

Tasca S4.01. Creació de Base de Dades

Fernando Poblete

Descripció

Partint d'alguns arxius CSV dissenyaràs i crearàs la teva 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 puguis realitzar les següents consultes:

- Al revisar las bases de datos, lo primero que llama la atención es que contamos con 3 tablas de 'users', cada una para un país diferente. Al revisar y comparar los datos entre tablas, notamos que las estructuras de las tablas son exactamente iguales, y que el identificador único, 'id', no se repite de una tabla a otra, por lo que **podemos unir los datos de las tres tablas en una sola** tabla de 'users', con 'id' como PRIMARY KEY.

Creación Tabla 'users'

Creamos la tabla 'users' luego de revisar los datos en el archivo a importar:

```
1  -- Creamos la base de datos
2  • CREATE DATABASE IF NOT EXISTS tasca4;
3  • USE tasca4;
4
5  /*CREACION E INGRESO DE DATOS TABLA USERS*/
6  -- Creación tabla 'users'
7  • CREATE TABLE IF NOT EXISTS users (
8      id INT AUTO_INCREMENT PRIMARY KEY,
9      name VARCHAR(20),
10     surname VARCHAR(20),
11     phone VARCHAR(20),
12     email VARCHAR(100),
13     birth_date DATE,
14     country VARCHAR(20),
15     city VARCHAR(40),
16     postal_code VARCHAR(12),
17     address VARCHAR(60)
18 );
19
```

Output

#	Time	Action	Message
✓ 1	00:02:09	CREATE DATABASE IF NOT EXISTS tasca4	1 row(s) affected
✓ 2	00:02:09	USE tasca4	0 row(s) affected
✓ 3	00:02:09	CREATE TABLE IF NOT EXISTS users (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(20), surname VARCHAR(20), phone VARCHAR(20), email VARCHAR(100), birth_date DATE, country VARCHAR(20), city VARCHAR(40), postal_code VARCHAR(12), address VARCHAR(60))	0 row(s) affected

Cargamos datos a la tabla 'users', haciendo la transformación de datos necesaria para la lectura de fechas:

```

19 • SHOW VARIABLES LIKE 'local_infile';
20
21 -- Luego de revisar la consistencia de los 3 archivos csv de users, importamos los registros de las 3 tablas en una sola
22 -- IMPORTACION DE DATOS TABLAS USERS A UNA MISMA TABLA
23 • LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\users_usa.csv"
24 INTO TABLE users
25 FIELDS TERMINATED BY ','
26 ENCLOSED BY '"'
27 LINES TERMINATED BY '\\r\\n'
28 IGNORE 1 ROWS
29 (id, name, surname, phone, email, @birth_date, country, city, postal_code, address)
30 SET birth_date = STR_TO_DATE(@birth_date, '%b %d, %Y');
31
32 • LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\users_uk.csv"
33 INTO TABLE users
34 FIELDS TERMINATED BY ','
35 ENCLOSED BY '"'
36 LINES TERMINATED BY '\\r\\n'
37 IGNORE 1 ROWS
38 (id, name, surname, phone, email, @birth_date, country, city, postal_code, address)
39 SET birth_date = STR_TO_DATE(@birth_date, '%b %d, %Y');
40
41 • LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\users_ca.csv"
42 INTO TABLE users
43 FIELDS TERMINATED BY ','
44 ENCLOSED BY '"'
45 LINES TERMINATED BY '\\r\\n'
46 IGNORE 1 ROWS
47 (id, name, surname, phone, email, @birth_date, country, city, postal_code, address)
48 SET birth_date = STR_TO_DATE(@birth_date, '%b %d, %Y');
49

```

Output

#	Time	Action	Message
✓ 1	23:12:30	LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\users_usa.csv" INTO TABLE users FIELDS TERMINATED BY ',' ENC...	150 row(s) affected Records: 150 Deleted: 0 Skipped: 0 Warnings: 0
✓ 2	23:12:30	LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\users_uk.csv" INTO TABLE users FIELDS TERMINATED BY ',' ENCL...	50 row(s) affected Records: 50 Deleted: 0 Skipped: 0 Warnings: 0
✓ 3	23:12:30	LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\users_ca.csv" INTO TABLE users FIELDS TERMINATED BY ',' ENCL...	75 row(s) affected Records: 75 Deleted: 0 Skipped: 0 Warnings: 0

Revisamos carga de tabla 'users':

```

49
50 -- Revisamos la creacion de la tabla 'users'
51 • SELECT *
52 FROM users;
53

```

Result Grid

	id	name	surname	phone	email	birth_date	country	city	postal_code	address
▶	1	Zeus	Gamble	1-282-581-0551	interdum.enim@protonmail.edu	1985-11-17	United States	Lowell	73544	348-7818 Sagittis St.
	2	Garrett	Mconnell	(718) 257-2412	integer.vitae.nibh@protonmail.org	1992-08-23	United States	Des Moines	59464	903 Sit Ave
	3	Ciaran	Harrison	(522) 598-1365	interdum.feugiat@aol.org	1998-04-29	United States	Columbus	56518	736-2063 Tellus St.
	4	Howard	Stafford	1-411-740-3269	ornare.egestas@icloud.edu	1989-02-18	United States	Kailua	77417	Ap #545-2244 Erat. Rd.
	5	Hayfa	Pierce	1-554-541-2077	et.malesuada.fames@hotmail.org	1998-09-26	United States	Sandy	31564	341-2821 Ultrices Av.
	6	Inel	Tuenn	(718) 788-8070	gravida.nunc.sed@vubhon.ca	1989-10-15	United States	Nashville	96838	888-7799 Amet Street

Output

#	Time	Action	Message
✓ 1	23:13:02	SELECT * FROM users LIMIT 0, 1000	275 row(s) returned

- Ahora procedemos a identificar cual será la tabla de hechos en nuestro modelo fijándonos en las columnas con nombre que contengan la palabra 'id' (revisando consistencia de codificación entre tablas).
 - Todas las tablas cuentan con identificadores únicos, los cuales se convertirán en nuestras PRIMARY KEYS (al crear todas las tablas e especificar sus PRIMARY KEYS, las acepta ya que son únicas en cada tabla).
 - **La tabla 'transactions' cuenta con identificadores que hacen referencias a todas las otras tablas, lo que junto con la naturaleza de sus datos (volumen y mediciones), la convierte en una tabla ideal para ser el centro de nuestro modelo de estrella como una tabla de hechos.**
 - Notamos que la tabla 'credit_cards' cuenta con una referencia a la tabla de 'users', pero que esta no nos es de mayor utilidad, ya que la tabla 'transactions' ya cuenta con referencias a 'credit_cards' y a 'users', por lo que decidimos no hacer uso de este dato, y no conectar las tablas 'users' y 'credit_cards' directamente.
 - Tomamos nota de diferencia de nombres, pero consistencia de codificación, para el identificador de la compañía, que se llama 'company_id' en la tabla 'company, y 'business_id' en la tabla de 'transaction'
- Como hemos identificado la tabla de hechos, procedemos a crear y poblar las tablas de dimensiones: 'credit_cards', 'companies' y 'products'

Creación Tabla 'credit_cards'

Creamos la tabla 'credit_cards' luego de revisar los datos en el archivo a importar:

```

54
55 /*CREACION E INGRESO DE DATOS TABLA CREDIT_CARDS*/
56 -- Creación tabla 'credit_cards'
57 CREATE TABLE IF NOT EXISTS credit_cards (
58     id VARCHAR(10) PRIMARY KEY,
59     user_id INT,
60     iban VARCHAR(40) UNIQUE,
61     pan VARCHAR(30) UNIQUE,
62     pin VARCHAR(12),
63     cvv VARCHAR(4),
64     track1 VARCHAR(60),
65     track2 VARCHAR(60),
66     expiring_date DATE
67 );
  
```

Output:

#	Time	Action	Message
1	23:31:21	CREATE TABLE IF NOT EXISTS credit_cards (id VARCHAR(10) PRIMARY KEY, user_id INT, iban VARCHAR(40) UNIQUE, panVA...	0 row(s) affected

Cargamos datos a la tabla 'credit_cards', haciendo la transformación de datos necesaria para la lectura de fechas, y revisamos la tabla:

tasca4 users

Limit to 1000 rows

```

68
69 -- IMPORTACION DE DATOS A TABLA CREDIT_CARDS
70 • LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\credit_cards.csv"
71 INTO TABLE credit_cards
72 FIELDS TERMINATED BY ','
73 ENCLOSED BY '"'
74 LINES TERMINATED BY '\n'
75 -- LINES TERMINATED BY '\\r\\n' (ESTO FALLA)
76 IGNORE 1 ROWS
77 (id, user_id, iban, pan, pin, cvv, track1, track2, @expiring_date)
78 SET expiring_date = STR_TO_DATE(@expiring_date, '%m/%d/%y');
79
80 -- Revisamos la creacion de la tabla 'credit cards'
81 • SELECT *
82 FROM credit_cards;
83

```

Result Grid

id	user_id	iban	pan	pin	cvv	track1	track2	expiring_date
CcU-2938	275	TR301950312213576817638661	5424465566813633	3257	984	%B8383712448554646^WovsxeJDpwiev^8604...	%B7653863056044187=800716333673	2022-10-30
CcU-2945	274	DO26854763748537475216568689	5142423821948828	9080	887	%B4621311609958661^UftuyfsSeimxn^06106...	%B4149568437843501=510714033071	2023-08-24
CcU-2952	273	BG45IVQL52710525608255	4556 453 55 5287	4598	438	%B2183285104307501^CddyttUxwfdq^5907...	%B6778580257827162=6906859740077	2021-06-29
CcU-2959	272	CR7242477244335841535	372461377349375	3583	667	%B7281111956795320^XocddjBdkecd^09016...	%B4246154489281853=280522391678	2023-02-24
CcU-2966	271	BG72LKTQ15627628377363	448566 886747 7265	4900	130	%B4728932322756223^JhlgvsuBmwgj^7202...	%B2318571115599881=890821578475	2024-10-29

credit_cards 6 x

Output

Action Output

#	Time	Action	Message
1	23:50:30	LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\credit_cards.csv" INTO TABLE credit_cards FIELDS TERMINATED B...	275 row(s) affected Records: 275 Deleted: 0 Skipped: 0 Warnings: 0
2	23:50:30	SELECT * FROM credit_cards LIMIT 0, 1000	275 row(s) returned

Creación Tabla 'companies'

Creamos la tabla 'companies' luego de revisar los datos en el archivo a importar. Importamos los datos y revisamos la importación:

The screenshot shows a MySQL Workbench window with a SQL editor and a results pane. The SQL editor contains the following code:

```
84 /*CREACION E INGRESO DE DATOS TABLA COMPANIES*/
85 -- Creación tabla 'companies'
86 CREATE TABLE IF NOT EXISTS companies (
87     company_id VARCHAR(10) PRIMARY KEY,
88     company_name VARCHAR(60),
89     phone VARCHAR(30) UNIQUE,
90     email VARCHAR(50) UNIQUE,
91     country VARCHAR(30),
92     website VARCHAR(60)
93 );
94
95 -- IMPORTACION DE DATOS A TABLA COMPANIES
96 LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\companies.csv"
97 INTO TABLE companies
98 FIELDS TERMINATED BY ','
99 ENCLOSED BY '"'
100 LINES TERMINATED BY '\r\n'
101 IGNORE 1 ROWS;
102
103 -- Revisamos la creacion de la tabla 'companies'
104 SELECT *
105 FROM companies;
```

The results pane shows a table with 7 rows of data:

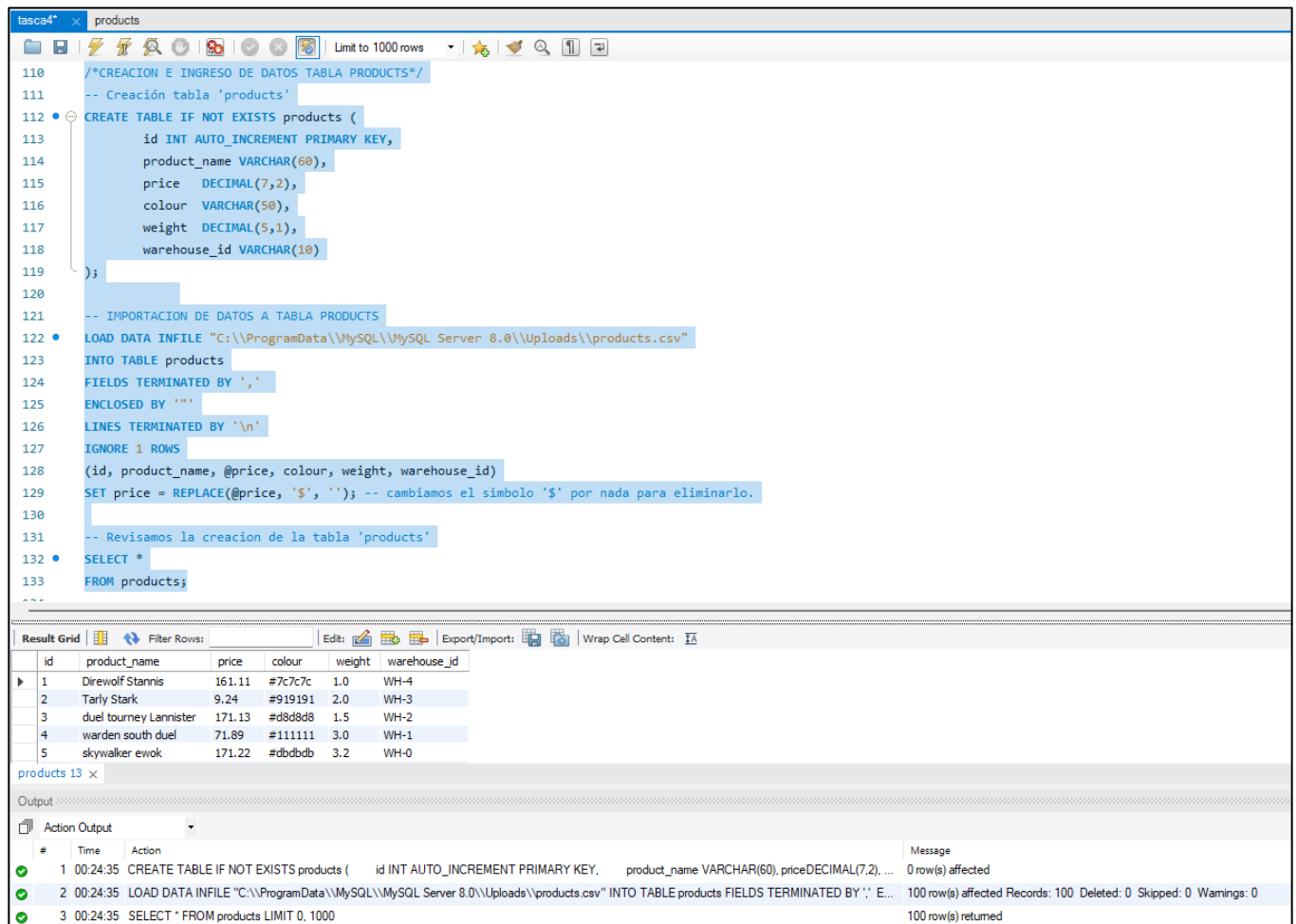
company_id	company_name	phone	email	country	website
b-2222	Ac Fermentum Incorporated	06 85 56 52 33	donec.porttitor.tellus@yahoo.net	Germany	https://instagram.com/site
b-2226	Magna A Neque Industries	04 14 44 64 62	risus.donec.nibh@idoud.org	Australia	https://whatsapp.com/group/9
b-2230	Fusce Corp.	08 14 97 58 85	risus@protonmail.edu	United States	https://pinterest.com/sub/cars
b-2234	Convalis In Incorporated	06 66 57 29 50	mauris.ut@aol.co.uk	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
b-2242	Donec Ltd	01 25 51 37 37	at.iaculis@hotmail.co.uk	Norway	https://nytimes.com/user/110

