

Tasca S4.01. Creació de Base de Dades

Fecha 04/11/2025

Andrey Ismagilov

Nivel 1

Descàrrega els arxius CSV, estudia'ls i dissenya una base de dades amb un esquema d'estrella que contingui, almenys 4 taules.

Vamos a crear la base de datos **transactions_sp4**.

Hay 6 archivos en el formato CSV:

1. **transactions.csv** contiene la información de cada venta realizada en línea. Usando este archivo podemos crear la tabla **transaction**.

```
|id;card_id;business_id;timestamp;amount;declined;product_ids;user_id;lat;longitude  
CDDA7E40-544D-47BB-A4ED-671D08A950D9;Ccs-6894;b-2466;2018-12-12 08:05:17;161.88;0;75, 73, 98;2313;59.62050974356148;16.559977155728436  
09456357-8E9B-475A-8257-87A023699964;Ccs-5135;b-2342;2024-05-20 22:49:08;171.13;0;3;554;45.76458841901318;4.843056518287656  
C47C7C84-C174-4973-A768-825A9DB5582A;Ccs-8415;b-2250;2018-11-04 23:16:18;497.29;0;92, 85, 36, 23;3834;52.06849608409912;4.301099382555438  
2FB526AA-3844-4DDF-AC09-2EBCC255EE76;Ccs-6553;b-2610;2022-06-17 09:17:25;344.15;0;5, 27;1972;39.47598513179192;-0.3764194439184784  
0D877407-B85E-40D2-9229-6BD6F2B8DF0A;Ccs-7678;b-2454;2020-11-29 07:29:16;435.1;0;38, 32, 82, 62;3097;51.43656985032189;5.478525648707596
```

Vemos que los valores se separan por punto y coma.

El encabezado del archivo tiene los nombres de las siguientes columnas:

1. **id** – el identificador de la transacción que será la variable **id** del tipo **VARCHAR(255)** en la tabla **transaction**. Será el **Primary Key** de la tabla **transaction**.
2. **card_id** – el identificador la tarjeta de crédito del tipo **VARCHAR(20)**. Será la **credit_card_id** en la tabla y tambien el **FOREIGN KEY** de la tabla **transaction** que estará vinculada con la variable **id** de la tabla **credit_cards** a continuación.
3. **business_id** – el identificador de la empresa que vende los productos en línea. Será el variable **company_id** del tipo **VARCHAR(20)** y el **FOREIGN KEY** de la tabla **transaction** que estará vinculada con la variable **id** de la tabla **companies** luego.
4. **timestamp** – timestamp de la transacción. Será la **timestamp** del tipo **VARCHAR(255)**.
5. **amount** – importe de la transacción. La **amount** del tipo **DECIMAL(10, 2)**.
6. **declined** – es igual a **0** si la transacción realizada e igual a **1** si rechazada. Será la **declined** del tipo **TINYINT**.
7. **product_ids** – la lista de los productos incluidos en la transacción (venta). Se separan por coma. Será la **product_ids** del tipo **VARCHAR(255)**.
8. **user_id** – el identificador del comprador. Será la **user_id** del tipo **INT** y el **FOREIGN KEY** de la tabla **transaction** que estará vinculada con la variable **id** de la tabla **data_users** luego.
9. **lat** - latitud del lugar donde se realizó la venta. Será la **lat** del tipo **FLOAT**.
10. **longitude** - longitud del lugar donde se realizó la venta. Será la **longitude** del tipo **FLOAT**.

2. **credit_cards.csv** guarda detalles sobre las tarjetas de crédito. Con este archivo creamos la tabla **credit_cards**.

```
|id,user_id,iban,pan,pin,cvv,track1,track2,expiring_date  
CcU-2938,275,TR301950312213576817638661,5424465566813633,3257,984,%B8383712448554646^WowsxejDpiew^86041142?7,%B7653863056044187=8007163336?3,10/30/22  
CcU-2945,274,D02685476374853747521656869,5142423821948828,9080,887,%B4621311609958661^UftuyfsSeimxn^0610628241?7,%B4149568437843501=5107140330?1,08/24/23  
CcU-2952,273,BG451VQL52710525608255,4556 453 55 5287,4598,438,%B2183285104307501^CddiytcUxwfdq^5907955430?9,%B6778580257827162=69068597400?7,06/29/21  
CcU-2959,272,CR7242477244335841535,372461377349375,3583,667,%B7281111956795320^XocddijBckecd^99016253?3,%B4246154489281853=2805223916?8,02/24/23  
CcU-2966,271,BG72LKTQ15627628377363,448566 886747 7265,4900,130,%B4728932322756223^JhlgyvuFbmwgj^72022894943?7,%B2318571115599881=8908215784?5,10/29/24
```

Vemos que los valores se separan por coma.

El encabezado del archivo tiene los nombres de las siguientes columnas:

1. **id** – el identificador de la tarjeta crédito que será la variable **id** del tipo **VARCHAR(20)** en la tabla **credit_cards** y el **Primary Key**.
2. **user_id** - el identificador del comprador. Será la **user_id** del tipo **INT**.
3. **iban** - un numero de cuenta bancaria vinculada a la tarjeta crédito, será la **iban** del tipo **VARCHAR(50)**.
4. **pan** - un número de la tarjeta que se encuentra impreso en el anverso (y a veces reverso) y que la identifica de forma única, será la **pan** del tipo **VARCHAR(50)**.
5. **pin** - un número de identificación personal de 4 dígitos que funciona como una clave secreta para autorizar transacciones, será la **pin** del tipo **VARCHAR(4)**.
6. **cvv** - un código de seguridad de 3 o 4 dígitos, será la **cvv** del tipo **VARCHAR(4)**.
7. **track1** – los datos en la banda magnética de la tarjeta de crédito, será la **track1** del tipo **VARCHAR(255)**.
8. **track2** - los datos en la banda magnética de la tarjeta de crédito, será la **track2** del tipo **VARCHAR(255)**.
9. **expiring_date** - la fecha de vencimiento de una tarjeta, será la **expiring_date** del tipo **VARCHAR(20)**.

3. **companies.csv** - las empresas que vendan sus productos en línea.

```
|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,Fuse 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
```

Vemos que los valores se separan por coma.

El encabezado del archivo tiene los nombres de las siguientes columnas:

1. **id** – el identificador de la empresa. Será el **Primary Key** de la tabla **companies** del tipo **VARCHAR(20)**.
2. **company_name** – nombre de la empresa. Será la **company_name** del tipo **VARCHAR(255)**.
3. **phone** – el numero de telefono. Será la **phone** del tipo **VARCHAR(15)**.
4. **email** – el correo electronico de la empresa. Será la **email** del tipo **VARCHAR(100)**.
5. **country** – el pais donde esta la empresa. Será la **country** del tipo **VARCHAR(100)**.
6. **website** – la pagina web de la empresa. Será la **website** del tipo **VARCHAR(255)**.

4. **european_users.csv** - la información detallada sobre compradores que viven en la zona europea.

```
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 Pede 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.
```

Los valores se separan por coma.

El encabezado del archivo tiene los nombres de las siguientes columnas:

1. **id** – el identificador del comprador. Será el **Primary Key** de la tabla **data_user** del tipo **INT**.
2. **name** – el nombre del comprador. Será la **name** del tipo **VARCHAR(100)**.
3. **surname** – el apellido del comprador. Será la **surname** del tipo **VARCHAR(100)**.
4. **phone** – el numero de telefono del comprador. Será la **phone** del tipo **VARCHAR(150)**.
5. **email** – el correo electronico del comprador. Será la **email** del tipo **VARCHAR(150)**.
6. **birth_date** – el cumpleaños del comprador. Será la **birth_date** del tipo **VARCHAR(100)**.
7. **country** – el pais donde esta la empresa. Será la **country** del tipo **VARCHAR(150)**.
8. **city** – la ciudad donde vive el comprador. Será la **city** del tipo **VARCHAR(150)**.
9. **postal_code** – el **codico postal del comprador**. Será la **postal_code** del tipo **VARCHAR(100)**.
10. **address** – la dirección del comprador. Será la **address** del tipo **VARCHAR(255)**.

5. **american_users.csv** - la información detallada sobre compradores que viven en la zona americana.

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

El encabezado del archivo tiene los nombres iguales que el archivo **european_users.csv** por eso combinaremos la tabla **american_users** y la **european_users** a la unica tabla **data_users** luego.

6. **products.csv** -almacena la información sobre cada producto que se venda en línea.

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

Vemos que los valores se separan por coma.

