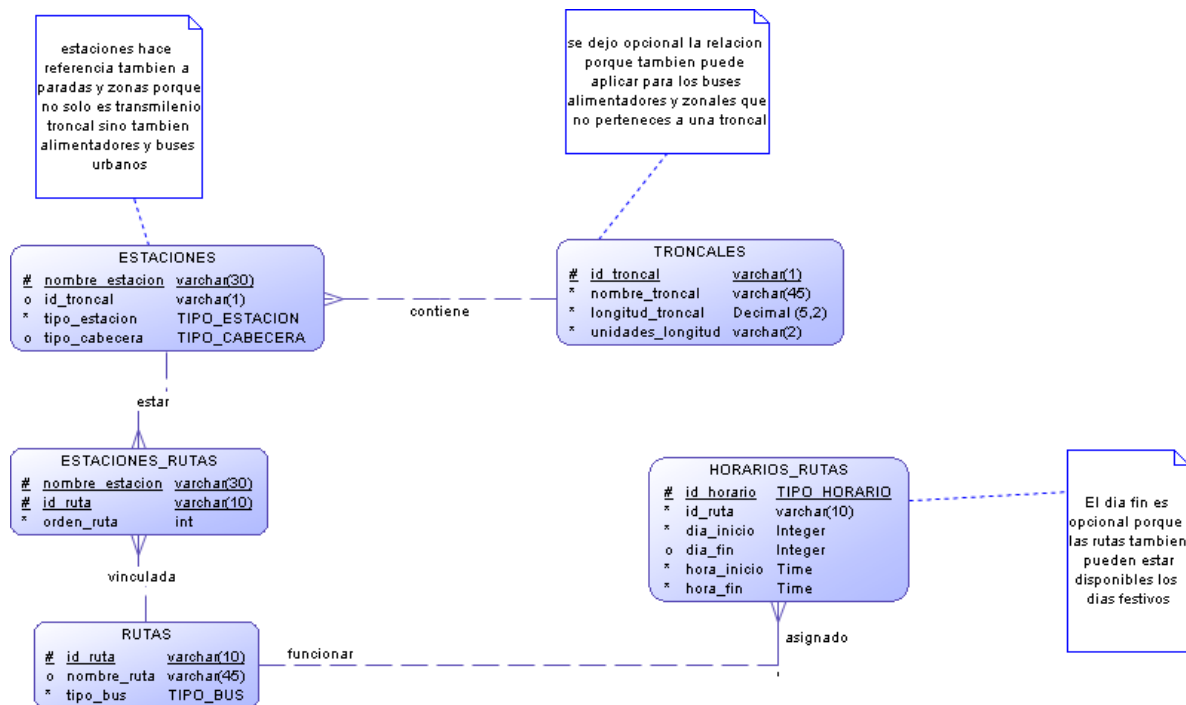


PROYECTO TRANSMILENIO PARA EL MODULO DE FRAMEWORK DE PERSISTENCIA




PRESENTADO POR: Cesar Andrés Patiño Nieto CODIGO: 201822962

Wilmer Ariel Avella Tuta CODIGO: 201823020

para empezar, es necesario modelar una base de datos donde se pueda evidenciar la infraestructura del Transmilenio para ello utilizamos el powerdesigner para modelar la base de datos

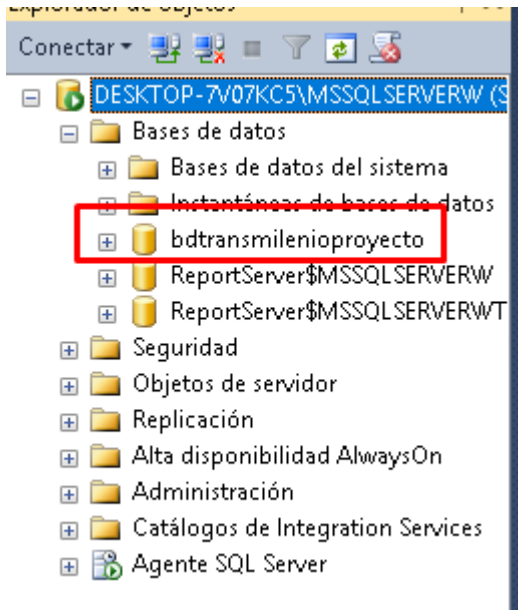


enseguida vamos a desarrollar los diferentes scripts para crear la base de datos del Transmilenio

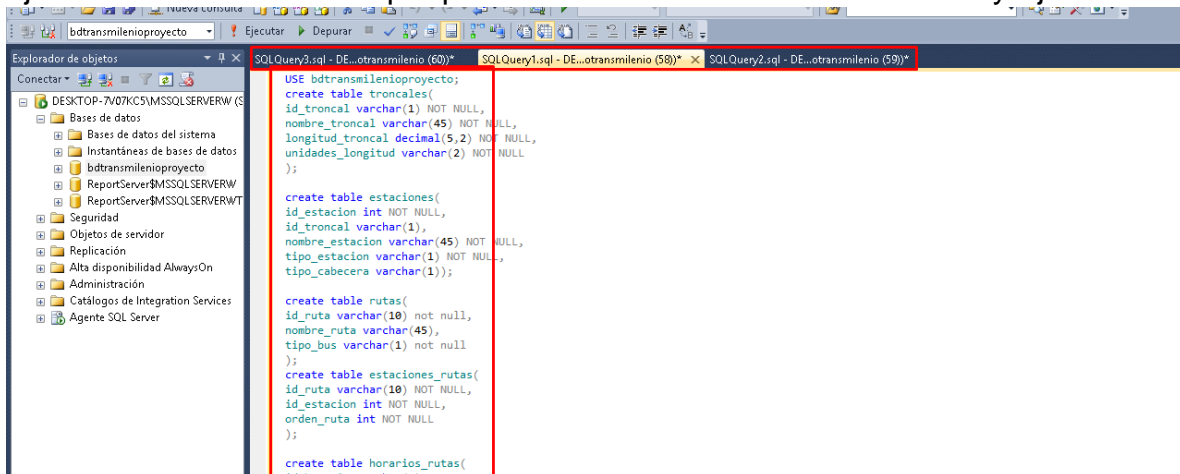
procedimientos almacenados 2019/04/03 10:43:12 m... 03/04/2019 10:43... carpeta de archivos				
	createrestriccionbdtransmilenioproyecto	03/04/2019 10:43	Archivo SQL	2 KB
	createtablesbdtransmilenioproyecto	03/04/2019 10:43	Archivo SQL	1 KB
	insercionesproyectotransmilenio	03/04/2019 10:43	Archivo SQL	1 KB

enseguida ya teniendo configurado el super usuario de sql server podemos crear la base de datos para ellos nos situamos en el super usuario creado y nos dirigimos a

la carpeta de bases de datos y creamos una base de datos con el nombre de bdtransmilenioproyecto



estando en ella damos clic derecho y le damos nueva consulta para copiar y poder ejecutar los diferentes scripts para la creación de la base de datos y ejecutamos



ya teniendo la base de datos configurada y cargada en el sql server vamos a configurar para crear el proyecto, para ello nos dirigimos a donde está la carpeta de NetBeans y en ella creamos una carpeta donde contara con los .jar necesarios para la conexión y el funcionamiento de la aplicación

Este equipo > Documentos >

Nombre	Fecha de modifica...	Tipo	Tamaño
DrExplain projects	11/12/2017 18:13	Carpeta de archivos	
eclipse	22/06/2018 21:09	Carpeta de archivos	
FeedbackHub	25/11/2017 14:34	Carpeta de archivos	
HiSuite	16/02/2018 17:20	Carpeta de archivos	
Lightshot	06/10/2018 13:17	Carpeta de archivos	
MEGAsync	03/03/2018 11:37	Carpeta de archivos	
MEGAsync Downloads	05/03/2018 21:26	Carpeta de archivos	
MobaXterm	13/09/2018 21:51	Carpeta de archivos	
My Cmaps	25/04/2018 17:13	Carpeta de archivos	
My ISO Files	26/03/2019 9:32	Carpeta de archivos	
NetBeansProjects	03/04/2019 14:17	Carpeta de archivos	
Plantillas personalizadas de Office	09/09/2017 17:46	Carpeta de archivos	
plsqldoc	26/02/2019 9:02	Carpeta de archivos	
Recursos	Fecha de creación: 09/09/2017 17:46 Carpeta vacía	Carpeta de archivos	
SQL Server Management Studio	28/03/2019 9:29	Carpeta de archivos	
Visual Studio 2010	26/03/2019 10:46	Carpeta de archivos	
Visual Studio 2015	18/09/2018 11:32	Carpeta de archivos	

