SPRINT 4: CREACIÓN DE BASES DE DATOS

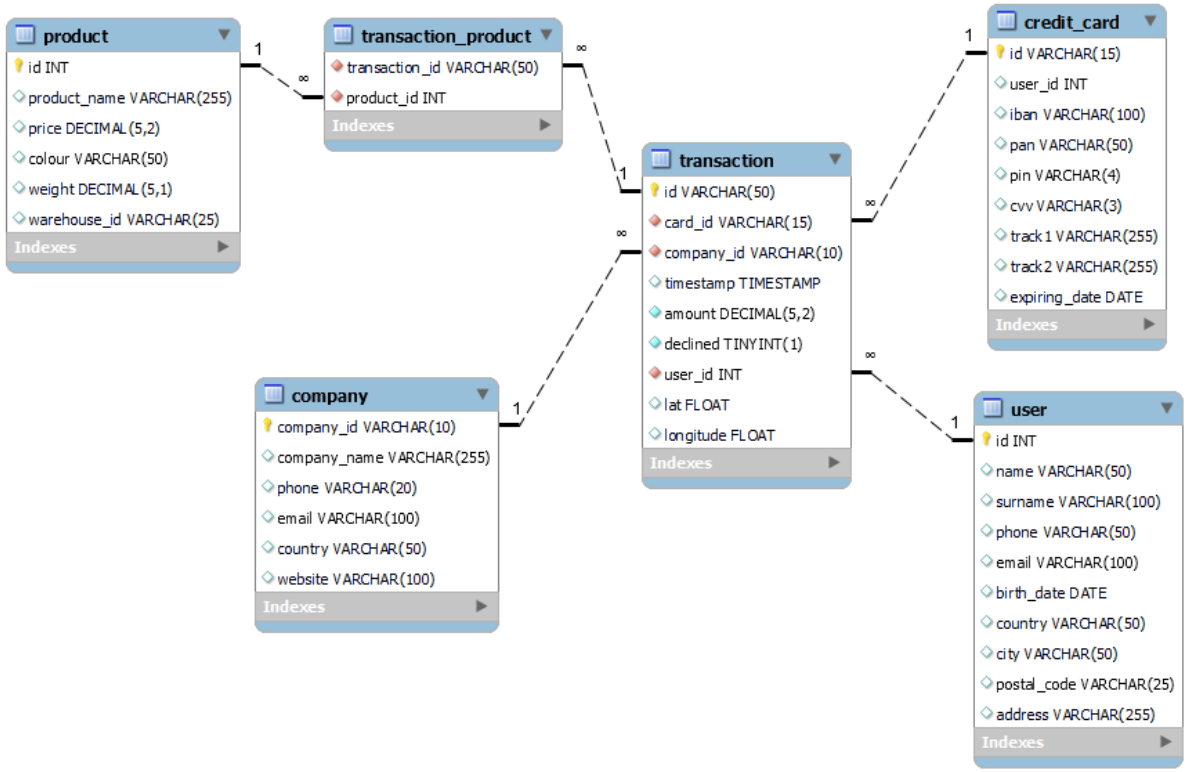
Descarga los archivos CSV, estúdialos y diseña una base de datos con un esquema de estrella que contenga, al menos, 4 tablas.

Los archivos CSV contenían registros sobre las transacciones, las empresas que participan en dichas transacciones, las tarjetas de crédito utilizadas, los usuarios que las realizaron (de tres países: Reino Unido, Estados Unidos y Canadá) y los productos vendidos. He creado una base de datos llamada *operations* y las tablas correspondientes para importar los datos de los archivos CSV usando LOAD DATA INFILE (véase script estructura_operationsdb.sql para más detalles).

Los registros de transacciones contenían varios identificativos de productos por transacción en una única columna, cosa que complicaba filtrar los datos de un único producto. Para solucionarlo, he creado una tabla nueva (transaction_product) para los ids de transacciones y los ids de productos. A continuación, he transformado la columna de product_ids de transactions.csv para separar cada producto de una transacción en una fila. *

He escogido los tipos de datos de las variables en función de la información que contenían. Por ejemplo, las columnas que contenían fechas las he definido como DATE, las que contenían números decimales (amount, price y weight) las he definido como DECIMAL, las que contenían id's de usuario/producto como INT, etc. **

En este modelo tenemos la tabla de hechos transaction, que conecta con las tablas dimensión credit_card ***, user y company con una relación N:1 (transaction N:1 dimensión). Para relacionar transaction con dimensión product tenemos la tabla intermedia transaction_product, que conecta N:1 con transaction y N:1 con product (véase diagrama a continuación).



