Tasca S4.01. Creació de Base de Dades

Hecho por: Yatmelis Freites

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:

```sql
CREATE DATABASE GeekStore;
USE GeekStore;

CREATE TABLE companies (
  company_id VARCHAR(100) PRIMARY KEY,
  company_name VARCHAR(50),
  phone VARCHAR(20),
  email VARCHAR(50),
  country VARCHAR(50),
  website VARCHAR(100)
);

CREATE TABLE credit_cards (
  id VARCHAR(20) PRIMARY KEY,
  user_id INT,
  iban VARCHAR(34) NOT NULL UNIQUE,
  pan VARCHAR(25) NOT NULL,
  pin INT(4) NOT NULL,
  cvv INT(3) NOT NULL,
  track1 VARCHAR(100) ,
  track2 VARCHAR(100),
  expiring_date VARCHAR(10) NOT NULL
);
```

Primero se declara **CREATE DATABASE** para generar mi base de datos, indicando el nombre a continuación y posteriormente **USE** para indicar que todas las acciones a continuación se realizarán para modificar esta base de datos en específico.

Luego con **CREATE TABLE** escribo el nombre de la tabla a crear, en primer lugar **companies** y entre paréntesis se insertan los campos que se encuentran en el archivo .csv, luego de haberlos analizado para determinar el tipo de dato que contiene, y al crear el campo **company_id,** se escribe a continuación **PRIMARY KEY para indicar que es la clave primaria de esta tabla.**

Para la tabla **credit_cards** se escribe la declaración **CREATE TABLE** y posteriormente se introducen los nombres del archivo .csv y a continuación se indica que **id** es la **PRIMARY KEY,** y que el campo **iban** es **NOT NULL** para evitar campos vacíos en caso de que existan y se indica que esta es una **clave UNIQUE**, es decir que solo existe un iban para cada tarjeta de crédito, este dato no se repite, y los campos **pan, pin, cvv** son igualmente **NOT NULL**, al igual que el campo **expiring_date** porque toda tarjeta posee una fecha de vencimiento

```sql
CREATE TABLE products(
    id VARCHAR(30) PRIMARY KEY ,
    product_name VARCHAR(20),
    price VARCHAR(20),
    colour VARCHAR(7),
    weight FLOAT,
    warehouse_id VARCHAR(7)
);

CREATE TABLE users (
id INT PRIMARY KEY,
name VARCHAR(30),
surname VARCHAR(30),
phone VARCHAR(20),
email VARCHAR(50),
birth_date VARCHAR(25) NOT NULL,
country VARCHAR(50),
city VARCHAR(50),
postal_code VARCHAR (15),
address VARCHAR(50)
);
```

Para la tabla **products** se escribe la declaración **CREATE TABLE** y posteriormente se introducen los nombres del archivo .csv y a continuación se indica que **id** es la **PRIMARY KEY,** se completan los tipos de datos para cada campo (**VARCHAR**) según correspondan en el archivo, en este caso solo **weight** posee un tipo de dato **FLOAT** porque el peso de los productos puede ser de tipo decimal.

