Tasca S4.01. Creació de bases 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

Creación de la base de datos

- Creamos una base de datos llamada “operativa”

```
CREATE DATABASE operativa;
```

- Seleccionamos la base de datos para trabajar en ella y añadir las tablas

```
USE operativa;
```

Creación de las tablas

- Creamos la tabla “companies”
  - Identificamos el campo “company”_id” como Primary Key o PK
  - Marcamos el campo “email” como UNIQUE al tratarse de un campo único que no puede tener duplicados, ya que cada dirección de correo

debe ser única. Además, al indexarse automáticamente también optimizamos el rendimiento de la base de datos al realizar búsquedas a través de estos campos.

    - El campo “phone” también podría ser un campo UNIQUE, pero al haber priorizado el campo anterior no es recomendable tener varios INDEX en la misma tabla al poder verse afectada la optimización de la base de datos.

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

- Creamos la tabla “credit_cards”
    - Identificamos “id” como Primary Key de la tabla
    - Marcamos el campo iban como UNIQUE

```
CREATE TABLE IF NOT EXISTS credit_cards (
    id varchar(20),
  user_id int,
  iban varchar(50) UNIQUE,
  pan varchar(4),
  pin varchar(4),
  cvv int,
  track1 varchar(100),
  track2 varchar(100),
  expiring_date varchar(20),
```

```
  PRIMARY KEY(id),
  INDEX(user_id)
  );
```

- Creamos la tabla “users”
    - Tenemos 3 archivos .csv de usuarios diferentes y cada uno corresponde a un país diferente (Canada, United Kingdom y United States). Al conservar la misma estructura de columnas, decidimos crear una única tabla en la cual insertaremos los datos de los tres archivos en lugar de crear tres tablas diferentes.
    - Asignamos el campo “id” como Primary Key.

```
 CREATE TABLE IF NOT EXISTS users (
     id int,
  name varchar(25),
  surname varchar (50),
  phone varchar (20),
  email varchar(75),
  birth_date varchar(20),
  country varchar (25),
  city varchar(25),
  postal_code varchar(20),
  address varchar(125),
  PRIMARY KEY (id),
 );
```

- Creamos la tabla “transactions”
    - Asignamos el campo “id” como Primary Key
    - Al tratarse de la tabla de hechos de la base de datos, creamos relaciones con las otras tablas mediante diversas Foreign Key, una por

cada tabla de dimensiones, que las vincula con su Primary Key en cada una de esas tablas.

```
CREATE TABLE IF NOT EXISTS transactions (
    id varchar(255),
    card_id varchar(20),
    business_id varchar(20),
    timestamp timestamp,
    amount decimal(10,2),
    declined tinyint(1),
    product_ids int,
    user_id int,
    lat decimal(10,14),
    longitude decimal(10,14),
    PRIMARY KEY (id),
);
```

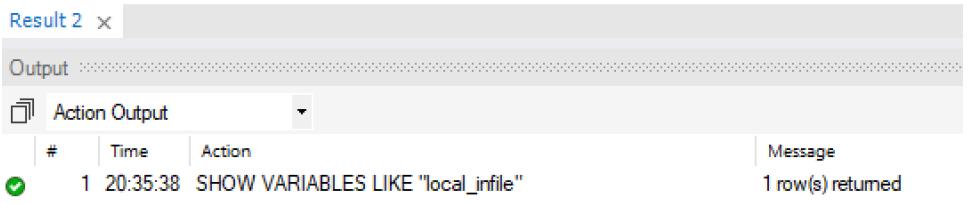
Activación de la carga local

Para poder cargar un archivo .csv en local, hemos de verificar si la opción de carga local está activada. Al realizar la comprobación, aparece como OFF, por lo que procederemos a su activación

```
SHOW VARIABLES LIKE "local_infile";  -- Comprobamos si la carga local está a

SET GLOBAL local_infile=1;
```

	Variable_name	Value
▶	local_infile	ON



Ahora tenemos la carga local activada a nivel general, pero también hemos de activarla en la propia base de datos y en MySQL Server.

Activación en la base de datos

1. En el menú superior de Workbench, vamos a “DATABASE” y después a “MANAGE CONNECTIONS”
2. Seleccionamos “Local Instance MySQL 80”
3. En la pestaña “advanced”, añadimos la siguiente línea:

   OPT_LOCAL_INFILE=1

Ya podemos proceder a insertar los registros para cada una de nuestras tablas.

Carga de registros en las tablas

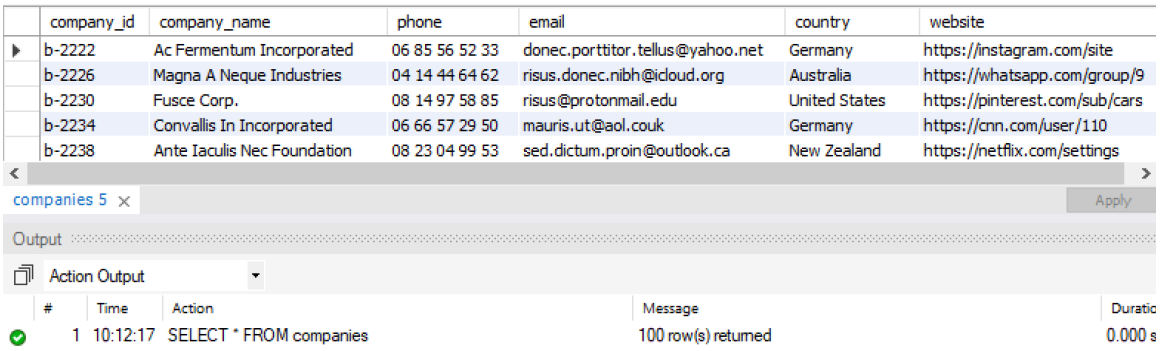
Carga de registros para la tabla “companies”

- Ignoramos la primera fila del .csv al tratarse de los encabezados de las columnas.

```
LOAD DATA LOCAL INFILE "C:/Users/Alex/Downloads/companies.csv"
INTO TABLE companies
FIELDS TERMINATED BY ','    -- Indica que los campos están separados por c
```

```
ENCLOSED BY '"'            -- Indica que los valores pueden estar entre comilla
LINES TERMINATED BY '\r\n'  -- Indica que cada fila termina con salto al estil
IGNORE 1 LINES;
```

- Comprobamos que los datos se hayan cargado correctamente



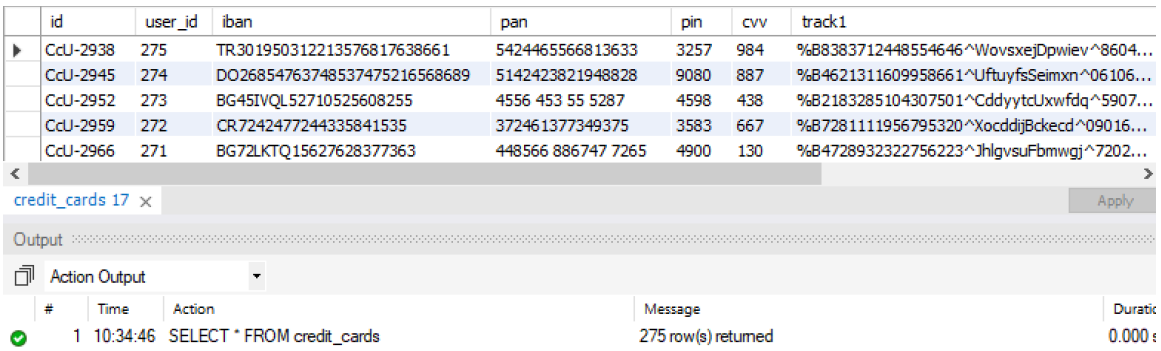
company_id	company_name	phone	email	country	website
b-2222	Ac Fermentum Incorporated	06 85 56 52 33	donec.porttitor.tellus@yahoo.net	Germany	https://instagram.com/site
b-2226	Magna A Neque Industries	04 14 44 64 62	risus.donec.nibh@icloud.org	Australia	https://whatsapp.com/group/9
b-2230	Fusce Corp.	08 14 97 58 85	risus@protonmail.edu	United States	https://pinterest.com/sub/cars
b-2234	Convallis In Incorporated	06 66 57 29 50	mauris.ut@aol.couk	Germany	https://cnn.com/user/110
b-2238	Ante Iaculis Nec Foundation	08 23 04 99 53	sed.dictum.proin@outlook.ca	New Zealand	https://netflix.com/settings

