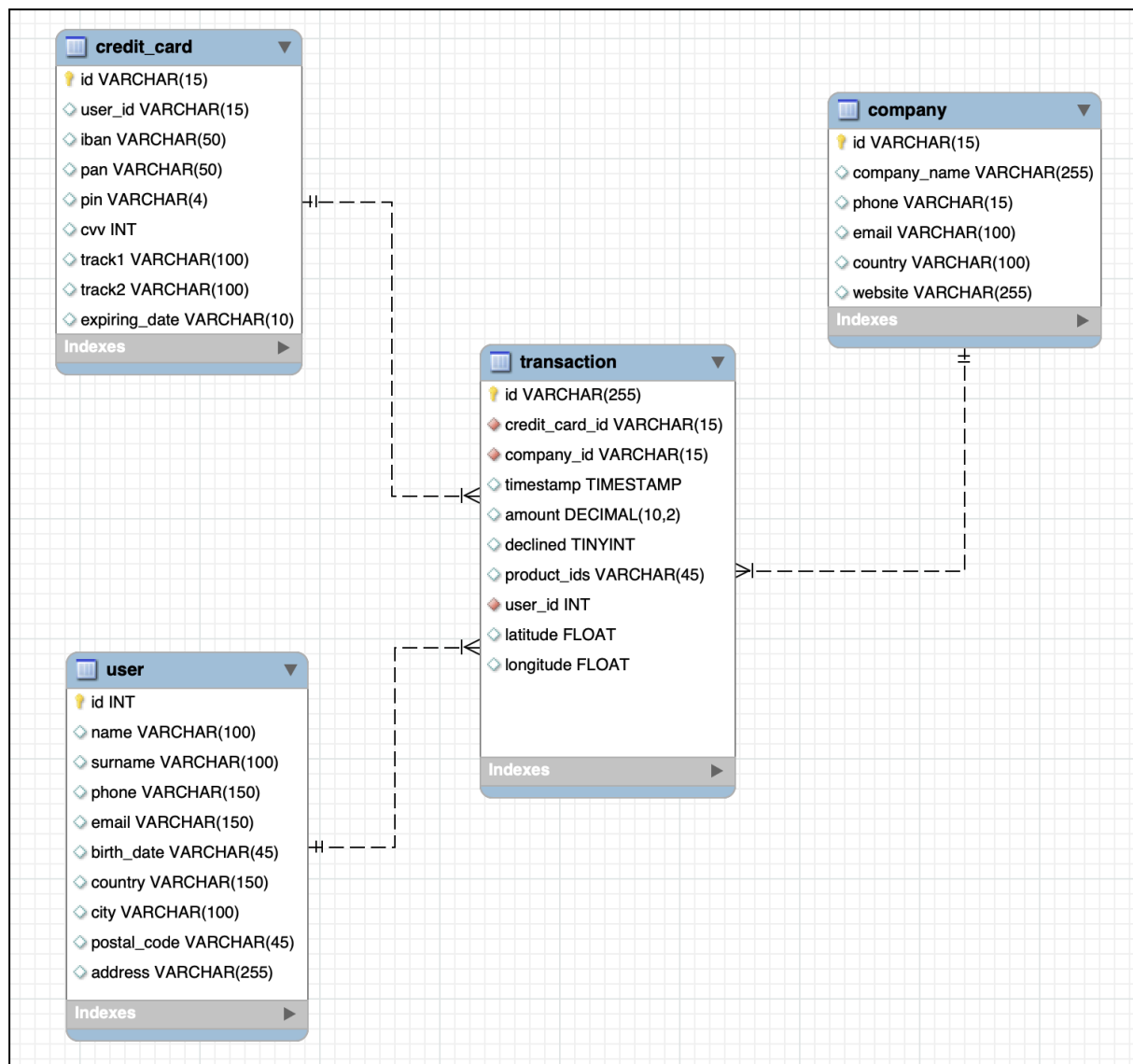


Tasca S4.01. Creació de Base de Dades

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 puguem realitzar les següents consultes:

Tras revisar los ficheros **transactions.csv**, **credit_cards.csv**, **companies.csv**, **users_ca.csv**, **users_uk.csv** y **users_usa.csv**, se diseña con la herramienta de diseño de MySQL Workbench un modelo de base de datos en estrella con la siguiente estructura:



En vez de usar la herramienta de ingeniería para crear la base de datos, vamos a crear todas las instancias y a cargar los ficheros por código.

Primero creamos la base de datos, intentando usar un conjunto de caracteres y colación más inclusivos posibles:

```
1 • CREATE DATABASE IF NOT EXISTS transactions_alex
2   DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci;
```

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|-----|----------|--|-------------------|------------------|
| ✓ 1 | 13:17:44 | CREATE DATABASE IF NOT EXISTS transaction... | 1 row(s) affected | 0.000 sec |

A continuació, creamos las tablas **company**, **credit_card** y **user** (nuestras **tablas de dimensiones**) para, posteriormente, crear la **tabla de hechos** **transaction** y las correspondientes claves foráneas.

- **credit_card_fk**, que relacionará la **id** de la tabla **credit_card** con el campo **credit_card_id** de la tabla **transaction** en una relación 1:n;
- **company_ik**, que relacionará la **id** de la tabla **company** con el campo **company_id** de la tabla **transaction** en una relación 1:n;
- **user_ik**, que relacionará la **id** de la tabla **user** con el campo **user_id** de la tabla **transaction** en una relación 1:n;

```
1 CREATE TABLE IF NOT EXISTS transactions_alex.company (
2   id VARCHAR(15) PRIMARY KEY,
3   company_name VARCHAR(255) NULL DEFAULT NULL,
4   phone VARCHAR(15) NULL DEFAULT NULL,
5   email VARCHAR(100) NULL DEFAULT NULL,
6   country VARCHAR(100) NULL DEFAULT NULL,
7   website VARCHAR(255) NULL DEFAULT NULL
8 );
```

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|-----|----------|--|-------------------|------------------|
| ✓ 1 | 13:31:10 | CREATE TABLE IF NOT EXISTS transactions_ale... | 0 row(s) affected | 0.047 sec |

(Antepongo el nombre de la base de datos a la tabla porque no será la primera vez que creo la tabla en la base de datos que no toca...)

```

1 CREATE TABLE IF NOT EXISTS transactions_alex.credit_card (
2     id VARCHAR(15) PRIMARY KEY,
3     user_id VARCHAR(15) NULL DEFAULT NULL,
4     iban VARCHAR(50) NULL DEFAULT NULL,
5     pan VARCHAR(50) NULL DEFAULT NULL,
6     pin VARCHAR(4) NULL DEFAULT NULL,
7     cvv VARCHAR(4) NULL DEFAULT NULL,
8     track1 VARCHAR(100) NULL DEFAULT NULL,
9     track2 VARCHAR(100) NULL DEFAULT NULL,
10    expiring_date VARCHAR(10) NULL DEFAULT NULL
11 );

```

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|--|-------------------|------------------|
| 1 | 13:41:38 | CREATE TABLE IF NOT EXISTS transactions_ale... | 0 row(s) affected | 0.125 sec |

```

1 CREATE TABLE IF NOT EXISTS transactions_alex.user (
2     id INT NOT NULL PRIMARY KEY,
3     name VARCHAR(100) NULL DEFAULT NULL,
4     surname VARCHAR(100) NULL DEFAULT NULL,
5     phone VARCHAR(150) NULL DEFAULT NULL,
6     email VARCHAR(150) NULL DEFAULT NULL,
7     birth_date VARCHAR(45) NULL DEFAULT NULL,
8     country VARCHAR(150) NULL DEFAULT NULL,
9     city VARCHAR(100) NULL DEFAULT NULL,
10    postal_code VARCHAR(45) NULL DEFAULT NULL,
11    address VARCHAR(255) NULL DEFAULT NULL
12 );

