>>>>>>>>>>>>>>>>>>>> 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,
        company_name VARCHAR(255),
        phone VARCHAR(15),
        email VARCHAR(150),
        country VARCHAR(150),
        website VARCHAR(150));

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

create table credit_card (
        id VARCHAR(20) PRIMARY KEY,
        user_id VARCHAR(20),
        iban VARCHAR(255),
        pan VARCHAR(45),
        pin CHAR(4),
        cvv CHAR(3),
        track1 VARCHAR(255),
        track2 VARCHAR(255),
        expiring_date varchar(255));

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

CREATE TABLE products (
        id INT PRIMARY KEY,
        product_name VARCHAR(100),
        price VARCHAR(10),
        colour VARCHAR(100),
        weight VARCHAR(100),
        warehouse_id VARCHAR(100));
-- ------------------------------------------------------

CREATE TABLE transactions (
        id VARCHAR(255) PRIMARY KEY,
        card_id VARCHAR(20),
        bussiness_id VARCHAR(150),
        timestamp varchar(150),
        amount DECIMAL(10,2),
        declined TINYINT(1),
        product_ids VARCHAR(20),
        user_id INT,
        lat VARCHAR(50),
        longitude VARCHAR(50));

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

CREATE TABLE users (
        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));
```

Debemos comentar que, analizando los campos o columnas de esta última tabla users, es el resultado de la unificación de los archivos **"users_ca, users_usa, users_uk"** la opción clara fué efectuar una importación unificada a una sola tabla **'users'**.
De esta forma simplifica el modelo y facilita la gestión de datos, bajando el tiempo e incrementando la eficiencia, 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, mas no que una transacción corresponda a varios usuarios.

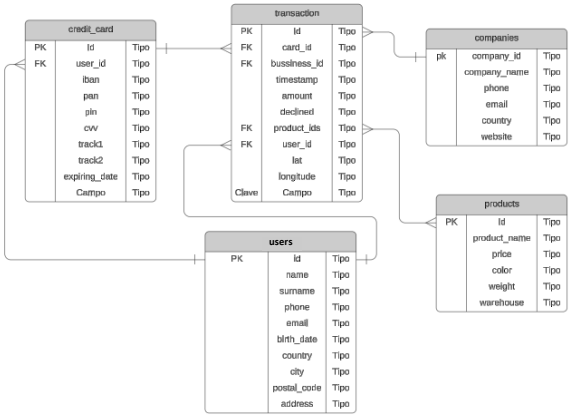

Diagrama conceptual inicial – todas las relaciones

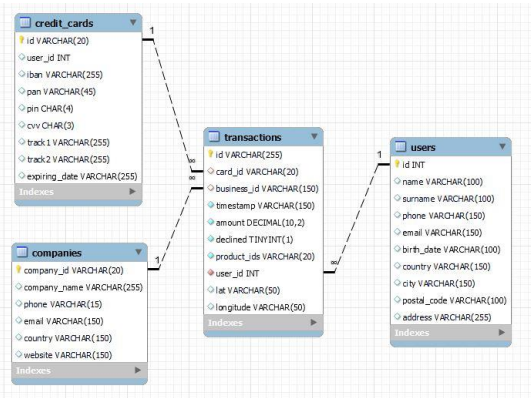

Diagrama de relaciones de FK en las tablas a usarse inicialmente

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

```
CREATE INDEX idx_company
        ON transaction(business_id);

CREATE INDEX idx_credit_card
        ON transaction(card_id);

CREATE INDEX idx_users
        ON transaction(user_id);

CREATE INDEX idx_products ON transaction(product_ids);

ALTER TABLE company
        ADD FOREIGN KEY (id) REFERENCES transactions(business_id);

ALTER TABLE credit_card
        ADD FOREIGN KEY (id) REFERENCES transactions(card_id);

ALTER TABLE users
        ADD FOREIGN KEY (id) REFERENCES transactions(user_id);
```

Además, ya que en los próximos ejercicios vamos a requerir crear una la tabla intermedia 'transactions_products', en este apartado explicamos el procedimiento y sus respectivas vinculaciones foráneas e indexaciones, procedemos con la importación de datos.

```
CREATE TABLE transactions_products (
    id VARCHAR (255),
    product_ids INT,

ALTER TABLE transactions_products (
```

```
    ADD CONSTRAINT product_ids FOREIGN KEY (product_ids) REFERENCES products (id),
    ADD CONSTRAINT id FOREIGN KEY (id) REFERENCES transactions (id));

