Clase 2: Tipos de JOIN y Lenguajes SQL 📚

Bienvenidos a la segunda sesión de SQL. En la clase pasada, aprendimos a crear una base de datos y a manipular datos de forma individual. Hoy, daremos un gran salto y aprenderemos a combinar datos de diferentes tablas usando **JOIN** y a entender los diferentes lenguajes que componen SQL.

---

1. El Poder de Unir Tablas: Tipos de JOIN

Los JOIN son el corazón del modelo relacional. Nos permiten vincular dos o más tablas y extraer datos combinados. Veremos los cuatro tipos principales.

- **INNER JOIN**: La unión más común. Devuelve solo las filas que tienen una coincidencia en ambas tablas. Es como una intersección de conjuntos.
    - **Ejemplo**: Encontrar el título de los libros junto con el nombre del autor.

```
SELECT
  libros.titulo,
  autores.nombre,
  autores.apellido
FROM libros
INNER JOIN autores
  ON libros.id_autor = autores.id_autor;
```

- **LEFT JOIN**: Devuelve todas las filas de la tabla "izquierda" (la que aparece primero en la consulta), y las filas coincidentes de la tabla "derecha". Si no hay coincidencia, los valores de la tabla derecha serán NULL.
    - **Ejemplo**: Obtener todos los autores y, si tienen, el título de sus libros. Para esto, agregaremos un autor sin libros para ver cómo funciona el LEFT JOIN.

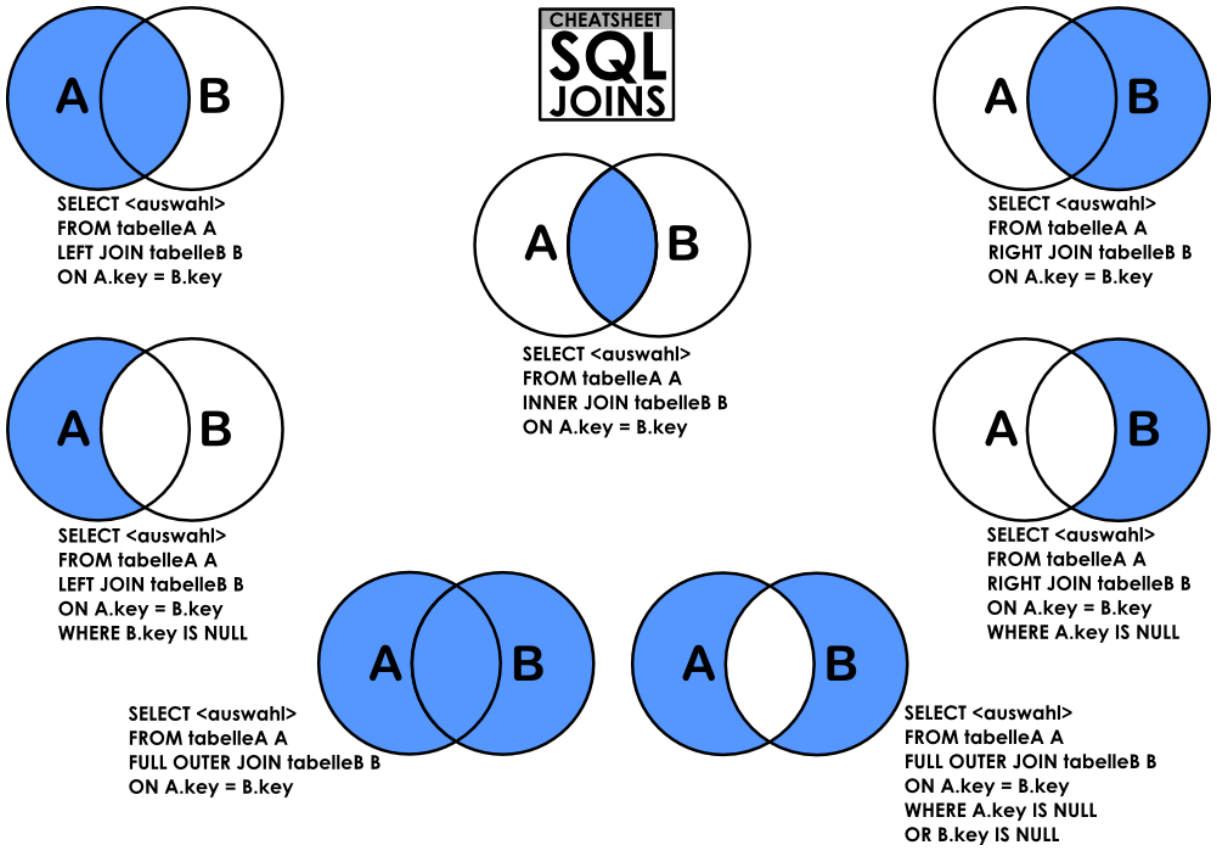
```
-- PASO 1: Agregar un autor sin libros para el ejemplo
INSERT INTO autores (nombre, apellido, nacionalidad) VALUES
('Stephen', 'Fry', 'Británica');
```

