

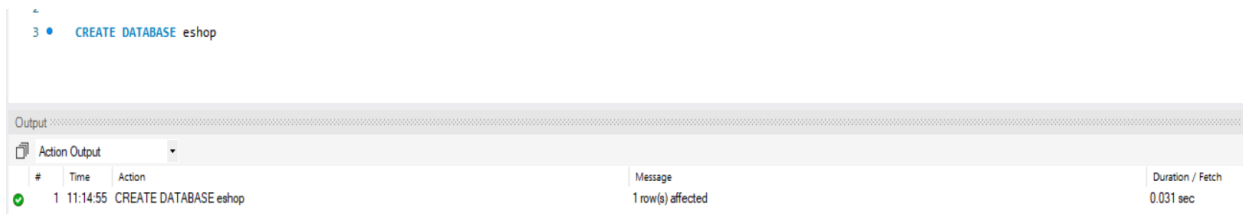
Autora: Giorgia Calvagna  
Fecha de envío: 23/01/2025  
Revisado por: Liubov Shubina

## Sprint 4: Modelat SQL

### Nivel 1:

- Partiendo de algunos archivos CSV diseñarán y crearás tu base de datos:

Empezaremos creando una nueva base de datos, que llamaremos *eshop*:



The screenshot shows a database management interface. At the top, a command bar displays '3 • CREATE DATABASE eshop'. Below this, an 'Output' section is visible. A dropdown menu is set to 'Action Output'. A table of results is shown with the following data:

#	Time	Action	Message	Duration / Fetch
1	11:14:55	CREATE DATABASE eshop	1 row(s) affected	0.031 sec

Una vez creada la base de datos, crearemos las respectivas tablas que la formarán.

Las tablas que crearemos son: *companies*, *credit\_cards*, *products*, *users*, *transactions*.

Tabla *companies*:

```
6 CREATE TABLE IF NOT EXISTS companies (  
7     company_id VARCHAR(15) PRIMARY KEY,  
8     company_name VARCHAR(255) NOT NULL,  
9     phone VARCHAR(15) NULL,  
10    email VARCHAR(100) NOT NULL,  
11    country VARCHAR(100) NULL,  
12    website VARCHAR(255) NULL  
13 );  
14  
15 LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/companies.csv'  
16 INTO TABLE companies  
17 FIELDS TERMINATED BY ','  
18 IGNORE 1 LINES;
```

Output

#	Time	Action	Message	Duration / Fetch
1	18:03:50	CREATE TABLE IF NOT EXISTS companies ( company_id VARCHAR(15) PRIMARY KEY, -- Chiave primar...	0 row(s) affected	0.047 sec
2	18:04:31	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/companies.csv' INTO TABLE com...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0	0.032 sec

Este código SQL tiene como objetivo gestionar datos relacionados con empresas en una base de datos de manera estructurada y eficiente. El primer bloque define la estructura de la tabla llamada "companies", asegurándose de que esta sea creada solo si no existe previamente. Dentro de esta tabla, se establecen varias columnas: "company\_id" como clave primaria, que sirve para identificar de manera única a cada empresa; "company\_name" para almacenar el nombre de la empresa, obligatorio; y otras columnas como "phone", "email", "country" y "website" para registrar información adicional, algunas de las cuales son opcionales.

El segundo bloque de código carga datos en esta tabla desde un archivo externo llamado "companies.csv". Este archivo debe estar ubicado en la ruta especificada y contiene la información que se desea importar. Los datos se separan por comas, y la primera línea del archivo, que normalmente contiene los encabezados, es ignorada durante la importación para evitar conflictos con el contenido real. Este proceso permite poblar rápidamente la tabla con datos preexistentes de forma automatizada.

Así es como se presenta la tabla *companies*:

20	•	SELECT *
21		FROM companies;
22		

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@idoud.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.co.uk	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 75 51 77 77	at.iaculis@hotmail.neu.de	Monaco	https://outlook.com/user/110

#	Time	Action	Message	Duration / Fetch
1	11:59:13	SELECT * FROM companies	100 row(s) returned	0.000 sec / 0.000 sec

Tabla *credit\_cards*:

20	•	CREATE TABLE IF NOT EXISTS credit_cards (
21		id VARCHAR(15) PRIMARY KEY,
22		user_id VARCHAR(100) NOT NULL,
23		iban VARCHAR(40) UNIQUE NOT NULL,
24		pan VARCHAR(45) NOT NULL,
25		pin VARCHAR(5) NULL,
26		cvv VARCHAR(4) NULL,
27		track1 VARCHAR(100) NULL,
28		track2 VARCHAR(100) NULL,
29		expiring_date VARCHAR(10) NULL
30		);
31		
32	•	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/credit_cards.csv'
33		INTO TABLE credit_cards
34		FIELDS TERMINATED BY ','
35		IGNORE 1 LINES;

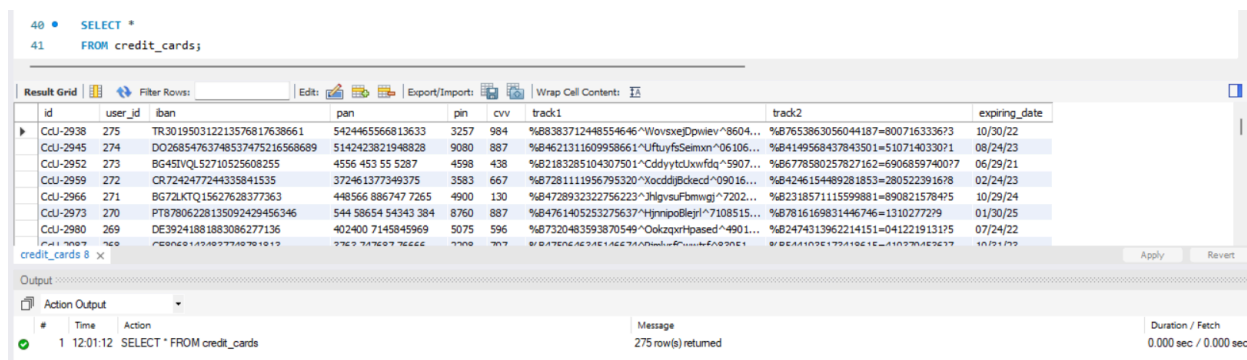
#	Time	Action	Message	Duration / Fetch
1	18:12:23	CREATE TABLE IF NOT EXISTS credit_cards ( id VARCHAR(15) PRIMARY KEY, user_id VARCHAR(100) NOT NULL, iban VARCHAR(40) UNIQUE NOT NULL, pan VARCHAR(45) NOT NULL, pin VARCHAR(5) NULL, cvv VARCHAR(4) NULL, track1 VARCHAR(100) NULL, track2 VARCHAR(100) NULL, expiring_date VARCHAR(10) NULL );	0 row(s) affected	0.062 sec
2	18:12:26	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/credit_cards.csv' INTO TABLE credit_cards FIELDS TERMINATED BY ',' IGNORE 1 LINES;	275 row(s) affected Records: 275 Deleted: 0 Skipped: 0 Warnings: 0	0.031 sec

Este código SQL se centra en informaciones relacionadas con tarjetas de crédito. El primer bloque define la estructura de una tabla llamada "credit\_cards" y se asegura de que sea creada solo si no existe previamente. Dentro de esta tabla, se establecen varias columnas: "id" como clave primaria para identificar de manera única cada tarjeta; "user\_id" para asociar la tarjeta a un usuario específico, obligatorio; "iban" como un identificador único y obligatorio para la cuenta bancaria asociada; "pan" para almacenar el número principal de la tarjeta, también obligatorio; y otras columnas como "pin", "cvv", "track1", "track2" y "expiring\_date" para registrar información adicional, todas ellas opcionales.

