

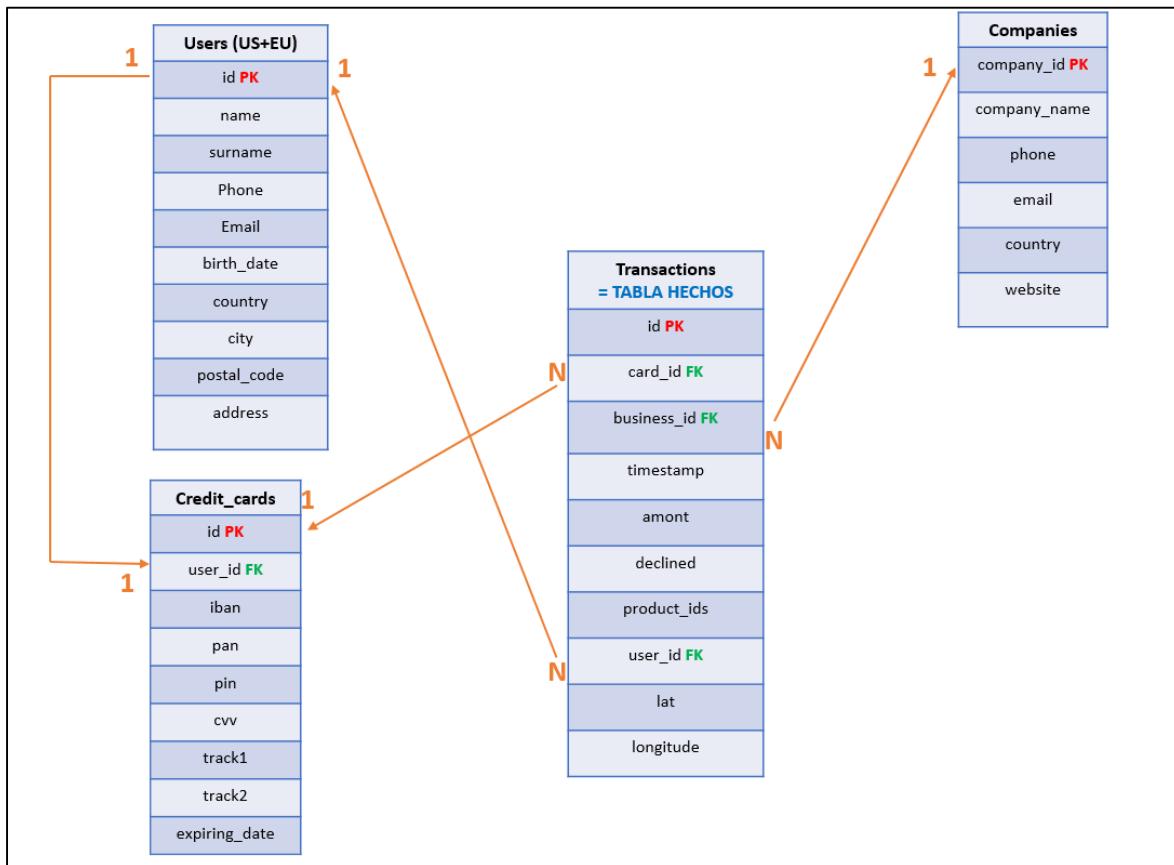
SPRINT 4 (MYSQL) MARINE FERNANDEZ

Partiendo de algunos archivos CSV diseñarás y crearás tu base de datos.

NIVEL 1

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

Diagrama



Esa base de datos se compone de 4 tablas : transactions, companies, credit_cards y users y tiene un **modelo en estrella**.

La tabla transactions es la **tabla de hechos**, es decir que almacena datos cuantitativos acerca de las ventas realizadas y cada fila representa una transacción (venta).

Las tablas companies, credit_cards y users son las tablas de dimensiones del modelo es decir que contienen detalles acerca de los hechos de la tabla transactions. Se relacionan con la tabla de hechos a través de sus Primary Key las cuales son Foreign Key en la tabla de hechos.

Detalle de cada tabla:

Users (US+EU)

- Id: identificador único del usuario.
- Name: nombre del usuario.
- Surname: apellido del usuario.
- Phone: número de teléfono del usuario.
- Email: dirección de correo electrónico del usuario.
- Birth_date: fecha de nacimiento del usuario.
- Country: país de residencia del usuario.
- City: ciudad donde reside el usuario.
- Postal_code: código postal del usuario.
- Address: dirección física del usuario.

Credit_cards

- Id: identificador único de la tarjeta bancaria.
- User_id: identificador del usuario propietario de la tarjeta.
- Iban: número internacional de cuenta bancaria asociado a la tarjeta.
- Pan: número completo de la tarjeta bancaria (impreso en la tarjeta).
- Pin: número de identificación personal utilizado para autorizar transacciones.
- Cvv: código de verificación usado para compras (asegura la posesión física de la tarjeta).
- Track1 / Track2: bandas magnéticas que contienen información codificada de la tarjeta.
- Expiring_date: fecha de vencimiento de la tarjeta bancaria.

Companies

- Company_id: identificador único de la empresa.
- Company_name: nombre de la empresa.
- Phone: número telefónico de la empresa.
- Email: correo electrónico de contacto de la empresa.
- Country: país donde opera la empresa.
- Website: sitio web oficial de la empresa.

Transactions (TABLA DE HECHOS)

- Id: identificador único de la transacción.
- Card_id: identificador de la tarjeta usada en la transacción (llave foránea hacia Credit_cards).
- Business_id: identificador de la empresa donde se realizó la compra.
- User_id: identificador del usuario que efectuó la transacción.
- Timestamp: fecha y hora exacta en que se realizó la transacción.
- Amount: importe total de la venta o pago efectuado.
- Declined: indica si la transacción fue rechazada (1) o aprobada (0).
- Product_ids: lista los identificadores de los productos incluidos en la compra.
- Lat: latitud del punto donde se realizó la transacción.
- Longitude: longitud del punto donde se realizó la transacción.

Relacion entre las tablas:

Relación entre users y credit_cards : una vez cargados los datos averiguaremos esa relación gracias a una query.

Relación entre users y transactions: Uno a Muchos (1:N)

Un usuario puede realizar muchas transacciones, pero cada transacción pertenece a un solo usuario.

Esta relación se configura con la **Foreign Key** user_id en la tabla transactions, la cual hace referencia al campo id de la tabla users donde es **Primary Key**.

Relación entre credit_cards y transactions: Uno a Muchos (1:N)

Una tarjeta de crédito puede ser usada en múltiples transacciones, pero cada transacción se realiza con una sola tarjeta.

Esta relación está dada por el campo card_id en transactions, que es una **Foreign Key** referenciando el id de credit_cards, donde es **Primary Key** .

Relación entre companies y transactions: Uno a Muchos (1:N)

Una empresa (comercio) puede registrar muchas transacciones, pero cada transacción corresponde a una sola empresa.

Esta relación se define con la **Foreign Key** business_id en la tabla transactions, que apunta al company_id de companies donde es **Primay Key**.

Creación de la Base de Datos, de las tablas y proceso de carga de los datos:

Primero hemos procedido a crear la base de datos en la cual crearemos las distintas tablas del modelo la hemos llamado sales.

