

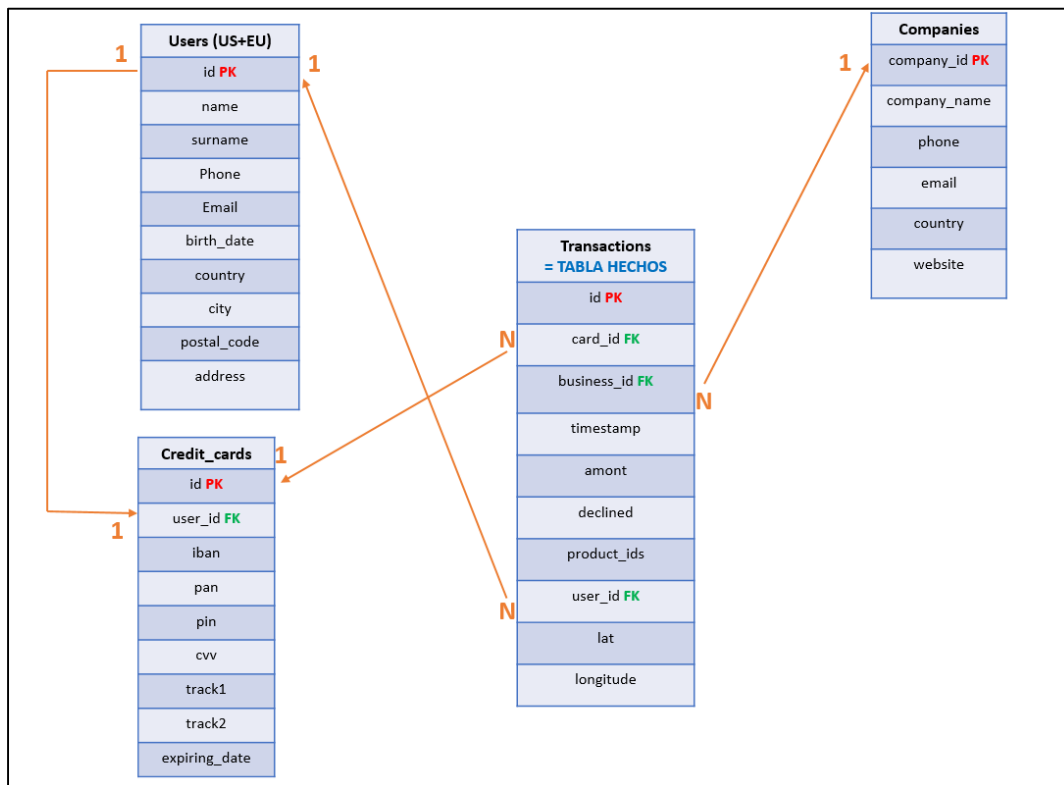
SPRINT 4 (MYSQL) MARINE FERNANDEZ

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 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 **Primary 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.

```
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 );
```

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) .

```
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 INT,  
25     user_id INT,  
26     lat FLOAT,  
27     longitude FLOAT  
28 ) ;
```

Para la **tabla transactions** hemos escogido el data type VARCHAR para las columnas id, credit_card, business_id 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 las columnas products_ids y 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 );

```

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

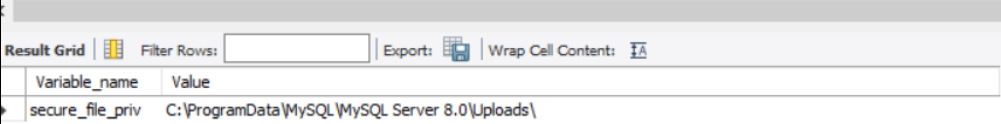
```

