```java
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class GestorFutbol {

    private static final String URL =
"jdbc:mysql://localhost:3306/futbol_db";
    private static final String USER = "root";
    private static final String PASS = "";

    private Connection conectar() throws SQLException {
        return DriverManager.getConnection(URL, USER,
PASS);
    }

    // ==========================================
    // 1. LISTAR (Read) - Reconstruye los objetos
    // ==========================================
    public ArrayList<Jugador> listarJugadores() {
        ArrayList<Jugador> lista = new ArrayList<>();
        // Hacemos JOIN para traer los datos del equipo
también
        String sql = "SELECT j.id, j.nombre, j.posicion,
e.nombre AS nom_eq, e.estadio " +
                     "FROM jugadores j " +
                     "JOIN equipos e ON j.id_equipo = e.id";

        try (Connection conn = conectar();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                // 1. Reconstruimos el objeto Equipo
                Equipo e = new
Equipo(rs.getString("nom_eq"), rs.getString("estadio"));

                // 2. Reconstruimos el objeto Jugador
                Jugador j = new
Jugador(rs.getString("nombre"), rs.getString("posicion"),
e);
                j.setId(rs.getInt("id")); // Guardamos el
ID de la BBDD

                lista.add(j);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return lista;
    }

    // ==========================================
    // 2. INSERTAR (Create)
    // ==========================================
    public void insertarJugador(Jugador j) {
        String sql = "INSERT INTO jugadores (nombre,
posicion, id_equipo) VALUES (?, ?, ?)";

        try (Connection conn = conectar()) {
            // 1. Obtenemos el ID del equipo (lo crea si
no existe)
            int idEquipo = gestionarEquipo(conn,
j.getEquipo());

            // 2. Insertamos al jugador usando ese ID
            try (PreparedStatement ps =
conn.prepareStatement(sql)) {
                ps.setString(1, j.getNombre());
                ps.setString(2, j.getPosicion());
                ps.setInt(3, idEquipo);
                ps.executeUpdate();
                System.out.println("✅ Jugador insertado:
" + j.getNombre());
            }
        } catch (SQLException e) {
            System.err.println("Error al insertar: " +
e.getMessage());
        }
    }

    // ==========================================
    // 3. ACTUALIZAR (Update)
    // ==========================================
    public void actualizarJugador(int idJugador, Jugador
nuevosDatos) {
        String sql = "UPDATE jugadores SET nombre = ?,
posicion = ?, id_equipo = ? WHERE id = ?";

        try (Connection conn = conectar()) {
            // 1. Verificamos el equipo de los nuevos
datos (por si cambió de equipo)
            int idEquipo = gestionarEquipo(conn,
nuevosDatos.getEquipo());

            // 2. Actualizamos
            try (PreparedStatement ps =
conn.prepareStatement(sql)) {
                ps.setString(1, nuevosDatos.getNombre());
                ps.setString(2,
nuevosDatos.getPosicion());
                ps.setInt(3, idEquipo);
                ps.setInt(4, idJugador);

                int filas = ps.executeUpdate();
                if (filas > 0) System.out.println("🔄
Jugador actualizado correctamente.");
                else System.out.println("⚠️ No se encontró
el jugador con ID " + idJugador);
            }
        } catch (SQLException e) {
            System.err.println("Error al actualizar: " +
e.getMessage());
        }
    }

    // ==========================================