El segundo bloque de código carga datos en esta tabla desde un archivo externo llamado "credit\_cards.csv". Este archivo debe estar ubicado en la ruta especificada y contiene la información que se desea importar. Los datos se separan por comas y la primera línea del archivo, que normalmente contiene los encabezados, es ignorada durante la importación para evitar conflictos con el

contenido real. Este proceso permite poblar la tabla con información preexistente de manera rápida y automatizada.

Así es como se presenta la tabla *credit\_cards*:



```
40 • SELECT *
41 FROM credit_cards;
```

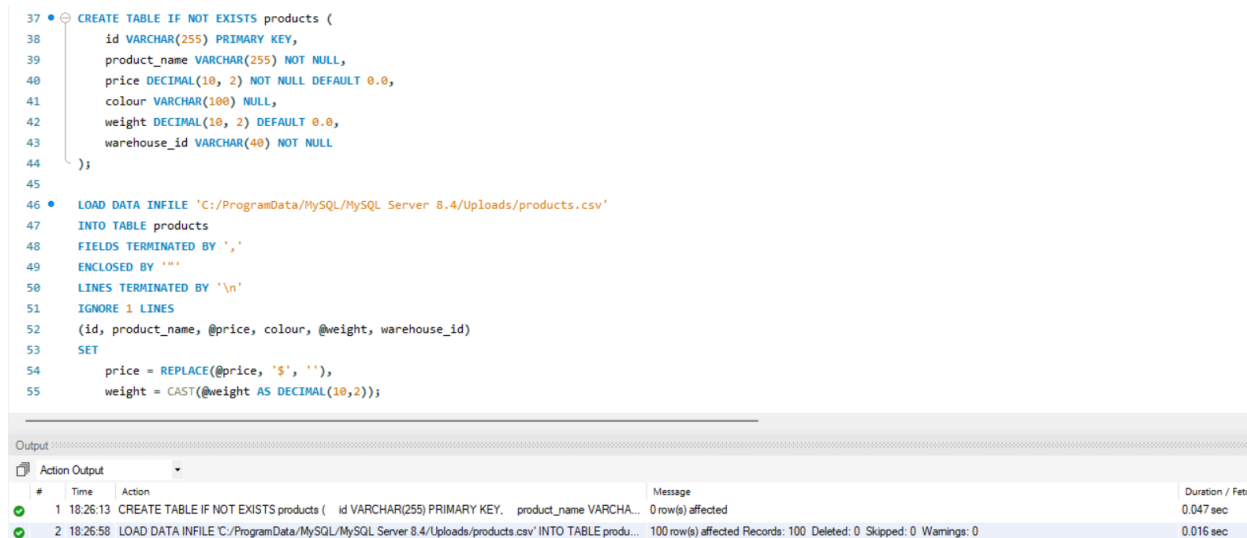
id	user_id	iban	pan	pin	cvv	track1	track2	expiring_date
CDU-2938	275	TR301950312213576817638661	5424465566813633	3257	984	%B8383712448554646~WovsxeJpWiev~8604...	%B7653863056044187=800716333673	10/30/22
CDU-2945	274	DO26854763748537475216568689	5142423821948828	9080	887	%B4621311609958661~UftuyfsSemxm~06106...	%B4149568437843501=510714033071	08/24/23
CDU-2952	273	BG45IVQL52710525608255	4556 453 55 5287	4598	438	%B2183285104307501~CddytdJxwfdq~5907...	%B6778580257827162=6906859740077	06/29/21
CDU-2959	272	CR7242477244333841535	372461377349375	3583	667	%B7281111956795320~XocddjBdecd~09016...	%B4246154489281853=280522391678	02/24/23
CDU-2966	271	BG72LKTQ15627628377363	448566 886747 7265	4900	130	%B4728932322756223~JhqvsvFbmvgi~7202...	%B2318571115599881=890821578475	10/29/24
CDU-2973	270	PT87806228135092429456346	544 58654 54343 384	8760	887	%B4761405253275637~Hjnpoblej~7108515...	%B7816169831446746=1310277279	01/30/25
CDU-2980	269	DE39241881883086277136	402400 7145845969	5075	596	%B7320483593870549~OokzqrHpsed~4901...	%B2474313962214151=041221913175	07/24/22

credit\_cards 0 x

Output

#	Time	Action	Message	Duration / Fetch
1	12:01:12	SELECT * FROM credit_cards	275 row(s) returned	0.000 sec / 0.000 sec

Tabla *products*:



```
37 • CREATE TABLE IF NOT EXISTS products (
38     id VARCHAR(255) PRIMARY KEY,
39     product_name VARCHAR(255) NOT NULL,
40     price DECIMAL(10, 2) NOT NULL DEFAULT 0.0,
41     colour VARCHAR(100) NULL,
42     weight DECIMAL(10, 2) DEFAULT 0.0,
43     warehouse_id VARCHAR(40) NOT NULL
44 );
45
46 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/products.csv'
47 INTO TABLE products
48 FIELDS TERMINATED BY ','
49 ENCLOSED BY '"'
50 LINES TERMINATED BY '\n'
51 IGNORE 1 LINES
52 (id, product_name, @price, colour, @weight, warehouse_id)
53 SET
54     price = REPLACE(@price, '$', ''),
55     weight = CAST(@weight AS DECIMAL(10,2));
```

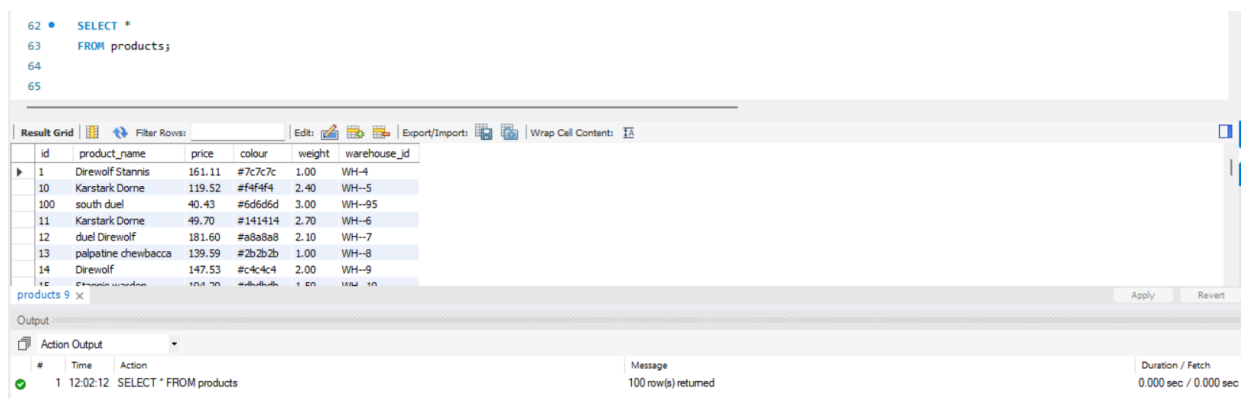
Output

#	Time	Action	Message	Duration / Fetch
1	18:26:13	CREATE TABLE IF NOT EXISTS products ( id VARCHAR(255) PRIMARY KEY, product_name VARCHA...	0 row(s) affected	0.047 sec
2	18:26:58	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/products.csv' INTO TABLE produ...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0	0.016 sec

Este código SQL presenta una particularidad en la forma en que maneja los datos durante su importación en la tabla "products". Aunque sigue la lógica de creación y carga de datos descrita anteriormente, se diferencia en que incluye instrucciones adicionales para transformar ciertos datos antes de almacenarlos. Las columnas que se importarán son: id, product\_name, price, colour, weight y warehouse\_id. Para las columnas price y weight, se utilizan variables temporales (@price y @weight) porque requieren un procesamiento especial a través del comando "SET".

