

SPRINT 4

Nivel 1

Creación de una nueva base de datos llamada “**R2TRANS**” con el comando **CREATE DATABASE** y asignando por default un set de caracteres que incluyen el uso de las tildes y la eñe: “*utf8mb4*”.

The screenshot shows the MySQL Workbench interface. In the SQL tab, two commands are run:

```

13 • CREATE DATABASE R2TRANS DEFAULT CHARACTER SET utf8mb4;
14 • show databases;

```

The results grid displays the list of databases, including the newly created 'R2TRANS' database. The status bar at the bottom indicates 10 row(s) returned and a duration of 0.0020 sec / 0.00001...

El siguiente paso para poder revisar los archivos .csv es importarlos a la base de datos.

Para este paso, en primer lugar hay que crear las tablas en las que más tarde se importarán los datos que necesitemos desde los archivos .csv. Creamos las tablas sin establecer *Primary Key* y que todos los campos acepten NULL puesto que no sabemos si los datos que vamos a importar están limpios y normalizados. Igualmente, estableceremos todos los campos en el formato VARCHAR para evitar problemas a la hora de importar los datos.

The screenshot shows the MySQL Workbench interface. In the SQL tab, a command is run to create the 'credit_cards' table:

```

11
12 -- Creación de las tablas donde importar los archivos .csv --
13 • CREATE TABLE IF NOT EXISTS credit_cards (
14     id varchar(15) NOT NULL,
15     user_id VARCHAR(10),
16     iban VARCHAR(50),
17     pan VARCHAR(30),
18     pin VARCHAR(4),
19     cvv VARCHAR(3),
20     track1 VARCHAR(255),
21     track2 VARCHAR(255),
22     expiring_date VARCHAR(15)
23 );

```

The results grid shows the table was created successfully with 0 row(s) affected. The status bar at the bottom indicates a duration of 0.014 sec.

En la imagen anterior se puede observar la creación de la tabla “*credit_cards*” y el mismo procedimiento se seguirá para el resto de tablas que vamos a importar. Una vez creada la tabla podemos optar a cargar los datos desde el archivo .csv con el comando **LOAD DATA INFILE ‘rutaarchivo’ INTO TABLE nombretabla** y con algunas especificaciones gracias a los siguientes parámetros:

- **FIELDS TERMINATED BY ',';** separación de campos por coma
- **ENCLOSED BY '\"';** los caracteres se pondrán entre comillas.
- **LINES TERMINATED BY '\n';** se identificará el cambio de línea con \n
- **IGNORE 1 ROWS;** que se utilizará para no importar la primera fila donde se especifica el nombre de los campos.

Una vez ejecutado estos comandos me dan un error. “*The MySQL server is running with the --secure-file-priv option so it cannot execute this statement*”

```

23    -- Importar los archivos .csv en la base de datos --
24 • LOAD DATA INFILE '/Users/rosi/Desktop/SQL_ITAcademy/SQL/Sprint_4/credit_cards.csv'
25   INTO TABLE credit_cards
26   FIELDS TERMINATED BY ','
27   ENCLOSED BY '\"'
28   LINES TERMINATED BY '\n'
29   IGNORE 1 ROWS;
30
100% 15:29 |
```

Action	Output	Time	Action	Response	Duration / Fetch Time
12	14:13:33	LOAD DATA INFILE '/Users/rosi/Desktop/SQL_ITAcademy/...		Error Code: 1290. The MySQL server is running with t...	0.0083 sec

Así que voy a la variable “`secure_file_priv`” para ver qué directorio está habilitado para poder importar los datos. Y me da el valor NULL, lo cuál indica que puedo importar archivos desde cualquier directorio. Sin embargo, sigo teniendo el mismo problema.

```

33 • show VARIABLES LIKE 'secure_file_priv';
100% 40:33 |
```

Variable_name	Value
secure_file_priv	NULL

Action	Output	Time	Action	Response	Duration / Fetch Time
17	14:26:44	show VARIABLES LIKE 'secure_file_priv'		1 row(s) returned	0.0038 sec / 0.00000...

Así que según la ayuda de ChatGPT, puedo cargar los datos de **LOCAL** utilizando el siguiente comando. **LOAD DATA LOCAL INFILE**. Esto también me da un error.

ERROR: Loading local data is disabled; this must be enabled on both the client and server sides. Así que intento habilitar la variable para importar data desde LOCAL.

Para solucionar este problema, se debe habilitar la opción de `local_infile` en el cliente. Lo cual hago con los siguientes comandos **SET GLOBAL local_infile = 1**, después de esto, pasa de estar OFF a estar **ON**, como puede verse en la imagen a continuación.

```

33 •  show VARIABLES LIKE 'secure_file_priv';
34 •  SET GLOBAL local_infile = 1;
35 •  show VARIABLES LIKE 'local_infile';

100% 29:34 | Result Grid Filter Rows: Search Export: Result Grid
Result Grid
Variable_name Value
local_infile ON
Result 8 Read Only
Action Output
Time Action Response Duration / Fetch Time
16 14:24:31 show VARIABLES LIKE 'local_infile' 1 row(s) returned 0.0058 sec / 0.00000...

```

Vuelvo a ejecutar el comando para cargar los archivos .csv y me da de nuevo un error en este caso el siguiente error:

Error Code: 2068. LOAD DATA LOCAL INFILE file request rejected due to restrictions on access.

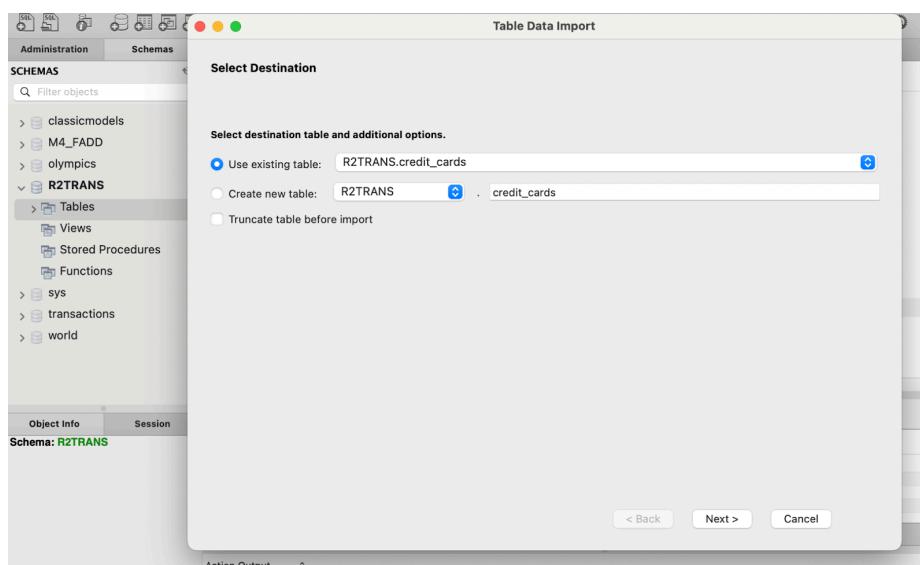
```

23      -- Importar los archivos .csv en la base de datos --
24 •  LOAD DATA LOCAL INFILE '/Users/rosi/Desktop/SQL_ITAcademy/SQL/Sprint_4/credit_cards.csv'
25    INTO TABLE credit_cards
26    FIELDS TERMINATED BY ','
27    ENCLOSED BY ""
28    LINES TERMINATED BY '\n'
29    IGNORE 1 ROWS;
30

100% 15:29 | Action Output
Action Output
Time Response Duration / Fetch Time
1 14:32:03 L Error Code: 2068. LOAD DATA LOCAL INFILE file request rejected due to restrictions on access. 0.00063 sec
/SQL_ITAcademy/SQL/Sprint_4/Sprint_4_Nivel_1_Rosi.sql

```

No tengo más ideas para conseguir solucionar el problema. Después de horas en ello, y de consultar a otras compañeras que también tienen Mac y que han tenido el mismo problema, sin encontrar solución, decidí importar los datos de los archivos .csv desde el “Table Data Import Wizard” de la interface de Workbench. Usando la tabla creada con anterioridad.



