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:

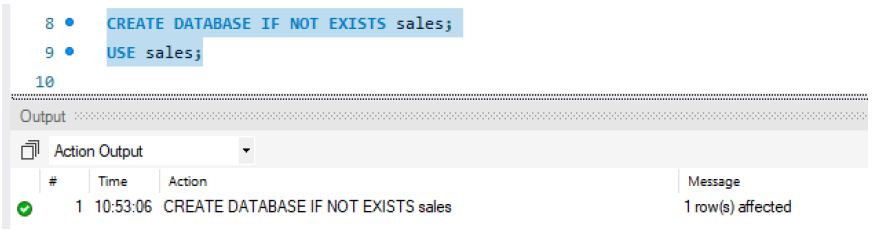
Descripción de la base de datos:

Inicialmente tenemos 7 tablas, pero 3 de ellas se pueden unificar en una, ya que se trata de 3 tablas con las mismas columnas de usuarios de nacionalidades de Canadá, USA y UK.

Tenemos por lo tanto 4 tablas: companies, users, credit_card y transactions.

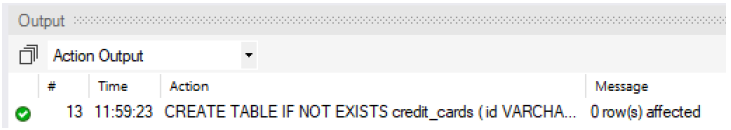
Se trata de una base de datos relacional estructurada en un esquema de modelo dimensional de estrella con la tabla 'transactions' como la tabla de hechos y las tablas 'users', credit_card', 'products' y 'companies' como las tablas de dimensiones que se relacionan con la de hechos 1 a N. Existe también una relación entre las tabla users y credit_cards 1 a N, que consideramos no es necesaria y procederemos a eliminar. Haremos también las modificaciones necesarias para optimizar y ordenar la base de datos: cambiar los tipos de birth_date y expiring_date de VARCHAR a DATE.

Creamos la base de datos a la que hemos llamado sales:
CREATE DATABASE IF NOT EXISTS sales;

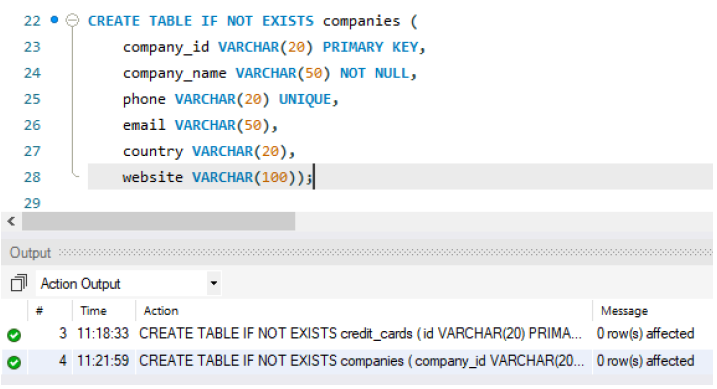


Creamos las tablas:
CREATE TABLE IF NOT EXISTS **credit_cards** (
id VARCHAR(20) PRIMARY KEY,
user_id INT NOT NULL,
iban VARCHAR(50) UNIQUE NOT NULL,
pan VARCHAR(50) UNIQUE NOT NULL,
pin VARCHAR(50) NOT NULL,
cvv INT UNIQUE NOT NULL,
track1 VARCHAR(50),
track2 VARCHAR(50),
expiring_date VARCHAR(10) NOT NULL);

```
DROP TABLE credit_cards;
CREATE TABLE IF NOT EXISTS credit_cards (
    id VARCHAR(20) PRIMARY KEY,
    user_id INT NOT NULL,
    iban VARCHAR(50) UNIQUE NOT NULL,
    pan VARCHAR(50) UNIQUE NOT NULL,
    pin VARCHAR(50) NOT NULL,
    cvv INT NOT NULL,
    track1 VARCHAR(50),
    track2 VARCHAR(50),
    expiring_date VARCHAR(10) NOT NULL);
```




```
CREATE TABLE IF NOT EXISTS companies (
    company_id VARCHAR(20) PRIMARY KEY,
    company_name VARCHAR(50) NOT NULL,
    phone VARCHAR(20) UNIQUE,
    email VARCHAR(50),
    country VARCHAR(20),
    website VARCHAR(100));
```




```
CREATE TABLE IF NOT EXISTS users (
    id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    surname VARCHAR(50) NOT NULL,
    phone VARCHAR(20) UNIQUE,
    email VARCHAR(50) UNIQUE NOT NULL,
    birth_date VARCHAR(20),
    country VARCHAR(20),
    city VARCHAR(20),
    postal_code VARCHAR(20),
    adress VARCHAR(100));
```

```
CREATE TABLE IF NOT EXISTS users (
    id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    surname VARCHAR(50) NOT NULL,
    phone VARCHAR(20) UNIQUE,
    email VARCHAR(50) UNIQUE NOT NULL,
    birth_date VARCHAR(10),
    country VARCHAR(20),
    city VARCHAR(20),
    postal_code VARCHAR(20),
    adress VARCHAR(100));
```

Output

Action Output

#	Time	Action	Message
21	12:21:50	DROP TABLE users	0 row(s) affected
22	12:21:58	CREATE TABLE IF NOT EXISTS users (id...	0 row(s) affected

```
CREATE TABLE IF NOT EXISTS transactions (
    id VARCHAR(100) PRIMARY KEY,
    card_id VARCHAR(20) NOT NULL,
    business_id VARCHAR(20) NOT NULL,
    timestamp TIMESTAMP,
    amount DECIMAL(10,2) NOT NULL,
    declined boolean,
    product_ids VARCHAR(50) NOT NULL,
    user_id INT NOT NULL,
    lat FLOAT,
    longitude FLOAT);
```

Output

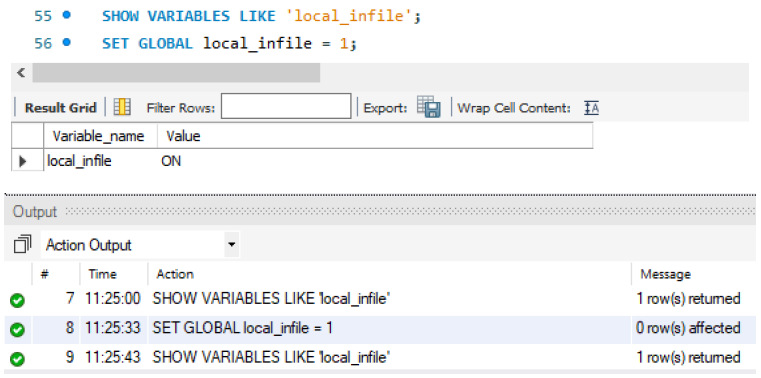
Action Output

#	Time	Action	Message
5	11:23:28	CREATE TABLE IF NOT EXISTS users (id INT PRIMARY KEY, name ...	0 row(s) affected
6	11:23:43	CREATE TABLE IF NOT EXISTS transactions (id VARCHAR(100) PRIM...	0 row(s) affected

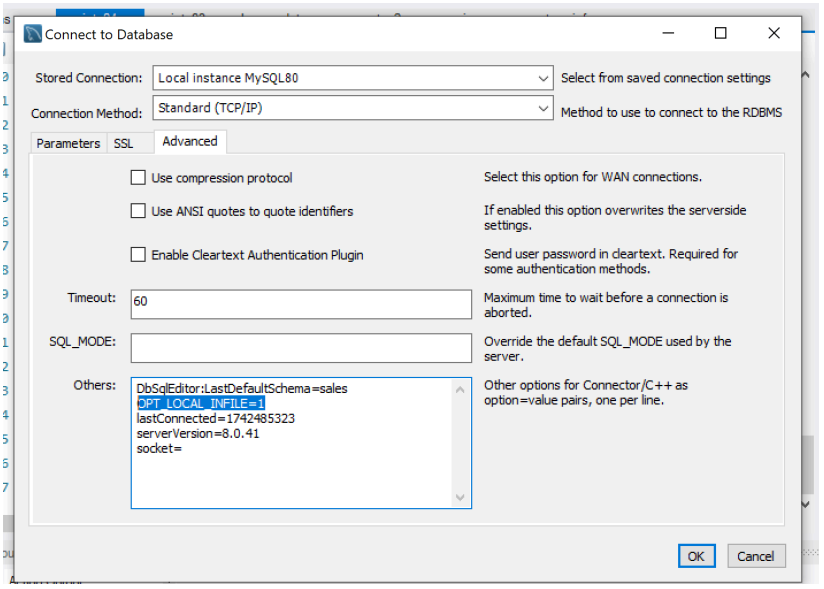
Inserción de datos:

Para poder insertar un archivo local para insertar los datos hay que verificar si la opción está activada. La respuesta inicial es negativa, aparece como OFF, por lo que la activamos. Comprobamos después de activarla y aparece como ON.

```
SHOW VARIABLES LIKE 'local_infile';
SET GLOBAL local_infile = 1;
```



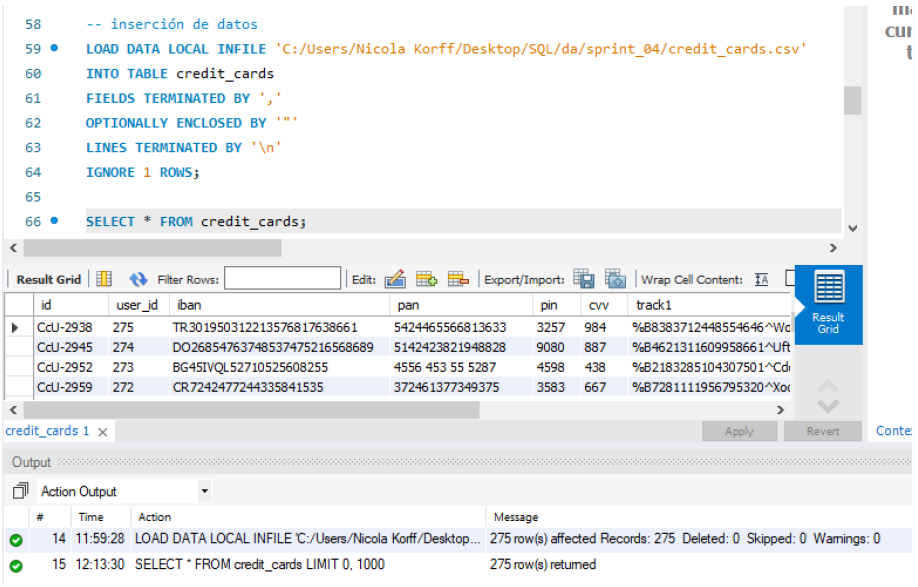
También hemos tenido que ir desde la barra de herramientas de Mysql a 'Database' a 'Manage connections' y en la pestaña 'Advanced' añadir en 'Others' el código: OPT_LOCAL_INFILE=1.



Insertamos los datos:

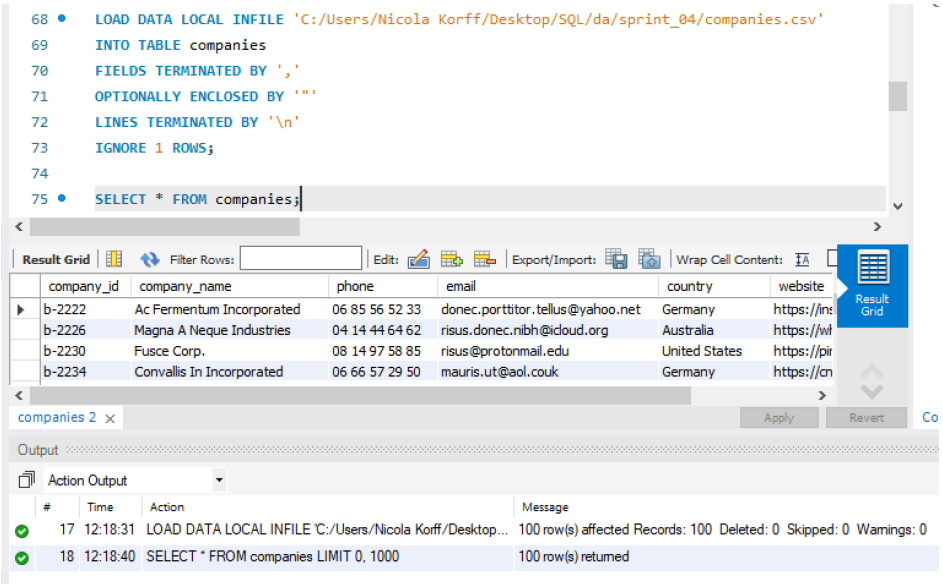
LOAD DATA LOCAL INFILE 'C:/Users/Nicola Korff/Desktop/SQL/da/sprint_04/credit_cards.csv'
INTO TABLE **credit_cards**
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

SELECT * FROM credit_cards;



LOAD DATA LOCAL INFILE 'C:/Users/Nicola Korff/Desktop/SQL/da/sprint_04/companies.csv'
INTO TABLE **companies**
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

SELECT * FROM companies;



Para la inserción de los datos user, daba error al insertar datos, al principio pensábamos que era porque había campos que contenían comas y llevaba a confusión, pero al mirar el excel hemos visto que la separación era correcta. Por lo tanto, hemos pensado que el problema podría radicar en el salto de línea y al cambiar \n a \r\n ha funcionado. Los archivos de credit_cards, companies y transactions han sido creados en formato Unix/Mac OS X y el de users en formato Windows. A continuación específico las diferencias entre los diferentes tipos de saltos de línea.

\r = CR (Retorno de carro) → Se usa como carácter de nueva línea en Mac OS antes de X
\n = LF (Salto de línea) → Se usa como carácter de nueva línea en Unix/Mac OS X

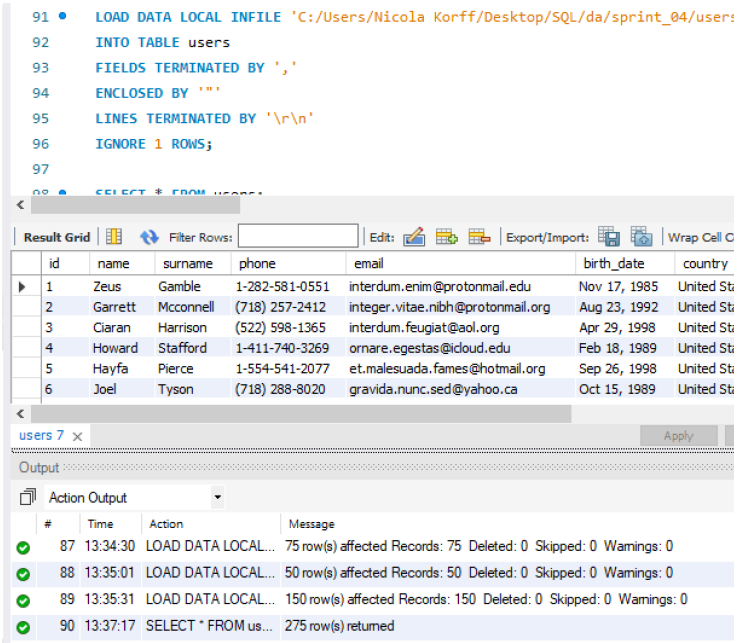
\r\n = CR + LF → Se usa como carácter de nueva línea en Windows

```
LOAD DATA LOCAL INFILE 'C:/Users/Nicola Korff/Desktop/SQL/da/sprint_04/users_ca.csv'
INTO TABLE users
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS;

LOAD DATA LOCAL INFILE 'C:/Users/Nicola Korff/Desktop/SQL/da/sprint_04/users_uk.csv'
INTO TABLE users
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS;

LOAD DATA LOCAL INFILE 'C:/Users/Nicola Korff/Desktop/SQL/da/sprint_04/users_usa.csv'
INTO TABLE users
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS;

SELECT * FROM users;
```




```
LOAD DATA LOCAL INFILE 'C:/Users/Nicola Korff/Desktop/SQL/da/sprint_04/transactions.csv'
INTO TABLE transactions
FIELDS TERMINATED BY ';'
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

SELECT * FROM transactions;
```

```
100 •  LOAD DATA LOCAL INFILE 'C:/Users/Nicola Korff/Desktop/SQL/da/sprint_04/transactions.csv'
101    INTO TABLE transactions
102    FIELDS TERMINATED BY ';'
103    ENCLOSED BY '"'
104    LINES TERMINATED BY '\n'
105    IGNORE 1 ROWS;
106
107 •  SELECT * FROM transactions;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	card_id	business_id	timestamp	amount	declined	product
02C6201E-D90A-1859-B4EE-88D2986D3B02	CcU-2938	b-2362	2021-08-28 23:42:24	466.92	0	71, 1, 19
0466A42E-47CF-8D24-FD01-C0B689713128	CcU-4219	b-2302	2021-07-26 07:29:18	49.53	0	47, 97, 4
063FBA79-99EC-66FB-29F7-25726D1764A5	CcU-2987	b-2250	2022-01-06 21:25:27	92.61	0	47, 67, 3
0668296C-CDB9-A883-76BC-2E4C44F8C8AE	CcU-3743	b-2618	2022-01-26 02:07:14	394.18	0	89, 83, 7

transactions 3 × Apply

Output

Action Output

#	Time	Action	Message
53	12:47:49	LOAD D...	587 row(s) affected Records: 587 Deleted: 0 Skipped: 0 Warnings: 0
54	12:47:53	SELECT...	587 row(s) returned

Modificación en las tablas:

CREDIT_CARDS

Eliminamos el campo credit_card.user_id:

ALTER TABLE credit_card DROP COLUMN user_id;