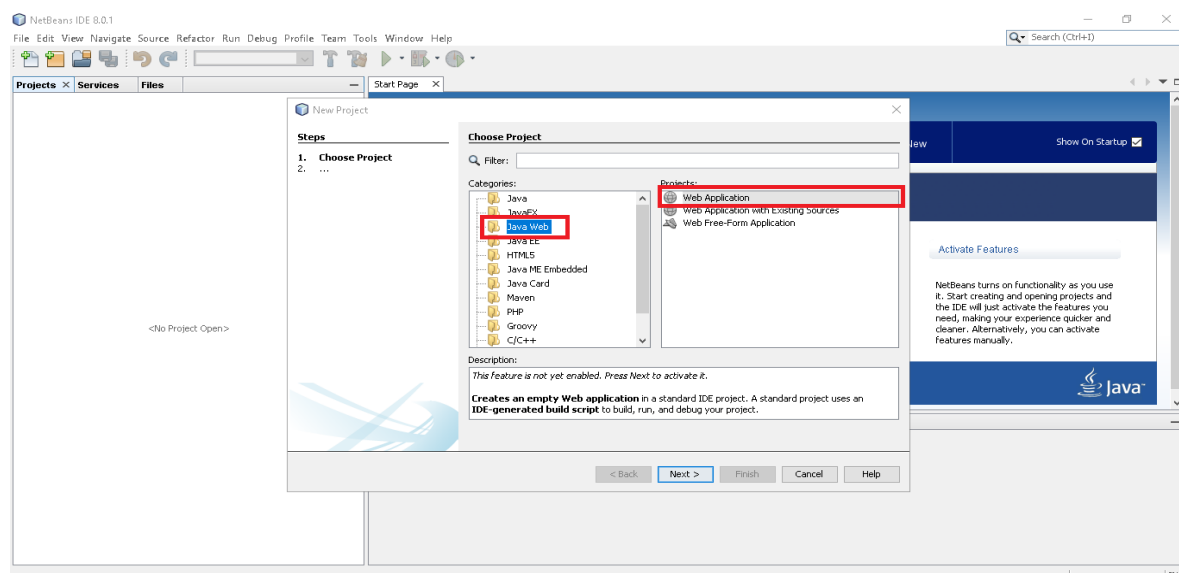
dentro de la carpeta recursos se encuentran los respectivos .jar

Portapapeles Organizar Nuevo Abrir Seleccionar

Este equipo > Documentos > Recursos > Recursos

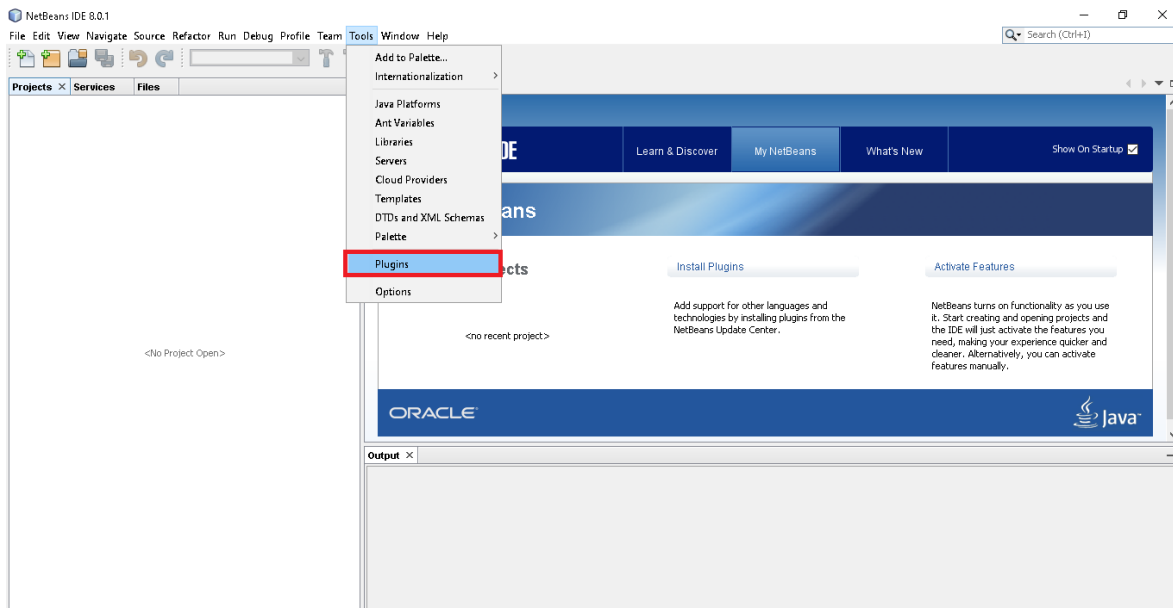
Nombre	Fecha de modifica...	Tipo	Tamaño
primefaces-5.0	25/03/2019 17:47	Executable Jar File	2.589 KB
primefaces-6.0	03/04/2019 14:52	Executable Jar File	3.914 KB
sqljdbc4	22/03/2019 22:49	Executable Jar File	571 KB
sqljdbc42	16/02/2018 18:02	Executable Jar File	878 KB

ya teniendo los diferentes componentes necesarios para poder seguir con el desarrollo de la aplicación nos dirigimos a NetBeans para crear el proyecto para ellos verificar si tenemos instalado java web en nuestro entorno de NetBeans

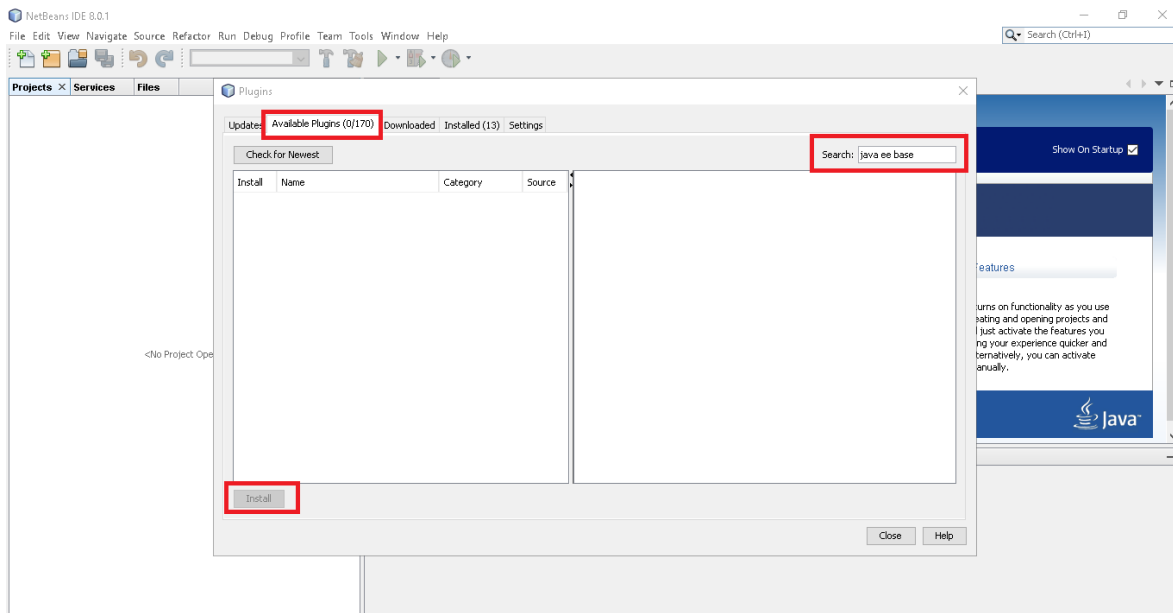


en caso de no tenerlo

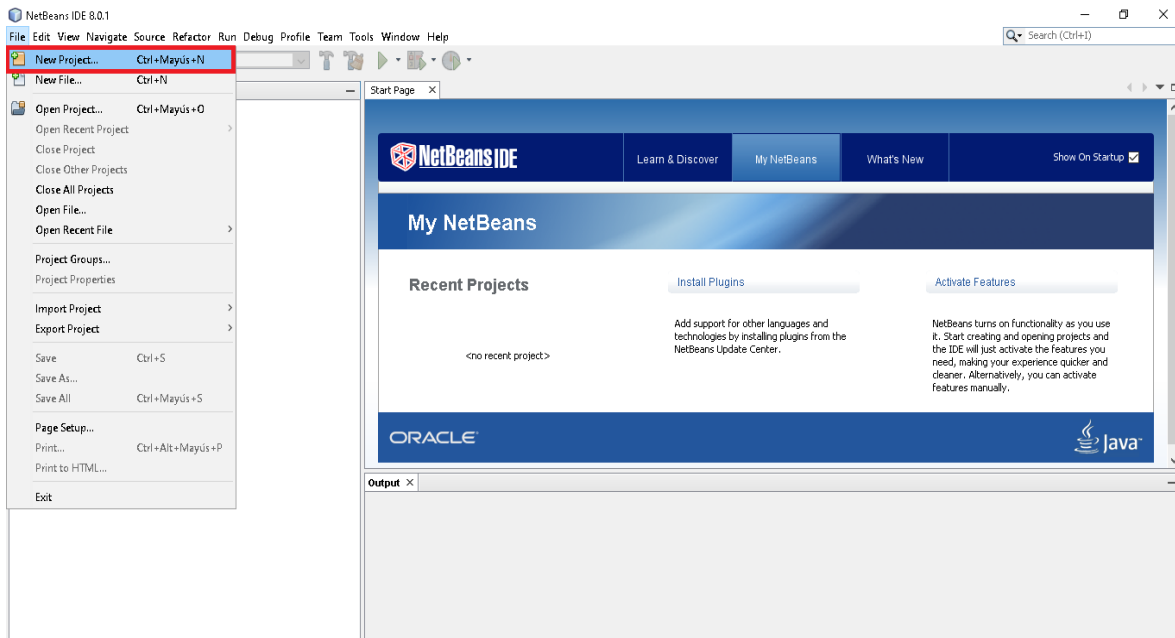
para ello vamos a la pestaña tools y plugin