Compruebo que los datos se han cargado correctamente.

```
22 •   SELECT * FROM credit_cards;
23 •   show columns from credit_cards;
24
25   -- Importar los archivos .csv en la base de datos --
```

id	user_id	iban	pan	pin	cvv	track1	track2
CcU-2952	273	BG45VQL52710525608255	4556 453 55 5287	4598	438	%B2183285104307501^CddyytcUxwfdq^59079...	%B67785802
CcU-2959	272	CR724277244335841535	372461377349375	3583	667	%B728111956795320^XocddijBckecl^090162...	%B42461544
CcU-2966	271	BG72LKTQ15627628377363	448566 886747 7265	4900	130	%B4728932322756223^JhlgvsuFbmwgl^72022...	%B23185711
CcU-2980	269	PT87806228135092429456346	544 58654 54343 384	8760	887	%B4761405253275637^HjlnipoBlejl^7108515...	%B78161698
CcU-2980	269	DE39241881883086277136	402400 7145845969	5075	596	%B7320483593870549^OokzoxrHnased^49017...	%B24743139

Creación de la tabla “company” y carga de los datos a través del “Table Data Import Wizard” y comprobación de que se han cargado los datos correctamente. Se han cargado un total de **100 registros de compañías**.

```
22 •   CREATE TABLE IF NOT EXISTS company (
23     company_id VARCHAR(15) NOT NULL,
24     company_name VARCHAR(255),
25     phone VARCHAR(15),
26     email VARCHAR(100),
27     country VARCHAR(100),
28     website VARCHAR(255)
29 );
30 •   show tables;
31 •   SELECT * FROM company;
32 •   show columns from credit_cards;
```

company_id	company_name	phone	email	country	website
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@ad.co.uk	Germany	https://cnn.com/user/110
b-2238	Ante facilis Nec Foundation	08 23 04 99 53	sed.dictum.proin@outlook.ca	New Zealand	https://netflix.com/settings
b-2242	Donec Ltd	01 25 51 37 37	at.iaculis@hotmail.co.uk	Norway	https://nytimes.com/user/110
b-2246	Sed.Nunc.ltd	02 62 64 73 48	nibh@yahoo.or...	United Kingdom	https://cnn.com/one

Creación de la tabla “TRANSACTION”:

```
31 •   CREATE TABLE IF NOT EXISTS transaction (
32     id VARCHAR(255) NOT NULL,
33     card_id VARCHAR(15),
34     business_id VARCHAR(20),
35     timestamp VARCHAR(50),
36     amount VARCHAR(20),
37     declined VARCHAR(2),
38     product_ids VARCHAR(255),
39     user_id VARCHAR(20),
40     lat VARCHAR(30),
41     longitude VARCHAR(30)
42 );
43 •   show tables;
44 •   SELECT * FROM transaction;
```

id	card_id	business_id	timestamp	amount	declined	product_i...	user_id	lat	longitude
----	---------	-------------	-----------	--------	----------	--------------	---------	-----	-----------

Ahora podemos comprobar que se han cargado correctamente **587 registros** de transacciones en la tabla.

44 • SELECT * FROM transaction;

Result Grid									
	id	card_id	business_id	timestamp	amount	declined	product_ids	user_id	lat
72997E96-DC2C-A4D7-7C24-66C302F8AE5A	CcU-2952	b-2230	2022-01-30 15:16:36	239.87	0	97, 41, 3	275	43.3584055296	-1
AB069F53-965E-A2A8-C06-C48CFD92501	CcU-2959	b-2234	2021-04-15 13:37:18	60.99	0	11, 13, 61, 29	275	1.6481916928	-1
2F3B6ABA-147D-EB0B-FE8D-9A4E2EA9DBD5	CcU-2966	b-2238	2021-10-18 06:12:03	33.81	0	47, 37, 11, 1	275	-43.4811227136	16
5B0EEFB86-B8A1-EFAA-5EE1-27E7DC8F54A4	CcU-2973	b-2242	2022-01-06 01:44:48	42.82	0	23, 19, 71	275	-64.1136375808	85
2R92RF1C-FC14-A760-0A75-871477649D6A	CcU-2980	b-2246	2021-08-10 08:14:49	383.73	0	59, 13, 23	275	-41.049559552	16

Creación de la tabla “*USERS*”

```
45 • - CREATE TABLE IF NOT EXISTS users (
46             id VARCHAR(255) NOT NULL,
47             name VARCHAR(20),
48             surname VARCHAR(20),
49             phone VARCHAR(20),
50             email VARCHAR(50),
51             birth_date VARCHAR(20),
52             country VARCHAR(50),
53             city VARCHAR(50),
54             postal_code VARCHAR(10),
55             address VARCHAR(255)
56         );
57 • show tables;
58 • SELECT * FROM users;
```

Importación de los datos del archivo `users_usa.csv` y comprobación de la carga de **150 registros**.

58 • SELECT * FROM users;

100% 21:58

Result Grid Filter Rows: Search Export:

id	name	surname	phone	email	birth_date	country	city	postal_code	address
1	Carroll	Wiesner	(407) 257-2712	ingenieria.kademica@protonmail.org	Aug 20, 1982	United States	Beechwood	55117	333 S. 1st Ave.
3	Ciaran	Harrison	(522) 598-1365	interdum.feugiat@aol.org	Apr 29, 1998	United States	Columbus	56518	736-2063 Tellus St.
4	Howard	Stafford	1-411-740-3269	ornare.egestas@icloud.edu	Feb 18, 1989	United States	Kailua	77417	Ap #545-2244 Erat.
5	Hayfa	Pierce	1-554-541-2077	et.malesuada.fames@hotmail.org	Sep 26, 1998	United States	Sandy	31564	341-2821 Ultrices A.
6	Joel	Tyson	(718) 288-8020	gravida.nunc.sed@yahoo.ca	Oct 15, 1989	United States	Nashville	96838	888-2799 Amet Str.
7	Rafael	Jimenez	(817) 689-0478	enet@outlook.ca	Dec 4, 1981	United States	Hillsboro	29874	8627.Malesuada.B.

users 11 Read Only

Action Output

Time Response Duration / Fetch Time

46 15:47:53 S 150 row(s) returned 0.00082 sec / 0.0000...

Al realizar un **select * from users**, podemos ver que nos devuelve **200 filas**, lo que indica que se han cargado los 50 registros del archivo *users_uk.csv* correctamente.

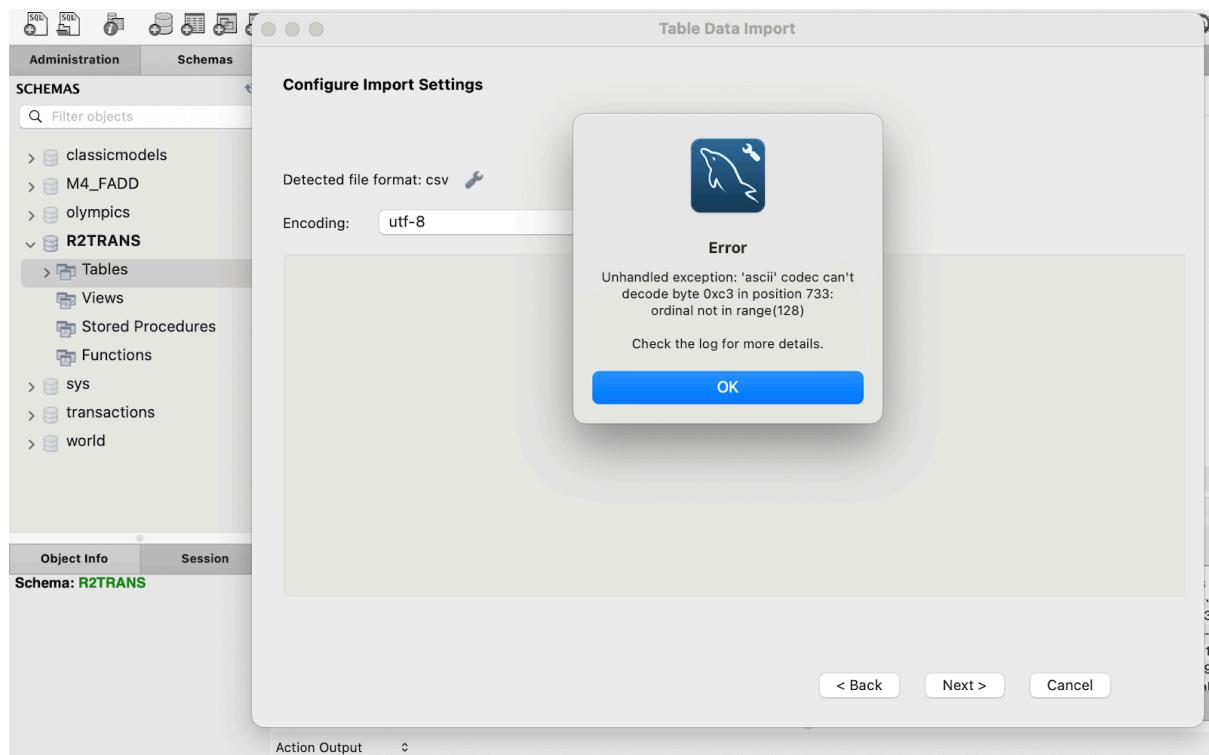
The screenshot shows the MySQL Workbench interface with a query editor window. The query is:

```
58 • SELECT * FROM users;
```

