

## SPRINT 3 – Data Analytics

A continuación, se presentan los resultados del Sprint #3 de la especialización Data Analytics de IT Academy, realizados por Rossemayr Castellanos entregado el día 01/10/2025. Se realizaron los 3 niveles del Sprint.

En este sprint se simula una situación empresarial en la que deberás realizar diversas manipulaciones en las tablas de una base de datos. Además, trabajarás con índices y vistas para optimizar consultas y organizar la información.

Continuarás trabajando con la base de datos que contiene información de un marketplace, un entorno similar a Amazon donde varias empresas venden sus productos a través de un canal online. En esta actividad, empezarás a trabajar con datos relacionados con tarjetas de crédito.

Añade las tablas al modelo según corresponda:

- Nivel 1: Tabla "credit\_card"
- Nivel 3: Tabla "user"

Observación: Para este Sprint no se considero diferenciar entre transacciones exitosas (declined = 0) y las transacciones no finalizadas o declinadas (declined = 1) .

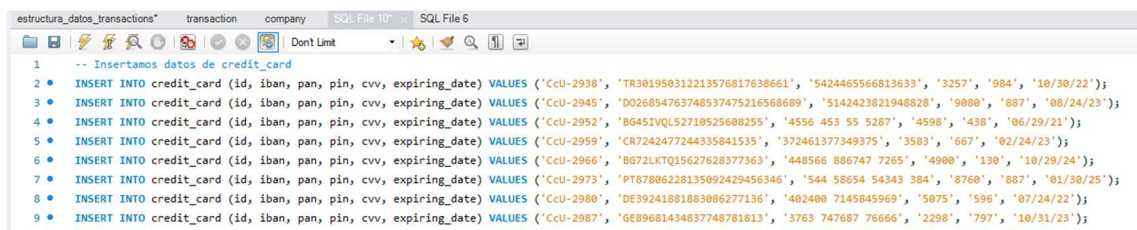
### NIVEL 1

#### 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 debe ser capaz de identificar de forma 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 del mismo.

En la base de datos **transactions** se hace referencia al pago con tarjeta de crédito, este campo se identifica como **credit\_card\_id** en la tabla **transaction**, por lo que podemos sugerir que este sería la Primary Key en la futura tabla **credit\_card** y en la tabla **transaction** es la Foreign Key que vincula ambas tablas.

Para diseñar la tabla debemos ver el archivo "datos\_introducir\_credit", esto para conocer el tipo de datos que contiene y establecer formato para cada campo.



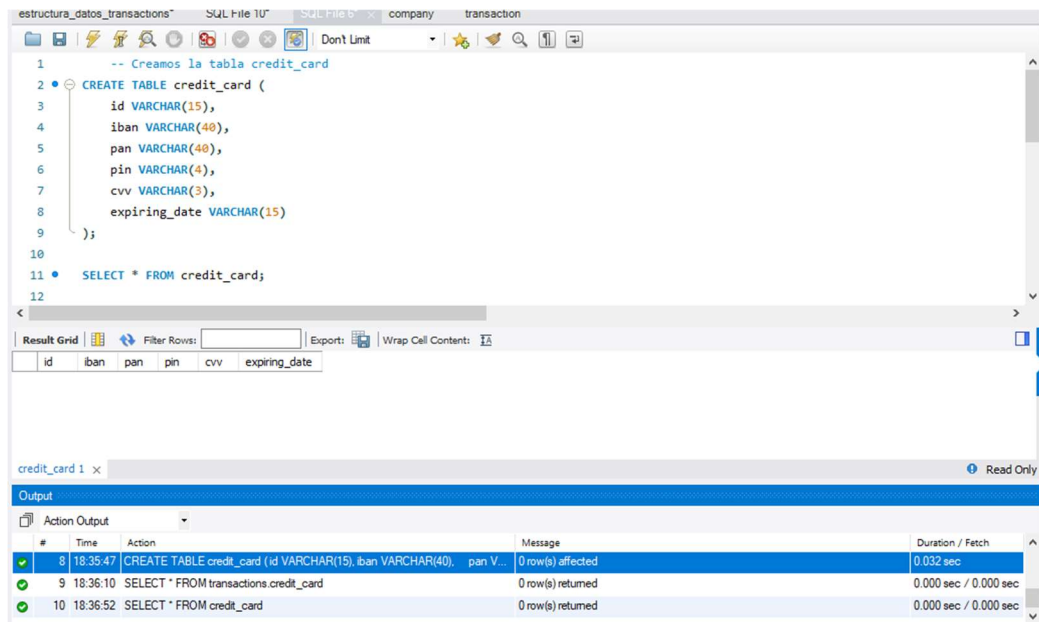
```
estructura_datos_transactions* transaction company SQL File 10 SQL File 6
1 -- Insertamos datos de credit_card
2 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ('CcU-2938', 'TR301950312213576817638661', '5424465566813633', '3257', '984', '10/30/22');
3 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ('CcU-2945', 'D026854763748537475216568689', '5142423821948828', '9080', '887', '08/24/23');
4 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ('CcU-2952', 'B645IVQL52710525608255', '4556 453 55 5287', '4598', '438', '06/29/21');
5 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ('CcU-2959', 'CR7242477244335841535', '372461377349375', '3583', '667', '02/24/23');
6 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ('CcU-2966', 'B672LKTQ15627628377363', '448566 886747 7265', '4900', '130', '10/29/24');
7 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ('CcU-2973', 'PT87806228135092429456346', '544 58654 54343 384', '8760', '887', '01/30/25');
8 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ('CcU-2980', 'DE39241881883086277136', '402400 7145845969', '5075', '596', '07/24/22');
9 INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES ('CcU-2987', 'GE89681434837748781813', '3763 747687 76666', '2298', '797', '10/31/23');
```

