Clase 1: Introducción a Bases de Datos Relacionales y Modelado de Datos 🧠

Esta clase es el primer contacto con el mundo de las bases de datos relacionales, y la dedicaremos a entender los conceptos fundamentales y a practicar el diseño de una base de datos simple. Al final de esta sesión de 3 horas, no solo sabrás qué es una tabla, sino también cómo crear una base de datos con reglas que garanticen la calidad de la información.

Instrucciones para la Clase: Preparando el Entorno 🛠️

Antes de empezar, necesitamos tener las herramientas correctas. Para trabajar con bases de datos relacionales, utilizaremos un **SGBD** y un cliente gráfico.

- **¿Qué es un SGBD?** Un **Sistema Gestor de Base de Datos (SGBD)** es un software que nos permite crear, mantener y manipular los datos en una base de datos. Para esta clase, usaremos **MySQL**.
- **¿Qué es XAMPP?** Para usar MySQL en tu computadora, la forma más sencilla es instalar **XAMPP**. XAMPP es un paquete de software libre que incluye el servidor web Apache, **MySQL** y PHP, todo en un solo instalador. Es ideal para empezar.
    - **Enlace de descarga**: https://www.apachefriends.org/es/index.html
- **¿Qué es un cliente gráfico?** Es una aplicación que nos permite interactuar con nuestra base de datos de manera visual, sin tener que escribir todos los comandos en la línea de comandos. Las opciones más populares son:
    - **MySQL Workbench**: Es la herramienta oficial y es ideal para modelar bases de datos (hacer los diagramas que veremos) y ejecutar consultas.
    - **Enlace de descarga**: https://www.mysql.com/products/workbench/
    - **DBeaver**: Una alternativa muy potente y versátil que soporta muchos tipos de bases de datos, no solo MySQL.
    - **Enlace de descarga**: https://dbeaver.io/download/

Recomendamos tener instalado **XAMPP** y **MySQL Workbench** para esta clase.

---

1. Fundamentos: ¿Qué es una Base de Datos Relacional?

Imagina que quieres almacenar información de forma organizada. Una Base de Datos Relacional es una forma de organizar esos datos en tablas que están conectadas entre sí. La clave del modelo relacional es que evita la repetición de datos, permitiendo una gestión más eficiente y segura.

- **Tabla (o Entidad): Una colección de datos sobre un tema específico. Se compone de filas y columnas. Por ejemplo, una tabla de `autores` o una tabla de `libros`.**

- **Fila (o Registro): Una única entrada en una tabla. En la tabla `autores`, una fila representaría a un autor específico.**
- **Columna (o Atributo): Un campo de datos que describe una característica de la entidad. En la tabla `autores`, las columnas serían `nombre` o `nacionalidad`.**

---

2. La Columna Vertebral de los Datos: Claves y Relaciones

Las claves son los identificadores que nos permiten conectar tablas y garantizar la integridad de los datos.

- **Clave Primaria (`Primary Key - PK`): Una columna (o conjunto de columnas) que identifica de forma única cada fila de una tabla. Su principal propósito es garantizar la integridad de la entidad: cada registro debe ser único y no puede existir un duplicado.**
  - **Características:**
    - **Única: No puede haber dos filas con el mismo valor en la clave primaria.**
    - **No Nula (`NOT NULL`): No puede estar vacía.**
  - **Ejemplo: En una tabla de `autores`, `id_autor` sería la clave primaria.**
- **Clave Foránea (`Foreign Key - FK`): Una columna en una tabla que hace referencia a la clave primaria de otra tabla. Actúa como un "puente" o "enlace". Su función es garantizar la integridad referencial, asegurando que las relaciones entre tablas sean válidas.**
  - **Ejemplo: En la tabla `libros`, la columna `id_autor` es una clave foránea que apunta a `id_autor` en la tabla `autores`.**
- **Cardinalidad: Describe la cantidad de instancias de una entidad que pueden estar asociadas con las instancias de otra.**
  - **1 a 1: Una instancia de la entidad A está relacionada con una y solo una instancia de la entidad B. (Ej: `persona` y `pasaporte`).**
  - **1 a muchos: Una instancia de A puede estar relacionada con múltiples instancias de B, pero B solo puede estar relacionada con una A. (Ej: `autor` y `libros`).**
  - **Muchos a muchos: Múltiples instancias de A pueden estar relacionadas con múltiples instancias de B. (Ej: `libros` y `lectores`, ya que un libro puede ser leído por muchos lectores y un lector puede leer muchos libros). Para modelar**