Posteriormente antes de poder cargar los datos de los archivos CSV hubo que crear las estructuras de las tablas, configurando los data type de cada columna cuando era posible.

```
1  #NIVEL 1
2  #Descarga los archivos CSV, estudiales y diseña una base de datos con un esquema de estrella que contenga, al menos 4 tablas de las que puedas realizar las siguientes
3
4 • CREATE DATABASE sales;
5
6 • CREATE TABLE companies (
7     company_id VARCHAR(15),
8     company_name VARCHAR(100),
9     phone VARCHAR(20),
10    email VARCHAR(100),
11    country VARCHAR(100),
12    website VARCHAR(100)
13 );
14
```

Output

Action Output	#	Time	Action	Message	Duration / Fetch
	1	11:49:32	CREATE DATABASE sales	1 row(s) affected	0.000 sec
	2	11:50:01	CREATE TABLE companies (company_id VARCHAR(15), company_name VARCHAR(100), phone V...)	0 row(s) affected	0.141 sec

Para la **tabla companies**, hemos escogido el data type VARCHAR ya que hemos notado que los datos cuentan con letras y números con distintas longitudes. En el caso de la columna phone hemos escogido VARCHAR también por si habría ahora o en el futuro un numero con un carácter especial (+ por ejemplo) .

The screenshot shows the MySQL Workbench interface. In the top pane, there is a code editor with the following SQL code:

```
17 • CREATE TABLE transactions (
18     id VARCHAR(100),
19     card_id VARCHAR(255),
20     business_id VARCHAR(255),
21     timestamp TIMESTAMP,
22     amount DECIMAL (10,2),
23     declined TINYINT(1),
24     product_ids VARCHAR(50),
25     user_id INT,
26     lat FLOAT,
27     longitude FLOAT
28 ) .
```

In the bottom pane, there is an "Output" window titled "Action Output". It contains a table with one row of data:

#	Time	Action	Message	Duration / Fetch
1	11:58:09	CREATE TABLE transactions (id VARCHAR(100), card_id VARCHAR(255), business_id VARCHAR(255), timestamp TIMESTAMP, amount DECIMAL (10,2), declined TINYINT(1), product_ids VARCHAR(50), user_id INT, lat FLOAT, longitude FLOAT) .	0 rows affected, 1 warning(s). 1681 Integer display width is deprecated and will be removed in a future release.	0.063 sec

Para la **tabla transactions** hemos escogido el data type VARCHAR para las columnas id, credit_card, business_id y product_ids ya que los datos de esos campos cuentan con letras y números con distintas longitudes.

Para la columna timestamps hemos elegido TIMESTAMP ya que se trata de una fecha con hora.

Para user_id hemos elegido INT ya que se tratan de números enteros.

Hemos configurado la columna amout como DECIMAL con hasta dos números después de la coma ya que hemos revisado los datos en el archivo CSV previamente para determinarlo.

Para la columna declined, al tratarse de un solo número hemos escogido el tipo TINYINT ya que se adapta mejor que un INT en este caso.

Finalmente para las tablas lat y longitude, al tratarse de datos con muchas decimales hemos escogido FLOAT sin límite de decimales.

```

30 • CREATE TABLE users (
31     id INT,
32     name VARCHAR(100),
33     surname VARCHAR(100),
34     phone VARCHAR(15),
35     email VARCHAR(100),
36     birth_date VARCHAR(255),
37     country VARCHAR(100),
38     city VARCHAR(100),
39     postal_code VARCHAR(100),
40     address VARCHAR(100)
41 );
42

```

Output

#	Time	Action	Message	Duration / Fetch
1	11:52:58	CREATE TABLE users (id INT, name VARCHAR(100), surname VARCHAR(100), phone VARCHAR(15), email VARCHAR(100), birth_date VARCHAR(255), country VARCHAR(100), city VARCHAR(100), postal_code VARCHAR(100), address VARCHAR(100));	0 row(s) affected	0.047 sec

Para la carga posterior de los datos de los archivos american users y european users hemos creado una sola tabla nombrada **users**.

Para el id hemos escogido el data type INT ya que se tratan de números enteros.

Para la columna birth_date hemos tenido que configurarla como VARCHAR porque si la configuramos como DATE, no se cargan los datos ya que tienen el siguiente formato : "Nov 17, 1985" y MYSQL no lo reconoce como fecha.

Para el resto de columnas hemos escogido VARCHAR ya que los datos de esos campos cuentan con letras y números con distintas longitudes.

```

43 • CREATE TABLE credit_cards (
44     id VARCHAR(30),
45     user_id INT,
46     iban VARCHAR(50),
47     pan VARCHAR(50),
48     pin VARCHAR(6),
49     cvv INT,
50     track1 VARCHAR(255),
51     track2 VARCHAR(255),
52     expiring_date VARCHAR(255)
53 );
54

```

Output

#	Time	Action	Message	Duration / Fetch
1	11:54:15	CREATE TABLE credit_cards (id VARCHAR(30), user_id INT, iban VARCHAR(50), pan VARCHAR(50), pin VARCHAR(6), cvv INT, track1 VARCHAR(255), track2 VARCHAR(255), expiring_date VARCHAR(255));	0 row(s) affected	0.063 sec

Para la tabla credit_cards, hemos configurado las columnas, iban, pan, pin, track1 y track 2 como VARCHAR ya que los datos de esos campos cuentan con letras y números y hemos adaptado la longitud en función de los datos del CSV previamente revisado.

Las columnas user_id y cvv como INT al tratarse de números enteros.

Carga de los datos:

```
55 #Para determinar dónde almacenar y desde dónde cargar los archivos CSV en MySQL, fue necesario consultar las rutas de archivos accesibles por el servidor.
56 #Esto se hizo utilizando el siguiente comando:
57
58 • SHOW VARIABLES LIKE 'secure_file_priv';
59
```

Result Grid | Filter Rows: [] | Export: [] | Wrap Cell Content: [] | Result Grid | Read Only

Variable_name	Value
secure_file_priv	C:\ProgramData\MySQL\MySQL Server 8.0\Uploads

Result 2 x Output

Action Output

#	Time	Action
1	11:55:02	SHOW VARIABLES LIKE 'secure_file_priv'

Message: 1 row(s) returned Duration / Fetch: 0.000 sec / 0.000 sec

Para determinar dónde almacenar y desde dónde cargar los archivos CSV en MySQL, fue necesario consultar las rutas de archivos accesibles por el servidor.

Posteriormente empezamos a cargar el primer archivo transactions.csv.

