

CASO DE ESTUDIO

Laboratorio 1

Responsables:

**Nicolás Alberto Henao Avendaño
Santiago Cadavid Bustamante
Sergio Andrés Castrillón Vásquez
Jerónimo Alzate Duque
Juan Fernando Giraldo Cardona
Aiber Andrés Ruiz Graciano**



Profesor:

Diego José Luis Botia Valderrama

**Medellín
Universidad de Antioquia
2017**

Introducción

El presente informe tiene como finalidad comprender el manejo básico de Java EE, para ello se resumen los conceptos vistos en el laboratorio número 1 del curso de *Arquitectura de Software* de la Universidad de Antioquia, siguiendo las indicaciones del profesor Ph. Diego Jose Luis Botia, mediante el uso de herramientas informáticas para el desarrollo de páginas Web.

El trabajo va dirigido a toda la comunidad que posea un conocimiento básico acerca de la programación orientada a objetos.

Objetivos

Objetivo general

Crear una aplicación CRUD que permita a los empleados manejar la información de un concesionario de vehículos.

Objetivos específicos

- Aplicar una arquitectura N capas para el desarrollo de una aplicación web.
- Configurar las bases de datos de los clientes, los vehículos y las ventas del concesionario.
- Conectar la base de datos al servidor de aplicaciones GlassFish.
- Generar recursos JDBC y JNDI
- Crear el modelo de clases con anotaciones JPA.
- Crear la Unidad de Persistencia
- Crear el componente Controller (Servlet)
- Diseñar el front-end de la aplicación
- Desplegar la aplicación en el servidor de aplicaciones.

MARCO TEÓRICO

- **JDBC (Java DataBase Connectivity):** Esta API se encarga de realizar una traducción entre una base de datos SQL y el lenguaje Java, gracias a esto se consigue una independencia entre los mismos. Se puede enviar instrucciones a la base de datos de manera transparente usando el lenguaje Java, logrando así uno de los pilares de Java: “write once, run anywhere”, lo que facilita el trabajo en empresas donde las necesidades y tecnologías usadas con respecto a las bases de datos son heterogéneas, se alcanza el objetivo de compactarlas en un solo lenguaje.
- **JNDI (Java naming and directory Interface):** Es una API que permite nombrar objetos, además de suministrar una interfaz para realizar búsquedas objetos en un entorno informático distribuido. Trabaja en conjunto con otras tecnologías que conforman Java Platform, Enterprise Edition (JavaEE).
- **Servlet de Java:** Es una clase de Java cuyo propósito es aumentar y mejorar las capacidades de los servidores web, gestionando solicitudes que los clientes realizan a través del navegador con el objetivo de generar páginas web de forma dinámica.
- **JSP (Java Server Page):** Es una tecnología que ayuda a los desarrolladores de software a desarrollar páginas web dinámicas basadas en HTML y XML. Tiene la misma filosofía de trabajo que PHP solamente que es desarrollado en lenguaje Java; además, son una forma alternativa de crear servlets, gracias a que el código JSP se traduce a código de servlet Java la primera vez que se le invoca y en adelante es el código del nuevo servlet que se ejecuta produciendo como salida el código HTML que compone la página web de respuesta.
- **JPA (Java Persistence Api):** Es un framework que maneja datos relacionales en aplicaciones Java a través de JavaEE o JavaSE. Realiza una abstracción que se hace sobre JDBC que nos permite tratar las tablas y sus relaciones, convirtiéndolas en clases de Java (que se les conoce como entities) a través de la conversión ORM (Object Relational Mapping) y puede configurarse a través de metadatos (xml o anotaciones); también permite hacer lo mismo en sentido contrario.
Cuando tenemos una entidad desde la base de datos y ciertos atributos no se van a utilizar en un principio, JPA nos permite decidir si se va a cargar inmediatamente esos atributos mediante la lectura denominada eager fetch, en caso de que vaya a esperar hasta utilizar esos atributos se le denominará

lazy fetch. Se podría pensar en dejar todos los atributos en lazy fetch, pero cuando los vaya a utilizar, el proveedor de persistencia tendrá que hacer una llamada a la base de datos para leerla, lo cual generaría un coste de ejecución bastante elevado; por eso se recomienda dejar este tipo de lectura para los objetos de gran tamaño y ciertos tipos de asociación entre tablas (uno a muchos y muchos a muchos).

- **JPQL (Java Persistence Query Language):** Es un lenguaje de consultas orientado a objetos que opera con objetos de entidad JPA en vez de hacerlo directamente en la base de datos. Se usa para hacer consultas con otras entidades almacenadas en una base de datos relacional, además soporta funciones como update, delete o select, inspirado en SQL.
- **Anotaciones JPA:** Algunas de las anotaciones más usadas son:
 - **@Entity:** Indica que se creará una entidad. Se coloca antes de escribir la clase
 - **@Id:** Indica la clave primaria
 - **@table (name = "tablaSQL"):** Representa de qué tabla se va a extraer la información.
 - **@Column:** Principalmente sirve para decir que la clave está asociada a un atributo de la tabla, pero además posee varias funcionalidades integradas en esa anotación, entre las que destacan: Indicar si el dato ingresado puede ser o no nulo, si existe un límite de caracteres o si cada fila de esa columna debe ser diferente a las demás; así:
`@Column (name="code", nullable=false, length=50, unique=true)`
 - **@GeneratedValue:** En caso de haber usado la expresión `AUTO_INCREMENT` en SQL, se generará esta anotación para representar que a ese dato subirá automáticamente.
 - **@Temporal:** Se usa generalmente en datos tipo `DATE` (SQL), dependiendo de la instrucción te dará información detallada de hora, segundos y milisegundos junto con la fecha (`TIMESTAMP`), si solo es la hora (`TIME`) o la fecha (`DATE`), así:
`@Temporal(TemporalType.TIMESTAMP)`
 - **@OneToMany:** Indica una relación unidireccional de a uno muchos. Se coloca para indicar que el elemento es una clave foránea.
 - **@ManyToOne:** Indica una relación unidireccional de muchos a uno.
 - **@JoinColumn:** Señala la columna que hará relación con otra tabla, generalmente va luego de instrucciones tipo `@OneToMany` o `@ManyToOne`

- **POJO (Plain Old Java Object):** Es una instancia de una clase que no extiende ni implementa nada en especial, por ejemplo: se implementa una clase Usuario, con sus atributos privados y sus respectivos setter y getter. Una instancia de esta clase es un objeto POJO.
- **DAO (Data Access Object) :** Patrón de diseño que suministra una interfaz común entre la aplicación y los dispositivos de almacenamiento de datos. Su uso está ligado principalmente a que cualquier objeto de negocio no necesita saber hacia dónde se dirige ni qué se va a hacer con la información que acarrea, de eso se encarga la interfaz. Esto implica que en momento de ejecución se ejecutarán más líneas de código, lo cual no es lo ideal en aplicaciones críticas con el rendimiento.
- **DTO (Data Transfer Object):** Es un objeto que únicamente transporta información entre procesos. A diferencia de DAO, DTO no contiene lógica del negocio; se podría asemejar a un carrito de compra que solo traslada información.
- **EJB (Enterprise Java Beans):** Es una API que pone a disposición del programador un modelo de componentes distribuidos del lado del servidor, en donde abstrae problemas como: concurrencia, transacciones, persistencia, seguridad, entre otras, lo cual permite que éste se centre en la lógica del negocio.
- **Session bean:** Para acceder a una aplicación que está desplegada en el un servidor, el cliente invoca métodos de session bean. Está dividida en tres tipos de session bean:
 - **Stateful:** El estado de la sesión es guardado, es decir, para cada usuario se va a tener un estado dependiendo de su interacción con el bean y las invocaciones a los métodos van a depender de este estado.
 - **Stateless:** A diferencia de stateful, este tipo de session bean no maneja variables que guarden datos sobre el estado de la sesión de cada usuario, el método de invocación cumple una función genérica para todos los clientes.
 - **Singleton:** Este session bean se usa cuando el estado de la sesión necesita ser compartido a través de la aplicación, ya que cumple una característica particular y es que es instanciado una sola vez durante la vida de la aplicación cuando ésta inicia, lo que le permite inicializar tareas de la aplicación, así como también finalizarlas cuando sea necesario ya que este session bean operará durante el transcurso de la aplicación.

- **Connection pool:** Se entiende como el manejo de un grupo de conexiones abiertas y reutilizables a una base de datos, de tal forma que se puedan realizar operaciones de consultas o actualizaciones de datos.
- **JPA Named-query:** Las named queries son consultas a la base de datos estáticamente definidas, las cuales constan de una consulta predefinida y un nombre que se le da. Son usadas para tener un código más organizado separando el código Java de las consultas JPQL.

FUNCIONALIDAD

La aplicación Concesionario se basa en una aplicación empresarial a la cual solo tienen acceso los empleados de la empresa Jaidiber SA. La base de datos cuenta con una tabla empleado, la cual son los que tienen derecho a logearse y trabajar con las funcionalidades según el pedido de los automóviles. Para ello, vamos a dejar claro que la empresa Jaidiber SA es la que hace el registro de los empleados a la base de datos por aparte. Ya estando dentro de la aplicación el empleado tiene el rol de manejar las transaccionalidades de los clientes, vehículos y ventas. Registrar, Actualizar, Buscar y Borrar son las funciones específicas que tiene el empleado.

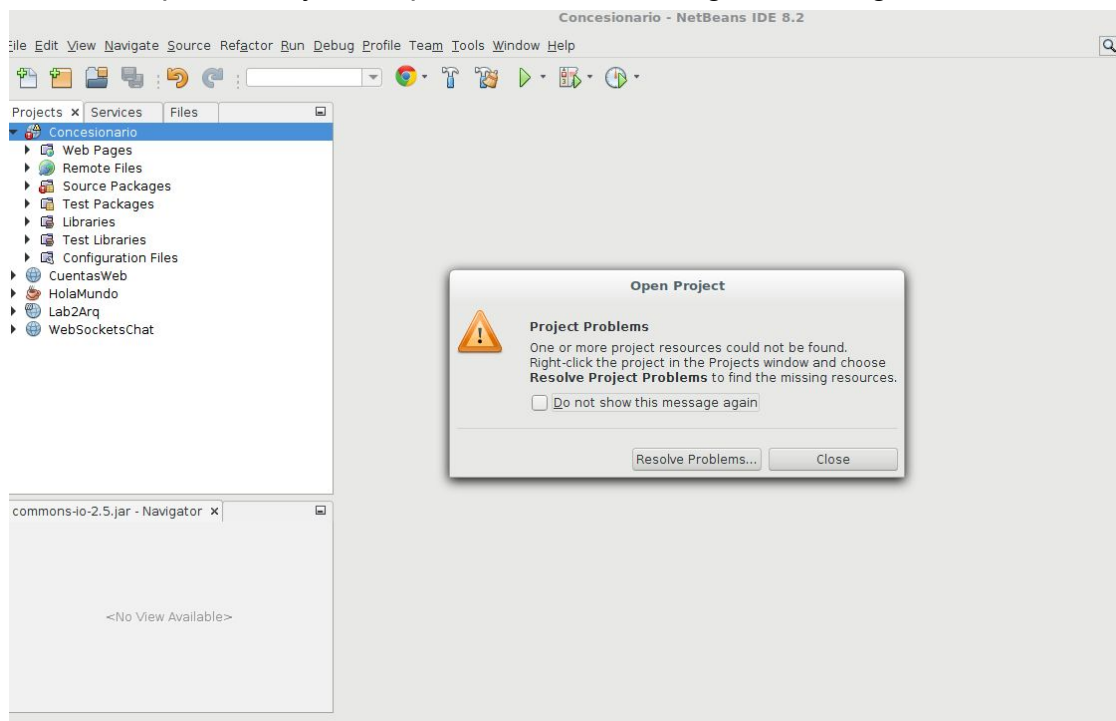
Si se desea logear, lo mejor es crear primero un registro empleado en la base de datos y ya empezar a disfrutar de la aplicación.

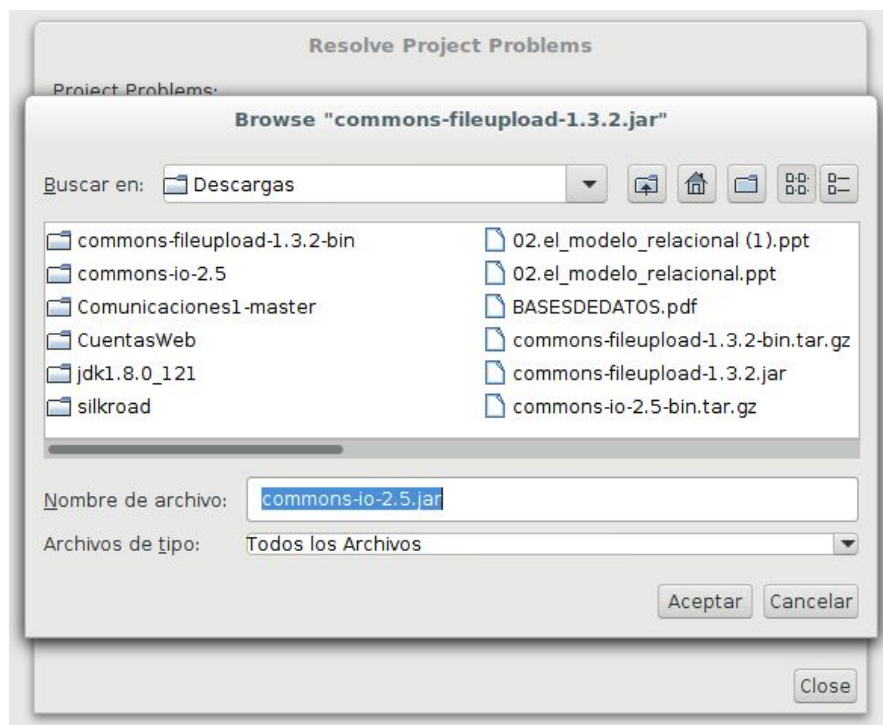
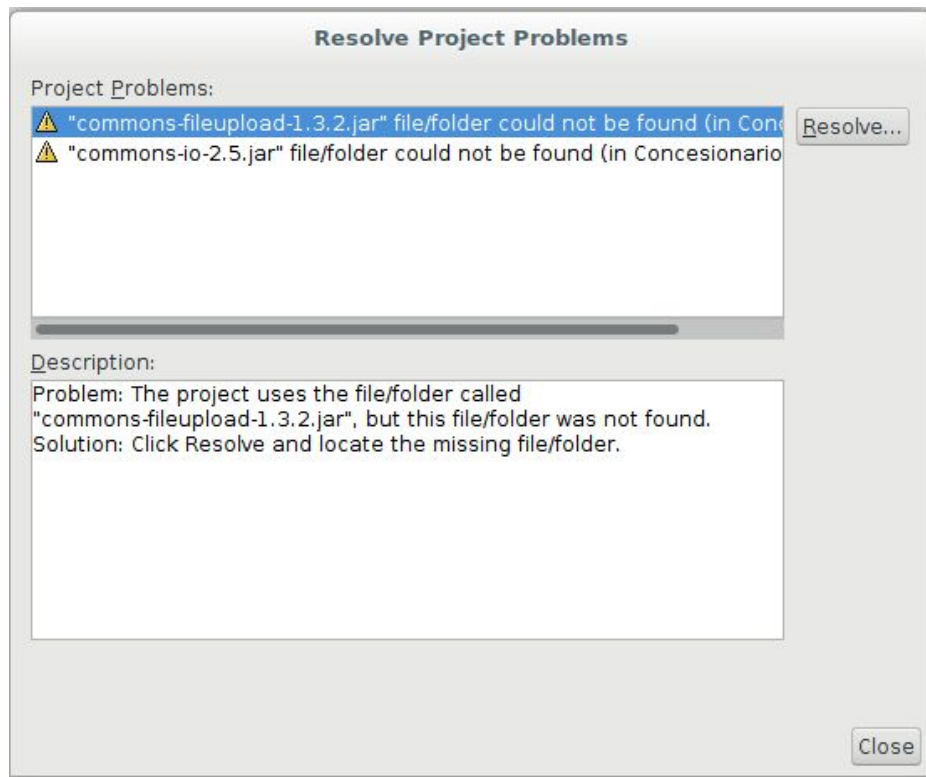
NOTA (importante): El programa requiere de dos librerías que se deben importar a la sección de librerías del netbeans en el proyecto. Son dos archivos .jar.

