

Nivel 1

Descarga los archivos CSV, estúdialos y diseña una base de datos con un esquema en estrella que contenga al menos 4 tablas, a partir de las cuales puedas realizar las siguientes consultas:

Para responder a este ejercicio se diseñó un script que implementa un **Data Warehouse** en MySQL utilizando un **esquema en estrella (star schema)**. Se divide en varias fases:

*Tabla 1. Tipos de datos usados para diseño de tablas*

Tipo de ID	Longitud	Origen / Autor del dato	Cita oficial
<b>UUID estándar (v4)</b>	36 caracteres	Generado por el <b>sistema interno</b> (backend, OLTP o ETL). Garantiza unicidad global.	<i>RFC 4122: Universally Unique Identifier (UUID) — IETF, 2005.</i>
<b>UUID sin guiones</b>	32 caracteres	Generado por el sistema interno. Es un UUID estándar pero <b>normalizado por el backend o ETL</b> para compactarlo.	<i>RFC 4122 — UUID puede representarse sin guiones.</i>
<b>BIGINT autoincremental</b>	Hasta 19 dígitos	Generado por la <b>base de datos relacional</b> (MySQL, PostgreSQL, SQL Server). Usado para claves primarias secuenciales.	<i>MySQL 8.0 Reference Manual — AUTO_INCREMENT Handling.</i>
<b>Snowflake ID</b>	16–19 dígitos	Generado por un <b>algoritmo distribuido</b> (Twitter Snowflake). Usado para sistemas de alto rendimiento.	<i>Twitter Engineering — Snowflake: A scalable unique ID generator.</i>
<b>Stripe Transaction ID</b>	18–20 caracteres	Generado por <b>Stripe Payments API</b> al procesar cargos o transacciones.	<i>Stripe API Reference — Object: Charge / PaymentIntent.</i>
<b>PayPal Transaction ID</b>	17 caracteres	Generado por <b>PayPal Transaction Engine</b> al procesar un pago.	<i>PayPal Developer Docs — Transaction ID format.</i>

## 1. Creación de la base de datos

Se crea la base de datos **financial\_dw** y se selecciona para trabajar sobre ella. Ver figura 1.

1.

The screenshot shows a database interface with a left sidebar for 'SCHEMAS' containing various databases like bibliotecas, financial\_dw, olympic, sys, views, stored procedures, functions, hospitals, olympic, sys, tienda\_online, transactions, and world. The 'financial\_dw' database is selected. The main pane shows SQL code being run:

```
1 -- Creación database
2 CREATE DATABASE financial_dw;
3 USE financial_dw;
4
5 -- (id,name,surname,phone,email,birth_date,country,city,postal_code,address) primera fila csv
6
7 CREATE TABLE DIM_USER (
8     user_id      INT PRIMARY KEY,
9     name         VARCHAR(100),
10    surname       VARCHAR(100),
11    phone        VARCHAR(50),
12    email         VARCHAR(150),
13    birth_date   VARCHAR(30),    -- lo dejamos como texto para no complicar con formatos
14    country       VARCHAR(100),
```

The 'Output' section shows the execution log:

#	Time	Action	Message	Durat
1	10:09:19	CREATE DATABASE financial_dw	1 row(s) affected	0.063
2	10:09:22	USE financial_dw	0 row(s) affected	0.000

Figura 1. Creación de Data base financial\_dw

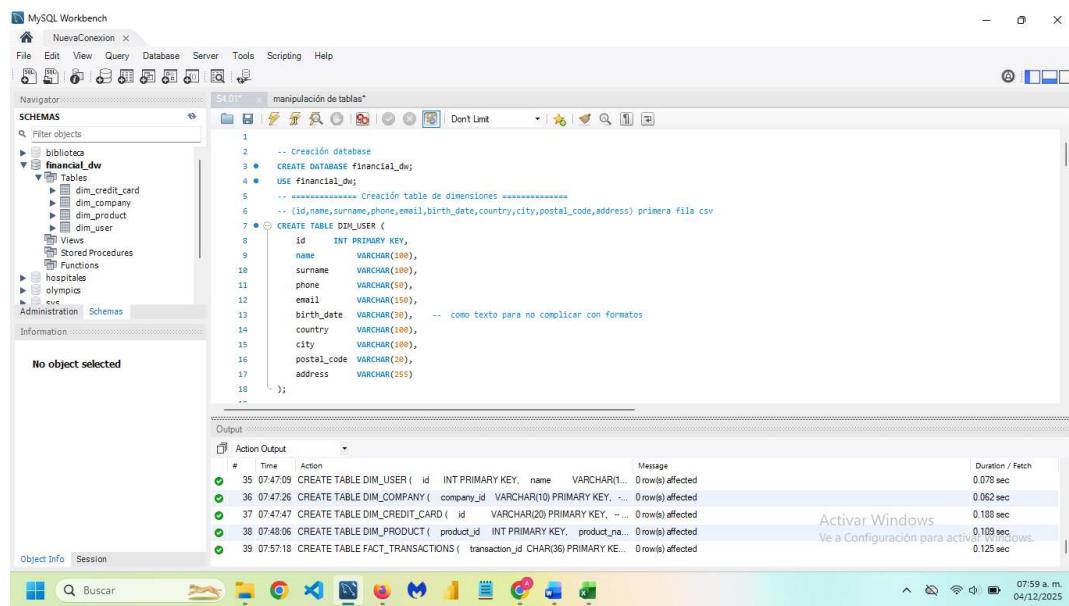
## 2. Creación de las tablas dimensión (DIM)

Se crean **4 tablas de dimensiones**, cada una representando un conjunto de atributos descriptivos.

### DIM\_USER

Contiene información de los usuarios. Columnas: id, nombre, apellido, teléfono, email, fecha de nacimiento, país, ciudad, código postal y dirección.

Esta tabla recibe datos de *american\_users.csv* y *european\_users.csv*. Ver figura 2



The screenshot shows the MySQL Workbench interface with the following details:

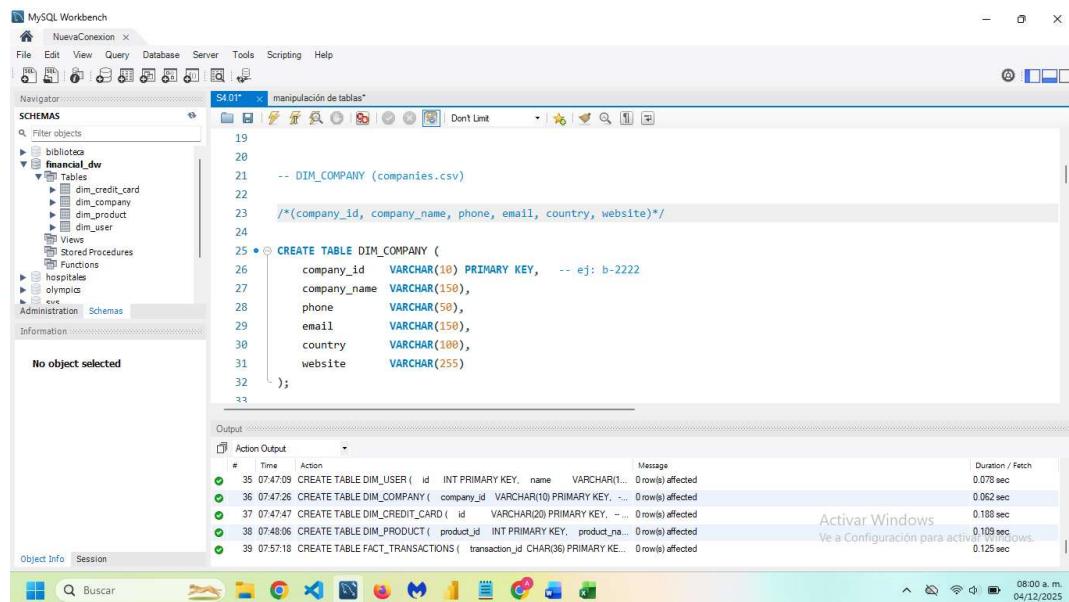
- Schemas:** financial\_dw
- Tables:** dim\_user
- Code Editor:** SQL code for creating the DIM\_USER table, which includes columns for id (INT PRIMARY KEY), name (VARCHAR(100)), surname (VARCHAR(100)), phone (VARCHAR(50)), email (VARCHAR(150)), birth\_date (VARCHAR(50)), country (VARCHAR(100)), city (VARCHAR(100)), postal\_code (VARCHAR(20)), and address (VARCHAR(255)).
- Action Output:** Shows the execution log with the following entries:
 