The output pane shows the following messages:

```
# Time Action Message
1 23:53:34 CREATE TABLE IF NOT EXISTS companies ( company_id VARCHAR(10) PRIMARY KEY, company_name VARCHAR(60), phoneVARC... 0 row(s) affected
2 23:53:34 LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\companies.csv" INTO TABLE companies FIELDS TERMINATED BY '...' 100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0
3 23:53:34 SELECT * FROM companies LIMIT 0, 1000 100 row(s) returned
```

Creación Tabla 'products'

Creamos la tabla 'products' luego de revisar los datos en el archivo a importar. Limpiamos el símbolo '\$' de la columna 'price', en caso de que debamos hacer cálculos numéricos con esta variable en un futuro.



```
110 /*CREACION E INGRESO DE DATOS TABLA PRODUCTS*/
111 -- Creación tabla 'products'
112 CREATE TABLE IF NOT EXISTS products (
113     id INT AUTO_INCREMENT PRIMARY KEY,
114     product_name VARCHAR(60),
115     price DECIMAL(7,2),
116     colour VARCHAR(50),
117     weight DECIMAL(5,1),
118     warehouse_id VARCHAR(10)
119 );
120
121 -- IMPORTACION DE DATOS A TABLA PRODUCTS
122 LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\products.csv"
123 INTO TABLE products
124 FIELDS TERMINATED BY ','
125 ENCLOSED BY '"'
126 LINES TERMINATED BY '\n'
127 IGNORE 1 ROWS
128 (id, product_name, @price, colour, weight, warehouse_id)
129 SET price = REPLACE(@price, '$', ''); -- cambiamos el simbolo '$' por nada para eliminarlo.
130
131 -- Revisamos la creacion de la tabla 'products'
132 SELECT *
133 FROM products;
```

	id	product_name	price	colour	weight	warehouse_id
1	161.11	Direwolf Stannis	161.11	#7c7c7c	1.0	WH-4
2	9.24	Tarly Stark	9.24	#919191	2.0	WH-3
3	171.13	duel tourney Lannister	171.13	#d8d8d8	1.5	WH-2
4	71.89	warden south duel	71.89	#111111	3.0	WH-1
5	171.22	skywalker ewok	171.22	#dbdbdb	3.2	WH-0

products 13 x

Output

Action Output

#	Time	Action	Message
1	00:24:35	CREATE TABLE IF NOT EXISTS products (id INT AUTO_INCREMENT PRIMARY KEY, product_name VARCHAR(60), price DECIMAL(7,2), ...	0 row(s) affected
2	00:24:35	LOAD DATA INFILE "C:\\ProgramData\\MySQL\\MySQL Server 8.0\\Uploads\\products.csv" INTO TABLE products FIELDS TERMINATED BY ',' E...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0
3	00:24:35	SELECT * FROM products LIMIT 0, 1000	100 row(s) returned

Análisis de cardinalidad

- A continuación, luego de haber creado y poblado todas nuestras tablas de dimensiones, revisamos la cardinalidad entre la tabla de hecho 'transactions', con cada una de las tablas de dimensiones, revisando los datos en 'transactions' y entendiendo la lógica que hay detrás:
 - **users:** Un usuario puede hacer múltiples transacciones, pero cada transacción está asociada a solamente 1 usuario. **'transactions:users -> N:1'**
 - **credit_cards:** Una tarjeta de crédito puede ser usada para múltiples transacciones, pero cada transacción solo está asociada a una tarjeta de crédito. **'transactions: credit_cards -> N:1'**

- **companies:** Una compañía puede estar asociada a diferentes transacciones, pero cada transacción solamente está asociada a una compañía. **'transactions: companies -> N:1'**
- **products:** Aquí tenemos un caso especial, ya que, aunque cada transacción tiene un identificador único, tenemos en la tabla de 'transactions' la columna llamada 'product_ids', que nos lista, en valores separados por ',' diferentes productos que puede haber en una transacción. Naturalmente por otro lado, cada producto en la tabla 'products' puede comprarse varias veces en transacciones diferentes. **'transactions: products -> N:M'**

¿Dónde van las claves foráneas?

- Para entender donde van las llaves foráneas, utilizamos el principio de dependencia lógica. Las tablas que pueden existir por si misma, sin información de otras tablas, no requieren de llaves foráneas.
- Por otro lado, las tablas que solo tienen sentido si se conectan con otras, llevan las llaves foráneas.
- En el caso de la tabla puente, esta depende completamente tanto de 'transactions' como de 'products', por lo que tendremos que crear dos llaves foráneas, que enlacen los identificadores en 'transactions' y 'products' contenidos en la tabla puente, con los identificadores únicos de cada tabla respectiva.
- Es importante recordar, que una tabla con llaves foráneas no se puede poblar con datos en sus PRIMARY KEYS, si no están poblados aun sus datos equivalentes en las diferentes tablas que las FOREIGN KEYS apuntan.
- **Por lo anterior, nuestros pasos finales para crear la Base de Datos serán:**
 1. Crear la tabla transaction con sus FOREIGN KEYS apuntando a 'users', 'credit_cards' y 'companies'. Poblar la tabla con datos y revisar estructura relacional.
 2. Crear la tabla puente con FOREIGN KEYS apuntando a 'transactions' y 'products'.
 3. Poblar la tabla puente

1. Creación Tabla 'transactions'

The screenshot shows a SQL script in the 'tasca4*' window. The script creates a table named 'transactions' with the following columns and constraints:

- id VARCHAR(40) PRIMARY KEY,
- card_id VARCHAR(10),
- business_id VARCHAR(10),
- timestamp TIMESTAMP,
- amount DECIMAL(10, 2),
- declined BOOLEAN,
- product_ids VARCHAR(50),
- user_id INT,
- lat FLOAT,
- longitude FLOAT,
- FOREIGN KEY (card_id) REFERENCES credit_cards(id),
- FOREIGN KEY (business_id) REFERENCES companies(company_id),
- FOREIGN KEY (user_id) REFERENCES users(id)

The script also includes a comment for data import and a LOAD DATA INFILE statement to load data from 'transactions.csv' into the 'transactions' table. The output window shows two actions:

- 1 16:43:50 CREATE TABLE IF NOT EXISTS transactions (id VARCHAR(40) PRIMARY KEY, card_id VARCHAR(10), business_id VARCHAR(10), timestamp TIMES... 0 row(s) affected
- 2 16:43:50 LOAD DATA INFILE "C:\ProgramData\MySQL\MySQL Server 8.0\Uploads\transactions.csv" INTO TABLE transactions FIELDS TERMINATED BY '...' 587 row(s) affected Records: 587 Deleted: 0 Skipped: 0 Warnings: 0

Revisamos carga de tabla 'transactions':

The screenshot shows the SQL script in the 'tasca4*' window. The script is a SELECT statement to verify the data load:

```
SELECT * FROM transactions;
```

