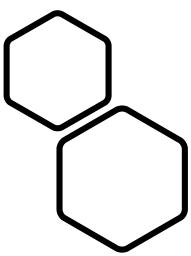




Cloud Computing-Smart Cities-High
Performance Computing



DR. GABRIEL LEON PAREDES

gleon@ups.edu.ec

www.linkedin.com/in/gabrielleonp

Cloud Computing, Smart Cities & High-Performance
Computing
Cuenca, Ecuador

APLICACIONES DISTRIBUIDAS

Persistencia JDBC

- Consiste en guardar los datos de forma permanente, normalmente Bases de Datos.
- **JDBC (Java Database Connectivity).** Son un conjunto de interfaces para independizar el acceso a las bases de datos relacionales desde las aplicaciones Java
- Se requiere la implementación específica del gestor de bases de datos utilizado



- MySQL
 - <http://www.mysql.com>
 - <http://dev.mysql.com>
- Versión: MySQL Community Server 8.0.19.
 - Instalación local (localhost) y puerto: 3306. Usuario root con clave en blanco
- MySQL Connector/J 8.0.19
 - <https://dev.mysql.com/downloads/connector/j/>
- Iniciar base de datos por consola
 - %MySQL%>mysql -u root
- Muestra las bases de datos existentes: show databases;
- Crea una base de datos llamada “jee”: create database jee;
- Borra una base de datos llamada “jee”: drop database jee;
- Se conecta a la base de datos “jee”: use jee;
- Muestra las tablas de la base de datos actual: show tables;
- Describe una tabla: describe MiTabla;
- Selecciona los registros de una tabla: select * from MiTabla;

JDBC

- Copiar el conector *.jar en la carpeta lib:
 - Proyecto JavaEE: /WEB-INF/lib
 - Proyecto Java: /lib (debe colgar directamente del proyecto) (se debe registrar con "build path")
- Importante: importar los interfaces, NO la implementación concreta
- Crear el "Class" del driver del controlador
- Registrar los controladores JDBC , cada fabricante tiene una clase diferente.
 - Class.forName("com.mysql.jdbc.Driver");
 - Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
- Abrir una conexión
 - Se puede crear una conexión a una BD concreta, mediante la clase DriverManager, con el método getConnection(url,user,pass). Este utiliza el driver registrado. Se debe proporcionar tres datos:
 - url: "jdbc:mysql://localhost:3306/jee" (suponiendo que la BD es "jee")
 - user: "root"
 - pass: ""
 - conexion = DriverManager.getConnection(url, user, pass);

JDBC

- Crear una instancia de la clase Statement
 - Esta clase encapsula el texto SQL de la petición a la BD
 - El objeto se crea mediante: **conexion.createStatement()**
- Ejecutar modificaciones o consultas en la base de datos
 - Se realizan con el objeto Statement. Dispone de dos métodos fundamentales:
 - **executeUpdate**. Para ejecutar INSERT, UPDATE o DELETE, y sentencias SQL que no devuelven nada (sentencias D CREATE, DROP...)
 - **executeQuery**. Para ejecutar sentencias SQL, que devuelve un único objeto ResultSet (SELECT...)
- Procesar resultados
 - **ResultSet** es una lista ordenada de tuplas
 - next(): próximo, previous(): previo, first(): primero, last(): último...
 - Cerrar objetos
 - Resultset, Statement y Connection: close()

```

9 public class HolajDBC {
10
11     public static void main(String[] args) {
12         Connection conexion = null;
13         Statement sentencia = null;
14         ResultSet result = null;
15
16         String url = "jdbc:mysql://localhost:3306/jee";
17         String user = "root";
18         String pass = "";
19
20     try {
21         Class.forName("com.mysql.cj.jdbc.Driver");
22         conexion = DriverManager.getConnection(url, user, pass);
23         sentencia = conexion.createStatement();
24     } catch (ClassNotFoundException e) {
25         System.out.println("Imposible cargar el driver: " + e.getMessage());
26     } catch (SQLException e) {
27         System.out.println("Imposible conectar: " + e.getMessage());
28     }
29
30     try {
31         sentencia.executeUpdate("DROP TABLE IF EXISTS tabla1");
32         sentencia.executeUpdate("CREATE TABLE tabla1 (id1 INT PRIMARY KEY, nombre CHAR(20) DEFAULT '-');");
33     } catch (SQLException e) {
34         System.out.println("Creacion de tabla fallida: " + e.getMessage());
35
36     try {
37         sentencia.executeUpdate("INSERT tabla1 (id1) VALUES (3)");
38         sentencia.executeUpdate("INSERT tabla1 VALUES (4, 'Jesus')");
39         sentencia.executeUpdate("INSERT tabla1 VALUES (5, 'Juan')");
40     } catch (SQLException e) {
41         System.out.println("Inserccion de datos de tabla fallida: " + e.getMessage());
42
43     try {
44         result = sentencia.executeQuery("SELECT * FROM tabla1");
45         while (result.next())
46             System.out.println("id1: " + result.getLong("id1") + ", nombre: " + result.getString("nombre"));
47     } catch (SQLException e) {
48         System.out.println("Consulta Fallida: " + e.getMessage());
49     }
50
51