En el caso de "price", se reemplaza (usando el comando "REPLACE") el símbolo "\$" de los valores importados con una cadena vacía ("") para asegurar que sean tratados como datos numéricos. Para "weight", con el comando "CAST" cualquier valor numérico (ya sea entero o decimal) es convertido al formato DECIMAL(10,2), garantizando consistencia en los datos almacenados. Además, se especifica que los campos en el archivo de origen están delimitados por comas y encerrados entre comillas dobles, y que cada línea termina con un salto de línea, lo que asegura una interpretación precisa del archivo durante la carga.

Así es como se presenta la tabla *products*:



The screenshot shows a database management tool interface. At the top, a SQL query is entered in a text area:

```
62 SELECT *
63 FROM products;
64
65
```

Below the query, a "Result Grid" displays the data from the *products* table. The grid has columns: *id*, *product\_name*, *price*, *colour*, *weight*, and *warehouse\_id*. The data is as follows:

id	product_name	price	colour	weight	warehouse_id
1	Direwolf Stannis	161.11	#7c7c7c	1.00	WH-4
10	Karstark Dome	119.52	#f4f4f4	2.40	WH-5
100	south duel	40.43	#6d6d6d	3.00	WH-95
11	Karstark Dome	49.70	#141414	2.70	WH-6
12	duel Direwolf	181.60	#a8a8a8	2.10	WH-7
13	palpatine chewbacca	139.59	#2b2b2b	1.00	WH-8
14	Direwolf	147.53	#c4c4c4	2.00	WH-9
16	Emmerys cardon	124.70	#d4d4d4	1.00	WH-10

Below the result grid, an "Output" section shows the execution details of the query:

#	Time	Action	Message	Duration / Fetch
1	12:02:12	SELECT * FROM products	100 row(s) returned	0.000 sec / 0.000 sec

Tabla *users*:

Los ficheros de que disponemos con respecto a la tabla de *users* son tres: *user\_ca*, *user\_uk*, *user\_usa*. La estructura de estas tablas es la misma, por lo que crearemos una única tabla de usuarios que las contenga a todas.

```
58 • CREATE TABLE IF NOT EXISTS users (  
59     id INT PRIMARY KEY,  
60     name VARCHAR(100) NOT NULL,  
61     surname VARCHAR(100) NOT NULL,  
62     phone VARCHAR(20),  
63     email VARCHAR(100),  
64     birth_date DATE,  
65     country VARCHAR(100),  
66     city VARCHAR(100),  
67     postal_code VARCHAR(20),  
68     address VARCHAR(255),  
69     INDEX idx_email (email),  
70     INDEX idx_phone (phone)  
71 );  
72  
73 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/users_ca.csv'  
74 INTO TABLE users  
75 FIELDS TERMINATED BY ','  
76 OPTIONALLY ENCLOSED BY ''''  
77 LINES TERMINATED BY '\r\n'  
78 IGNORE 1 LINES  
79 (id, name, surname, phone, email, @birth_date, country, city, postal_code, address)  
80 SET birth_date = STR_TO_DATE(TRIM(BOTH '''' FROM @birth_date), '%b %d, %Y');  
81  
82 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/users_uk.csv'  
83 INTO TABLE users  
84 FIELDS TERMINATED BY ','  
85 OPTIONALLY ENCLOSED BY ''''  
86 LINES TERMINATED BY '\r\n'  
87 IGNORE 1 LINES  
88 (id, name, surname, phone, email, @birth_date, country, city, postal_code, address)  
89 SET birth_date = STR_TO_DATE(TRIM(BOTH '''' FROM @birth_date), '%b %d, %Y');  
90  
91 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/users_usa.csv'  
92 INTO TABLE users  
93 FIELDS TERMINATED BY ','  
94 OPTIONALLY ENCLOSED BY ''''  
95 LINES TERMINATED BY '\r\n'  
96 IGNORE 1 LINES  
97 (id, name, surname, phone, email, @birth_date, country, city, postal_code, address)  
98 SET birth_date = STR_TO_DATE(TRIM(BOTH '''' FROM @birth_date), '%b %d, %Y');  
99
```

Output

#	Time	Action	Message	Duration / Fetch
1	18:41:17	CREATE TABLE IF NOT EXISTS users ( id INT PRIMARY KEY, name VARCHAR(100) NOT NULL, su...	0 row(s) affected	0.063 sec
2	18:41:57	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/users_ca.csv' INTO TABLE users ...	75 row(s) affected Records: 75 Deleted: 0 Skipped: 0 Warnings: 0	0.016 sec
3	18:42:14	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/users_uk.csv' INTO TABLE users ...	50 row(s) affected Records: 50 Deleted: 0 Skipped: 0 Warnings: 0	0.031 sec
4	18:42:32	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/users_usa.csv' INTO TABLE user...	150 row(s) affected Records: 150 Deleted: 0 Skipped: 0 Warnings: 0	0.031 sec

Este código SQL se centra en la gestión de los datos relacionados con usuarios, consolidando información de distintas regiones en una sola tabla llamada "users". La estructura de la tabla incluye campos clave como "id", "name" y "surname" para identificar a los usuarios, junto con otros campos adicionales

como "phone", "email", "birth\_date", "country", "city", "postal\_code" y "address" para registrar información personal y de contacto. Además, se crean índices en las columnas "email" y "phone" para optimizar las consultas que involucren estos campos, mejorando el rendimiento en la búsqueda y filtrado de datos.

El código también incluye instrucciones para cargar datos desde tres archivos de origen: "users\_ca.csv", "users\_uk.csv" y "users\_usa.csv". Aunque estos archivos contienen datos de diferentes regiones (Canadá, Reino Unido y Estados Unidos), comparten la misma estructura, lo que facilita su integración en una única tabla (*users*). Durante el proceso de carga, los datos se separan por comas, se manejan cadenas opcionalmente encerradas entre comillas dobles y cada línea finaliza con un salto de línea.

Este código realiza una serie de transformaciones para convertir una fecha desde un texto a un formato de fecha válido en la base de datos.

id	name	phone	email	birth_date	country	city	postal_code	address
201	Lola Powers	018-139-4717	ante.blandit@outlook.eu	"Mar 20, 2000"	Canada	Rigolet	V6T 6M7	154-5415 Auctor St.

Comencemos con la variable temporal `@birth_date`, que contiene la fecha tal como se lee del archivo CSV, en un formato como "Mar 20, 2000".

A continuación, se aplica la función `TRIM`, que elimina las comillas dobles tanto al principio como al final del texto almacenado en `@birth_date`. En el ejemplo "Mar 20, 2000", después del `TRIM` tendremos Mar 20, 2000, sin las comillas.

Después entra en juego la función `STR_TO_DATE`, que toma este texto limpio y lo convierte en una fecha real siguiendo el patrón especificado `'%b %d, %Y'`. En este patrón, `%b` representa el mes en formato abreviado como "Mar", `%d` representa el día del mes como un número del 1 al 31, y `%Y` representa el año con cuatro dígitos.

Finalmente, todo el resultado de estas transformaciones se asigna a la columna *birth\_date* de la tabla mediante el comando `SET`. De esta manera, si empezamos con el texto "Mar 20, 2000", al final del proceso tendremos una fecha en formato 2000-03-20 almacenada correctamente en la base de datos.