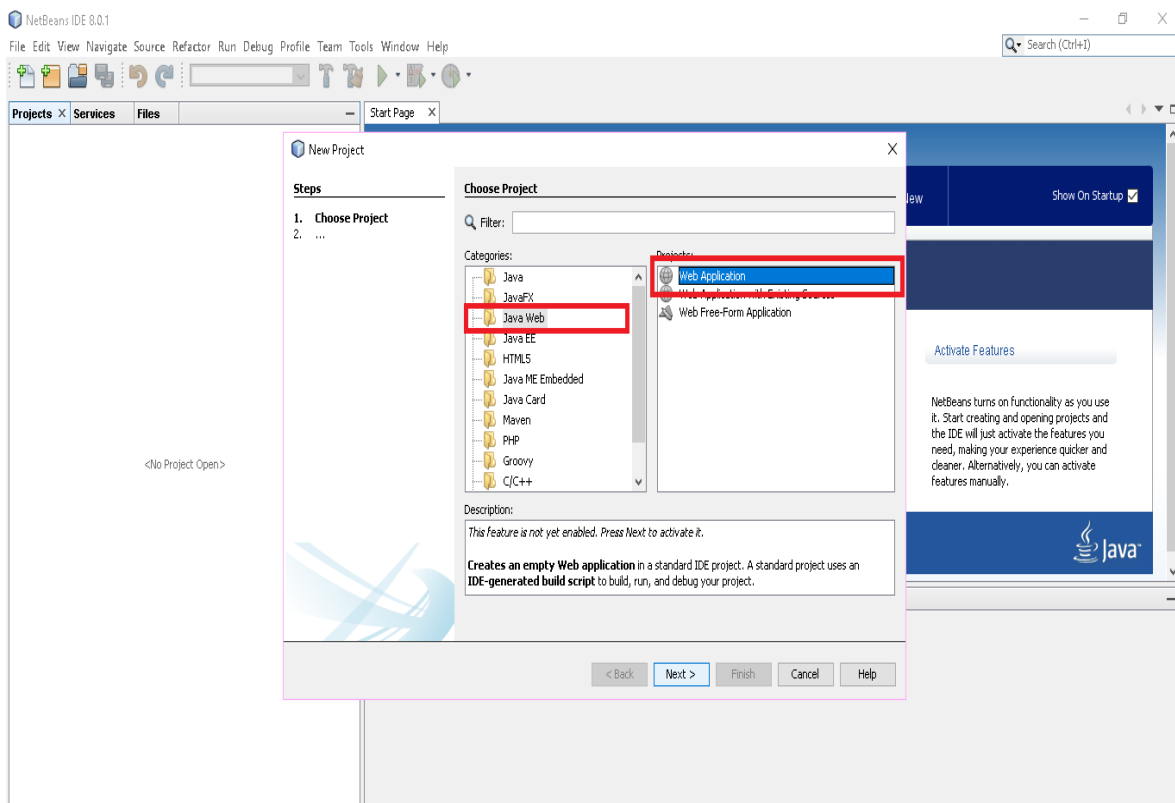
vamos a updates available y escribimos en la parte superior derecha java EE base y si aparece lo seleccionamos y damos install para nuestro caso la version NetBeans 8.1 al instalarlo nos instala los componentes web



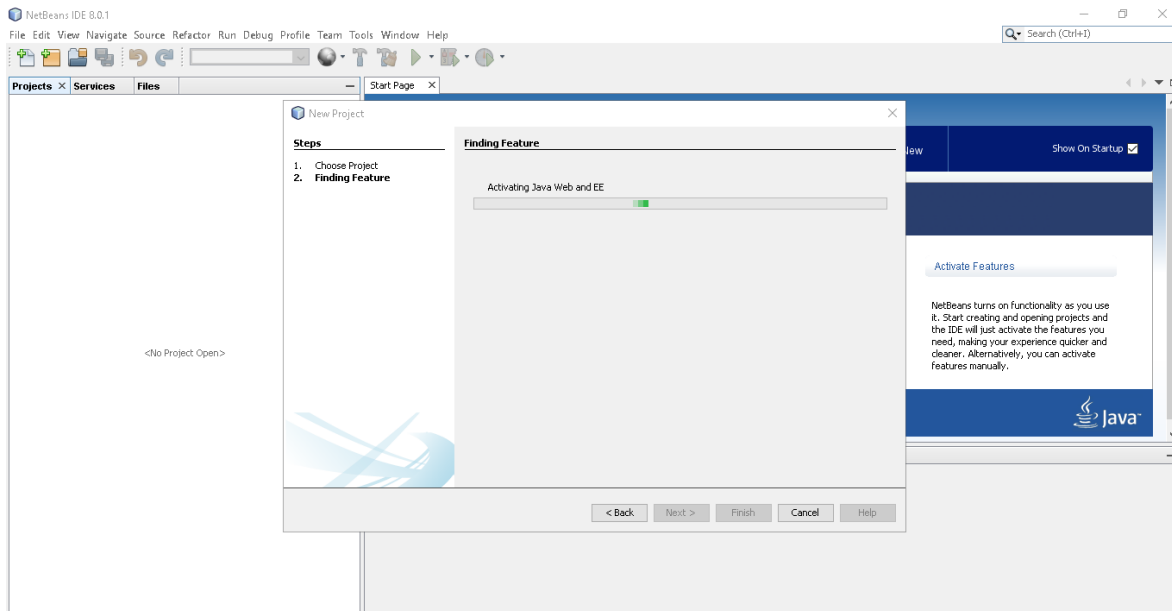
ahora vamos a crear nuestro proyecto vamos a file new project



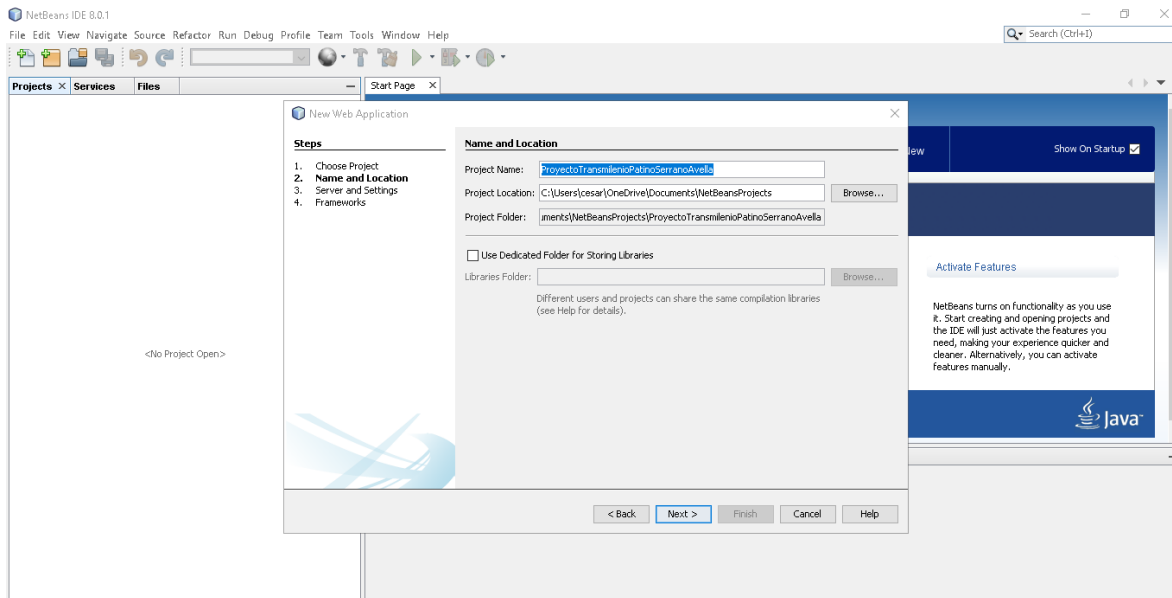
seleccionamos java web y web application



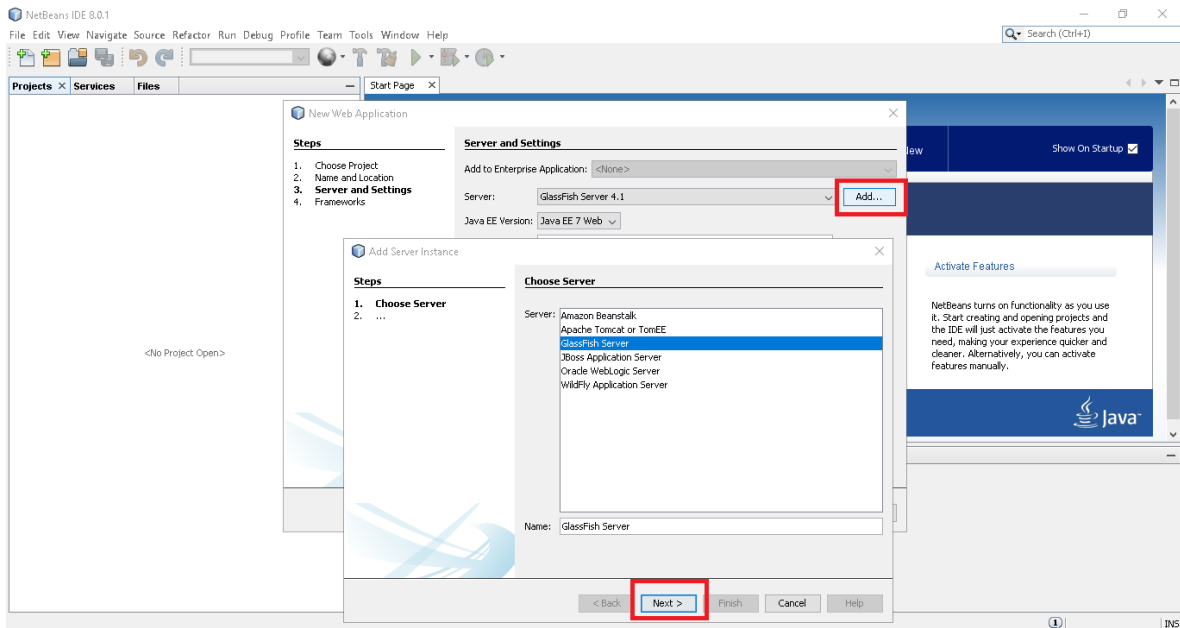
esperamos que le isntale complementos de web



