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

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

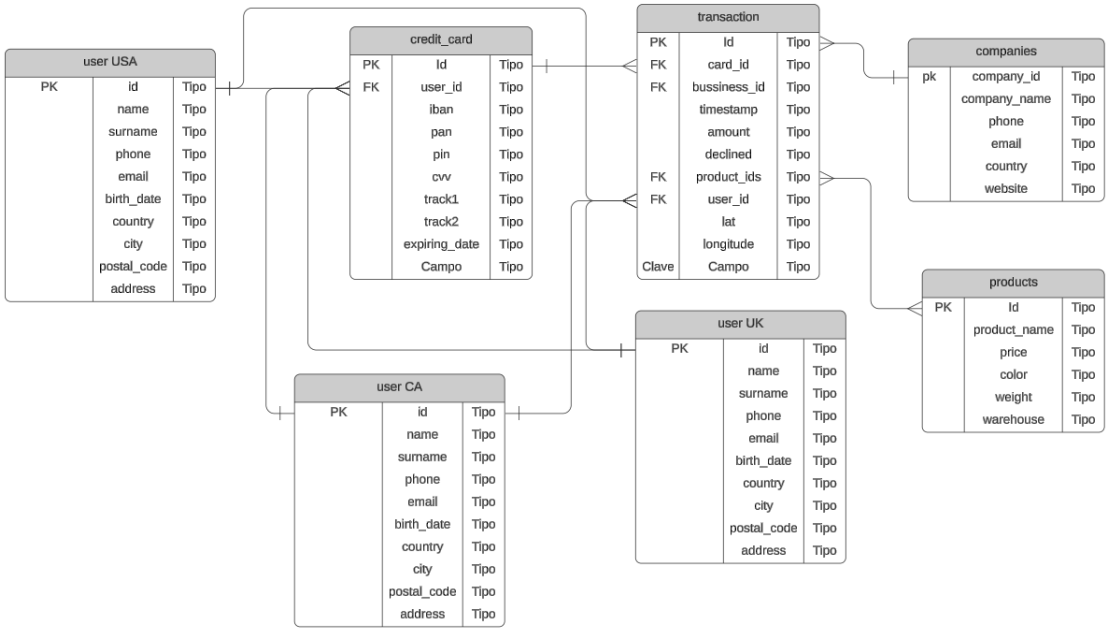
========== NIVEL 1 ==========

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 companies (
	company_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 int,
	iban varchar(255) null,
	pan char(45) null,
	pin char(4) null,
	cvv char(3) null,
	track1 varchar(255),
	track2 varchar(255),
	expiring_date varchar(255) null
	);

	alter table credit_card
	ADD FOREIGN KEY (user_id) references users(id);
	-- -------------------------------------------------------

	create table products (
	id int auto_increment PRIMARY KEY not null,
	product_name varchar(100) not null,
	price varchar(10) null,
	color varchar(100) null,
	weight double null,
	warehouse_id varchar(100) not null
	);
	-- -------------------------------------------------------

	create table users (
	id int auto_increment PRIMARY KEY not null,
	name varchar(100) null,
	surname varchar(100) null,
	phone varchar(150) null,
	email varchar(150) null,
	birth_date varchar(100) null,
	country varchar(150) null,
	city varchar (150) null,
	postal_code varchar(100) null,
	address varchar(255) null
	);
	-- -------------------------------------------------------
```

```
CREATE TABLE transaction (
id varchar(255) PRIMARY KEY not null,
card_id varchar(20) null,
bussiness_id varchar(150) null,
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 not null,
lat varchar(50) null,
longitude varchar(50) null);

alter table transaction
ADD FOREIGN KEY (card_id) references credit_card(id),
ADD FOREIGN KEY (bussiness_id) references companies(company_id),
ADD FOREIGN KEY (product_ids) references products(id),
ADD FOREIGN KEY (user_id) references users(id);
```

Indagando los datos facilitados, se establece una primera idea del diagrama de relación de entidades de las siguientes tablas:



Analizando los campos o columnas de las Tablas users **“users_ca, users_usa, users_uk”** evidenciando sus símiles, 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.

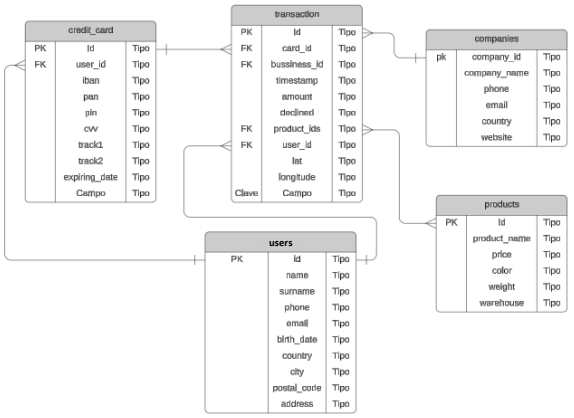


Diagrama conceptual inicial – todas las relaciones

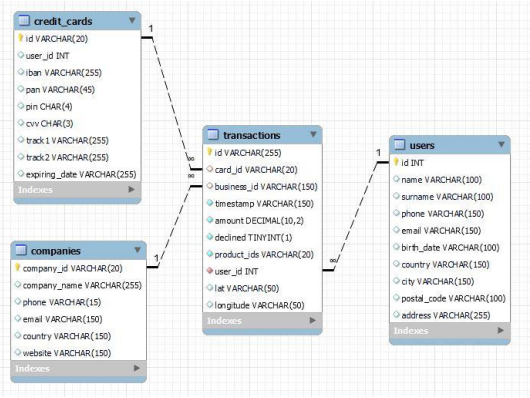


Diagrama de relaciones de FK en las tablas a usarse inicialmente

Ejercicio 1

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

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

Opción consulta sencilla solo con la tabla transactions

