Clase 3: Consultas Avanzadas y Conectividad 📚

Bienvenidos a la última clase de SQL. En esta sesión, llevaremos nuestras habilidades al siguiente nivel, combinando todo lo aprendido para resolver problemas complejos y conectar nuestra base de datos con una aplicación web real.

---

1. Ejercicios de Consultas Avanzadas con `JOIN`

Para esta clase, desafiaremos a los estudiantes con ejercicios que combinan múltiples tablas y funciones de agregación. Aquí están los enunciados y las soluciones:

- **Libros con más préstamos**: Muestra el título de los 5 libros que han sido prestados más veces, junto con la cantidad total de préstamos para cada uno.

```sql
SELECT
  l.titulo,
  COUNT(p.id_prestamo) AS total_prestamos
FROM libros AS l
INNER JOIN prestamos AS p
  ON l.id_libro = p.id_libro
GROUP BY
  l.titulo
ORDER BY
  total_prestamos DESC
LIMIT 5;
```

- **Libros más solicitados y sus lectores**: ¿Cuáles son los libros más pedidos y quiénes los solicitaron? Muestra solo los títulos de los libros que han sido prestados al menos 3 veces.

```sql
SELECT
  l.titulo,
  GROUP_CONCAT(CONCAT(r.nombre, ' ', r.apellido) SEPARATOR ', ') AS lectores,
  COUNT(p.id_prestamo) AS total_solicitudes
FROM libros AS l
INNER JOIN prestamos AS p
  ON l.id_libro = p.id_libro
INNER JOIN lectores AS r
  ON p.id_lector = r.id_lector
GROUP BY
  l.titulo
HAVING
  total_solicitudes >= 3
ORDER BY
  total_solicitudes DESC;
```

- **Lectores que leyeron más de dos libros del mismo autor**: Muestra el nombre de los lectores que han tomado prestados más de dos libros de un mismo autor.

```
SELECT
  r.nombre,
  r.apellido,
  a.nombre AS nombre_autor
FROM lectores AS r
INNER JOIN prestamos AS p
  ON r.id_lector = p.id_lector
INNER JOIN libros AS l
  ON p.id_libro = l.id_libro
INNER JOIN autores AS a
  ON l.id_autor = a.id_autor
GROUP BY
  r.id_lector, a.id_autor
HAVING
  COUNT(l.id_libro) > 2;
```

- **Épocas de mayor demanda de libros**: Determina en qué años se realizaron la mayor cantidad de préstamos.

```
SELECT
  YEAR(fecha_prestamo) AS año,
  COUNT(id_prestamo) AS total_prestamos
FROM prestamos
GROUP BY
  año
ORDER BY
  total_prestamos DESC;
```

- **Libros con préstamos más largos**: Identifica el título de los 5 libros que pasaron más tiempo prestados.

```
SELECT
  l.titulo,
  DATEDIFF(p.fecha_devolucion, p.fecha_prestamo) AS dias_prestado
FROM libros AS l
INNER JOIN prestamos AS p
  ON l.id_libro = p.id_libro
WHERE
  p.fecha_devolucion IS NOT NULL
ORDER BY
  dias_prestado DESC
LIMIT 5;
```

---

2. Conceptos Clave: Comodines

Comodines en SQL (`LIKE`)

Los comodines son caracteres especiales que se usan con la cláusula `LIKE` para buscar patrones de texto en cadenas.

- `%` **(porcentaje)**: Representa cero o más caracteres.
  - `LIKE 'A%'`: Busca textos que **empiezan con** 'A'.
  - `LIKE '%ez'`: Busca textos que **terminan en** 'ez'.
  - `LIKE '%en%'`: Busca textos que **contienen** 'en'.
- `_` **(guión bajo)**: Representa un solo carácter.
  - `LIKE '_o%'`: Busca textos cuya **segunda letra** es 'o'.

---

3. Conectando SQL con JavaScript: Un Proyecto Básico de Express

Para mostrar cómo se aplican estas consultas en un entorno real, te presento un proyecto básico de Express para crear una API que se conecta a la base de datos `biblioteca`.

- **Objetivo**: Crear un servidor web con Express que ejecuta consultas SQL y devuelve los resultados en formato **JSON**.
- **Requerimientos**: `express`, `mysql2` y `cors`.
- **Práctica recomendada**: **Centralizar las consultas SQL** en un archivo de constantes para mejorar la organización y la legibilidad del código.

Estructura del proyecto:

```
/express-sql
|-- index.js  (Servidor principal)
|-- db.js     (Módulo de conexión a la base de datos)
|-- queries.js (Archivo para guardar las consultas SQL)
|-- package.json
```

Ejemplo de código (`index.js`):

JavaScript

```
// index.js
const express = require('express');
const cors = require('cors');
const app = express();
const db = require('./db');
const { authorQueries } = require('./queries');

app.use(express.json());
app.use(cors());

app.get('/autores', (req, res) => {
```

```
  db.query(authorQueries.getAllAuthors, (err, results) => {
    if (err) {
      return res.status(500).send('Error');
    }
    res.json(results);
  });
});

app.listen(3000, () => {
  console.log(`Servidor en http://localhost:3000`);
});
```

---

4. Preguntas Clave y Conceptos Adicionales

- **Deadlock**: Es un problema en bases de datos donde dos o más transacciones se bloquean mutuamente, impidiendo que cualquiera de ellas avance. Los DBMS modernos detectan y resuelven esto abortando una de las transacciones.
- **Out of Range Column**: Este error ocurre al intentar insertar un valor que no cabe en el tipo de dato de la columna. Por ejemplo, en la columna `año_publicacion` con tipo `YEAR` (rango limitado), no se puede insertar un año como 1813. La solución es cambiar el tipo de dato a `SMALLINT` o `INT`.
- **localhost**: No es la conexión, sino el **host** o la dirección desde donde se permite la conexión. Se refiere a la propia computadora (IP `127.0.0.1`), asegurando que la conexión sea local y segura.