

S4.01. Creación de Base de Datos

Creación de DB y adecuación

Primero creamos la base de datos (a la cual llamaremos bussines) y las tablas correspondientes a los archivos CSV proporcionados.

Usaremos el comando CREATE TABLE, al cual añadiremos IF NOT EXISTS para que cree las tablas solo en caso de que no existan.

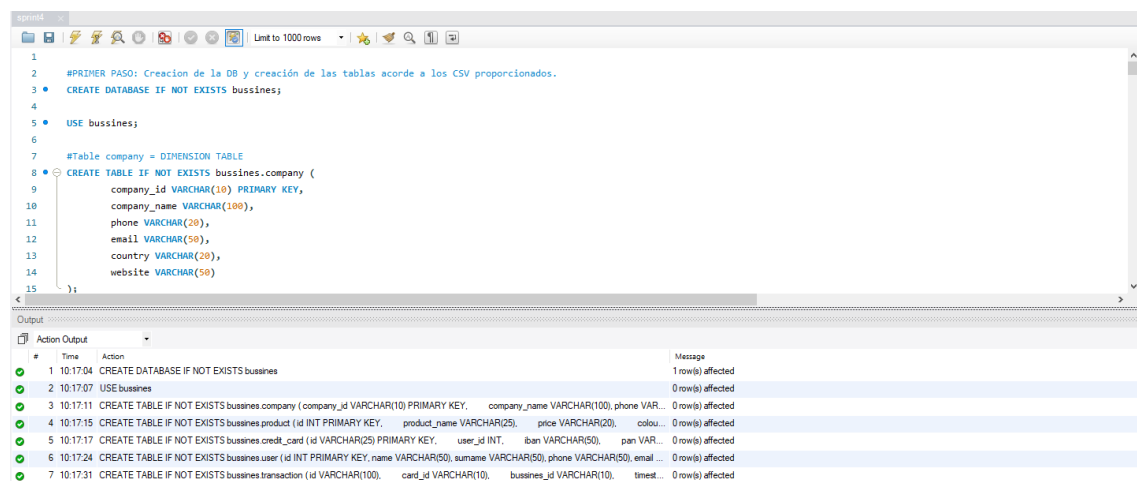
Según los archivos proporcionados tendremos un modelo en estrella con la tabla de *transaction* como **FACT TABLE**, y las tablas *user*, *product*, *credit_card* y *company* como **DIMENSION TABLE**.

Los archivos CSV se podrían cargar o bien mediante comandos o mediante el *import wizard* del *MySQL Workbench*. En este caso usaremos el *import wizard*.

Creación de tablas:

Bussines.company = DIMENSION TABLE

PRIMARY KEY = Company_id

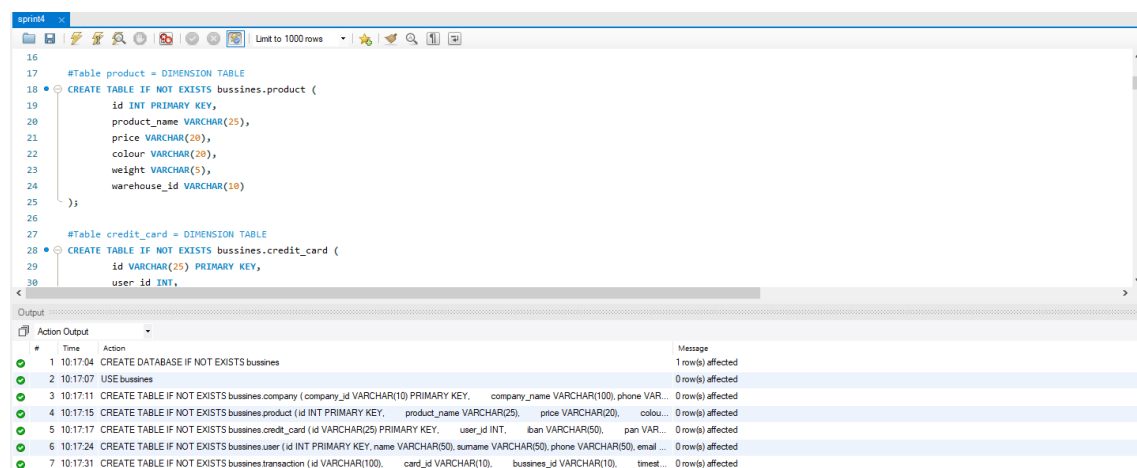


```
1
2 #PRIMER PASO: Creación de la DB y creación de las tablas acorde a los CSV proporcionados.
3 CREATE DATABASE IF NOT EXISTS bussines;
4
5 USE bussines;
6
7 #Table company = DIMENSION TABLE
8 CREATE TABLE IF NOT EXISTS bussines.company (
9     company_id VARCHAR(10) PRIMARY KEY,
10    company_name VARCHAR(100),
11    phone VARCHAR(20),
12    email VARCHAR(50),
13    country VARCHAR(20),
14    website VARCHAR(50)
15 );
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

#	Time	Action	Message
1	10:17:04	CREATE DATABASE IF NOT EXISTS bussines	1 row(s) affected
2	10:17:07	USE bussines	0 row(s) affected
3	10:17:11	CREATE TABLE IF NOT EXISTS bussines.company (company_id VARCHAR(10) PRIMARY KEY, company_name VARCHAR(100), phone VAR...	0 row(s) affected
4	10:17:15	CREATE TABLE IF NOT EXISTS bussines.product (id INT PRIMARY KEY, product_name VARCHAR(25), price VARCHAR(20), colou...	0 row(s) affected
5	10:17:17	CREATE TABLE IF NOT EXISTS bussines.credit_card (id INT PRIMARY KEY, user_id INT, iban VARCHAR(50), pan VAR...	0 row(s) affected
6	10:17:24	CREATE TABLE IF NOT EXISTS bussines.user (id INT PRIMARY KEY, name VARCHAR(50), surname VARCHAR(50), phone VARCHAR(50), email ...	0 row(s) affected
7	10:17:31	CREATE TABLE IF NOT EXISTS bussines.transaction (id VARCHAR(100), card_id VARCHAR(10), bussines_id VARCHAR(10), timest...	0 row(s) affected

Bussines.product = DIMENSION TABLE

PRIMARY KEY = id