- **RIGHT JOIN**: Lo opuesto a LEFT JOIN. Devuelve todas las filas de la tabla "derecha" y las coincidentes de la izquierda. Si no hay coincidencia, los valores de la tabla izquierda serán NULL.
    - **Ejemplo**: Encontrar todos los libros y, si tienen, los datos del autor (sería igual a un INNER JOIN si todos los libros tienen autor).

```
SELECT
  libros.titulo,
  autores.nombre,
  autores.apellido
FROM libros
RIGHT JOIN autores
  ON libros.id_autor = autores.id_autor;
```

- **FULL OUTER JOIN**: Devuelve todas las filas cuando hay una coincidencia en una de las tablas. Combina los resultados de LEFT y RIGHT JOIN. En MySQL, se logra combinando un LEFT JOIN y un RIGHT JOIN con UNION.
    - **Ejemplo**: Obtener la lista completa de libros y la lista completa de autores, con sus coincidencias.

```
SELECT
  autores.nombre,
  autores.apellido,
  libros.titulo
FROM autores
LEFT JOIN libros
  ON autores.id_autor = libros.id_autor
UNION
SELECT
  autores.nombre,
  autores.apellido,
  libros.titulo
FROM autores
RIGHT JOIN libros
  ON autores.id_autor = libros.id_autor;
```



---

2. Los Lenguajes de SQL: DDL, DML y DCL

SQL no es un lenguaje único, sino que se divide en tres categorías o sublenguajes, cada uno con un propósito distinto.

- **DDL (Data Definition Language)**: **Define** la estructura de la base de datos. Piense en esto como el esqueleto. Se utiliza para crear, modificar o eliminar la estructura de los objetos de la base de datos.
    - **Comandos comunes**: `CREATE`, `ALTER`, `DROP`.
    - **Ejemplo**: El `CREATE TABLE` que usaste en la primera clase es un comando DDL.
- **DML (Data Manipulation Language)**: **Manipula** los datos dentro de la estructura. Es el lenguaje que usas para interactuar con la información.
    - **Comandos comunes**: `SELECT`, `INSERT`, `UPDATE`, `DELETE`.
    - **Ejemplo**: Todas las consultas que hiciste en los ejercicios de la clase 1 (`INSERT`, `UPDATE`, `DELETE`, `SELECT`) son comandos DML.
- **DCL (Data Control Language)**: **Controla** el acceso y los permisos de los usuarios sobre los datos. Es el lenguaje de seguridad.
    - **Comandos comunes**: `GRANT`, `REVOKE`.
    - **Ejemplo**: `GRANT SELECT ON libros TO 'usuario_biblioteca'@'localhost';` (Otorga permiso de lectura a un usuario). `REVOKE` hace lo contrario.

---

3. Agrupación y Filtrado de Grupos

Repasaremos y profundizaremos en `GROUP BY` y `HAVING`.

- **`GROUP BY`**: Agrupa filas con los mismos valores para calcular funciones de agregación. Es esencial para obtener resúmenes.
    - **Ejemplo**: Contar cuántos libros tiene cada autor.

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

- **`HAVING`**: Filtra los resultados después de que los grupos han sido creados. No puede ser reemplazado por `WHERE` cuando se usan funciones de agregación.
    - **Ejemplo**: Encontrar los autores que han escrito más de un libro.

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

4. Subconsultas: Consultas dentro de Consultas

Una **subconsulta** es una consulta `SELECT` anidada dentro de otra consulta. Son una herramienta muy poderosa y se pueden usar en diferentes partes de una *query*.

Tipos de Subconsultas

1. **Subconsultas en la cláusula `WHERE`**: Es la más común. Se usa para filtrar los resultados de la consulta principal basándose en los resultados de la subconsulta.
    - **Ejemplo 1**: Encuentra los libros que se publicaron en el mismo año que el libro "IT".

```
SELECT titulo, año_publicacion
FROM libros
WHERE año_publicacion = (
  SELECT año_publicacion
  FROM libros
```

```
  WHERE titulo = 'IT'
);
```

- **Ejemplo 2**: Encuentra los lectores que han prestado algún libro.

```
SELECT nombre, apellido
FROM lectores
WHERE id_lector IN (
  SELECT id_lector
  FROM prestamos
);
```

2. **Subconsultas en la cláusula `FROM`**: También conocidas como **tablas derivadas**. Se tratan como una tabla temporal y se usan para agrupar o resumir datos antes de unirlos o filtrarlos.
    - **Ejemplo**: Obtén el nombre de los autores que han escrito más de un libro.

```
SELECT nombre, apellido
FROM autores AS a
INNER JOIN (
  SELECT id_autor
  FROM libros
  GROUP BY id_autor
  HAVING COUNT(*) > 1
) AS autores_con_multiples_libros
ON a.id_autor = autores_con_multiples_libros.id_autor;
```

3. **Subconsultas en la cláusula `SELECT`**: Devuelven un valor único para cada fila de la consulta principal.
    - **Ejemplo**: Muestra el nombre de cada lector junto con la cantidad total de libros que ha prestado.

```
SELECT
  nombre,
  apellido,
  (
    SELECT COUNT(*)
    FROM prestamos
    WHERE prestamos.id_lector = lectores.id_lector
  ) AS total_prestamos
FROM lectores;
```

Subconsultas: Consultas dentro de Consultas

Una **subconsulta** es una consulta SELECT anidada dentro de otra consulta. Son una herramienta muy poderosa y se pueden usar en diferentes partes de una *query*.

Tipos de Subconsultas

1. **Subconsultas en la cláusula WHERE**: Es la más común. Se usa para filtrar los resultados de la consulta principal basándose en los resultados de la subconsulta.
    - **Ejemplo 1**: Encuentra los libros que se publicaron en el mismo año que el libro "IT".

```
SELECT titulo, año_publicacion
FROM libros
WHERE año_publicacion = (
  SELECT año_publicacion
  FROM libros
  WHERE titulo = 'IT'
);
```

Ejemplo 2: Encuentra los lectores que han prestado algún libro.

```
SELECT nombre, apellido
FROM lectores
WHERE id_lector IN (
  SELECT id_lector
  FROM prestamos
);
```

2. **Subconsultas en la cláusula FROM**: También conocidas como **tablas derivadas**. Se tratan como una tabla temporal y se usan para agrupar o resumir datos antes de unirlos o filtrarlos.
    - **Ejemplo**: Obtén el nombre de los autores que han escrito más de un libro.

```
SELECT nombre, apellido
FROM autores AS a
INNER JOIN (
  SELECT id_autor
  FROM libros
  GROUP BY id_autor
  HAVING COUNT(*) > 1
) AS autores_con_multiples_libros
ON a.id_autor = autores_con_multiples_libros.id_autor;
```

3. **Subconsultas en la cláusula `SELECT`**: Devuelven un valor único para cada fila de la consulta principal.
    - **Ejemplo**: Muestra el nombre de cada lector junto con la cantidad total de libros que ha prestado.

```
SELECT
  nombre,
  apellido,
  (
    SELECT COUNT(*)
    FROM prestamos
    WHERE prestamos.id_lector = lectores.id_lector
  ) AS total_prestamos
FROM lectores;
```

Diferencia entre Subconsultas y `JOIN`

Aunque a menudo se pueden usar para obtener resultados similares, las subconsultas y los `JOIN` no son lo mismo. La diferencia principal radica en su **enfoque y rendimiento**:

- **`JOIN`**: Está diseñado para **combinar datos** de dos o más tablas basándose en una relación de columnas. Es muy eficiente para este propósito. Piense en los `JOIN`s como la forma principal de vincular tablas. Es la herramienta preferida para unir tablas de forma horizontal.
- **Subconsulta**: Se utiliza para **filtrar o calcular datos** de una consulta principal, actuando como un dato de apoyo. La subconsulta se ejecuta primero para devolver un valor o un conjunto de valores, que luego son utilizados por la consulta principal. Las subconsultas son más versátiles, pero a menudo menos eficientes que un `JOIN` para unir grandes conjuntos de datos, ya que pueden ejecutar la subconsulta para cada fila de la consulta externa.

En resumen:

- **Usa `JOIN`** para combinar filas de dos o más tablas. Es más legible y, en la mayoría de los casos, más eficiente para este propósito.
- **Usa subconsultas** cuando necesites un valor o un conjunto de valores que no puedes obtener directamente de la consulta principal. Son ideales para filtrar datos usando un resultado dinámico o para obtener valores de agregación por separado.

---

5. Ejercicios Prácticos para la Clase 2

Aquí tienes una serie de ejercicios prácticos para que tus estudiantes apliquen los conceptos de `JOIN` y agrupaciones. Las consignas están diseñadas para que utilicen las tablas `autores`, `libros`, `lectores` y `prestamos` de la base de datos `biblioteca`.

Ejercicios de JOIN

1. **INNER JOIN**: Muestra el título de todos los libros junto con el nombre y apellido de su autor. Solo se deben mostrar los libros que tienen un autor asignado.
2. **LEFT JOIN**: Lista todos los autores de la base de datos y, si han escrito algún libro, muestra su título. Si un autor no tiene libros registrados, su título de libro debe aparecer como NULL.
3. **RIGHT JOIN**: Muestra todos los libros y, si tienen un autor, el nombre y apellido del mismo. Si hay autores que no han escrito libros, sus datos aparecerán como NULL.
4. **JOIN + WHERE**: Muestra el título de los libros que han sido prestados, junto con el nombre del lector que los tiene. La consulta debe mostrar solo los préstamos que aún no han sido devueltos (fecha_devolucion es NULL).

Ejercicios de Agrupación y Filtrado (GROUP BY y HAVING)

1. **GROUP BY + COUNT**: Para cada lector, cuenta la cantidad total de libros que ha prestado. La salida debe mostrar el id_lector y la cantidad de préstamos.
2. **GROUP BY + AVG**: Calcula el promedio de préstamos por lector.
3. **GROUP BY + HAVING**: Encuentra los id_lector de aquellos lectores que han tomado prestados más de un libro en total.
4. **GROUP BY + HAVING (con JOIN de apoyo)**: Muestra el nombre completo de los lectores que han tomado prestado al menos un libro, junto con el total de libros que tienen en préstamo.

Ejercicios de DML, DDL y DCL

1. **DDL (ALTER)**: En la tabla autores, agrega una nueva columna llamada fecha_nacimiento de tipo DATE.
2. **DML (UPDATE)**: Actualiza la fecha_nacimiento del autor 'Gabriel García Márquez' a '1927-03-06'.
3. **DCL (GRANT)**: Crea un usuario llamado lector_anonimo con la contraseña 12345 y otórgale permisos para seleccionar datos de la tabla libros.
4. **DCL (REVOKE)**: Revoca el permiso de lectura (SELECT) sobre la tabla libros al usuario lector_anonimo.

---

6. Datos para la Base de Datos "Biblioteca"

Para los ejercicios, necesitarás una base de datos con información. Aquí están las queries para poblar las tablas con datos de ejemplo.

```
-- Elimina la base de datos 'biblioteca' si ya existe para crearla desde cero.
DROP DATABASE IF EXISTS biblioteca;

-- Crea la base de datos
CREATE DATABASE biblioteca;
USE biblioteca;

-- Tabla de Autores
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
CREATE TABLE libros (
 id_libro INT PRIMARY KEY AUTO_INCREMENT,
 titulo VARCHAR(255) NOT NULL,
 isbn VARCHAR(13) UNIQUE NOT NULL,
 año_publicacion YEAR,
 id_autor INT NOT NULL,
 FOREIGN KEY (id_autor) REFERENCES autores(id_autor) ON DELETE RESTRICT
);

-- Tabla de Lectores
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
);

-- Tabla de Préstamos
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,
 fecha_devolucion DATE,
 FOREIGN KEY (id_libro) REFERENCES libros(id_libro) ON DELETE CASCADE,
 FOREIGN KEY (id_lector) REFERENCES lectores(id_lector) ON DELETE CASCADE,
 CHECK (fecha_devolucion IS NULL OR fecha_devolucion >= fecha_prestamo)
);

-- POBAR LAS TABLAS

-- Autores
INSERT INTO autores (nombre, apellido, nacionalidad) VALUES
('Stephen', 'King', 'Estadounidense'),
('Agatha', 'Christie', 'Británica'),
('Jules', 'Verne', 'Francesa'),
('Antoine de', 'Saint-Exupéry', 'Francesa'),
('Edgar Allan', 'Poe', 'Estadounidense'),
('Isaac', 'Asimov', 'Estadounidense'),
('Gabriel', 'García Márquez', 'Colombiana'),
('Jane', 'Austen', 'Británica'),
('Haruki', 'Murakami', 'Japonés'),
('Herman', 'Melville', 'Estadounidense'),
```

```
('Stephen', 'Fry', 'Británica');

-- Libros
INSERT INTO libros (titulo, isbn, año_publicacion, id_autor) VALUES
('IT', '978-0451419736', 1986, 1),
('El Resplandor', '978-0385121675', 1977, 1),
('Asesinato en el Orient Express', '978-0062073491', 1934, 2),
('Diez negritos', '978-0062073484', 1939, 2),
('La vuelta al mundo en ochenta días', '978-0007421189', 1872, 3),
('Viaje al centro de la Tierra', '978-0140449279', 1864, 3),
('El Principito', '978-3125732159', 1943, 4),
('El cuervo', '978-0679782414', 1845, 5),
('Yo, robot', '978-0553294385', 1950, 6),
('Fundación', '978-0553805977', 1951, 6),
('El amor en los tiempos del cólera', '978-0307389731', 1985, 7),
('Orgullo y prejuicio', '978-0141439518', 1813, 8),
('Kafka en la orilla', '978-0307275362', 2002, 9),
('Moby Dick', '978-0142437247', 1851, 10);

-- Lectores
INSERT INTO lectores (nombre, apellido, email) VALUES
('Ana', 'García', 'ana.garcia@email.com'),
('Luis', 'Ramírez', 'luis.ramirez@email.com'),
('Sofía', 'Fernández', 'sofia.fernandez@email.com'),
('Daniel', 'Torres', 'daniel.torres@email.com'),
('Valeria', 'Ruiz', 'valeria.ruiz@email.com'),
('Pablo', 'Jiménez', 'pablo.jimenez@email.com'),
('Laura', 'Vargas', 'laura.vargas@email.com');

-- Préstamos
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'),
(5, 3, '2025-08-10', NULL),
(2, 4, '2025-08-15', '2025-08-20'),
(4, 5, '2025-07-28', '2025-08-10'),
(6, 6, '2025-08-05', '2025-08-20'),
(7, 7, '2025-08-12', NULL),
(8, 1, '2025-08-18', NULL),
(9, 2, '2025-08-19', NULL),
(10, 3, '2025-08-17', NULL),
(1, 5, '2025-08-14', NULL),
(3, 7, '2025-08-11', '2025-08-19'),
(11, 4, '2025-08-10', '2025-08-18'),
(12, 6, '2025-08-15', NULL),
(13, 1, '2025-08-16', NULL),
(14, 2, '2025-08-17', '2025-08-20');
```

ACID

Propiedades que garantizan fiabilidad e integridad de las transacciones en una db.
Cada letra de ACID corresponde a una propiedad que las bd relacionales deben cumplir.

A: Atomicidad-Atomicity:
Todo o nada. Una transacción se completa o no se realiza.
Si una parte de la transacción falla, todo el proceso debe revertirse (rollback).
Rollback: La base vuelve al estado en que estaba antes de que la transacción comenzará.

C: Consistencia-Consistency:
Asegura que una transacción lleve a la db de un estado valido a otro estado valido.
Respeta las reglas, restricciones y triggers definidos.
Después de toda transacción la db va a estar en un estado coherente, donde sus reglas siguen cumpliendose.

I: Isolation-Aislamiento:
Las transacciones se realizan de forma independiente unas de otras.
De forma secuencial aunque se ejecuten de forma concurrente.
Los cambios de una transacción no están disponibles para otras transacciones hasta que esta no se haya completado. Evita inconsistencias temporales.

D: Durabilidad-Durability:
Una vez que una transacción fue confirmada (commit) los cambios son permanentes incluso si ocurre un fallo.
Fallos: corte de energía o caídas de servidor.
Usando logs o registros podemos almacenar y recuperar las transacciones confirmadas.

Persistencia - Guardado y almacenamiento de los datos.