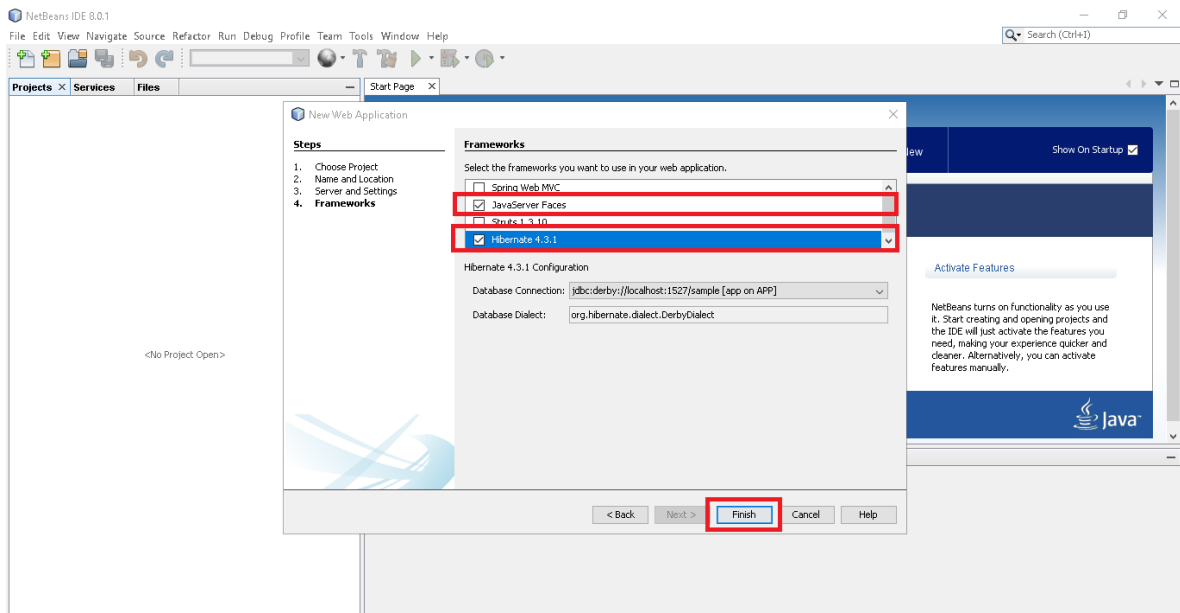
le colocamos el nombre a nuestro proyecto que va a ser proyectotransmileniopatinoavella



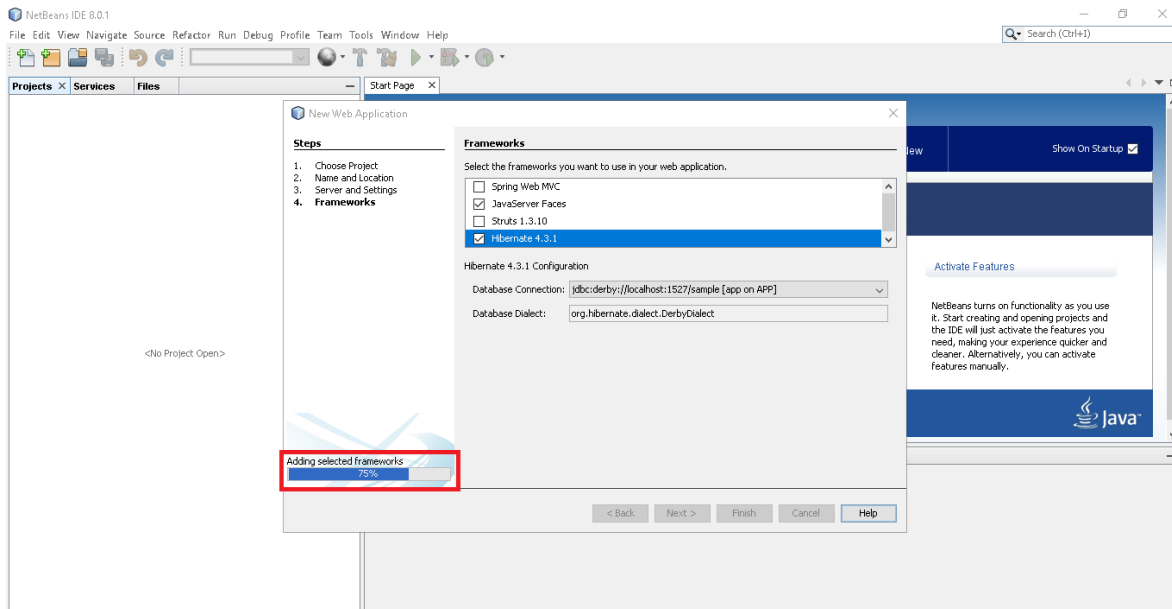
luego es importante ejecutar un servidor de aplicaciones para ello seleccionamos el que nos instala el NetBeans con esto podemos ejecutar las aplicaciones jsp y jsf (es importante porque o si no es imposible ver nuestras aplicaciones hechas en jsf o jsp) que es glashfish existen otros servidores de aplicaciones como tom cat y jboss si queremos instalarlo solo es seleccionarlo darle next, darle el nombre y listo



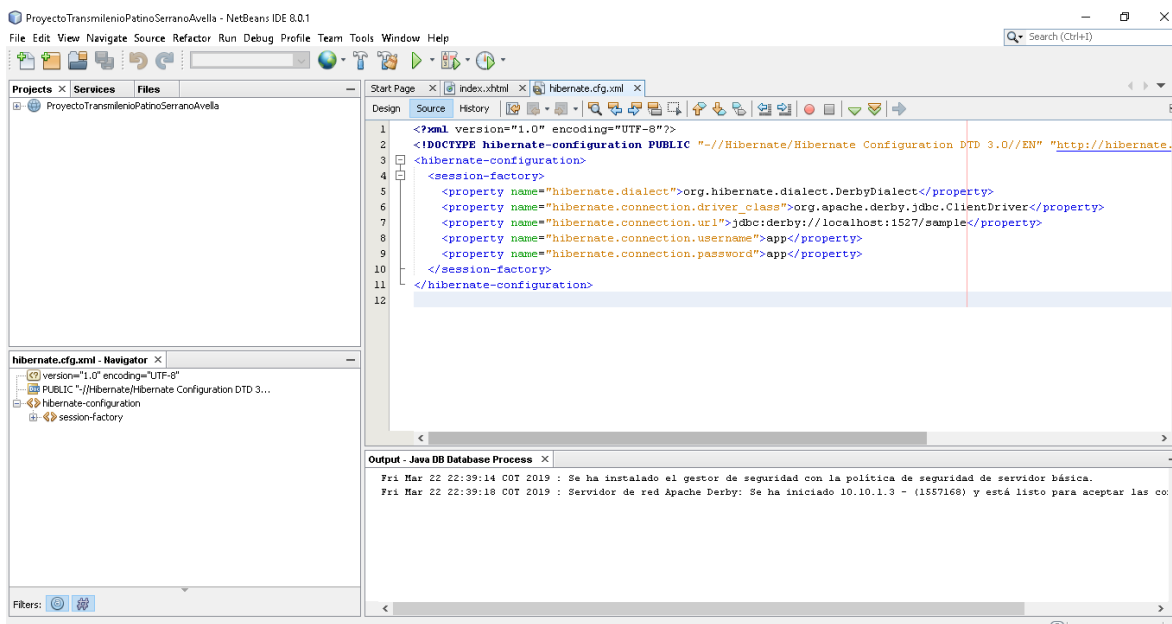
ahora vamos a seleccionar los framework para nuestro proyecto nosotros vamos a seleccionar el framework jsf(java server face) y el framework de persistencia hibernate y le damos finalizar



