

## DESCRIPCIÓN

En este sprint, se simula una situación empresarial en la que debes realizar diversas manipulaciones en las tablas de la base de datos. A su vez, tendrás que trabajar con índices y vistas. En esta actividad, continuarás trabajando con la base de datos que contiene información de una empresa dedicada a la venta de productos online. En esta tarea, empezarás a trabajar con información relacionada con tarjetas de crédito.

## NIVEL I

### EJERCICIO I

Tu tarea es diseñar y crear una tabla llamada "credit\_card" que almacene detalles cruciales sobre las tarjetas de crédito. La nueva tabla debe ser capaz de identificar de forma única cada tarjeta y establecer una relación adecuada con las otras dos tablas ("transaction" y "compañero"). 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 del mismo.

El ejercicio consiste en crear una tabla llamada "credit\_card" dentro de la base de datos *Transactions*, estableciendo las relaciones con las demás tablas existentes, para luego poder importarle los datos que ya vienen en el archivo "datos\_introducir\_credit".

#### 1- Selección de base de datos

Para comenzar con el ejercicio, tenemos que seleccionar la base de datos con la que vamos a trabajar. Para ello hice la siguiente declaración:

```
USE transactions;
```

696 21:36:16 USE transactions 0 row(s) affected

#### 2- Creación de una tabla

Una vez seleccionada la base de datos (punto 1), procedí con el proceso de creación de la tabla *credit\_card*. Como primer paso, definí una declaración de Indexación (index). Es una declaración que se utiliza para crear índices en tablas, para que la recuperación de los datos en una base de datos sea con mayor rapidez. Es decir, sirve para acelerar las consultas (optimizar).

```
CREATE INDEX idx_credit_card ON transaction (credit_card_id);
```

697 21:38:37 CREATE INDEX idx\_credit\_card ON transaction (credit\_card\_id) 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

- **CREATE INDEX:** declaración que indica la creación de índice en la tabla.
- **Idx\_credit\_card:** es el nombre del índice
- **ON transaction:** indica en cual tabla se creará el índice, en este caso es en la tabla *Transaction*.
- **(credit\_card\_id):** es la columna sobre la cual se crea el índice en la tabla *Transaction*. Este campo ya existe en dicha tabla.

Después de crear los índices, procedí con la creación de la tabla `credit_card`. Para ello comencé indicando:

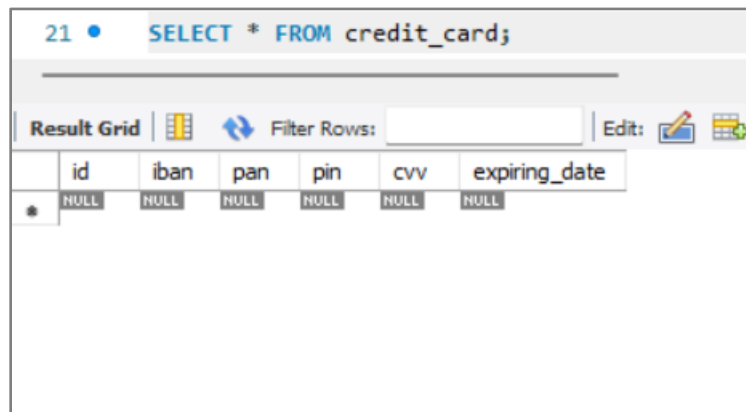
```
13 • CREATE TABLE IF NOT EXISTS credit_card (  
14     id VARCHAR(20) PRIMARY KEY,  
15     iban VARCHAR(50),  
16     pan VARCHAR (30),  
17     pin VARCHAR (4),  
18     cvv INT,  
19     expiring_date VARCHAR (30)  
20 );
```

698 21:41:17 CREATE TABLE IF NOT EXISTS credit\_card (id VARCHAR(20) PRIMARY KEY, iban VARCHAR(50), ... 0 row(s) affected

- **CREATE TABLE IF NOT EXISTS credit\_card:** declaración que indica la creación de una tabla en la base de datos, con la condición de que dicha tabla no existe. Luego de esta primer parte, coloque el nombre de la nueva tabla a crear, en este caso es “credit\_card”.
- **CREACION DE CAMPOS:** luego de la declaración, procedí a definir los campos que compondrán la tabla, teniendo en cuenta la información proporcionada en el archivo “datos\_introducir\_credit”. Los campos creados con sus especificaciones son las siguientes:
  - **id VARCHAR(20) PRIMARY KEY:** identificador único para cada tarjeta de crédito, con la restricción **PRIMARY KEY** para asegurar la unicidad y evitar valores null. Es de tipo **VARCHAR** con una longitud de hasta 20 caracteres, permitiendo letras, números y caracteres especiales.
  - **iban VARCHAR(50):** (International Bank Account Number). Campo que almacena el número de cuenta internacional asociada la tarjeta. Lo definí como un tipo de dato **VACHAR** con una longitud de 50 caracteres que pueden contener letras, números y caracteres especiales.
  - **pan VARCHAR (30):** (Primary Account Number de la tarjeta). Corresponde al número principal de la cuenta. Lo definí como un tipo de dato **VACHAR** con una longitud de 30 caracteres que pueden contener letras, números y caracteres especiales.
  - **pin VARCHAR (4):** corresponde al **PIN** de la tarjeta y lo definí como un tipo de dato **VARCHAR** con una longitud de 4 caracteres que pueden contener letras, números y caracteres especiales.
  - **cvv INT:** código de verificación de la tarjeta y lo definí como un tipo de dato de un valor entero (**INT**).
  - **expiring\_date VARCHAR (30):** campo que contiene la fecha de expiración de la tarjeta. Temporalmente, lo definí como **VARCHAR (30)** para admitir los datos de fecha que no coinciden con el formato **DATE** de SQL, con la intención de insertar inicialmente los datos y luego transformarlos y ajustar el tipo de dato a **DATE** cuando la estructura de los registros esté adaptada.

Una vez ejecutada esta declaración de creación de tablas, podemos verificar que la tabla se ha creado correctamente, junto con todos sus campos/columnas correspondientes, mediante la siguiente instrucción:

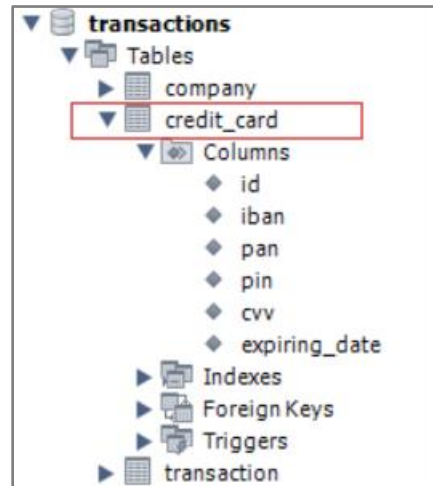
```
SELECT * FROM credit_card;
```



	id	iban	pan	pin	cvv	expiring_date
*	NULL	NULL	NULL	NULL	NULL	NULL

699 21:42:39 SELECT \* FROM credit\_card 0 row(s) returned

Aquí podemos ver cómo se creó la tabla dentro de la base de datos *Transactions*.



### 3- Importación de datos a la tabla “credit card”

Tras definir la estructura de la nueva tabla *credit\_card*, procedí a importar los datos proporcionados en el archivo “*datos\_introducir\_credit*” a la tabla. Para ello, lo que hice fue abrir el archivo “*datos\_introducir\_credit.sql*” en MyWorkbench, controlé que el código que estaba escrito en ese archivo estuviera correcto para insertar los registros (INSERT INTO...) y que el nombre de las columnas coincidiera con la tabla que había creado. Luego de dicha verificación, ejecuté el código para que los datos que contenía el archivo ingresaran a la tabla *credit\_card* de la base de datos *transactions*:

S3\_Ejercicios

datos\_introducir\_credit

Don't Limit

```

1
2 -- Insertamos datos de credit_card
3 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2938', 'TR301950312213576817638661', '54244655
4 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2945', 'D026854763748537475216568689', '514242
5 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2952', 'B645IVQL52710525608255', '4556 453 55
6 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2959', 'CR7242477244335841535', '3724613773493
7 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2966', 'B672LKTQ15627628377363', '448566 88674
8 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2973', 'PT87806228135092429456346', '544 58654
9 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2980', 'DE39241881883086277136', '402400 71458
10 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2987', 'GE89681434837748781813', '3763 747687
11 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-2994', 'BH62714428368066765294', '344283273252
12 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3001', 'CY49087426654774581266832110', '511722
13 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3008', 'LU507216693616119230', '44857444644338
14 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3015', 'PS119398216295715968342456821', '3784
15 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3022', 'GT9169516285056977423121857', '5164 1
16 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3029', 'AZ62317413982441418123739746', '3429 2
17 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3036', 'AZ39336002925842865843941994', '3768 4
18 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3043', 'TN6488143310514852179535', '455676 643
19 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3050', 'FR5167744369175836831854477', '4024007
20 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3057', 'LU931822574697545215', '3484 621767 21
21 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3064', 'PS146965545449253377627273133', '3467
22 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3071', 'N08923814763512', '3464 789562 23352',
23 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3078', 'IS025127145884623279548733', '4539 322
24 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3085', 'BE63114723972437', '5266 3346 1135 168
25 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3092', 'R065LS0D1166122125447487', '3488 75422
26 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3099', 'PT26105275356823705537218', '448 55418
27 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-3106', 'AT684251637751136592', '34954714639528

```

✓	288	12-12-23	INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-4835', 'PT34592...	1 row(s) affected
✓	289	12-12-23	INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-4842', 'SA21567...	1 row(s) affected
✓	290	12-12-23	INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-4849', 'SE28131...	1 row(s) affected
✓	291	12-12-23	INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ( 'CcU-4856', 'TR37387...	1 row(s) affected

Para verificar que la importación se realizó correctamente, ejecuté la siguiente instrucción:

```
SELECT * FROM credit_card;
```

21 • `SELECT * FROM credit_card;`

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

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
CcU-3001	CY49087426654774581266832110	511722 924833 2244	9562	867	09/16/22
CcU-3008	LU507216693616119230	4485744464433884	1856	740	04/05/25
CcU-3015	PS119398216295715968342456821	3784 662233 17389	3246	822	01/31/22
CcU-3022	GT91695162850556977423121857	5164 1379 4842 3951	5610	342	04/25/25
CcU-3029	AT62317413987441418123730746	3479 270566 77631	9708	505	09/07/23

credit\_card 2 x

979 21:45:26 `SELECT * FROM credit_card` 275 row(s) returned

#### 4- Actualización de Formato y Tipo de Datos en "expiring\_date"

Como mencioné en el segundo punto, fue necesario definir la columna `expiring_date` como un `VARCHAR (30)` temporalmente para poder crear la tabla `credit_card` con los datos proporcionados en su formato original. Una vez creada la tabla, continué con los pasos correspondientes para ajustar la estructura y asegurar que `expiring_date` tuviera las propiedades correctas para almacenar datos en formato de fecha (`DATE`). A continuación, explico dichos pasos:

- A. Conversión de los registros de la columna "expiring\_date": en esta declaración lo que hago es estandarizar el formato de los registros que se encuentran en dicha columna, a un tipo de dato `DATE`. Para ello, realice las siguientes declaraciones:

```
24 • SET SQL_safe_updates = 0;
25
26 • UPDATE credit_card
27   SET expiring_date = STR_TO_DATE(expiring_date, '%m/%d/%y')
28   WHERE expiring_date IS NOT NULL;
29
30 • SET SQL_safe_updates = 1;
```



✓	980	21:47:10	SET SQL_safe_updates = 0	0 row(s) affected
✓	981	21:47:14	UPDATE credit_card SET expiring_date = STR_TO_DATE(expiring_date, '%m/%d/%y') WHERE expiring_d...	275 row(s) affected Rows matched: 275 Changed: 275 Warnings: 0
✓	982	21:47:19	SET SQL_safe_updates = 1	0 row(s) affected

## PRIMER PASO

- **SET SQL\_safe\_updates = 0:** debido a que mi usuario tiene activado el modo seguro de actualizaciones, para poder realizar esta tarea, tuve que desactivar dicho modo a través de esta declaración. Donde establezco que la función SQL\_safe\_update es igual a 0, es decir lo desactivo.