Esto asegura la consistencia y la integridad de los datos importados.

Así es como se presenta la tabla *users*:

201 • `select *`  
202 `from users;`

	id	name	surname	phone	email	birth_date	country	city	postal_code	address
1	Zeus	Gamble		1-282-581-0551	interdum.enim@protonmail.edu	1985-11-17	United States	Lowell	73544	348-7818 Sagittis St.
2	Garrett	Mcconnell		(718) 257-2412	integer.vitae.nibh@protonmail.org	1992-08-23	United States	Des Moines	59464	903 Sit Ave
3	Ciaran	Harrison		(522) 598-1365	interdum.feugiat@aol.org	1998-04-29	United States	Columbus	56518	736-2063 Tellus St.
4	Howard	Stafford		1-411-740-3269	omare.egestas@cloud.edu	1989-02-18	United States	Kailua	77417	Ap #545-2244 Erat. Rd.
5	Hayfa	Pierce		1-554-541-2077	et.malesuada.fames@hotmail.org	1998-09-26	United States	Sandy	31564	341-2821 Ultrices Av.
6	Joel	Tyson		(718) 288-8020	gravida.nunc.sed@yahoo.ca	1989-10-15	United States	Nashville	96838	888-2799 Amet Street

users 2 x

Output

#	Time	Action	Message	Duration / Fetch
1	11:53:10	select * from users	275 row(s) returned	0.000 sec / 0.000 sec

Tabla *transactions*:

```

100 • CREATE TABLE IF NOT EXISTS transactions (
101     id VARCHAR(255) PRIMARY KEY,
102     card_id VARCHAR(40) NOT NULL,
103     business_id VARCHAR(40) NOT NULL,
104     timestamp TIMESTAMP NOT NULL,
105     amount DECIMAL(10, 2) NOT NULL,
106     declined BOOLEAN NOT NULL,
107     product_ids VARCHAR(40) NULL,
108     user_id INT NOT NULL,
109     lat FLOAT,
110     longitude FLOAT,
111     CONSTRAINT fk_transactions_credit_cards FOREIGN KEY (card_id) REFERENCES credit_cards(id),
112     CONSTRAINT fk_transactions_companies FOREIGN KEY (business_id) REFERENCES companies(company_id),
113     CONSTRAINT fk_transactions_users FOREIGN KEY (user_id) REFERENCES users(id),
114     INDEX idx_card_id (card_id),
115     INDEX idx_business_id (business_id),
116     INDEX idx_user_id (user_id)
117 );
118
119 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/transactions.csv'
120 INTO TABLE transactions
121 FIELDS TERMINATED BY ','
122 ENCLOSED BY ''

```

Output

#	Time	Action	Message	Duration / Fetch
1	19:16:07	CREATE TABLE IF NOT EXISTS transactions ( id VARCHAR(255) PRIMARY KEY, card_id VARCHAR(40) NOT NULL, business_id VARCHAR(40) NOT NULL, timestamp TIMESTAMP NOT NULL, amount DECIMAL(10, 2) NOT NULL, declined BOOLEAN NOT NULL, product_ids VARCHAR(40) NULL, user_id INT NOT NULL, lat FLOAT, longitude FLOAT, CONSTRAINT fk_transactions_credit_cards FOREIGN KEY (card_id) REFERENCES credit_cards(id), CONSTRAINT fk_transactions_companies FOREIGN KEY (business_id) REFERENCES companies(company_id), CONSTRAINT fk_transactions_users FOREIGN KEY (user_id) REFERENCES users(id), INDEX idx_card_id (card_id), INDEX idx_business_id (business_id), INDEX idx_user_id (user_id) )	0 row(s) affected	0.109 sec
2	19:16:53	LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/transactions.csv' INTO TABLE transactions	587 row(s) affected Records: 587 Deleted: 0 Skipped: 0 Warnings: 0	0.109 sec

La tabla "transactions" se define como la *fact table* del sistema, lo que significa que sirve como el punto central para consolidar y analizar información relacionada con transacciones financieras. Contiene detalles clave de cada transacción, como su identificador único ("id"), el monto ("amount"), la fecha y hora ("timestamp"), así como el estado de la transacción ("declined", que indica si fue aprobada o rechazada).

Un aspecto fundamental de esta tabla es la declaración de las claves externas ("foreign keys"), que establecen relaciones con otras tablas del sistema:



- La columna "card\_id" está vinculada con la tabla "credit\_cards" a través de la clave externa "fk\_transactions\_credit\_cards", lo que permite rastrear qué tarjeta de crédito se utilizó en cada transacción.
- La columna "business\_id" referencia la tabla "companies" mediante la clave externa "fk\_transactions\_companies", conectando cada transacción con la empresa o comercio involucrado.
- La columna "user\_id" está relacionada con la tabla "users" mediante la clave externa "fk\_transactions\_users", lo que permite asociar cada transacción con el usuario que la realizó.

Estas claves externas no sólo garantizan la integridad referencial entre las tablas, sino que también permiten realizar consultas y análisis más complejos al vincular datos de diferentes dimensiones (usuarios, empresas y tarjetas de crédito) de manera estructurada.

Adicionalmente, se incluyen índices en las columnas "card\_id", "business\_id" y "user\_id" para optimizar la velocidad de las consultas relacionadas con estas claves. Finalmente, los campos "lat" y "longitude" permiten almacenar la ubicación geográfica donde se realizó cada transacción, añadiendo un contexto espacial a los datos registrados.

Así es como se presenta la tabla *transactions*:

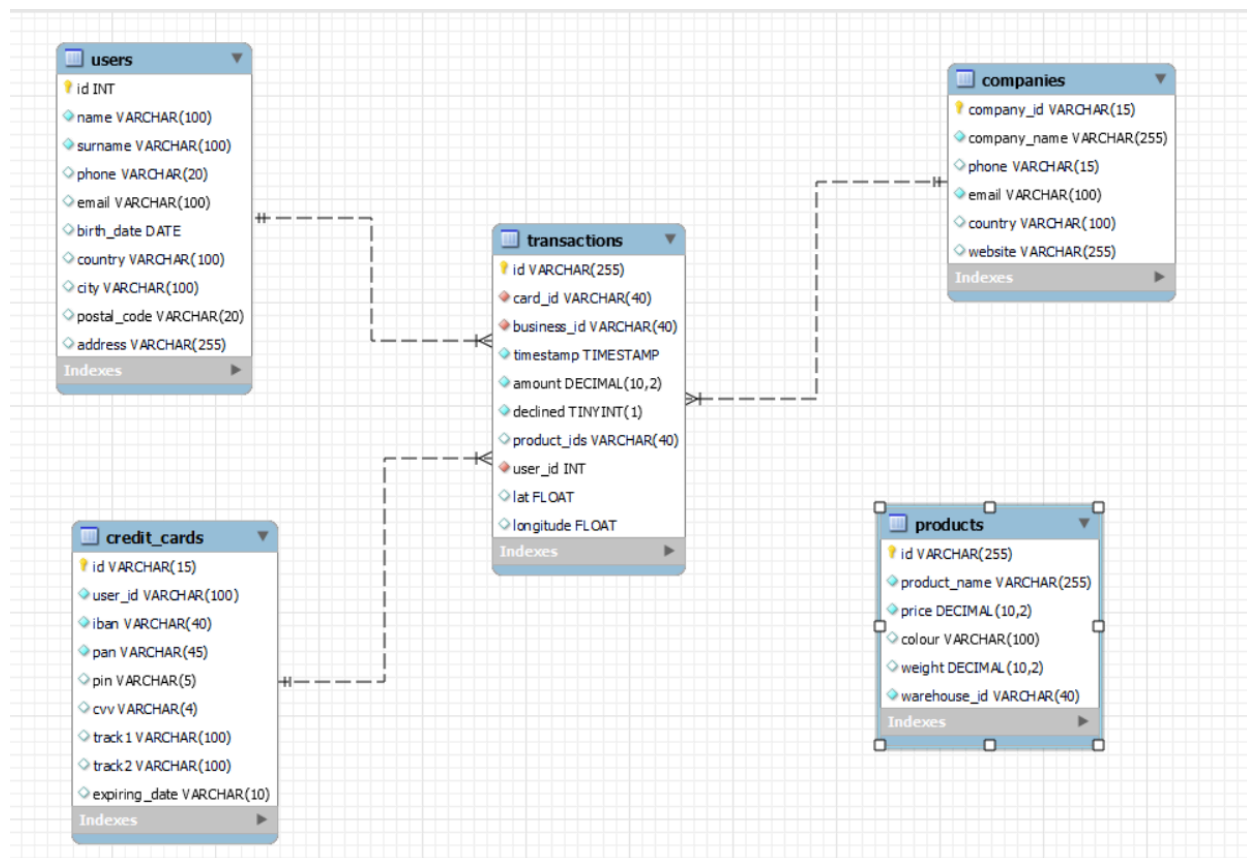
id	card_id	business_id	timestamp	amount	declined	product_ids	user_id	lat	longitude
02C6201E-D90A-1859-84EE-88D2986D3802	CcU-2938	b-2362	2021-08-28 23:42:24	466.92	0	71, 1, 19	92	81.9185	-12.5276
0466A42E-47CF-8D24-FD01-C0B689713128	CcU-4219	b-2302	2021-07-26 07:29:18	49.53	0	47, 97, 43	170	-43.9695	-117.525
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-D684-0DD0-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-6F27-28B7-13B5-9385B2B3B8E2	CcU-3071	b-2298	2021-05-11 20:40:06	303.05	1	67, 7	275	71.1706	10.5757

