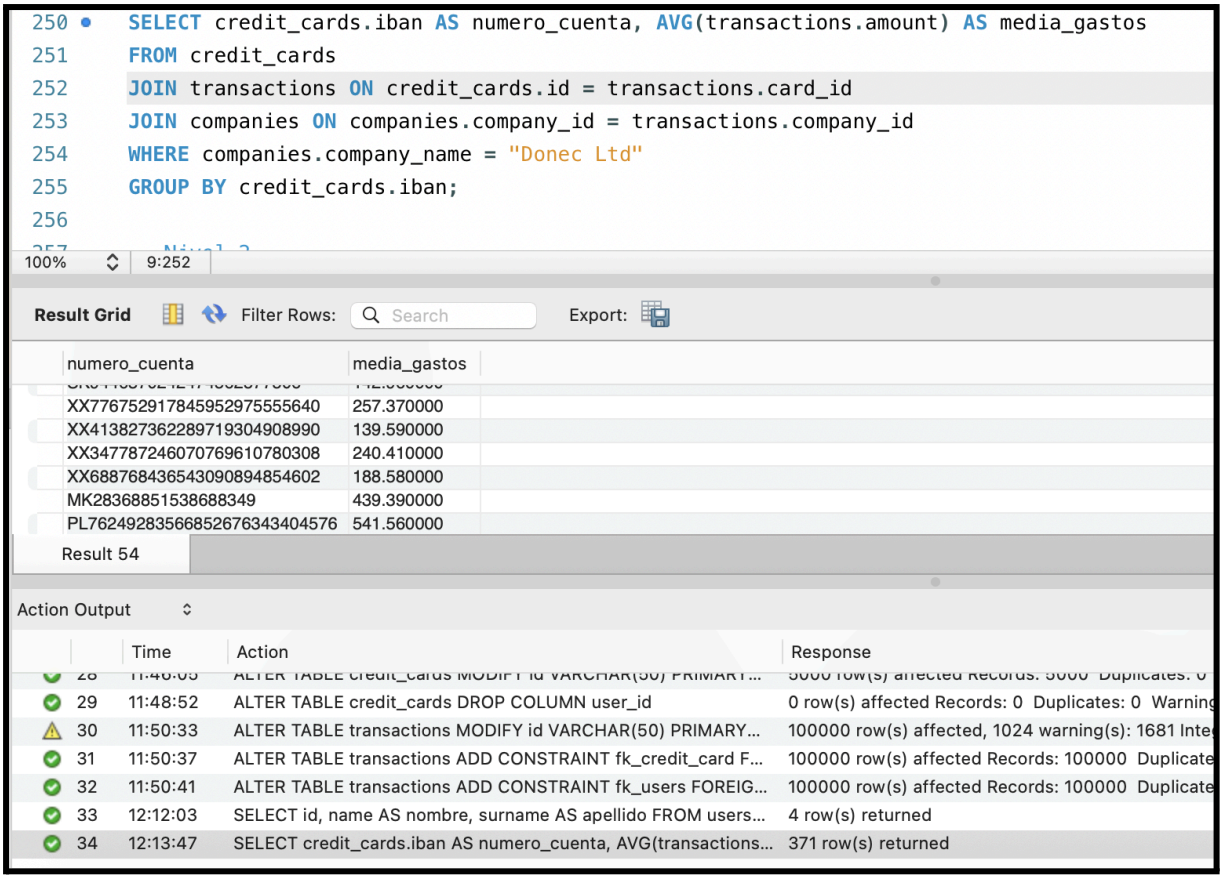
La subconsulta interna:

- Agrupa los registros de **transactions** por el campo **user_id** usando **GROUP BY**.
- Cuenta cuántas transacciones tiene cada usuario con **COUNT(id)**.
- Luego, mediante **HAVING COUNT(id) > 80**, se filtran solo los usuarios con más de 80 operaciones.

Finalmente, el resultado muestra los nombres y apellidos de todos los usuarios que cumplen esta condición, combinando información de **users** (datos personales) y **transactions** (cantidad de operaciones).