Carga de registros para la tabla “credit_cards”

- Ignoramos la primera fila del .csv al tratarse de los encabezados de las columnas.
- Al inspeccionar el .csv con Notepadd++ hemos visto que fue creado con Unix. Eso repercute en el salto de línea, por lo que en este caso sería “/n”

```
LOAD DATA LOCAL INFILE "C:/Users/Alex/Downloads/credit_cards.csv"
INTO TABLE companies
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'  -- El .csv fue creado con Unix,Linux o Mac
IGNORE 1 LINES;
```

- Comprobamos que los datos se hayan cargado correctamente



id	user_id	iban	pan	pin	cvv	track1
CcU-2938	275	TR301950312213576817638661	5424465566813633	3257	984	%B8383712448554646^WovsxejDpwiev^8604...
CcU-2945	274	DO26854763748537475216568689	5142423821948828	9080	887	%B4621311609958661^UftuyfsSeimxn^06106...
CcU-2952	273	BG45IVQL52710525608255	4556 453 55 5287	4598	438	%B2183285104307501^CddyytcUxwfdq^5907...
CcU-2959	272	CR7242477244335841535	372461377349375	3583	667	%B7281111956795320^XocddijBckecd^09016...
CcU-2966	271	BG72LKTQ15627628377363	448566 886747 7265	4900	130	%B4728932322756223^JhlgvsuFbmwgj^7202...

credit_cards 17 ×

Apply

Output

Action Output

#	Time	Action	Message	Durati
1	10:34:46	SELECT * FROM credit_cards	275 row(s) returned	0.000 s

Carga de registros para la tabla “users”

- Carga para “users_usa.csv”

```
LOAD DATA LOCAL INFILE "C:/Users/Alex/Downloads/users_usa.csv"
INTO TABLE users
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES;
```

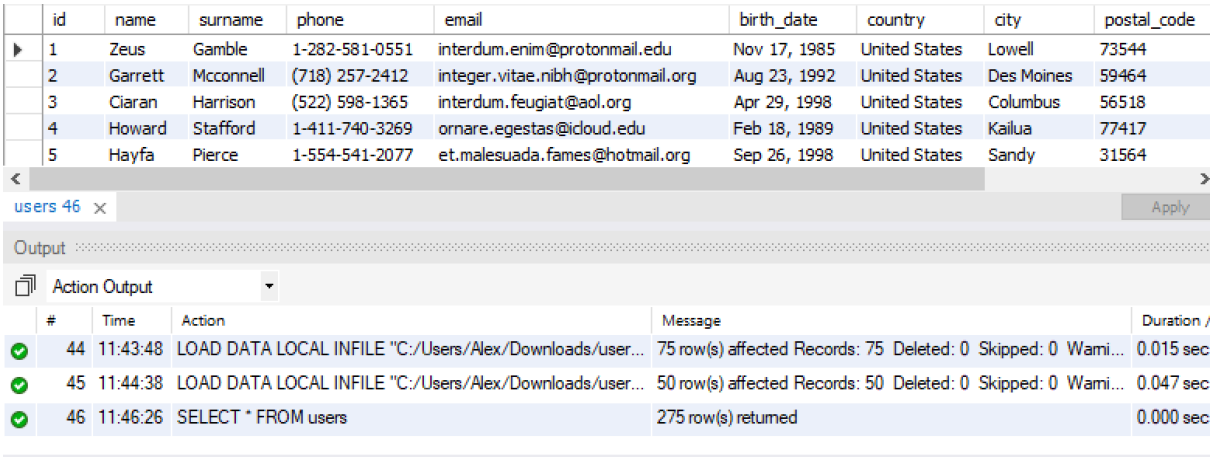
- Carga para “users_ca.csv”

```
LOAD DATA LOCAL INFILE "C:/Users/Alex/Downloads/users_ca.csv"
INTO TABLE users
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES;
```

- Carga para “users_uk.csv”

```
LOAD DATA LOCAL INFILE "C:/Users/Alex/Downloads/users_uk.csv"
INTO TABLE users
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES;
```

- Comprobamos que los datos se han cargado correctamente



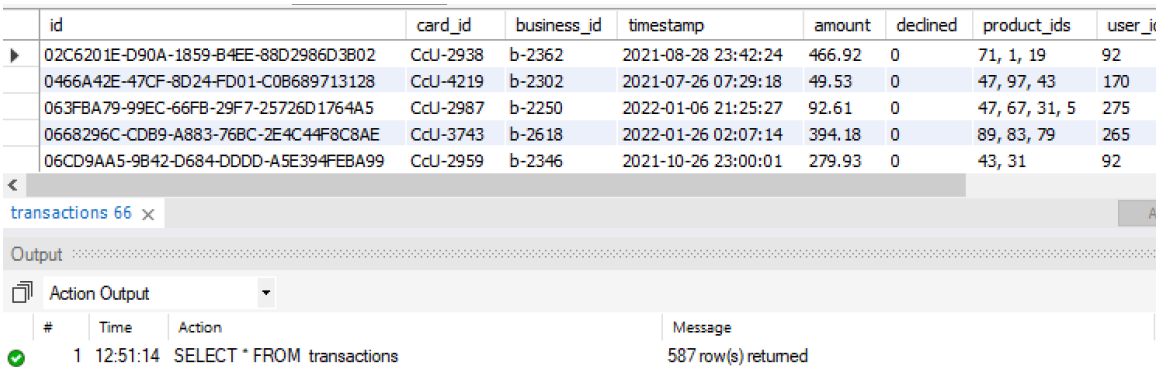
id	name	surname	phone	email	birth_date	country	city	postal_code
1	Zeus	Gamble	1-282-581-0551	interdum.enim@protonmail.edu	Nov 17, 1985	United States	Lowell	73544
2	Garrett	Mcconnell	(718) 257-2412	integer.vitae.nibh@protonmail.org	Aug 23, 1992	United States	Des Moines	59464
3	Ciaran	Harrison	(522) 598-1365	interdum.feugiat@aol.org	Apr 29, 1998	United States	Columbus	56518
4	Howard	Stafford	1-411-740-3269	ornare.egestas@icloud.edu	Feb 18, 1989	United States	Kailua	77417
5	Hayfa	Pierce	1-554-541-2077	et.malesuada.fames@hotmail.org	Sep 26, 1998	United States	Sandy	31564

