

Autora: Giorgia Calvagna

Fecha de envío: 03/01/2025

Revisado por: Raquel Limpo Martínez

Tasca S3 - Manipulació de taules

Nivel 1, Ejercicio 1

- Tu tarea es diseñar y crear una tabla llamada "credit_card" que almacene detalles cruciales sobre las tarjetas de crédito. La nueva tabla tiene que ser capaz de identificar de manera única cada tarjeta y establecer una relación adecuada con las otras dos tablas ("transaction" y "company"). Después de crear la tabla será necesario que ingreses la información del documento denominado "*datos_introducir_credit*". Recuerda mostrar el diagrama y realizar una breve descripción de este.

Comenzamos creando la nueva tabla *credit_card* en la base de datos de *transactions*.

Podemos crear esta nueva tabla escribiendo el código a través del comando CREATE TABLE IF NOT EXISTS:

```
4 • USE transactions;
5
6 • CREATE TABLE IF NOT EXISTS credit_card (
7     id VARCHAR(15) PRIMARY KEY,
8     iban VARCHAR(40) NOT NULL,
9     pan VARCHAR(45) NOT NULL,
10    pin INT NOT NULL,
11    cvv INT NOT NULL,
12    expiring_date VARCHAR(20) NOT NULL
13 );
14
```

Output			
Action Output			
#	Time	Action	Message
✓ 1	13:36:50	USE transactions	0 row(s) affected
✓ 2	13:36:50	CREATE TABLE IF NOT EXISTS credit_card (id VARCHAR(15) PRIMARY KEY, iban VARCH...	0 row(s) affected

En función de los datos que almacenemos, debemos elegir el tipo de datos adecuado para optimizar y ahorrar espacio. En este caso, se eligieron los datos INT y VARCHAR.

El tipo de dato VARCHAR se utiliza para campos que contienen texto o datos alfanuméricos, es decir, valores que pueden incluir letras, números o incluso símbolos especiales. Aunque a veces un campo pueda parecer numérico, como un ID o un número de cuenta, usamos VARCHAR en estos casos porque:

1. El valor puede contener letras o símbolos además de números (por ejemplo, un IBAN que puede incluir letras).
2. La longitud es variable, lo que significa que no todos los valores ocuparán la misma cantidad de espacio.
3. No queremos que el valor se trate matemáticamente. Por ejemplo, un ID que consiste en números no debe sumarse ni restarse, ya que es un identificador único, no un número en sentido matemático.

Por otro lado, el tipo de dato INT se utiliza para valores estrictamente numéricos, es decir, números enteros. En nuestra tabla, usamos INT para campos como el PIN y el CVV porque:

1. Estos valores son puramente numéricos y no contienen letras ni símbolos.
2. El tipo INT es más eficiente para almacenar números, ya que ocupa menos espacio que un VARCHAR.
3. Campos como el PIN (normalmente 4 dígitos) o el CVV (3 o 4 dígitos) tienen una longitud limitada y fija, por lo que no necesitamos un campo de texto flexible, aunque usar VARCHAR asegura que el formato del PIN se mantenga tal como fue ingresado, preservando cualquier cero inicial y evitando errores al comparar o mostrar los datos.

Por lo tanto, INT es la opción ideal para estos campos, ya que optimiza el almacenamiento y garantiza que solo se ingresen valores numéricos.

Se ha elegido especificar NOT NULL en las columnas para garantizar que ningún dato importante quede vacío, especialmente en campos críticos como el IBAN, PAN, PIN o CVV. Sin embargo, también podría haberse utilizado la opción DEFAULT para asignar valores por defecto en caso de que no se proporcionen datos al momento de insertar un registro. Esto ofrecería mayor flexibilidad, sobre todo si más adelante se

planea cambiar estos campos para establecer valores predeterminados según las necesidades del sistema.






Si hiciera falta, podemos editar la información de la tabla mediante la función ALTER TABLE para añadir columnas o cambiar tipo de datos.

Ahora que la tabla *credit_card* ha sido creada, podemos cargar los valores utilizando el archivo *datos_introducir_credit*:

```
1  |
2  -- Insertamos datos de credit_card
3  • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2938', 'TR301950312213576817638661', '5424465566813633', '3257', '984', '10/30/22');
4  • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2945', 'DO26854763748537475216568689', '5142423821948828', '9080', '887', '08/24/23');
5  • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2952', 'BG45IVQL52710525608255', '4556 453 55 5287', '4598', '438', '06/29/21');
6  • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2959', 'CR7242477244335841535', '372461377349375', '3583', '667', '02/24/23');
7  • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2966', 'BG72LKTQ15627628377363', '448566 886747 7265', '4900', '130', '10/29/24');
8  • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2973', 'PT87806228135092429456346', '544 58654 54343 384', '8760', '887', '01/30/25');
9  • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2980', 'DE39241881883086277136', '402400 7145845969', '5075', '596', '07/24/22');
10 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2987', 'GE89681434837748781813', '3763 747687 76666', '2298', '797', '10/31/23');
11 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2994', 'BH62714428368066765294', '344283273252593', '7545', '595', '02/28/22');
12 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3001', 'CY49087426654774581266832110', '511722 924833 2244', '9562', '867', '09/16/22');
13 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3008', 'LU507216693616119230', '4485744464433884', '1856', '740', '04/05/25');
14 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3015', 'PS119398216295715968342456821', '3784 662233 17389', '3246', '822', '01/31/22');
15 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3022', 'GT9169516280556977423121857', '5164 1379 4842 3951', '5610', '342', '04/25/25');
```

Una vez cargados todos los datos de la tabla *credit_card*, podemos seleccionar todos los valores para verificar si hay errores:

```
15 • select *
16   from credit_card;
17 |
```