commons-io-2.5.jar

commons-fileupload-1.3.2.jar

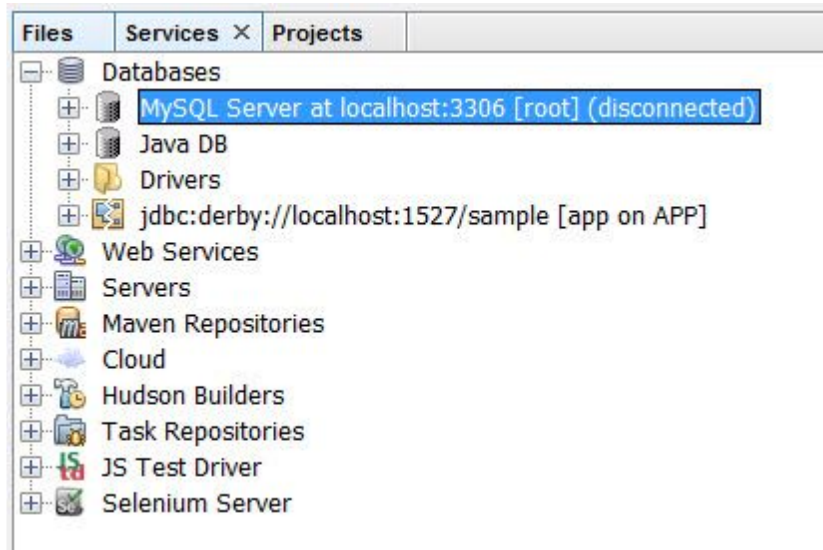
Para facilidad se enviará anexado al proyecto, y cuando se abra el proyecto le dan en resolve problems y las importan, como en la siguiente imagen.



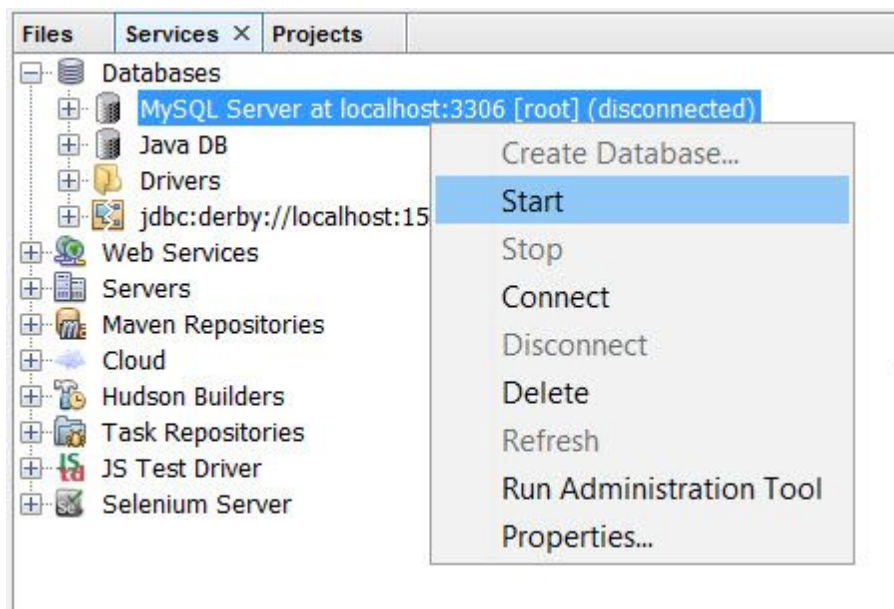


PASO A PASO

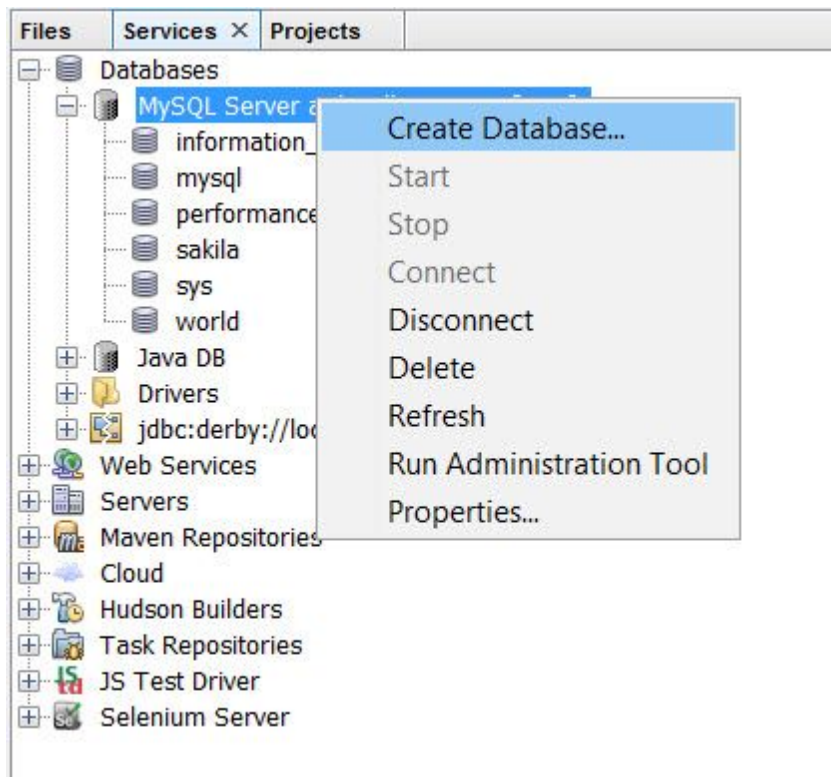
Se ejecuta NetBeans y se dirige a la pestaña Services.



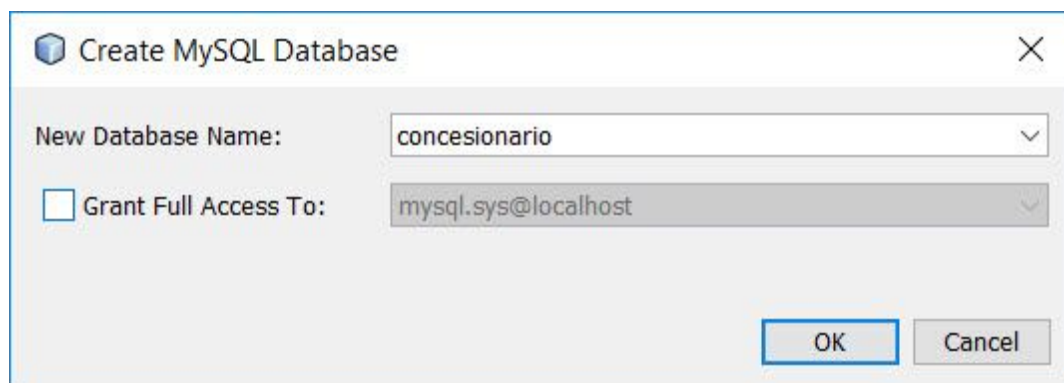
Se debe iniciar el motor de bases de datos para poder crear la base de datos y las tablas que contiene.



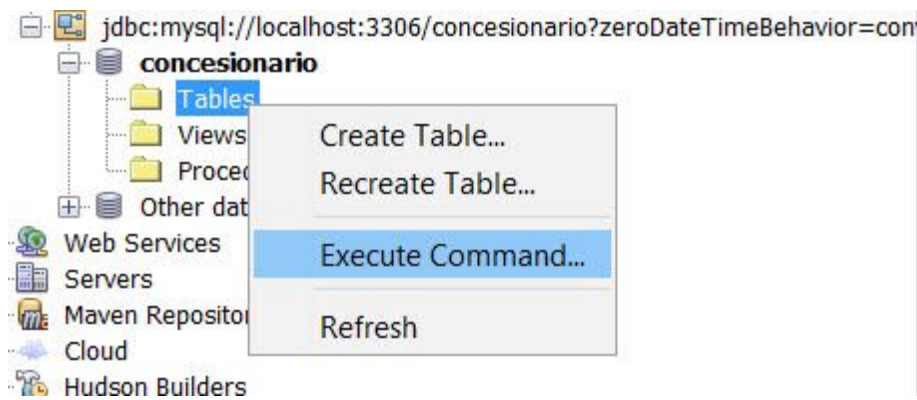
Se crea la base de datos y se pone un nombre a esta.



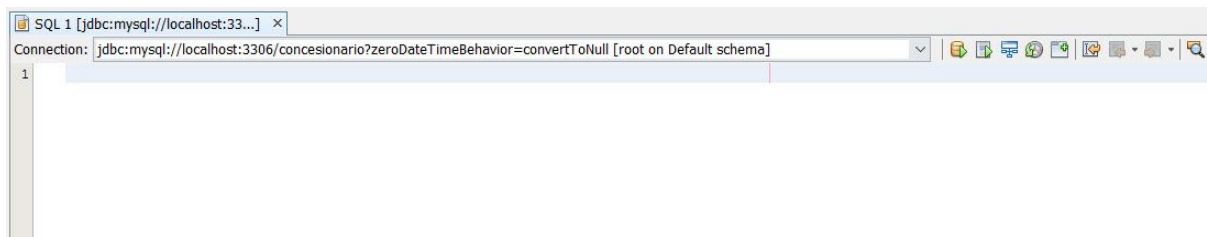
Nombre de la base de datos, en este caso la llamamos concesionario.



Nos dirigimos al nodo de la base de datos creada y en la carpeta tablas, se presiona el botón derecho del ratón y se clikea "Execute Command..."



Se abrirá una página como la siguiente, para ingresar el código SQL correspondiente para la creación de las tablas.



Se ejecuta el código SQL que muestra la imagen.

```
SQL 1 [jdbc:mysql://localhost:33...]  
Connection: jdbc:mysql://localhost:3306/concesionario?zeroDateTimeBehavior=convertToNull [root on Default schema]  
1 CREATE TABLE cliente (  
2     id VARCHAR(50) NOT NULL,  
3     tipo_id VARCHAR(20) NOT NULL,  
4     nombre VARCHAR(50) NOT NULL,  
5     direccion VARCHAR(50) NOT NULL,  
6     telefono VARCHAR(20) NOT NULL,  
7     email VARCHAR(50) NOT NULL,  
8     PRIMARY KEY (id, tipo_id));  
9  
10 CREATE TABLE empleado (  
11     id INT NOT NULL AUTO_INCREMENT,  
12     cedula VARCHAR(50) NOT NULL,  
13     tipo_cedula VARCHAR(20) NOT NULL,  
14     nombre VARCHAR(50) NOT NULL,  
15     username VARCHAR(50) NOT NULL,  
16     password VARCHAR(50) NOT NULL,  
17     email VARCHAR(50) NOT NULL,  
18     direccion VARCHAR(50) NOT NULL,  
19     telefono VARCHAR(20) NOT NULL,  
20     PRIMARY KEY (id));  
21  
22 CREATE TABLE vehiculo (  
23     placa VARCHAR(10) NOT NULL,  
24     marca VARCHAR(30) NOT NULL,  
25     referencia VARCHAR(30) NOT NULL,  
26     modelo VARCHAR(10) NOT NULL,  
27     color VARCHAR(20) NOT NULL,  
28     precio INT NOT NULL,  
29     imagen VARCHAR(50),  
30     PRIMARY KEY (placa));  
31  
32 CREATE TABLE venta (  
33     codigo INT NOT NULL AUTO_INCREMENT,  
34     id_vehiculo VARCHAR(10) NOT NULL,  
35     id_cliente VARCHAR(50) NOT NULL,  
36     tipo_id_cliente VARCHAR(20) NOT NULL,  
37     id_empleado INT NOT NULL,  
38     nuevo TINYINT(1) NOT NULL,  
39     fecha VARCHAR(20) NOT NULL,  
40     PRIMARY KEY (codigo));
```

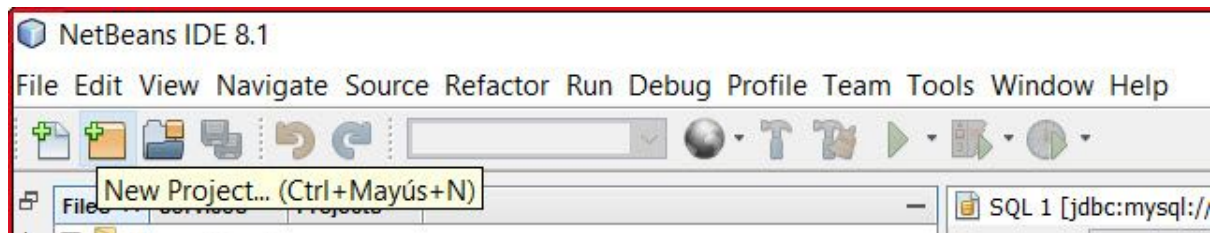
CREATE TABLE cliente (id VARCHAR(50) NOT NULL, tipo_id VARCHAR(20) NOT NULL, nombre VARCHAR(50) NOT NULL, direccion VARCHAR(50) NOT NULL, telefono VARCHAR(20) NOT NULL, email VARCHAR(50) NOT NULL, PRIMARY KEY (id, tipo_id));

CREATE TABLE empleado (id INT NOT NULL AUTO_INCREMENT, cedula VARCHAR(50) NOT NULL, tipo_cedula VARCHAR(20) NOT NULL, nombre VARCHAR(50) NOT NULL, username VARCHAR(50) NOT NULL, password VARCHAR(50) NOT NULL, email VARCHAR(50) NOT NULL, direccion VARCHAR(50) NOT NULL, telefono VARCHAR(20) NOT NULL, PRIMARY KEY (id));

