Repaso Acceso a Datos — Examen (sin Spring/Hibernate)

Objetivo: entender el flujo App → DAO → JDBC → BD + intercambio con CSV/JSON/XML

Mentalidad: no memorizar; reconstruir el flujo y practicar con código.

1. Enfoque general

Acceso a Datos trata de **mover datos con seguridad y coherencia**. Entiende las capas y cómo se comunican: **Aplicación** → **DAO** → **JDBC** → **BD**, y también la lectura/escritura de **ficheros** (CSV, JSON, XML).

2. Arquitectura por capas y patrón DAO

Separa la lógica de negocio de la persistencia. Un **DAO** (Data Access Object) expone métodos CRUD para una entidad sin que el resto de la app sepa SQL. Beneficios: testabilidad, mantenimiento y posibilidad de cambiar tecnología sin romper la app.

```
// Interfaz DAO
public interface AlumnoDAO {
    Optional<Alumno> findById(long id);
    List<Alumno> findAll();
    long insert(Alumno a);
    boolean update(Alumno a);
    boolean delete(long id);
}
```

3. SQL esencial para el examen

Consultas básicas (SELECT / INSERT / UPDATE / DELETE):

```
-- SELECT con filtro, orden y límite
SELECT id, nombre, edad
FROM alumnos
WHERE edad >= ?
ORDER BY nombre ASC
LIMIT 10;

-- INSERT / UPDATE / DELETE
INSERT INTO alumnos(nombre, edad) VALUES(?, ?);
UPDATE alumnos SET edad = ? WHERE id = ?;
DELETE FROM alumnos WHERE id = ?;
```

Joins típicos:

```
-- INNER JOIN: solo coincidencias
SELECT l.titulo, a.nombre
FROM libros l
JOIN autores a ON l.autor_id = a.id;

-- LEFT JOIN: todo de la izq, nulos si no coincide
SELECT a.nombre, l.titulo
FROM autores a
LEFT JOIN libros l ON l.autor_id = a.id;
```

Claves y restricciones:

- **PK** (PRIMARY KEY), **FK** (FOREIGN KEY).
- **UNIQUE**, **NOT NULL**, **CHECK**, **DEFAULT**.

- **ACID**: Atomicidad, Consistencia, Aislamiento, Durabilidad.

4. JDBC a fondo (sin frameworks)

Flujo típico de acceso:

```
// 1) Conexión (DriverManager); JDBC 4+ carga drivers automáticamente si están en el classpa
String url = "jdbc:mysql://localhost:3306/biblioteca?useSSL=false&serverTimezone=UTC";
try (Connection con = DriverManager.getConnection(url, "user", "pass")) {

    // 2) Preparar sentencia con parámetros (?) para evitar inyección SQL
    String sql = "SELECT id, titulo FROM libros WHERE autor_id = ? AND anio >= ?";
    try (PreparedStatement ps = con.prepareStatement(sql)) {
        ps.setLong(1, autorId);
        ps.setInt(2, anioMin);

        // 3) Ejecutar y procesar resultados
        try (ResultSet rs = ps.executeQuery()) {
            List<Libro> lista = new ArrayList<>();
            while (rs.next()) {
                Libro l = new Libro(rs.getLong("id"), rs.getString("titulo"));
                lista.add(l);
            }
            return lista;
        }
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

Transacciones (autocommit/commit/rollback):

```
try (Connection con = DriverManager.getConnection(url, "user", "pass")) {
    con.setAutoCommit(false); // manejo manual de transacción

    try (PreparedStatement ps1 = con.prepareStatement(
            "UPDATE cuentas SET saldo = saldo - ? WHERE id = ?");
         PreparedStatement ps2 = con.prepareStatement(
            "UPDATE cuentas SET saldo = saldo + ? WHERE id = ?")) {

        // transferencia 100€ de 1 -> 2
        ps1.setBigDecimal(1, new BigDecimal("100.00"));
        ps1.setLong(2, 1L);
        ps1.executeUpdate();

        ps2.setBigDecimal(1, new BigDecimal("100.00"));
        ps2.setLong(2, 2L);
        ps2.executeUpdate();

        con.commit(); // todo OK
    } catch (SQLException ex) {
        con.rollback(); // revierte todo si algo falla
        throw ex;
    } finally {
        con.setAutoCommit(true);
    }
}
```

Buenas prácticas con JDBC:

- **PreparedStatement** siempre (evita inyección, tipado fuerte con setInt, setString...).

- Usa **try-with-resources** para cerrar Connection, PreparedStatement y ResultSet.
- **Parametriza** consultas; nunca concatenes strings con datos de usuario.
- Para varias operaciones relacionadas, **desactiva autocommit** y usa commit/rollback.
- Comprueba **executeUpdate()** para ver cuántas filas se afectan.

5. Ficheros: CSV / JSON / XML

CSV rápido (java.nio + split):