    // 4. ELIMINAR (Delete)
    // ==========================================
    public void eliminarJugador(int idJugador) {
        String sql = "DELETE FROM jugadores WHERE id = ?";

        try (Connection conn = conectar();
             PreparedStatement ps =
conn.prepareStatement(sql)) {

            ps.setInt(1, idJugador);
            int filas = ps.executeUpdate();

            if (filas > 0) System.out.println("❌ Jugador
eliminado.");
            else System.out.println("⚠️ ID no
encontrado.");

        } catch (SQLException e) {
            System.err.println("Error al eliminar: " +
e.getMessage());
        }
    }

    // ==========================================
    // MÉTODO AUXILIAR (El cerebro de la relación)
    // ==========================================
    /**
     * Busca si el equipo existe por nombre.
     * Si existe -> Devuelve su ID.
     * Si NO existe -> Lo inserta y devuelve el nuevo ID
generado.
     */
    private int gestionarEquipo(Connection conn, Equipo
equipo) throws SQLException {
        // 1. Intentar buscar el equipo
        String sqlBuscar = "SELECT id FROM equipos WHERE
nombre = ?";
        try (PreparedStatement ps =
conn.prepareStatement(sqlBuscar)) {
            ps.setString(1, equipo.getNombre());
            ResultSet rs = ps.executeQuery();
            if (rs.next()) {
                return rs.getInt("id"); // ¡Ya existía!
Devolvemos su ID.
            }
        }

        // 2. Si no existía, lo insertamos
        String sqlInsertar = "INSERT INTO equipos (nombre,
estadio) VALUES (?, ?)";
        try (PreparedStatement ps =
conn.prepareStatement(sqlInsertar,
Statement.RETURN_GENERATED_KEYS)) {
            ps.setString(1, equipo.getNombre());
            ps.setString(2, equipo.getEstadio());
            ps.executeUpdate();

            ResultSet rs = ps.getGeneratedKeys();
            if (rs.next()) {
                System.out.println("   (Equipo nuevo
creado: " + equipo.getNombre() + ")");
```

```
            return rs.getInt(1); // Devolvemos el
nuevo ID
        }
    }

    throw new SQLException("No se pudo obtener el ID
del equipo.");
  }
}
```

```
try (Connection con = DriverManager.getConnection(url, user,
pass)) {
    con.setAutoCommit(false); // Desactivar autocommit →
inicia transacción

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

    ps1.setDouble(1, 500.0);
    ps1.setInt(2, 1);
    ps1.executeUpdate();

    ps2.setDouble(1, 500.0);
    ps2.setInt(2, 2);
    ps2.executeUpdate();

    con.commit(); // Todo correcto
    System.out.println("Transacción completada
correctamente");

} catch (SQLException e) {
    System.err.println("Error en transacción: " +
e.getMessage());
    try {
        con.rollback(); // Revertir todos los cambios
        System.err.println("Transacción revertida");
    } catch (SQLException ex) {
        System.err.println("Error al hacer rollback: " +
ex.getMessage());
    }
} finally {
    con.setAutoCommit(true); // Vuelvo a dejar el valor por
defecto
}
```

```
con.setAutoCommit(false);  // Inicia transacción
    // operaciones SQL
con.commit();              // Confirma cambios
con.rollback();            // Revierte si hay error
con.setAutoCommit(true);   // Se vuelve al valor inicial
```

```
try {
    Connection con = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/tienda", "root", "");
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery("SELECT * FROM producto");
} catch (SQLException e) {
    System.out.println("Error en la base de datos: " +
e.getMessage());
    System.out.println("Código de error: " +
e.getErrorCode());
    System.out.println("Estado SQL: " + e.getSQLState());
}
```