## SEGUNDO PASO

- **UPDATE credit\_card:** declaración que especifica que se desea modificar los registros existentes en la tabla *credit\_card*.
- **SET expiring\_date = STR\_TO\_DATE(expiring\_date, '%m/%d/%y');** establece que la columna *expiring\_date* debe ser actualizada utilizando la función *STR\_TO\_DATE*, la cual transforma los valores de la columna a un tipo de dato *DATE*. Dentro de la función, se especifica que los valores de *expiring\_date* deben interpretarse con el siguiente formato de fecha:
  - %m: mes
  - %d: día
  - %y: año.
- **WHERE expiring\_date IS NOT NULL:** aplique la cláusula *WHERE* para limitar la actualización de los registros de la columna *expiring\_date* a aquellos que tienen un valor, es decir que no sean *NULL*.

En conclusión, esta declaración convierte los registros que tienen un formato de fecha como 10/30/2024 en el formato estándar *DATE*, resultando en 2024-10-30

## TERCER PASO

- **SET SQL\_safe\_updates = 1:** luego de haber realizado la actualización de los registros del campo *expiring\_date*, vuelvo a activar el modo seguro de actualizaciones en SQL. Es la misma declaración que el primer paso, pero esta vez la igualación de la función *SQL\_safe\_update* es igual a 1, es decir lo activo.

## COMPROBACION

Una vez realizado los pasos anteriores, lo que hice fue comprobar que efectivamente los registros del campo *expiring\_date* fueron actualizados, para ello hice la siguiente declaración:

```
SELECT expiring_date FROM credit_card;
```

```
32 • SELECT expiring_date FROM credit_card;
33
```

expiring_date
2022-10-30
2023-08-24
2021-06-29
2023-02-24
2024-10-29
2025-01-30
2022-07-24
2023-10-31
2022-02-28
2022-09-16
2025-04-05
2022-01-31
2025-04-25
2023-09-02
2025-10-27
2025-06-07
2023-10-09

credit\_card 4 x

983 21:48:24 SELECT expiring\_date FROM credit\_card 275 row(s) returned

- B. Modificación del tipo de dato de la columna “expiring\_date”: una vez transformado los registros de la columna “expiring\_date” a un formato compatible con el tipo de dato DATE, lo que queda para finalizar con esta tarea es modificar la información del tipo de dato de dicha columna. Es decir, de cambiar VARCHAR (30) a DATE. Para ello, hice la siguiente declaración:

```
34 • ALTER TABLE credit_card
35     MODIFY COLUMN expiring_date DATE;
```

984 22:28:39 ALTER TABLE credit\_card MODIFY COLUMN expiring\_date DATE 275 row(s) affected Records: 275 Duplicates: 0 Warnings: 0

- **ALTER TABLE credit\_card**: cláusula que indica que se va a realizar una modificación en la estructura de la tabla credit\_card.
- **MODIFY COLUMN expiring\_date DATE**: especifica que la modificación se centrará en la columna expiring\_date, transformándola en un tipo de dato DATE.

A continuación, comprobamos dicha modificación:

Table: credit\_card

Columns:	
id	varchar(20) PK
iban	varchar(50)
pan	varchar(30)
pin	varchar(4)
cvv	int
expiring_date	date

## 5- Creando relación con tabla transaction.

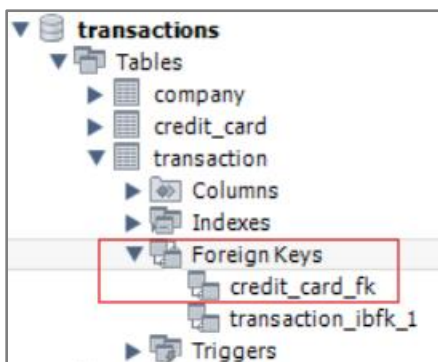
Ya creada en la base de datos la nueva dimensión (*credit\_card*), lo que queda es relacionarla con la tabla de hechos que en este caso es la tabla *transaction*. Para ello hice la siguiente declaración:

```
42 • ALTER TABLE transaction
43     ADD CONSTRAINT credit_card_fk
44     FOREIGN KEY (credit_card_id) REFERENCES credit_card(id);
```

986 23:22:18 ALTER TABLE transaction ADD CONSTRAINT credit\_card\_fk FOREIGN KEY (credit\_card\_id) REFERENCES credit\_card(id) 587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0

- **ALTER TABLE transaction:** cláusula que indica que se va a realizar una modificación en la estructura de la tabla *transaction*.
- **ADD CONSTRAINT credit\_card\_fk:** función que agrega una restricción de clave foránea con el nombre *credit\_card\_fk*
- **FOREIGN KEY (credit\_card\_id) REFERENCES credit\_card(id):** función que establece que el campo *credit\_card\_id* de la tabla *transaction* debe coincidir con el campo *id* de la tabla *credit\_card*.

Una vez ejecutada esta declaración, a continuación, podemos corroborar que se creó dicha relación en la tabla *transaction*:



## 6- Ajuste de relación entre las tablas user y transaction

Para complementar este ejercicio importe la tabla contenida en el documento “estructura\_datos\_user” y “datos\_introducir\_user” a MySQL Workbench, para luego analizarla y mostrar su esquema en el siguiente punto. Al importar la estructura, observé que la relación de clave foránea entre la tabla *user* y la tabla *transaction* se ha configurado de manera incorrecta. En la definición de la estructura de la tabla *user*, se declaró que **FOREIGN KEY (id) REFERENCES transaction (user\_id)** corresponde a la misma tabla *user*, lo cual no es adecuado. En realidad, la tabla *user* debe actuar como dimensión, mientras que la tabla *transaction* es la tabla de hechos. Por lo tanto, esta relación debe ser corregida para reflejar correctamente la estructura del modelo de datos.



```
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
```

Para corregir esta situación lo primero que hice fue modificar la tabla user, eliminando la clave foránea

```
115 ALTER TABLE user  
116 DROP FOREIGN KEY user_ibfk_1;  
117
```

1301 17:24:55 ALTER TABLE user DROP FOREIGN KEY user\_ibfk\_1 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

- **ALTER TABLE user:** declaración que indica que se quiere modificar la tabla user.
- **DROP FOREIGN KEY user\_ibfk\_1:** función que se utiliza para eliminar una clave foránea. En este caso se indica que se quiere eliminar la FK user\_ibfk\_1 (nombre de la clave foránea).

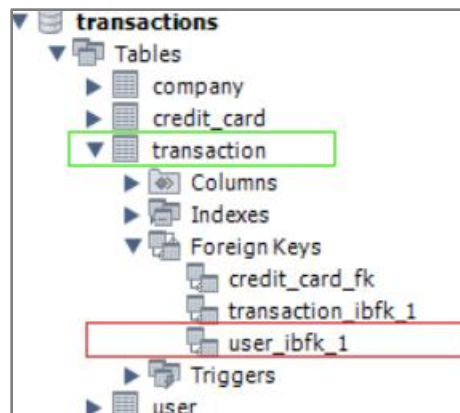
Luego de realizar esta eliminación, continué con la modificación de la tabla transaction para crear la relación con la tabla user:

```
119 ALTER TABLE transaction  
120 ADD CONSTRAINT user_ibfk_1  
121 FOREIGN KEY (user_id) REFERENCES user(id);  
122
```

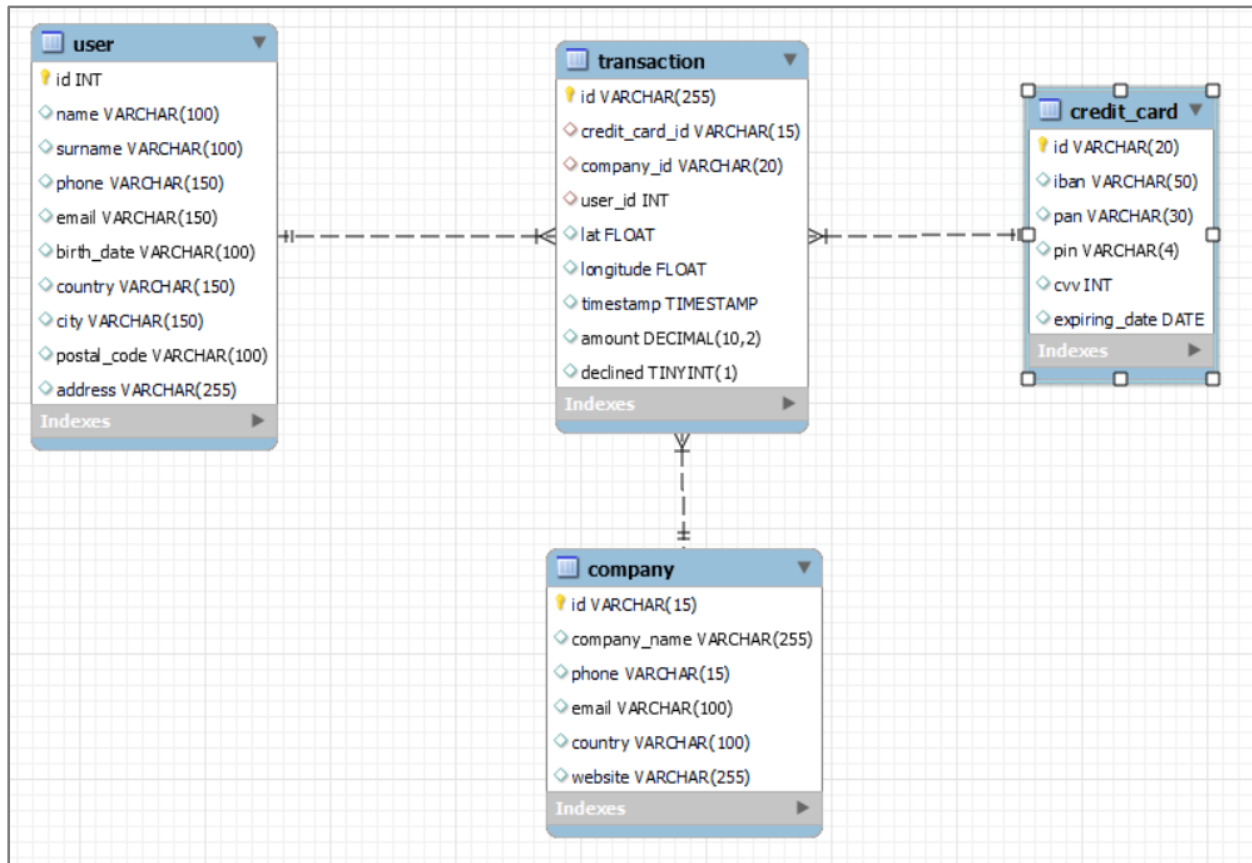
1302 17:29:35 ALTER TABLE transaction ADD CONSTRAINT user\_ibfk\_1 FOREIGN KEY (user\_id) REFERENCES u... 587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0

- **ALTER TABLE transaction:** cláusula que indica que se va a realizar una modificación en la estructura de la tabla transaction.
- **ADD CONSTRAINT user\_ibfk\_1:** función que agrega una restricción de clave foránea con el nombre user\_ibfk\_1.
- **FOREIGN KEY (user\_id) REFERENCES user(id):** función que establece que el campo user\_id de la tabla transaction debe coincidir con el campo id de la tabla user.