* NOTA 1: He realizado la transformación de productos por fila en MySQL. Para conseguir esta modificación, he necesitado los siguientes pasos:

- Crear una tabla temporal para cargar las columnas transaction_id y product_ids (en formato VARCHAR, al ser múltiples valores separados por coma; no se detectaría como INT)
- Importar las columnas mencionadas a esta tabla temporal

```
CREATE TEMPORARY TABLE transaction_product_temp (
    transaction_id VARCHAR(50),
    product_ids VARCHAR(100)
);

LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/transactions.csv'
INTO TABLE transaction_product_temp
FIELDS TERMINATED BY ';'
LINES TERMINATED BY '\r\n'
IGNORE 1 ROWS
(transaction_id, @dummy, @dummy, @dummy, @dummy, @dummy, product_ids, @dummy, @dummy, @dummy);
```

- Separar los productos en filas separadas e insertar los registros en la tabla transaction_product:
    - Creamos una columna, llamada n, para establecer un índice interno que nos ayudará a separar cada registro de la columna product_ids; la relacionamos con la tabla temporal con los datos de product_ids (en VARCHAR) con la condición de que la longitud product_ids – longitud product_ids (sin comas) sea superior o igual a n-1. Por ejemplo:
        - Product_ids 71, 41 ⟶ char_length 6 – char_length 5 = 1 ≥ n-1
            - n=2 porque tenemos dos registros de products_ids
        - Product_ids 47, 37, 11, 1 ⟶ char_length 13 – char_length 10 = 3 ≥ n-1
            - n=4 porque tenemos cuatro registros de product_ids
    - Separamos los registros de product_ids con SUBSTRING_INDEX. Por ejemplo:
        - Product_ids 71, 41
            - **n=1** ⟶ **71**
            - **n=2** ⟶ 71, 41 ⟶ SUBSTRING_INDEX ((71, 41), ‘,’ -1) ⟶ **41**
            - n=3: nada porque no se cumpliría que la longitud de caracteres es superior que n-1 ⟶ 1 no es superior que 2
        - Product_ids 47, 37, 11, 1
            - **n=1** ⟶ **47**
            - **n=2** ⟶ 47, 37 ⟶ SUBSTRING_INDEX ((47, 37), ‘,’ -1) ⟶ **37**
            - **n=3** ⟶ 47, 37, 11 ⟶ SUBSTRING_INDEX ((47, 37, 11), ‘,’ -1) ⟶ **11**
            - **n=4** ⟶ 47, 37, 11, 1 ⟶ SUBSTRING_INDEX ((47, 37, 11, 1), ‘,’ -1) ⟶ **1**

      Si el número generado (n) es menor o igual al número de elementos en la lista, se extrae el elemento correspondiente de la cadena product_ids. Si no, esa fila de la subconsulta numbers no contribuirá a la extracción de elementos para esa fila de la tabla.

```
INSERT INTO transaction_product (transaction_id, product_id)
SELECT  transaction_id,
        SUBSTRING_INDEX(SUBSTRING_INDEX(product_ids, ',', numbers.n), ',', -1) AS product_id
FROM transaction_product_temp
JOIN (  SELECT 1 AS n
        UNION ALL SELECT 2
        UNION ALL SELECT 3
        UNION ALL SELECT 4  ) AS numbers
ON CHAR_LENGTH(product_ids) - CHAR_LENGTH(REPLACE(product_ids, ',', '')) >= n - 1;
```

- Eliminar la tabla temporal creada

```
DROP TEMPORARY TABLE transaction_product_temp;
```

** NOTA 2: Para garantizar una carga precisa de los datos, he agregado cláusulas SET para ajustar datatype de las columnas del archivo CSV de origen.

- Para modificar el formato de fechas de origen a YYYY-MM-DD (DATE): STR_TO_DATE; modificación similar para birth_date (tabla user)

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/credit_cards.csv'
INTO TABLE credit_card
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(id, user_id, iban, pan, pin, cvv, track1, track2, @expiring_date)
SET expiring_date=STR_TO_DATE(@expiring_date,'%c/%d/%y');
```

- Para quitar el símbolo de $ de la columna price y, así, se pueda detectar como DECIMAL: REPLACE

```
LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/products.csv'
INTO TABLE product
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(id, product_name, @price, colour, weight, warehouse_id)
SET price=REPLACE(@price,'$','');
```

*** NOTA 3: La tabla credit_card contiene el identificador de los usuarios que son propietarios de las tarjetas de crédito. No he establecido una relación entre las tablas credit_card y user ya que se formaría una relación cíclica en mi base de datos.

NIVEL 1

Ejercicio 1. Realiza una subconsulta que muestre a todos los usuarios con más de 30 transacciones utilizando al menos 2 tablas.

Subconsulta: cuento la cantidad de registros en la tabla transaction con la condición de que los identificadores correspondan a los presentes en la tabla user. El conteo se realiza en función de los usuarios (GROUP BY user.id).

Como filtro, aplico que la cantidad de transacciones sea superior a 30.

Selecciono todos los campos de la tabla user para tener datos personales completos de los usuarios.