Como se observa en la imagen se cuenta con 6 campos

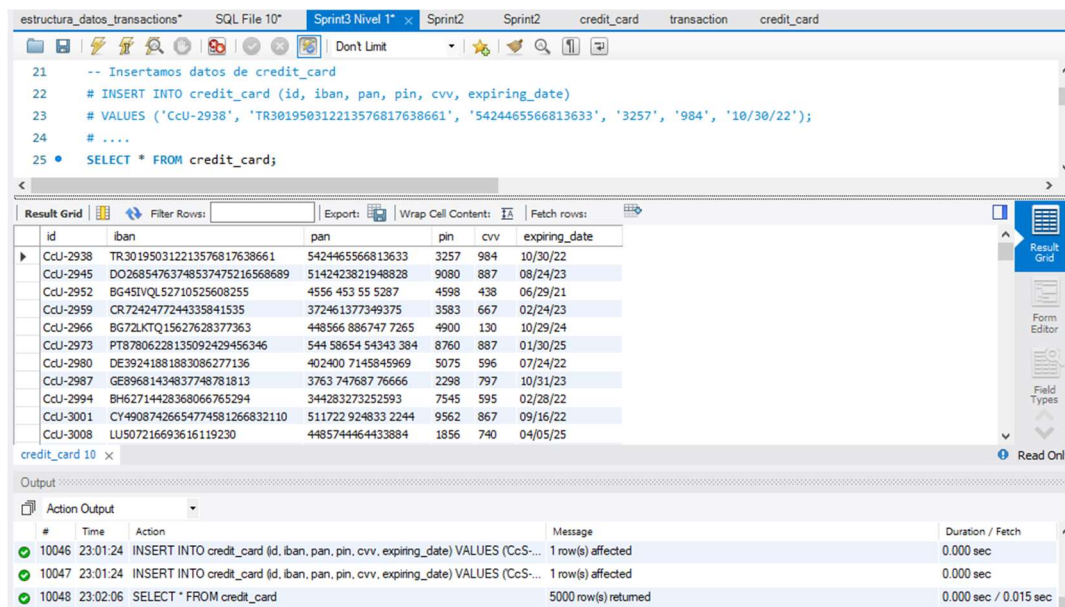
- **id**: número identificador del código de la tarjeta que sería la clave primaria y única, debemos revisar que el formato coincida con **credit\_card\_id** de la tabla **transaction**.

- *iban*: alfanumérico, corresponde a la cuenta iban asociado a la tarjeta,
- *pan*: numeración desconocida de momento.
- *pin*: código clave de la tarjeta.
- *cvv*: código de seguridad de la tarjeta.
- *expiring\_date*: fecha de caducidad de la tarjeta.

De la observación anterior se definió la siguiente estructura de datos.

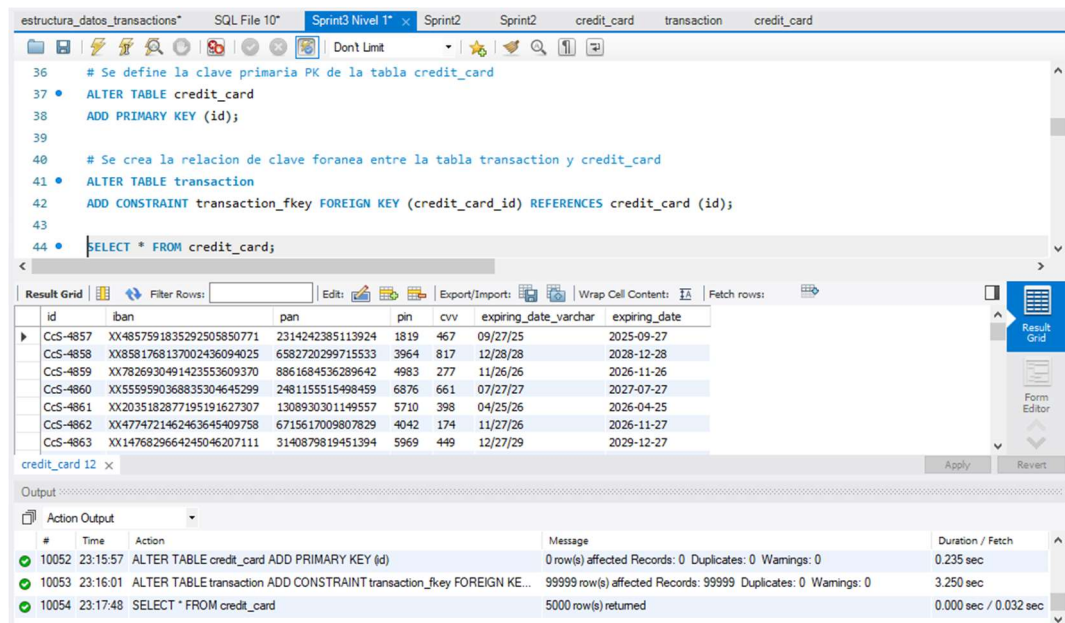


A continuación, se introducen los datos del archivo "*datos\_introducir\_credit*" y se ejecuta el script para luego visualizar los datos cargados en la tabla **credit\_card**.

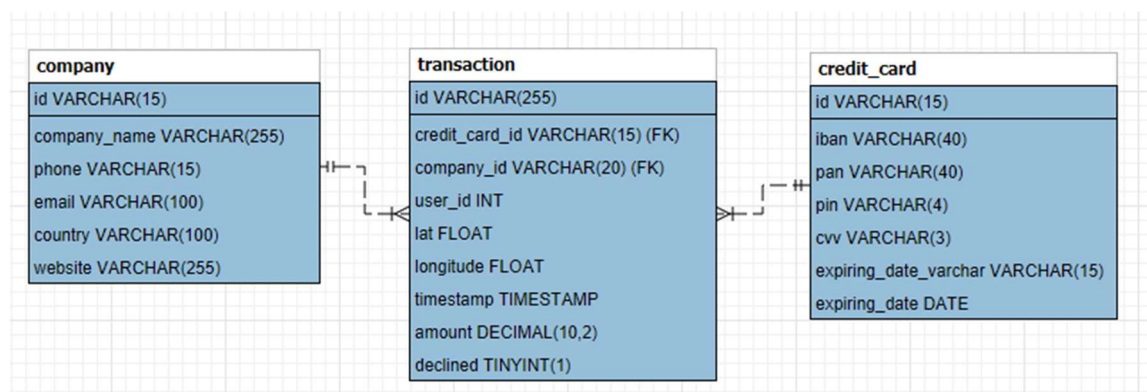


Ahora se establecen las claves primaria y foránea que definirán la relación de la tabla **credit\_card** con el resto de las tablas. Como se menciona anteriormente el campo *id* de la tabla **credit\_card** es la que vincula aparentemente a través de la tabla **transaction** con la base de datos por medio del campo *credit\_card\_id*. En la imagen se presentan las sentencias que permiten definir la

Primary Key (**credit\_card.id**) y la Foreign Key (**transaction.credit\_card\_id**) como vínculo entre ambas tablas (**credit\_card** y **transaction**). Se utilizó el comando ALTER TABLE y ADD para establecer ambas claves y las restricciones para referenciar a la tabla **credit\_card**.