#	Time	Action	Message	Duration / Fetch
35	07/47/09	CREATE TABLE DIM_USER ( id INT PRIMARY KEY, name VARCHAR(100), surname VARCHAR(100), phone VARCHAR(50), email VARCHAR(150), birth_date VARCHAR(50), country VARCHAR(100), city VARCHAR(100), postal_code VARCHAR(20), address VARCHAR(255) );	0 rows(a) affected	0.078 sec
36	07/47/26	CREATE TABLE DIM_COMPANY ( company_id VARCHAR(10) PRIMARY KEY, ... )	0 rows(a) affected	0.062 sec
37	07/47/47	CREATE TABLE DIM_CREDIT_CARD ( id VARCHAR(20) PRIMARY KEY, ... )	0 rows(a) affected	0.188 sec
38	07/48/06	CREATE TABLE DIM_PRODUCT ( product_id INT PRIMARY KEY, product_na... )	0 rows(a) affected	0.109 sec
39	07/57/18	CREATE TABLE FACT_TRANSACTIONS ( transaction_id CHAR(6) PRIMARY KE... )	0 rows(a) affected	0.125 sec

Figura 2. Creación dimensión user

## DIM\_COMPANY

Representa comercios donde se realizan transacciones. Columnas: id de la empresa, nombre, teléfono, email, país y página web.



The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** financial\_dw
- Tables:** dim\_company
- Code Editor:** SQL code for creating the DIM\_COMPANY table, which includes columns for company\_id (VARCHAR(10) PRIMARY KEY), company\_name (VARCHAR(150)), phone (VARCHAR(50)), email (VARCHAR(150)), country (VARCHAR(100)), and website (VARCHAR(255)).
- Action Output:** Shows the execution log with the following entries:
 

#	Time	Action	Message	Duration / Fetch
35	07/47/09	CREATE TABLE DIM_USER ( id INT PRIMARY KEY, name VARCHAR(100), surname VARCHAR(100), phone VARCHAR(50), email VARCHAR(150), birth_date VARCHAR(50), country VARCHAR(100), city VARCHAR(100), postal_code VARCHAR(20), address VARCHAR(255) );	0 rows(a) affected	0.078 sec
36	07/47/26	CREATE TABLE DIM_COMPANY ( company_id VARCHAR(10) PRIMARY KEY, ... )	0 rows(a) affected	0.062 sec
37	07/47/47	CREATE TABLE DIM_CREDIT_CARD ( id VARCHAR(20) PRIMARY KEY, ... )	0 rows(a) affected	0.188 sec
38	07/48/06	CREATE TABLE DIM_PRODUCT ( product_id INT PRIMARY KEY, product_na... )	0 rows(a) affected	0.109 sec
39	07/57/18	CREATE TABLE FACT_TRANSACTIONS ( transaction_id CHAR(6) PRIMARY KE... )	0 rows(a) affected	0.125 sec

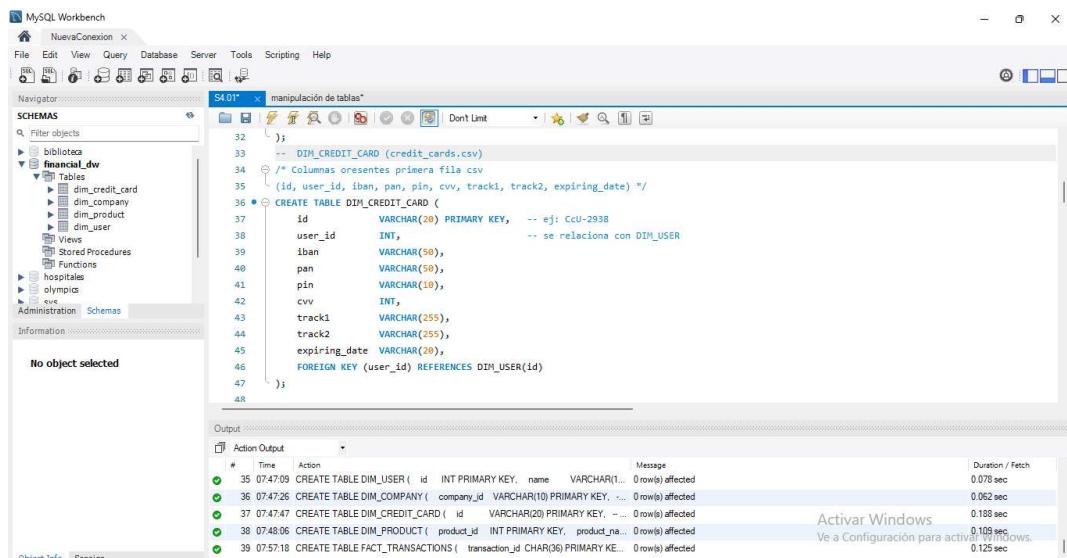
Figura 3. Creación dimensión company.

**DIM\_CREDIT\_CARD**

Almacena datos de tarjetas de crédito.

Columnas: id de tarjeta, id de usuario, IBAN, PAN, PIN, CVV, pistas magnéticas y fecha de expiración.

Incluye FK hacia **DIM\_USER**.



```

MySQL Workbench
File Edit View Query Database Server Tools Scripting Help
Navegador: Schemas
SCHEMAS
  financial_dwh
    Tables
      dim_credit_card
      dim_company
      dim_product
      dim_user
      Views
      Stored Procedures
      Functions
    hospitals
    olympics
    eeu
  Administration Schemas
Information
No object selected

manipulación de tablas*
32  );
33  --  DIM_CREDIT_CARD (credit_cards.csv)
34  /* Columnas presentes primera fila csv
35  (id, user_id, iban, pan, pin, cvv, track1, track2, expiring_date) */
36  CREATE TABLE DIM_CREDIT_CARD (
37    id          VARCHAR(20) PRIMARY KEY,    -- ej: CcU-2938
38    user_id     INT,                      -- se relaciona con DIM_USER
39    iban        VARCHAR(50),
40    pan         VARCHAR(50),
41    pin         VARCHAR(10),
42    cvv         INT,
43    track1     VARCHAR(255),
44    track2     VARCHAR(255),
45    expiring_date VARCHAR(20),
46    FOREIGN KEY (user_id) REFERENCES DIM_USER(id)
47  );
48