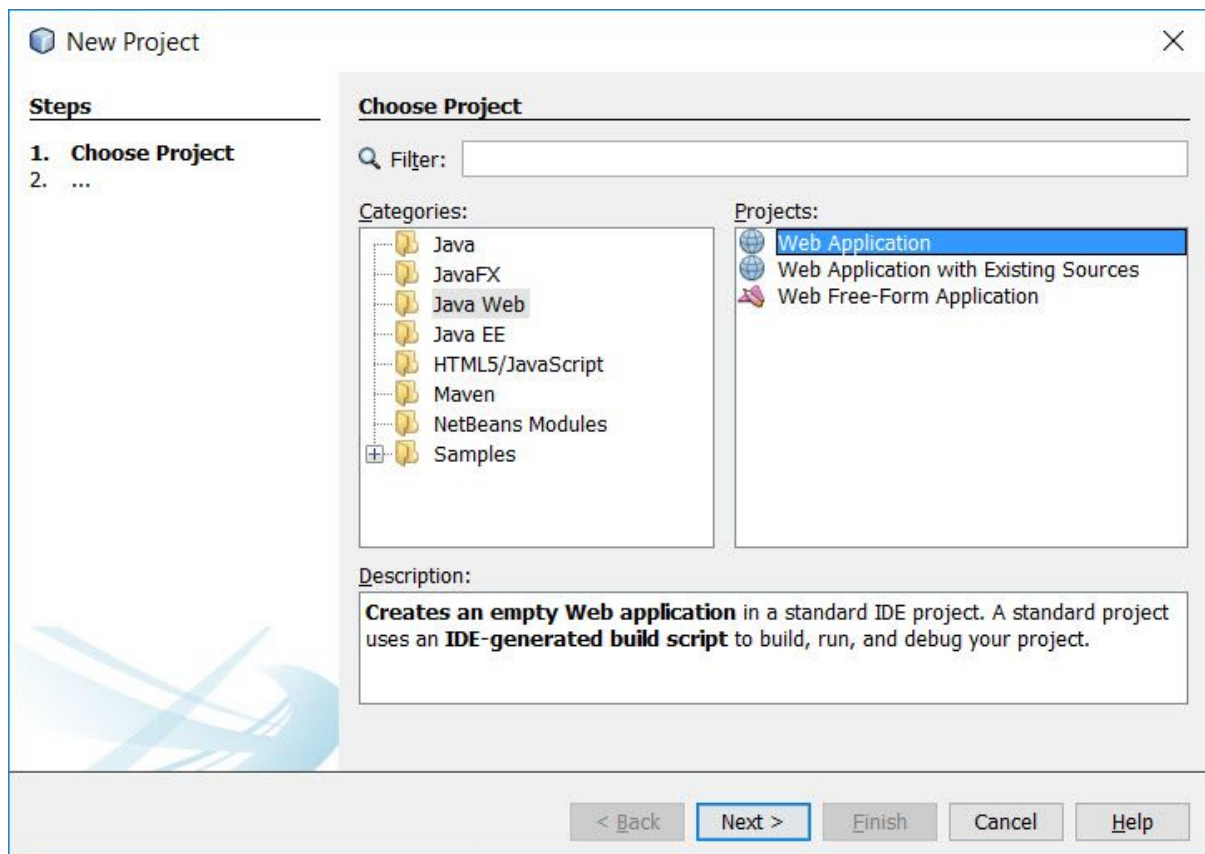
CREATE TABLE vehiculo (placa VARCHAR(10) NOT NULL, marca VARCHAR(30) NOT NULL, referencia VARCHAR(30) NOT NULL, modelo VARCHAR(10) NOT NULL, color VARCHAR(20) NOT NULL, precio INT NOT NULL, imagen LONGBLOB, PRIMARY KEY (placa));

```
CREATE TABLE venta (codigo INT NOT NULL AUTO_INCREMENT, id_vehiculo  
VARCHAR(10) NOT NULL, id_cliente VARCHAR(50) NOT NULL, tipo_id_cliente  
VARCHAR(20) NOT NULL, id_empleado INT NOT NULL, nuevo TINYINT(1) NOT  
NULL, fecha VARCHAR(20) NOT NULL, PRIMARY KEY (codigo));
```

Luego se procede a crear el proyecto.



Se selecciona dentro de la categoría Java Web la opción Web Application.



Se pone un nombre al proyecto y se continúa el proceso presionando "Next >".

New Web Application

Steps

1. Choose Project
- 2. Name and Location**
3. Server and Settings
4. Frameworks

Name and Location

Project Name:

Project Location:

Project Folder:

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

< Back **Next >** Finish Cancel Help

Se dejan los valores por defecto (Glassfish Server 4.1.1 y Java EE 7 Web) y se presiona Finish.

New Web Application

Steps

1. Choose Project
2. Name and Location
- 3. Server and Settings**
4. Frameworks

Server and Settings

Add to Enterprise Application:

















Server:

Java EE Version:

Context Path:

< Back Next > **Finish** Cancel Help

Para crear un JDBC Connection pool, se dirige a la dirección donde se tiene instalado Glassfish y se abre el documento domain.xml.

> Este equipo > Windows (C:) > Archivos de programa > glassfish-4.1.1 > glassfish > domains > domain1 > config				
Nombre	Fecha de modifica...	Tipo	Tamaño	
 admin-keyfile	19/09/2015 5:07 a....	Archivo	1 KB	
 cacerts.jks	19/09/2015 5:07 a....	Archivo JKS	80 KB	
 default-logging.properties	19/09/2015 5:07 a....	Archivo PROPERTI...	5 KB	
 default-web	19/09/2015 5:07 a....	Documento XML	50 KB	
 domain	19/09/2015 5:07 a....	Documento XML	33 KB	
 domain-passwords	19/09/2015 5:07 a....	Archivo	1 KB	
 glassfish-acc	19/09/2015 5:07 a....	Documento XML	4 KB	
 javaee.server.policy	19/09/2015 5:07 a....	Archivo POLICY	4 KB	
 keyfile	19/09/2015 5:07 a....	Archivo	2 KB	
 keystore.jks	19/09/2015 5:07 a....	Archivo JKS	5 KB	
 logging.properties	19/09/2015 5:07 a....	Archivo PROPERTI...	6 KB	
 login.conf	19/09/2015 5:07 a....	Archivo CONF	3 KB	
 restrict.server.policy	19/09/2015 5:07 a....	Archivo POLICY	3 KB	
 server.policy	19/09/2015 5:07 a....	Archivo POLICY	7 KB	
 wss-server-config-1.0	19/09/2015 5:07 a....	Documento XML	8 KB	
 wss-server-config-2.0	19/09/2015 5:07 a....	Documento XML	8 KB	

Dentro de este documento se escriben las siguientes líneas de código, después de la etiqueta <resources>.

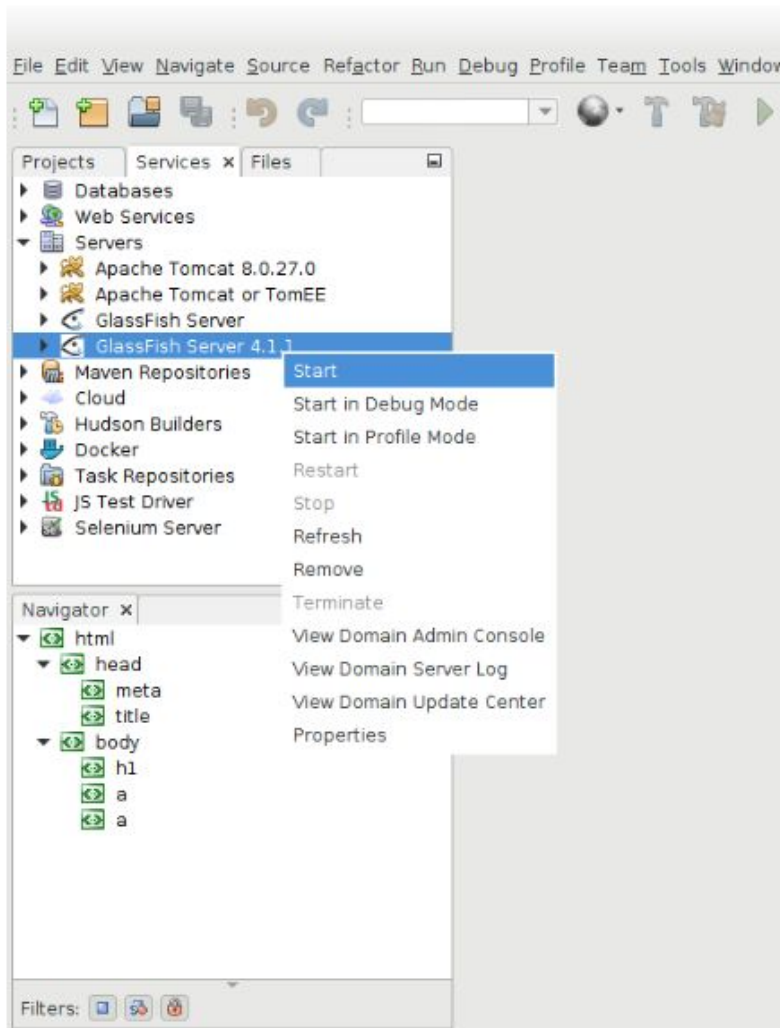

```

</security-configurations>
<system-applications />
<resources>
  <jdbc-connection-pool datasource-
classname="com.mysql.jdbc.jdbc2.optional.MysqlDataSource"
name="connection-poolConcesionario" wrap-jdbc-objects="false"
connection-validation-method="auto-commit" res-
type="javax.sql.DataSource">
    <property name="URL"
value="jdbc:mysql://localhost:3306/concesionario?
zeroDateTimeBehavior=convertToNull"></property>
    <property name="driverClass" value="com.mysql.jdbc.Driver">
</property>
    <property name="Password" value="root"></property>
    <property name="portNumber" value="3306"></property>
    <property name="databaseName" value="concesionario">
</property>
    <property name="User" value="root"></property>
    <property name="serverName" value="localhost"></property>
  </jdbc-connection-pool>

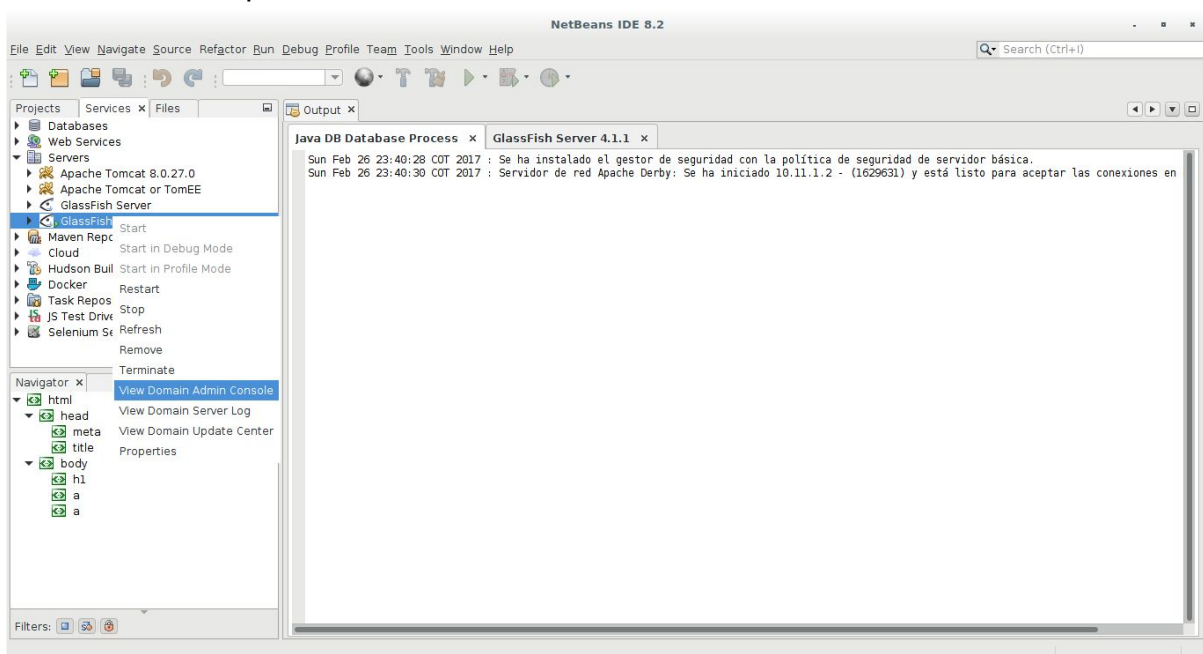
  <jdbc-resource pool-name="connection-poolAccount" jndi-
name="jdbc/account"></jdbc-resource>
  <jdbc-resource pool-name="__TimerPool" jndi-name="jdbc/
__TimerPool" object-type="system-admin" />
  <jdbc-resource pool-name="DerbyPool" jndi-name="jdbc/
default" object-type="system-all" />

```

Se inicia el servidor Glassfish haciendo click derecho sobre este en el botón Start.



Se da clic en la opción “View Domain Admin Console”.



La opción “Ping” debe estar seleccionada y el resto de parámetros se cambian según la necesidad del proyecto, en este caso se dejaron los valores por defecto.

The screenshot shows the 'Edit JDBC Connection Pool' dialog box in the GlassFish Server Open Source Edition. The 'General' tab is selected. The 'Pool Name' is 'connection-poolConcesionario'. The 'Resource Type' is 'javax.sql.DataSource'. The 'Datasource Classname' is 'com.mysql.jdbc.jdbc2.optional.MysqlDataSource'. The 'Driver Classname' is empty. The 'Ping' checkbox is checked and labeled 'Enabled'. The 'Deployment Order' is '100'. The 'Description' is empty. The 'Pool Settings' section shows 'Initial and Minimum Pool Size' as '8' connections.

Home | About... | User: admin | Domain: domain1 | Server: localhost | Help

GlassFish™ Server Open Source Edition

Nodes
Applications
Lifecycle Modules
Monitoring Data
Resources
Concurrent Resources
Connectors
JDBC
JDBC Resources
jdbc/_TimerPool
jdbc/_default
jdbc/account
jdbc/concesionario
jdbc/sample
JDBC Connection Pools
DerbyPool
SamplePool
_TimerPool
connection-poolAccou
connection-poolConce

General | Advanced | Additional Properties

Edit JDBC Connection Pool [Save] [Cancel]

Modify an existing JDBC connection pool. A JDBC connection pool is a group of reusable connections for a particular database.
[Load Defaults] [Flush] [Ping]

* Indicates required field

General Settings

Pool Name: connection-poolConcesionario

Resource Type: javax.sql.DataSource
Must be specified if the datasource class implements more than 1 of the interface.

Datasource Classname: com.mysql.jdbc.jdbc2.optional.MysqlDataSource
Vendor-specific classname that implements the DataSource and/or XADataSource APIs

Driver Classname:
Vendor-specific classname that implements the java.sql.Driver interface.

Ping: ☒ Enabled
When enabled, the pool is pinged during creation or reconfiguration to identify and warn of any erroneous values for its attributes

Deployment Order: 100
Specifies the loading order of the resource at server startup. Lower numbers are loaded first.