Ejercicio 2

Muestra la media de *amount* por IBAN de las tarjetas de crédito en la compañía **Donec Ltd**, utilizando al menos 2 tablas.



```
250 •  SELECT credit_cards.iban AS numero_cuenta, AVG(transactions.amount) AS media_gastos
251    FROM credit_cards
252    JOIN transactions ON credit_cards.id = transactions.card_id
253    JOIN companies ON companies.company_id = transactions.company_id
254    WHERE companies.company_name = "Donec Ltd"
255    GROUP BY credit_cards.iban;
256
```

100% 9:252

Result Grid | Filter Rows: Search | Export:

numero_cuenta	media_gastos
XX776752917845952975555640	257.370000
XX413827362289719304908990	139.590000
XX347787246070769610780308	240.410000
XX688768436543090894854602	188.580000
MK28368851538688349	439.390000
PL762492835668526763434045 76	541.560000

Result 54

Action Output

	Time	Action	Response
29	11:48:52	ALTER TABLE credit_cards DROP COLUMN user_id	0 row(s) affected Records: 0 Duplicates: 0 Warning
30	11:50:33	ALTER TABLE transactions MODIFY id VARCHAR(50) PRIMARY...	100000 row(s) affected, 1024 warning(s): 1681 Inte
31	11:50:37	ALTER TABLE transactions ADD CONSTRAINT fk_credit_card F...	100000 row(s) affected Records: 100000 Duplicate
32	11:50:41	ALTER TABLE transactions ADD CONSTRAINT fk_users FOREIG...	100000 row(s) affected Records: 100000 Duplicate
33	12:12:03	SELECT id, name AS nombre, surname AS apellido FROM users...	4 row(s) returned
34	12:13:47	SELECT credit_cards.iban AS numero_cuenta, AVG(transactions...	371 row(s) returned

Se inicia la consulta seleccionando el campo **iban** de la tabla **credit_cards**, renombrado como *numero_cuenta*, y se calcula la **media de amount** con la función **AVG()**, renombrada como *media_gastos*.

Se realiza un **JOIN** entre las tablas:

- **credit_cards** y **transactions**, unidas por el campo `card_id`, para relacionar cada transacción con la tarjeta utilizada.
- **transactions** y **companies**, unidas por `company_id`, para identificar en qué empresa se efectuó cada operación.

Se aplica un **filtro con WHERE** para limitar los resultados únicamente a las transacciones de la empresa **"Donec Ltd"**.

Finalmente, se usa **GROUP BY credit_cards.iban** para agrupar los resultados por número de cuenta (IBAN) y obtener la media de gasto por cada tarjeta utilizada en esa compañía.

Nivel 2

Crea una nueva tabla que refleje el estado de las tarjetas de crédito en función de si las tres últimas transacciones han sido declinadas: en ese caso es **inactiva**; si al menos una no ha sido rechazada, entonces es **activa**. A partir de esta tabla, responde:

Ejercicio 1

¿Cuántas tarjetas están activas?

```sql
CREATE TABLE validez_tarjeta AS
SELECT card_id AS tarjeta_id, SUM(contador.declined) AS cantidad_transacciones_validas,
    CASE
        WHEN SUM(contador.declined) = 3 THEN 'Tarjeta Invalida'
        WHEN SUM(contador.declined) < 3 THEN 'Tarjeta Valida'
    END AS validez
FROM (  SELECT card_id, declined
        FROM transactions t
        WHERE (
                SELECT COUNT(*)
                FROM transactions t2
                WHERE t2.card_id = t.card_id
                AND t2.timestamp > t.timestamp
                ) < 3
        ) AS contador
GROUP BY card_id;
```

100% 16:270

Action Output

	Time	Action	Response
30	11:50:33	ALTER TABLE transactions MODIFY id VARCHAR(50) PRIMARY...	100000 row(s) affected, 1024 warning(s): 1681 Integer display width
31	11:50:37	ALTER TABLE transactions ADD CONSTRAINT fk_credit_card F...	100000 row(s) affected Records: 100000 Duplicates: 0 Warnings: 0
32	11:50:41	ALTER TABLE transactions ADD CONSTRAINT fk_users FOREIG...	100000 row(s) affected Records: 100000 Duplicates: 0 Warnings: 0
33	12:12:03	SELECT id, name AS nombre, surname AS apellido FROM users...	4 row(s) returned
34	12:13:47	SELECT credit_cards.iban AS numero_cuenta, AVG(transactions...	371 row(s) returned
35	12:15:59	CREATE TABLE validez_tarjeta AS SELECT card_id AS tarjeta_i...	5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0

```sql
SELECT COUNT(*)
FROM validez_tarjeta
WHERE validez = "Tarjeta Valida";
```

