TECNOLÓGICA DEL ORIENTE
INSTITUCIÓN DE EDUCACIÓN SUPERIOR

DESARROLLO WEB

INTERACCIÓN CON BASES DE DATOS EN APLICACIONES WEB

INTERACCIÓN CON BASES DE DATOS EN APLICACIONES WEB: USO DE ORMS Y DRIVERS

La interacción entre aplicaciones web y bases de datos se facilita mediante Object-Relational Mapping (ORM) frameworks y drivers especializados que abstraen las complejidades de la comunicación directa con los sistemas de base de datos. Los ORMs como Sequelize para Node.js, SQLAlchemy para Python, o Mongoose para MongoDB, proporcionan una capa de abstracción que permite a los desarrolladores interactuar con bases de datos utilizando objetos y métodos del lenguaje de programación en lugar de escribir SQL o consultas específicas de base de datos directamente. Esta abstracción mejora la productividad del desarrollo, reduce errores de programación y facilita el mantenimiento del código.

Los ORMs ofrecen múltiples beneficios incluyendo protección automática contra inyección SQL, gestión de conexiones de base de datos, migración de esquemas automatizada, validación de datos y caché automático de consultas. Sin embargo, también introducen sobrecarga de rendimiento y pueden generar consultas SQL ineficientes si no se configuran correctamente. Los drivers nativos, por otro lado, proporcionan control total sobre las consultas y optimización de rendimiento, pero requieren mayor experticia en SQL y gestión manual de aspectos como pooling de conexiones y validación de datos. Granados La Paz (2023), destaca que las aplicaciones web del entorno servidor deben balancear productividad de desarrollo con optimización de rendimiento al seleccionar estrategias de acceso a datos.

En proyectos reales de ingeniería de software, la selección entre ORMs y drivers nativos depende de factores como la complejidad de las consultas, requisitos de rendimiento, experiencia del equipo y tiempo de desarrollo disponible. Aplicaciones con consultas simples CRUD frecuentemente se benefician de ORMs por su rapidez de desarrollo, mientras que sistemas que requieren consultas complejas u optimización extrema de rendimiento pueden preferir drivers nativos. Muchas aplicaciones adoptan enfoques híbridos utilizando ORMs para la mayoría de las operaciones y drivers nativos para consultas críticas de rendimiento.

Ejercicio práctico. Implementar la misma funcionalidad usando ORM (Sequelize) y driver nativo para Node.js.

Paso 1. Configurar Sequelize: const { Sequelize, DataTypes } = require('sequelize'); const sequelize = new Sequelize('database', 'username', 'password', {host: 'localhost', dialect: 'mysql'}); const Usuario = sequelize.define('Usuario', {nombre: DataTypes.STRING, email: DataTypes.STRING}); await sequelize.sync();.

Paso 2. Operaciones con ORM: // Crear usuario: const nuevoUsuario = await Usuario.create({nombre: 'Juan', email: 'juan@email.com'}); // Buscar usuarios: const usuarios = await Usuario.findAll({where: {nombre: {[Op.like]: '%Juan%'}}}); // Actualizar: await Usuario.update({email: 'nuevo@email.com'}, {where: {id: 1}});.

Paso 3. Implementar con driver nativo: const mysql = require('mysql2/promise'); const connection = await mysql.createConnection({host: 'localhost', user: 'username', password: 'password', database: 'database'}); // Crear usuario:

await connection.execute('INSERT INTO usuarios (nombre, email) VALUES (?, ?)', ['Juan', 'juan@email.com']); // Buscar: const [rows] = await connection.execute('SELECT * FROM usuarios WHERE nombre LIKE ?', ['%Juan%']);.

Paso 4. Comparar rendimiento y facilidad de uso.

Resultado. Se muestran las diferencias en productividad de desarrollo, rendimiento y control entre ORMs y drivers nativos, proporcionando criterios para la selección apropiada, según el contexto del proyecto.