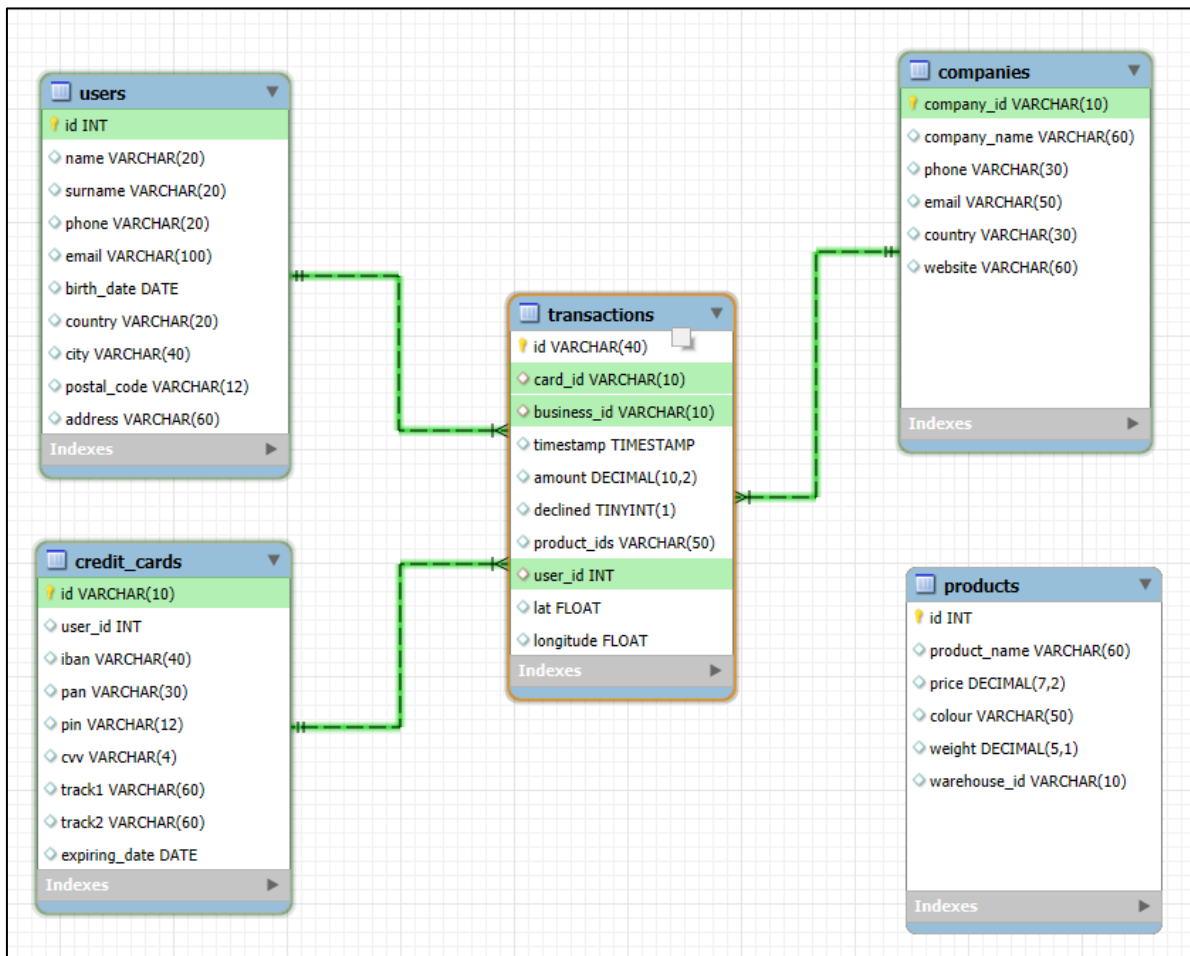
The output window shows the result grid with 15 rows of data. The columns are: id, card_id, business_id, timestamp, amount, declined, product_ids, user_id, lat, and longitude. The data is as follows:

id	card_id	business_id	timestamp	amount	declined	product_ids	user_id	lat	longitude
02C6201E-D90A-1859-B4EE-88D2986D3B02	CdU-2938	b-2362	2021-08-28 23:42:24	466.92	0	71, 1, 19	92	81.9185	-12.5276
0466A42E-47CF-8D24-FD01-C0B689713128	CdU-4219	b-2302	2021-07-26 07:29:18	49.53	0	47, 97, 43	170	-43.9695	-117.525
063FBA79-99EC-66FB-29F7-25726D1764A5	CdU-2987	b-2250	2022-01-06 21:25:27	92.61	0	47, 67, 31, 5	275	-81.2227	-129.05
0668296C-CDB9-A883-76BC-2E4C4F8C8AE	CdU-3743	b-2618	2022-01-26 02:07:14	394.18	0	89, 83, 79	265	-34.3593	-100.556
06CD9AA5-9B42-D684-DDDD-A5E394FEBA99	CdU-2959	b-2346	2021-10-26 23:00:01	279.93	0	43, 31	92	33.7381	158.298
07A46D48-31A3-7E87-65B9-0DA902AD109F	CdU-3225	b-2386	2021-06-28 21:11:42	340.87	1	47, 23	272	38.8342	92.1905
09DE92CE-6F27-2BB7-13B5-9385B2B388E2	CdU-3071	b-2298	2021-05-11 20:40:06	303.05	1	67, 7	275	71.1706	10.5757
0A476ED9-0C13-1962-F87B-D3563924B539	CdU-4359	b-2302	2022-02-26 20:33:54	430.49	0	29, 41, 11	221	-56.4901	114.801

The output window shows one action:

- 1 16:46:07 SELECT * FROM transactions LIMIT 0, 1000 587 row(s) returned

Revisamos Diagrama Relacional al incluir nuestras primeras FOREIGN KEYS:



2. Creación tabla puente

- Debido a la relación N:M entre 'transactions' y 'products', se hace necesario crear una tabla puente para conectar ambas tablas. Esta tabla la llamaremos 'transactions_products', y tendrá una llave primaria compuesta, por las llaves primarias de 'transactions' y 'products'. Esto significa que **cada registro de esta nueva tabla contiene una combinación de transacción y producto única**.
- Es importante notar que esta definición restringe nuestro modelo a que solamente se pueda comprar una unidad del mismo producto en una misma transacción. Con los datos que contamos esto no representa un problema, pero de requerir adaptar el modelo a poder comprar múltiples unidades en una sola transacción, podríamos agregar una columna de cantidad 'quantity' en la tabla puente. Como se nos ha solicitado crear un modelo de estrella, no incorporamos esta columna, ya que rompería el esquema solicitado.
- Finalmente, las FOREIGN KEYS de esta tabla, serán los mismos componentes de la PRIMARY KEY COMPUESTA, pero por separado. Así, cada combinación de transacción-producto, hará referencia por separado, a la información de la transacción y el producto, en las tablas 'transactions' y 'products' respectivamente.

tasca4+ × products credit_cards

Limit to 1000 rows

```

167 -- CREACION DE LA TABLA PUENTE 'transactions_products'
168 CREATE TABLE transactions_products (
169     id_transaction VARCHAR(40),
170     id_product INT,
171     -- quantity INT NOT NULL, -- No lo incorporamos por ahora
172     PRIMARY KEY (id_transaction, id_product),
173     FOREIGN KEY (id_transaction) REFERENCES transactions(id),
174     FOREIGN KEY (id_product) REFERENCES products(id)
175 );
176
177 DESCRIBE transactions_products;

```

Result Grid | Filter Rows: | Exports: | Wrap Cell Content: [IA](#)

Field	Type	Null	Key	Default	Extra
id_transaction	varchar(40)	NO	PRI	NULL	
id_product	int	NO	PRI	NULL	

Result 16 ×

Output

Action Output

#	Time	Action	Message
✓ 1	17:05:19	CREATE TABLE transactions_products (id_transaction VARCHAR(40), id_product INT, -- quantity INT NOT NULL, -- No lo incorporamos por ah...	0 row(s) affected
✓ 2	17:05:55	DESCRIBE transactions_products	2 row(s) returned

3. Cargar datos a tabla puente

- Para cargar la tabla puente utilizamos necesitamos descomponer la columna 'product_ids' de la tabla 'transactions', en varios registros, uno para cada producto.
- El problema principal es que desconocemos la cantidad de productos que puede contener una transacción. **Utilizamos la función JSON_TABLE()**, que al recibir un listado de número en formato [4,5,7], puede generar un array con esos números, el cual luego puede generar un registro para cada elemento del array.
- Usamos la función CONCAT junto con REPLACE para formatear la información, agregando paréntesis cuadrados al registro de 'products_id' y eliminando los espacios en blanco entre los números, los cuales causan errores en esta función.
- **El símbolo \$ en las funciones JSON es el punto de referencia donde se inicia el documento.**
- Usando '\$[*]' indicamos que queremos acceder a todos los datos del arreglo construido. Los arreglos comienzan enumerando sus ítems desde el número cero. Si usáramos '\$[0]' obtendríamos el primer ítem del arreglo, mientras que '\$[*]' hace referencia a todo el arreglo.
- Con 'PATH '\$[*]'' indicamos que queremos obtener todos los elementos del arreglo JASON y convertirlos en registros separados.