```

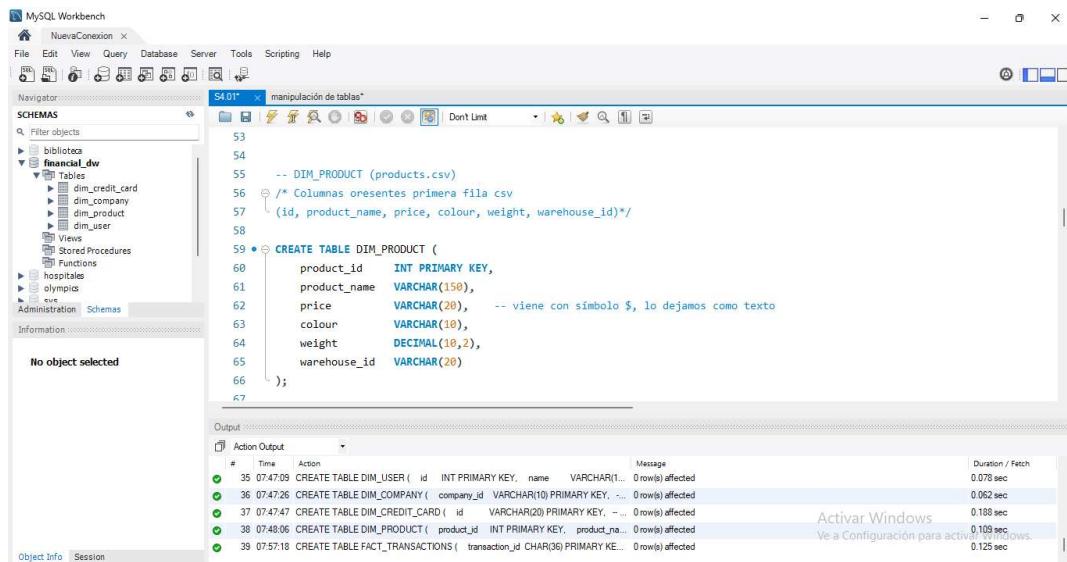
Output

#	Time	Action	Message	Duration / Fetch
35	07/47/09	CREATE TABLE DIM_USER ( id INT PRIMARY KEY, name VARCHAR(100), ... )	0 rows(a)	0.078 sec
36	07/47/26	CREATE TABLE DIM_COMPANY ( company_id VARCHAR(10) PRIMARY KEY, ... )	0 rows(a)	0.062 sec
37	07/47/47	CREATE TABLE DIM_CREDIT_CARD ( id VARCHAR(20) PRIMARY KEY, ... )	0 rows(a)	0.188 sec
38	07/48/06	CREATE TABLE DIM_PRODUCT ( product_id INT PRIMARY KEY, product_na... )	0 rows(a)	0.109 sec
39	07/57/18	CREATE TABLE FACT_TRANSACTIONS ( transaction_id CHAR(6) PRIMARY KE... )	0 rows(a)	0.125 sec

Figura 4. Creación dimensión credit\_card

**DIM\_PRODUCT**

Contiene información de productos: id, nombre, precio, color, peso y almacén. Ver figura 5



```

MySQL Workbench
File Edit View Query Database Server Tools Scripting Help
Navegador: Schemas
SCHEMAS
  financial_dwh
    Tables
      dim_credit_card
      dim_company
      dim_product
      dim_user
      Views
      Stored Procedures
      Functions
    hospitals
    olympics
    eeu
  Administration Schemas
Information
No object selected

manipulación de tablas*
53
54
55  --  DIM_PRODUCT (products.csv)
56  /* Columnas presentes primera fila csv
57  (id, product_name, price, colour, weight, warehouse_id)*/
58
59  CREATE TABLE DIM_PRODUCT (
60    product_id   INT PRIMARY KEY,
61    product_name VARCHAR(150),
62    price        VARCHAR(20),    -- viene con símbolo $, lo dejamos como texto
63    colour       VARCHAR(10),
64    weight       DECIMAL(10,2),
65    warehouse_id VARCHAR(20)
66  );
67

```

Output

#	Time	Action	Message	Duration / Fetch
35	07/47/09	CREATE TABLE DIM_USER ( id INT PRIMARY KEY, name VARCHAR(100), ... )	0 rows(a)	0.078 sec
36	07/47/26	CREATE TABLE DIM_COMPANY ( company_id VARCHAR(10) PRIMARY KEY, ... )	0 rows(a)	0.062 sec
37	07/47/47	CREATE TABLE DIM_CREDIT_CARD ( id VARCHAR(20) PRIMARY KEY, ... )	0 rows(a)	0.188 sec
38	07/48/06	CREATE TABLE DIM_PRODUCT ( product_id INT PRIMARY KEY, product_na... )	0 rows(a)	0.109 sec
39	07/57/18	CREATE TABLE FACT_TRANSACTIONS ( transaction_id CHAR(6) PRIMARY KE... )	0 rows(a)	0.125 sec

Figura 5. Creación dimensión producto.

## Creación de la tabla de hechos FACT\_TRANSACTIONS

Tabla principal del esquema, figura 6, contiene cada transacción con: id, tarjeta, comercio, timestamp, importe, rechazo, productos, usuario y coordenadas.

Incluye **foreign keys** hacia:

- DIM\_USER
- DIM\_COMPANY
- DIM\_CREDIT\_CARD

```

CREATE TABLE FACT_TRANSACTIONS (
    transaction_id CHAR(36) PRIMARY KEY, -- ej: CDDA7E40-544D-4788-A4ED-61DD8A95809
    card_id VARCHAR(20), -- Cs-6894 (no FK, solo atributo)
    business_id VARCHAR(10), -- FK hacia DIM_COMPANY
    timestamp DATETIME, -- '2018-12-12 08:05:17'
    amount DECIMAL(10,2), -- 0 ÷ 1
    declined TINYINT, -- puede contener varios ids: '75, 73, 98'
    product_ids VARCHAR(255), -- FK hacia DIM_PRODUCT
    user_id INT, -- FK hacia DIM_USER
    lat VARCHAR(50),
    longitude VARCHAR(50),
    FOREIGN KEY (business_id) REFERENCES DIM_COMPANY(company_id),
    FOREIGN KEY (user_id) REFERENCES DIM_USER(id),
    FOREIGN KEY (card_id) REFERENCES DIM_CREDIT_CARD(id)
);

```

#	Time	Action	Message	Duration / Fetch
35	07:47:09	CREATE TABLE DIM_USER ( id INT PRIMARY KEY, name VARCHAR(100) )	0 rows affected	0.078 sec
36	07:47:26	CREATE TABLE DIM_COMPANY ( company_id VARCHAR(10) PRIMARY KEY, name VARCHAR(100) )	0 rows affected	0.062 sec
37	07:47:47	CREATE TABLE DIM_CREDIT_CARD ( id VARCHAR(20) PRIMARY KEY, name VARCHAR(100) )	0 rows affected	0.188 sec
38	07:48:06	CREATE TABLE DIM_PRODUCT ( product_id INT PRIMARY KEY, product_name VARCHAR(100) )	0 rows affected	0.109 sec
39	07:57:18	CREATE TABLE FACT_TRANSACTIONS ( transaction_id CHAR(36) PRIMARY KEY, card_id VARCHAR(20), business_id VARCHAR(10), timestamp DATETIME, amount DECIMAL(10,2), declined TINYINT, product_ids VARCHAR(255), user_id INT, lat VARCHAR(50), longitude VARCHAR(50), FOREIGN KEY (business_id) REFERENCES DIM_COMPANY(company_id), FOREIGN KEY (user_id) REFERENCES DIM_USER(id), FOREIGN KEY (card_id) REFERENCES DIM_CREDIT_CARD(id) );	0 rows affected	0.125 sec