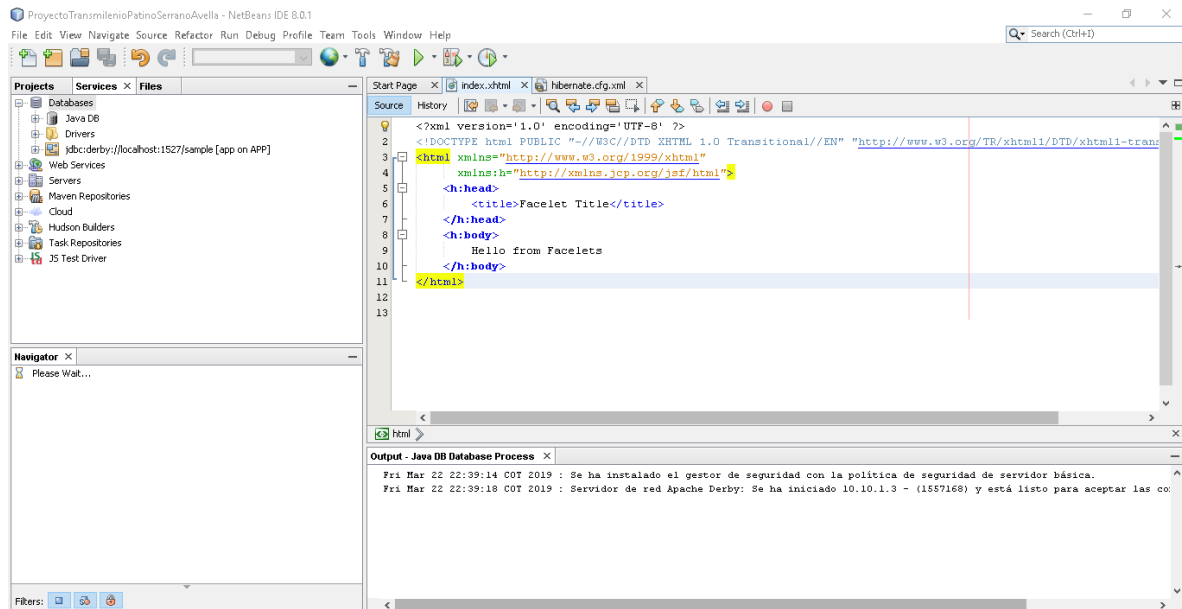
como vemos nos dice que esta instalando los framework seleccionados



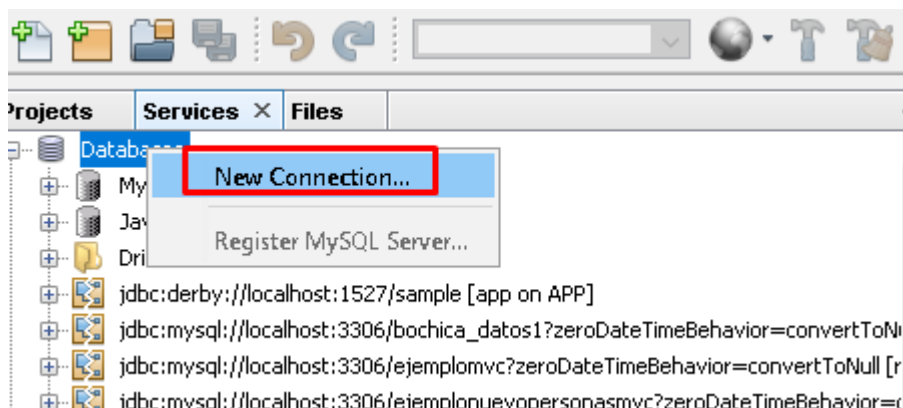
como vemos nos creo un archivo xml que es el de hibernate donde vamos hacer la conexión con la base de datos y la configuración de persistencia con los objetos que vamos a asociar a nuestras entidades de la base de datos



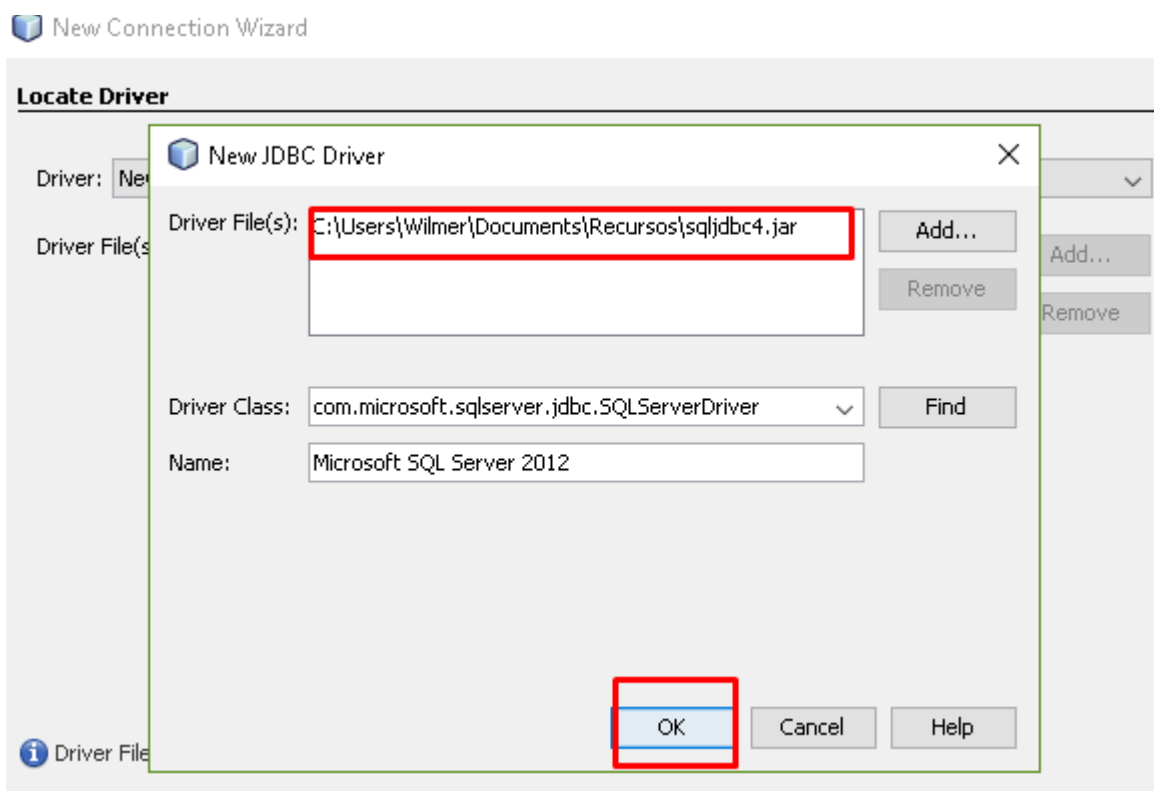
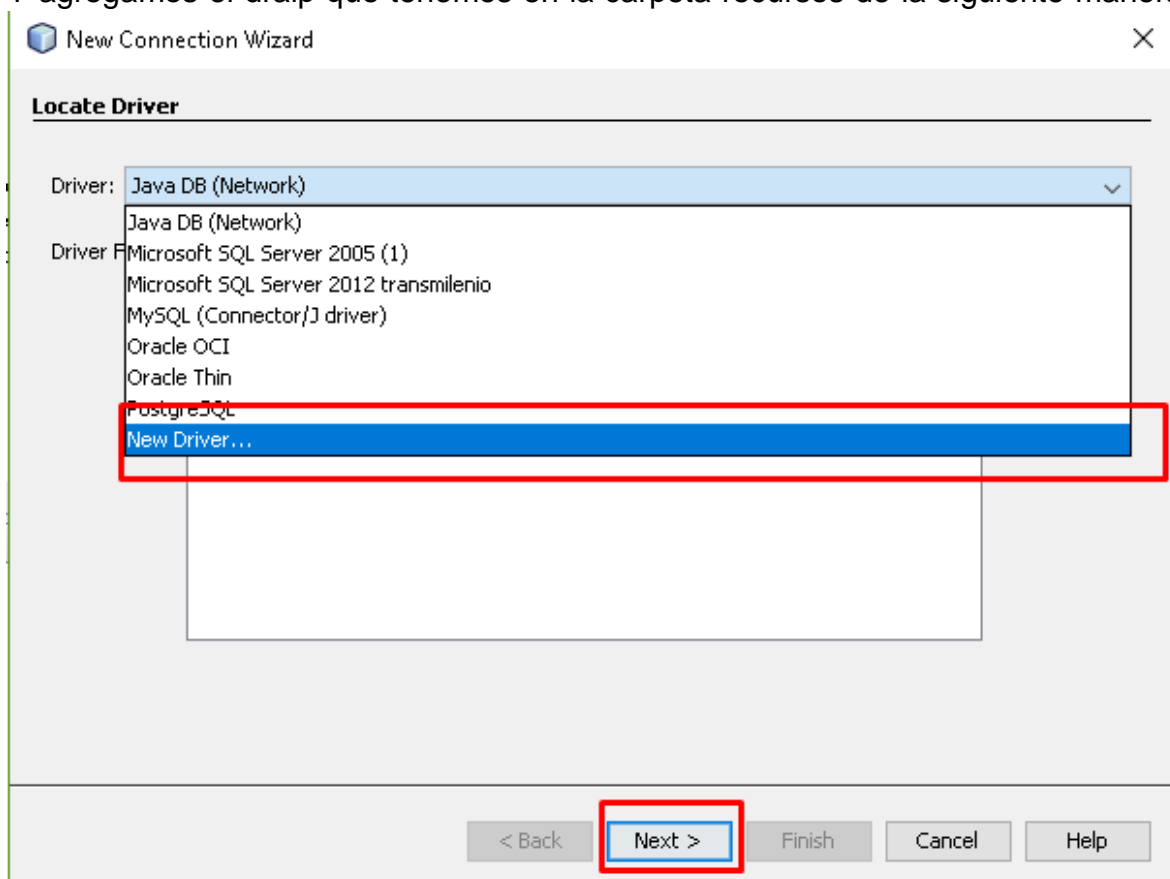
nos creó también el index.xhtml que es el que nos creó al seleccionar jsf