En la figura anexa se presenta el diagrama entidad-relación generado para la base de datos **transactions**, con la adición de la nueva tabla **credit\_card**.



11. Diagrama Entidad - Relación para base de datos **transactions** NIVEL 1

La clave que vincula ambas tablas es el código de la tarjeta de crédito siendo clave primaria en la tabla **credit\_card** (**id**) y clave foránea en la tabla **transaction** (**credit\_card\_id**).

La relación entre ambas tablas es de 1: N, de 1 a muchos similar a la relación que tiene la tabla **transaction** con la tabla **company**. Si se analiza la relación una tarjeta puede realizar múltiples compras (transacciones) registradas en la tabla **transaction**, sin embargo, una venta o transacción es cargada a una única tarjeta de la tabla **credit\_card**.

## Ejercicio 2

El departamento de Recursos Humanos ha identificado un error en el número de cuenta asociado a su tarjeta de crédito con ID CcU-2938. La información que debe mostrarse para este registro es: TR323456312213576817699999. Recuerda mostrar que el cambio se realizó.

```
estructura_datos_transactions* SQL File 10* Sprint3 Nivel 1* Sprint2 Sprint2 credit_card transaction credit_card
52 ## Ejercicio 2
53 # El departamento de Recursos Humanos ha identificado un error en el número de cuenta asociado
54 # a su tarjeta de crédito con ID CcU-2938. La información que debe mostrarse para este registro
55 # es: TR323456312213576817699999. Recuerda mostrar que el cambio se realizó.
56 • UPDATE credit_card
57 SET iban = "TR323456312213576817699999"
58 WHERE id = "CcU-2938";
59
60 • SELECT id, iban
61 FROM credit_card
62 WHERE id = "CcU-2938";
```

Result Grid

id	iban
CcU-2938	TR323456312213576817699999
NULL	NULL

credit\_card 13 x

Output

#	Time	Action	Message	Duration / Fetch
10055	23:20:31	UPDATE credit_card SET iban = "TR323456312213576817699999" WHERE id = "CcU-2938";	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.016 sec
10056	23:20:34	SELECT id, iban FROM credit_card WHERE id = "CcU-2938";	1 row(s) returned	0.000 sec / 0.000 sec

En este ejercicio se busca modificar el registro de una cuenta asociado a la tarjeta "CcU-2938", para aplicarlo se utiliza el comando UPDATE junto con WHERE para filtrar el registro a actualizar, adicionalmente se aplica un SELECT para comprobar que el cambio en la cuenta asociada a la tarjeta fue ejecutado.

## Ejercicio 3

En la tabla "transaction" ingresa una nueva transacción con la siguiente información:

Id	108B1D1D-5B23-A76C-55EF-C568E49A99DD
credit card id	CcU-9999
company id	b-9999
user id	9999
lato	829.999
longitud	-117.999
amunt	111.11
declined	0

En este ejercicio se debe ingresar este nuevo registro en la tabla **transaction** a través del comando INSERT INTO. El problema es que genera error por integridad referencial principio que garantiza la validez de las relaciones entre tablas, este concepto evita inconsistencias, como la creación de registros con referencia a datos que no existen, es decir, una Foreign Key debe corresponder aun valor valido en la Primary Key de otra tabla.



En este caso tanto *credit\_car\_id* como *company\_id* son Foreign Key en la tabla **transaction**, el error ocurre porque estos datos no existen en las tablas **credit\_card** y **company**, respectivamente. Primero se deben crear ambos registros en ambas tablas por medio del INSERT INTO. Después si se puede insertar el registro nuevo en la tabla **transaction** sin problemas.

```

64 # Ejercicio 3
65 # En la tabla "transaction" ingresa una nueva transacción con la siguiente información:
66 # Id 10881D1D-5B23-A76C-55EF-C568E49A99DD # credit_card_id CcU-9999 # company_id b-9999 # user_id 9999
67 # lat 829.999 # longitude -117.999 # amount 111.11 # declined 0
68 • INSERT INTO company (id) VALUES ('b-9999'); -- Introducir el registro en la tabla company (integridad referencial)
69 • INSERT INTO credit_card (id) VALUES ('CcU-9999'); -- Introducir el registro en la tabla credit_card (integridad referencial)
70 • INSERT INTO transaction (id, credit_card_id, company_id, user_id, lat, longitude, timestamp, amount, declined)
71 VALUES ('10881D1D-5B23-A76C-55EF-C568E49A99DD', 'CcU-9999', 'b-9999', '9999', '829.999', '-117.999', NOW(), '111.11', '0');
72 • SELECT * FROM transaction WHERE company_id = 'b-9999';
  
```

id	credit_card_id	company_id	user_id	lat	longitude	timestamp	amount	declined
10881D1D-5B23-A76C-55EF-C568E49A99DD	CcU-9999	b-9999	9999	829.999	-117.999	2025-09-25 23:31:53	111.11	0

#	Time	Action	Message	Duration / Fetch
10061	23:30:38	INSERT INTO company (id) VALUES ('b-9999')	1 row(s) affected	0.000 sec
10062	23:30:48	INSERT INTO transaction (id, credit_card_id, company_id, user_id, lat, longitude, timestamp, amount, declined)	Error Code: 1452. Cannot add or update a child row: a foreign ...	0.000 sec
10063	23:31:48	INSERT INTO credit_card (id) VALUES ('CcU-9999')	1 row(s) affected	0.000 sec
10064	23:31:53	INSERT INTO transaction (id, credit_card_id, company_id, user_id, lat, longitude, timestamp, amount, declined)	1 row(s) affected	0.016 sec
10065	23:32:16	SELECT * FROM transaction WHERE company_id = 'b-9999'	1 row(s) returned	0.000 sec / 0.000 sec

#### Ejercicio 4

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

```

79 # Ejercicio 4
80 # Desde recursos humanos te solicitan eliminar la columna "pan" de la tabla credit_card.
81 # Recuerda mostrar el cambio realizado.
82
83 • ALTER TABLE credit_card
84 DROP COLUMN pan;
85
86 • SELECT * FROM credit_card;
87
  
