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

Nivel 1:

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

- Empezamos por crear la base de datos:

```
2  CREATE DATABASE IF NOT EXISTS transacciones;
```

Output

Action Output

#	Time	Action	Message
1	21:38:17	CREATE DATABASE IF NOT EXISTS transacciones	1 row(s) affected

- Luego creamos las tablas 'products', 'companies', 'users' y 'credit_card'.

```
8  USE transacciones;
9  CREATE TABLE IF NOT EXISTS products (
10     id VARCHAR(20) PRIMARY KEY,
11     product_name VARCHAR(100),
12     price VARCHAR(20),
13     colour VARCHAR(20),
14     weight VARCHAR(20),
15     warehouse_id VARCHAR(20)
16 );
```

Output

Action Output

#	Time	Action	Message
1	21:40:05	USE transacciones	0 row(s) affected
2	21:40:05	CREATE TABLE IF NOT EXISTS products (id VAR...	0 row(s) affected

```
19 CREATE TABLE IF NOT EXISTS companies (
20     company_id VARCHAR(20) PRIMARY KEY,
21     company_name VARCHAR(100),
22     phone VARCHAR(15),
23     email VARCHAR(100),
24     country VARCHAR(100),
25     website VARCHAR(100) NULL
26 );
```

Output

Action Output

#	Time	Action	Message
1	21:43:29	CREATE TABLE IF NOT EXISTS companies (com...	0 row(s) affected

```
29 CREATE TABLE IF NOT EXISTS users (
30     id VARCHAR(100) PRIMARY KEY,
31     name VARCHAR(100),
32     surname VARCHAR(100),
33     phone VARCHAR(30),
34     email VARCHAR(100),
35     birth_date VARCHAR(15),
36     country VARCHAR(100),
37     city VARCHAR(100),
38     postal_code VARCHAR(15),
39     address VARCHAR(150)
40 );
```

Output

Action Output

#	Time	Action	Message
1	21:44:28	CREATE TABLE IF NOT EXISTS users (id VARCH...	0 row(s) affected

```sql
CREATE TABLE IF NOT EXISTS credit_card (
    id VARCHAR(20) PRIMARY KEY,
    user_id VARCHAR(100),
    iban VARCHAR(50) NULL,
    pan VARCHAR(50),
    pin VARCHAR(4),
    cvv VARCHAR(4),
    track1 VARCHAR(100) NULL,
    track2 VARCHAR(100) NULL,
    expiring_date VARCHAR(14)
);
```

Output

Action Output

#	Time	Action	Message
1	12:55:11	CREATE TABLE IF NOT EXISTS credit_card (id VARCHAR(20) PRIMARY KEY, user_id VARCHAR(100), ...	0 row(s) affected

- Creamos la tabla transactions a la cual le añadimos todas las relaciones con las otras tablas a través de la FK:

```sql
CREATE TABLE IF NOT EXISTS transactions(
    id VARCHAR(100) PRIMARY KEY,
    card_id VARCHAR(20),
    business_id VARCHAR(20),
    timestamp VARCHAR(50),
    amount DECIMAL(10, 2),
    declined BOOLEAN,
    product_ids VARCHAR(20),
    user_id VARCHAR(100) REFERENCES user(id),
    lat VARCHAR(100),
    longitude VARCHAR(100),
    CONSTRAINT fk_credit_card FOREIGN KEY (card_id) REFERENCES credit_card(id),
    CONSTRAINT fk_company FOREIGN KEY (business_id) REFERENCES companies(company_id),
    CONSTRAINT fk_users FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Output

Action Output

#	Time	Action	Message
1	21:53:34	CREATE TABLE IF NOT EXISTS transactions(id V...	0 row(s) affected

Es importante señalar que, como se observará más adelante en el esquema presentado, aún no se han establecido las relaciones entre la tabla `products` y las demás tablas de nuestro modelo. Esta decisión se ha tomado así, dado que los datos de la columna `product_ids` de la tabla `transactions` están almacenados en forma de una lista acumulada dentro de una misma columna, en lugar de estar normalizados en filas individuales.

Hasta el momento, no se ha realizado ningún tipo de manipulación o transformación sobre estos datos, ya que su tratamiento forma parte de una de las tareas a abordar posteriormente en este sprint. En dicha tarea, se buscará descomponer esta información para poder establecer correctamente las conexiones necesarias con la tabla `products`, y así optimizar la estructura de la base de datos y mejorar su funcionalidad y eficiencia.

- Ahora pasamos a **cargar los datos** proporcionados en cada tabla. Empezaremos con la tabla 'users' en la cual cargaremos los datos tanto de los los users de UK, USA y Canadá:

```
USE transacciones;
LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.4\\Uploads\\users_uk.csv" INTO TABLE users
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES;

USE transacciones;
LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.4\\Uploads\\users_usa.csv" INTO TABLE users
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES;

USE transacciones;
LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.4\\Uploads\\users_ca.csv"INTO TABLE users
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES;
```

Output

Action Output

#	Time	Action	Message
1	22:11:07	USE transacciones	0 row(s) affected
2	22:11:07	LOAD DATA INFILE "C:\\ProgramData\\MySQL\\...	50 row(s) affected Records: 50 Deleted: 0 Skipped: 0 Warnings: 0
3	22:11:07	USE transacciones	0 row(s) affected
4	22:11:07	LOAD DATA INFILE "C:\\ProgramData\\MySQL\\...	150 row(s) affected Records: 150 Deleted: 0 Skipped: 0 Warnings: 0
5	22:11:07	USE transacciones	0 row(s) affected
6	22:11:07	LOAD DATA INFILE "C:\\ProgramData\\MySQL\\...	75 row(s) affected Records: 75 Deleted: 0 Skipped: 0 Warnings: 0

- Continuamos cargando datos, ahora de las tablas 'companies' y 'credit_card', haciendo lo mismo que para la anterior tabla:

```
USE transacciones;
LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.4\\Uploads\\companies.csv" INTO TABLE companies
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES;
```

Output

Action Output

#	Time	Action	Message
1	22:12:56	USE transacciones	0 row(s) affected
2	22:12:56	LOAD DATA INFILE "C:\\ProgramData\\MySQL\\...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0

```
USE transacciones;
LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.4\\Uploads\\credit_cards.csv" INTO TABLE credit_card
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES;
```

Output

Action Output

#	Time	Action	Message
1	22:15:06	USE transacciones	0 row(s) affected
2	22:15:06	LOAD DATA INFILE "C:\\ProgramData\\MySQL\\...	275 row(s) affected Records: 275 Deleted: 0 Skipped: 0 Warnings: 0

- A diferencia de las anteriores tablas y en vista de que la tabla 'products' no contiene ningún dato encerrado entre comillas, no indicamos ENCLOSED BY ni LINES TERMINATED BY:

```
USE transacciones;
LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.4\\Uploads\\products.csv" INTO TABLE products
  FIELDS TERMINATED BY ','
  IGNORE 1 LINES;
```

Output

Action Output

#	Time	Action	Message
1	22:20:48	USE transacciones	0 row(s) affected
2	22:20:48	LOAD DATA INFILE "C:\\ProgramData\\MySQL\\...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0

- Vemos que los valores de la columna 'price' de la tabla 'products' tienen un signo "$" antes del monto, por lo cual lo eliminamos y además convertimos el tipo de datos a decimal para posteriormente poder hacer cálculos con ellos.

```
119 •  UPDATE products SET price=REPLACE(price,'$','');
120 •  ALTER TABLE products
121    MODIFY COLUMN price DECIMAL(9,2),
122    MODIFY COLUMN weight DECIMAL(4,2);
```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
22	11:27:44	UPDATE products SET price=REPLACE(price,'$','')	100 row(s) affected Rows matched: 100 Changed: 100 Warnings: 0	0.016 sec
23	11:27:44	ALTER TABLE products MODIFY COLUMN price DECIMAL(9,2), MODI...	100 row(s) affected Records: 100 Duplicates: 0 Warnings: 0	0.094 sec

- Finalmente, encontramos que los datos de la tabla 'transactions' están separados por punto y coma y no por comas, además, los valores de algunas columnas están encerrados entre comillas simples y no entre comillas dobles, con fue el caso de las anteriores tablas, por lo tanto hacemos estos dos pequeños cambios:

```
120 •  USE transacciones;
121 •  LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.4\\Uploads\\transactions.csv" INTO TABLE transactions
122      FIELDS TERMINATED BY ';' ENCLOSED BY '"'
123      IGNORE 1 LINES;
```

Output

Action Output

#	Time	Action	Message
1	22:23:48	USE transacciones	0 row(s) affected
2	22:23:48	LOAD DATA INFILE "C:\\ProgramData\\MySQL\\...	587 row(s) affected Records: 587 Deleted: 0 Skipped: 0 Warnings: 0

- Ahora cambiamos los datos de la columna timestamp que estaban en VARCHAR a TIMESTAMP

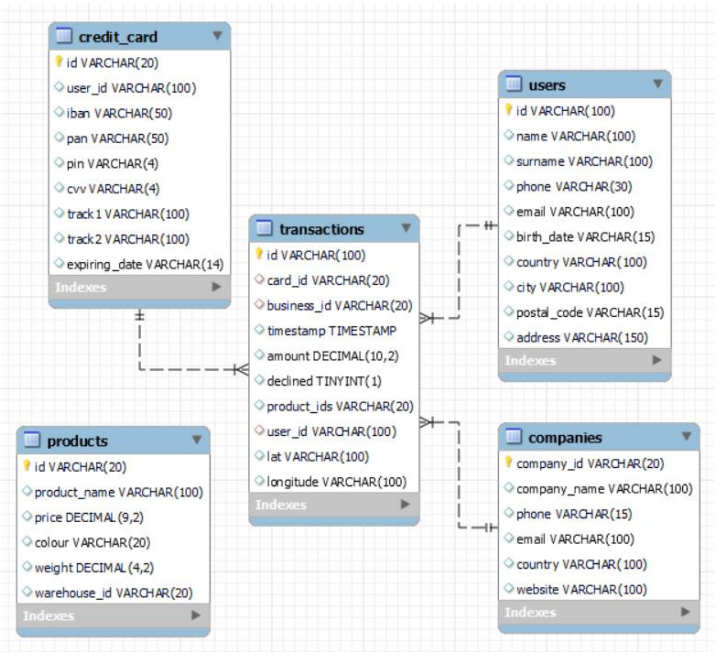
```
132 •  ALTER TABLE transactions CHANGE timestamp timestamp TIMESTAMP;
```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	11:32:30	ALTER TABLE transactions CHANGE timestamp timestamp TIMESTAMP	587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0	0.203 sec

Después de crear nuestra base de datos, las tablas y cargar los datos, nuestro esquema quedaría de la siguiente manera:



Ejercicio 1

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

```
133 •  SELECT *
134    FROM users
135    WHERE users.id IN (SELECT t.user_id
136                       FROM transactions t
137                       GROUP BY t.user_id
138                       HAVING COUNT(t.id) > 30);
139
140
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	name	surname	phone	email	birth_date	country	city	postal_code	address
267	Ocean	Nelson	079-481-2745	aenean@yahoo.com	Dec 26, 1991	Canada	Charlottetown	85X 3P4	Ap #732-8357 Pede, Rd.
272	Hedwig	Gilbert	064-204-8788	sem.eget@icloud.edu	Apr 16, 1991	Canada	Tuktoyaktuk	Q4C 3G7	P.O. Box 496, 5145 Sapien Road
275	Kenyon	Hartman	082-871-7248	convallis.ante.lectus@yahoo.com	Aug 3, 1982	Canada	Richmond	R8H 2K2	8564 Facilisi. St.
92	Lynn	Riddle	1-387-885-4057	vitae.aliquet@outlook.edu	Sep 21, 1984	United States	Bozeman	61871	P.O. Box 712, 7907 Est St.

users 20 ×

Output

Action Output

#	Time	Action	Message
1	16:09:21	SELECT * FROM users WHERE users.id IN (SELECT t.user_id FROM transactions t GROUP...	4 row(s) returned

- He seleccionado todos los datos de la tabla 'users'.
- Hice la subconsulta en el WHERE para que me filtre por los 'users' que tengan más de 30 transacciones en la tabla 'transactions'.