SQL File 8*

```

179 -- IMPORTACION DE DATOS A TABLA TRANSACTIONS_PRODUCTS
180 INSERT INTO transactions_products (id_transaction, id_product)
181 SELECT
182     t.id AS id_transaction, -- lo podemos obtener directamente de la tabla 'transactions'
183     CAST(j.id_product2 AS UNSIGNED) AS id_product -- hacemos un cambio de tipo de datos con CAST, de lo obtenido en JSON_TABLE a un numero positivo
184 FROM transactions AS t
185 JOIN JSON_TABLE(
186     CONCAT('[', REPLACE(t.product_ids, ' ', ', '), ']'), -- agrego parentesis cuadrados, y elimino espacios entre productos
187     '$[*]' -- todos los elementos dentro del array
188     COLUMNS (id_product2 VARCHAR(255) PATH '$') -- los elementos seleccionados los quiero como registros separados
189 ) AS j;
190
191 SELECT *
192 FROM transactions_products;

```

Result Grid

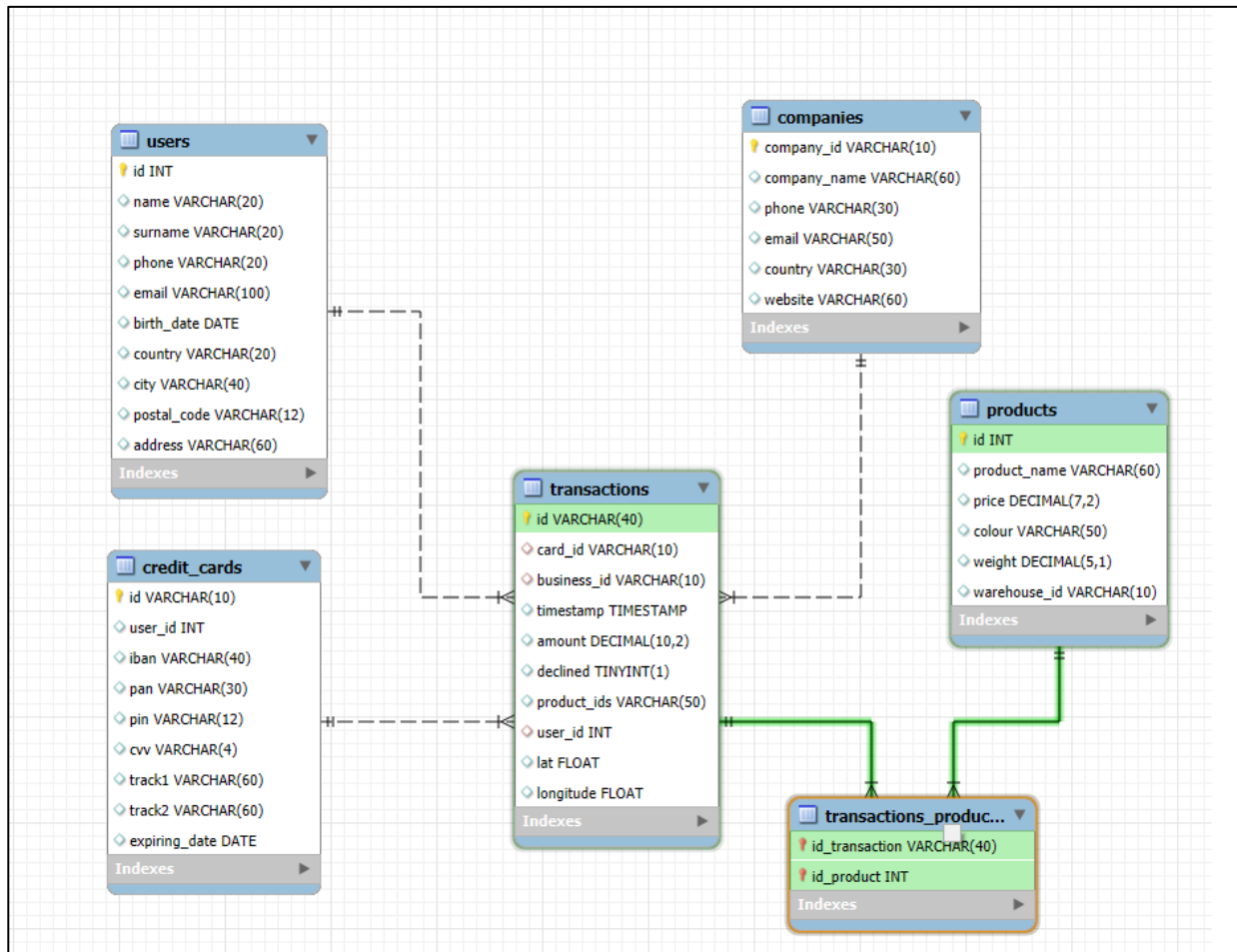
id_transaction	id_product
02C6201E-D90A-1859-B4EE-88D2986D3B02	1
122DC333-E19F-D629-DCD8-9C54CF1EBB9A	1
1753A288-9FC1-52E6-5C39-A1FFB97B0D3A	1
1A6CECFB-2E3A-65A3-72D9-2FDB58A1E4BA	1
1EA2B262-D507-AD14-4374-4D532967113F	1
23CF8ED3-402C-7C54-59CD-DB505C5CCCCCE	1
2A5A3001-104F-1D1F-7852-5BA80186986F	1
2F386AB6-147D-EB08-FE8D-9A4E2EA9DBD5	1
2F499B4D-4DC7-B337-010D-8B7471812A80	1
2F74E99B-9337-7F74-5208-5DEC81A174FD	1
331A8A52-52D4-D323-0388-1A97C982E441	1
35DE2442-5AEB-9C73-E8DE-BC31DEA8C4DE	1

transactions_products 57 x

Output

#	Time	Action	Message
1	00:58:49	INSERT INTO transactions_products (id_transaction, id_product) SELECT t.id AS id_transaction, -- lo podemos obtener directamente de la tabla '...	1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0
2	00:58:49	SELECT * FROM transactions_products LIMIT 0, 1000	1000 row(s) returned

Incluimos el modelo relacional que nos ha quedado, al unir todas las tablas correspondientes:



Ahora con nuestra base de datos completa, iniciamos las consultas solicitadas.

- Exercici 1

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

- Para esta subconsulta, vamos a la tabla 'transactions' y agrupamos los datos por 'user_id'. Haciendo un simple conteo en cada grupo, podemos ver cuantas veces se repite el 'user_id' en la tabla de transacciones.
- Utilizando la clausula HAVING, imponemos condiciones sobre la agrupación, para obtener los resultados de los usuarios que tengan más de 30 transacciones.
- La anterior consulta, sirve como input a la consulta exterior, que nos entrega toda la información de los usuarios, sin repeticiones, donde el 'id' de usuario se encuentra en el listado entregado por la subconsulta anterior.