Tuvimos que hacer algunas correcciones como cambiar el sentido de las barras invertidas (subrayadas) del nombre de la ruta y ponerlas dobles : C:\ProgramData\MySQL\MySQL Server 8.0\Uploads.

```
60 • USE sales;
61 • LOAD DATA INFILE 'C://ProgramData//MySQL//MySQL Server 8.0//Uploads//transactions.csv'
62     INTO TABLE transactions
63     FIELDS TERMINATED BY ';'
64     OPTIONALLY ENCLOSED BY ""
65     LINES TERMINATED BY '\n'
66     IGNORE 1 ROWS;
67
68     #verificación
69 •     SELECT*
70     FROM transactions;
```

Output

Action Output

#	Time	Action
1	12:00:45	LOAD DATA INFILE 'C://ProgramData//MySQL//MySQL Server 8.0//Uploads//transactions.csv' INTO T... 100000 row(s) affected Records: 100000 Deleted: 0 Skipped: 0 Warnings: 0

Message: 100000 row(s) affected Records: 100000 Deleted: 0 Skipped: 0 Warnings: 0 Duration / Fetch: 2.813 sec

```
68     #verificación
69 •     SELECT*
70     FROM transactions;
```

Result Grid | Filter Rows: [] | Export: [] | Wrap Cell Content: [] | Fetch rows: [] | Result Grid | Read Only

id	card_id	business_id	timestamp	amount	declined	product_ids	user_id	lat	longitude
CDDA7E40-544D-47EB-A4ED-671D08A950D9	Ccs-6894	b-2466	2018-12-12 08:05:17	161.88	0	75, 73, 98	2313	59.6205	16.56
09456357-8E9B-475A-8257-87A023699964	Ccs-5135	b-2342	2024-05-20 22:49:08	171.13	0	3	554	45.7646	4.84306
C47C7C84-C174-4973-A768-825A9D85582A	Ccs-8415	b-2250	2018-11-04 23:16:18	497.29	0	92, 85, 36, 23	3834	52.0685	4.3011
2FB526AA-3844-4DDF-AC09-2EBC255EE76	Ccs-6553	b-2610	2022-06-17 09:17:25	344.15	0	5, 27	1972	39.476	-0.376419
00877407-B85E-40D2-9229-6BD6FB6DF0A	Ccs-7678	b-2454	2020-11-29 07:29:16	435.10	0	38, 32, 82, 62	3097	51.4366	5.47853

Result 4 x Output

Action Output

#	Time	Action
1	12:04:53	SELECT* FROM transactions

Message: 100000 row(s) returned Duration / Fetch: 0.000 sec / 0.297 sec

Cargamos los datos tomando en cuenta que en este archivo CSV (si lo abrimos en notebook) los campos están separados por ";", los valores pueden estar entrecomillados y las filas terminan por un salto a la línea. Además, se salta la primera línea ya que es la que contiene los encabezados.

Y procedemos de igual manera con el resto de archivos, con la diferencia de que los campos están separados por una coma.

```

72 • LOAD DATA INFILE 'C://ProgramData//MySQL//MySQL Server 8.0//Uploads//companies.csv'
73   INTO TABLE companies
74   FIELDS TERMINATED BY ','
75   OPTIONALLY ENCLOSED BY ""
76   LINES TERMINATED BY '\n'
77   IGNORE 1 ROWS;
78
79   #verificación
80 • SELECT*
81   FROM companies;
82

```

Output:

#	Time	Action	Message	Duration / Fetch
1	12:01:33	LOAD DATA INFILE 'C://ProgramData//MySQL//MySQL Server 8.0//Uploads//companies.csv' INTO TA...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0	0.062 sec

```

79   #verificación
80 • SELECT*
81   FROM companies;
82

```

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

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@cloud.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.co.uk	Germany	https://crn.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

Output:

#	Time	Action	Message	Duration / Fetch
1	12:05:43	SELECT* FROM companies	100 row(s) returned	0.000 sec / 0.000 sec

```

83 • LOAD DATA INFILE 'C://ProgramData//MySQL//MySQL Server 8.0//Uploads//credit_cards.csv'
84   INTO TABLE credit_cards
85   FIELDS TERMINATED BY ','
86   OPTIONALLY ENCLOSED BY ""
87   LINES TERMINATED BY '\n'
88   IGNORE 1 ROWS;
89
90   #verificación
91 • SELECT*
92   FROM credit_cards;
93

```

Output:

#	Time	Action	Message	Duration / Fetch
1	12:04:00	LOAD DATA INFILE 'C://ProgramData//MySQL//MySQL Server 8.0//Uploads//credit_cards.csv' INTO TA...	5000 row(s) affected Records: 5000 Deleted: 0 Skipped: 0 Warnings: 0	0.203 sec

```

90   #verificación
91 • SELECT*
92   FROM credit_cards;
93

```

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

id	user_id	iban	pan	pin	cvv	track1	track2	expiring_date
CcU-2938	275	TR301950312213576817638661	5424465566813633	3257	984	%#68383712448554646~WovsxejDpwiev^8604...	%#87653863056044187=800716333673	10/30/22
CcU-2945	274	DQ26854763748537475216568689	5142423821948828	9080	887	%#8462131609958661^UfuyfsSeimnx^06106...	%#84149568437843501=510714033071	08/24/23
CcU-2952	273	BG49IVQL52710525608255	4556 453 55 5287	4598	438	%#82183285104307501^CddyytcUxwfdq^9907...	%#86778580257827162=690685974007	06/29/21
CcU-2959	272	CR724277244335841535	372461377349375	3583	667	%#87281111956795320^Xocddjbkcedc^09016...	%#8424615489281853=280522391678	02/24/23
CcU-2966	271	BG72LKTQ15627628377363	448566 886747 7265	4900	130	%#84728932322756223^Jhgvufbmwg^7202...	%#82318571115599881=890821578475	10/29/24

Output:

#	Time	Action	Message	Duration / Fetch
1	12:06:28	SELECT* FROM credit_cards	5000 row(s) returned	0.000 sec / 0.016 sec

```

94 • LOAD DATA INFILE 'C://ProgramData//MySQL//MySQL Server 8.0//Uploads//european_users.csv'
95   INTO TABLE users
96   FIELDS TERMINATED BY ','
97   OPTIONALLY ENCLOSED BY ""
98   LINES TERMINATED BY '\n'
99   IGNORE 1 ROWS;
100
101
102 • LOAD DATA INFILE 'C://ProgramData//MySQL//MySQL Server 8.0//Uploads//american_users.csv'
103   INTO TABLE users
104   FIELDS TERMINATED BY ','
105   OPTIONALLY ENCLOSED BY ""
106   LINES TERMINATED BY '\n'
107   IGNORE 1 ROWS;
108
< Output
  Action Output
    # Time Action
    1 12:07:17 LOAD DATA INFILE C://ProgramData//MySQL//MySQL Server 8.0//Uploads//european_users.csv' INT... 3990 row(s) affected Records: 3990 Deleted: 0 Skipped: 0 Warnings: 0 0.250 sec
    2 12:07:21 LOAD DATA INFILE C://ProgramData//MySQL//MySQL Server 8.0//Uploads//american_users.csv' INT... 1010 row(s) affected Records: 1010 Deleted: 0 Skipped: 0 Warnings: 0 0.047 sec

```

```

109  #verificación
110 • SELECT*
111   FROM users;
< Output
  Result Grid | Filter Rows: [ ] Export: [ ] Wrap Cell Content: [ ] Fetch rows: [ ]
    id name surname phone email birth_date country city postal_code address
    151 Meghan Hayden 0800 746 6747 arcu.vel@hotmail.ca Jul 2, 1980 United Kingdom London EC1A 1BB Ap #432-4493 Aliquet Rd.
    152 Hakeem Alford (0111) 367 0184 adipiscing.ligula@google.edu Sep 30, 1979 United Kingdom Birmingham B1 1AA 551-8930 Lobortis Street
    153 Keegan Pugh (016977) 3851 sodales.nisi@aol.org Jul 27, 1994 United Kingdom London EC1A 1BB Ap #312-5898 Consectetuer St.
    154 Cooper Bullock (021) 2521 6627 et@outlook.net Nov 2, 1986 United Kingdom Manchester M1 1AE 872-1866 Pepe Rd.
    155 Joshua Russell 055 4409 5286 justo.nec.ante@outlook.edu Jan 23, 1984 United Kingdom Manchester M1 1AE Ap #285-4727 Auctor. Av.
    156 Remedios Case 055 3114 1566 mollis.phasellus.libero@aol.com Oct 9, 1994 United Kingdom Liverpool L1 1AA 479-3690 Turpis Road
    157 Philo Carev 0800 640 6251 phasellus@yahoo.net Oct 10, 1992 United Kingdom Manchester M1 1AE 196-1103 Ouisoue Street
  users 7 < Read Only
  Action Output
    # Time Action
    1 12:08:16 SELECT* FROM users
      Message
      5000 row(s) returned
      Duration / Fetch
      0.000 sec / 0.016 sec

