

IT ACADEMY

SPRINT 4: **Creació de Bases de Dades.**

Sergi Alcolea de la Gala







Nivell 1

Descàrrega els arxius CSV, estudia'ls i dissenya una base de dades amb un esquema d'estrella que contingui, almenys 4 taules de les quals puguis realitzar les següents consultes:

Antes de poder realizar las subconsultas, primero debemos crear nuestra base de datos propia des de la cual operar.

→ Nos descargamos los archivos:

▼ hoy (6)

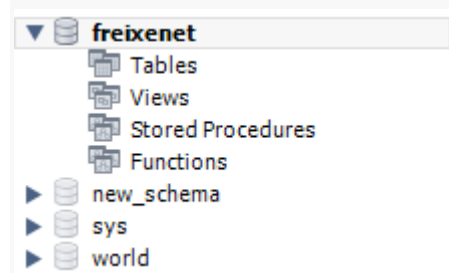
 transactions.csv	13/10/2025 10:34	Full de càlcul de l'...	12.911 KB
 products.csv	13/10/2025 10:34	Full de càlcul de l'...	5 KB
 european_users.csv	13/10/2025 10:34	Full de càlcul de l'...	451 KB
 credit_cards.csv	13/10/2025 10:34	Full de càlcul de l'...	805 KB
 companies.csv	13/10/2025 10:34	Full de càlcul de l'...	11 KB
 american_users.csv	13/10/2025 10:34	Full de càlcul de l'...	118 KB

Procedemos a crear la base de datos a la que denominaremos como 'Freixenet' (porque la lista de productos parecen nombres de licores y otras bebidas alcohólicas.)

```
37 • CREATE SCHEMA IF NOT EXISTS `Freixenet`; -- 1. Crear la base de datos (Schema) usando acentos g
38 • USE `Freixenet`; -- 2. Seleccionar la base de datos para trabajar
39 • SET default_storage_engine = INNODB; -- 3. Establecer el motor de almacenamiento correctamente (
```

Output					
Action Output					
#	Time	Action	Message	Duration / Fetch	
✓ 4	11:34:02	CREATE SCHEMA IF NOT EXISTS 'Freixenet'	1 row(s) affected	0.000 sec	
✓ 5	11:34:02	USE 'Freixenet'	0 row(s) affected	0.000 sec	
✓ 6	11:34:02	SET default_storage_engine = INNODB	0 row(s) affected	0.000 sec	

Vemos que el schema se ha creado correctamente:



Ahora ya podemos empezar a trabajar sobre la base de datos. Sin embargo, todavía es necesario crear las tablas de datos e insertar los datos correspondientes a cada una antes de poder empezar a hacer consultas y subconsultas.

El ejercicio pide que creamos una estructura de estrella en el diagrama de datos. Esto quiere decir que, tras crear nuestra base de datos propia, crearemos tablas a partir de ellas, que se dividan en:

- Fact table: la “hechos” Contendrá los datos cuantitativos y las claves foráneas a las dimensiones.
- Dimension Tables: Necesitamos tablas para responder al quién, qué, cuándo, dónde y cómo de cada transacción.

Así pues, empezaremos creando las siguientes tablas, que serán la base de todo el modelo donde definimos la estructura de estrella que almacenará los datos limpios y organizados:

★ TABLA PRODUCTOS:

En los archivos CSV que contienen los datos fuente, podemos ver los campos y el tipo de registro que se han utilizado.

	A	B	C	D	E	F	G	H	I	J	K
1	id	product_name	price	colour	weight	warehouse_id					
2		1 Direwolf Stannis	\$161.11	#7c7c7c		1 WH-4					
3		2 Tarly Stark	\$9.24	#919191		2 WH-3					
4		3 duel tourney Lannister	\$171.13	#d8d8d8	1.5	WH-2					
5		4 warden south duel	\$71.89	#111111		3 WH-1					

Podemos emplearlos como guía para la correcta definición de los campos de la tabla en MySQL, ya que es esencial garantizar el mismo orden y nombre de los mismos para asegurar una importación de los datos de forma fácil y segura.

```

28 • CREATE TABLE product (
29     id VARCHAR(50) PRIMARY KEY, -- Usamos el ID original del CSV como PK
30     product_name VARCHAR(255) NOT NULL,
31     price_unit DECIMAL(10, 2),
32     colour VARCHAR(50),
33     weight_kg DECIMAL(5, 2),
34     warehouse_id VARCHAR(50) -- Renombrado y usado como ID único
35 );

```

Output

#	Time	Action	Message	Duration / Fetch
✓ 37	12:53:17	SELECT * FROM freixenet.card_status_simple	5000 row(s) returned	0.000 sec / 0.000 sec
✓ 38	12:53:45	SELECT * FROM freixenet.card_status_simple W...	4995 row(s) returned	0.000 sec / 0.015 sec
✓ 39	13:01:06	SELECT product.id, product.product_name, ...	100 row(s) returned	0.219 sec / 0.000 sec
✓ 40	13:08:37	DROP DATABASE 'freixenet'	7 row(s) affected	0.078 sec
✓ 41	13:08:44	CREATE SCHEMA IF NOT EXISTS 'Freixenet'	1 row(s) affected	0.000 sec
✓ 42	13:26:33	CREATE TABLE product (id VARCHAR(50) P...	0 row(s) affected	0.016 sec

Ahora que la estructura de la tabla está creada, ahora simplemente insertamos los datos correspondientes del archivo CSV:

```
39 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/products.csv'
40 INTO TABLE product
41 FIELDS TERMINATED BY ','
42 ENCLOSED BY '"'
43 LINES TERMINATED BY '\n'
44 IGNORE 1 ROWS
45 (id, product_name, @price, colour, weight_kg, warehouse_id)
46 SET price_unit = REPLACE(@price, '$', ''); -- Limpieza de dato en la carga
47
48 • SELECT * FROM product;
```

	id	product_name	price_unit	colour	weight_kg	warehouse_id
▶	1	Direwolf Stannis	161.11	#7c7c7c	1.00	WH-4
	10	Karstark Dorne	119.52	#f4f4f4	2.40	WH--5
	100	south duel	40.43	#6d6d6d	3.00	WH--95
	11	Karstark Dorne	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	147.52	#f4f4f4	2.00	WH--9

product 2 x

Apply Revert

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 40	13:08:37	DROP DATABASE 'freixenet'	7 row(s) affected	0.078 sec
✓ 41	13:08:44	CREATE SCHEMA IF NOT EXISTS 'Freixenet'	1 row(s) affected	0.000 sec
✓ 42	13:26:33	CREATE TABLE product (id VARCHAR(50) P...	0 row(s) affected	0.016 sec
✓ 43	13:30:02	LOAD DATA INFILE 'C:/ProgramData/MySQL/M...	100 row(s) affected Records: 100 Deleted: 0 Ski...	0.000 sec