```
SELECT user_id, COUNT(*) AS total_transactions
FROM transactions
GROUP BY user_id
HAVING COUNT(*) > 30
ORDER BY total_transactions asc;
```

Opción consulta sencilla solo con la tabla transactions y montos totales de 2 decimales 'round'

```
SELECT user_id, COUNT(*) AS total_transactions, ROUND(SUM(amount), 2) as monto_total
FROM transactions
GROUP BY user_id
HAVING COUNT(*) > 30 AND sum(amount)
ORDER BY total_transactions asc;
```

Opción consulta sencilla con Union (2 tablas)

```
SELECT 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;
```

Opción consulta sencilla con Union (2 tablas) concatenando los nombres

```
select 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;
```

Opción con subconsulta dentro de INNER JOIN

```
SELECT u.name, u.surname, 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;
```

Opción con subconsulta con INNER JOIN y con alias 'nombres completos'

```
select concat(NAME, ' ', SURNAME) AS "nombres completos", total_transacciones
from users as u
inner join (
select user_id, count(*) as total_transacciones
from transactions
```

```
group by user_id
having count(*) >30
) t ON u.id = t.user_id;
```

Opción con subconsulta con INNER JOIN, con alias 'nombres completos' y el monto total de giro en 2 decimales 'format'

```
select 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;
```

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

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, AVG(t.amount) AS promedio_suma_transacciones, cc.iban  -- funciona
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:

Creación de nueva tabla

```
CREATE TABLE card_status (
  id INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
  card_id VARCHAR(20) NOT NULL,
  status VARCHAR(10) NOT NULL DEFAULT 'active',
  KEY idx_id2 (id),
  CONSTRAINT fk_card_reference FOREIGN KEY (card_id) REFERENCES credit_cards (id)  -- Renamed constraint name
);
```

Ejercicio 1

¿Cuántas tarjetas están activas?

```
select *
from credit_cards
where expiring_date > DATE_FORMAT(CURRENT_DATE,'%m/%d/%Y');

SELECT *
FROM credit_cards
WHERE expiring_date > DATE_FORMAT(CURRENT_DATE, '%m/%d/%4');

SELECT *
FROM credit_cards
WHERE expiring_date > (CURRENT_DATE, '%m/%d/%Y');
```

========== **NIVEL 3** ==========

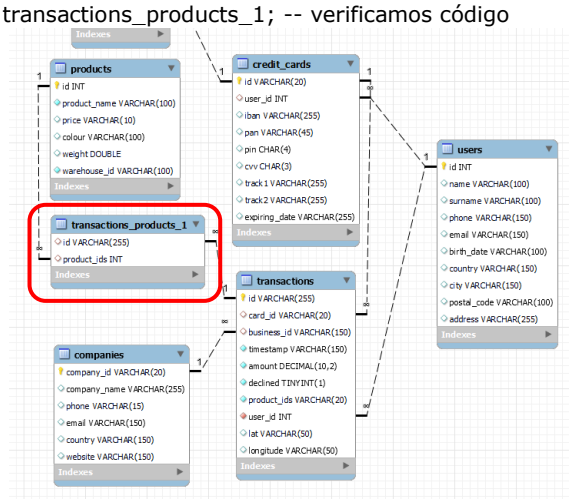
Crea una tabla con la que podamos unir los datos del nuevo archivo products.csv con la base de datos creada, teniendo en cuenta que desde transaction tienes product_ids. Genera la siguiente consulta:

Ejercicio 1

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

Después de crear tabla intermedia 'transactions_products_1' con sus respectivas vinculaciones foráneas e indexaciones, hacemos la importación de datos
procedemos a INDEXAR

```
CREATE TABLE transactions_products_2 (
  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_1; -- verificamos código
```



consulta opción 1 usando la tabla products y la tabla intermedia transactions_products

```
select DISTINCT product_name, count(product_ids) as cantidad_ventas
from products p
inner join transactions_products tp on p.id = tp.product_ids
group by product_name
order by cantidad_ventas desc;
```

consulta opción 2 usando la tabla products y la tabla intermedia transactions_products*/

```
select DISTINCT product_name, count(tp.id) as cantidad_ventas
from products p
inner join transactions_products tp on p.id = tp.product_ids
group by product_name
order by cantidad_ventas desc;
```

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 producto 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
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

Nota: Diagrama Final