The results grid displays 200 rows of data from the 'users' table. The columns are: id, name, surname, phone, email, birth_date, country, city, postal_code, and address. The data includes various names like Ciaran, Howard, Hayfa, Joel, and Rafael, along with their respective details such as phone numbers and email addresses.

At the bottom of the interface, there is an 'Action Output' section showing the time (15:49:48) and the number of rows returned (200). The duration of the fetch is listed as 0.00087 sec / 0.0002...

Sin embargo con el archivo de *users_ca.csv* me da un error al intentar importarlo



He abierto el archivo con la aplicación *Sublime Text* y he guardado el archivo con el **encoding utf-8**, pero sigue apareciendo el mismo error.

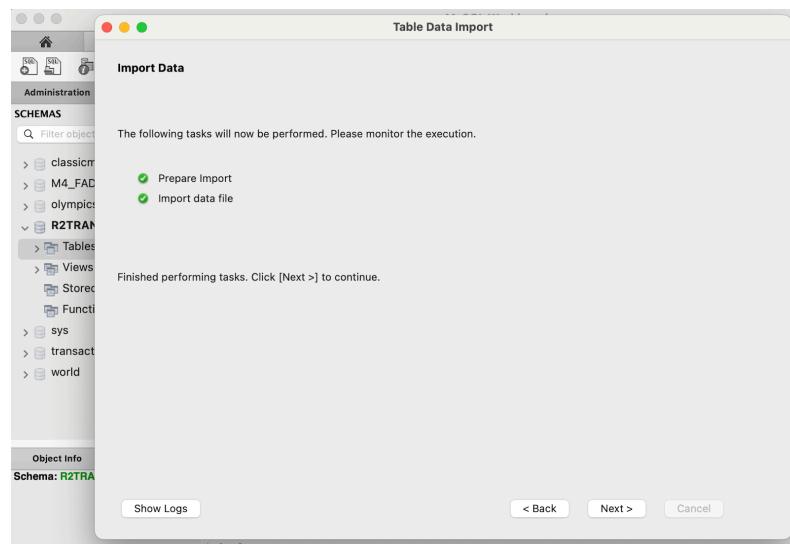
También agregué algunos comandos a mi configuración de MySQL, que es la que se muestra a continuación pero sin éxito.

```
ini
[mysqld]
character-set-server=utf8mb4
collation-server=utf8mb4_unicode_ci
```

Finalmente, gracias a la ayuda de una compañera, supe que había un carácter dentro del texto que NO pertenecía al formato utf-8. Era una é, como se puede observar en la imagen a continuación.

```
couk,"Jan 6, 1982",Canada,Baddeck,K3X 6Z5,441-8969 Rhoncus Road
look.ca,"Oct 15, 1994",Canada,Maple Creek,Y2C 9E6,"517-6759 Ut,
lor@google.ca,"May 7, 1983",Canada.Dieppe,E7S 4P8,Ap #916-8051 A
tlook.net,"Nov 17, 2000",Canada,Cuéb c City,B4K 0J6,8588 Massa.
@google.edu,"Sep 22, 1981",Canada,Kae-Edzo,20Y 8L2,Ap #636-8055
b 23, 1993",Canada,Annapolis Royal,S4Y 8V5,Ap #983-6042 Amet Str
cidunt@yahoo.ca,"Dec 14, 1991",Canada,Cambridge Bay,93Z 5S5,Ap #
Feb 16, 2001",Canada,Ottawa,A1S 9W6,601-6142 Etiam St.
```

Una vez reemplazado este carácter por otro que si acepta utf-8, fue posible cargar los datos del archivo user_ca.csv.



Al hacer un ***SELECT * FROM users***, se puede observar en los registros que ya aparecen usuarios en los que el país es Canada y, además, el total de usuarios ha subido hasta **275 en total**. Anteriormente, había un total de 200 registros de usuarios.

73 • ***SELECT * FROM users;***

100% 21:73

Result Grid										
Edit: Export/Import:										
	<i>id</i>	<i>name</i>	<i>surname</i>	<i>phone</i>	<i>email</i>	<i>birth_date</i>	<i>country</i>	<i>city</i>	<i>postal_code</i>	<i>address</i>
	210	Slade	Poole	084-771-1363	amet@icloud.com	Feb 16, 2001	Canada	Ottawa	A1S 9W6	601-6142 Etiam St.
	211	Larissa	Carpenter	066-617-3711	congue@aol.org	Jan 21, 1998	Canada	Cumberla...	S5Y 2L8	7285 Sed St.
	212	Zeph	Schmidt	056-157-7412	vestibulum.lorem@hotmail.ca	Apr 10, 1986	Canada	Fort Smith	V3G 8B3	4756 Tempor Rd.
	213	Blake	Strickland	067-339-3024	ac.libero.nec@yahoo.com	Apr 15, 1983	Canada	Mission	R0V 9R2	P.O. Box 207, 6843 Imp...

Action Output

	Time	Response	Duration / Fetch T
21	11:56:23	S OK	0.000 sec
22	11:56:26	P OK	0.000 sec
23	11:56:26	D OK	0.000 sec
24	11:57:44	S 275 row(s) returned	0.0013 sec / 0.000

Por el momento hemos creado las cuatro tablas que necesitaremos para realizar los ejercicios 1 y 2 de este nivel que son: “company”, “users”, “credit_cards” y “transaction”.

```

57 •    ''
58 •  show tables;
59 13:57
Result Grid Filter Rows: Search Export: 
Tables_in_r2tra...
company
credit_cards
transaction
users
Result 16
Action Output
Time Action Response Duration / Fetch Time
20 16:25:47 show tables 4 row(s) returned 0.0020 sec / 0.0000...
Read Only

```

Para poder hacer el diseño del modelo de la base de datos, se necesita estudiar las tablas implicadas y lo haré con algunas consultas en SQL para determinar por qué campos se relacionan las tablas.

```

59 •  SELECT * FROM users;
60 •  SELECT * FROM transaction;
61 •  SELECT * FROM credit_cards;
62 •  SELECT * FROM company;
63 •  SELECT count(id) FROM credit_cards;
64 •  show columns from credit_cards;
65 23:62
Result Grid Filter Rows: Search Export: 
company_id company_name phone email country website
a-2220 Magna Ataque Industries 01 77 77 37 32 nesciendocumbre@outlook.org Australia https://magnatakeapp.com.ng/capo
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
b-2242 Donec Ltd 01 25 51 37 37 at.iaculis@hotmail.co.uk Norway https://nytimes.com/user/110
company 17
Action Output
Time Action Response Duration / Fetch Time
21 16:30:09 SELECT * FROM company 100 row(s) returned 0.00091 sec / 0.0000...
Read Only

```

Este estudio nos muestra que tenemos un **modelo en ESTRELLA**, en el que tenemos una tabla de hechos o métricas que es la tabla de “TRANSACTION”, en esta tabla encontramos los campos que podremos usar como métricas y las Foreign Key, es decir, los campos que nos ayudan a relacionar esta tabla con el resto de tablas que son las tablas de dimensiones (“COMPANY”, “CREDIT_CARDS” y “COMPANY”). Como es habitual en los modelos de estrella, tenemos una relación de MUCHOS (N) en la tabla de hechos con una relación de UNO (1) en las tablas de dimensiones.