Output: Action Output

Message: 587 row(s) returned

Duration / Fetch: 0.000 sec / 0.000 sec

El diagrama del schema *eshop* de momento se presenta así:

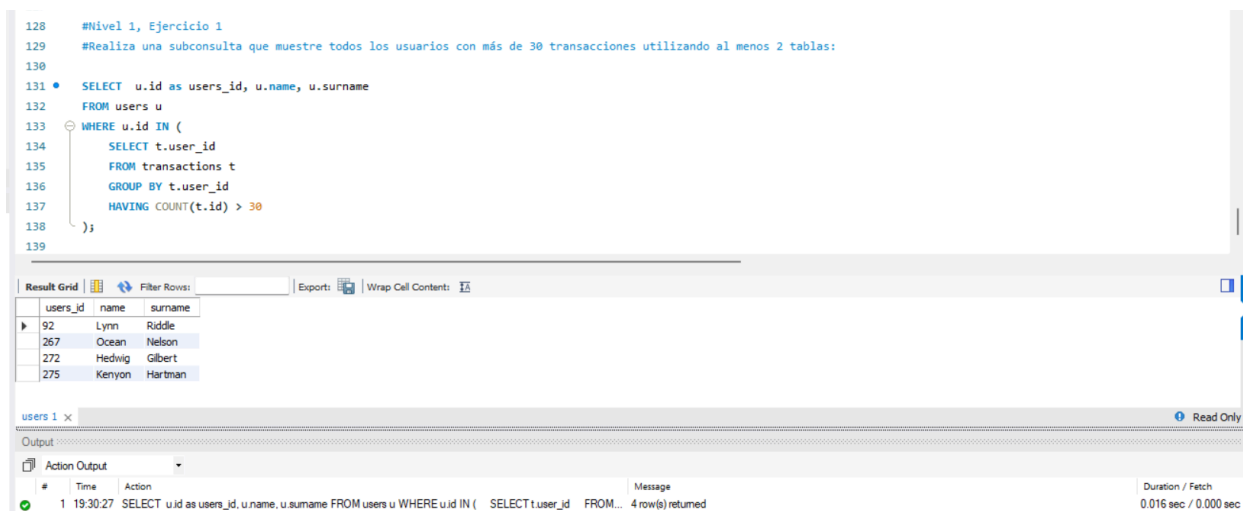


Como puede verse, la tabla *products* no tiene conexión con la tabla *transactions* porque las informaciones de la clave primaria de la tabla *products* (id) están presente en la tabla *transactions*, pero necesitan una tabla puente para crear una conexión ya que la columna *products\_ids* de la tabla *transactions* contiene más de una información de id separada por una coma y esto impide establecer una conexión:

product_ids
59
71, 41
97, 41, 3
11, 13, 61, 29
47, 37, 11, 1
23, 19, 71
59, 13, 23
47, 67, 31, 5
17, 13, 73

## Nivel 1, Ejercicio 1

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



The screenshot shows a SQL IDE interface. The top pane contains a SQL query with line numbers 128 to 139. The query is as follows:

```
128 #Nivel 1, Ejercicio 1
129 #Realiza una subconsulta que muestre todos los usuarios con más de 30 transacciones utilizando al menos 2 tablas:
130
131 • SELECT u.id as users_id, u.name, u.surname
132 FROM users u
133 WHERE u.id IN (
134     SELECT t.user_id
135     FROM transactions t
136     GROUP BY t.user_id
137     HAVING COUNT(t.id) > 30
138 );
139
```

The bottom pane shows the 'Result Grid' with 4 rows of data. The columns are 'users\_id', 'name', and 'surname'.

users_id	name	surname
92	Lynn	Riddle
267	Ocean	Nelson
272	Hedwig	Gilbert
275	Kenyon	Hartman

Below the result grid, there is an 'Output' section showing the execution details of the query. It indicates that the query was executed at 19:30:27, took 0.016 seconds, and returned 4 rows.

Este código realiza una consulta que muestra todos los usuarios que han realizado más de 30 transacciones. Se utilizan las tablas *users* y *transactions*. La consulta principal selecciona los datos de los usuarios, como su ID, nombre y apellido, de la tabla *users*.

Dentro de esta consulta, hay una subconsulta que trabaja con la tabla *transactions*. Esta subconsulta agrupa las transacciones por el ID del usuario (*t.user\_id*) y cuenta cuántas transacciones ha realizado cada usuario. Luego, utiliza la cláusula *HAVING* para filtrar aquellos usuarios que tienen más de 30 transacciones.

Finalmente, la consulta principal utiliza la condición *IN* para seleccionar solo los usuarios cuyo ID aparece en el resultado de la subconsulta, es decir, aquellos que han realizado más de 30 transacciones. De esta manera, se obtiene la lista de usuarios que cumplen con este criterio.

## Nivel 1, Ejercicio 2

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

```
143 • SELECT cc.id as credit_card_id, c.company_name, cc.iban, ROUND(AVG(t.amount), 2) AS avg_amount
144 FROM credit_cards cc
145 JOIN transactions t
146 ON cc.id = t.card_id
147 JOIN companies c
148 ON c.company_id = t.business_id
149 WHERE c.company_name = 'Donec Ltd'
150 GROUP BY cc.iban, cc.id;
```

credit_card_id	company_name	iban	avg_amount
CCU-2973	Donec Ltd	PT87806228135092429456346	203.72

Result 4 ×

Output

#	Time	Action	Message	Duration / Fetch
1	19:38:59	SELECT cc.id as credit_card_id, c.company_name, cc.iban, ROUND(AVG(t.amount), 2) AS avg_amount FR...	1 row(s) returned	0.000 sec / 0.000 sec

Este código realiza una consulta para mostrar la media del monto (amount) de las transacciones por IBAN de las tarjetas de crédito utilizadas en la compañía llamada "Donec Ltd". La consulta utiliza tres tablas: *credit\_cards*, *transactions* y *companies*.

