>>>>>>>>>>>>>>>>>>>> S P R I N T 4 <<<<<<<<<<<<<<<<<<<<

Partiendo de algunos archivos CSV diseñarás y crearás tu base de datos.

========== NIVEL 1 ==========

Descarga los archivos CSV, estúdiales y diseña una base de datos con un esquema de estrella que contenga, al menos 4 tablas de las que puedas realizar las siguientes consultas:

Código:
```
CREATE DATABASE db_SPRINT4;
-- ------------------------------------------------------------

CREATE TABLE company (
        id VARCHAR(20) PRIMARY KEY not null,
        company_name VARCHAR(255) null,
        phone VARCHAR(15) null,
        email VARCHAR(150) null,
        country VARCHAR(150) null,
        website VARCHAR(150) null
);

-- ------------------------------------------------------

create table credit_card (
        id VARCHAR(20) PRIMARY KEY not null,
        user_id VARCHAR(20),
        iban VARCHAR(255) null,
        pan VARCHAR(45) null,
        pin CHAR(4) null,
        cvv CHAR(3) null,
        track1 VARCHAR(255),
        track2 VARCHAR(255),
        expiring_date varchar(255) null
);

-- ------------------------------------------------------

CREATE TABLE products (
        id INT PRIMARY KEY not null,
        product_name VARCHAR(100),
        price VARCHAR(10),
        colour VARCHAR(100),
        weight VARCHAR(100),
        warehouse_id VARCHAR(100),
);
-- ------------------------------------------------------

CREATE TABLE users (
        id INT PRIMARY KEY not null,
        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)
);
-- ------------------------------------------------------

CREATE TABLE transactions (
        id VARCHAR(255) PRIMARY KEY NOT NULL,
```

```
    card_id VARCHAR(20),
    bussiness_id VARCHAR(150),
    timestamp varchar(150) NOT NULL,
    amount DECIMAL(10,2) NOT NULL,
    declined TINYINT(1) NOT NULL DEFAULT 0,
    product_ids VARCHAR(20) NOT NULL,
    user_id INT,
    lat VARCHAR(50),
    longitude VARCHAR(50)
);
```

conforme la elaboración de todas las tablas necesarias, se planifica as relaciones entre ellas, se establece la indexación con sus respectivos foreign key y evitarnos posibles inconvenientes

```
CREATE INDEX idx_company
    ON transaction(business_id);
ALTER TABLE company
    ADD FOREIGN KEY (id) REFERENCES transactions(business_id);

CREATE INDEX idx_credit_card
    ON transaction(card_id);
ALTER TABLE credit_card
    ADD FOREIGN KEY (id) REFERENCES transactions(card_id);

CREATE INDEX idx_users
    ON transaction(user_id);
ALTER TABLE users
    ADD FOREIGN KEY (id) REFERENCES transactions(user_id);

CREATE INDEX idx_products
    ON transaction(product_ids);
ALTER TABLE products
    ADD FOREIGN KEY (id) REFERENCES transactions(product_ids);
```

Debemos comentar que, analizando los campos o columnas de las tablas users **“users_ca, users_usa, users_uk”** la opción clara es poder hacer una importación unificada a una sola tabla **‘users’**.

De esta forma simplifica el modelo y facilita la gestión de datos, reduce la redundancia y mejora la integridad de datos, permite un análisis más eficiente de los datos de usuarios a nivel global, la relación entre "Usuarios" y "Transacciones" se mantiene de uno a muchos (1:M), donde un usuario puede realizar muchas transacciones.