```

Ejemplo

uso JDBC

Transacciones

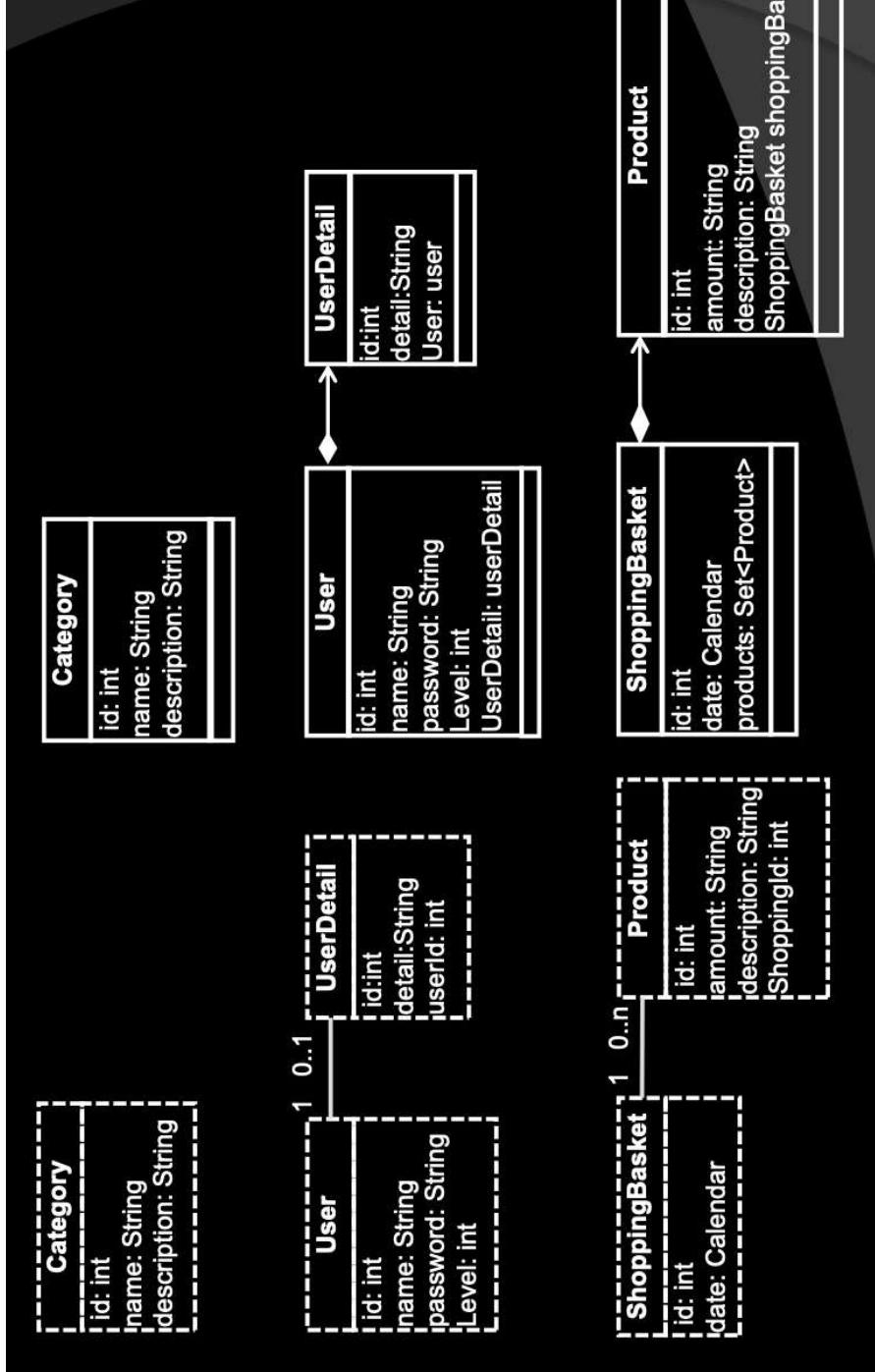
- Un conjunto de acciones tratadas como atómicas
- Sentencia SQL es atómica
- El alcance está limitado a una conexión
- Transacción
 - BEGIN
 - COMMIT o ROLLBACK
- Lecturas consistentes por otras conexiones, los cambios sin confirmar no aparecen
- Lecturas por la propia conexión, los cambios sin confirmar, aparecen realizados

