

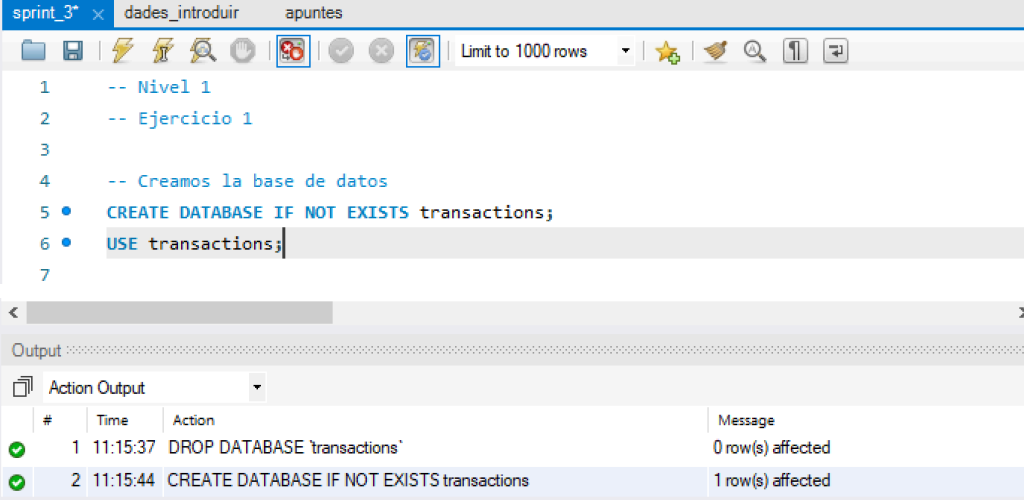
NIVELL 1

Exercici 1

La teva tasca és dissenyar i crear una taula anomenada "credit_card" que emmagatzemi detalls crucials sobre les targetes de crèdit. La nova taula ha de ser capaç d'identificar de manera única cada targeta i establir una relació adequada amb les altres dues taules ("transaction" i "company"). Després de crear la taula serà necessari que ingressis la informació del document denominat "dades_introduir_credit". Recorda mostrar el diagrama i realitzar una breu descripció d'aquest.

Empezamos creando la base de datos y las tablas con sus relaciones:

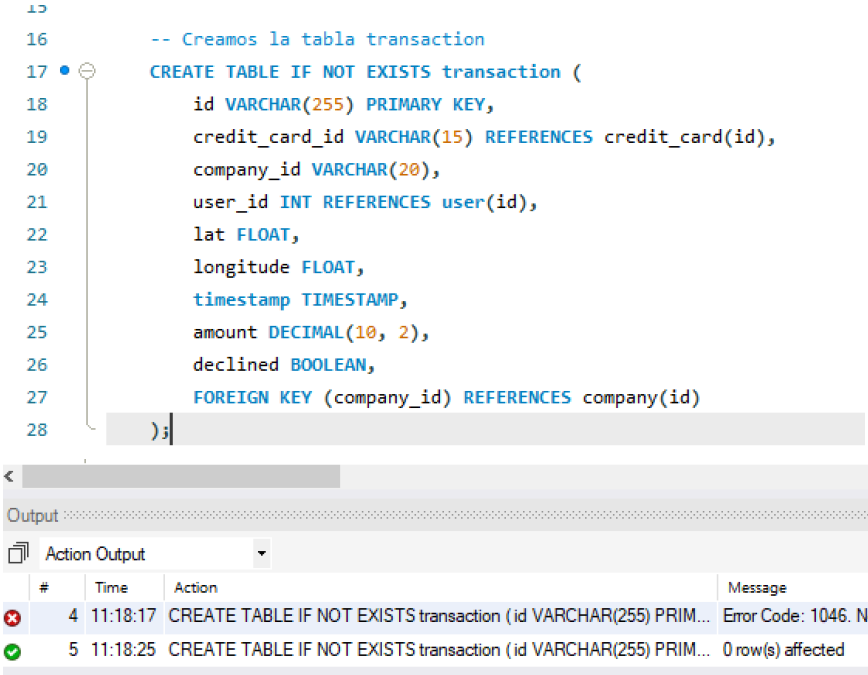
Creación de la base de dato transactions:



```
1  -- Nivel 1
2  -- Ejercicio 1
3
4  -- Creamos la base de datos
5  • CREATE DATABASE IF NOT EXISTS transactions;
6  • USE transactions;
7
```

#	Time	Action	Message
✓ 1	11:15:37	DROP DATABASE `transactions`	0 row(s) affected
✓ 2	11:15:44	CREATE DATABASE IF NOT EXISTS transactions	1 row(s) affected

Creación de la tabla transaction:



```
15
16  -- Creamos la tabla transaction
17  • CREATE TABLE IF NOT EXISTS transaction (
18      id VARCHAR(255) PRIMARY KEY,
19      credit_card_id VARCHAR(15) REFERENCES credit_card(id),
20      company_id VARCHAR(20),
21      user_id INT REFERENCES user(id),
22      lat FLOAT,
23      longitude FLOAT,
24      timestamp TIMESTAMP,
25      amount DECIMAL(10, 2),
26      declined BOOLEAN,
27      FOREIGN KEY (company_id) REFERENCES company(id)
28  );
```

#	Time	Action	Message
✗ 4	11:18:17	CREATE TABLE IF NOT EXISTS transaction (id VARCHAR(255) PRIM...	Error Code: 1046. No database selected
✓ 5	11:18:25	CREATE TABLE IF NOT EXISTS transaction (id VARCHAR(255) PRIM...	0 row(s) affected

Creación de la tabla company:

```
sprint_3* x  dades_introduir  apuntes
[Icons] Limit to 1000 rows [Icons]
16 timestamp TIMESTAMP,
17 amount DECIMAL(10, 2),
18 declined BOOLEAN);
19
20 -- Creamos la tabla company
21 • CREATE TABLE IF NOT EXISTS company (
22     id VARCHAR(15) PRIMARY KEY,
23     company_name VARCHAR(255),
24     phone VARCHAR(15),
25     email VARCHAR(100),
26     country VARCHAR(100),
27     website VARCHAR(255));
28
29 -- insertamos los datos de las tablas
30 -- mostramos el resultado
31 • SHOW TABLES FROM transactions;
32 • SHOW COLUMNS FROM company;
33
34 -- relacionamos las tablas
<
Output
Action Output
# Time Action Message
6 11:19:54 CREATE TABLE IF NOT EXISTS transaction (id VARCHAR(255) PRIM... 0 row(s) affected, 1
7 11:20:07 CREATE TABLE IF NOT EXISTS company (id VARCHAR(15) PRIMAR... 0 row(s) affected
```