Primero, la consulta selecciona el ID de la tarjeta de crédito (cc.id), el nombre de la compañía (c.company\_name), el IBAN de la tarjeta de crédito (cc.iban), y la media de los montos de las transacciones (ROUND(AVG(t.amount), 2) AS avg\_amount). La función AVG calcula el promedio de los montos, y ROUND redondea el resultado a dos decimales.

Luego, se realiza una combinación (JOIN) entre la tabla *credit\_cards* y *transactions* usando el ID de la tarjeta de crédito (cc.id = t.card\_id). Después, se hace otro JOIN con la tabla *companies* usando el ID de la empresa (c.company\_id = t.business\_id).

La condición WHERE c.company\_name = 'Donec Ltd' filtra los resultados para mostrar solo las transacciones asociadas a la compañía "Donec Ltd".

Finalmente, la consulta agrupa los resultados por el IBAN de la tarjeta de crédito y su ID, para que el promedio del monto se calcule por cada tarjeta. De esta manera, se obtiene la media de los montos de las transacciones realizadas con las tarjetas de crédito asociadas a la compañía "Donec Ltd".

## Nivel 2, Ejercicio 1

- 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: ¿Cuántas tarjetas están activas?

```
155 CREATE TABLE card_activity (  
156     card_id VARCHAR(15) PRIMARY KEY,  
157     declined ENUM('YES', 'NO') NOT NULL,  
158     FOREIGN KEY (card_id) REFERENCES credit_cards(id)  
159 );  
160
```

Output

#	Time	Action	Message	Duration / Fetch
1	19:59:20	CREATE TABLE card_activity ( card_id VARCHAR(15) PRIMARY KEY, declined ENUM('YES', 'NO') NO...	0 row(s) affected	0.062 sec

El código proporcionado crea una nueva tabla llamada *card\_activity* que tiene como objetivo reflejar el estado de las tarjetas de crédito en función de si las últimas tres transacciones fueron declinadas o no. A continuación, explico cada parte del código y cómo cumple con la solicitud del ejercicio.

**CREATE TABLE card\_activity:** este comando crea una nueva tabla llamada *card\_activity*. El nombre refleja que se trata de la actividad o estado de las tarjetas.

**card\_id VARCHAR(15) PRIMARY KEY:** la tabla tiene una columna llamada *card\_id* que almacena el identificador de la tarjeta de crédito. Esta columna es de tipo VARCHAR(15), lo que significa que puede almacenar cadenas de texto de hasta 15 caracteres (suficiente para un número de tarjeta de crédito). Esta columna se establece como clave primaria (PRIMARY KEY), lo que significa que no puede haber registros duplicados de tarjetas en la tabla.

**declined ENUM('YES', 'NO') NOT NULL:** esta columna llamada *declined* almacena el estado de la tarjeta, que se determina en función de si las últimas tres transacciones fueron declinadas o no. El tipo de dato ENUM('YES', 'NO') permite que la columna solo contenga uno de estos dos valores, lo que representa si las transacciones fueron rechazadas (YES) o aceptadas (NO). La restricción NOT NULL asegura que siempre haya un valor en esta columna.

FOREIGN KEY (card\_id) REFERENCES credit\_cards(id): Se establece una clave externa (FOREIGN KEY) en la columna *card\_id*, que referencia el campo *id* de la tabla *credit\_cards*. Esto asegura que solo se puedan insertar en la tabla *card\_activity* tarjetas que existan previamente en la tabla *credit\_cards*, lo que mantiene la integridad referencial entre ambas tablas.

```
161 INSERT INTO card_activity (card_id, declined)
162 SELECT t.card_id, IF(COUNT(*) = 3 AND SUM(t.declined) = 3, 'YES', 'NO') AS declined
163 FROM (SELECT card_id, declined
164       FROM transactions
165       ORDER BY timestamp DESC) t
166 GROUP BY t.card_id;
167
```

card_id	declined
CdJ-2938	NO
CdJ-2945	NO
CdJ-2952	NO
CdJ-2959	NO
CdJ-2966	NO

Result 5 x

Output

#	Time	Action	Message	Duration / Fetch
1	20:10:25	INSERT INTO card_activity (card_id, declined) SELECT t.card_id, IF(COUNT(*) = 3 AND SUM(t.declined) = 3...	275 row(s) affected Records: 275 Duplicates: 0 Warnings: 0	0.032 sec
2	20:11:21	SELECT t.card_id, IF(COUNT(*) = 3 AND SUM(t.declined) = 3, 'YES', 'NO') AS declined FROM (SELECT card...	275 row(s) returned	0.000 sec / 0.000 sec
3	20:13:40	SELECT t.card_id, IF(COUNT(*) = 3 AND SUM(t.declined) = 3, 'YES', 'NO') AS declined FROM (SELECT card...	275 row(s) returned	0.016 sec / 0.000 sec

La consulta SQL tiene como objetivo analizar el comportamiento de las tarjetas de crédito registradas en la tabla "transactions" y determinar si, para cada tarjeta, las últimas tres transacciones fueron rechazadas.

En la subconsulta interna, se seleccionan las columnas *card\_id* y *declined* de la tabla *transactions* y se ordenan las transacciones por la columna *timestamp* en orden descendente ("ORDER BY timestamp DESC"). Esto asegura que las transacciones más recientes sean evaluadas primero.

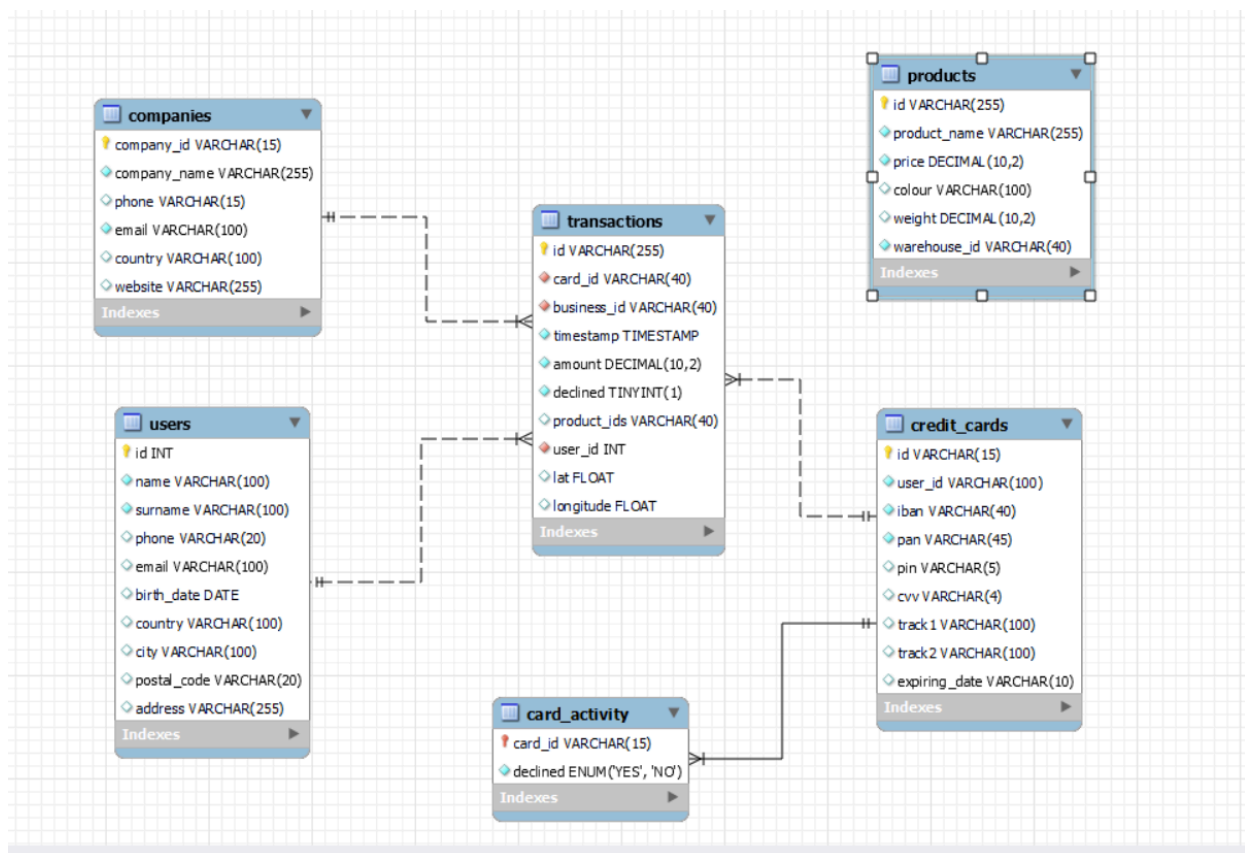
La consulta externa agrupa los datos por *card\_id* ("GROUP BY t.card\_id"), analizando cada tarjeta de manera independiente. Para cada tarjeta, se calculan dos valores:

- La función "COUNT(\*)" cuenta el número total de transacciones evaluadas.
- La función "SUM(t.declined)" suma los valores de la columna "declined", que representa el estado de rechazo de las transacciones (0 para aprobada, 1 para rechazada).

La condición "IF(COUNT(\*) = 3 AND SUM(t.declined) = 3, 'YES', 'NO')" evalúa si existen exactamente tres transacciones recientes ("COUNT(\*) = 3") y si todas ellas fueron rechazadas ("SUM(t.declined) = 3"). Si ambas condiciones se cumplen, se devuelve "YES"; de lo contrario, "NO".

El resultado de esta consulta evidencia que no existen tarjetas con tres transacciones consecutivas rechazadas aplicando los criterios indicados, así que las 275 están activas (no "declined").

Esto es como se muestra el diagrama después de introducir la tabla *card\_activity*:



### Nivel 3, Ejercicio 1

- 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 transactions tenemos product\_ids. Genera la siguiente consulta: necesitamos conocer el número de veces que se ha vendido cada producto.

Actualmente, la columna *product\_ids* en la tabla *transactions* parece violar la regla de normalización (ya que contiene múltiples valores separados por comas). Una solución podría ser crear una tabla de asociación.

Empezamos creando la tabla *sold\_products*:

```
171 CREATE TABLE sold_products (  
172     transaction_id VARCHAR(255),  
173     product_id VARCHAR(255),  
174     PRIMARY KEY (transaction_id, product_id),  
175     FOREIGN KEY (transaction_id) REFERENCES transactions(id),  
176     FOREIGN KEY (product_id) REFERENCES products(id)  
177 );  
178
```

Output

#	Time	Action	Message	Duration / Fetch
1	16:57:11	CREATE TABLE sold_products ( transaction_id VARCHAR(255), product_id VARCHAR(255), PRIMAR	0 row(s) affected	0.203 sec

Esta nueva tabla almacena la relación entre cada transacción y los productos comprados en esa transacción. Esto permite manejar de manera eficiente los datos originalmente almacenados en la columna *product\_ids* de la tabla *transactions*, donde se encontraban como una lista separada por comas.

```
179 INSERT INTO sold_products (transaction_id, product_id)  
180 SELECT t.id AS transaction_id, p.id AS product_id  
181 FROM transactions t  
182 CROSS JOIN products p  
183 WHERE FIND_IN_SET(p.id, REPLACE(t.product_ids, ',', ',')) > 0;
```

Output

#	Time	Action	Message	Duration / Fetch
1	16:36:11	INSERT INTO sold_products (transaction_id, product_id) SELECT t.id AS transaction_id, p.id AS product_id F...	1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0	0.140 sec



transaction_id	product_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
063FBA79-99EC-66FB-29F7-25726D1764A5	67
063FBA79-99EC-66FB-29F7-25726D1764A5	5
063FBA79-99EC-66FB-29F7-25726D1764A5	47
063FBA79-99EC-66FB-29F7-25726D1764A5	31
0668296C-CD89-A883-76BC-3E4C44F8CBAE	89
0668296C-CD89-A883-76BC-3E4C44F8CBAE	83
0668296C-CD89-A883-76BC-3E4C44F8CBAE	79
06CD9AA5-9B42-D684-DDDD-A5E394FEB999	43

Result 7 Read Only

Output

#	Time	Action	Message	Duration / Fetch
1	16:39:30	SELECT t.id AS transaction_id, p.id AS product_id FROM transactions t CROSS JOIN products p WHERE FI...	1457 row(s) returned	0.000 sec / 0.000 sec

Este código tiene como objetivo insertar en la tabla *sold\_products* las combinaciones válidas entre transacciones y productos vendidos. Para lograrlo, utiliza una consulta que compara cada transacción con cada producto mediante un CROSS JOIN, que genera todas las combinaciones posibles entre ambas tablas. A partir de estas combinaciones, el filtro WHERE asegura que sólo se seleccionen aquellas en las que el producto efectivamente está listado en la columna `product\_ids` de la transacción.

La función `FIND\_IN\_SET(p.id, REPLACE(t.product\_ids, ' ', ','))` es clave. Busca si el identificador del producto (`p.id`) se encuentra dentro de la lista de productos asociados a la transacción (`t.product\_ids`). Aquí, el uso de `REPLACE` elimina los espacios innecesarios dentro de las listas para evitar errores en la comparación.

El filtro `> 0` es imprescindible porque la función `FIND\_IN\_SET` devuelve un valor positivo (empezando desde 1) si el producto está presente en la lista de `product\_ids`. Si el producto no está en la lista, la función devuelve `0`. Sin este filtro, se incluirían combinaciones incorrectas, lo que resultaría en datos erróneos en la tabla `sold\_products`. Por lo tanto, el `> 0` asegura que solo se procesen las asociaciones reales entre transacciones y productos vendidos.

Finalmente, cada combinación válida de transacción-producto se inserta en la tabla `sold\_products`, que actúa como un registro detallado de qué productos específicos fueron vendidos en cada transacción.

Ahora solo queda contestar a la pregunta del ejercicio sobre el número de veces que se han vendido los productos. Para hacerlo utilizaremos este código:

```
189 • SELECT p.id AS product_id, p.product_name, COUNT(sp.transaction_id) AS product_sales
190 FROM products p
191 LEFT JOIN sold_products sp
192 ON p.id = sp.product_id
193 LEFT JOIN transactions t
194 ON sp.transaction_id = t.id
195 WHERE t.declined = 0
196 GROUP BY p.id, p.product_name
197 ORDER BY product_sales DESC;
198
199
```

product_id	product_name	product_sales
23	riverlands north	60
67	Winterfell	59
2	Tarly Stark	56
17	skywalker ewok sith	54
43	duel	54

Result 5 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	10:42:16	SELECT p.id AS product_id, p.product_name, COUNT(sp.transaction_id) AS product_sales FROM products p...	26 row(s) returned	0.016 sec / 0.000 sec

Este código SQL tiene como objetivo generar un informe que muestre una lista de productos junto con el número total de ventas asociadas a cada uno, ordenando los resultados en orden descendente según el número de ventas (*product\_sales*). La consulta realiza varias operaciones para unir diferentes tablas y obtener los datos necesarios para el análisis.