#	Time	Action	Message	Duration /
44	11:43:48	LOAD DATA LOCAL INFILE "C:/Users/Alex/Downloads/user...	75 row(s) affected Records: 75 Deleted: 0 Skipped: 0 Wami...	0.015 sec
45	11:44:38	LOAD DATA LOCAL INFILE "C:/Users/Alex/Downloads/user...	50 row(s) affected Records: 50 Deleted: 0 Skipped: 0 Wami...	0.047 sec
46	11:46:26	SELECT * FROM users	275 row(s) returned	0.000 sec

Carga de registros para la tabla “transaction”

```
LOAD DATA LOCAL INFILE "C:/Users/Alex/Downloads/transactions.csv"
INTO TABLE transactions
FIELDS TERMINATED BY ';'
ENCLOSED BY '"'
LINES TERMINATED BY '\r\n'
IGNORE 1 LINES;
```

- Comprobamos que la carga se ha realizado correctamente



id	card_id	business_id	timestamp	amount	declined	product_ids	user_i
02C6201E-D90A-1859-B4EE-88D2986D3B02	CcU-2938	b-2362	2021-08-28 23:42:24	466.92	0	71, 1, 19	92
0466A42E-47CF-8D24-FD01-C0B689713128	CcU-4219	b-2302	2021-07-26 07:29:18	49.53	0	47, 97, 43	170
063FBA79-99EC-66FB-29F7-25726D1764A5	CcU-2987	b-2250	2022-01-06 21:25:27	92.61	0	47, 67, 31, 5	275
0668296C-CDB9-A883-76BC-2E4C44F8C8AE	CcU-3743	b-2618	2022-01-26 02:07:14	394.18	0	89, 83, 79	265
06CD9AA5-9B42-D684-DDDD-A5E394FEBA99	CcU-2959	b-2346	2021-10-26 23:00:01	279.93	0	43, 31	92

transactions 66 ×

Output

Action Output

#	Time	Action	Message
1	12:51:14	SELECT * FROM transactions	587 row(s) returned

Creación de las relaciones entre tablas

La tabla “transactions” es la tabla de hechos, mientras que sus dimensiones son las tablas “companies”, “users”, y “credit_cards”.

Procedemos a crear las relaciones entre la tabla de hechos y las de dimensiones para configurar un esquema de estrella

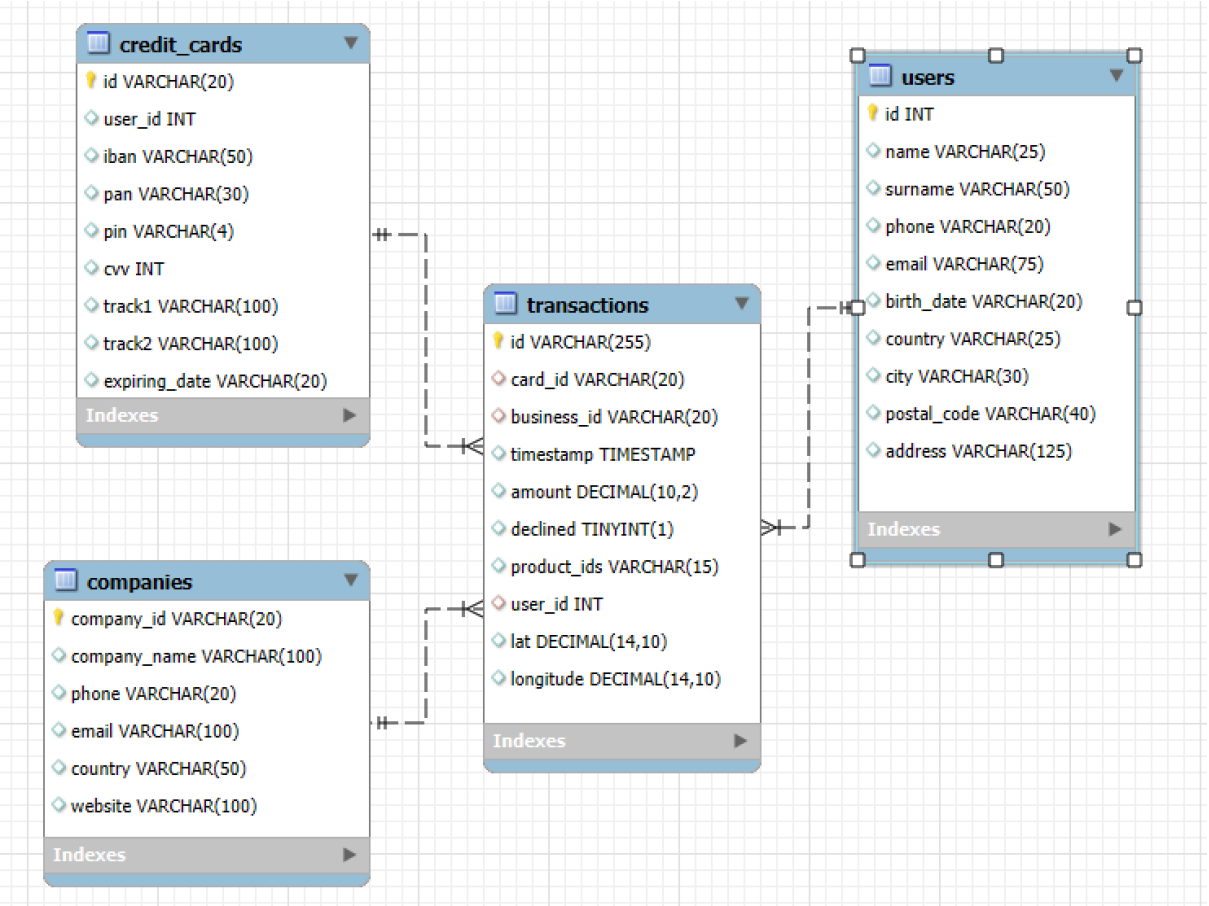
```
-- Relación con "credit_cards"
ALTER TABLE transactions
ADD CONSTRAINT transactions_ibfk_1
FOREIGN KEY (card_id) REFERENCES credit_cards(id);

-- Relación con "companies"
ALTER TABLE transactions
ADD CONSTRAINT transactions_ibfk_2
FOREIGN KEY (business_id) REFERENCES companies(company_id);

-- Relación con "users"
```

```
ALTER TABLE transactions
ADD CONSTRAINT transactions_ibfk_3
FOREIGN KEY (user_id) REFERENCES users(id);
```

Diagrama ER



Exercici 1

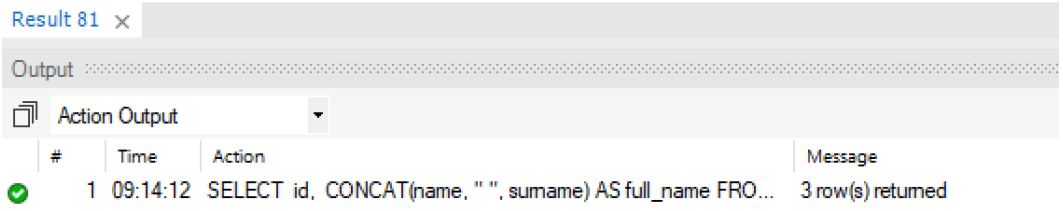
Realitza una subconsulta que mostri tots els usuaris amb més de 30 transaccions utilitzant almenys 2 taules.