Una vez ejecutada esta declaración, a continuación, podemos corroborar que se creó dicha relación en la tabla transaction:



## 7- Relación Entre Las Tablas:



Como podemos ver en la imagen, este modelo de datos corresponde a un modelo denominado “Estrella”. Este tipo de modelo organiza la estructura de la base de datos en dos (2) tipos principales de tablas: Tabla de Dimensiones y Tabla de hechos.

Las tablas company (tabla de dimensión) y transaction (tabla de hechos) están interrelacionadas a través de una relación uno a muchos (1:N), donde la clave primaria id de la tabla company se utiliza como clave

foránea `company_id` en la tabla `transaction`. Esto significa que una única empresa puede realizar múltiples transacciones.

Lo mismo pasa con las tablas `credit_card` (tabla de dimensión) y `transaction` (tabla de hechos) que están interrelacionadas a través de una relación uno a muchos (1:N), donde la clave primaria `id` de la tabla `credit_card` se utiliza como clave foránea `credit_card_id` en la tabla `transaction`. Es decir, que una única tarjeta de crédito puede estar asociada con múltiples transacciones.

También las tablas `user` (tabla de dimensión) y `transaction` (tabla de hechos) que están interrelacionadas a través de una relación uno a muchos (1:N), donde la clave primaria `id` de la tabla `user` se utiliza como clave foránea `user_id` en la tabla `transaction`. Es decir, que un usuario puede realizar múltiples transacciones.

## 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 debe mostrarse para este registro es: `R323456312213576817699999`. Recuerda mostrar que el cambio se realizó.

Para realizar este ejercicio, utilicé la declaración `UPDATE`, que se emplea para modificar registros existentes en una tabla de base de datos. A continuación, explico la misma:

```
51 • UPDATE credit_card
52   SET iban = 'R323456312213576817699999'
53   WHERE id = 'CcU-2938';
```

1272 13:10:50 UPDATE credit\_card SET iban = 'R323456312213576817699999' WHERE id = 'CcU-2938' 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

- `UPDATE credit_card`: indico que se realizará una actualización en los registros de la tabla `credit_card`.
- `SET iban = 'R323456312213576817699999'`: especifico el campo y el nuevo valor/registro que voy a actualizar. En este caso, el campo `iban` es el que se modificará con el nuevo valor `'R323456312213576817699999'` que se quiere mostrar en la tabla.
- `WHERE id = 'CcU-2938'`: agregué la cláusula `WHERE` para condicionar la actualización. Es decir, que la modificación se aplique únicamente al campo `id` que cuyo registro sea igual a `'CcU-2938'`

Una vez ejecutada esta declaración, comprobé que dicho `UPDATE` se haya realizado correctamente, mediante la siguiente declaración:

```
55 • SELECT id, iban
56   FROM credit_card
57   WHERE id = 'CcU-2938';
```

Result Grid		Filter Rows:	Ed
id	iban		
CcU-2938	R323456312213576817699999		

1273 13:13:08 SELECT id, iban FROM credit\_card WHERE id = 'CcU-2938' 1 row(s) returned

- **SELECT id, iban:** selecciono los campos/columnas que quiero mostrar, en este caso es *id* e *iban*.
- **FROM credit\_card:** indico la tabla de la cual obtendremos los datos, en este caso es la tabla *credit\_card*.
- **WHERE id = 'CcU-2938':** condiono el resultado con la cláusula WHERE donde especifico que la consulta solo debe traer aquellos registros cuyo *id* sea igual a 'CcU-2938'.

## 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

Antes de proceder con la inserción de los datos solicitados, realicé un análisis de la estructura y los registros existentes en la tabla *transaction* con el objetivo de familiarizarme con su diseño y contenido. Este proceso me permitió identificar cualquier ajuste necesario para asegurar que los nuevos datos cumplan con las especificaciones y restricciones de la tabla. El procedimiento de control fue el siguiente:

### I- Revisión de la estructura de la tabla:

Existen varias maneras de realizar esta tarea de análisis y revisión de la estructura de las tablas. Algunas de las opciones disponibles incluyen:

- Revisión de Diagramas de Tablas de Base de Datos: su consulta permite visualizar los campos, relaciones, claves primarias y foráneas, así como otros atributos clave, facilita una comprensión integral de la estructura y las interdependencias de las tablas.
- Investigación en MySQL Workbench: utilizar la sección SCHEMAS y la opción INFORMATION en MySQL Workbench para examinar las características de las tablas y su contenido.
- Consultas SQL para Inspección: haciendo declaraciones con DESCRIBE podemos observar la estructura de las tablas y sus restricciones. Con SELECT podemos consultar para analizar los registros actuales en las tablas.

71 • DESCRIBE transaction;

Result Grid | Filter Rows: | Export: | Wrap Cell C

Field	Type	Null	Key	Default	Extra
id	varchar(255)	NO	PRI	NULL	
credit_card_id	varchar(15)	YES	MUL	NULL	
company_id	varchar(20)	YES	MUL	NULL	
user_id	int	YES	MUL	NULL	
lat	float	YES		NULL	
longitude	float	YES		NULL	
timestamp	timestamp	YES		NULL	
amount	decimal(10,2)	YES		NULL	
declined	tinyint(1)	YES		NULL	

Result 14 x

✓ 1275 14:41:14 DESCRIBE transaction 9 row(s) returned

- **DESCRIBE transaction:** es una función que permite obtener información detallada de la estructura de la tabla. En este caso estamos solicitando información de la tabla *transaction*. Al ejecutar esta declaración, obtuve el siguiente resultado:
- ✓ **FIELD (campos):** proporciona el nombre de todos los campos presente en la tabla. En este caso son nueve (9) campos:
    - Id
    - Credit\_card\_id
    - Company\_id
    - User\_id
    - Lat
    - Longitude
    - Timestamp
    - Amount
    - Declined
  - ✓ **TYPE (tipo):** indica el tipo de dato que cada columna puede almacenar. Para la tabla *transaction*, los tipos de datos son los siguientes:
    - *id*, *credit\_card\_id* y *company\_id* => VACHAR (SIZE) → almacena string con una longitud determinada.
    - *user\_id* => INT → almacena números enteros sin decimales.
    - *Lat* y *longitude* => FLOAT → almacena números que pueden tener decimales.
    - *Timestamp* => TIMESTAMP → almacena fechas y horas.
    - *Amount* => DECIMAL (SIZE, D) → almacena números decimales con precisión exacta.
    - *Declined* => TINYINT (SIZE) → almacena números muy pequeños.
  - ✓ **NULL (nulo):** indica si las columnas permiten valores nulos. En este caso, el único campo que no lo permite es el campo *id*.

- ✓ KEY (clave): informa sobre las claves primarias (PK), las claves foráneas (FK) o si la columna forma parte de un Índice (MUL). En este caso, el campo *id* es una PK y los campos *credit\_card\_id*, *company\_id* y *user\_id* son MUL, es decir que estos campos pueden tener valores duplicados.
- ✓ DEFAULT (predeterminado): muestra el valor predeterminado asignado a la columna si no se proporciona un valor. En este caso, todos los campos están configurados con un valor predeterminado de NULL, lo que implica que, si no se especifica un valor al insertar un registro, la base de datos asignará automáticamente un valor nulo.
- ✓ EXTRA: información adicional que se puede incluir para entender mejor los campos. En esta tabla, no se presenta información extra.

## 2- Consulta de Registros existente:

Consulté los registros actuales en la tabla para observar los valores típicos de cada columna, asegurándome de que los nuevos datos sean consistentes con los valores existentes.

74 • `SELECT * FROM transaction;`

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

	id	credit_card_id	company_id	user_id	lat	longitude	timestamp	amount	declined
▶	02C6201E-D90A-1...	CdJ-2938	b-2362	92	81.9185	-12.5276	2021-08-28 23:42:24	466.92	0
	0466A42E-47CF-8...	CdJ-4219	b-2302	170	-43.9695	-117.525	2021-07-26 07:29:18	49.53	0
	063FBA79-99EC-6...	CdJ-2987	b-2250	275	-81.2227	-129.05	2022-01-06 21:25:27	92.61	0
	0668296C-CDB9-A...	CdJ-3743	b-2618	265	-34.3593	-100.556	2022-01-26 02:07:14	394.18	0
	06CD9AA5-9B42-D...	CdJ-2959	b-2346	92	33.7381	158.298	2021-10-26 23:00:01	279.93	0
	07A46D48-31A3-7...	CdJ-3225	b-2386	272	38.8342	92.1905	2021-06-28 21:11:42	340.87	1
	09DE92CE-6F27-2...	CdJ-3071	b-2298	275	71.1706	10.5757	2021-05-11 20:40:06	303.05	1
	0A476ED9-0C13-1...	CdJ-4359	b-2302	221	-56.4901	114.801	2022-02-26 20:33:54	430.49	0
	08EB80B7-9D66-1...	CdJ-3141	b-2338	272	23.3264	-13.6037	2022-03-04 14:54:35	288.81	1
	0C7C3A33-9947-3...	CdJ-3309	b-2434	272	63.3615	-68.6667	2021-04-10 20:58:41	103.44	1

1276 15:41:44 `SELECT * FROM transaction` 587 row(s) returned

- `SELECT * FROM transaction`: declaración para consultar todos los registros existentes en la tabla *transaction*.

De la consulta realizada, se pudo identificar que no todos los registros a ingresar coinciden con los campos existentes en la tabla *transaction*. A continuación, se detallan las modificaciones necesarias para garantizar la correcta inserción de los datos:

- Latitud (lat) => el valor 829.999 no es congruente con el formato esperado para los registros de latitud en la tabla. Por lo tanto, será necesario ajustar este valor a 82.9999 antes de realizar la inserción, asegurando así que se cumplan las especificaciones de formato. En este caso al ser un único dato a modificar, lo realizaré de manera manual.
- Timestamp => aunque la información de fecha y hora no está especificada en el enunciado, este campo es fundamental para la tabla *transaction*, ya que registra momentos temporales relacionados con cada transacción. A pesar de que la estructura de la tabla indica que este campo puede tener un valor predeterminado de NULL, es recomendable evitar que quede nulo por la importancia antes mencionada.



Por lo tanto, cuando tenga que insertar los registros a la tabla *transaction*, aplicaré la función `CURRENT_TIMESTAMP` al momento de insertar el registro.

### 3- Verificación de Relaciones entre Tablas:

En esta etapa, realicé un rastreo para verificar que los registros correspondientes a los campos de claves foráneas realmente existan en las tablas relacionadas. Este procedimiento me asegura que los datos que estoy ingresando en la tabla *transaction* se correspondan con registros válidos en sus respectivas tablas madre.