Ejemplo

```
49
50     try { // begin
51         conexion.setAutoCommit(false);
52         // sentencias SQL
53         sentencia = conexion.createStatement();
54         sentencia.executeUpdate("INSERT tabla1 VALUES (8,'Trans1')");
55         sentencia.executeUpdate("INSERT tabla1 VALUES (8,'Trans2')");
56         // Si se llega a este punto, todo ha ido bien
57         conexion.commit();
58     } catch (SQLException e) {
59         try { // Hay problemas, se deshace todo
60             conexion.rollback();
61             System.out.println("Deshaciendo... " + e.getMessage());
62         } catch (SQLException e1) {
63             System.out.println("ERROR (rollback): " + e1.getMessage());
64         }
65     }
66 }
67
68 }
```

Patrón de diseño DAO (Data Access Object)

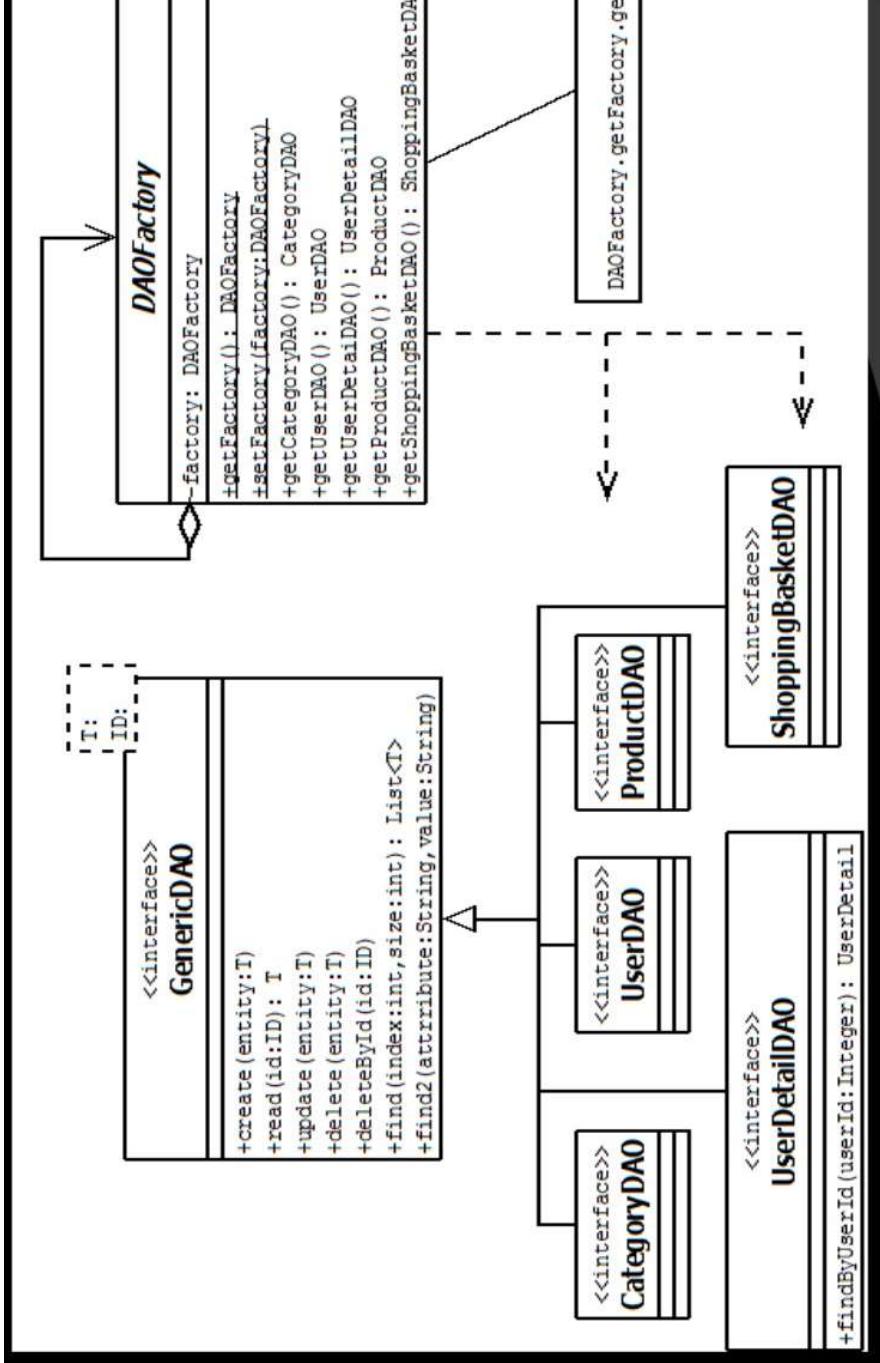
- Objetivo: abstraer y encapsular el acceso a la capa de persistencia (Bases de Datos), permitiendo desacoplar la capa de negocio con la capa de persistencia
- Adapta el modelo relacional de BD con el modelo de objetos
- Permite el intercambio de implementaciones de persistencia sin afectar a la capa de negocio.