```
109 •  ALTER TABLE credit_cards DROP COLUMN user_id;
110 •  SHOW COLUMNS FROM credit_cards;
```

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

Field	Type	Null	Key	Default	Extra
id	varchar(20)	NO	PRI	NULL	
iban	varchar(50)	NO	UNI	NULL	
pan	varchar(50)	NO	UNI	NULL	
pin	varchar(50)	NO		NULL	
cvv	int	NO		NULL	
track1	varchar(50)	YES		NULL	
track2	varchar(50)	YES		NULL	
expiring_date	varchar(10)	NO		NULL	

Result 50 ×

Output

Action Output

#	Time	Action	Message
294	16:32:48	ALTER TABLE credit_cards DROP COLU...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
295	16:33:31	SHOW COLUMNS FROM credit_cards	8 row(s) returned

Modificamos el tipo de dato de la columna expiring_date:

ALTER TABLE credit_cards ADD COLUMN expiring_date_temp DATE;
UPDATE credit_cards SET expiring_date_temp = STR_TO_DATE(expiring_date, '%m/%d/%y');
ALTER TABLE credit_cards DROP expiring_date;
ALTER TABLE credit_cards CHANGE expiring_date_temp expiring_date DATE NOT NULL;
SHOW COLUMNS FROM credit_cards;

```
111     -- Modificamos el tipo de dato de la columna expiring_date:
112 •   ALTER TABLE credit_cards ADD COLUMN expiring_date_temp DATE;
113 •   UPDATE credit_cards SET expiring_date_temp = STR_TO_DATE(expiring_date, '%m/%d/%y');
114 •   ALTER TABLE credit_cards DROP expiring_date;
115 •   ALTER TABLE credit_cards CHANGE expiring_date_temp expiring_date DATE NOT NULL;
116 •   SHOW COLUMNS FROM credit_cards;
117
```

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

Field	Type	Null	Key	Default	Extra
id	varchar(20)	NO	PRI	NULL	
iban	varchar(50)	NO	UNI	NULL	
pan	varchar(50)	NO	UNI	NULL	
pin	varchar(50)	NO		NULL	
cvv	int	NO		NULL	
track1	varchar(50)	YES		NULL	
track2	varchar(50)	YES		NULL	
expiring_date	date	NO		NULL	

Result 52

Output

Action Output

#	Time	Action	Message
312	16:44:35	ALTER TABLE credit_cards ADD COLUM...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
313	16:44:38	UPDATE credit_cards SET expiring_date_...	275 row(s) affected Rows matched: 275 Changed: 275 Warnings: 0
314	16:44:41	ALTER TABLE credit_cards DROP expirin...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
315	16:44:45	ALTER TABLE credit_cards CHANGE expi...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
316	16:44:47	SHOW COLUMNS FROM credit_cards	8 row(s) returned

USERS

En user modificamos el tipo de dato de la columna birth_date:

ALTER TABLE user ADD COLUMN birth_date_temp DATE;
UPDATE user SET birth_date_temp = STR_TO_DATE(birth_date, '%b %d, %Y');
ALTER TABLE user DROP birth_date;
ALTER TABLE user CHANGE birth_date_temp birth_date DATE NOT NULL;

```
112     -- En user modificamos el tipo de dato de la columna birth_date:
113 •   ALTER TABLE users ADD COLUMN birth_date_temp DATE;
114 •   UPDATE users SET birth_date_temp = STR_TO_DATE(birth_date, '%b %d, %Y');
115 •   ALTER TABLE users DROP birth_date;
116 •   ALTER TABLE users CHANGE birth_date_temp birth_date DATE NOT NULL;
117 •   SHOW COLUMNS FROM users;
```

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

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	
name	varchar(50)	YES		NULL	
surname	varchar(50)	YES		NULL	
phone	varchar(20)	YES		NULL	
email	varchar(255)	YES		NULL	
country	varchar(100)	YES		NULL	
city	varchar(100)	YES		NULL	
postal_code	varchar(50)	YES		NULL	
adress	varchar(255)	YES		NULL	
birth_date	date	NO		NULL	

Result 51

Output

Action Output

#	Time	Action	Message
303	16:37:00	ALTER TABLE users ADD COLUMN birth...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
304	16:37:03	UPDATE users SET birth_date_temp = ST...	275 row(s) affected Rows matched: 275 Changed: 275 Warnings: 0
305	16:37:07	ALTER TABLE users DROP birth_date	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
306	16:37:10	ALTER TABLE users CHANGE birth_date...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
307	16:37:13	SHOW COLUMNS FROM users	10 row(s) returned

Mostrar que se ha creado correctamente:
La base de datos:
SHOW TABLES FROM sales;

```
116 •  SHOW TABLES FROM sales;
```

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

Tables_in_sales
companies
credit_cards
transactions
users

Result 8

Output

Action Output

#	Time	Action	Message
97	13:41:27	ALTER TABLE credit_...	275 row(s) affected Records: 275 Duplicates: 0 Warnings: 0
98	13:45:23	SHOW TABLES FRO...	4 row(s) returned

Las tablas:
SHOW COLUMNS FROM credit_cards;

```
128 •  SHOW COLUMNS FROM credit_cards;
```

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

Field	Type	Null	Key	Default	Extra
id	varchar(20)	NO	PRI	NULL	
iban	varchar(50)	NO	UNI	NULL	
pan	varchar(50)	NO	UNI	NULL	
pin	varchar(50)	NO		NULL	
cvv	int	NO		NULL	
track1	varchar(50)	YES		NULL	
track2	varchar(50)	YES		NULL	
expiring_date	date	NO		NULL	

SHOW COLUMNS FROM companies;

```
129 •  SHOW COLUMNS FROM companies;
```

Result Grid | Filter Rows: | Export: | Wrap C

Field	Type	Null	Key	Default	Extra
company_id	varchar(20)	NO	PRI	NULL	
company_name	varchar(50)	NO		NULL	
phone	varchar(20)	YES	UNI	NULL	
email	varchar(50)	YES		NULL	
country	varchar(20)	YES		NULL	
website	varchar(100)	YES		NULL	

SHOW COLUMNS FROM users;

```
130 •  SHOW COLUMNS FROM users;
```

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default
id	int	NO	PRI	NULL
name	varchar(50)	YES		NULL
surname	varchar(50)	YES		NULL
phone	varchar(20)	YES		NULL
email	varchar(255)	YES		NULL
country	varchar(100)	YES		NULL
city	varchar(100)	YES		NULL
postal_code	varchar(50)	YES		NULL
adress	varchar(255)	YES		NULL
birth_date	date	NO		NULL

SHOW COLUMNS FROM transactions;

```
121 •  SHOW COLUMNS FROM transactions;
```

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

Field	Type	Null	Key	Default	Extra
id	varchar(100)	NO	PRI	NULL	
card_id	varchar(20)	NO	MUL	NULL	
business_id	varchar(20)	NO	MUL	NULL	
timestamp	timestamp	YES		NULL	
amount	decimal(10,2)	NO		NULL	
declined	tinyint(1)	YES		NULL	
product_ids	varchar(50)	NO		NULL	
user_id	int	NO	MUL	NULL	
lat	float	YES		NULL	
longitude	float	YES		NULL	

Result 16

Output

Action Output

#	Time	Action	Message
106	13:53:01	SHOW COLUMNS FROM users	10 row(s) returned
107	13:53:52	SHOW COLUMNS FROM transactions	10 row(s) returned

Creamos las relaciones de Foreign keys entre las tablas

ALTER TABLE transactions
ADD CONSTRAINT fk_transactions_credit_cards FOREIGN KEY (card_id) REFERENCES credit_cards (id),
ADD CONSTRAINT fk_transactions_companies FOREIGN KEY (business_id) REFERENCES companies (company_id),
ADD CONSTRAINT fk_transactions_users FOREIGN KEY (user_id) REFERENCES users (id);

Comprobamos que se han creado correctamente

SHOW INDEXES FROM transactions;