El encabezado del archivo tiene los nombres de las siguientes columnas:

1. **id** – el codico del producto. Será la variable **id** de la tabla **products** del tipo **INT** y el **PRIMARY KEY** de la tabla.
2. **product_name** – el nombre del producto. Será la variable **product_name** del tipo **VARCHAR(50)**.
3. **price** – el precio del producto. Será la variable **price** del tipo **DECIMAL(10, 2)**.
4. **colour** – el codico de color del producto. Será la variable **colour** del tipo **VARCHAR(20)**.
5. **weight** – el peso del producto. Será la variable **weight** del tipo **VARCHAR(20)**.
6. **warehouse_id** – el identificador de almacén. Será la variable **warehouse_id** del tipo **VARCHAR(20)**.

Eligimos 5 de ellos: el **transactions.csv** que será la tabla **transaction** (la tabla de hechos de la esquema de estrella), el **credit_cards.csv** (será la tabla **credit_cards**), el **companies.csv** (la tabla **companies**), el **european_users.csv** y el **american_users.csv** que formaran la tabla **data_user**.

Al principio los tipos de columnas de todas las tablas seran de **VARCHAR** para que los datos vaya cargar sin errores. Luego cambiamos los tipos de columnas con datos numericos.

Paso 1. Creamos la database transactions_sp4.

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'Schemas' section, there is a tree view with several schemas like 'hosptiales', 'nens_nenes', 'olympics', 'sys', 'transactions', and 'transactions_sp4'. The 'transactions_sp4' schema is expanded, showing 'Tables', 'Views', 'Stored Procedures', and 'Functions'. The main pane displays SQL code for creating the database:

```

1 ##### NIVEL 1 #####
2 # Descàrrega els arxius CSV, estudia's i dissenya una base de dades amb un esquema d'estrella que contingui,
3 # almenys 4 taules.
4
5 • DROP DATABASE transactions_sp4;
6
7 • CREATE DATABASE IF NOT EXISTS transactions_sp4;
8
9 • USE transactions_sp4;
10

```

The 'Output' tab at the bottom shows the execution results:

#	Time	Action	Message
1	11:20:28	CREATE DATABASE IF NOT EXISTS transactions_sp4	1 row(s) affected
2	11:20:32	USE transactions_sp4	0 row(s) affected

Paso 2. Creamos la tabla transaction y luego insertamos los datos del archivo transactions.csv en la tabla.

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'Schemas' section, there is a tree view with several schemas like 'hosptiales', 'nens_nenes', 'olympics', 'sys', 'transactions', and 'transactions_sp4'. The 'transactions_sp4' schema is expanded, showing 'Tables', 'Views', 'Stored Procedures', and 'Functions'. The main pane displays SQL code for creating the 'transaction' table and loading data from a CSV file:

```

11 -- Creamos la tabla "transaction"
12 • CREATE TABLE transaction (
13     id          VARCHAR(255) NULL,
14     card_id     VARCHAR(20) NULL,
15     company_id  VARCHAR(20) NULL,
16     timestamp   VARCHAR(255) NULL,
17     amount      VARCHAR(255) NULL,
18     declined    VARCHAR(4) NULL,
19     product_ids VARCHAR(255) NULL,
20     user_id     VARCHAR(20) NULL,
21     lat         VARCHAR(255) NULL,
22     longitude   VARCHAR(255) NULL
23 );
24 -- Insertamos los datos del archivo transactions.csv en la tabla "transaction"
25 • LOAD DATA
26   INFILE 'C:\\transactions.csv'
27   -- C:\\Users\\evgen\\Downloads\\Barcelona Activa\\Data Analytics\\Sprint 4 SQL
28   -- 'C:\\\\Users\\\\evgen\\\\Downloads\\\\Barcelona Activa\\\\Data Analytics\\\\Sprint 4 SQL\\\\transactions.csv'
29   INTO TABLE transaction
30   FIELDS TERMINATED BY ';'
31   IGNORE 1 ROWS;

```

The 'Output' tab at the bottom shows the execution results:

#	Time	Action	Message
1	11:29:33	CREATE TABLE transaction (id VARCHAR(255) NULL, card_id VARCHAR(20) NULL, company_id ... 0 row(s) affected	0 row(s) affected
2	11:30:00	LOAD DATA INFILE 'C:\\transactions.csv' - C:\\Users\\evgen\\Downloads\\Barcelona Activa\\Data Analytics\\Sprint 4 SQL\\transactions.csv' INTO TABLE transaction FIELDS TERMINATED BY ';' IGNORE 1 ROWS;	100000 row(s) affected Records: 100000 Deleted: 0 Skipped: 0 Warnings: 0

Se ve, que la **transaction** tiene 100000 registros.

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'Schemas' section, there is a tree view with several schemas like 'hosptiales', 'nens_nenes', 'olympics', 'sys', 'transactions', and 'transactions_sp4'. The 'transactions_sp4' schema is expanded, showing 'Tables', 'Views', 'Stored Procedures', and 'Functions'. The main pane displays SQL code for selecting all rows from the 'transaction' table:

```

32
33 •  SELECT *
34   FROM transaction;
35
36

```

The 'Result Grid' tab at the bottom shows the data from the 'transaction' table:

	id	card_id	company_id	timestamp	amount	declined	product_ids	user_id	lat	longitude
1	CDDA7E40-54HD-47B8-A4ED-671DD8A950D9	Ccs-6894	b-2466	2018-12-12 08:05:17	161.88	0	75, 73, 98	2313	59.62050974356148	16.55997715
2	09456357-8E9B-475A-8257-87A023699964	Ccs-5135	b-2342	2024-05-20 22:49:08	171.13	0	3	554	45.76458841901318	4.843056518
3	C47C7C84-C174-4973-A76B-825A9DB5582A	Ccs-8415	b-2250	2018-11-04 23:16:18	497.29	0	92, 85, 36, 23	3834	52.06849608409912	4.301099382
4	2FB526AA-3844-4DDF-AC09-2EBCC255EE76	Ccs-6553	b-2610	2022-06-17 09:17:25	344.15	0	5, 27	1972	39.47598513179192	-0.37641944
5	00R77407-R85F-40D2-9229-6FD06F2RR0f0A	Ccs-7678	b-2454	2020-11-29 07:29:16	435.1	0	38, 32, 82, 62	3097	51.436569850132189	5.478525648

The 'Output' tab at the bottom shows the execution results:

#	Time	Action	Message
1	11:33:08	SELECT * FROM transaction	100000 row(s) returned

Paso 3. Creamos la tabla credit_cards e insertamos los datos del archivo credit_cards.csv.

Screenshot of MySQL Workbench showing the creation of the credit_cards table and the import of data from credit_cards.csv.

```

38 -- Creamos la tabla "credit_cards"
39 • CREATE TABLE IF NOT EXISTS credit_cards (
40     id          VARCHAR(20) NULL,
41     user_id    VARCHAR(20) NULL,
42     iban        VARCHAR(50) NULL,
43     pan         VARCHAR(50) NULL,
44     pin         VARCHAR(4)  NULL,
45     cvv         VARCHAR(4)  NULL,
46     track1      VARCHAR(255) NULL,
47     track2      VARCHAR(255) NULL,
48     expiring_date VARCHAR(20) NULL
49 );
50 -- Insertamos los datos del archivo credit_cards.csv en la tabla "credit_cards"
51 • LOAD DATA
52   INFILE 'C:\\credit_cards.csv'
53   INTO TABLE credit_cards
54   FIELDS TERMINATED BY ','
55   IGNORE 1 ROWS;

```

Output:

#	Time	Action	Message
1	11:37:16	CREATE TABLE IF NOT EXISTS credit_cards (id VARCHAR(20) NULL, user_id VARCHAR(20) N...)	0 row(s) affected
2	11:37:24	LOAD DATA INFILE C:\\credit_cards.csv INTO TABLE credit_cards FIELDS TERMINATED BY ',' IGNORE 1 R...	5000 row(s) affected Records: 5000 Deleted: 0 Skipped: 0 Warnings: 0

La tabla credit_cards tiene 5000 filas.