Opción 1

- Creamos una subconsulta que nos devuelve el número de usuario de la tabla “transactions”, basado en un filtraje combinado de HAVING y WHERE.
- En WHERE filtramos por todas aquellas transacciones que no han sido declinadas, y en HAVING filtramos por un contaje de todas las transacciones que son mayores a 30 y que cumplen el parámetro anterior. Agrupamos por número de usuario.
- En la consulta principal seleccionamos el id de usuario, el nombre y el apellido de la tabla “users”. Usamos la función CONCAT en el nombre y apellido para que se muestren de forma conjunta en una única tabla y le damos un alias para mayor claridad.
- Unimos la consulta y subconsulta con WHERE y IN a través del campo “id”, que es PK de “users” y que se une a “transactions” a través de “user_id”, su FK

```
SELECT
   id,
   CONCAT(name, " ", surname) AS full_name
FROM users
WHERE id IN (
                  SELECT user_id
                  FROM transactions
                  WHERE declined = 0
                  GROUP BY user_id
                  HAVING COUNT(id) > 30);
```

id	full_name
92	Lynn Riddle
267	Ocean Nelson
272	Hedwig Gilbert

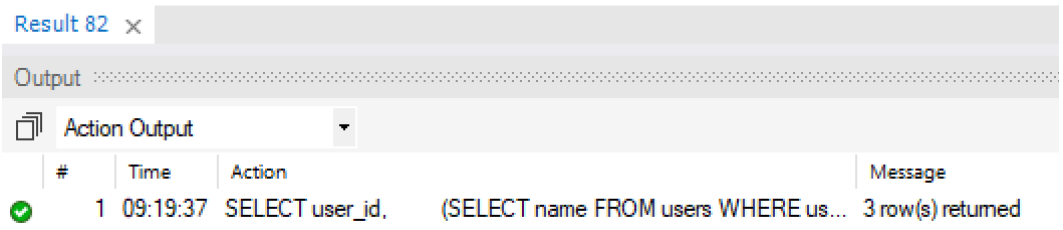


Opción 2

- Nuestra tabla principal es “transactions” y configuramos la consulta principal para que nos devuelva el número de usuario y el contaje de todas las transacciones filtradas por aquellas que no hayan sido declinadas y por las que cuyo contaje sea superior a 30.
- Para obtener también el nombre de usuario y el apellido que solo se encuentran en la tabla “users”, añadimos un par de subconsultas en SELECT seleccionando estos campos de esa tabla, igualando ambas tablas por su PK y FK.
- De esta forma, además de los nombres podremos obtener el número de transacciones en una columna.

```
SELECT
    user_id,
    (SELECT name FROM users WHERE users.id = t.user_id) AS name,
    (SELECT surname FROM users WHERE users.id = t.user_id) AS surname,
    COUNT(t.id) AS num_trans
FROM transactions AS t
WHERE declined = 0
GROUP BY user_id
HAVING num_trans > 30;
```

user_id	name	surname	num_trans
92	Lynn	Riddle	39
267	Ocean	Nelson	39
272	Hedwig	Gilbert	38



Entre ambas opciones, la opción 1 es más eficiente al usar una única subconsulta. Sin embargo, la opción 2 es más completa al mostrar también la cantidad exacta de transacciones realizadas de los tres clientes que coinciden con el criterio de haber realizado más de 30 transacciones.

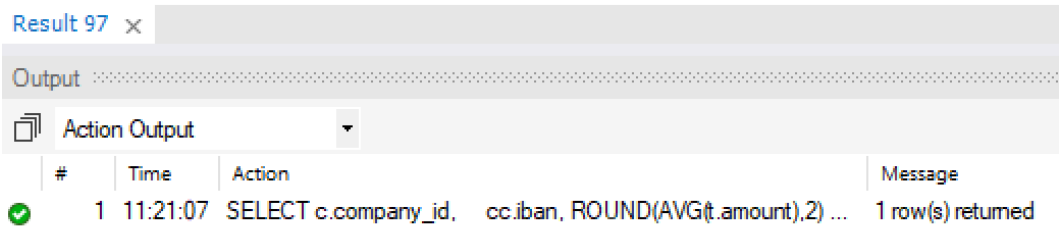
Exercici 2

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

- Unimos la tablas “company”, “transactions” y “credit_cards” mediante una INNER JOIN mediante sus PK y FK.
- Seleccionamos los campos “company_id” y “company_name” de la tabla “companies”, además de calcular la media de “amount” de la tabla “transactions” y seleccionar el campo “iban” de “credit_cards”.
- Filtramos por el identificador de Donec Ltd, ya que al tratarse de una PK la consulta será más eficiente, y por todas aquellas transacciones que no hayan sido declinadas.
- Finalmente, filtramos por el campo “iban” de la tabla “credit_card”.

```
SELECT
    c.company_id,
    c.company_name,
    ROUND(AVG(t.amount),2) AS avg_amount,
    cc.iban
FROM companies AS c
JOIN transactions AS t
    ON c.company_id = t.business_id
JOIN credit_cards AS cc
    ON t.card_id = cc.id
WHERE c.company_id = "b-2242"
    AND declined = 0
GROUP BY cc.iban;
```

company_id	iban	avg_amount
b-2242	PT87806228135092429456346	42.82



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

1. Creamos la tabla en la cual insertaremos el identificador de tarjeta y su estado
    a. Asignamos al campo un valor varchar(20), idéntico al de la tabla “credit_card”, y le asignamos la categoría de PK.
    b. Creamos el campo “status” como un enum o lista seleccionable, identificando dos posibles opciones: “active” o “inactive”, dependiendo de la consulta que usaremos para obtener los datos que necesitamos para añadirlos a la tabla.
    c. Vinculamos esta tabla con “credit_cards” a través del identificador de la tarjeta presente en ambas tablas

```
CREATE table credit_card_status (
    credit_card_id varchar(20) PRIMARY KEY,
    status enum("active", "inactive") NOT NULL
    );
```

- Comprobamos que la tabla se haya creado correctamente

```
DESCRIBE credit_card_status;
```

	Field	Type	Null	Key	Default	Extra
▶	credit_card_id	varchar(20)	NO	PRI	NULL	
	status	enum('active','inactive')	NO		NULL	

Result 99

Output

Action Output

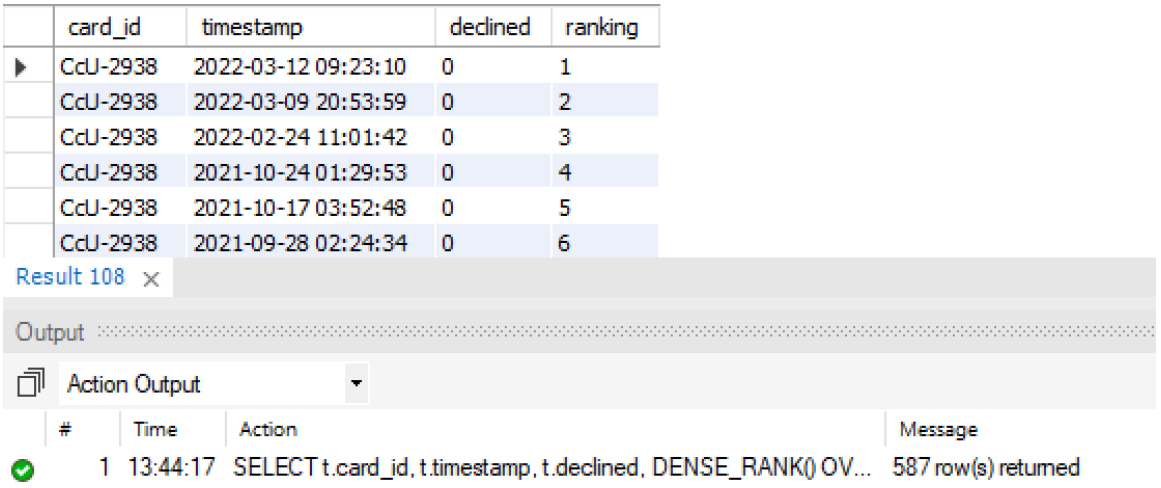
#	Time	Action	Message
1	11:29:31	DESCRIBE credit_card_status	2 row(s) returned