```

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|--|-------------------|------------------|
| 1 | 13:36:48 | CREATE TABLE IF NOT EXISTS transactions_ale... | 0 row(s) affected | 0.062 sec |

```
1 CREATE TABLE IF NOT EXISTS transactions_alex.transaction (  
2     id VARCHAR(255) PRIMARY KEY,  
3     credit_card_id VARCHAR(15) NOT NULL,  
4     company_id VARCHAR(15) NOT NULL,  
5     timestamp TIMESTAMP NULL DEFAULT NULL,  
6     amount DECIMAL(10,2) NULL DEFAULT NULL,  
7     declined TINYINT,  
8     product_ids VARCHAR(45) NULL DEFAULT NULL,  
9     user_id INT NOT NULL,  
10    latitude FLOAT NULL DEFAULT NULL,  
11    longitude FLOAT NULL DEFAULT NULL,  
12    FOREIGN KEY (company_id) REFERENCES company(id),  
13    FOREIGN KEY (credit_card_id) REFERENCES credit_card(id),  
14    FOREIGN KEY (user_id) REFERENCES user(id)  
15 );
```

Output

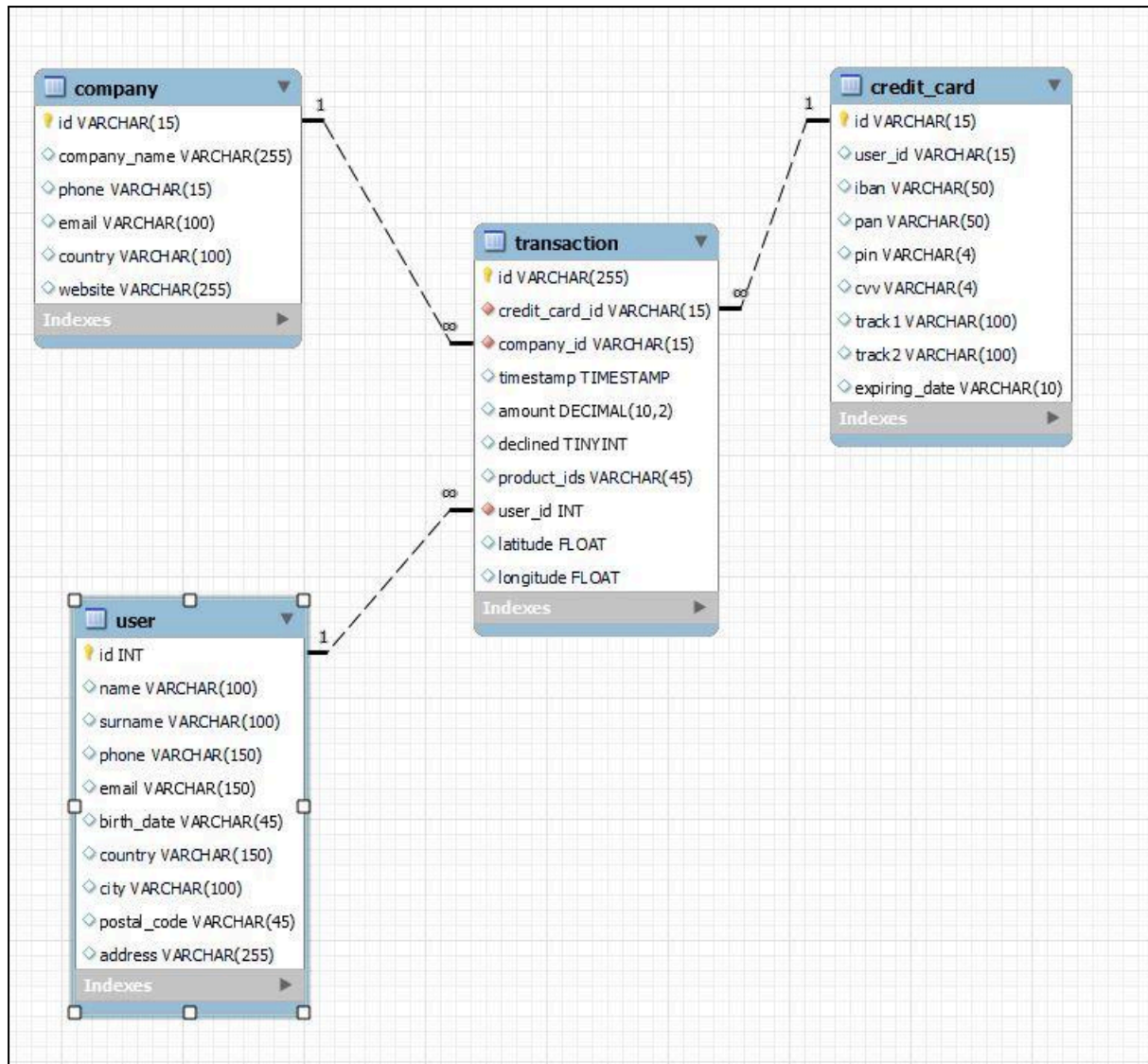
Action Output

| # | Time | Action | Message | Duration / Fetch |
|-----|----------|---|-------------------|------------------|
| ✓ 1 | 13:49:39 | CREATE TABLE IF NOT EXISTS transactions_alex.trans... | 0 row(s) affected | 0.109 sec |

Las métricas que recoge **transaction** son:

- **timestamp**: fecha y hora en que se realiza la transacción;
- **amount**: el importe de la transacción;
- **latitude**: la latitud geográfica desde donde se realiza la transacción;
- **longitude**: la longitud geográfica desde donde se realiza la transacción.

Ejecutamos la ingeniería inversa y comprobamos que hemos obtenido el modelo diseñado:

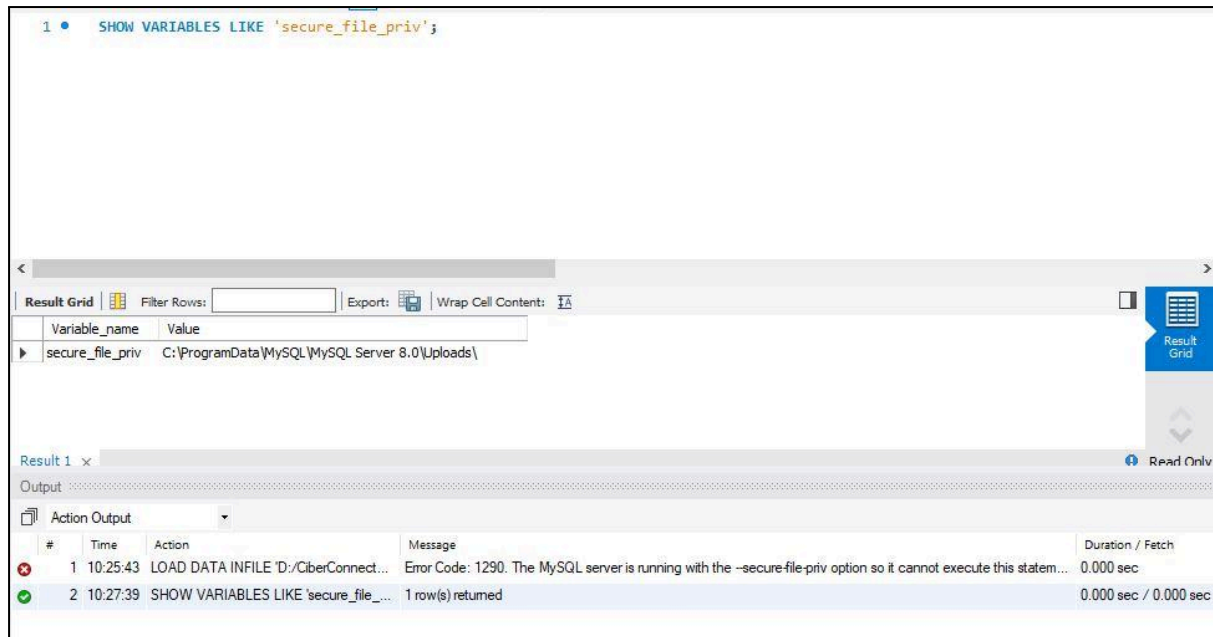


Pasamos a incorporar los datos de los ficheros a sus respectivas tablas. Hemos de tener en cuenta las características de cada fichero csv para parametrizar correctamente la sentencia de importación: separadores, finales de línea...