```

id	iban	pin	cvv	expiring_date_varchar	expiring_date
CcS-4857	XX4857591835292505850771	1819	467	09/27/25	2025-09-27
CcS-4858	XX8581768137002436094025	3964	817	12/28/28	2028-12-28
CcS-4859	XX7826930491423553609370	4983	277	11/26/26	2026-11-26
CcS-4860	XX5559590368835304645299	6876	661	07/27/27	2027-07-27
CcS-4861	XX2035182877195191627307	5710	398	04/25/26	2026-04-25
CcS-4862	XX4774721462463645409758	4042	174	11/27/26	2026-11-27
CcS-4863	XX1476829664245046207111	5969	449	12/27/29	2029-12-27
CcS-4864	XX8380298893385731196159	8481	139	02/28/26	2026-02-28

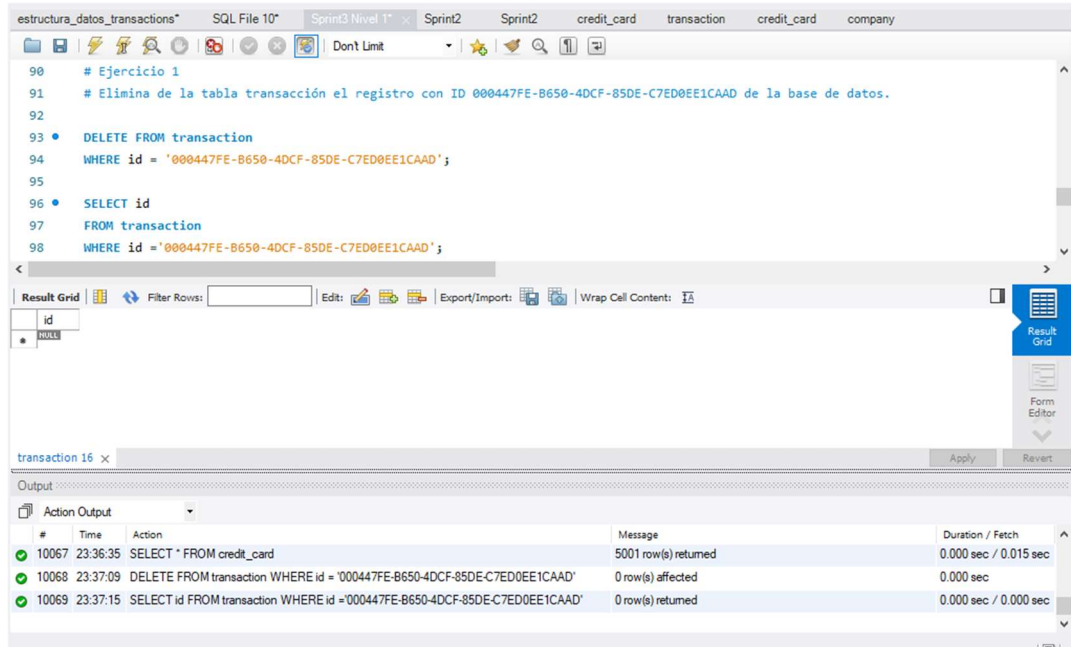
#	Time	Action	Message	Duration / Fetch
10065	23:32:16	SELECT * FROM transaction WHERE company_id = 'b-9999'	1 row(s) returned	0.000 sec / 0.000 sec
10066	23:36:31	ALTER TABLE credit_card DROP COLUMN pan	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.047 sec
10067	23:36:35	SELECT * FROM credit_card	5001 row(s) returned	0.000 sec / 0.015 sec

En este ejercicio se modifica la tabla **credit\_card** eliminando con el comando DROP la columna "pan", luego con el SELECT se verifica la eliminación del campo.

## NIVEL 2

### Ejercicio 1

Elimina de la tabla transacción el registro con ID 000447FE-B650-4DCF-85DE-C7ED0EE1CAAD de la base de datos.



En este ejercicio se elimina con el comando DELETE un registro de la tabla **transaction** correspondiente al *id* del enunciado el cual se ubica usando el comando WHERE, luego con el SELECT se verifica la eliminación del registro, filtrando a través de la búsqueda de *id*.

### 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.

En este ejercicio se debe generar la consulta solicitada para al inicio de la misma aplicar el comando CREATE VIEW con em nombre que del enunciado “**VistaMarketing**”. Las vistas funcionan como tablas virtuales que pueden ser consultadas como otra tabla de la base de datos, para ello se llaman a través del FROM al igual que el resto de tablas.

En la consulta se unen datos de la tabla **transaction** y **company** usando el comando JOIN y ON, luego se agrupa por empresas con el comando GROUP BY y para finalmente reflejar su promedio de ventas (*avg\_amount*), priorizando las empresas con mayores ventas con el comando ORDER BY.

Al realizar la sentencia genero error por el uso de ORDER BY, la documentación indica que se puede usar en la creación de la vista, pero no será ejecuta dentro de la misma, por lo que sugiere

aplicar el ORDER BY en la consulta de la vista. En este caso se indicó en la vista para mostrar los procesos aplicados a los datos y también el en SELECT al realizar la consulta de la vista “VistaMarketing”.

The screenshot shows the SQL Server Enterprise Manager interface. The top pane displays the SQL script for creating and querying the VistaMarketing view. The bottom pane shows the results of the execution.

```
87 # Ejercicio 2
88 # La sección de marketing desea tener acceso a información específica para realizar análisis y estrategias efectivas. Se ha solicitado
89 # crear una vista que proporcione detalles clave sobre las compañías y sus transacciones. Será necesaria que crees una vista
90 # llamada VistaMarketing que contenga la siguiente información: Nombre de la compañía. Teléfono de contacto. País de residencia.
91 # Media de compra realizado por cada compañía. Presenta la vista creada, ordenando los datos de mayor a menor promedio de compra.
92 • CREATE VIEW VistaMarketing AS
93 SELECT c.company_name, c.phone, c.country, ROUND(AVG(t.amount), 2) AS avg_amount
94 FROM company AS c
95 LEFT JOIN transaction AS t
96 ON c.id = t.company_id
97 GROUP BY c.company_name, c.phone, c.country
98 ORDER BY avg_amount DESC;
99 • SELECT * FROM VistaMarketing ORDER BY avg_amount DESC;
```

company_name	phone	country	avg_amount
Ac Fermentum Incorporated	06 85 56 52 33	Germany	284.87
Pretium Neque Corp.	07 77 48 55 28	Australia	276.16
Urnna Convallis Associates	06 01 24 77 04	United States	274.24
At Associates	09 56 61 10 65	New Zealand	272.21
Metus Vitae Associates	08 25 44 40 66	Australia	270.08