Ejercicio 2

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

Hice una consulta mediante un JOIN entre las tablas 'companies', 'credit_card', 'transactions', tomando de la tabla 'company' el nombre de la misma, de 'credit_card' el 'iban' y el 'card_id' y el cálculo del 'amount' de la tabla de 'transactions'. Luego en el WHERE filtré por la compañía indicada y también filtré por transacciones no declinadas, ya que desde mi perspectiva no tendría sentido incluir en la media el monto de las transacciones que no han sido efectivas. Finalmente agrupamos por 'card_id'.

```
163 •  SELECT com.company_name, cc.iban, t.card_id, ROUND(AVG(t.amount),2) AS Monto
164    FROM credit_card cc
165    JOIN transactions t
166    ON t.card_id = cc.id
167    JOIN companies com
168    ON t.business_id = com.company_id
169    WHERE com.company_name = 'Donec Ltd' and t.declined = 0
170    GROUP BY t.card_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

company_name	iban	card_id	Monto
Donec Ltd	PT87806228135092429456346	CcU-2973	42.82

Result 7 ×

Output

Action Output

#	Time	Action	Message
1	12:40:59	SELECT com.company_name, cc.iban, t.card_id, ROUND(AVG(t.amount),2) AS Monto FROM credit_card...	1 row(s) returned
2	12:41:18	SELECT com.company_name, cc.iban, t.card_id, ROUND(AVG(t.amount),2) AS Monto FROM credit_card...	1 row(s) returned

Nivel 2:

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:

Primero creamos la tabla, agrego el id las tarjetas y una columna para insertar en ella la clasificación de “activas” e “inactivas”.

```
159 • CREATE TABLE cc_status(
160       card_id VARCHAR(20),
161       estado VARCHAR(20)
162   );
```

```
Output
Action Output
#   Time      Action
1   16:12:16  CREATE TABLE cc_status( card_id VARCHAR(20), estado VARCHAR(20) )
```

Luego, para insertar los datos en la tabla, utilicé un CASE para establecer la condición que muestra el estado de las tarjetas en una columna calculada. Usé SUM(declined) >= 3 para marcar las tarjetas como 'INACTIVA', ya que solo se sumarían los valores de 1 correspondientes a transacciones declinadas. Para las demás tarjetas, es decir, aquellas con menos de 3 transacciones declinadas, añadí un ELSE que las marca como 'ACTIVA'.

A continuación, en el FROM, hice una subconsulta en la que, por un lado, realicé un JOIN para traer las columnas card_id y declined de la tabla transactions, y el id de la tabla credit_card. Por otro lado, utilicé ROW_NUMBER para numerar las filas, y con PARTITION BY agrupé el conteo de filas por card_id y por declined. De esta forma, se genera un conteo independiente para cada tarjeta según el tipo de valor en la columna declined. Después, lo ordené por timestamp DESC para que el conteo comience desde la fecha más reciente.

Posteriormente, añadí una cláusula WHERE para filtrar las filas cuyo resultado de ROW_NUMBER sea menor o igual a tres, de manera que solo se tomen en cuenta las tres últimas transacciones de cada tarjeta. Finalmente, agrupé los resultados por card_id.

```
164 •   INSERT INTO cc_status(card_id, estado)
165     SELECT card_id,
166         CASE
167             WHEN SUM(declined) >= 3 THEN 'INACTIVA'
168             ELSE 'ACTIVA'
169         END AS estado
170     FROM (SELECT cc.id AS card_id, t.declined,
171           ROW_NUMBER() OVER (PARTITION BY cc.id, t.declined
172           ORDER BY timestamp DESC) AS NumTipo
173           FROM transactions t
174           JOIN credit_card cc
175           ON t.card_id = cc.id) AS conteo
176     WHERE NumTipo <= 3
177     GROUP BY card_id;
```

```
Output
Action Output
#   Time      Action                                                  Message
1   20:13:03  INSERT INTO cc_status(card_id, estado) SELECT card_id,  CASE  WHE...  275 row(s) affected Records: 275
```

- Le asigno a la columna card_id la condición de UNIQUE y además agrego card_id como fk también de cc_status:

```
181 •  ALTER TABLE cc_status
182    ADD UNIQUE (card_id);
183
184 •  ALTER TABLE transactions
185    ADD CONSTRAINT fk_card_id
186    FOREIGN KEY (card_id) REFERENCES cc_status(card_id);
```

Output

Action Output

#	Time	Action	Message
2	16:14:43	ALTER TABLE cc_status ADD UNIQUE (card_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
3	16:14:43	ALTER TABLE transactions ADD CONSTRAINT fk_card_id FOREIGN KEY (card_id) REFERENCES cc...	587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0

Visualizamos la tabla:

```
186 •  SELECT *
187    FROM cc_status;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

card_id	estado
CcU-2938	ACTIVA
CcU-2945	ACTIVA
CcU-2952	ACTIVA
CcU-2959	ACTIVA
CcU-2966	ACTIVA
CcU-2973	ACTIVA
CcU-2980	ACTIVA
CcU-2987	ACTIVA
CcU-2994	ACTIVA
CcU-3001	ACTIVA

cc_status 25

Output

Action Output

#	Time	Action	Message
1	20:14:28	SELECT * FROM cc_status LIMIT 0, 1000	275 row(s) returned

Veo que todas las tarjetas me dan "ACTIVA", por lo cual paso a comprobar mediante una consulta de la tabla transactions, contado cantidad de registros por card_id donde declined = 1 (operación declinada) sea igual o mayor a 3, y verificamos que ninguna tarjeta tiene 3 o más transacciones declinadas.

```
183 •  SELECT card_id, COUNT(*)
184    FROM transactions
185    WHERE (DECLINED = 1) >= 3
186    GROUP BY card_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

card_id	COUNT(*)

Result 20

Output

Action Output

#	Time	Action	Message
1	12:46:58	SELECT card_id, COUNT(*) FROM transactions WHERE (DECLINED = 1) >= 3 GROUP BY card_id LIMI...	0 row(s) returned

Ahora sigo comprobando, pero ahora consultamos todas las tarjetas que tienen al menos una transacción = 1.

Verifico de forma más manual que todas las tarjetas que tiene declined = 1 solo tienen una transacción declinada, por lo cual concluyo que la tabla de cc_status ha sido creada correctamente.

```
191 •   SELECT card_id, COUNT(*)
192     FROM transactions
193     WHERE declined = 1
194     GROUP BY card_id;
195
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

card_id	COUNT(*)
CcU-3512	1
CcU-3519	1
CcU-3526	1
CcU-3533	1
CcU-3540	1

Result 35

Output

Action Output

#	Time	Action	Message
1	13:20:41	SELECT card_id, COUNT(*) FROM transactions WHERE declined = 1 GROUP BY card_id LIMIT 0, 1000	87 row(s) returned

Ejercicio 1

¿Cuántas tarjetas están activas?

```
198 •   SELECT COUNT(card_id)
199     FROM cc_status
200     WHERE estado = 'Activa';
201
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

COUNT(card_id)
275

Result 34

Output

Action Output

#	Time	Action	Message
1	13:19:40	SELECT COUNT(card_id) FROM cc_status WHERE estado = 'Activa' LIMIT 0, 1000	1 row(s) returned

Tal y como hemos visto en la creación de la tabla, todas las tarjetas están “Activas”, por lo tanto el COUNT no da 275 que es el número total de tarjetas en la base de datos.

Nivel 3:

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

- Primero creamos la tabla intermedia para poder conectar las tablas de trasactions y products

```
226 • CREATE TABLE tabla_conexion (
227     id VARCHAR(100),
228     product_ids VARCHAR(20)
229     );
```

```
Output
Action Output
#  Time      Action                                                                                  Message
1  22:46:18  CREATE TABLE tabla_conexion ( id VARCHAR(100), product_ids VARCHAR(20) )                0 row(s) affected
```

- Ahora desglosamos los datos y los ingresamos así a la tabla intermedia.

- Empezamos usando un SUBSTRING_INDEX anidado para extraer los datos uno a uno.

- Aplicamos un TRIM para eliminar los espacios en blanco que quedan al extraer los datos.

- Con el JOIN y los UNION se crean las filas en donde insertaremos los datos descompuestos.

- Finalmente, en el ON, hacemos una resta entre el número total de caracteres y el número de caracteres sin coma en cada una de nuestras celdas. Esto nos permite calcular el número de comas que hay en cada celda. Se compara con el valor n de esa celda para poder determinar cuántas veces debemos descomponer cada celda.

```
221 •  INSERT INTO tabla_conexion (id, product_ids)
222    SELECT id, TRIM(SUBSTRING_INDEX(SUBSTRING_INDEX(t.product_ids, ',', numpartes.n), ',', -1))
223    FROM transactions t
224    JOIN (SELECT 1 n UNION SELECT 2 UNION SELECT 3 UNION SELECT 4 UNION SELECT 5) AS numpartes
225    ON CHAR_LENGTH(t.product_ids) - CHAR_LENGTH(REPLACE(t.product_ids, ',', '')) >= numpartes.n - 1;
```

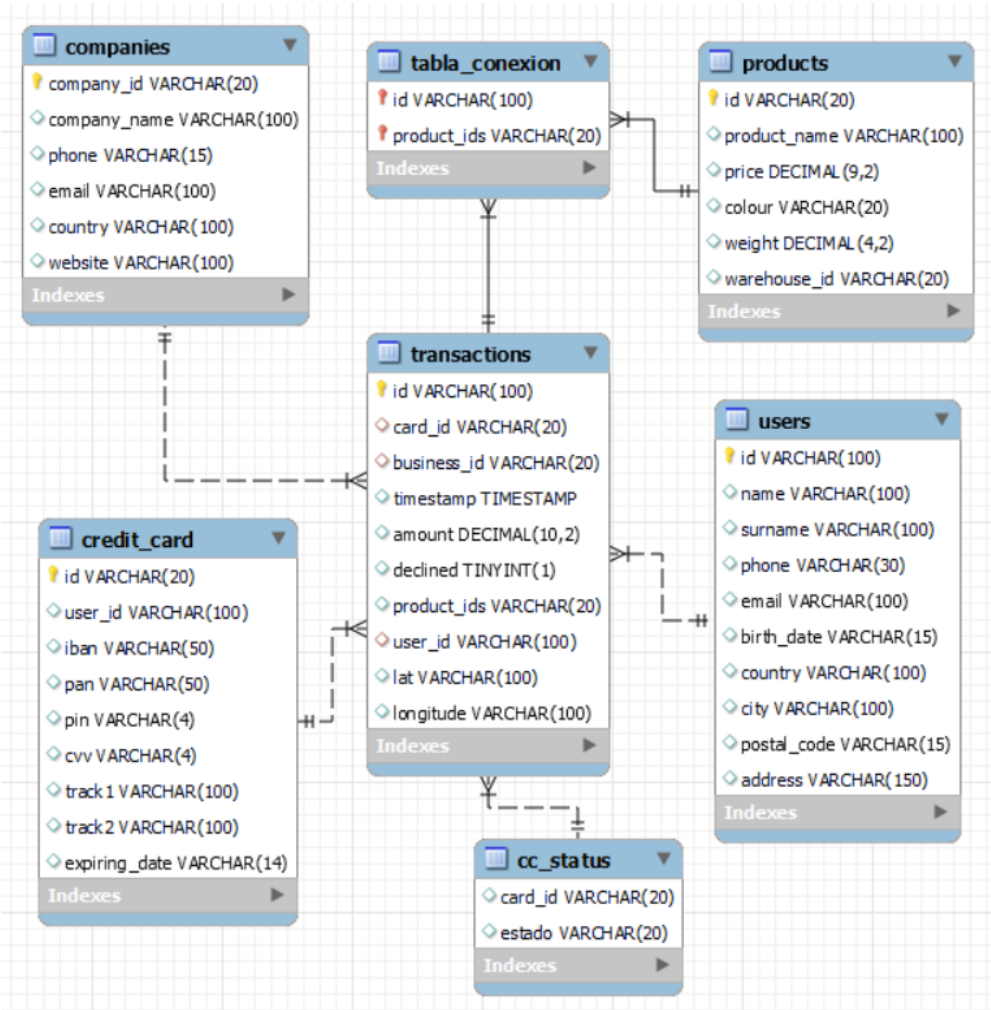
```
Output
Action Output
#  Time      Action                                                                                  Message
1  20:31:43  INSERT INTO tabla_conexion (id, product_ids)  SELECT id, TRIM(SUBST...  1457 row(s) affected Records: 1457
```

Ahora agregamos la PK y las FK