Siempre nos podemos encontrar con algún problema, como que tengas el parámetro de seguridad del motor de la base de datos activo (`--secure-file-priv`) y haya que desactivarlo. En este caso, vemos que este parámetro nos permite importar ficheros desde la ruta **C:\ProgramData\MySQL\MySQL Server 8.0\Uploads** del ordenador del aula:

(Hay que tener cuidado con la sintaxis de la sentencia; es mejor escribir **SHOW VARIABLES LIKE 'secure%'** que no equivocarse en la ortografía de la variable, como me había pasado: `secure-file-priv` no enseña ningún resultado y, claro, uno puede perder el tiempo en los archivos de configuración).

Buscamos dónde está el archivo de configuración del motor de base de datos:



Si no fuese así, intentamos localizar el fichero de configuración **my.ini** en esta carpeta en Windows, o **my.cnf** en el caso de Mac, buscamos el parámetro **secure_file_priv** (o lo añadimos, en caso de que no exista, en la sección **mysqld** del fichero) y le indicamos la dirección donde están guardados nuestros ficheros.

(Nota: Trabajando con Mac, la instalación de Workbench no crea ningún archivo de configuración; cuando se entra en la opción **Instance / Options File** de la pestaña **Administration**, Workbench crea un archivo **my.cnf** vacío en la ruta `/etc` que, para que nos funcione la importación, hay que rellenar con las siguientes líneas:

```
[mysqld]
local_infile=1
secure_file_priv=""
```

La primera línea nos permite importar archivos de la máquina, y la segunda línea nos sitúa en la raíz del Mac.)

Importaremos los archivos **.csv** mediante la instrucción **LOAD FILE INTO table**, teniendo en cuenta, para cada archivo, cuál es el carácter de separación, el de final de línea, y omitiendo la línea de encabezado.

Empezamos por la tabla **company**, importando el fichero **companies.csv**:

```

66  -- Importamos el fichero companies.csv a la tabla company
67  • LOAD DATA
68  INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/companies.csv'
69  INTO TABLE transactions_alex.company
70  FIELDS TERMINATED BY ','
71  LINES TERMINATED BY '\n'
72  IGNORE 1 LINES;

```

Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|---|--|------------------|
| 1 | 11:02:06 | LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Up... | 100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0 | 0.046 sec |

Importa 100 registros, que coincide con los que hay en el csv tras descartar el encabezado.

Seguimos con la importación de **credit_cards.csv** en la tabla **credit_card**:

```

1  • LOAD DATA
2  INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/credit_cards.csv'
3  INTO TABLE transactions_alex.credit_card
4  FIELDS TERMINATED BY ','
5  LINES TERMINATED BY '\n'
6  IGNORE 1 LINES;
7

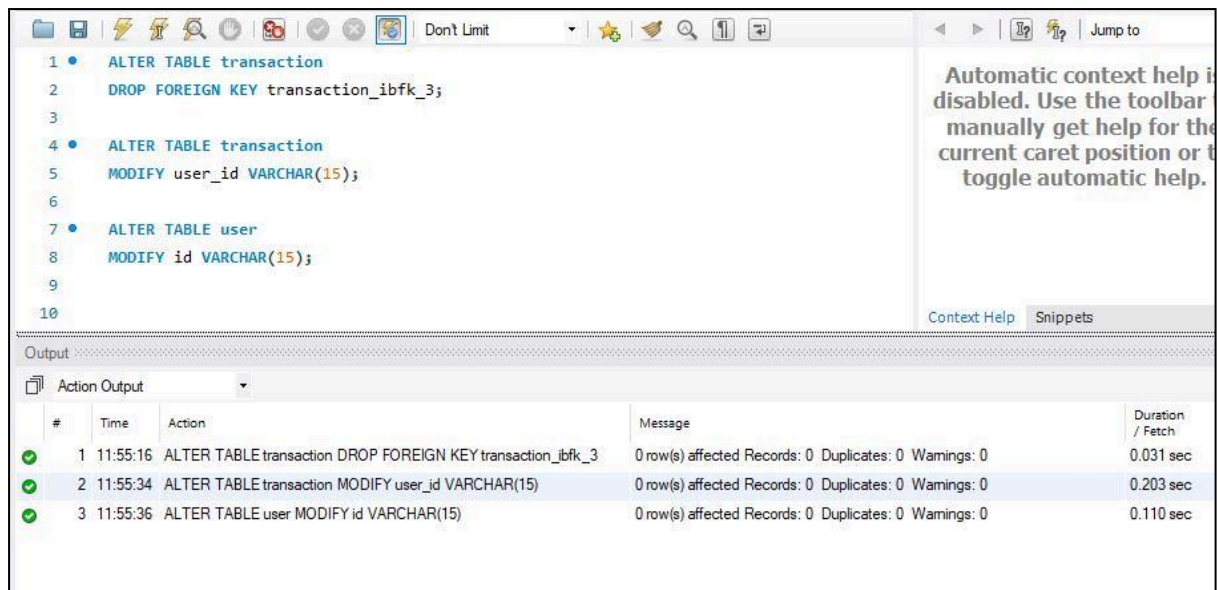
```

Output

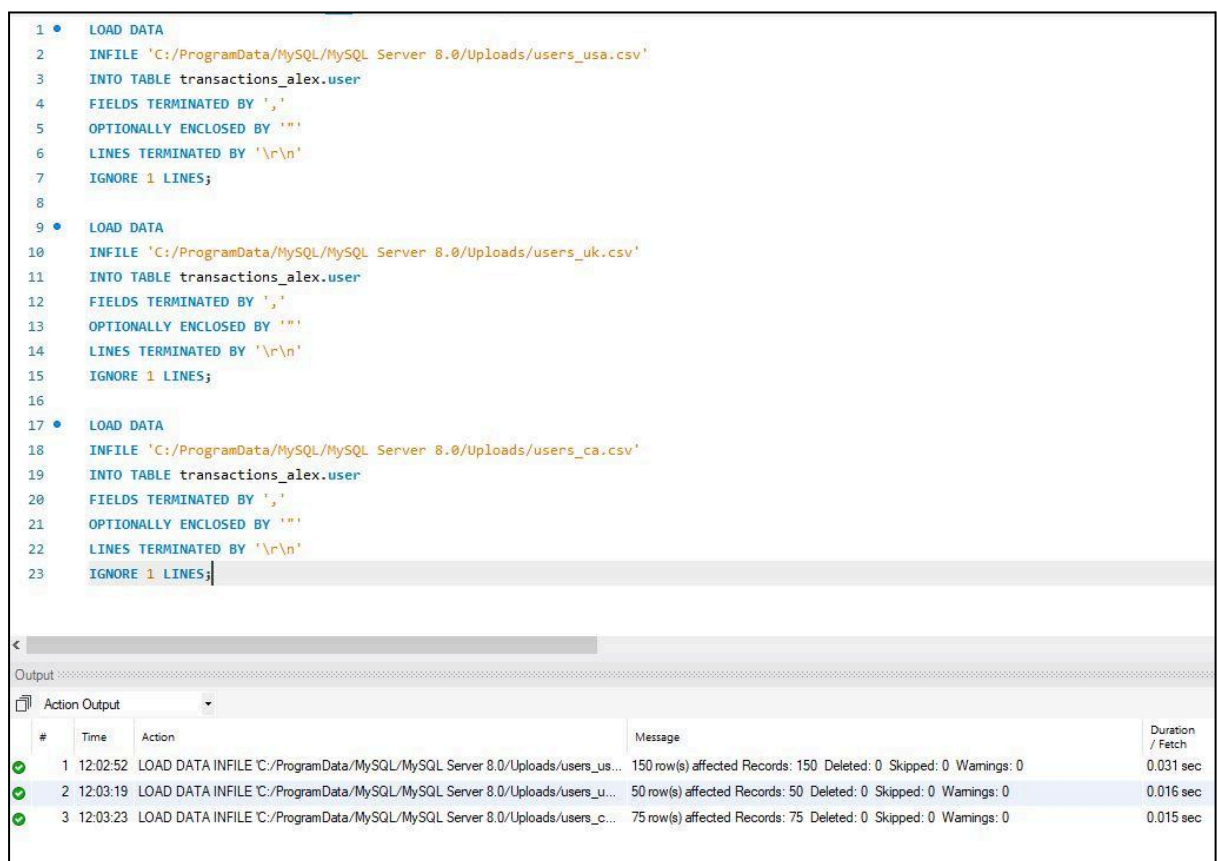
| # | Time | Action | Message | Duration / Fetch |
|---|----------|---|--|------------------|
| 1 | 11:08:34 | LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Up... | 275 row(s) affected Records: 275 Deleted: 0 Skipped: 0 Warnings: 0 | 0.063 sec |

275 registros, que coincide con los registros del fichero (exceptuando el encabezado).