```
Path p = Path.of("alumnos.csv");  // nombre,edad
try (var br = Files.newBufferedReader(p)) {
    String linea;
    while ((linea = br.readLine()) != null) {
        String[] t = linea.split(",");
        Alumno a = new Alumno(t[0], Integer.parseInt(t[1]));
        // procesar...
    }
}
```

JSON (Jackson) — sin frameworks:

```
// Gradle/Maven: com.fasterxml.jackson.core:jackson-databind
ObjectMapper om = new ObjectMapper();

// Escribir lista a JSON
List<Alumno> lista = List.of(new Alumno("Ana", 20), new Alumno("Luis", 22));
om.writerWithDefaultPrettyPrinter().writeValue(Path.of("alumnos.json").toFile(), lista);

// Leer desde JSON
List<Alumno> leidos = om.readValue(Path.of("alumnos.json").toFile(),
    new TypeReference<List<Alumno>>() {});
```

JSON (Gson) — alternativa:

```
Gson gson = new GsonBuilder().setPrettyPrinting().create();

// Escribir
String json = gson.toJson(lista);
Files.writeString(Path.of("alumnos.json"), json);

// Leer (array -> lista)
Alumno[] arr = gson.fromJson(Files.readString(Path.of("alumnos.json")), Alumno[].class);
List<Alumno> leidos = Arrays.asList(arr);
```

XML (JAXB) — mapeo básico:

```
@XmlRootElement
public class Alumno {
    public String nombre;
    public int edad;
    public Alumno() {}
    public Alumno(String n, int e) { this.nombre = n; this.edad = e; }
}

// Escribir
JAXBContext ctx = JAXBContext.newInstance(Alumno.class);
Marshaller marshaller = ctx.createMarshaller();
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
marshaller.marshal(new Alumno("Ana", 20), Path.of("alumno.xml").toFile());

// Leer
```

```
Unmarshaller u = ctx.createUnmarshaller();
Alumno a = (Alumno) u.unmarshal(Path.of("alumno.xml").toFile());
```

6. Configuración externa (properties) y utilidades

```
// db.properties
url=jdbc:mysql://localhost:3306/biblioteca
user=root
pass=toor

// Cargar properties
Properties pr = new Properties();
try (var in = Files.newInputStream(Path.of("db.properties"))) {
    pr.load(in);
}
String url = pr.getProperty("url");
```

7. Proyecto unificador (CLI)

Mini app “Gestor de Biblioteca”: entidades **Libro**, **Autor**, **Prestamo**. Capa DAO con JDBC. Exporta préstamos a JSON/XML. Menú por consola. Prueba inserts, selects, transacciones.

```
// Esqueleto del menú (simplificado)
while (true) {
    System.out.println("1) Alta libro  2) Listar  3) Prestar  4) Exportar JSON  0) Salir");
    int op = Integer.parseInt(scan.nextLine());
    switch (op) {
        case 1 -> servicioLibros.alta();
        case 2 -> servicioLibros.listar();
        case 3 -> servicioPrestamos.prestar();
        case 4 -> exportadorJSON.exportarPrestamos();
        case 0 -> { return; }
    }
}
```

8. Mapas mentales y preguntas rápidas

```
Flujo mental:
App → DAO → JDBC → SQL → BD
         ■ Ficheros (CSV/JSON/XML)

Preguntas express:
• ¿Qué devuelve executeQuery() y executeUpdate()?
• ¿Cómo previenes inyección SQL?
• ¿Cuándo usarías commit/rollback?
• ¿Cómo leerías una lista de objetos desde JSON?
• ¿Diferencia entre INNER JOIN y LEFT JOIN?
```

9. Cheat-Sheet de examen

- Pipeline típico JDBC: *getConnection* → *prepare* → *set params* → *execute* → *mapear ResultSet* → *cerrar*.
- Siempre PreparedStatement, nunca concatenar strings con datos.
- Una operación = una transacción coherente (desactiva autocommit si agrupa varias).
- CSV: split por coma; JSON: Jackson/Gson; XML: JAXB.
- Config fuera del código (properties).

Tip final: escribe 1 ejemplo **completo** de CRUD con JDBC antes del examen. Si puedes reproducirlo de memoria, lo tienes.