Description:

Pool Settings

Initial and Minimum Pool Size: 8 Connections
Minimum and initial number of connections maintained in the pool

The screenshot shows the 'Edit JDBC Connection Pool' dialog box in the GlassFish Server Open Source Edition, with the 'Advanced' tab selected. The 'Deployment Order' is '100'. The 'Description' is empty. The 'Pool Settings' section shows 'Initial and Minimum Pool Size' as '8' connections, 'Maximum Pool Size' as '32' connections, 'Pool Resize Quantity' as '2' connections, 'Idle Timeout' as '300' seconds, and 'Max Wait Time' as '60000' milliseconds. The 'Transaction' section shows 'Non Transactional Connections' as 'Enabled', 'Transaction Isolation' as a dropdown menu, and 'Isolation Level' as 'Guaranteed'.

Home | About... | User: admin | Domain: domain1 | Server: localhost | Help

GlassFish™ Server Open Source Edition

Nodes
Applications
Lifecycle Modules
Monitoring Data
Resources
Concurrent Resources
Connectors
JDBC
JDBC Resources
jdbc/_TimerPool
jdbc/_default
jdbc/account
jdbc/concesionario
jdbc/sample
JDBC Connection Pools
DerbyPool
SamplePool
_TimerPool
connection-poolAccou
connection-poolConce

General | Advanced | Additional Properties

Deployment Order: 100
Specifies the loading order of the resource at server startup. Lower numbers are loaded first.

Description:

Pool Settings

Initial and Minimum Pool Size: 8 Connections
Minimum and initial number of connections maintained in the pool

Maximum Pool Size: 32 Connections
Maximum number of connections that can be created to satisfy client requests

Pool Resize Quantity: 2 Connections
Number of connections to be removed when pool idle timeout expires

Idle Timeout: 300 Seconds
Maximum time that connection can remain idle in the pool

Max Wait Time: 60000 Milliseconds
Amount of time caller waits before connection timeout is sent

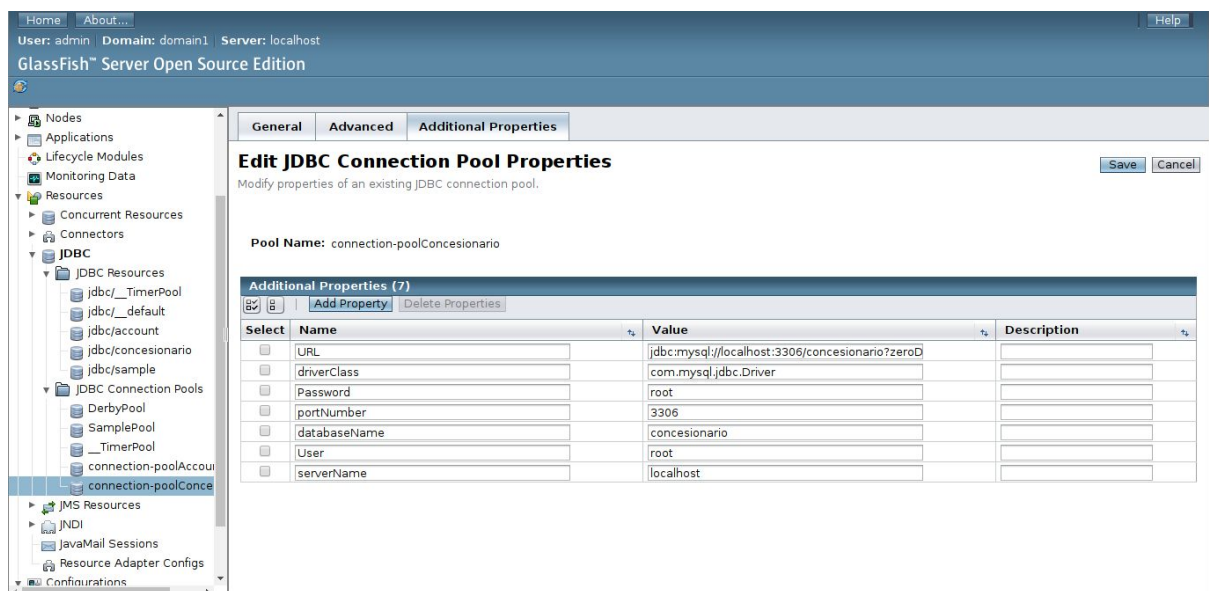
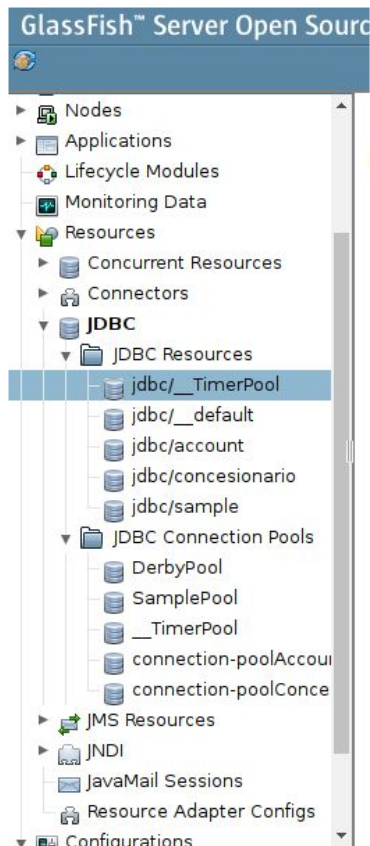
Transaction

Non Transactional Connections: ☒ Enabled
Returns non-transactional connections

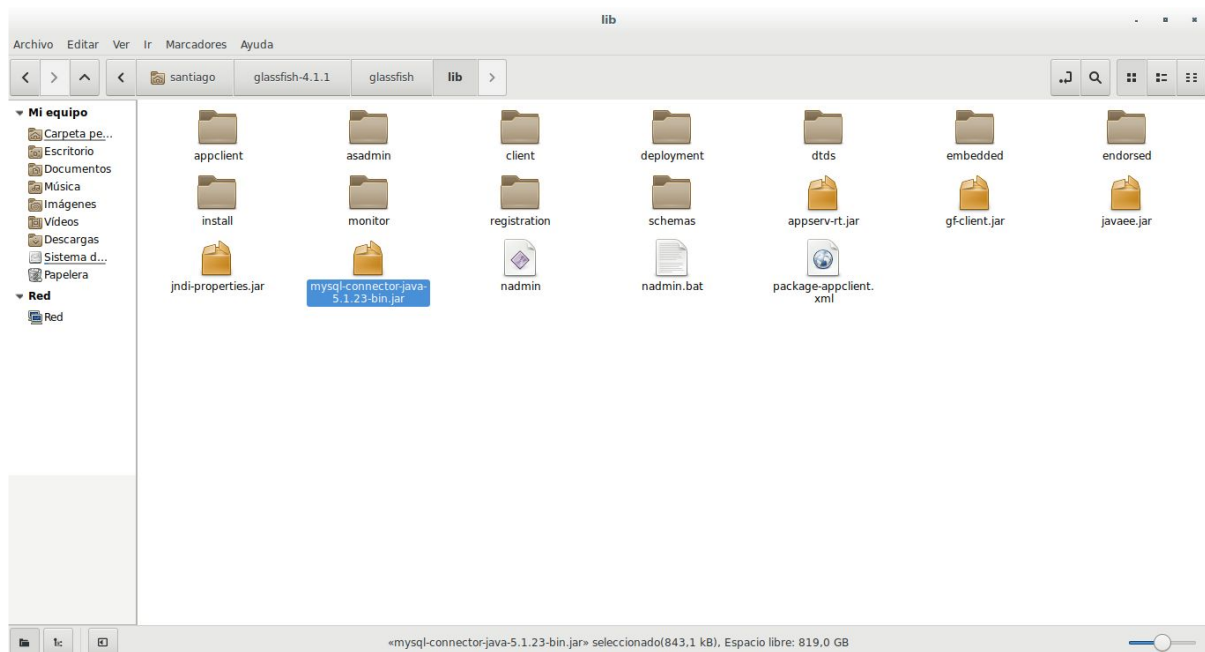
Transaction Isolation:
If unspecified, use default level for JDBC Driver

Isolation Level: ☒ Guaranteed
All connections use same isolation level; requires Transaction Isolation

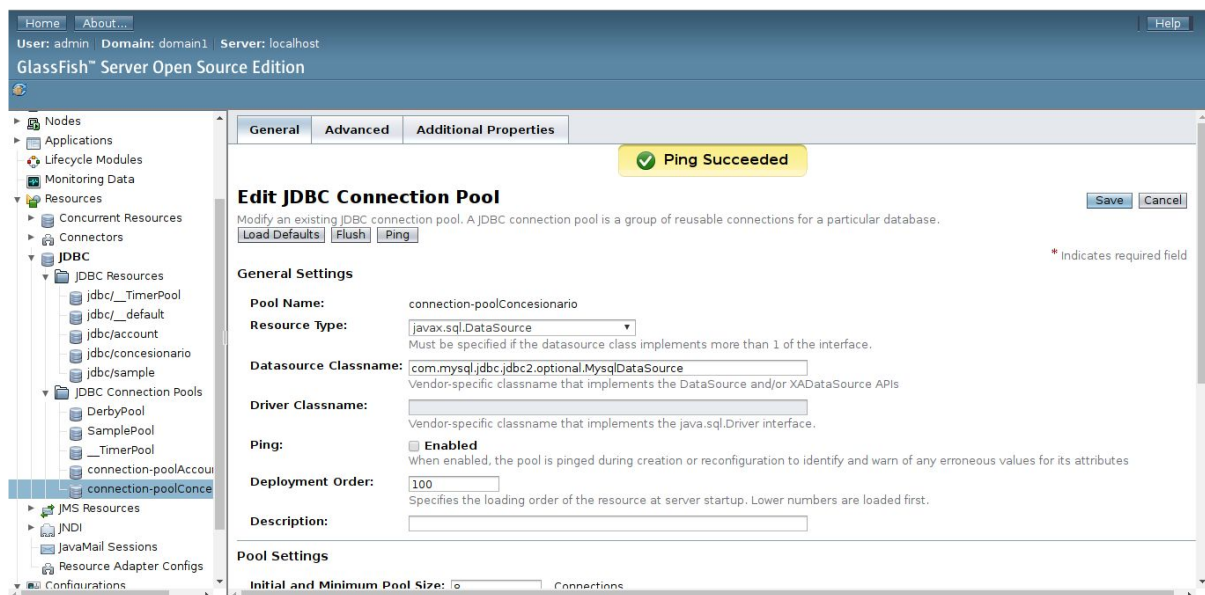
[Save] [Cancel]



Luego para verificar el pool de conecciones se pega el .jar de MySQL el cual se descarga de <http://www.java2s.com/Code/Jar/m/Downloadmysqlconnectorjava5123binjar.htm> en cual se pega en la carpeta librerias de Glassfish donde se encuentre instalado el servidor.



Una vez pegado el .jar de MySQL ya es posible realizar el pool de conexiones.



Después de que se han guardado los datos, se reinicia el servidor de aplicaciones desde Netbeans, luego se abre la consola de administración y se verifica que se hayan guardado los diferentes cambios que se realizaron en el pool de conexiones JDBC.

Edit JDBC Resource

Edit an existing JDBC data source.

[Load Defaults](#)

JNDI Name: jdbc/concesionario

Pool Name: connection-poolConcesionario ▼

Use the [JDBC Connection Pools](#) page to create new pools

Deployment Order: 100

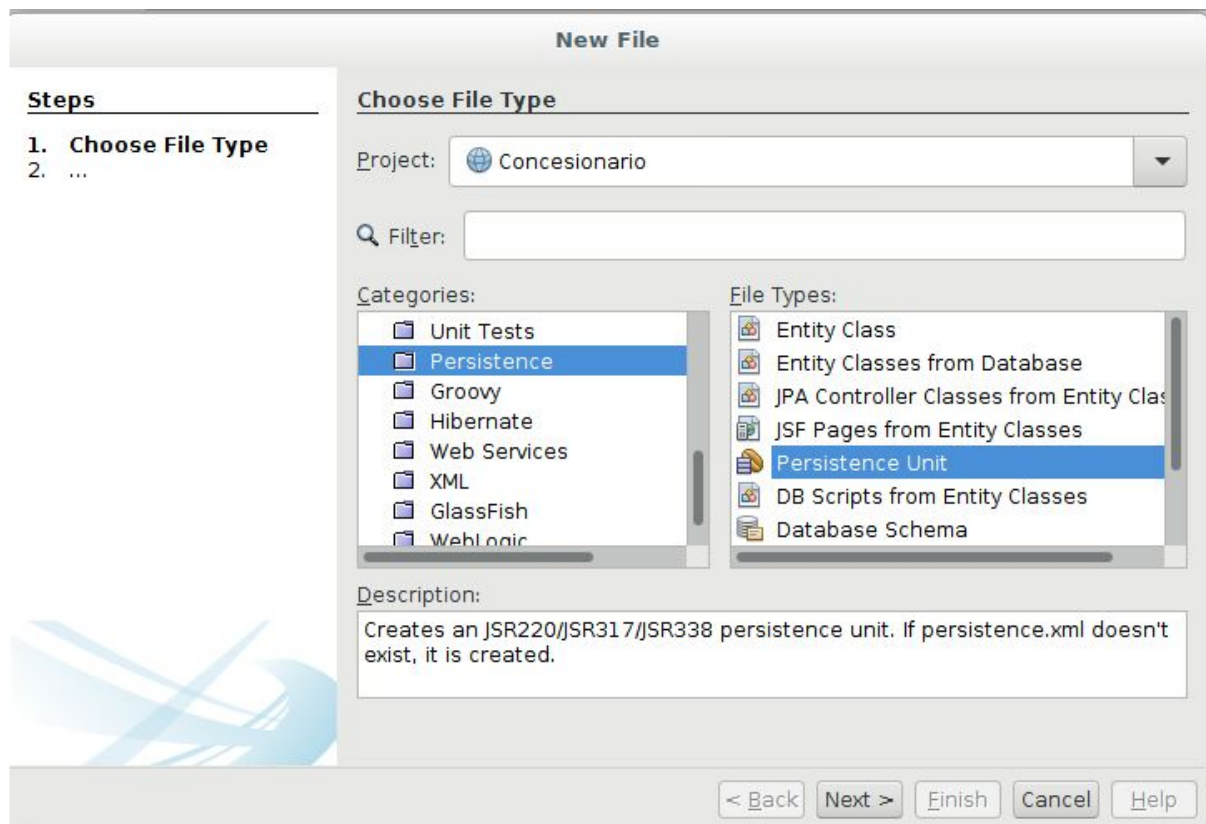
Specifies the loading order of the resource at server startup. Lower numbers are loaded first.

Description:

Status: ☒ Enabled

A continuación se agrega la unidad de persistencia.

click derecho sobre el proyecto -> new -> other -> persistence -> persistence unit.



por defecto en el nombre aparece P.U , el proveedor de persistencia será EclipseLink, se activa la API Java Transaction API, y en la estrategia de generación de tablas none.

New Persistence Unit

Steps

1. Choose File Type
2. **Provider and Database**

Provider and Database

Persistence Unit Name:

Specify the persistence provider and database for entity classes.

Persistence Provider:

Data Source:

☒ Use Java Transaction APIs

Table Generation Strategy: ☐ Create ☐ Drop and Create ☒ None

< Back Next > Finish Cancel Help

Una vez creado se podrá observar el contenido del archivo XML.