Vamos con la tabla **users**. Aquí tenemos que incorporar los datos de tres ficheros: **users_usa.csv**, **users_uk.csv** y **user_ca.csv**. Me encuentro con que la importación falla porque, al campo **id** se le está pasando un carácter en vez de un número, así que cambio el tipo de dato de **id** de la tabla **user** y de **user_id** de la tabla **transaction**, eliminando previamente la clave foránea:



Tenemos que añadir la instrucción **ENCLOSED** para que la instrucción identifique los campos entrecomillados como un campo.



Verificamos con los archivos que el número de registros es correcto. Vamos a por la tabla **transaction**:


```
1 • LOAD DATA
2   INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/transactions.csv'
3   INTO TABLE transactions_alex.transaction
4   FIELDS TERMINATED BY ';'
5   LINES TERMINATED BY '\n'
6   IGNORE 1 LINES;
7
```

Output :

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|---|--|------------------|
| 1 | 12:11:55 | LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL... | 587 row(s) affected Records: 587 Deleted: 0 Skipped: ... | 0.188 sec |

La base de datos no está normalizada, pues en algunos registros de la tabla **transaction** observamos varios valores en el campo **product_ids**. De momento, guardamos los valores en formato texto en este campo y más adelante los exportaremos a otra tabla intermedia.

Recuperamos la clave foránea:

```

1 ALTER TABLE transaction
2 ADD CONSTRAINT FOREIGN KEY (user_id)
3 REFERENCES user(id);
4

```

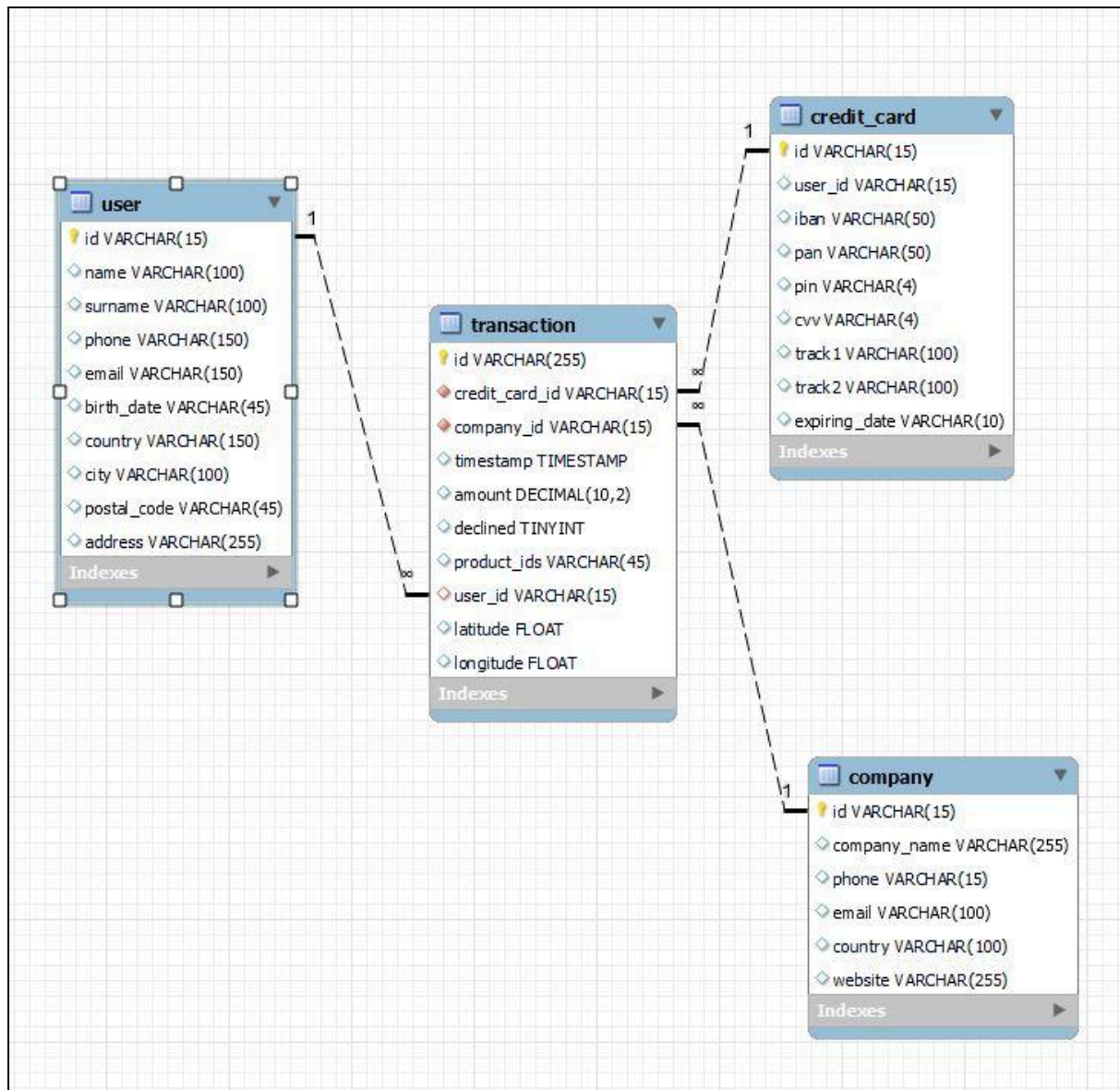
Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|---|---|------------------|
| 1 | 12:14:15 | ALTER TABLE transaction ADD CONSTRAINT FOR... | 587 row(s) affected Records: 587 Duplicates: 0 Wamin... | 0.328 sec |

La tabla **transaction** será nuestra **tabla de hechos**, y las tablas **user**, **company** y **credit_card** serán las **tablas de dimensiones**, que guardan los datos de los usuarios, empresas y tarjetas de créditos, respectivamente, de cada uno de los registros (transacciones) de nuestra tabla de hechos.

Así queda finalmente la base de datos:



- Exercici 1

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

CORRECCIÓ:

```
1 • SELECT u.name, u.surname, u.phone, u.email
2 FROM user u
3 WHERE u.id IN
4     (SELECT t.user_id
5      FROM transaction t
6      GROUP BY t.user_id
7      HAVING COUNT(t.id) > 30)
8 ORDER BY surname;
```

The screenshot shows a database interface with a query editor at the top and a results grid below. The query is a SQL statement that selects user information from the 'user' table, filtered by a subquery that identifies users with more than 30 transactions in the 'transaction' table. The results grid displays four rows of user data.

| | name | surname | phone | email |
|---|--------|---------|----------------|---------------------------------|
| ▶ | Hedwig | Gilbert | 064-204-8788 | sem.eget@icloud.edu |
| | Kenyon | Hartman | 082-871-7248 | convallis.ante.lectus@yahoo.com |
| | Ocean | Nelson | 079-481-2745 | aenean@yahoo.com |
| | Lynn | Riddle | 1-387-885-4057 | vitae.aliquet@outlook.edu |

Below the results grid, there is an 'Output' section showing the execution details of the query. It indicates that the query was executed at 12:03:28 and returned 4 rows.

| # | Time | Action | Message | Duration / Fetch |
|-----|----------|--|-------------------|-----------------------|
| ✓ 1 | 12:03:28 | SELECT u.name, u.surname, u.phone, u.email FR... | 4 row(s) returned | 0.000 sec / 0.000 sec |

En la subconsulta anidada más profunda identificamos los **user_id** que tienen más de 30 transacciones mediante la función de agregación **COUNT** por **id** de la tabla **transactions** agrupado por **user_id**. Sobre esta subconsulta realizamos otra consulta para quedarnos solo con los **user_id** que nos interesan, y el resultado de esta consulta lo usamos como filtro para obtener los datos de estos usuarios de la tabla **user**.