Ahora toca crear estas relaciones entre las tablas para poder crear las consultas que se nos piden.

- **company 1 : N transaction**

Primero, creamos la PK en la tabla “company”

```

80      -- Creación de relaciones entre tablas --
81      # company 1:N transaction
82 •  ALTER TABLE company
83      ADD CONSTRAINT company_id Primary key(company_id);
84 •  show columns from company;
85

```

Result Grid

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

Action Output

Time	Action	Response	Duration / Fetch Time
24 17:48:32	show columns from company	6 row(s) returned	0.0028 sec / 0.00003...

Después, creamos la FK en la tabla “transaction” relacionada con el campo **company_id** de la tabla “company”.

```

86      -- creación de Clave foránea en tabla transaction --
87 •  ALTER TABLE transaction
88      ADD CONSTRAINT business_id FOREIGN KEY (business_id)REFERENCES company(company_id);
89 •  show columns from transaction;
90

```

Result Grid

Field	Type	Null	Key	Default	Extra
id	varchar(255)	NO		NULL	
card_id	varchar(15)	YES		NULL	
business_id	varchar(20)	YES	MUL	NULL	
timestamp	varchar(50)	YES		NULL	
amount	varchar(20)	YES		NULL	

Action Output

Time	Action	Response	Duration / Fetch Time
26 17:54:15	show columns from transaction	10 row(s) returned	0.0017 sec / 0.00001...

Antes de continuar con las siguientes relaciones, creo la Clave Primaria de la tabla “transaction”.

```

91 •  ALTER TABLE transaction
92      ADD CONSTRAINT id Primary key(id);
93 •  show columns from transaction;

```

Result Grid

Field	Type	Null	Key	Default	Extra
id	varchar(255)	NO	PRI	NULL	
card_id	varchar(15)	YES		NULL	
business_id	varchar(20)	YES	MUL	NULL	
timestamp	varchar(50)	YES		NULL	
amount	varchar(20)	YES		NULL	

Action Output

Time	Action	Response	Duration / Feti
28 17:57:04	show columns from transaction	10 row(s) returned	0.0025 sec / 0

- users 1 : N transaction

Creación de la PK en la tabla “users” en el campo ***id***.

```
95      # users 1:N transaction
96      -- Creación PK en la tabla users --
97 •  ALTER TABLE users
98      ADD CONSTRAINT id Primary key(id);
99 •  show columns from users;
```

The screenshot shows the MySQL Workbench interface with the SQL editor tab open. The query window contains the code for creating a primary key on the 'id' column of the 'users' table. Below the editor is a 'Result Grid' pane displaying the table structure with the new primary key constraint applied. The status bar at the bottom indicates the operation was completed successfully.

Field	Type	Null	Key	Default	Extra
id	varchar(255)	NO	PRI	NULL	
name	varchar(20)	YES		NULL	
surname	varchar(20)	YES		NULL	
phone	varchar(20)	YES		NULL	
email	varchar(50)	YES		NULL	

Creación de la FK en la tabla “transaction” en el campo ***user_id*** que en este caso está relacionada con el campo ***id*** de la tabla “users”.

```
102 •  ALTER TABLE transaction
103      ADD CONSTRAINT user_id FOREIGN KEY (user_id) REFERENCES users(id);
104 •  show columns from transaction;
```

The screenshot shows the MySQL Workbench interface with the SQL editor tab open. The query window contains the code for creating a foreign key constraint on the 'user_id' column of the 'transaction' table, referencing the 'id' column of the 'users' table. Below the editor is a 'Result Grid' pane displaying the table structure with the new foreign key constraint applied. The status bar at the bottom indicates the operation was completed successfully.

Field	Type	Null	Key	Default	Extra
declined	varchar(2)	YES		NULL	
product_ids	varchar(255)	YES		NULL	
user_id	varchar(20)	YES	MUL	NULL	
lat	varchar(30)	YES		NULL	
longitud	varchar(30)	YES		NULL	

- credit_cards 1 : N transaction

```
107      # credit_cards 1:N transaction
108      -- creación de Clave primaria en tabla credit_cards --
109 •  ALTER TABLE credit_cards
110      ADD CONSTRAINT id Primary key(id);
111 •  show columns from credit_cards;
112      -- creación de Clave foránea en tabla transaction --
```

The screenshot shows the MySQL Workbench interface with the SQL editor tab open. The query window contains the code for creating a primary key on the 'id' column of the 'credit_cards' table and a foreign key constraint on the 'user_id' column referencing the 'id' column of the 'users' table. Below the editor is a 'Result Grid' pane displaying the table structure with both constraints applied. The status bar at the bottom indicates the operation was completed successfully.

Field	Type	Null	Key	Default	Extra
id	varchar(15)	NO	PRI	NULL	
user_id	varchar(10)	YES		NULL	
iban	varchar(50)	YES		NULL	
pan	varchar(30)	YES		NULL	
pin	varchar(4)	YES		NULL	

```

112      -- creación de Clave foránea en tabla transaction --
113 • ALTER TABLE transaction
114     ADD CONSTRAINT card_id FOREIGN KEY (card_id)REFERENCES credit_cards(id);
115 • show columns from transaction;

```

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

- Result Grid:** Shows the columns of the `transaction` table. The columns are: id (varchar(255), NO, PRI, NULL), card_id (varchar(15), YES, MUL, NULL), business_id (varchar(20), YES, MUL, NULL), timestamp (varchar(50), YES, NULL, NULL), and amount (varchar(20), YES, NULL, NULL). A total of 25 rows are shown.
- Action Output:** Displays the execution results of the SQL statements. It shows 38 rows affected at 15:08:21, with a duration of 0.0034 seconds.

• users 1 : 1 credit_cards

Ahora vamos a crear una relación 1:1 entre las tablas “`users`” y “`credit_cards`”. Esto significa que se relacionan por campos en los cuales hay la misma cantidad de registros, no puede haber registros con las claves que las relacionan repetidas. En este caso, estas dos tablas se relacionan por el campo “`id`” de la tabla “`users`” con el campo “`user_id`” de la tabla “`credit_cards`”.

En este clase estableceremos como FK el campo “`user_id`” de la tabla “`credit_cards`”, pero antes de establecer esta relación, especificaremos que este campo debe ser UNIQUE, es decir no acepta valores repetidos.

```

117      # users 1:1 credit_cards
118      -- Creación de la restricción UNIQUE a la que será la FK de la tabla credit_cards --
119 • ALTER TABLE credit_cards
120     ADD CONSTRAINT user_id UNIQUE(user_id);
121 • describe credit_cards;

```

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

- Result Grid:** Shows the columns of the `credit_cards` table. The columns are: id (varchar(15), NO, PRI, NULL), user_id (varchar(10), YES, UNI, NULL), and iban (varchar(50), YES, NULL, NULL). A total of 34 rows are shown.
- Action Output:** Displays the execution results of the SQL statements. It shows 50 rows affected at 16:15:37, with a duration of 0.0026 seconds.

Y añado la clave foránea en la tabla “`credit_cards`” en el campo “`user_id`”.

```

121      -- creación de Clave foránea en tabla credit_cards --
122 • ALTER TABLE credit_cards
123     ADD CONSTRAINT fk_user_id FOREIGN KEY (user_id)REFERENCES users(id);

```

A continuación, se adjunta el **Diagrama Entidad Relación** para la base de datos “R2TRANS”, que muestra el modelo en estrella explicado anteriormente.

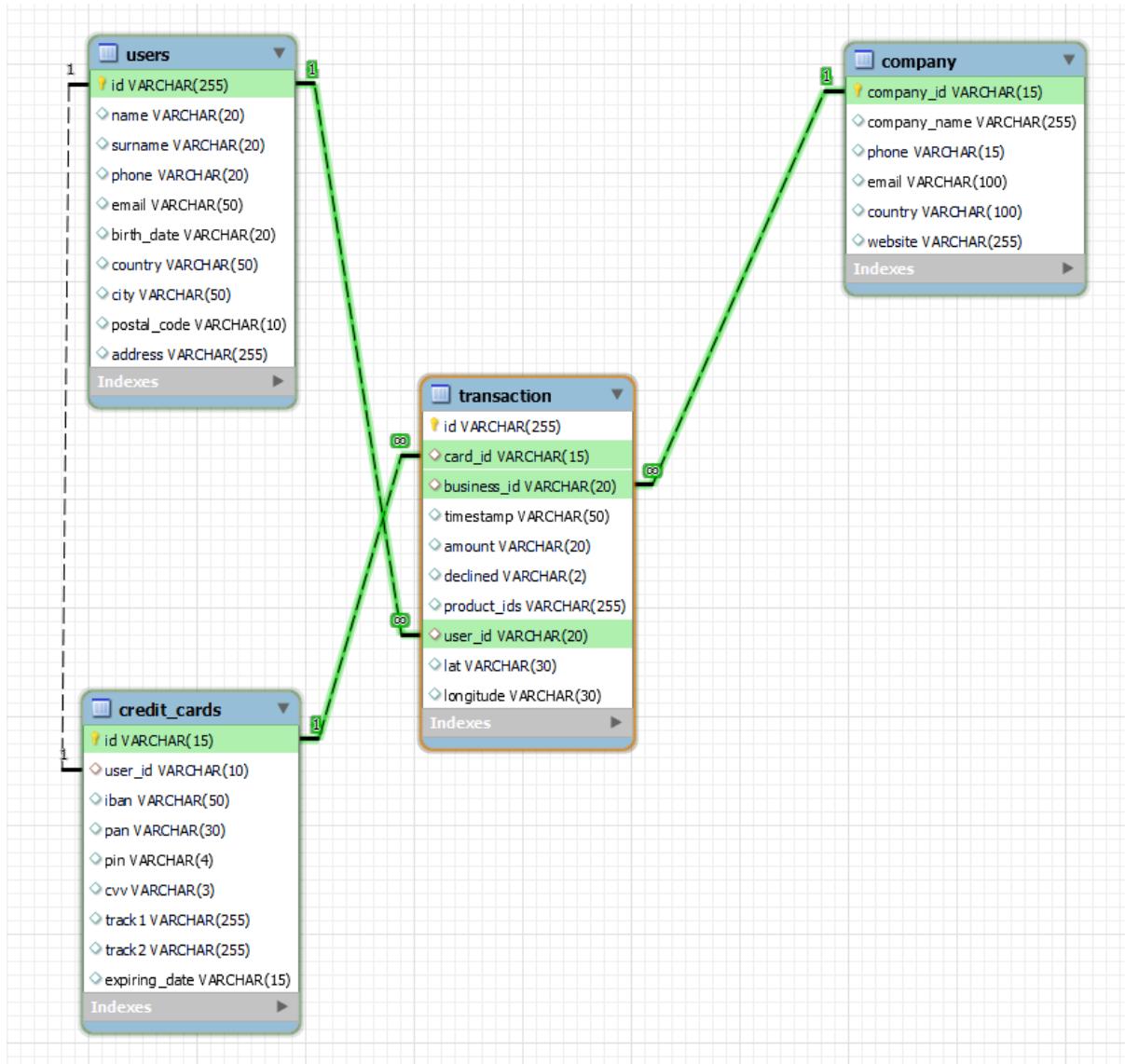


Fig. 1: Diagrama Entidad Relación

Ejercicio 1: Usuarios con más de 30 transacciones.

Para resolver este ejercicio, una vez creada todas las relaciones entre las tablas, necesitamos la tabla “users” y la tabla “transaction”. Gracias a un **INNER JOIN** podremos hacer el recuento de transacciones realizadas por cada usuario (utilizando un **GROUP BY** por identificador de usuario) y con un **HAVING** seleccionamos los usuarios que cumplen la condición de tener más de 30 transacciones. Nos muestra un **total de 4 usuarios** que tienen más de 30 transacciones.

Se puede hacer con una SUBCONSULTA:

```

126  # Ejercicio 1
127  -- Muestra los usuarios que hayan realizado más de 30 transacciones --
128  -- con SUBCONSULTA --
129 •  SELECT users.name
130  FROM users
131  WHERE (SELECT COUNT(transaction.id)
132      FROM transaction
133      WHERE transaction.user_id = users.id) > 30
134  GROUP BY users.name;
135

```

Result Grid

name
Ocean
Hedwig
Kenyon
Lynn

Action Output

Time	Action	Response	Duration / Fetch Time
10 10:22:07	SELECT users.name FROM users WHERE (SELECT COUN...	4 row(s) returned	0.0028 sec / 0.00001...

Y también es posible hacer la consulta con una JOIN:

```

126  # Ejercicio 1
127  -- Muestra los usuarios que hayan realizado más de 30 transacciones --
128 •  SELECT users.name, count(transaction.id) AS NumTransacciones
129  FROM users
130  JOIN transaction
131  ON users.id = transaction.user_id
132  GROUP BY users.id
133  HAVING NumTransacciones >30
134  ORDER BY NumTransacciones DESC;
135

```

Result Grid

name	NumTransacciones
Hedwig	76
Ocean	52
Kenyon	48
Lynn	39

Action Output

Time	Action	Response	Duration / Fetch Time
37 13:17:59	S 4 row(s) returned		0.0081 sec / 0.00001...
38 13:19:18	S 4 row(s) returned		0.0040 sec / 0.00001...

Ejercicio 2: Cantidad media de compra de la compañía DONEC Ltd.

Para resolver este ejercicio, agrupamos los registros por *iban* y *nombre de la compañía*. Podemos hacer la agrupación de estos dos campos, puesto que para cada compañía, tenemos un único iban. Como solo necesitamos los datos de la compañía “Donec Ltd” incluimos la condición en un **WHERE** y hacemos la media (**AVG**) de las cantidades que encontramos en la tabla “transaction” gracias a la unión de las tablas “transaction” y “company”. La media de compras realizadas por la compañía Donec es de 203,72 euros.

```

121  # Ejercicio 2
122  -- Cantidad media por IBAN de las tarjetas de la compañía Donec Ltd. --
123
124 •  SELECT company_name, iban, round(avg(amount),2) AS MediaCompraDonec
125  FROM credit_cards
126  JOIN transaction
127  ON transaction.card_id = credit_cards.id
128  JOIN company
129  ON transaction.business_id = company.company_id
130  WHERE company_name = "Donec Ltd"
131  GROUP BY iban, company_name;
132
133  ◇ 29:131
100% ◇ 29:131

Result Grid Filter Rows: Search Export:
company_name iban MediaCompraDon...
Donec Ltd PT87806228135092429456346 203.72

Result 33 Read
Action Output ◇
Time Action Response Duration / Fetch
48 15:31:20 S 1 row(s) returned 0.0011 sec / 0.00

```

Nivel 2

Creación de una nueva Tabla con el ESTADO DE LAS TARJETAS.

Primero, creo una tabla llamada “**EstadoTarjetas**” en la que voy a guardar el resultado de la consulta sobre las tarjetas que están activas según las condiciones dadas.

```

> desglose_transacti... 156
> EstadoTarjetas 157  # Nivel 2
> numbers 158  # Ejercicio 1
> products 159  -- Creación de una tabla con el estado de las tarjetas --
> transaction 160  # ACTIVA si ha tenido 2 o menos transacciones denegadas en las últimas tres transacciones
> users 161  # DESACTIVA si las tres últimas transacciones han sido denegadas.
162  -- Creación de una tabla --
163 • CREATE TABLE EstadoTarjetas (
164    card_id VARCHAR(10),
165    Estado_Tarjeta VARCHAR(10)
166 );
167

100% ◇ 3:166

Action Output ◇
Time Action Response Duration / Fetch Time
12 10:25:55 CREATE TABLE EstadoTarjetas ( card_id VARCHAR(10),... 0 row(s) affected 0.0072 sec

```

Para recopilar las últimas tres transacciones vamos a hacer primero de todo una selección en la que se pueda tener acceso a los siguientes campos: el identificador de la tarjeta (“*card_id*”), la fecha (“*timestamp*”) y el campo “*declined*” que nos informa sobre si la transacción ha sido realizada o no. Para esto, creo una vista llamada “**Ultimas3transacciones**”.

Para poder acceder a las últimas tres transacciones para cada tarjeta, utilizo la función **ROW_NUMBER**, es una función que nos ayuda a asignar con un número secuencial a cada una de las transacciones realizadas por un tarjeta en concreto y comienza la asignación de números para la siguiente tarjeta.

ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) as rn

Esta función nos crea una asignación de número llamada **r n** para cada una de las transacciones ordenadas por la fecha descendente para cada una de las tarjetas. Finalmente, se limita la tabla a sólo las últimas tres transacciones con la cláusula **WHERE rn <= 3**

card_id	timestamp	rn
CcU.2945	2022-03-12	1
CcU.2945	2022-01-27	2
CcU.2945	2021-10-23	3
CcU-2999	2023-01-13	1
CcU-2999	2022-12-10	2
CcU-2999	2022-02-01	3

Aquí en la imagen se puede observar la selección que se ha guardado como “**“Ultimas3transacciones”**. Esta selección va a ser la subconsulta que voy a incluir en la consulta final.

```

145  # Nivel 2
146  # Ejercicio 1
147  -- Creación de una tabla con las columnas card_id, fecha y si han sido declinadas o no las transacciones --
148 • CREATE VIEW Ultimas3transacciones AS
149  SELECT card_id, timestamp, declined,
150  FROM (SELECT card_id, timestamp, declined,
151  ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) as rn
152  FROM transaction ) as subquery
153  WHERE rn<= 3
154  ORDER BY card_id, timestamp;
155 • SELECT * FROM ultimas3transacciones;
```

The screenshot shows a database management tool interface. At the top, there is a code editor window displaying the SQL code for creating a view named 'Ultimas3transacciones'. Below the code editor is a 'Result Grid' section showing the data from the view. The grid has columns: 'card_id', 'timestamp', and 'declined'. The data includes several rows for different card IDs, their timestamps, and their 'declined' status (0 or 1). Below the result grid is an 'Action Output' section which shows a single query execution: 'SELECT * FROM ultimas3transacciones' with a response of '376 row(s) returned'.

card_id	timestamp	declined
CcU-2938	2022-02-24 11:01:42	0
CcU-2938	2022-03-09 20:53:59	0
CcU-2938	2022-03-12 09:23:10	0
CcU-2945	2021-06-15 00:26:29	1
CcU-2945	2022-02-04 15:52:56	0
CcU-2952	2021-05-06 05:33:39	1
CcU-2952	2022-01-30 15:16:36	0
CcU-2959	2022-02-28 00:10:50	0

Después, uso un **CASE WHEN** para generar una nueva columna en la que se muestran las tarjetas como

- **ACTIVA:** las tarjetas que en las últimas tres transacciones no han sido ninguna de ellas denegadas.
- **DESACTIVA:** las tarjetas que al menos una transacción de las tres últimas ha sido denegada.

```

168    -- Insertar los datos en la nueva tabla --
169 • INSERT INTO EstadoTarjetas (card_id, Estado_Tarjeta)
170     SELECT card_id,
171         CASE WHEN Sum(declined) = 3 THEN 'DESACTIVA'
172             ELSE 'ACTIVA' END AS Estado_Tarjeta
173     FROM (SELECT card_id, timestamp, declined
174             FROM (SELECT card_id, timestamp, declined,
175                 ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) as rn
176                 FROM transaction ) as subquery
177             WHERE rn<= 3
178             ORDER BY card_id, timestamp) AS ultimas3transacciones
179     GROUP BY card_id;
180 • SELECT * FROM EstadoTarjetas;

```

100%	18:179	...
Action Output		
	Time	Action
13	10:29:03	INSERT INTO EstadoTarjetas (card_id, Estado_Tarjeta) SE... 275 row(s) affected Records: 275 Duplicates: 0 War... 0.022 sec

Y por último incluyo una instrucción de **INSERT INTO** para incluirlas en la tabla creada anteriormente “EstadoTarjetas”.

Ejercicio 1: ¿Cuántas tarjetas están activas?

Para esto, se puede hacer un **COUNT** del campo “Estado_Tarjeta” de la vista que hemos creado anteriormente, con la condición de que este campo sea “**ACTIVA**”. Esta consulta nos devuelve que hay **275 tarjetas que están activas** según las condiciones planteadas.

```

186    -- Cuentas tarjetas están activas --
187 • SELECT count(Estado_Tarjeta)
188     FROM EstadoTarjetas
189     WHERE Estado_Tarjeta = 'ACTIVA';

```

100%	33:189	...
Result Grid		
	Filter Rows:	Search
	Export:	grid
count(Estado_Tarje...		Result Grid
275		Read Only
Result 70		
Action Output		
	Time	Action
171	12:35:13	SELECT * FROM Esta... 275 row(s) returned
172	12:39:13	SELECT count(Estad... 1 row(s) returned

Nivel 3

Ejercicio 1: ¿Cuántas veces se ha vendido un producto?

Para empezar, creo la nueva tabla “products”, con los campos que se identifican al abrir el documento *products.csv* en un excel. Al ejecutar **show tables**, se puede ver que la tabla “products” ha sido creada.

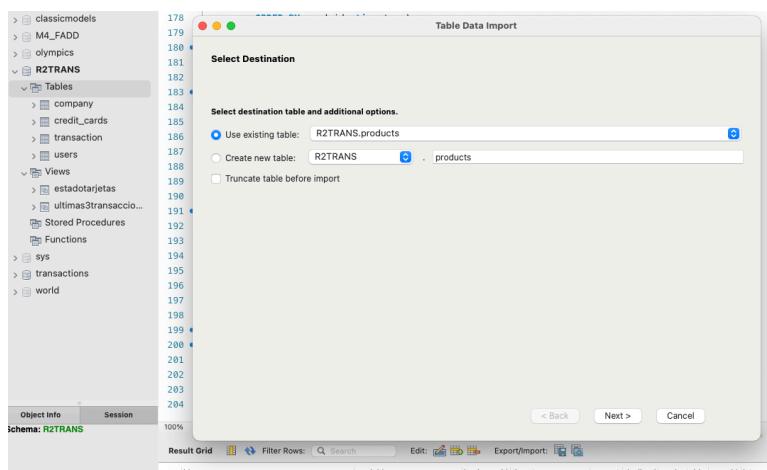
```

191 • CREATE TABLE IF NOT EXISTS products (
192     id VARCHAR(10) NOT NULL,
193     product_name VARCHAR(50),
194     price VARCHAR(20),
195     colour VARCHAR(20),
196     weight VARCHAR(10),
197     warehouse_id VARCHAR(10)
198 );
199 • show tables;
200
  
```

Result Grid | Filter Rows: | Search | Export: | Action Output | Time | Action | Response | 7 row(s) |

94 20:11:39 show tables

Ahora procedo a insertar los datos del archivo *products.csv* en la tabla “*products*” con el *Table Data Import Wizard* de Workbench.



Al hacer un ***SELECT * FROM products***, se observa que los datos se han introducido correctamente y que se han incluido ***un total de 100 registros*** en la tabla “*products*”.

```

200 • select * from products;
201
  
```

Result Grid | Filter Rows: | Search | Export: | Action Output | Time | Action | Response | 100 row(s) returned |

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
6	dooku solo	\$136.60	#c4c4c4	0.8	WH-1
7	north of Casterly	\$63.33	#b7b7b7	0.6	WH-2
8	Winterfell	\$32.37	#383838	1.4	WH-3
9	Winterfell	\$76.40	#b5b5b5	1.2	WH-4

products 59

105 15:15:42 select * from products

A continuación, voy a crear la relación entre la tabla “*products*” que es otra tabla de dimensiones que se añade al modelo en estrella, y la tabla “*transaction*” que es la tabla de métricas.

Los campos por los que se relacionan estas tablas son `transaction.product_ids = products.id`. Sin embargo, vamos a encontrar un problema a la hora de crear esta relación porque el campo “`product_ids`” de la tabla “`transaction`” presenta id de varios productos separados por comas. Es decir, la granularidad de este campo es de bajo nivel y antes de crear la relación es necesario aumentar el nivel de granularidad de esta tabla. Lo que significa dividir un registro del campo “`product_ids`” en varios registros para poder contabilizar individualmente cada uno de los productos.

Podemos crear la PK en la tabla “`products`”.

```
202  # products 1:N transaction
203  -- creación de Clave primaria en tabla products --
204 • ALTER TABLE products
205  ADD CONSTRAINT id Primary key(id);
206 • show columns from products;
207
```

Result Grid | Filter Rows: Search | Export:

Field	Type	Null	Key	Default	Extra
<code>id</code>	<code>varchar(10)</code>	NO	PRI	<code>NULL</code>	
<code>product_name</code>	<code>varchar(50)</code>	YES		<code>NULL</code>	
<code>price</code>	<code>varchar(20)</code>	YES		<code>NULL</code>	
<code>colour</code>	<code>varchar(20)</code>	YES		<code>NULL</code>	
<code>weight</code>	<code>varchar(10)</code>	YES		<code>NULL</code>	
<code>warehouse_id</code>	<code>varchar(10)</code>	YES		<code>NULL</code>	

Result 62

Action Output | Time | Action | Response

109 15:26:25 show columns from products 6 row(s) returned

Sin embargo, devuelve un error al intentar crear la relación foránea con la tabla “`transaction`”.

```
208  -- creación de Clave foránea en tabla transaction --
209 • ALTER TABLE transaction
210  ADD CONSTRAINT product_ids FOREIGN KEY (product_ids)REFERENCES products(id);
211
```

Action Output | Time | Action | Response

110 15:27:36 ALTER TABLE transaction ADD CONSTRAINT product_ids FOREIGN... Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('r2trans'. '#sql-247d_65', CONSTRAINT 'product_id'

Por lo tanto, debo descomponer los valores del campo “`product_ids`” de la tabla “`transaction`”. Para esto hay que realizar varios pasos:

- **Crear una tabla llamada “desglose_transaction”** en la que se va a almacenar los resultados de descomponer el campo “`product_ids`” (campo en el que se encuentran varios identificadores de productos) en un nuevo campo “`product_id`” en el que se inserta solo un identificador de producto, obtenido de desglosar el campo anterior.

```
216 • CREATE TABLE desglose_transaction (
217  id VARCHAR(36),
218  product_id INT
219 );
```

Action Output | # | Time | Action | Message | Duration / F...

98 10:53:18 ALTER TABLE desglose_transaction ADD CONSTRAINT id FOREIGN... 1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0 0.109 sec

99 10:57:19 show columns from desglose_transaction ALTER TABLE desglose_transaction ADD CONSTRAINT id FOREIGN KEY (id)REFERENCES transaction(id) 0.000 sec / C...

100 11:07:55 DROP TABLE 'r2trans'.desglose_transaction 0.016 sec

101 11:08:20 CREATE TABLE desglose_transaction (id VARCHAR(36), produc... 0 row(s) affected 0.016 sec

- **Crear una tabla de números y llenarla con valores.** En este caso, era suficiente con una tabla de número de 0 a 9, puesto que no hay ningún campo “*product_ids*” que contenga más de 10 identificadores de productos.

```

230    -- Crea una tabla de números si no la tienes ya
231 • CREATE TABLE numbers (
232     n INT
233 );
234
235    -- Llena la tabla de números con algunos valores
236 • INSERT INTO numbers (n) VALUES (0), (1), (2), (3), (4), (5), (6), (7), (8), (9);
237 • SELECT * FROM numbers;
238
100% 23:237 | Result Grid Filter Rows: Search Export: Read C
  n
  0
  1
  2
  3
numbers 20
Action Output
Time Action Response Duration / Fetch Ti
57 17:25:07 CREATE TEMPORARY TA... 0 row(s) affected 0.0026 sec
58 17:31:44 SELECT * FROM numbers 10 row(s) returned 0.00068 sec / 0.00

```

- **Usar la tabla de números como contador para dividir los ids de productos.** Para este fin, se va a utilizar el comando **SUBSTRING_INDEX(str, delim, count)** que devuelve una subcadena desde el inicio de la cadena (*str*) hasta el (*count*) separado por el delimitador (*delim*). En nuestro caso:
 - *str = product_ids* (sería la lista de identificadores de productos)
 - *delim = ','* porque están separados por una coma.
 - *count = numbers.n + 1* en el que *numbers.n* es un valor de la tabla que hemos creado de números, y va tomando números hasta (*n+1*)

Y además, uso de nuevo el comando **SUBSTRING_INDEX(..., ',', -1)** que aplico a la subcadena resultante del anterior comando SUBSTRING_INDEX para extraer el último elemento de la cadena después de la coma. Finalmente con un **CAST**, lo convierto de cadena a entero.

Para crear un registro por cada uno de los IDs que hay en el campo “*product_ids*” en el JOIN se combina la tabla “transaction” con la tabla “numbers” cuando se cumpla un condición:

CHAR_LENGTH(product_ids) - CHAR_LENGTH(REPLACE(product_ids, ',', '')) >= numbers.n

Esta condición calcula la longitud del string “*product_ids*” ***CHAR_LENGTH(product_ids)*** y le resta la longitud del mismo string sin comas ***CHAR_LENGTH(REPLACE(product_ids, ',', ''))***, dando lugar a la cantidad de comas que hay en el campo y esto debe ser superior o igual al número de la tabla “*numbers*” que corresponda.

Así el resultado de la JOIN es la creación de diversas filas para cada registro en la tabla “*transaction*” en la que se inserta cada uno de los IDs del campo “*product_ids*”.

```

230      -- Ahora usa la tabla de números para dividir los ids de productos --
231 •  INSERT INTO desglose_transaction (id, product_id)
232     SELECT
233       id,
234       CAST(SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', numbers.n + 1), ',', -1) AS UNSIGNED)
235     FROM
236       transaction
237     JOIN
238       numbers ON CHAR_LENGTH(product_ids) - CHAR_LENGTH(REPLACE(product_ids, ',', '')) >= numbers.n;
239
240 •  SELeCT id, card_id, product_ids FROM transaction

```

Output				
Action Output				
#	Time	Action	Message	Duration /
99	10:57:19	show columns from desglose_transaction	8 row(s) returned	0.000 sec
100	11:07:55	DROP TABLE r2trans'.desglose_transaction'	0 row(s) affected	0.016 sec
101	11:08:20	CREATE TABLE desglose_transaction (id VARCHAR(36), produc...	0 row(s) affected	0.016 sec
102	11:08:28	INSERT INTO desglose_transaction (id,product_id) SELECT id, ...	1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0	0.031 sec

Para comprobar que se ha desglosado correctamente el campo “*product_ids*”. Selecciono una transacción específica. Se puede observar que su campo “*product_ids*” contiene tres identificadores de productos: ‘71, 1, 19’

```

255 •  SELeCT id, card_id, product_ids FROM transaction
256   WHERE id = '02C6201E-D90A-1859-B4EE-88D2986D3B02';
257

```

Result Grid			
transaction 37			
Action Output			
id	card_id	product_ids	
02C6201E-D90A-1859-B4EE-88D2986D3B02	CcU-2938	71, 1, 19	
HULL	NULL	NULL	
Edit: Export/Import:			
84	18:18:17	SELECT card_id, pro...	3 row(s) returned
85	18:19:25	SELeCT id, card_id,...	1 row(s) returned

Si selecciono esa misma transacción en la tabla “*desglose_transaction*”, se observa que ahora aparecen 3 registros, uno para cada uno de los IDs de los diferentes productos. Por lo tanto, se ha realizado con éxito el desglose del campo “*product_ids*”.

```

255 •   SELECT card_id, product_id, id FROM desglose_transaction
256 WHERE id = '02C6201E-D90A-1859-B4EE-88D2986D3B02';
100%  51:256
Result Grid Filter Rows: Search Export:
card_id product_id id
CcU-2938 19 02C6201E-D90A-1859-B4EE-88D2986D3B02
CcU-2938 1 02C6201E-D90A-1859-B4EE-88D2986D3B02
CcU-2938 71 02C6201E-D90A-1859-B4EE-88D2986D3B02
desglose_transaction 45
Action Output
Time Action Response
100 19:06:58 INSERT INTO desglose... 1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0
101 19:08:03 SELECT card_id, pro... 3 row(s) returned

```

Ahora necesitamos **crear la relación 1:N** entre la nueva tabla “*desglose_transaction*” y la tabla “*products*” pero antes debo asegurarme que los campos a relacionar tengan el mismo tipo de dato. Así que después de comprobar que tenían tipos diferentes, he cambiado el campo “*id*” de la tabla “*products*” a tipo de dato INT como el campo “*product_id*” en la tabla “*desglose_transaction*”.

```

267 -- Cambio el tipo de dato del campo ID de la tabla PRODUCTS para poder crear la clave foránea --
268 • ALTER TABLE products CHANGE id id INT;
269 • show columns from products;
100%  28:269
Result Grid Filter Rows: Search Export:
Field Type Null Key Default Extra
id int NO PRI NULL
product_name varchar(50) YES NULL
price varchar(20) YES NULL
colour varchar(20) YES NULL
weight varchar(10) YES NULL
Result 54
Action Output
Time Action Response Duration / Fetch 1
115 19:15:25 ALTER TABLE produc... 100 row(s) affected Records: 100 Duplicates: 0 Warnings: 0 0.028 sec
116 19:15:43 show columns from p... 6 row(s) returned 0.0017 sec / 0.000

```

Finalmente, es posible crear la clave foránea (FK) en la nueva tabla “*desglose_transaction*” en el campo “*product_id*”, relacionado con la clave primaria de la tabla “*products*” que es el campo “*id*”.

Para acabar de conectar estas dos tablas a la base de datos “*R2TRANS*” creo una relación 1:N entre la tabla “*transaction*” y la tabla “*desglose_transaction*”, creando una FK en la tabla “*desglose_transaction*” relacionada con el campo “*id*” de la tabla “*transaction*”.

```

252     # creación de Clave foránea en tabla desglose_transaction con tabla products --
253 • ALTER TABLE desglose_transaction
254     ADD CONSTRAINT fk2_product_id FOREIGN KEY (product_id)REFERENCES products(id);
255
256
257     # creación de Clave foránea en tabla desglose_transaction con tabla transaction --
258 • ALTER TABLE desglose_transaction
259     ADD CONSTRAINT id FOREIGN KEY (id)REFERENCES transaction(id);
260 • show columns from desglose_transaction;
261

```

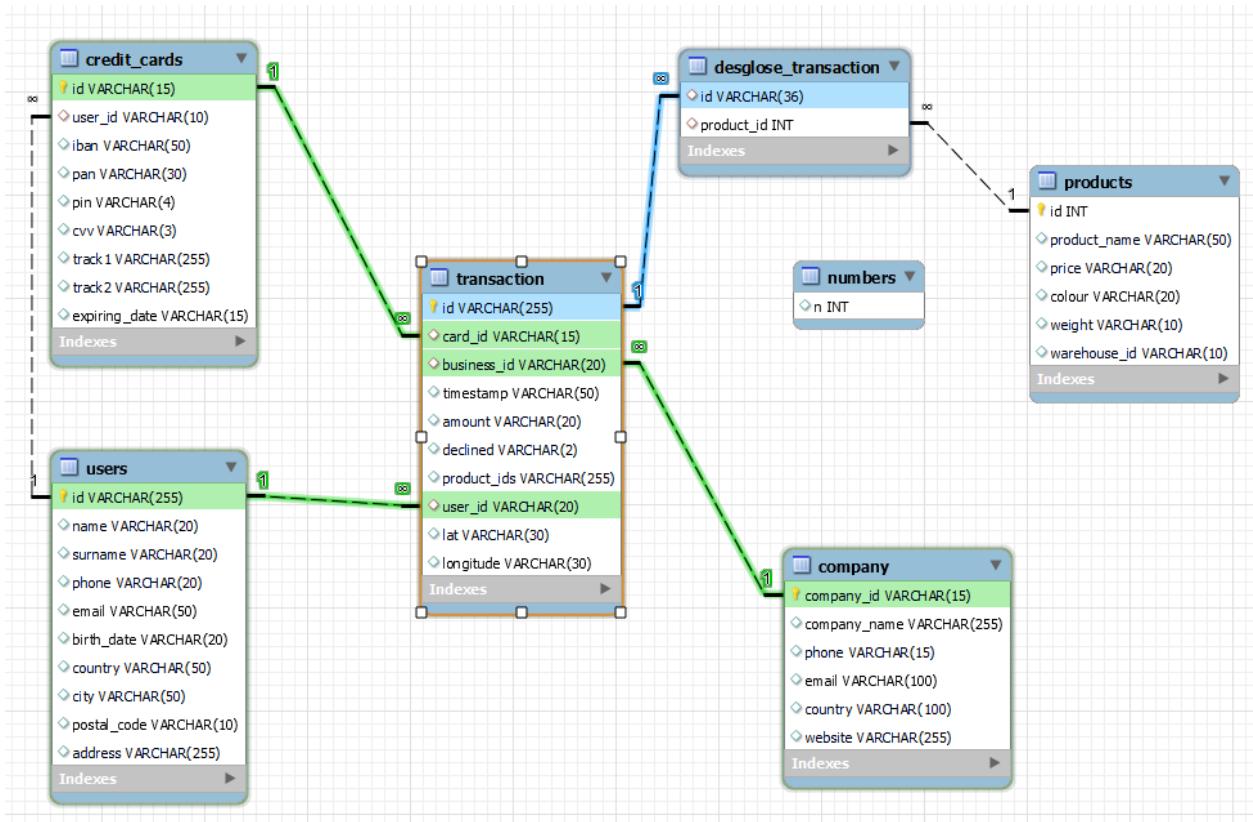
The screenshot shows the MySQL Workbench interface. In the top pane, there is a 'Result Grid' showing the columns of the 'desglose_transaction' table: 'id' (varchar(36), YES, MUL, NULL) and 'product_id' (int, YES, MUL, NULL). Below this is a 'Result 13' section and an 'Output' pane containing the execution history of the commands.

Field	Type	Null	Key	Default	Extra
id	varchar(36)	YES	MUL	NULL	
product_id	int	YES	MUL	NULL	

Action Output:

- 104 11:14:03 ALTER TABLE desglose_transaction ADD CONSTRAINT fk2_product_id FOREIGN KEY (product_id)REFERENCES products(id); 1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0 0.093 sec
- 105 11:14:10 ALTER TABLE desglose_transaction ADD CONSTRAINT id FOREIGN KEY (id)REFERENCES transaction(id); 1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0 0.125 sec
- 106 11:14:13 show columns from desglose_transaction 2 row(s) returned 0.016 sec / 0.000

De nuevo, una imagen del **Diagrama Entidad Relación** de la base de datos “**R2TRANS**” después de las últimas modificaciones, con la tabla de productos y la nueva tabla creada (“*desglose_transaction*”) para relacionar la tabla de dimensiones “*products*” con la tabla de hechos “*transaction*”.



Por último, realizo la consulta para determinar el número de veces que se ha vendido cada uno de los productos. Gracias a una JOIN de las tablas “*products*” y “*desglose_transaction*” y agrupando por “*product_name*”, se puede hacer un COUNT de los productos vendidos por el campo “*product_id*” de la tabla “*desglose_transaction*”.

```
274    -- ¿Cuántas veces se ha vendido cada producto? --
275 • SELECT product_name, count(product_id) AS VecesVendido
276   FROM products
277   JOIN desglose_transaction
278     ON products.id = desglose_transaction.product_id
279   GROUP BY product_name
280   ORDER BY VecesVendido DESC;
```

The screenshot shows a database query results interface. At the top, there is a code editor window with the SQL query. Below it is a results grid table with two columns: 'product_name' and 'VecesVendido'. The data includes rows for Direwolf Stannis (106), skywalker ewok (100), riverlands north (68), Winterfell (68), Direwolf riverlands the (66), Tarly Stark (65), duel (65), and Tully (62). A sidebar on the right shows icons for 'Result Grid' and 'Form Editor'. Below the results grid is a section titled 'Action Output' with a table showing two actions: SELECT statements at times 19:25:13 and 19:25:22, both returning 24 row(s). The duration for each action is listed as 0.0069 sec / 0.00001... and 0.0042 sec / 0.00001... respectively.

product_name	VecesVendido
Direwolf Stannis	106
skywalker ewok	100
riverlands north	68
Winterfell	68
Direwolf riverlands the	66
Tarly Stark	65
duel	65
Tully	62

Action Output	Time	Action	Response	Duration / Fetch Time
119	19:25:13	SELECT product_na...	24 row(s) returned	0.0069 sec / 0.00001...
120	19:25:22	SELECT product_na...	24 row(s) returned	0.0042 sec / 0.00001...