Rodriguez Cabral, Juan Maria DNI: 35.321.415

Coderhouse

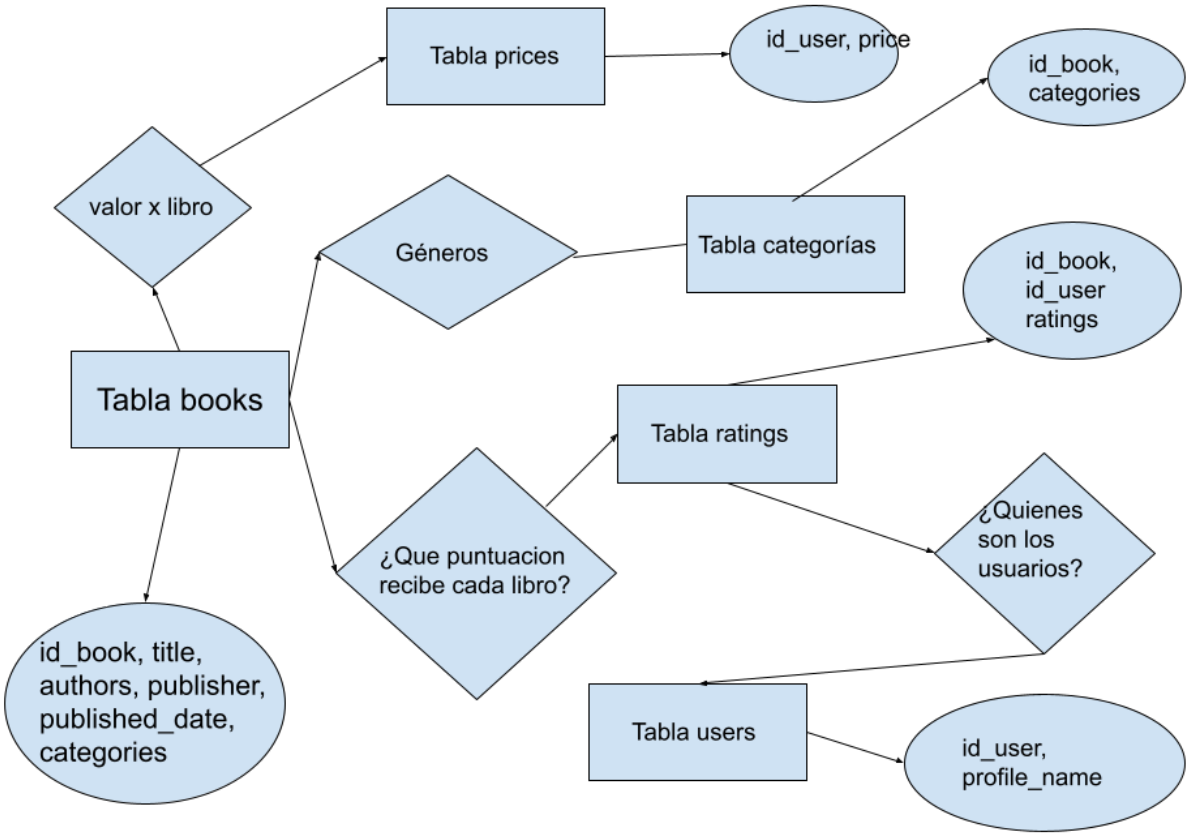
1° Entrega de proyecto final

Base de datos elegida, y temática de la misma:

La BD elegida originalmente contiene una lista de libros, con diversos datos tales como el título, autor, año de publicación, entre otros.
El Dataset incluía una segunda tabla con 3 millones de puntuaciones de usuarios de la plataforma Amazon.

Debido al gran tamaño de la base de datos, opte por una reducción de la información, cuidando de no dejar datos huérfanos. También de pasar de 2 tablas, a 5, las cuales detallaré más adelante. Todo este proceso fue llevado a cabo a través de Python, usando la librería Pandas. El código está disponible en Github, así como el link al dataset que se encuentra alojado en la plataforma Kaggle.

Diagrama entidad-relacion de la BD ya normalizada:



Descripción de las tablas

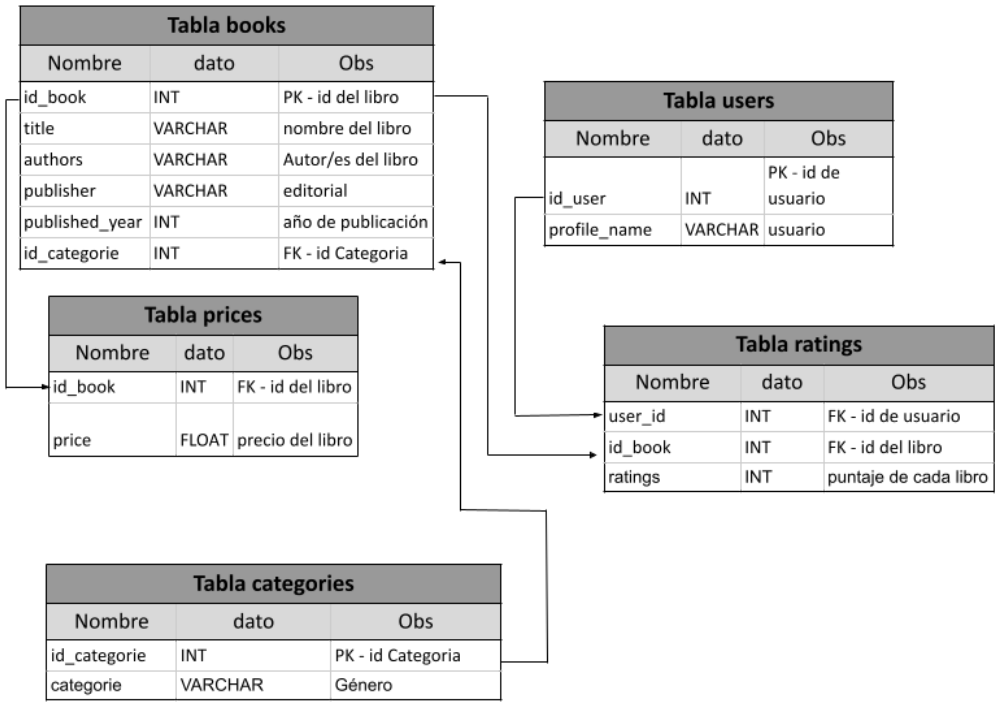


Tabla books

Nombre	dato	Obs
id_book	INT	PK - id del libro
title	VARCHAR	nombre del libro
authors	VARCHAR	Autor/es del libro
publisher	VARCHAR	editorial
published_year	INT	año de publicación
id_categorie	INT	FK - id Categoria

Tabla users

Nombre	dato	Obs
id_user	INT	PK - id de usuario
profile_name	VARCHAR	usuario

Tabla prices

Nombre	dato	Obs
id_book	INT	FK - id del libro
price	FLOAT	precio del libro

Tabla ratings

Nombre	dato	Obs
user_id	INT	FK - id de usuario
id_book	INT	FK - id del libro
ratings	INT	puntaje de cada libro

Tabla categories

Nombre	dato	Obs
id_categorie	INT	PK - id Categoria
categorie	VARCHAR	Género

Script de creación de Schema y Tablas

```
CREATE SCHEMA IF NOT EXISTS books;

USE books;

CREATE TABLE IF NOT EXISTS categories (
id_categories INT NOT NULL PRIMARY KEY,
categories VARCHAR (255)
);

CREATE TABLE IF NOT EXISTS books (
id_book INT NOT NULL PRIMARY KEY,
title VARCHAR (500),
authors VARCHAR (500),
publisher VARCHAR (500),
published_year INT NOT NULL,
id_categories INT NOT NULL, FOREIGN KEY (id_categories) REFERENCES categories
(id_categories)
);


CREATE TABLE IF NOT EXISTS prices (
id_book INT NOT NULL,
FOREIGN KEY (id_book) REFERENCES books (id_book),
```

```
price FLOAT NOT NULL
);

CREATE TABLE IF NOT EXISTS users (
id_user INT NOT NULL PRIMARY KEY,
profile_name VARCHAR(255)
);

CREATE TABLE IF NOT EXISTS ratings (
id_user INT NOT NULL,
FOREIGN KEY (id_user) REFERENCES users (id_user),
id_book INT NOT NULL,
FOREIGN KEY (id_book) REFERENCES books (id_book),
score INT NOT NULL
)
```

Script de inserción de filas

Aclaración: Teniendo en cuenta que las tablas tienen varias miles de filas, voy a dejar en este espacio una muestra de inserción de cada una de las tablas, que consta de 5 filas por tabla. En el repositorio de Github se puede encontrar el script completo.

Insert table categories:

```
use books;
INSERT INTO categories (id_categories, categories) VALUES (1, 'Fiction');
INSERT INTO categories (id_categories, categories) VALUES (2, 'Religion');
INSERT INTO categories (id_categories, categories) VALUES (3, 'Biography &
Autobiography');
INSERT INTO categories (id_categories, categories) VALUES (4, 'Technology &
Engineering');
INSERT INTO categories (id_categories, categories) VALUES (5, 'Law');
```