Obtenemos los usuarios que tienen más de 30 transacciones mediante una subconsulta en la que, mediante la sentencia **HAVING**, imponemos la condición de que tengan más de 30 registros en la tabla **transaction**. Así comprobamos que no es necesario que la función de agregación aparezca en la sentencia **SELECT**, que fue lo que me llevó a crear otra consulta anidada.

- Exercici 2

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

```

1 • SELECT cc.iban IBAN, ROUND(AVG(t.amount),2) Media
2 FROM transaction t
3 JOIN credit_card cc
4     ON cc.id = t.credit_card_id
5 JOIN company c
6     ON c.id = t.company_id
7 WHERE c.company_name = 'Donec Ltd'
8 GROUP BY t.credit_card_id;

```

| IBAN | Media |
|---------------------------|--------|
| PT87806228135092429456346 | 203.72 |

Result 7 x

Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|--|-------------------|-----------------------|
| 1 | 11:01:19 | SELECT cc.iban IBAN, ROUND(AVG(t.amount),2) Media FROM transact... | 1 row(s) returned | 0.000 sec / 0.000 sec |

Hacemos una **JOIN** entre la tabla **transaction** y **company**, filtrando la **id** de la empresa cuyo nombre sea 'Donec Ltd'; unimos estos resultados mediante otra **JOIN** con la tabla **credit_card**, uniendo por la **id** de las tarjetas de crédito que pudiera tener la empresa Donec Ltd. Calculamos la media y agrupamos por el identificador de la tarjeta de crédito, ya que cada **credit_card_id** tendrá asignado un único IBAN.

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:

Exercici 1

Quantes targetes estan actives?

CORRECCIÓ:

Primero creamos una tabla temporal a la que llamaremos **credit_card_ordered**, donde almacenaremos las transacciones agrupadas por tarjeta de crédito, las ordenaremos de reciente a antigua y les asignaremos un número de orden mediante la función **ROW_NUMBER**. A continuación, ejecutamos la consulta para confirmar que los registros que almacenamos cumplen con el formato que estamos buscando.

```

1 • CREATE TEMPORARY TABLE IF NOT EXISTS credit_card_ordered AS (
2     SELECT
3         id,
4         credit_card_id,
5         declined,
6         timestamp,
7         ROW_NUMBER() OVER(PARTITION BY credit_card_id ORDER BY timestamp DESC) AS numorden
8     FROM transaction
9 )
10
11 • SELECT * FROM credit_card_ordered;

```

| id | credit_card_id | declined | timestamp | numorden |
|--------------------------------------|----------------|----------|---------------------|----------|
| AD85A78A-8829-5746-93A0-8B7A792EBC18 | CcU-2938 | 0 | 2022-03-12 09:23:10 | 1 |
| F1A598A2-86C5-50A9-F1CE-FB1D69866C39 | CcU-2938 | 0 | 2022-03-09 20:53:59 | 2 |
| 55166D02-D74C-6A63-6C54-8678467649B4 | CcU-2938 | 0 | 2022-02-24 11:01:42 | 3 |
| 6575E457-EF8B-B50A-AD85-68ED6FE98BE1 | CcU-2938 | 0 | 2021-10-24 01:29:53 | 4 |
| C9185110-96F7-193C-D4B4-D1086A63744A | CcU-2938 | 0 | 2021-10-17 03:52:48 | 5 |
| E6873589-5339-4189-E984-9F4B54C844A4 | CcU-2938 | 0 | 2021-09-28 02:24:34 | 6 |
| 18B2472B-DA8B-36F7-B0F2-7DDA688B73D1 | CcU-2938 | 0 | 2021-09-24 08:33:44 | 7 |
| C3516B1E-4923-B2DF-E6E2-B07548480CCF | CcU-2938 | 0 | 2021-09-18 00:31:49 | 8 |
| 17561134-37A4-E316-2C3C-C5541EDAD761 | CcU-2938 | 0 | 2021-09-15 05:23:32 | 9 |
| 02C6201E-D90A-1859-B4EE-88D2986D3B02 | CcU-2938 | 0 | 2021-08-28 23:42:24 | 10 |
| AC777376-9329-BE65-285C-505D7C83A262 | CcU-2938 | 0 | 2021-07-27 17:14:52 | 11 |

| # | Time | Action | Message | Duration / Fetch |
|---|----------|---|--|-----------------------|
| 1 | 13:12:18 | CREATE TEMPORARY TABLE IF NOT EXISTS credit_card_ordered A... | 587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0 | 0.000 sec |
| 2 | 13:12:21 | SELECT * FROM credit_card_ordered | 587 row(s) returned | 0.016 sec / 0.000 sec |

A continuació, alimentamos la tabla **credit_card_status** con las tres últimas transacciones de cada tarjeta y discriminamos si la suma de **declined** es igual o inferior (bueno, diferente, ya que no va a superar nunca a tres) a tres, en cuyo caso se considerará **activa** o **inactiva**,

respectivamente. Añado datos de la tarjeta, como IBAN y nombre, que pueda resultarnos interesantes. Comprobamos los datos de la tabla.

```

1 • CREATE TABLE IF NOT EXISTS credit_card_status AS (
2     SELECT
3         cco.credit_card_id AS 'Id. tarjeta',
4         cc.iban,
5         u.name,
6         u.surname,
7         CASE
8             WHEN SUM(declined) = 3 THEN 'Inactiva'
9             ELSE 'Activa'
10        END AS Estado
11    FROM credit_card_ordered cco
12    JOIN credit_card cc ON cco.credit_card_id = cc.id
13    JOIN user u ON cc.user_id = u.id
14    WHERE numorden in (1,2,3)
15    GROUP BY credit_card_id
16 );
17
18 • SELECT * FROM credit_card_status;

```

| Id. tarjeta | iban | name | surname | Estado |
|-------------|-------------------------------|---------|-------------|--------|
| CcU-2938 | TR301950312213576817638661 | Kenyon | Hartman | Activa |
| CcU-2945 | DO26854763748537475216568689 | Jameson | Hunt | Activa |
| CcU-2952 | BG45IVQL52710525608255 | Hilary | Ferguson | Activa |
| CcU-2959 | CR7242477244335841535 | Hedwig | Gilbert | Activa |
| CcU-2966 | BG72LKTQ15627628377363 | Leandra | Cherry | Activa |
| CcU-2973 | PT87806228135092429456346 | Elton | Roberson | Activa |
| CcU-2980 | DE39241881883086277136 | Haley | Fitzpatrick | Activa |
| CcU-2987 | GE89681434837748781813 | Clark | Olson | Activa |
| CcU-2994 | BH62714428368066765294 | Ocean | Nelson | Activa |
| CcU-3001 | CY49087426654774581266832110 | Aiko | Chaney | Activa |
| CcU-3008 | LU507216693616119230 | Chloe | Keith | Activa |
| CcU-3015 | PS119398216295715968342456821 | Keiko | Guerra | Activa |
| CcU-3022 | GT91695162850556977423121857 | Ima | Hendricks | Activa |
| CcU-3029 | AZ62317413982441418123739746 | Brett | Kirby | Activa |

credit_card_status 7 x

Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|---|--|-----------------------|
| 1 | 13:18:20 | CREATE TABLE IF NOT EXISTS credit_card_status AS (SELECT cco.credit_card_id A... | 275 row(s) affected Records: 275 Duplicates: 0 Warnings: 0 | 0.062 sec |
| 2 | 13:18:39 | SELECT * FROM credit_card_status | 275 row(s) returned | 0.000 sec / 0.000 sec |

Contamos el número de tarjetas activas con la siguiente consulta:

```

1 • SELECT estado, COUNT(estado) 'Núm. tarjetas por estado'
2   FROM credit_card_status
3  GROUP BY estado;