2. Creamos la consulta necesaria para obtener los resultados que necesitamos para insertarlos en la tabla creada

Subconsulta para obtener un contaje de todos los movimientos de las tarjetas sin agrupar los datos en un único registro

- Creamos una consulta para seleccionar los campos “card_id”, “timestamp” y “declined” de la tabla “transactions”.
- Usamos la window function DENSE_RANK() junto a PARTITION BY en el campo “card_id” para crear una nueva columna que agrupe las transacciones por tarjeta, asignándoles una numeración desde 1, y así seguir teniendo contexto de cada transacción en la que la tarjeta fue usada.
- Añadimos un ORDER BY en orden descendente para el campo “timestamp” para ordenar las transacciones de la más reciente a las más antigua.
- El número del ranking se reiniciará cuando cambie el identificador de la tarjeta.

```
SELECT
    t.card_id,
    t.timestamp,
    t.declined,
    DENSE_RANK() OVER(PARTITION BY t.card_id ORDER BY t.timestamp DESC
FROM transactions AS t;
```



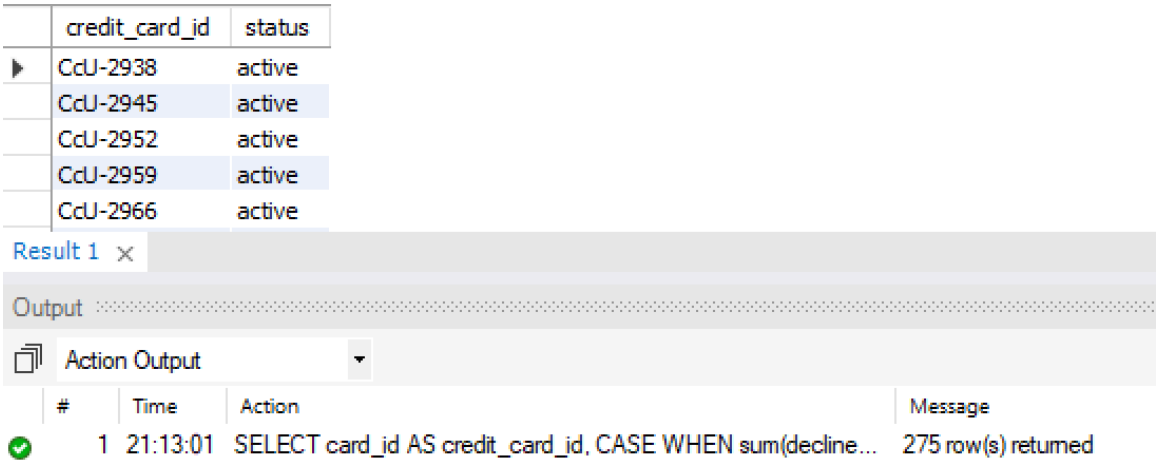
card_id	timestamp	declined	ranking
CcU-2938	2022-03-12 09:23:10	0	1
CcU-2938	2022-03-09 20:53:59	0	2
CcU-2938	2022-02-24 11:01:42	0	3
CcU-2938	2021-10-24 01:29:53	0	4
CcU-2938	2021-10-17 03:52:48	0	5
CcU-2938	2021-09-28 02:24:34	0	6

Consulta para obtener los campos que necesitamos para insertarlos en la tabla

- Seleccionamos el campo “card_id” para obtener el identificador de tarjeta
- Mediante un CASE, indicamos que si esa tarjeta ha sido declinada 3 veces o más (valor 1 o TRUE) , será marcada como “inactive”. Cualquier otro resultado, será marcado como “active”.
- Filtramos por las 3 últimas transacciones realizadas, previamente ordenadas en orden descendiente en la subconsulta

```
SELECT
    card_id AS credit_card_id,
    CASE
        WHEN sum(declined) >= 3 THEN "inactive"
        ELSE "active"
        END AS status
FROM (SELECT
            t.card_id,
            t.timestamp,
            t.declined,
            DENSE_RANK() OVER(PARTITION BY t.card_id ORDER BY t.timestamp
            FROM transactions AS t
      ) AS rt
```

```
WHERE ranking <= 3  --Filtramos por las 3 últ. transacciones (las hemos orde
GROUP BY credit_card_id;
```



El resultado de la consulta nos devuelve que las 275 tarjetas diferentes que están registradas están activas.

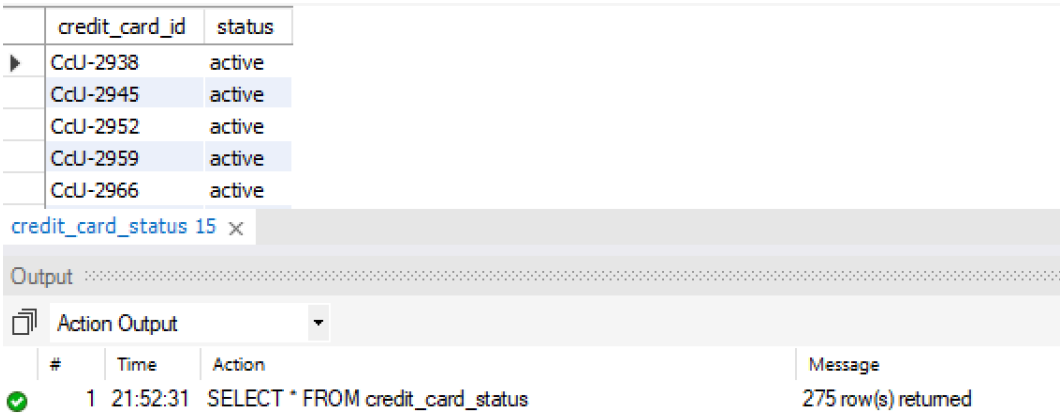
Inserción de la consulta en la tabla creada

```
INSERT INTO credit_card_status (credit_card_id, status)
SELECT
   card_id AS credit_card_id,
   CASE
      WHEN sum(declined) < 3 THEN "inactive"
      ELSE "active"
      END AS status
FROM (
         SELECT
            card_id,
            timestamp,
            declined,
            DENSE_RANK() OVER(PARTITION BY card_id ORDER BY timestamp D
         FROM transactions
```

```
        ) AS rt
WHERE ranking <= 3  --Filtramos por las 3 últ. transacciones (las hemos orde
GROUP BY credit_card_id;
```

- Comprobamos que los datos se han insertado correctamente

```
SELECT *
FROM credit_card_status;
```



Relación entre “credit_card_status” y “credit_cards”

- Relacionamos ambas tablas a través de la PK “id” de la tabla “credit_cards”, que hace de FK del PK “credit_card_id” de “credit_card_status”
- Se trata de una relación de 1 a N

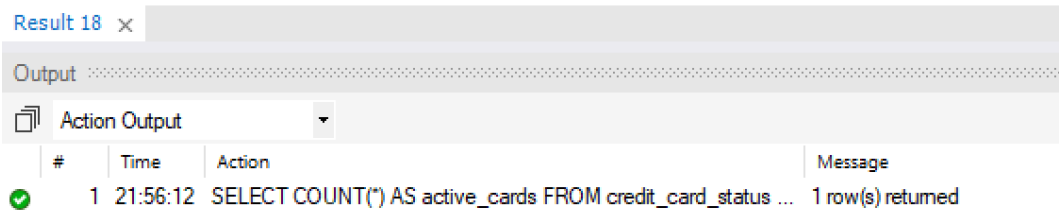
```
ALTER TABLE credit_cards
ADD CONSTRAINT fk_ccards_ccards_status
FOREIGN KEY(id) REFERENCES credit_card_status(credit_card_id);
```

Exercici 1

Quantes targetes estan actives?

```
SELECT COUNT(*) AS active_cards
FROM credit_card_status
WHERE status = "active";
```

	active_cards
▶	275



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.