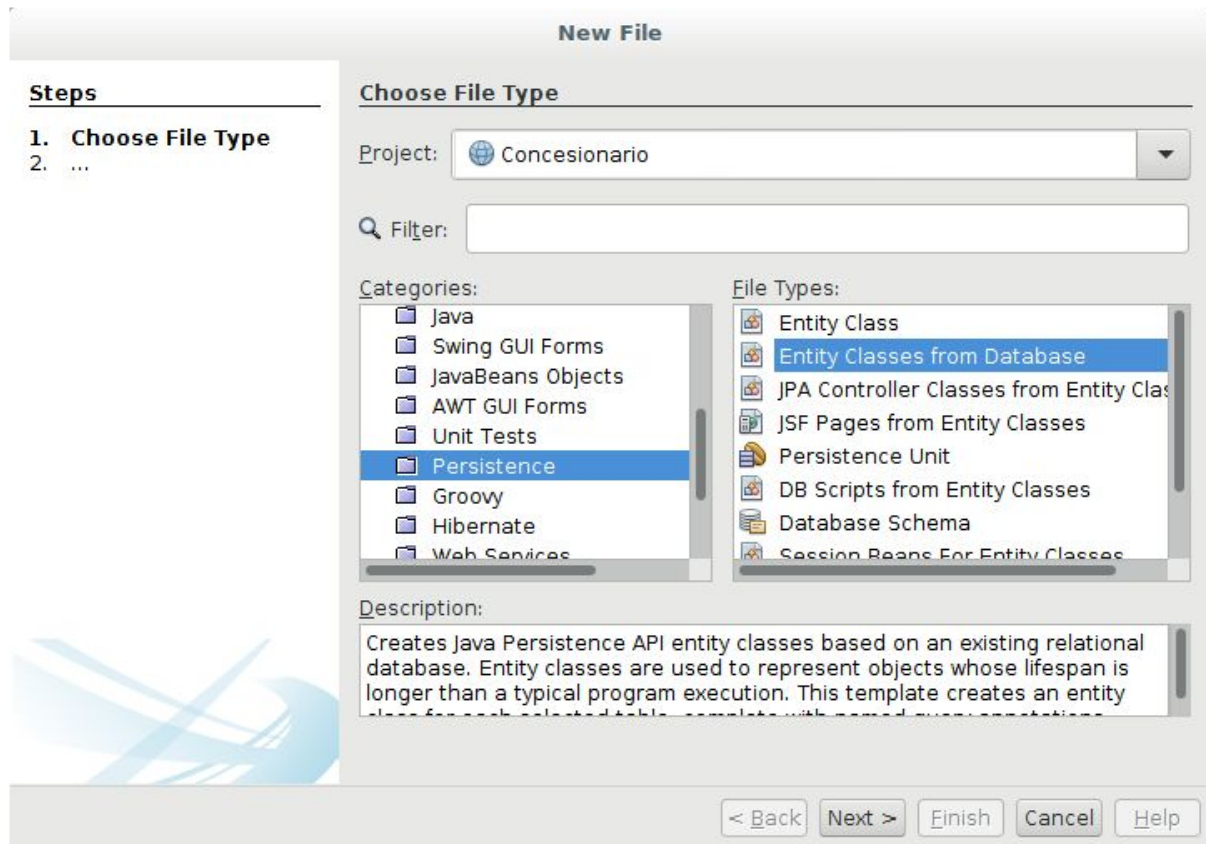
```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence.xml">
3   <persistence-unit name="ConcesionarioPU" transaction-type="JTA">
4     <jta-data-source>jdbc/concesionario</jta-data-source>
5     <exclude-unlisted-classes>>false</exclude-unlisted-classes>
6     <properties/>
7   </persistence-unit>
8 </persistence>
9

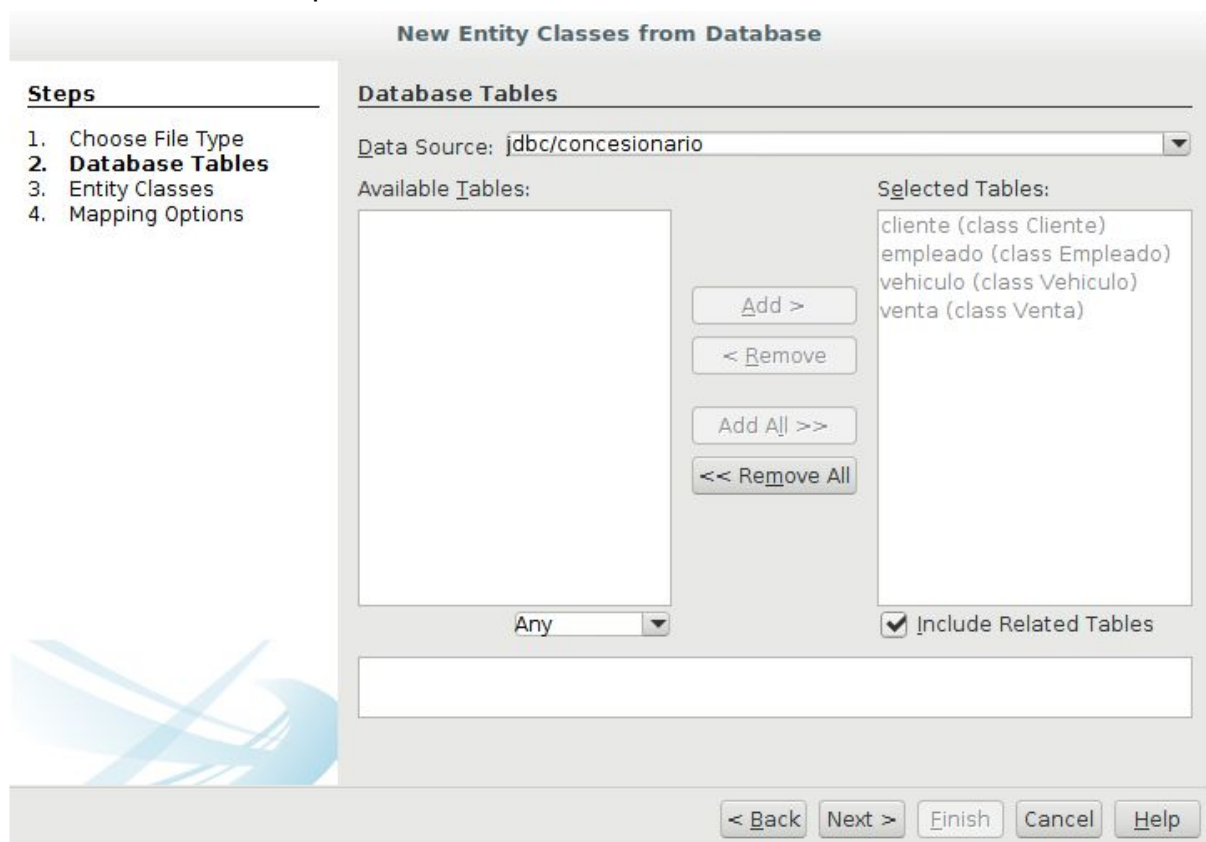
```

A continuación se creará una clase Java el cual permitirá crear los POJOS correspondientes a las tablas de la bases de datos. Esta clase representará el modelo del dominio de persistencia.

Se hace click derecho sobre el proyecto y seleccione New → Other→ Persistence → Entity Classes from Database.



Se añaden las tablas para las cuales se desea crear un POJO.



En esta ventana aparece cómo se realizará el Mapeo entre la tabla de la BD y la Clase del Modelo del dominio. el nombre del paquete se definió como com.udea.logica y se seleccionan las opciones Generate Named Query Annotation for Persistence Fields, y Generate JAXB Annotations.

New Entity Classes from Database

Steps

1. Choose File Type
2. Database Tables
- 3. Entity Classes**
4. Mapping Options

Entity Classes

Specify the names and the location of the entity classes.

Class Names:

Database Table	Class Name	Generation Type
cliente	Cliente	Update
empleado	Empleado	Update
vehiculo	Vehiculo	Update
venta	Venta	Update

Project: Concesionario

Location: Source Packages

Package: com.udea.logica

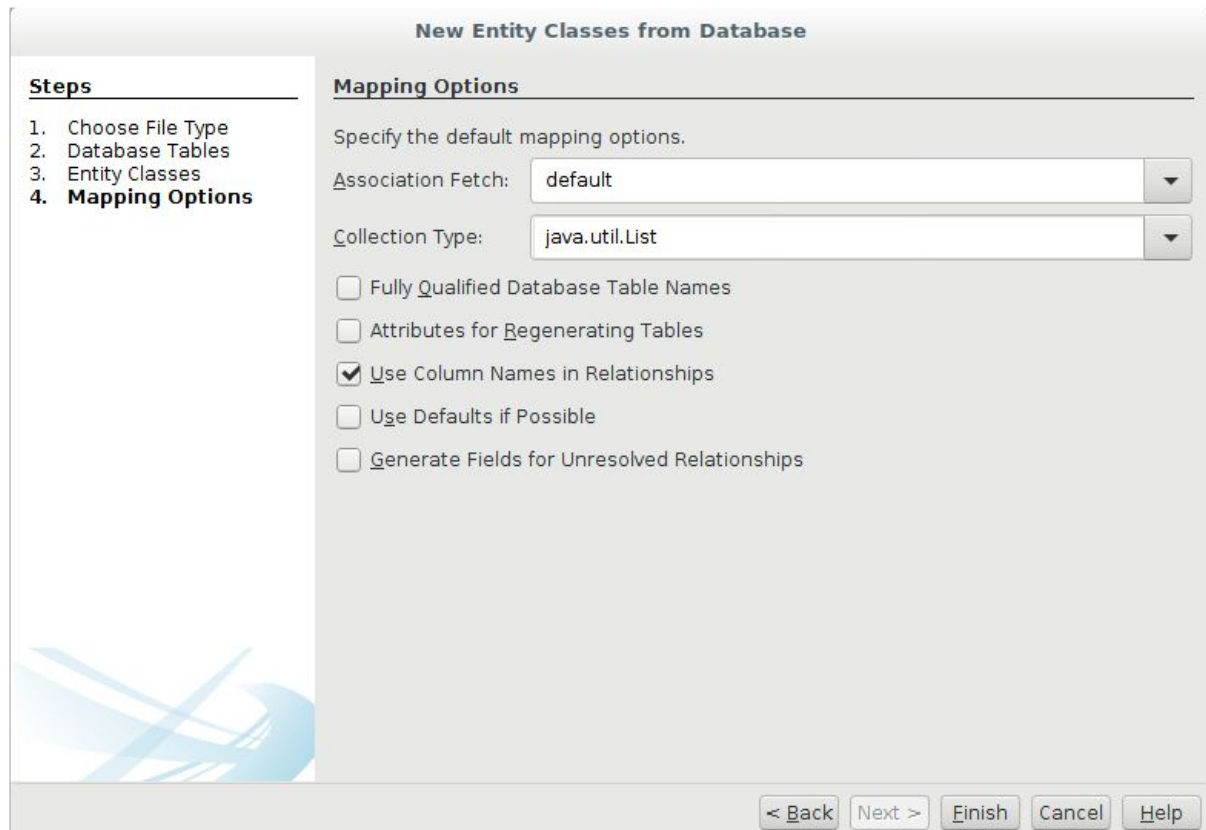
☒ Generate Named Query Annotations for Persistent Fields

☒ Generate JAXB Annotations

☐ Generate MappedSuperclasses instead of Entities

< Back Next > Finish Cancel Help

En la siguiente ventana se dejan todos los valores por defecto.



En el código de la clase del Entity Bean llamado Account podrá observar varias anotaciones JPA como **@Entity**, **@Table**, **@NamedQueries**, etc; que indican el comportamiento del Entity Bean.

Cliente.java

```

28 @Entity
29 @Table(name = "cliente")
30 @XmlRootElement
31 @NamedQueries({
32     @NamedQuery(name = "Cliente.findAll", query = "SELECT c FROM Cliente c")
33     , @NamedQuery(name = "Cliente.findById", query = "SELECT c FROM Cliente c WHERE c.clientePK.id = :id")
34     , @NamedQuery(name = "Cliente.findByTipoId", query = "SELECT c FROM Cliente c WHERE c.clientePK.tipoId = :tipoId")
35     , @NamedQuery(name = "Cliente.findByNombre", query = "SELECT c FROM Cliente c WHERE c.nombre = :nombre")
36     , @NamedQuery(name = "Cliente.findByDireccion", query = "SELECT c FROM Cliente c WHERE c.direccion = :direccion")
37     , @NamedQuery(name = "Cliente.findByTelefono", query = "SELECT c FROM Cliente c WHERE c.telefono = :telefono")
38     , @NamedQuery(name = "Cliente.findByEmail", query = "SELECT c FROM Cliente c WHERE c.email = :email"}})
39 public class Cliente implements Serializable {
40
41     @OneToMany(cascade = CascadeType.ALL, mappedBy = "cliente")
42     private List<Venta> ventaList;
43
44     private static final long serialVersionUID = 1L;
45     @EmbeddedId
46     protected ClientePK clientePK;
47     @Basic(optional = false)
48     @NotNull
49     @Size(min = 1, max = 50)
50     @Column(name = "nombre")
51     private String nombre;
52     @Basic(optional = false)
53     @NotNull
54     @Size(min = 1, max = 50)
55     @Column(name = "direccion")
56     private String direccion;
57     @Basic(optional = false)
58     @NotNull
59     @Size(min = 1, max = 20)
60     @Column(name = "telefono")

```

ClientePK.java

```

19 @Embeddable
20 public class ClientePK implements Serializable {
21
22     @Basic(optional = false)
23     @NotNull
24     @Size(min = 1, max = 50)
25     @Column(name = "id")
26     private String id;
27     @Basic(optional = false)
28     @NotNull
29     @Size(min = 1, max = 20)
30     @Column(name = "tipo_id")
31     private String tipoId;
32
33     public ClientePK() {
34     }
35
36     public ClientePK(String id, String tipoId) {
37         this.id = id;
38         this.tipoId = tipoId;
39     }
40
41     public String getId() {
42         return id;
43     }
44
45     public void setId(String id) {
46         this.id = id;
47     }
48
49     public String getTipoId() {
50         return tipoId;