```

Una vez todos los archivos CSV cargados, procedemos a configurar las Primary Key y modificar los data type de los campos que no habíamos podido configurar antes de cargarlos:

Tabla transactions

```

113  #Procedemos a realizar los cambios en cada tabla, tipo de dato en algunos caso y configuracion de los PK
114
115  #- CAMBIOS TABLA TRANSACTIONS
116  #configuracion de PK
117 • ALTER TABLE transactions
118   ADD CONSTRAINT Pk_transactions PRIMARY KEY (id);
119
< Output
  Action Output
    # Time Action
    1 12:09:11 ALTER TABLE transactions ADD CONSTRAINT Pk_transactions PRIMARY KEY (id)
      Message
      0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
      Duration / Fetch
      2.156 sec

```

```

119
120 • DESCRIBE transactions;
121
< Output
  Result Grid | Filter Rows: [ ] Export: [ ] Wrap Cell Content: [ ]
    Field Type Null Key Default Extra
    id varchar(100) NO PRI NULL
    card_id varchar(255) YES NULL
    business_id varchar(255) YES NULL
    timestamp timestamp YES NULL
    amount decimal(10,2) YES NULL
    declined tinyint(1) YES NULL
    product_ids varchar(50) YES NULL
    user_id int YES NULL
  Result 8 < Read Only
  Action Output
    # Time Action
    1 12:10:22 DESCRIBE transactions
      Message
      10 row(s) returned
      Duration / Fetch
      0.016 sec / 0.000 sec

```

Tabla users

```

122  #CAMBIOS TABLA USERS
123 • ALTER TABLE users
124 ADD CONSTRAINT Pk_users PRIMARY KEY (id);
125
126 #Cambiamos el data type de la columna birth_date
127 • SET SQL_SAFE_UPDATES = 0; #Se tiene que configurar el safe update mode para poder usar update sin una clausula WHERE
128 • UPDATE users
129 SET birth_date = STR_TO_DATE(birth_date, "%b %d, %Y");
130
< -----
Output
  Action Output
    # Time Action
    1 12:27:33 ALTER TABLE users ADD CONSTRAINT Pk_users PRIMARY KEY (id) Message
                                                0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 Duration / Fetch
    2 12:27:37 SET SQL_SAFE_UPDATES = 0 0 row(s) affected 0.000 sec
    3 12:27:39 UPDATE users SET birth_date = STR_TO_DATE(birth_date, "%b %d, %Y") 5000 row(s) affected Rows matched: 5000 Changed: 5000 Warnings: 0 0.266 sec
  
```



```

131 • ALTER TABLE users
132 MODIFY COLUMN birth_date DATE;
133
134 #verificacion
135 • DESCRIBE users;
136
< -----
Result Grid | Filter Rows: [ ] Export: [ ] Wrap Cell Content: [ ]
  Field Type Null Key Default Extra
  phone varchar(15) YES NULL
  email varchar(100) YES NULL
  birth_date date YES NULL
  country varchar(100) YES NULL
  city varchar(100) YES NULL
  postal_code varchar(100) YES NULL
  address varchar(100) YES NULL
  Result 15 x
  
```



```

131 #verificacion
132 • SELECT*
133 FROM users;
< -----
Result Grid | Filter Rows: [ ] Edit: [ ] Export/Import: [ ] Wrap Cell Content: [ ] Fetch rows: [ ]
  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 New York 10001 348-7818 Sagittis St.
  2 Garrett McConnell (718) 257-2412 integer.vitae.nibh@protonmail.org 1992-08-23 United States Philadelphia 19101 903 Sit Ave.
  3 Ciaran Harrison (522) 598-1365 interdum.feugiat@aol.org 1998-04-20 United States Houston 77001 736-2063 Tellus St.
  4 Howard Stafford 1-411-740-3269 ornare.egestas@cloud.edu 1989-02-18 United States Phoenix 85001 Ap #545-224 Erat. Rd.
  5 Hayfa Pierce 1-554-541-2077 et.malesuada.fames@hotmail.org 1998-09-26 United States Philadelphia 19101 341-2821 Ultrices Av.
  6 Joel Tyson (718) 288-8020 gravida.nunc.sed@yahoo.ca 1989-10-15 United States San Jose 95101 888-2799 Amet Street
  7 Rafael Jimenez (817) 689-0478 eget@outlook.ca 1981-12-04 United States Chicago 60601 8627 Malesuada Rd.
  8 Nissim Frank (692) 157-3469 egestas.aliquam.fringilla@google.ca 1993-08-01 United States New York 10001 Ap #251-7144 Integer St.
  users 9 x
  
```



```

131 #verificacion
132 • SELECT*
133 FROM users;
< -----
Action Output
  # Time Action
  1 12:29:26 ALTER TABLE users MODIFY COLUMN birth_date DATE Message
                                                5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0 Duration / Fetch
  2 12:29:28 DESCRIBE users 10 row(s) returned 0.219 sec
  0.016 sec / 0
  
```



```

131 #verificacion
132 • SELECT*
133 FROM users;
< -----
Result Grid | Filter Rows: [ ] Form Editor
  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 New York 10001 348-7818 Sagittis St.
  2 Garrett McConnell (718) 257-2412 integer.vitae.nibh@protonmail.org 1992-08-23 United States Philadelphia 19101 903 Sit Ave.
  3 Ciaran Harrison (522) 598-1365 interdum.feugiat@aol.org 1998-04-20 United States Houston 77001 736-2063 Tellus St.
  4 Howard Stafford 1-411-740-3269 ornare.egestas@cloud.edu 1989-02-18 United States Phoenix 85001 Ap #545-224 Erat. Rd.
  5 Hayfa Pierce 1-554-541-2077 et.malesuada.fames@hotmail.org 1998-09-26 United States Philadelphia 19101 341-2821 Ultrices Av.
  6 Joel Tyson (718) 288-8020 gravida.nunc.sed@yahoo.ca 1989-10-15 United States San Jose 95101 888-2799 Amet Street
  7 Rafael Jimenez (817) 689-0478 eget@outlook.ca 1981-12-04 United States Chicago 60601 8627 Malesuada Rd.
  8 Nissim Frank (692) 157-3469 egestas.aliquam.fringilla@google.ca 1993-08-01 United States New York 10001 Ap #251-7144 Integer St.
  users 9 x
  
```



```

131 #verificacion
132 • SELECT*
133 FROM users;
< -----
Action Output
  # Time Action
  1 12:13:31 SELECT* FROM users Message
                                                5000 row(s) returned Duration / Fetch
  0.000 sec / 0.031 sec
  
```

Para modificar el data type de VARCHAR a DATE de las columnas que contienen datos relacionados con fechas (birth_date en esa tabla y expiring_date en la tabla credit_cards) hemos usado la función STR_TO_DATE ya que nos permite personalizar el formato inicial de la fecha que queremos convertir de string a date, lo cual no sería posible usando DATE.