- Creamos la tabla “products”
  - Asignamos el valor de Primary Key al campo “id”

```
CREATE TABLE IF NOT EXISTS products (
    id int,
  product_name varchar(50),
  price varchar(10),
  colour varchar(10),
  weight decimal(6,2),
```

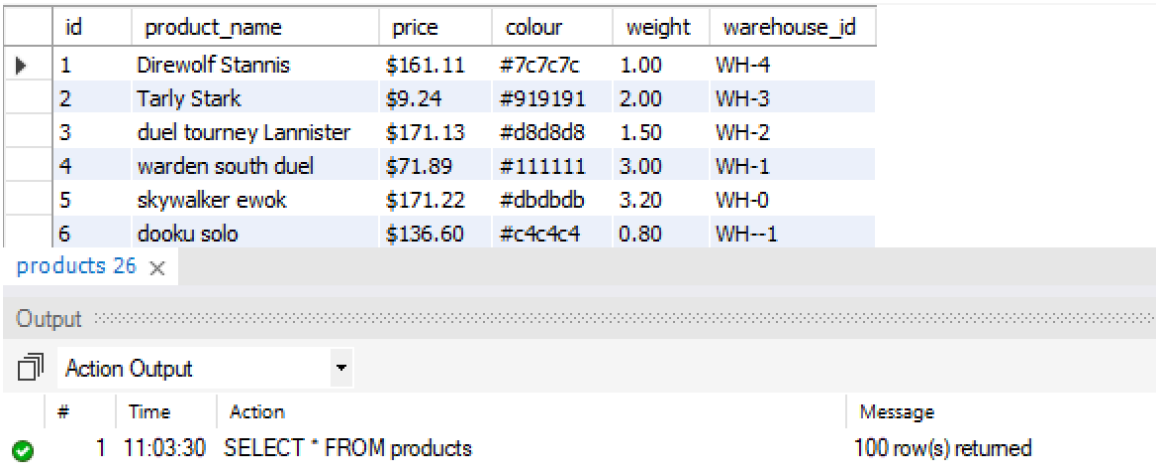
```
  warehouse_id varchar(10),
  PRIMARY KEY (id)
);
```

Carga de registros para la tabla “products”

- Ignoramos la primera fila del .csv al tratarse del encabezado de las columnas

```
LOAD DATA LOCAL INFILE "C:/Users/Alex/Downloads/products.csv"
INTO TABLE products
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 LINES;
```

- Comprobamos que los datos se han cargado correctamente



id	product_name	price	colour	weight	warehouse_id
1	Direwolf Stannis	$161.11	#7c7c7c	1.00	WH-4
2	Tarly Stark	$9.24	#919191	2.00	WH-3
3	duel tourney Lannister	$171.13	#d8d8d8	1.50	WH-2
4	warden south duel	$71.89	#111111	3.00	WH-1
5	skywalker ewok	$171.22	#dbdbdb	3.20	WH-0
6	dooku solo	$136.60	#c4c4c4	0.80	WH--1

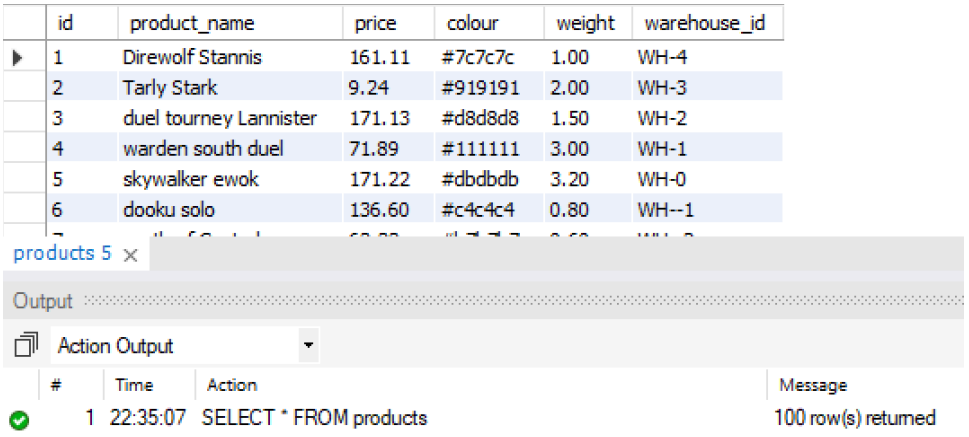
Normalización de la tabla

El campo “price” es un varchar y queremos transformarlo a decimal.

- Para ello, primero eliminamos el simbolo del dolar del campo con la función TRIM, y añadimos LEADING porque el símbolo del dolar está al principio de la cadena (si estuviera al final usaríamos TRAILING)

```
UPDATE products
SET price = TRIM(LEADING '$' FROM price);
```

- Comprobamos que el cambio se ha aplicado correctamente



id	product_name	price	colour	weight	warehouse_id
1	Direwolf Stannis	161.11	#7c7c7c	1.00	WH-4
2	Tarly Stark	9.24	#919191	2.00	WH-3
3	duel tourney Lannister	171.13	#d8d8d8	1.50	WH-2
4	warden south duel	71.89	#111111	3.00	WH-1
5	skywalker ewok	171.22	#dbdbdb	3.20	WH-0
6	dooku solo	136.60	#c4c4c4	0.80	WH--1

- A continuación, le asignamos un valor numérico decimal de 8 dígitos de los cuales 2 son decimales.

```
ALTER TABLE products
MODIFY COLUMN price dec(8,2);
```