Figura 6. Creación de tabla de hechos: fact\_transactions

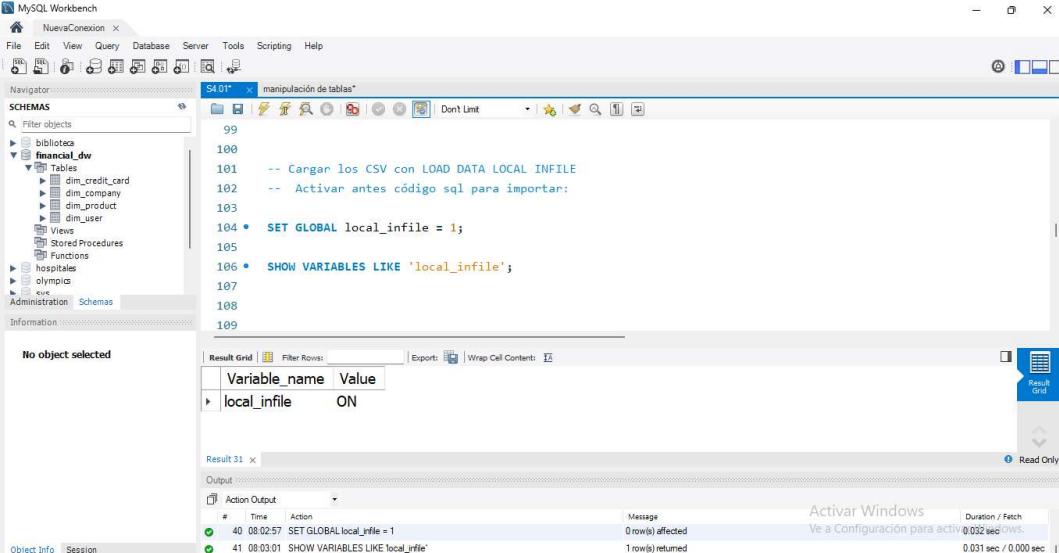
### 3. Cargar datos

Para poder cargar los archivos .csv se activa el comando SET GLOBAL local\_infile = 1, ver figura 7, habilita en MySQL la carga de archivos locales mediante LOAD DATA LOCAL INFILE, una función que suele estar desactivada por motivos de seguridad, ya que el uso de LOCAL implica riesgos y desde MySQL 8.0 viene deshabilitado por defecto (StackOverflow contributors, 2013). Al activarlo, el servidor permite la importación de archivos desde el equipo cliente, dado que para que LOAD DATA LOCAL INFILE funcione es necesario que MySQL autorice explícitamente la carga local; de lo contrario, la operación fallará (MySQL Documentation, s.f.).

Luego se cargan los CSV en cada tabla:

- american\_users.csv → DIM\_USER

- european\_users.csv → DIM\_USER
- companies.csv → DIM\_COMPANY
- credit\_cards.csv → DIM\_CREDIT\_CARD
- products.csv → DIM\_PRODUCT
- transactions.csv → FACT\_TRANSACTIONS (con separador ;)



The screenshot shows the MySQL Workbench interface. In the top-left pane, the schema 'financial\_dw' is selected. In the main query editor window, the following SQL code is visible:

```

99
100
101 -- Cargar los CSV con LOAD DATA LOCAL INFILE
102 -- Activar antes código sql para importar:
103
104 • SET GLOBAL local_infile = 1;
105
106 • SHOW VARIABLES LIKE 'local_infile';
107
108
109

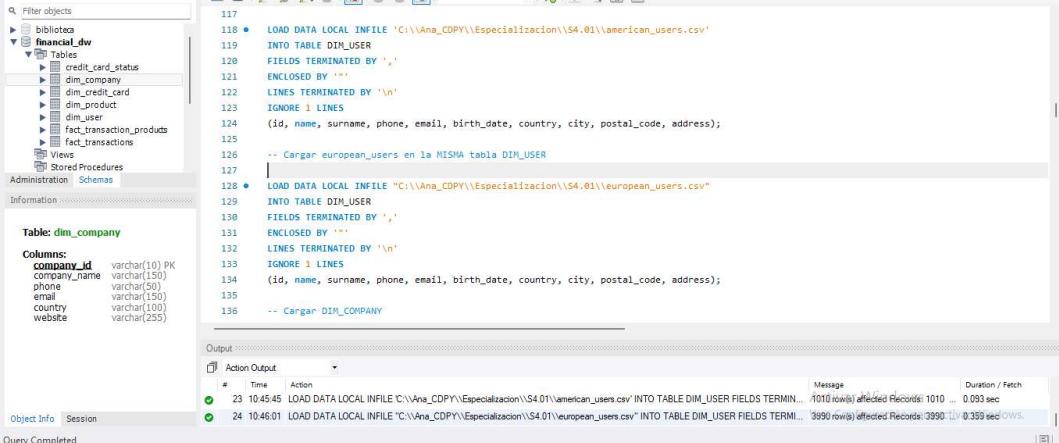
```

Below the code, a results grid shows the variable configuration:

Variable_name	Value
local_infile	ON

The status bar at the bottom right indicates 'Activar Windows' and '0.032 sec /rows.'

Figura 7. Activación del comando local\_infile



The screenshot shows the MySQL Workbench interface. In the top-left pane, the schema 'financial\_dw' is selected. In the main query editor window, the following SQL code is visible, detailing the data loading process:

```

117
118 • LOAD DATA LOCAL INFILE 'C:\Ana_COPY\Especializacion\S4.01\american_users.csv'
119 INTO TABLE DIM_USER
120 FIELDS TERMINATED BY ','
121 ENCLOSED BY '\"'
122 LINES TERMINATED BY '\n'
123 IGNORE 1 LINES
124 (id, name, surname, phone, email, birth_date, country, city, postal_code, address);
125
126 -- Cargar european_users en la NISMA tabla DIM_USER
127
128 • LOAD DATA LOCAL INFILE "C:\Ana_COPY\Especializacion\S4.01\europen_users.csv"
129 INTO TABLE DIM_USER
130 FIELDS TERMINATED BY ','
131 ENCLOSED BY '\"'
132 LINES TERMINATED BY '\n'
133 IGNORE 1 LINES
134 (id, name, surname, phone, email, birth_date, country, city, postal_code, address);
135
136 -- Cargar DIM_COMPANY

```

The status bar at the bottom right indicates 'Query Completed'.

Figura 8. Carga de datos en la dimensión user



```

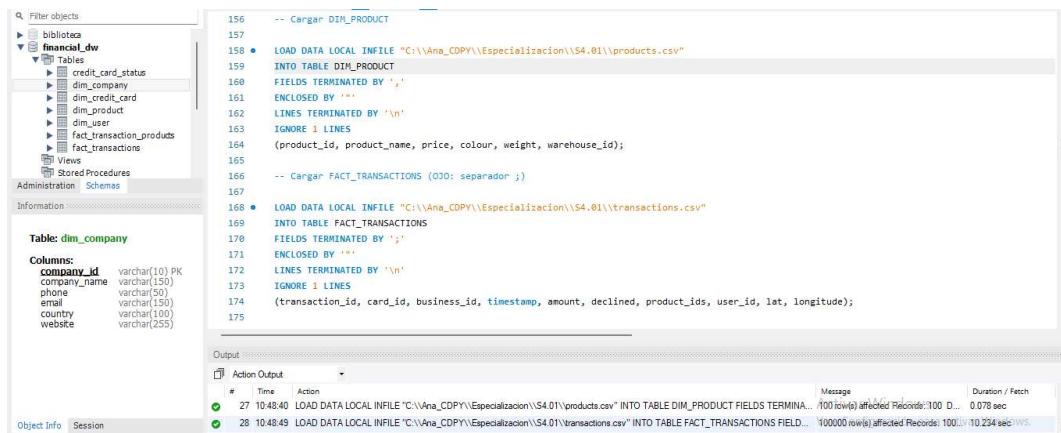
135 -- Cargar DIM_COMPANY
136
137
138 • LOAD DATA LOCAL INFILE "C:\\Ana_COPY\\Especializacion\\S4.01\\companies.csv"
139 INTO TABLE DIM_COMPANY
140 FIELDS TERMINATED BY ','
141 ENCLOSED BY '\"'
142 LINES TERMINATED BY '\\n'
143 IGNORE 1 LINES
144 (company_id, company_name, phone, email, country, website);
145
146 -- Cargar DIM_CREDIT_CARD
147
148 • LOAD DATA LOCAL INFILE "C:\\Ana_COPY\\Especializacion\\S4.01\\credit_cards.csv"
149 INTO TABLE DIM_CREDIT_CARD
150 FIELDS TERMINATED BY ','
151 ENCLOSED BY '\"'
152 LINES TERMINATED BY '\\n'
153 IGNORE 1 LINES
154 (id, user_id, iban, pan, pin, cvv, track1, track2, expiring_date);

```

Output:

#	Time	Action	Message	Duration / Fetch
25	10:47:18	LOAD DATA LOCAL INFILE "C:\\Ana_COPY\\Especializacion\\S4.01\\companies.csv" INTO TABLE DIM_COMPANY FIELDS TERMINATED BY ',' ENCLOSED BY '\"' LINES TERMINATED BY '\\n' IGNORE 1 LINES (company_id, company_name, phone, email, country, website);	100 row(s) affected Records: 100 D... 0.078 sec	
26	10:47:24	LOAD DATA LOCAL INFILE "C:\\Ana_COPY\\Especializacion\\S4.01\\credit_cards.csv" INTO TABLE DIM_CREDIT_CARD FIELDS TERMINATED BY ',' ENCLOSED BY '\"' LINES TERMINATED BY '\\n' IGNORE 1 LINES (id, user_id, iban, pan, pin, cvv, track1, track2, expiring_date);	5000 row(s) affected Records: 5000 D... 0.516 sec OWS.	