```
16
17 #Table product = DIMENSION TABLE
18 CREATE TABLE IF NOT EXISTS bussines.product (
19     id INT PRIMARY KEY,
20     product_name VARCHAR(25),
21     price VARCHAR(20),
22     colour VARCHAR(20),
23     weight VARCHAR(5),
24     warehouse_id VARCHAR(10)
25 );
26
27 #Table credit_card = DIMENSION TABLE
28 CREATE TABLE IF NOT EXISTS bussines.credit_card (
29     id VARCHAR(25) PRIMARY KEY,
30     user_id INT,
31     iban VARCHAR(50),
32     pan VARCHAR(16),
33     exp_date VARCHAR(10),
34     cvv2 VARCHAR(3)
35 );
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

#	Time	Action	Message
1	10:17:04	CREATE DATABASE IF NOT EXISTS bussines	1 row(s) affected
2	10:17:07	USE bussines	0 row(s) affected
3	10:17:11	CREATE TABLE IF NOT EXISTS bussines.company (company_id VARCHAR(10) PRIMARY KEY, company_name VARCHAR(100), phone VAR...	0 row(s) affected
4	10:17:15	CREATE TABLE IF NOT EXISTS bussines.product (id INT PRIMARY KEY, product_name VARCHAR(25), price VARCHAR(20), colou...	0 row(s) affected
5	10:17:17	CREATE TABLE IF NOT EXISTS bussines.credit_card (id INT PRIMARY KEY, user_id INT, iban VARCHAR(50), pan VAR...	0 row(s) affected
6	10:17:24	CREATE TABLE IF NOT EXISTS bussines.user (id INT PRIMARY KEY, name VARCHAR(50), surname VARCHAR(50), phone VARCHAR(50), email ...	0 row(s) affected
7	10:17:31	CREATE TABLE IF NOT EXISTS bussines.transaction (id VARCHAR(100), card_id VARCHAR(10), bussines_id VARCHAR(10), timest...	0 row(s) affected

Bussines.credit_card = DIMENSION TABLE

PRIMARY KEY = id

```
26
27 #Table credit_card = DIMENSION TABLE
28 CREATE TABLE IF NOT EXISTS bussines.credit_card (
29     id VARCHAR(25) PRIMARY KEY,
30     user_id INT,
31     iban VARCHAR(50),
32     pan VARCHAR(50),
33     pin VARCHAR(50),
34     cvv VARCHAR(50),
35     track1 VARCHAR(100),
36     track2 VARCHAR(100),
37     expiring_date VARCHAR(10)
38 );
39
40
```

Output

#	Time	Action	Message
1	10:17:04	CREATE DATABASE IF NOT EXISTS bussines	1 row(s) affected
2	10:17:07	USE bussines	0 row(s) affected
3	10:17:11	CREATE TABLE IF NOT EXISTS bussines.company (company_id VARCHAR(10) PRIMARY KEY, company_name VARCHAR(100), phone VAR...	0 row(s) affected
4	10:17:15	CREATE TABLE IF NOT EXISTS bussines.product (id INT PRIMARY KEY, product_name VARCHAR(25), price VARCHAR(20), colou...	0 row(s) affected
5	10:17:17	CREATE TABLE IF NOT EXISTS bussines.credit_card (id VARCHAR(25) PRIMARY KEY, user_id INT, iban VARCHAR(50), pan VAR...	0 row(s) affected
6	10:17:24	CREATE TABLE IF NOT EXISTS bussines.user (id INT PRIMARY KEY, name VARCHAR(50), surname VARCHAR(50), phone VARCHAR(50), email ...	0 row(s) affected
7	10:17:31	CREATE TABLE IF NOT EXISTS bussines.transaction (id VARCHAR(100), card_id VARCHAR(10), bussines_id VARCHAR(10), timest...	0 row(s) affected

Bussines.user = DIMENSION TABLE

PRIMARY KEY = id

```
41 #Table user = DIMENSION TABLE
42 CREATE TABLE IF NOT EXISTS bussines.user (
43     id INT PRIMARY KEY,
44     name VARCHAR(50),
45     surname VARCHAR(50),
46     phone VARCHAR(50),
47     email VARCHAR(50),
48     birth_date VARCHAR(25),
49     country VARCHAR(50),
50     city VARCHAR(50),
51     postal_code VARCHAR(25),
52     address VARCHAR(50)
53 );
54
55 #Table transaction = FACT TABLE
```

Output

#	Time	Action	Message
1	10:17:04	CREATE DATABASE IF NOT EXISTS bussines	1 row(s) affected
2	10:17:07	USE bussines	0 row(s) affected
3	10:17:11	CREATE TABLE IF NOT EXISTS bussines.company (company_id VARCHAR(10) PRIMARY KEY, company_name VARCHAR(100), phone VAR...	0 row(s) affected
4	10:17:15	CREATE TABLE IF NOT EXISTS bussines.product (id INT PRIMARY KEY, product_name VARCHAR(25), price VARCHAR(20), colou...	0 row(s) affected
5	10:17:17	CREATE TABLE IF NOT EXISTS bussines.credit_card (id VARCHAR(25) PRIMARY KEY, user_id INT, iban VARCHAR(50), pan VAR...	0 row(s) affected
6	10:17:24	CREATE TABLE IF NOT EXISTS bussines.user (id INT PRIMARY KEY, name VARCHAR(50), surname VARCHAR(50), phone VARCHAR(50), email ...	0 row(s) affected
7	10:17:31	CREATE TABLE IF NOT EXISTS bussines.transaction (id VARCHAR(100), card_id VARCHAR(10), bussines_id VARCHAR(10), timest...	0 row(s) affected

Bussines.transaction = FACT TABLE

PRIMARY KEY = id

```
55 #Table transaction = FACT TABLE
56 CREATE TABLE IF NOT EXISTS bussines.transaction (
57     id VARCHAR(100),
58     card_id VARCHAR(10),
59     bussines_id VARCHAR(10),
60     timestamp TIMESTAMP,
61     amount DECIMAL(10, 2),
62     declined BOOLEAN,
63     product_id VARCHAR(15),
64     user_id INT,
65     lat VARCHAR(20),
66     longitude VARCHAR(20),
67     FOREIGN KEY (card_id) REFERENCES credit_card(id),
68     FOREIGN KEY (user_id) REFERENCES user(id),
69     FOREIGN KEY (bussines_id) REFERENCES company(company_id)
70 );
```