Configuración y Patrón Básico

1. Establecer la Conexión (Boilerplate)

Este es el código estándar para conectarse. Si te piden un método estático para obtener conexión, usa este modelo.
import java.sql.*;

```
public class DatabaseConnection {
    // Ajusta la URL según el nombre de tu BBDD en el examen
    private static final String URL = "jdbc:mysql://localhost:3306/nombre_bbdd";
    private static final String USER = "root";
    private static final String PASSWORD = "";
    public static Connection getConnection() throws SQLException {
        // En versiones modernas de JDBC no hace falta Class.forName(...)
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
}
```

2. La Estructura "Try-with-Resources" (OBLIGATORIO)

El examen pide **"Gestionar la conexión y liberar recursos"**. La forma más profesional y segura (evita que te bajen nota por no cerrar conexiones) es esta estructura. Cierra automáticamente ResultSet, PreparedStatement y Connection.
String sql = "TU SENTENCIA SQL AQUÍ";

```
// 1. Abrir conexión y PreparedStatement dentro del paréntesis del try
try (Connection conn = DatabaseConnection.getConnection();
     PreparedStatement stmt = conn.prepareStatement(sql)) {
    // 2. Setear valores (si hay ?)
    stmt.setInt(1, valor);
    // 3. Ejecutar
    stmt.executeUpdate(); // Para INSERT, UPDATE, DELETE
    // O
    ResultSet rs = stmt.executeQuery(); // Para SELECT
} catch (SQLException e) {
    e.printStackTrace(); // O lanzar una IOException personalizada
}
```

Escritura (INSERT/UPDATE) y ArrayLists

1. Insertar una lista de Objetos (ArrayList -> BBDD)

El profesor dijo: *"Os doy un ArrayList y os pido que rellenéis una BBDD"*.
Estrategia: Recorres la lista y ejecutas el INSERT por cada elemento.
// Suponiamos que recibes: List<Alumno> listaAlumnos

```
public void guardarLista(List<Alumno> listaAlumnos) throws SQLException {
    String sql = "INSERT INTO alumnos (id, nombre, email) VALUES (?, ?, ?)";
    try (Connection conn = DatabaseConnection.getConnection();
         PreparedStatement stmt = conn.prepareStatement(sql)) {
        // Desactivar auto-commit si quieres que sea una transacción única (opcional pero recomendado)
        conn.setAutoCommit(false);

        for (Alumno alu : listaAlumnos) {
            stmt.setInt(1, alu.getId());
            stmt.setString(2, alu.getNombre());
            stmt.setString(3, alu.getEmail());

            stmt.executeUpdate(); // Ejecuta la inserción
        }
        conn.commit(); // Confirmar todos los cambios al final
    } catch (SQLException e) {
        // Manejar error y rollback si usaste transacción
        throw e;
    }
}
```

2. INSERT con "ON DUPLICATE KEY UPDATE"

Usado en tu proyecto para **Crear o Modificar** en la misma sentencia. Muy útil si te piden "Guardar" (sea nuevo o existente).

```
String sql = "INSERT INTO asignaturas (id, nombre, creditos) VALUES (?, ?, ?) " +
             "ON DUPLICATE KEY UPDATE nombre = ?, creditos = ?";
try (Connection conn = DatabaseConnection.getConnection();
     PreparedStatement stmt = conn.prepareStatement(sql)) {
    // Valores para el INSERT
    stmt.setInt(1, asignatura.getId());
    stmt.setString(2, asignatura.getNombre());
    stmt.setInt(3, asignatura.getCreditos());
    // Valores para el UPDATE (si el ID ya existe)
    stmt.setString(4, asignatura.getNombre());
    stmt.setInt(5, asignatura.getCreditos());
    int filasAfectadas = stmt.executeUpdate();
    System.out.println("Filas afectadas: " + filasAfectadas);
}
```

3. Eliminar (DELETE)

```
String sql = "DELETE FROM alumnos WHERE id = ?";
try (Connection conn = DatabaseConnection.getConnection();
     PreparedStatement stmt = conn.prepareStatement(sql)) {
    stmt.setInt(1, idAlumno);
    int afectados = stmt.executeUpdate();
    if (afectados > 0) System.out.println("Eliminado correctamente");
    else System.out.println("No se encontró el ID");
}
```