esto, se utiliza una tabla intermedia o pivote (`prestamos` en nuestro caso) que contiene las claves foráneas de ambas tablas.

---

3. El ADN de los Datos: Tipos y Restricciones

Cuando creamos una columna, le asignamos un tipo de dato para definir qué tipo de información va a contener (números, texto, fechas, etc.). Además, le podemos aplicar restricciones para asegurar la calidad de esos datos.

Tipos de Datos Comunes en SQL

- **Tipos Numéricos:**
  - `INT`: Para números enteros (ej. `123`, `id_autor`).
  - `DECIMAL(p, s)`: Para números con decimales. `p` es la cantidad total de dígitos y `s` es la cantidad de decimales. (Ej: `DECIMAL(5, 2)` para precios).
  - `FLOAT` / `DOUBLE`: Para números con punto flotante (menos precisos que `DECIMAL`).
- **Tipos de Cadena de Caracteres:**
  - `VARCHAR(n)`: Para texto de longitud variable, donde `n` es la cantidad máxima de caracteres (ej. `VARCHAR(100)` para un nombre).
  - `CHAR(n)`: Para texto de longitud fija. Si el texto es más corto, se rellena con espacios.
  - `TEXT`: Para cadenas de texto muy largas, como el contenido de una reseña.
  - `ENUM`: Para un conjunto de valores predefinidos (ej. `ENUM('activo', 'inactivo')`).
- **Tipos de Fecha y Hora:**
  - `DATE`: Para almacenar solo la fecha (ej. `'2025-08-12'`).
  - `TIME`: Para almacenar solo la hora.
  - `DATETIME`: Para almacenar la fecha y la hora (ej. `'2025-08-12 10:30:00'`).
  - `YEAR`: Para almacenar un año.
- **Otros Tipos:**
  - `BOOLEAN`: Para valores de verdadero o falso (`TRUE`/`FALSE`). En MySQL se representa a menudo con `TINYINT(1)`.

Restricciones (`Constraints`) Clave

- **`PRIMARY KEY`: Una columna que es `UNIQUE` y `NOT NULL` a la vez. Es el identificador principal.**
- **`FOREIGN KEY`: Asegura la integridad referencial.**
- **`NOT NULL`: Asegura que la columna nunca puede tener un valor nulo.**
- **`UNIQUE`: Todos los valores en la columna deben ser únicos, pero puede ser nula.**
- **`DEFAULT`: Define un valor por defecto si no se especifica uno al insertar un registro.**
- **`CHECK`: Permite definir una condición que todos los valores en la columna deben cumplir.**
- **`ON DELETE [ACCIÓN]`: Define qué sucede con los registros dependientes cuando se borra un registro padre.**
    - **`CASCADE`: Borra en cascada. Si se borra el padre, los hijos también se borran.**
    - **`RESTRICT`: Evita el borrado. Si hay registros hijos, no se puede borrar el registro padre.**
    - **`SET NULL`: Establece el valor de la clave foránea en `NULL` en los registros hijos.**

---

4. De la Teoría a la Práctica: Normalización

La normalización es el proceso de diseñar la estructura de una base de datos para reducir la redundancia y mejorar la integridad. Es la razón por la que dividimos la información en múltiples tablas conectadas.

- **Problema de la redundancia: Si en una tabla `libros` guardamos el nombre y la nacionalidad del autor en cada fila, al actualizar la nacionalidad de un autor tendríamos que cambiarla en todos sus libros. Esto es ineficiente y propenso a errores.**
- **Solución: Normalizamos, creando una tabla `autores` separada y conectándola con la tabla `libros` a través de una clave foránea.**

---

5. Actividad Práctica: Creando la Base de Datos "Biblioteca"

A continuación, usaremos todas las herramientas teóricas que hemos aprendido para crear una base de datos funcional en MySQL.