Result Grid						
Filter Rows: <input type="text"/>						
Edit:    Export/Imports:   Wrap Cell Content: <input checked="" type="checkbox"/>						
	id	iban	pan	pin	cvv	expiring_date
▶	CcU-2938	TR301950312213576817638661	5424465566813633	3257	984	10/30/22
	CcU-2945	DO26854763748537475216568689	5142423821948828	9080	887	08/24/23
	CcU-2952	BG45IVQL52710525608255	4556 453 55 5287	4598	438	06/29/21
	CcU-2959	CR7242477244335841535	372461377349375	3583	667	02/24/23
	CcU-2966	BG72LKTQ15627628377363	448566 886747 7265	4900	130	10/29/24
	CcU-2973	PT87806228135092429456346	544 58654 54343 384	8760	887	01/30/25
	CcU-2980	DE39241881883086277136	402400 7145845969	5075	596	07/24/22
	CcU-2987	GE89681434837748781813	3763 747687 76666	2298	797	10/31/23
	CcU-2994	BH62714428368066765294	344283273252593	7545	595	02/28/22

credit_card 3 x

Output:

Action Output

#	Time	Action	Message
✓ 1	18:49:02	select * from credit_card LIMIT 0, 1000	275 row(s) returned

La consulta nos devuelve 275 resultados y coloca correctamente los valores bajo las columnas que hemos creado.

La tabla de dimensión *credit_card* se relaciona con la tabla de hecho *transaction* a través de la primary key *id* que aparece como foreign key en la tabla *transaction* bajo el nombre *credit_card_id*.

Para que en la estructura de datos de la tabla de hechos *transaction* se registre la segunda clave foránea que hace referencia a la tabla *credit_card*, debemos ejecutar el siguiente comando:

```
ALTER TABLE transaction
ADD CONSTRAINT fk_credit_card_id
FOREIGN KEY (credit_card_id)
REFERENCES credit_card(id);
```

Gracias a este comando, hemos creado una conexión entre la nueva tabla de dimensiones *credit_card* y la tabla de hechos *transaction*, que ya estaba conectada a la tabla *company* a través de la clave foránea *company_id*:

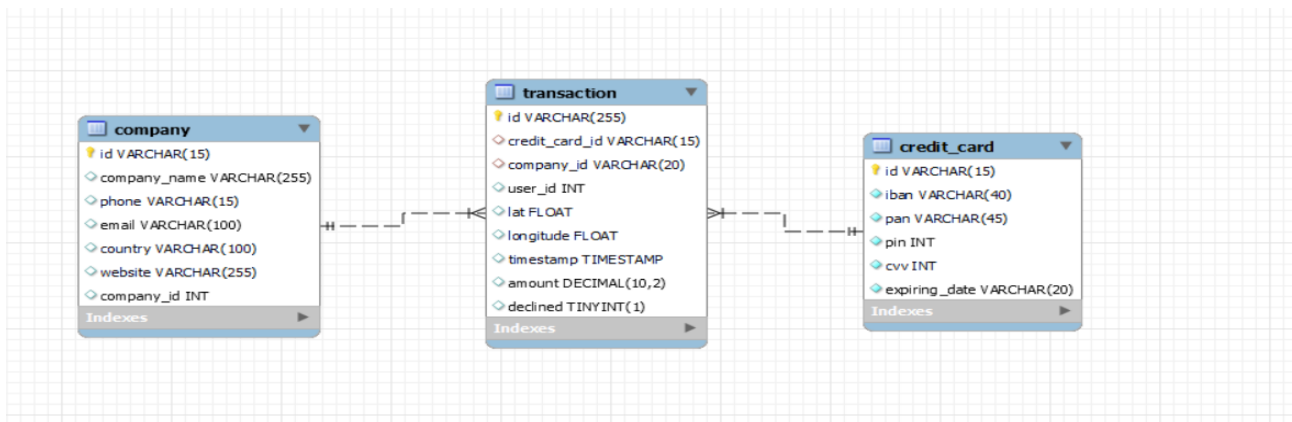
```
-- Creamos la tabla transaction
CREATE TABLE IF NOT EXISTS transaction (
  id VARCHAR(255) PRIMARY KEY,
  credit_card_id VARCHAR(15) REFERENCES credit_card(id),
  company_id VARCHAR(20),
  user_id INT REFERENCES user(id),
  lat FLOAT,
  longitude FLOAT,
  timestamp TIMESTAMP,
  amount DECIMAL(10, 2),
  declined BOOLEAN,
  FOREIGN KEY (company_id) REFERENCES company(id)
);
```

Se decide crear un índice sobre la columna *credit_card_id* en la tabla *transaction* para optimizar las consultas que utilizan esta columna como criterio de búsqueda o filtrado. Esto mejora significativamente el rendimiento del sistema, especialmente en bases de datos grandes donde se realizan muchas operaciones de lectura. El índice permite acceder más rápidamente a los registros relacionados con una tarjeta de crédito específica, reduciendo el tiempo de ejecución de las consultas y mejorando la eficiencia global del sistema.

```
29 • CREATE INDEX idx_credit_card
30 ON transaction(credit_card_id);
31
```

Output			
Action Output			
#	Time	Action	Message
1	16:03:11	CREATE INDEX idx_credit_card ON transaction(credit_card_id)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
			Duration / Fetch 0.141 sec

Ahora que hemos ejecutado las conexiones entre la tabla *credit_card* y la tabla *transaction*, a través de Reverse Engineer de MySQL Workbench podemos visualizar el diagrama actualizado:

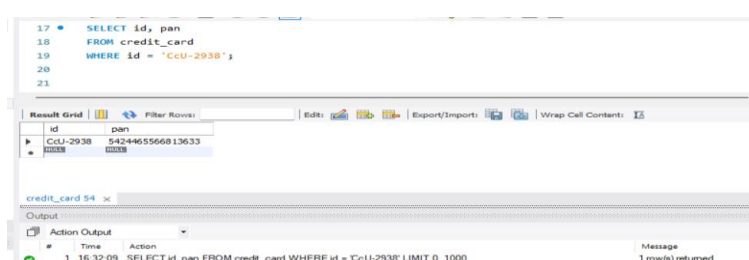


La tabla de hechos *transaction* está relacionada con la tabla *company* a través de la clave foránea *company_id*, lo que indica que cada transacción está asociada a una empresa específica. Además, la tabla *transaction* está conectada con la tabla *credit_card* mediante la clave foránea *credit_card_id*, lo que significa que cada transacción está asociada a una tarjeta de crédito. En ambas relaciones, la relación es de tipo uno a muchos: una empresa puede tener múltiples transacciones, y una tarjeta de crédito también puede estar asociada a múltiples transacciones.

Nivel 1, Ejercicio 2

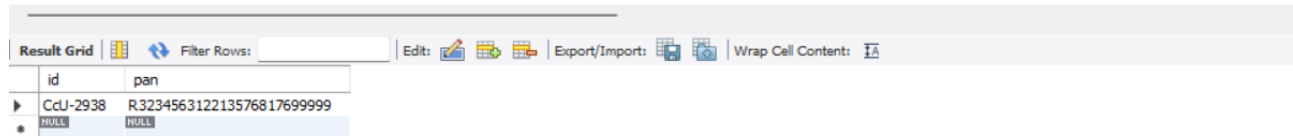
- El departamento de Recursos Humanos ha identificado un error en el número de cuenta del usuario con ID CcU-2938. La información que tiene que mostrarse para este registro es: R323456312213576817699999. Recuerda mostrar que el cambio se realizó:

Antes de realizar la modificación, se consulta el valor actual del número de cuenta correspondiente al ID CcU-2938 para poder asegurarse, en una fase posterior, de que el cambio se ejecute correctamente:

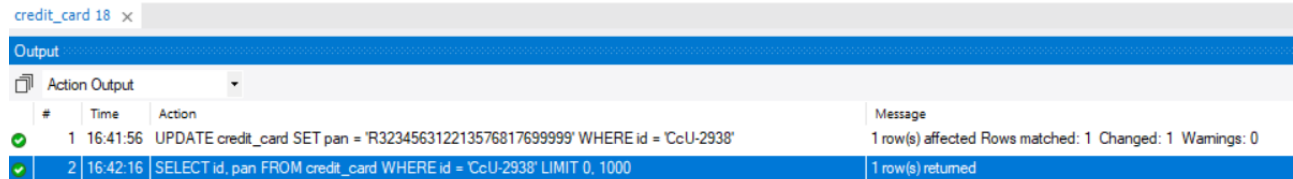


Luego, se procede a actualizar el valor utilizando el comando UPDATE con SET para cambiar el número de cuenta:

```
36 • UPDATE credit_card
37   SET pan = 'R323456312213576817699999'
38   WHERE id = 'CcU-2938';
39
40 • SELECT id, pan
41   FROM credit_card
42   WHERE id = 'CcU-2938';
```



id	pan
CcU-2938	R323456312213576817699999



#	Time	Action	Message
1	16:41:56	UPDATE credit_card SET pan = 'R323456312213576817699999' WHERE id = 'CcU-2938'	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0
2	16:42:16	SELECT id, pan FROM credit_card WHERE id = 'CcU-2938' LIMIT 0, 1000	1 row(s) returned

Se utilizó el comando UPDATE para corregir el número de cuenta del ID CcU-2938. Dentro del comando, utilizamos SET para asignar el nuevo valor R323456312213576817699999 al campo *pan*. La cláusula SET es fundamental porque nos permite especificar qué columna queremos modificar y qué valor debe tener. Para asegurarnos de que solo se actualizara el registro del usuario específico, utilizamos WHERE para establecer la condición que filtra el registro por el *id* correspondiente. De esta forma, evitamos modificar accidentalmente otros registros. Finalmente, con una consulta SELECT, verificamos que el cambio se haya realizado correctamente.

Este proceso es clave para actualizar un dato de manera precisa sin afectar a otros registros, y el uso de SET es esencial para definir qué columna y qué valor se deben cambiar.

Nivel 1, Ejercicio 3

- En la tabla "transaction" ingresa un nuevo usuario con la siguiente información:

id	108B1D1D-5B23-A76C-55EF-C568E49A99DD
credit_card_id	CcU-9999
company_id	b-9999
user_id	9999
lat	829.999
longitude	-117.999
amount	111.11
declined	0

Si intentamos ingresar este nuevo usuario, MySQL enseñará el siguiente error:

```
47
48 • INSERT INTO transaction (id, credit_card_id, company_id, user_id, lat, longitude, amount, declined)
49 VALUES ('108B1D1D-5B23-A76C-55EF-C568E49A99DD', 'CcU-9999', 'b-9999', '9999', '829.999', '-117.999', '111.11', '0');
50
51
52
```

Output

#	Time	Action	Message	Duration / Fetch
1	15:10:32	INSERT INTO transac	Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('transactions', 'transaction', CONSTRAINT 'transaction_ibfk_1' FOREIGN KEY ('company_id') REFERENCES 'company' ('id'))	

El error que se está observando (Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails) indica que se está intentando insertar un valor en una columna que tiene una clave foránea, pero el valor que se está intentando insertar no existe en la tabla de referencia.