Primero, se seleccionan tres columnas principales en el SELECT. El *p.id* representa el identificador único de cada producto, el *p.product\_name* muestra el nombre del producto, y el `COUNT(sp.transaction_id)` calcula el número total de transacciones en las que el producto ha sido vendido, con el alias *product\_sales* para facilitar la interpretación.

La consulta utiliza un LEFT JOIN entre las tablas *products* y *sold\_products*. Este paso asegura que todos los productos de la tabla *products* se incluyan en el resultado, incluso si no han sido vendidos, ya que el LEFT JOIN garantiza que los datos de la tabla *products* prevalezcan, aunque no haya correspondencias en *sold\_products*. El criterio de unión, `p.id = sp.product_id`, establece la relación entre el identificador del producto y las transacciones en las que está incluido.

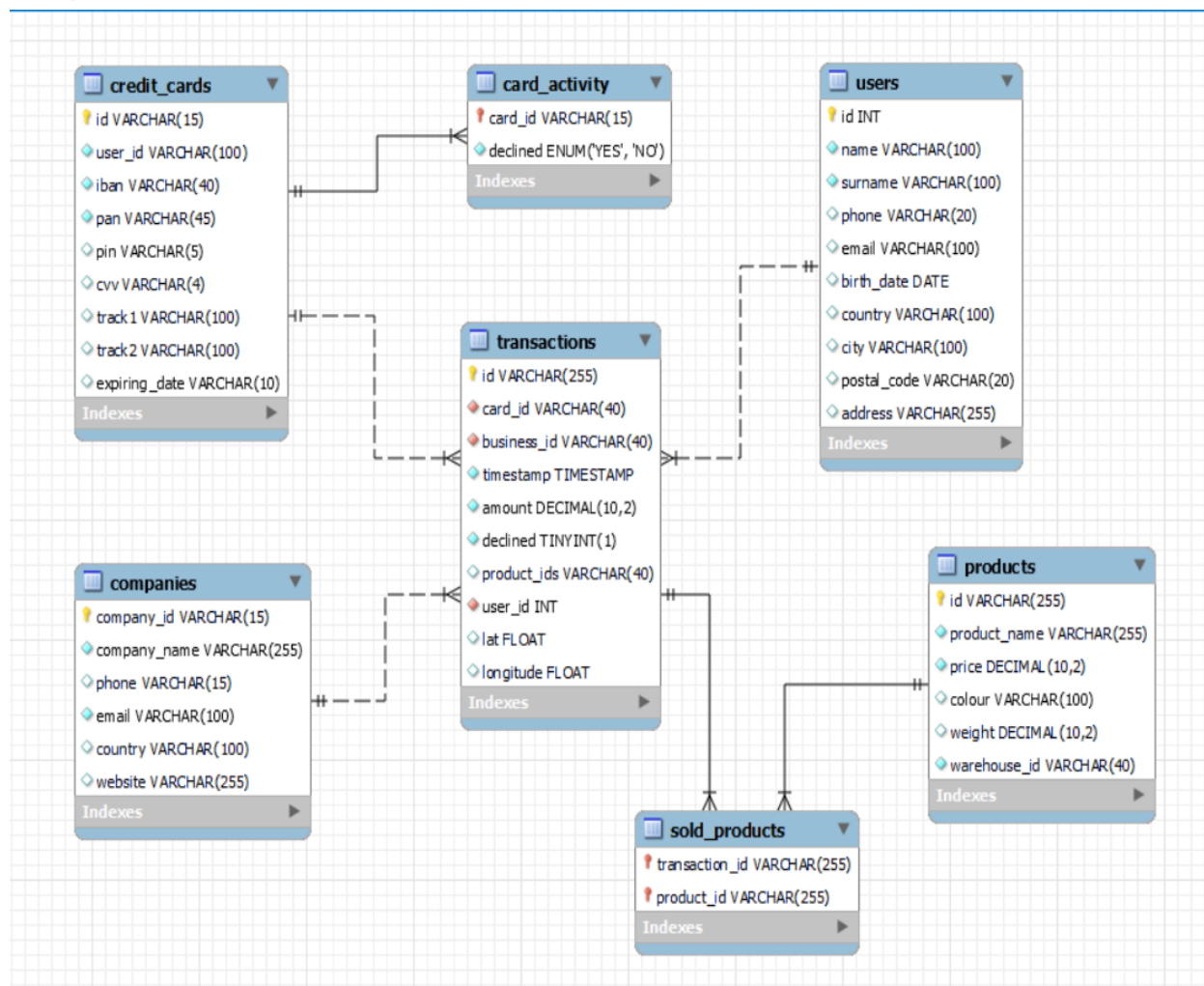
A continuación, se realiza un segundo LEFT JOIN con la tabla *transactions*. Este paso asocia cada transacción a los productos vendidos, pero con una condición adicional.

En el WHERE indicamos que las transacciones consideradas deben tener el

campo *t.declined* igual a 0, lo que indica que la transacción no fue rechazada. Esto permite filtrar únicamente las ventas exitosas.

Finalmente, los resultados se agrupan por el identificador del producto (*p.id*) y su nombre (*p.product\_name*) usando la cláusula GROUP BY. Esto asegura que cada producto aparezca una sola vez en la salida, con su respectivo conteo de ventas calculado a partir del número de transacciones relacionadas. La cláusula ORDER BY *product\_sales* DESC organiza los productos de mayor a menor número de ventas, destacando aquellos con mejor desempeño.

Ahora que todas las tablas han sido creadas, este es como aparece el diagrama completo:



En este esquema, la tabla *transactions* actúa como la tabla de hechos (fact table) y se conecta con varias tablas de dimensiones. Cada transacción está relacionada con un usuario a través de la columna *user\_id* y con una tarjeta de crédito mediante la columna *card\_id*. La tabla *sold\_products* establece una relación muchos-a-muchos entre las transacciones y los productos vendidos, permitiendo rastrear qué productos están asociados a cada transacción. Las empresas (*companies*) se relacionan indirectamente con las transacciones mediante el identificador *business\_id*. Finalmente, la actividad de las tarjetas (*card\_activity*) está vinculada a las tarjetas de crédito (*credit\_cards*) y proporciona información adicional sobre el estado de cada tarjeta.

En cuanto a las relaciones representadas en el diagrama, es importante notar la diferencia entre las líneas continuas y las líneas discontinuas. Las líneas continuas representan relaciones obligatorias, normalmente implementadas mediante claves foráneas (foreign keys) con restricciones de integridad referencial. Esto implica que, para cada registro en la tabla secundaria, debe existir un registro relacionado en la tabla principal. Por ejemplo, en la relación entre *sold\_products* y *transactions*, cada producto vendido en *sold\_products* debe estar asociado a una transacción válida en la tabla *transactions*.

Por el contrario, las líneas discontinuas indican relaciones opcionales o condicionales, lo que significa que no siempre es obligatorio que exista un registro coincidente en ambas tablas. Estas relaciones permiten que ciertos registros de una tabla no estén vinculados a registros específicos en la otra, reflejando así vínculos más flexibles o dependientes de condiciones adicionales.

Esta distinción entre líneas continuas y discontinuas es esencial para interpretar correctamente el modelo relacional, ya que define cómo interactúan los datos entre las tablas y qué reglas deben cumplirse en las relaciones.