Transacciones y Fechas

1. Gestión de Transacciones (IMPORTANTE)

Ejemplo Modelo (Basado en tu SQLMatriculaRepository):

```
public void guardarMatriculaCompleta(Matricula matricula) throws IOException {
    Connection conn = null; // Declarar fuera para poder usarla en el catch
    try {
        conn = DatabaseConnection.getConnection();
        conn.setAutoCommit(false); // 1. INICIAR TRANSACCIÓN
        // --- PASO A: Guardar Cabecera (Matrícula) ---
        String sqlMat = "INSERT INTO matriculas (id, id_alumno, fecha) VALUES (?, ?, ?)";
        try (PreparedStatement stmt = conn.prepareStatement(sqlMat)) {
            stmt.setInt(1, matricula.getId());
            stmt.setInt(2, matricula.getAlumno().getId());
            stmt.setDate(3, Date.valueOf(matricula.getFecha())); // Ojo a la conversión de fecha
            stmt.executeUpdate();
        }
        // --- PASO B: Guardar Líneas (Asignaturas) ---
        String sqlRel = "INSERT INTO matricula_asignatura (id_matricula, id_asignatura) VALUES (?, ?)";
        try (PreparedStatement stmt = conn.prepareStatement(sqlRel)) {
            for (Asignatura asig : matricula.getAsignaturas()) {
                stmt.setInt(1, matricula.getId());
                stmt.setInt(2, asig.getId());
                stmt.executeUpdate();
            }
        }
        conn.commit(); // 2. CONFIRMAR CAMBIOS (Todo ha ido bien)
    } catch (SQLException e) {
        if (conn != null) {
            try {
                conn.rollback(); // 3. REVERTIR SI HAY ERROR
                System.out.println("Rollback ejecutado.");
            } catch (SQLException ex) { ex.printStackTrace(); }
        }
        throw new IOException("Error en transacción", e);
    } finally {
        // 4. RESTAURAR ESTADO Y CERRAR
        if (conn != null) {
            try {
                conn.setAutoCommit(true);
                conn.close();
            } catch (SQLException ex) { ex.printStackTrace(); }
        }
    }
}
```

2. Manejo de Fechas (LocalDate vs SQL Date)

En Java solemos usar LocalDate, pero JDBC necesita java.sql.Date.
De Java a SQL (Para guardar):
Java

```
LocalDate fechaJava = LocalDate.now();
stmt.setDate(indice, Date.valueOf(fechaJava));
```

De SQL a Java (Para leer):
Java

```
java.sql.Date fechaSql = rs.getDate("fecha");
LocalDate fechaJava = fechaSql.toLocalDate();
```

Lectura (SELECT) y Teoría Resumida

1. Leer Datos (SELECT) y Mapear a Objetos

El examen pide **"Mostrar los resultados obtenidos"**.

```
public List<Alumno> buscarTodos() throws IOException {
    List<Alumno> lista = new ArrayList<>();
    String sql = "SELECT * FROM alumnos"; // O "WHERE id = ?"
    try (Connection conn = DatabaseConnection.getConnection();
         Statement stmt = conn.createStatement(); // O PreparedStatement si hay parámetros
         ResultSet rs = stmt.executeQuery(sql)) {
        // Iterar mientras haya filas
        while (rs.next()) {
            // 1. Extraer datos de las columnas
            int id = rs.getInt("id"); // Nombre de la columna en BBDD
            String nombre = rs.getString("nombre");
            String email = rs.getString("email");

            // 2. Crear objeto Java
            Alumno alumno = new Alumno(id, nombre, email);

            // 3. Añadir a la lista
            lista.add(alumno);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return lista;
}
```

2. Métodos clave de JDBC

Método	Uso Principal	Retorno
executeQuery()	Para **SELECT**.	ResultSet (la tabla de resultados).
executeUpdate()	Para **INSERT, UPDATE, DELETE**.	int (número de filas afectadas).
execute()	Genérico (si no sabes qué viene).	boolean (true si devuelve ResultSet).

4. Tipos de Datos (Java <-> SQL

- int -> getInt()
- String (VARCHAR/TEXT) -> getString()
- double (DECIMAL/FLOAT) -> getDouble()
- boolean (TINYINT) -> getBoolean()
- LocalDate (DATE) -> getDate()