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 a
56 #Esto se hizo utilizando el siguiente comando:
57
58 • SHOW VARIABLES LIKE 'secure_file_priv';
59
```



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

Result 11 x

Output

Action Output

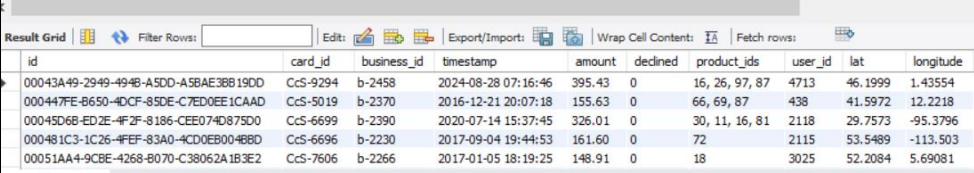
#	Time	Action	Message
10	12:52:34	SHOW VARIABLES LIKE 'secure_file_priv'	1 row(s) returned

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;
71
```



id	card_id	business_id	timestamp	amount	declined	product_ids	user_id	lat	longitude
00043A49-2949-494B-A5DD-ASBAE38B19DD	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-8SDE-C7ED0EE1CAAD	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 14 x

Output

Action Output

#	Time	Action	Message
13	12:58:31	SELECT* FROM transactions	100000 row(s) returned

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

```

Result Grid

company_id	company_name	phone	email	country	website
b-2222	Ac Fermentum Incorporated	06 85 56 52 33	donec.porttitor.tellus@yahoo.net	Germany	https://instagram.com/site
b-2226	Magna A Neque Industries	04 14 44 64 62	risus.donec.nibh@icloud.org	Australia	https://whatsapp.com/group/9
b-2230	Fusce Corp.	08 14 97 58 85	risus@protonmail.edu	United States	https://pinterest.com/sub/cars
b-2234	Convallis In Incorporated	06 66 57 29 50	mauris.ut@aol.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

companies 17 x

Output

Action Output

#	Time	Action	Message
15	12:58:46	SELECT* FROM transactions	100000 row(s) returned

```

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;

```

Result Grid

id	user_id	iban	pan	pin	cvv	track1	track2	expiring_date
CcS-4857	276	XX4857591835292505850771	2314242385113924	1819	467	%B2314242385113924^LWCBUDLWCBUD^22...	%B2314242385113924=2410101518363164?	2025-09-27
CcS-4858	277	XX8581768137002436094025	6582720299715533	3964	817	%B6582720299715533^TIQMVTIQMVI^2404...	%B6582720299715533=241110104546272?	2028-12-28
CcS-4859	278	XX7826930491423553609370	8861684536289642	4983	277	%B8861684536289642^COFBGDCOFBGD^280...	%B8861684536289642=2502101761665371?	2026-11-26
CcS-4860	279	XX5559390368835304645299	2481155515498459	6876	661	%B2481155515498459^TIJTUTIJUTU^31040...	%B2481155515498459=2602101514414395?	2027-07-27
CcS-4861	280	XX2035182877195191627307	1308930301149557	5710	398	%B1308930301149557^HPOBNZHPOBNZ^330...	%B1308930301149557=2805101751305028?	2026-04-25

credit_cards 20 x

Output

Action Output

#	Time	Action	Message
19	13:07:00	SELECT* FROM credit_cards	5000 row(s) returned

```

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
109 #verificación

```

Result Grid

id	name	surname	phone	email	country	city	postal_code	address	birth_date
1	Zeus	Gamble	1-282-581-0551	interdum.enim@protonmail.edu	United States	New York	10001	348-7818 Sagittis St.	1985-11-17
2	Garrett	Mcconnell	(718) 257-2412	integer.vitae.nibh@protonmail.org	United States	Philadelphia	19101	903 Sit Ave	1992-08-23
3	Ciaran	Harrison	(522) 598-1365	interdum.feugiat@aol.org	United States	Houston	77001	736-2063 Tellus St.	1998-04-29
4	Howard	Stafford	1-411-740-3269	ornare.egestas@icloud.edu	United States	Phoenix	85001	Ap #545-2244 Erat. Rd.	1989-02-18
5	Hayfa	Pierce	1-554-541-2077	et.malesuada.fames@hotmail.org	United States	Philadelphia	19101	341-2821 Ultrices Av.	1998-09-26

users 23 x

Output

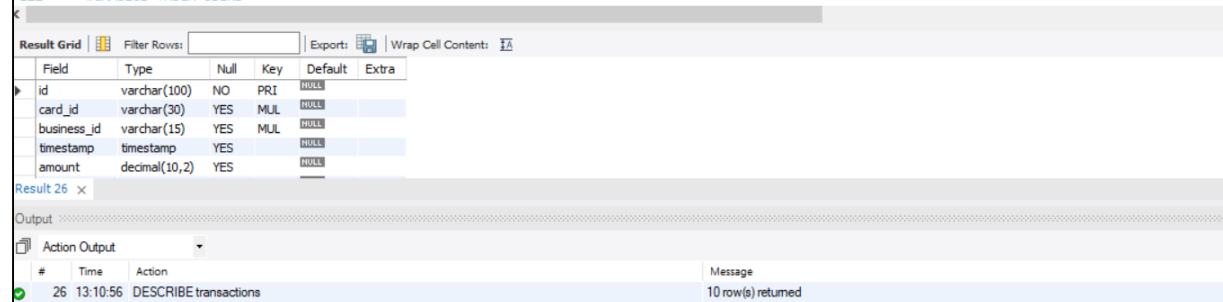
Action Output

#	Time	Action	Message
22	13:07:43	SELECT* FROM users	5000 row(s) returned

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

