## **Sección Teórica — Base de Datos Oracle**

### **🔹 ¿Qué es una base de datos relacional?**

Las bases de datos relacionales son aquellas que organizan los datos en tablas que pueden relacionarse entre sí. Las relaciones se dan a través de claves. En estas, las filas representan los registros y las columnas representan atributos.

### **🔹 ¿Qué es una clave primaria y una clave foránea en Oracle?**

* **Clave primaria:** Identifica de forma única cada una de las filas de una tabla. No puede ser nula (not nullable) y debe ser única (unique).
* **Clave foránea:** Es el campo que referencia la clave primaria de otra tabla.

### **🔹 Escribe una consulta SQL para seleccionar todos los registros de una tabla llamada cliente que vivan en la ciudad de Guayaquil y Quito.**

SELECT \* FROM cliente

WHERE ciudad IN ('Guayaquil', 'Quito');

### **🔹 Explica la diferencia entre INNER JOIN, LEFT JOIN y RIGHT JOIN.**

* INNER JOIN: Devuelve sólo las coincidencias entre ambas tablas.
* LEFT JOIN: Devuelve todos los registros de la tabla izquierda junto a los que coinciden de la tabla derecha.
* RIGHT JOIN: Devuelve todos los registros de la tabla derecha junto a los que coinciden de la tabla izquierda.

### **🔹 ¿Qué es un índice en Oracle y cómo mejora el rendimiento?**

Un índice es una estructura que mejora la velocidad de las consultas ya que permite búsquedas rápidas en columnas específicas. Se utiliza usualmente para evitar escaneos completos de tabla que toman mucho más tiempo.

### **🔹 Escribe una consulta SQL que devuelva los 5 clientes con mayor saldo en la tabla facturas.**

SELECT cliente\_id, nombre, saldo

FROM facturas

ORDER BY saldo DESC

FETCH FIRST 5 ROWS ONLY;

### **🔹 Explicar y grafique un ejemplo de normalización de una base de datos de facturación en tercera forma normal (3FN). Cite las entidades Clientes, Productos y Facturas (al menos defina 3 campos por tabla).**

* **Clientes:** id\_cliente, nombre, email
* **Productos:** id\_producto, nombre, precio
* **Facturas:** id\_factura, id\_cliente, fecha

**Ejemplo en 3FN:**

* Cada tabla tiene clave primaria.
* No hay valores repetidos innecesarios.
* Cada campo depende solo de la clave primaria.

Diagrama:

****

### **🔹 ¿Cómo funcionan los triggers en Oracle? Escribe un ejemplo de un trigger que registre cada vez que se inserta un nuevo usuario en la tabla usuarios.**

Un TRIGGER es un bloque PL/SQL que se ejecuta automáticamente ante ciertos eventos (INSERT, UPDATE, DELETE).

**Ej:**

CREATE OR REPLACE TRIGGER log\_insert\_usuario

AFTER INSERT ON usuarios

FOR EACH ROW

BEGIN

INSERT INTO log\_usuarios (id\_usuario, fecha)

VALUES (:NEW.id, SYSDATE);

END;

### **🔹 Explica cómo optimizar una consulta compleja en Oracle y qué herramientas utilizarías para analizar su rendimiento.**

**Estrategias:**

* Crear índices.
* Evitar SELECT \*.
* Usar EXISTS en lugar de IN si hay muchas filas.

**Herramientas:**

* EXPLAIN PLAN
* Oracle SQL Developer
* AWR Reports, SQL Trace

## **Sección Teórica — Java**

### **🔹 ¿Qué diferencia hay entre una variable int y una variable Integer en Java?**

* int: dato de tipo primitivo lo que implica mayor eficiencia.
* Integer: ya no es primitivo, sino una clase Wrapper, permite usar métodos y trabajar con colecciones como List<Integer>.

### **🔹 Explica qué es la herencia en Java y da un ejemplo.**

Es un mecanismo donde una clase hija, como lo dice su nombre, hereda atributos y métodos de una clase padre.

**Ejemplo:**

class Animal {

void sonido() { System.out.println("Hace sonido"); }

}

class Perro extends Animal {

void sonido() { System.out.println("Ladra"); }

}

### **🔹 ¿Para qué sirve la palabra clave final en Java?**

* **en una Variable:** indica que no puede cambiar su valor.
* **en un Método:** indica que no puede ser sobrescrito.
* **en una Clase:** indica que no puede ser heredada.

### 

### 

### 

### 

### **🔹 ¿Cuál es la diferencia entre una interfaz y una clase abstracta en Java?**

Una interfaz define un conjunto de métodos que una clase debe implementar, actuando como un contrato de comportamiento. Mientras que la clase abstracta puede contener tanto métodos abstractos (sin implementación) como métodos concretos (con implementación), y también actúa como una plantilla para otras clases.