- Comprobamos que el cambio se ha aplicado correctamente

```
DESCRIBE products;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	
product_name	varchar(50)	YES		NULL	
price	decimal(8,2)	YES		NULL	
colour	varchar(10)	YES		NULL	
weight	decimal(6,2)	YES		NULL	
warehouse_id	varchar(10)	YES		NULL	

Result 6

Output

Action Output

#	Time	Action	Message
1	22:35:07	SELECT * FROM products	100 row(s) returned
2	22:36:23	DESCRIBE products	6 row(s) returned

Creación de una tabla intermedia entre “transactions” y “products”

La tabla “transactions” tiene un campo llamado “product_ids” con más de un valor en el campo, recogiendo que una misma transacción puede haber servido para comprar más de un producto.

“Product_ids” es un campo mal normalizado porque almacena varios valores almacenados separados por comas. Este diseño de la tabla no cumple la primera norma formal (1NF), en la cual debemos asegurarnos que cada columna contiene valores atómicos o indivisibles.

En este caso, es aconsejable crear una tabla intermedia en la cual pueda recogerse que una misma transacción puede contener varios productos (un producto por registro) y eliminar el campo “product_ids” de “transactions”. Esta tabla intermedia también nos ayudará a establecer la correcta relación entre tablas.

Normalización del campo “product_ids” de “transactions” para cumplir la 1NF

- Los distintos valores se encuentran separados por comas y en algunos registros se trata de 2 valores, en otros de 3 y en otros de 4.

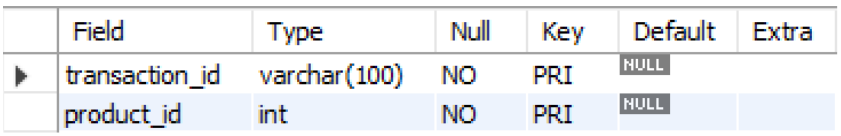
- Vamos a suponer que sabemos que el campo tiene varios valores, pero no sabemos el máximo de valores presentes en un campo entre todos ellos. Supongamos que contiene hasta un máximo de 7 valores.

Creación de la tabla intermedia “orders”

- Para poder repetir o bien el id de transacción o el identificador de producto en diferentes registros, necesitamos que la combinación de ambos sea la Primary Key de la tabla (Primary Key compuesta).

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

- Comprobamos que la tabla se haya creado correctamente



	Field	Type	Null	Key	Default	Extra
▶	transaction_id	varchar(100)	NO	PRI	NULL	
	product_id	int	NO	PRI	NULL	

Normalización del campo multivalor “product_ids”

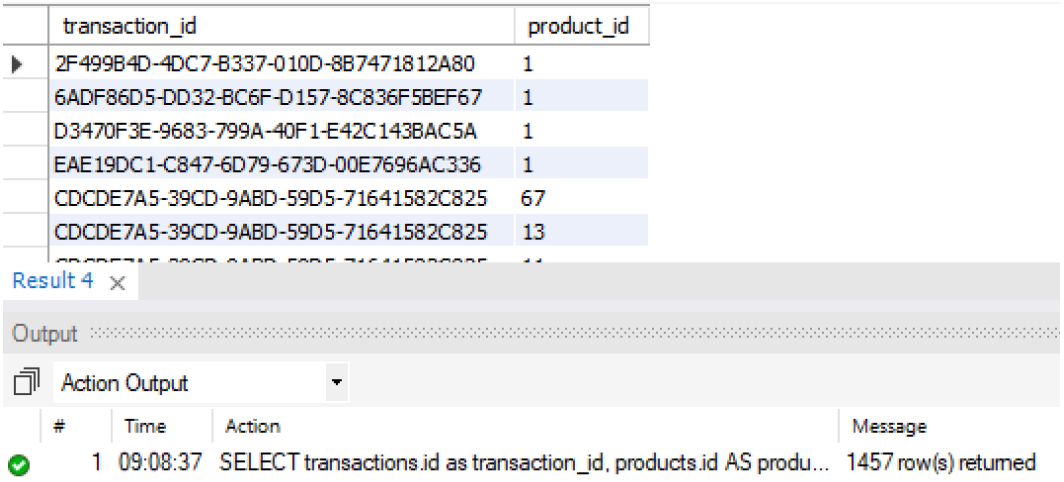
- Después de probar varias opciones para repartir los múltiples valores que contiene “product_ids” en varios campos para cumplir con la norma 1NF sin éxito, me quedo con la opción de unir las tablas “transactions” y “products” mediante un JOIN usando FIND_IN_SET.
    - Seleccionamos el campo identificador de las transacciones de la tabla “transactions”, y el campo identificador de los productos de la tabla

“products”, la tabla de dimensiones en la que tenemos un identificador de producto por registro.

- Hacemos un INNER JOIN de ambas tablas y las unimos mediante la función para cadenas de texto FIND IN SET, que buscará si los valores de “products.id”, su primera variable, se encuentran en “transactions.product_ids”, su segunda variable, y de forma interna nos devolverá su posición en el último campo, devolviéndonos un 0 si no hay una coincidencia entre valores.
- Dentro de FIND IN SET usamos la función REPLACE anidando en la segunda variable mencionada. Con esto conseguimos eliminar todos los espacios vacíos dentro de ese campo.
- Finalmente, usamos el operador lógico “>” para obtener las coincidencias entre campos, que corresponderán a los valores mayores que 0.

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

- Comprobamos que nos devuelve los resultados correctos



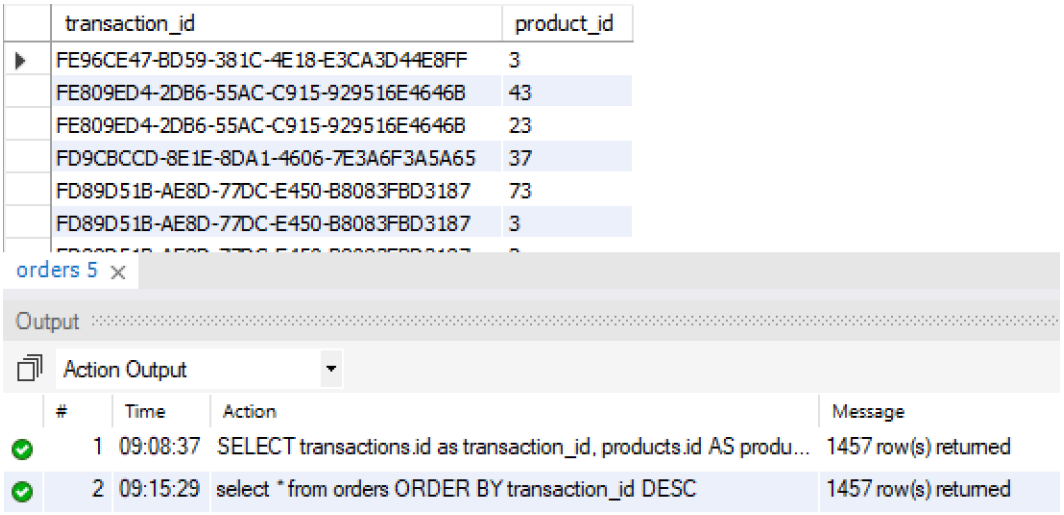
transaction_id	product_id
2F499B4D-4DC7-B337-010D-8B7471812A80	1
6ADF86D5-DD32-BC6F-D157-8C836F5BEF67	1
D3470F3E-9683-799A-40F1-E42C143BAC5A	1
EAE19DC1-C847-6D79-673D-00E7696AC336	1
CDCDE7A5-39CD-9ABD-59D5-71641582C825	67
CDCDE7A5-39CD-9ABD-59D5-71641582C825	13

- Insertamos los resultados en la tabla intermedia “orders” que hemos creado anteriormente mediante el comando INSERT INTO.

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

- Comprobamos que los datos se hayan insertado correctamente