```
134     -- Creamos las relaciones de Foreign keys entre las tablas
135 •   ALTER TABLE transactions
136     ADD CONSTRAINT fk_transactions_credit_cards FOREIGN KEY (card_id) REFERENCES credit_cards (id),
137     ADD CONSTRAINT fk_transactions_companies FOREIGN KEY (business_id) REFERENCES companies (company_id),
138     ADD CONSTRAINT fk_transactions_users FOREIGN KEY (user_id) REFERENCES users (id);
139
140     -- Comprobamos que se han creado correctamente
141 •   SHOW INDEXES FROM transactions;
```

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

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed
transactions	0	PRIMARY	1	id	A	587	NULL	NULL
transactions	1	fk_transactions_credit_cards	1	card_id	A	275	NULL	NULL
transactions	1	fk_transactions_companies	1	business_id	A	100	NULL	NULL
transactions	1	fk_transactions_users	1	user_id	A	216	NULL	NULL

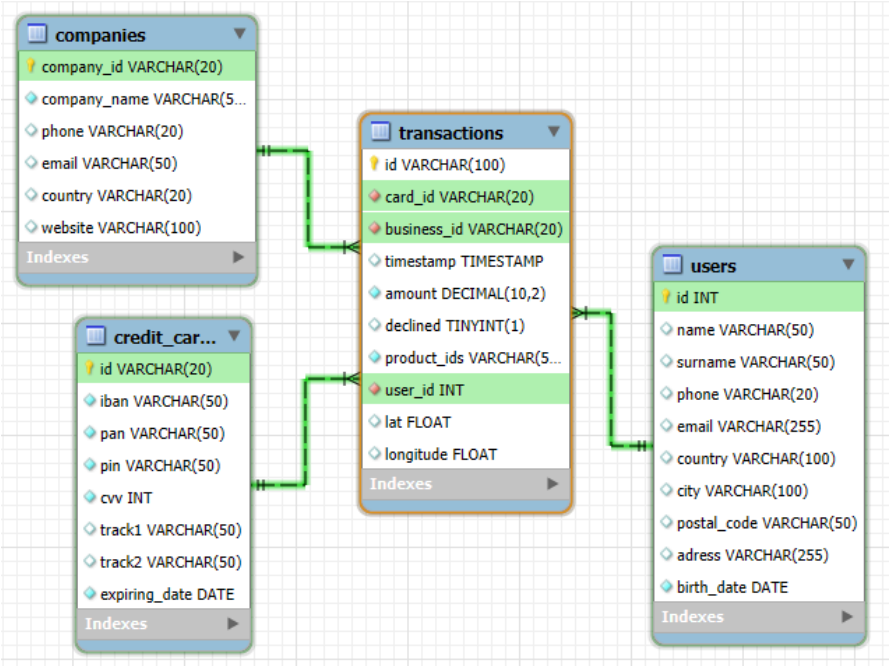
Result 56

Output

Action Output

#	Time	Action	Message
324	16:54:29	ALTER TABLE transactions ADD CONST...	587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0
325	16:54:44	SHOW INDEXES FROM transactions	4 row(s) returned

Esquema final de la base de datos 'sales':

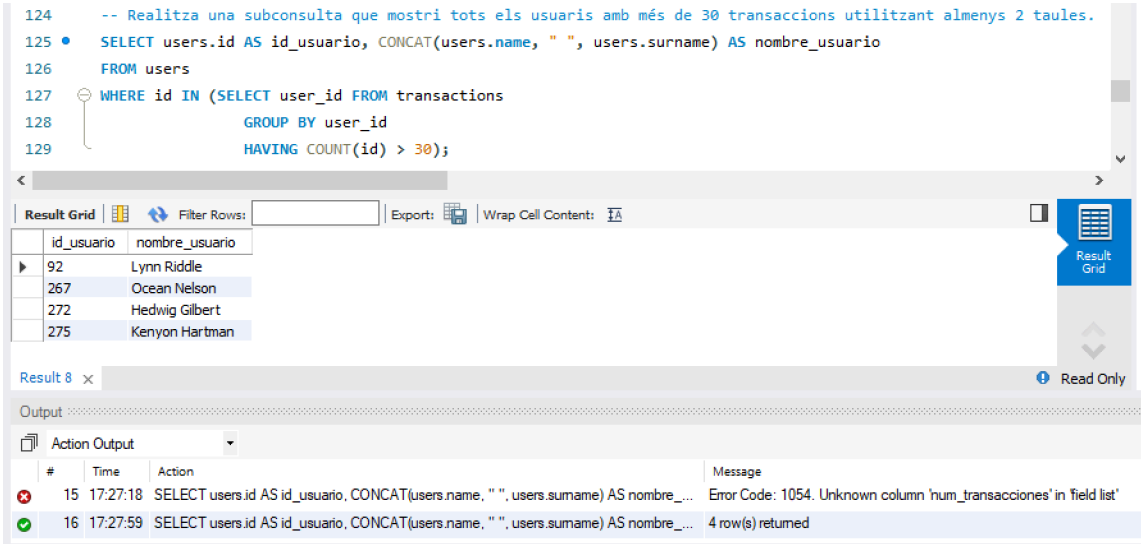


- Exercici 1

Realitza una subconsulta que mostri tots els usuaris amb més de 30 transaccions utilitzant almenys 2 taules.

Seleccionamos el id de usuario y el nombre y apellido que con el método CONCAT unimos para que aparezcan juntos en una sola columna, filtramos a través de una subconsulta donde seleccionamos y agrupamos por el id de usuario y filtramos a los usuarios con más de 30 transacciones con un COUNT y HAVING.

```
SELECT users.id AS id_usuario, CONCAT(users.name, " ", users.surname) AS nombre_usuario
FROM users
WHERE id IN (SELECT user_id FROM transactions
                                    GROUP BY user_id
                                    HAVING COUNT(id) > 30);
```

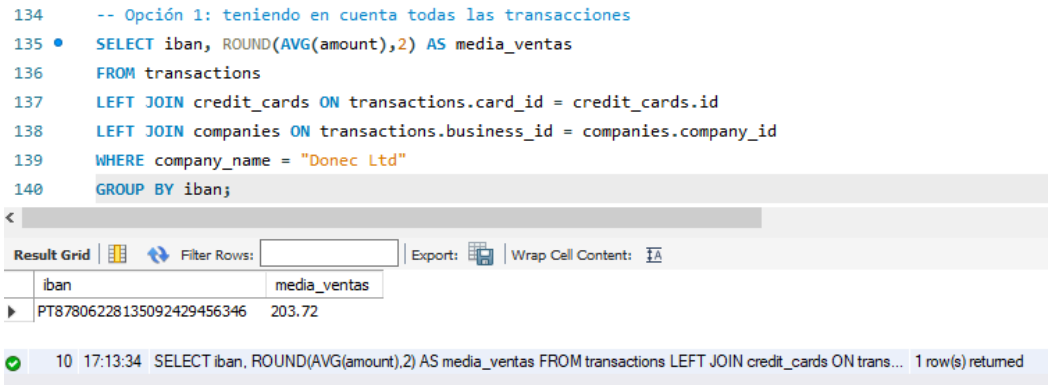


- Exercici 2

Mostra la mitjana d'amount per IBAN de les targetes de crèdit a la companyia Donec Ltd, utilitza almenys 2 taules.

Seleccionamos el iban y hacemos la media del amount redondeada. Unimos las tablas transactions, credit_cards y companies por id de tarjeta e id de empresa con un LEFT JOIN. Filtramos con WHERE por el nombre de empresa deseado y agrupamos por iban.

-- Opción 1: teniendo en cuenta todas las transacciones
SELECT iban, ROUND(AVG(amount),2) AS media_ventas
FROM transactions
LEFT JOIN credit_cards ON transactions.card_id = credit_cards.id
LEFT JOIN companies ON transactions.business_id = companies.company_id
WHERE company_name = "Donec Ltd"
GROUP BY iban;



Opción 2: hacemos lo mismo, pero descartando las transacciones con pagos rechazados
SELECT iban, ROUND(AVG(amount),2) AS media_ventas
FROM transactions
JOIN credit_cards ON transactions.card_id = credit_cards.id
JOIN companies ON transactions.business_id = companies.company_id
WHERE company_name = “Donec Ltd” AND declined = 0
GROUP BY iban;

Obtenemos un resultado distinto:

```
142     -- Opción 2: hacemos lo mismo, pero descartando las transacciones con pagos rechazados
143     SELECT iban, ROUND(AVG(amount),2) AS media_ventas
144     FROM transactions
145     LEFT JOIN credit_cards ON transactions.card_id = credit_cards.id
146     LEFT JOIN companies ON transactions.business_id = companies.company_id
147     WHERE company_name = "Donec Ltd" AND declined = 0
148     GROUP BY iban;
```

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

iban	media_ventas
PT87806228135092429456346	42.82

12 17:18:36 SELECT iban, ROUND(AVG(amount),2) AS media_ventas FROM transactions LEFT JOIN credit_cards ON trans... 1 row(s) returned

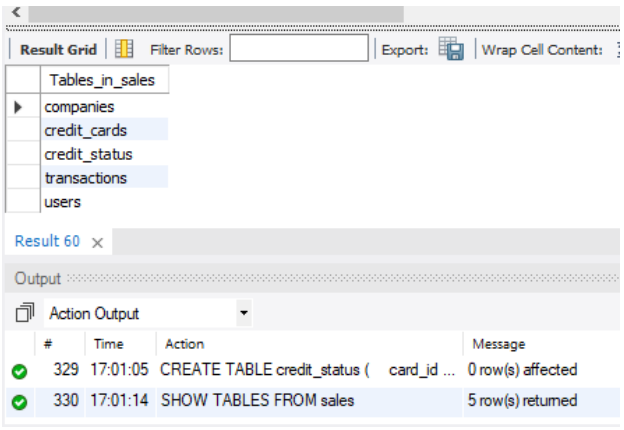
Nivell 2

Crea una nova taula que reflecteixi l'estat de les targetes de crèdit basat en si les últimes tres transaccions van ser declinades i genera la següent consulta:

Creamos la nueva tabla que es actualizable:
Creamos la nueva tabla llamada credit_status que muestre el estado como activa o bloqueada de las tarjetas basado a si en las últimas 3 transacciones han sido declinadas o no.
Utilizamos card_id tipo VARCHAR como Primary Key (mismo que el id de la tabla credit_cards con la que la relacionamos), el iban tipo VARCHAR como en la tabla credit_cards y creamos una columna nueva llamada status que podría ser tipo Boolean, pero hemos pensado que mejor tipo ENUM ya que permite mostrar elementos determinados, en este caso dos grupos: ‘activa’ para las tarjetas que han pasado el filtro como que no han sido declinadas en las últimas 3 transacciones y ’bloqueada’ para aquellas tarjetas que han pasado el filtro como que sí tienen sus últimas 3 transacciones declinadas.
Relacionamos la tabla 1 a N con la tabla credit_cards. Comprobamos que se ha realizado correctamente

```
CREATE TABLE credit_status (
   card_id VARCHAR(20) PRIMARY KEY,
   status ENUM('activa', ‘bloqueada’) NOT NULL,
   FOREIGN KEY (card_id) REFERENCES credit_cards(id));
```

```
174     -- creamos la tabla
175     CREATE TABLE credit_status (
176         card_id VARCHAR(20) PRIMARY KEY,
177         status ENUM('activa', 'bloqueada') NOT NULL);
178
179        -- Comprobamos que se ha creado correctamente
180         SHOW TABLES FROM sales;
```



Hacemos la consulta para identificar las transacciones declinadas e insertar en la tabla:
Insertamos en la tabla credit_status los datos en las columnas pertinentes: card_id y status que sacamos de la siguiente consulta y utilizando para insertar entre bloqueada y activa mediante el método CASE.
Para la consulta utilizamos DENSE_RANK() para ordenar las transacciones según el timestamp descendentemente y particionando por el card_id. A esto lo llamamos rango. A esta subconsulta le filtramos que seleccione las primeras 3. Al hacer el CASE filtramos los declinados donde cogemos la suma de los declinados del rango limitado a 3. Por lo que aquellos que tengan menos de 3 (3 transacciones últimas declinadas) son consideradas 'activa' en status. Aquellas card_id que tengan la suma de declinados = 3 serán entonces marcadas como 'bloqueada' en status.

```
INSERT INTO credit_status (card_id, status)
SELECT card_id,
CASE WHEN sum(declined) < 3
        THEN 'activa'
   ELSE 'bloqueada'
END AS status
FROM (SELECT transactions.card_id, transactions.timestamp, declined,
        DENSE_RANK() OVER (PARTITION BY transactions.card_id
                                        ORDER BY transactions.timestamp DESC) AS rango
        FROM transactions) AS consulta_rango
WHERE rango <= 3
GROUP BY card_id
ORDER BY card_id;
```

```sql
-- creamos la tabla
CREATE TABLE credit_status (
    card_id VARCHAR(20) PRIMARY KEY,
    status ENUM('activa', 'bloqueada') NOT NULL);

-- Comprobamos que se ha creado correctamente
SHOW TABLES FROM sales;

-- Insertar en la tabla los datos filtrados
INSERT INTO credit_status (card_id, status)
SELECT card_id,
CASE WHEN sum(declined) < 3
    THEN 'activa'
    ELSE 'bloqueada'
END AS status
FROM (SELECT transactions.card_id, transactions.timestamp, declined,
    DENSE_RANK() OVER (PARTITION BY transactions.card_id
                       ORDER BY transactions.timestamp DESC) AS rango
FROM transactions) AS consulta_rango
WHERE rango <= 3
GROUP BY card_id
ORDER BY card_id;
```

Output

Action Output

#	Time	Action	Message
143	17:24:42	INSERT INTO credit_status (card_id, iban, status) SELECT card_id, CASE WHEN sum(declined) < 3 THEN 'a...	Error Code: 1054. Unknown column 'iban' in 'field list'
144	17:25:09	INSERT INTO credit_status (card_id, status) SELECT card_id, CASE WHEN sum(declined) < 3 THEN 'activa' ...	275 row(s) affected Records: 275 Duplicates: 0 Warnings: 0

Comprobación:
Efectivamente, las 275 tarjetas que hay están activas.

```sql
                       ORDER BY transactions.timestamp DESC) AS rango
FROM transactions) AS consulta_rango
WHERE rango <= 3
GROUP BY card_id
ORDER BY card_id;

SELECT * FROM credit_status;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

card_id	status
CcU-2938	activa
CcU-2945	activa
CcU-2952	activa
CcU-2959	activa
CcU-2966	activa
CcU-2973	activa
CcU-2980	activa

credit_status 30

Output

Action Output

#	Time	Action	Message
144	17:25:09	INSERT INTO credit_status (card_id, status) SELECT card_id, CASE WHEN sum(declined) < 3 THEN 'activa' ...	275 row(s) affected Records
145	17:36:23	SELECT * FROM credit_status LIMIT 0, 1000	275 row(s) returned

Añadimos el Foreign Key
ALTER TABLE credit_status
ADD CONSTRAINT fk_credit_status_credit_cards FOREIGN KEY (card_id) REFERENCES credit_cards (id);

SHOW INDEXES FROM credit_status;

```
200     -- Añadimos el Foreign Key
201 •  ALTER TABLE credit_status
202     ADD CONSTRAINT fk_credit_status_credit_cards FOREIGN KEY (card_id) REFERENCES credit_cards (id);
203
204 •  SHOW INDEXES FROM credit_status;
205
```

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

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visib
credit_status	0	PRIMARY	1	card_id	A	0	NULL	NULL		BTREE			YES

Result 31

Output

Action Output

#	Time	Action	Message
146	17:44:12	ALTER TABLE credit_status ADD CONSTRAINT fk_credit_status_credit_cards FOREIGN KEY (card_id) REF...	275 row(s) affected Records: 275 Duplicates: 0 Warnings: 0
147	17:44:30	SHOW INDEXES FROM credit_status	1 row(s) returned

Exercici 1

Quantes targetes estan actives?