100% 12:284

Result Grid Filter Rows: Search Export:

COUNT(*)
4995

Result 55

Action Output

	Time	Action	Response
31	11:50:37	ALTER TABLE transactions ADD CONSTRAINT fk_credit_card F...	100000 row(s) affected
32	11:50:41	ALTER TABLE transactions ADD CONSTRAINT fk_users FOREIG...	100000 row(s) affected
33	12:12:03	SELECT id, name AS nombre, surname AS apellido FROM users...	4 row(s) returned
34	12:13:47	SELECT credit_cards.iban AS numero_cuenta, AVG(transactions...	371 row(s) returned
35	12:15:59	CREATE TABLE validez_tarjeta AS SELECT card_id AS tarjeta_i...	5000 row(s) affected R
36	12:16:23	SELECT COUNT(*) FROM validez_tarjeta WHERE validez = "Tarj...	1 row(s) returned

Primero se crea una tabla llamada **validez_tarjeta** a partir de una subconsulta que analiza las tres últimas transacciones de cada tarjeta.

En la subconsulta interna:

- Se seleccionan los campos **card_id** y **declined** de la tabla **transactions**.
- Se utiliza una condición con **COUNT()** dentro de un **WHERE** correlacionado para identificar las **últimas tres transacciones** por tarjeta, comparando las fechas (**timestamp**) entre sí.
- La condición `t2.timestamp > t.timestamp` permite contar cuántas transacciones posteriores existen, y al filtrar con `< 3` se obtienen únicamente las tres más recientes.

En la consulta principal:

- Se agrupan los resultados por **card_id**.
- Se calcula la suma de los valores del campo **declined**, ya que cada transacción declinada tiene un valor de 1.
- Mediante un **CASE**, se determina la validez de la tarjeta:
    - Si la suma es **3**, significa que las tres transacciones fueron rechazadas → *Tarjeta Invalida*.
    - Si la suma es menor que **3**, al menos una fue aprobada → *Tarjeta Valida*.

Al final solo realizamos COMPLETAR

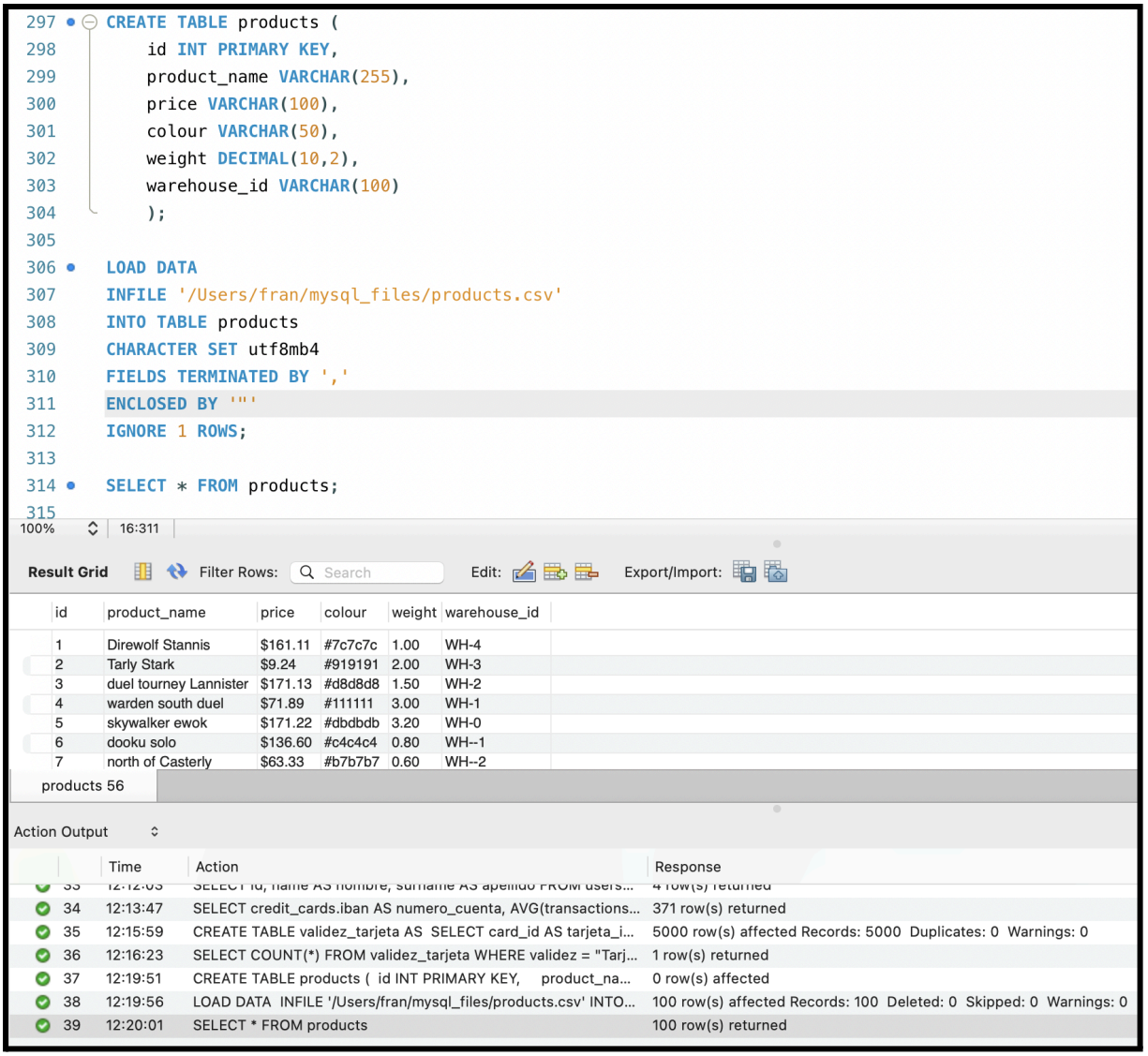
La consulta muestra el número total de tarjetas **activas** en el sistema, es decir, aquellas que **no tuvieron las tres últimas transacciones declinadas**.