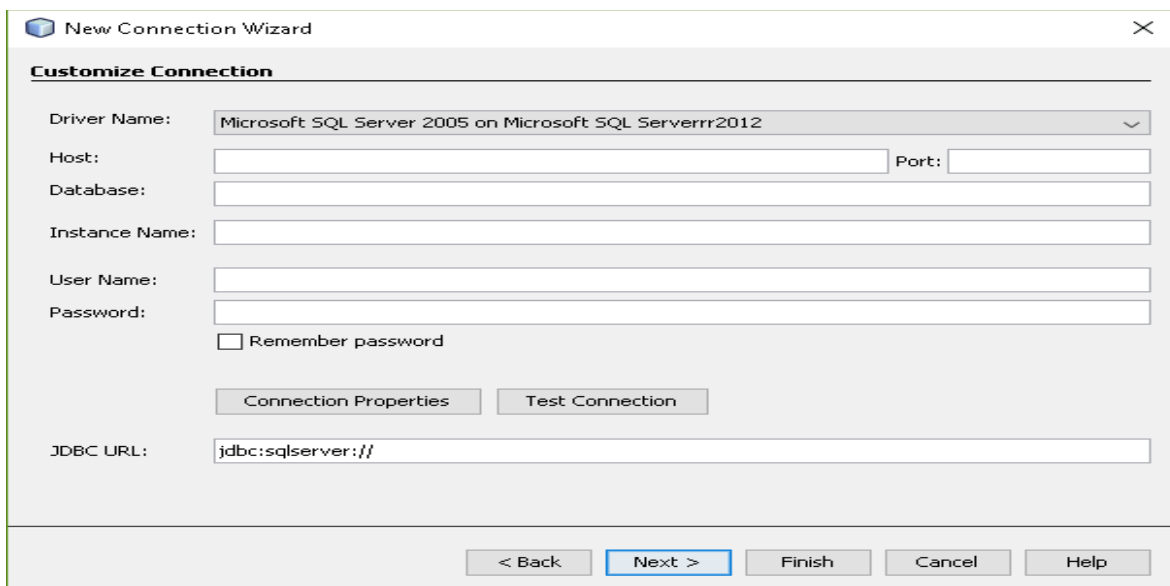
Ya creado el proyecto es necesario que crear una conexión para poder tener conexión con nuestra base de datos que está en sql server para ellos nos dirigimos a services y le damos nueva conexión



Y agregamos el draip que tenemos en la carpeta recursos de la siguiente manera



Enseguida nos saldrá la configuración para poder conectarnos a sql server 2012



New Connection Wizard

Customize Connection

Driver Name: Microsoft SQL Server 2005 on Microsoft SQL Serverrrr2012

Host: Port:

Database:

Instance Name:

User Name:

Password:

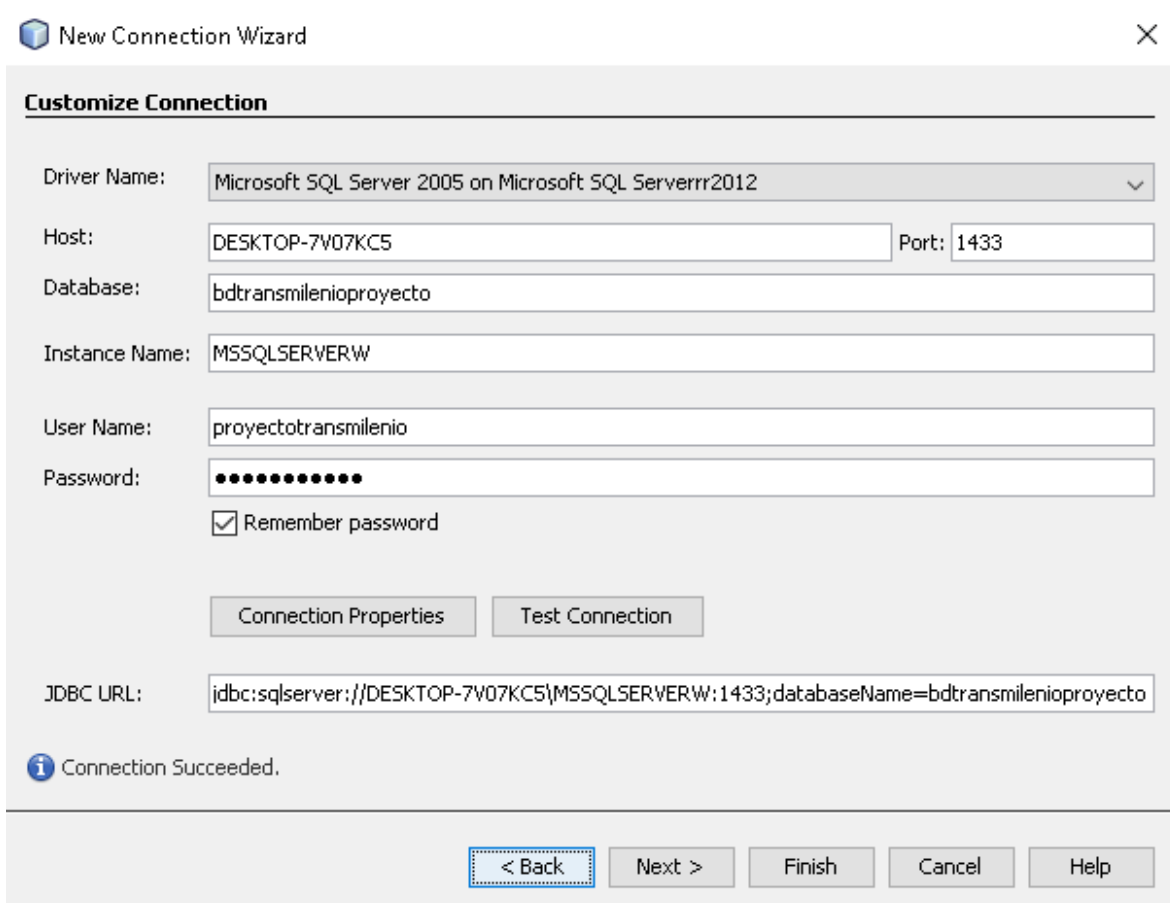
☐ Remember password

Connection Properties Test Connection

JDBC URL: jdbc:sqlserver://

< Back Next > Finish Cancel Help

Se completa el formulario correspondiente para seguir con la configuración y verificamos con el tes de conexión



New Connection Wizard

Customize Connection

Driver Name: Microsoft SQL Server 2005 on Microsoft SQL Serverrrr2012

Host: DESKTOP-7V07KC5 Port: 1433

Database: bdtransmilenioproyecto

Instance Name: MSSQLSERVERW

User Name: proyectotransmilenio

Password: ●●●●●●●●

☒ Remember password

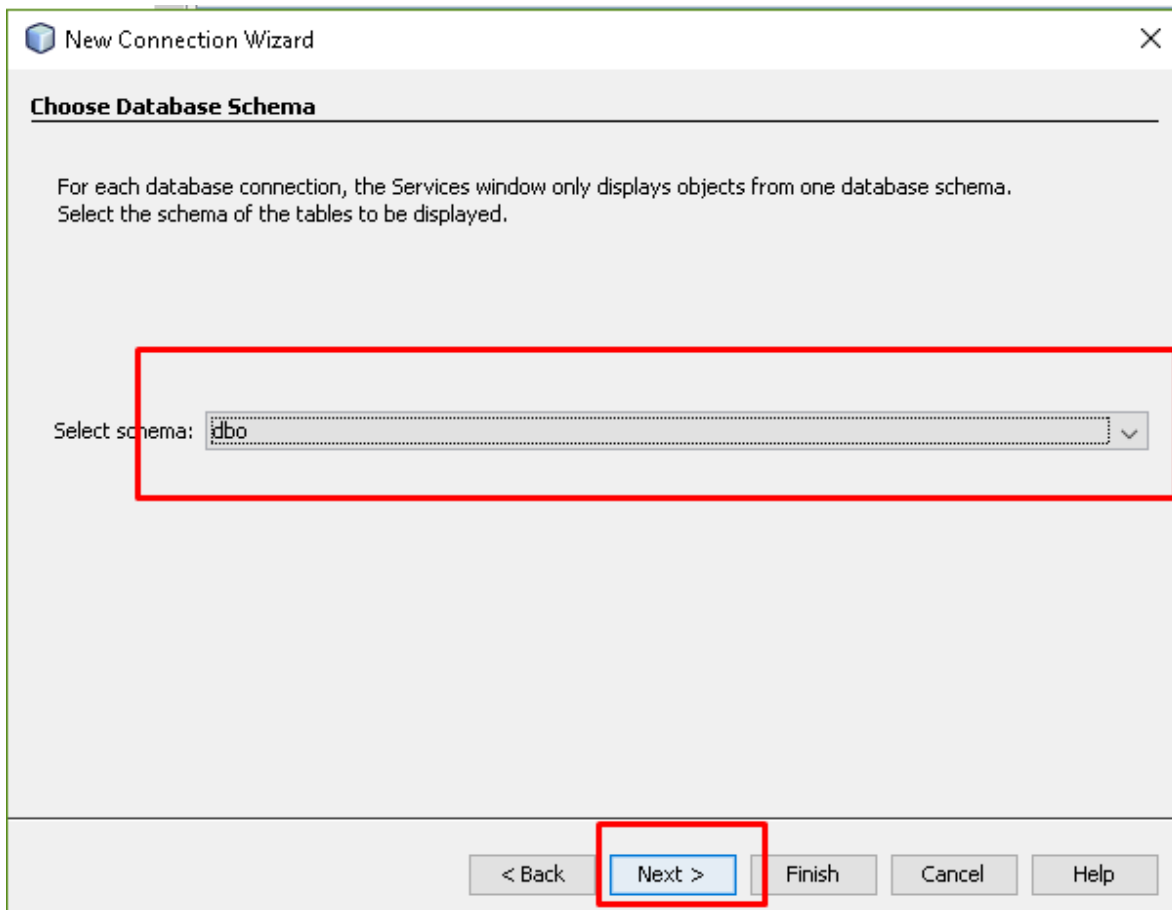
Connection Properties Test Connection

JDBC URL: jdbc:sqlserver://DESKTOP-7V07KC5\MSSQLSERVERW:1433;databaseName=bdtransmilenioproyecto