Hay varias cosas a aclarar en este segundo proceso:

- 1) **LOAD DATA INFILE** marca la dirección para leer el archivo. Si MySQL está configurado para mirar siempre en la carpeta donde se guardan los archivos para trabajar, se puede simplemente poner el nombre del mismo (en este caso. "product.csv". Pero es más seguro copiar y pegar la dirección completa. Lee las columnas y las asigna a la tabla.
- 2) **IGNORE 1 ROWS** salta la primera fila, que es la cabecera (los títulos de las columnas).
- 3) El truco clave está aquí: **SET price_unit = REPLACE(@price, '\$', '')**. La columna de precio en el CSV venía con el símbolo del dólar (ej: \$15.99). Para poder guardarla como un número **DECIMAL**, debemos limpiarla durante la carga. Guardamos el valor original en una variable temporal (**@price**) y luego, con **REPLACE**, le quitamos el símbolo \$ antes de insertarlo en la columna final **price_unit**. Esto es una pequeña transformación (la 'T' de ETL) hecha directamente en la carga.

Procedemos a hacer lo mismo con el resto de tablas:

★ TABLA COMPANYY

```
52 • CREATE TABLE company (  
53     id VARCHAR(50) PRIMARY KEY,  
54     company_name VARCHAR(255) NOT NULL,  
55     country VARCHAR(100),  
56     phone VARCHAR(50),  
57     email VARCHAR(255),  
58     website VARCHAR(255)  
59 );  
60  
61 -- insertamos:  
62 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/companies.csv'  
63 INTO TABLE company  
64 FIELDS TERMINATED BY ','  
65 ENCLOSED BY ''''  
66 LINES TERMINATED BY '\n'  
67 IGNORE 1 ROWS  
68 (id, company_name, phone, email, country, website); -- NOTA CLAVE: La lista de columnas (en pa  
69
```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 44	13:30:22	SELECT * FROM freixenet.product	100 row(s) returned	0.000 sec / 0.000 sec
✓ 45	13:31:31	SELECT * FROM product	100 row(s) returned	0.000 sec / 0.000 sec
✓ 46	13:44:11	CREATE TABLE company (id VARCHAR(50) ...	0 row(s) affected	0.016 sec
✓ 47	13:44:20	LOAD DATA INFILE 'C:/ProgramData/MySQL/M...	100 row(s) affected Records: 100 Deleted: 0 Ski...	0.015 sec

★ TABLA CREDIT_CARD

```

72 • CREATE TABLE credit_card (
73     id VARCHAR(50) PRIMARY KEY NOT NULL,
74     user_id VARCHAR(50),
75     iban VARCHAR(50),
76     pan VARCHAR(50),
77     pin VARCHAR(50),
78     cvv VARCHAR(10),
79     track1 VARCHAR(255),
80     track2 VARCHAR(255),
81     expiring_date VARCHAR(20)
82 );
83
84 -- Insertamos:
85
86 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/credit_cards.csv'
87 INTO TABLE credit_card
88 FIELDS TERMINATED BY ','
89 ENCLOSED BY '"'
90 LINES TERMINATED BY '\n'
91 IGNORE 1 ROWS;
92
93 • SELECT * FROM freixenet.credit_card;

```

id	user_id	iban	pan	pin	cvv	track1
CcS-4857	276	XX4857591835292505850771	2314242385113924	1819	467	%B2314242385113924^LWCBUDLW
CcS-4858	277	XX8581768137002436094025	6582720299715533	3964	817	%B6582720299715533^TIQMVTIQM
CcS-4859	278	XX7826930491423553609370	8861684536289642	4983	277	%B8861684536289642^COFBGDCOF
CcS-4860	279	XX5559590368835304645299	2481155515498459	6876	661	%B2481155515498459^TIUJTUTIUJT
CcS-4861	280	XX2035182877195191627307	1308930301149557	5710	398	%B1308930301149557^HPOBNZHPOE

credit_card 3 x Apply Revert

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 48	13:46:00	CREATE TABLE credit_card (id VARCHAR(50...	0 row(s) affected	0.015 sec
✓ 49	13:46:05	LOAD DATA INFILE 'C:/ProgramData/MySQL/M...	5000 row(s) affected Records: 5000 Deleted: 0 ...	0.125 sec
✓ 50	13:48:13	SELECT * FROM freixenet.credit_card	5000 row(s) returned	0.000 sec / 0.000 sec
✓ 51	13:48:54	SELECT * FROM freixenet.credit_card	5000 row(s) returned	0.000 sec / 0.000 sec

★ TABLA USER:

```

98 • CREATE TABLE user (
99     id INT PRIMARY KEY, -- Usamos el ID original como PK
100     first_name VARCHAR(100) NOT NULL,
101     last_name VARCHAR(100) NOT NULL,
102     phone VARCHAR(50),
103     email VARCHAR(255),
104     birth_date DATE,
105     country VARCHAR(100) NOT NULL,
106     continent VARCHAR(50) NOT NULL,
107     city VARCHAR(100),
108     postal_code VARCHAR(20),
109     address VARCHAR(255)
110 );
111 -- Insertamos datos de users AMERICANOS:
112 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/american_users.csv'
113 INTO TABLE user
114 FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'
115 IGNORE 1 ROWS
116 (id, first_name, last_name, phone, email, @birth_date_raw, country, city, postal_code, address)
117 SET
118     continent = 'America',
119     birth_date = STR_TO_DATE(@birth_date_raw, '%b %d, %Y');
120
121 -- Insertamos users EUROPEOS:
122 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/european_users.csv'
123 INTO TABLE user
124 FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'
125 IGNORE 1 ROWS
126 (id, first_name, last_name, phone, email, @birth_date_raw, country, city, postal_code, address)
127 SET
128     continent = 'Europe',
129     birth_date = STR_TO_DATE(@birth_date_raw, '%b %d, %Y');

