RODRIGUEZ CABRAL, JUAN MARIA

CODERHOUSE

# SQL

Entrega de proyecto final para Coderhouse

# Indice

# Indice

# Indice

01

# DB

Elección de la temática, y del dataset

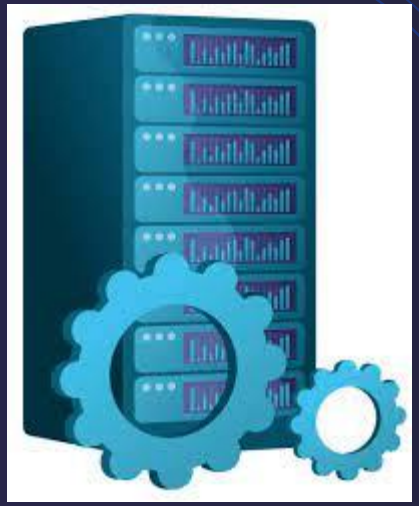
# SOBRE EL DATASET

El Dataset elegido originalmente se puede encontrar en la plataforma Kaggle, contiene 2 tablas con información acerca de libros y reseñas de los usuarios de la plataforma amazon.


Link al dataset:
https://www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews

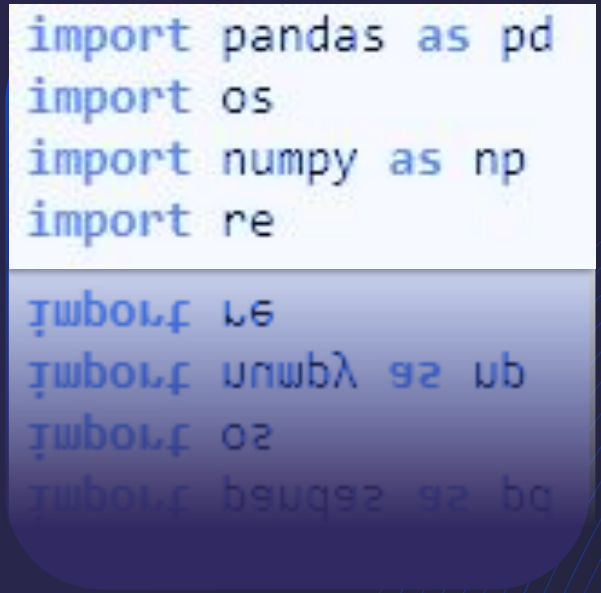
# SOBRE EL DATASET

El objetivo del presente trabajo es poder crear una base de datos a partir de la información recopilada en el dataset de la plataforma, buscando indicadores generales acerca de los libros, los puntajes recibidos, que libros son más populares, o más caros. También buscar los exponentes principales de cada género, como ya se verá más adelante, entre otros.
También se espera mostrar procesos para automatizar y mejorar las funcionalidades de la BD en cuestión.