Consulta SQL Completa y Corregida para la Base de Datos "Biblioteca"

Aquí está el script completo y funcional, que incluye la instrucción `DROP DATABASE` para empezar desde cero si es necesario, y la tabla `prestamos` con la sintaxis corregida para el modelador EER de MySQL Workbench.

```
-- Elimina la base de datos 'biblioteca' si ya existe para crearla desde cero.

DROP DATABASE IF EXISTS biblioteca;

-- Crear la base de datos (esquema)
CREATE DATABASE biblioteca;
USE biblioteca;

-- -----------------------------------------------------
-- Tabla de Autores: Se aplican PK y NOT NULL
-- -----------------------------------------------------
CREATE TABLE autores (
    id_autor INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    nacionalidad VARCHAR(50)
);

-- -----------------------------------------------------
-- Tabla de Libros: Aplica PK, UNIQUE, FK y NOT NULL
-- -----------------------------------------------------
CREATE TABLE libros (
    id_libro INT PRIMARY KEY AUTO_INCREMENT,
    titulo VARCHAR(255) NOT NULL,
    isbn VARCHAR(13) UNIQUE NOT NULL, -- El ISBN es único por cada libro
    año_publicacion YEAR,
    id_autor INT NOT NULL, -- Cada libro debe tener un autor
    FOREIGN KEY (id_autor) REFERENCES autores(id_autor)
        ON DELETE RESTRICT -- No se puede borrar un autor si tiene libros asociados
);

-- -----------------------------------------------------
-- Tabla de Lectores: Aplica PK, NOT NULL y UNIQUE
-- -----------------------------------------------------
CREATE TABLE lectores (
    id_lector INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL -- El email debe ser único para cada lector
);

-- -----------------------------------------------------
-- Tabla de Préstamos: La tabla de unión (muchos a muchos)
-- -----------------------------------------------------
```

```
CREATE TABLE prestamos (
    id_prestamo INT PRIMARY KEY AUTO_INCREMENT,
    id_libro INT NOT NULL,
    id_lector INT NOT NULL,
    fecha_prestamo DATE NOT NULL,  -- Corregido para compatibilidad con el modelador EER
    fecha_devolucion DATE,
    FOREIGN KEY (id_libro) REFERENCES libros(id_libro)
        ON DELETE CASCADE, -- Si se borra un libro, se borran sus registros de préstamo
    FOREIGN KEY (id_lector) REFERENCES lectores(id_lector)
        ON DELETE CASCADE, -- Si se borra un lector, se borran sus registros de préstamo
    CHECK (fecha_devolucion IS NULL OR fecha_devolucion >= fecha_prestamo)
);
```

Clase 2: Primeros Pasos con SQL y MySQL 💻

Esta clase es la transición de la teoría a la práctica de consulta de datos. Ahora que la base de datos está modelada, los estudiantes aprenderán a interactuar con ella.

1. Diferencias entre Relaciones "Identifying" y "Non-Identifying"

Este es un concepto crucial para entender la naturaleza de las relaciones en un diagrama Entidad-Relación (ERD).

- **Relación Identifying (Línea Continua)**: En una relación de identificación, la clave foránea (FK) **también es parte de la clave primaria** (PK) en la tabla hija. Esto significa que un registro en la tabla hija no puede existir sin un registro padre. La dependencia es fuerte. Se usa para entidades débiles que no pueden ser identificadas de forma única sin su padre.
- **Relación Non-Identifying (Línea Punteada)**: En una relación de no identificación, la clave foránea (FK) **no es parte de la clave primaria** en la tabla hija. El registro de la tabla hija puede ser identificado de forma única por su propia clave primaria, independientemente del registro padre. La dependencia es más débil. Es la relación más común.

2. Las Cláusulas Fundamentales de SQL

- **SELECT**: La base de cualquier consulta. Aprenderemos a seleccionar columnas específicas o todas (*).
- **FROM**: De qué tabla se obtienen los datos.
- **WHERE**: Filtrado de registros. Usaremos operadores para buscar datos que cumplan una condición específica.
- **ORDER BY**: Ordenar los resultados de forma ascendente (ASC) o descendente (DESC).

3. Operadores, Condicionales y Funciones de Agregación

- **Operadores de Comparación y Lógicos**: Utilizar AND, OR, NOT, >, <, =, != y otros operadores para construir consultas más complejas.
- **Funciones de Agregación**:
    - COUNT(): Contar registros en una tabla.

    - SUM(): Sumar los valores de una columna.
    - AVG(): Calcular el promedio.
    - MIN() y MAX(): Encontrar el valor mínimo y máximo de una columna.

4. Agrupación y Filtrado de Grupos

- **GROUP BY**: Agrupar registros que tienen valores idénticos para una o más columnas.
- **HAVING**: Filtrar registros después de que han sido agrupados. Se diferencia de WHERE en que HAVING opera sobre los resultados de las funciones de agregación.

Ejemplos de uso 💻

Usaremos la base de datos de la biblioteca como ejemplo para mostrar cómo se usan las cláusulas en consultas reales.

- **SELECT**: Para obtener el título y el ISBN de todos los libros.

```
SELECT titulo, isbn FROM libros;
```

- **FROM**: Se usa junto con SELECT.

```
SELECT * FROM autores;
```

- **WHERE**: Para encontrar los libros publicados después del año 2000.

```
SELECT titulo FROM libros WHERE año_publicacion > 2000;
```

- **ORDER BY**: Para listar a los lectores ordenados por su apellido de forma alfabética.

```
SELECT nombre, apellido FROM lectores ORDER BY apellido ASC;
```

---

3. Operadores, Condicionales y Funciones de Agregación

Estos elementos nos permiten construir consultas más complejas y obtener información valiosa de los datos.

- **Operadores de Comparación y Lógicos**: Para encontrar libros de un autor específico que se hayan publicado en un año concreto.

```
SELECT titulo FROM libros WHERE id_autor = 1 AND año_publicacion = 1967
```

- **Funciones de Agregación**:
    - COUNT(): Para saber cuántos libros ha escrito el autor con id_autor = 1.

```
SELECT COUNT(*) FROM libros WHERE id_autor = 1;
```

    - MIN() y MAX(): Para encontrar el año de publicación del primer y el último libro.

```
SELECT MIN(año_publicacion) AS primer_libro, MAX(año_publicacion) AS ultimo_libro FROM libros;
```

---

4. Agrupación y Filtrado de Grupos

Estas cláusulas son fundamentales para obtener estadísticas y resúmenes de los datos.

- **GROUP BY**: Para contar cuántos libros tiene cada autor. La consulta agrupa los libros por id_autor y luego cuenta las filas en cada grupo.

```
SELECT id_autor, COUNT(*) AS total_libros FROM libros GROUP BY id_autor;
```

- **HAVING**: Para encontrar los autores que han escrito más de un libro.

```
SELECT id_autor, COUNT(*) AS total_libros FROM libros GROUP BY id_autor HAVING COUNT(*) > 1;
```

La diferencia clave es que WHERE filtra filas individuales, mientras que HAVING filtra los resultados de los grupos creados por GROUP BY.

Para que puedas empezar a trabajar con tu base de datos biblioteca, aquí tienes las queries INSERT que te permitirán poblar las tablas con datos de ejemplo. He tenido en cuenta las relaciones y claves foráneas para que puedas ejecutar el script completo sin errores.

Queries para la Base de Datos "Biblioteca"

Este script insertará datos en las tablas autores, libros, lectores y prestamos, respetando la integridad referencial y las claves foráneas.

Poblar la tabla autores

Primero, insertamos los autores. Como id_autor es AUTO_INCREMENT, no necesitamos especificarlo en el INSERT.

```
INSERT INTO autores (nombre, apellido, nacionalidad) VALUES
('Gabriel', 'García Márquez', 'Colombiana'),
('J.R.R.', 'Tolkien', 'Británica'),
('George', 'Orwell', 'Británica'),
```

```
('Jane', 'Austen', 'Británica'),
('Haruki', 'Murakami', 'Japonesa');
```

Poblar la tabla `libros`

Ahora, insertamos los libros. Aquí es crucial que el `id_autor` que asignemos exista en la tabla `autores` que acabamos de llenar.

- `id_autor = 1` corresponde a 'Gabriel García Márquez'.
- `id_autor = 2` corresponde a 'J.R.R. Tolkien'.
- `id_autor = 3` corresponde a 'George Orwell'.
- `id_autor = 4` corresponde a 'Jane Austen'.

```
INSERT INTO libros (titulo, isbn, año_publicacion, id_autor) VALUES
('Cien años de soledad', '978-0307474161', 1967, 1),
('El amor en los tiempos del cólera', '978-0307389731', 1985, 1),
('El Señor de los Anillos', '978-0618053267', 1954, 2),
('1984', '978-0451524935', 1949, 3),
('Orgullo y Prejuicio', '978-0141439518', 1813, 4);
```

Poblar la tabla `lectores`

Insertamos algunos lectores. El `id_lector` también es `AUTO_INCREMENT`.

```
INSERT INTO lectores (nombre, apellido, email) VALUES
('Juan', 'Pérez', 'juan.perez@email.com'),
('María', 'Gómez', 'maria.gomez@email.com'),
('Carlos', 'López', 'carlos.lopez@email.com');
```

Poblar la tabla `prestamos`

Finalmente, insertamos los registros de préstamos. Esta es la tabla intermedia que relaciona `libros` y `lectores`. Necesitamos los IDs de ambas tablas.

- `id_libro = 1` ('Cien años de soledad') prestado al `id_lector = 1` ('Juan Pérez').
- `id_libro = 3` ('El Señor de los Anillos') prestado al `id_lector = 2` ('María Gómez').
- `id_libro = 4` ('1984') prestado al `id_lector = 1` ('Juan Pérez').

```
INSERT INTO prestamos (id_libro, id_lector, fecha_prestamo, fecha_devolucion) VALUES
(1, 1, '2025-08-01', '2025-08-15'),
(3, 2, '2025-07-20', '2025-08-05'),
(4, 1, '2025-08-10', NULL);
```

Consignas de Problemas a Resolver

1. Manipulación de Datos (DML)

1. **Ejercicio de `INSERT`**: Agrega un nuevo autor a la tabla `autores`. El autor es "**Isabel Allende**", de nacionalidad "**Chilena**".

2. **Ejercicio de `UPDATE`**: La lectora "**María Gómez**" ha cambiado su correo electrónico. Actualiza su dirección de email a "**maria.gomez.new@email.com**" en la tabla `lectores`.
3. **Ejercicio de `DELETE`**: Por un error administrativo, se necesita eliminar al autor "**Haruki Murakami**". Borra su registro de la tabla `autores`.

---

2. Funciones, Agrupación y Consultas Avanzadas

1. **Ejercicio de `COUNT`**: ¿Cuántos libros hay en total en la base de datos?
2. **Ejercicio de `MIN` y `MAX`**: ¿Cuál es el año de publicación más antiguo y el más reciente de todos los libros disponibles?
3. **Ejercicio de `YEAR`**: Utiliza la columna `fecha_prestamo` de la tabla `prestamos` para contar cuántos préstamos se realizaron en el año 2025.
4. **Ejercicio de `ORDER BY`**: Muestra la lista completa de libros, ordenada de forma descendente por el año de publicación.
5. **Ejercicio de `GROUP BY`**: Cuenta la cantidad de libros que ha escrito cada autor. Muestra el `id_autor` y el total de libros por cada uno.
6. **Ejercicio de `HAVING`**: Basado en el ejercicio anterior, encuentra el `id_autor` de aquellos autores que han escrito más de un libro.