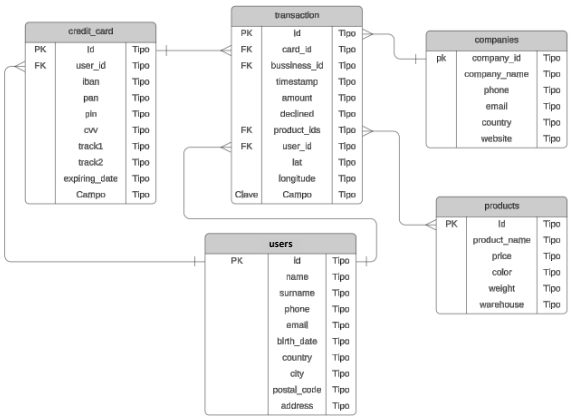

Diagrama conceptual inicial – todas las relaciones

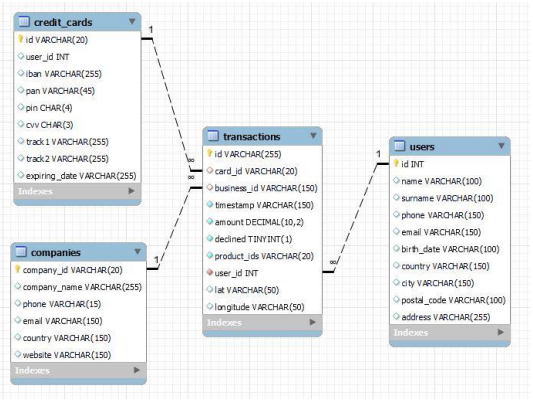

Diagrama de relaciones de FK en las tablas a usarse inicialmente

Ejercicio 1

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

```
SELECT id,
        concat(NAME, ' ',SURNAME) AS "nombres completos",(
            SELECT count(transactions.id)
            FROM transactions
            WHERE users.id = transactions.user_id) AS transacciones
FROM users
WHERE id IN (
        SELECT user_id
    FROM transactions
    GROUP BY user_id
    HAVING COUNT(id) > 30)

ORDER BY transacciones DESC;
```

id	nombres completos	transacciones
272	Hedwig Gilbert	76
267	Ocean Nelson	52
275	Kenyon Hartman	48
92	Lynn Riddle	39

Como una buena práctica en la profundización en el dominio del manejo de datos, desgloso algunas opciones

Opción consulta sencilla con JOIN

```
SELECT u.id,
        u.name as nombre,
        u.surname AS apellido,
        count(t.id) as total_transaccions
FROM users AS u
INNER JOIN transactions AS t ON u.id = t.user_id
GROUP BY u.id, u.name, u.surname
HAVING COUNT(t.id) > 30
ORDER BY total_transaccions DESC;
```

id	nombre	apellido	total_transaccions
272	Hedwig	Gilbert	76
267	Ocean	Nelson	52
275	Kenyon	Hartman	48
92	Lynn	Riddle	39

Opción consulta sencilla con JOIN concatenando los nombres

```
SELECT t.user_id,
        concat(NAME, ' ',SURNAME) AS "nombres completos",
        count(t.id) AS total_transaccions
FROM users AS u
INNER JOIN transactions AS t ON u.id = t.user_id
GROUP BY u.id, u.name, u.surname
HAVING  count(t.id) >30
ORDER BY total_transaccions DESC;
```

user_id	nombres completos	total_transaccions
272	Hedwig Gilbert	76
267	Ocean Nelson	52
275	Kenyon Hartman	48
92	Lynn Riddle	39

Opción con subconsulta dentro de INNER JOIN concatenando los nombres

```
SELECT t.user_id,
        concat(NAME, ' ',SURNAME) AS "nombres completos",
        total_transactions
FROM users AS u
INNER JOIN (
        SELECT user_id, COUNT(*) AS total_transactions
        FROM transactions
        GROUP BY user_id
        HAVING COUNT(*) > 30
        ) AS t ON u.id = t.user_id
ORDER BY total_transactions DESC;
```

user_id	nombres completos	total_transactions
272	Hedwig Gilbert	76
267	Ocean Nelson	52
275	Kenyon Hartman	48
92	Lynn Riddle	39

Opción con **subconsulta** dentro de INNER JOIN, **concatenando** los nombres y el **monto total** con 2 decimales

```
SELECT t.user_id,
        concat(u.NAME, ' ',u.SURNAME) AS "nombres completos",
   total_transacciones,
   monto_total
FROM users AS u
INNER JOIN (
        SELECT user_id, count(*) AS total_transacciones,
                FORMAT (SUM(t.amount), 'f2') AS monto_total
        FROM transactions AS t
        GROUP BY user_id
        HAVING count(*) >30
        ) AS t ON u.id = t.user_id
ORDER BY total_transacciones DESC;
```

user_id	nombres completos	total_transacciones	monto_total
272	Hedwig Gilbert	76	18,351
267	Ocean Nelson	52	13,052
275	Kenyon Hartman	48	12,012
92	Lynn Riddle	39	11,452

Ejercicio 2

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

```
SELECT co.company_name,
        cc.iban,
        ROUND(AVG(t.amount),2) AS promedio_suma_transacciones
FROM companies co
INNER JOIN transactions t ON co.company_id = t.business_id
INNER JOIN credit_cards cc ON t.card_id = cc.id
WHERE co.company_name = 'Donec Ltd'
GROUP BY co.company_name, cc.iban;
```

company_name	promedio_suma_transacciones	iban
Donec Ltd	203.715000	PT87806228135092429456346