# ACLARACION

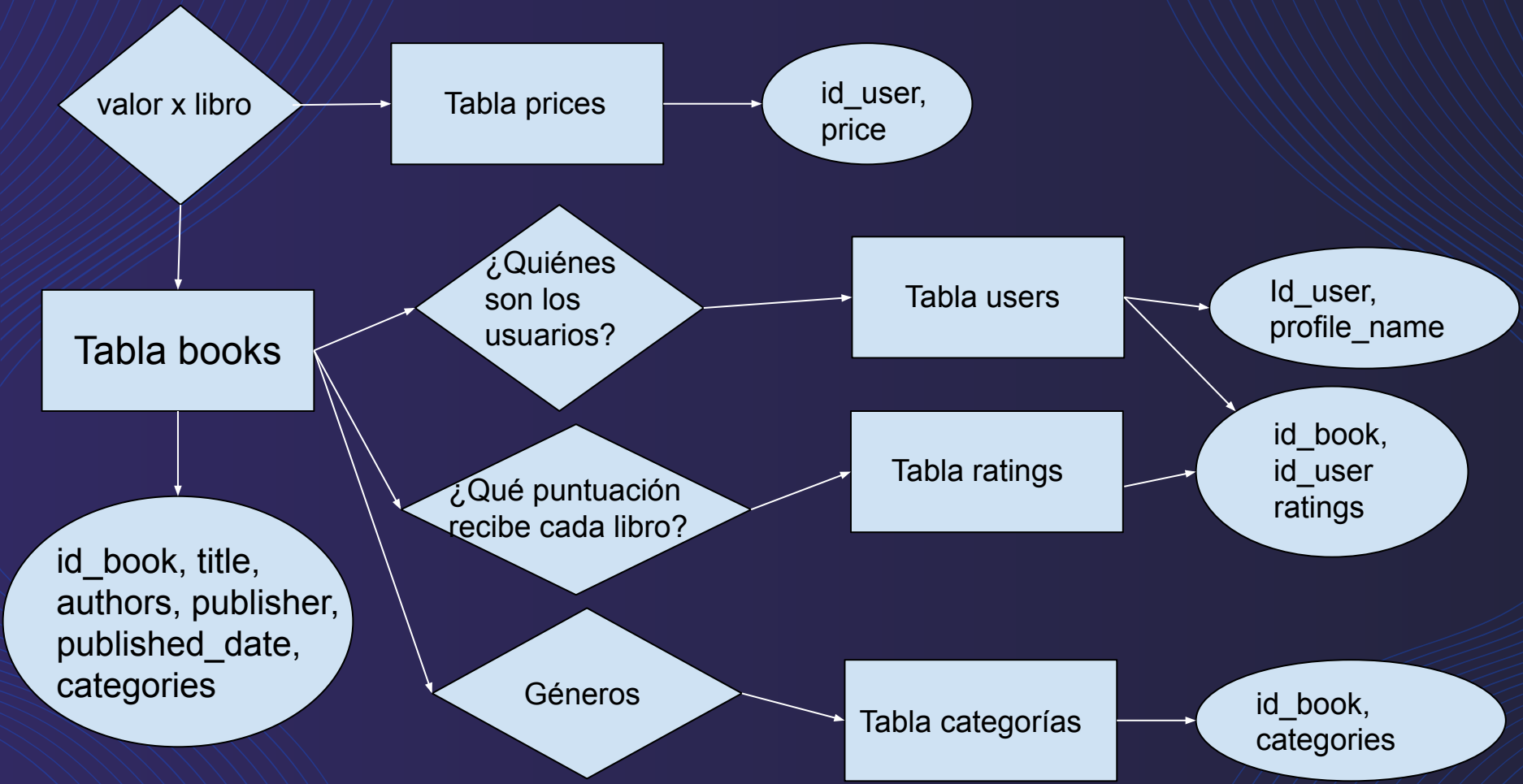
Debido al gran tamaño del dataset, el mismo fue limpiado y normalizado con Python, y su contenido recortado para facilitar el trabajo con MySQL.
La base original contiene 3 millones de filas. De esta manera, se optó por aplicar un proceso de submuestreo al dataset con el fin de simplificar su manipulación y análisis. Para mayor información, se recomienda recurrir al código, el cual está compartido y comentado en el presente repositorio.