En este caso, el error se refiere al campo *company_id*, que tiene una clave foránea vinculada a la tabla *company(id)*. El valor 'b-9999' que se está intentando insertar como *company_id* en la tabla *transaction* no existe en la tabla *company*.

Por esta razón, insertamos este nuevo valor de *id* en la tabla *company*:

```
51 • INSERT INTO company (id) VALUES ('b-9999');
52
```

Output

#	Time	Action	Message	Duration / Fetch
1	15:10:32	INSERT INTO transaction (id, credit_card_id, company_id, user_id, lat, longitude, amount, declined) VALUES ...	Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails ('transactions', 'transaction', C...	0.047 sec
2	15:19:46	INSERT INTO company (id) VALUES ('b-9999')	1 row(s) affected	0.063 sec

Para que no vuelva a aparecer el mismo error, tenemos que hacer lo mismo también con la tabla *credit_card*. Así que insertamos el nuevo valor *id* también en la tabla *credit_card*:

```
56 • INSERT INTO credit_card (id) VALUES ('CcU-9999');
57
58 • ALTER TABLE credit_card
59 MODIFY COLUMN iban VARCHAR(40) DEFAULT NULL;
60
61
```

Output				
Action Output				
#	Time	Action	Message	
✖ 1	15:28:44	INSERT INTO credit_card (id) VALUES ('CcU-9999')	Error Code: 1364. Field 'iban' doesn't have a default value	
✔ 2	16:02:46	ALTER TABLE credit_card MODIFY COLUMN iban VARCHAR(40) DEFAULT NULL	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	

Dado que, tras el intento de insertar el dato, se señala un error que indica que el campo *id* no tiene un valor por defecto (default), realizamos la modificación necesaria para que el valor *id* pueda registrarse correctamente.

Para evitar el mismo error, además de para el campo *iban*, extendemos la modificación a todos los campos de la tabla *credit card*, utilizando *DEFAULT NULL* y cambiando a *VARCHAR* todos los tipos de datos:

```
ALTER TABLE credit_card
MODIFY COLUMN iban VARCHAR(40) DEFAULT NULL,
MODIFY COLUMN pan VARCHAR(45) DEFAULT NULL,
MODIFY COLUMN pin VARCHAR(10) DEFAULT NULL,
MODIFY COLUMN cvv VARCHAR(5) DEFAULT NULL,
MODIFY COLUMN expiring_date VARCHAR(20) DEFAULT NULL;
```

Una vez realizada esta modificación, podremos insertar el nuevo valor en la columna *id* de la tabla *credit_card*.

Una vez insertado este dato, podremos ingresar la información requerida por el ejercicio sin encontrarnos con el mencionado error 1452:

```
70 • INSERT INTO credit_card (id) VALUES ('CcU-9999');
71
72 • INSERT INTO transaction (id, credit_card_id, company_id, user_id, lat, longitude, amount, declined)
73 VALUES ('10881D1D-5B23-A76C-55EF-C568E49A990D', 'CcU-9999', 'b-9999', '9999', '829.999', '-117.999', '111.11', '0');
74
```

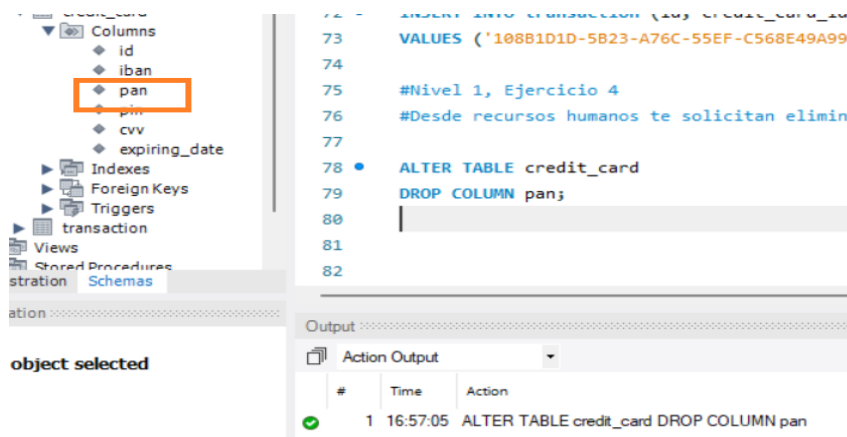
Output				
Action Output				
#	Time	Action	Message	Duration / Fetch
✔ 1	16:27:44	INSERT INTO credit_card (id) VALUES ('CcU-9999')	1 row(s) affected	0.015 sec
✔ 2	16:27:55	INSERT INTO transaction (id, credit_card_id, company_id, user_id, lat, longitude, amount, declined) VALUES ('1...	1 row(s) affected	0.062 sec

Nivel 1, Ejercicio 4

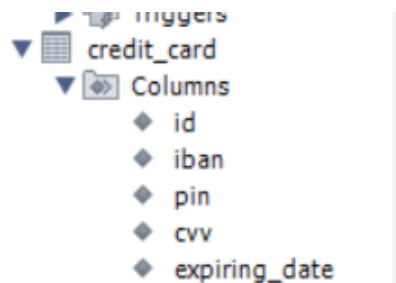
- Desde recursos humanos te solicitan eliminar la columna "pan" de la tabla *credit_card*. Recuerda mostrar el cambio realizado:

Recursos Humanos solicitó eliminar la columna *pan* porque, probablemente, las informaciones son obsoletas o no requeridas.

Usamos ALTER TABLE junto con DROP COLUMN para eliminar una columna porque esta es la forma estándar y más directa en SQL para modificar una tabla existente. Esta instrucción permite eliminar una columna específica sin afectar los demás datos de la tabla.