SELECT *
FROM transactions_products;       -- visualizamos la nueva tabla

SHOW CREATE TABLE transactions_products; -- verificamos codigo
```

Al importar los datos desde tabla de 'transactions' encontramos inconvenientes en la importación con el archivo 'cant_product.csv', ya que la data (product_ids) en la tabla origen se encontraba varios 'ids' separados por comas en la misma fila de cada producto, lo evidencia la imagen siguiente:

id	card_id	business_id	timestamp	amount	declined	product_ids	user_id	lat	longitude
108B1D1D-5E	CcU-2938	b-2222	##########	293.57	0	59	275	##########	##########
7DC26247-20	CcU-2945	b-2226	##########	312.5	0	71, 41	275	##########	##########
72997E96-DC	CcU-2952	b-2230	##########	239.87	0	97, 41, 3	275	##########	##########
AB069F53-96	CcU-2959	b-2234	##########	60.99	0	11, 13, 61, 29	275	##########	##########
2F3B6AB6-14	CcU-2966	b-2238	##########	33.81	0	47, 37, 11, 1	275	##########	##########
5B0EEF86-B8	CcU-2973	b-2242	##########	42.82	0	23, 19, 71	275	##########	##########
2B928E1C-EC	CcU-2980	b-2246	##########	383.73	0	59, 13, 23	275	##########	##########

Fracción de la tabla

Para ello, se acudió a una herramienta externa, se estableció un pequeño codigo en Python, para poder separar cada 'product_ids' agrupados en cada fila con su nombre de producto (id) ubicados en la tabla 'transactions'

Codigo:

```
import csv

with open('cant_product.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=';')
    for row in csv_reader:
        for pro in row[1].split(','):
            print(row[0].strip(),';', pro.strip())
```

Abrir el archivo CSV: with open('cant_product.csv') abre el archivo CSV llamado 'cant_product.csv' en modo lectura y lo asigna a la variable csv_file. La declaración with asegura el manejo adecuado del archivo, cerrándolo automáticamente incluso si hay errores.

Crear un objeto lector CSV: csv_reader = csv.reader(csv_file, delimiter=';') este objeto se utilizará para iterar sobre las filas del archivo CSV, el delimiter especifica que los datos del archivo CSV están separados por punto y coma (;) en lugar de la coma predeterminada (,).

Recorriendo filas: for row in csv_reader:itera sobre cada fila en el archivo CSV. La variable rowcontendrá una lista que contiene los valores de cada celda de la fila actual.

Procesamiento de cada producto seguido: for pro in row[1].split(',') se encarga a iterar sobre una lista de productos dentro del segundo elemento (índice 1) de la fila actual. Esto supone que la segunda columna contiene nombres de productos separados por comas. El split(',') método que divide la cadena según comas, creando una lista de nombres de productos individuales.

Impresión de salida formateada: print(row[0].strip(), ';', pro.strip()) logra imprimir lo siguiente:

- ***row[0].strip()*** El primer elemento (índice 0) de la fila, con los espacios en blanco iniciales y finales eliminados mediante el método strip(), esto represente una categoría de producto.
- ***';'*** Un punto y coma para separar la categoría (cada product_ids) y el nombre del producto.
- ***pro.strip()*** El nombre del producto actual del bucle interno, con los espacios en blanco iniciales y finales eliminados usando strip().

Este código procesa/importa desde un archivo CSV donde la segunda columna contiene product_ids separados por comas. Recorre cada fila, extrae la categoría (c/product_ids) y los nombres de los productos e imprime cada nombre de producto junto con su categoría (c/product_ids), separados por un punto y coma.

id	product_ids
0466A42E-47CF-8D24-FD01-C0B689713128	47
0466A42E-47CF-8D24-FD01-C0B689713128	97
0466A42E-47CF-8D24-FD01-C0B689713128	43
063FBA79-99EC-66FB-29F7-25726D1764A5	47
063FBA79-99EC-66FB-29F7-25726D1764A5	67
063FBA79-99EC-66FB-29F7-25726D1764A5	31
063FBA79-99EC-66FB-29F7-25726D1764A5	5
0668296C-CDB9-A883-76BC-2E4C44F8C8AE	89
0668296C-CDB9-A883-76BC-2E4C44F8C8AE	83
0668296C-CDB9-A883-76BC-2E4C44F8C8AE	79
0A476ED9-0C13-1962-F87B-D3563924B539	29
0A476ED9-0C13-1962-F87B-D3563924B539	41
0A476ED9-0C13-1962-F87B-D3563924B539	11
1017AA59-3D5F-7A4C-1992-D151A8D1FA0A	37
1017AA59-3D5F-7A4C-1992-D151A8D1FA0A	13
108B1D1D-5B23-A76C-55EF-C568E49A05DD	59
122DC333-E19F-D629-DCD8-9C54CF1EBB9A	1
122DC333-E19F-D629-DCD8-9C54CF1EBB9A	67
122DC333-E19F-D629-DCD8-9C54CF1EBB9A	19
135267BA-2E7D-957C-C42C-6450A2B3ED54	11
135267BA-2E7D-957C-C42C-6450A2B3ED54	71

Ejercicio 1

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