```
115  #- CAMBIOS TABLA TRANSACTIONS
116  #configuracion de PK
117  • ALTER TABLE transactions
118    ADD CONSTRAINT Pk_transactions PRIMARY KEY (id);
119
120  • DESCRIBE transactions;
121  #CAMBIOS TABLA USERS
```



Field	Type	Null	Key	Default	Extra
id	varchar(100)	NO	PRI		NULL
card_id	varchar(30)	YES	MUL		NULL
business_id	varchar(15)	YES	MUL		NULL
timestamp	timestamp	YES			NULL
amount	decimal(10,2)	YES			NULL

Result 26 x

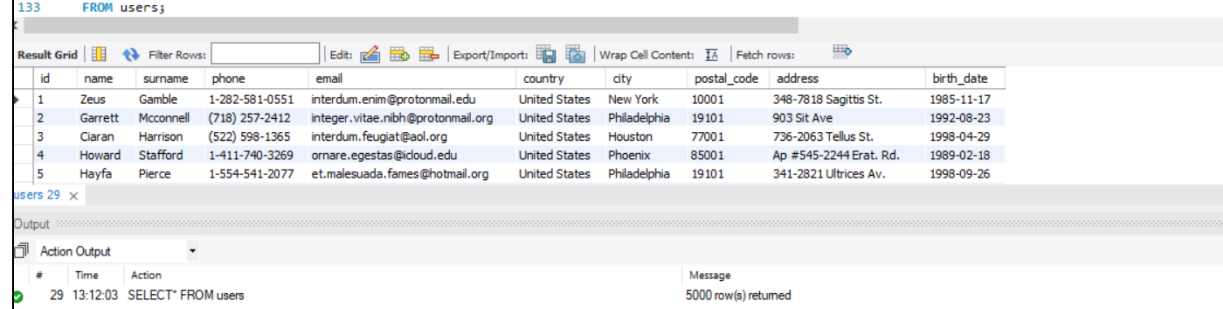
Output

Action Output

#	Time	Action	Message
26	13:10:56	DESCRIBE transactions	10 row(s) returned

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
131  #verificacion
132  • SELECT*
133    FROM users;
```



id	name	surname	phone	email	country	city	postal_code	address	birth_date
1	Zeus	Gamble	1-282-581-0551	interdum.enim@protonmail.edu	United States	New York	10001	348-7818 Sagittis St.	1985-11-17
2	Garrett	Mcconnell	(718) 257-2412	integer.vitae.nibh@protonmail.org	United States	Philadelphia	19101	903 Sit Ave	1992-08-23
3	Ciaran	Harrison	(522) 598-1365	interdum.feugiat@aol.org	United States	Houston	77001	736-2063 Tellus St.	1998-04-29
4	Howard	Stafford	1-411-740-3269	omare.egestas@icloud.edu	United States	Phoenix	85001	Ap #545-2244 Erat. Rd.	1989-02-18
5	Hayfa	Pierce	1-554-541-2077	et.malesuada.fames@hotmail.org	United States	Philadelphia	19101	341-2821 Ultrices Av.	1998-09-26

users 29 x

Output

Action Output

#	Time	Action	Message
29	13:12:03	SELECT* FROM users	5000 row(s) returned

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

```

137 #CAMBIOS TABLA CREDIT_CARD
138
139 • ALTER TABLE credit_cards
140   ADD CONSTRAINT Pk_credit_cards PRIMARY KEY (id);
141
142 #Cambiamos el formato de la columna expiring_date
143 • UPDATE credit_cards
144   SET expiring_date = STR_TO_DATE (expiring_date, "%m/%d/%y");
145
146 #verificaciones
147 • SELECT*
148   FROM credit_cards;
149
150 • DESCRIBE credit_cards;
151
152 #CAMBIOS TABLA COMPANIES

```

Result Grid

id	user_id	iban	pan	pin	cvv	track1	track2	expiring_date
CcS-4857	276	XX4857591835292505850771	2314242385113924	1819	467	%B2314242385113924~LWCBUDLWCBUD^22...	%B2314242385113924=2410101518363164?	2025-09-27
CcS-4858	277	XX8581768137002436094025	6582720299715533	3964	817	%B6582720299715533~TIQMVTITQMV1^2404...	%B6582720299715533=2411101104546272?	2028-12-28
CcS-4859	278	XX7826930491423553609370	8861684536289642	4983	277	%B8861684536289642~COPBGDCOPBGD^280...	%B8861684536289642=2502101761665371?	2026-11-26
CcS-4860	279	XX5559590368835304645299	2481155515498459	6876	661	%B2481155515498459~TIJUTUTIJUTU^31040...	%B2481155515498459=2602101514414395?	2027-07-27
CcS-4861	280	XX2035182877195191627307	1308930301149557	5710	398	%B1308930301149557~HPOBNZHPOBNZ^330...	%B1308930301149557=2805101751305028?	2026-04-25

credit_cards 32 x

Output

Action Output

#	Time	Action	Message
32	13:14:41	SELECT* FROM credit_cards	5000 row(s) returned

Tabla companies

```

152 #CAMBIOS TABLA COMPANIES
153
154 • ALTER TABLE companies
155   ADD CONSTRAINT Pk_companies PRIMARY KEY (company_id);
156
157 #verificacion
158 • DESCRIBE companies;
159

```

Result Grid

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

Result 34 x

Output

Action Output

#	Time	Action	Message
34	13:14:42	SELECT* FROM credit_cards	5000 row(s) returned

Finalmente, procedemos a relacionar las tablas entre sí:

```
160 #Procedemos a relacionar las tablas entre sí : configuración de foreign keys en la tabla transactions y relacionamos las tablas credit
161 • ALTER TABLE transactions
162   ADD CONSTRAINT Fk_companies_transactions
163   FOREIGN KEY (business_id) REFERENCES companies(company_id);
164
165 • ALTER TABLE transactions
166   ADD CONSTRAINT Fk_users_transactions
167   FOREIGN KEY (user_id) REFERENCES users(id);
168
169 • ALTER TABLE transactions
170   ADD CONSTRAINT Fk_credit_cards_transactions
171   FOREIGN KEY (card_id) REFERENCES credit_cards(id);
172
173 • ALTER TABLE credit_cards
174   ADD CONSTRAINT Fk_users_credit_cards
175   FOREIGN KEY (user_id) REFERENCES users(id);
```

```
177 #verificación
178 • DESCRIBE transactions;
179 • DESCRIBE credit_cards;
180
```

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

Field	Type	Null	Key	Default	Extra
id	varchar(100)	NO	PRI	NULL	
card_id	varchar(30)	YES	MUL	NULL	
business_id	varchar(15)	YES	MUL	NULL	
timestamp	timestamp	YES		NULL	
amount	decimal(10,2)	YES		NULL	

Result 38 x

Output

Action Output

#	Time	Action	Message
38	13:18:31	DESCRIBE transactions	10 row(s) returned

```
177 #verificación
178 • DESCRIBE transactions;
179 • DESCRIBE credit_cards;
180
```

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

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	

Result 41 x

Output

Action Output

#	Time	Action	Message
41	13:18:57	DESCRIBE credit_cards	9 row(s) returned

```

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: [IA](#)

user_id	COUNT(*)
---------	----------

Result 45 x

Output

Action Output

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

EJECICIO 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: [IA](#)

id	name	surname	phone	email	country	city	postal_code	address	birth_date
185	Molly	Gilliam	0800 120 8023	donec@outlook.couk	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	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

EJECICIO 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

iban	Media_importe
XX383017813919620199366352	680.69
XX637706357397570394973913	680.01
XX971393971465292202312259	645.46
XX171847116928892375969307	628.89
XX225424638818542406223575	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:

```
230 #CREATE TABLE declined_transactions AS
231
232 #Creamos una tabla que indique si las 3 transacciones mas recientes de cada credit id fueron declinadas ==> tarjeta inactiva sino activa
233 #creamos varias CTE, para tener mejor organizacion y legibilidad
234 • CREATE TABLE IF NOT EXISTS Credit_cards_status AS(
235 WITH Transacciones_ordenadas AS (
236 SELECT
237 t.id,
238 t.card_id,
239 ROW_NUMBER() OVER (PARTITION BY t.card_id ORDER BY t.timestamp DESC) AS Ranking_transacciones
240 FROM transactions AS t),
241
242 Tres_ultimas_transacciones AS (
243 SELECT
244 t.card_id,
245 t.declined,
246 t.timestamp
247 FROM Transacciones_ordenadas
248 JOIN transactions AS t
249 ON t.id = Transacciones_ordenadas .id
250 WHERE Ranking_transacciones <=3),
251
252 Transacciones_declinadas AS (
253 SELECT
254 Tres_ultimas_transacciones.card_id,
255 SUM(Tres_ultimas_transacciones.declined) AS Cantidad_declinadas
256 FROM Tres_ultimas_transacciones
257 GROUP BY Tres_ultimas_transacciones.card_id)
```

258	
259	SELECT
260	Transacciones_declinadas.card_id,
261	CASE
262	WHEN Transacciones_declinadas.Cantidad_declinadas = 3 THEN "Tarjeta inactiva"
263	ELSE "Tarjeta activa"
264	END AS Estado_tarjeta
265	FROM Transacciones_declinadas);
266	
267	SELECT*
268	FROM credit_cards_status;
269	

card_id	Estado_tarjeta
CcS-4857	Tarjeta activa
CcS-4858	Tarjeta activa
CcS-4859	Tarjeta activa
CcS-4860	Tarjeta activa
CcS-4861	Tarjeta activa

#	Time	Action	Message
4	14:50:30	SELECT* FROM credit_cards_status	5000 row(s) returned

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.

```

294 #Configuramos la PK y FK de esa tabla intermedia
295 • ALTER TABLE credit_cards_status
296 ADD CONSTRAINT Pk_credit_cards_status
297 PRIMARY KEY (card_id);
298
299 • ALTER TABLE credit_cards_status
300 ADD CONSTRAINT FK_CreditCardsStatus_CreditCards
301 FOREIGN KEY (card_id) REFERENCES credit_cards(id);
302
303 • DESCRIBE credit_cards_status;
304

```

Result Grid

Field	Type	Null	Key	Default	Extra
card_id	varchar(255)	NO	PRI	NULL	
Estado_tarjeta	varchar(16)	NO			

Result 66 x

Output

Action Output

#	Time	Action	Message
95	23:50:55	DESCRIBE credit_cards_status	2 row(s) returned

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.

EJECICIO 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

```

Result Grid

Cantidad_tarjetas_activas
4995

Result 10 x

Output

Action Output

#	Time	Action	Message
9	15:49:33	SELECT COUNT(c.Estado_tarjeta) AS Cantidad_tarjetas_activas FROM credit_cards_status AS c WHE...	1 row(s) returned

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

```
312 #Primero creamos la tabla products y cargamos los datos siguiendo los mismos pasos que para el nivel 1
313 CREATE TABLE products (
314     id INT,
315     product_name VARCHAR(50),
316     price VARCHAR(10),
317     colour VARCHAR(20),
318     weight DECIMAL(10,2),
319     warehouse_id VARCHAR(20));
320
321 LOAD DATA INFILE 'C://ProgramData//MySQL//MySQL Server 8.0//Uploads//products.csv'
322 INTO TABLE products
323 FIELDS TERMINATED BY ','
324 OPTIONALLY ENCLOSED BY '"'
325 LINES TERMINATED BY '\n'
326 IGNORE 1 ROWS;
```

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 columna product_name, colour y warehouse_id se ha 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.

```

328 #cambiamos el data type de la columna price, primero quitando el símbolo dolard y posteriormente convertimos a DECIMAL
329 • UPDATE products
330 SET price = REPLACE(price, "$", "");
331 • ALTER TABLE products
332 MODIFY COLUMN price DECIMAL(10,2);
333
334 #verificamos:
335 • SELECT *
336 FROM products;
337 • DESCRIBE products;
338
339 #Anadimos la PK
340 • ALTER TABLE products
341 ADD CONSTRAINT Pk_products
342 PRIMARY KEY (id);

```

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

Result 15 x

Output

#	Time	Action	Message
14	16:01:07	DESCRIBE products	6 row(s) returned

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

```

353 #transformamos la columna products_id de transactions a formato JSON ARRAY creando una nueva columna en transactions
354 • ALTER TABLE transactions
355 ADD COLUMN separated_products VARCHAR(255);
356
357 • UPDATE transactions
358 SET separated_products = REPLACE (product_ids, ',', '');
359
360 • UPDATE transactions
361 SET separated_products = CONCAT('["',separated_products,']');
362
363 • CREATE TABLE IF NOT EXISTS transactionsProducts
364 SELECT t.id AS transaction_id, j.product_id
365 FROM transactions AS t
366 CROSS JOIN
367 JSON_TABLE (t.separated_products, "$[*]"
368 COLUMNS (product_id INT PATH "$")) AS j;
369

```

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.

```
371 #Configuramos la PK y relacionamos esa tabla con la tabla products mediante FK
372 • ALTER TABLE transactionsProducts
373   ADD CONSTRAINT Pk_transactionsProducts
374   PRIMARY KEY (transaction_id, product_id);
375
376 • ALTER TABLE transactionsProducts
377   ADD CONSTRAINT Fk_transactionsProducts_products
378   FOREIGN KEY (product_id) REFERENCES products (id);
379
380 • ALTER TABLE transactionsProducts
381   ADD CONSTRAINT Fk_transactionsProducts_transactions
382   FOREIGN KEY (transaction_id) REFERENCES transactions(id);
383
384 • DESCRIBE transactionsProducts;
```

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

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

Result 70 x

Output

Action Output

#	Time	Action	Message
104	00:10:53	DESCRIBE transactionsProducts	2 row(s) returned

Configuramos la **Primary Key** compuesta de esa nueva tabla intermediaria 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`).

```

374 #procedemos a eliminar la columna separated_products de la tabla transactions para dejarla como estaba inicialmente
375 • ALTER TABLE transactions
376 DROP COLUMN separated_products;
377
378 #verificacion:
379 • SELECT*
380 FROM transactions;
381
382

```

Result Grid

id	card_id	business_id	timestamp	amount	declined	product_ids	user_id	lat	longitude
00043A49-2949-494B-A5DD-A5BAE38B19DD	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-C7ED0EE1CAAD	CcS-5019	b-2370	2016-12-21 20:07:18	155.63	0	66, 69, 87	438	41.5972	12.2218
00045D68-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
0008A312-EDFE-4A4F-BC99-E9C92EC3CA4D	CcU-3358	b-2598	2023-09-23 04:51:43	294.59	0	35, 33, 19	215	53.5535	-113.499

transactions 37 x

Output

Action Output

#	Time	Action	Message
52	21:51:41	SELECT* FROM transactions	100000 row(s) returned

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

EJECICIO 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

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

Action Output

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