Tabla credit_cards

```

140  #CAMBIOS TABLA CREDIT_CARD
141
142 • ALTER TABLE credit_cards
143   ADD CONSTRAINT Pk_credit_cards PRIMARY KEY (id);
144
145   #Cambiamos el formato de la columna expiring_date
146 • UPDATE credit_cards
147   SET expiring_date = STR_TO_DATE (expiring_date, "%m/%d/%y");
148
149 • ALTER TABLE credit_cards
150   MODIFY COLUMN expiring_date DATE;
151
< ----- >
Output:
  Action Output
  # Time Action
  ✓ 1 12:22:51 ALTER TABLE credit_cards ADD CONSTRAINT Pk_credit_cards PRIMARY KEY (id) Message
  0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 Duration / Fetch
  0.297 sec
  ✓ 2 12:22:54 UPDATE credit_cards SET expiring_date = STR_TO_DATE (expiring_date, "%m/%d/%y") 5000 row(s) affected Rows matched: 5000 Changed: 5000 Warnings: 0 0.219 sec
  ✓ 3 12:23:03 ALTER TABLE credit_cards MODIFY COLUMN expiring_date DATE 5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0 0.219 sec

```

```

146  #verificaciones
147 • SELECT*
148   FROM credit_cards;
< ----- >
Result Grid | Filter Rows: [ ] | Edit: [ ] [ ] [ ] Export/Import: [ ] [ ] Wrap Cell Content: [ ] Fetch rows: [ ]
  id user_id iban pan pin cvv track1 track2 expiring_date
  ▶ CcS-4857 276 XX485759183529250580771 2314242385113924 1819 467 %B2314242385113924=24101015183631647 2025-09-27
  CcS-4858 277 XX8581768137002436094025 6582720299715533 3964 817 %B6582720299715533=2411011045462722 2028-12-28
  CcS-4859 278 XX7826930491423553609370 8861684536289642 4983 277 %B8861684536289642=C0FBGDCOFBGD^280... 2026-11-26
  CcS-4860 279 XX5559590368835304645299 2481155515498459 6876 661 %B2481155515498459^TJUTUTIUTU^31040... 2027-07-27
  CcS-4861 280 XX20351828771951627307 1308930301149557 5710 398 %B1308930301149557^HPOBNZHPOBNZ^330... 2026-04-25
  CcS-4862 281 XX4774721462463645409758 6715617009807829 4042 174 %B6715617009807829^LDMWTD.LDMWTD^33... 2026-11-27
  CcS-4863 282 XX1476823964245046207111 3140879819451394 5969 449 %B3140879819451394^OXXJODOXXJOD^230... 2029-12-27
  CcS-4864 283 XX8380298893385731196159 5793672133649114 4881 139 %B5793672133649114=23061018063671017 2026-02-28
credit_cards 13 x
Result 13 x
Output:
  Action Output
  # Time Action
  ✓ 1 12:19:02 SELECT* FROM credit_cards Message
  5000 row(s) returned Duration / Fetch
  0.000 sec / 0.016 sec

```

```

156 • DESCRIBE credit_cards;
< ----- >
Result Grid | Filter Rows: [ ] | Export: [ ] Wrap Cell Content: [ ]
  Field Type Null Key Default Extra
  ▶ id varchar(30) NO PRI NULL
  user_id int YES NULL
  iban varchar(50) YES NULL
  pan varchar(50) YES NULL
  pin varchar(6) YES NULL
  cvv int YES NULL
  track1 varchar(255) YES NULL
  track2 varchar(255) YES NULL
  expiring_date date YES NULL
Result 14 x
Output:
  Action Output
  # Time Action
  ✓ 2 12:22:54 UPDATE credit_cards SET expiring_date = STR_TO_DATE (expiring_date, "%m/%d/%y") Message
  5000 row(s) affected Rows matched: 5000 Changed: 5000 Warnings: 0 Duration / Fetch
  0.219 sec
  ✓ 3 12:23:03 ALTER TABLE credit_cards MODIFY COLUMN expiring_date DATE 5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0 0.219 sec
  ✓ 4 12:23:46 DESCRIBE credit_cards 9 row(s) returned 0.000 sec / 0.000 sec

```

Tabla companies

```

160  #CAMBIOS TABLA COMPANIES
161
162 • ALTER TABLE companies
163   ADD CONSTRAINT Pk_companies PRIMARY KEY (company_id);
164
165  #verificacion
166 • DESCRIBE companies;
167

```

Result Grid

Field	Type	Null	Key	Default	Extra
company_id	varchar(15)	NO	PRI	NULL	
company_name	varchar(100)	YES		NULL	
phone	varchar(20)	YES		NULL	
email	varchar(100)	YES		NULL	
country	varchar(100)	YES		NULL	
website	varchar(100)	YES		NULL	

Action Output

#	Time	Action	Message	Duration / Fetch
1	12:31:00	ALTER TABLE companies ADD CONSTRAINT Pk_companies PRIMARY KEY (company_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.188 sec
2	12:31:04	DESCRIBE companies	6 row(s) returned	0.000 sec / 0.000 sec

Finalmente, procedemos a relacionar las tablas entre sí:

```

168  #Procedemos a relacionar las tablas entre sí : configuración de foreign keys en la tabla transactions y relacionamos las tablas credit_cards y users
169 • ALTER TABLE transactions
170   ADD CONSTRAINT Fk_companies_transactions
171   FOREIGN KEY (business_id) REFERENCES companies(company_id);
172
173 • ALTER TABLE transactions
174   ADD CONSTRAINT Fk_users_transactions
175   FOREIGN KEY (user_id) REFERENCES users(id);
176
177 • ALTER TABLE transactions
178   ADD CONSTRAINT Fk_credit_cards_transactions
179   FOREIGN KEY (card_id) REFERENCES credit_cards(id);
180
181 • ALTER TABLE credit_cards
182   ADD CONSTRAINT Fk_users_credit_cards
183   FOREIGN KEY (user_id) REFERENCES users(id);
184

```

Action Output

#	Time	Action	Message	Duration / Fetch
1	12:32:03	ALTER TABLE transactions ADD CONSTRAINT Fk_companies_transactions FOREIGN KEY (business_id) REFERENCES companies(company_id);	100000 row(s) affected Records: 100000 Duplicates: 0 Warnings: 0	3.203 sec
2	12:32:09	ALTER TABLE transactions ADD CONSTRAINT Fk_users_transactions FOREIGN KEY (user_id) REFERENCES users(id);	100000 row(s) affected Records: 100000 Duplicates: 0 Warnings: 0	2.609 sec
3	12:32:12	ALTER TABLE transactions ADD CONSTRAINT Fk_credit_cards_transactions FOREIGN KEY (card_id) REFERENCES credit_cards(id);	100000 row(s) affected Records: 100000 Duplicates: 0 Warnings: 0	3.063 sec
4	12:32:16	ALTER TABLE credit_cards ADD CONSTRAINT Fk_users_credit_cards FOREIGN KEY (user_id) REFERENCES users(id);	5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0	0.281 sec

```

185  #verificación
186 • DESCRIBE transactions;
187 • DESCRIBE credit_cards;
188