```
SELECT u.*,
        (SELECT count(t.id)
        FROM transactions t
        WHERE u.id = t.user_id) AS cantidad_transacciones
FROM users u
GROUP BY id
HAVING cantidad_transacciones > 30;
```

id	name	surname	phone	email	birth_date	country	city	postal_code	address	cantidad_transacciones
92	Lynn	Riddle	1-387-885-4057	vitae.aliquet@outlook.edu	Sep 21, 1984	United States	Bozeman	61871	P.O. Box 712, 7907 Est St.	39
267	Ocean	Nelson	079-481-2745	aenean@yahoo.com	Dec 26, 1991	Canada	Charlottetown	85X 3P4	Ap #732-8357 Pede, Rd.	52
272	Hedwig	Gilbert	064-204-8788	sem.eget@icloud.edu	Apr 16, 1991	Canada	Tuktoyaktuk	Q4C 3G7	P.O. Box 496, 5145 Sapien Road	76
275	Kenyon	Hartman	082-871-7248	convallis.ante.lectus@yahoo.com	Aug 3, 1982	Canada	Richmond	R8H 2K2	8564 Facilisi. St.	48

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 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, la misma para el objetivo planteado

CREATE TABLE card_status (
        card_id VARCHAR(15),
        status VARCHAR(50));

-- Se ingresa los datos de tablas establecidas condicionándolos a 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) <= 2 THEN 'tarjeta activa'
            ELSE 'tarjeta desactivada'
        END AS estado_tarjeta
FROM transacciones_tarjeta
WHERE row_transaction <= 3
GROUP BY numero_tarjeta;
```

```
SELECT card_id as numero_tarjeta,
        status AS estado_tarjeta
FROM card_status;
```

PARTITION BY la función separa cada tarjeta individualmente en relacion a sus índices, asi recorremos todas las tajetas.

ORDER BY timestamp DESC ordenamos las transacciones en cada grupo desde la mas actual a la más antigua, filtrándolas a posterior conforme el número de veces reciente.

'WHERE row_transaction <= 3' para establecer si una tarjeta esta activa o inactiva, debemos verificar si existen 3 transacciones actuales/ultimas/recientes por tarjeta, de esta forma con la introducción de datos a sta nueva tabla, logramos 275 datos.

numero_tarjeta	estado_tarjeta
CcU-2938	tarjeta activa
CcU-2945	tarjeta activa
CcU-2952	tarjeta activa
CcU-2959	tarjeta activa
CcU-2966	tarjeta activa
CcU-2973	tarjeta activa
CcU-2980	tarjeta activa
CcU-2987	tarjeta activa
CcU-2994	tarjeta activa
CcU-3001	tarjeta activa
CcU-3008	tarjeta activa
CcU-3015	tarjeta activa
CcU-3022	tarjeta activa
CcU-3029	tarjeta activa
CcU-3036	tarjeta activa
CcU-3043	tarjeta activa
CcU-3050	tarjeta activa
CcU-3057	tarjeta activa
CcU-3064	tarjeta activa
CcU-3071	tarjeta activa
CcU-3078	tarjeta activa
CcU-3085	tarjeta activa

Ejercicio 1

¿Cuántas tarjetas están activas?

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

tarjetas activas
275

========== 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.

```
CREATE TABLE transactions_products_1 (
  id VARCHAR (255),
  product_ids INT);
```

-- cargamos e introducimos la data desde el archivo csv modificado (explicado al inicio)

```
LOAD DATA LOCAL INFILE
'D:\SH ESPAÑA\CURSOS - CAPACITACIÓN\IT ACADEMY\DATA ANALYTICS\SPRINT
4\DESCARGADOS\transactions_products.csv'
INTO TABLE transactions_products_2
FIELDS TERMINATED BY  ","
ENCLOSED BY '"'
```

```
LINES TERMINATED BY ";"
IGNORE 1 ROWS;
```

id	product_ids
108B1D1D-5B23-A76C-55EF-C568E49A05DD	59
7DC26247-20EC-53FE-E555-B6C2E55CA5D5	71
7DC26247-20EC-53FE-E555-B6C2E55CA5D5	41
72997E96-DC2C-A4D7-7C24-66C302F8AE5A	97
72997E96-DC2C-A4D7-7C24-66C302F8AE5A	41
72997E96-DC2C-A4D7-7C24-66C302F8AE5A	3
AB069F53-965E-A2A8-CE06-CA8C4FD92501	11
AB069F53-965E-A2A8-CE06-CA8C4FD92501	13
AB069F53-965E-A2A8-CE06-CA8C4FD92501	61
AB069F53-965E-A2A8-CE06-CA8C4FD92501	29
2F3B6AB6-147D-EB0B-FE8D-9A4E2EA9DBD5	47
2F3B6AB6-147D-EB0B-FE8D-9A4E2EA9DBD5	37
2F3B6AB6-147D-EB0B-FE8D-9A4E2EA9DBD5	11
2F3B6AB6-147D-EB0B-FE8D-9A4E2EA9DBD5	1

De esta forma, el diagrama o modelo general de todo el sistema de tablas que se logra en el proceso de los ejercicios con las relaciones foráneas resultantes conforme la necesidad y desarrollo previsto en el sinnúmero de consultas, queda de la siguiente forma

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 un producto, se encuentre varias versiones de éste artículo bajo otras características

En esta consulta intentaremos descubrir la sospecha de la existencia posible de un producto a varios tipos de productos ya sea por tamaño color, por alguna razón lo evidencia en warehouse, de esta forma nos aseguramos, 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
ORDER BY p.id ASC;
```

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

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