Nivel 3

Crea una tabla con la cual 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.



```
297 • CREATE TABLE products (
298       id INT PRIMARY KEY,
299       product_name VARCHAR(255),
300       price VARCHAR(100),
301       colour VARCHAR(50),
302       weight DECIMAL(10,2),
303       warehouse_id VARCHAR(100)
304       );
305
306 •   LOAD DATA
307     INFILE '/Users/fran/mysql_files/products.csv'
308     INTO TABLE products
309     CHARACTER SET utf8mb4
310     FIELDS TERMINATED BY ','
311     ENCLOSED BY '"'
312     IGNORE 1 ROWS;
313
314 •   SELECT * FROM products;
315
```

100% 16:311

Result Grid Filter Rows: Search Edit: Export/Import:

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	#dbdbdb	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

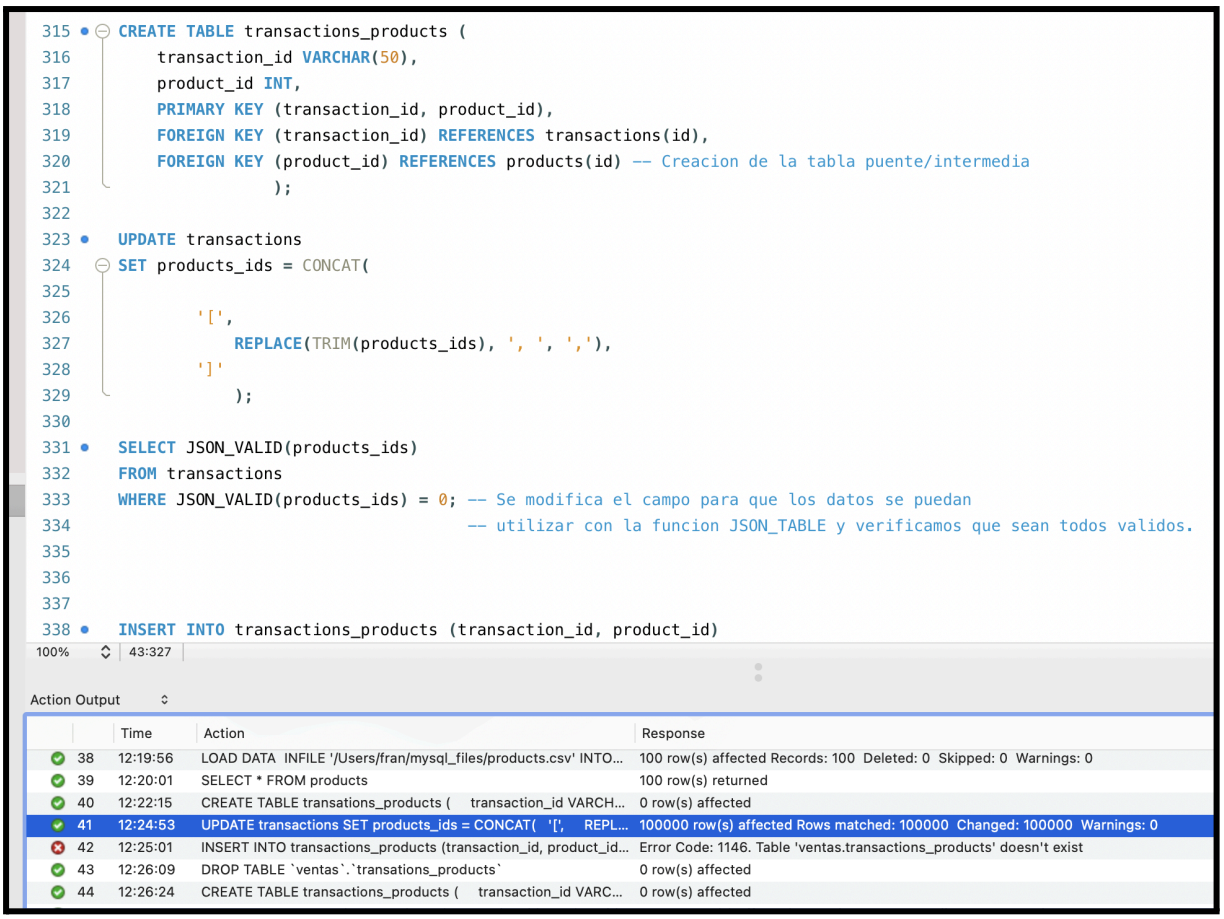
products 56

Action Output

	Time	Action	Response
34	12:13:47	SELECT credit_cards.iban AS numero_cuenta, AVG(transactions...	371 row(s) returned
35	12:15:59	CREATE TABLE validez_tarjeta AS SELECT card_id AS tarjeta_i...	5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0
36	12:16:23	SELECT COUNT(*) FROM validez_tarjeta WHERE validez = "Tarj...	1 row(s) returned
37	12:19:51	CREATE TABLE products (id INT PRIMARY KEY, product_na...	0 row(s) affected
38	12:19:56	LOAD DATA INFILE '/Users/fran/mysql_files/products.csv' INTO...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0