Insert table books:

```
INSERT INTO books (id_book, title, authors, publisher, published_year, id_categories)
VALUES (1, 'Whispers of the Wicked Saints', 'Veronica Haddon', 'iUniverse', 2005, 1);
INSERT INTO books (id_book, title, authors, publisher, published_year, id_categories)
VALUES (2, 'The Church of Christ: A Biblical Ecclesiology for Today', 'Everett Ferguson',
'Wm. B. Eerdmans Publishing', 1996, 2);
INSERT INTO books (id_book, title, authors, publisher, published_year, id_categories)
VALUES (3, 'Saint Hyacinth of Poland', 'Mary Fabyan Windeatt', 'Tan Books & Pub', 2009, 3);
INSERT INTO books (id_book, title, authors, publisher, published_year, id_categories)
VALUES (4, 'Vector Quantization and Signal Compression (The Springer International Series
in Engineering and Computer Science)', 'Allen Gersho, Robert M. Gray', 'Springer Science &
Business Media', 2012, 4);
```

Insert table prices:

```
INSERT INTO prices (id_book, price) VALUES (1, 10.95);
INSERT INTO prices (id_book, price) VALUES (2, 25.97);
INSERT INTO prices (id_book, price) VALUES (3, 13.95);
INSERT INTO prices (id_book, price) VALUES (4, 76.94);
INSERT INTO prices (id_book, price) VALUES (5, 14.95);
```

Insert Table Users:

```
INSERT INTO users (id_user, profile_name) VALUES (1, 'Book Reader');
INSERT INTO users (id_user, profile_name) VALUES (2, 'V. Powell');
INSERT INTO users (id_user, profile_name) VALUES (3, 'LoveToRead Actually Read
Books');
INSERT INTO users (id_user, profile_name) VALUES (4, 'Clara');
INSERT INTO users (id_user, profile_name) VALUES (5, 'Tonya');
```

Insert table Ratings:

```
INSERT INTO ratings (id_user, id_book,score) VALUES (1, 1, 1);
INSERT INTO ratings (id_user, id_book,score) VALUES (2, 1, 4);
INSERT INTO ratings (id_user, id_book,score) VALUES (3, 1, 1);
INSERT INTO ratings (id_user, id_book,score) VALUES (4, 1, 5);
INSERT INTO ratings (id_user, id_book,score) VALUES (5, 1, 5);
```

2° Entrega de proyecto final

Listado de vistas

1- Esta es la View más sencilla de todas. Simplemente creo una vista que traerá una query que indica que libros fueron publicados en 2001. Implica traer todas las columnas que conforman la tabla books.

```
CREATE VIEW año_2001 AS
select * FROM books
WHERE published_year = 2001;
```

2- Como su nombre indica, se trata de una view que trae los libros mas caros. Puntualmente los que tienen un precio superior a 500. está conformada por la columna title(tabla books) y la columna prices(tabla prices)

```
CREATE VIEW libros_muy_caros AS
SELECT
```

```
        title,
    prices.price
FROM books
JOIN prices ON books.id_book = prices.id_book
WHERE price > 500;
```

3- Está View traerá como resultado un listado de libros cuya categoria indique que corresponden a Technology & Engineering, y, al mismo tiempo, que su fecha de publicación sea posterior o igual al año 2010. Esta tabla estará ordenada por año de publicación, de más reciente a más antiguo. Esta conformada por las columnas title, authors, published_year (tabla books). Para poder buscar por la categoria Technology & Engineering, usamos un JOIN con la tabla categories.

```
CREATE VIEW Libros_tecnologia AS
SELECT
    title,
    authors,
    published_year
FROM books
JOIN categories ON books.id_categories = categories.id_categories
WHERE categories = 'Technology & Engineering' AND published_year >= 2010
ORDER BY published_year DESC;
```

4 - Esta view devolvera una lista de los 100 libros mas votados por los usuarios. Para esto, usamos las columnas title, authors (tabla books), y la columna id_book (tabla ratings) la cual se trabaja con la función de agregación COUNT en esta ultima columna. Se joinea las tablas books y ratings, y se ordena en forma descendente según la cantidad de votos.

```
CREATE VIEW libros_mas_votados AS
SELECT
    books.title,
    books.authors,
    COUNT(ratings.id_book) AS cantidad_de_votos
FROM books
JOIN ratings ON books.id_book = ratings.id_book
GROUP BY title, authors
ORDER BY cantidad_de_votos DESC
LIMIT 100;
```

5 - La última view devolverá un listado de los 25 usuarios que más puntuaciones dieron a los libros. Para esto se utiliza las columnas profile_name (tabla users) y id_user (FK de la tabla ratings), se agrupa por profile_name, y se procede a usar la función de agregación COUNT en id_user. Por último se ordena de forma descendente según la cantidad de puntuaciones que haya realizado cada usuario.

```
CREATE VIEW TOP_25_USUARIOS AS
SELECT
```

```
    users.profile_name AS usuario,
    COUNT(ratings.id_user) AS cantidad_puntuaciones
FROM users
JOIN ratings ON users.id_user = ratings.id_user
GROUP BY profile_name
ORDER BY cantidad_puntuaciones DESC
LIMIT 25;
```