```

Result Grid

| estado | Núm. tarjetas por estado |
|--------|--------------------------|
| Activa | 275 |

Result 13 x

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|--|-------------------|-----------------------|
| 1 | 13:48:06 | SELECT estado, COUNT(estado) 'Núm. tarjetas por estado' FROM credit_card_st... | 1 row(s) returned | 0.000 sec / 0.000 sec |

El número de tarjetas activas coincide con el número de registros en la tabla **credit_card**, con lo que todas las tarjetas que tenemos en la base de datos están activas. Para comprobar que esta consulta funciona en caso de tener una tarjeta inactiva, creamos una tarjeta nueva y tres transacciones rechazadas, y comprobamos que esta tarjeta queda excluida de la consulta.

```

1 • INSERT INTO credit_card
2   VALUES ('CcU-9999', '275', 'CE01234 5678 9012 3456', '0123 4567 8901 2345', 9999, 000, '%tarari', '%tarara', '12/28/28');
3
4 • INSERT INTO transaction
5   VALUES ('02C6201E-D90A-1859-B4EE-88D2986D3B03', 'CcU-9999', 'b-2422', '2024-10-12 00:00:00', 500.00, 1, '70', 275, 80, 80);
6 • INSERT INTO transaction
7   VALUES ('02C6201E-D90A-1859-B4EE-88D2986D3B04', 'CcU-9999', 'b-2422', '2024-10-12 00:00:00', 400.00, 1, '69', 275, 80, 80);
8 • INSERT INTO transaction
9   VALUES ('02C6201E-D90A-1859-B4EE-88D2986D3B05', 'CcU-9999', 'b-2422', '2024-10-12 00:00:00', 300.00, 1, '6', 275, 80, 80);
10
11 • SELECT COUNT(id) FROM credit_card;

```

Result Grid

| COUNT(id) |
|-----------|
| 276 |

Result 10 x

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|--|-------------------|-----------------------|
| 3 | 13:16:41 | INSERT INTO transaction VALUES (02C6201E-D90A-1859-B4EE-88D2986D3... | 1 row(s) affected | 0.000 sec |
| 4 | 13:16:41 | INSERT INTO transaction VALUES (02C6201E-D90A-1859-B4EE-88D2986D3... | 1 row(s) affected | 0.016 sec |
| 5 | 13:17:29 | SELECT COUNT(id) FROM credit_card | 1 row(s) returned | 0.000 sec / 0.000 sec |

Volvemos a crear las tablas **credit_card_ordered** y **credit_card_status**:

CORRECCIÓN:

```

1 • drop table credit_card_status;
2 • drop table credit_card_ordered;
3
4 • CREATE TEMPORARY TABLE IF NOT EXISTS credit_card_ordered AS (
5     SELECT
6         id,
7         credit_card_id,
8         declined,
9         timestamp,
10        ROW_NUMBER() OVER(PARTITION BY credit_card_id ORDER BY timestamp DESC) AS numorden
11    FROM transaction
12 ) ;

```

Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|---|--|------------------|
| 1 | 10:49:17 | DROP TABLE credit_card_status | 0 row(s) affected | 0.078 sec |
| 2 | 10:49:21 | CREATE TEMPORARY TABLE IF NOT EXISTS credit_card_ordered... | 0 row(s) affected, 1 warning(s): 1050 Table 'credit_card_ordered' already exists | 0.000 sec |
| 3 | 10:49:59 | DROP TABLE credit_card_ordered | 0 row(s) affected | 0.000 sec |
| 4 | 10:50:12 | CREATE TEMPORARY TABLE IF NOT EXISTS credit_card_ordered... | 590 row(s) affected Records: 590 Duplicates: 0 Warnings: 0 | 0.000 sec |

En la última acción comprobamos que tenemos tres registros más, correspondientes a las tres nuevas transacciones.

```

1 • CREATE TABLE IF NOT EXISTS credit_card_status AS (
2     SELECT
3         cco.credit_card_id AS 'Id. tarjeta',
4         cc.iban,
5         u.name,
6         u.surname,
7         CASE
8             WHEN SUM(declined) = 3 THEN 'Inactiva'
9             ELSE 'Activa'
10        END AS Estado
11    FROM credit_card_ordered cco
12    JOIN credit_card cc ON cco.credit_card_id = cc.id
13    JOIN user u ON cc.user_id = u.id
14    WHERE numorden in (1,2,3)
15    GROUP BY credit_card_id
16 ) ;

```

Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|--|--|------------------|
| 1 | 11:00:58 | CREATE TABLE IF NOT EXISTS credit_card_status AS (SELECT c... | 276 row(s) affected Records: 276 Duplicates: 0 Warnings: 0 | 0.078 sec |

Comprobamos que tenemos una línea más correspondiente a la nueva tarjeta de crédito.

Consultamos la tabla **credit_card_status**:

```

1  -- Comprobamos de nuevo los datos de la tabla credit_card_status
2  • SELECT * FROM credit_card_status;

```

| Id. tarjeta | iban | name | surname | Estado |
|-------------|------------------------------|----------|-----------|----------|
| CcU-9999 | CE01234 5678 9012 3456 | Kenyon | Hartman | Inactiva |
| CcU-4856 | TR373872558313545667124286 | Zeus | Gamble | Activa |
| CcU-4849 | SE2813123487163628531121 | Garrett | Mcconnell | Activa |
| CcU-4842 | SA2156708581957118818229 | Ciaran | Harrison | Activa |
| CcU-4835 | PT34592171131763200132583 | Howard | Stafford | Activa |
| CcU-4828 | BG11LMJ30149367569464 | Hayfa | Pierce | Activa |
| CcU-4821 | LT253147505686466784 | Joel | Tyson | Activa |
| CcU-4814 | MR4845282437847152280636374 | Rafael | Jimenez | Activa |
| CcU-4807 | LB19298318715580851625676971 | Nissim | Franks | Activa |
| CcU-4800 | SI97824334522161436 | Mannix | Mcdain | Activa |
| CcU-4793 | HU95215627749276573565556322 | Robert | Mccarthy | Activa |
| CcU-4786 | SI51703104173167515 | Joan | Baird | Activa |
| CcU-4779 | FI9109231810971761 | Benedict | Wheeler | Activa |

credit_card_status 8 x

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|----------------------------------|---------------------|-----------------------|
| 1 | 11:07:53 | SELECT * FROM credit_card_status | 276 row(s) returned | 0.000 sec / 0.000 sec |

Y comprobamos la cantidad de tarjetas activas e inactivas:

```

1  -- ¿Cuántas tarjetas están activas?
2  SELECT estado, COUNT(estado) 'Núm. tarjetas por estado'
3  FROM credit_card_status
4  GROUP BY estado;

```

| estado | Núm. tarjetas por estado |
|----------|--------------------------|
| Activa | 275 |
| Inactiva | 1 |

Result 9 x

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|---|-------------------|-----------------------|
| 1 | 11:10:28 | SELECT estado, COUNT(estado) 'Núm. tarjetas por estado' FROM c... | 2 row(s) returned | 0.000 sec / 0.000 sec |

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:

Exercici 1

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

Como decíamos, la base de datos no está normalizada porque el campo **products_ids** de la tabla **transaction** alberga varios valores. Para normalizarla, crearemos una tabla intermedia entre **transaction** y **product** que genere una línea por transacción y producto vendido en dicha transacción; es decir, si en una transacción se han vendido tres productos, tendremos tres líneas, una por producto, con el mismo id de la transacción.