Si vamos a actualizar el *schema*, podemos averiguar que el campo *pan* ha sido eliminado:



Para comprobarlo de otra forma, utilizamos SELECT * para llamar a todos los campos de la tabla *credit_card* y verificar que el campo *pan* no esté presente:

```

80
81 • SELECT *
82 FROM credit_card;
83

```

id	iban	pin	cvv	expiring_date
CcU-2938	TR301950312213576817638661	3257	984	10/30/22
CcU-2945	DO26854763748537475216568689	9080	887	08/24/23
CcU-2952	BG45IVQL52710525608255	4598	438	06/29/21
CcU-2959	CR7242477244335841535	3583	667	02/24/23
CcU-2966	BG72LKTQ15627628377363	4900	130	10/29/24
CcU-2973	PT87806228135092429456346	8760	887	01/30/25
CcU-2980	DE30741881883086777136	5075	506	07/24/22

credit_card 1 x

Output

#	Time	Action	Message
1	16:57:05	ALTER TABLE credit_card DROP COLUMN pin	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
2	17:00:30	SELECT * FROM credit_card LIMIT 0, 1000	276 row(s) returned

Nivel 2, Ejercicio 1

- Elimina de la tabla *transaction* el registro con ID 02C6201E-D90A-1859-B4EE-88D2986D3B02 de la base de datos:

Antes de eliminar el dato, es útil verificar si el registro con el ID especificado existe en la tabla *transaction*; esto se hace utilizando una consulta SELECT.

```

3
4 • SELECT id
5 FROM transaction
6 WHERE id = '02C6201E-D90A-1859-B4EE-88D2986D3B02';

```

id
02C6201E-D90A-1859-B4EE-88D2986D3B02
NULL

transaction 1 x

Output

#	Time	Action	Message
1	17:22:44	SELECT id FROM transaction WHERE id = '02C6201E-D90A-1859-B4EE-88D2986D3B02' LIMIT 0, 1000	1 row(s) returned

Una vez confirmado que el registro existe, se utiliza el comando DELETE junto con una cláusula WHERE para indicar específicamente cuál registro se desea eliminar. En este caso, se elimina el registro cuyo ID es '02C6201E-D90A-1859-B4EE-88D2986D3B02':

```
8 • DELETE FROM transaction
9 WHERE id = '02C6201E-D90A-1859-B4EE-88D2986D3B02';
10
```

#	Time	Action	Message	Duration / Fetch
1	17:38:55	DELETE FROM transaction WHERE id = '02C6201E-D90A-1859-B4EE-88D2986D3B02'	1 row(s) affected	0.062 sec

Después de ejecutar el comando, se realiza una nueva consulta SELECT para comprobar que el registro ya no está en la tabla. Si la consulta no devuelve resultados, se puede confirmar que el proceso de eliminación se completó con éxito:

```
11 • SELECT id
12 FROM transaction
13 WHERE id = '02C6201E-D90A-1859-B4EE-88D2986D3B02';
```

#	Time	Action	Message	Duration / Fetch
1	17:38:55	DELETE FROM transaction WHERE id = '02C6201E-D90A-1859-B4EE-88D2986D3B02'	1 row(s) affected	0.062 sec
2	17:42:54	SELECT id FROM transaction WHERE id = '02C6201E-D90A-1859-B4EE-88D2986D3B02' LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

Nivel 2, Ejercicio 2

- La sección de marketing desea tener acceso a información específica para realizar análisis y estrategias efectivas. Se ha solicitado crear una vista que proporcione detalles clave sobre las compañías y sus transacciones. Será necesaria que crees una vista llamada VistaMarketing que contenga la siguiente información: nombre de la compañía, teléfono de contacto, país de residencia, media de compra realizado por cada compañía. Presenta la vista creada, ordenando los datos de mayor a menor media de compra:

Para crear la vista utilizamos el comando CREATE VIEW y le damos el nombre de VistaMarketing:

```
CREATE VIEW VistaMarketing AS
```

Para realizar la consulta que satisface las necesidades del equipo de marketing, primero se identifican las tablas necesarias. En este caso, se trabaja con las tablas *company* y *transaction*. Luego, se seleccionan las columnas clave que se deben incluir en la vista: el nombre de la compañía, el teléfono de contacto, el país de residencia y la media de las compras asociadas a cada compañía. Puede ser útil añadir el id de la compañía para que posteriormente se pueda realizar una agrupación a través de este dato:

Se utiliza una función de agregación, en este caso AVG(), para calcular la media de los valores en la columna *amount* dentro de la tabla *transaction*. A través de ROUND se redondea el promedio calculado a dos cifras decimales.

Se realiza una unión (JOIN) entre las tablas *company* y *transaction* utilizando el campo en común *company_id*, lo que permite conectar las transacciones con sus respectivas compañías y se da un alias a las tablas para que el código sea más legible.

Para evitar errores, se asegura que todas las demás columnas que no son calculadas estén incluidas en una cláusula GROUP BY.

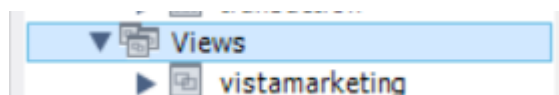
Se agrega una ordenación de los resultados de mayor a menor utilizando la columna calculada de la media, lo que facilita el análisis para el equipo de marketing.

```
104 • CREATE VIEW VistaMarketing AS
105 SELECT c.id, c.company_name, c.phone, c.country, ROUND(avg(t.amount), 2) as avg_amount
106 FROM company c
107 JOIN transaction t
108 ON c.id = t.company_id
109 GROUP BY c.id, c.company_name, c.phone, c.country
110 ORDER BY avg_amount DESC;
111
```