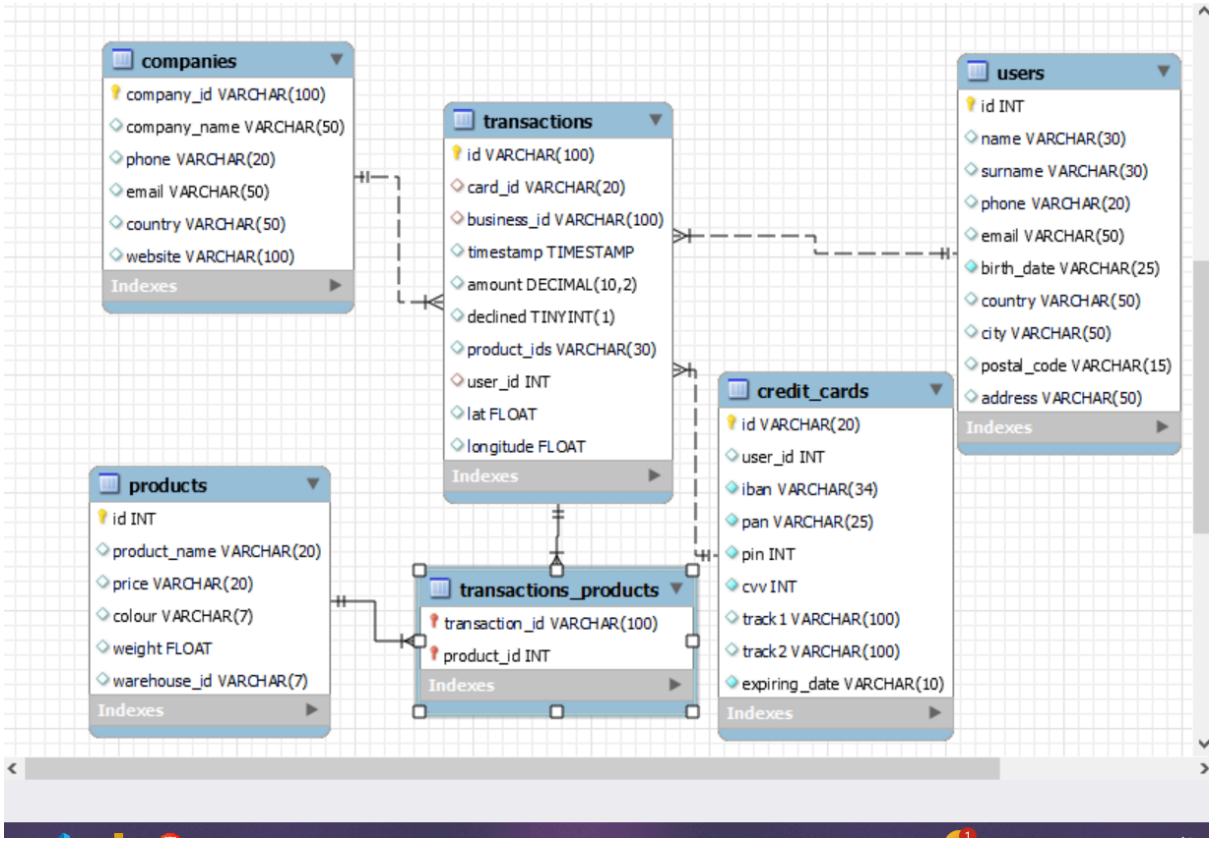
Para la tabla **users** se escribe la declaración **CREATE TABLE** y posteriormente se introducen los nombres del archivo .csv y a continuación se indica que **id** es la **PRIMARY KEY,** y que el campo **birth_date** es **NOT NULL**, porque toda persona tiene una una fecha de nacimiento, y todos los campos a excepción de **id** son de tipo **VARCHAR**.

```sql
CREATE TABLE transactions (
    id VARCHAR(100) PRIMARY KEY,
    card_id VARCHAR(20),
    business_id VARCHAR(100),
    timestamp TIMESTAMP,
    amount DECIMAL(10, 2),
    declined TINYINT(1),
    product_ids VARCHAR(30),
    user_id INT,
    lat FLOAT,
    longitude FLOAT,
    FOREIGN KEY (card_id) REFERENCES credit_cards(id),
    FOREIGN KEY (business_id) REFERENCES companies(company_id),
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Por último, de la tabla **transactions** se escribe la declaración **CREATE TABLE** y posteriormente se introducen los nombres del archivo .csv y a continuación se indica que **id** es la **PRIMARY KEY de tipo VARCHAR,** y que los campo **card_id y business_id** son de tipo **VARCHAR.** El campo **timestamp** es de tipo **TIMESTAMP, amount** de tipo **DECIMAL, indicando entre paréntesis el numero de dígitos aceptados, y luego de la coma el número de decimales** para que sea más fácil de leer. Para el campo **declined** se indica que es de tipo **TINYINT (**ya que con 1 on 0 se indica si la transacción fue aprobada o rechazada**), product_ids** es de tipo **VARCHAR, user_id** de tipo **INT** ya que se indica con números y **lat y long** son **FLOATS** porque son de tipo decimal.

Se ha identificado que esta es la tabla de hechos y por lo tanto se agregan todas las **FOREIGN KEYS,** referenciando a sus respectivos campos en las tablas de dimensiones.

Cabe resaltar que se complicaba la importación de los datos de **transactions** de manera correcta dado que los datos de la columna **product_ids** eran diferentes a los de la tabla **product(id), por lo tanto no existe como clave foránea en esta tabla**



Para generar el modelo correcto, se siguieron los siguientes pasos:

```
ALTER TABLE products
MODIFY COLUMN id INT;
```

- Se alteró la tabla products para cambiar el tipo de dato de VARCHAR a INT

```
50  CREATE TABLE transactions_products (
51      transaction_id VARCHAR(100),
52      product_id VARCHAR(30),
53      PRIMARY KEY (transaction_id, product_id)
54  );
55
56  ALTER TABLE transactions_products
57  MODIFY COLUMN  product_id INT;
```

- Se creó una nueva tabla con solo las primary keys, para generar un puente entre las tablas transactions y products.
- Posteriormente se alteró también en el campo de product_id para que coincidiera con la columna id de la tabla product.
- Luego se agregaron los datos haciendo uso de una query.

```sql
INSERT INTO transactions_products (transaction_id, product_id)
SELECT
        t.id,
        SUBSTRING_INDEX(SUBSTRING_INDEX(t.product_ids, ',', n.n), ',', -1) as product_id
FROM (SELECT 1 as n UNION ALL SELECT 2 UNION ALL SELECT 3 UNION ALL SELECT 4
    ) n
JOIN transactions t
        ON n.n <= 1 + (LENGTH(t.product_ids) - LENGTH(REPLACE(t.product_ids, ',', '')))
ORDER BY t.id, n.n;
```

- Esta consulta SELECCIONA el id de transaction y con SUBSTRING_INDEX genera una secuencia de números del 1 al 4 y luego une cada número con las filas de la tabla transactions.Se realiza un join de estos números generados y para cada fila en la tabla transactions, extrae ID de producto individuales de la columna product_ids separada por comas y los devuelve junto con el ID de transacción. Finalmente, ordena los resultados basándose en el ID de transacción y la secuencia de números.

```sql
--
DELETE FROM transactions_products
WHERE product_id NOT IN (SELECT id FROM products);

ALTER TABLE transactions_products
ADD CONSTRAINT fk_transaction_id
FOREIGN KEY (transaction_id) REFERENCES transactions(id);

ALTER TABLE transactions_products
ADD CONSTRAINT fk_product_id
FOREIGN KEY (product_id) REFERENCES products(id);

--
```

- Por último se eliminan los records de transaction_products en las que product_id no se encuentra en la selección de los ids de products;
- Se altera la nueva tabla para agregar constraints para las foreign keys y poder realizar la conexión con las dos tablas.

Nivell 1

Exercici 1

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

```
SELECT u.id, u.name, u.surname, u.email
FROM geekstore.users u
WHERE u.id IN (
    SELECT t.user_id
    FROM geekstore.transactions t
    GROUP BY t.user_id
    HAVING COUNT(t.user_id) > 30
);
```

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

id	name	surname	email
92	Lynn	Riddle	vitae.aliquet@outlook.edu
267	Ocean	Nelson	aenean@yahoo.com
272	Hedwig	Gilbert	sem.eget@icloud.edu
275	Kenyon	Hartman	convallis.ante.lectus@yahoo.com
NULL	NULL	NULL	NULL

Se escribe **SELECT del id, name, surname e email** de la tabla users y se usa la letra **u** como alias y se realiza la subconsulta utilizando un **WHERE** para filtrar id.

Dentro de la subconsulta, se **SELECCIONA user_id** de la tabla **transactions**, usando la letra **t** como alias y se agrupa igualmente por **user_id,** usando **HAVING COUNT** (t.user_id) **y se realiza la comparación con mayor a (>)** 30 para indicar que se debe filtrar según este criterio del número de transacciones.

Exercici 2

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

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

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

mediana	iban
203.715000	PT87806228135092429456346

Se escribe **SELECT AVG** de **transactions.amount** (donde se usa la letra **t** como alias) y se realiza un **JOIN** de **companies** (donde se usa la letra **c** como alias) **donde transaction.business id es igual a companies.company_id,** se realiza otro **JOIN** de la tabla **credit_cards (**donde se usa cc como alias) **donde credit_cards.id es igual a transactions.card_id** utilizando un **WHERE** para filtrar el nombre de la compañía Donec Ltd, por último se agrupa por iban.

Nivell 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:

Exercici 1

Quantes targetes estan actives?

Nivell 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:

Exercici 1

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