Unión Europea
Fondo Social Europeo
Iniciativa de Empleo Juvenil
El FSE invierte en tu futuro

Tema 14-3

Caso práctico

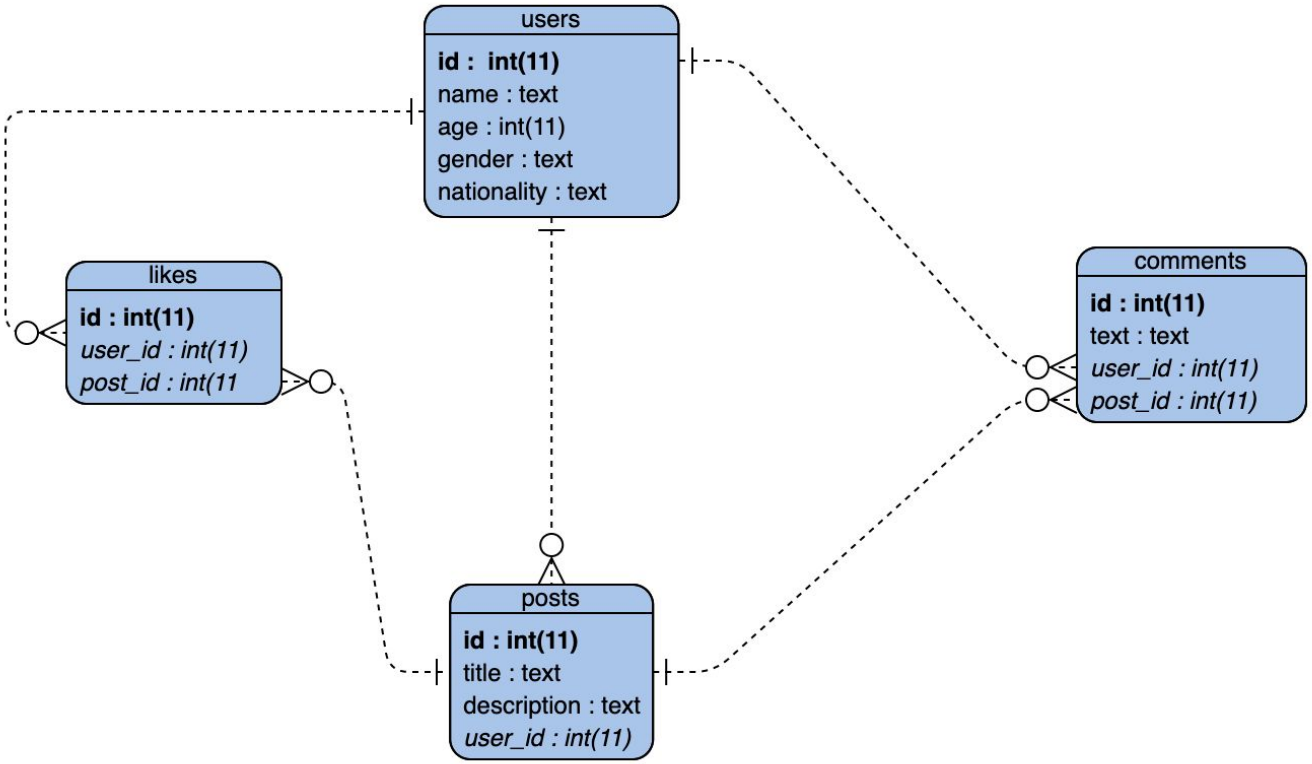
Ejemplo base de datos red social

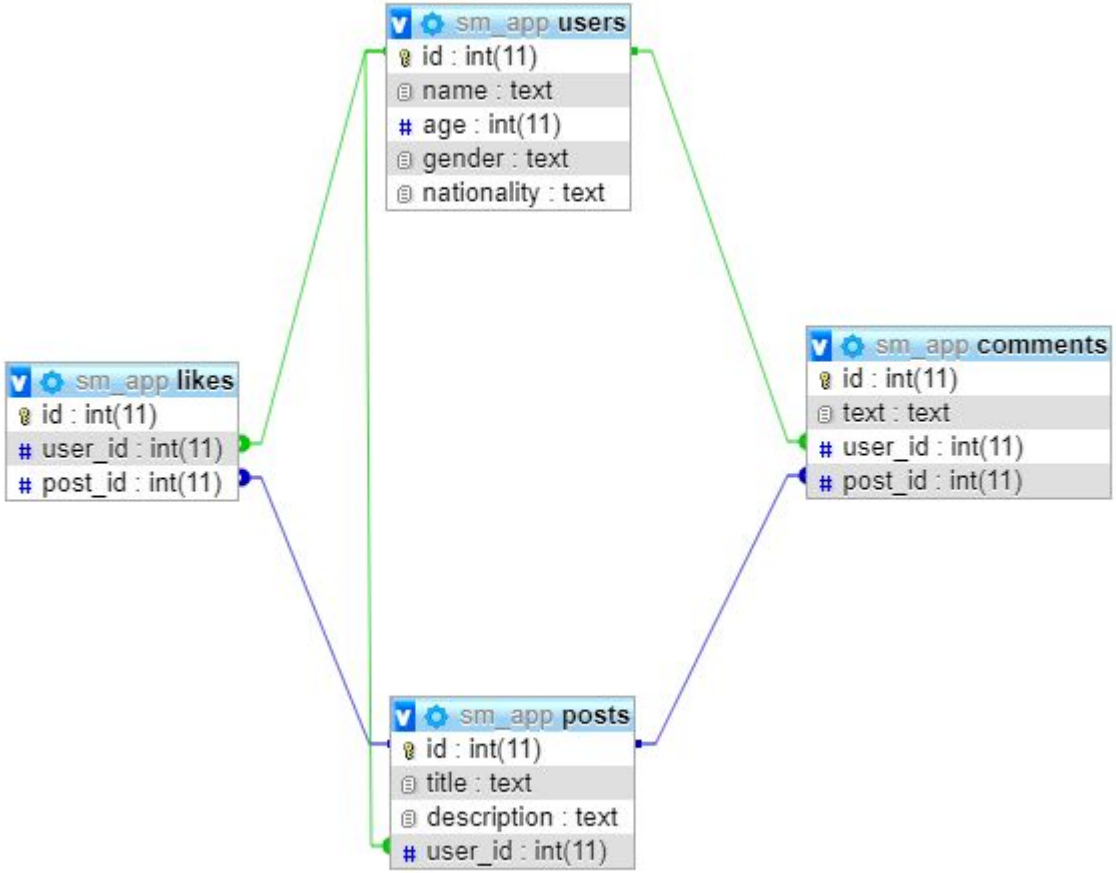
Vamos a crear y trabajar una pequeña base de datos para una app de social media. La base de datos tendrá 4 tablas:

1. usuarios: users
2. publicaciones: posts
3. comentarios: comments
4. me gusta: likes

Diagrama de base de datos

A la hora de diseñar una BBDD es conveniente hacer un diagrama con el esquema de las tablas y las dependencias (ERD: Entity Relationship Diagram)




sm_app users
id : int(11)
name : text
age : int(11)
gender : text
nationality : text
sm_app likes
id : int(11)
user_id : int(11)
post_id : int(11)
sm_app comments
id : int(11)
text : text
user_id : int(11)
post_id : int(11)
sm_app posts
id : int(11)
title : text
description : text
user_id : int(11)

Conexión a la base de datos

sm_app.py

```python
import sqlite3
from sqlite3 import Error

def create_connection(path):
    connection = None
    try:
        connection = sqlite3.connect(path)
        print("La conexión a la base de datos se ha ejecutado correctamente")
    except Error as e:
        print(f"Ha ocurrido el error '{e}'")

    return connection

connection = create_connection("sm_app.db")
connection.close()
```