```

Result Grid

Field	Type	Null	Key	Default	Extra
card_id	varchar(255)	YES	MUL	NULL	
business_id	varchar(255)	YES	MUL	NULL	
timestamp	timestamp	YES		NULL	
amount	decimal(10,2)	YES		NULL	
declined	tinyint(1)	YES		NULL	
product_ids	varchar(50)	YES		NULL	
user_id	int	YES	MUL	NULL	
lat	float	YES		NULL	
longitude	float	YES		NULL	

Action Output

#	Time	Action	Message	Duration / Fetch
1	12:34:01	DESCRIBE transactions	10 row(s) returned	0.016 sec / 0.000 sec

```

185      #verificación
186 •  DESCRIBE transactions;
187 •  DESCRIBE credit_cards;
188

```

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

Field	Type	Null	Key	Default	Extra
id	varchar(30)	NO	PRI	NULL	
user_id	int	YES	MUL	NULL	
iban	varchar(50)	YES		NULL	
pan	varchar(50)	YES		NULL	
pin	varchar(6)	YES		NULL	
cvv	int	YES		NULL	
track1	varchar(255)	YES		NULL	
track2	varchar(255)	YES		NULL	
expiring_date	date	YES		NULL	

Result 18 x

Output :::::::::::::::::::::

Action Output

#	Time	Action
1	12:34:53	DESCRIBE credit_cards

Message 9 row(s) returned Duration / F 0.000 sec / 0

```

181      #para asegurarnos de las relaciones y mas precisamente de la relacion entre credit_cards y users hacemos la siguiente query :
182 •  SELECT user_id , COUNT(*)
183     FROM credit_cards
184     GROUP BY user_id
185     HAVING COUNT(*) >1;
186
187      #asi sabemos que un usuario esta asociado a una sola tarjeta
188

```

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

user_id	COUNT(*)

Result 45 x

Output :::::::::::::::::::::

Action Output

#	Time	Action
45	13:19:31	SELECT user_id , COUNT(*) FROM credit_cards GROUP BY user_id HAVING COUNT(*) >1

Message 0 row(s) returned

Para saber como relacionar la tabla users y credit_cards (1 a 1, 1 a N o N a 1) buscamos si habían usuarios asociados a más de una tarjeta la query que vemos en el cuadro arriba, no fue el caso así que determinamos que la relación entre esas dos tablas es de 1 a 1.

EJERCICIO 1

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

```
192 •   SELECT *
193     FROM users AS u
194     WHERE EXISTS ( SELECT 1
195         FROM transactions AS t
196         WHERE t.user_id = u.id
197             AND t.declined = 0
198         GROUP BY t.user_id
199         HAVING COUNT(t.id) > 80
200     );
201
```

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

id	name	surname	phone	email	country	city	postal_code	address	birth_date
185	Molly	Gilliam	0800 120 8023	donec@outlook.co.uk	United Kingdom	London	EC1A 1BB	P.O. Box 202, 5638 Mi Rd.	1993-12-21
289	Dxwgi	Hwcru	+98-309-8797	dxwgi.hwcru@example.com	Germany	Stuttgart	70173	82 Hwcru Street	1976-08-20
318	Bnyr	Astuw	+33-120-9644	bnyr.astuw@example.com	Italy	Genoa	16100	53 Astuw Street	1974-05-03
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

users 6 x

Output ::::::::::::::::::::

Action Output

#	Time	Action	Message
5	10:22:40	SELECT * FROM users AS u WHERE EXISTS (SELECT 1 FROM transactions AS t	... 3 row(s) returned

EJERCICIO 2

Muestra la media de amount por IBAN de las tarjetas de crédito en la compañía Donec Ltd., utiliza por lo menos 2 tablas.

```
214      #EJERCICIO 2
215      #Muestra la media de amount por IBAN de las tarjetas de crédito en la compañía Donec Ltd., utiliza por lo menos 2 tablas.
216
217 •   SELECT c1.iban, ROUND(AVG(t.amount),2) AS Media_importe
218     FROM transactions AS t
219     JOIN companies AS c
220     ON t.business_id= c.company_id
221     JOIN credit_cards AS c1
222     ON c1.id= t.card_id
223     WHERE c.company_name = "Donec Ltd" AND t.declined = 0
224     GROUP BY c1.iban
225     ORDER BY Media_importe DESC ;
226
```

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

iban	Media_importe
Xx383017813919620199366352	680.69
Xx637706357397570394973913	680.01
Xx971393971465292202312259	645.46
Xx171847116928892375969307	628.89
Xx25424638818542406223575	608.68

Result 53 x

Output ::::::::::::::::::::

Action Output

#	Time	Action	Message
53	13:23:27	SELECT c1.iban, ROUND(AVG(t.amount),2) AS Media_importe FROM transactions AS t JOIN companie...	370 row(s) returned

NIVEL 2

Crea una nueva tabla que refleje el estado de las tarjetas de crédito basado en si las tres últimas transacciones han sido declinadas entonces es inactivo, si al menos una no es rechazada entonces es activo. Partiendo de esta tabla responde:

```
239 • CREATE TABLE IF NOT EXISTS Credit_cards_status AS(
240   WITH Transacciones_ordenadas AS (
241     SELECT
242       t.id,
243       t.card_id,
244       ROW_NUMBER() OVER (PARTITION BY t.card_id ORDER BY t.timestamp DESC) AS Ranking_transacciones
245     FROM transactions AS t),
246
247   Tres_ultimas_transacciones AS (
248     SELECT
249       t.card_id,
250       t.declined,
251       t.timestamp
252     FROM Transacciones_ordenadas
253     JOIN transactions AS t
254     ON t.id = Transacciones_ordenadas.id
255     WHERE Ranking_transacciones <=3),
256
257   Transacciones_declinadas AS (
258     SELECT
259       Tres_ultimas_transacciones.card_id,
260       SUM(Tres_ultimas_transacciones.declined) AS Cantidad_declinadas
261     FROM Tres_ultimas_transacciones
262     GROUP BY Tres_ultimas_transacciones.card_id)
263
264   SELECT
265     Transacciones_declinadas.card_id,
266     CASE
267       WHEN Transacciones_declinadas.Cantidad_declinadas = 3 THEN "Tarjeta inactiva"
268       ELSE "Tarjeta activa"
269     END AS Estado_tarjeta
270   FROM Transacciones_declinadas);
271
<
Output :>
Action Output
# Time Action
1 12:39:07 CREATE TABLE IF NOT EXISTS Credit_cards_status AS(WITH Transacciones_ordenadas AS ( SELE... 5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0
Duration / F
0.704 sec
Result Grid | Filter Rows: [ ] | Export: [ ] | Wrap Cell Content: [ ] | Fetch rows: [ ]
card_id Estado_tarjeta
CCS-4857 Tarjeta activa
CCS-4858 Tarjeta activa
CCS-4859 Tarjeta activa
CCS-4860 Tarjeta activa
CCS-4861 Tarjeta activa
CCS-4862 Tarjeta activa
CCS-4863 Tarjeta activa
CCS-4864 Tarjeta activa
credit_cards_status 20 ×
Output :>
Action Output
# Time Action
1 12:41:32 SELECT* FROM credit_cards_status
Message
5000 row(s) returned
Duration / F
0.000 sec / 0.0
```

Para crear una tabla que indique si una tarjeta está activa o inactiva, según si sus 3 transacciones más recientes fueron declinadas o no hemos procedido con subconsultas, creando CTE (Common Table Expression) con WITH según los siguientes pasos:

- Primero hemos creado la CTE **Transacciones_ordenadas** usando una función ventana que asigna un número de orden a cada transacción por tarjeta (card_id), ordenadas desde la más reciente a la más antigua.
- Hemos obtenido una tabla con las transacciones numeradas según su antigüedad, siendo 1 la más reciente)
- Creamos una segunda CTE **Tres_ultimas_transacciones** que selecciona solo las 3 transacciones más recientes de cada tarjeta (según el ranking anterior) e incluimos la columna declined para tener la información de si fueron rechazadas o no.
- Creamos una tercera CTE **Transacciones_declinadas** que suma la cantidad de transacciones declinadas dentro de la CTE **Tres_ultimas_transacciones**.
- Finalmente, la tabla **Credit_card_status** se generó a partir de la CTE Transacciones_declinadas utilizando una expresión CASE que clasifica las tarjetas como activas o inactivas según si sus tres últimas transacciones fueron declinadas.

```

302  #Configuramos la PK y FK de esa tabla intermedia
303 • ALTER TABLE credit_cards_status
304   ADD CONSTRAINT Pk_credit_cards_status
305   PRIMARY KEY (card_id);
306
307 • ALTER TABLE credit_cards_status
308   ADD CONSTRAINT FK_CreditCardsStatus_CreditCards
309   FOREIGN KEY (card_id) REFERENCES credit_cards(id);
310
311 • DESCRIBE credit_cards_status;
312
313
< -----
Output ::::::::::::::::::::
Action Output
# | Time | Action
1 12:42:42 ALTER TABLE credit_cards_status ADD CONSTRAINT Pk_credit_cards_status PRIMARY KEY(card_id) 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0 0.187 sec
2 12:42:44 ALTER TABLE credit_cards_status ADD CONSTRAINT FK_CreditCardsStatus_CreditCards FOREIGN KEY... 5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0 0.250 sec

```

```

311 • DESCRIBE credit_cards_status;
< -----
Result Grid | Filter Rows: [ ] Export: [ ] | Wrap Cell Content: [ ]
Field Type Null Key Default Extra
card_id varchar(255) NO PRI NULL
Estado_tarjeta varchar(16) NO

Result 21 x
Output ::::::::::::::::::::
Action Output
# | Time | Action
1 12:44:03 DESCRIBE credit_cards_status
Message
2 row(s) returned
Duration / Fetch
0.000 sec / 0.0

```

Una vez la tabla creada procedemos a configurar su Primary Key (card_id) y vincularla con la tabla credit_cards mediante Foreign Key teniendo una relación de 1 a 1.

EJERCICIO 1

¿Cuántas tarjetas están activas?

```
302 #¿Cuántas tarjetas están activas?
303
304 • SELECT COUNT(c.Estado_tarjeta) AS Cantidad_tarjetas_activas
305   FROM credit_cards_status AS c
306  WHERE c.Estado_tarjeta = "Tarjeta activa";
307
308 ... 2
Result Grid | Filter Rows: [ ] Export: [ ] Wrap Cell Content: [ ]
Cantidad_tarjetas_activas
4995
Result 10 x
Output
Action Output
# Time Action
9 15:49:33 SELECT COUNT(c.Estado_tarjeta) AS Cantidad_tarjetas_activas FROM credit_cards_status AS c WHE... Message
1 row(s) returned
Duration / Fe
```

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 tienes product_ids. Genera la siguiente consulta:

Creación y carga de los datos de la tabla products

```
328 #Primero creamos la tabla products y cargamos los datos siguiendo los mismos pasos que para el nivel 1
329 • CREATE TABLE products (
330   id INT,
331   product_name VARCHAR(50),
332   price VARCHAR(10),
333   colour VARCHAR(20),
334   weight DECIMAL(10,2),
335   warehouse_id VARCHAR(20));
336
337 • LOAD DATA INFILE 'C://ProgramData//MySQL//MySQL Server 8.0//Uploads//products.csv'
338   INTO TABLE products
339   FIELDS TERMINATED BY ','
340   OPTIONALLY ENCLOSED BY ""
341   LINES TERMINATED BY '\n'
342   IGNORE 1 ROWS;
343
344 #cambiamos el data type de la columna price, primero quitando el símbolo dolard y posteriormente convertimos a DECIMAL
< Output
Action Output
# Time Action
1 12:45:57 CREATE TABLE products (id INT, product_name VARCHAR(50), price VARCHAR(10), colour VARCHAR... 0 row(s) affected Duration / Fe
0.062 sec
2 12:45:59 LOAD DATA INFILE 'C://ProgramData//MySQL//MySQL Server 8.0//Uploads//products.csv' INTO TABL... 100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0 Duration / Fe
0.062 sec
```

Para cargar los datos del archivo CSV products hemos procedido de la misma manera que para el nivel 1, es decir crear previamente la tabla y configurar los data type de las columnas para cargar posteriormente los datos.

La columna id se ha definido como INT, ya que contiene identificadores numéricos enteros para cada producto.

Las columnas product_name, colour y warehouse_id se han configurado como VARCHAR dado que almacenan datos que cuentan con letras y números con distintas longitudes.

Para la columna price, se eligió el tipo VARCHAR para poder cargar los datos, ya que los precios contienen el símbolo “\$” y nos daría error si intentáramos configurarla como DECIMAL.

En el caso de weight, se usó DECIMAL(10,2), permitiendo registrar valores numéricos con hasta dos decimales.

```

344  #cambiamos el data type de la columna price, primero quitando el símbolo dolard y posteriormente convertimos a DECIMAL
345 • UPDATE products
346   SET price = REPLACE(price, "$", "");
347
348 • ALTER TABLE products
349   MODIFY COLUMN price DECIMAL(10,2);
350
351  #verificamos:
352 • DESCRIBE products;
353

```

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

Field	Type	Null	Key	Default	Extra
id	int	YES		NULL	
product_name	varchar(50)	YES		NULL	
price	decimal(10,2)	YES		NULL	
colour	varchar(20)	YES		NULL	
weight	decimal(10,2)	YES		NULL	
warehouse_id	varchar(20)	YES		NULL	

Result 22 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	12:47:26	UPDATE products SET price = REPLACE(price, "\$", "")	100 row(s) affected Rows matched: 100 Changed: 100 Warnings: 0	0.032 sec
2	12:47:29	ALTER TABLE products MODIFY COLUMN price DECIMAL(10,2)	100 row(s) affected Records: 100 Duplicates: 0 Warnings: 0	0.078 sec
3	12:47:53	DESCRIBE products	6 row(s) returned	0.000 sec / 0

```

351  #verificamos:
352 • DESCRIBE products;
353
354 • SELECT *
355   FROM products;
356
357

```

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

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	#e4e4c4	0.80	WH--1
7	north of Casterly	63.33	#b7b7b7	0.60	WH--2
R	Minterfall	37.37	#383838	1.40	WH--3

products 24 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	12:49:03	SELECT * FROM products	100 row(s) returned	0.000 sec / 0

```

358 #Anadimos la PK
359 • ALTER TABLE products
360 ADD CONSTRAINT Pk_products
361 PRIMARY KEY (id);
362
363 • DESCRIBE products;
364

```

Result Grid

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	
product_name	varchar(50)	YES		NULL	
price	decimal(10,2)	YES		NULL	
colour	varchar(20)	YES		NULL	
weight	decimal(10,2)	YES		NULL	
warehouse_id	varchar(20)	YES		NULL	

Result 25

Output

#	Time	Action	Message	Duration / Fetch
1	12:50:28	ALTER TABLE products ADD CONSTRAINT Pk_products PRIMARY KEY (id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.109 sec
2	12:50:31	DESCRIBE products	6 row(s) returned	0.000 sec / 0

Una vez cargados los datos, procedemos a cambiar el data type de la columna price de VARCHAR a DECIMAL (10,2), quitando previamente todos los símbolos “\$”.