Output

#	Time	Action	Message	Duration / Fetch
1	16:22:09	CREATE VIEW VistaMarketing AS SELECT c.id, c.company_name, c.phone, c.country, ROUND(avg(t.amoun...	0 row(s) affected	0.047 sec

Después de actualizar el schema, saldrá el nombre de la vista que se acaba de crear:



A través del comando SELECT se puede llamar a la pantalla los datos guardados en la view *VistaMarketing*:

```
114 • SELECT * FROM VistaMarketing;
115
```

id	company_name	phone	country	avg_amount
b-2398	Eget Ipsum Ltd	03 67 44 56 72	United States	473.08
b-2310	Non Magna LLC	06 71 73 13 17	United Kingdom	468.35
b-2410	Sed Id Limited	07 28 18 18 13	United States	461.21
b-2454	Justo Eu Arcu Ltd	08 42 56 71 52	Italy	443.64
b-2458	Eget Tincidunt Dui Institute	05 35 93 32 44	Netherlands	442.52
b-2522	Viverra Donec Foundation	03 33 12 32 73	United Kingdom	442.28

Output: Action Output

#	Time	Action	Message
1	16:35:39	SELECT * FROM VistaMarketing LIMIT 0, 1000	101 row(s) returned

Nivel 2, Ejercicio 3

- Filtra la vista VistaMarketing para mostrar solo las compañías que tienen su país de residencia en "Germany":

```
32 • SELECT *
33 FROM VistaMarketing
34 WHERE country = 'Germany';
```

id	company_name	phone	country	avg_amount
b-2566	Aliquam PC	01 45 73 52 16	Germany	385.265000
b-2358	Ac Industries	09 34 65 40 60	Germany	289.645000
b-2614	Rutrum Non Inc.	02 66 31 61 09	Germany	266.900000
b-2302	Nunc Interdum Incorporated	05 18 15 48 13	Germany	244.025238
b-2306	Augue Foundation	06 88 43 15 63	Germany	240.800000
b-2222	Ac Fermentum Incorporated	06 85 56 52 33	Germany	206.465000

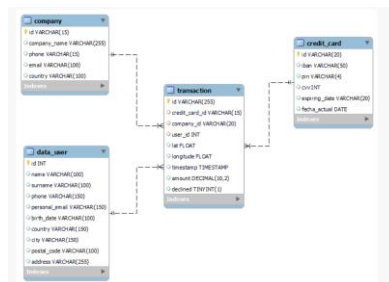
Output: Action Output

#	Time	Action	Message	Duration / Fetch
1	20:10:56	SELECT * FROM VistaMarketing LIMIT 0, 1000	101 row(s) returned	0.015 sec / 0.000 sec
2	20:19:59	SELECT * FROM VistaMarketing WHERE country = 'Germany' LIMIT 0, 1000	8 row(s) returned	0.000 sec / 0.000 sec

Se realiza una extracción de datos desde la vista *VistaMarketing* para obtener únicamente la información relacionada con las empresas alemanas. Se aplica un filtro mediante la cláusula `WHERE country = 'Germany'` que permite seleccionar exclusivamente los registros correspondientes a Alemania, descartando los datos de otros países presentes en la vista.

Nivel 3, Ejercicio 1

- La próxima semana tendrás una nueva reunión con los gerentes de marketing. Un compañero de tu equipo realizó modificaciones en la base de datos, pero no recuerda como las realizó. Te pide que lo ayudes a dejar los comandos ejecutados para obtener el siguiente diagrama:



En esta actividad, es necesario que describas el "paso a paso" de las tareas realizadas. Es importante realizar descripciones sencillas, simples y fáciles de comprender. Para realizar esta actividad tendrás que trabajar con los archivos denominados "estructura_datos_user" y "datos_introducir_user":

Siguiendo las indicaciones del diagrama compartido, comenzamos añadiendo la columna faltante *fecha_actual* a la tabla *credit_card* para que quede completa:

```
45 ALTER TABLE credit_card
46 ADD COLUMN fecha_actual DATE DEFAULT NULL;
47
```

#	Time	Action	Message	Dura
1	16:29:28	ALTER TABLE credit_card ADD COLUMN fecha_actual DATE DEFAULT NULL	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.03'

Para realizar esta operación, utilizamos el comando ALTER TABLE seguido del nombre de la tabla *credit_card*. Luego, añadimos la nueva columna con ADD COLUMN especificando su nombre, *fecha_actual*, su tipo de dato, DATE, y el valor por defecto, que será NULL. Esto permite incluir la columna sin afectar los datos existentes en la tabla.

Para garantizar la correspondencia con el diagrama compartido, se cambia el tipo de dato de la columna *pin* en la tabla *credit_card*, pasando a VARCHAR(4). Una vez realizada la modificación, se verifica utilizando el comando DESCRIBE para confirmar que el cambio se haya aplicado correctamente:

```
47
48 • ALTER TABLE credit_card
49   MODIFY COLUMN pin VARCHAR(4);
50
51 • DESCRIBE credit_card;
```

Field	Type	Null	Key	Default	Extra
id	varchar(15)	NO	PRI	NULL	
iban	varchar(40)	YES		NULL	
pin	varchar(4)	YES		NULL	
cvv	varchar(5)	YES		NULL	
expiring_date	varchar(20)	YES		NULL	
fecha_actual	date	YES		NULL	

Result 1 x

Output

#	Time	Action	Message
1	16:29:28	ALTER TABLE credit_card ADD COLUMN fecha_actual DATE DEFAULT NULL	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
2	16:40:52	ALTER TABLE credit_card MODIFY COLUMN pin VARCHAR(4)	276 row(s) affected Records: 276 Duplicates: 0 Warnings: 0
3	16:41:33	DESCRIBE credit_card	6 row(s) returned

Una vez completada la tabla *credit_card*, para asegurar que las tablas sean coherentes con el diagrama proporcionado, se decide eliminar los campos *website* y *company_id* de la tabla *company*, ya que no están reflejados en la estructura del diagrama que debemos implementar:

```
• ALTER TABLE company
  DROP COLUMN website,
  DROP COLUMN company_id;
```

Esta instrucción usa el comando ALTER TABLE para modificar la estructura de la tabla *company*, eliminando las columnas mencionadas con DROP COLUMN. Esto asegura que la tabla tenga únicamente los campos requeridos según el diseño del diagrama.