Output

#	Time	Action	Message	Duration / Fetch
24	19:35:06	CREATE VIEW VistaMarketing AS SELECT c.company_name, c.phone, c.country, ROUND(AVG(t.amount), 2) A...	0 row(s) affected	0.000 sec
25	19:35:12	SELECT * FROM VistaMarketing ORDER BY avg_amount DESC	101 row(s) returned	0.609 sec / 0.000 sec

### Ejercicio 3

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

The screenshot shows the SQL Server Enterprise Manager interface. The top pane displays the SQL script for filtering the VistaMarketing view. The bottom pane shows the results of the execution.

```
115
116 # Ejercicio 3
117 # Filtra la vista VistaMarketing para mostrar sólo las compañías que tienen su país de residencia en "Germany"
118
119 • SELECT *
120 FROM VistaMarketing
121 WHERE country = 'Germany';
```

company_name	phone	country	avg_amount
Ac Fermentum Incorporated	06 85 56 52 33	Germany	284.87
Nunc Interdum Incorporated	05 18 15 48 13	Germany	259.32
Convallis In Incorporated	06 66 57 29 50	Germany	257.75
Ac Industries	09 34 65 40 60	Germany	255.15
Rutrum Non Inc.	02 66 31 61 09	Germany	255.14
Auctor Mauris Corp.	05 62 87 14 41	Germany	254.77
Augue Foundation	06 88 43 15 63	Germany	253.51
Aliquam PC	01 45 73 52 16	Germany	253.14

Output

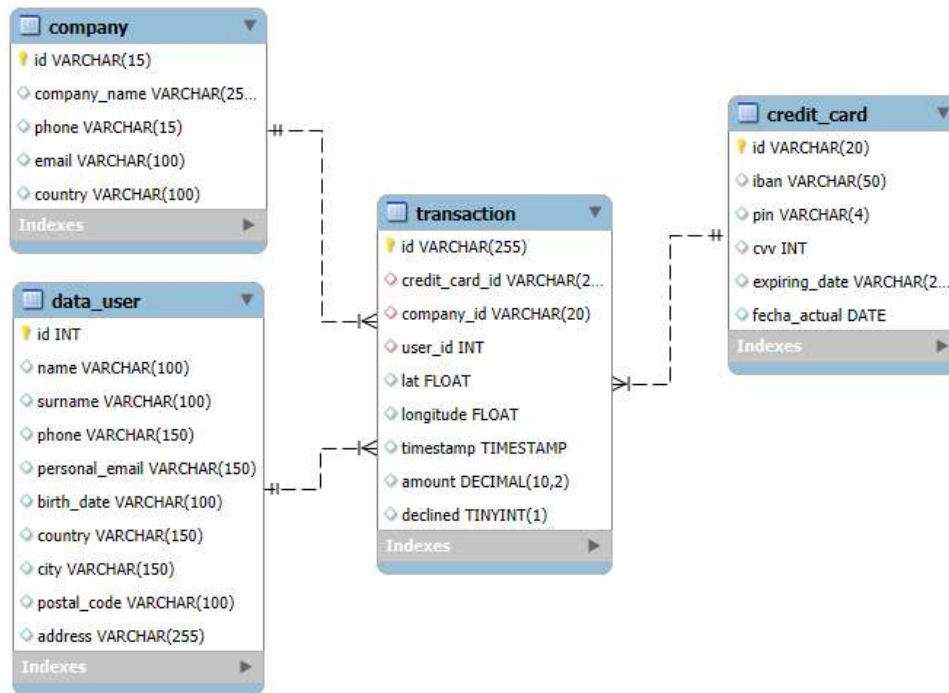
#	Time	Action	Message	Duration / Fetch
10071	23:38:52	CREATE VIEW VistaMarketing AS SELECT c.company_name, c.phone, c.country, ROUND(A...	0 row(s) affected	0.016 sec
10072	23:38:58	SELECT * FROM VistaMarketing	101 row(s) returned	0.641 sec / 0.000 sec
10073	23:40:09	SELECT * FROM VistaMarketing WHERE country = 'Germany'	8 row(s) returned	0.140 sec / 0.000 sec

En este ejercicio se realiza una consulta sobre vista “VistaMarketing” creada en el ejercicio anterior. En este caso se solicita filtrar las empresas alemanas de la vista, por tal motivo se aplica el comando WHERE para indicar al país (“country”) como Alemania (“Germany”).

## 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 cómo las realizó. Te pide que le 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 deberás trabajar con los archivos denominados "estructura\_datos\_user" y "datos\_introducir\_user"

Recuerda seguir trabajando sobre el modelo y las tablas con las que ya has trabajado hasta ahora.

Este ejercicio se divide en 2 partes:

- 1.- Cargar en la base de datos **transactions** la tabla **data\_user** ya que no esta y
- 2.- Aplicar las sentencias necesarias para imitar el diagrama propuesto en el enunciado.

Se inicia con la creación de la tabla **user** dentro de la base de datos a través de los 2 archivos: "estructura\_datos\_user.sql". y "datos\_introducir\_sprint3\_user.sql".

Primero se ejecuta el script con la estructura para dar los parámetros de los campos que componen la tabla **user** y luego se ejecuta el script del archivo de datos a introducir que contiene los registros para llenar la tabla **user**.

En la imagen a continuación se aprecia los campos indicados en la estructura de los datos de la tabla de nombre: **user** donde:

- **id**: número identificador del código del usuario, asignado como Primary Key, se debe verificar que el formato coincida con **user\_id** de la tabla **transaction** y establecer las restricciones y relaciones pertinentes (Foreign Key).
- **name**: nombre del usuario,



- *surname*: apellido del usuario,
- *phone*: teléfono del usuario,
- *email*: correo electrónico del usuario,
- *birth\_date*: fecha de nacimiento del usuario,
- *country*: país donde se localiza o reside el usuario,
- *city*: ciudad donde se localiza o reside el usuario,
- *postal\_code*: código postal del lugar donde se localiza o reside el usuario y
- *address*: dirección del sitio donde se localiza o reside el usuario.

```

1  # NIVEL 3
2  # Ejercicio 1
3  CREATE TABLE IF NOT EXISTS user (
4      id CHAR(10) PRIMARY KEY,
5      name VARCHAR(100),
6      surname VARCHAR(100),
7      phone VARCHAR(150),
8      email VARCHAR(150),
9      birth_date VARCHAR(100),
10     country VARCHAR(150),
11     city VARCHAR(150),
12     postal_code VARCHAR(100),
13     address VARCHAR(255)
14 );
  
```

Output

Date	Time	SQL
2025-09-30	00:19:29	CREATE TABLE IF NOT EXISTS user (id CHAR(10) PRIMARY KEY, name VARCHAR(100), surname VARCHAR(100), phone VARCHAR(150), em...