Creación de las tablas

Primero definimos la función para capturar la excepción de un posible error

```
def execute_query(connection, query):
    cursor = connection.cursor()
    try:
        cursor.execute(query)
        connection.commit()
        print("Consulta ejecutada con éxito")
    except Error as e:
        print(f"Ha ocurrido el error '{e}'")
```

Para crear la primera tabla, asignamos la cadena de la instrucción SQL a una variable y la pasamos como argumento a la función que acabamos de crear

```
create_users_table = """
CREATE TABLE IF NOT EXISTS users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  age INTEGER,
  gender TEXT,
  nationality TEXT
);
"""
execute_query(connection, create_users_table)
```

Introducimos también la tabla posts

```
create_users_table = """
CREATE TABLE IF NOT EXISTS users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  age INTEGER,
  gender TEXT,
  nationality TEXT
);
"""

create_posts_table = """
CREATE TABLE IF NOT EXISTS posts(
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  title TEXT NOT NULL,
  description TEXT NOT NULL,
  user_id INTEGER NOT NULL,
  FOREIGN KEY (user_id) REFERENCES users (id)
);
"""

execute_query(connection, create_users_table)
execute_query(connection, create_posts_table)
```

Y finalmente, las de comments y likes, con sus correspondientes llamadas a execute_query

```
create_comments_table = """
CREATE TABLE IF NOT EXISTS comments (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  text TEXT NOT NULL,
  user_id INTEGER NOT NULL,
  post_id INTEGER NOT NULL,
  FOREIGN KEY (user_id) REFERENCES users (id) FOREIGN KEY (post_id) REFERENCES posts (id)
);
"""

create_likes_table = """
CREATE TABLE IF NOT EXISTS likes (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  user_id INTEGER NOT NULL,
  post_id integer NOT NULL,
  FOREIGN KEY (user_id) REFERENCES users (id) FOREIGN KEY (post_id) REFERENCES posts (id)
);
"""
```

Y finalmente, las de comments y likes, con sus correspondientes llamadas a execute_query

```
execute_query(connection, create_users_table)
execute_query(connection, create_posts_table)
execute_query(connection, create_comments_table)
execute_query(connection, create_likes_table)
```

Ahora escribimos el código SQL para insertar los registros de cada tabla. Empezamos por la de los usuarios:

```
create_users = """
INSERT INTO
  users (name, age, gender, nationality)
VALUES
  ('John', 25, 'hombre', 'USA'),
  ('Anna', 32, 'mujer', 'France'),
  ('Mary', 35, 'mujer', 'England'),
  ('Miguel', 40, 'hombre', 'España'),
  ('Isabel', 21, 'mujer', 'Canada');

"""
```

Código SQL para registros de los posts:

```
create_posts = """
INSERT INTO
  posts (title, description, user_id)
VALUES
  ("Happy", "I am feeling very happy today", 1),
  ("Hot Weather", "The weather is very hot today", 2),
  ("Help", "I need some help with my work", 2),
  ("Great News", "I am getting married", 1),
  ("Interesting Game", "It was a fantastic game of tennis", 5),
  ("Party", "Anyone up for a late-night party today?", 3);
"""
```

Código SQL para registros de los comentarios:

```
create_comments = """
INSERT INTO
  comments (text, user_id, post_id)
VALUES
  ('Count me in', 1, 6),
  ('What sort of help?', 5, 3),
  ('Congrats buddy', 2, 4),
  ('I was rooting for Nadal though', 4, 5),
  ('Help with your thesis?', 2, 3),
  ('Many congratulations', 5, 4);
"""
```

Código SQL para registros de los “me gusta” o likes:

```
create_likes = """
INSERT INTO
  likes (user_id, post_id)
VALUES
  (1, 6),
  (2, 3),
  (1, 5),
  (5, 4),
  (2, 4),
  (4, 2),
  (3, 6);
"""
```

A continuación, recurro a la función que había creado para hacer consultas para insertar los registros en cada tabla

```
# Crea los registros en las tablas que se habían creado
execute_query(connection, create_users)
execute_query(connection, create_posts)
execute_query(connection, create_comments)
execute_query(connection, create_likes)
```