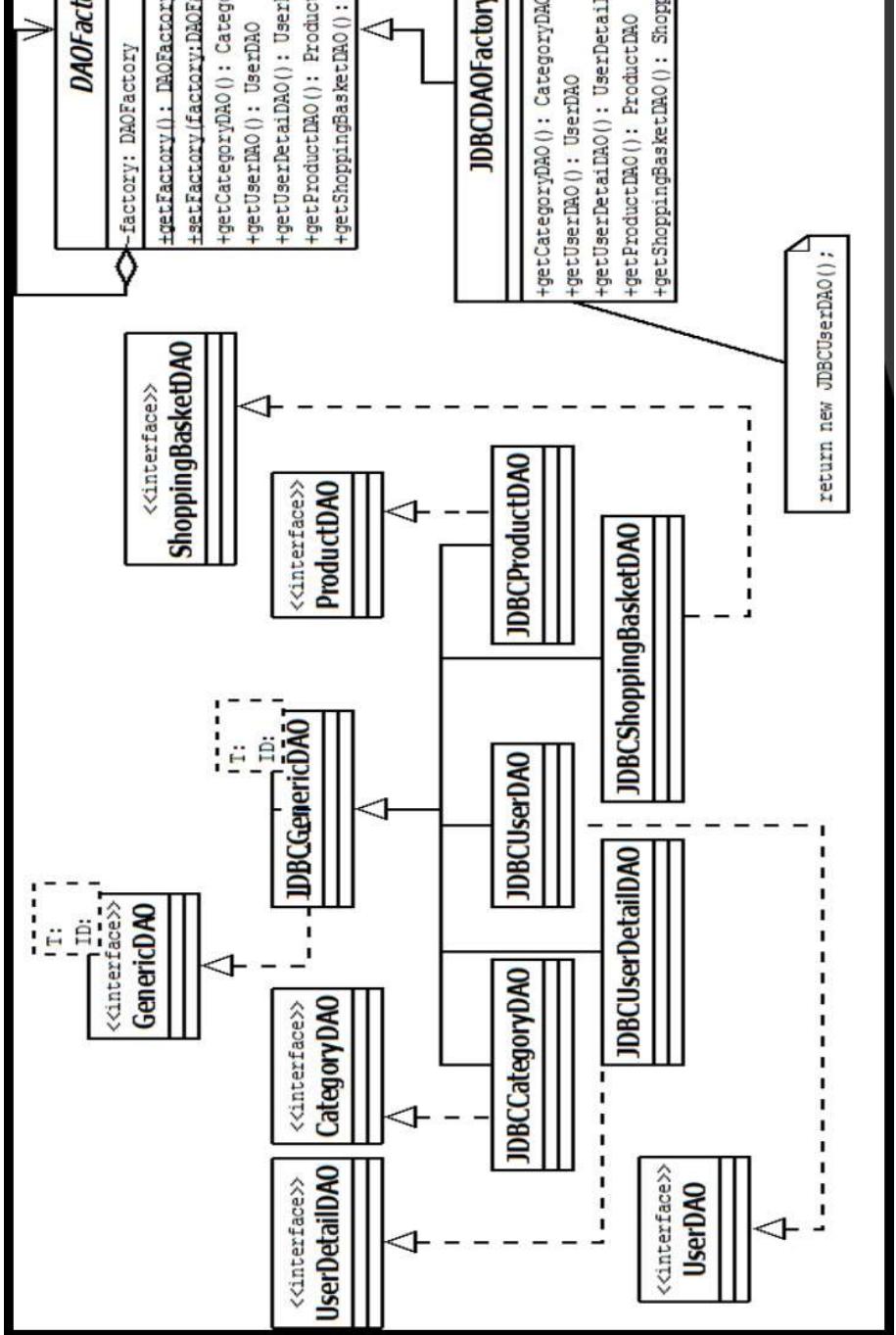
Aplicaciones Distribuidas - Dr. Gabriel A. León Paredes