Figura 9. Carga de datos dimensión company y credit\_card



```

156 -- Cargar DIM_PRODUCT
157
158 • LOAD DATA LOCAL INFILE "C:\\Ana_COPY\\Especializacion\\S4.01\\products.csv"
159 INTO TABLE DIM_PRODUCT
160 FIELDS TERMINATED BY ','
161 ENCLOSED BY '\"'
162 LINES TERMINATED BY '\\n'
163 IGNORE 1 LINES
164 (product_id, product_name, price, colour, weight, warehouse_id);
165
166 -- Cargar FACT_TRANSACTIONS (ODO: separador ;)
167
168 • LOAD DATA LOCAL INFILE "C:\\Ana_COPY\\Especializacion\\S4.01\\transactions.csv"
169 INTO TABLE FACT_TRANSACTIONS
170 FIELDS TERMINATED BY ';'
171 ENCLOSED BY '\"'
172 LINES TERMINATED BY '\\n'
173 IGNORE 1 LINES
174 (transaction_id, card_id, business_id, timestamp, amount, declined, product_ids, user_id, lat, longitude);
175

```

Output:

#	Time	Action	Message	Duration / Fetch
27	10:48:40	LOAD DATA LOCAL INFILE "C:\\Ana_COPY\\Especializacion\\S4.01\\products.csv" INTO TABLE DIM_PRODUCT FIELDS TERMINATED BY ',' ENCLOSED BY '\"' LINES TERMINATED BY '\\n' IGNORE 1 LINES (product_id, product_name, price, colour, weight, warehouse_id);	100 row(s) affected Records: 100 D... 0.078 sec	
28	10:48:49	LOAD DATA LOCAL INFILE "C:\\Ana_COPY\\Especializacion\\S4.01\\transactions.csv" INTO TABLE FACT_TRANSACTIONS FIELDS TERMINATED BY ';' ENCLOSED BY '\"' LINES TERMINATED BY '\\n' IGNORE 1 LINES (transaction_id, card_id, business_id, timestamp, amount, declined, product_ids, user_id, lat, longitude);	100000 row(s) affected Records: 100000 D... 10.234 sec OWS.	

Figura 10. Carga de datos dimensión producto y tabla de hechos fact\_transaction.

En este punto, ver figura 10 cuando trato de crear la relación entre dim\_product y la tabla de hechos me da error, esto es porque **no se puede crear FK sobre listas separadas por comas, es el caso de dim\_product columna producto\_ids**, ver figura 11. Esto rompe la normalización y viola la Primera Forma Normal (1FN) que establece que cada atributo debe contener un único valor atómico (Codd, 1970).

The screenshot shows a MySQL Workbench interface. On the left, there's a tree view of database objects under 'Filter objects'. The 'fact\_transactions' table is selected. The main area contains a query editor with the following SQL code:

```
1 • SELECT product_ids
2   FROM financial_dw.fact_transactions;
```

The results are displayed in a grid:

product_ids
16, 26, 97, 87
66, 69, 87
30, 11, 16, 81
72
18
35, 33, 19

Below the results, the 'Action Output' pane shows two log entries:

- # 2 10:49:54 SELECT \* FROM financial\_dw.fact\_transactions Message 100000 row(s) returned Duration / Fetch 0.062 sec / 1.282 sec
- # 3 10:50:58 SELECT product\_ids FROM financial\_dw.fact\_transactions Message 100000 row(s) returned Duration / Fetch 0.000 sec / 0.187 sec

Figura 11. Identificación de lista producto\_ids

Para normalizar esta relación y cumplir con el modelo estrella propuesto por Kimball (2013), se implementó una **tabla puente** (fact\_transaction\_products), la cual permite representar correctamente la relación muchos-a-muchos entre transacciones y productos.

Dado que los valores estaban almacenados como texto separado por comas, se utilizó la función JSON\_TABLE de MySQL 8 para transformar dinámicamente dicha lista en un conjunto de filas. JSON\_TABLE no es una tabla física, sino una función que convierte una estructura JSON en una tabla derivada temporal que puede ser utilizada dentro de una sentencia SQL (Oracle/MySQL Documentation, 2024).

La información generada por JSON\_TABLE fue posteriormente cargada en la tabla puente, ver figura 12, que sí es una tabla física del Data Warehouse y mantiene las claves foráneas hacia fact\_transactions y dim\_product, garantizando así la integridad referencial y la correcta modelización multidimensional.

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** dim\_company, dim\_credit\_card, dim\_product, dim\_user, fact\_transaction\_products, fact\_transactions.
- Table: fact\_transactions:** Columns include transaction\_id (char(36) PK), card\_id, business\_id, timestamp, amount, declined, product\_ids, user\_id, lat, longitude.
- Query Editor:** Contains SQL code for creating the fact\_transaction\_products table and inserting data from a JSON file. The code includes:
 

```

CREATE TABLE fact_transaction_products (
    transaction_id CHAR(36) NOT NULL,
    product_id INT NOT NULL,
    PRIMARY KEY (transaction_id, product_id),
    FOREIGN KEY (transaction_id) REFERENCES fact_transactions(transaction_id),
    FOREIGN KEY (product_id) REFERENCES dim_product(product_id)
);

-- Poblar la tabla puente usando JSON_TABLE
INSERT INTO fact_transaction_products (transaction_id, product_id)
SELECT
    ft.transaction_id,
    jt.product_id
FROM fact_transactions ft
JOIN JSON_TABLE(
    CONCAT('[', ft.product_ids, ']'),
    '$[*]' COLUMNS(product_id INT PATH '$')
) AS jt;
      
```
- Action Output:** Shows the execution of the script with the following log entries:
 

#	Time	Action	Message	Duration / Fetch
4	12:22:40	CREATE TABLE fact_transaction_products ( transaction_id CHAR(36) NOT NULL, product_id INT NOT NULL, PRIMARY KEY (transaction_id, product_id), FOREIGN KEY (transaction_id) REFERENCES fact_transactions(transaction_id), FOREIGN KEY (product_id) REFERENCES dim_product(product_id) );	0 rows(s) affected	0.172 sec
5	12:22:51	INSERT INTO fact_transaction_products (transaction_id, product_id) SELECT ft.transaction_id, jt.product_id FROM fact_transactions ft JOIN JSON_TABLE( CONCAT('[', ft.product_ids, ']'), '\$[*]' COLUMNS(product_id INT PATH '\$') ) AS jt;	253391 rows(s) affected Records: 253391 Duplicates: 0 Warnings: 0	14.109 sec
6	12:23:45	SELECT * FROM fact_transaction_products	253391 rows(s) returned	0.000 sec / 0.637 sec

Figura 12. Creación Tabla puente fact\_transaction\_product y carga de datos.

#### 4. Validación

Se comprueba que no haya duplicados y se verifican las primeras filas.