Output

#	Time	Action	Message
1	10:17:04	CREATE DATABASE IF NOT EXISTS bussines	1 row(s) affected
2	10:17:07	USE bussines	0 row(s) affected
3	10:17:11	CREATE TABLE IF NOT EXISTS bussines.company (company_id VARCHAR(10) PRIMARY KEY, company_name VARCHAR(100), phone VAR...	0 row(s) affected
4	10:17:15	CREATE TABLE IF NOT EXISTS bussines.product (id INT PRIMARY KEY, product_name VARCHAR(25), price VARCHAR(20), colou...	0 row(s) affected
5	10:17:17	CREATE TABLE IF NOT EXISTS bussines.credit_card (id VARCHAR(25) PRIMARY KEY, user_id INT, iban VARCHAR(50), pan VAR...	0 row(s) affected
6	10:17:24	CREATE TABLE IF NOT EXISTS bussines.user (id INT PRIMARY KEY, name VARCHAR(50), surname VARCHAR(50), phone VARCHAR(50), email ...	0 row(s) affected
7	10:17:31	CREATE TABLE IF NOT EXISTS bussines.transaction (id VARCHAR(100), card_id VARCHAR(10), bussines_id VARCHAR(10), timest...	0 row(s) affected

En la **FACT TABLE** también estableceremos los **FOREIGN KEY**, es decir las relaciones de esta tabla con sus dimensiones. Serán las siguientes:

1-FOREIGN KEY (card_id) REFERENCES credit_card(id) – Relación N to ONE.

2-FOREIGN KEY (user_id) REFERENCES user(id) – Relación N to ONE.

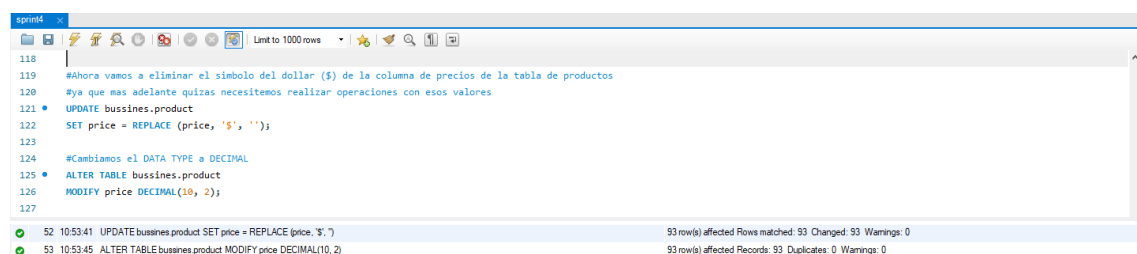
3-FOREIGN KEY (bussines_id) REFERENCES company(company_id) – Relación N to ONE.

No estableceremos una relación entre la tabla *product* y la FACT TABLE. Eso es debido a que en la tabla *transaction* la columna *product_ids* puede contener más de un *id* de producto. Esto establece una relación de MANY to MANY (N to N), la cual vamos a solucionar en el siguiente paso.

Adecuación de la base de datos

Vamos a realizar varias acciones para limpiar y adecuar los datos de forma que sea mas fácil trabajar con ellos luego.

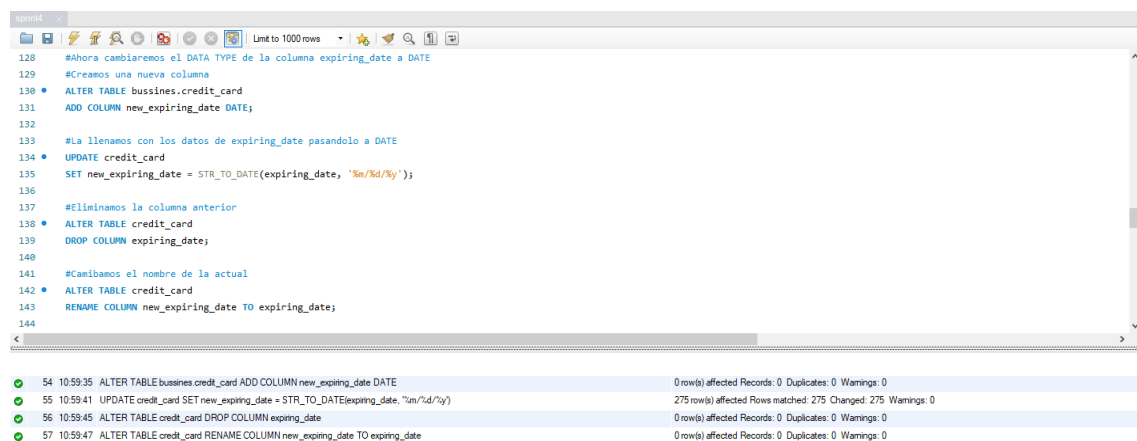
1.Primeramente vamos a sacar el símbolo del dolar (\$) de la columna de precios de la tabla de productos; de esta forma nos será posible realizar operaciones con la columna en caso de que lo necesitemos.



```
118
119 #Ahora vamos a eliminar el símbolo del dolar ($) de la columna de precios de la tabla de productos
120 #ya que mas adelante quizas necesitemos realizar operaciones con esos valores
121 • UPDATE bussines.product
122   SET price = REPLACE (price, '$', '');
123
124 #Cambiamos el DATA TYPE a DECIMAL
125 • ALTER TABLE bussines.product
126   MODIFY price DECIMAL(10, 2);
127
```

52	10:53:41	UPDATE bussines.product SET price = REPLACE (price, '\$', '');	93 row(s) affected Rows matched: 93 Changed: 93 Warnings: 0
53	10:53:45	ALTER TABLE bussines.product MODIFY price DECIMAL(10, 2);	93 row(s) affected Records: 93 Duplicates: 0 Warnings: 0