Insertamos los datos:

```
sprint_3*  dades_introduir x  apuntes
[Icons] Limit to 1000 rows
1 • USE transactions;
2
3 -- Insertamos datos de company
4 • INSERT INTO company (id, company_name, phone, email, country, website) VALUES (
5 • INSERT INTO company (id, company_name, phone, email, country, website) VALUES (
```

Output

Action Output

#	Time	Action	Message
694	11:28:53	INSERT INTO transaction (id, credit_card_id, company_id, user_id, lat, l...	1 row(s) affected
695	11:28:53	INSERT INTO transaction (id, credit_card_id, company_id, user_id, lat, l...	1 row(s) affected

Creamos la tabla credit card:

Creamos un índice para optimizar recursos. Hacemos un DROP TABLE IF EXISTS para eliminar si hubiera una tabla anterior hecha con el mismo nombre para poder crear la nueva. Creamos las diferentes columnas. Utilizando una numeración específica de VARCHAR para que haya espacio al cambio, pero reducido para ocupar el mínimo de espacio posible.

```

CREATE INDEX idx_credit_card_id ON transaction(credit_card_id);
DROP TABLE IF EXISTS credit_card;
CREATE TABLE IF NOT EXISTS credit_card (
    id VARCHAR(20) PRIMARY KEY,
    iban VARCHAR(50),
    pan VARCHAR(50),
    pin VARCHAR(50),
    cvv INT,
    expiring_date VARCHAR(20);

```

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

```

36
37 • DROP TABLE IF EXISTS credit_card;
38 • CREATE TABLE IF NOT EXISTS credit_card (
39     id VARCHAR(20) PRIMARY KEY,
40     iban VARCHAR(50),
41     pan VARCHAR(50),
42     pin VARCHAR(50),
43     cvv INT,
44     expiring_date VARCHAR(20));
45

```

Below the query editor is an "Output" panel with a dropdown menu set to "Action Output". It displays a table with the following data:

#	Time	Action	Message
1252	12:10:24	DROP TABLE IF EXISTS credit_card	0 row(s) affected
1253	12:10:29	CREATE TABLE IF NOT EXISTS credit_card (id VARCHAR(20) PRI...	0 row(s) affected

Insertamos los datos:

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

```

1
2 -- Insertamos datos de credit_card
3 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES (
4 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES (
5 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES (
6 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES (
7 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES (
8 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES (
9 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES (
10 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES (
11 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES (
12 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES (
13 • INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALUES (

```

Below the query editor is an "Output" panel with a dropdown menu set to "Action Output". It displays a table with the following data:

#	Time	Action	Message
1250	12:07:53	INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALU...	1 row(s) affected
1251	12:07:53	INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALU...	1 row(s) affected

Creamos la relación con la tabla transaction:

Relacionamos las tablas una vez insertados los valores para que se puedan vincular los valores y no tengamos errores de relación donde puede suceder que si no coinciden los valores se genere una relación inversa 1 a N transaction-credit_card.

The screenshot shows a SQL IDE with a query editor and an output window. The query editor contains the following SQL code:

```
48 -- relacionamos las tablas
49 ALTER TABLE transaction ADD FOREIGN KEY (credit_card_id) REFERENCES credit_card(id);
50
```

The output window shows the results of the execution:

#	Time	Action	Message
2501	12:27:04	INSERT INTO credit_card (id, iban, pan, pin, cvv, expiring_date) VALU...	1 row(s) affected
2502	12:29:21	ALTER TABLE transaction ADD FOREIGN KEY (credit_card_id) REFE...	587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0

Mostramos las tablas y sus columnas:

The screenshots show the execution of several SQL statements to display the database structure:

```
52 SHOW TABLES FROM transactions;
53 SHOW COLUMNS FROM company;
54 SHOW COLUMNS FROM credit_card;
55 SHOW COLUMNS FROM transaction;
```

The results are shown in the following tables:

Table
company
credit_card
transaction

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		NULL	
lat	float	YES		NULL	
longitude	float	YES		NULL	
timestamp	timestamp	YES		NULL	
amount	decimal(10,2)	YES		NULL	
declined	tinyint(1)	YES		NULL	

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

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	

Descripción de la base de datos

La base de datos contiene las tablas transaction, company y credit_card. Se trata de una base de datos relacional de modelo dimensional con esquema en estrella donde la tabla transaction es la tabla de hechos y las otras dos son las tablas dimensionales.

Las tablas se relacionan como tablas de dimensiones (company y credit_card) 1 a N con la tabla de hechos (transaction) a través de la PK de las tablas de de dimensiones, la columna id en ambos casos y las FK de las columnas de la tabla de hechos. Las relaciones 1 a N o many son relaciones en las que puede haber múltiples transacciones por empresas y tarjetas de crédito, pero cada transacción es única.

- **Tabla de hechos: Transaction** con id (Primary Key), credit_card_id, company_id (Foreign Key), user_id, lat, longitude, timestamp, amount, declined. Contiene datos medibles y dinámicos, es donde ocurren las operaciones principales.

Columna	Tipo de Dato	Características
id	VARCHAR (255) (PK)	Identificador único de la transacción, clave primaria.
credit_card_id	VARCHAR(150) (FK)	Clave foránea que referencia credit_card(id), establece la relación 1:N con la tabla Credit_card por su id.
company_id	VARCHAR (20) (FK)	Clave foránea que referencia Company(id), establece la relación 1:N con la tabla Company por su id.
user_id	INT (FK)	Clave foránea que referencia User(id), establece la relación 1:N con la tabla User por su id.
lat	FLOAT	Latitud de la ubicación de la transacción .
longitude	FLOAT	Longitud de la ubicación de la transacción.
timestamp	TIMESTAMP	Fecha y hora exacta de la transacción.
amount	DECIMAL(10,2)	Monto de la transacción con 2 decimales
declined	TINYINT o BOOLEAN (1)	Indica si la transacción fue rechazada (TRUE = rechazada, FALSE = aprobada).

- **Tabla de dimensiones: Company** con id (Primary Key), company_name, phone, email, country, website. Contiene información estática y descriptiva de las empresas, para contextualizar los datos de la tabla de hechos.

Columna	Tipo de Dato	Características
id	VARCHAR (15) (PK)	Identificador único, clave primaria.
company_name	VARCHAR(255)	Nombre de la empresa, texto de longitud variable (hasta 255 caracteres).
phone	VARCHAR(15)	Teléfono de la empresa, texto de longitud variable (hasta 255 caracteres).
email	VARCHAR(100)	Correo electrónico de la empresa. (Información sensible).
country	VARCHAR(100)	País de la empresa.
website	VARCHAR(255)	URL del sitio web de la empresa

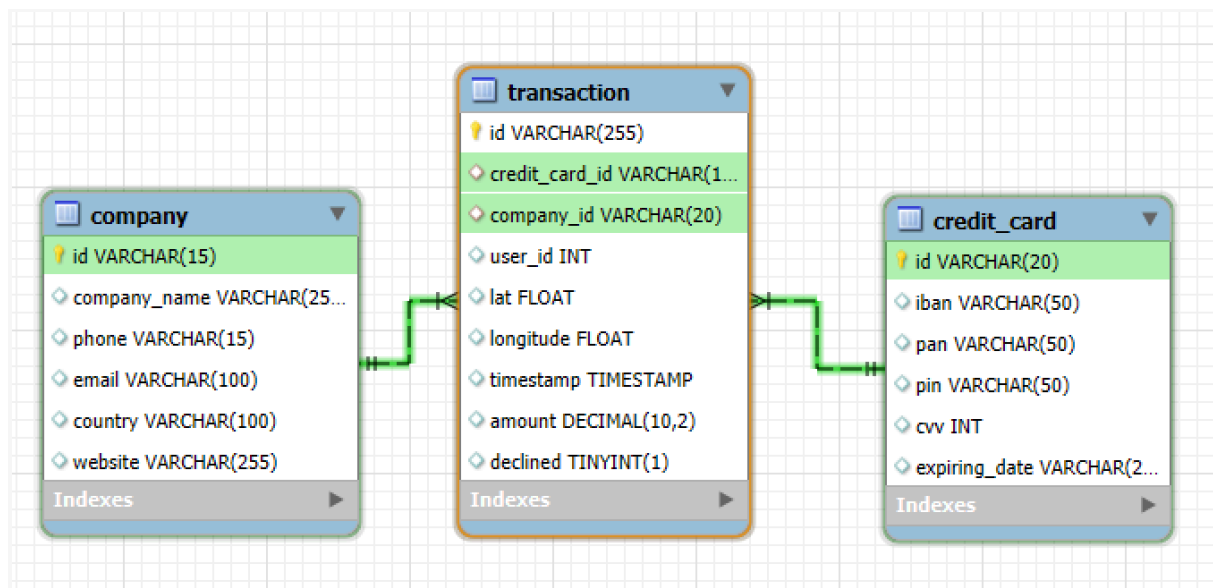
- **Tabla de dimensiones: Credit_card** con id (Primary Key), iban, pan, pin, cvv y expiring_date. Contiene información estática y descriptiva de los específicos de las tarjetas de crédito utilizadas en los pagos, para contextualizar los datos de la tabla de hechos. Toda la información es sensible menos la fecha de caducidad.

Columna	Tipo de Dato	Características
id	VARCHAR(20)(PK)	Identificador único, clave primaria.
iban	VARCHAR(50)	International Bank Account Number, texto de longitud variable (hasta 50 caracteres).
pan	VARCHAR(50)	Personal Account Number, texto de longitud variable (hasta 50 caracteres).
pin	VARCHAR(50)	pin de la tarjeta, texto de longitud variable (hasta 50 caracteres).
cvv	INT	cvv de la tarjeta, dato numérico entero.
expiring_date	VARCHAR(20)	Fecha de caducidad de la tarjeta, texto de longitud variable (hasta 20 caracteres).

Mostramos los índices de las tablas:

56 • **SHOW INDEXES FROM transaction;**

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality
transaction	0	PRIMARY	1	id	A	587
transaction	1	company_id	1	company_id	A	100
transaction	1	idx_credit_card_id	1	credit_card_id	A	275



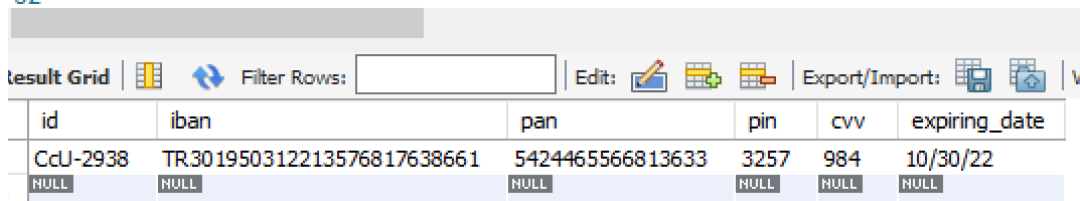
Exercici 2

El departament de Recursos Humans ha identificat un error en el número de compte de l'usuari amb ID CcU-2938. La informació que ha de mostrar-se per a aquest registre és: R323456312213576817699999. Recorda mostrar que el canvi es va realitzar.

Buscamos el usuario para comprobar que está y que efectivamente el número de cuenta es incorrecto.

```
SELECT * FROM credit_card WHERE id = "CcU-2938";
```

```
58 -- Ejercicio 2
59 -- Modificar el iban del usuario con id "CcU-2938"
60 • SELECT * FROM credit_card
61 WHERE id = "CcU-2938";
62
```



id	iban	pan	pin	cvv	expiring_date
CcU-2938	TR301950312213576817638661	5424465566813633	3257	984	10/30/22
NULL	NULL	NULL	NULL	NULL	NULL

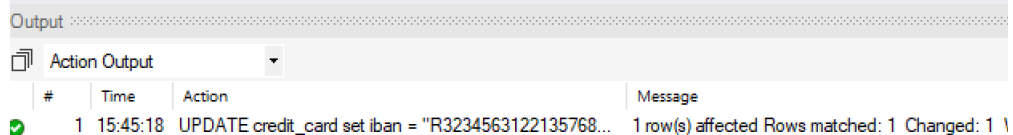
Con UPDATE indicamos que queremos modificar o actualizar la tabla con datos diferentes a los que hay. Con SET indicamos lo que queremos modificar y con el WHERE filtramos o especificamos el registro al queremos hacer la modificación.

```
UPDATE credit_card
SET iban = "R323456312213576817699999"
WHERE id = "CcU-2938";
```

```
17 -- Ejercicio 2
18 -- Modificar el iban del usuario con id "CcU-2938"
19 • SELECT * FROM credit_card
20 WHERE id = "CcU-2938";
21
22 • UPDATE credit_card set
23 iban = "R323456312213576817699999"
24 WHERE id = "CcU-2938";
25
26 -- Ejercicio 3
27 -- ingresar un nuevo usuario
28 • SELECT * FROM transaction;
```

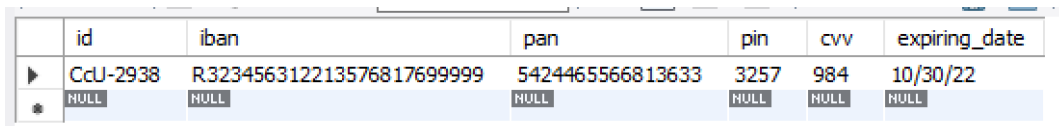
Automatically
disabled
manual
current
toggle

Context Help



#	Time	Action	Message
1	15:45:18	UPDATE credit_card set iban = "R323456312213576817699999"	1 row(s) affected Rows matched: 1 Changed: 1

Comprobamos que el cambio se ha realizado correctamente:



id	iban	pan	pin	cvv	expiring_date
CcU-2938	R323456312213576817699999	5424465566813633	3257	984	10/30/22
NULL	NULL	NULL	NULL	NULL	NULL

Exercici 3

En la taula "transaction" ingresa un nou usuari amb la següent informació:

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

Comprobamos que efectivamente el usuario no existe:

```
69 • SELECT * FROM transaction
70 WHERE id = "108B1D1D-5B23-A76C-55EF-C568E49A99DD";
71
```

Result Grid

	id	credit_card_id	company_id	user_id	lat	longitude	timestamp	amount	declined
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
72 • SELECT * FROM credit_card
73 WHERE id = "CcU-9999";
74
```

Result Grid

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

```
75 • SELECT * FROM company
76 WHERE id = "b-9999";
77
```

Result Grid

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

Podemos observar que los datos no están registrados en ninguna de las tablas.

Para poder agregar el registro correctamente en la tabla transaction, debemos añadirlo también en las tablas company y credit_card en las columnas que se relacionan como PK-FK.

Insertamos el id pertinente en la tabla company en la columna id:

INSERT INTO company (id) VALUES ('b-9999');

```
78 • INSERT INTO company (id)
79 VALUES ('b-9999');
```

Output

Action Output

#	Time	Action	Message
✓ 2517	13:43:00	INSERT INTO company (id) VALUES ('b-9999')	1 row(s) affected

Insertamos el id pertinente en la tabla credit_card en la columna id:

```
INSERT INTO credit_card (id) VALUES ('CcU-9999');
```

81	•	INSERT INTO credit_card (id)
82		VALUES ('CcU-9999');

#	Time	Action	Message
✓ 2519	13:46:56	INSERT INTO credit_card (id) VALUES ('CcU-9999')	1 row(s) affected

Finalmente insertamos los datos de la nueva transacción. No aparece el dato de la columna Timestamp, que podríamos añadir con la función NOW() que pondría la fecha actual, pero hemos preferido dejarlo en blanco, lo que dará un valor NULL.

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

84	•	INSERT INTO transaction (id, credit_card_id, company_id, user_id, lat, longitude, amount, declined)
85		VALUES ('108B1D1D-5B23-A76C-55EF-C568E49A99DD', 'CcU-9999', 'b-9999', 9999, 829.999, -117.999, 111.11, 0);
86		

#	Time	Action	Message
✓ 2519	13:46:56	INSERT INTO credit_card (id) VALUES ('CcU-9999')	1 row(s) affected
✓ 2520	13:48:56	INSERT INTO transaction (id, credit_card_id, company_id, user_id, lat, longitude, amo...	1 row(s) affected

Comprobamos que el usuario se haya creado correctamente:

En la tabla transaction: Resultado Ok

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

#	Time	Action	Message
✓ 2520	13:48:56	INSERT INTO transaction (id, credit_card_id, company_id, user_id, lat, longitude, ...	1 row(s) affected
✓ 2521	13:51:53	SELECT * FROM transaction WHERE id = "108B1D1D-5B23-A76C-55EF-C568E...	1 row(s) returned

En la tabla company y credit_card: Resultado Ok

75 •	SELECT * FROM company
76	WHERE id = "b-9999";
77	

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

72 •	SELECT * FROM credit_card
73	WHERE id = "CcU-9999";
74	

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

Exercici 4

Des de recursos humans et sol·liciten eliminar la columna "pan" de la taula credit_card. Recorda mostrar el canvi realitzat.

Comprobamos que la columna exista:

90 •	SHOW COLUMNS FROM credit_card;
91	

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

Con ALTER TABLE indicamos que queremos modificar la tabla, con DROP COLUMN eliminamos la columna.

ALTER TABLE credit_card DROP COLUMN pan;

92 •	ALTER TABLE credit_card DROP COLUMN pan;
93	

#	Time	Action	Message
✓ 2525	13:56:16	SHOW COLUMNS FROM credit_card	6 row(s) returned
✓ 2526	13:59:55	ALTER TABLE credit_card DROP COLUMN pan	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Comprobamos que no aparece de nuevo la columna:

90 •	SHOW COLUMNS FROM credit_card;
------	--------------------------------

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

Nivell 2

Exercici 1

Elimina de la taula transaction el registre amb ID 02C6201E-D90A-1859-B4EE-88D2986D3B02 de la base de dades.

Comprobamos si el registro existe:

```
97 • SELECT * FROM transaction WHERE id = "02C6201E-D90A-1859-B4EE-88D2986D3B02";
```

id	credit_card_id	company_id	user_id	lat	longitude	timestamp	amount
02C6201E-D90A-1859-B4EE-88D2986D3B02	CcU-2938	b-2362	92	81.9185	-12.5276	2021-08-28 23:42:24	466.92
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Utilizamos DELETE FROM para decir que queremos eliminar datos de la tabla transaction y con WHERE id filtramos el registro a eliminar.

DELETE FROM transaction WHERE id = "02C6201E-D90A-1859-B4EE-88D2986D3B02";

```
97 • SELECT * FROM transaction WHERE id = "02C6201E-D90A-1859-B4EE-88D2986D3B02";
```

#	Time	Action	Message
2528	16:08:26	SELECT * FROM transaction WHERE id = "02C6201E-D90A-1859-B4EE-88D2986D3B02";	1 row(s) returned
2529	16:08:59	DELETE FROM transaction WHERE id = "02C6201E-D90A-1859-B4EE-88D2986D3B02";	1 row(s) affected

Comprobamos si el registro ha sido eliminado correctamente:

```
97 • SELECT * FROM transaction WHERE id = "02C6201E-D90A-1859-B4EE-88D2986D3B02";
```

id	credit_card_id	company_id	user_id	lat	longitude	timestamp	amount	declined
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Exercici 2

La secció de màrqueting desitja tenir accés a informació específica per a realitzar anàlisi i estratègies efectives. S'ha sol·licitat crear una vista que proporcioni detalls clau sobre les companyies i les seves transaccions. Serà necessària que creïs una vista anomenada VistaMarketing que contingui la següent informació: Nom de la companyia. Telèfon de contacte. País de residència. Mitjana de compra realitzat per cada companyia. Presenta la vista creada, ordenant les dades de major a menor mitjana de compra.

Utilizamos CREATE VIEW AS para crear la vista. Seleccionamos el nombre de la compañía, el teléfono y país, además de calcular la media por compras de las tablas company y transaction unidas por un INNER JOIN por el id de la compañía. Lo agrupamos por los datos que queremos ver y lo ordenamos descendientemente por la media para ver las más altas primero.

CREATE VIEW VistaMarketing AS

SELECT company_name, company.phone, company.country, AVG(amount) as media_compra

```

FROM company
INNER JOIN transaction
ON company.id = transaction.company_id
GROUP BY company_name, company.phone, company.country
ORDER BY media_compra DESC;

```

```

103 • CREATE VIEW VistaMarketing AS
104 SELECT company_name, company.phone, company.country, AVG(amount) as media_compra
105 FROM company
106 INNER JOIN transaction
107 ON company.id = transaction.company_id
108 GROUP BY company_name, company.phone, company.country
109 ORDER BY media_compra DESC;
110

```

#	Time	Action	Message
✓ 2530	16:10:19	SELECT * FROM transaction WHERE id = "02C6201E-D90A-1859-B4EE-...	0 row(s) returned
✓ 2531	16:11:33	CREATE VIEW VistaMarketing AS SELECT company_name, company.ph...	0 row(s) affected

Hacemos un SELECT para mostrarlo en pantalla. En la imagen de la izquierda observamos en schemas el VIEW creado.

SCHEMAS

Filter objects

- sakila
- sys
- transactions
 - Tables
 - Views
 - vistamarketing
 - company_name
 - phone
 - country
 - media_compra
 - Stored Procedures
 - Functions
- world

Administration Schemas

```

111 • SELECT * FROM VistaMarketing;
112

```

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

company_name	phone	country	media_compra
Eget Ipsum Ltd	03 67 44 56 72	United States	473.075000
Non Magna LLC	06 71 73 13 17	United Kingdom	468.345000
Sed Id Limited	07 28 18 18 13	United States	461.210000
Justo Eu Arcu Ltd	08 42 56 71 52	Italy	443.635000
Eget Tincidunt Dui Institute	Eget Tincidunt Dui Institute	ands	442.520000

VistaMarketing 20 x

Output

Action Output

#	Time	Action	Message
✓ 2531	16:11:33	CREATE VIEW VistaMarketing AS SELECT company_name, company.ph...	0 row(s) affected
✓ 2532	16:13:32	SELECT * FROM VistaMarketing LIMIT 0, 1000	101 row(s) returned

Exercici 3

Filtra la vista VistaMarketing per a mostrar només les companyies que tenen el seu país de residència en "Germany"

Filtramos con el WHERE las compañías cuya residencia sea Alemania del VIEW que hemos creado.

```

SELECT * FROM VistaMarketing
WHERE country = "Germany";

```

```

115 • SELECT * FROM VistaMarketing
116 WHERE country = "Germany";
117

```

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

	company_name	phone	country	media_compra
▶	Aliquam PC	01 45 73 52 16	Germany	385.265000
	Ac Industries	09 34 65 40 60	Germany	289.645000
	Rutrum Non Inc.	02 66 31 61 09	Germany	266.900000
	Nunc Interdum Incorporated	05 18 15 48 13	Germany	244.025238
	Augue Foundation	06 88 43 15 63	Germany	240.800000

VistaMarketing 21 x

Output

Action Output

#	Time	Action	Message
✓ 2532	16:13:32	SELECT * FROM VistaMarketing LIMIT 0, 1000	101 row(s) returned
✓ 2533	16:17:39	SELECT * FROM VistaMarketing WHERE country = "Germany" LIMIT 0, 1...	8 row(s) returned

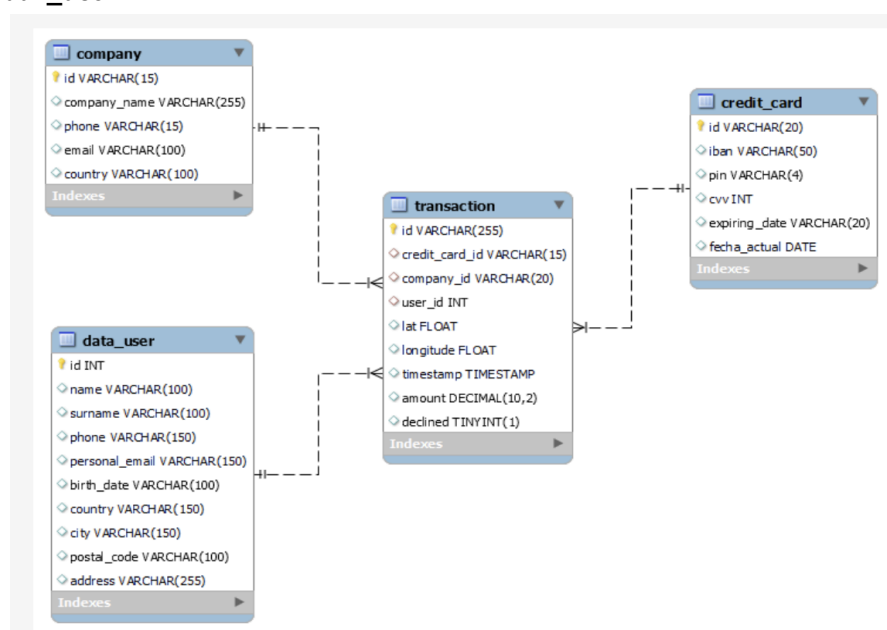
Nivell 3

Exercici 1

La setmana vinent tindràs una nova reunió amb els gerents de màrqueting. Un company del teu equip va realitzar modificacions en la base de dades, però no recorda com les va realitzar. Et demana que l'ajudis a deixar els comandos executats per a obtenir el següent diagrama:

Recordatori

En aquesta activitat, és necessari que descriguis el "pas a pas" de les tasques realitzades. És important realitzar descripcions senzilles, simples i fàcils de comprendre. Per a realitzar aquesta activitat hauràs de treballar amb els arxius denominats "estructura_dades_user" i "dades_introduir_user"



Primero comprobamos que la tabla no exista:

The screenshot shows a database interface. At the top, a query is entered: `SHOW TABLES FROM transactions;`. Below the query, a result grid displays the tables in the 'transactions' database: 'company', 'credit_card', 'transaction', and 'vistamarketing'. Below the result grid, the 'Output' tab is selected, showing an 'Action Output' log. The log contains two entries: a failed action (red cross) at 16:25:41 with the message 'Error Code: 1049. Ur', and a successful action (green checkmark) at 16:26:08 with the message '4 row(s) returned'.

Luego creamos la tabla con el siguiente código:

Creamos un índice para optimizar recursos, creamos la estructura comentada.

```
CREATE INDEX idx_user_id ON transaction(user_id);
CREATE TABLE IF NOT EXISTS user (
    id INT PRIMARY KEY,
    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),
    FOREIGN KEY(id) REFERENCES transaction(user_id);
```

The screenshot shows a database interface with a SQL editor and an output window. The SQL editor contains the following code:

```
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 );
```

 Below the editor, the 'Output' tab is selected, showing an 'Action Output' log. The log contains two entries: a successful action (green checkmark) at 16:43:29 with the message '0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0', and another successful action (green checkmark) at 16:43:38 with the message '0 row(s) affected'.

A continuación cargamos los datos:

```
4
3 -- Insertamos datos de user
4 • INSERT INTO user (id, name, surname, phone, email, birth_date, country, city, postal_code, ad
5 • INSERT INTO user (id, name, surname, phone, email, birth_date, country, city, postal_code, ad
6 • INSERT INTO user (id, name, surname, phone, email, birth_date, country, city, postal_code, ad
7 • INSERT INTO user (id, name, surname, phone, email, birth_date, country, city, postal_code, ad
8 • INSERT INTO user (id, name, surname, phone, email, birth_date, country, city, postal_code, ad
9 • INSERT INTO user (id, name, surname, phone, email, birth_date, country, city, postal_code, ad
10 • INSERT INTO user (id, name, surname, phone, email, birth_date, country, city, postal_code, ad
11 • INSERT INTO user (id, name, surname, phone, email, birth_date, country, city, postal_code, ad
12 • INSERT INTO user (id, name, surname, phone, email, birth_date, country, city, postal_code, ad
13 • INSERT INTO user (id, name, surname, phone, email, birth_date, country, city, postal_code, ad
14 • INSERT INTO user (id, name, surname, phone, email, birth_date, country, city, postal_code, ad
15 • INSERT INTO user (id, name, surname, phone, email, birth_date, country, city, postal_code, ad
16 • INSERT INTO user (id, name, surname, phone, email, birth_date, country, city, postal_code, ad
```

Output

Action Output

#	Time	Action	Message
✓ 2814	16:44:49	INSERT INTO user (id, name, surname, phone, email, birth_date, country, ...	1 row(s) affected
✓ 2815	16:44:49	SET foreign_key_checks = 1	0 row(s) affected

Se observa en el diagrama dado que el nombre de la tabla es distinto, por lo que le cambiamos el nombre a la tabla de user a data_user utilizando ALTER TABLE y RENAME TO.

ALTER TABLE user RENAME TO data_user;

```
138 -- Insertar datos
139 • ALTER TABLE user RENAME TO data_user;
140
```

Output

Action Output

#	Time	Action	Message
✓ 2815	16:44:49	SET foreign_key_checks = 1	0 row(s) affected
✓ 2816	16:48:40	ALTER TABLE user RENAME TO data_user	0 row(s) affected

Comprobamos que se haya hecho el cambio:

```
143 • SHOW TABLES FROM transactions;
144
```

Result Grid

Tables_in_transactions
company
credit_card
data_user
transaction
vistamarketing

Comprobamos las columnas de las diferentes tablas para comparar con el diagrama y ver qué otros cambios se requieren:

SHOW COLUMNS FROM transaction;

154 • `SHOW COLUMNS FROM transaction;`

155

Result Grid | Filter Rows: | Export: | Wrap 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	

transaction

- 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)

Indexes

Todo correcto.

`SHOW COLUMNS FROM company;`

154 • `SHOW COLUMNS FROM company;`

155

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

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	

company

- id VARCHAR(15)
- company_name VARCHAR(255)
- phone VARCHAR(15)
- email VARCHAR(100)
- country VARCHAR(100)

Indexes

Vemos que la columna website ya no está, por lo que procedemos a eliminarla:
`ALTER TABLE company DROP COLUMN website;`

158 • `ALTER TABLE company DROP COLUMN website;`

Output

Action Output

#	Time	Action	Message
2828	17:20:06	SHOW COLUMNS FROM company	6 row(s) returned
2829	17:23:55	ALTER TABLE company DROP COLUMN website	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Comprobamos que se haya llevado a cabo correctamente:

154 • `SHOW COLUMNS FROM company;`

155

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

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	

Comprobamos la columna data_user:
`SHOW COLUMNS FROM data_user;`

159 • `SHOW COLUMNS FROM data_user;`

Field	Type	Null	Key	Default	E
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	

data_user

- id INT
- name VARCHAR(100)
- surname VARCHAR(100)
- phone VARCHAR(150)
- personal_email VARCHAR(150)
- birth_date VARCHAR(100)
- country VARCHAR(150)
- city VARCHAR(150)
- postal_code VARCHAR(100)
- address VARCHAR(255)

Indexes

Vemos que se ha modificado el nombre de la columna email a personal_mail.

Lo modificamos:

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

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

Output

Action Output

#	Time	Action	Message
2832	17:27:42	SHOW COLUMNS FROM data_user	10 row(s) returned
2833	17:31:09	ALTER TABLE data_user CHANGE email personal_email VARCHAR(150)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Comprobamos que el cambio se haya llevado a cabo:

159 • `SHOW COLUMNS FROM data_user;`

Field	Type	Null	Key	Default
id	int	NO	PRI	NULL
name	varchar(100)	YES		NULL
surname	varchar(100)	YES		NULL
phone	varchar(150)	YES		NULL
personal_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

Comprobamos la columna credit_card:

`SHOW COLUMNS FROM credit_card;`

164 • `SHOW COLUMNS FROM credit_card;`

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

credit_card

- id VARCHAR(20)
- iban VARCHAR(50)
- pin VARCHAR(4)
- cvv INT
- expiring_date VARCHAR(20)
- fecha_actual DATE

Indexes

Observamos que se ha creado una columna nueva denominada fecha_actual.
Procedemos a crearla:

```
ALTER TABLE credit_card ADD COLUMN fecha_actual DATE;
```

(Si quisiéramos rellenarla con la fecha actual utilizamos el siguiente código:
UPDATE credit_card SET fecha_actual = CURDATE();)

```
166 -- Crear la columna fecha_actual
167 ALTER TABLE credit_card ADD COLUMN fecha_actual DATE;
```

Output

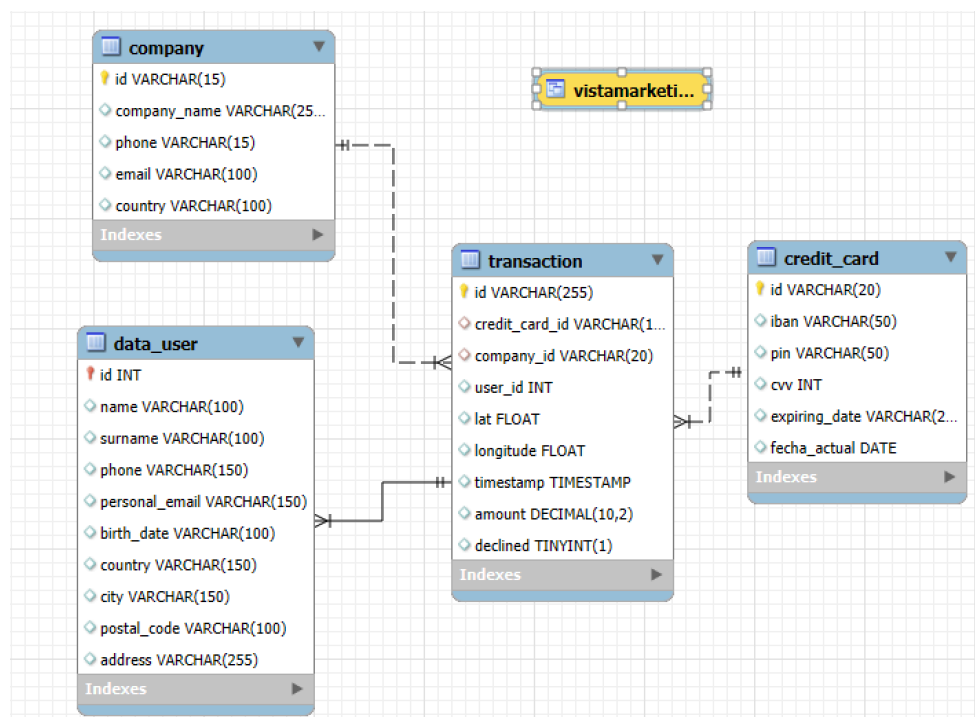
#	Time	Action	Message
2835	17:33:38	SHOW COLUMNS FROM credit_card	5 row(s) returned
2836	18:39:23	ALTER TABLE credit_card ADD COLUMN fecha_actual DATE	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Comprobamos que se haya creado correctamente:

```
158 SHOW COLUMNS FROM credit_card;
```

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

Al ir a comprobar la situación del diagrama, se ha observado que la relación entre data_user y transaction está invertida:



Para que la relación 1 a N sea correcta, deberemos eliminar la relación actual, añadir el registro hecho anteriormente en la tabla transaction con user_id = 9999, ya que si los datos no coinciden y hay más datos no existentes en la tabla de transaction que en la de data_user, se generará la relación de forma inversa siendo ésta la que tenemos en estos momentos: 1 a N de transaction a data_user.

Primero debemos eliminar la relación:

ALTER TABLE data_user DROP FOREIGN KEY data_user_ibfk_1;

168 • ALTER TABLE data_user DROP FOREIGN KEY data_user_ibfk_1;

Output

Action Output

#	Time	Action	Message
4106	19:21:17	ALTER TABLE data_user DROP FOREIGN KEY transaction_ibfk_1	Error Code: 1091. Can't DROP 'transaction_ibfk_1': check
4107	19:22:03	ALTER TABLE data_user DROP FOREIGN KEY data_user_ibfk_1	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Comprobamos que se ha eliminado correctamente: data_user está sin relaciones

data_user

id	name	surname	phone	personal_email	birth_date	country	city	postal_code	address
INT	VARCHAR(100)	VARCHAR(100)	VARCHAR(150)	VARCHAR(150)	VARCHAR(100)	VARCHAR(150)	VARCHAR(150)	VARCHAR(100)	VARCHAR(255)

Indexes

Procedemos a crear el registro de id = 99999:

Comprobamos primero si el usuario existe en la tabla data_user:

146 • SELECT * FROM data_user WHERE id = 9999;

147

Result Grid

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

Registramos al usuario:

INSERT INTO data_user (id) VALUES (9999);

146 • SELECT * FROM data_user WHERE id = 9999;

147

Output

Action Output

#	Time	Action	Message
2821	17:05:32	INSERT INTO data_user (id) VALUES ('9999')	Error Code: 1054. Unknown column '9999' in 'field list'
2822	17:05:58	INSERT INTO data_user (id) VALUES (9999)	1 row(s) affected

Comprobamos que se ha creado el registro:

```
146 • SELECT * FROM data_user WHERE id = 9999;
```

```
147
```

	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

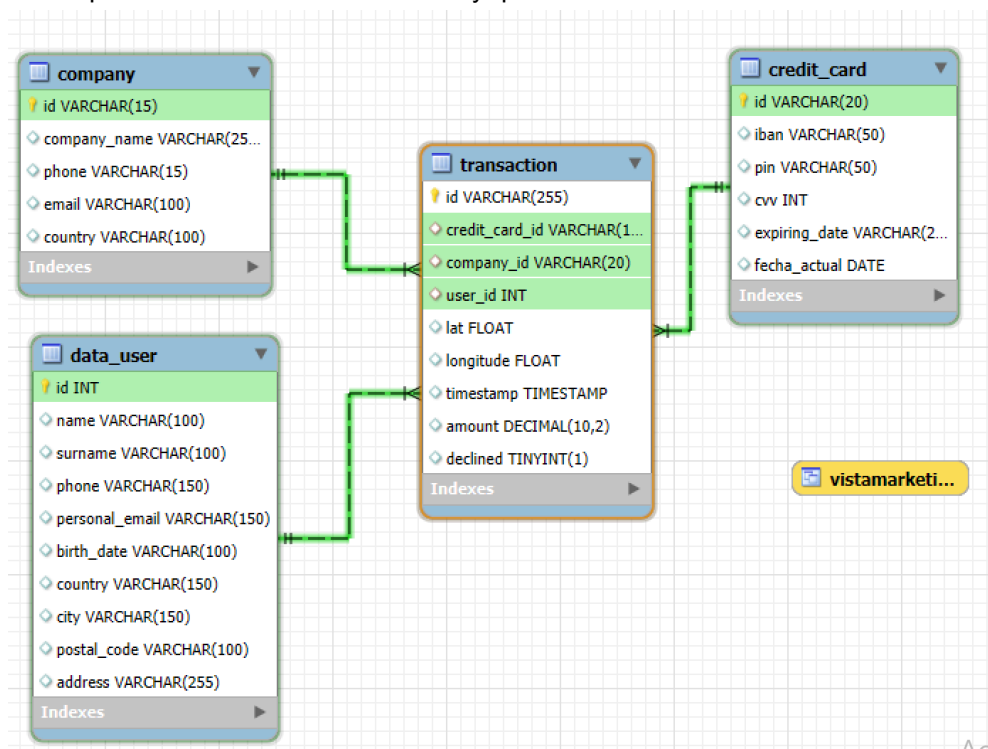
Finalmente vinculamos las tablas:

ALTER TABLE transaction ADD FOREIGN KEY (user_id) REFERENCES data_user(id);

```
150 • ALTER TABLE transaction ADD FOREIGN KEY (user_id) REFERENCES data_user(id);
```

#	Time	Action	Message
✖ 2824	17:10:55	ALTER TABLE transactions ADD FOREIGN KEY (user_id) REFERENCES d...	Error Code: 1146. Table 'transactions.transactions' doesn't exist
✔ 2825	17:11:09	ALTER TABLE transaction ADD FOREIGN KEY (user_id) REFERENCES d...	587 row(s) affected Records: 587 Duplicates: 0 Warnings: 0

Comprobamos que se ha creado correctamente y que toda la base de datos es correcta:



Descripción de la tabla:

A las tres tablas que teníamos antes añadimos una cuarta tabla: data_user. Se trata de una tabla de dimensiones que se relaciona con transaction 1 a N. A continuación describimos la tabla:

- **Tabla de dimensiones: data_user** con id (Primary Key), name, surname, phone, personal_email, birth_date, country, city, postal_code y address. Contiene información

estática, personal y descriptiva de los usuarios, para contextualizar los datos de la tabla de hechos.

Columna	Tipo de Dato	Características
id	INT (PK)	Identificador único, clave primaria.
name	VARCHAR(100)	Nombre del usuario, texto de longitud variable (hasta 100 caracteres).
surname	VARCHAR(100)	Apellido del usuario, texto de longitud variable (hasta 100 caracteres).
phone	VARCHAR(150)	Teléfono del usuario, texto de longitud variable (hasta 150 caracteres). Información sensible.
personal_email	VARCHAR(150)	Email del usuario, texto de longitud variable (hasta 150 caracteres). Información sensible.
birth_date	VARCHAR(100)	Fecha de nacimiento del usuario, texto de longitud variable (hasta 100 caracteres).
country	VARCHAR(150)	País de residencia del usuario, texto de longitud variable (hasta 150 caracteres).
city	VARCHAR(150)	Ciudad de residencia del usuario, texto de longitud variable (hasta 150 caracteres).
postal_code	VARCHAR(100)	Código postal del usuario, texto de longitud variable (hasta 150 caracteres).
adress	VARCHAR(255)	Dirección del usuario, texto de longitud variable (hasta 150 caracteres). Información sensible.

Exercici 2

L'empresa també et sol·licita crear una vista anomenada "InformeTecnico" que contingui la següent informació:

- *ID de la transacció*
- *Nom de l'usuari/ària*
- *Cognom de l'usuari/ària*
- *IBAN de la targeta de crèdit usada.*
- *Nom de la companyia de la transacció realitzada.*
- *Assegura't d'incloure informació rellevant de totes dues taules i utilitza àlies per a canviar de nom columnes segons sigui necessari.*

Mostra els resultats de la vista, ordena els resultats de manera descendent en funció de la variable ID de transaction

Como en un ejercicio anterior, creamos un VIEW, donde seleccionamos el id de la transacción, el nombre de usuario, el apellido, el iban, el nombre de la compañía, si la transacción utilizamos INNER JOINS para unificar las tablas y lo ordenamos por el id de transacción de forma descendiente.

```
CREATE VIEW InformeTecnico AS
```

```
SELECT transaction.id AS transaccion, CONCAT(data_user.name, " ", surname) AS usuario,
```

company_name AS empresa, iban, declined AS declinada, SUM(amount) AS total_compras
FROM transaction

INNER JOIN data_user ON transaction.user_id = data_user.id

INNER JOIN credit_card ON transaction.credit_card_id = credit_card.id

INNER JOIN company ON transaction.company_id = company.id

GROUP BY transaccion

ORDER BY transaction.id DESC;

SELECT * FROM InformeTecnico;

```

182 • CREATE VIEW InformeTecnico AS
183 SELECT transaction.id AS transaccion, CONCAT(data_user.name, " ", surname) AS usuario,
184 company_name AS empresa, iban, declined AS declinada, SUM(amount) AS total_compras
185 FROM transaction
186 INNER JOIN data_user ON transaction.user_id = data_user.id
187 INNER JOIN credit_card ON transaction.credit_card_id = credit_card.id
188 INNER JOIN company ON transaction.company_id = company.id
189 GROUP BY transaccion
190 ORDER BY transaction.id DESC;
191
192 • SELECT * FROM InformeTecnico;

```

transaction	usuario	empresa	iban	ded
FE96CE47-BD59-381C-4E18-E3CA3D-4E8FF	Kenyon Hartman	Magna A Neque Industries	DO26854763748537475216568689	1
FE809ED4-2DB6-55AC-C915-929516E46468	Molly Gilliam	Nunc Interdum Incorporated	SE2813123487163628531121	0
FD9CBCCD-8E1E-8DA1-4606-7E3A6F3A5A65	Linus Willis	Nunc Interdum Incorporated	KW9485332754781757886242955643	0
FD89D51B-AE8D-77DC-E450-B8083FBD3187	Hilda Levy	Malesuada PC	LT053237077744561475	0
FD2E8957-414B-BEEC-E9AD-59AA7ABA6290	Hedwio Gilbert	Neque Tellus Imperdiet Corp.	GE84848451582810541526	0

#	Time	Action	Message
12	11:11:22	CREATE VIEW InformeTecnico AS SELECT transaction.id AS transaccion, CONC...	0 row(s) affected
13	11:11:24	SELECT * FROM InformeTecnico LIMIT 0, 1000	587 row(s) returned

SCHEMAS

- Filter objects
- Functions
 - sys
- transactions
 - Tables
 - company
 - credit_card
 - data_user
 - transaction
 - Views
 - informetecnico
 - transaccion
 - usuario
 - empresa
 - iban
 - declinada
 - total_compras
 - vistamarketing
 - Stored Procedures
 - Functions
 - world

Administration Schemas