### **🔹 Explica el concepto de try-catch-finally en el manejo de excepciones en Java.**

Se usa para manejar excepciones y definir el comportamiento de la aplicación en caso de haber una excepción:

try {

int x = 1 / 0;

} catch (ArithmeticException e) {

System.out.println("Error: " + e.getMessage());

} finally {

System.out.println("Se ejecuta siempre");

}

### **🔹 Implementa una función en Java que reciba una lista de enteros y devuelva otra lista con los números ordenados de menor a mayor.**

public List<Integer> ordenar(List<Integer> lista) {

Collections.sort(lista);

return lista;

}

### **🔹 ¿Qué es el patrón de diseño Singleton y cómo lo implementarías en Java?**

Es un patrón que asegura que una clase tenga una única instancia y proporciona un punto de acceso global a esta instancia:

public class Singleton {

private static Singleton instancia;

private Singleton() {}

public static Singleton getInstancia() {

if (instancia == null) {

instancia = new Singleton();

}

return instancia;

}

}

### **🔹 Explica qué es el Garbage Collector en Java y cómo funciona.**

Es un proceso automático de Java que libera memoria eliminando objetos que ya no se usan, que se ejecuta en segundo plano.

### **🔹 Implementa un servicio REST en Spring Boot que devuelva una lista de usuarios en formato JSON.**

@RestController

@RequestMapping("/usuarios")

public class UsuarioController {

@GetMapping

public List<Usuario> listar() {

return usuarioService.findAll();

}

}

### **🔹 ¿Cómo funciona la anotación @Transactional en Spring Boot y qué problemas podría generar su uso incorrecto?**

Maneja transacciones en métodos. Si ocurre una excepción, se hace rollback, volviendo al estado anterior.

**Si se usa mal:**

* No se hace rollback por excepciones no verificadas.
* Métodos llamados dentro de la misma clase pueden no aplicar transacción.

## **Sección Teórica — Node.js**

### **🔹 ¿Qué es Node.js y en qué se diferencia de otros entornos de ejecución?**

Es un entorno de ejecución de JavaScript del lado del servidor. Usa el motor V8 de Chrome. Es no bloqueante y orientado a eventos.

### **🔹 ¿Qué es npm y para qué se utiliza?**

NPM significa Node Package Manager. Se usa para instalar dependencias (librerías) en proyectos.

### **🔹 Escribe un script en Node.js que imprima "Hola, Mundo" en la consola.**

console.log("Hola, Mundo");

### **🔹 Explica qué es el event loop en Node.js.**

El event loop permite que Node.js maneje múltiples operaciones asincrónicas en un hilo usando una cola de eventos.

### **🔹 ¿Cómo manejarías errores en una aplicación con Express.js?**

app.use((err, req, res, next) => {

res.status(500).json({ error: err.message });

});

### **🔹 Escribe un código en Node.js que cree un servidor HTTP básico con Express.**

const express = require('express');

const app = express();

app.get('/', (req, res) => res.send("Hola, Mundo"));

app.listen(3000);

### **🔹 ¿Cómo manejarías autenticación y autorización en una API desarrollada con Node.js?**

* **Para Autenticación utilizaría:** Validar identidad (login, JWT, etc.).
* **Para Autorización:** Permitir acceso a recursos según rol (admin, user, etc.).

### **🔹 Explica las diferencias entre callbacks, promesas y async/await en** [**Node.js**](http://node.js)**.**

Un callback es una función que se pasa como argumento a otra función y se ejecuta cuando una tarea asincrónica termina.

// Ejemplo de un Callback

fs.readFile('file.txt', (err, data) => {});

Una promesa representa un valor que puede estar disponible en el momento, en el futuro o nunca.

// Ejemplo de Promesa

fetch('/api').then(res => res.json());

Async/await es una sintaxis que simplifica el uso de promesas escribiendo código asincrónico de manera similar al sincrónico

// Ejemplo de Async/Await

async function getData() {

const res = await fetch('/api');

}

## **Sección Teórica — Angular**

### **🔹 ¿Qué es Angular y para qué se utiliza?**

Framework frontend de Google para construir SPA (Single Page Applications) o aplicaciones de una sola página usando TypeScript.

### **🔹 ¿Qué es un componente en Angular?**

Un componente es la únidad básica en Angular. Incluye HTML, CSS y lógica en TypeScript.

@Component({

selector: 'app-ejemplo',

templateUrl: './ejemplo.component.html'

})

### **🔹 Explica qué es ngModel y cómo se usa en Angular.**

El ngModel es una directiva de enlace en ambos sentidos entre la vista y el modelo.

<input [(ngModel)]="nombre">

donde nombre se puede visualizar en la vista como el contenido del input, y támbien se encuentra en memoria en el modelo.

### **🔹 Explica la diferencia entre @Input() y @Output() en Angular.**

* @Input(): recibe datos del componente padre.
* @Output(): emite eventos al componente padre.

### **🔹 ¿Cómo se maneja el enrutamiento en Angular?**

El enrutamiento se configura generalmente en el archivo app-routing.module.ts con rutas y componentes.

Ej:

{ path: 'clientes', component: ClienteListComponent }

### **🔹 Escribe un servicio en Angular que realice una petición HTTP a una API.**

getClientes(): Observable<Cliente[]> {

return this.http.get<Cliente[]>('/api/clientes');

}

### **🔹 Explica qué es el ciclo de vida de un componente en Angular y sus métodos principales.**

Métodos principales:

* ngOnInit(): se ejecuta al iniciar.
* ngOnChanges(): se ejecuta al cambiar inputs.
* ngOnDestroy(): al destruirse el componente.

### **🔹 ¿Cómo manejarías el estado global en una aplicación Angular?**

La opción más completa a mi parecer es NgRx (o Redux para Angular).