- **CREDIT CARD ID (FK):**

```
79 • SELECT*
80 FROM credit_card
81 WHERE id = 'CcU-9999';
```

Result Grid

id	iban	pan	pin	cvv	expiring_date
NULL	NULL	NULL	NULL	NULL	NULL

✓ 1283 16:15:17 SELECT\* FROM credit\_card WHERE id = 'CcU-9999' 0 row(s) returned

- `SELECT* FROM credit_card`: declaración para consultar todos los campos existentes en la tabla *credit\_card*.
- `WHERE id = 'CcU-9999'`: coloque la cláusula `WHERE` para filtrar el resultado considerando que el `id` sea igual al nuevo valor que se está ingresando.

Esta consulta indica que el valor especificado no existe en la tabla *credit\_card*, lo que implica la necesidad de ingresarlo para mantener la integridad referencial con la tabla *transaction*. A continuación, presento la declaración de inserción correspondiente:

```
96 • INSERT INTO credit_card (id)
97 VALUE ('CcU-9999');
98
99 • SELECT*
100 FROM credit_card
101 WHERE id = 'CcU-9999';
```

Result Grid

id	iban	pan	pin	cvv	expiring_date
CcU-9999	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL

✓	1293	17:03:09	INSERT INTO credit_card (id) VALUE ('CcU-9999')	1 row(s) affected
✓	1294	17:04:08	SELECT* FROM credit_card WHERE id = 'CcU-9999'	1 row(s) returned

- **INSERT INTO credit\_card (id):** cláusula que indica que se quiere ingresar un nuevo registro en el campo id de la tabla credit\_card.
- **VALUE ('CcU-9999'):** especificación del nuevo valor a ingresar, en este caso es 'CcU-9999'.

Una vez ejecutada esta declaración, seguí con su verificación:

- **SELECT\* FROM credit\_card:** declaración para consultar todos los campos existentes en la tabla credit\_card.
- **WHERE id = 'CcU-9999':** coloque la cláusula WHERE para filtrar el resultado considerando que el id sea igual al nuevo valor que se está ingresando.

Esta consulta ha confirmado que el id = 'CcU-9999' se ha insertado correctamente en la tabla *credit\_card*. Por otro lado, debido a la ausencia de datos adicionales necesarios para completar las columnas existentes, los valores correspondientes aparecen como NULL, conforme a la configuración predeterminada de la tabla (default). Este valor NULL permite la posterior actualización de la tabla con la información relevante. Este registro se ha creado con el propósito de establecer una relación con la tabla *transaction*, asegurando así la integridad referencial entre ambas tablas.

- **COMPANY ID (FK):**

83 • SELECT\*

84 FROM company

85 WHERE id = 'b-9999';

96

Result Grid

Filter Rows:

Edit:

Export/Import:

W

id	company_name	phone	email	country	website
NULL	NULL	NULL	NULL	NULL	NULL

✓	1285	16:22:26	SELECT* FROM company WHERE id = 'b-9999'	0 row(s) returned
---	------	----------	--	-------------------

- **SELECT\* FROM company:** declaración para consultar todos los campos existentes en la tabla company.
- **WHERE id = 'b-9999':** coloque la cláusula WHERE para filtrar el resultado considerando que el id sea igual al nuevo valor que se está ingresando.

Esta consulta indica que el valor especificado no existe en la tabla *company*, lo que implica la necesidad de ingresarlo para mantener la integridad referencial con la tabla *transaction*. A continuación, presento la declaración de inserción correspondiente:

```
105 • INSERT INTO company (id)
106     VALUE ('b-9999');
107
108 • SELECT*
109     FROM company
110     WHERE id = 'b-9999';
111
```

id	company_name	phone	email	country	website
b-9999	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL

✓	1295	17:07:36	INSERT INTO company (id) VALUE (b-9999)	1 row(s) affected
✓	1296	17:07:45	SELECT* FROM company WHERE id = 'b-9999'	1 row(s) returned

- **INSERT INTO company (id):** cláusula que indica que se quiere ingresar un nuevo registro en el campo id de la tabla company.
- **VALUE ('b-9999');** especificación del nuevo valor a ingresar, en este caso es 'b-9999'.

Una vez ejecutada esta declaración, seguí con su verificación:

- **SELECT\* FROM company:** declaración para consultar todos los campos existentes en la tabla company.
- **WHERE id = 'b-9999':** coloque la cláusula WHERE para filtrar el resultado considerando que el id sea igual al nuevo valor que se está ingresando.

Esta consulta ha confirmado que el id = 'b-9999' se ha insertado correctamente en la tabla *company*. Por otro lado, debido a la ausencia de datos adicionales necesarios para completar las columnas existentes, los valores correspondientes aparecen como NULL, conforme a la configuración predeterminada de la tabla (default). Este valor NULL permite la posterior actualización de la tabla con la información relevante. Este registro se ha creado con el propósito de establecer una relación con la tabla *transaction*, asegurando así la integridad referencial entre ambas tablas.

- **USER ID (FK):**

```
87 • SELECT*
88     FROM user
89     WHERE id = '9999';
90
```

id	name	surname	phone	email	birth_date	country	city	postal_code	address
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

✓	1286	16:24:52	SELECT* FROM user WHERE id = '9999'	0 row(s) returned
---	------	----------	-------------------------------------	-------------------

- `SELECT* FROM user`: declaración para consultar todos los campos existentes en la tabla *user*.
- `WHERE id = '9999'`: coloque la cláusula `WHERE` para filtrar el resultado considerando que el `id` sea igual al nuevo valor que se está ingresando.

Esta consulta indica que el valor especificado no existe en la tabla *user*, lo que implica la necesidad de ingresarlo para mantener la integridad referencial con la tabla *transaction*. A continuación, presento la declaración de inserción correspondiente:

```
125 • INSERT INTO user (id)
126     VALUE ('9999');
127
128 • SELECT*
129     FROM user
130     WHERE id = '9999';
131
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

	id	name	surname	phone	email	birth_date	country	city	postal_code	address
▶	9999	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

✓	1303	17:34:56	INSERT INTO user (id) VALUE ('9999')	1 row(s) affected
✓	1304	17:35:02	SELECT* FROM user WHERE id = '9999'	1 row(s) returned

- `INSERT INTO user (id)`: cláusula que indica que se quiere ingresar un nuevo registro en el campo `id` de la tabla *user*.
- `VALUE ('9999')`: especificación del nuevo valor a ingresar, en este caso es `'9999'`.

Una vez ejecutada esta declaración, seguí con su verificación:

- `SELECT* FROM user`: declaración para consultar todos los campos existentes en la tabla *user*.
- `WHERE id = '9999'`: coloque la cláusula `WHERE` para filtrar el resultado considerando que el `id` sea igual al nuevo valor que se está ingresando.

Esta consulta ha confirmado que el `id = '9999'` se ha insertado correctamente en la tabla *user*. Por otro lado, debido a la ausencia de datos adicionales necesarios para completar las columnas existentes, los valores correspondientes aparecen como `NULL`, conforme a la configuración predeterminada de la tabla (default). Este valor `NULL` permite la posterior actualización de la tabla con la información relevante. Este registro se ha creado con el propósito de establecer una relación con la tabla *transaction*, asegurando así la integridad referencial entre ambas tablas.

#### 4- Inserción de los nuevos registros a la tabla *transaction*:

Como paso final para completar el ejercicio, corresponde ingresar los nuevos registros en la tabla *transaction*, siguiendo las consideraciones mencionadas en los puntos anteriores. A continuación, se presenta la declaración de inserción:

```
134 • INSERT INTO transaction (id, credit_card_id, company_id, user_id, lat, longitude, timestamp, amount, declined)
135     VALUES ('108B1D1D-5B23-A76C-55EF-C568E49A99DD', 'CcU-9999', 'b-9999', '9999', 82.9999, -117.999, CURRENT_TIMESTAMP, 111.11, 0);
136
```

1316 21:19:18 INSERT INTO transaction (id, credit\_card\_id, company\_id, user\_id, lat, longitude, timestamp, amount, d... 1 row(s) affected

- `INSERT INTO transaction (id, credit_card_id, company_id, user_id, lat, longitude, timestamp, amount, declined)`: declaración que indica que se quiere insertar nuevos registros en la tabla *transaction* especificando en que columnas. En este caso, se ingresará nuevos registros en todos sus campos.
- `VALUES ('108B1D1D-5B23-A76C-55EF-C568E49A99DD', 'CcU-9999', 'b-9999', '9999', 82.9999, -117.999, CURRENT_TIMESTAMP, 111.11, 0)`: este conjunto de valores especifica los datos a ingresar en cada campos de la tabla *transaction*. Estos valores son dados en el enunciado, con excepción del campo *timestamp*, para el cual utilicé la función `CURRENT_TIMESTAMP` (mencionada anteriormente) que registra automáticamente la fecha y hora actual en el momento de la inserción. Otro tema de aclaración, es el valor *Lat*, ajusté manualmente a 82.9999 para alinear el registro con el formato requerido. Dado que esta fue una corrección puntual en un único registro, lo realicé directamente en el valor insertado. Sin embargo, si hubiera sido necesario ajustar múltiples registros, tendría que haber utilizado la declaración `UPDATE` para modificar los valores en la tabla de manera eficiente.

Terminada la inserción de valores, realicé la consulta para corroborar los datos ingresados:

```
138 • SELECT * FROM transaction
139     WHERE id = '108B1D1D-5B23-A76C-55EF-C568E49A99DD';
140
```

id	credit_card_id	company_id	user_id	lat	longitude	timestamp	amount	declined
108B1D1D-5B23-A...	CcU-9999	b-9999	9999	82.9999	-117.999	2024-11-02 21:19:18	111.11	0
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

1317 21:20:30 SELECT \* FROM transaction WHERE id = '108B1D1D-5B23-A76C-55EF-C568E49A99DD' 1 row(s) returned

- `SELECT* FROM transaction`: declaración para consultar todos los campos existentes en la tabla *transaction*.
- `WHERE id = '108B1D1D-5B23-A76C-55EF-C568E49A99DD'`: coloque la cláusula `WHERE` para filtrar el resultado considerando que el *id* sea igual al nuevo valor ingresado.

## EJERCICIO 4

Desde recursos humanos te solicitan eliminar la columna "pan" de la tabla *credit\_card*. Recuerda mostrar el cambio realizado.

Para cumplir con la solicitud de Recursos Humanos de eliminar la columna *pan* de la tabla *credit\_card*, ejecuté la siguiente declaración:

```
131 • ALTER TABLE credit_card
132     DROP COLUMN pan;
133
```

1319 22:06:17 ALTER TABLE credit\_card DROP COLUMN pan 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

- **ALTER TABLE credit\_card**: declaración que indica que se quiere modificar la tabla *credit\_card*.
- **DROP COLUMN pan**: función que se utiliza para eliminar una columna de una tabla. En este caso se indica que se quiere eliminar la columna *pan* de la tabla mencionada anteriormente.

Para corroborar dicha eliminación de columna, hice la siguiente declaración:

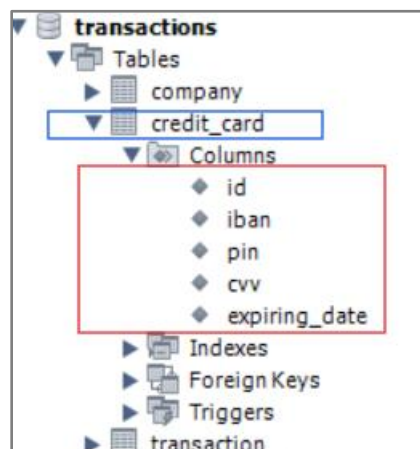
```
136 • DESCRIBE credit_card;
```

Field	Type	Null	Key	Default	Extra
id	varchar(20)	NO	PRI	NULL	
iban	varchar(50)	YES		NULL	
pin	varchar(4)	YES		NULL	
cvv	int	YES		NULL	
expiring_date	date	YES		NULL	

✓ 1320 22:08:41 DESCRIBE credit\_card 5 row(s) returned

- **DESCRIBE credit\_card**: es una función que permite obtener información detallada de la estructura de la tabla. En este caso estamos solicitando información de la tabla *credit\_card*. Al ejecutarla, podemos visualizar todos los campos que componen esta tabla. En este caso, la ejecución muestra que actualmente hay solo cinco campos en la tabla, y la columna *pan* ya no aparece en la lista de campos. Esto confirma que el cambio se realizó correctamente, eliminando la columna *pan*, la cual fue previamente creada y mencionada en pasos anteriores.

Aquí podemos ver que la columna ya no se encuentra en la tabla *credit\_card* dentro de la base de datos *Transactions*.





## NIVEL 2

### EJERCICIO I

Elimina de la tabla transacción el registro con ID 02C6201E-D90A-1859-B4EE-88D2986D3B02 de la base de datos.

Para poder eliminar de la tabla transacción el registro indicado, utilice la siguiente declaración:

```
144 • DELETE FROM transaction
145 WHERE id= '02C6201E-D90A-1859-B4EE-88D2986D3B02';
146
```

1 22:32:16 DELETE FROM transaction WHERE id= '02C6201E-D90A-1859-B4EE-88D2986D3B02' 1 row(s) affected

- **DELETE FROM transaction:** declaración que indica que se quiere eliminar un registro de la tabla *transaction*.
- **WHERE id= '02C6201E-D90A-1859-B4EE-88D2986D3B02';** coloque la cláusula WHERE asegurar que sólo se elimine el registro id = ID 02C6201E-D90A-1859-B4EE-88D2986D3B02.

Para corroborar dicha eliminación, realice la siguiente consulta:

```
148 • SELECT *
149 FROM transaction
150 WHERE id= '02C6201E-D90A-1859-B4EE-88D2986D3B02';
```

Result Grid								
Filter Rows: <input type="text"/>								
Edit:								
Export/Import:								
Wrap Cell Content:								
id	credit_card_id	company_id	user_id	lat	longitude	timestamp	amount	declined
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

2 22:33:13 SELECT \* FROM transaction WHERE id= '02C6201E-D90A-1859-B4EE-88D2986D3B02' 0 row(s) returned

- **SELECT\* FROM transaction:** declaración para consultar todos los campos existentes en la tabla *transaction*.
- **WHERE id= '02C6201E-D90A-1859-B4EE-88D2986D3B02';** coloque la cláusula WHERE para filtrar el resultado considerando que el id = '02C6201E-D90A-1859-B4EE-88D2986D3B02'.

Esta consulta no devolvió el valor que se quería eliminar, y devolvió valores nulos.

## 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 promedio de compra.

El objetivo de este ejercicio es presentar una vista estructurada llamada “VistaMarketing”, para que permita a al departamento de marketing acceder de manera rápida a la información que solicitan en el enunciado. Para ello hice la siguiente declaración:

```
155 • CREATE VIEW VistaMarketing AS
156     SELECT company_name, phone, country, avg(amount) AS Media_de_compras
157     FROM company
158     JOIN transaction
159     ON company.id=company_id
160     WHERE declined=0
161     GROUP BY 1,2,3
162     ORDER BY Media_de_compras DESC;
163
```

18 23:05:32 CREATE VIEW VistaMarketing AS SELECT company\_name, phone, country, avg(amount) AS Media\_de\_compras 0 row(s) affected

- **CREATE VIEW VistaMarketing AS:** declaración que indica la creación de una vista. En este caso es una tabla virtual llamada “Vistamarketing”
- **SELECT company\_name, phone, country, avg(amount) AS Media\_de\_compras:** aquí seleccione los campos que compondrán dicha vista de acuerdo a los requerimientos del enunciado. Los campos seleccionados son:
  - Company\_name:
  - Phone
  - Country
  - avg(amount) AS Media\_de\_compras=> este valor es el calculo de promedio de las compras realizadas por cada compañía.
- **FROM company JOIN transaction ON company.id=company\_id:** función que une la table company con la table transaction a través de la clave foránea company\_id.
- **WHERE declined=0:** cláusula WHERE para enfocar los resultados sólo a transacciones aprobadas.
- **GROUP BY 1,2,3:** agrupa los campos por compañía para poder hacer el calculo de la media de compras de cada una de ellas.
- **ORDER BY Media\_de\_compras DESC:** ordena los datos por el campo Media\_de\_compras de manera descendente, es decir de mayor a menor.

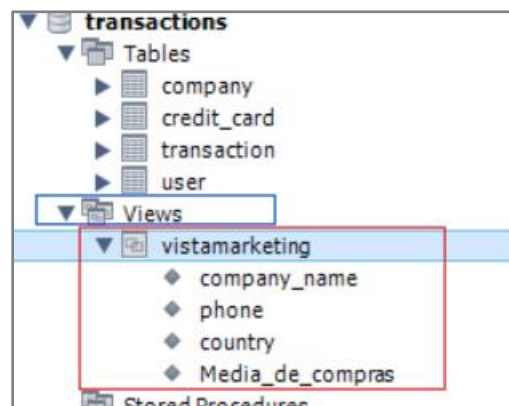
Consulta de la vista:

164 • `SELECT* FROM vistamarketing;`

	company_name	phone	country	Media_de_compras
▶	Eget Ipsum Ltd	03 67 44 ...	United States	481.860000
	Sed Id Limited	07 28 18 ...	United States	477.510000
	Neque Tellus Incorpora...	04 43 18 ...	Ireland	477.100000
	Nunc Sit Incorporated	07 28 42 ...	Norway	461.830000
	Non Magna LLC	06 71 73 ...	United Kingdom	458.740000
	Maecenas Malesuada F...	09 38 53 ...	Netherlands	451.290000
	Erat LLP	03 18 88 ...	Netherlands	448.440000
	Tortor Nunc Commodo ...	05 35 92 ...	United States	447.110000
	Justo Eu Arcu Ltd	08 42 56 ...	Italy	444.160000
	Pede Cum Ltd	07 62 26 ...	Norway	442.320000
	Vestibulum Lorem PC	02 02 87 ...	Belgium	428.400000

✓ 97 15:30:51 `SELECT* FROM vistamarketing` 101 row(s) returned

El resultado de esta declaración es la creación de una vista en la base de datos Transactions, la cual estará disponible para que el departamento de marketing la consulte y realice análisis de datos relevantes:



### EJERCICIO 3

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

Para este ejercicio, hice la siguiente declaración:

```
167 • SELECT *
168 FROM VistaMarketing
169 WHERE COUNTRY='Germany';
170
```

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

company_name	phone	country	Media_de_compras
Ac Industries	09 34 65 ...	Germany	396.150000
Auctor Mauris Corp.	05 62 87 ...	Germany	308.990000
Ac Fermentum Incorpor...	06 85 56 ...	Germany	293.570000
Aliquam PC	01 45 73 ...	Germany	280.340000
Rutrum Non Inc.	02 66 31 ...	Germany	266.900000
Nunc Interdum Incorpo...	05 18 15 ...	Germany	242.947692
Convallis In Incorporated	06 66 57 ...	Germany	60.990000
Augue Foundation	06 88 43 ...	Germany	15.050000

✓ 19 23:38:21 SELECT \* FROM VistaMarketing WHERE COUNTRY='Germany' 8 row(s) returned

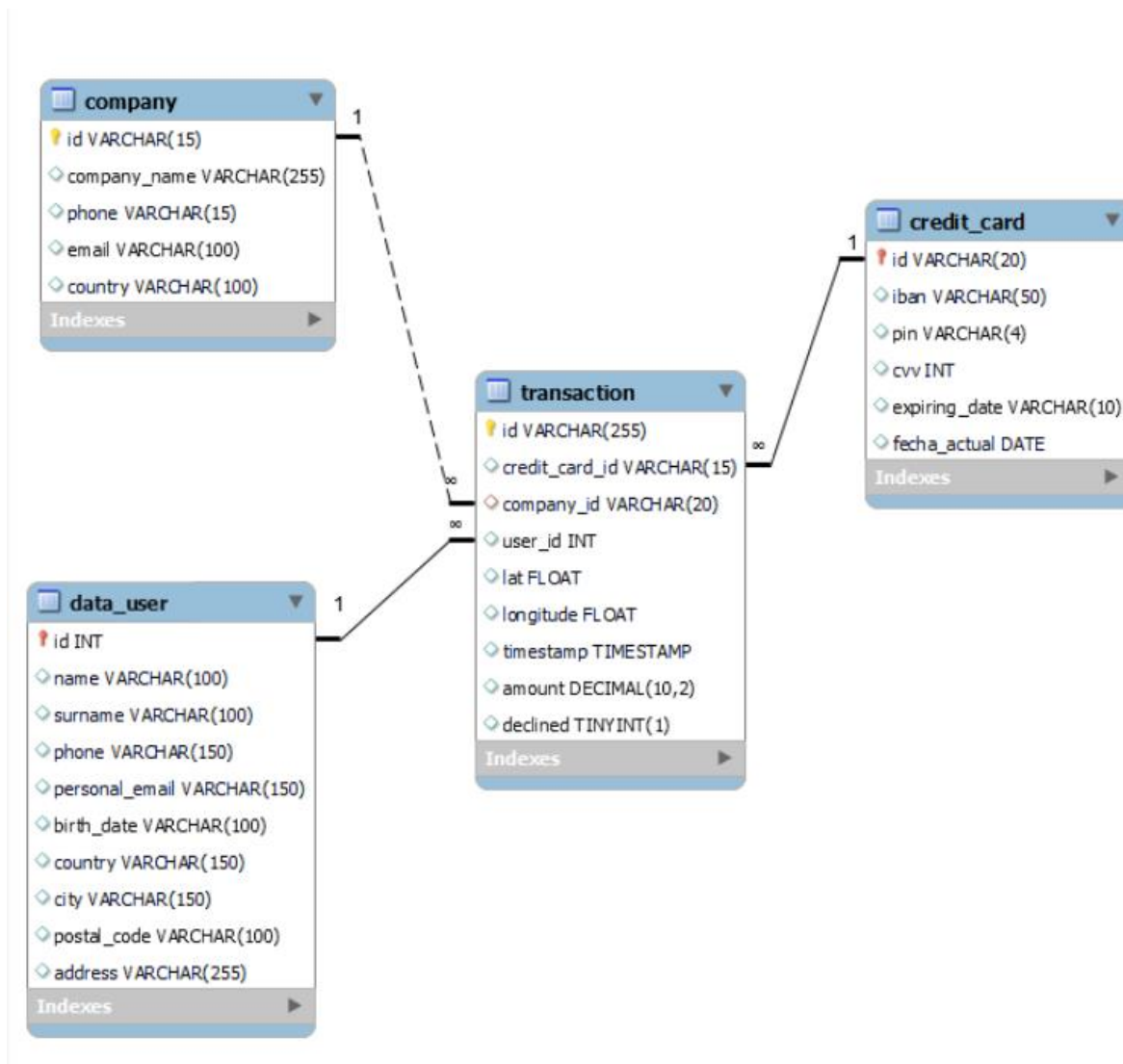
- **SELECT\*** **FROM VistaMarketing**: declaración para consultar todos los campos existentes en la vista *VistaMarketing*.
- **WHERE COUNTRY='Germany'**: cláusula Where que filtra los resultados. En este caso la condición es que se muestren aquellas compañías que se encuentran en Alemania (country='Germany').

Esta declaración da como resultado, que hay sólo 8 compañías que residen en Alemania.

## NIVEL 3

### EJERCICIO I

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 cómo las realizó. Te pide que le ayudes a dejar los comandos ejecutados para obtener el siguiente diagrama:



Para apoyar a mi compañero en la reconstrucción de las modificaciones realizadas en la base de datos *transactions*, voy a proceder paso a paso. Primero, revisaré las tablas de dimensiones (*company*, *credit\_card* y *data\_user*) y luego la tabla de hechos (*transaction*) para definir las modificaciones necesarias y crear los comandos y ejecutarlos.

Los siguientes puntos son los que aplicaré en cada una de las tablas a analizar:

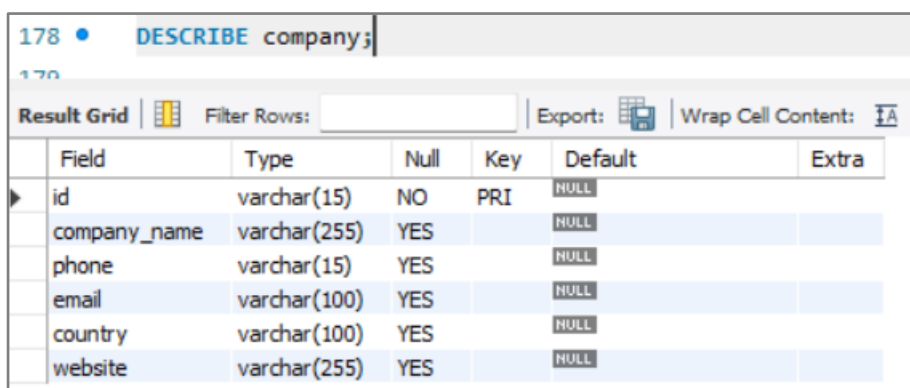
- 1- **Revisión de la estructura de la tabla:** consiste en revisar la estructura actual de las tablas.
- 2- **Comparación de estructura de tablas:** después del paso anterior, compararé la estructura actual de cada tabla con la estructura final indicada en el diagrama. Esto me permitirá identificar las diferencias entre ambas tablas.
- 3- **Definición de modificaciones y creación y ejecución de comandos:** con base a las diferencias identificadas en el paso anterior, procederé a definir las modificaciones necesarias para que las tablas de la base de datos se ajusten al diseño solicitado. Luego pasaré a crear y ejecutar los comandos necesarios para ajustar la estructura de las tablas conforme al diagrama solicitado en el enunciado.

## TABLAS DIMENSIONES

- **TABLA COMPANYY**

### I- **Revisión de la estructura de la tabla:**

Para hacer esta revisión de estructura hice la siguiente declaración:



The screenshot shows a database query window with the command `DESCRIBE company;` entered. Below the command, there is a 'Result Grid' showing the structure of the 'company' table. The grid has columns for Field, Type, Null, Key, Default, and Extra. The data is as follows:

Field	Type	Null	Key	Default	Extra
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	
website	varchar(255)	YES		NULL	

2 17:30:02 DESCRIBE company 6 row(s) returned

- **DESCRIBE company:** es una función que permite obtener información detallada de la estructura de la tabla. En este caso estamos solicitando información de la tabla *company*. Al ejecutar esta declaración, obtuve el siguiente resultado:
  - ✓ **FIELD (campos):** tiene un total de 6 columnas denominadas:
    - Id
    - Company\_name
    - Phone
    - Email country
    - Website
  - ✓ **TYPE (tipo):** los tipos de datos definidos en la tabla *company*, son los siguientes:
    - Id => VACHAR (15)
    - Company\_name => VACHAR (255)
    - Phone => VACHAR (15)
    - Email => VACHAR (100)
    - Country => VACHAR (100)
    - Website => VACHAR (255)



- ✓ NULL (nulo): el único campo que no permite valores nulos es el campo *id*. Los otros cinco (5) campos, permite valores nulos.
- ✓ KEY (clave): el campo *id* es la clave primaria (PK). Los demás no son clave.
- ✓ DEFAULT (predeterminado): todos los campos de esta tabla *company* están configurados con un valor predeterminado de NULL, lo que implica que, si no se especifica un valor al insertar un registro, la base de datos asignará automáticamente un valor nulo.
- ✓ EXTRA: en esta tabla, no se presenta información extra.

## 2- Comparación de estructura de tablas:

- Diferencias encontradas:
  - La versión final de la tabla *company* no incluye el campo *WEBSITE*, por lo que el total de campos en esta tabla es cinco (5).

## 3- Definición de modificaciones y creación y ejecución de comandos:

- Modificaciones a realizar:
  - Eliminación de la columna *WEBSITE* de la tabla *company*.
- Creación y ejecución de comandos:
  - ELIMINACION DE COLUMNA

```
182 • ALTER TABLE company
183 DROP COLUMN website;
184
```

3 18:20:37 ALTER TABLE company DROP COLUMN website 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

- **ALTER TABLE company:** declaración que indica que se quiere modificar la tabla *company*.
- **DROP COLUMN website:** función que se utiliza para eliminar una columna de una tabla. En este caso se indica que se quiere eliminar la columna *website* de la tabla mencionada anteriormente.

Para corroborar dicha eliminación de columna, hice la siguiente declaración:

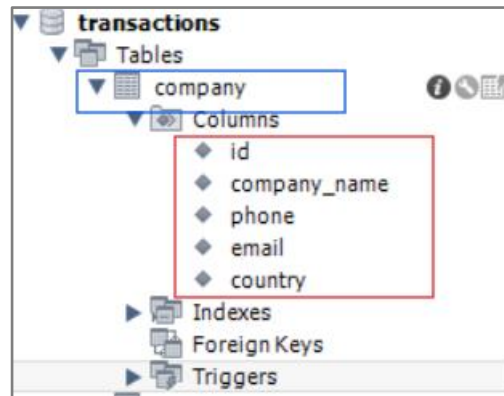
```
186 • DESCRIBE company;
187
```

Field	Type	Null	Key	Default	Extra
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	

4 18:23:10 DESCRIBE company 5 row(s) returned

- **DESCRIBE company:** es una función que permite obtener información detallada de la estructura de la tabla. En este caso estamos solicitando información de la tabla *company*. Al ejecutarla, podemos visualizar todos los campos que componen esta tabla. En este caso, la ejecución muestra que actualmente hay solo cinco campos en la tabla, y la columna *website* ya no aparece en la lista de campos. Esto confirma que el cambio se realizó correctamente.

Aquí podemos ver que la columna ya no se encuentra en la tabla *company* dentro de la base de datos *Transactions*.



- **CREDIT CARD**

- I- Revisión de la estructura de la tabla:**

Para hacer esta revisión de estructura hice la siguiente declaración:

Field	Type	Null	Key	Default	Extra
id	varchar(20)	NO	PRI	NULL	
iban	varchar(50)	YES		NULL	
pin	varchar(4)	YES		NULL	
cvv	int	YES		NULL	
expiring_date	date	YES		NULL	

✓	5	18:33:55	DESCRIBE credit_card	5 row(s) returned
---	---	----------	----------------------	-------------------

- **DESCRIBE credit\_card:** es una función que permite obtener información detallada de la estructura de la tabla. En este caso estamos solicitando información de la tabla *credit\_card*. Al ejecutar esta declaración, obtuve el siguiente resultado:
  - ✓ **FIELD (campos):** tiene un total de cinco (5) columnas denominadas:
    - Id
    - iban
    - pin
    - cvv

- `expiring_date`
- ✓ **TYPE (tipo):** los tipos de datos definidos en la tabla `credit_card`, son los siguientes:
  - `Id => VACHAR (20)`
  - `iban => VACHAR (50)`
  - `pin => INT`
  - `cvv => INT`
  - `expiring_date => date`
- ✓ **NULL (nulo):** el único campo que no permite valores nulos es el campo `id`. Los otros cuatro (4) campos, permite valores nulos.
- ✓ **KEY (clave):** el campo `id` es la clave primaria (PK). Los demás no son clave.
- ✓ **DEFAULT (predeterminado):** todos los campos de esta tabla `credit_card` están configurados con un valor predeterminado de NULL, lo que implica que, si no se especifica un valor al insertar un registro, la base de datos asignará automáticamente un valor nulo.
- ✓ **EXTRA:** en esta tabla, no se presenta información extra.

## 2- Comparación de estructura de tablas:

- Diferencias encontradas:
  - La versión final de la tabla `credit_card` en el campo `expiring_date` el tipo de dato es `VACHAR (10)`, mientras que en la tabla actual de `credit_card` el tipo de dato de `expiring_date` es `DATE`.
  - La versión final de la tabla `credit_card` hay una columna de más denominada **fecha\_actual** con tipo de dato **DATE**. Es decir, la tabla `credit_card` final tiene un total de seis (6) campos/columnas.

## 3- Definición de modificaciones y creación y ejecución de comandos:

- Modificaciones a realizar:
  - a) Alteración del tipo de dato de los campos `pin` y `expiring_date`, Los tipos de a ingresar con `VACHAR (4)` y `VACHAR (10)`, respectivamente.
  - b) Agregar una la columna **fecha\_actual** con tipo de dato **DATE**, a la presente tabla.
- Creación y ejecución de comandos:
  - a) ALTERACION DEL TIPOS DE DATOS

```
197 • ALTER TABLE credit_card
198     MODIFY COLUMN expiring_date VARCHAR (10);
```

100 10:11:36 ALTER TABLE credit\_card MODIFY COLUMN expiring\_date VARCHAR (10) 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

- **ALTER TABLE credit\_card:** cláusula que indica que se va a realizar una modificación en la estructura de la tabla `credit_card`.
- **MODIFY COLUMN expiring\_date VARCHAR (10):** especifica que la modificación se centrará en la columna `expiring_date`, transformándola en un tipo de dato `VACHAR (10)`.

b) AGREGACION DE COLUMNA