```
SELECT  u.*,
        (SELECT COUNT(id)
        FROM transaction t
        WHERE t.user_id = u.id) AS num_transacciones
FROM user u
GROUP BY u.id
HAVING num_transacciones > 30;
```

Obtenemos un total de cuatro usuarios que cumplen el criterio establecido: Lynn Riddle (id 92), Ocean Nelson (id 267), Hedwig Gilbert (id 272) y Kenyon Hartman (id 275). Sus datos personales y la cantidad de transacciones realizadas por cada uno de ellos se muestran en la tabla a continuación.

id	name	surname	phone	email	birth_date	country	city	postal_code	address	num_transacciones
92	Lynn	Riddle	1-387-885-4...	vitae.aliquet@outl...	1984-09-21	United States	Bozeman	61871	P.O. Box 712, ...	39
267	Ocean	Nelson	079-481-2745	aenean@yahoo.c...	1991-12-26	Canada	Charlottetown	85X 3P4	Ap #732-8357...	52
272	Hedwig	Gilbert	064-204-8788	sem.eget@icloud....	1991-04-16	Canada	Tuktoyaktuk	Q4C 3G7	P.O. Box 496, ...	76
275	Kenyon	Hartman	082-871-7248	convallis.ante.lect...	1982-08-03	Canada	Richmond	R8H 2K2	8564 Facilisi. St.	48

Ejercicio 2. Muestra el promedio de las transacciones por IBAN de las tarjetas de crédito en la compañía Donec Ltd. utilizando al menos 2 tablas.

En la tabla de transacciones tenemos id_company pero no su nombre, por lo que tenemos que recurrir a la tabla company mediante una subconsulta para conocer el identificador de la compañía Donec Ltd.

Una vez sabemos el id, podemos calcular el promedio de las transacciones de esta compañía en función de las tarjetas de crédito utilizadas (GROUP BY card_id).

Al tener ya el promedio por tarjeta, solo falta relacionarlo con la tabla credit_card para tener el número de IBAN asociado a la tarjeta.

```
WITH avg_trans_card AS
(
SELECT card_id, ROUND(AVG(amount),2) AS avg_amount
FROM transaction
WHERE company_id = (SELECT company_id
                    FROM company
                    WHERE company_name='Donec Ltd')
GROUP BY card_id
)
SELECT iban, atc.avg_amount AS promedio_transaccion
FROM credit_card cc
JOIN avg_trans_card atc
ON cc.id=atc.card_id;
```

En este caso, la compañía Donec Ltd. tenía tan solo dos transacciones y las dos se realizaron con la misma tarjeta de crédito. Por tanto, como resultado obtenemos un número iban (mostrado en la tabla) con un promedio de 203.72 euros.

iban	promedio_transaccion
PT87806228135092429456346	203.72

NIVEL 2

Crea una nueva tabla que refleje el estado de las tarjetas de crédito basado en si las últimas tres transacciones fueron declinadas.

Para crear una tabla que muestre el estado de las tarjetas, debemos conseguir las últimas tres transacciones realizadas con las tarjetas. Para ello, seleccionamos información relevante, como es el id de la tarjeta, timestamp (fecha y hora de la transacción) y declined (0: aceptada, 1: rechazada).

Dado que el número de transacciones por tarjeta varían, debemos aplicarle un "índice" interno a cada fila de transacción de una determinada tarjeta con la función ROW_NUMBER ().

Al usar PARTITION BY card_id, estamos indicando a la función que separe las asignaciones de números secuenciales para cada tarjeta individualmente. Es decir, cuando terminan las transacciones de una tarjeta y comienzan las de otra, el índice se reinicia a 1 para la nueva tarjeta. Esto asegura que cada tarjeta tenga su propio conjunto de índices únicos para sus transacciones.

Además, al especificar ORDER BY timestamp DESC, estamos ordenando las transacciones dentro de cada grupo de tarjeta de más recientes a las más antiguas.

Para determinar si una tarjeta está activa o no, hay que considerar las 3 transacciones más recientes de cada tarjeta (row_transaction <= 3).

En la tabla creada tendremos dos columnas, la primera con el identificador de la tarjeta y la segunda con el estado de la tarjeta: 'operativa' o 'inoperativa' (asignamos el estado con CASE).

- Si las tres transacciones son rechazadas, la suma de declined sería 3, por tanto 'inoperativa'
- Si hay una o dos transacciones rechazadas de las tres a considerar, la suma de declined sería <= 2, por tanto 'operativa'

Al ejecutar el código, insertamos un total de 275 registros de tarjeta con su respectivo estado en la tabla card_status.

```sql
CREATE TABLE IF NOT EXISTS card_status (
    id_tarjeta VARCHAR(15),
    estado_tarjeta VARCHAR(50)
);

INSERT INTO card_status (id_tarjeta, estado_tarjeta)
WITH trans_card AS
(
SELECT  card_id,
        timestamp,
        declined,
        ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS row_transaction
FROM transaction
)
SELECT  card_id AS id_tarjeta,
        CASE
            WHEN SUM(declined) <= 2 THEN 'operativa'
            ELSE 'inoperativa'
        END AS estado_tarjeta
FROM trans_card
WHERE row_transaction <= 3
GROUP BY id_tarjeta;
```

<u>Ejercicio 1.</u> ¿Cuántas tarjetas están activas?

Una vez tenemos creada la tabla con el estado de las tarjetas, contamos cuántas están operativas.

```sql
SELECT COUNT(*) AS 'cantidad tarjetas activas'
FROM card_status
WHERE estado_tarjeta='operativa';
```

Como resultado, obtenemos 275 tarjetas activas. Cabe destacar que en la tabla creada tenemos un total de 275 tarjetas; por tanto, todas las tarjetas presentes en card_status están activas.

cantidad tarjetas activas
275

NIVEL 3

<u>Ejercicio 1.</u> Necesitamos conocer el número de veces que se ha vendido cada producto.

En mi informe he querido separar en columnas separadas la cantidad de transacciones aceptadas y la cantidad de rechazadas para cada producto, además del total de ventas.

Con la tabla transaction_product podemos saber la cantidad total de transacciones que se han registrado para un determinado producto; pero, tal como he diseñado la tabla, no podemos distinguir si esas transacciones fueron aceptadas o rechazadas. Para ello, tenemos que recurrir a la tabla transaction mediante una JOIN.

Además, para saber el nombre del producto tenemos que recurrir a la tabla product mediante otra JOIN. En este caso utilizo RIGHT JOIN para que me aparezcan todos los nombres de los productos, incluso aquellos que no tienen ninguna transacción registrada en la tabla transaction_product.

Por tanto, desde la tabla intermedia transaction_product relaciono la tabla transaction con una INNER JOIN y la tabla product con RIGHT JOIN.

Para calcular la cantidad de transacciones, utilizo la función SUM con la condición descrita en CASE.

- En la columna cantidad_vendida aplico la condición de que todas las transacciones aceptadas (declined 0) tengan valor 1 y las rechazadas 0, por lo que sumando las veces que aparece cada producto obtenemos el total de ventas exitosas.
- En la columna cantidad_rechazada aplico la condición de que todas las transacciones rechazadas (declined 1) tengan valor 1 y las aceptadas 0, por lo que la suma daría el total de ventas rechazadas para cada producto.

En la columna cantidad_total aplico la función COUNT sin ninguna condición para calcular la cantidad total de transacciones registradas para cada producto.

Finalmente ordeno los resultados de mayor a menor cantidad total de transacciones y, en caso de empate, por el nombre del producto en orden alfabético.

NOTA: Al agrupar por el nombre del producto, estoy asumiendo que los productos con el mismo nombre, pero diferentes identificadores, corresponden al mismo artículo, con posibles variaciones en peso y/o color.

```sql
SELECT  p.product_name AS producto,
        SUM(CASE WHEN t.declined=0 THEN 1 ELSE 0 END) AS cantidad_vendida,
        SUM(CASE when t.declined=1 THEN 1 ELSE 0 END) AS cantidad_rechazada,
        COUNT(tp.product_id) AS cantidad_total
FROM transaction_product tp
JOIN transaction t
    ON tp.transaction_id=t.id
RIGHT JOIN product p
    ON tp.product_id=p.id
GROUP BY producto
ORDER BY cantidad_total DESC, producto;
```

Con esta consulta, obtenemos 70 registros, lo que corresponde a la cantidad total de productos con nombre único en la tabla product. De estos, unicamente 24 productos tienen transacciones registradas, por lo que los 46 restantes aparecen en la lista de resultados, pero con cantidad 0.

Como podemos observar en la tabla a continuación, el producto más vendido es Direwolf Stannis, con un total de 106 transacciones registradas (86 exitosas y 20 rechazadas).

producto	cantidad_vendida	cantidad_rechazada	cantidad_total
Direwolf Stannis	86	20	106
skywalker ewok	88	12	100
riverlands north	60	8	68
Winterfell	59	9	68
Direwolf riverlands the	52	14	66