Ahora se muestra en la imagen de un SELECT que permite visualizar como queda la tabla **user** después de ejecutar el segundo script para la carga de datos dentro de la tabla.

```

130 # Ejercicio 1
131 # La próxima semana tendrás una nueva reunión con los gerentes de marketing. Un compañero de tu equipo realizó
132 # modificaciones en la base de datos, pero no recuerda cómo las realizó. Te pide que le ayudes a dejar
133 # los comandos ejecutados para obtener el siguiente diagrama anexo en el informe
134
135 -- Se ejecutan los script entregados de tabla user ("estructura datos user.sql", "datos introducir sprint3 user.sql")
136
137 SELECT * FROM user;
138
  
```

id	name	surname	phone	email	birth_date	country	city	postal_code	address
1	Zeus	Gamble	1-282-581-0551	interdum.enim@protonmail.edu	Nov 17, 1985	United States	New York	10001	348-7818 Sagittis St.
10	Robert	Mccarthy	(324) 746-6771	fermentum@protonmail.com	Apr 30, 1984	United States	San Jose	95101	P.O. Box 773
100	Melodie	Mclean	1-677-221-7152	risus.varius@google.ca	Sep 15, 1989	United States	San Jose	95101	Ap #644-0492 Sagittis St.
1000	Ankivr	Qbunrbp	+48-258-9936	ankivr.qbunrbp@example.com	May 17, 1970	Germany	Stuttgart	70173	215 Qbunrbp St
1001	Nfrirb	Oydaibvg	+94-121-2522	nfrirb.oydaibvg@example.com	Mar 4, 1994	Germany	Cologne	50667	121 Oydaibvg St
1002	Ijbfnd	Jdddhvyp	+70-120-3668	ijbfnd.jdddhvyp@example.com	Sep 27, 2001	Germany	Munich	80331	412 Jdddhvyp St
1003	Uycig	Sfbdymzj	+58-123-6968	uycig.sfbdymzj@example.com	Jan 20, 1981	Germany	Stuttgart	70173	735 Sfbdymzj St
1004	Uycig	Sfbdymzj	+58-123-6968	uycig.sfbdymzj@example.com	Jan 20, 1981	Germany	Stuttgart	70173	735 Sfbdymzj St

Output

#	Time	Action	Message	Duration / Fetch
25127	00:23:01	INSERT INTO user (id, name, surname, phone, email, birth_date, country, city, postal_code, address)	1 row(s) affected	0.000 sec
25128	00:23:01	INSERT INTO user (id, name, surname, phone, email, birth_date, country, city, postal_code, address)	1 row(s) affected	0.000 sec
25129	00:26:02	SELECT * FROM user	5000 row(s) returned	0.000 sec / 0.031 sec

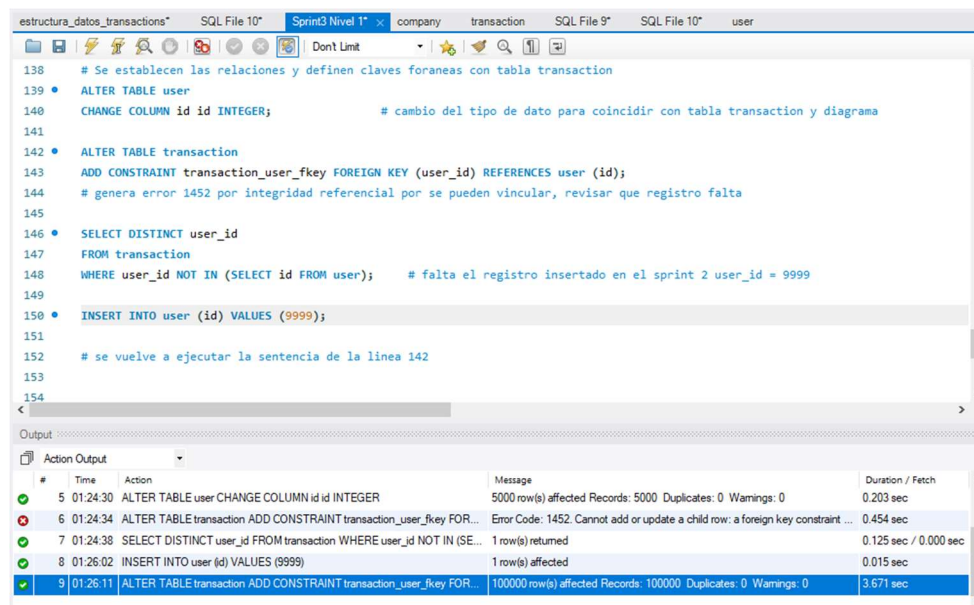
Para la nueva tabla **user** falta establecer la relación que la vincula con la base de datos **transactions**, ya se asoció el campo de *id* como Primary Key durante la creación de la tabla, ahora se debe identificar el campo en las otras tablas con el cual se relación. En la tabla **transaction**

existe en campo *user\_id* que aparentemente seria la vinculación, sin embargo, el formato de datos es diferente.

Se procede a modificar el formato de campo *id* a INTEGER, como lo establece el diagrama y está en la tabla **transaction**. En la imagen anexa se aprecia el cambio de formato con los comandos ALTER TABLE y CHANGE COLUMN.

Luego se crea la sentencia para realizar la restricción y vinculo entre las tablas **user** y **transaction** a través de la asignación de Foreign Key, usando los comandos ALTER TABLE, ADD CONSTRAINT, FOREIGN KEY y REFERENCES.