2.A continuación cambiaremos el **DATA TYPE** de la columna *expiring_date* en la tabla *credit_card*. Lo pasaremos a **DATE**.



```
128 #Ahora cambiaremos el DATA TYPE de la columna expiring_date a DATE
129 #Creamos una nueva columna
130 • ALTER TABLE bussines.credit_card
131   ADD COLUMN new_expiring_date DATE;
132
133 #La llenamos con los datos de expiring_date pasandolo a DATE
134 • UPDATE credit_card
135   SET new_expiring_date = STR_TO_DATE(expiring_date, '%m/%d/%y');
136
137 #Eliminamos la columna anterior
138 • ALTER TABLE credit_card
139   DROP COLUMN expiring_date;
140
141 #Cambiamos el nombre de la actual
142 • ALTER TABLE credit_card
143   RENAME COLUMN new_expiring_date TO expiring_date;
144
```

54	10:59:35	ALTER TABLE bussines.credit_card ADD COLUMN new_expiring_date DATE	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
55	10:59:41	UPDATE credit_card SET new_expiring_date = STR_TO_DATE(expiring_date, '%m/%d/%y')	275 row(s) affected Rows matched: 275 Changed: 275 Warnings: 0
56	10:59:45	ALTER TABLE credit_card DROP COLUMN expiring_date	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
57	10:59:47	ALTER TABLE credit_card RENAME COLUMN new_expiring_date TO expiring_date	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Crearemos una nueva columna en la misma tabla, la llenaremos con los datos de la columna *expiring_date* usando la expresión **STR_TO_DATE**, luego eliminaremos la columna *expiring_date* y renombraremos la nueva columna creada a *expiring_date*.

3.Creación de junction table entre transaction y product.

En sistemas de bases de datos relacionales lo más apropiado cuando se encuentran relaciones de tipo **N to N** es crear una tabla intermedia (*junction table*) que permita extraer información de ambas tablas. Esta *junction table* tendrá relaciones **N to ONE** con la tabla *product* y la tabla *transaction*. Eso nos permitirá simplificar *queries* a la hora de recuperar información y mantener la integridad estructural y lógica de la base de datos, ya que podremos crear **FOREIGN KEYS** entre las tablas que preservaran la consistencia de la base de datos a la hora de introducir nuevos datos en la misma.

Empezaremos separando la columna *product_ids* de la tabla *transaction* en 4 columnas diferentes. Para ello crearemos 4 nuevas columnas que podrán tener un valor **INT** o **NULL** en función de si una *transaction* tiene entre 1 o 4 product ids. Estas columnas se llamarán *product_id1*, *product_id2*, *product_id3*, *product_id4*.

```
spring4 - bussines_table_check
73
74 #Separación de la columna product_ids en 4 columnas diferentes product_idn (n = 1 - 4)
75 #Primero añadimos las nuevas columnas para los product_ids
76 • ALTER TABLE transaction
77   ADD COLUMN product_id1 VARCHAR(20);
78
79 • ALTER TABLE transaction
80   ADD COLUMN product_id2 VARCHAR(20);
81
82 • ALTER TABLE transaction
83   ADD COLUMN product_id3 VARCHAR(20);
84
85 • ALTER TABLE transaction
86   ADD COLUMN product_id4 VARCHAR(20);
87
```

58	11:15:27	ALTER TABLE transaction ADD COLUMN product_id1 VARCHAR(20)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
59	11:15:36	ALTER TABLE transaction ADD COLUMN product_id2 VARCHAR(20)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
60	11:15:38	ALTER TABLE transaction ADD COLUMN product_id3 VARCHAR(20)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
61	11:15:39	ALTER TABLE transaction ADD COLUMN product_id4 VARCHAR(20)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Ahora rellenamos estas columnas con el comando **UPDATE** y la función **SUBSTRING_INDEX**.

```
spring4 - bussines_table_check
90
91 #Con estos UPDATES separamos los products ids en columnas
92 #y nos aseguramos que los valores sean NULL en las columnas que lo requieran
93 • UPDATE bussines.transaction
94   SET
95     product_id1 = SUBSTRING_INDEX(product_id, ',', 1);
96
97 • UPDATE bussines.transaction
98   SET
99     product_id2 = SUBSTRING_INDEX(SUBSTRING_INDEX(product_id, ',', 2), ',', -1)
100   WHERE product_id LIKE '%,%';
101
102 • UPDATE bussines.transaction
103   SET
104     product_id3 = SUBSTRING_INDEX(SUBSTRING_INDEX(product_id, ',', 3), ',', -1)
105   WHERE product_id LIKE '%,%,%';
106
107 • UPDATE bussines.transaction
108   SET
109     product_id4 = SUBSTRING_INDEX(SUBSTRING_INDEX(product_id, ',', -1), ',', -1)
110   WHERE product_id LIKE '%,%,%,%';
111
```

63	11:16:42	UPDATE bussines.transaction SET product_id1 = SUBSTRING_INDEX(product_id, ',', 1)	587 row(s) affected Rows matched: 587 Changed: 587 Warnings: 0
64	11:16:47	UPDATE bussines.transaction SET product_id2 = SUBSTRING_INDEX(SUBSTRING_INDEX(product_id, ',', 2), ',', -1) WHERE product_id LIKE '%,%'	477 row(s) affected Rows matched: 477 Changed: 477 Warnings: 0
65	11:16:50	UPDATE bussines.transaction SET product_id3 = SUBSTRING_INDEX(SUBSTRING_INDEX(product_id, ',', 3), ',', -1) WHERE product_id LIKE '%,%,%'	291 row(s) affected Rows matched: 291 Changed: 291 Warnings: 0
66	11:16:53	UPDATE bussines.transaction SET product_id4 = SUBSTRING_INDEX(SUBSTRING_INDEX(product_id, ',', -1), ',', -1) WHERE product_id LIKE '%,%,%,%'	102 row(s) affected Rows matched: 102 Changed: 102 Warnings: 0

Finalmente definimos la **DATA** de las nuevas columnas como **INT**.

```
spring4 - bussines_table_check
111
112 #Una vez separados los products ids podemos cambiar su formato a INT
113 • ALTER TABLE bussines.transaction
114   MODIFY product_id1 INT,
115   MODIFY product_id2 INT,
116   MODIFY product_id3 INT,
117   MODIFY product_id4 INT;
118
```

68	11:16:34	ALTER TABLE bussines.transaction MODIFY product_id1 INT, MODIFY product_id2 INT, MODIFY product_id3 INT, MODIFY product_id4 INT	587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0
----	----------	---	--

Una vez separados los *product_id* en diferentes columnas vamos a crear la *junction table* a la cual vamos a llamar *transaction_product*. Esta tabla va a contener las dos **PRIMARY KEYS** de las tablas *transaction* y *product*; y va a contener cada pareja de *transaction_id* y *product_id* que exista en la tabla *transaction*. Es decir, si la transacción con id 120 contiene las product ids '15, 25, 35', nuestra nueva tabla va a contener las rows 120-15, 120-25, 120-35; representando así cada par existente entre *transaction_id* y *product_id* en la tabla de *transaction*.