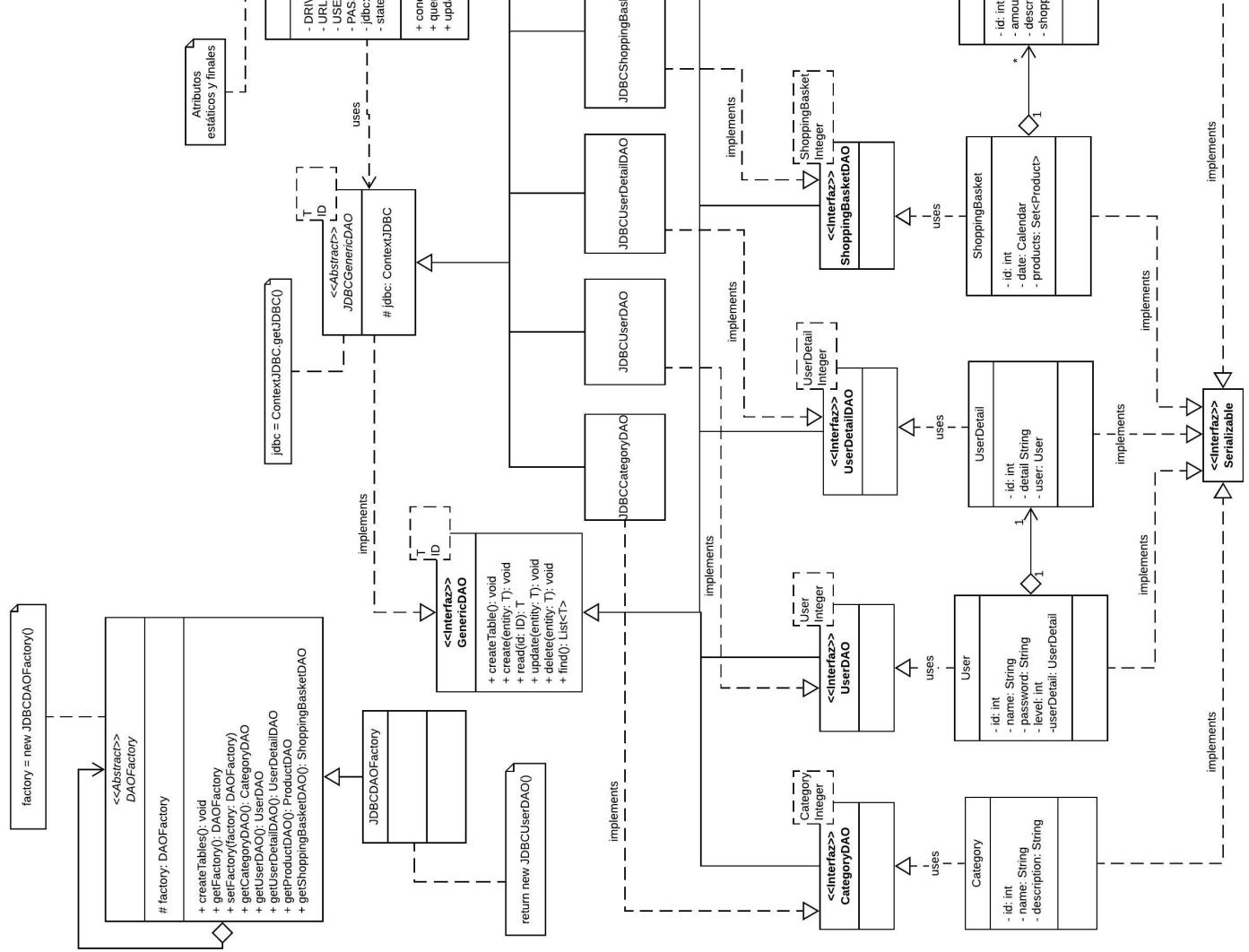
Ejemplo: Patrón DAO

Ejemplo: Patrón DAO



Ejemplo: Patrón DAO





Patrón DAO

```

public interface GenericDAO<T, ID> {
    public void createTable();
    public void create(T entity);
    public T read(ID id);
    public void update(T entity);
    public void delete(T entity);
    public List<T> find();
}

public interface CategoryDAO extends GenericDAO<Category, Integer> { }

public class Category implements Serializable {
    private int id; private String name; private String description;
    public Category(int id, String name, String description) {
        this.setId(id); this.setName(name); this.setDescription(description);
    }
    public int getId() { return this.id; }
    public void setId(int id) { this.id = id; }
    public String getName() { return this.name; }
    public void setName(String name) { this.name = name; }
    public String getDescription() { return this.description; }
    public void setDescription(String description) { this.description = description; }
    @Override String toString() {
        return "Category [id=" + id + ", name=" + name + ", description=" + description + "]";
    }
}