02

# Diagrama ER

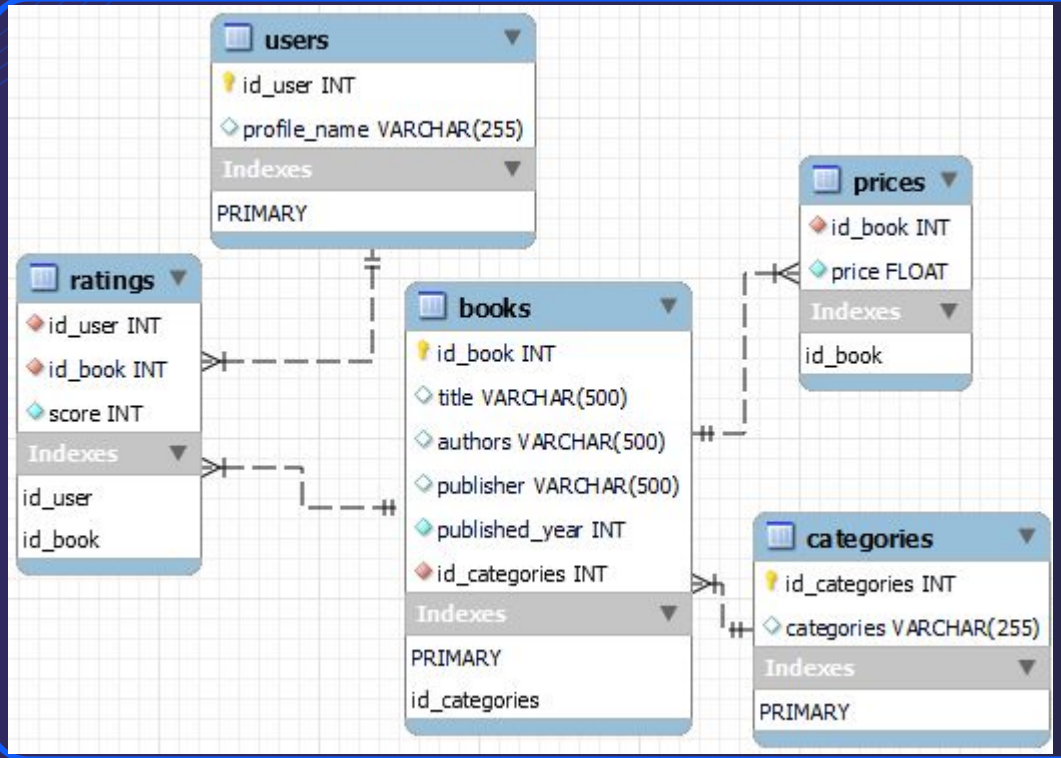
Diagrama ER de la BD ya normalizada


valor x libro
Tabla prices
id_user,
price
Tabla books
¿Quiénes
son los
usuarios?
Tabla users
Id_user,
profile_name
id_book,
id_user
ratings
¿Qué puntuación
recibe cada libro?
Tabla ratings
id_book, title,
authors, publisher,
published_date,
categories
Géneros
Tabla categorías
id_book,
categories

03

# Tablas

Descripción esquemática de las tablas y su composición




Como bien se mencionó antes, en este diagrama se encuentran presentes las 5 tablas, con sus Primary keys (PK) y Foreign Keys (FK).

La tabla books es la tabla principal del proyecto.

04

# SCHEMA

Script de creación y documentación del Schema y de las tablas.

# Creacion del Schema

```
CREATE SCHEMA IF NOT EXISTS books;
```

```
USE books;
```

Con el primer script doy creación al schema, y con el segundo declara que en este script se usara la BD “books”.

# CREACIÓN DE LAS TABLAS

En los siguientes scripts, se dará creación a las 5 tablas que van a componer el proyecto.

Primero se crea la tabla “categories”, la cual contiene una PK, que será referenciada por la tabla central del proyecto: books, la cual es ingresada a continuación.

```
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)
);
```

Luego se procede creado las 3 tablas faltantes:

Prices: contiene lo id de los libros, junto con su precio de mercado.

Users: Tabla donde se encuentra almacenado el nickname de cada usuario que puntúa los libros.

Ratings: Donde está plasmada la puntuación que cada usuario da a cada libro.

```
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
)
```

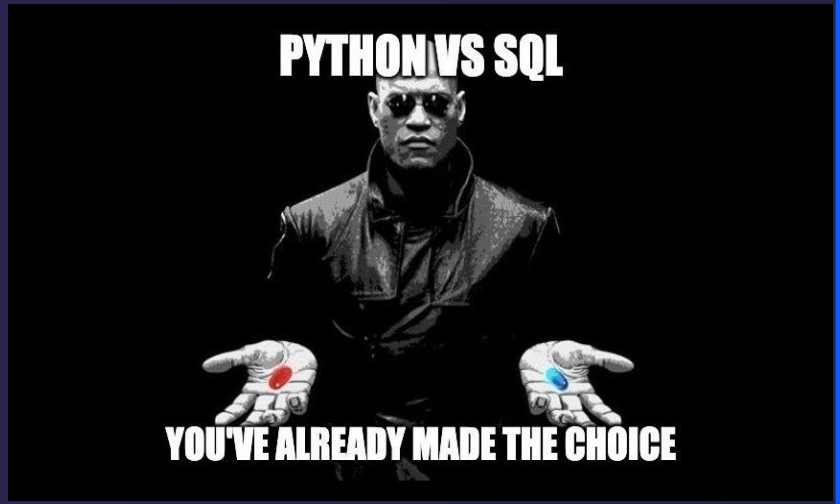
05

# Inserts

Script de creación y documentación de Inserts.