39	12:20:01	SELECT * FROM products	100 row(s) returned

Creación de la tabla `products`:
Se crea una nueva tabla con los campos del archivo `products.csv`, incluyendo información sobre el producto, su precio, color, peso y almacén (**warehouse_id**).
El campo **id** se define como **PRIMARY KEY** para identificar de manera única cada producto.

Carga de datos desde el CSV:
Se utiliza **LOAD DATA INFILE** para importar el archivo `products.csv` a la tabla **products**, especificando la codificación (`utf8mb4`), delimitador de campos y omitiendo la fila de encabezados con **IGNORE 1 ROWS**.



```sql
CREATE TABLE transactions_products (
    transaction_id VARCHAR(50),
    product_id INT,
    PRIMARY KEY (transaction_id, product_id),
    FOREIGN KEY (transaction_id) REFERENCES transactions(id),
    FOREIGN KEY (product_id) REFERENCES products(id) -- Creacion de la tabla puente/intermedia
                );

UPDATE transactions
SET products_ids = CONCAT(

        '[',
            REPLACE(TRIM(products_ids), ', ', ','),
        ']'
            );

SELECT JSON_VALID(products_ids)
FROM transactions
WHERE JSON_VALID(products_ids) = 0; -- Se modifica el campo para que los datos se puedan
                                    -- utilizar con la funcion JSON_TABLE y verificamos que sean todos validos.



INSERT INTO transactions_products (transaction_id, product_id)
```

100% 43:327

Action Output

	Time	Action	Response
38	12:19:56	LOAD DATA INFILE '/Users/fran/mysql_files/products.csv' INTO...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0
39	12:20:01	SELECT * FROM products	100 row(s) returned
40	12:22:15	CREATE TABLE transations_products (transaction_id VARCH...	0 row(s) affected
41	12:24:53	UPDATE transactions SET products_ids = CONCAT('[', REPL...	100000 row(s) affected Rows matched: 100000 Changed: 100000 Warnings: 0
42	12:25:01	INSERT INTO transactions_products (transaction_id, product_id...	Error Code: 1146. Table 'ventas.transactions_products' doesn't exist
43	12:26:09	DROP TABLE `ventas`.`transations_products`	0 row(s) affected
44	12:26:24	CREATE TABLE transactions_products (transaction_id VARC...	0 row(s) affected

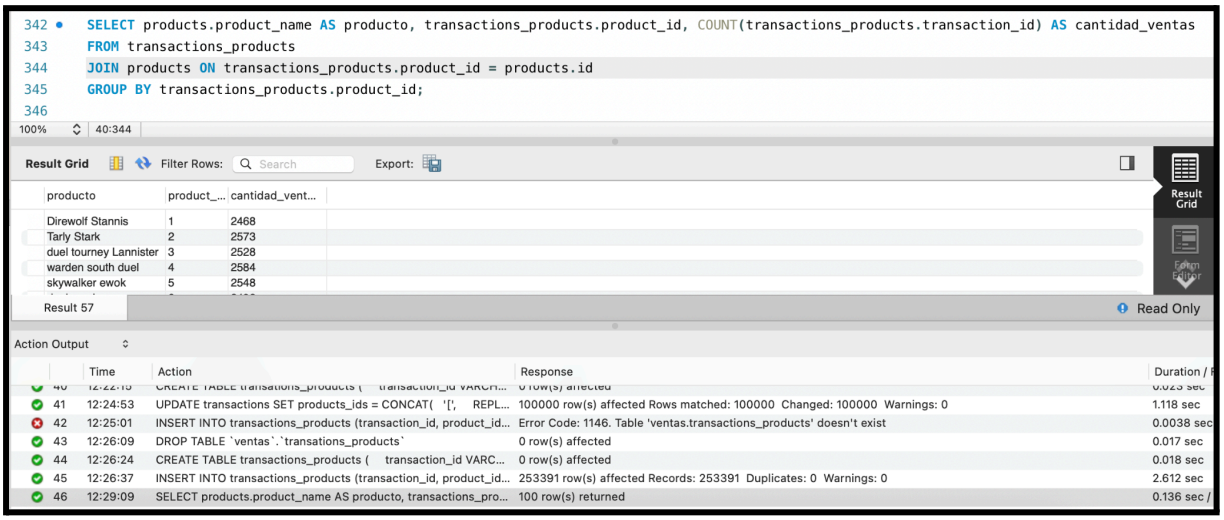
Creación de la tabla intermedia `transactions_products`:
Dado que la relación entre transacciones y productos es **muchos a muchos (N:N)**, se crea una tabla intermedia con dos claves foráneas:

- `transaction_id` (referencia a **transactions**)
- `product_id` (referencia a **products**)
  Además, se establece una **clave primaria compuesta** por ambos campos para evitar duplicados.

Normalización del campo `products_ids`:
En la tabla **transactions**, el campo `products_ids` contenía varios identificadores separados por comas.
Con **UPDATE** y **REPLACE()**, se ajusta el formato para que adopte una estructura válida de **JSON array** (ejemplo: `[1,2,3]`).
Luego se valida con **JSON_VALID()** para comprobar que los datos sean interpretables como JSON.

Descomposición del JSON a filas individuales:
Se utiliza la función **JSON_TABLE()**, que permite extraer los valores del array JSON en filas separadas.

- Cada producto dentro de `products_ids` se convierte en una fila con su `product_id`.
- Esta información se inserta en la tabla intermedia **transactions_products**, junto con el `transaction_id` correspondiente.



Cálculo de ventas por producto:
Finalmente, se realiza una consulta que:

- Une **transactions_products** con **products** mediante el `product_id`.
- Agrupa los resultados por producto con **GROUP BY**.
- Calcula cuántas veces aparece cada producto en transacciones con **COUNT()**, lo que representa la cantidad total de ventas.