========== NIVEL 2 ==========

Crea una nueva tabla que refleje el estado de las tarjetas de crédito basado en si las últimas tres transacciones fueron declinadas y genera la siguiente consulta:

```
-- se crea una nueva tabla
CREATE TABLE card_status (
        id INT,
        card_id VARCHAR(15),
   status VARCHAR(50));
```

```
-- Se ingresa los datos de tablas establecidas condicionandolo con filtros según el pedido

INSERT INTO card_status (card_id, status)
(WITH transacciones_tarjeta AS (
SELECT card_id,
          timestamp,
          declined,
          ROW_NUMBER() OVER (PARTITION BY card_id ORDER BY timestamp DESC) AS
row_transaction
FROM transactions)
SELECT card_id AS numero_tarjeta,
   CASE
      WHEN SUM(declined) <= 3 THEN 'tarjeta activa'
      ELSE 'tarjeta desactivada'
   END AS estado tarjeta
FROM transacciones_tarjeta
WHERE row_transaction <= 3
GROUP BY numero_tarjeta
HAVING COUNT(numero_tarjeta) = 3);

SELECT card_id, status FROM card_status;  -- verificando resultados previos
```

Ejercicio 1

¿Cuántas tarjetas están activas?

```
SELECT COUNT(*) AS 'tarjetas activas'
FROM card_status
WHERE estado_tarjeta ='tarjeta activa';
```

tarjetas activas
9

========== NIVEL 3 ==========

Crea una tabla con la que podamos unir los datos del nuevo archivo products.csv con la base de datos creada, teniendo en cuenta que desde transaction tienes product_ids. Genera la siguiente consulta:

Ejercicio 1

Necesitamos conocer el número de veces que se ha vendido cada producto.

Después de crear tabla intermedia 'transactions_products' con sus respectivas vinculaciones foráneas e indexaciones, hacemos la importación de datos.

```
CREATE TABLE transactions_products (
    id VARCHAR (255) DEFAULT NULL,
    product_ids INT DEFAULT NULL,
    KEY idx_id1 (id),
    KEY idx_id2 (product_ids),
    CONSTRAINT product_ids FOREIGN KEY (product_ids) REFERENCES products (id),
    CONSTRAINT id FOREIGN KEY (id) REFERENCES transactions (id)
);
SELECT * FROM transactions_products;       -- visualizamos la nueva tabla
SHOW CREATE TABLE transactions_products; -- verificamos codigo
```

Fracción del diagrama

consulta usando **COUNT** para el conteo por **product_ids**

```
SELECT DISTINCT product_name,
        count(product_ids) AS cantidad_ventas -- conteo por product_ids
FROM products p
INNER JOIN transactions_products tp ON p.id = tp.product_ids
GROUP BY product_name
ORDER BY cantidad_ventas DESC;
```

consulta usando **COUNT** para el conteo por **id**

```
SELECT DISTINCT product_name,
        count(tp.id) AS cantidad_ventas -- conteo por id
FROM products p
INNER JOIN transactions_products tp ON p.id = tp.product_ids
GROUP BY product_name
ORDER BY cantidad_ventas DESC;
```

En la comprobación de las 2 opciones, ya sea por **id** y **product_ids,** obtenemos los mismos resultados, mostramos la tabla:

product_name	cantidad_ventas
Direwolf Stannis	106
skywalker ewok	100
riverlands north	68
Winterfell	68
Direwolf riverlands the	66
Tarly Stark	65
duel	65
Tully	62
jinn Winterfell	61
skywalker ewok sith	61
palpatine chewbacca	60
kingsblood Littlefinger...	58
Winterfell Lannister	57
duel tourney	57

NOTA: surge ciertas dudas que bajo el nombre de 1 producto se encuentre varias versiones de producto según otras características

En esta consulta intentamos cubrir la sospecha de que existan bajo un mismo nombre varios tipos de productos ya sea por tamaño color, por alguna razón lo evidencia en warehouse, de esta forma para asegurarnos, usamos **DISTINCT** y lo agrupamos por **id**

```
SELECT DISTINCT p.id,
        count(product_ids) as conteo_ventas
FROM transactions_products as tp
INNER JOIN products as p ON tP.product_ids = p.id
GROUP BY p.id
```

id	conteo_ventas
1	61
2	65
3	51
5	49
7	54
11	48
13	60
17	61
19	49
23	68
29	49
31	47

```
ORDER BY p.id ASC;
```

Ahora solo agregamos el nombre para conocer a pesar que se repita en algunas filas, pero se diferencia por el id según su tipología

```
SELECT DISTINCT p.id,
        p.product_name,
        count(tp.id) AS cantidad_ventas
FROM products p
INNER JOIN transactions_products tp ON p.id = tp.product_ids
GROUP BY id
ORDER BY id ASC;
```

id	product_name	cantidad_ventas
1	Direwolf Stannis	61
2	Tarly Stark	65
3	duel tourney Lannister	51
5	skywalker ewok	49
7	north of Casterly	54
11	Karstark Dorne	48
13	palpatine chewbacca	60
17	skywalker ewok sith	61
19	dooku solo	49
23	riverlands north	68
29	Tully maester Tarly	49
31	Lannister	47
37	Direwolf Littlefinger	51

Fracción de la tabla

Nota: Diagrama Final