Finalmente configuramos la **Primary Key** de la tabla product.

Creación una tabla con la que podamos unir los datos del nuevo archivo products.csv con la base de datos creada

```

365 #transformamos la columna products_id de transactions a formato JSON ARRAY creando una nueva columna en transactions
366 • ALTER TABLE transactions
367 ADD COLUMN separated_products VARCHAR(255);
368
369 • UPDATE transactions
370 SET separated_products = REPLACE (product_ids, ',', '"', '"');
371
372 • UPDATE transactions
373 SET separated_products = CONCAT('[',separated_products,']');
374
375 • CREATE TABLE IF NOT EXISTS transactionsProducts
376 SELECT t.id AS transaction_id, j.product_id
377 FROM transactions AS t
378 CROSS JOIN
379 JSON_TABLE (t.separated_products, '$[*]')
380   COLUMNS (product_id INT PATH "$") AS j;
381

```

Output

#	Time	Action	Message	Duration / Fetch
2	12:52:08	UPDATE transactions SET separated_products = REPLACE (product_ids, ',', '"', '"')	100000 row(s) affected Rows matched: 100000 Changed: 100000 Warnings: 0	6.656 sec
3	12:52:28	UPDATE transactions SET separated_products = CONCAT('[',separated_products,']')	100000 row(s) affected Rows matched: 100000 Changed: 100000 Warnings: 0	4.141 sec
4	12:52:35	CREATE TABLE IF NOT EXISTS transactionsProducts SELECT t.id AS transaction_id, j.product_id FRO...	253391 row(s) affected Records: 253391 Duplicates: 0 Warnings: 0	3.313 sec

Para poder crear la tabla, primero es necesario normalizar la tabla transactions, ya que inicialmente almacena un registro por cada transacción, y en la columna product_ids pueden aparecer varios identificadores de producto en una misma fila.

Para resolverlo, se utilizó la función JSON_TABLE, creando una columna intermedia denominada separated_products en la tabla transactions. Esta columna transforma el contenido de product_ids al formato array de JSON mediante el uso de las funciones REPLACE y CONCAT.

Posteriormente, hemos creado la tabla **transactionsProducts** para descomponer los datos del campo separated_products de la tabla transactions, que contiene una lista de productos ahora en formato JSON dentro de una misma transacción.

Hicimos una CROSS JOIN entre la tabla transactions y un JSON_TABLE que nos permitió convertir el contenido JSON de la columna separated_products en una tabla relacional.

El resultado es una nueva tabla **transactionsProducts** donde cada fila representa una combinación entre una transacción (transaction_id) y un producto (product_id), permitiendo relacionar fácilmente qué productos participaron en cada transacción. En otras palabras, obtenemos una mayor granularidad de los datos.

The screenshot shows the MySQL Workbench interface with two tabs: 'Output' and 'Result Grid'. In the 'Output' tab, three SQL statements are listed:

```
383 #Configuramos la PK y relacionamos esa tabla con la tabla products y transactions mediante FK
384 • ALTER TABLE transactionsProducts
385   ADD CONSTRAINT Pk_transactionsProducts
386   PRIMARY KEY (transaction_id, product_id);
387
388 • ALTER TABLE transactionsProducts
389   ADD CONSTRAINT Fk_transactionsProducts_products
390   FOREIGN KEY (product_id) REFERENCES products (id);
391
392 • ALTER TABLE transactionsProducts
393   ADD CONSTRAINT Fk_transactionsProducts_transactions
394   FOREIGN KEY (transaction_id) REFERENCES transactions(id);
395
```

In the 'Result Grid' tab, the table structure is displayed:

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

Below the table structure, the results of the DESCRIBE command are shown:

```
396 • DESCRIBE transactionsProducts;
397
Result 26 ×
```

The 'Output' tab also shows the results of the DESCRIBE command:

#	Time	Action
1	12:53:50	DESCRIBE transactionsProducts

Message: 2 row(s) returned

Configuramos la **Primary Key** compuesta de esa nueva tabla intermedia y la relacionamos como Foreign Key con la tabla products y transactions mediante la columna id de la tabla products y la columna id de transactions.

Entre transactions y TransactionsProducts hay una relación uno a muchos (una transacción puede tener muchos registros en TransactionsProducts) y entre products y TransactionsProducts también hay una relación uno a muchos (un producto puede aparecer en muchas filas de TransactionsProducts).

```

398     #procedemos a eliminar la columna separated_products de la tabla transactions para dejarla como estaba inicialmente
399 •   ALTER TABLE transactions
400     DROP COLUMN separated_products;
401
402     #verificación:
403 •   SELECT*
404     FROM transactions;
405
406

```

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

ID	card_id	business_id	timestamp	amount	declined	product_ids	user_id	lat	longitude
00043AA4-2949-494B-ASDD-A5BAE3BB19DD	CcS-9294	b-2458	2024-08-28 07:16:46	395.43	0	16, 26, 97, 87	4713	46.1999	1.43554
000447FE-B650-4DCF-85DE-C7ED0EE1CAA	CcS-5019	b-2370	2016-12-21 20:07:18	155.63	0	66, 69, 87	438	41.5972	12.2218
00045D6B-ED2E-4F2F-8186-CEE074D875D0	CcS-6699	b-2390	2020-07-14 15:37:45	326.01	0	30, 11, 16, 81	2118	29.7573	-95.3796
000481C3-1C26-4FEF-83A0-4CD0EB004BBD	CcS-6696	b-2230	2017-09-04 19:44:53	161.60	0	72	2115	53.5489	-113.503
00051AA4-9CBE-4268-B070-C38062A1B3E2	CcS-7606	b-2266	2017-01-05 18:19:25	148.91	0	18	3025	52.2084	5.69081

transactions 27 x

Output:

#	Time	Action	Message	Duration / F
1	12:55:40	ALTER TABLE transactions DROP COLUMN separated_products	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.172 sec
2	12:55:44	SELECT* FROM transactions	100000 row(s) returned	0.015 sec / 0

Finalmente eliminamos la columna que habíamos creado para convertir la columna de product_ids de transactions a formato JSON.

EJERCICIO 1

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

```

387 •   SELECT tp.product_id, p.product_name, COUNT(*) AS Cantidad_ventas
388     FROM transactionsProducts AS tp
389     JOIN transactions AS t
390     ON t.id = tp.transaction_id
391     JOIN products AS p
392     ON p.id = tp.product_id
393     WHERE t.declined = 0
394     GROUP BY tp.product_id
395     ORDER BY tp.product_id;
396

```

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

product_id	product_name	Cantidad_ventas
1	Direwolf Stannis	2467
2	Tarly Stark	2562
3	duel tourney Lannister	2520
4	warden south duel	2573
5	skywalker ewok	2543
6	dooku solo	2487

Result 41 x

Output:

#	Time	Action	Message	Duration
56	21:59:17	SELECT tp.product_id, p.product_name, COUNT(*) AS Cantidad_ventas FROM transactionsProducts A...	100 row(s) returned	1.219