```

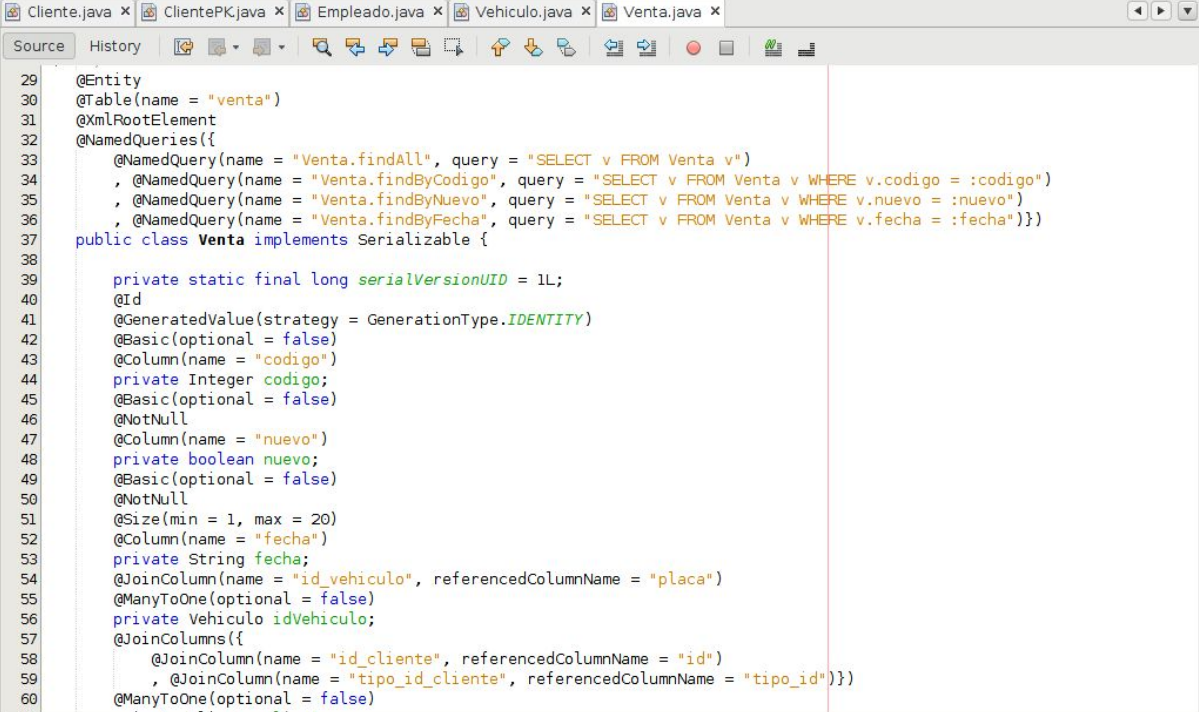
Empleado.java

```
Cliente.java x ClientePK.java x Empleado.java x
Source History
30 @Entity
31 @Table(name = "empleado")
32 @XmlRootElement
33 @NamedQueries({
34     @NamedQuery(name = "Empleado.findAll", query = "SELECT e FROM Empleado e")
35     , @NamedQuery(name = "Empleado.findById", query = "SELECT e FROM Empleado e WHERE e.id = :id")
36     , @NamedQuery(name = "Empleado.findByCedula", query = "SELECT e FROM Empleado e WHERE e.cedula = :cedula")
37     , @NamedQuery(name = "Empleado.findByTipoCedula", query = "SELECT e FROM Empleado e WHERE e.tipoCedula = :tipoCedula")
38     , @NamedQuery(name = "Empleado.findByNombre", query = "SELECT e FROM Empleado e WHERE e.nombre = :nombre")
39     , @NamedQuery(name = "Empleado.findByUsername", query = "SELECT e FROM Empleado e WHERE e.username = :username")
40     , @NamedQuery(name = "Empleado.findByPassword", query = "SELECT e FROM Empleado e WHERE e.password = :password")
41     , @NamedQuery(name = "Empleado.findByEmail", query = "SELECT e FROM Empleado e WHERE e.email = :email")
42     , @NamedQuery(name = "Empleado.findByDireccion", query = "SELECT e FROM Empleado e WHERE e.direccion = :direccion")
43     , @NamedQuery(name = "Empleado.findByTelefono", query = "SELECT e FROM Empleado e WHERE e.telefono = :telefono"))
44 public class Empleado implements Serializable {
45
46     @OneToMany(cascade = CascadeType.ALL, mappedBy = "idEmpleado")
47     private List<Venta> ventaList;
48
49     private static final long serialVersionUID = 1L;
50     @Id
51     @GeneratedValue(strategy = GenerationType.IDENTITY)
52     @Basic(optional = false)
53     @Column(name = "id")
54     private Integer id;
55     @Basic(optional = false)
56     @NotNull
57     @Size(min = 1, max = 50)
58     @Column(name = "cedula")
59     private String cedula;
60     @Basic(optional = false)
61     @NotNull
```

Vehiculo.java

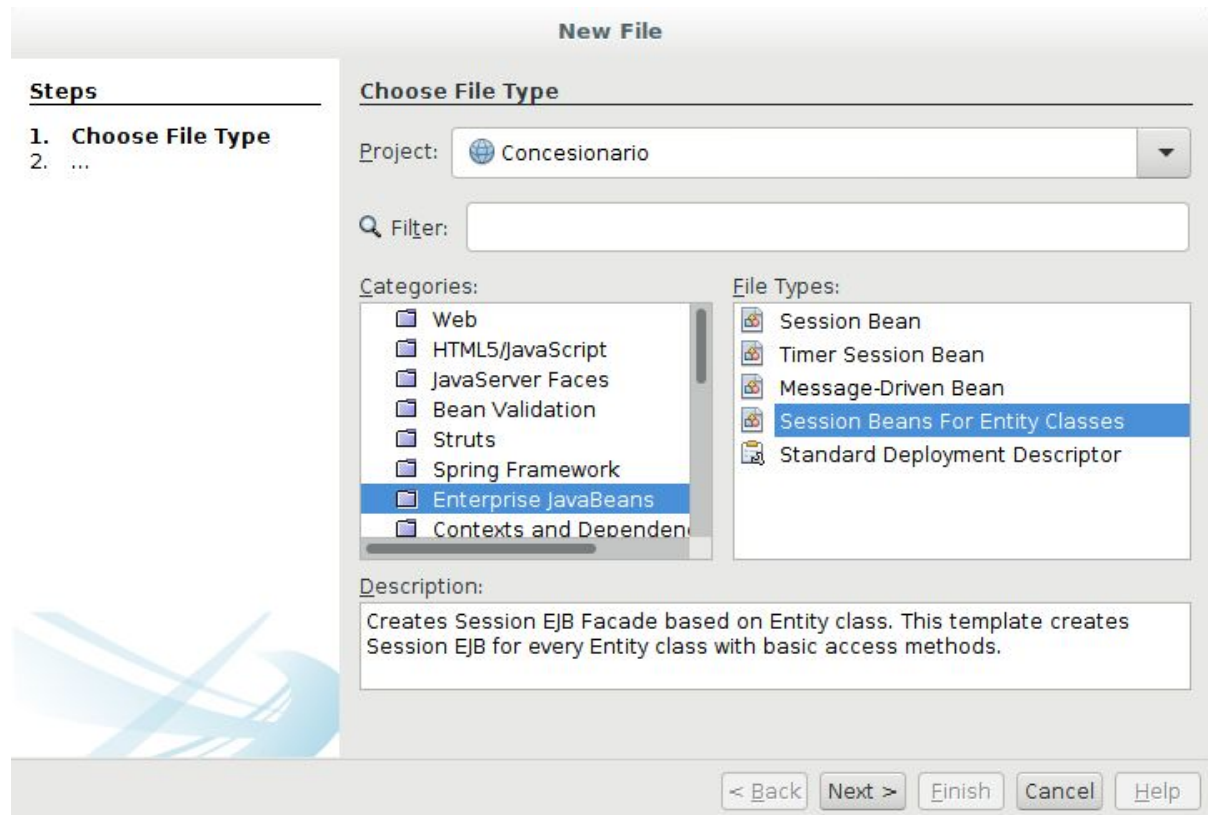
```
Cliente.java x ClientePK.java x Empleado.java x Vehiculo.java x
Source History
28 @Entity
29 @Table(name = "vehiculo")
30 @XmlRootElement
31 @NamedQueries({
32     @NamedQuery(name = "Vehiculo.findAll", query = "SELECT v FROM Vehiculo v")
33     , @NamedQuery(name = "Vehiculo.findByPlaca", query = "SELECT v FROM Vehiculo v WHERE v.placa = :placa")
34     , @NamedQuery(name = "Vehiculo.findByMarca", query = "SELECT v FROM Vehiculo v WHERE v.marca = :marca")
35     , @NamedQuery(name = "Vehiculo.findByReferencia", query = "SELECT v FROM Vehiculo v WHERE v.referencia = :referencia")
36     , @NamedQuery(name = "Vehiculo.findByModelo", query = "SELECT v FROM Vehiculo v WHERE v.modelo = :modelo")
37     , @NamedQuery(name = "Vehiculo.findByColor", query = "SELECT v FROM Vehiculo v WHERE v.color = :color")
38     , @NamedQuery(name = "Vehiculo.findByPrecio", query = "SELECT v FROM Vehiculo v WHERE v.precio = :precio")
39     , @NamedQuery(name = "Vehiculo.findByImagen", query = "SELECT v FROM Vehiculo v WHERE v.imagen = :imagen"))
40 public class Vehiculo implements Serializable {
41
42     @OneToMany(cascade = CascadeType.ALL, mappedBy = "idVehiculo")
43     private List<Venta> ventaList;
44
45     private static final long serialVersionUID = 1L;
46     @Id
47     @Basic(optional = false)
48     @NotNull
49     @Size(min = 1, max = 10)
50     @Column(name = "placa")
51     private String placa;
52     @Basic(optional = false)
53     @NotNull
54     @Size(min = 1, max = 30)
55     @Column(name = "marca")
56     private String marca;
57     @Basic(optional = false)
58     @NotNull
59     @Size(min = 1, max = 30)
```

Venta.java

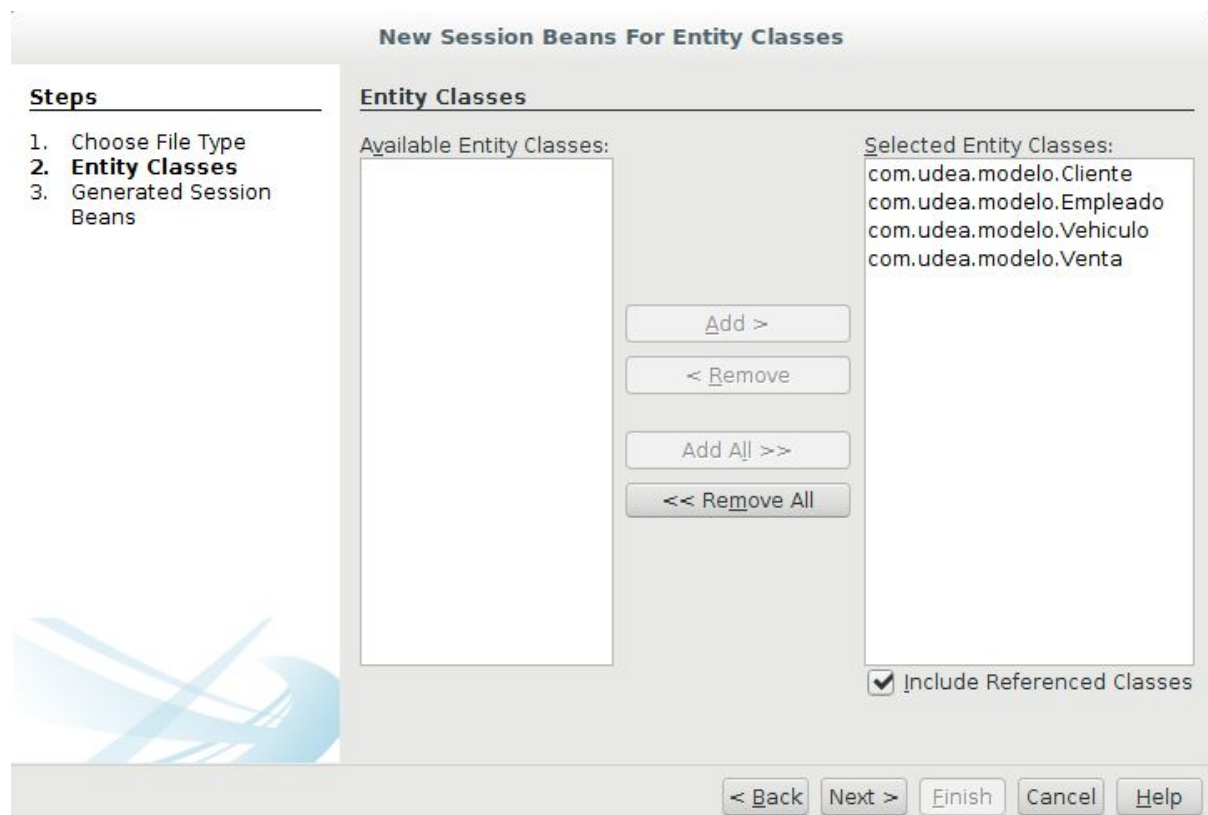


```
29 @Entity
30 @Table(name = "venta")
31 @XmlRootElement
32 @NamedQueries({
33     @NamedQuery(name = "Venta.findAll", query = "SELECT v FROM Venta v")
34     , @NamedQuery(name = "Venta.findByCodigo", query = "SELECT v FROM Venta v WHERE v.codigo = :codigo")
35     , @NamedQuery(name = "Venta.findByNuevo", query = "SELECT v FROM Venta v WHERE v.nuevo = :nuevo")
36     , @NamedQuery(name = "Venta.findByFecha", query = "SELECT v FROM Venta v WHERE v.fecha = :fecha"}})
37 public class Venta implements Serializable {
38
39     private static final long serialVersionUID = 1L;
40     @Id
41     @GeneratedValue(strategy = GenerationType.IDENTITY)
42     @Basic(optional = false)
43     @Column(name = "codigo")
44     private Integer codigo;
45     @Basic(optional = false)
46     @NotNull
47     @Column(name = "nuevo")
48     private boolean nuevo;
49     @Basic(optional = false)
50     @NotNull
51     @Size(min = 1, max = 20)
52     @Column(name = "fecha")
53     private String fecha;
54     @JoinColumn(name = "id_vehiculo", referencedColumnName = "placa")
55     @ManyToOne(optional = false)
56     private Vehiculo idVehiculo;
57     @JoinColumns({
58         @JoinColumn(name = "id_cliente", referencedColumnName = "id")
59         , @JoinColumn(name = "tipo_id_cliente", referencedColumnName = "tipo_id")})
60     @ManyToOne(optional = false)
```

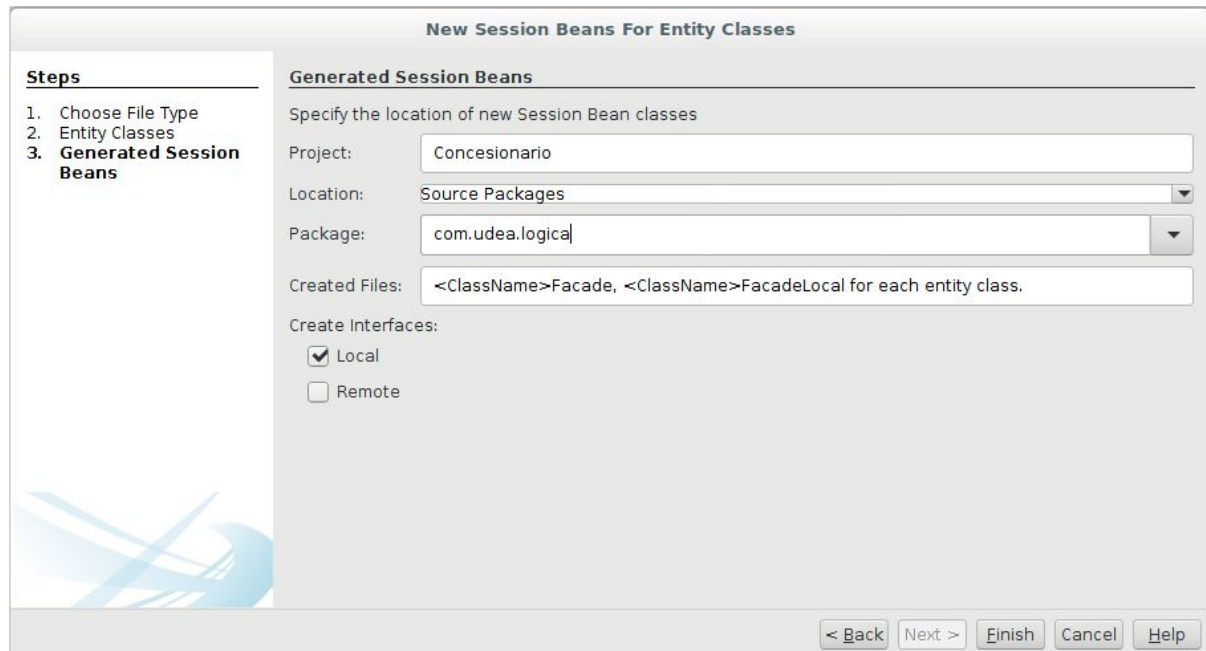
A continuación se agrega nuestra capa de lógica del negocio, Se hace click derecho sobre el proyecto y seleccione New → Other → Enterprise JavaBeans → Session Beans For Entity Classes.