Creemos la tabla **product** e importamos el fichero **products.csv**:

```

1  -- Creamos la tabla product
2
3  CREATE TABLE IF NOT EXISTS product (
4      id VARCHAR(15) PRIMARY KEY,
5      product_name VARCHAR(255),
6      price VARCHAR(10),
7      colour VARCHAR(10),
8      wheight DECIMAL(5,1),
9      warehouse_id VARCHAR(10)
10 );
11
12 -- Importamos el fichero products.csv a la tabla product
13
14 LOAD DATA
15 INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/products.csv'
16 INTO TABLE transactions_alex.product
17 FIELDS TERMINATED BY ','
18 LINES TERMINATED BY '\n'
19 IGNORE 1 LINES;
20

```

Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|--|--|------------------|
| 1 | 12:23:13 | CREATE TABLE IF NOT EXISTS product (id VARCHAR(15) ... | 0 row(s) affected | 0.062 sec |
| 2 | 12:23:13 | LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server... | 100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0 | 0.016 sec |

Una vez rellena tabla padre **products** con las características de los productos, crearemos una tabla intermedia, que llamaremos **transaction_products**, y que posteriormente poblaremos con tantas líneas por transacción como productos se hayan vendido en ella. Crearemos las claves foráneas hacia las tablas padre **transaction** y **product** para asegurarnos la integridad de los datos suministrados en **transaction_product**.

```
1 CREATE TABLE IF NOT EXISTS transaction_products (  
2     transaction_id VARCHAR(255),  
3     product_id VARCHAR(15),  
4     FOREIGN KEY (transaction_id) REFERENCES transaction(id),  
5     FOREIGN KEY (product_id) REFERENCES product(id)  
6 );
```

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|---|-------------------|------------------|
| 1 | 12:24:54 | CREATE TABLE IF NOT EXISTS transaction_products (trans... | 0 row(s) affected | 0.078 sec |

Para poder llenar esta tabla con n productos por cada transacción tendremos que extraer los números de producto del campo **product_ids** de la tabla **transaction**. Sabemos que en toda transacción se ha vendido como mínimo un producto, pero desconocemos cuántos productos se han vendido en cada transacción. Para saberlo, dado que los id de productos están separados por una coma, contaremos cuántas comas hay en cada cadena de caracteres; el número de registros será igual al número de comas +1. El cálculo de las comas lo realizaremos contando la longitud de cada registro en **product_ids** y le restamos la longitud de ese mismo registro sustrayéndole los caracteres ','.

Una vez hecho el cálculo, esa información la almacenaremos en una tabla temporal que llamaremos **transaction_numproducts**. Comprobamos a continuación que la tabla contiene los registros que buscamos:

1 • CREATE TABLE transaction_numproducts (
2 id VARCHAR(255),
3 product_ids VARCHAR(45),
4 NumProds INT
5)
6 SELECT
7 id,
8 product_ids,
9 LENGTH(product_ids) - LENGTH(REPLACE(product_ids, ',', '')) + 1 AS NumProds
10 FROM transaction;
11
12 • SELECT * FROM transaction_numproducts;

100% 39:12

Result Grid

Filter Rows: Search Export:

| | id | product_ids | NumProds |
|---|--------------------------------------|----------------|----------|
| ▶ | 02C6201E-D90A-1859-B4EE-88D2986D3B02 | 71, 1, 19 | 3 |
| | 0466A42E-47CF-8D24-FD01-C0B689713128 | 47, 97, 43 | 3 |
| | 063FBA79-99EC-66FB-29F7-25726D1764A5 | 47, 67, 31, 5 | 4 |
| | 0668296C-CDB9-A883-76BC-2E4C44F8C8AE | 89, 83, 79 | 3 |
| | 06CD9AA5-9B42-D684-DDDD-A5E394FEBA99 | 43, 31 | 2 |
| | 07A46D48-31A3-7E87-65B9-0DA902AD109F | 47, 23 | 2 |
| | 09DE92CE-6F27-2BB7-13B5-9385B2B3B8E2 | 67, 7 | 2 |
| | 0A476ED9-0C13-1962-F87B-D3563924B539 | 29, 41, 11 | 3 |
| | 0BEB80B7-9D66-1707-CE4B-9DC7E71914B5 | 19, 41, 29, 3 | 4 |
| | 0C7C3A33-9947-3BC1-846D-7BE3D0D17598 | 89, 31 | 2 |
| | 0CE957A6-CCAA-2B7A-6839-8A4B1B324853 | 83, 43, 73, 61 | 4 |
| | 0DD2E608-5C9E-D1B3-4999-B99F43AD735A | 7, 47, 17 | 3 |
| | 1017AA59-3D5F-7A4C-1992-D151A8D1FA0A | 37, 13 | 2 |
| | 1026DA24-8929-31F1-8250-D7BA805C13D2 | 89, 11, 97, 79 | 4 |
| | 108B1D1D-5B23-A76C-55EF-C568E49A05DD | 59 | 1 |
| | 10A9B07A-810C-76EB-4D15-12C6CC128037 | 43, 83 | 2 |
| | 11ABED97-EA12-1B9A-96F0-A93ACC172179 | 29 | 1 |
| | 122DC333-E19F-D629-DCD8-9C54CF1EBB9A | 1, 67, 19 | 3 |
| | 133B82CC-DE62-8604-2D11-3DC5449E0A5F | 29 | 1 |
| | 135267BA-2E7D-957C-C42C-6450A2B3ED54 | 11, 71 | 2 |
| | 13DCC69F-EA07-E52B-8309-D474C6281E80 | 97, 29, 23 | 3 |
| | 13FBB312-B283-7976-DA47-14DE5986218A | 11, 29, 43, 79 | 4 |

transaction_numproducts 3 Read Only

Action Output

| | Time | Action | Response | Duration / Fetch Time |
|-----|----------|--|--------------------------|-------------------------|
| ✓ 1 | 21:35:31 | CREATE TABLE transaction_numproducts (id VARCHAR(255), product_ids VARCHAR(45), NumProds INT... | 587 row(s) affected R... | 0.016 sec |
| ✓ 2 | 21:35:54 | SELECT * FROM transaction_numproducts LIMIT 0, 10000 | 587 row(s) returned | 0.0011 sec / 0.00036... |

CORRECCIÓN:

Aprovecho esta tabla temporal para crear otra tabla temporal, **numcontrol**, que listará las diferentes cantidades de productos que se venden por transacción. Este listado nos será útil para la posterior extracción de productos en la tabla de desambiguación.

```

1 • CREATE TEMPORARY TABLE IF NOT EXISTS numcontrol
2   SELECT DISTINCT(NumProds) num
3   FROM transaction_numproducts
4   ORDER BY num ASC;
5
6   -- Mostramos los datos de esta nueva tabla de control
7 • SELECT * FROM numcontrol;
    
```

The screenshot shows a database interface with a 'Result Grid' displaying the following data:

| num |
|-----|
| 1 |
| 2 |
| 3 |
| 4 |

The 'Action Output' section shows the following results:

| # | Time | Action | Message | Duration / Fetch |
|---|----------|--|--|-----------------------|
| 1 | 13:48:13 | CREATE TEMPORARY TABLE IF NOT EXISTS numcontrol SELEC... | 4 row(s) affected Records: 4 Duplicates: 0 Warnings: 0 | 0.000 sec |
| 2 | 13:48:13 | SELECT * FROM numcontrol | 4 row(s) returned | 0.000 sec / 0.000 sec |

Mantenemos las comprobaciones sobre la tabla **transaction_numproducts**:

Vemos que el número máximo de productos por transacción es 4, tal como podemos ver también en la visualización de la tabla **numcontrol**:

```

1 • SELECT MAX(NumProds) AS 'Máx. prods. por transacción'
2   FROM transaction_numproducts;
    
```

The screenshot shows a database interface with a 'Result Grid' displaying the following data:

| Máx. prods. por transacc... |
|-----------------------------|
| 4 |

The 'Action Output' section shows the following results:

| # | Time | Action | Response | Duration / Fetch Time |
|---|----------|---|-------------------|------------------------|
| 1 | 22:10:18 | SELECT MAX(NumProds) AS 'Máx. prods. por transacción' FROM transaction_numpr... | 1 row(s) returned | 0.00058 sec / 0.000... |

Y el número de productos vendidos en total será:

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

```
1 SELECT SUM(NumProds) AS 'Total productos'
2 FROM transaction_numproducts;
```