```

Result Grid								
	id	first_name	last_name	phone	email	birth_date	country	continent
▶	1	Zeus	Gamble	1-282-581-0551	interdum.enim@protonmail.edu	1985-11-17	United States	America
	2	Garrett	Mcconnell	(718) 257-2412	integer.vitae.nibh@protonmail.org	1992-08-23	United States	America
	3	Ciaran	Harrison	(522) 598-1365	interdum.feugiat@aol.org	1998-04-29	United States	America
	4	Howard	Stafford	1-411-740-3269	ornare.egestas@icloud.edu	1989-02-18	United States	America

user 4 x

Apply

Revert

Output



Action Output

#	Time	Action	Message	Duration / Fetch
✓ 52	13:51:04	CREATE TABLE user (id INT PRIMARY KEY, ...	0 row(s) affected	0.031 sec
✓ 53	13:51:08	LOAD DATA INFILE 'C:/ProgramData/MySQL/M...	1010 row(s) affected Records: 1010 Deleted: 0 ...	0.015 sec
✓ 54	13:51:22	LOAD DATA INFILE 'C:/ProgramData/MySQL/M...	3990 row(s) affected Records: 3990 Deleted: 0 ...	0.188 sec

En cuanto a los users, podríamos hacer dos tablas separadas (users americanos y users europeos). Pero una buena práctica del modelo dimensional es combinar datos semejantes mediante **consolidación** en una misma tabla (siempre y cuando tengan los mismos campos).

Podemos convertir la diferencia entre Europeos y Americanos en un simple atributo de la dimensión (una categoría dicotómica), pero para ello hará falta añadir un campo que no existía: **continente**.

Al igual que con el precio, las fechas venían en formato texto (ej: "Jan 15, 1990"). La función **STR_TO_DATE** convierte este texto en un formato de fecha estándar de MySQL (**DATE**). Esto es crucial para poder hacer cálculos con fechas más adelante (como calcular la edad de un usuario o filtrar por mes de nacimiento).

★ TABLA PRODUCTS:

```
135 • CREATE TABLE IF NOT EXISTS transaction (
136     -- Clave Primaria
137     id VARCHAR(255) PRIMARY KEY,
138
139     -- Claves Foráneas (referencias)
140     card_id VARCHAR(50),          -- Referencia a credit_card.id
141     company_id VARCHAR(50) NOT NULL, -- Referencia a company.id
142
143     -- Datos Transaccionales
144     timestamp DATETIME,          -- Fecha y hora de la transacción
145     amount DECIMAL(10, 2) NOT NULL, -- Monto de la transacción
146     declined BOOLEAN,            -- 0 (No) o 1 (Sí)
147     product_ids VARCHAR(500),    -- Lista de IDs de productos
148     user_id INT NOT NULL,        -- Referencia a user.id
149
150     -- Datos Geográficos
151     lat DECIMAL(11, 8),          -- Latitud
152     longitude DECIMAL(11, 8)    -- Longitud
153 );
154
155 -- Insertamos todos los datos:
156
157 • LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.4/Uploads/transactions.csv'
158 INTO TABLE transaction
159 FIELDS TERMINATED BY ';'
160 ENCLOSED BY '"'
161 LINES TERMINATED BY '\n'
162 IGNORE 1 ROWS;
163
164 • SELECT * FROM freixenet.transaction;
165
```

id	card_id	company_id	timestamp	amount	declined	product_id
00043A49-2949-494B-A5DD-A5BAE3BB19DD	CcS-9294	b-2458	2024-08-28 07:16:46	395.43	0	16, 26, 97,
000447FE-B650-4DCF-85DE-C7ED0EE1CAAD	CcS-5019	b-2370	2016-12-21 20:07:18	155.63	0	66, 69, 87,
00045D6B-ED2E-4F2F-8186-CEE074D875D0	CcS-6699	b-2390	2020-07-14 15:37:45	326.01	0	30, 11, 16,
000481C3-1C26-4FEF-83A0-4CD0EB004BBD	CcS-6696	b-2230	2017-09-04 19:44:53	161.60	0	72

Output				
Action Output				
#	Time	Action	Message	Duration / Fetch
✓ 58	13:53:31	SELECT * FROM freixenet.user	5000 row(s) returned	0.000 sec / 0.015 sec
✓ 59	14:01:26	CREATE TABLE IF NOT EXISTS transaction (...	0 row(s) affected	0.110 sec
⚠ 60	14:01:51	LOAD DATA INFILE 'C:/ProgramData/MySQL/M...		2.313 sec

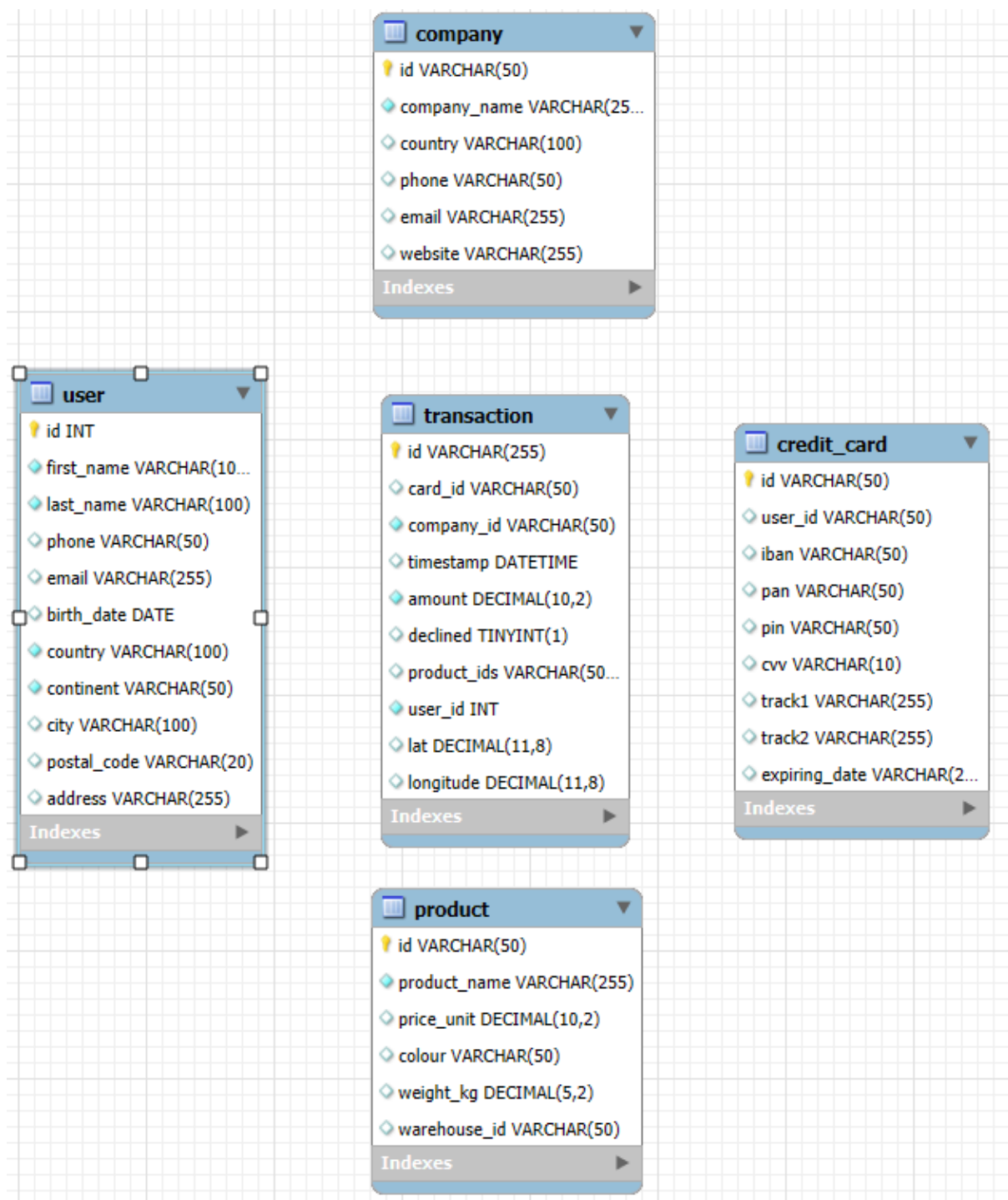
Finalmente crearemos la ÚLTIMA TABLA: transacciones. Esta será la tabla maestra y la más importante: No describe "cosas", sino que registra **eventos**, es decir, las transacciones. Conecta todas las dimensiones que hemos creado antes y, por ello, será el centro de nuestra estrella:

- **id** `VARCHAR(255)` **PRIMARY KEY**: Cada transacción tiene su propio identificador único.
- **Columnas Clave (los conectores)**: Las columnas `card_id`, `company_id`, y `user_id` son las más importantes. De momento, son simples columnas que contienen los IDs de las otras tablas. Más adelante, las convertiremos en **Claves Foráneas (Foreign Keys)** para crear la conexión formal.
- **Métricas**: `amount` (el importe) y `declined` (si fue rechazada) son los valores numéricos que normalmente se analizan. Son el "corazón" de la tabla de hechos. Son los datos que sumaremos, promediaremos, etc.

Una vez creadas todas las tablas, veremos que disponemos de todas ellas en el navegador:

Sprint4		SQL File 5*		credit_card	user	transaction	freixenet x				
Info	Tables	Columns	Indexes	Triggers	Views	Stored Procedures	Functions	Grants	Events		
Name		Engine	Version	Row Format		Rows	Avg Row Length		Data Length		Max Data
company		InnoDB	10	Dynamic		100	163		16.0 KiB		
credit_card		InnoDB	10	Dynamic		4923	322		1.5 MiB		
product		InnoDB	10	Dynamic		100	163		16.0 KiB		
transaction		InnoDB	10	Dynamic		99422	195		18.6 MiB		
user		InnoDB	10	Dynamic		5311	496		2.5 MiB		

Obtendremos el siguiente diagrama:



Efectivamente, tiene forma de estrella, aunque las tablas están dispersas. Hay que conectar las tablas dimensionales (company, credit_card, user y product) a la tabla de hechos (transaction), mediante FOREIGN KEYS:

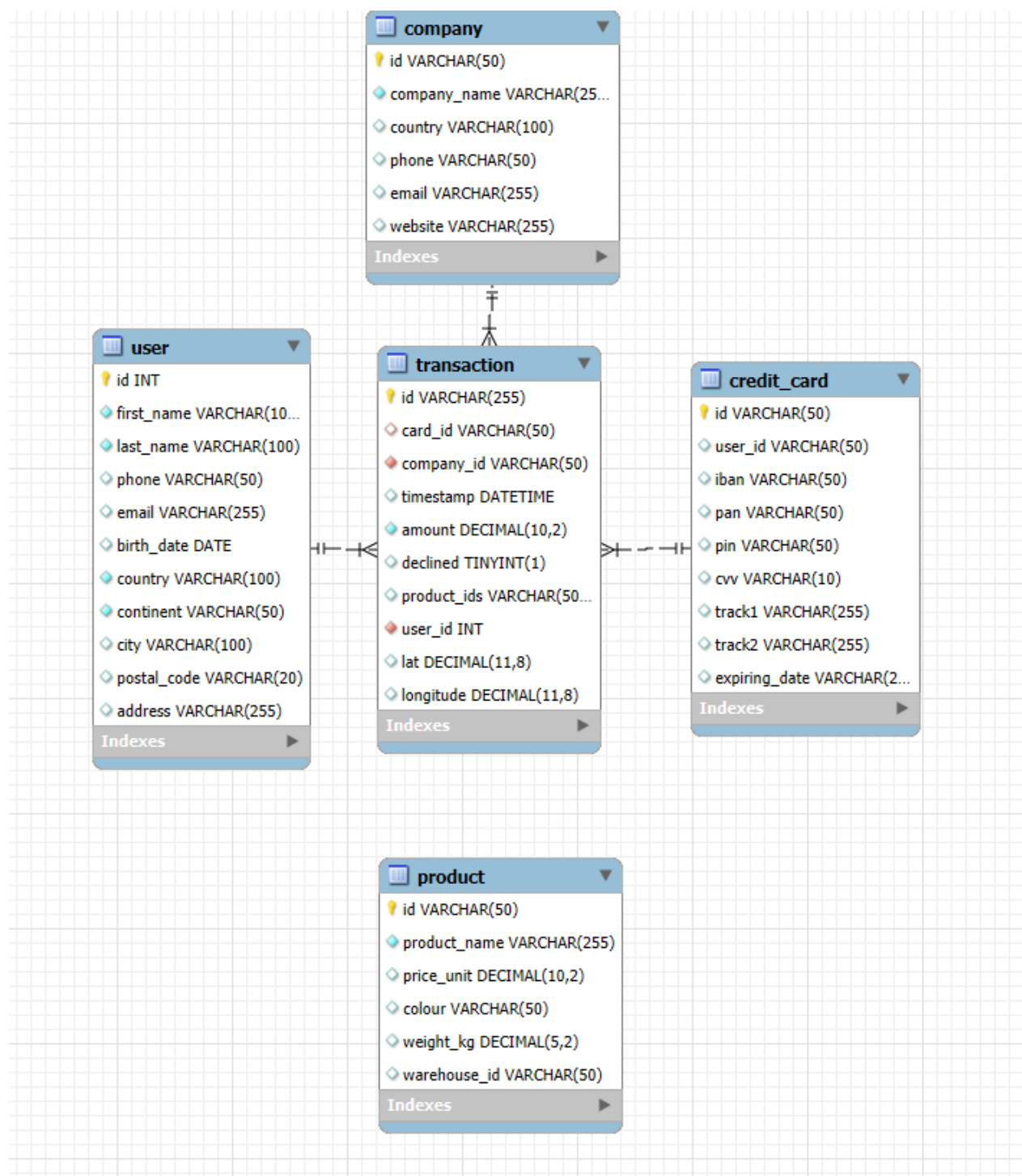
```
170 -- 1. Unir a la dimensión USER
171 • ALTER TABLE transaction
172     ADD CONSTRAINT fk_user
173     FOREIGN KEY (user_id) REFERENCES user(id);
174
175 -- 2. Unir a la dimensión COMPANY
176 • ALTER TABLE transaction
177     ADD CONSTRAINT fk_company
178     FOREIGN KEY (company_id) REFERENCES company(id);
179
180 -- 3. Unir a la dimensión CREDIT_CARD
181 • ALTER TABLE transaction
182     ADD CONSTRAINT fk_card
183     FOREIGN KEY (card_id) REFERENCES credit_card(id); -- Esta FK falló la primera vez porque ]
184
```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 65	14:12:37	ALTER TABLE transaction ADD CONSTRAIN...	100000 row(s) affected Records: 100000 Duplic...	1.312 sec
✓ 66	14:12:40	ALTER TABLE transaction ADD CONSTRAIN...	100000 row(s) affected Records: 100000 Duplic...	1.782 sec
✓ 67	14:12:43	ALTER TABLE transaction ADD CONSTRAIN...	100000 row(s) affected Records: 100000 Duplic...	2.563 sec

Así es como obtendremos nuestro diagrama de estrella:



Por qué la tabla "product" no se encuentra unida a transaction como las demás? Una única transacción puede incluir múltiples productos (por ejemplo, una compra de tres artículos diferentes). La tabla **transaction** almacena todos los IDs de producto en una sola columna de texto (**product_ids**), lo cual es una estructura "plana" que impide usar una **FOREIGN KEY** estándar. Las claves foráneas solo pueden manejar relaciones Uno a Muchos (1:N), donde cada fila de la tabla de hechos apunta a una única fila de la tabla de dimensión. Para modelar correctamente esta relación M:M, se requeriría un paso intermedio posterior para crear una tabla puente (o de ítems de línea), que es la razón por la que se difiere su vinculación. Es algo que dejaremos para más adelante (el nivel 3).

Exercici 1

Realitza una subconsulta que mostri tots els usuaris amb més de 80 transaccions utilitzant almenys 2 taules.

El objetivo es obtener una lista de usuarios. La consulta empieza con **FROM user**, lo que significa que su punto de partida es la tabla de usuarios. La idea es recorrer esta tabla, fila por fila (es decir, usuario por usuario), y decidir si cada uno cumple la condición para aparecer en el resultado final. Para los que sí la cumplan, mostraremos su **ID, nombre y apellido**.

```
11 • SELECT
12     user.id,
13     user.first_name,
14     user.last_name,
15     -- 1. Subconsulta per calcular el recompte (es calcula una vegada per a cada usuari)
16     (
17         SELECT COUNT(transaction.id)
18         FROM transaction
19         WHERE transaction.user_id = user.id -- Correlació per comptar només les transaccions d'a
20     ) AS num_transactions
21 FROM
22     user
23 WHERE
24     -- 2. Subconsulta per aplicar el filtre (el mateix càlcul es repeteix, però el filtre és ràp
25     (
26         SELECT COUNT(transaction.id)
27         FROM transaction
28         WHERE transaction.user_id = user.id
29     ) > 80
30 ORDER BY
31     num_transactions DESC;
```

Result Grid

	id	first_name	last_name	num_transactions
▶	185	Molly	Gilliam	110
	289	Dxwgi	Hwcru	94
	318	Bnyr	Astuw	91
	454	Sfzzoh	Xgvfridxs	81

Result 1 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 1	11:27:50	SELECT user.id, userfirst_name, user.last_...	4 row(s) returned	0.203 sec / 0.000 sec

Esta es una doble subconsulta dentro de otra subconsulta, cuyo truco se encuentra en:

```
(
    SELECT COUNT(transaction.id)
    FROM transaction
    WHERE transaction.user_id = user.id
```

Esto Significa que la subconsulta **depende de la consulta principal**. Fíjate en la condición **WHERE transaction.user_id = user.id**. El valor **user.id** viene de la fila que la consulta principal (**FROM user**) está procesando en ese momento.

Nivell 2

Crea una nova taula que reflecteixi l'estat de les targetes de crèdit basat en *si les tres últimes transaccions han estat declinades aleshores és inactiu, si almenys una no és rebutjada aleshores és actiu*. Partint d'aquesta taula respon:

Exercici 1

Quantes targetes estan actives?

Primero, debemos crear la tabla:

The screenshot shows a database management tool interface. On the left, a tree view displays the database structure, including tables like 'company', 'credit_card', 'product', 'transaction', and 'user'. The main area shows a SQL script for creating a table 'card_status_simple' and inserting data from a subquery. The subquery selects the 'card_id' and the timestamp of the third most recent transaction for each card. The script is as follows:

```
23 • CREATE TABLE card_status_simple AS
24 SELECT
25     transaction_actual.card_id,
26     (
27         SELECT transaction_tercera.timestamp
28         FROM transaction transaction_tercera -- Alias para la subconsulta (antes t3)
29         WHERE transaction_tercera.card_id = transaction_actual.card_id
30         ORDER BY transaction_tercera.timestamp DESC
31         LIMIT 1 OFFSET 2 -- Selecciona la 3ª fila más reciente (offset 2)
32     ) AS third_recent_timestamp -- La subconsulta encuentra la fecha de la 3ª transacción más
33 FROM
34     transaction transaction_actual -- Alias para la consulta principal (antes t)
35 GROUP BY
36     transaction_actual.card_id; -- Agrupamos para obtener una sola fila por cada tarjeta
37
38 • SELECT * FROM card_status_simple
```

Below the script, a 'Result Grid' shows the data for the 'card_status_simple' table:

card_id	third_recent_timestamp
CcS-4857	2024-07-27 10:50:49
CcS-4858	2023-12-10 09:21:55
CcS-4859	2023-03-11 06:20:59
CcS-4860	2023-07-14 06:59:27
CcS-4861	2024-02-05 21:17:58
CcS-4862	2024-01-05 01:22:49

At the bottom, an 'Action Output' table shows the execution results of the SQL script:

#	Time	Action	Message	Duration / Fetch
2	11:56:11	SELECT user.id, user.first_name, user.last...	4 row(s) returned	0.047 sec / 0.000 sec
3	11:56:25	CREATE TABLE card_status_simple AS SELECT...	5000 row(s) affected Records: 5000 Duplicates: ...	0.985 sec
4	11:57:21	SELECT * FROM card_status_simple	5000 row(s) returned	0.000 sec / 0.016 sec

A continuació serà necessari calcular el estat final de cada transacció hecha por cada tarjeta de crédito. Para ello, añadimos una columna a la tabla creada que contenga el número de rechazos **contando a partir de la tercera transacción**:

```

42 • ALTER TABLE card_status_simple
43 ADD COLUMN status_targeta VARCHAR(10); -- Afegim la columna de l'estat

```

card_id	third_recent_timestamp	status_targeta
CcS-4857	2024-07-27 10:50:49	NULL
CcS-4858	2023-12-10 09:21:55	NULL
CcS-4859	2023-03-11 06:20:59	NULL
CcS-4860	2023-07-14 06:59:27	NULL
CcS-4861	2024-02-05 21:17:58	NULL
CcS-4862	2024-01-05 01:22:49	NULL

card_status_simple5 x

Output

#	Time	Action	Message	Duration / Fetch
6	12:03:16	SELECT * FROM card_status_simple	5000 row(s) returned	0.000 sec / 0.000 sec
7	12:31:21	ALTER TABLE card_status_simple ADD COLUMN...	0 row(s) affected Records: 0 Duplicates: 0 Warn...	0.015 sec
8	12:31:29	SELECT * FROM card_status_simple	5000 row(s) returned	0.000 sec / 0.000 sec

Por último, actualizamos el estado de los datos basándonos en las condiciones:

```

231 SET status_targeta = (
232     -- Subconsulta que comprova les condicions
233     SELECT
234         CASE
235             -- CAS 1: Si third_recent_timestamp és NULL (la targeta té < 3 transaccions), és
236             WHEN cs.third_recent_timestamp IS NULL THEN 'Actiu'
237
238             -- CAS 2: Comprovem si alguna transacció (des de la 3a més recent) NO va ser rebutjada
239             WHEN EXISTS (
240                 SELECT 1
241                 FROM transaction t_check
242                 WHERE
243                     t_check.card_id = cs.card_id AND -- Per a la targeta actual
244                     t_check.timestamp >= cs.third_recent_timestamp AND -- Des de la tercera t
245                     t_check.declined = 0 -- ...algun rebuig és 0 (actiu)
246             ) THEN 'Actiu'
247
248             -- CAS 3: Si no es compleix cap de les anteriors, significa que les 3 o més recen
249             ELSE 'Inactiu'
250         END
251 );
252
253 • SELECT * FROM card_status_simple;

```

card_id	third_recent_timestamp	status_targeta
CcS-4857	2024-07-27 10:50:49	Actiu
CcS-4858	2023-12-10 09:21:55	Actiu
CcS-4859	2023-03-11 06:20:59	Actiu
CcS-4860	2023-07-14 06:59:27	Actiu
CcS-4861	2024-02-05 21:17:58	Actiu
CcS-4862	2024-01-05 01:22:49	Actiu

tatus_simple1 x

Output

#	Time	Action	Message	Duration / Fetch
11	12:36:29	UPDATE card_status_simple cs SET status_targ...	5000 row(s) affected Rows matched: 5000 Chan...	0.282 sec
12	12:37:43	SELECT * FROM card_status_simple	5000 row(s) returned	0.000 sec / 0.016 sec
13	12:38:24	SELECT * FROM card_status_simple	5000 row(s) returned	0.000 sec / 0.016 sec

En este segmento analizamos varios aspectos:

El primer bloque de código actualiza la tabla `card_status_simple` para asignar un estado a cada tarjeta.

- **SET SQL_SAFE_UPDATES = 0;**: Esto es como quitar un seguro de MySQL que, por defecto, te impide modificar una tabla entera de golpe. Se necesita para que la siguiente orden funcione.

El **UPDATE** establece el `status_targeta` siguiendo estas **tres reglas en orden**:

1. **¿La tarjeta tiene menos de 3 transacciones?**
 - Si es así (`third_recent_timestamp IS NULL`), se considera **'Actiu'** automáticamente.
 - **Si tiene 3 o más transacciones, ¿alguna de las 3 últimas fue exitosa?**
 - El código revisa las 3 transacciones más recientes. Si encuentra que **al menos una** de ellas *no* fue rechazada (`declined = 0`), la tarjeta se considera **'Actiu'**.
2. **Si no se cumplen las dos reglas anteriores...**
 - Esto solo puede significar una cosa: la tarjeta tiene 3 o más transacciones y **todas las 3 más recientes fueron rechazadas**. En este caso, la tarjeta se considera **'Inactiu'**.

Después, hemos contado las tarjetas activas (`SELECT COUNT`). Es mucho más simple, porque una vez que todas las tarjetas han sido etiquetadas, esta consulta hace lo siguiente.

1. Va a la tabla `card_status_simple`.
2. Filtra para quedarse solo con las filas donde `status_targeta` es igual a **'Actiu'**.
3. **Cuenta cuántas filas** cumplen esa condición y muestra el resultado.

The screenshot shows a SQL IDE interface. The top pane contains a SQL query:

```
255 -- C) Quantes targetes estan actives?
256 • SELECT
257     COUNT(css.card_id) AS total_targetes_actives
258 FROM
259     card_status_simple css
260 WHERE
261     css.status_targeta = 'Actiu';
262
```

The bottom pane shows the 'Result Grid' with one row of data:

	total_targetes_actives
▶	4995

Below the result grid, there is an 'Output' section with a tab for 'Action Output'.

#	Time	Action	Message	Duration / Fetch
✓ 12	12:37:43	SELECT * FROM card_status_simple	5000 row(s) returned	0.000 sec / 0.016 sec
✓ 13	12:38:24	SELECT * FROM card_status_simple	5000 row(s) returned	0.000 sec / 0.016 sec
✓ 14	12:58:06	SELECT COUNT(css.card_id) AS total_targete...	1 row(s) returned	0.016 sec / 0.000 sec

Al inicio del Sprint, cuando intentábamos armar la base de datos, me recomendaron que dejara la tabla de **product** para el final. Dicha tabla no se puede vincular directamente a la tabla de hechos (“transaction”) mediante una Foreign Key, porque hay que resolver una relación de Muchos a Muchos (M:M). Al fin y al cabo, cada producto tiene su propia ID, pero además, cada transacción puede involucrar varias unidades de un producto que también tendrán su propia ID.

Por ello, es necesario tener una tabla puente que haga de intermediario entre ambas tablas:

The screenshot shows a database management interface for a project named 'freixenet'. On the left, a tree view shows the database structure, including tables like 'card_status_simple', 'company', 'credit_card', 'product', 'product_transaction', 'transaction', and 'user'. The 'product_transaction' table is expanded, showing its columns: 'product_id' and 'transaction_id'. Below the tree, the SQL editor shows the following code:

```
265 • CREATE TABLE product_transaction (
266     product_id VARCHAR(50) NOT NULL,
267     transaction_id VARCHAR(255) NOT NULL,
268     PRIMARY KEY (product_id, transaction_id)
269 );
270
271 • Select * from product_transaction;
```

Below the SQL editor, the 'Result Grid' shows a single row with 'product_id' and 'transaction_id' both set to 'NULL'. Below that, the 'Output' pane shows the 'Action Output' table:

#	Time	Action	Message
✓ 14	12:58:06	SELECT COUNT(css.card_id) AS total_targete...	1 row(s) returned
✓ 15	13:15:08	CREATE TABLE product_transaction (product...	0 row(s) affected
✓ 16	13:15:39	Select * from product_transaction	0 row(s) returned

Una vez creada, añadimos las FOREIGN KEYS:

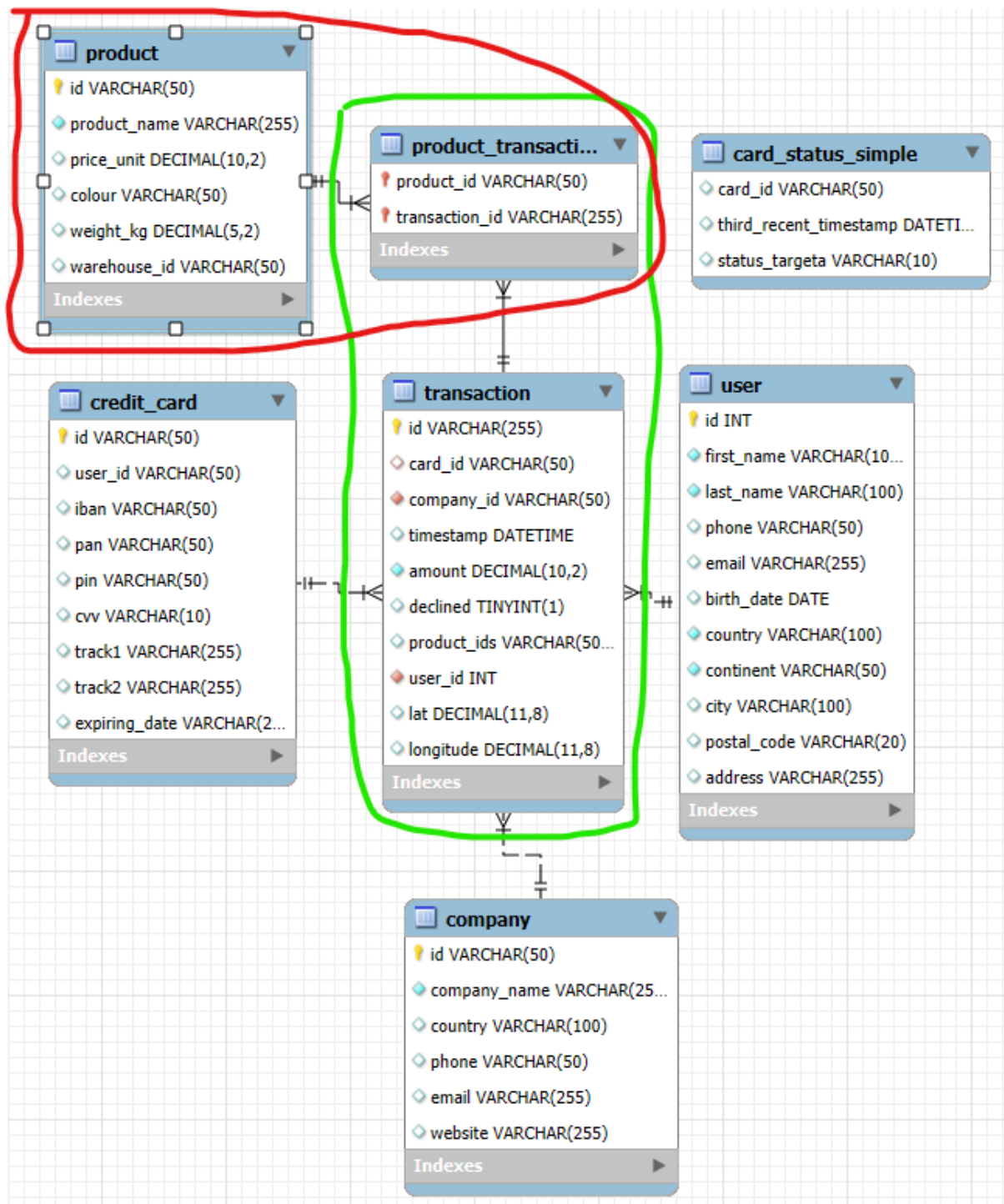
The screenshot shows the SQL editor with the following code:

```
273 -- 1. Afegim la clau forana a la taula de fets (Transaction)
274 • ALTER TABLE product_transaction
275     ADD CONSTRAINT fk_product_transaction
276     FOREIGN KEY (transaction_id) REFERENCES transaction(id);
277
278 -- 2. Afegir la clau forana a la taula de dimensions (Product)
279 • ALTER TABLE product_transaction
280     ADD CONSTRAINT fk_pt_product
281     FOREIGN KEY (product_id) REFERENCES product(id);
282
```

Below the SQL editor, the 'Output' pane shows the 'Action Output' table:

#	Time	Action	Message	Duration / Fetch
✓ 16	13:15:39	Select * from product_transaction	0 row(s) returned	0.000 sec / 0.000 sec
✓ 17	13:21:43	ALTER TABLE product_transaction ADD CON...	0 row(s) affected Records: 0 Duplicates: 0 Wam...	0.047 sec
✓ 18	13:21:53	ALTER TABLE product_transaction ADD CON...	0 row(s) affected Records: 0 Duplicates: 0 Wam...	0.047 sec

Nos debería quedar, entonces el siguiente diagrama:



Ahora llega la parte más complicada del ejercicio. La tabla `transaction` tiene los IDs de producto en una sola celda de texto (ej: "`prod1, prod2, prod3`"). Necesitamos "desempaquetar" esa lista y crear una fila en nuestra tabla puente por cada producto.

```

288 • INSERT INTO product_transaction (transaction_id, product_id)
289 SELECT transaction.id AS transaction_id,
290 TRIM(SUBSTRING_INDEX(SUBSTRING_INDEX(transaction.product_ids, ',', n.n), ',', -1)) AS product_id
291 FROM transaction
292 JOIN
293 (SELECT 1 n UNION ALL SELECT 2 UNION ALL SELECT 3 UNION ALL SELECT 4 UNION ALL SELECT 5) n
294 ON CHAR_LENGTH(transaction.product_ids) - CHAR_LENGTH(REPLACE(transaction.product_ids, ',', '')) >= n.n - 1
295 WHERE TRIM(SUBSTRING_INDEX(SUBSTRING_INDEX(transaction.product_ids, ',', n.n), ',', -1)) IS NOT NULL
296 AND TRIM(SUBSTRING_INDEX(SUBSTRING_INDEX(transaction.product_ids, ',', n.n), ',', -1)) != '';
297
298 • SELECT * FROM product_transaction;

```

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

Fetch rows:

	product_id	transaction_id
▶	16	00043A49-2949-494B-A5DD-A5BAE3BB19DD
	26	00043A49-2949-494B-A5DD-A5BAE3BB19DD
	87	00043A49-2949-494B-A5DD-A5BAE3BB19DD
	97	00043A49-2949-494B-A5DD-A5BAE3BB19DD
	66	000447FE-B650-4DCF-85DE-C7ED0EE1CAAD
	69	000447FE-B650-4DCF-85DE-C7ED0EE1CAAD

product_transaction 4

Apply

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 18	13:21:53	ALTER TABLE product_transaction ADD CONSTRAINT fk_pt_product FOREIGN KEY (product_id) REFERENCES product(id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.047 sec
✗ 19	13:29:44	INSERT INTO product_transaction (transaction_id, product_id) SELECT transaction.id AS transaction_id, TRIM(SUBSTRING_INDEX(SUBSTR...	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use ...	0.000 sec
✓ 20	13:35:46	INSERT INTO product_transaction (transaction_id, product_id) SELECT transaction.id AS transaction_id, TRIM(SUBSTRING_INDEX(SUBSTR...	253391 row(s) affected Records: 253391 Duplicates: 0 Warnings: 0	5.984 sec

Paso a paso, esto es lo que hace:

1. **La "Tabla de Números" (JOIN ... n):** Crea una tabla temporal con números del 1 al 5. Su único propósito es **multiplicar las filas** de la tabla `transaction`.
2. **La Condición de Multiplicación (ON ...):** Esta es la clave. Cuenta cuántas comas hay en el campo `product_ids`. Si una transacción tiene 3 productos (y por tanto 2 comas), esta condición hace que esa fila se una con los números 1, 2 y 3 de la tabla temporal. El resultado es que **la fila de la transacción se triplica**.
3. **La Extracción del ID (SUBSTRING_INDEX):** Ahora que tenemos tres copias de la fila de la transacción (una con el número 1, otra con el 2 y otra con el 3), esta fórmula extrae el **primer, segundo y tercer ID de producto** de la lista `product_ids` respectivamente.
4. **Los Filtros de Seguridad (WHERE):** Simplemente se asegura de que no se inserten filas vacías o nulas si la lista de productos tiene algún formato extraño (como una coma al final).

Entonces, como resultado final, Si teníamos una transacción `t1` con productos "`p1`, `p2`, `p3`", este `INSERT` genera automáticamente tres filas nuevas y las inserta en `product_transaction`.

Por último, consultamos rápidamente cuántas veces se ha vendido cada producto y comprobamos:

```
301 • SELECT
302     product.id,
303     product.product_name,
304     COUNT(product_transaction.product_id) AS Nombre_de_vendes
305 FROM product_transaction
306 JOIN product
307     ON product_transaction.product_id = product.id
308 GROUP BY product.id, product.product_name
309 ORDER BY Nombre_de_vendes DESC, product.product_name ASC;
```

Result Grid

	id	product_name	Nombre_de_vendes
▶	52	riverlands the duel	2654
	29	Tully maester Tarly	2635
	21	duel Direwolf	2609
	16	the duel warden	2608
	66	mustafar jinn	2601
	87	sith Jade	2598

Result 5 ×

Output

#	Time	Action	Message	Duration / Fetch
✗ 21	13:36:14	SELECT * FROM product_transaction - EXERCI...	Error Code: 1064. You have an error in your SQL ...	0.000 sec
✓ 22	13:36:22	SELECT * FROM product_transaction	253391 row(s) returned	0.000 sec / 0.078 sec
✓ 23	13:43:24	SELECT product.id, product.product_name, ...	100 row(s) returned	0.234 sec / 0.000 sec