SELECT COUNT(*) AS tarjetas_activas
FROM credit_card_status
WHERE status = 'activa';

```
206     -- Ejercicio 1
207     -- Quantes targetes estan actives?
208 •  SELECT COUNT(*) AS targetas_activas
209     FROM credit_status
210     WHERE status = 'activa';
```

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

targetas_activas
275

Result 32

Output

Action Output

#	Time	Action	Message
147	17:44:30	SHOW INDEXES FROM credit_status	1 row(s) returned
148	17:45:48	SELECT COUNT(*) AS targetas_activas FROM credit_status WHERE status = 'activa' LIMIT 0, 1000	1 row(s) returned

Nivell 3

Crea una taula amb la qual puguem unir les dades del nou arxiu products.csv amb la base de dades creada, tenint en compte que des de transaction tens product_ids. Genera la següent consulta:

Primero crearemos la tabla products:

```
CREATE TABLE IF NOT EXISTS products (
    id INT PRIMARY KEY,
    product_name VARCHAR(50) NOT NULL,
    price VARCHAR(10) NOT NULL,
```

color VARCHAR(20),
weight FLOAT,
warehouse_id VARCHAR(10));

```
185     -- Primero crearemos la tabla products:
186 ●   CREATE TABLE IF NOT EXISTS products (
187         id INT PRIMARY KEY,
188         product_name VARCHAR(50) NOT NULL,
189         price VARCHAR(10) NOT NULL,
190         color VARCHAR(20),
191         weight FLOAT,
192         warehouse_id VARCHAR(10));
193
```

Output

Action Output

#	Time	Action	Message
95	12:23:15	SELECT transactions.timestamp, t...	587 row(s) returned
96	12:30:26	CREATE TABLE IF NOT EXISTS ...	0 row(s) affected

Insertamos los datos en la tabla products:
LOAD DATA INFILE 'C:/Users/Nicola Korff/Desktop/SQL/da/sprint_04/products.csv'
INTO TABLE products
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS;

```
202     -- Insertamos los datos en la tabla products:
203 ●   LOAD DATA LOCAL INFILE 'C:/Users/Nicola Korff/Desktop/SQL/da/sprint_04/products.cs
204     INTO TABLE products
205     FIELDS TERMINATED BY ','
206     ENCLOSED BY '"'
207     LINES TERMINATED BY '\n'
208     IGNORE 1 ROWS;
209
210 ●   SELECT * FROM products;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	product_name	price	color	weight	warehouse_id
1	Direwolf Stannis	$161.11	#7c7c7c	1	WH-4
2	Tarly Stark	$9.24	#919191	2	WH-3
3	duel tourney Lannister	$171.13	#d8d8d8	1.5	WH-2
4	warden south duel	$71.89	#111111	3	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

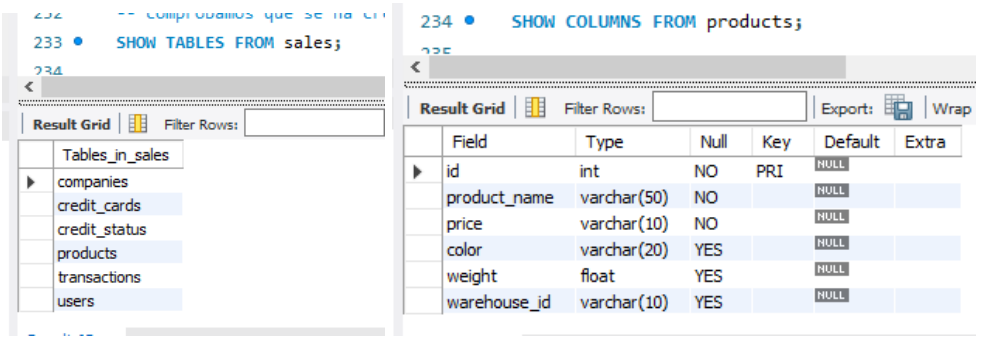
products 20 | Apply

Output

Action Output

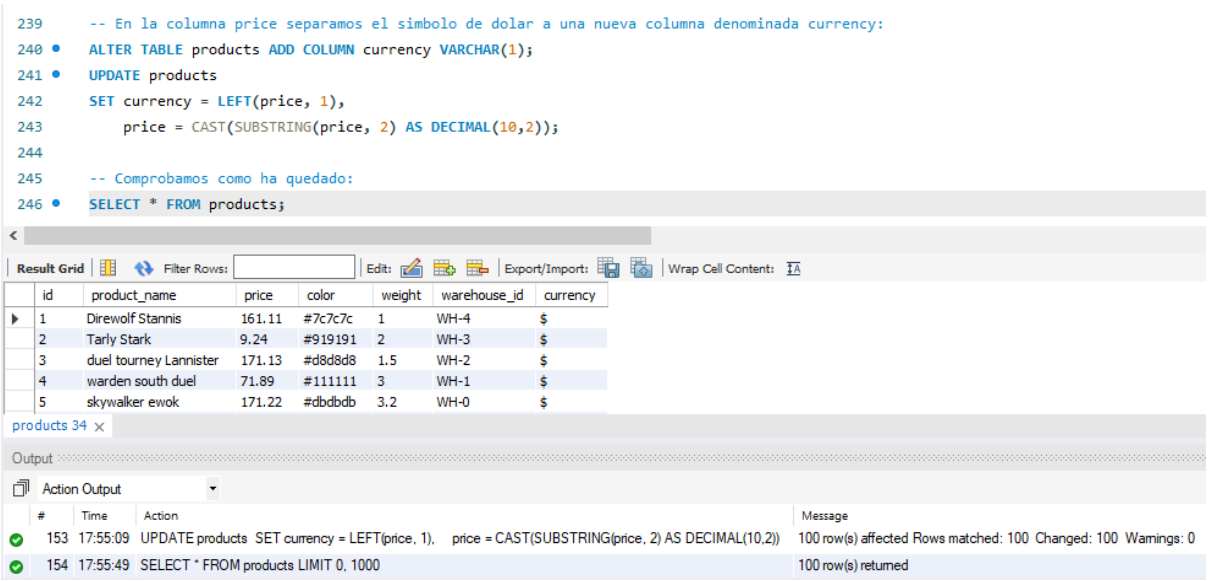
#	Time	Action	Message
103	12:39:29	LOAD DATA LOCAL INFILE 'C:/Use...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0
104	12:40:59	SELECT * FROM products LIMIT 0, ...	100 row(s) returned

Comprobamos que se ha creado correctamente:
SHOW TABLES FROM sales;
SHOW COLUMNS FROM products;



En la columna price separamos el símbolo de dolar a una nueva columna denominada currency:

```
ALTER TABLE products ADD COLUMN currency VARCHAR(1);
UPDATE products
SET currency = LEFT(price, 1),
    price = CAST(SUBSTRING(price, 2) AS DECIMAL(10,2));
```



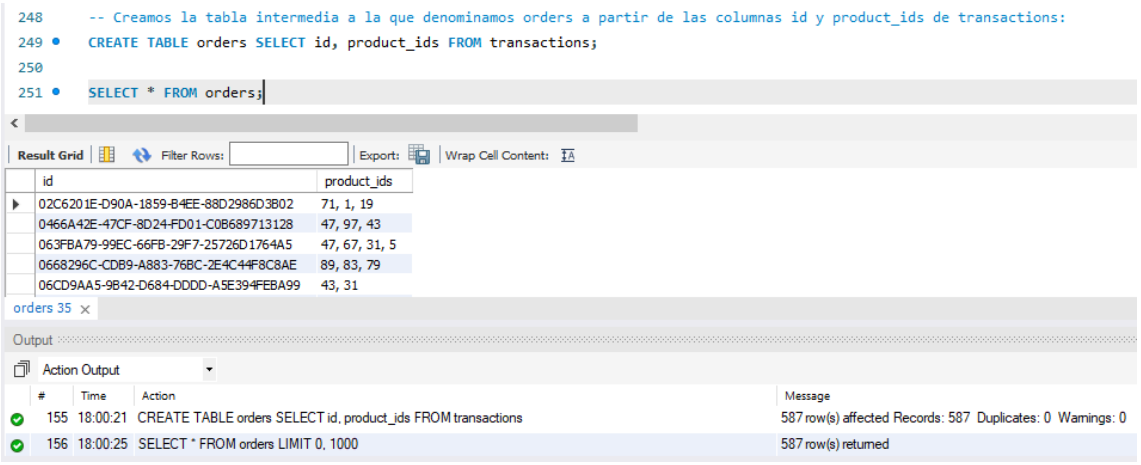
Para integrar los datos de products.csv en la base de datos y relacionarlos con transactions.product_ids, necesitamos una tabla intermedia para gestionar la relación entre transacciones y productos, ya que product_ids puede contener múltiples valores por transacción en un mismo campo.

Crearemos la tabla orders desde transacciones importando los el id de la transacción y los productos de la columna products_id:

CREATE TABLE orders SELECT id, products_id FROM transaction;

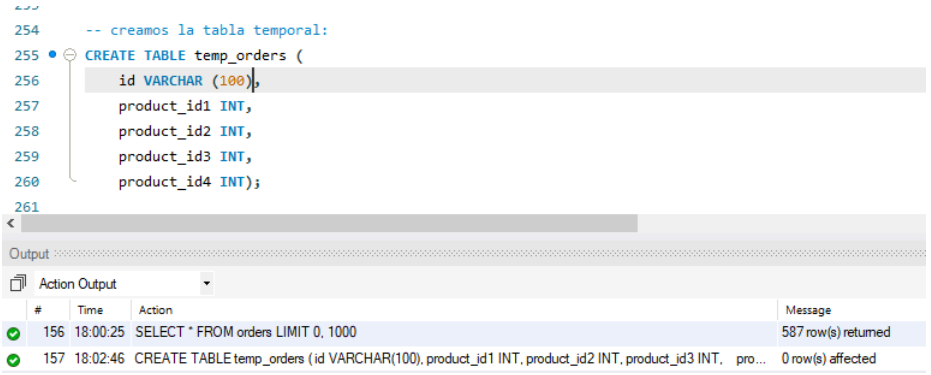
Comprobamos:

```
SELECT * FROM orders;
```



Creamos una tabla temporal a la que denominamos temp_orders donde podemos insertar los valores de los products_id separados por comas

```
CREATE TABLE temp_orders (
        id VARCHAR(100),
        product_id1 INT,
        product_id2 INT,
        product_id3 INT,
        product_id4 INT );
```



Insertamos los datos en la tabla temps_orders, extrayendo y separando los valores de la columna product_ids de la tabla orders que acabamos de crear, que contiene los ids de los productos separados por comas como en la tabla transactions. Esto lo haremos mediante la función SUBSTRING_INDEX doble. El último argumento de la función que va de 1 a 4 devuelve los elementos de la fila. 1 devuelve el primer elemento; 2 obtiene los primeros 2 elementos y con -1 tomamos el último de los que obtuvimos; lo mismo con 3 y 4. Es decir, cada fila de orders se transformará en una fila en temp_orders, donde los valores de product_ids se dividirán en hasta 4 columnas. En caso de no haber valor se convierte en NULL. Con CAST... AS UNSIGNED transformamos los valores extraídos de tipo VARCHAR y los convierte en números enteros (UNSIGNED) para que se almacenen correctamente en la tabla.

```
INSERT INTO temp_orders (id, product_id1, product_id2, product_id3, product_id4)
SELECT id,
   CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', 1), ',', -1) AS UNSIGNED) AS product_id1,
   CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', 2), ',', -1) AS UNSIGNED) AS product_id2,
```

CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', 3), ',', -1) AS UNSIGNED) AS product_id3,
CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', 4), ',', -1) AS UNSIGNED) AS product_id4
FROM orders;

```
262     -- insertamos los productos_id de la tabla
263 •   INSERT INTO temp_orders (id, product_id1, product_id2, product_id3, product_id4)
264     SELECT id,
265         CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', 1), ',', -1) AS UNSIGNED) AS product_id1,
266         CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', 2), ',', -1) AS UNSIGNED) AS product_id2,
267         CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', 3), ',', -1) AS UNSIGNED) AS product_id3,
268         CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', 4), ',', -1) AS UNSIGNED) AS product_id4
269     FROM orders;
270
271 •   SELECT * FROM temp_orders;
```

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

id	product_id1	product_id2	product_id3	product_id4
02C6201E-D90A-1859-B4EE-88D2986D3B02	71	1	19	19
0466A42E-47CF-8D24-FD01-C0B689713128	47	97	43	43
063FBA79-99EC-66FB-29F7-25726D1764A5	47	67	31	5
0668296C-CDB9-A883-76BC-2E4C44F8C8AE	89	83	79	79
06CD9AA5-9B42-D684-DDDD-A5E394FEBA99	43	31	31	31

temp_orders 36

Output

Action Output

#	Time	Action	Message
158	18:10:10	INSERT INTO temp_orders (id, product_id1, product_id2, product_id3, product_id4) SELECT id, CAST(SUB...	587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0
159	18:10:15	SELECT * FROM temp_orders LIMIT 0, 1000	587 row(s) returned

Una vez resuelta la separación de los valores, se deben traspasar todo a una mismo campo respetando sus ids, por lo que creamos una tabla auxiliar a la que denominamos trasp_order para traspasar los valores desde la tabla temp_orders. Luego eliminaremos la tabla orders para poder renombrar la tabla trasp_order como orders. Y finalmente eliminaremos también la tabla temporal.

Creamos la tabla de traspaso:

```
CREATE TABLE trasp_order (
        id VARCHAR(100),
   product_id INT);
```

Insertamos los valores desde la tabla temp_orders:

Para insertar los datos, normalizamos la tabla de multiples productos en diferentes columnas (product_id1, product_id2, product_id3, product_id4) y los convertimos en filas individuales dentro de la tabla trasp_order. Con el método UNION ALL combinamos las diferentes consultas.

```
INSERT INTO trasp_order (id, product_id)
SELECT id, product_id1
FROM temp_orders
WHERE product_id1 IS NOT NULL
UNION ALL
SELECT id, product_id2
FROM temp_orders
WHERE product_id2 IS NOT NULL
UNION ALL
SELECT id, product_id3
FROM temp_orders
WHERE product_id3 IS NOT NULL
```

UNION ALL
SELECT id, product_id4
FROM temp_orders
WHERE product_id4 IS NOT NULL;

```
275 • CREATE TABLE trasp_order (
276       id VARCHAR(100),
277       product_id INT);
278
279   -- Insertamos los valores:
280 • INSERT INTO trasp_order (id, product_id)
281   SELECT id, product_id1
282   FROM temp_orders
283   WHERE product_id1 IS NOT NULL
284   UNION ALL
285   SELECT id, product_id2
286   FROM temp_orders
287   WHERE product_id2 IS NOT NULL
288   UNION ALL
289   SELECT id, product_id3
290   FROM temp_orders
291   WHERE product_id3 IS NOT NULL
292   UNION ALL
293   SELECT id, product_id4
294   FROM temp_orders
295   WHERE product_id4 IS NOT NULL;
296
```

Output

Action Output

#	Time	Action	Message
161	18:16:02	CREATE TABLE trasp_order (id VARCHAR(100), product_id INT)	0 row(s) affected
162	18:16:26	INSERT INTO trasp_order (id, product_id) SELECT id, product_id1 FROM temp_orders WHERE product_id1 ...	2348 row(s) affected Records: 2348 Duplicates: 0 Warnings: 0

Comprobamos:
SELECT * FROM trasp_order;

```
297 • SELECT * FROM trasp_order;
```

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

id	product_id
02C6201E-D90A-1859-B4EE-88D2986D3B02	71
0466A42E-47CF-8D24-FD01-C0B689713128	47
063FBA79-99EC-66FB-29F7-25726D1764A5	47
0668296C-CDB9-A883-76BC-2E4C44F8C8AE	89
06CD9AA5-9B42-D684-DDDD-A5E394FEBA99	43

trasp_order 37 ×

Output

Action Output

#	Time	Action	Message
162	18:16:26	INSERT INTO trasp_order (id, product_id) SELECT id, product_id1 FROM temp_orders WHERE product_id1 ...	2348 row(s) affected F
163	18:17:04	SELECT * FROM trasp_order LIMIT 0, 1000	1000 row(s) returned

Eliminamos la tabla original "orders" la cual será reemplazada con "trasp_order"
DROP table orders;

Renombrar la tabla "trasp_order" a su nombre final "orders".
ALTER TABLE trasp_order RENAME orders;

Finalmente eliminamos la tabla temporal "temp_orders"
DROP TABLE temp_orders;

```
299     -- Borrar la tabla original "orders" la cual será reemplazada con "trasp_order"
300 •   DROP table orders;
301
302     -- Renombrar la tabla "trasp_order" a su valor final "orders".
303 •   ALTER TABLE trasp_order RENAME orders;
304
305     -- Borrar tabla temporal "temp_orders"
306 •   DROP TABLE temp_orders;
```

Output

Action Output

#	Time	Action	Message
163	18:17:04	SELECT * FROM trasp_order LIMIT 0, 1000	1000 row(s) returned
164	18:17:57	DROP table orders	0 row(s) affected
165	18:18:03	ALTER TABLE trasp_order RENAME orders	0 row(s) affected
166	18:18:09	DROP TABLE temp_orders	0 row(s) affected

Comprobamos cómo han quedado las tablas:

```
309 •   SHOW TABLES FROM sales;
310
```

Result Grid | Filter Rows: | Export:

Tables_in_sales
companies
credit_cards
credit_status
orders
products
transactions
users

Result 39

Mostramos cómo ha quedado
SELECT * FROM orders ORDER BY id;