Below the query area, there is a toolbar with options like 'Result Grid', 'Filter Rows', 'Search', and 'Export'. The 'Result Grid' is selected, and it displays a table with one column, 'Total product...', and one row with the value '1457'. To the right of the table, there are icons for 'Result Grid', 'Form Editor', and 'Field Types'. Below the table, there is a 'Result 11' tab and a 'Read Only' button. At the bottom, there is an 'Action Output' section with a table showing the execution details:

| | Time | Action | Response | Duration / Fetch Time |
|-----|----------|--|-------------------|-------------------------|
| ✓ 1 | 22:11:34 | SELECT SUM(NumProds) AS 'Total productos' FROM tran... | 1 row(s) returned | 0.00067 sec / 0.0000... |

La idea de nuestra consulta es seleccionar el primer producto del campo **products_id** y emparentarlo con su correspondiente **transaction_id**. La extracción del campo la conseguimos aplicando la función **SUBSTRING_INDEX** hasta la primera ocurrencia del signo de separación (coma):

SUBSTRING_INDEX(product_ids, ',', 1)

Una vez hemos introducido en **transaction_product** el primer (o único) producto del campo **product_ids** de cada transacción, preparamos una segunda consulta en la que aumentamos el parámetro de ocurrencia del separador para ampliar así la cadena de texto hasta el segundo (tercer y cuarto) producto y, a esa cadena, le aplicamos otro **SUBSTRING_INDEX()** para extraer el id de producto que queda a la derecha de la cadena de texto, cosa que conseguimos cambiando el parámetro de ocurrencia a -1.;

Posteriormente, seleccionaremos el segundo producto de **products_id** para aquellos registros cuyo **NumProd > 1**, esto lo hacemos aplicando la función hasta la segunda coma y, sobre esta cadena (que contiene todos los productos hasta la segunda coma), le aplicamos de nuevo la función, pero recortando desde la derecha hasta la primera coma que se encuentra, es decir:

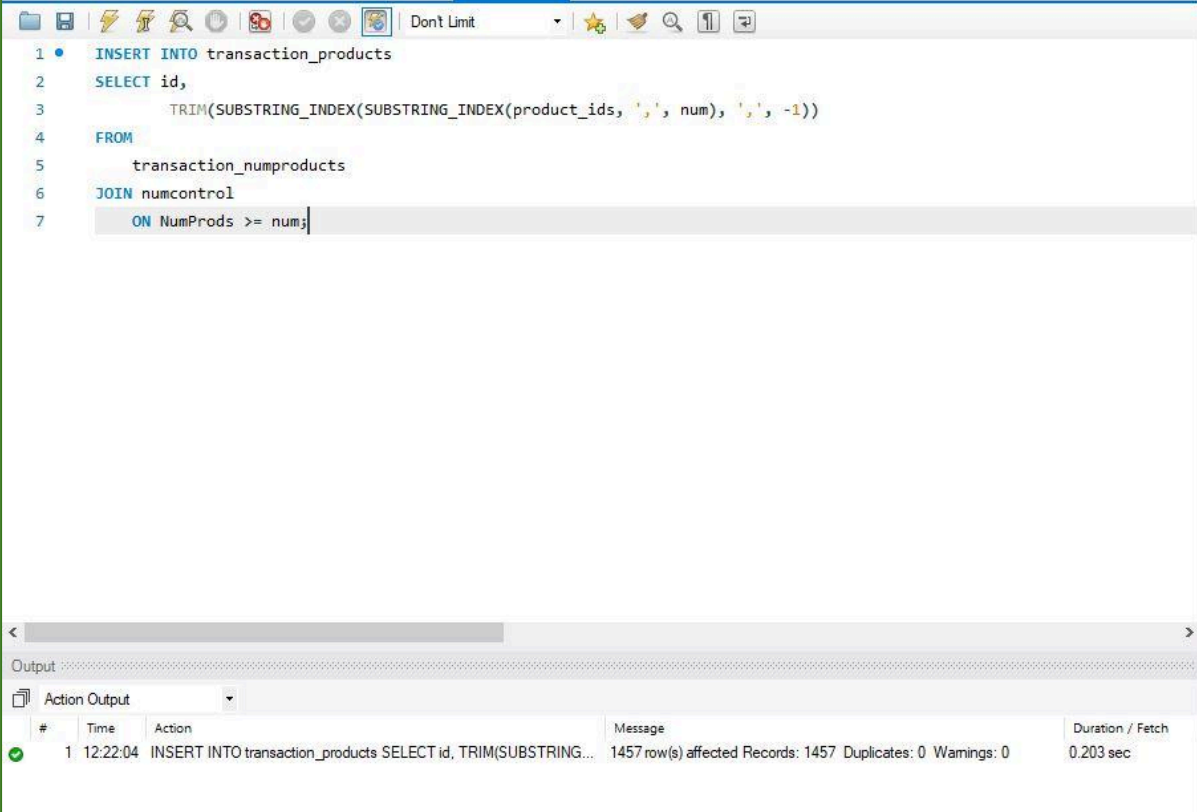
SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', 2), ',', -1)

a continuación, el tercer producto de **products_id** para los registros con **NumProd > 2**, que, aplicando el mismo procedimiento, obtendremos así:

SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', 3), ',', -1)

Así hasta llegar al máximo número de productos; esto es, hasta **numcontrol.num = 4** en nuestro caso.

Si ejecutamos esta consulta y la insertamos en la tabla **transaction_products**, la sintaxis sería la siguiente:



```
1 • INSERT INTO transaction_products
2   SELECT id,
3          TRIM(SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', num), ',', -1))
4   FROM
5     transaction_numproducts
6  JOIN numcontrol
7    ON NumProds >= num;
```

Output

| # | Time | Action | Message | Duration / Fetch |
|---|----------|---|--|------------------|
| 1 | 12:22:04 | INSERT INTO transaction_products SELECT id, TRIM(SUBSTRING... | 1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0 | 0.203 sec |

Ejecutamos una consulta sobre la tabla **transaction_products** y comprobamos que el número de registros en la tabla coincide con el número de productos vendidos que calculamos anteriormente.

Para contestar a la pregunta del ejercicio, contamos cuántas líneas de transacciones, agrupado por producto, hay en la tabla **transaction_products**, excluyendo las transacciones rechazadas:

1 • SELECT
2 tp.product_id Id,
3 p.product_name Producto,
4 p.colour Color,
5 p.price Precio,
6 COUNT(tp.transaction_id) NumVentas
7 FROM
8 transaction_products tp
9 JOIN
10 product p ON p.id = tp.product_id
11 JOIN
12 transaction t ON t.id = tp.transaction_id
13 WHERE
14 t.declined = 0
15 GROUP BY Id
16 ORDER BY NumVentas DESC;

Result Grid
Filter Rows:
Export:
Wrap Cell Content:
Id Producto Color Precio NumVentas
23 riverlands north #545454 \$169.96 60
67 Winterfell #1c1c1c \$195.94 59
2 Tarly Stark #919191 \$9.24 56
17 skywalker ewok sith #7c7c7c \$91.89 54
43 duel #5b5b5b \$59.80 54
97 jinn Winterfell #bababa \$65.25 53
79 Direwolf riverlands the #b2b2b2 \$132.86 52
1 Direwolf Stannis #7c7c7c \$161.11 51
13 palpatine chewbacca #2b2b2b \$139.59 51
47 Tully #919191 \$82.15 50
61 Winterfell Lannister #848484 \$28.01 50
41 Lannister Barratheon #f9f9f9 \$141.01 48
Result 44 x
Output
Action Output
Time Action Message Duration / Fetch
1 12:59:57 SELECT tp.product_id Id, p.product_name Producto, p.colour Color, ... 26 row(s) returned 0.000 sec / 0.000 s

Agrupo por **product_id** y le añado la descripción de color y precio de cada **product_id**, ya que hay varios productos con el mismo nombre y color diferente.

Comprobamos que no todos los productos se han vendido, y que los productos que ofrece nuestra empresa se centran en dos sagas de fantasía y ciencia ficción como *Juego de tronos* y *Star Wars* que tantas alegrías (y algún disgusto) me han dado :)

Para acabar, echemos un último vistazo a la estructura de la base de datos, incluyendo la tabla **product**, la tabla intermedia y las auxiliares:

