Solucionario: Interacción Node.js + MySQL (Fauna DB)

Guía de Implementación Profesional

1. Bloque 1: Operaciones CRUD Básicas

```javascript
// 1. Conexion básica y verificación
const mysql = require('mysql2/promise');
const pool = mysql.createPool({
    host: 'localhost', user: 'root', password: '', database: 'fauna_db'
});

// 2. Insertar clase Amphibia
await db.query('INSERT INTO clases (nombre) VALUES (?)', ['Amphibia']);

// 3. Recuperar todos los habitats
const [habitats] = await db.query('SELECT * FROM habitats');

// 4. Insertar tres familias (Bulk insert)
const familias = [['Canidae', 1], ['Hominidae', 1], ['Equidae', 1]];
await db.query('INSERT INTO familias (nombre, clase_id) VALUES ?', [familias]);

// 5. Update nombre cientifico
await db.query('UPDATE especies SET nombre_cientifico = ? WHERE nombre_comun = ?',
    ['Panthera tigris tigris', 'Tigre']);

// 6. Eliminar clima Árido
await db.query('DELETE FROM habitats WHERE clima = ?', ['Árido']);

// 7. Filtrado poblacion > 10.000
const [pob] = await db.query('SELECT nombre_comun FROM especies WHERE poblacion_estimada > 10000');

// 8. Búsqueda con LIKE
const [conA] = await db.query('SELECT * FROM especies WHERE nombre_comun LIKE ?', ['%a%']);

// 9. Ordenación descendente
const [fam] = await db.query('SELECT * FROM familias WHERE id > 2 ORDER BY nombre ASC');

// 10. Count total
const [[{total}]] = await db.query('SELECT COUNT(*) as total FROM especies');
```

2. Bloque 2: Consultas Relacionales (JOINS)

```javascript
// 11. Inner Join Especie - Familia
const [res11] = await db.query(`
    SELECT e.nombre_comun, f.nombre as familia
    FROM especies e JOIN familias f ON e.familia_id = f.id`);

// 12. Triple Join (Especie -> Familia -> Clase)
const [res12] = await db.query(`
    SELECT e.nombre_comun, f.nombre as familia, c.nombre as clase
    FROM especies e
    JOIN familias f ON e.familia_id = f.id
    JOIN clases c ON f.clase_id = c.id`);

// 13. Promedio poblacion Mamiferos (Clase ID 1)
const [[{promedio}]] = await db.query(`
    SELECT AVG(e.poblacion_estimada) as promedio
    FROM especies e JOIN familias f ON e.familia_id = f.id WHERE f.clase_id = 1`);

// 14. Habitats de una especie (Muchos a Muchos)
const [res14] = await db.query(`
    SELECT h.nombre FROM habitats h
    JOIN especie_habitat eh ON h.id = eh.habitat_id
    WHERE eh.especie_id = ?`, [1]);

// 15. Población máxima y mínima
```

```
const [[stats]] = await db.query(`
    SELECT MAX(poblacion_estimada) as maximo, MIN(poblacion_estimada) as minimo
    FROM especies`);
console.log(`Máximo: ${stats.maximo}, Mínimo: ${stats.minimo}`);

// 16. Group By: Especies por familia
const [res16] = await db.query(`
    SELECT f.nombre, COUNT(e.id) as num_especies
    FROM familias f LEFT JOIN especies e ON f.id = e.familia_id GROUP BY f.id`);

// 17. Filtrado con HAVING
const [clases] = await db.query(`
    SELECT c.nombre, COUNT(f.id) as total_familias
    FROM clases c
    JOIN familias f ON c.id = f.clase_id
    GROUP BY c.id
    HAVING total_familias > 2`);

// 18. LEFT JOIN y detección de nulos
const [huerfanos] = await db.query(`
    SELECT e.nombre_comun
    FROM especies e
    LEFT JOIN especie_habitat eh ON e.id = eh.especie_id
    WHERE eh.habitat_id IS NULL`);

// 19. Manipulación de strings (UPPER y LENGTH)
const [nombres] = await db.query(`
    SELECT UPPER(nombre_cientifico) as nombre_up, LENGTH(nombre_cientifico) as longitud
    FROM especies`);

// 20. Concatenación de cadenas
const [listado] = await db.query(`
    SELECT CONCAT(nombre_comun, ' (', nombre_cientifico, ')') as ficha_tecnica
    FROM especies`);
```

---

3. Bloque 3: Seguridad y Arquitectura

```
// 21. Manejo de errores con try-catch
try {
    const [rows] = await db.query('SELECT * FROM tabla_inexistente');
} catch (error) {
    console.error('Error en la consulta:', error.message);
// Aquí se gestionaría el error (ej. enviar logs o respuesta 500)
}

// 22. Aprovechamiento del pool de conexiones
// El pool gestiona las conexiones automáticamente
const [res1, res2, res3] = await Promise.all([
    db.query('SELECT * FROM clases'),
    db.query('SELECT * FROM familias'),
    db.query('SELECT * FROM habitats')
]);

// 23. Prepared statements para evitar inyección de SQL
const idBuscado = 5;
// El driver se encarga de escapar el valor para evitar SQL Injection
const [especie] = await db.execute('SELECT * FROM especies WHERE id = ?', [idBuscado]);

```

---

4. Bloque 4: Express e Interacción con el Cliente

```
// 24. API: Listar Clases
app.get('/api/clases', async (req, res) => {
    const [rows] = await db.query('SELECT * FROM clases');
    res.json(rows);
});

// 25. API: Filtrar especies por Familia
app.get('/api/especies/:familiaId', async (req, res) => {
    const [rows] = await db.query('SELECT * FROM especies WHERE familia_id = ?', [req.params.familiaId]);
    res.json(rows);
});

// 26, 27 y 28. Lógica del Cliente (Frontend)
```

```
/*
HTML:
<select id="selFam"></select>
<input type="text" id="buscador" placeholder="Buscar especie...">
<table id="res"></table>
*/

// Cargar select al inicio
async function init() {
    const fams = await (await fetch('/api/familias')).json();
    document.getElementById('selFam').innerHTML =
        fams.map(f => `<option value="${f.id}">${f.nombre}</option>`).join('');
}

// Escuchar cambios para actualizar tabla
document.getElementById('selFam').addEventListener('change', async (e) => {
    const esp = await (await fetch(`/api/especies/${e.target.value}`)).json();
    document.getElementById('res').innerHTML = esp.map(a => `<tr><td>${a.nombre_comun}</td></tr>`).join('');
});

// Buscador en tiempo real (Ejercicio 28)
document.getElementById('buscador').addEventListener('input', async (e) => {
    const query = e.target.value;
    if(query.length < 2) return;
    const res = await fetch(`/api/especies/search?q=${query}`);
    const data = await res.json();
    // Actualizar vista...
});

// 29. Inserción POST
app.post('/api/especies', async (req, res) => {
    // Desestructuramos los campos recibidos en el JSON del body
    const { nombre_comun, nombre_cientifico, familia_id, estado_conservacion, poblacion_estimada } = req.body;

    const sql = `INSERT INTO especies
    (nombre_comun, nombre_cientifico, familia_id, estado_conservacion, poblacion_estimada)
    VALUES (?, ?, ?, ?, ?)`;

    try {
        const [result] =
          await db.query(sql, [nombre_comun, nombre_cientifico, familia_id, estado_conservacion, poblacion_estimada]);
        // Respondemos con el ID generado y estado 201 (Created)
        res.status(201).json({ id: result.insertId, mensaje: "Registro creado con éxito" });
    } catch (err) {
        res.status(500).json({ error: "No se pudo insertar el registro" });
    }
});

// 30. Actualización con PUT
app.put('/api/especies/:id', async (req, res) => {
    const id = req.params.id;
    const { estado_conservacion, poblacion_estimada } = req.body;
    const sql = "UPDATE especies SET estado_conservacion = ?, poblacion_estimada = ? WHERE id = ?";
    try {
        const [result] = await db.query(sql, [estado_conservacion, poblacion_estimada, id]);
        if (result.affectedRows === 0) {
            return res.status(404).json({ mensaje: "Especie no encontrada" });
        }
        res.json({ mensaje: "Especie actualizada correctamente" });
    } catch (err) {
        res.status(500).json({ error: "Error al actualizar los datos" });
    }
});
```