```

GenericDAO

Category

Category

Context

```
public class ContextJDBC {  
    private static final String DRIVER = "com.mysql.jdbc.Driver";  
    private static final String URL = "jdbc:mysql://localhost:3306/jee";  
    private static final String USER = "root"; private static final String PASS = "";  
  
    private static ContextJDBC jdbc = null;  
    private Statement statement = null;  
  
    public ContextJDBC() { this.connect(); }  
    protected static ContextJDBC getJDBC() {  
        if (jdbc == null) jdbc = new ContextJDBC();  
        return jdbc;  
    }  
  
    private void connect() {  
        try {  
            Class.forName(DRIVER);  
            Connection connection = DriverManager.getConnection(URL, USER, PASS);  
            this.statement = connection.createStatement();  
        } catch (ClassNotFoundException e) {  
            System.out.println("">>>WARNING (JDBC:connect)...problemas con el driver");  
        } catch (SQLException e) { System.out.println("">>>WARNING (JDBC:connect)...problemas con la BD"); }  
    }  
  
    public ResultSet query(String sql){  
        try { return this.statement.executeQuery(sql); }  
        catch (SQLException e) { System.out.println("">>>WARNING (JDBC:query): ----" + sql + "----" + e); }  
        return null;  
    }  
  
    public boolean update(String sql) {  
        try { this.statement.executeUpdate(sql); return true; }  
        catch (SQLException e) { System.out.println("">>>WARNING (JDBC:update)... actualización: ----" + sql + "----" + e); }  
        return false;  
    }  
}
```

```

public abstract class JDBCGenericDAO<T, ID> implements GenericDAO<T, ID> {
    protected ContextJDBC jdbc = ContextJDBC.getJDBC();
}

public class JDBCCategoryDAO extends JDBCGenericDAO<Category, Integer> implements CategoryDAO {
    @Override public void createTable() {
        jdbc.update("DROP TABLE IF EXISTS Category ");
        jdbc.update("CREATE TABLE Category (" + "ID INT NOT NULL," + "DESCRIPTION VARCHAR(255)," +
                    "NAME VARCHAR(255)," + "PRIMARY KEY (ID))");
    }

    @Override public void create(Category category) {
        jdbc.update("INSERT Category VALUES (" + category.getId() + "," + category.getName() + +
                    " ,'" + category.getDescription() + "')");
    }

    @Override public Category read(Integer id) {
        Category c = null;
        ResultSet rs = jdbc.query("SELECT * FROM Category WHERE id=" + id);
        try {
            if (rs != null && rs.next()) {
                c = new Category(rs.getInt("id"), rs.getString("name"), rs.getString("description"));
            }
        } catch (SQLException e) {
            System.out.println(">>>WARNING (Category:read):" + e.getMessage());
        }
        return c;
    }
}

```

```

@Override public void update(Category category) {
    jdbc.update("UPDATE Category SET name='"
        + category.getName() + "' ,description='"
        + category.getDescription() + "' WHERE id='"
        + category.getId() + "'");
}

@Override public void delete(Category category) {
    jdbc.update("DELETE FROM Category WHERE id='"
        + category.getId());
}

@Override public List<Category> find() {
    List<Category> list = new ArrayList<Category>();
    ResultSet rs = jdbc.query("SELECT * FROM Category");
    try {
        while (rs.next()) {
            list.add(new Category(rs.getInt("id"),
                rs.getString("name"),
                rs.getString("description")));
        }
    } catch (SQLException e) {
        System.out.println(">>>WARNING (JDBCCategoryDAO:find):"
            + e.getMessage());
    }
    return list;
}

```

JDBCCCategory

(2-2)

```

public class JDBCUserDAO extends JDBCGenericDAO<User, Integer> implements dao.UserDAO {
    @Override public void createTable() {
        jdbc.update("DROP TABLE IF EXISTS UserDetail ");
        jdbc.update("DROP TABLE IF EXISTS User ");
        jdbc.update("CREATE TABLE User (ID INT NOT NULL, LEVEL INT,"
                + "NAME VARCHAR(255),PASSWORD VARCHAR(255),PRIMARY KEY (ID))");
        DAOFactory.getFactory().getUserDetailDAO().createTable();
    }

    @Override public void create(User user) {
        jdbc.update("INSERT User VALUES (" + user.getId() + "," + user.getLevel() + ","
                + user.getName() + "," + user.getPassword() + ")");
        UserDetail ud = user.getDetail();
        if (ud != null) DAOFactory.getFactory().getUserDetailDAO().create(ud);
    }

    @Override public User read(Integer id) {
        User u = null;
        ResultSet rs = jdbc.query("SELECT * FROM User WHERE id=" + id);
        try {
            if (rs != null && rs.next()) {
                u = new User(rs.getInt("id"), rs.getString("name"), rs.getString("password"));
                u.setLevel(rs.getInt("level"));
            }
        } catch (SQLException e) { System.out.println(">>>WARNING (UserDAO:read): " + e.getMessage()); }
        if (u == null) return null;
        UserDetail ud = DAOFactory.getFactory().getUserDetailDAO().findByUserId(id);
        if (ud != null) ud.setUser(u);
        u.setDetail(ud);
        return u;
    }
}

```

JDBCUse (1-2)

```

@Override public void update(User user) {
    UserDetailDAO uddAO = DAOFactory.getFactory().getUserDetailDAO();
    UserDetail ud = uddAO.findById(user.getId());
    jdbc.update("UPDATE User SET name='"
            + user.getName() + "' ,password='"
            + user.getPassword() + "' ,level='"
            + user.getLevel() + " WHERE id='"
            + user.getId() + "'");
    if (user.getDetail() == null && ud != null) uddAO.delete(ud);
    else if (user.getDetail() != null && ud == null) uddAO.create(user.getDetail());
    else if (user.getDetail() != null && ud != null) uddAO.update(user.getDetail());
}

@Override public void delete(User user) {
    if (user.getDetail() != null)
        DAOFactory.getFactory().getUserDetailDAO().delete(user.getDetail());
    jdbc.update("DELETE FROM User WHERE id='"
            + user.getId());
}

@Override
public List<User> find() {
    return null;
}
}

```

JDBC Use (2-2)

DAOFactory

```
public abstract class DAOFactory {  
    protected static DAOFactory factory = new JDBCDAOFactory();  
    public static DAOFactory getFactory() {  
        return factory;  
    }  
    public void createTables() {}  
    public abstract UserDao getUserDAO();  
    public abstract UserDetailDAO getUserDetailDAO();  
    public abstract CategoryDAO getCategoryDAO();  
}  
  
public class JDBCDAOFactory extends DAOFactory {  
    @Override public void createTables() {  
        this.getCategoryDAO().createTable();  
        this.getUserDAO().createTable();  
    }  
    @Override public CategoryDAO getCategoryDAO() { return new JDBCCategoryDAO(); }  
    @Override public UserDao getUserDAO() { return new JDBCUserDAO(); }  
    @Override public UserDetailDAO getUserDetailDAO() { return new JDBCUserDetailDAO(); }  
}
```

```
public class TestCategory {  
    public static void main(String[] args) {  
        CategoryDAO cd = DAOFactory.getFactory().getCategoryDAO();  
        cd.createTable();  
  
        Category c1, c2, c3, c4, c5;  
        c1 = new Category(1, "uno", "Categoría uno");  
        c2 = new Category(2, "dos", "Categoría dos");  
        c3 = new Category(3, "tres", "Categoría tres");  
        cd.create(c1); cd.create(c2); cd.create(c3);  
        cd.delete(c2);  
        c4 = cd.read(3);  
        c4.setDescription("Nueva...");  
        cd.update(c4);  
        c5 = new Category(1, "uno", "Otro de uno");  
        cd.update(c5);  
        cd.create(new Category(6, "seis", "Categoría seis"));  
        System.out.println(cd.find());  
    }  
}
```

Test

EjemploDAO

Actualizado

<https://bit.ly/3eQlTD>

Ejemplo JEE Persona Servlet + JSP + JDBC

<https://bit.ly/35DBIDj>

Referencias

“Desarrollo de aplicaciones web distribuidas código abierto (Java EE): Principios tecnológicos”
Bernal, J., UPM, 2012.

“Java EE 8 Development with Eclipse”, Kulkarni, R., Packt, 3th Edition, 2018.