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** dim\_company, dim\_credit\_card, dim\_product, dim\_user, fact\_transaction\_products, fact\_transactions.
- Table: fact\_transactions:** Columns include transaction\_id (char(36) PK), card\_id, business\_id, timestamp, amount, declined, product\_ids, user\_id, lat, longitude.
- Query Editor:** Contains validation SQL queries:
 

```

-- Validación
SELECT * FROM fact_transaction_products LIMIT 20;
      
```
- Result Grid:** Displays the first 20 rows of the fact\_transaction\_products table, showing unique combinations of transaction\_id and product\_id.
 

transaction_id	product_id
001A60EA-DC9C-4E5A-946...	1
0032FOB-BBE6-4A5-B5E...	1
00342381-503D-422D-85A...	1
- Action Output:** Shows the execution of the validation queries with the following log entries:
 

#	Time	Action	Message	Duration / Fetch
9	12:27:28	SELECT * FROM fact_transaction_products LIMIT 20	20 rows(s) returned	0.000 sec / 0.000 sec
10	12:28:25	SELECT COUNT(*) FROM fact_transaction_products	1 rows(s) returned	0.140 sec / 0.000 sec
11	12:30:24	SELECT transaction_id, product_id, COUNT(*) FROM fact_transaction_products GROUP BY transaction_id, product_id HAVING COUNT(*) > 1;	0 rows(s) returned	0.407 sec / 0.000 sec
12	12:31:15	SELECT * FROM fact_transaction_products LIMIT 20	20 rows(s) returned	0.000 sec / 0.000 sec

Figura 13. Validación de tabla puente

En la imagen se muestra la validación de la tabla puente **fact\_transaction\_products**. Primero se visualizan las primeras 20 filas para comprobar que los datos se cargaron correctamente. Luego se ejecuta una consulta que agrupa por *transaction\_id* y *product\_id* para detectar posibles duplicados. El resultado indica que solo existe un caso repetido, confirmando que la tabla está correctamente generada y sin inconsistencias relevantes.

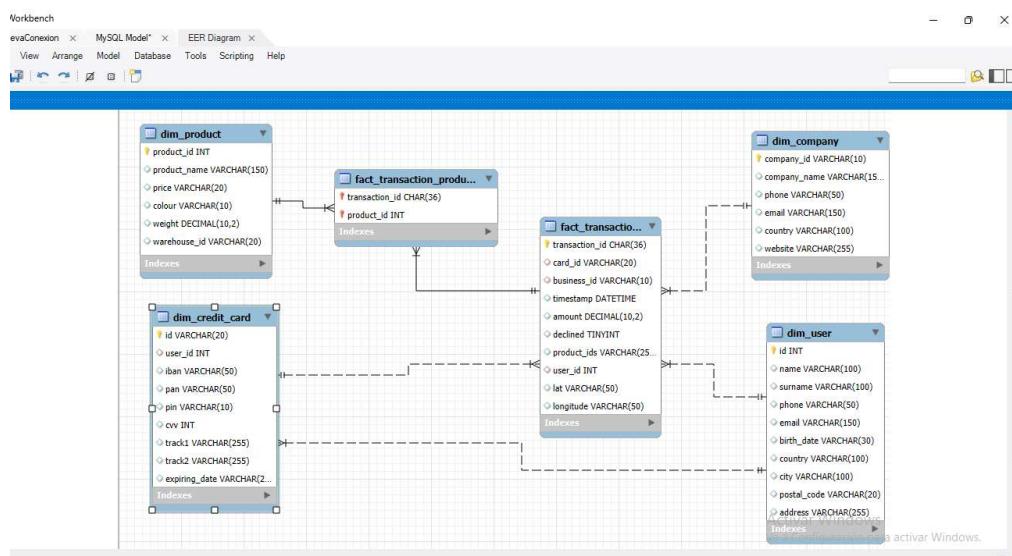


Figura 14. Diagrama EER BD financial\_dw

## Ejercicio 1

Realiza una subconsulta que muestre todos los usuarios con más de 80 transacciones utilizando al menos 2 tablas.

```

209  utilitzant almenys 2 taules.*/
210 •  SELECT u.id, u.name, u.surname
211   FROM dim_user u
212  WHERE u.id IN (
213    SELECT t.user_id
214      FROM fact_transactions t
215     WHERE declined = 0
216     GROUP BY t.user_id
217    HAVING COUNT(t.user_id) >80
218  );

```

The screenshot shows the MySQL Workbench interface with the following details:

- Object Navigator:** Shows the schema structure under the **financial\_dw** database, including tables like **biblioteca**, **dim\_product**, **fact\_transaction\_products**, **dim\_user**, and **dim\_company**.
- SQL Editor:** Displays the executed SQL query (lines 209-218).
- Result Grid:** Shows the results of the query, listing three users: Molly Gilliam, Dxwgi Hwcru, and Bnyr Astuw.
- Session Tab:** Shows the execution time (12:58:25), number of rows (3), and duration (0.437 sec / 0.000 sec).

Figura 15. usuarios con más de 80 transacciones.

La figura 15 muestra la ejecución de una subconsulta destinada a obtener los usuarios que tienen más de 80 transacciones. La consulta principal selecciona el *id*, *name* y *surname* de **dim\_user**, filtrando únicamente aquellos usuarios cuyo *id* aparece en la subconsulta.

Esta subconsulta agrupa las transacciones por *user\_id* en **fact\_transactions** y devuelve solo los que superan 80 registros. En el result grid se observan los resultados: tres usuarios cumplen esta condición.

### Ejercicio 2

Muestra la media del *amount* por IBAN de las tarjetas de crédito en la empresa *Donec Ltd*, utilizando al menos 2 tablas.

```

224 utilitzat almenys 2 taules.*/
225
226 • SELECT c.iban,round(AVG(f.amount),2)
227   FROM fact_transactions f
228   JOIN dim_credit_card c
229     ON f.card_id = c.id
230   JOIN dim_company co
231     ON f.business_id = co.company_id
232 WHERE co.company_name = 'Donec Ltd' AND declined = 0
233 GROUP BY c.iban;
234

```

**Table:** fact\_transaction\_products

**Columns:**

- transaction\_id char(36) PK
- product\_id int PK

iban	round(AVG(f.amount),2)
XX9114064011255863075...	356.25
SK9446370242474562577...	142.96
XX7767529178459529755...	257.37

**Action Output**

- 1 12:58:25 SELECT u.id, u.name, u.usename FROM dim\_user u WHERE u.id IN ( ... ) 3 row(s) returned
- 2 13:00:01 SELECT c.iban,round(AVG(f.amount),2) FROM fact\_transactions f JOIN dim\_credit\_card ... 370 row(s) returned

Activar Windows  
Ve a Configuración para activar.

Duration / Fetch  
0.437 sec / 0.000 sec  
0.015 sec / 0.000 sec

Figura 16. media del importe (*amount*) por IBAN de las tarjetas de crédito que han realizado transacciones en la empresa **Donec Ltd**.

En la figura 16 se ejecuta una consulta que calcula la **media del importe (amount) por IBAN** de las tarjetas de crédito que han realizado transacciones en la empresa **Donec Ltd**. Para ello se unen las tablas **fact\_transactions**, **dim\_credit\_card** y **dim\_company** mediante sus claves correspondientes. El resultado muestra tres IBAN junto con el promedio de los importes asociados a cada uno.

### Nivel 2

Crea una nueva tabla que refleje el estado de las tarjetas de crédito basándose en si las tres últimas transacciones han sido rechazadas; en ese caso la tarjeta es **inactiva**. Si al menos una de ellas no ha sido rechazada, entonces es **activa**. A partir de esta tabla, responde:

## Ejercicio 1

¿Cuántas tarjetas están activas?

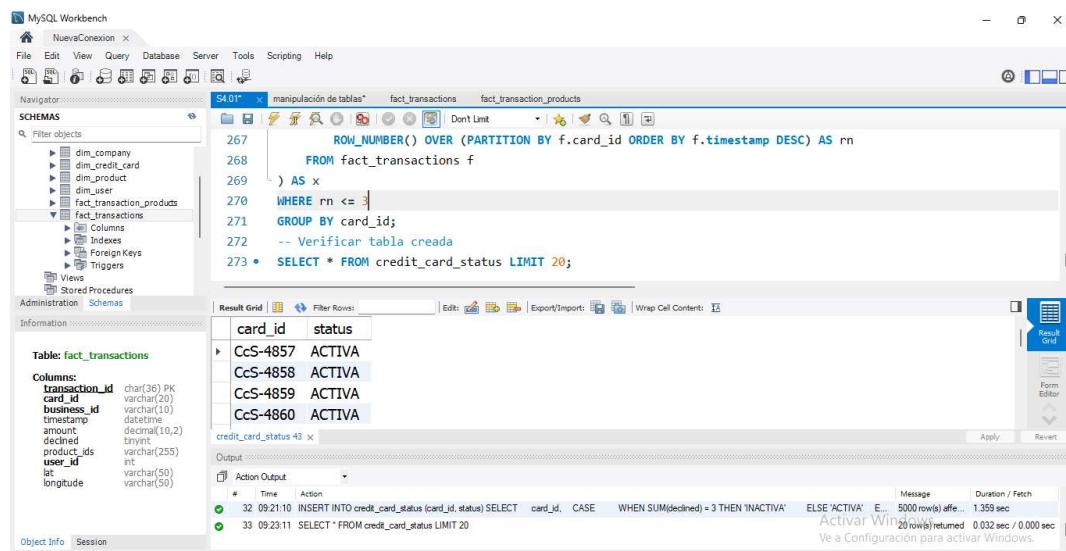
En este ejercicio se creó la tabla **credit\_card\_status** para clasificar las tarjetas de crédito como *ACTIVAS* o *INACTIVAS* según sus tres últimas transacciones, ver figura 17. Mediante una subconsulta que utiliza ROW\_NUMBER(), se seleccionaron las tres operaciones más recientes por tarjeta. Luego, con una expresión CASE, se determinó su estado: una tarjeta se considera **INACTIVA** si las tres transacciones fueron rechazadas; en caso contrario, se clasifica como **ACTIVA**, ver figura 18. Finalmente, se verificó la correcta inserción de los datos consultando los primeros registros de la tabla, ver figura 19.

The screenshot shows the MySQL Workbench interface with a query editor window titled "S4.01\*". The code in the editor creates the **credit\_card\_status** table and inserts data into it based on the last three transactions for each card. The table has columns **card\_id** and **status**. The status is determined by a CASE statement that checks if the sum of declined transactions is 3 (then **INACTIVA**) or 0 (then **ACTIVA**). The insert query uses a subquery to get the last three transactions for each card and a window function to count the number of declines. The output pane shows the execution of the create table command and the insert command, both of which completed successfully.

```
CREATE TABLE credit_card_status (
    card_id VARCHAR(20) PRIMARY KEY,
    status VARCHAR(10)
);

INSERT INTO credit_card_status (card_id, status)
SELECT
    card_id,
    CASE
        WHEN SUM(declined) = 3 THEN 'INACTIVA'
        ELSE 'ACTIVA'
    END AS status
FROM (
    SELECT
        f.card_id,
        f.declined,
        ROW_NUMBER() OVER (PARTITION BY f.card_id ORDER BY f.timestamp DESC) AS rn
    FROM fact_transactions f
) AS x
WHERE rn <= 3
GROUP BY card_id;
```

Figura 17. Creación tabla **credit\_card\_status**



```

MySQL Workbench
File Edit View Query Database Server Tools Scripting Help
Schemas
Table: fact_transactions
Columns:
transaction_id char(36) PK
card_id varchar(20)
business_id varchar(10)
timestamp datetime
amount decimal(10,2)
declined tinyint
product_ids varchar(255)
user_id int
lat varchar(50)
longitude varchar(50)

SELECT * FROM credit_card_status LIMIT 20;
  
```

The screenshot shows the MySQL Workbench interface with a query editor window titled "manipulación de tablas". The query is as follows:

```

ROW_NUMBER() OVER (PARTITION BY f.card_id ORDER BY f.timestamp DESC) AS rn
FROM fact_transactions f
) AS x
WHERE rn <= 3
GROUP BY card_id;
-- Verificar tabla creada
SELECT * FROM credit_card_status LIMIT 20;
  
```

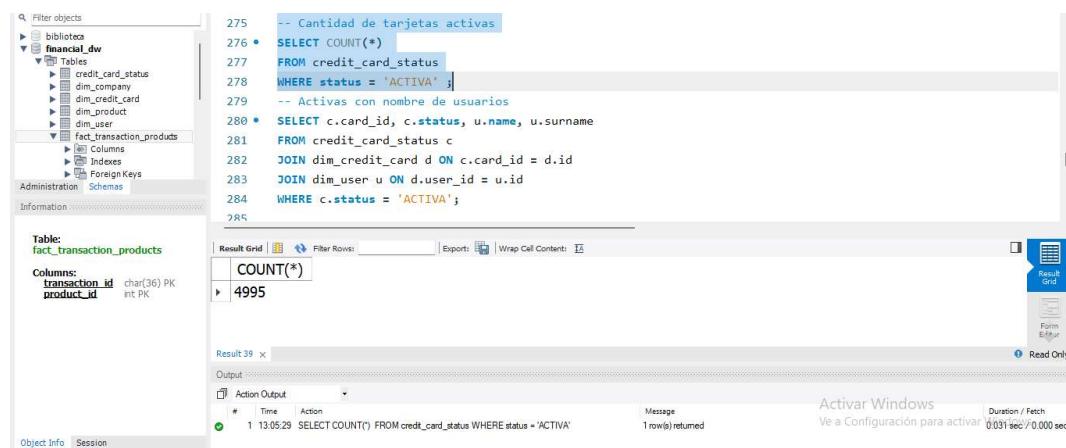
The results grid displays four rows of data:

card_id	status
Ccs-4857	ACTIVA
Ccs-4858	ACTIVA
Ccs-4859	ACTIVA
Ccs-4860	ACTIVA

The status column shows all entries as "ACTIVA". Below the results, the "Output" pane shows two log entries:

- 32 09:21:10 INSERT INTO credit\_card\_status (card\_id, status) SELECT card\_id, CASE WHEN SUM(declined) = 3 THEN 'INACTIVA' ELSE 'ACTIVA' END FROM fact\_transactions GROUP BY card\_id; 5000 row(s) affected. 1.359 sec
- 33 09:23:11 SELECT \* FROM credit\_card\_status LIMIT 20 rows returned. 0.032 sec / 0.000 sec

Figura 18. Determinación del estado de las tarjetas-



```

MySQL Workbench
Table: fact_transaction_products
Columns:
transaction_id char(36) PK
product_id int PK

SELECT COUNT(*)
FROM credit_card_status
WHERE status = 'ACTIVA'
  
```

The screenshot shows the MySQL Workbench interface with a query editor window titled "manipulación de tablas". The query is as follows:

```

-- Cantidad de tarjetas activas
SELECT COUNT(*)
FROM credit_card_status
WHERE status = 'ACTIVA'
  
```

The results grid displays one row of data:

COUNT(*)
4995

The output pane shows a single log entry:

- 1 13:05:29 SELECT COUNT(\*) FROM credit\_card\_status WHERE status = 'ACTIVA' 1 row(s) returned. 0.031 sec / 0.000 sec

Figura 19. Consulta cantidad de tarjetas inactivas

En la figura 20 podemos observar las tarjetas inactivas con el nombre de usuarios

The screenshot shows the MySQL Workbench interface. On the left, the 'Schemas' tree view is open, showing the 'financial\_dw' schema with several tables like 'credit\_card\_status', 'dim\_company', etc. In the center, a query editor window displays the following SQL code:

```

275 -- Cantidad de tarjetas activas
276 • SELECT COUNT(*)
277 FROM credit_card_status
278 WHERE status = 'ACTIVA';
279 -- Activas con nombre de usuarios
280 • SELECT c.card_id, c.status, u.name, u.surname
281 FROM credit_card_status c
282 JOIN dim_credit_card d ON c.card_id = d.id
283 JOIN dim_user u ON d.user_id = u.id
284 WHERE c.status = 'ACTIVA';
285

```

Below the query editor is a result grid titled 'fact\_transaction\_products' with columns: card\_id, status, name, surname. The data shows three rows:

card_id	status	name	surname
CcS-4857	ACTIVA	Erxbtmt	Cnzfc
CcS-4858	ACTIVA	Vsveckm	Ghyee
CcS-4859	ACTIVA	Jndiens	Rcfior

At the bottom, the 'Output' pane shows two log entries:

- Action Output: # 1 13:05:29 Action Message 1 row(s) returned
- # 2 13:06:26 Action Message 4995 row(s) returned

On the right side of the interface, there are various toolbars and a status bar indicating 'Read Only' mode.