Ahora, para poder seleccionar registros para mostrar resultados de consultas, defino una función que utiliza fetchall()

```
# Define una función para seleccionar registros con fetchall
def execute_read_query(connection, query):
    cursor = connection.cursor()
    result = None
    try:
        cursor.execute(query)
        result = cursor.fetchall()
        return result
    except Error as e:
        print(f"Ha ocurrido el error '{e}'")
```

De esta forma, podemos mostrar en pantalla todos los registros de la tabla usuarios con un for, tras haberlos seleccionado

```
# Selecciona todos los registros de la tabla users:
select_users = "SELECT * from users"
users = execute_read_query(connection, select_users)
# Muestra los registros
for user in users:
    print(user)
```

Y lo mismo con los posts

```
# Selecciona todos los registros de la tabla posts
select_posts = "SELECT * FROM posts"
posts = execute_read_query(connection, select_posts)
# Muestra los registros de la tabla posts
for post in posts:
    print(post)
```

Pero lo interesante es hacer consultas que aporten información más adaptada a las decisiones que se van a tomar. Borramos las consultas anteriores y creamos una con la instrucción JOIN, que nos permite relacionar dos tablas

```python
# Define una función para seleccionar registros con fetchall
def execute_read_query(connection, query):
    cursor = connection.cursor()
    result = None
    try:
        cursor.execute(query)
        result = cursor.fetchall()
        return result
    except Error as e:
        print(f"Ha ocurrido el error '{e}'")

select_users_posts = """
SELECT
  users.id,
  users.name,
  posts.description
FROM
  posts
  INNER JOIN users ON users.id = posts.user_id
"""

users_posts = execute_read_query(connection, select_users_posts)

for users_post in users_posts:
    print(users_post)
```

Podemos, incluso, relacionar tres tablas:

```
# Consulta una selección resultado de relacionar las tablas users, posts y comments
select_posts_comments_users = """
SELECT
  posts.description as post,
  text as comment,
  name
FROM
  posts
  INNER JOIN comments ON posts.id = comments.post_id
  INNER JOIN users ON users.id = comments.user_id
"""

posts_comments_users = execute_read_query(
    connection, select_posts_comments_users
)

for posts_comments_user in posts_comments_users:
    print(posts_comments_user)
```

Llegados a este punto, a veces va a venir bien mostrar el nombre de los campos de las columnas involucradas

```
# Muestra los encabezamientos de las columnas
cursor = connection.cursor()
cursor.execute(select_posts_comments_users)
cursor.fetchall()

column_names = [description[0] for description in cursor.description]
print(column_names)
```

Utilizando WHERE, podemos hacer consultas más específicas, como por ejemplo, mostrar los posts y cuántos likes ha recibido

```
# Muestra cada post y el número total de likes que ha recibido
select_post_likes = """
SELECT
  description as Post,
  COUNT(likes.id) as Likes
FROM
  likes,
  posts
WHERE
  posts.id = likes.post_id
GROUP BY
  likes.post_id
"""

post_likes = execute_read_query(connection, select_post_likes)

for post_like in post_likes:
    print(post_like)
```

Vemos ahora cómo modificar uno de los comentarios. Primero lo mostramos, para ubicarnos, y después utilizamos UPDATE/SET

```
# Muestra uno de los posts, que se va a modificar después
select_post_description = "SELECT description FROM posts WHERE id = 2"

post_description = execute_read_query(connection, select_post_description)

for description in post_description:
    print(description)

# Actualiza el post que se acaba de seleccionar
update_post_description = """
UPDATE
  posts
SET
  description = "The weather has become pleasant now"
WHERE
  id = 2
"""

execute_query(connection, update_post_description)
```

Análogamente, podemos borrar un comentario:

```
# Borra el comentario con el id 5
delete_comment = "DELETE FROM comments WHERE id = 5"
execute_query(connection, delete_comment)
```

carmenbvg@gmail.com

```
1  def gratitude():
2      print("Thank you.")
3
```