Al intentar ejecutar se genera un error code 1452, similar al indicado en el Ejercicio 3 del Nivel 1 por integridad referencial, ya que existen inconsistencias entre los registros de **transaction.user\_id** y **user.id**. Se soluciona verificando cual es el registro con un SELECT DISTINCT y WHERE, para después agregarlo con un INSERT INTO a la tabla. En este caso coincidía precisamente con el registro introducido en el ejercicio 3 del Nivel 1 ya citado. Se genera una sentencia con INSERT INTO para agregar el registro faltante *id = 9999* a la tabla **user**. Ahora se intenta nuevamente ejecutar la sentencia del párrafo anterior para asignar la Foreign Key, como se presenta en la imagen anexa en la última línea de la ventana "Action Output".



```
138 # Se establecen las relaciones y definen claves foraneas con tabla transaction
139 ALTER TABLE user
140 CHANGE COLUMN id id INTEGER; # cambio del tipo de dato para coincidir con tabla transaction y diagrama
141
142 ALTER TABLE transaction
143 ADD CONSTRAINT transaction_user_fkey FOREIGN KEY (user_id) REFERENCES user (id);
144 # genera error 1452 por integridad referencial por se pueden vincular, revisar que registro falta
145
146 SELECT DISTINCT user_id
147 FROM transaction
148 WHERE user_id NOT IN (SELECT id FROM user); # falta el registro insertado en el sprint 2 user_id = 9999
149
150 INSERT INTO user (id) VALUES (9999);
151
152 # se vuelve a ejecutar la sentencia de la linea 142
153
154
```

#	Time	Action	Message	Duration / Fetch
5	01:24:30	ALTER TABLE user CHANGE COLUMN id id INTEGER	5000 row(s) affected Records: 5000 Duplicates: 0 Warnings: 0	0.203 sec
6	01:24:34	ALTER TABLE transaction ADD CONSTRAINT transaction_user_fkey FOR...	Error Code: 1452. Cannot add or update a child row: a foreign key constraint ...	0.454 sec
7	01:24:38	SELECT DISTINCT user_id FROM transaction WHERE user_id NOT IN (SE...	1 row(s) returned	0.125 sec / 0.000 sec
8	01:26:02	INSERT INTO user (id) VALUES (9999)	1 row(s) affected	0.015 sec
9	01:26:11	ALTER TABLE transaction ADD CONSTRAINT transaction_user_fkey FOR...	100000 row(s) affected Records: 100000 Duplicates: 0 Warnings: 0	3.671 sec

Finalizada la creación de la nueva tabla **user** dentro de la base de datos, ahora se realizarán las modificaciones necesarias para reproducir el diagrama anexo en el enunciado.

Se deben hacer cambios en nombres, formatos tipo y longitud, eliminar e insertar otros campos. Para separar las tareas se dividirá por tablas para indicar los cambios que se realizaran:

Tabla **credit\_card**:

- editar extensión VARCHAR en *id*, *ibn* y *expiring\_date\_var*,
- modificar formato a INTEGER en *cvv*,
- eliminar *expiring\_date* formato DATE,
- renombrar *expiring\_date\_varchar* como *expiring\_date* y
- crear campo *fecha actual* en formato DATE.

Tabla **company**:

- eliminar el campo *website*.

Tabla **user**:

- renombrar la tabla de *user* a *data\_user*.

Tabla **transation**:

- editar extensión del formato VARCHAR en *credit\_card\_id*.

En las 2 imágenes siguientes se presentan las sentencias que permitieron generar el resto de los cambios en la base de datos y las tablas indicadas.

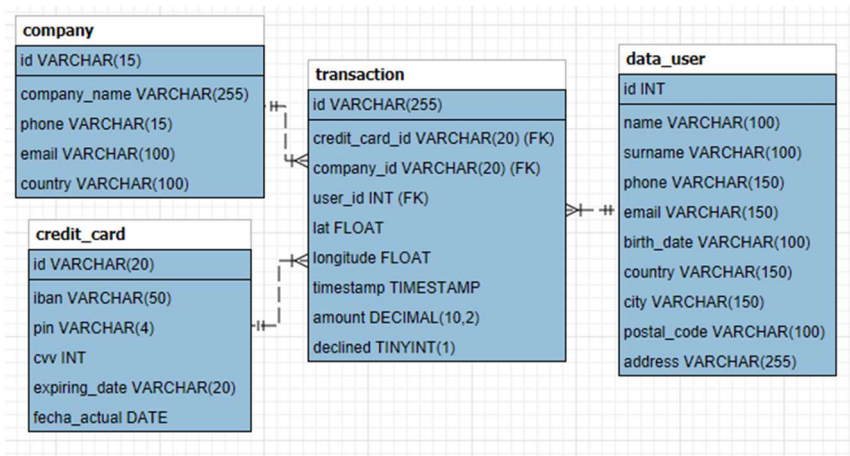
```
152
153 # Ahora se ajustan el tipo de dato, extension y nombres para la tabla credit_card este como en el diagrama
154 • ALTER TABLE credit_card
155   CHANGE COLUMN id id VARCHAR(20);
156 • ALTER TABLE credit_card
157   MODIFY COLUMN iban VARCHAR(50);
158 • ALTER TABLE credit_card
159   MODIFY COLUMN cvv INTEGER;
160 • ALTER TABLE credit_card
161   MODIFY COLUMN expiring_date_varchar VARCHAR(20);
162 • ALTER TABLE credit_card
163   DROP COLUMN expiring_date;
164 • ALTER TABLE credit_card
165   RENAME COLUMN expiring_date_varchar TO expiring_date;
166
```

#	Time	Action	Message	Duration / Fetch
3	10:40:28	SELECT * FROM InformeTecnico	100000 row(s) returned	1.125 sec / 0.109 sec
4	10:44:54	ALTER TABLE credit_card CHANGE COLUMN id id VARCHAR(20)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.062 sec
5	10:45:45	ALTER TABLE credit_card MODIFY COLUMN iban VARCHAR(50)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.031 sec
6	10:46:55	ALTER TABLE credit_card MODIFY COLUMN cvv INTEGER	5001 row(s) affected Records: 5001 Duplicates: 0 Warnings: 0	0.234 sec
7	10:48:28	ALTER TABLE credit_card MODIFY COLUMN expiring_date_varchar VARCH...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.031 sec
8	10:53:18	ALTER TABLE credit_card DROP COLUMN expiring_date	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.047 sec

```
157
158 # Ahora se ejecutan el resto de los cambios a las tablas para imitar la imagen del diagrama presentado
159
160 # creacion del campo fecha actual con formato DATE
161 • ALTER TABLE credit_card
162   ADD COLUMN fecha_actual DATE DEFAULT (CURRENT_DATE);
163
164 # se elimina el campo website
165 • ALTER TABLE company
166   DROP COLUMN website;
167
168 # se renombra la Tabla user
169 • RENAME TABLE user TO data_user;
170
171 # se modifica la longitud del formato VARCHAR
172 • ALTER TABLE transation
173   MODIFY COLUMN credit_card_id VARCHAR(20);
174
```

#	Time	Action	Message	Duration / Fetch
7	22:27:09	ALTER TABLE credit_card ADD COLUMN fecha_actual DATE DEFAULT (CUR...	5001 row(s) affected Records: 5001 Duplicates: 0 Warnings: 0	0.187 sec
8	22:27:20	ALTER TABLE company DROP COLUMN website	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.078 sec
9	22:27:25	RENAME TABLE user TO data_user	0 row(s) affected	0.032 sec
10	22:27:36	ALTER TABLE transation MODIFY COLUMN credit_card_id VARCHAR(20)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.032 sec

Ahora se presenta en la imagen anexa como queda el diagrama de entidad relación de la base de datos con los cambios aplicados.