Figura 20. Tarjetas inactivas con usuarios respectivos.

### Nivel 3

Crea una tabla con la que podamos unir los datos del nuevo archivo *products.csv* con la base de datos creada, teniendo en cuenta que desde *transaction* dispones de *product\_ids*. Genera la siguiente consulta:

#### Ejercicio 1

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

En el modelo no es necesario crear una nueva tabla puente porque ya existe **fact\_transaction\_products**, que cumple exactamente la función de relacionar cada transacción con los productos incluidos en ella, evitando duplicidad de datos y manteniendo la integridad del diseño. La imagen mostrada evidencia que esta tabla funciona correctamente: en MySQL Workbench se observa la consulta SQL que une **fact\_transaction\_products** con **dim\_product** para contar cuántas veces se vendió cada producto, y los resultados muestran productos como *riverlands the duel* con 2654 ventas, ver figura 21, lo que confirma que la estructura actual permite obtener correctamente el número de ventas por producto sin necesidad de crear una nueva tabla.

The screenshot shows the MySQL Workbench interface. On the left, the 'Object Navigator' displays the schema structure under 'biblioteca' and 'financial\_dw'. The 'Tables' section lists several tables including 'credit\_card\_status', 'dim\_company', 'dim\_credit\_card', 'dim\_product', 'fact\_transaction\_products', and 'fact\_transactions'. Below this, the 'Table: fact\_transaction\_products' is selected, showing its columns: 'transaction\_id' (char(36) PK) and 'product\_id' (int PK). The main workspace shows a query editor with the following SQL code:

```

292 •  SELECT
293   p.product_id,
294   p.product_name,
295   COUNT(*) AS veces_vendido
296   FROM fact_transaction_products ftp
297   JOIN dim_product p
298   ON ftp.product_id = p.product_id
299   JOIN fact_transactions f
300   ON ftp.transaction_id = f.transaction_id
301   WHERE f.declined = 0
302   GROUP BY p.product_id, p.product_name
303   ORDER BY veces_vendido DESC;
304
305  /*
  */

```

The 'Result Grid' shows the output of the query:

product_id	product_name	veces_vendido
52	riverlands the duel	2642
29	Tully maester Tarly	2627
21	duel Direwolf	2603

At the bottom, the 'Session' tab is active, showing the execution details: # 1 13:07:55 Action SELECT p.product\_id, p.product\_name, COUNT(\*) AS veces\_vendido FROM fa... 100 row(s) returned. The status bar indicates 'Activar Windows' and 'Duration / Fetch 2.469 sec / 0.000 sec'.

Figura 21. Número de veces que se ha vendido cada producto.

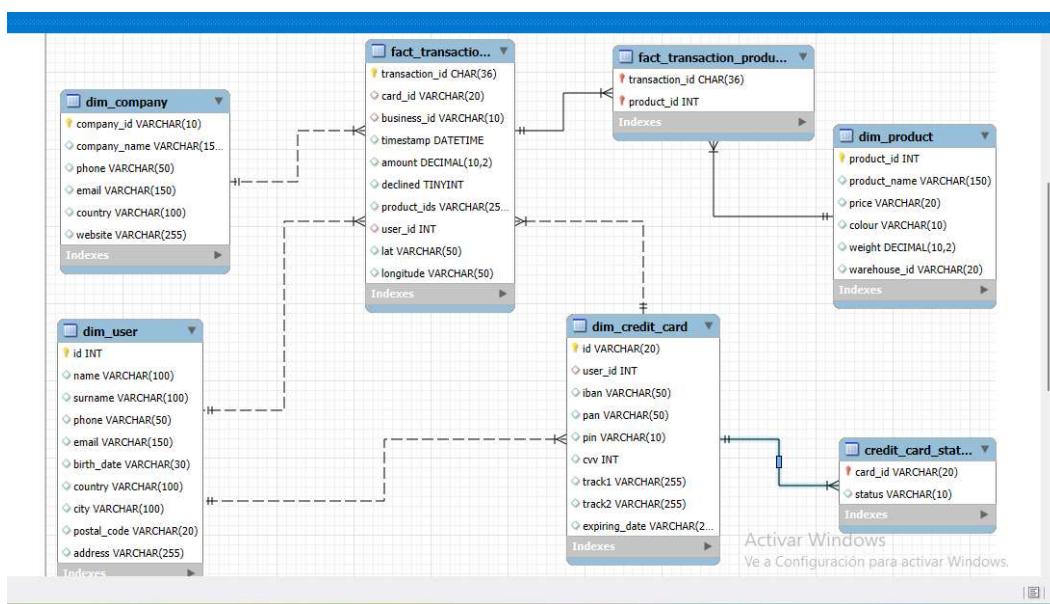


Figura 22. Diagrama EER final de BD financial\_dw.

En la figura 14 y 22, se puede observar que en MySQL Workbench las líneas sólidas representan *identifying relationships*, donde la clave foránea forma parte de la clave primaria, como en la tabla puente *fact\_transaction\_products*, mientras que las líneas punteadas indican *non-identifying relationships*, en las que la clave foránea no participa en la clave primaria, tal como ocurre en las relaciones entre *fact\_transactions* y las tablas de dimensiones. Para comprobar por qué el diagrama mostraba líneas punteadas, se

verificó el motor InnoDB, se revisaron y limpiaron las claves foráneas duplicadas y se reinsertaron correctamente; tras ello se confirmó que las líneas punteadas no eran un error, sino el comportamiento esperado, ya que las relaciones no son identificadoras. Esto coincide con la documentación oficial de MySQL Workbench: “*non-identifying relationships are displayed using dashed lines, as the foreign key does not participate in the primary key of the child table*” (MySQL Workbench Manual, EER Diagrams).

Tabla 2. Tabla 2. Interpretación de las líneas de relación en Workbench

Línea	¿Cuándo aparece?	¿Por qué?
Sólida	Cuando un atributo es PK + FK	La identidad depende del padre
Punteada	Cuando la FK <b>no es parte de la PK</b>	La tabla hija tiene identidad propia

**REFERENCIAS**

- Codd, E. F. (1970). *A relational model of data for large shared data banks*. **Communications of the ACM**, 13(6), 377–387.  
<https://doi.org/10.1145/362384.362685>
- Kimball, R., & Ross, M. (2013). *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling* (3rd ed.). John Wiley & Sons.
- Oracle Corporation. (2024). *MySQL 8.0 Reference Manual: JSON\_TABLE Function*.  
<https://dev.mysql.com/doc/refman/8.0/en/json-table-functions.html>
- Oracle Corporation. (2024). *Oracle Database SQL Language Reference: JSON\_TABLE*.  
[https://docs.oracle.com/en/database/oracle/oracle-database/23/sqlrf/JSON\\_TABLE.html](https://docs.oracle.com/en/database/oracle/oracle-database/23/sqlrf/JSON_TABLE.html)
- OpenAI. (2025). *ChatGPT* (versión gpt-5.1) [Modelo de lenguaje de gran escala].  
<https://chat.openai.com/>