```
SELECT *
FROM orders
ORDER BY transaction_id DESC;
```



transaction_id	product_id
FE96CE47-BD59-381C-4E18-E3CA3D44E8FF	3
FE809ED4-2DB6-55AC-C915-929516E4646B	43
FE809ED4-2DB6-55AC-C915-929516E4646B	23
FD9CBCCD-8E1E-8DA1-4606-7E3A6F3A5A65	37
FD89D51B-AE8D-77DC-E450-B8083FBD3187	73
FD89D51B-AE8D-77DC-E450-B8083FBD3187	3

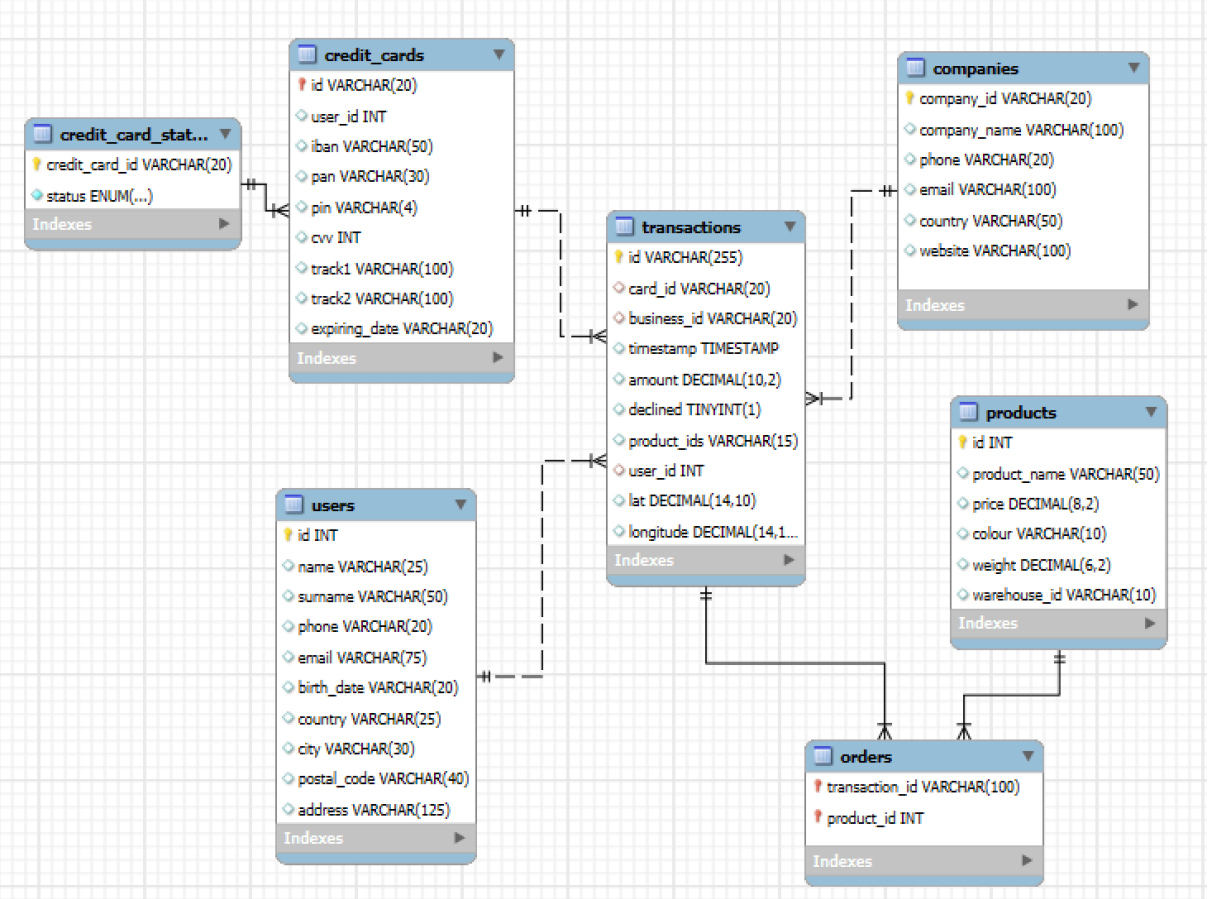
Relación entre las tabla intermedia “orders” y el resto de tablas

La tabla “orders” tendrá dos relaciones, una hacia “transactions” y otra hacia “orders”, actuando como tabla intermedia que servirá como puente de unión entre ambas tablas.

```
-- Relación entre "orders" y "products"
ALTER TABLE orders
ADD CONSTRAINT fk_orders_products
FOREIGN KEY (product_id) REFERENCES products(id);

-- Relación entre "orders" y "transactions"
ALTER TABLE orders
ADD CONSTRAINT fk_orders_transactions
FOREIGN KEY (transaction_id) REFERENCES transactions(id);
```

Diagrama ER



A diferencia de las relaciones entre las otras tablas que tienen una línea punteada uniéndolas, la relación entre “products”, “orders” y transactions” muestra una línea sólida entre tablas.

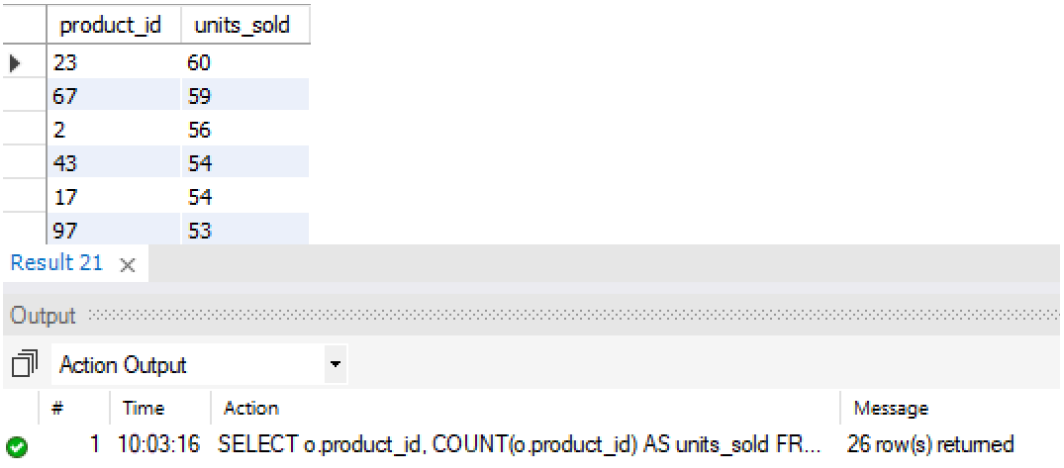
Esto es como consecuencia de la relación identificada que une a las tres tablas, donde la tabla intermedia no puede ser correctamente identificada sin su tabla madre, teniendo además una Primary Key compuesta hecha de las dos Primary Key de las dos tablas madre que vincula.

Exercici 1

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

- Seleccionamos el campo “product_id” de la tabla y realizamos un conteo sobre el mismo campo para obtener el identificador de producto y el número de veces que aparece en la tabla “orders”.
- Para filtrar por aquellas transacciones que no han sido declinadas en WHERE, hacemos un JOIN con transactions para recuperar el campo “declined”.
- Finalmente, agrupamos lo resultados por identificador de producto y ordenamos los resultados por unidades vendidas de mayor a menor.

```
SELECT
    o.product_id,
  COUNT(o.product_id) AS units_sold
FROM orders AS o
JOIN transactions as t
    ON t.id = o.transaction_id
WHERE t.declined = 0
GROUP BY o.product_id
ORDER BY units_sold DESC;
```



product_id	units_sold
23	60
67	59
2	56
43	54
17	54
97	53

Result 21

Output

Action Output

#	Time	Action	Message
1	10:03:16	SELECT o.product_id, COUNT(o.product_id) AS units_sold FR...	26 row(s) returned