

# Tarea S4.01. Creación de Bases de Datos

## Nivel 1

**Descarga los archivos CSV, estúdialos 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:**

A continuación mostramos la base de datos con las tablas creadas y el diagrama con esquema de estrella. Para crear la base de datos simplemente usamos CREATE DATABASE con el nombre de la base de datos y luego escribimos USE y el mismo nombre de la base de datos con la que vamos a operar.

```
#NIVEL 1
#Crear Base de datos
CREATE DATABASE transactions_db;
USE transactions_db;
```

Para crear las tablas siempre empezaremos usando la cláusula CREATE TABLE IF NOT EXISTS y el nombre de la tabla. Empezamos con las tablas que contengan solamente claves primarias. La primera será users. Escribimos los nombres de las variables y le asignamos un tipo de dato. Para datos de carácter alfanumérico que pueden variar en longitud o espacio, usamos VARCHAR( ) y asignamos un número que representa este espacio o longitud que ocupan los datos. Asegurémonos que el número es suficientemente largo ya que si el número de caracteres de una fila supera esta cifra, nos dará error. Si queremos evitar ocupar mucho espacio, también podemos ajustar a una cifra que sabemos no se superará en ninguna fila. La primera variable suele ser una clave primaria id. Si sabemos que hay variables que todas las filas tienen la misma longitud, usamos CHAR(longitud de caracteres). Nos aseguramos siempre de acabar cada tabla con “);”.

```

6      # Tabla users
7 • ○ CREATE TABLE IF NOT EXISTS users (
8          id VARCHAR(3) PRIMARY KEY,
9          name VARCHAR(50),
10         surname VARCHAR(20),
11         phone VARCHAR(20),
12         email VARCHAR(50),
13         birth_date VARCHAR(20),
14         country VARCHAR(20),
15         city VARCHAR(30),
16         postal_code VARCHAR(25),
17         address VARCHAR(50)
18     );

```

La siguiente tabla, `credit_cards`, contiene una clave foránea al final. La variable `user_id` hacer referencia a la `id` de la tabla `users`, de modo que usamos `FOREIGN KEY (user_id) REFERENCES users(id)` para que queden relacionadas ambas tablas.

```

20     # Tabla credit_cards
21 • ○ CREATE TABLE IF NOT EXISTS credit_cards (
22         id CHAR(8) PRIMARY KEY,
23         user_id CHAR(3),
24         iban VARCHAR(34),
25         pan VARCHAR(19),
26         pin CHAR(4),
27         cvv CHAR(3),
28         track1 VARCHAR(50),
29         track2 VARCHAR(50),
30         expiring_date VARCHAR(10),
31         FOREIGN KEY (user_id) REFERENCES users(id)
32     );

```

---

```

34     # Tabla companies
35 • ○ CREATE TABLE IF NOT EXISTS companies (
36         company_id CHAR(6) PRIMARY KEY,
37         company_name VARCHAR(35),
38         phone VARCHAR(30),
39         email VARCHAR(40),
40         country VARCHAR(30),
41         website VARCHAR(50)
42     );

```

En la tabla transactions, incluimos 3 claves foráneas, las variables id de cada una de las 3 tablas creadas anteriormente. Así todas estarán relacionadas y formarán el esquema de estrella requerido. Veremos que algunas variables contienen tipos de dato distintos como timestamp para representar fechas y horas, la variable declined será booleana con 1s y 0s para representar si la transacción está declinada o no y variables de coordenadas como lat y longitud con float o amount con DECIMAL para representar valores con decimales.

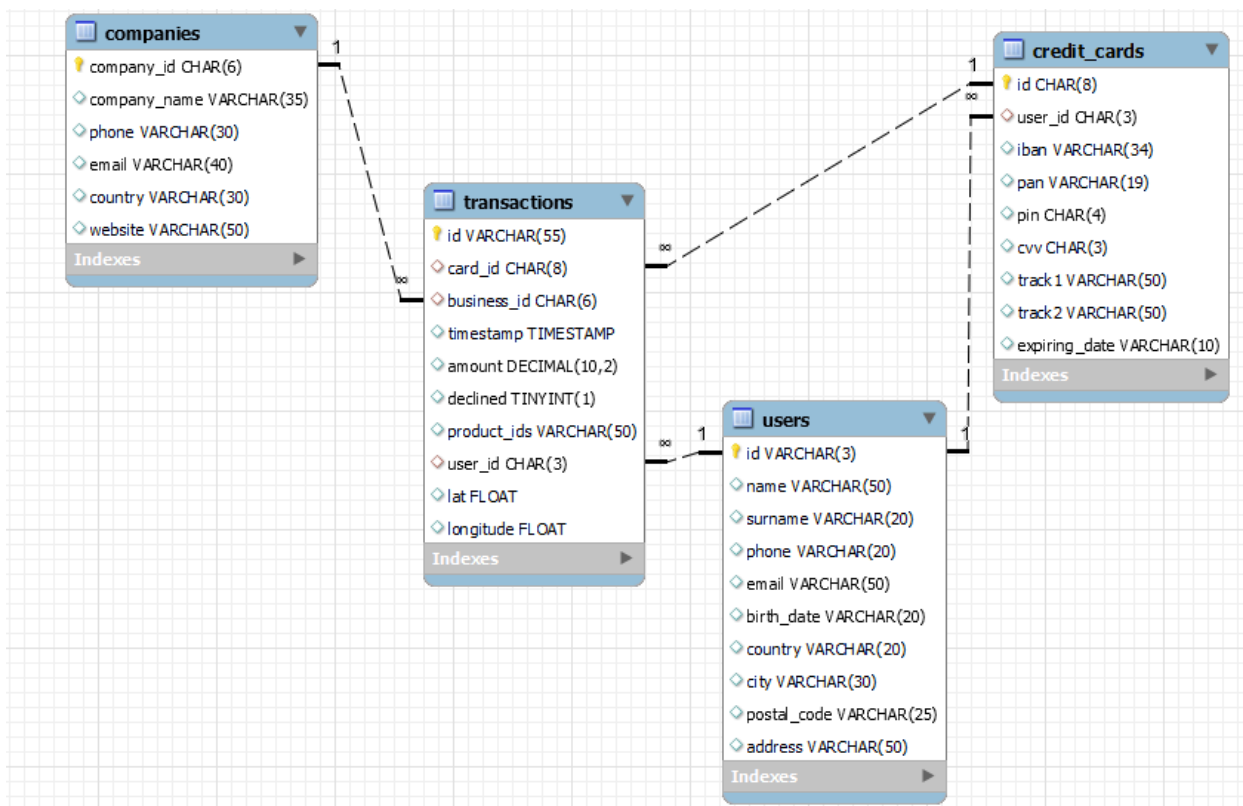
```

44     # Tabla transactions
45 • ○ CREATE TABLE IF NOT EXISTS transactions (
46         id VARCHAR(55) PRIMARY KEY,
47         card_id CHAR(8),
48         business_id CHAR(6),
49         timestamp TIMESTAMP,
50         amount DECIMAL(10, 2),
51         declined BOOLEAN,
52         product_ids VARCHAR(50),
53         user_id CHAR(3),
54         lat FLOAT,
55         longitude FLOAT,
56         FOREIGN KEY (card_id) REFERENCES credit_cards(id),
57         FOREIGN KEY (business_id) REFERENCES companies(company_id),
58         FOREIGN KEY (user_id) REFERENCES users(id)
59     );

```

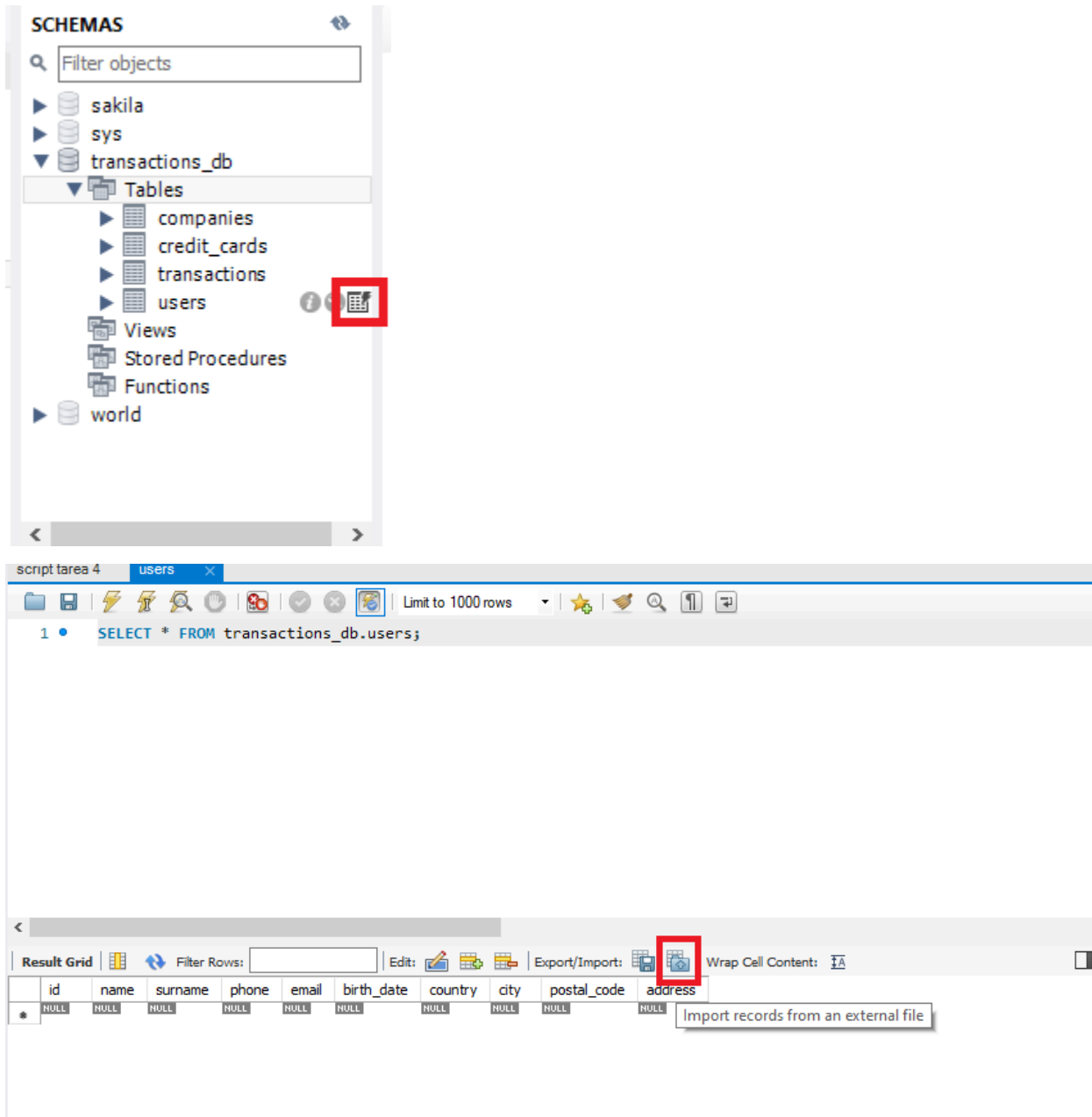
Output			
Action Output			
#	Time	Action	Message
✓ 1	12:43:03	CREATE DATABASE transactions_db	1 row(s) affected
✓ 2	12:43:06	USE transactions_db	0 row(s) affected
✓ 3	12:43:15	CREATE TABLE IF NOT EXISTS users ( id VARCHAR(3) PRIMARY KEY, name VARCH...	0 row(s) affected
✓ 4	12:43:28	CREATE TABLE IF NOT EXISTS credit_cards ( id CHAR(8) PRIMARY KEY, user_id CHA...	0 row(s) affected
✓ 5	12:43:33	CREATE TABLE IF NOT EXISTS companies ( company_id CHAR(6) PRIMARY KEY, co...	0 row(s) affected
✓ 6	12:43:37	CREATE TABLE IF NOT EXISTS transactions ( id VARCHAR(55) PRIMARY KEY, card_i...	0 row(s) affected

Una vez ejecutamos cada tabla y nos aseguramos que están correctamente creadas, vamos a database, reverse engineer y hacemos click en next, seleccionamos la base de datos transactions\_db y seguimos haciendo click en next/finish hasta que nos aparezca el diagrama. Veremos que las relaciones son de N:1 entre transactions y el resto de tablas. Lo que significa que un usuario, una tarjeta de crédito o una compañía pueden tener varias transacciones. Entre users y credit\_cards también hay una relación de 1:N, un usuario puede tener varias tarjetas de crédito.



Una vez tenemos la base de datos, las tablas y el esquema, importamos los datos de los archivos csv disponibles en el enunciado. Si hacemos click en la tabla que aparece a la derecha de cada

tabla creada en los Schemas con las distintas bases de datos y click en import records from an external file, podemos coger la ubicación del archivo csv con los datos e importarlos en la tabla correspondiente. Hay que prestar atención en la parte donde seleccionamos la tabla donde se introducirán los datos, si nos equivocamos de nombre de tabla, se importarán en el lugar equivocado.



Aquí vemos cómo se han importado los valores para la tabla users. Al tener 3 archivos csv, comprobamos con excel que el número total de filas que contiene la tabla en Workbench es el mismo que el total de filas de los 3 archivos csv de 'users\_usa', 'users\_uk' y 'users\_ca'.

The screenshot shows the MySQL Workbench interface. At the top, a script editor window titled 'script tarea 4' contains the SQL query: `SELECT * FROM transactions_db.users;`. Below the script editor, the 'Result Grid' displays the query results. The results are shown in a table with the following columns: id, name, surname, phone, email, birth\_date, country, city, postal\_code, and address. The table contains 10 rows of data. Below the result grid, the 'Output' tab is selected, showing the execution log. The log shows three steps: 1. 'PREPARE stmt FROM 'INSERT INTO 'transactions\_db'.users' ('id','name','surname','phon...' OK. 2. 'DEALLOCATE PREPARE stmt' OK. 3. 'SELECT \* FROM transactions\_db.users LIMIT 0, 1000' 275 row(s) returned.

	id	name	surname	phone	email	birth_date	country	city	postal_code	address
1	1	Zeus	Gamble	1-282-581-0551	interdum.enim@protonmail.edu	Nov 17, 1985	United States	Lowell	73544	348-7818 Sagitt
10	10	Robert	Mccarthy	(324) 746-6771	fermentum@protonmail.com	Apr 30, 1984	United States	Eugene	85526	P.O. Box 773, 3
100	100	Melodie	Mclean	1-677-221-7152	risus.varius@google.ca	Sep 15, 1989	United States	College	11838	Ap #644-8492 S
101	101	Sarah	Beck	(358) 691-4345	vitae.risus@aol.couk	Apr 9, 1983	United States	Great Falls	67129	665-9047 In, R
102	102	Jasper	Landry	1-397-765-1118	consectetuer.euismod@aol.org	Apr 16, 1982	United States	Columbus	11595	Ap #374-7325 S
103	103	Upton	Chavez	(227) 785-6484	euismod.est@aol.ca	Mar 15, 1986	United States	Essex	95631	1990 Vel, Av.
104	104	Martha	Barlow	(732) 326-5448	vulputate@hotmail.net	Oct 29, 1988	United States	Chicago	41512	Ap #311-7103 I
105	105	Hashim	Rose	(858) 313-6727	urna@idoud.com	Mar 28, 1983	United States	Tacoma	99632	8034 Tortor, Ro
106	106	Tanner	Valenzuela	1-346-421-3135	nascetur.ridiculus@google.net	Apr 6, 1993	United States	Naperville	31130	Ap #114-2616 f

Hacemos exactamente lo mismo para el resto de tablas con sus respectivos datos importados de los archivos csv, proporcionados por el enunciado.

## Ejercicio 1

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

Para realizar dicha consulta, seleccionamos el nombre y apellido de la tabla usuarios con el criterio WHERE id IN con la subconsulta que nos seleccione la id del usuario de la tabla transactions agrupado por id de usuario y usando la cláusula HAVING, contamos por id con COUNT(id) el número de transacciones que han realizado. Si queremos seleccionar solamente aquellos usuarios que hayan realizado más de 30, ponemos > 30 al final.

```

61  #Ejercicio 1
62  • SELECT name, surname
63  FROM users
64  WHERE id IN (
65      SELECT user_id
66      FROM transactions
67      GROUP BY user_id
68      HAVING COUNT(id) > 30
69  );
70

```

Result Grid

	name	surname
▶	Ocean	Nelson
	Hedwig	Gilbert
	Kenyon	Hartman
	Lynn	Riddle

users 2 x

Output

Action Output

#	Time	Action	Message
✓ 1	11:55:04	SELECT name, surname FROM users WHERE id IN ( SELECT user_id FROM transactions ...	4 row(s) returned

## Ejercicio 2

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

Para este ejercicio seleccionamos variables usando alias ya que utilizaremos JOINS para unir la tabla transactions con credit\_cards y companies por medio de las ids de éstas. Seleccionamos de la tabla credit cards (alias cc) la variable iban, para calcular el importe promedio, usamos ROUND(AVG(t.amount, 2) para obtener el cálculo redondeado a 2 decimales, lo renombramos todo con el alias avg\_amount. Ponemos en la cláusula FROM la variable con las claves foráneas transactions y hacemos el doble JOIN de credit\_cards y transactions por medio de cc.id = t.card\_id y companies con transactions por medio de t.business\_id = c.company\_id. Usamos el criterio WHERE con el nombre de la empresa 'Donec Ltd' para ver la media de importe de las tarjetas de crédito de dicha compañía. Agrupamos por la variable de credit\_cards iban.

```

71 #Ejercicio 2
72 • SELECT cc.iban, ROUND(AVG(t.amount), 2) AS avg_amount
73 FROM transactions t
74 JOIN credit_cards cc ON cc.id = t.card_id
75 JOIN companies c ON t.business_id = c.company_id
76 WHERE c.company_name = 'Donec Ltd'
77 GROUP BY cc.iban;
78

```

iban	avg_amount
PT87806228135092429456346	203.72

Result 4 x

Output

Action Output

#	Time	Action	Message
1	12:09:52	SELECT cc.iban, ROUND(AVG(t.amount), 2) AS avg_amount FROM transactions t JOIN cred...	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:**

Tal como nos piden, creamos una tabla llamada card\_status con una variable card\_id que figure como clave primaria, con referencia a la variable id de la tabla credit\_cards y otra variable status que nos represente el estado de la tarjeta de crédito (activada o desactivada), según el criterio establecido en el enunciado.

```

81 #NIVEL 2
82 #Ejercicio 1
83 • CREATE TABLE IF NOT EXISTS card_status (
84     card_id CHAR(8) PRIMARY KEY,
85     status VARCHAR(20),
86     FOREIGN KEY (card_id) REFERENCES credit_cards(id)
87 );

```

Output

Action Output

#	Time	Action	Message
2	13:22:54	CREATE TABLE IF NOT EXISTS card_status ( card_id CHAR(8) PRIMARY KEY, status V...	0 row(s) affected



Una vez tenemos la tabla creada, haremos la introducción de los datos. Para ello usamos INSERT INTO card\_status (card\_id, status) y en lugar de escribir VALUES como hemos hecho en anteriores ocasiones, haremos una consulta seleccionando card\_id, usaremos CASE WHEN SUM(declined) = COUNT(\*) THEN 'not activated' para determinar que si la suma de todas las transacciones declinadas que cogemos iguala el número total de filas contadas en la selección, la tarjeta está desactivada. Si hay un solo 0, estará activada (ELSE activated'). Finalizamos con END AS status para que se introduzcan los datos correctamente en dicha columna. Para saber de donde coger estos datos, en FROM usaremos una subconsulta seleccionando card\_id, declined, timestamp y una columna nueva que deberemos crear para poder coger las 3 o menos de 3 transacciones más recientes de cada card\_id. Creí que usando ORDER BY timestamp y LIMIT 3 podría establecer el criterio que nos dan como requisito para saber el estado de cada tarjeta. Pero haciendo esto solo cogemos las 3 primeras filas. El resto de card\_ids no se seleccionan. Consultando online vi que se puede establecer dicho criterio si creamos una columna que enumere las transacciones para cada card\_id. La cláusula es ROW\_NUMBER( ) OVER(PARTITION BY card\_id ORDER BY timestamp DESC) AS row\_num. Esto nos añadirá dicha columna con un índice que se auto incrementará para cada card\_id, ordenado según la fecha de más reciente a menos. Las otras columnas seleccionadas las cogemos de la tabla transactions. Cerramos paréntesis de la subconsulta y le añadimos un alias AS LastThreeTransactions. En este caso, si no añadimos alias a la subconsulta nos da error. Añadimos debajo el criterio WHERE row\_num <= 3 para que se quede con las 3 o menos de 3 transacciones más recientes de la subconsulta y GROUP BY card\_id. Vemos cómo queda la tabla card\_status.

```
87  #Insertar valores de status según las tres últimas transacciones
88  • INSERT INTO card_status (card_id, status)
89  SELECT card_id, CASE WHEN SUM(declined) = COUNT(*) THEN 'not activated' ELSE 'activated' END AS status
90  FROM (SELECT card_id, declined, timestamp,
91          ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS row_num
92       FROM transactions) AS LastThreeTransactions
93  WHERE row_num <= 3
94  GROUP BY card_id;
```

Output			
Action Output			
#	Time	Action	Message
1	14:24:10	INSERT INTO card_status (card_id, status) SELECT card_id, CASE WHEN SUM(declined) = ...	275 row(s) affected Records: 275 Duplicates: 0 Warnings: 0

Hacemos una consulta para ver si los datos están bien introducidos

```

96 • SELECT *
97 FROM card_status;
98

```

card_id	status
CcU-2938	activated
CcU-2945	activated
CcU-2952	activated
CcU-2959	activated
CcU-2966	activated

Output

#	Time	Action	Message
2	14:24:14	SELECT * FROM card_status LIMIT 0, 1000	275 row(s) returned

## Ejercicio 1

### ¿Cuántas tarjetas están activas?

Para responder esta pregunta, simplemente hacemos `SELECT COUNT(*) AS active_cards` de la tabla `card_status` `WHERE status = 'activated'` y nos contará el total de tarjetas activas.

```

99 #Número de tarjetas activas
100 • SELECT COUNT(*) AS active_cards
101 FROM card_status
102 WHERE status = 'activated';
103

```

active_cards
275

Result 3 x Read Only

Output

#	Time	Action	Message
1	14:57:40	SELECT COUNT(*) AS active_cards FROM card_status WHERE status = 'activated' LIMIT 0,...	1 row(s) returned

Según la consulta, todas están activadas, no hay ninguna con sus últimas 3 o menos de 3 transacciones declinadas.

## 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:**

Hacemos el mismo código que usamos para las tablas anteriores. Añadimos las columnas y nos aseguramos de incluir el tipo de dato adecuado para que no hayan errores al importar los datos del archivo csv. Id será la primary key.

```
106 • CREATE TABLE IF NOT EXISTS products (  
107     id INT PRIMARY KEY,  
108     product_name VARCHAR(50),  
109     price VARCHAR(20),  
110     colour CHAR(7),  
111     weight DECIMAL(4, 1),  
112     warehouse_id VARCHAR(10)  
113 );  
114
```

Output			
Action Output			
#	Time	Action	Message
✓ 1	15:01:36	CREATE TABLE IF NOT EXISTS products ( id INT PRIMARY KEY, product_name VA...	0 row(s) affected

Importamos los datos y visualizamos la tabla para ver que todo está correcto.

115 • **SELECT \***

116 **FROM products;**

117

The screenshot shows a database interface with a 'Result Grid' and an 'Output' section. The 'Result Grid' displays a table with 6 columns: id, product\_name, price, colour, weight, and warehouse\_id. The 'Output' section shows a log of actions, including a successful SQL query execution.

id	product_name	price	colour	weight	warehouse_id
1	Direwolf Stannis	\$161.11	#7c7c7c	1.0	WH-4
2	Tarly Stark	\$9.24	#919191	2.0	WH-3
3	duel tourney Lannister	\$171.13	#d8d8d8	1.5	WH-2
4	warden south duel	\$71.89	#111111	3.0	WH-1
5	skywalker ewok	\$171.22	#dbdbdb	3.2	WH-0
6	dooku solo	\$136.60	#c4c4c4	0.8	WH--1
7	north of Casterly	\$63.33	#b7b7b7	0.6	WH--2
8	Winterfell	\$32.37	#383838	1.4	WH--3
9	Winterfell	\$76.40	#5b5b5b	1.2	WH--4
10	Karstark Dorne	\$119.52	#f4f4f4	2.4	WH--5
11	Karstark Dorne	\$49.70	#141414	2.7	WH--6
12	duel Direwolf	\$181.60	#a8a8a8	2.1	WH--7
13	palpatine chewbacca	\$139.59	#2b2b2b	1.0	WH--8

products 6 x

Output

Action Output

#	Time	Action	Message
1	15:05:57	SELECT * FROM products LIMIT 0, 1000	100 row(s) returned

Una vez tenemos la tabla products, debemos relacionarla con transactions. Si nos fijamos en la columna product\_ids de transactions, hay varios valores separados por comas en una misma fila o registro. Esto nos puede traer problemas si queremos hacer la relación con products o hacer consultas que impliquen ambas tablas. De modo que hay que separar los valores y hacer que aparezca un solo product\_id por fila. Debemos separar y duplicar el resto de datos de cada fila o registro. Ignoro por completo la forma de hacerlo, así que consulté online y vi que se puede hacer de forma manual en excel usando fórmulas con cláusulas como SPLIT y FLATTEN. Split separa los valores de product\_ids y los coloca en distintas columnas, flatten los convierte en filas. Sería cuestión de añadir el resto de valores manualmente copiando y pegando. También se puede usar SQL pero la query es demasiado larga o complicada en mi opinión. Sabiendo que vamos a manejar Python y librerías como pandas, he pensado que la solución más rápida y fácil de entender en cuanto a código, sería ésta: Creamos un notebook en Jupyter, importamos el archivo en pandas, hacemos la transformación y generamos un nuevo archivo limpio para poder importar en MySQL. Aquí está el código:

```

In [1]: 1 import pandas as pd
2
3 # Cargar el archivo desde la ubicación especificada
4 file_path = r'C:\Users\Joan Salas Dalmáu\Downloads\transactions.xlsx'
5
6 # Leer el archivo en un DataFrame
7 df = pd.read_excel(file_path)
8
9 # La columna 'product_ids' contiene valores separados por comas
10 # Separamos los valores de 'product_ids' en listas
11 df['product_ids'] = df['product_ids'].str.split(',')
12
13 # Usamos explode para crear nuevas filas por cada elemento de la lista en 'product_ids'
14 df = df.explode('product_ids')
15
16 # Eliminar filas con valores nulos (NaN)
17 df = df.dropna(subset=['product_ids'])
18
19 # Convertimos los valores de 'product_ids' a int
20 df['product_ids'] = df['product_ids'].astype(int)
21
22 # Guardamos el nuevo DataFrame en un archivo Excel limpio
23 cleaned_file_path = r'C:\Users\Joan Salas Dalmáu\Downloads\transactions_cleaned.xlsx'
24 df.to_excel(cleaned_file_path, index=False)
25
26 # Mostrar el DataFrame resultante
27 df

```

Out[1]:

	id	card_id	business_id	timestamp	amount	declined	product_ids	user_id	lat	longitude
1	7DC26247-20EC-53FE-E555-B6C2E55CA5D5	CcU-2945	b-2226	2022-02-04 15:52:56	312.50	0	71	275	58.936718	-76.817110
1	7DC26247-20EC-53FE-E555-B6C2E55CA5D5	CcU-2945	b-2226	2022-02-04 15:52:56	312.50	0	41	275	58.936718	-76.817110
2	72997E96-DC2C-A4D7-7C24-66C302F8AE5A	CcU-2952	b-2230	2022-01-30 15:16:36	239.87	0	97	275	43.358406	-17.657968
2	72997E96-DC2C-A4D7-7C24-66C302F8AE5A	CcU-2952	b-2230	2022-01-30 15:16:36	239.87	0	41	275	43.358406	-17.657968
2	72997E96-DC2C-A4D7-7C24-66C302F8AE5A	CcU-2952	b-2230	2022-01-30 15:16:36	239.87	0	3	275	43.358406	-17.657968
...	...	...	...	...	...	...	...	...	...	...
585	A4D0D84F-4622-BB83-E6B6-51E545D4A217	CcU-3533	b-2562	2021-05-14 16:59:27	395.81	1	71	267	-24.640383	-69.876969
585	A4D0D84F-4622-BB83-E6B6-51E545D4A217	CcU-3533	b-2562	2021-05-14 16:59:27	395.81	1	43	267	-24.640383	-69.876969
585	A4D0D84F-4622-BB83-E6B6-51E545D4A217	CcU-3533	b-2562	2021-05-14 16:59:27	395.81	1	2	267	-24.640383	-69.876969
586	9FBB3D61-D3C2-E5BB-4BC3-6CC83C718D34	CcU-3540	b-2566	2021-04-19 21:33:41	490.19	1	53	267	75.211672	-171.044656
586	9FBB3D61-D3C2-E5BB-4BC3-6CC83C718D34	CcU-3540	b-2566	2021-04-19 21:33:41	490.19	1	17	267	75.211672	-171.044656

1248 rows x 10 columns

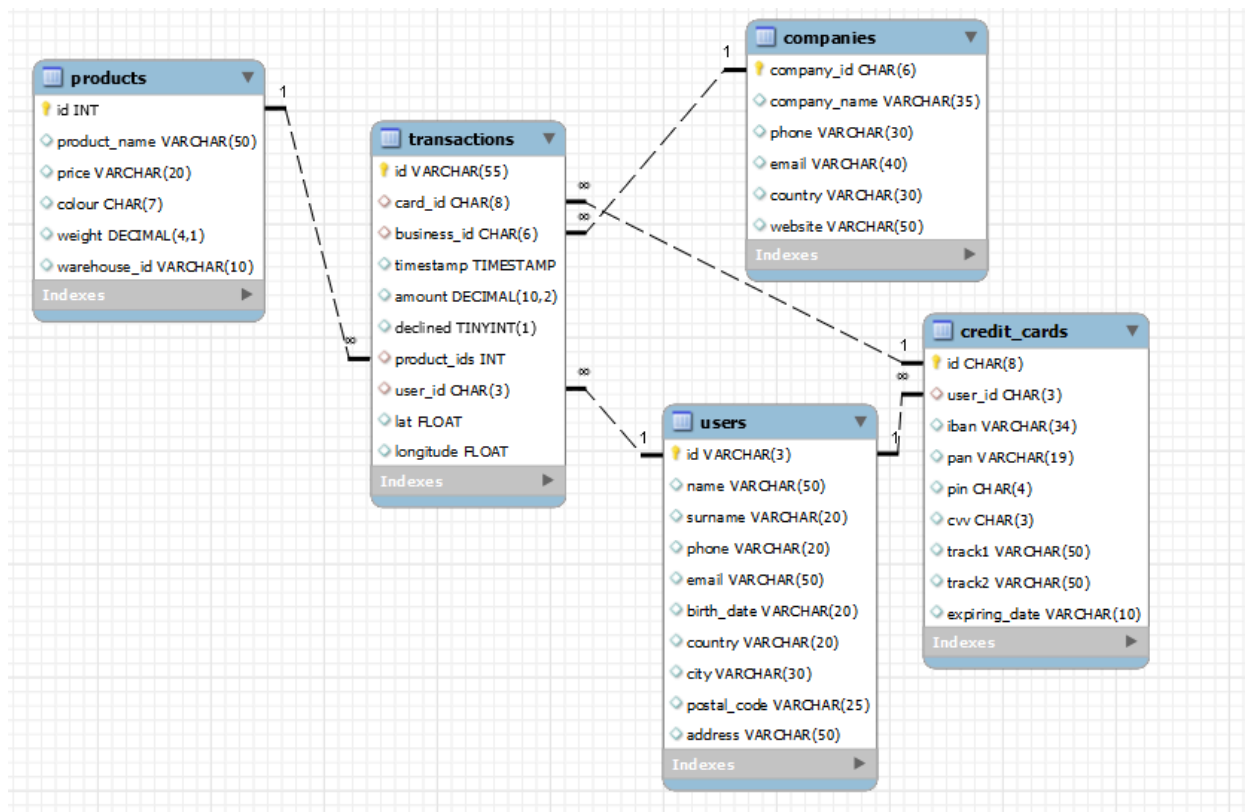
Ahora ya tenemos un solo valor en product\_id con el resto de columnas duplicadas. Eliminamos los datos de la tabla transactions, añadimos la clave foránea en transactions e importamos el archivo limpiado.

```

118 #Eliminamos datos e importamos tabla con product_ids limpiada (un solo valor por fila)
119 • TRUNCATE TABLE transactions;
120
121 #Añadimos restricción y clave foránea para relacionar la tabla transactions con products
122 • ALTER TABLE transactions
123   ADD CONSTRAINT fk_product_ids
124   FOREIGN KEY (product_ids) REFERENCES products(id);
125

```

No incluyo screenshot de la ejecución del código en action output pero puedo confirmar que se ejecuta correctamente ya que si vuelvo a generar un diagrama con database > reverse engineer, aparecen todas las tablas relacionadas según las claves establecidas.



Veamos la tabla limpia con la columna de product\_ids con un valor por registro.

script tarea 4\* transactions x

Limit to 1000 rows

1 • `SELECT * FROM transactions_db.transactions;`

Result Grid

	id	card_id	business_id	timestamp	amount	declined	product_ids	user_id	lat	lon
▶	02C6201E-D90A-1859-B4EE-88D2986D3B02	CcU-2938	b-2362	2021-08-28 23:42:24	466.92	0	71	92	81.9185	-12.1
	0466A42E-47CF-8D24-FD01-C0B689713128	CcU-4219	b-2302	2021-07-26 07:29:18	49.53	0	47	170	-43.9695	-11.7
	063FBA79-99EC-66FB-29F7-25726D1764A5	CcU-2987	b-2250	2022-01-06 21:25:27	92.61	0	47	275	-81.2227	-12.5
	0668296C-CDB9-A883-76BC-2E4C44F8C8AE	CcU-3743	b-2618	2022-01-26 02:07:14	394.18	0	89	265	-34.3593	-10.0
	06CD9AA5-9B42-D684-DDDD-A5E394FEBA99	CcU-2959	b-2346	2021-10-26 23:00:01	279.93	0	43	92	33.7381	158
	07A46D48-31A3-7E87-65B9-0DA902AD109F	CcU-3225	b-2386	2021-06-28 21:11:42	340.87	1	47	272	38.8342	92.1
	09DE92CE-6F27-2BB7-13B5-9385B2B3B8E2	CcU-3071	b-2298	2021-05-11 20:40:06	303.05	1	67	275	71.1706	10.5
	0A476ED9-0C13-1962-F87B-D3563924B539	CcU-4359	b-2302	2022-02-26 20:33:54	430.49	0	29	221	-56.4901	114
	08EB80B7-9D66-1707-CE4B-9DC7E71914B5	CcU-3141	b-2338	2022-03-04 14:54:35	288.81	1	19	272	23.3264	-13.
	0C7C3A33-9947-3BC1-846D-7BE3D0D17598	CcU-3309	b-2434	2021-04-10 20:58:41	103.44	1	89	272	63.3615	-68.
	0CE957A6-CCAA-2B7A-6839-8A4B1B324853	CcU-3435	b-2506	2022-02-02 07:29:36	428.69	1	83	269	-69.3537	-10.
	0DD2E608-5C9E-D1B3-4999-B99F43AD735A	CcU-2959	b-2234	2021-04-17 05:30:17	252.47	1	7	275	9.68811	130

transactions 1 x

Output

Action Output

#	Time	Action	Message
✓ 1	13:31:16	SELECT * FROM transactions_db.transactions LIMIT 0, 1000	436 row(s) returned

## Ejercicio 1

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

Para saber el número de ventas de cada producto haremos la siguiente consulta, seleccionamos de la tabla products con el alias 'p' product\_name, usamos la cláusula COUNT(t.product\_ids) AS sales de la tabla transactions para que nos dé el número de ventas de cada producto. Juntamos products y transactions por p.id = t.product\_ids, usamos el criterio WHERE declined = 0 para coger las transacciones que se han hecho efectivas, agrupamos por nombre de producto y ordenamos por ventas de mayor a menor para ver los más vendidos primero.

```

126 # número de veces que se ha vendido cada producto
127 • SELECT p.product_name, COUNT(t.product_ids) AS sales
128 FROM transactions t
129 JOIN products p ON p.id = t.product_ids
130 WHERE declined = 0
131 GROUP BY p.product_name
132 ORDER BY sales DESC;

```


 Filter Rows: 
 Export: 
 Wrap Cell Content: 

	product_name	sales
▶	skywalker ewok	24
	riverlands north	23
	Direwolf riverlands the	23
	Direwolf Stannis	21
	Winterfell Lannister	21
	duel	20
	skywalker ewok sith	20

Result 5 ×

Output



Action Output

#	Time	Action	Message
✓ 1	13:18:56	SELECT p.product_name, COUNT(t.product_ids) AS sales FROM transactions t JOIN produc...	24 row(s) returned