Ahora podemos crear la tabla *data_user* utilizando las indicaciones del archivo *estructura_datos_user*:

```
5 CREATE TABLE IF NOT EXISTS user (  
6     id INT PRIMARY KEY,  
7     name VARCHAR(100),  
8     surname VARCHAR(100),  
9     phone VARCHAR(150),  
10    email VARCHAR(150),  
11    birth_date VARCHAR(100),  
12    country VARCHAR(150),  
13    city VARCHAR(150),  
14    postal_code VARCHAR(100),  
15    address VARCHAR(255),  
16    FOREIGN KEY(id) REFERENCES transaction(user_id)  
17 );  
18  
19
```

Output

#	Time	Action	Message	Duration / Fets
1	17:21:16	CREATE TABLE IF NOT EXISTS user (id INT PRIMARY KEY, name VARCHAR(100), surname...	Error Code: 6125. Failed to add the foreign key constraint. Missing unique key for constraint 'user_ibfk_1' in the...	0.000 sec

El sistema devuelve el error 6125. Esto ocurre porque necesitamos asegurarnos de que la columna *id* de la tabla *data_user* tenga valores únicos en la columna *user_id* de la tabla *transaction*.

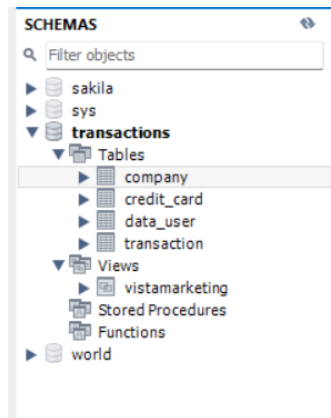
Por eso, vamos a crear la tabla sin indicar ahora la clave foránea, para insertarla en otro momento:

```
12 CREATE TABLE IF NOT EXISTS data_user (  
13     id INT PRIMARY KEY,  
14     name VARCHAR(100),  
15     surname VARCHAR(100),  
16     phone VARCHAR(150),  
17     email VARCHAR(150),  
18     birth_date VARCHAR(100),  
19     country VARCHAR(150),  
20     city VARCHAR(150),  
21     postal_code VARCHAR(100),  
22     address VARCHAR(255)  
23 );  
24  
25  
26
```

Output

#	Time	Action	Message
1	13:03:21	CREATE TABLE IF NOT EXISTS data_user (id INT PRIMARY KEY, name VARCHAR(100), surname ...	0 row(s) affected

Actualizamos el esquema para verificar que la tabla `data_user` haya sido creada:



Después de crear la estructura de la tabla *data_user* y antes de llenarla con los valores correspondientes, establecemos una conexión con la tabla de hechos *transaction*, definiendo la clave foránea que servirá para conectar ambas tablas:

```
24
25 • ALTER TABLE transaction
26 ADD CONSTRAINT fk_user_transaction
27 FOREIGN KEY (user_id) REFERENCES data_user(id);
28
29
```

Output :

Action Output

#	Time	Action	Message
1	15:17:20	ALTER TABLE transaction ADD CONSTRAINT fk_user_transaction FOREIGN KEY (user_id) REFERENCES data_user(id);	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Ahora, si intentamos insertar en la tabla *data_user* los valores del archivo *datos_introducir_user*, el sistema generará un error porque la tabla que hemos creado se llama *data_user*, mientras que en las sentencias INSERT INTO del archivo se hace referencia a *user*.

```
3      -- Insertamos datos de user
4  ●   INSERT INTO user (id, name, surname,
5  ●   INSERT INTO user (id, name, surname,
```

Por esta razón, para añadir a la tabla los valores que le corresponden, tenemos que reemplazar el nombre de la tabla en cada registro usando el comando CTRL + H para hacer buscar *INSERT INTO user* y reemplazar con *INSERT INTO data user*:

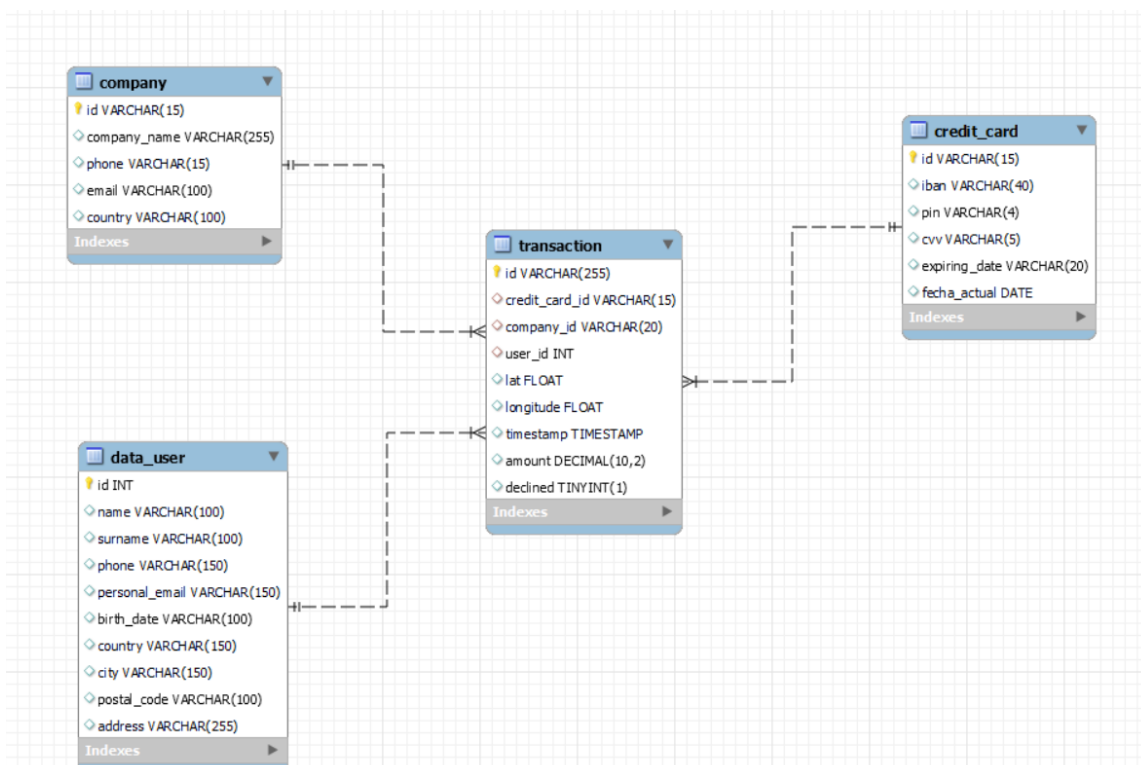
```
Q|- INSERT INTO user
Replace All Replace INSERT INTO data_user

1 • SET foreign_key_checks = 0;
2
3 -- Insertamos datos de user
4 • INSERT INTO data_user (id, name, surname, phone, email, birth_date, country, city, postal_code, address) VALUES ( "1"
5 • INSERT INTO data_user (id, name, surname, phone, email, birth_date, country, city, postal_code, address) VALUES ( "2"
6 • INSERT INTO data_user (id, name, surname, phone, email, birth_date, country, city, postal_code, address) VALUES ( "3"
7 • INSERT INTO data_user (id, name, surname, phone, email, birth_date, country, city, postal_code, address) VALUES ( "4"
8 • INSERT INTO data_user (id, name, surname, phone, email, birth_date, country, city, postal_code, address) VALUES ( "5"
9 • INSERT INTO data_user (id, name, surname, phone, email, birth_date, country, city, postal_code, address) VALUES ( "6"
10 • INSERT INTO data_user (id, name, surname, phone, email, birth_date, country, city, postal_code, address) VALUES ( "7"
11 • INSERT INTO data_user (id, name, surname, phone, email, birth_date, country, city, postal_code, address) VALUES ( "8"
12 • INSERT INTO data_user (id, name, surname, phone, email, birth_date, country, city, postal_code, address) VALUES ( "9"
13 • INSERT INTO data_user (id, name, surname, phone, email, birth_date, country, city, postal_code, address) VALUES ( "1"
14 • INSERT INTO data_user (id, name, surname, phone, email, birth_date, country, city, postal_code, address) VALUES ( "1"
```

Después de una revisión, se observa que el campo *email* de la tabla *data_user* en el diagrama que tenemos que respetar aparece con el nombre de *personal_email*, por lo que ejecutamos el comando para realizar esta modificación:

```
ALTER TABLE data_user
CHANGE COLUMN email personal_email VARCHAR(150);
```

Después de estas acciones el diagrama, realizado a través del comando *Reverse Engineer*, aparece así:



Nivel 3, Ejercicio 2

- La empresa también te solicita crear una vista llamada "InformeTecnico" que contenga la siguiente información:
 - o ID de la transacción
 - o Nombre del usuario/aria
 - o Apellido del usuario/aria
 - o IBAN de la tarjeta de crédito usada.
 - o Nombre de la compañía de la transacción realizada.Asegúrate de incluir información relevante de ambas tablas y utiliza alias para cambiar de nombre columnas según sea necesario.

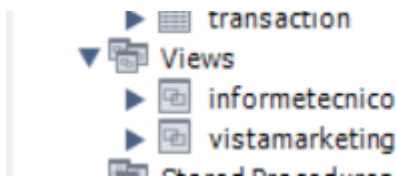
Muestra los resultados de la vista, ordena los resultados de manera descendente en función de la variable ID de *transaction*:

El código que se va a crear realiza una vista llamada *InformeTecnico* que recopila información clave relacionada con las transacciones de la base de datos *transactions*. La vista combina datos de varias tablas mediante uniones (JOIN), permitiendo acceder a detalles específicos de cada transacción. En particular, selecciona el ID de la transacción, el nombre y apellido del usuario, el IBAN de la tarjeta de crédito utilizada y el nombre de la compañía asociada. Los resultados están organizados en orden descendente según el ID de la transacción, lo que facilita consultar primero las transacciones más recientes.

```
--
31 • CREATE VIEW InformeTecnico AS
32 SELECT t.id AS id_transaction, u.name AS user_name, u.surname AS user_surname, cr.iban, c.company_name
33 FROM transaction t
34 JOIN company c
35 ON t.company_id = c.id
36 JOIN credit_card cr
37 ON t.credit_card_id = cr.id
38 JOIN data_user u
39 ON t.user_id = u.id
40 ORDER BY id_transaction DESC;
41
42
```

Output			
Action Output			
#	Time	Action	Message
1	19:06:37	CREATE VIEW InformeTecnico AS SELECT t.id AS id_transaction, u.name AS user_name, u.surname AS use...	0 row(s) affected

Ahora las vistas de la base de datos *transactions* son las siguientes:



y el diagrama actualizado aparece como sigue:

