Tarea S4.01. Creació de Base de Dades

Nivel 1

Descàrrega els arxius CSV, estudia'ls i dissenya una base de dades amb un esquema d'estrella que contingui, almenys 4 taules de les quals puguis realitzar les següents consultes:

Reviso las columnas y el tipo de datos de cada una de las tablas adjuntas.

"Transaction" es la tabla central del modelo, y de ella se desprenden las tablas "companies", "credit_cards", "products" y "data_users" (consecuencia de la unión de las 3 tablas users_ca, users_uk y users_usa).

TABLA 1	**TABLA 2**	**TABLA 3**	**TABLA 4**	**TABLA 5**
transactions	**companies**	**credit_cards**	**products**	**data_users**
transaction_id (pk) *	company_id (pk)	credit_card_id (pk) *	product_id (pk) *	user_id (pk) *
credit_card_id (fk)	company_name	user_id (fk)	product_name	name
~~business~~company_id (fk) *	phone	iban	price	surname
product_ids (fk)	email	pin	colour	phone
user_id (fk)	country	pan	weight	email
timestamp	website	cvv	warehouse_id	birth_date
amount		track1		country
declined		track2		city
latitude		expiring_date		postal_code
longitude				address

Para no tener problemas con la carga de los datos, el orden y nombre de los campos se crean tal y como están expresados en los archivos csv, luego reorganizo y modifico los nombres para facilitar las relaciones entre tablas. Los campos que se modificarán están marcados con asteriscos en la tabla anterior.

```sql
        -- Creo la nueva base de datos
CREATE DATABASE IF NOT EXISTS new_transactions; -- para no borrar la bbdd anterior y trabajar sobre una nueva
USE new_transactions;

        -- Creo las 5 tablas:

CREATE TABLE IF NOT EXISTS transactions(
    id VARCHAR(255) NOT NULL,       -- Cambiar nombre
    card_id VARCHAR(15),            -- Cambiar nombre
    business_id VARCHAR(20),        -- Cambiar nombre
    timestamp TIMESTAMP,
    amount DECIMAL(10,2),
    declined BOOLEAN,
    product_ids VARCHAR(100),
    user_id INT,
    lat FLOAT,                      -- Cambiar nombre
    longitude FLOAT,
    PRIMARY KEY (id)                -- Falta añadir las FK
);

CREATE TABLE IF NOT EXISTS companies(
    company_id VARCHAR(15),
    company_name VARCHAR(255),
    phone VARCHAR(15),
    email VARCHAR(100),
    country VARCHAR(100),
    website VARCHAR(100),
    PRIMARY KEY (company_id)        -- Falta añadir las FK
);


CREATE TABLE IF NOT EXISTS credit_cards(
    id VARCHAR(20),
    user_id INT,
    iban VARCHAR(50),
    pan VARCHAR(20),
    pin VARCHAR(4),
    cvv VARCHAR(3),
    track1 VARCHAR(100),
    track2 VARCHAR(100),
    expiring_date DATE,
    PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS products(
    id INT,
    product_name VARCHAR(100),
    price VARCHAR(10),              -- porque están guardados con el símbolo $ --> luego se tendrá que convertir a decimal
    colour VARCHAR(10),
    weight DECIMAL(10,2),
    warehouse_id VARCHAR(10),
    PRIMARY KEY (id)
);

```

```sql
CREATE TABLE IF NOT EXISTS data_users(
    id INT,
    name VARCHAR(100),
    surname VARCHAR(100),
    phone VARCHAR(50),
    email VARCHAR(150),
    birth_date VARCHAR(20),         -- convertir a date
    country VARCHAR(50),
    city VARCHAR(100),
    postal_code VARCHAR(10),
    address VARCHAR (150),
    PRIMARY KEY (id)
);


    -- Cargo los datos correspondientes en cada tabla
            -- Import records from an external file -- hecho
```

La relación entre las tablas credit_cards – transactions, companies – transactions y data_users – transactions es de tipo 1 a n, y están relacionadas mediante sus PK que funcionan como FK en la tabla principal.

```sql
    -- Modificaciones
ALTER TABLE credit_cards
RENAME COLUMN id TO credit_card_id;
UPDATE credit_cards
SET expiring_date = STR_TO_DATE(expiring_date, '%m/%d/%y');
ALTER TABLE credit_cards
MODIFY COLUMN expiring_date DATE;

ALTER TABLE products
RENAME COLUMN id TO product_id;

ALTER TABLE data_users
RENAME COLUMN id TO user_id;
UPDATE data_users
SET birth_date = STR_TO_DATE(birth_date, '%b %d, %Y');
ALTER TABLE data_users
MODIFY COLUMN birth_date DATE;

ALTER TABLE transactions
RENAME COLUMN id TO transaction_id,
RENAME COLUMN business_id TO company_id,
RENAME COLUMN card_id TO credit_card_id,
RENAME COLUMN product_ids TO product_list;

    -- Añadir FKs --> relaciones 1 a n entre tablas
SET foreign_key_checks = 0;         -- Desactivar primero los fk-checks y volver a activarlos después de las modificaciones.
ALTER TABLE transactions
ADD CONSTRAINT fk_credit_cards_transactions
FOREIGN KEY (credit_card_id) REFERENCES credit_cards(credit_card_id),
ADD CONSTRAINT fk_companies_transactions
FOREIGN KEY (company_id) REFERENCES companies(company_id),
ADD CONSTRAINT fk_data_users_transactions
FOREIGN KEY (user_id) REFERENCES data_users(user_id);
SET foreign_key_checks = 1;
```

La relación entre las tablas products – transactions es más compleja, ya que la columna transactions.product_ids es una lista de ids de productos, por lo que la relación entre las tablas es n a n. Para evitar problemas, se va a crear una tabla intermedia que relacione directamente transaction_id con product_id, donde cada una de ellas será una FK relacionada respectivamente con las tablas transactions y products, y a la vez cada combinación transaction_id – product_id es única y actuará como PK.

Primero reviso toda la columna y observo que como máximo hay 4 ids en cada lista. Divido la lista de ids por las comas con la función SUBSTRING_INDEX y empiezo seleccionando el del final (por eso hay un -1). Para saber cuántos ids hay en la lista, defino el índice n, que corresponde al número de comas + 1 = total productos.

Con la función UNION puedo unir en diferentes filas consecutivas el id de la lista (empezando por el final) con su transaction_id correspondiente. Uso la función CHARACTER_LENGHT para relacionar la longitud de la lista con el número de filas generadas.

```sql
-- Crear tabla intermedia trans_prod
CREATE TABLE trans_prod(
    transaction_id VARCHAR(255),
    product_id INT,
    PRIMARY KEY (transaction_id, product_id),
    FOREIGN KEY (transaction_id) REFERENCES transactions(transaction_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
    );

    -- Para poder rellenar esta tabla, es necesario descomponer la columna product_list (strings functions) y crear compilaciones únicas transaction_id - product_id
        -- Funciones SUBSTRING_INDEX(string, delimiter, number) y CHARACTER_LENGTH(string)
INSERT INTO trans_prod (transaction_id, product_id)
SELECT
    t.transaction_id,
    SUBSTRING_INDEX(SUBSTRING_INDEX(t.product_list, ',', n), ',', -1) AS product_id
FROM
    transactions AS t
JOIN
    (SELECT 1 AS n UNION SELECT 2 UNION SELECT 3 UNION SELECT 4) n  -- Máximo 4 productos en las listas, los concateno uno en cada fila
    ON CHARACTER_LENGTH(t.product_list) - CHARACTER_LENGTH(REPLACE(t.product_list, ',', '')) >= n - 1;  -- filtro que conecta cada conjunto n con el valor de la lista de productos
                                -- índice de n = número de comas + 1. Así incluye una nueva fila por cada producto en la lista
        -- Compruebo:
      SELECT * FROM new_transactions.trans_prod;
```

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

transaction_id	product_id
02C6201E-D90A-1859-B4EE-88D2986D3B02	1
122DC333-E19F-D629-DCD8-9C54CF1EBB9A	1
1753A288-9FC1-52E6-5C39-A1FFB97B0D3A	1
1A6CECFB-2E3A-65A3-72D9-2FDB58A1E4BA	1
1EA2B262-D507-AD14-4374-4D5329671 13F	1

trans_prod 5

Output

Action Output

#	Time	Action	Message
1	10:46:35	SELECT * FROM new_transactions.trans_prod	1457 row(s) returned

Falta eliminar el símbolo de dólar ($) en la columna products.price y cambiar el tipo de dato de VARCHAR a DECIMAL(10,2):

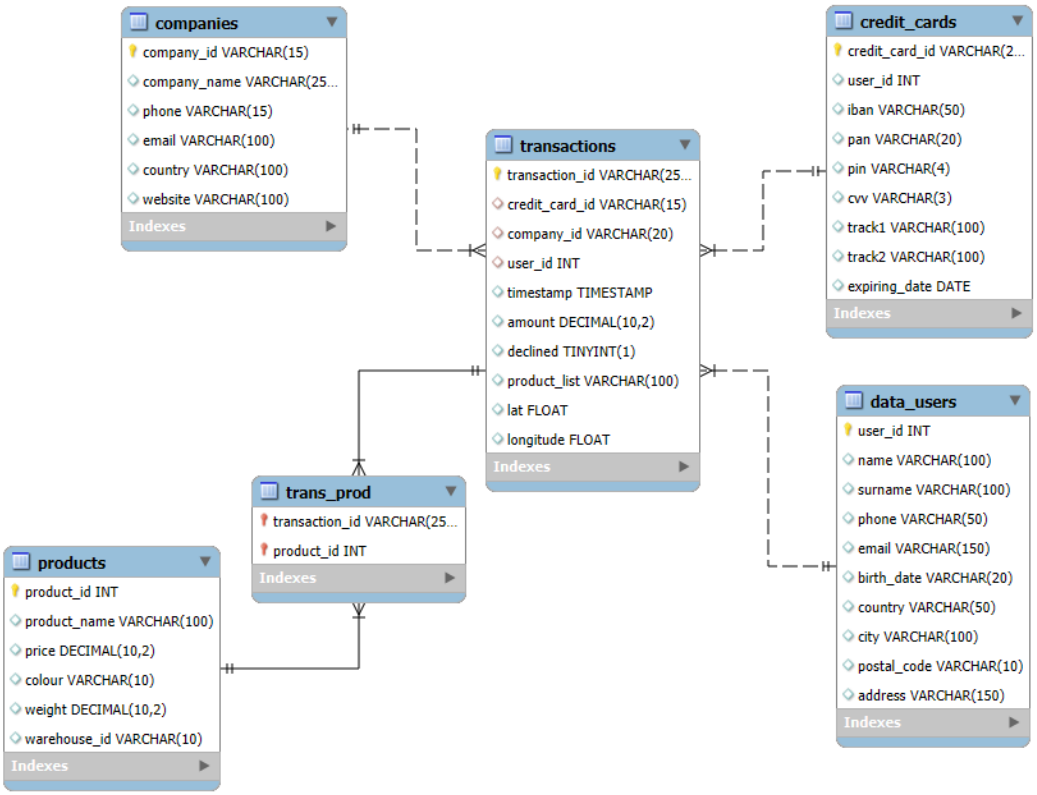
```sql
    -- Eliminar símbolo $ de products.price y convertir datos a decimal(10,2)
UPDATE products
SET price = REPLACE(price, '$', '');
ALTER TABLE products
```

Output

Action Output

#	Time	Action	Message
157	19:31:57	ALTER TABLE price MODIFY price DECIMAL(10,2)	Error Code: 1146. Table 'new_transactions.price' doesn't exist
158	19:32:04	ALTER TABLE products MODIFY price DECIMAL(10,2)	100 row(s) affected Records: 100 Duplicates: 0 Warnings: 0

Por último, el modelo finalizado quedaría así:



Ejercicio 1 –

Realitza una subconsulta que mostri tots els usuaris amb més de 30 transaccions utilitzant almenys 2 taules.

Usando las tablas transactions y data_users, he podido determinar que solo hay 3 usuarios que tienen más de 30 transacciones aceptadas (transactions.declined = 0).

```sql
SELECT t.user_id,
    (SELECT name
    FROM data_users AS du
    WHERE du.user_id = t.user_id) AS name,
    (SELECT surname
    FROM data_users AS du
    WHERE du.user_id = t.user_id) AS surname,
    COUNT(t.transaction_id) AS total_transactions
FROM transactions AS t
WHERE t.declined = 0
GROUP BY t.user_id,
    (SELECT name
    FROM data_users AS du
    WHERE du.user_id = t.user_id),
    (SELECT surname
    FROM data_users AS du
    WHERE du.user_id = t.user_id)
HAVING COUNT(t.transaction_id) > 30; -- 3 usuarios
```

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

user_id	name	surname	total_transactions
92	Lynn	Riddle	39
267	Ocean	Nelson	39
272	Hedwig	Gilbert	38

Result 14

Output

Action Output

#	Time	Action	Message
1	11:24:52	SELECT t.user_id, (SELECT name FROM data_users AS du WHERE du.user_id = t.user_id) AS name, (SELECT surname FROM...	3 row(s) returned

```sql
-- De forma más simplificada se podría hacer:
SELECT du.user_id, du.name, du.surname
FROM data_users AS du
WHERE du.user_id IN (
        SELECT t.user_id
        FROM transactions AS t
        WHERE t.declined = 0
        GROUP BY t.user_id
        HAVING COUNT(t.transaction_id) > 30);   -- 3 usuarios
```

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

user_id	name	surname
92	Lynn	Riddle
267	Ocean	Nelson
272	Hedwig	Gilbert
NULL	NULL	NULL

data_users 16

Output

Action Output

#	Time	Action	Message
1	11:28:50	SELECT du.user_id, du.name, du.surname FROM data_users AS du WHERE du.user_id IN (SELECT t.user_id FROM transactions AS t W...	3 row(s) returned

Ejercicio 2 –

Mostra la mitjana d'amount per IBAN de les targetes de crèdit a la companyia Donec Ltd, utilitza almenys 2 taules.

Los pagos a esta compañía se hicieron solo con una tarjeta de crédito, el valor medio de las cantidades es el siguiente:

```sql
SELECT cc.iban, ROUND(AVG(t.amount),2) AS average_amount
FROM transactions AS t
JOIN credit_cards AS cc
ON t.credit_card_id = cc.credit_card_id
JOIN companies AS c
ON t.company_id = c.company_id
WHERE c.company_name = 'Donec Ltd' AND t.declined = 0
GROUP BY cc.iban;
```

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

iban	average_amount
PT87806228135092429456346	42.82

Result 16

Output

Action Output

#	Time	Action	Message
1	21:22:23	SELECT cc.iban, ROUND(AVG(t.amount),2) AS average_amount FROM transactions AS t JOIN credit_cards AS cc ON t.credit_card_id = cc.credit_c...	1 row(s) returned

Nivel 2

Crea una nova taula que reflecteixi l'estat de les targetes de crèdit basat en si les últimes tres transaccions van ser declinades i genera la següent consulta:

Primero creo la tabla de dos columnas y marco la PK y FK correspondientes.

Para rellenar el contenido de la tabla status voy a hacer uso de una función ventana para acotar solamente las últimas 3 transacciones. Con una función CASE WHEN voy a sumar cuántas transacciones fueron rechazadas y, en caso de sumar 3, la tarjeta se cataloga como inactiva.

```sql
        -- nueva tabla con 2 columnas. una de ellas ya existe en tabla cc, la otra es nueva y generada a partir de otras tablas --> CASE WHEN
        -- 3 últimas transacciones rechazadas: active = false = 0.
        -- ordenarlas por timestamp desc para tener las 3 últimas.
        -- con la función ventana determino solo esas 3 últimas (ordenadas time desc)
CREATE TABLE card_status(
    credit_card_id VARCHAR(20),
    active BOOLEAN,
    PRIMARY KEY (credit_card_id),
    FOREIGN KEY (credit_card_id) REFERENCES credit_cards(credit_card_id)
    );

INSERT INTO card_status(credit_card_id, active)
SELECT
    t2.credit_card_id,
    CASE            -- suma 1 con cada transferencia rechazada. Si el resultado da 3 --> tarjeta inactiva
        WHEN SUM(t2.declined) = 3 THEN 0
        ELSE 1
    END AS status
FROM (          -- subQ alias t2 en FROM --> row_number para asignar número de fila
    SELECT
        credit_card_id,
        declined,
        ROW_NUMBER() OVER (PARTITION BY credit_card_id ORDER BY timestamp DESC) AS row_num      -- función ventana ROW_NUMBER
    FROM transactions
) AS t2
WHERE t2.row_num <= 3       -- filtramos sólo las 3 últimas transacciones
GROUP BY t2.credit_card_id;
```

Ejercicio 1 –

Quantes targetes estan actives?

Todas las tarjetas están activas.

```sql
/* Ejercicio 1:
Quantes targetes estan actives? */
SELECT COUNT(active) AS active_cards
FROM card_status
WHERE active = 1;   -- 275 activas, es decir, todas.
```

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

active_cards
275

Result 4

Output

Action Output

#	Time	Action	Message
1	10:15:54	SELECT COUNT(active) AS active_cards FROM card_status WHERE active = 1	1 row(s) returned

Nivel 3

Crea una taula amb la qual puguem unir les dades del nou arxiu products.csv amb la base de dades creada, tenint en compte que des de transaction tens product_ids. Genera la següent consulta:

La tabla trans_prod se ha creado anteriormente.

Ejercicio 1 –

Necessitem conèixer el nombre de vegades que s'ha venut cada producte.

```sql
SELECT
    p.product_name,
    COUNT(tp.product_id) AS total_sales
FROM products AS p
JOIN trans_prod AS tp
ON p.product_id = tp.product_id
JOIN transactions AS t
ON tp.transaction_id = t.transaction_id
WHERE t.declined = 0
GROUP BY p.product_id, p.product_name
ORDER BY total_sales DESC;
```

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

product_name	total_sales
riverlands north	60
Winterfell	59
Tarly Stark	56
duel	54
skywalker ewok sith	54
jinn Winterfell	53
Direwolf riverlands the	52

Result 42

Output

Action Output

#	Time	Action	Message
1	22:42:21	SELECT p.product_name, COUNT(tp.product_id) AS total_sales FROM products AS p JOIN trans_prod AS tp ON p.product_id = tp.product_id ...	26 row(s) returned