```
246 •  ALTER TABLE tabla_conexion
247    ADD PRIMARY KEY (id, product_ids),
248    ADD FOREIGN KEY (id) REFERENCES transactions(id),
249    ADD FOREIGN KEY (product_ids) REFERENCES products(id);
```

```
Output
Action Output
#  Time      Action                                                                                  Message
1  23:05:45  ALTER TABLE tabla_conexion ADD PRIMARY KEY (id, product_ids), ADD FOREIGN KEY (id) REFERENCES transactions(id), ADD FOREIGN KE...  1457 row(s) affected Records: 1457  Duplicates: 0  Warnings: 0
```

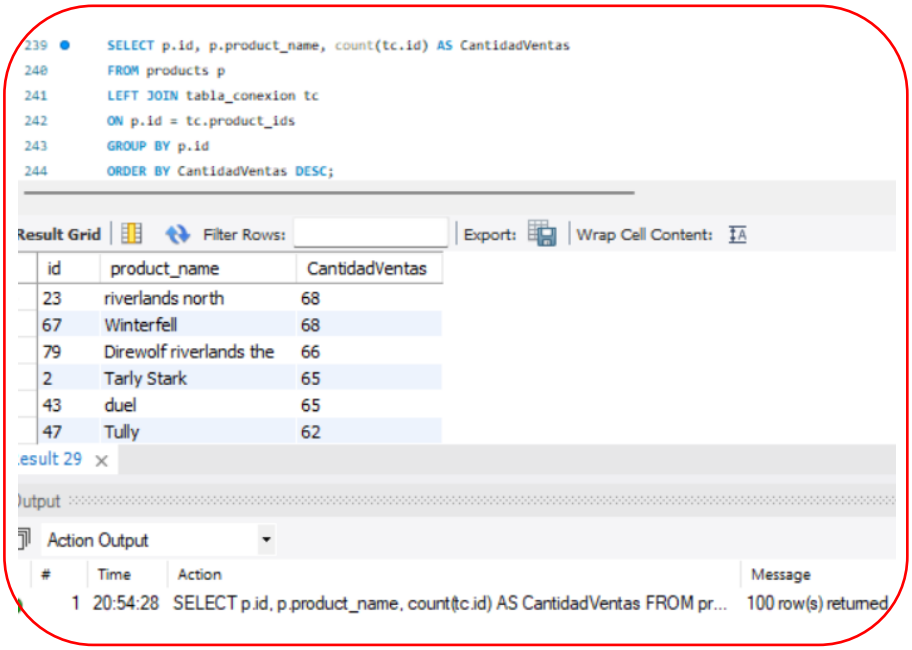
Así quedaría nuestro esquema:



Ejercicio 1

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

- Seleccionamos los datos que se quieren mostrar, haciendo un COUNT en la columna 'id' de la 'tabla_conexion'.
- Hacemos un JOIN entre la tabla products y tabla_conexion.
- Agrupamos por el 'id' de producto y, finalmente, ordenamos de forma descendente.



```
239 ●   SELECT p.id, p.product_name, count(tc.id) AS CantidadVentas
240     FROM products p
241     LEFT JOIN tabla_conexion tc
242     ON p.id = tc.product_ids
243     GROUP BY p.id
244     ORDER BY CantidadVentas DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	product_name	CantidadVentas
23	riverlands north	68
67	Winterfell	68
79	Direwolf riverlands the	66
2	Tarly Stark	65
43	duel	65
47	Tully	62

esult 29 ×

utput

Action Output

#	Time	Action	Message
1	20:54:28	SELECT p.id, p.product_name, count(tc.id) AS CantidadVentas FROM pr...	100 row(s) returned