Se añaden las clases entidad hacia la derecha.

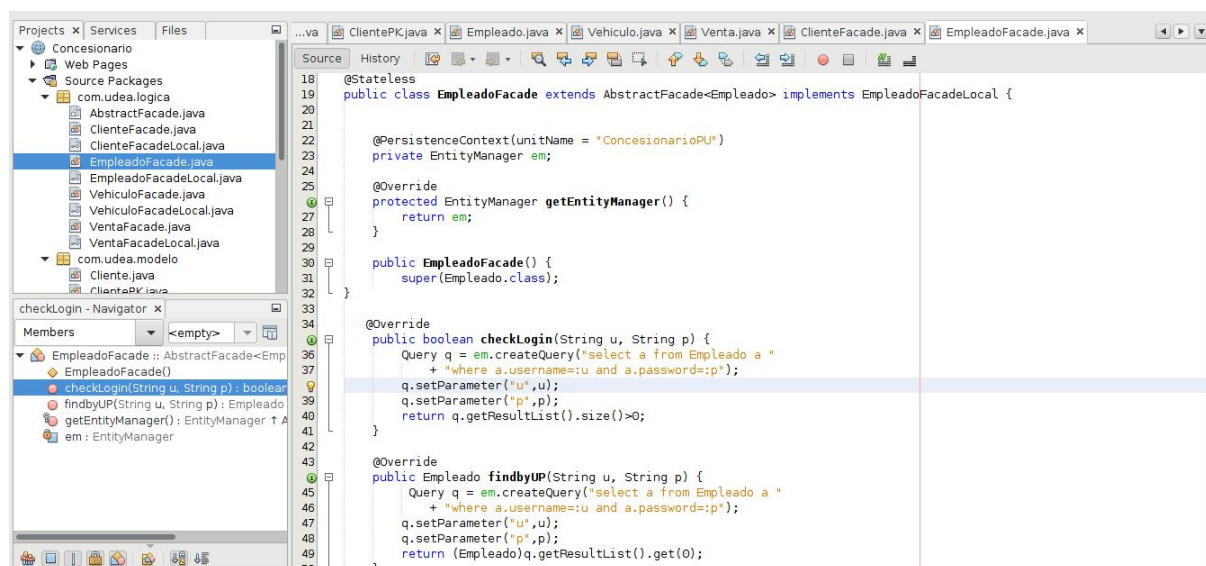


Finalmente se pone el nombre del paquete el cual en este caso lo llamamos com.udea.logica y las interfaces se crean de tipo local.

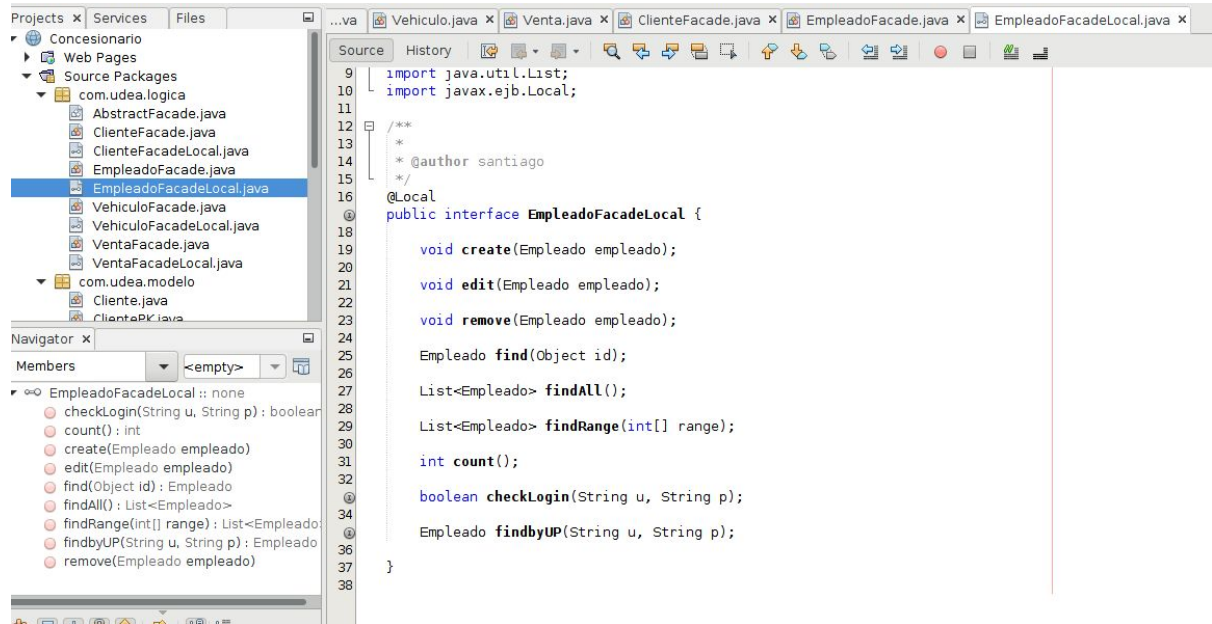


Se crea automaticamente una clase AbstractFacade.java y por cada tabla de la base de datos se crea una Clase Facade y una FacadeLocal.

EmpleadoFacade.java



EmpleadoFacadeLocal.java



Después de esto ya se pueden agregar métodos en las clases Facade, para este proyecto se insertó un método checkLogin, tal como se muestra en la siguiente figura.

Add Business Method...


Name:

Return Type:

Parameters ☒ Exceptions ☐

Name	Type	Final
u	java.lang.String	<input type="checkbox"/>
p	java.lang.String	<input type="checkbox"/>

Use in Interface: ☒ Local ☐ Remote ☐ Both

 Such method already exists

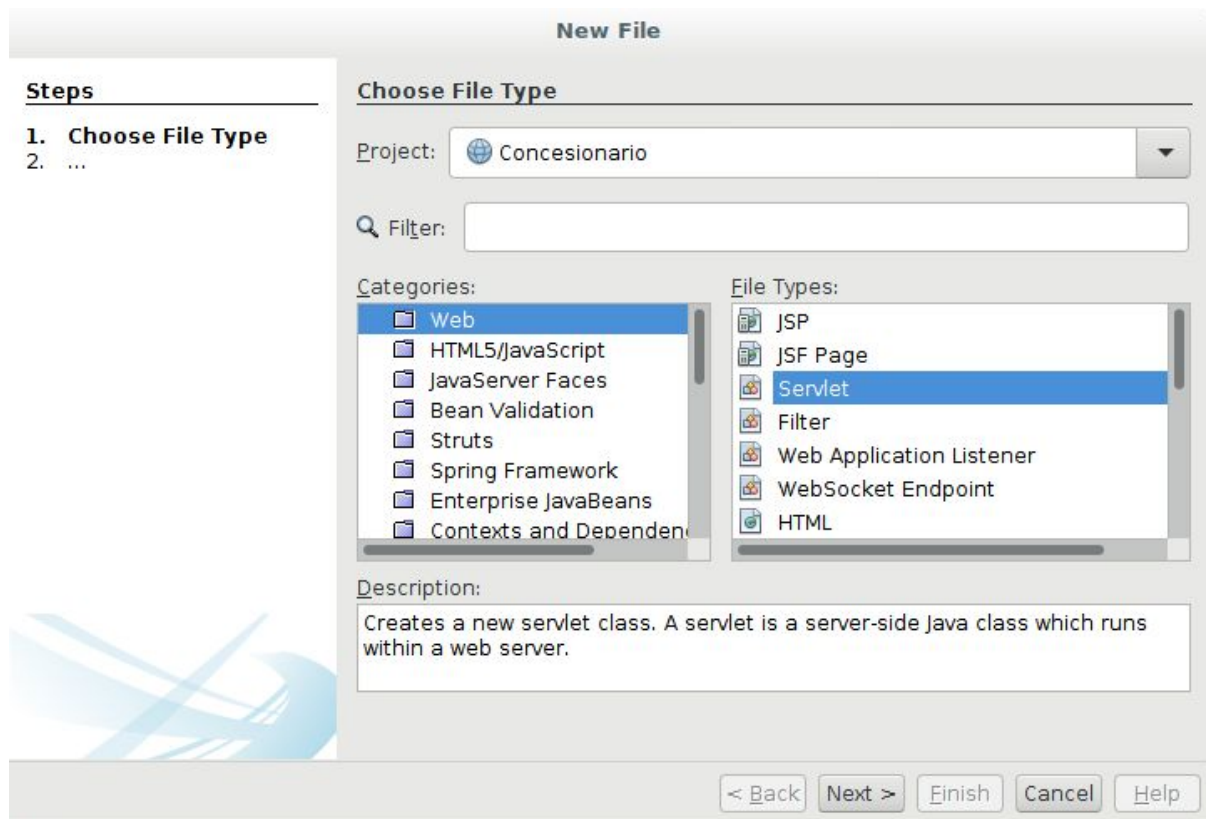
Este método retorna un booleano dependiendo del Query, es decir, si la consulta a la base de datos fue fructífera, retorna true, de lo contrario retorna false.

```
@Override
public boolean checkLogin(String u, String p) {
    Query q = em.createQuery("select a from Empleado a "
        + "where a.username=:u and a.password=:p");
    q.setParameter("u",u);
    q.setParameter("p",p);
    return q.getResultList().size()>0;
}

@Override
public Empleado findbyUP(String u, String p) {
    Query q = em.createQuery("select a from Empleado a "
        + "where a.username=:u and a.password=:p");
    q.setParameter("u",u);
    q.setParameter("p",p);
    return (Empleado)q.getResultList().get(0);
}
```

El siguiente paso fue crear los controladores del proyecto, utilizamos un controlador por cada tabla de la base de datos.

Se hace click derecho sobre el respectivo nodo del proyecto. Seleccione New -> Other -> Web -> Servlet.



Se define el nombre de la clase y el paquete en el cual se almacenará, en este caso será EmpleadoServlet y com.udea.servlet respectivamente.

New Servlet

Steps

1. Choose File Type
- 2. Name and Location**
3. Configure Servlet Deployment

Name and Location

Class Name:

Project:

Location:

Package:

Created File:

< Back Next > Finish Cancel Help

En la siguiente ventana se verifica que esté activado la opción de usar el descriptor de despliegue (web.xml).

New Servlet

Steps

1. Choose File Type
2. Name and Location
- 3. Configure Servlet Deployment**

Configure Servlet Deployment

Register the Servlet with the application by giving the Servlet an internal name (Servlet Name). Then specify patterns that identify the URLs that invoke the Servlet. Separate multiple patterns with commas.

☒ Add information to deployment descriptor (web.xml)

Class Name:

Servlet Name:

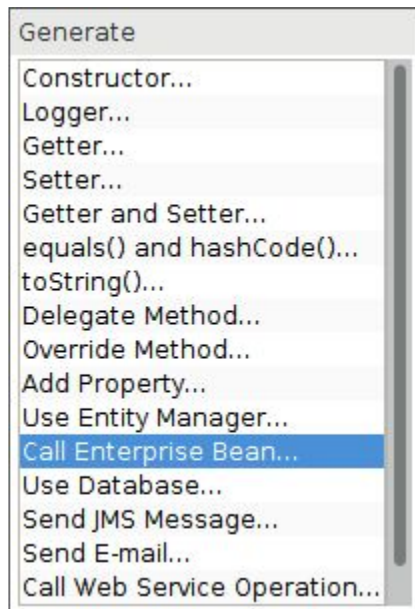
URL Pattern(s):

Initialization Parameters:

Name	Value	
<input type="text"/>		<input type="button" value="New"/>
		<input type="button" value="Edit..."/>
		<input type="button" value="Delete"/>

< Back Next > Finish Cancel Help

Sobre el editor código de la clase, haga click derecho y seleccione Insert Code luego seleccione la opción Call Enterprise Bean y agregue la Interfaz Local de la fachada de AccountFacade.



Call Enterprise Bean

Select an enterprise bean from open projects.