Primero creamos la tabla:

```

118
119 #Creación de JUNCTION TABLE entre transaction y product
120 CREATE TABLE transaction_product (
121     transaction_id VARCHAR(100),
122     product_id INT,
123     PRIMARY KEY (transaction_id, product_id)
124 );
125
69 11:29:33 CREATE TABLE transaction_product (transaction_id VARCHAR(100), product_id INT, PRIMARY KEY (transaction_id, product_id)) 0 row(s) affected

```

Ahora introducimos los datos en la tabla creada. Como hemos separado las *product_id* en diferentes columnas este proceso es muy sencillo, con un simple **SELECT** se relacionan las *product_id* y las *transaction_id*.

```

125
126 INSERT INTO transaction_product (transaction_id, product_id)
127 SELECT id, product_id
128 FROM (
129     SELECT id, product_id1 AS product_id FROM transaction WHERE product_id1 IS NOT NULL
130     UNION
131     SELECT id, product_id2 AS product_id FROM transaction WHERE product_id2 IS NOT NULL
132     UNION
133     SELECT id, product_id3 AS product_id FROM transaction WHERE product_id3 IS NOT NULL
134     UNION
135     SELECT id, product_id4 AS product_id FROM transaction WHERE product_id4 IS NOT NULL
136 ) AS unions;
137
70 11:30:46 INSERT INTO transaction_product (transaction_id, product_id) SELECT id, product_id FROM ( SELECT id, product_id1 AS product_id FROM trans... 1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0

```

La nueva *junction table transaction_product* tiene el siguiente formato:

	transaction_id	product_id
▶	02C6201E-D90A-1859-B4EE-88D2986D3B02	1
	02C6201E-D90A-1859-B4EE-88D2986D3B02	19
	02C6201E-D90A-1859-B4EE-88D2986D3B02	71
	0466A42E-47CF-8D24-FD01-C0B689713128	43
	0466A42E-47CF-8D24-FD01-C0B689713128	47
	0466A42E-47CF-8D24-FD01-C0B689713128	97
	063FBA79-99EC-66FB-29F7-25726D1764A5	5
	063FBA79-99EC-66FB-29F7-25726D1764A5	31
	063FBA79-99EC-66FB-29F7-25726D1764A5	47
	063FBA79-99EC-66FB-29F7-25726D1764A5	67
	0668296C-CDB9-A883-76BC-2E4C44F8C8AE	79
	0668296C-CDB9-A883-76BC-2E4C44F8C8AE	83
	0668296C-CDB9-A883-76BC-2E4C44F8C8AE	89

Ahora podemos establecer **FOREIGN KEYS** entre las tablas *transaction*, *product* y *transaction_product*.

Creamos un índice en la tabla *transaction*, en la columna *id*. A continuación establecemos la relación entre *transaction.id* y *transaction_product.transaction_id*.

```
sprint4 bussines_table_check
152
153 • CREATE INDEX idx_transaction_id ON transaction(id);
154
155 • ALTER TABLE transaction_product
156   ADD FOREIGN KEY (transaction_id) REFERENCES transaction(id);
```

Cuando intentamos establecer la FOREIGN KEY entre las tablas *transaction_product* y *product*, nos aparece un error que nos indica que la FOREIGN KEY CONSTRAINT no se está respetando. Como estamos intentando establecer la FOREIGN KEY desde la tabla *transaction_product* esto nos indica que debe haber algunos *product_id* en la tabla *transaction*, que ahora cuentan en la tabla *transaction_product*, que no existen en la tabla *product*. Con una simple query comprobamos cuales son los *product_id* que no constan en la tabla *product* y los introducimos en la misma.

```
sprint4 bussines_table_check
248
249 • SELECT product.id, transaction_product.product_id
250   FROM transaction_product
251  LEFT JOIN product ON product.id = transaction_product.product_id
252  WHERE product.id IS NULL;
253
```

75 11:46:58 SELECT product.id, transaction_product.product_id FROM transaction_product LEFT JOIN product ON product.id = transaction_product.product_id... 111 row(s) returned

id	product_id
NULL	41
NULL	41
NULL	41
NULL	53
NULL	41
NULL	53
NULL	53
NULL	53
NULL	53
NULL	53
NULL	53

Comprobamos que son las ids 41 y 53. Las introducimos en la tabla *product*, y así podremos establecer la FOREIGN KEY entre las dos tablas. Si esto fuera un caso real contactaríamos con el departamento de ventas para ver si se ha producido algún error o si falta productos en la base de datos, pero en este caso nos limitaremos a crear los *products* aunque estos no contengan información en la tabla más allá de las ids.

```
sprint4 bussines_table_check
147
148 • INSERT INTO product(id)
149   VALUES(41);
150
151 • INSERT INTO product(id)
152   VALUES(53);
```