```
202 • ALTER TABLE credit_card  
203 ADD fecha_actual DATE;
```

7 19:27:49 ALTER TABLE credit\_card ADD fecha\_actual DATE 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

- **ALTER TABLE credit\_card:** cláusula que indica que se va a realizar una modificación en la estructura de la tabla credit\_card.
- **ADD fecha\_actual DATE:** especifica que se agregará (ADD) una nueva columna denominada fecha\_actual y que los datos que almacenará es de tipo DATE.

Para corroborar la modificación de los tipos de datos y la agregación de columna a la presente tabla, hice la siguiente declaración:

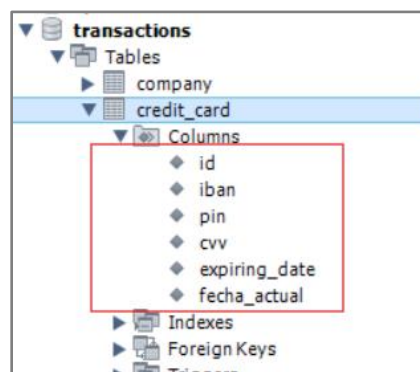
```
206 • DESCRIBE credit_card;
```

Field	Type	Null	Key	Default	Extra
id	varchar(20)	NO	PRI	NULL	
iban	varchar(50)	YES		NULL	
pin	varchar(4)	YES		NULL	
cvv	int	YES		NULL	
expiring_date	varchar(10)	YES		NULL	
fecha_actual	date	YES		NULL	

8 19:32:40 DESCRIBE credit\_card 6 row(s) returned

- **DESCRIBE credit\_card:** es una función que permite obtener información detallada de la estructura de la tabla. En este caso estamos solicitando información de la tabla credit\_card. Al ejecutarla, podemos visualizar todos los campos que componen esta tabla. En este caso, la ejecución muestra que actualmente hay seis (6) campos en la tabla, debido a la agregación del campo fecha\_actual con tipo de dato DATE y que el campo expiring\_date ha modificado su tipo de dato a VARCHAR (10) respectivamente. Esto confirma que los cambios se realizaron correctamente.

Aquí podemos ver que la nueva columna se encuentra en la tabla credit\_card dentro de la base de datos Transactions.



- **DATA USER**

### I- Revisión de la estructura de la tabla:

Para hacer esta revisión de estructura hice la siguiente declaración:

211 • DESCRIBE user;

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

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

9 19:43:32 DESCRIBE user 10 row(s) returned

➤ **DESCRIBE user:** es una función que permite obtener información detallada de la estructura de la tabla. En este caso estamos solicitando información de la tabla user. Al ejecutar esta declaración, obtuve el siguiente resultado:

- ✓ **FIELD (campos):** tiene un total de diez (10) columnas denominadas:
  - Id
  - name
  - surname
  - phone
  - email
  - birth\_date
  - country
  - city
  - postal\_code
  - address
- ✓ **TYPE (tipo):** los tipos de datos definidos en la tabla user son los siguientes:
  - Id => INT
  - Name => VARCHAR (100)
  - Surname => VARCHAR (100)
  - Phone => VARCHAR (150)
  - Email => VARCHAR (150)
  - birth\_date => VARCHAR (100)
  - country => VARCHAR (150)
  - city => VARCHAR (150)
  - postal\_code => VARCHAR (100)

- address => VARCHAR (255)
- ✓ NULL (nulo): el único campo que no permite valores nulos es el campo *id*. Los otros nueve (9) campos, permite valores nulos.
- ✓ KEY (clave): el campo *id* es la clave primaria (PK). Los demás no son clave.
- ✓ DEFAULT (predeterminado): todos los campos de esta tabla *user* están configurados con un valor predeterminado de NULL, lo que implica que, si no se especifica un valor al insertar un registro, la base de datos asignará automáticamente un valor nulo.
- ✓ EXTRA: en esta tabla, no se presenta información extra.

## 2- Comparación de estructura de tablas:

- Diferencias encontradas:
  - a) La versión final de esta tabla tiene diferente nombre. La tabla original se llama **USER** y la tabla final se llama **DATA\_USER**.
  - b) La versión final de la tabla *data\_user* modificó el nombre de la columna *email*. Ahora se denomina **personal\_email**.

## 3- Definición de modificaciones y creación y ejecución de comandos:

- Modificaciones a realizar:
  - a) Renombrar la tabla **USER** a **DATA\_USER**
  - b) Renombrar la columna **EMAIL** a **PERSONAL\_EMAIL**.

- Creación y ejecución de comandos:

- a) ALTERACION DEL NOMBRE DE UNA TABLA

```
216 • RENAME TABLE user TO data_user;
```

✓ 10 20:08:06 RENAME TABLE user TO data\_user 0 row(s) affected

- **RENAME TABLE user:** función que indica que se desea renombrar/cambiar el nombre de una tabla. En este caso, se desea cambiar el nombre de la tabla *user*.
- **TO data\_user:** es parte de la función que especifica cual es el nuevo nombre que se quiere asignar a la tabla. En este caso es *data\_user*.

- b) ALTERACION DEL NOMBRE DE UNA COLUMNA

```
220 • ALTER TABLE data_user  
221 RENAME COLUMN email TO personal_email;
```

✓ 11 20:18:55 ALTER TABLE data\_user RENAME COLUMN email TO personal\_email 0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

- **ALTER TABLE data\_user:** cláusula que indica que se va a realizar una modificación en la estructura de la tabla *data\_user*.



- **RENAME COLUMN** `email TO personal_email`; función que indica que se desea renombrar a una columna. En este caso, se quiere cambiar el nombre de la columna `email` por el nombre `personal_email`.

Para corroborar la modificación de los nombres de la tabla y de la columna a la presente tabla, hice la siguiente declaración:

```
225 • DESCRIBE data_user;  
226
```

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

12 20:23:20 DESCRIBE data\_user 10 row(s) returned

- **DESCRIBE data\_user**: es una función que permite obtener información detallada de la estructura de la tabla. En este caso estamos solicitando información de la tabla `data_user`. Al ejecutarla, podemos visualizar todos los campos que componen esta tabla. En este caso, la ejecución muestra que sigue manteniendo los diez (10) campos en la tabla, pero el campo `email` ahora se llama `personal_email`. Además, para hacer esta consulta ya tuve que utilizar el nuevo nombre de la tabla que es `data_user`. Esto confirma que los cambios se realizaron correctamente.

Aquí podemos ver que el nuevo nombre de la tabla y de la columna que se encuentra dentro de la base de datos *Transactions*.

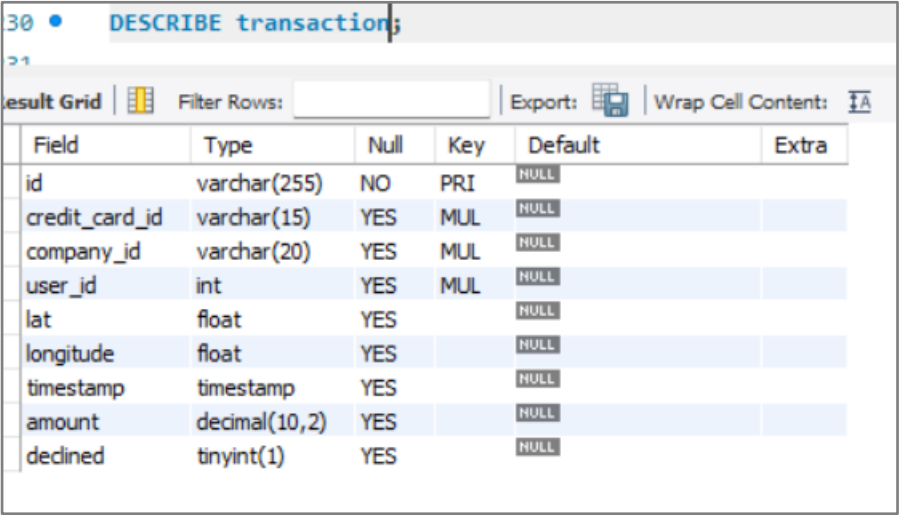
▼	transactions
▼	Tables
▶	company
▶	credit_card
▼	data_user
▼	Columns
◆	id
◆	name
◆	surname
◆	phone
◆	personal_email
◆	birth_date
◆	country
◆	city
◆	postal_code
◆	address

## TABLAS DE HECHOS

### • TRANSACTION

#### I- Revisión de la estructura de la tabla:

Para hacer esta revisión de estructura hice la siguiente declaración:



Field	Type	Null	Key	Default	Extra
id	varchar(255)	NO	PRI	<a href="#">NULL</a>	
credit_card_id	varchar(15)	YES	MUL	<a href="#">NULL</a>	
company_id	varchar(20)	YES	MUL	<a href="#">NULL</a>	
user_id	int	YES	MUL	<a href="#">NULL</a>	
lat	float	YES		<a href="#">NULL</a>	
longitude	float	YES		<a href="#">NULL</a>	
timestamp	timestamp	YES		<a href="#">NULL</a>	
amount	decimal(10,2)	YES		<a href="#">NULL</a>	
declined	tinyint(1)	YES		<a href="#">NULL</a>	

13 20:31:25 DESCRIBE transaction 9 row(s) returned

- **DESCRIBE transaction:** es una función que permite obtener información detallada de la estructura de la tabla. En este caso estamos solicitando información de la tabla *transaction*. Al ejecutar esta declaración, obtuve el siguiente resultado:

- ✓ **FIELD (campos):** tiene un total de nueve (9) columnas denominadas:
  - Id
  - Credit\_card\_id
  - Company\_id
  - User\_id
  - lat
  - longitude
  - timestamp
  - amount
  - declined
- ✓ **TYPE (tipos):** los tipos de datos definidos en la tabla *transaction* son los siguientes:
  - Id => VARCHAR (255)
  - Credit\_card\_id => VARCHAR (15)
  - Company\_id => VARCHAR (20)
  - User\_id => INT
  - Lat => FLOAT
  - Longitude => FLOAT
  - Timestamp => TIMESTAMP

- Amount => DECIMAL (10,2)
  - Declined => TINYINT (1)
- ✓ NULL (nulo): el único campo que no permite valores nulos es el campo *id*. Los otros ocho (8) campos, permite valores nulos.
- ✓ KEY (clave): informa sobre las claves primarias (PK), las claves foráneas (FK) o si la columna forma parte de un Índice (MUL). En este caso, el campo *id* es una PK y los campos *credit\_card\_id*, *company\_id* y *user\_id* son MUL, es decir que estos campos pueden tener valores duplicados.
- ✓ DEFAULT (predeterminado): todos los campos de esta tabla *transaction* están configurados con un valor predeterminado de NULL, lo que implica que, si no se especifica un valor al insertar un registro, la base de datos asignará automáticamente un valor nulo.
- ✓ EXTRA: en esta tabla, no se presenta información extra.

## 2- Comparación de estructura de tablas:

- Diferencias encontradas:
- a) La versión final de esta tabla presenta diferencias en cuanto a las relaciones con las tablas de dimensiones *data\_user* y *credit\_card*. En el diagrama final, estas relaciones son obligatorias, lo que se refleja en las líneas sólidas que las conectan. Al contrario, en nuestro modelo original, dichas relaciones eran opcionales, representadas por líneas punteadas. Esto indica que, en la versión final, cada transacción exige que tenga un usuario y una tarjeta asociada (relaciones obligatorias), mientras que en el modelo original no se requiere esa vinculación obligatoria.

## 3- Definición de modificaciones y creación y ejecución de comandos:

- Modificaciones a realizar:
- Establecer la obligatoriedad de la relación entre las tablas de dimensiones *data\_user* y *credit\_card* con la tabla de hechos *transaction*. Para lograr esto, se debe modificar la tabla *transaction*, añadiendo la restricción NOT NULL a las columnas *user\_id* y *credit\_card\_id*. Esto asegurará que no se pueda insertar un registro en la tabla *transaction* sin una referencia válida a un usuario y a una tarjeta de crédito.

- Creación y ejecución de comandos:

- a) AGREGACION DE RESTRICCIONES DE COLUMNAS

```
235 • ALTER TABLE transaction
236     MODIFY COLUMN user_id INT NOT NULL;
237
238 • ALTER TABLE transaction
239     MODIFY COLUMN credit_card_id VARCHAR(15) NOT NULL;
```

✓	24 22:13:46	ALTER TABLE transaction MODIFY COLUMN user_id INT NOT NULL	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0
✓	25 22:14:06	ALTER TABLE transaction MODIFY COLUMN credit_card_id VARCHAR(15) NOT NULL	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

- **ALTER TABLE transaction:** cláusula que indica que se va a realizar una modificación en la estructura de la tabla transaction.
- **MODIFY COLUMN user\_id INT NOT NULL:** especifica que la modificación se centrará en la columna user\_id, manteniendo su tipo de dato INT pero agregando la restricción NOT NULL.
- **MODIFY COLUMN credit\_card\_id VARCHAR (15) NOT NULL:** especifica que la modificación se centrará en la columna credit\_card\_id, manteniendo su tipo de dato VARCHAR (15) pero agregando la restricción NOT NULL.

Para corroborar las modificaciones realizadas agregando restricciones a los campos que son claves foráneas en la tabla transaction, hice la siguiente declaración:

```
242 • DESCRIBE transaction;
243
```

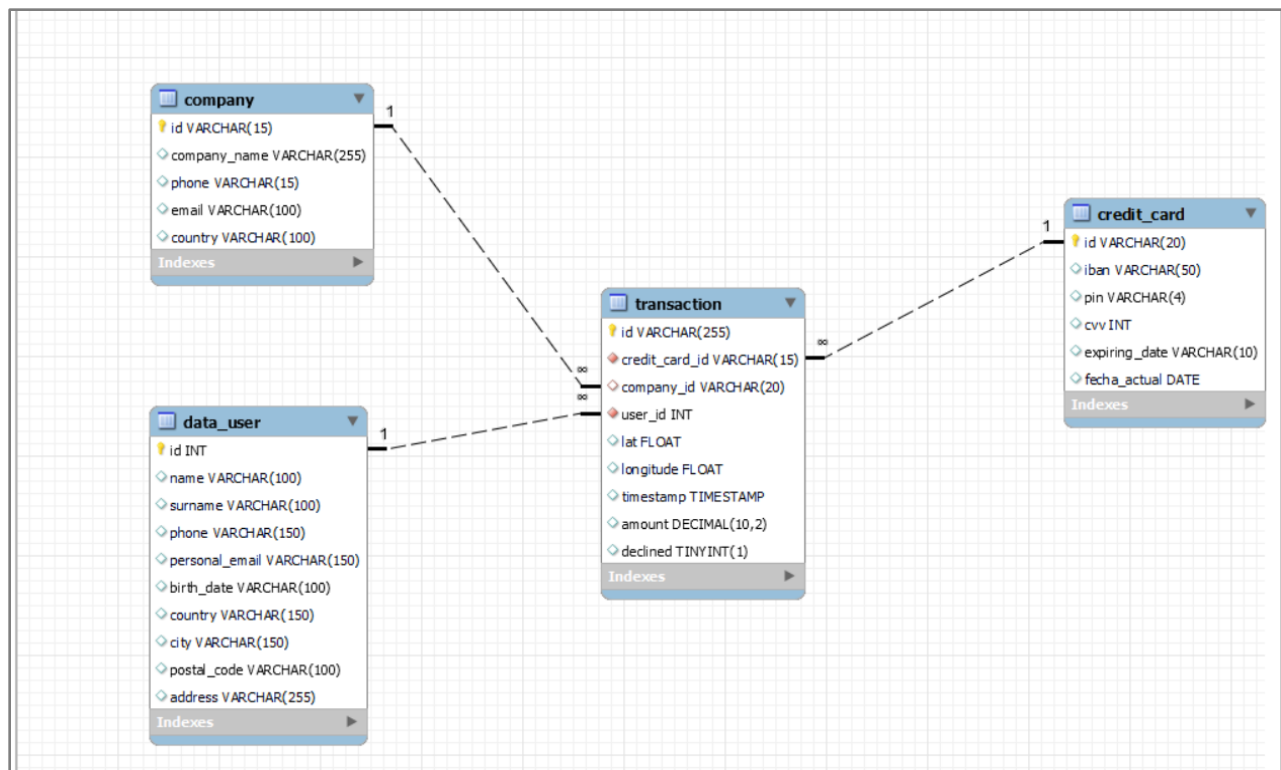
Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

Field	Type	Null	Key	Default	Extra
id	varchar(255)	NO	PRI		
credit_card_id	varchar(15)	NO	MUL		
company_id	varchar(20)	YES	MUL		
user_id	int	NO	MUL		
lat	float	YES			
longitude	float	YES			
timestamp	timestamp	YES			
amount	decimal(10,2)	YES			
declined	tinyint(1)	YES			

✓ 27 22:15:43 DESCRIBE transaction 9 row(s) returned

- **DESCRIBE transaction:** es una función que permite obtener información detallada de la estructura de la tabla. En este caso estamos solicitando información de la tabla *transaction*. Al ejecutarla, podemos visualizar todos los campos que componen esta tabla. En este caso, la ejecución muestra que sigue manteniendo los nueve (9) campos en la tabla, pero los campos credit\_card\_id y user\_id ahora tienen como restricción NOT NULL. Esto confirma que los cambios se realizaron correctamente.

Finalizado el análisis de cada tabla, podemos corroborar que las modificaciones realizadas fueron las correctas ya que en el siguiente diagrama podemos ver como cada tabla coincide con la tabla del enunciado:



- Aclaración: en el enunciado, las relaciones obligatorias están representadas por líneas sólidas, aunque en este diagrama no se reflejan de esa forma. Los diamantes de las claves foráneas `credit_card_id` y `user_id` aparecen en rojo tras modificar la obligatoriedad de la relación, mientras que `company_id` no se pinta, posiblemente indicando que su relación no es obligatoria. Esta diferencia en líneas y colores de diamantes podría deberse a una característica de la versión de MySQL Workbench que estoy utilizando.

## EJERCICIO 2

La empresa también te solicita crear una vista llamada "Informe Técnico" que contenga la siguiente información:

**ID de la transacción**

**Nombre del usuario/a**

**Apellido del usuario/a**

**IBAN de la tarjeta de crédito usada.**

**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 forma descendente en función de la variable ID de transacción.**

Para resolver el ejercicio, realice la siguiente declaración:

```
266 • CREATE VIEW InformeTecnico AS
267 SELECT transaction.id AS ID_Transaction, amount as Amount_transaction, timestamp AS Fecha_Hora, declined , name AS Name_user,
    surname as Surname_user, iban AS Iban_tarjeta, data_user.country as Country_user, email as Emails_user, company_name, company.
    country AS Country_company
268 FROM transaction
269 JOIN data_user
270 ON user_id=data_user.id
271 JOIN credit_card
272 ON credit_card_id= credit_card.id
273 JOIN company
274 ON company_id=company.id
275 ORDER BY 1 DESC;
```

94 15:00:39 CREATE VIEW InformeTecnico AS SELECT transaction.id AS ID\_Transaction, amount as Amount\_tra... 0 row(s) affected

- **CREATE VIEW InformeTecnico AS:** declaración que indica la creación de una vista. En este caso es una tabla virtual llamada “InformeTecnico”
- **SELECT transaction.id AS ID\_Transaction, amount as Amount\_transaction, timestamp AS Fecha\_Hora, declined , name AS Name\_user, surname as Surname\_user, iban AS Iban\_tarjeta, data\_user.country as Country\_user, email as Emails\_user, company\_name, company.country AS Country\_company:** aquí seleccione los campos que compondrán dicha vista de acuerdo a los requerimientos del enunciado. Los campos seleccionados son:
  - Transaction.id => la columna la renombre con la cláusula AS ID\_Transaction
  - Amount => la columna la renombre con la cláusula AS Amount\_transaction
  - Timestamp => la columna la renombre con la cláusula AS Fecha\_Hora
  - Declined
  - Name => la columna la renombre con la cláusula AS Name\_user
  - Surname => la columna la renombre con la cláusula as Surname\_user
  - Iban => la columna la renombre con la cláusula AS Iban\_tarjeta
  - Data\_user.country Iban => la columna la renombre con la cláusula as Country\_user
  - Email => la columna la renombre con la cláusula as Emails\_user
  - Company\_name
  - Company.country => la columna la renombre con la cláusula AS Country\_company
- **FROM transaction JOIN data\_user ON user\_id=data\_user.id:** función que une la tabla transaction con la tabla data\_user a través de la clave foránea user\_id.
- **JOIN credit\_card ON credit\_card\_id= credit\_card.id:** función que une la tabla transaction con la tabla credite\_card a través de la clave foránea credit\_card\_id.
- **JOIN company ON company\_id=company.id:** función que une la tabla transaction con la tabla company a través de la clave foránea company\_id.
- **ORDER BY 1 DESC:** ordena los datos por el campo ID\_Transaction de manera descendente, es decir de mayor a menor.

Los resultados de la vista son los siguientes:

```
SELECT* FROM InformeTecnico;
```

ID_Transaction	Amount_transaction	Fecha_Hora	declined	Name_user	Surname_user	Iban_tarjeta
FE96CE47-8D59-381C-4E18-E3CA3D44E8FF	480.13	2021-06-15 00:26:29	1	Kenyon	Hartman	DO26854763748537475216568689
FE809ED4-2DB6-55AC-C915-929516E4646B	219.83	2021-11-09 21:35:40	0	Molly	Gilliam	SE2813123487163628531121
FD9CBCCD-8E1E-8DA1-4606-7E3A6F3A5A65	42.32	2021-06-13 11:41:17	0	Linus	Willis	KW9485332754781757886242955643
FD89D51B-AE8D-77DC-E450-B8083FBD3187	200.72	2022-03-16 02:35:05	0	Hilda	Levy	LT053237077744561475
FD2E8957-414B-8EEC-E9AD-59AA7A8A6290	78.29	2022-03-13 00:27:34	0	Hedwig	Gilbert	GE84848451582810541526
FCE2AB9A-271D-2BDC-9E49-8DD92A373391	335.56	2022-02-06 22:48:41	0	Hakeem	Alford	MD1234119525145401270486
FBD7E0D6-8A68-F5BC-0CA9-EA4B8760100C	207.09	2021-04-29 14:17:50	1	Hedwig	Gilbert	MU4132333444534342541344788855
FAC76A80-8448-69AA-E892-426C2F12621C	304.95	2021-05-30 21:10:55	0	Slade	Poole	MT05JWCF58868200575771634583813
FAAD3FFC-1A17-E141-43D3-359A5BA7CB3B	149.84	2021-10-24 20:16:23	0	Hedwig	Gilbert	GE90157928843338134463
FA053936-75D8-85FA-490D-9B624E1B920A	151.32	2021-07-06 10:18:35	0	Hedwig	Gilbert	GT02497653655330848247645975

Country_user	Emails_user	company_name	Country_company
Canada	risus.donec.nibh@icloud.org	Magna A Neque Industries	Australia
United Kingdom	non@outlook.com	Nunc Interdum Incorporated	Germany
Canada	non@outlook.com	Nunc Interdum Incorporated	Germany
Canada	cras.lorem.lorem@outlook.com	Malesuada PC	Ireland
Canada	vestibulum.ante.ipsum@aol.edu	Neque Tellus Imperdiet Corp.	Ireland
United Kingdom	non@outlook.com	Nunc Interdum Incorporated	Germany
Canada	bibendum.sed.est@yahoo.net	Mauris Id Inc.	Ireland
Canada	dui@aol.ca	Arcu LLP	Norway
Canada	enim.gravida.sit@hotmail.net	Lorem Eu Incorporated	Canada
Canada	urna.ut.tincidunt@yahoo.edu	Non Justo Corp.	Sweden

95 15:02:05 SELECT\* FROM InformeTecnico 587 row(s) returned

El resultado de esta declaración es la creación de una vista en la base de datos Transactions, la cual estará disponible para consultar y realizar los análisis de datos relevantes:

Views
informetecnico
ID_Transaction
Amount_transaction
Fecha_Hora
declined
Name_user
Surname_user
Iban_tarjeta
Country_user
Emails_user
company_name
Country_company
vistamarketing