- ▶ WebSocketsChat
- ▶ Lab2Arq
- ▶ CuentasWeb
- ▼ Concesionario
 - ClienteFacade
 - EmpleadoFacade
 - VehiculoFacade
 - VentaFacade

Reference Name:

Referenced Interface: ☐ No interface ☒ Local ☐ Remote

```

public class EmpleadoServlet extends HttpServlet {

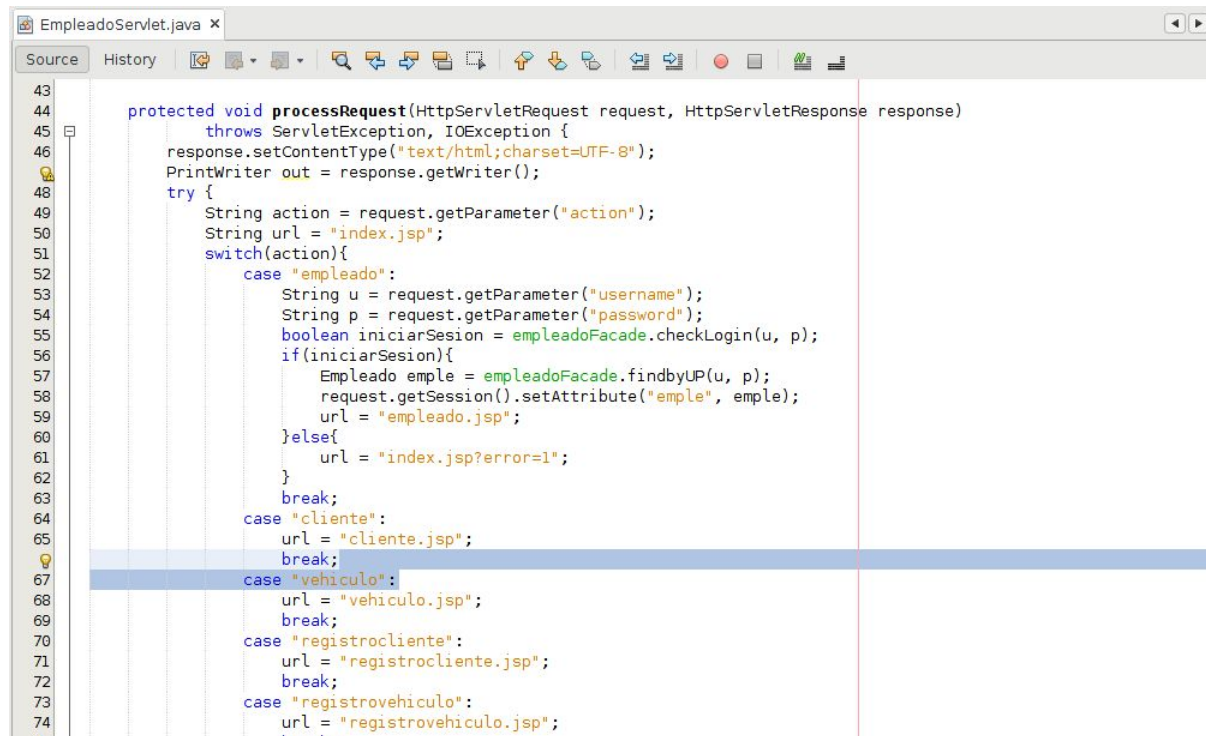
    @EJB
    private EmpleadoFacadeLocal empleadoFacade;

    /**
     * Processes requests for both HTTP GET and POST
     * methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
    }
}

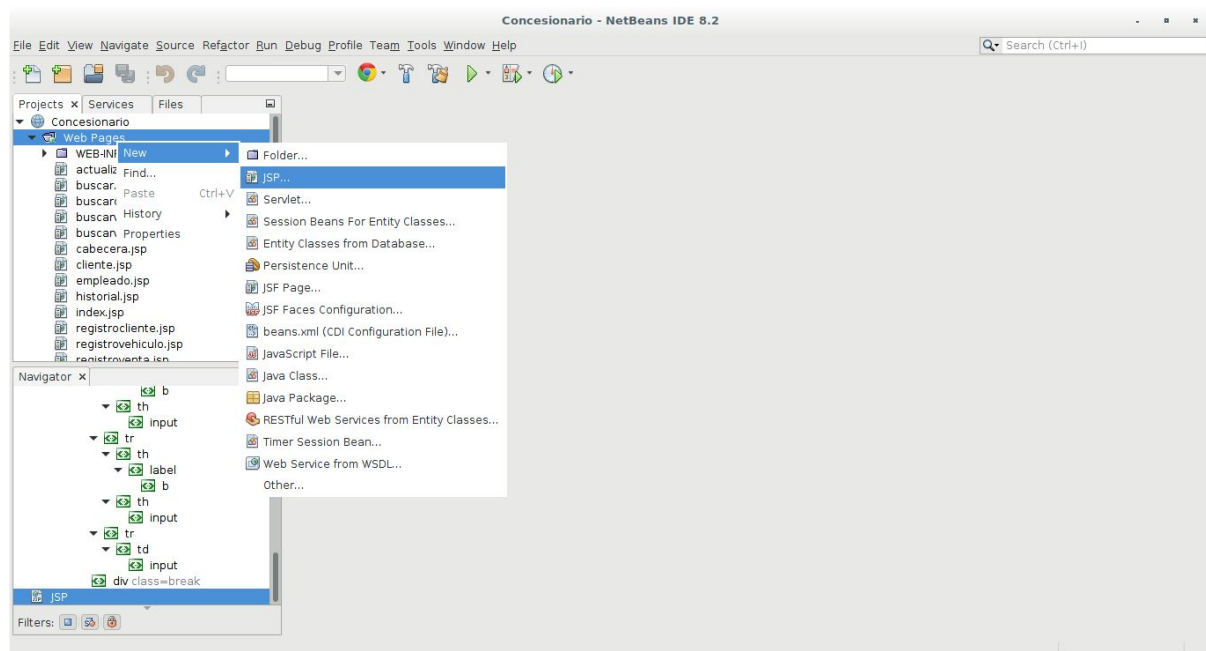
```

Después se procede a elaborar el código de que permite generar el control de los eventos (CRUD), se hace en los controladores que la lógica de la aplicación requiera utilizar.



```
43
44
45     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
46         throws ServletException, IOException {
47         response.setContentType("text/html;charset=UTF-8");
48         PrintWriter out = response.getWriter();
49         try {
50             String action = request.getParameter("action");
51             String url = "index.jsp";
52             switch(action){
53                 case "empleado":
54                     String u = request.getParameter("username");
55                     String p = request.getParameter("password");
56                     boolean iniciarSesion = empleadoFacade.checkLogin(u, p);
57                     if(iniciarSesion){
58                         Empleado emple = empleadoFacade.findbyUP(u, p);
59                         request.getSession().setAttribute("emple", emple);
60                         url = "empleado.jsp";
61                     }else{
62                         url = "index.jsp?error=1";
63                     }
64                     break;
65                 case "cliente":
66                     url = "cliente.jsp";
67                     break;
68                 case "vehiculo":
69                     url = "vehiculo.jsp";
70                     break;
71                 case "registrocliente":
72                     url = "registrocliente.jsp";
73                     break;
74                 case "registrovehiculo":
75                     url = "registrovehiculo.jsp";
76                     break;
77             }
78             out.print(url);
79         } catch (Exception ex) {
80             out.print(ex.getMessage());
81         }
82     }
83 }
```

Finalmente se procedió a crear las vistas del proyecto, de la siguiente manera:
Sobre el nodo principal del proyecto haga click derecho luego seleccione New -> Other -> Web -> JSP page



Se coloca el nombre de la vis

New JSP

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

File Name:

Project:

Location:

Folder:

Browse...

Created File:

Options:

- ☒ JSP File (Standard Syntax) ☐ Create as a JSP Segment
- ☐ JSP Document (XML Syntax)

Description:

A JSP file using JSP standard syntax.

< Back Next > Finish Cancel Help

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Projects x Services Files

Concesionario

- Web Pages
 - WEB-INF
 - actualizarcliente.jsp
 - buscar.jsp
 - buscarcliente.jsp
 - buscarveh.jsp
 - buscarvehiculo.jsp
 - cabecera.jsp
 - cliente.jsp
 - empleado.jsp
 - historial.jsp
 - index.jsp
 - registrocliente.jsp
 - registrovehiculo.jsp
 - registroventa.jsp

Navigator x

- CSS
 - Classes
 - .break
 - .container
 - .well
- HTML
 - html
 - head
 - meta
 - title
 - body
 - font
 - h1

Filters: [Icons]

Source History

```
1 <%@page contentType="text/html" pageEncoding="UTF-8"%>
2 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
3
4 <!DOCTYPE html>
5 <html>
6 <head>
7 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
8 <title>JSP Page</title>
9 </head>
10 <body>
11 <c:if test="${param.error==1}">
12 <font color="red">Usuario Invalido. Intentelo de nuevo</font>
13 </c:if>
14 <h1>Hola mundo!</h1>
15 <div class="container well">
16
17 <div align="center">
18 <h1>Ingresar</h1>
19 <form action="EmpleadoServlet?action=empleado" method="post">
20 <table>
21 <tr>
22 <th><label><b>Usuario: </b></label></th>
23 <th>
24 <input type="text" placeholder="Nombre de usuario" name="username" required="" />
25 </th>
26 </tr>
27 <tr>
28 <th><label><b>Contraseña: </b></label></th>
29 <th>
30 <input type="password" placeholder="Contraseña" name="password" required="" />
31 </th>
32 </tr>
33 </table>
34 </div>
35 </div>
36
37 </body>
38 </html>
```

HERRAMIENTAS NECESARIAS

Para el desarrollo de este aplicativo, se requiere un sistema operativo, un entorno de desarrollo integrado (IDE), el lenguaje de programación JAVA, un gestor de base de datos, el servidor de aplicaciones Glassfish, y un navegador que soporte HTML5.

Sistema Operativo:

Windows 7 o superior

Linux

Mac OS

Al ser JAVA un lenguaje de programación con máquina virtual puede ser utilizado en cualquier sistema operativo sin problema alguno.

Entorno de Desarrollo Integrado:

Para completar con éxito el desarrollo, se usó el IDE NetBeans, gracias a su facilidad de uso, y de sus complementos preinstalados, que ahorran esfuerzo, generando de manera automatizada, líneas de código en el momento de programar. Además se ha usado porque es un software gratuito.

Lenguaje de Programación(JAVA):

Se ha utilizado Java por ser un lenguaje de alto nivel y por ser el más utilizado en el mundo. Además, Java Enterprise Edition nos ofrece una plataforma con grandes utilidades para el desarrollo de aplicaciones empresariales.

Para usar el lenguaje Java se requiere instalar el software JDK (recomendamos la versión 8).

Gestor de Base de Datos:

Se utiliza MySQL al ser un open source, y por su gran seguridad en el mapeo de los datos (guardar información en el disco duro).

Servidor de aplicaciones(GlassFish):

Se ha utilizado GlassFish por brindar una gran transaccionalidad a la base de datos y dar concurrencia a ella. Además de tener el beneficio de ser un software gratuito.

Navegador web:

Se deja a elección del usuario final, la única condición que debe cumplir es soportar HTML5. Se recomienda el uso de Google Chrome, ya que soporta con facilidad este lenguaje de enmarquetado y otras tecnologías de desarrollo web.

Front-End:

El framework utilizado fue Materialize el cual está desarrollado en SAAS y hace uso de HTML5, CSS3 y JavaScript. Las vistas que se presentarán a continuación son básicamente la percepción que va a tener el usuario final con nuestra aplicación.

Primero tenemos el login, el cual deja observar sobre la parte superior un “header” con el nombre de una supuesta empress, luego están los formularios, donde sólo pueden ingresar los Empleados que ya se encuentren registrados en la base de datos. Ya en la parte inferior nos encontramos con un footer el cual contiene información relevante sobre la empresa.

Concesionaria Jaidiber S.A

Ingresar

Username:

Password:

INGRESAR >

Concesionaria Jaidiber S.A

Aplicación que permite manejar la información de un concesionario de vehículos

Enlaces de interes

Site del curso

localhost:8080/Concesionario/index.jsp

Luego está registrar cliente, acá ya se ha efectuado el login por parte de empleado, y como se puede observar, sobre la parte izquierda del header hay unos botones de despliegue y por medio de ellos es posible llegar a otras vistas similares a esta como lo son “Registro venta” y “Registro Vehículo”.

Concesionaria Jaidiber S.A

Cientes Vehiculos Venta Cerrar sesión

Registrar cliente

Tipo de identificación*

Número de Identificación*

Nombres y Apellidos*

Dirección

Teléfono

E-Mail

REGISTRAR >

Concesionaria Jaidiber S.A

Aplicación que permite manejar la información de un concesionario de vehículos

Enlaces de interes

Site del curso

localhost:8080/Concesionario/index.jsp

Por último están las vistas de búsqueda, las cuales acceden a la base de datos y nos brindan toda la información allí contenida, y como si fuera poco también se pueden hacer algunas consultas básicas sobre estas, también está disponible la opción de actualizar y eliminar, dependiendo de la vista.

Concesionaria Jaidiber S.A

Clientes

Vehiculos

Venta

Cerrar sesión

Buscar cliente

Tipo de identificación*

▼

Número de Identificación

BUSCAR

Q

Tipo de documento	Número de documento	Nombres y Apellidos	Dirección	Teléfono	Email	
Cédula	12167643789	Sergio Andrés Castrillón	Cra 35 # 70-67	3356443	sergio@hotmail.com	Actualizar
NIT	234563435	Nicock Albert Gutenberg	Carrera 78 # 30-12	4534465	nicock@udea.edu.co	Actualizar
Pasaporte	5676545678	Jerónimo ALzte	calle 67 # 78-12	65784678	jero@udea.edu.co	Actualizar
C. Extranjería	989898989898	Andrew Gracciano	calle 78- # 89-32	324534	andrew@udea.edu.co	Actualizar

BIBLIOGRAFÍA

- Oracle.com. (2017). Java Servlet Technology Overview. [online] Available at: <http://www.oracle.com/technetwork/java/javaee/servlet/index.html> [Accessed 25 Feb. 2017].
- Oracle.com. (2017). Java SE Technologies - Database. [online] Available at: <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html> [Accessed 26 Feb. 2017].
- Oracle.com. (2017). Java SE - Core Technologies - Java Naming and Directory Interface (JNDI). [online] Available at: <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140184.html> [Accessed 27 Feb. 2017].
- Oracle.com. (2017). JavaServer Pages Technology. [online] Available at: <http://www.oracle.com/technetwork/java/jsp-138432.html> [Accessed 27 Feb. 2017].
- Oracle.com. (2017). JavaServer Pages Technology. [online] Available at: <http://www.oracle.com/technetwork/java/javaee/jsp/index.html> [Accessed 27 Feb. 2017].
- Es.wikipedia.org. (2017). JavaServer Pages. [online] Available at: https://es.wikipedia.org/wiki/JavaServer_Pages [Accessed 28 Feb. 2017].
- Davidmarco.es. (2017). Archivo - davidmarco.es. [online] Available at: <http://www.davidmarco.es/archivo/tutorial-jpa20> [Accessed 28 Feb. 2017].
- Ibm.com. (2017). IBM Knowledge Center. [online] Available at: http://www.ibm.com/support/knowledgecenter/es/SSRTLW_7.5.5/com.ibm.jee5.doc/topics/rjpaannotations.html [Accessed 27 Feb. 2017].
- Yuan, M. (2017). POJO Application Frameworks: Spring Vs. EJB 3.0 - O'Reilly Media. [online] Onjava.com. Available at: <http://www.onjava.com/pub/a/onjava/2005/06/29/spring-ejb3.html> [Accessed 25 Feb. 2017].
- Msdn.microsoft.com. (2017). Data Transfer Object. [online] Available at: <https://msdn.microsoft.com/en-us/library/ms978717.aspx> [Accessed 28 Feb. 2017].
- Oracle.com. (2017). Core J2EE Patterns - Data Access Object. [online] Available at: <http://www.oracle.com/technetwork/java/dataaccessobject-138824.html> [Accessed 26 Feb. 2017].
- Docs.oracle.com. (2017). What Is a Session Bean? - The Java EE 6 Tutorial. [online] Available at: <http://docs.oracle.com/javaee/6/tutorial/doc/gipjg.html> [Accessed 27 Feb. 2017].
- S.R.L, E. (2017). Base de conocimiento de Epidata : Teoría workshop de EJB 2.0. [online] Epidataconsulting.com. Available at:

<http://www.epidataconsulting.com/tikiwiki/tiki-index.php?page=Teor%C3%ADa+workshop+de+EJB+2.0> [Accessed 26 Feb. 2017].

- Objectdb.com. (2017). JPA Named Queries (@NamedQuery, @NamedQueries annotations). [online] Available at: <http://www.objectdb.com/java/jpa/query/named> [Accessed 26 Feb. 2017].