Screenshot of MySQL Workbench showing the creation of the companies table and the import of data from companies.csv.

```

56
57 • SELECT *
58   FROM credit_cards;
59
60

```

Result Grid:

id	user_id	iban	pan	pin	cvv	track1	track2	expiring_date
CdU-2938	275	TR30195031213576817638661	5424465566813633	3257	984	%B8383712448554646~WovxsxeDpwiev~...	%B7653863056044187=800716333673	10/30/22
CdU-2945	274	DO2685476374853747521656...	5142423821948828	9080	887	%B4621311609958661~UfufseGeman~0...	%B4149568437843501=51071403301	08/24/23
CdU-2952	273	BG45IVQL5210525608255	4556 453 55 5287	4980	438	%B2183285104307501^CddyytclJxwfdq...	%B6778380257827162=690685974007	06/29/21
CdU-2959	272	OR724247724433584153	372461377349375	3883	667	%B728111156795320*xcoddjBkecd~0...	%B424615489281853=280522391678	02/24/23

Output:

#	Time	Action	Message
1	11:40:18	SELECT * FROM credit_cards	5000 row(s) returned

Paso 4. Creamos la tabla companies e insertamos los datos del archivo companies.csv.

Screenshot of MySQL Workbench showing the creation of the companies table and the import of data from companies.csv.

```

61 ##### companies #####
62 -- Creamos la tabla companies
63 • CREATE TABLE IF NOT EXISTS companies (
64     company_id    VARCHAR(20) NULL,
65     company_name  VARCHAR(255) NULL,
66     phone         VARCHAR(15) NULL,
67     email         VARCHAR(100) NULL,
68     country       VARCHAR(100) NULL,
69     website       VARCHAR(255) NULL
70 );
71 -- Insertamos los datos del archivo companies.csv en la tabla "companies"
72 • LOAD DATA
73   INFILE 'C:\\companies.csv'
74   INTO TABLE companies
75   FIELDS TERMINATED BY ','
76   IGNORE 1 ROWS;

```

Output:

#	Time	Action	Message
1	11:51:38	CREATE TABLE IF NOT EXISTS companies (company_id VARCHAR(20) NULL, company_name VARCHA...	0 row(s) affected
2	11:51:47	LOAD DATA INFILE C:\\companies.csv INTO TABLE companies FIELDS TERMINATED BY ',' IGNORE 1 R...	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0

La tabla companies tiene 100 filas.

Screenshot of MySQL Workbench showing the creation of the companies table and the import of data from companies.csv.

```

77
78 • SELECT *
79   FROM companies;
80
81

```

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

Output:

#	Time	Action	Message
1	11:54:08	SELECT * FROM companies	100 row(s) returned

Paso 5. Creamos la tabla american_users e insertamos los datos del archivo american_users.csv.

```

82 ##### american_users #####
83 -- Creamos la tabla companies
84 CREATE TABLE IF NOT EXISTS american_users (
85     id      VARCHAR(20) NULL,
86     name    VARCHAR(100) NULL,
87     surname VARCHAR(100) NULL,
88     phone   VARCHAR(150) NULL,
89     email   VARCHAR(150) NULL,
90     birth_date VARCHAR(100) NULL,
91     country  VARCHAR(150) NULL,
92     city    VARCHAR(150) NULL,
93     postal_code VARCHAR(100) NULL,
94     address  VARCHAR(255) NULL
95 );
96 -- Insertamos los datos del archivo american_users.csv en la tabla "american_users"
97 LOAD DATA
98 INFILE 'C:\\american_users.csv'
99 INTO TABLE american_users
100 FIELDS TERMINATED BY ','
101 ENCLOSED BY ''
102 IGNORE 1 ROWS;

```

Output

#	Time	Action	Message
1	12:01:06	CREATE TABLE IF NOT EXISTS american_users (id VARCHAR(20) NULL, name VARCHAR(100) ... 0 rows) affected	
2	12:01:16	LOAD DATA INFILE C:\\american_users.csv INTO TABLE american_users FIELDS TERMINATED BY '' ENCL... 1010 row(s) affected Records: 1010 Deleted: 0 Skipped: 0 Warnings: 0	

La american_users tiene 1010 registros.

```

104
105 SELECT *
106 FROM american_users;
107

```

Result Grid

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

Output

#	Time	Action	Message
1	12:05:16	SELECT * FROM american_users	1010 row(s) returned

Paso 6. Creamos la tabla european_users e insertamos los datos del archivo european_users.csv.

```

109 ##### european_users #####
110 -- Creamos la tabla companies
111 CREATE TABLE IF NOT EXISTS european_users (
112     id      VARCHAR(20) NULL,
113     name    VARCHAR(100) NULL,
114     surname VARCHAR(100) NULL,
115     phone   VARCHAR(150) NULL,
116     email   VARCHAR(150) NULL,
117     birth_date VARCHAR(100) NULL,
118     country  VARCHAR(150) NULL,
119     city    VARCHAR(150) NULL,
120     postal_code VARCHAR(100) NULL,
121     address  VARCHAR(255) NULL
122 );
123 -- Insertamos los datos del archivo european_users.csv en la tabla "european_users"
124 LOAD DATA
125 INFILE 'C:\\\\european_users.csv'
126 INTO TABLE european_users
127 FIELDS TERMINATED BY ','
128 ENCLOSED BY ''
129 IGNORE 1 ROWS;

```

Output

#	Time	Action	Message
1	12:12:19	CREATE TABLE IF NOT EXISTS european_users (id VARCHAR(20) NULL, name VARCHAR(100) ... 0 rows) affected	
2	12:12:27	LOAD DATA INFILE C:\\\\european_users.csv INTO TABLE european_users FIELDS TERMINATED BY '' ENCL... 3990 row(s) affected Records: 3990 Deleted: 0 Skipped: 0 Warnings: 0	

La european_users tiene 3990 registros.

```

131
132 SELECT *
133 FROM european_users;
134
135

```

Result Grid

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	Ihaemeen	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 Consectetur St.
154	Cooper	Bullock	(021) 2521 6627	et@outlook.net	Nov 2, 1986	United Kingdom	Manchester	M1 1AE	872-1866 Pede Rd.
155	Joshua	Russell	055 409 5286	justo.nec.ante@outlook.edu	Jan 23, 1984	United Kingdom	Manchester	M1 1AE	Ap #285-4727 Auctor. Av.

Output

#	Time	Action	Message
1	12:16:07	SELECT * FROM european_users	3990 row(s) returned

Paso 7. Creamos la tabla `data_user` combinando la tabla `american_users` y la `european_users`.

Antes de hacerlo miramos las columnas de ambas tablas. Vemos que las tienen las mismas columnas.

`american_users`

The screenshot shows the MySQL Workbench interface with the 'Result Grid' tab selected. It displays the columns of the `american_users` table:

Field	Type	Null	Key	Default	Extra
id	varchar(20)	YES		NULL	
name	varchar(100)	YES		NULL	
surname	varchar(100)	YES		NULL	
phone	varchar(150)	YES		NULL	
email	varchar(150)	YES		NULL	
birth_date	varchar(100)	YES		NULL	
country	varchar(150)	YES		NULL	
city	varchar(150)	YES		NULL	
postal_code	varchar(100)	YES		NULL	
address	varchar(255)	YES		NULL	

Below the table structure, the 'Action Output' pane shows the result of the `SHOW COLUMNS FROM american_users;` query:

#	Time	Action
1	12:34:32	SHOW COLUMNS FROM american_users

Message: 10 row(s) returned

`european_users`

The screenshot shows the MySQL Workbench interface with the 'Result Grid' tab selected. It displays the columns of the `european_users` table:

Field	Type	Null	Key	Default	Extra
id	varchar(20)	YES		NULL	
name	varchar(100)	YES		NULL	
surname	varchar(100)	YES		NULL	
phone	varchar(150)	YES		NULL	
email	varchar(150)	YES		NULL	
birth_date	varchar(100)	YES		NULL	
country	varchar(150)	YES		NULL	
city	varchar(150)	YES		NULL	
postal_code	varchar(100)	YES		NULL	
address	varchar(255)	YES		NULL	

Below the table structure, the 'Action Output' pane shows the result of the `SHOW COLUMNS FROM european_users;` query:

#	Time	Action
1	13:32:41	SHOW COLUMNS FROM european_users

Message: 10 row(s) returned

Vamos a comprobar que la columna combinada `id` tendra los valores únicos.

The screenshot shows the MySQL Workbench interface with the SQL editor tab selected. A query is being run to verify that the combined `id` values from both tables are unique:

```
144 -- Ademas si combinamos las columnas "id" de ambas tablas, observamos que el valor "id" es unico y representa cada user concreto.
145 • SELECT id, COUNT(*) FROM (
146     SELECT id
147     FROM american_users
148     UNION
149     SELECT id
150     FROM european_users) t
151     GROUP BY id
152     HAVING COUNT(*) > 1;
153
```

The results are shown in the 'Result Grid' tab:

id	COUNT(*)

Below the results, the 'Action Output' pane shows the query and its execution message:

#	Time	Action
1	13:37:38	SELECT id, COUNT(*) FROM (SELECT id FROM american_users UNION SELECT id FROM european_users) ...

Message: 0 row(s) returned

Combinamos las creando la nueva tabla **data_user**.

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'schemas' tab, there is a tree view of database objects. Under the 'transactions_sp4' schema, the 'Tables' node is expanded, showing 'american_users', 'companies', 'credit_cards', 'data_user', 'european_users', and 'transaction'. The main pane displays the following SQL code:

```
154 -- Podemos combinar las dos tablas a la una data_user.
155 • CREATE TABLE data_user AS
156     SELECT *
157     FROM american_users
158     UNION
159     SELECT *
160     FROM european_users;
```

The output pane shows the execution results:

#	Time	Action
1	13:40:04	CREATE TABLE data_user AS SELECT * FROM american_users UNION SELECT * FROM european_users

Message: 5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0

Eliminamos las tablas **american_users** y **european_users**.

The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'schemas' tab, there is a tree view of database objects. Under the 'transactions_sp4' schema, the 'Tables' node is expanded, showing 'companies', 'credit_cards', 'data_user', and 'transaction'. The main pane displays the following SQL code:

```
162
163 • DROP TABLE american_users, european_users;
164 • SHOW TABLES FROM transactions_sp4;
165
166
```

The result grid shows the tables in the 'transactions_sp4' schema:

Tables_in_transactions_sp4
companies
credit_cards
data_user
transaction

The output pane shows the execution results:

#	Time	Action
1	13:50:12	DROP TABLE american_users, european_users
2	13:50:15	SHOW TABLES FROM transactions_sp4

Message: 0 row(s) affected
4 row(s) returned

Modificamos el tipo de la columna **id** de la tabla **data_user** a INT, para que se puede relacionar con la tabla **transaction**.

The screenshot shows the MySQL Workbench interface. In the left sidebar, there is a tree view of database objects. The main pane displays the following SQL code:

```
166
167 -- Los valores de "id" son numeros de 1 a 5000. Entonces modificamos el tipo de la variable a INT.
168 • ALTER TABLE data_user
169     MODIFY id INT NOT NULL;
170
```

The output pane shows the execution results:

#	Time	Action
1	13:57:56	ALTER TABLE data_user MODIFY id INT NOT NULL

Message: 5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0

Creamos el Primary Key de la tabla **data_user**.

The screenshot shows the MySQL Workbench interface. In the left sidebar, there is a tree view of database objects. The main pane displays the following SQL code:

```
172 -- y creamos el Primary KEY de la tabla data_user
173 • ALTER TABLE data_user
174     ADD PRIMARY KEY (id);
175
176 • SHOW COLUMNS FROM data_user;
```

The result grid shows the columns of the 'data_user' table:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	
name	varchar(100)	YES		NULL	
surname	varchar(100)	YES		NULL	
phone	varchar(150)	YES		NULL	
email	varchar(150)	YES		NULL	
birth_date	varchar(100)	YES		NULL	
country	varchar(150)	YES		NULL	
city	varchar(150)	YES		NULL	
postal_code	varchar(100)	YES		NULL	
address	varchar(255)	YES		NULL	

The output pane shows the execution results:

#	Time	Action
1	14:00:40	ALTER TABLE data_user ADD PRIMARY KEY (id)
2	14:00:44	SHOW COLUMNS FROM data_user

Message: 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
10 row(s) returned

Paso 8. Modificamos la tabla credit_cards.

Comprobamos que no hay valores repetidos en la columna id.

Sprint 4 Andrey Ismagilov* × SQL File 6* SQL File 7* Sprint 4 change secure file priv...
179 ##### credit_cards #####
180 -- Comprobamos que no hay valores repetidos en la columna "id"
181
182 • SELECT id, COUNT(*)
183 FROM credit_cards
184 GROUP BY id
185 HAVING COUNT(*) > 1;

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Result Grid
Output
Action Output
Time Action Message
1 11:09:51 SELECT id, COUNT(*) FROM credit_cards GROUP BY id HAVING COUNT(*) > 1 0 row(s) returned

Creamos el Primary Key de la tabla credit_cards.

Sprint 4 Andrey Ismagilov* × SQL File 6* SQL File 7* Sprint 4 change secure file priv...
187 -- Creamos el PRIMARY KEY de la tabla "credit_cards"
188 • ALTER TABLE credit_cards
189 MODIFY id VARCHAR(20) NOT NULL;
190 • ALTER TABLE credit_cards
191 ADD PRIMARY KEY(id);
192

Output
Action Output
Time Action Message
1 11:11:29 ALTER TABLE credit_cards MODIFY id VARCHAR(20) NOT NULL 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
2 11:11:34 ALTER TABLE credit_cards ADD PRIMARY KEY(id) 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

La tabla credit_cards modificada.

```
Sprint 4 Andrey Ismagilov * SQL File 6* SQL File 7* Sprint 4 change secure file priv...
192
193 -- La tabla "credit_card" modificada
194 • SHOW COLUMNS FROM credit_cards;
195
196
```

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

Field	Type	Null	Key	Default	Extra
id	varchar(20)	NO	PRI	NULL	
user_id	varchar(20)	YES		NULL	
iban	varchar(50)	YES		NULL	
pan	varchar(50)	YES		NULL	
pin	varchar(4)	YES		NULL	
cvv	varchar(4)	YES		NULL	
track1	varchar(255)	YES		NULL	
track2	varchar(255)	YES		NULL	
expiring_date	varchar(20)	YES		NULL	

Result 9 × Output: □ Action Output # Time Action Message
1 11:13:32 SHOW COLUMNS FROM credit_cards 9 row(s) returned

Paso 9. Modificamos la tabla companies.

Comprobamos que no hay valores repetidos en la columna id.

```
Sprint 4 Andrey Ismagilov * SQL File 6* SQL File 7* Sprint 4 change secure file priv...
197 ##### companies #####
198 -- Comprobamos que no hay valores repetidos en la columna "id"
199
200 • SELECT company_id, COUNT(*)
201   FROM companies
202   GROUP BY company_id
203   HAVING COUNT(*) > 1;
204
```

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

company_id	COUNT(*)

Output: □ Action Output # Time Action Message
1 11:18:28 SELECT company_id, COUNT(*) FROM companies GROUP BY company_id HAVING COUNT(*) > 1 0 row(s) returned

Renombramos la columna company_id a la id y creamos el PRIMARY KEY de la tabla companies.

```
Sprint 4 Andrey Ismagilov * SQL File 6* SQL File 7* Sprint 4 change secure file priv...
205 -- Renombramos la columna "company_id" a "id"
206 • ALTER TABLE companies
207   RENAME COLUMN company_id TO id;
208
209 -- Creamos el PRIMARY KEY de la tabla "companies"
210 --
211 • ALTER TABLE companies
212   MODIFY id VARCHAR(20) NOT NULL;
213 • ALTER TABLE companies
214   ADD PRIMARY KEY(id);
215
```

Output: □ Action Output # Time Action Message
1 11:21:53 ALTER TABLE companies RENAME COLUMN company_id TO id 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
2 11:22:03 ALTER TABLE companies MODIFY id VARCHAR(20) NOT NULL 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
3 11:22:07 ALTER TABLE companies ADD PRIMARY KEY(id); 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

La tabla companies modificada.

```
Sprint 4 Andrey Ismagilov * SQL File 6* SQL File 7* Sprint 4 change secure file priv...
215
216 -- La tabala "companies"
217 • SHOW COLUMNS FROM companies;
218
219
```

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

Field	Type	Null	Key	Default	Extra
id	varchar(20)	NO	PRI	NULL	
company_name	varchar(255)	YES		NULL	
phone	varchar(15)	YES		NULL	
email	varchar(100)	YES		NULL	
country	varchar(100)	YES		NULL	
website	varchar(255)	YES		NULL	

Result 11 × Output: □ Action Output # Time Action Message
1 11:23:35 SHOW COLUMNS FROM companies 6 row(s) returned

Paso 10. Modificamos la tabla **transaction**.

Comprobamos que no hay valores repetidos en la columna **id**.

```
217 • SHOW COLUMNS FROM transaction;
218
219 -- Comprobamos que no hay valores repetidos en la columna "id"
220 • SELECT id, COUNT(*)
221   FROM transaction
222   GROUP BY id
223   HAVING COUNT(*) > 1;
224
```

The screenshot shows the MySQL Workbench interface with the SQL editor tab open. The code above is pasted into the editor. Below the editor is a 'Result Grid' pane with a single row labeled 'id' and 'COUNT(*)'. A 'Result Grid' button is also visible in the top right corner of the pane.

Creamos el **Primary Key** de la tabla **transaction**.

```
225 -- Creamos el PRIMARY KEY de la tabla "transaction"
226 • ALTER TABLE transaction
227   MODIFY id VARCHAR(255) NOT NULL;
228 • ALTER TABLE transaction
229   ADD PRIMARY KEY(id);
230
```

The screenshot shows the MySQL Workbench interface with the SQL editor tab open. The code above is pasted into the editor. Below the editor is an 'Output' pane showing two successful actions: 'ALTER TABLE transaction MODIFY id VARCHAR(255) NOT NULL' and 'ALTER TABLE transaction ADD PRIMARY KEY(id)'. Each action has a timestamp, duration, and message indicating 0 rows affected, 0 records, 0 duplicates, and 0 warnings.

Renombramos la columna **card_id** y modificamos los tipos de varias columnas de la tabla **transaction**.

La **card_id** a la **credit_card_id**.

```
231 -- Renombramos la columna "card_id" a la "credit_card_id"
232 • ALTER TABLE transaction
233   RENAME COLUMN card_id TO credit_card_id;
234
```

The screenshot shows the MySQL Workbench interface with the SQL editor tab open. The code above is pasted into the editor. Below the editor is an 'Output' pane showing one successful action: 'ALTER TABLE transaction RENAME COLUMN card_id TO credit_card_id'. The message indicates 0 rows affected, 0 records, 0 duplicates, and 0 warnings.

Modificamos el tipo de la columna **user_id** a **INT** para que podamos vincularla con la columna **id** de la tabla **data_user** a continuación.

```
235 -- Modificamos el tipo de la columna "user_id" a INT para que podamos vincularla con la columna "id" de la tabla "data_user" a continuacion.
236 --
237 • ALTER TABLE transaction
238   MODIFY user_id INT;
239
```

The screenshot shows the MySQL Workbench interface with the SQL editor tab open. The code above is pasted into the editor. Below the editor is an 'Output' pane showing one successful action: 'ALTER TABLE transaction MODIFY user_id INT'. The message indicates 100000 rows affected, 100000 records, 0 duplicates, and 0 warnings.

Modificamos el tipo de la variable **amount** de **VARCHAR(255)** a **DECIMAL(10,2)**.

```
239 -- Modificamos el tipo de la variable "amount" de VARCHAR(255) a DECIMAL(10,2)
240 • ALTER TABLE transaction
241   MODIFY amount DECIMAL(10,2);
242
```

The screenshot shows the MySQL Workbench interface with the SQL editor tab open. The code above is pasted into the editor. Below the editor is an 'Output' pane showing one successful action: 'ALTER TABLE transaction MODIFY amount DECIMAL(10,2)'. The message indicates 100000 rows affected, 100000 records, 0 duplicates, and 0 warnings.

Modificamos los tipos varias variables:

- **lat** de **VARCHAR(255)** a **FLOAT**.
- **longitude** de **VARCHAR(255)** a **FLOAT**.
- **declined** de **VARCHAR(4)** a **TINYINT(1)**.
- **timestamp** de **VARCHAR(255)** a **TIMESTAMP**.

Sprint 4 Andrey Ismagilov × SQL File 6* SQL File 7* Sprint 4 change secure file priv...

```

244  -- Modificamos
245  -- el tipo de la variable "lat" de VARCHAR(255) a FLOAT,
246  -- el tipo de la variable "longitude" de VARCHAR(255) a FLOAT,
247  -- el tipo de la variable "declined" de VARCHAR(4) a TINYINT(1),
248  -- el tipo de la variable "timestamp" de VARCHAR(255) a TIMESTAMP.
249
250 • ALTER TABLE transaction
251   MODIFY lat FLOAT,
252   MODIFY longitude FLOAT,
253   MODIFY declined TINYINT,
254   MODIFY timestamp TIMESTAMP;
255

```

Output:

Action Output

#	Time	Action	Message
1	10:39:30	ALTER TABLE transaction MODIFY lat FLOAT, MODIFY longitude FLOAT, MODIFY declined TINYINT, MODIF...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Al final tenemos la tabla **transaction** preparada para hacer la esquema de estrella.

Sprint 4 Andrey Ismagilov × SQL File 6* SQL File 7* Sprint 4 change secure file priv...

```

256
257  -- Al final tenemos la tabla "transaction" preparada para hacer la esquema de estrella.
258 • SHOW COLUMNS FROM transaction;
259
260

```

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

Field	Type	Null	Key	Default	Extra
id	varchar(255)	NO	PRI	NULL	
credit_card_id	varchar(20)	YES		NULL	
company_id	varchar(20)	YES		NULL	
timestamp	timestamp	YES		NULL	
amount	decimal(10,2)	YES		NULL	
declined	tinyint	YES		NULL	
product_ids	varchar(255)	YES		NULL	
user_id	int	YES		NULL	
lat	float	YES		NULL	
longitude	float	YES		NULL	

Result 6 × Read Only

Output:

Action Output

#	Time	Action	Message
1	10:40:53	SHOW COLUMNS FROM transaction	10 row(s) returned

Paso 11. Creamos la esquema de estrella.

Vemos que la tabla "transaction" es la tabla de hechos del esquema de estrella.

- Contiene campos claves que se unen a las tablas de dimensiones: credit_card_id, company_id, user_id.
- Contiene métricas que podemos medir y analizar: amount, timestamp, declined, lat, longitude.
- Cada fila de la tabla representa la transaccion unica de la tarjeta credito.

Vinculamos la columna **id** de la tabla **data_user** con la **user_id** de la "transaction".

Sprint 4 Andrey Ismagilov × SQL File 6* SQL File 7* Sprint 4 change secure file priv...

```

261  ##### el esquema de estrella #####
262  -- Creamos la esquema de estrella.
263  -- Vemos que la tabla "transaction" es la tabla de hechos del esquema de estrella.
264  -- Contiene campos claves que se unen a las tablas de dimensiones: credit_card_id, company_id, user_id.
265  -- Contiene métricas que podemos media y analizar: amount, timestamp, declined, lat, longitude.
266  -- Cada fila de la tabla representa la transaccion unica de la tarjeta credito.
267
268  -- Vinculamos la columna "id" de la tabla "data_user" con la "user_id" de la "transaction"
269 • ALTER TABLE transaction
270   ADD CONSTRAINT fk_transaction_user_id
271   FOREIGN KEY (user_id)
272   REFERENCES data_user (id);
273

```

Output:

Action Output

#	Time	Action	Message
1	10:51:52	ALTER TABLE transaction ADD CONSTRAINT fk_transaction_user_id FOREIGN KEY (user_id) REFERENCES ...	100000 row(s) affected Records: 100000 Duplicates: 0 Warnings: 0

Vinculamos la columna **id** de la tabla **companies** con la **company_id** de la **transaction**.

```
Sprint 4 Andrey Ismagilov* SQL File 6* SQL File 7* Sprint 4 change secure file priv...
280 -- Vinculamos la columna "id" de la tabla "companies" con la "company_id" de la "transaction"
281 • ALTER TABLE transaction
282 ADD CONSTRAINT fk_transaction_company_id
283 FOREIGN KEY (company_id)
284 REFERENCES companies (id);

Output :::::
Action Output
# Time Action
1 11:44:05 ALTER TABLE transaction ADD CONSTRAINT fk_transaction_company_id FOREIGN KEY (company_id) REFE... 100000 row(s) affected Records: 100000 Duplicates: 0 Warnings: 0
```

Vinculamos la columna **id** de la tabla **credit_cards** con la **credit_card_id** de la **transaction**.

```
Sprint 4 Andrey Ismagilov* SQL File 6* SQL File 7* Sprint 4 change secure file priv...
286
287 -- Vinculamos la columna "id" de la tabla "credit_cards" con la "credit_card_id" de la "transaction"
288
289 • ALTER TABLE transaction
290 ADD CONSTRAINT fk_transaction_credit_card_id
291 FOREIGN KEY (credit_card_id)
292 REFERENCES credit_cards (id);
293

Output :::::
Action Output
# Time Action
1 11:48:02 ALTER TABLE transaction ADD CONSTRAINT fk_transaction_credit_card_id FOREIGN KEY (credit_card_id) R... 100000 row(s) affected Records: 100000 Duplicates: 0 Warnings: 0
```

Vamos a mirar los **FOREIGN KEYS** creados.

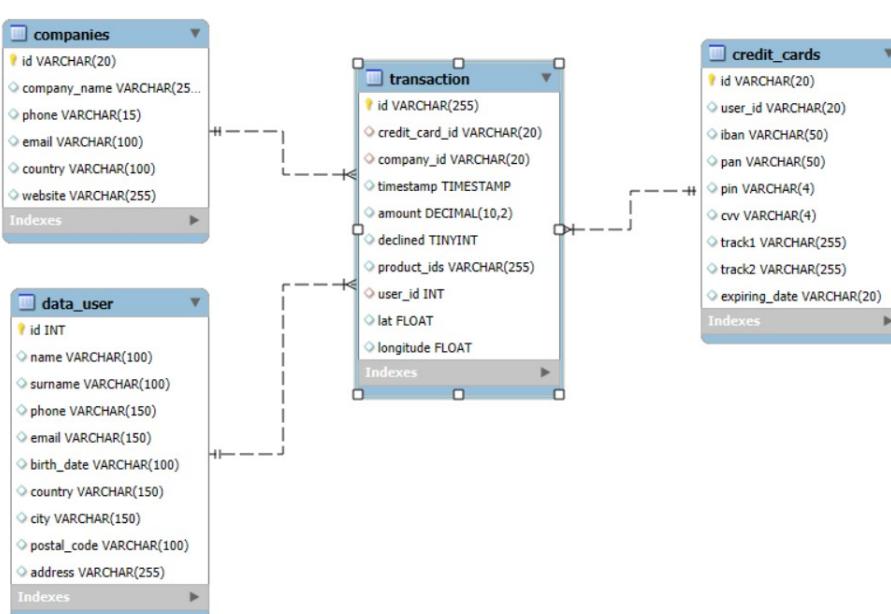
```
Sprint 4 Andrey Ismagilov* SQL File 6* SQL File 7* Sprint 4 change secure file priv...
292 -- Miramos los FOREIGN KEYS creados.
293 • SELECT
294     table_name,
295     column_name,
296     constraint_name,
297     referenced_table_name,
298     referenced_column_name
299     FROM
300     INFORMATION_SCHEMA.KEY_COLUMN_USAGE
301 WHERE
302     referenced_table_schema = 'transactions_sp4' ;
303

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


| TABLE_NAME  | COLUMN_NAME    | CONSTRAINT_NAME               | REFERENCED_TABLE_NAME | REFERENCED_COLUMN_NAME |
|-------------|----------------|-------------------------------|-----------------------|------------------------|
| transaction | company_id     | fk_transaction_company_id     | companies             | id                     |
| transaction | credit_card_id | fk_transaction_credit_card_id | credit_cards          | id                     |
| transaction | user_id        | fk_transaction_user_id        | data_user             | id                     |


Output :::::
Action Output
# Time Action
1 11:49:33 SELECT table_name, column_name, constraint_name, referenced_table_name, referenced_colu... 3 row(s) returned
```

Al final hemos creado la esquema estrella que se representa en el siguiente diagrama de la base de datos **transaction_sp4**.



Exercici 1

Realitza una subconsulta que mostri tots els usuaris amb més de 80 transaccions utilitzant almenys 2 taules.

The screenshot shows a SQL editor interface with several tabs at the top: Sprint 4 Andrey Ismagilov, credit_cards_declined_3_ult_tr..., SQL File 6*, SQL File 7*, Sprint 4 change secure file priv..., and transaction_products. The main area contains the following SQL code:

```
304  # Exercici 1
305  # Realitza una subconsulta que mostri tots els usuaris amb més de 80 transaccions utilitzant almenys 2 taules.
306 •  SELECT u.id, u.name, u.surname
307  FROM data_user u
308  WHERE EXISTS (
309      SELECT t.user_id, COUNT(t.user_id) AS num_transacciones
310      FROM transaction t
311      WHERE t.user_id = u.id
312      GROUP BY t.user_id
313      HAVING num_transacciones > 80
314  );
315
```

Below the code, there is a Result Grid showing the following data:

id	name	surname
185	Molly	Gillian
289	Dxwgi	Hwcru
318	Bnyr	Astuw
454	Sfzzoh	Xgvfridxs
*	NULL	NULL

The Output pane shows the following log entry:

#	Time	Action	Message
1	13:08:47	SELECT u.id, u.name, u.surname FROM data_user u WHERE EXISTS (SELECT t.user_id, COUNT(t.user_id) ...)	4 row(s) returned

Lo he hecho con un JOIN tambien tan solo para comparar los resultados.

The screenshot shows a SQL editor interface with several tabs at the top: Sprint 4 Andrey Ismagilov, SQL File 6*, SQL File 7*, Sprint 4 change secure file priv..., and transaction_products. The main area contains the following SQL code:

```
305  # Realitza una subconsulta que mostri tots els usuaris amb més de 80 transaccions utilitzant almenys 2 taules.
306 •  SELECT t.user_id, u.name, u.surname, count(*) AS num_transacciones
307  FROM transaction t
308  JOIN data_user u
309  WHERE u.id = t.user_id
310  GROUP BY t.user_id, u.name, u.surname
311  HAVING num_transacciones > 80
312  ORDER BY num_transacciones DESC;
```

Below the code, there is a Result Grid showing the following data:

user_id	name	surname	num_transacciones
185	Molly	Gillian	110
289	Dxwgi	Hwcru	94
318	Bnyr	Astuw	91
454	Sfzzoh	Xgvfridxs	81

The Output pane shows the following log entry:

#	Time	Action	Message
1	12:46:28	SELECT t.user_id, u.name, u.surname, count(*) AS num_transacciones FROM transaction t JOIN data_user u W...	4 row(s) returned

Exercici 2

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

The screenshot shows a SQL IDE interface with the following details:

Query Editor:

```
314 # Exercici 2
315 # Mostra la mitjana d'amount per IBAN de les targetes de crèdit a la companyia Donec Ltd, utilitza almenys 2 taules.
316 • SELECT c.company_name, cc.iban, ROUND(AVG(amount), 2) AS media_amount
317 FROM transaction t
318 JOIN credit_cards cc
319 ON cc.id = t.credit_card_id
320 JOIN companies c
321 ON c.id = t.company_id
322 WHERE c.company_name = "Donec Ltd" AND t.declined = 0
323 GROUP BY cc.iban
324 ORDER BY media_amount DESC;
```

Result Grid:

company_name	iban	media_amount
Donec Ltd	XX383017813919620199366352	680.69
Donec Ltd	XX637706357397570394973913	680.01
Donec Ltd	XX971393971465292202312259	645.46
Donec Ltd	XX171847116928892375969307	628.89
Donec Ltd	XX225424638818542406223575	608.68
Donec Ltd	XX748890729057195711766071	607.29
Donec Ltd	XX06145625706677291902177	605.11

Action Output:

#	Time	Action	Message
1	12:48:26	SELECT c.company_name, cc.iban, ROUND(AVG(amount), 2) AS media_amount FROM transaction t JOIN cred...	370 row(s) returned

Nivel 2

Crea una nova taula que reflecteixi l'estat de les targetes de crèdit basat en si les tres últimes transaccions han estat declinades aleshores és inactiu, si almenys una no és rebutjada aleshores és actiu.

Creamos la vista **credit_cards_declined_3_ult_trans** para guardar el resultado intermedio - las tarjetas con tres ultimas transacciones rechazadas

```
Sprint 4 Andrey Ismagilov  SQL File 6*  SQL File 7*  Sprint 4 change secure file priv...
[File] [New] [Open] [Save] [Save All] [Close] [Dont Limit] [Find] [Replace] [Help] [Copy] [Paste] [Print] [Exit]
326 ##### NIVEL 2 #####
327 # Crea una nova taula que reflecteixi l'estat de les targetes de crèdit basat en si les tres últimes transaccions han estat declinades
328 # aleshores és inactiu, si almenys una no és rebutjada aleshores és actiu.
329
330 -- Creamos la vista para guardar el resultado intermedio - las tarjetas con tres ultimas transacciones rechazadas
331 • CREATE VIEW credit_cards_declined_3_ult_trans AS
332     SELECT credit_card_id, SUM(declined) AS sum_declined_3_ult_trans
333     FROM (
334         SELECT id, credit_card_id, timestamp, declined,
335             ROW_NUMBER() OVER (PARTITION BY credit_card_id ORDER BY timestamp DESC) AS num_fila
336         FROM transaction
337     ) t
338     WHERE num_fila < 4
339     GROUP BY credit_card_id
340     HAVING sum_declined_3_ult_trans = 3;
```

Output:

Action Output	#	Time	Action	Message
1 12:53:15 CREATE VIEW credit_cards_declined_3_ult_trans AS SELECT credit_card_id, SUM(declined) AS sum_declined_... 0 row(s) affected				

La vista **credit_cards_declined_3_ult_trans**.

Navigator: credit_cards_declined_3_ult_trans

SCHEMAS: transactions_sp4

View: credit_cards_declined_3_ult_trans

Columns: credit_card_id, sum_declined_3_ult_trans

credit_card_id	sum_declined_3_ult_trans
CcS-4870	3
CcS-4899	3
CcS-4998	3
CcS-5035	3
CcU-3568	3

Output:

Action Output	#	Time	Action	Message
1 12:56:51 SELECT * FROM transactions_sp4.credit_cards_declined_3_ult_trans 5 row(s) returned				

Creamos la tabla **credit_card_status** que repersenta el estado de la tarjetas de crédito.

```
Sprint 4 Andrey Ismagilov  Sprint 4 change secure file priv...  credit_cards_declined_3_ult_trans  SQL File 6*  SQL File 7*  transaction_products
[File] [New] [Open] [Save] [Save All] [Close] [Dont Limit] [Find] [Replace] [Help] [Copy] [Paste] [Print] [Exit]
353 -- Creamos la tabla "credit_card_status" que repersenta el estado de la tarjetas de crédito
354 • CREATE TABLE IF NOT EXISTS credit_card_status AS
355     SELECT id AS cc_id,
356     CASE
357         WHEN EXISTS (
358             SELECT credit_card_id
359             FROM credit_cards_declined_3_ult_trans
360             WHERE id = credit_card_id)
361             THEN "Inactive"
362             ELSE "Active"
363         END AS status
364     FROM credit_cards;
365
```

Output:

Action Output	#	Time	Action	Message
1 21:50:49 CREATE TABLE IF NOT EXISTS credit_card_status AS SELECT id AS cc_id, CASE WHEN EXISTS (SELECT ... 5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0				

La tabla credit_card_status.

Sprint 4 Andrey Ismagilov x Sprint 4 change secure file priv... credit_cards_declined_3_ult_tr... SQL File 6* SQL File 7* transaction_products

```

368
369 •   SELECT *
370     FROM credit_card_status
371     ORDER BY status DESC;
372
373

```

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

cc_id	status
CcS-4998	Inactive
CcS-4870	Inactive
CcS-5035	Inactive
CcS-4899	Inactive
CcU-3568	Inactive
CcS-4858	Active
CcS-4859	Active

credit_card_status 89 x

Output:

Action Output

#	Time	Action	Message
1	21:52:34	SELECT * FROM credit_card_status ORDER BY status DESC	5000 row(s) returned

Vinculamos la columna **id** de la tabla **credit_cards** con la **cc_id** de la tabla **credit_card_status**.

Sprint 4 Andrey Ismagilov x Sprint 4 change secure file priv... credit_cards_declined_3_ult_tr... SQL File 6* SQL File 7* transaction_products credit_card_status

```

375 -- Vinculamos la columna "id" de la tabla "credit_cards" con la cc_id de la tabla "credit_card_status".
376 • ALTER TABLE credit_cards
377     ADD CONSTRAINT fk_credit_cards_id
378     FOREIGN KEY (id)
379     REFERENCES credit_card_status(cc_id);
380 -- Miramos los FOREIGN KEYS creados.
381 • SELECT
382     table_name,
383     column_name,
384     constraint_name,
385     referenced_table_name,
386     referenced_column_name
387     FROM
388     INFORMATION_SCHEMA.KEY_COLUMN_USAGE
389     WHERE
390     referenced_table_schema = 'transactions_sp4';

```

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

TABLE_NAME	COLUMN_NAME	CONSTRAINT_NAME	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME
credit_cards	id	fk_credit_cards_id	credit_card_status	cc_id
transaction	company_id	fk_transaction_company_id	companies	id
transaction	credit_card_id	fk_transaction_credit_card_id	credit_cards	id
transaction	user_id	fk_transaction_user_id	data_user	id

KEY_COLUMN_USAGE 76 x

Output:

Action Output

#	Time	Action	Message
1	19:09:05	ALTER TABLE credit_cards ADD CONSTRAINT fk_credit_cards_id FOREIGN KEY (id) REFERENCES credit_c...	5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0
2	19:09:08	SELECT table_name, column_name, constraint_name, referenced_table_name, referenced_colu...	4 row(s) returned

Exercici 1

Quantes targetes estan actives?

Sprint 4 Andrey Ismagilov x credit_cards_declined_3_ult_tr... SQL File 6* SQL File 7* Sprint 4 change secure file priv...

```

362 # Exercici 1
363 #Quantes targetes estan actives?
364 • SELECT COUNT(*) AS num_tarjetas_activas
365     FROM credit_card_status
366     WHERE status = "Active";
367

```

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

num_tarjetas_activas
4995

Result 25 x

Output:

Action Output

#	Time	Action	Message
1	13:06:29	SELECT COUNT(*) AS num_tarjetas_activas FROM credit_card_status WHERE status = "Active"	1 row(s) returned

Nivel 3

Crea una taula amb la qual puguem unir les dades del nou arxiu products.csv amb la base de dades creada, tenint en compte que des de transaction tens product_ids.

Creamos la tabla products y insertamos los datos del archivo products.csv en la tabla.

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the schema 'transactions_sp4' with a selected table 'products'. The main pane shows the SQL editor with the following code:

```
369 ##### NIVEL 3 #####
370 # Crea una taula amb la qual puguem unir les dades del nou arxiu products.csv amb la base de dades creada,
371 # tenint en compte que des de transaction tens product_ids.
372 • CREATE TABLE IF NOT EXISTS products (
373     id INT,
374     product_name VARCHAR(50),
375     price VARCHAR(255),
376     colour VARCHAR(20),
377     weight VARCHAR(20),
378     warehouse_id VARCHAR(20)
379 );
380
381 • LOAD DATA
382 INFILE 'C:\products.csv'
383 INTO TABLE products
384 FIELDS TERMINATED BY ','
385 IGNORE 1 ROWS;
```

The 'Output' pane shows the execution results:

#	Time	Action	Message
1	13:10:48	CREATE TABLE IF NOT EXISTS products (id INT, product_name VARCHAR(50), price VARCHAR(255), colour VARCHAR(20), weight VARCHAR(20), warehouse_id VARCHAR(20))	0 row(s) affected
2	13:12:37	LOAD DATA INFILE 'C:\products.csv' INTO TABLE products FIELDS TERMINATED BY ',' IGNORE 1 ROWS	100 row(s) affected Records: 100 Deleted: 0 Skipped: 0 Warnings: 0

La tabla products.

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
387 • SELECT *
388   FROM products;
```

The 'Result Grid' pane displays the data:

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

The 'Output' pane shows the execution results:

#	Time	Action	Message
1	13:15:04	SELECT * FROM products	100 row(s) returned

Creamos la tabla temporal trans_prod_temp con la columna products_ids_json que tendrá los valores en formato JSON para que podamos extraer los valores de product_ids con JSON_TABLE() función a continuacion.

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following code:

```
390 -- Creamos la tabla temporal "trans_prod_temp" con la columna "products_ids_json" que tendrá los valores en formato JSON
391 -- para que podamos extraer los valores de "product_ids" con JSON_TABLE() funcion a continuacion
392 • CREATE TEMPORARY TABLE IF NOT EXISTS trans_prod_temp AS
393   SELECT id, product_ids, CONCAT('[', product_ids, ']') AS products_ids_json
394   FROM transaction;
395
```

The 'Output' pane shows the execution results:

#	Time	Action	Message
1	13:17:24	CREATE TEMPORARY TABLE IF NOT EXISTS trans_prod_temp AS SELECT id, product_ids, CONCAT('[', prod...)	100000 row(s) affected Records: 100000 Duplicates: 0 Warnings: 0

La tabla temporal trans_prod_temp.

Sprint 4 Andrey Ismagilov credit_cards_declined_3_ult_tra... SQL File 6* SQL File 7* Sprint 4 change secure file priv...

```

397
398 • SELECT *
399   FROM trans_prod_temp;
400
401

```

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

	id	product_ids	products_ids_json
▶	00043A49-2949-494B-A5DD-A5BAE3BB19DD	16, 26, 97, 87	[16, 26, 97, 87]
	000447FE-B650-4DCF-85DE-C7ED0EE1CAAD	66, 69, 87	[66, 69, 87]
	00045D68-ED2E-4F2F-8186-CEE074D875D0	30, 11, 16, 81	[30, 11, 16, 81]
	000481C3-1C26-4FFB-83A0-4C00EB004BBD	72	[72]
	00051AA4-9CBE-4268-B070-C38062A1B3E2	18	[18]
	0008A312-EDFE-4A4F-BC99-E9C92EC3CA4D	35, 33, 19	[35, 33, 19]
	0009A151-98CF-4E31-9053-A468FF77FAAB	93, 55, 28, 91	[93, 55, 28, 91]

trans_prod_temp 28 ×

Output:

Action Output # Time Action Message

1 13:20:36 SELECT * FROM trans_prod_temp 100000 row(s) returned

Creamos la nueva tabla transaction_products con la columna transaction_id y la product_id.

Sprint 4 Andrey Ismagilov credit_cards_declined_3_ult_tra... SQL File 6* SQL File 7* Sprint 4 change secure file priv...

Navigator Schemas

transactions transactions_sp4

- Tables
 - companies
 - credit_card_status
 - credit_cards
 - data_user
 - products
 - transaction
 - transaction_products
- Views
- credit_cards_decline
- Stored Procedures

Administration Schemas Information

Table: transaction_products

Columns:

transaction_id	product_id
varchar(255)	int

transaction_products

```

402 -- Creamos la nueva tabla "transaction_products" con la columna "transaction_id" y la "product_id"
403 • CREATE TABLE IF NOT EXISTS transaction_products AS
404   SELECT id AS transaction_id, product_id from
405     trans_prod_temp,
406     JSON_TABLE (
407       products_ids_json,
408       '$[*]' COLUMNS(
409         product_id INT PATH '$'
410       )) AS t;
411

```

Output:

Action Output # Time Action Message

1 13:23:35 CREATE TABLE IF NOT EXISTS transaction_products AS SELECT id AS transaction_id, product_id from trans_p... 253391 row(s) affected Records: 253391 Duplicates: 0 Warnings: 0

La tabla transaction_products.

Sprint 4 Andrey Ismagilov credit_cards_declined_3_ult_tra... SQL File 6* SQL File 7* Sprint 4 change secure file priv...

```

414
415 • SELECT *
416   FROM transaction_products;
417
418

```

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

	transaction_id	product_id
▶	00043A49-2949-494B-A5DD-A5BAE3BB19DD	16
	00043A49-2949-494B-A5DD-A5BAE3BB19DD	26
	00043A49-2949-494B-A5DD-A5BAE3BB19DD	97
	00043A49-2949-494B-A5DD-A5BAE3BB19DD	87
	000447FE-B650-4DCF-85DE-C7ED0EE1CAAD	66
	000447FE-B650-4DCF-85DE-C7ED0EE1CAAD	69

transaction_products30 ×

Output:

Action Output # Time Action Message

1 13:27:57 SELECT * FROM transaction_products 253391 row(s) returned

Unimos la tabla **products** con las tablas creadas utilizando la tabla **transaction_products** como la tabla-puente.

Para ello vinculamos la columna **transaction_id** de la tabla **transaction_products** con la **id** de la **transaction** y luego vinculamos la columna "id" de la **products** con la columna **product_id** de la **transaction_products**.

```
Sprint 4 Andrey Ismagilov x Sprint 4 change secure file priv... credit_cards_declined_3_ult_tr... SQL File 6* SQL File 7* transaction_products credit_card_status
448 -- Vinculamos la columna "transaction_id" de la tabla "transaction_products" con la "id" de la tabla "transaction".
449 • ALTER TABLE transaction_products
450 ADD CONSTRAINT fk_transaction_products_transaction_id
451 FOREIGN KEY(transaction_id)
452 REFERENCES transaction(id);
453
454 -- Vinculamos la columna "id" de la tabla "products" con la columna "product_id" de la tabla "transaction_products".
455 • ALTER TABLE products
456 ADD PRIMARY KEY(id);
457
458 • ALTER TABLE transaction_products
459 ADD CONSTRAINT fk_transaction_products_product_id
460 FOREIGN KEY(product_id)
461 REFERENCES products(id);
462
```

Output:

Action Output		
#	Time	Action
1	21:00:56	ALTER TABLE transaction_products ADD CONSTRAINT fk_transaction_products_transaction_id FOREIGN KEY(... 253391 row(s) affected Records: 253391 Duplicates: 0 Warnings: 0
2	21:01:02	ALTER TABLE products ADD PRIMARY KEY(id) 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
3	21:01:09	ALTER TABLE transaction_products ADD CONSTRAINT fk_transaction_products_product_id FOREIGN KEY(pr... 253391 row(s) affected Records: 253391 Duplicates: 0 Warnings: 0

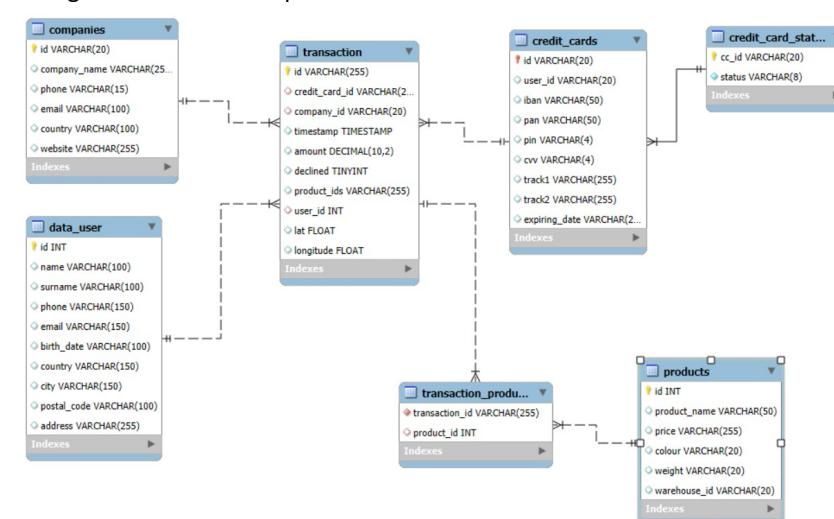
Miramos los **FOREIGN KEYS** creadas.

```
Sprint 4 Andrey Ismagilov* x Sprint 4 change secure file priv... credit_cards_declined_3_ult_tr... SQL File 6* SQL File 7* transaction_products credit_card_status
463 -- Miramos los FOREIGN KEYS creados.
464 • SELECT
465     table_name,
466     column_name,
467     constraint_name,
468     referenced_table_name,
469     referenced_column_name
470 FROM
471     INFORMATION_SCHEMA.KEY_COLUMN_USAGE
472 WHERE
473     referenced_table_schema = 'transactions_sp4' AND table_name = 'transaction_products';
474
```

Result Grid Filter Rows: Export: Wrap Cell Content: Message					
TABLE_NAME	COLUMN_NAME	CONSTRAINT_NAME	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME	
transaction_products	product_id	fk_transaction_products_product_id	products	id	
transaction_products	transaction_id	fk_transaction_products_transaction_id	transaction	id	

KEY_COLUMN_USAGE 84 x Read Only					
Output:					
Action Output					
1	21:05:00	SELECT	table_name, column_name, constraint_name, referenced_table_name, referenced_colu...	2 row(s) returned	

El diagrama final de la esquema entidad-relacion con todas las tablas creadas.



Exercici 1

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

The screenshot shows the Oracle SQL Developer interface. At the top, there are tabs for 'Sprint 4 Andrey Ismagilov', 'credit_cards_declined_3_ult_tra...', 'SQL File 6*', 'SQL File 7*', 'Sprint 4 change secure file priv...', and 'transaction_products'. Below the tabs is a toolbar with various icons. The main area contains the following SQL code:

```
430 # Exercici 1
431 # Necessitem conèixer el nombre de vegades que s'ha venut cada producte.
432 • SELECT tp.product_id, p.product_name, count(*) AS num_veces_vendido
433   FROM transaction_products tp
434   JOIN products p
435   ON tp.product_id = p.id
436   JOIN transaction t
437   ON t.id = tp.transaction_id
438   WHERE t.declined = 0
439   GROUP By tp.product_id, p.product_name
440   ORDER BY num_veces_vendido DESC;
441
```

Below the code is a 'Result Grid' table with the following data:

product_id	product_name	num_veces_vendido
52	riverlands the duel	2642
29	Tully maester Tarly	2627
21	duel Direwolf	2603
16	the duel warden	2602
33	duel warden	2593

At the bottom left, there is an 'Output' section showing the execution log:

Action Output	Message
1	14:06:16 SELECT tp.product_id, p.product_name, count(*) AS num_veces_vendido FROM transaction_products tp JOIN ... 100 row(s) returned