```
311 •   SELECT * FROM orders ORDER BY id;
312
313     -- Le cambiamos el nombre a la columna, la vacia
```

Result Grid | Filter Rows: | Export: | Wrap C

id	product_id
02C6201E-D90A-1859-B4EE-88D2986D3B02	71
02C6201E-D90A-1859-B4EE-88D2986D3B02	1
02C6201E-D90A-1859-B4EE-88D2986D3B02	19
02C6201E-D90A-1859-B4EE-88D2986D3B02	19
0466A42E-47CF-8D24-FD01-C0B689713128	47
0466A42E-47CF-8D24-FD01-C0B689713128	43
0466A42E-47CF-8D24-FD01-C0B689713128	97
0466A42E-47CF-8D24-FD01-C0B689713128	43

orders 40

Output

Finalmente añadimos los Foreign Keys:
Creamos un índice de id en orders para vincular con la Foreign Key de la tabla transactions y el Primary Key de la tabla products.

ALTER TABLE orders
ADD CONSTRAINT fk_orders_products FOREIGN KEY (product_id) REFERENCES products (id),
ADD CONSTRAINT fk_orders_transactions FOREIGN KEY (id) REFERENCES transactions (id);

Comprobamos:

SHOW INDEXES FROM orders;

```
315     -- Creamos un índice de id en orders para vincular con la Foreign Key de la tabla transactions.
316 •  ALTER TABLE orders
317     ADD CONSTRAINT fk_orders_products FOREIGN KEY (product_id) REFERENCES products (id),
318     ADD CONSTRAINT fk_orders_transactions FOREIGN KEY (id) REFERENCES transactions (id);
319
320     -- Comprobamos
321 •  SHOW INDEXES FROM orders;
```

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

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Vis
orders	1	fk_orders_products	1	product_id	A	26	NULL	NULL	YES	BTREE			YES
orders	1	fk_orders_transactions	1	id	A	587	NULL	NULL	YES	BTREE			YES

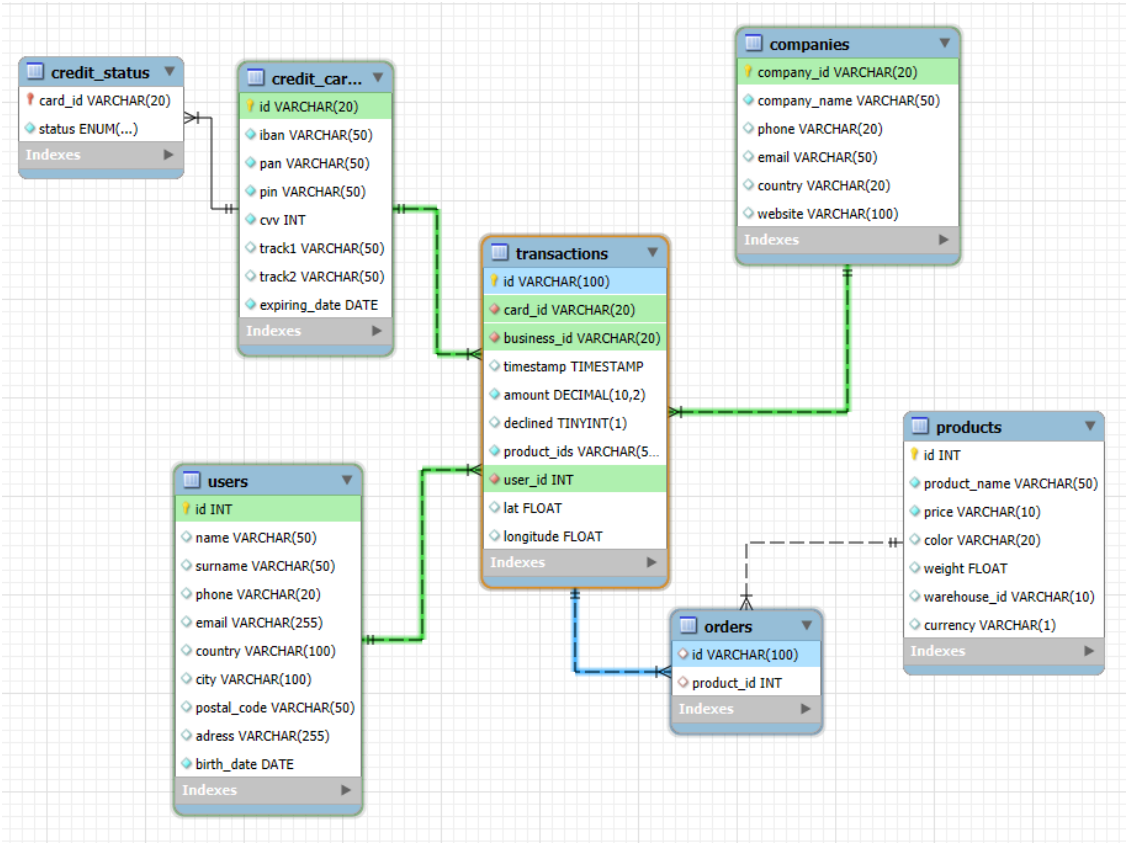
Result 41

Output

Action Output

#	Time	Action	Message
171	18:24:16	ALTER TABLE orders ADD CONSTRAINT fk_orders_products FOREIGN KEY (product_id) REFERENCES pr...	2348 row(s) affected Records: 2348 Duplicates: 0 Warnings: 0
172	18:24:21	SHOW INDEXES FROM orders	2 row(s) returned

ESQUEMA FINAL:

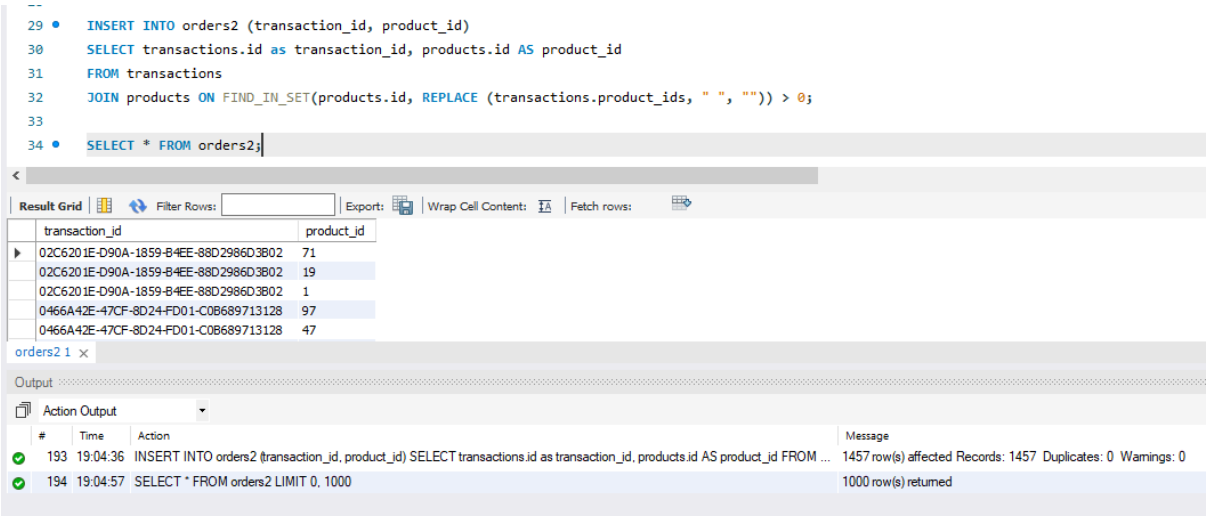


OPCIÓN 2 más sencilla:

```
CREATE TABLE IF NOT EXISTS orders2 (
        transaction_id VARCHAR(100) NOT NULL,
    product_id INT NOT NULL);
```

```
INSERT INTO orders2 (transaction_id, product_id)
SELECT transactions.id as transaction_id, products.id AS product_id
FROM transactions
JOIN products ON FIND_IN_SET(products.id, REPLACE (transactions.product_ids, " ", "")) > 0;

SELECT * FROM orders2;
```



Exercici 1

Necessitem conèixer el nombre de vegades que s'ha venut cada producte.

```
SELECT product_id, count(orders.product_id) AS cantidad_ventas
FROM transactions
INNER JOIN orders ON transactions.id = orders.id
WHERE transactions.declined = 0
GROUP BY product_id
ORDER BY cantidad_ventas DESC;
```