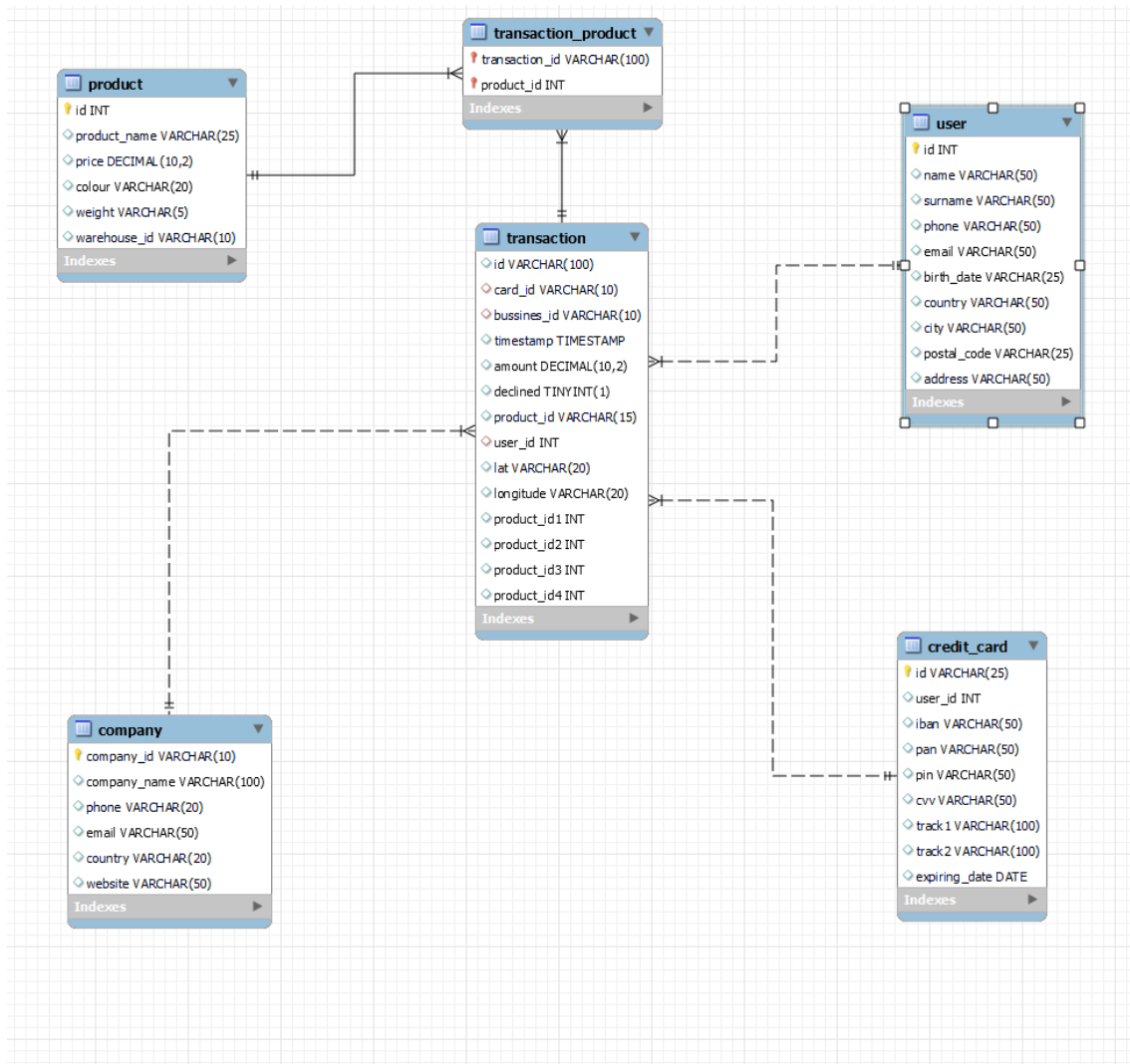
76 11:55:45 INSERT INTO product(id) VALUES(41) 1 row(s) affected
77 11:55:47 INSERT INTO product(id) VALUES(53) 1 row(s) affected

```
sprint4 bussines_table_check
138 #Intentamos estavlecer una FOREIGN KEY entre tablas transaction_product y product pero no deja
139 • ALTER TABLE transaction_product
140   ADD FOREIGN KEY (product_id) REFERENCES product(id);
141
```

78 11:56:39 ALTER TABLE transaction_product ADD FOREIGN KEY (product_id) REFERENCES product(id) 1457 row(s) affected Records: 1457 Duplicates: 0 Warnings: 0

Con este paso final hemos acabado de adecuar nuestra base de datos y esta está lista para ser trabajada.

Este es el esquema final de nuestra base de datos.

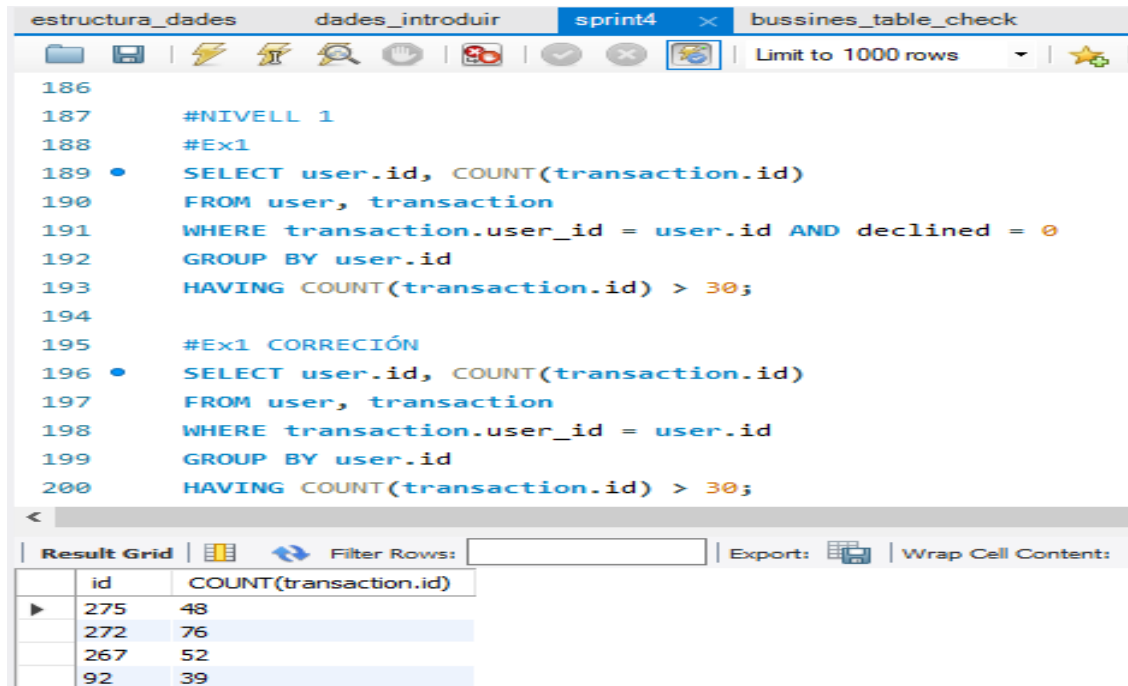


Se trata de un **modelo en estrella** con la tabla *transaction* como FACT TABLE i cuatro DIMENSION TABLES (*user*, *credit_card*, *company*, *product*) además de una tabla de unión (*junction table*) *transaction_product* que nos soluciona la relación **N to N** entre las tablas *transaction* y *product*, estableciendo ella misma relaciones **N to ONE** con las tablas *transaction* y *product*.