The screenshot shows a database IDE window titled 'tasca4'. The SQL editor contains the following query:

```
201
202 /* NIVEL 1 - EJERCICIO 1: Subconsulta*/
203 /*Realitza una subconsulta que mostri tots els usuaris amb més de 30 transaccions utilitzant almenys 2 taules.*/
204 SELECT DISTINCT *
205 FROM users AS u
206 WHERE u.id IN
207 (
208     SELECT user_id
209     FROM transactions
210     GROUP BY user_id
211     HAVING COUNT(id) > 30
212 );
213
```

Below the editor is the 'Result Grid' showing the results of the query. The table has 11 columns: id, name, surname, phone, email, birth_date, country, city, postal_code, and address. The results are as follows:

id	name	surname	phone	email	birth_date	country	city	postal_code	address
92	Lynn	Riddle	1-387-885-4057	vitae.aliquet@outlook.edu	1984-09-21	United States	Bozeman	61871	P.O. Box 712, 7907 Est St.
267	Ocean	Nelson	079-481-2745	aenean@yahoo.com	1991-12-26	Canada	Charlottetown	85X 3P4	Ap #732-8357 Pede, Rd.
272	Hedwig	Gilbert	064-204-8788	sem.eget@icloud.edu	1991-04-16	Canada	Tuktoyaktuk	Q4C 3G7	P.O. Box 496, 5145 Sapien Road
275	Kenyon	Hartman	082-871-7248	convallis.ante.lectus@yahoo.com	1982-08-03	Canada	Richmond	R8H 2K2	8564 Facilisi. St.
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Below the result grid, there is an 'Output' section showing the execution details:

```
users 19 x
Output
Action Output
# Time Action Message
1 18:06:17 SELECT DISTINCT * FROM users AS u WHERE u.id IN ( SELECT user_id FROM transactions GROUP BY user_id HAVING COUNT(id) > 30 )... 4 row(s) returned
```

- Exercici 2

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

- Primero creamos una subconsulta, para obtener el 'company_id' de la compañía cuyo nombre es 'Donec Ltd'.
- Luego, vinculamos las tablas 'credit_cards' y 'transactions' con un JOIN, y seleccionamos 'iban' y 'media_compras' (como el promedio de 'amount' en cada 'iban'.
- Agrupamos por iban y ordenamos los resultados de manera descendente.
- En este caso el listado es de solo un registro, lo que significa que la compañía 'Donec Ltd' solamente hay una tarjeta de crédito vinculada a esta compañía en las transacciones.

The screenshot shows a SQL IDE window titled 'tasca4*'. The query editor contains the following SQL code:

```
215 /* NIVEL 1 - EJERCICIO 2 Consulta*/
216 /*Mostra la mitjana d'amount per IBAN de les targetes de crèdit a la companyia Donec Ltd, utilitza almenys 2 taules.*/
217 • SELECT iban, ROUND(AVG(t.amount),2) AS media_compras -- agregar nombre compañía cuando esté todo conectado.
218 FROM credit_cards AS cc
219 INNER JOIN transactions AS t
220 ON cc.id = t.card_id
221 AND business_id IN
222 (
223     SELECT company_id
224     FROM companies
225     WHERE company_name = 'Donec Ltd'
226 )
227 )
228 GROUP BY iban
229 ORDER BY media_compras DESC;
230
```

Below the query editor, the 'Result Grid' shows the following data:

iban	media_compras
PT87806228135092429456346	203.72

At the bottom, the 'Output' section shows the execution message:

```
1 18:35:11 SELECT iban, ROUND(AVG(t.amount),2) AS media_compras -- agregar nombre compañía cuando esté todo conectado. FROM credit_cards AS cc INNE... 1 row(s) returned
```

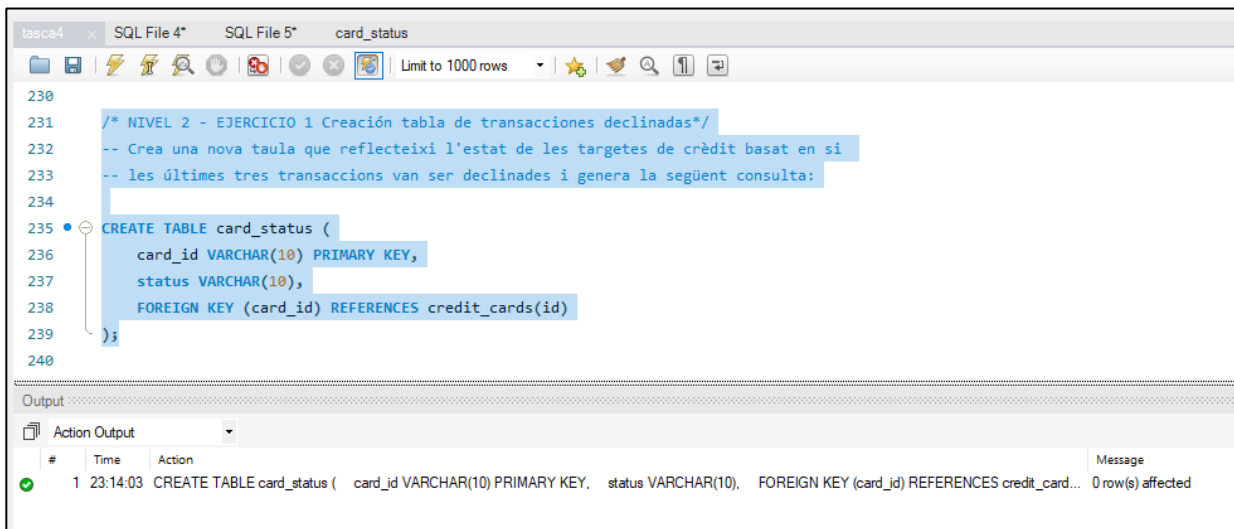
Nivell 2

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:

Exercici 1

Quantes targetes estan actives?

- Definimos una tarjeta como 'inactiva' cuando sus ultimas 3 transacciones han sido declinadas. Así, podremos generar una alerta, para avisarle al cliente que su tarjeta ya no funciona.
- Primero que nada, creamos una nueva tabla, que contenga los identificadores únicos de cada tarjeta de crédito 'card_id' y una variable llamada 'status', que indica si las últimas 3 transacciones han sido declinadas (tarjeta inactiva) o no (activa). Luego trabajaremos para poder completar esta tabla y realizar la consulta solicitada.
- En esta nueva tabla, el identificador 'card_id', funciona como llave primaria y foránea al mismo tiempo, referenciando a la tabla 'credit_cards'. Esto, porque cada estado de tarjeta está vinculado siempre a una sola tarjeta, en una relación 1:1 (siempre que tengamos una tarjeta en la tabla 'credit_cards' podremos consultar su estado en 'card_status').



The screenshot shows a SQL IDE window with a file named 'card_status'. The SQL code is as follows:

```
230
231 /* NIVEL 2 - EJERCICIO 1 Creación tabla de transacciones declinadas*/
232 -- Crea una nova taula que reflecteixi l'estat de les targetes de crèdit basat en si
233 -- les últimes tres transaccions van ser declinades i genera la següent consulta:
234
235 CREATE TABLE card_status (
236     card_id VARCHAR(10) PRIMARY KEY,
237     status VARCHAR(10),
238     FOREIGN KEY (card_id) REFERENCES credit_cards(id)
239 );
240
```

The 'Output' pane at the bottom shows the execution result:

#	Time	Action	Message
1	23:14:03	CREATE TABLE card_status (card_id VARCHAR(10) PRIMARY KEY, status VARCHAR(10), FOREIGN KEY (card_id) REFERENCES credit_card...	0 row(s) affected

- Para conseguir la información deseada tenemos que analizar las transacciones de cada tarjeta de crédito por separado, ordenando las transacciones por fecha y evaluar las 3 ultimas transacciones.
- Lo primero que haremos, es crear una subconsulta, que nos entregue, separado para cada tarjeta, las transacciones ordenadas de más reciente a más antigua, indicando si esta fue declinada o no.
- Además, crearemos una columna que enumere, dentro de cada tarjeta, que numero de transacción es cada registro (siendo 1 el registro más nuevo).

- Utilizamos la función de ventana ROW_NUMBER(), agrupando los datos en cada ventana por 'card_id', y ordenando por 'timestamp' de forma descendiente, realizando así un ranking de tiempo sin repetición.
- Para verificar que lo hemos hecho bien, incluimos a 'timestamp' en el resultado de nuestra consulta, y si resulta, lo eliminaremos de la subconsulta en el código final.
- Bajando un poco en los resultados, podemos verificar que la enumeración se reinicia al cambiar la tarjeta que está siendo evaluada.

SQL Query:

```

/* NIVEL 2 - EJERCICIO 1 Creación tabla de transacciones declinadas*/
-- 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:
-- Subconsulta para ranking de transacciones por tarjeta, desde más nueva a más antigua.
SELECT card_id,
       declined,
       ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS orden,
       timestamp
FROM transactions;

```

card_id	declined	orden	timestamp
CcU-2938	0	6	2021-09-28 02:24:34
CcU-2938	0	7	2021-09-24 08:33:44
CcU-2938	0	8	2021-09-18 00:31:49
CcU-2938	0	9	2021-09-15 05:23:32
CcU-2938	0	10	2021-08-28 23:42:24
CcU-2938	0	11	2021-07-27 17:14:52
CcU-2938	0	12	2021-07-25 12:34:59
CcU-2938	0	13	2021-07-18 08:20:59
CcU-2938	0	14	2021-07-11 00:19:27
CcU-2938	0	15	2021-07-07 17:43:16
CcU-2938	0	16	2021-07-03 19:56:27
CcU-2938	0	17	2021-05-28 15:21:36
CcU-2938	0	18	2021-05-19 01:05:28
CcU-2938	1	19	2021-05-09 10:25:08
CcU-2938	0	20	2021-04-18 11:05:12
CcU-2938	0	21	2021-04-09 19:23:41
CcU-2938	0	22	2021-04-01 07:27:49
CcU-2938	0	23	2021-03-28 01:11:44
CcU-2938	0	24	2021-03-23 01:12:06
CcU-2945	0	1	2022-02-04 15:52:56
CcU-2945	1	2	2021-06-15 00:26:29
CcU-2952	0	1	2022-01-30 15:16:36
CcU-2952	1	2	2021-05-06 05:33:39
CcU-2959	0	1	2022-03-16 14:01:36

Annotations:

- Transacciones más recientes:** Indicated by a blue box around the top 23 transactions for card_id CcU-2938.
- Transacciones más antiguas:** Indicated by a blue box around the bottom 2 transactions for card_id CcU-2938.
- Reinicio de contador 'orden':** Indicated by a red box around the first transaction for card_id CcU-2945, showing the counter resets to 1.

Output:

```

1 21:34:35 SELECT card_id, declined, ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS orden, timestamp FROM ... 587 row(s) returned

```

- Ahora que tenemos las transacciones ordenadas por fecha, para cada tarjeta de crédito, usaremos la consulta anterior como una subconsulta con la cláusula WITH.
- Comenzamos definiendo la subconsulta anterior como 'consultas_ordenadas' y la usamos en una nueva consulta, la cual, **agrupando nuevamente por card_id**, suma las consultas solo en los casos (con la clausula CASE) en que 'declined' es verdadero, para las transacciones rankeadas iguales o menores que 3.

SQL File 10" x

Limit to 1000 rows

```

239
240
241 -- Obtenemos un listado de todas las tarjetas, y de cuantas de las tres ultimas 3 transacciones han sido declinadas
242 WITH tarjetas_ordenadas AS
243 (
244     SELECT
245         card_id,
246         declined,
247         ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS orden
248     FROM transactions
249 )
250 SELECT card_id,
251 SUM(CASE WHEN declined = TRUE THEN 1 ELSE 0 END) AS suma_ultimas_tres_declinadas
252 FROM tarjetas_ordenadas
253 WHERE orden <= 3
254 GROUP BY card_id
255 ORDER BY suma_ultimas_tres_declinadas DESC;
256

```

Result Grid

card_id	suma_ultimas_tres_declinadas
CdU-2945	1
CdU-2952	1
CdU-2966	1
CdU-2973	1
CdU-2980	1
CdU-2987	1
CdU-3001	1
CdU-3008	1
CdU-3015	1
CdU-3022	1
CdU-3029	1
CdU-3036	1
CdU-3043	1
CdU-3050	1
CdU-3057	1

Result 12 x

Output

Action Output

#	Time	Action	Message
1	16:05:55	WITH tarjetas_ordenadas AS (SELECT card_id, declined, ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS ord...	275 row(s) returned

- Finalmente, para cargar la tabla, combinamos el resultado anterior como otra subconsulta a la cual llamaremos 'ultimas_tres'.
- Hacemos una consulta en la que pedimos el 'card_id' y creamos las categorías 'inactiva' o 'activa' para 'status', según la suma de las últimas tres transacciones declinadas.
- Para ingresar los datos a la tabla, al comienzo de las tres consultas, usamos la clausula INSERT, para ingresar los datos a 'card_id' y 'status', a la tabla 'card_status' que creamos anteriormente.

SQL File 10*

Limit to 1000 rows

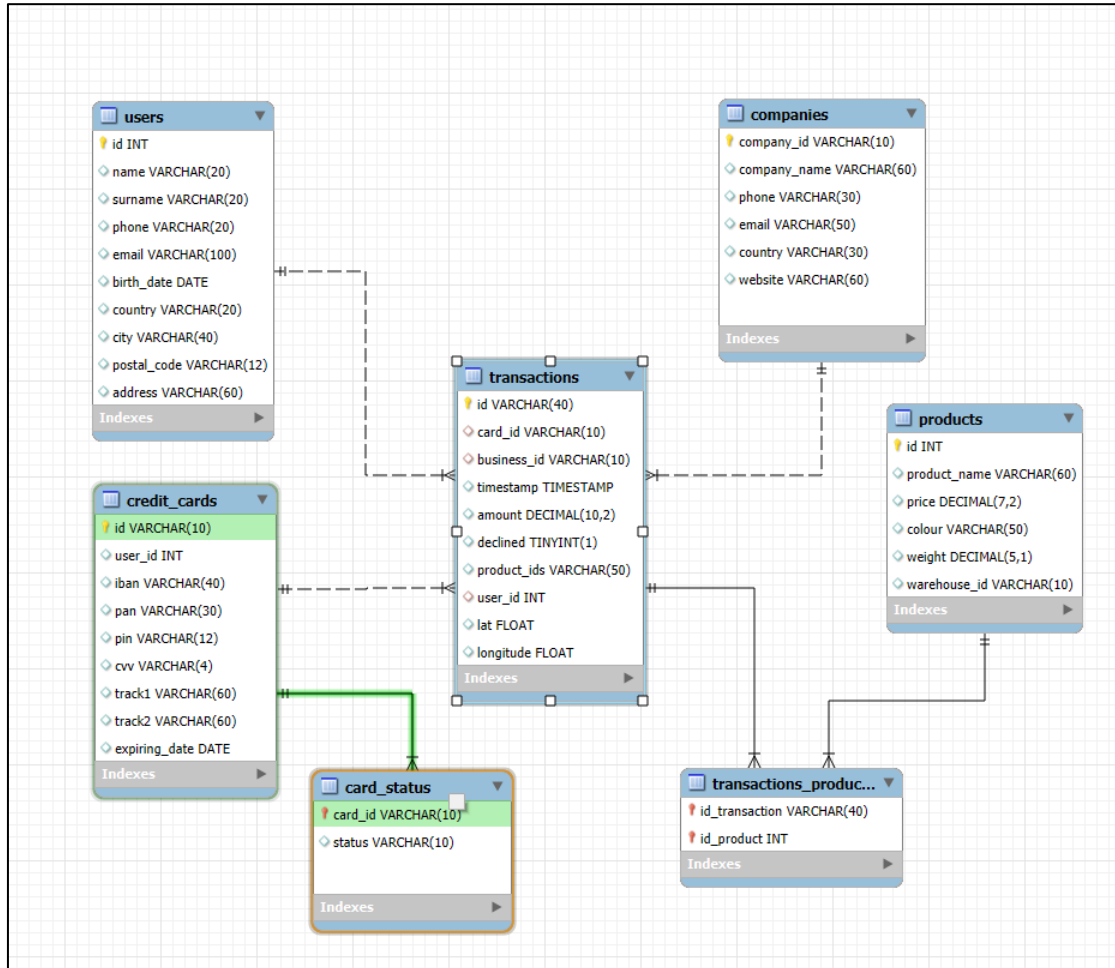
```
256
257 -- Ingreso de datos a tabla 'card_status'
258 • INSERT INTO card_status (card_id, status)
259 WITH tarjetas_ordenadas AS (
260     SELECT
261     card_id,
262     declined,
263     ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS orden
264     FROM transactions
265 ),
266 ultimas_tres AS (
267     SELECT
268     card_id,
269     SUM(CASE WHEN declined = TRUE THEN 1 ELSE 0 END) AS suma_ultimas_tres_declinadas
270     FROM tarjetas_ordenadas
271     WHERE orden <= 3
272     GROUP BY card_id)
273 SELECT card_id,
274 CASE WHEN suma_ultimas_tres_declinadas = 3 THEN 'inactiva' ELSE 'activa' END AS status
275 FROM ultimas_tres;
276 ---
277
```

Output

Action Output

#	Time	Action	Message
1	16:07:04	INSERT INTO card_status (card_id, status) WITH tarjetas_ordenadas AS (SELECT card_id, declined, ROW_NUMBER() OVER (PARTITION BY ca...	275 row(s) affected Records: 275 Duplicates: 0 Warnings: 0

- Revisamos el esquema relacional antes de realizar la consulta solicitada. Notamos que aunque la relación entre 'credit_cards' y 'card_status' aparece como 1:N en el diagrama, en realidad esta relación siempre será 1:1. Cualquier tarjeta de crédito que esté en 'credit_card' ha realizado al menos 1 transacción, y por lo tanto se puede definir un estado de la tarjeta para ella.



Ahora finalmente realizamos la consulta correspondiente para saber cuántas tarjetas están activas, y obtenemos que **todas las 275 tarjetas de la base de datos están activas.**

```

SQL File 7*
288
289 -- Consulta: Cuántas tarjetas están activas?
290 • SELECT COUNT(*) AS numero_tarjetas_activas
291 FROM card_status
292 WHERE status = 'activa';

```

numero_tarjetas_activas
275

```

Output
Action Output
# Time Action Message
1 23:53:50 SELECT COUNT(*) AS numero_tarjetas_activas FROM card_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:

Exercici 1

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

- Ya hemos creado la tabla puente necesaria y la hemos poblado con datos.
- Revisamos los datos de la tabla 'transaction_products' ordenando los datos por 'id_product'. Notamos que cada transacción ha sido convertida en varios registros, que combinan el identificador de la transacción, con el identificador de cada producto que hay en ella, por separado. En el listado, podemos ver que los productos se repiten en múltiples transacciones:

The screenshot shows a SQL IDE window titled 'tasca4*' with a file named 'SQL File 7*'. The query editor contains the following SQL code:

```
293
294  /* NIVEL 3 - EJERCICIO 1: Creación y consulta en tabla puente*/
295  /*Necessitem conèixer el nombre de vegades que s'ha venut cada producte*/
296
297  SELECT *
298  FROM transactions_products
299  ORDER BY id_product;
```

The 'Result Grid' shows the results of the query. It has two columns: 'id_transaction' and 'id_product'. The data is as follows:

id_transaction	id_product
D5E51BD3-4284-8097-8887-4A656C3992A8	1
DA247F32-9723-9BB6-B770-BC39C252CE15	1
E2D88D98-9328-ACDA-39A3-802C7BB36D34	1
E308729C-4A62-08C9-45C6-58C9479D4626	1
E6E14C3A-1AE7-7B62-57D8-16A6F19C995F	1
E8A862CE-AC8B-A489-620C-DDD7FEB16AA2	1
EAC77318-72E4-51E3-AB5A-9BCCBA39C8F2	1
EAE19DC1-C847-6D79-673D-00E7696AC336	1
EE2D8157-D63E-6A91-A363-9CF68B530AD3	1
F5ACD74B-4275-5AA1-2414-6EF417636B98	1
FD89D51B-AE8D-77DC-E450-88083FBD3187	1
14CAE5B5-8FB1-3E4A-4C85-0EA4167534F4	2
167DE938-B1C2-428D-FA3D-96A45CD41B22	2
1717FD68-ADAD-7082-A748-9112BE892CCC	2
17561134-37A4-E316-2C3C-C5541EDAD761	2
1788C881-EE67-9BAB-377A-4C78E772160B	2
1A6CECFB-2E3A-65A3-72D9-2FDB58A1E4BA	2
1C9EE7A2-EA73-9E54-DAC7-D08632543C68	2
24EB2B70-3602-FDA8-54D5-9CD7F3CA523C	2
32A9BDCA-1390-2BE1-82C8-AD514E38A203	2

The 'Output' section at the bottom shows the execution details:

#	Time	Action	Message
1	00:03:25	SELECT * FROM transactions_products ORDER BY id_product LIMIT 0, 1000	1000 row(s) returned

- Ahora basta con realizar la consulta, contando la cantidad de 'id_transaction' que hay en un grupo, al agrupar por 'id_producto'. Vamos a asumir que no solamente se requiere de la cantidad por código de producto, sino que deseamos tener toda la información de los productos junto a su cantidad, lo que nos obliga a hacer una JOIN de la tabla puente con la tabla 'products'.

tasca4* x SQL File 7*

Limit to 1000 rows

```

300
301 • SELECT p.*, COUNT(id_transaction) AS cantidad_ventas_producto
302 FROM transactions_products AS tp
303 INNER JOIN products AS p
304 ON tp.id_product = p.id
305 GROUP BY id_product;

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: [IA](#)

	id	product_name	price	colour	weight	warehouse_id	cantidad_ventas_producto
1	Direwolf Stannis	161.11	#7c7c7c	1.0	WH-4	61	
2	Tarly Stark	9.24	#919191	2.0	WH-3	65	
3	duel tourney Lannister	171.13	#d8d8d8	1.5	WH-2	51	
5	skywalker ewok	171.22	#dbdbdb	3.2	WH-0	49	
7	north of Casterly	63.33	#b7b7b7	0.6	WH--2	54	
11	Karstark Dorne	49.70	#141414	2.7	WH--6	48	
13	palpatine chewbacca	139.59	#2b2b2b	1.0	WH--8	60	
17	skywalker ewok sith	91.89	#7c7c7c	3.2	WH--12	61	
19	dooku solo	60.33	#3f3f3f	0.6	WH--14	49	
23	riverlands north	169.96	#545454	2.7	WH--18	68	
29	Tully maester Tarly	167.20	#111111	3.2	WH--24	49	
31	Lannister	85.02	#3f3f3f	0.6	WH--26	47	
37	Direwolf Littlefinger	26.66	#aaaaaa	1.0	WH--32	51	
41	Lannister Barratheon ...	141.01	#fcfcfc	3.2	WH--36	53	
43	duel	59.80	#5b5b5b	0.6	WH--38	65	
47	Tully	82.15	#919191	2.7	WH--42	62	
53	kingsblood Littlefinger ...	137.81	#3a3a3a	3.2	WH--48	58	
59	Direwolf Stannis	114.77	#cbcbcb	2.7	WH--54	45	
61	Winterfell Lannister	28.01	#848484	1.0	WH--56	57	
67	Winterfell	195.94	#1c1c1c	0.6	WH--62	68	
71	Tully Dorne	103.73	#424242	2.7	WH--66	54	
73	Dorne bastard	114.09	#848484	1.0	WH--68	47	

Result 56 x

Output

Action Output

#	Time	Action	Message
1	00:11:08	SELECT p.*, COUNT(id_transaction) AS cantidad_ventas_producto FROM transactions_products AS tp INNER JOIN products AS p ON tp.id_produ...	26 row(s) returned