## Ejercicio 2

La empresa también le pide crear una vista llamada "InformeTecnico" 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.
- Asegúrese de incluir información relevante de las tablas que conocerá y utilice 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.

```

    # Ejercicio 2
    CREATE VIEW InformeTecnico AS
    SELECT t.id AS Num_transaction, u.name AS Name_user, u.surname AS Lastname_user, cc.iban AS Num_iban_Credit_card, c.company_name AS name_company_selling
    FROM transaction AS t
    JOIN data_user AS u
    ON t.user_id = u.id
    JOIN credit_card AS cc
    ON t.credit_card_id = cc.id
    JOIN company AS c
    ON t.company_id = c.id
    ORDER BY t.id DESC;
    SELECT * FROM InformeTecnico ORDER BY Num_transaction DESC;
  
```

Num_transaction	Name_user	Lastname_user	Num_iban_Credit_card	name_company_selling
FFFD31D6-9495-47CE-854A-7D88E1CC274B	Bmrgli	Tprvmrc	XX794814451211289182490922	Turpis Company
FFFCF76D-ECF0-4985-A2D0-82A7B75998FC	Dfied	Vlqcdl	XX636251701647892036676034	Amet Nulla Donec Corporation
FFFC9E8D-27C7-4ADE-98F2-7533EF4DF126	Securp	Faofvqfy	XX162677143304223631437567	Nunc Interdum Incorporated
FFFB270D-F53A-4D5D-9666-E5307C53CC84	Ggzpa	Uirzjuh	XX395114267082019952567052	Viverra Donec Foundation
FFF9E3CE-234E-408C-A8EF-F9CAD577224A	Yshmq	Zpsjsleed	XX8845462156537570367941	Convallis In Incorporated

InformeTecnico 2 x

Output

Action Output

#	Time	Action	Message	Duration / Fetch
14	23:53:58	CREATE VIEW InformeTecnico AS SELECT t.id AS Num_transaction, u.name ...	0 row(s) affected	0.015 sec
15	23:54:00	SELECT * FROM InformeTecnico ORDER BY Num_transaction DESC	100000 row(s) returned	1.094 sec / 0.110 sec

En este ejercicio se arma la consulta solicitada en el enunciado para almacenarla en la vista “InformeTecnico”. En la sentencia se piden datos de todas las tablas de la base de datos, por lo que se aplica varios comandos JOIN para unir las distintas tablas.

Se utilizaron alias descriptivos de los campos consultados y se indico el ORDER BY tanto en la creación de la vista como en la consulta de la misma como se requería por el **transaction.id**.

### OBSERVACION ADICIONAL

En el Nivel 1 Ejercicio 1, al cargar los datos en la tabla **credit\_card**, se identifico que el campo *expiring\_date* correspondiente a la fecha de caducidad de la tarjeta venia en un formato diferente al DATE, venia en formato regional, el cual no reconoce **mysql** como formato de fecha.

Para solucionar el problema y pensando a futuro que sería conveniente tener el campo en formato fecha, se modificó la estructura para cargarlo como VARCHAR. Luego se realizaron los cambios necesarios con los comandos STR\_TO\_DATE, REPLACE y GET\_FORMAT para poder guardar los datos con formato DATE. En la imagen final del Sprint se muestra como se ejecutaron las sentencias para aplicar los cambios en el formato fecha.

Se considero mantener los datos de formato VARCHAR en una columna de nombre *expiring\_date\_varchar*. Lo cual resulto muy útil al final ya que para este Sprint no se requería el cambio de formato a fecha para ese campo.

The screenshot shows a SQL IDE interface with a query editor and a result grid. The query editor contains the following SQL commands:

```
26 # Se debe cambiar el formato de los datos de fecha, se cargaron como varchar para luego modificar
27 • ALTER TABLE credit_card
28   RENAME COLUMN expiring_date TO expiring_date_varchar; -- Se renombra para mantener nombre original
29
30 # Se crea la columna con formato DATE, transformando la columna inicial que viene en formato regional mm/dd/yy
31 • ALTER TABLE credit_card
32   ADD COLUMN expiring_date DATE AS (STR_TO_DATE(REPLACE(expiring_date_varchar, '/', '.'), GET_FORMAT(date, 'USA')));
33
34 • SELECT * FROM credit_card;
```

The result grid displays the following data:

id	iban	pan	pin	cvv	expiring_date_varchar	expiring_date
CcU-2938	TR301950312213576817638661	5424465566813633	3257	984	10/30/22	2022-10-30
CcU-2945	DO26854763748537475216568689	5142423821948828	9080	887	08/24/23	2023-08-24
CcU-2952	BG451VQL52710525608255	4556 453 55 5287	4598	438	06/29/21	2021-06-29
CcU-2959	CR7242477244335841535	372461377349375	3583	667	02/24/23	2023-02-24
CcU-2966	BG72LKTQ15627628377363	448566 886747 7265	4900	130	10/29/24	2024-10-29
CcU-2973	PT87806228135092429456346	544 58654 54343 384	8760	887	01/30/25	2025-01-30

The output section shows the execution of the SQL commands with the following details:

#	Time	Action	Message	Duration / Fetch
10049	23:12:44	ALTER TABLE credit_card RENAME COLUMN expiring_date TO expiring_date...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.031 sec
10050	23:12:50	ALTER TABLE credit_card ADD COLUMN expiring_date DATE AS (STR_TO_D...	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.031 sec
10051	23:12:56	SELECT * FROM credit_card	5000 row(s) returned	0.000 sec / 0.015 sec