NIVELL 1

Ex1

Con un simple **COUNT** podemos y la cláusula **HAVING** para establecer que tenga más de 30 transacciones podemos extraer la información.



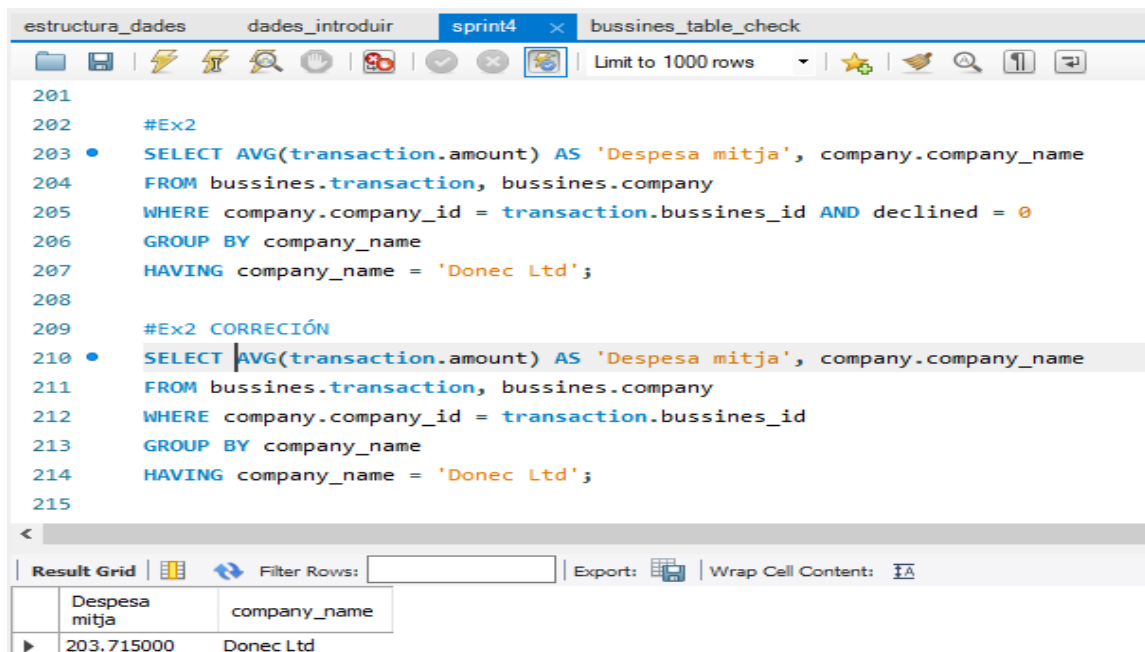
```
186
187 #NIVELL 1
188 #Ex1
189 • SELECT user.id, COUNT(transaction.id)
190 FROM user, transaction
191 WHERE transaction.user_id = user.id AND declined = 0
192 GROUP BY user.id
193 HAVING COUNT(transaction.id) > 30;
194
195 #Ex1 CORRECCIÓ
196 • SELECT user.id, COUNT(transaction.id)
197 FROM user, transaction
198 WHERE transaction.user_id = user.id
199 GROUP BY user.id
200 HAVING COUNT(transaction.id) > 30;
```

	id	COUNT(transaction.id)
▶	275	48
	272	76
	267	52
	92	39

*Se ha añadido la corrección sin la condición de 'declined = 0'.

Ex2

Con la fórmula **AVG** y la cláusula **HAVING** para establecer que el nombre de la compañía sea 'Donec Ltd' podemos extraer la información.



```
201
202 #Ex2
203 • SELECT AVG(transaction.amount) AS 'Despesa mitja', company.company_name
204 FROM bussines.transaction, bussines.company
205 WHERE company.company_id = transaction.bussines_id AND declined = 0
206 GROUP BY company_name
207 HAVING company_name = 'Donec Ltd';
208
209 #Ex2 CORRECCIÓ
210 • SELECT AVG(transaction.amount) AS 'Despesa mitja', company.company_name
211 FROM bussines.transaction, bussines.company
212 WHERE company.company_id = transaction.bussines_id
213 GROUP BY company_name
214 HAVING company_name = 'Donec Ltd';
215
```

	Despesa mitja	company_name
▶	203.715000	Donec Ltd

*Se ha añadido la corrección sin la condición de 'declined = 0'.

NIVELL 2

Ex1

Este query va a ser un poco más complejo.

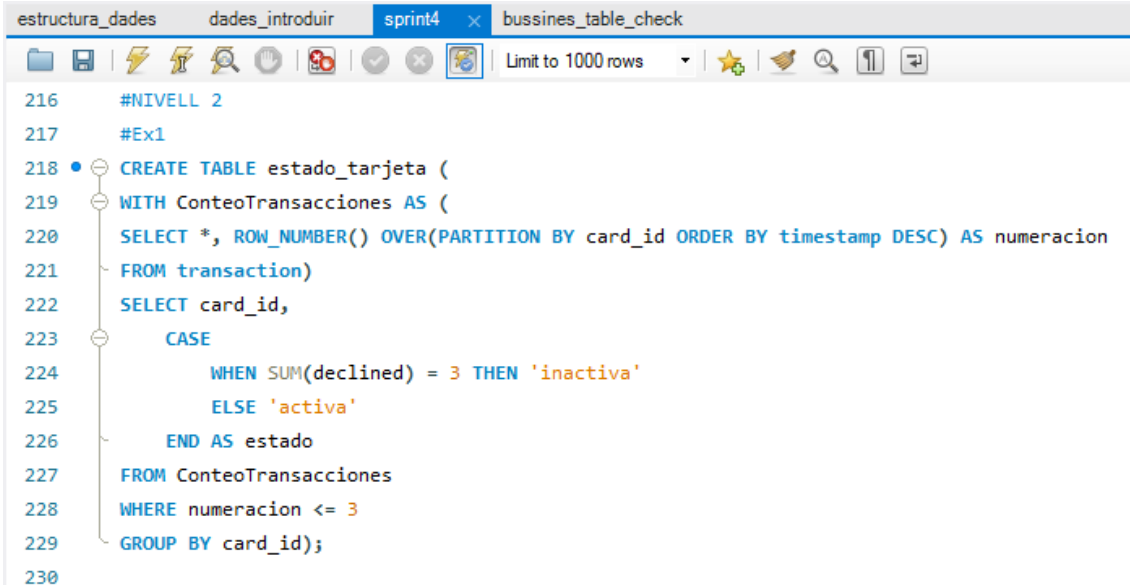
Cuando aparece una columna condicional como en este caso sabemos que probablemente vamos a tener que utilizar el **CASE**. En este caso la condición es que las ultimas 3 transacciones hayan sido declinadas, de este modo sabremos que la tarjeta esta inactiva.

Para extraer las últimas 3 transacciones vamos a tener que numerar las *rows* de la tabla *transaction*. Para ello vamos a usar la cláusula **ROW_NUMBER()**, y esta numeración la vamos a hacer sobre (**OVER**) la columna *card_id*, ordenándola por *timestamp*, para que nos la ordene temporalmente.

La query finalmente va a ser de la siguiente forma:

Primero utilizaremos **ROW_NUMBER** para numerar y ordenar temporalmente por *card_id* las transacciones. A esto lo llamaremos *numeració*, y servirá para establecer nuestra condición (últimas 3 transacciones). Con esto haremos un **SELECT** de las *card_id* utilizando un **CASE** donde la condición será que la **SUM(declined)** sea menor que 3 (ya que para que una tarjeta este inactiva esta tiene que tener las últimas tres transacciones declinadas). Finalmente, con el **WHERE** estableceremos que *numeració* <= 3, para quedarnos solo con las 3 últimas transacciones.

Todo esto irá dentro de un **CREATE TABLE** para que nos lo transforme todo en una tabla, a la cual llamaremos *estado_tarjeta*.



```
216 #NIVELL 2
217 #Ex1
218 CREATE TABLE estado_tarjeta (
219 WITH ConteoTransacciones AS (
220 SELECT *, ROW_NUMBER() OVER(PARTITION BY card_id ORDER BY timestamp DESC) AS numeracion
221 FROM transaction)
222 SELECT card_id,
223 CASE
224 WHEN SUM(declined) = 3 THEN 'inactiva'
225 ELSE 'activa'
226 END AS estado
227 FROM ConteoTransacciones
228 WHERE numeracion <= 3
229 GROUP BY card_id);
230
```







La tabla resultante tiene este formato:

Result Grid			Filter Rows:
	card_id	estado	
▶	CcU-2938	activa	
	CcU-2945	activa	
	CcU-2952	activa	
	CcU-2959	activa	
	CcU-2966	activa	
	CcU-2973	activa	
	CcU-2980	activa	
	CcU-2987	activa	
	CcU-2994	activa	
	CcU-3001	activa	
	CcU-3008	activa	
	CcU-3015	activa	
	CcU-3022	activa	
	CcU-3029	activa	
	CcU-3036	activa	
	CcU-3043	activa	
	CcU-3050	activa	





estado_tarjeta 1 x

Con un **COUNT** podemos saber cuántas tarjetas activas tenemos.

estructura_dades dades_introduir **sprint4** x bussines_table_check

 Limit to 1000 rows     

230
231 • **SELECT** COUNT(estado) AS 'numero de tarjetas activas'
232 **FROM** estado_tarjeta
233 **WHERE** estado = 'activa';
234

Result Grid   Filter Rows: Export:  Wrap Cell Content: 

	numero de tarjetas activas
▶	275

NIVELL 3

Ex1

Gracias a la *junction table* que hemos creado antes (*transaction_product*) la query necesaria para saber cuántos productos hemos vendido es un simple **COUNT** agrupado por *product_id*.

The screenshot shows a database IDE with a toolbar at the top containing icons for file operations, execution, and search. Below the toolbar, there are tabs for 'estructura_dades', 'dades_introduir', 'sprint4', and 'bussines_table_check'. The 'sprint4' tab is active, displaying two SQL queries. The first query (lines 235-240) counts products by ID. The second query (lines 241-245), labeled '#Ex1 CORRECCIÓ', joins the 'transaction_product' and 'product' tables to show product names. Below the queries is a 'Result Grid' with columns 'COUNT(product_id)', 'product_id', and 'product_name'. It contains 20 rows of data, including product names like 'Direwolf Stannis', 'Tully Stark', and 'Winterfell Lannister'. Rows with IDs 41 and 53 show 'NULL' for the product name. At the bottom, a tab labeled 'Result 10' is visible.

```
235 #NIVELL 3
236 #Ex1
237 • SELECT COUNT(product_id), product_id
238 FROM transaction_product
239 GROUP BY product_id;
240
241 #Ex1 CORRECCIÓ
242 • SELECT COUNT(product_id), product_id, product_name
243 FROM transaction_product, product
244 WHERE transaction_product.product_id = product.id
245 GROUP BY product_id, product_name;
```

	COUNT(product_id)	product_id	product_name
▶	61	1	Direwolf Stannis
	65	2	Tarly Stark
	51	3	duel tourney Lannister
	49	5	skywalker ewok
	54	7	north of Casterly
	48	11	Karstark Dorne
	60	13	palpatine chewbacca
	61	17	skywalker ewok sith
	49	19	dooku solo
	68	23	riverlands north
	49	29	Tully maester Tarly
	47	31	Lannister
	51	37	Direwolf Littlefinger
	53	41	NULL
	65	43	duel
	62	47	Tully
	58	53	NULL
	45	59	Direwolf Stannis
	57	61	Winterfell Lannister
	68	67	Winterfell
	54	71	Tully Dorne
	47	73	Dorne bastard
	66	79	Direwolf riverlands the

*Se ha añadido una corrección para mostrar el nombre del producto en la misma *query*. Los productos con *id* 41 y 53 no tienen nombre porque son los que hemos añadido nosotros para poder establecer las *FOREIGN KEYS* entre las tablas *transaction_product* y *product*, ya que los registros 41 y 53 existían en la tabla *transaction* pero no en la *product*.