Una vez que las tablas ya se encuentran creadas, resta poblarlas. El orden para la inserción de datos en cada tabla será el mismo que el de la creación de las tablas.

Cabe mencionar que la sintaxis para cada insert ha sido creada en el script de Python, al igual que nos id necesarios. Esto es importante  de mencionar ya que no han sido declarados como autoincrementales en la creación de la tabla.



```sql
INSERT INTO categories (id_categories, categories) VALUES (1, 'Religion');
INSERT INTO categories (id_categories, categories) VALUES (2, 'Music');
INSERT INTO categories (id_categories, categories) VALUES (3, 'Education');
INSERT INTO categories (id_categories, categories) VALUES (4, 'Fiction');
INSERT INTO categories (id_categories, categories) VALUES (5, 'Business & Economics');
```

```sql
INSERT INTO books (id_book, title, authors, publisher, published_year, id_categories)
VALUES (1, 'Under the Overpass: A Journey of Faith on the Streets of America', 'Mike Yankoski', 'Multnomah', 2009, 1);
INSERT INTO books (id_book, title, authors, publisher, published_year, id_categories)
VALUES (2, 'Magic Music from the Telharmonium', 'Reynold Weidenaar', 'Reynold Weidenaar', 1995, 2);
INSERT INTO books (id_book, title, authors, publisher, published_year, id_categories)
VALUES (3, 'Widening the Circle: Culturally Relevant Pedagogy for American Indian Children',
 'Beverly J. Klug, Patricia T. Whitfield', 'Routledge', 2012, 3);
INSERT INTO books (id_book, title, authors, publisher, published_year, id_categories)
VALUES (4, 'Lure of the Mountains: The Frontier Life of a Mountain Man', 'Wayde Bulow', 'iUniverse', 2001, 4);
INSERT INTO books (id_book, title, authors, publisher, published_year, id_categories)
VALUES (5, 'Asian Brand Strategy: How Asia Builds Strong Brands', 'M. Roll', 'Springer', 2016, 5);
```

```sql
INSERT INTO prices (id_book, price) VALUES (1, 10.19);
INSERT INTO prices (id_book, price) VALUES (2, 80.0);
INSERT INTO prices (id_book, price) VALUES (3, 160.0);
INSERT INTO prices (id_book, price) VALUES (4, 14.95);
INSERT INTO prices (id_book, price) VALUES (5, 54.35);
```

Los Insert visualizados son una muestra a modo de ejemplo, ya que cada tabla tiene al menos mil filas.

```sql
INSERT INTO users (id_user, profile_name) VALUES (1, 'T.S.');
INSERT INTO users (id_user, profile_name) VALUES (2, 'Randall Biddle');
INSERT INTO users (id_user, profile_name) VALUES (3, 'M. Character justabunchofcharacters');
INSERT INTO users (id_user, profile_name) VALUES (4, 'Minnesota Mom');
INSERT INTO users (id_user, profile_name) VALUES (5, 'Robert');
```

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

06

# VIEWS

Script de creación y documentación de las Views

# ¿Qué son las views?

Una view es una consulta almacenada que se presenta como una tabla virtual. Esta se define mediante una consulta SQL y se guarda en la base de datos. Actúa como una tabla lógica que se deriva de los datos de una o más tablas subyacentes. Su finalidad es permitir simplificar y personalizar la forma en que se accede y se presenta la información almacenada en la base de datos.

Se han creado 6 views, cada uno con una finalidad diferente, los cuales se verán a continuación.

# libros_tecnologia

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

Esta view  muestra valores de una sola tabla, pero para poder mostrar los resultados, debe buscar en la tabla categories un valor en específico. El resultado final será una tabla con todos los libros cuyo género corresponda a 'Technology & Engineering', y su fecha de publicación haya sido posterior a 2010.

# libros_mas_votados

Como bien indica el nombre, esta view devuelve una lista de los 100 libros con más cantidad de votos según los usuarios. Surge de joinear la tabla ratings, con la tabla books. A su vez, esta ordenada de meyor a menor, segun cantidad de votos que haya recibido un libros.

```sql
CREATE VIEW libros_mas_votados AS
SELECT
    cat.id_categories,
    bo.title AS libro,
    bo.authors AS autor,
    COUNT(rat.id_book) AS cantidad_de_votos_por_libro
FROM categories cat
JOIN books bo ON cat.id_categories = bo.id_categories
JOIN ratings rat ON bo.id_book = rat.id_book
GROUP BY cat.id_categories, bo.title, bo.authors
ORDER BY cantidad_de_votos_por_libro DESC;
```

# top_25_usuarios

```sql
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;
```

Similar a la view anterior, pero enfocada en los usuarios, esta view devolvera los los nicknames de los usuarios que mas votos hayan realizado en la plataforma. Nuevamente, ordenado de mayor a menor, con un tope de 25 usuarios mostrados.

# publishers

Esta view, devolvera una lista de publishers. La información que contendrá la tabla será, la cantidad de libros publicados por la editorial, el promedio de precio de venta al público, y por último la cantidad de votos que ha recibido cada editorial por el total de libros votados. Esto permitirá deducir cuales son las editoriales mas populares según los usuarios de amazon

```sql
CREATE VIEW publishers AS
SELECT
        bo.publisher,
        COUNT(bo.publisher) AS total_libros,
        ROUND(AVG(pr.price),2) AS precio_promedio,
        COUNT(ra.score) AS cantidad_votos
FROM books bo
JOIN prices pr ON bo.id_book = pr.id_book
JOIN ratings ra ON bo.id_book = ra.id_book
GROUP BY bo.publisher
ORDER BY cantidad_votos DESC;
```

# generos_mas_populares

```sql
CREATE VIEW generos_mas_populares AS
SELECT
        cat.id_categories,
        cat.categories AS generos,
        COUNT(rat.score) AS cantidad_puntuaciones_por_genero
FROM categories cat
JOIN books bo ON cat.id_categories = bo.id_categories
JOIN ratings rat ON bo.id_book = rat.id_book
GROUP BY id_categories
ORDER BY cantidad_puntuaciones_por_genero DESC;
```

Esta view traera como resultado una lista de los géneros literarios más populares entre los usuarios de Amazon, junto con la cantidad de votos (estos referenciados a los libros de cada género) que hayan recibido.

# libro_mas_popular_segun_genero

Esta view se crea a partir de la información recopilada en otras 2 views (generos_mas_populares y libros_mas_votados). Tiene como objetivo mostrar en una misma lista, tanto el género literario más popular, como el mayor exponente de ese género, según los usuarios

```sql
CREATE VIEW libro_mas_popular_segun_genero AS
SELECT
    gmp.generos,
    gmp.cantidad_puntuaciones_por_genero AS A,
    lmv.libro,
    lmv.autor,
    lmv.cantidad_de_votos_por_libro AS B
FROM generos_mas_populares gmp
JOIN libros_mas_votados lmv ON gmp.id_categories = lmv.id_categories
JOIN (
    SELECT id_categories, MAX(cantidad_de_votos_por_libro) AS max_votos
    FROM libros_mas_votados
    GROUP BY id_categories
) lmv_max ON lmv.id_categories = lmv_max.id_categories
    AND lmv.cantidad_de_votos_por_libro = lmv_max.max_votos
ORDER BY B DESC
LIMIT 10;
```

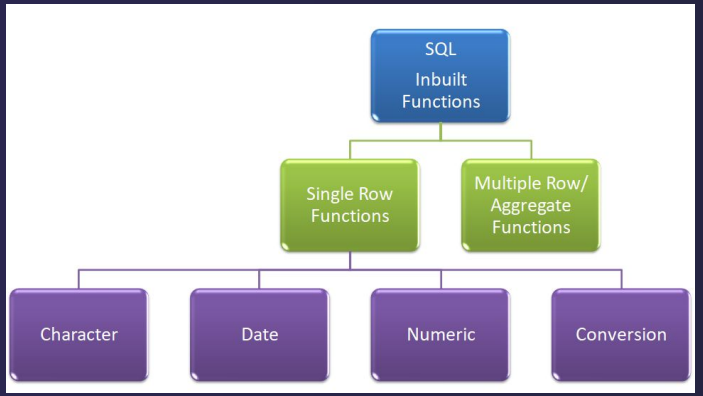
07

# FUNCTIONS

Script de creación y documentación de las Functions

# ¿Que son las functions?

Una function es un bloque de código SQL que acepta parámetros de entrada, realiza cálculos o manipulaciones en los datos y devuelve un valor como resultado. Pueden ser funciones integradas proporcionadas por el gestor  o funciones definidas por el usuario. Las funciones pueden ser utilizadas en consultas SQL para realizar operaciones sobre los datos.



# Calculadora IMC

```
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);
```

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 (Índice de Masa Muscular)

# promedioRating

```
DELIMITER $$
USE `books`$$
CREATE FUNCTION promedioRating(book_name VARCHAR(500))
RETURNS INTEGER
READS SQL DATA
BEGIN
    DECLARE avg_ratings INT;
    SELECT ROUND(AVG(score), 2) AS puntaje_promedio INTO avg_ratings
    FROM ratings
    JOIN books on ratings.id_book = books.id_book
    WHERE title = book_name;
    RETURN avg_ratings;
END$$

DELIMITER ;
```

Esta funcion traera el voto promedio de un libro, segun los usuarios de Amazon que hayan votado por el mismo.

```
SELECT promedioRating('Lure of the Mountains: The Frontier Life of a Mountain Man');
```

# 08

# SP

Script de creación y documentación de los SP

# ¿Qué son los Stored Procedures?

Un Stored Procedure es un conjunto de instrucciones SQL que se guarda en la base de datos y se puede llamar y ejecutar varias veces. Estos procedimientos almacenados en el gestor de base permiten encapsular lógica empresarial compleja dentro de la base de datos. Pueden aceptar parámetros de entrada, realizar operaciones en la base de datos y devolver resultados.

# 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 ;
```

El Stored Procedure elegido, sirve para poder insertar una nueva línea en la tabla books, sin tener que especificar la sentencia insert into, ni hacer mención de los nombres de las columnas, únicamente los datos del libro a ingresar

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

Al llamar al Stored Procedure para ingresar un nuevo titulo (en este caso, El Nombre de la Rosa), como resultado recibiremos este mensaje, siempre y cuando, el libro no existiese ya en la tabla.

| mensaje |
| --- |
| Libro insertado correctamente |

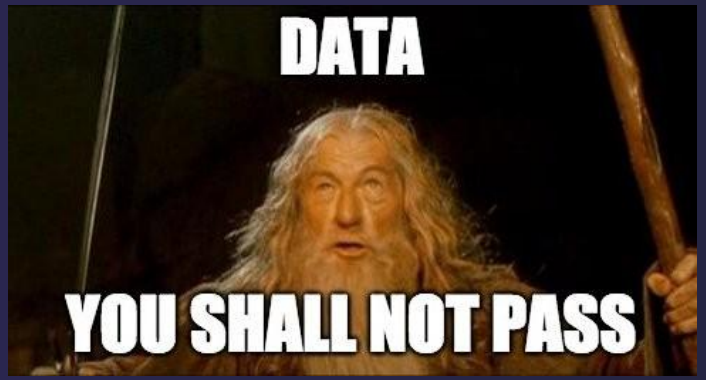
09

# TRIGGERS

Script de creación y documentación de los Triggers

# ¿Qué son los triggers?

Un trigger es un objeto que se asocia a una tabla y se activa automáticamente cuando ocurren ciertos eventos en esa tabla, como la inserción, actualización o eliminación de registros. Los triggers se utilizan para realizar acciones adicionales en la base de datos en respuesta a esos eventos, como validar datos, actualizar otras tablas o generar registros de auditoría.



# validar_categoria

El trigger elegido sirve para validar categorías. SI un libro ingresado tiene un id_categorie que todavía no haya sido creado, se recibirá un mensaje de error.

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

# Caso de ejemplo

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

Al intentar insertar el libro “El cementerio de praga”, con id_categorie = 2000, la cual no existe, aparecerá el siguiente mensaje de error:


En cambio,al realizar la insercion , con id_categorie = 1, la inserción se realizara de forma correcta, dandose el siguiente desenlace:

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


| | mensaje |
|---|---|
| ▶ | Libro insertado correctamente |

10

# DLC

Script y documentación sobre Data Control Language

# Caso de ejemplo

```
START TRANSACTION;
DELETE FROM books WHERE id_book = 33915;
SAVEPOINT eliminacionRegistro33915;
-- ROLLBACK TO eliminacionRegistro33915;
COMMIT;

START TRANSACTION;
-- Inserción de los registros en la tabla RATINGS
INSERT INTO RATINGS (id_user, id_book, score) VALUES (4, 33914, 4);
INSERT INTO RATINGS (id_user, id_book, score) VALUES (5, 33914, 4);
INSERT INTO RATINGS (id_user, id_book, score) VALUES (7, 33914, 4);
INSERT INTO RATINGS (id_user, id_book, score) VALUES (9, 33914, 5);
SAVEPOINT insercionPrimeros4Registros;
INSERT INTO RATINGS (id_user, id_book, score) VALUES (15, 33914, 5);
INSERT INTO RATINGS (id_user, id_book, score) VALUES (21, 33914, 5);
INSERT INTO RATINGS (id_user, id_book, score) VALUES (85, 33914, 5);
INSERT INTO RATINGS (id_user, id_book, score) VALUES (314, 33914, 5);
SAVEPOINT insercionUltimos4Registros;
-- ROLLBACK TO punto1;
COMMIT;
```

Este proceso consta de varios pasos:

1 – Iniciar la transaccion con “START TRANSACTION”

2 – Se realiza una eliminación de un registro específico en la tabla books utilizando el comando DELETE.

# Caso de ejemplo

3 - Se establece un punto de guardado utilizando el comando SAVEPOINT. Esto permite volver atrás hasta este punto en caso de necesitarlo.

4 -Se muestra el ejemplo de cómo volver hasta el punto de guardado, utilizando el comando ROLLBACK TO.

5 -Se confirma la transacción y se aplican los cambios realizados con el comando COMMIT.

6 - Se inicia otra transacción y se realizan inserciones en la tabla RATINGS. Se establecen dos puntos de guardado.

# Caso de ejemplo

7 - Al igual que en el caso anterior, se realiza un rollback hasta los puntos de guardado.

8 - Finalmente, se confirma la transacción y se aplican los cambios realizados.

Para terminar, los registros en la consola se veran asi

| | | | |
|---|---|---|---|
| 135 | 16:43:14 | INSERT INTO RATINGS (id_user, id_book, score) VALUES (7, 33914, 4) | 1 row(s) affected |
| 136 | 16:43:14 | INSERT INTO RATINGS (id_user, id_book, score) VALUES (9, 33914, 5) | 1 row(s) affected |
| 137 | 16:43:14 | SAVEPOINT insercionPrimeros4Registros | 0 row(s) affected |
| 138 | 16:43:14 | INSERT INTO RATINGS (id_user, id_book, score) VALUES (15, 33914, 5) | 1 row(s) affected |
| 139 | 16:43:14 | INSERT INTO RATINGS (id_user, id_book, score) VALUES (21, 33914, 5) | 1 row(s) affected |
| 140 | 16:43:14 | INSERT INTO RATINGS (id_user, id_book, score) VALUES (85, 33914, 5) | 1 row(s) affected |
| 141 | 16:43:14 | INSERT INTO RATINGS (id_user, id_book, score) VALUES (314, 33914, 5) | 1 row(s) affected |
| 142 | 16:43:14 | SAVEPOINT insercionUltimos4Registros | 0 row(s) affected |
| 143 | 16:43:14 | COMMIT | 0 row(s) affected |

11

# TCL

Script y documentacion sobre Transaction Control Language

# Creacion de usuario

Se trabajan 2 casos diferentes, con distintos niveles de permiso.

```
CREATE USER 'NicolasRodriguez'@'localhost' IDENTIFIED BY 'Coderhouse1234';
GRANT SELECT ON *.* TO 'NicolasRodriguez'@'localhost';
```

```
CREATE USER 'FlorenciaMoreno'@'localhost' IDENTIFIED BY 'Coderhouse123456';
GRANT SELECT, INSERT, UPDATE ON *.* TO 'FlorenciaMoreno'@'localhost';
```

En el primer caso, se crea un usuario (NicolasRodriguez), al cual se le otorgan únicamente permisos para visualización de las tablas. En el segundo caso, mediante el GRANT SELECT, se otorgan permisos puntuales. Además de lectura de las tablas, también podrá insertar y eliminar registros de las mismas.

# Gestion de permisos

Para poder corroboar los permisos de cada usuario, procedemos a realizar la siguiente sintaxis:

```
SHOW GRANTS FOR 'root'@'localhost';
SHOW GRANTS FOR 'NicolasRodriguez'@'localhost';
SHOW GRANTS FOR 'FlorenciaMoreno'@'localhost';
```

Por último, y solamente para estar seguros(ya que anteriormente, este permiso no fue asignado), con la sentencia REVOKE podemos eliminar los permisos que no queremos conceder. En este caso puntual, queremos evitar que ambos usuarios puedan hacer uso de la sentencia DELETE.

```
REVOKE DELETE ON *.* FROM 'NicolasRodriguez'@'localhost';
```

```
REVOKE DELETE ON *.* FROM 'FlorenciaMoreno'@'localhost';
```

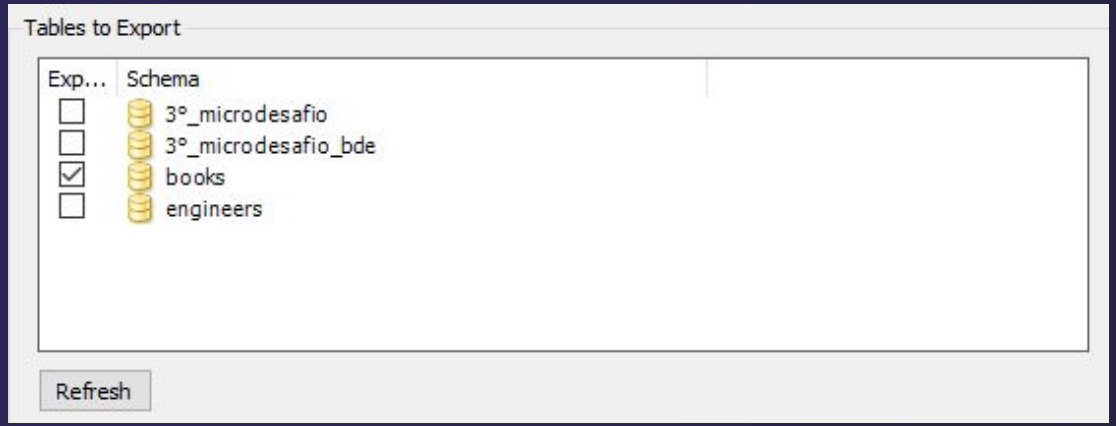
12

# BACKUP

Script y documentación sobre Backup

# Backup

Existen varias formas de crear un Backup de una BD. El que se vera a continuacion se puede gestionar desde el Workbench de MySQL. Se selecciona Data Export, dentro del apartado de adminitracion de nuestro Schema, se selecciona la BD a la cual queremos realizarle el Backup, y luego se procede a seleccionar algunos aspectos puntuales.

# Backup

Como se puede ver en el ejemplo, seleccionamos los objetos a exportar (en este caso, todos) y la carpeta donde queremos se almacene el Backup. Por ultimo se selecciona Start Export, y se realizara el proceso de Backup.