Listado de funciones

1 - Esta primera función hace el cálculo de IMC (índice de masa corporal). Para ejecutarlo hay que declarar en el select tanto peso como altura. La consulta devolverá el valor final del IMC. El objetivo de esta función, si bien no esta relacionada con la base de datos que estoy usando, era probar las funcionalidades de esta herramienta.

```
USE `books`;
DROP function IF EXISTS `calculadoraIMC`;

DELIMITER $$
USE `books`$$
CREATE FUNCTION calculadoraIMC(peso FLOAT, altura FLOAT)
RETURNS VARCHAR(255)
NO SQL
BEGIN
DECLARE imc FLOAT;
  SET imc = peso / (altura * altura);
  RETURN CONCAT('Tu índice de masa corporal es: ', imc);
END$$

DELIMITER ;

SELECT calculadoraIMC(75, 1.8);
#aca se declara peso y altura, el resultado de esto sera:


  Tu índice de masa corporal es:
  23.1481
```

2 - Esta función traerá la cantidad de libros escritos por autor, según los valores que encuentre en la tabla. Lo único que necesitamos poner es una parte del nombre o apellido del autor a la hora de llamar a la función. Esta función utiliza las columnas de la tabla books.

```
DROP function IF EXISTS `cantidadLibrosPorAutor`;

DELIMITER $$
USE `books`$$
```

```
CREATE FUNCTION cantidadLibrosPorAutor(author_name VARCHAR(500))
RETURNS INTEGER
READS SQL DATA
BEGIN
   DECLARE cant_libros INT;
   SELECT COUNT(title) INTO cant_libros
   FROM books
   WHERE authors LIKE CONCAT('%', author_name, '%');
   RETURN cant_libros;
END$$

DELIMITER ;

SELECT cantidadLibrosPorAutor('Shakespeare');
```

Stored Procedures

El SP elegido sirve para insertar nuevos libros en la tabla books, sin necesitar usar las sentencias INSERT INTO.
Temas a mejorar:
Debería hacer un SP que me permita ingresar valores en las demás tablas. O, ver de crear un nuevo SP donde pueda ingresar todo junto. Osea, en un mismo SP, ingresar los datos de tabla books + precios + ratings.
Por otro lado, debería hacer que id_book se ingrese de forma incremental automática. Ese tema me esta fallando

```
USE `books`;
DROP procedure IF EXISTS `InsertarLibro`;

DELIMITER $$
USE `books`$$
CREATE PROCEDURE InsertarLibro(
   IN p_id_book INT,
   IN p_title VARCHAR(1000),
   IN p_authors VARCHAR(1000),
   IN p_publisher VARCHAR(1000),
   IN p_published_year INT,
   IN p_id_categories INT
)
BEGIN
   DECLARE v_id_book INT;

   SELECT id_book INTO v_id_book FROM books WHERE title = p_title;

   IF v_id_book IS NOT NULL THEN
      SELECT 'El libro ya existe en la tabla' AS mensaje;
```

```
    ELSE
        INSERT INTO books(id_book, title, authors, publisher, published_year, id_categories)
        VALUES(p_id_book, p_title, p_authors, p_publisher, p_published_year,
p_id_categories);
        SELECT 'Libro insertado correctamente' AS mensaje;
    END IF;
END;$$

DELIMITER ;
```

Caso de prueba:

```
CALL spInsertarLibro(33914, 'El nombre de la rosa', 'Umberto Eco', 'Lumen', 1980, 140);
```

Trigger

Opte por elegir un Trigger que se complemente con el SP elegido. En este caso, antes de que se lleve a cabo el insert, este Trigger verificará si el número de id_categories ingresado ya existe en la tabla categories. En caso de que no, la inserción del libro se frenará y recibiré un aviso de error.

```
DELIMITER $$
CREATE TRIGGER validar_categoria
BEFORE INSERT ON books
FOR EACH ROW
BEGIN
    DECLARE categoria_valida INT;
    SELECT id_categories INTO categoria_valida FROM categories WHERE id_categories =
NEW.id_categories;
    IF categoria_valida IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La categoría especificada no
existe.';
    END IF;
END$$

DELIMITER ;
```

Casos de prueba (utilizando el SP previamente creado):

```
CALL spInsertarLibro(33915, 'El Cementerio de Praga', 'Umberto Eco', 'Lumen', 2010,
2000);
```

En este caso, el id_categories 2000 no existe, con que no podre realizar la inserción.

```
CALL spInsertarLibro(33915, 'El Cementerio de Praga', 'Umberto Eco', 'Lumen', 2010, 1);
```

Este sera un caso de exito