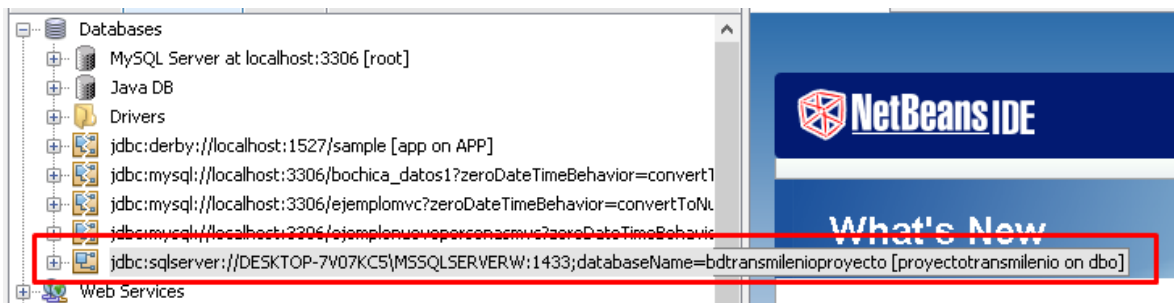
i Connection Succeeded.

< Back Next > Finish Cancel Help

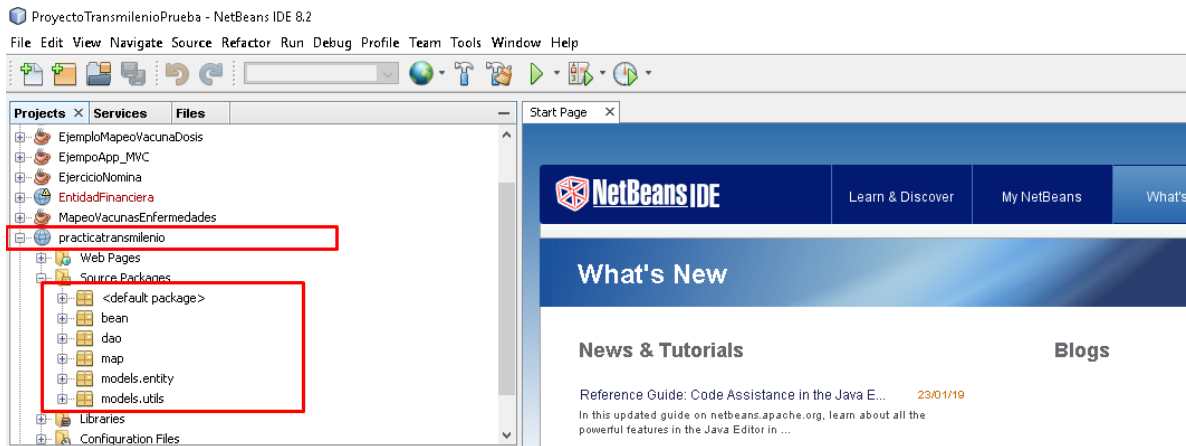
Enseguida seleccionamos la opción de dbo que contienen los mayores privilegios posibles en la base de datos; por lo tanto, tienen privilegios suficientes para crear y administrar la geodatabase.



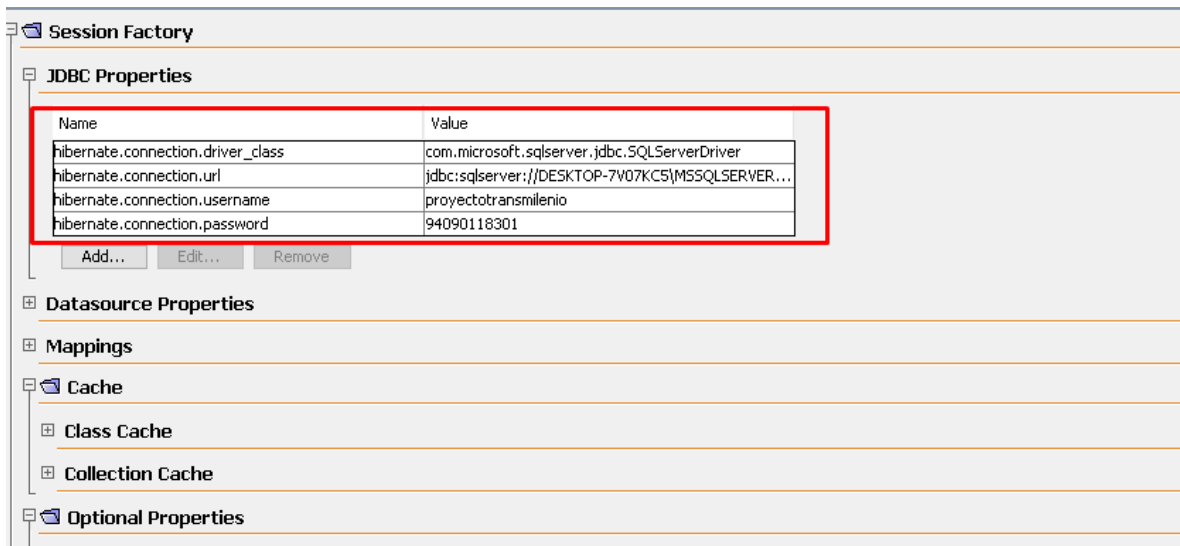
Y le damos finish y automáticamente nos crea la conexión con el motor de bases de datos



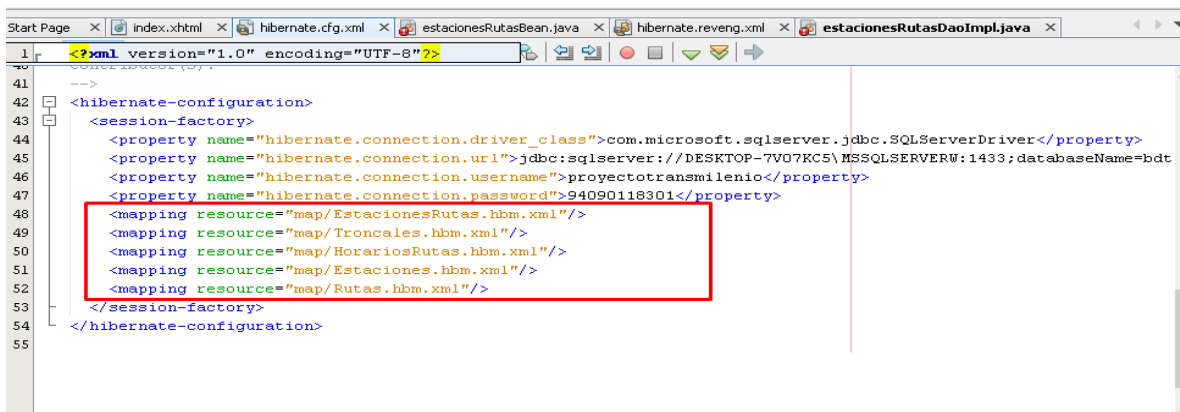
Ya teniendo la configuración de conexión nos dirigimos al proyecto y creamos los diferentes paquetes para el desarrollo de la aplicación



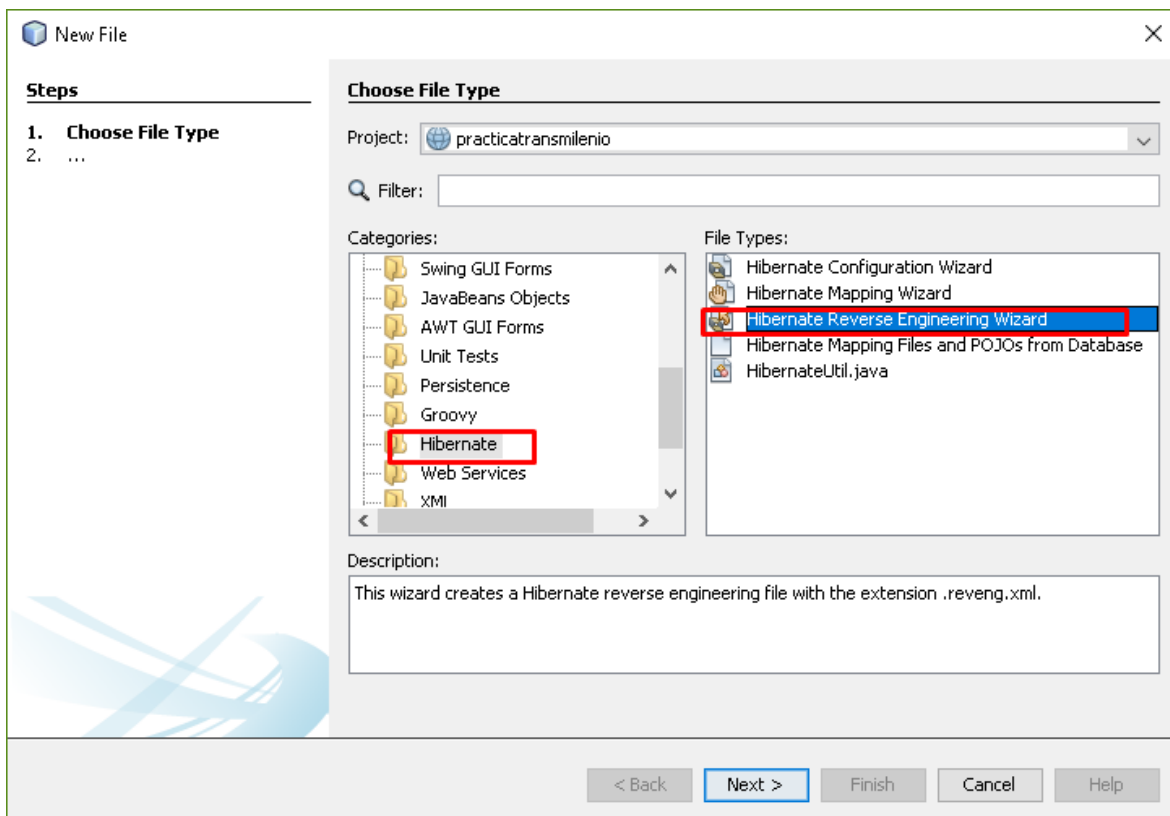
Después de haber creado los paquetes vamos a generar la configuración del hibernate para verificar si nos está haciendo la respectiva conexión con la base de datos y poder desarrollar el proceso de ingeniería inversa y los pojos para las diferentes tablas de nuestra base de datos



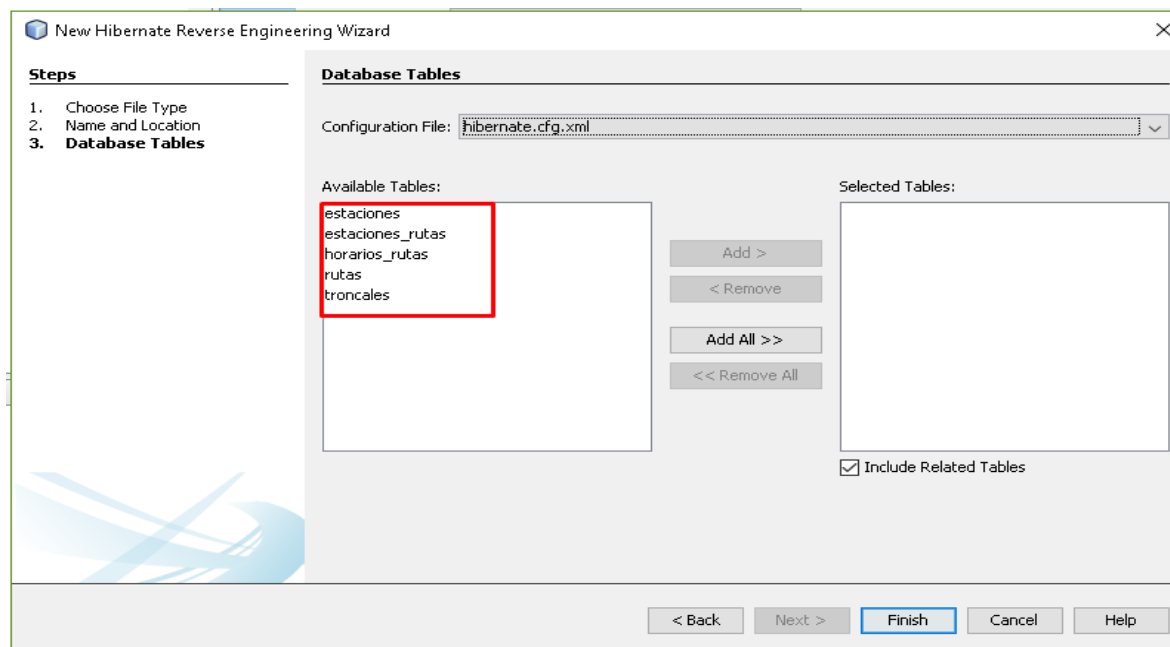
Dentro de esta configuración se puede observar las tablas que queremos mapear

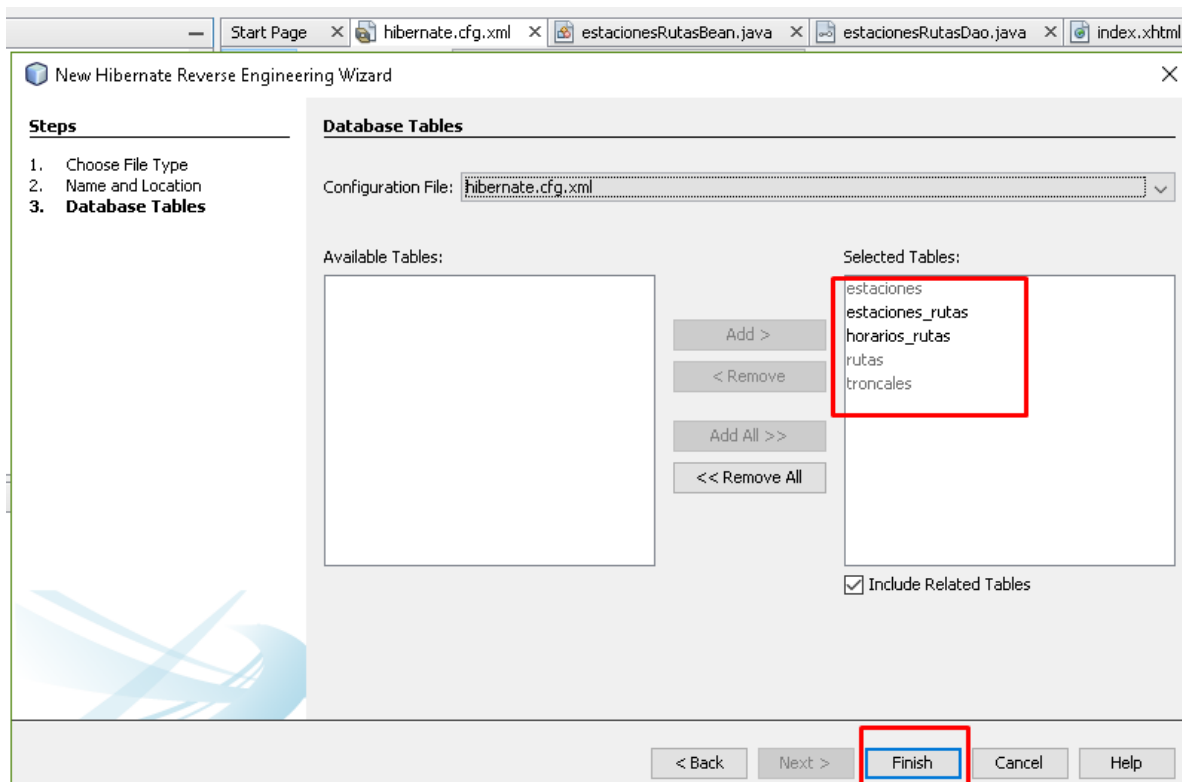


Enseguida vamos a realizar ingeniería inversa para poder mapear posteriormente nuestras tablas de la base de datos y que puedan ser reconocidas por la aplicación para ello nos situamos en el paquete de map

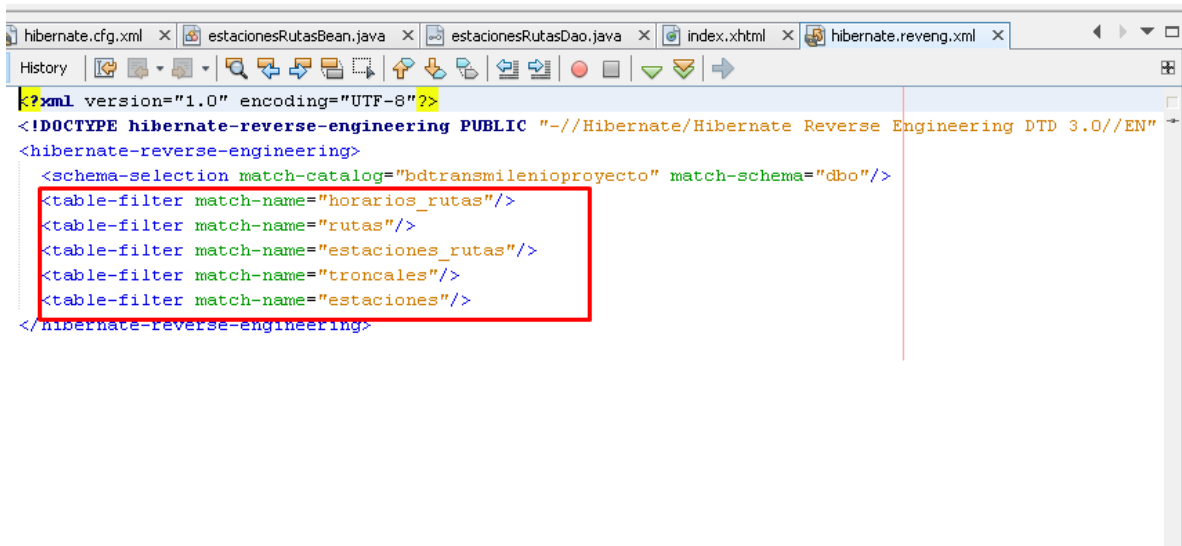


Después de ello nos cargara las diferentes tablas de nuestra base de datos y seleccionamos aquellas tablas a las cuales queremos mapear

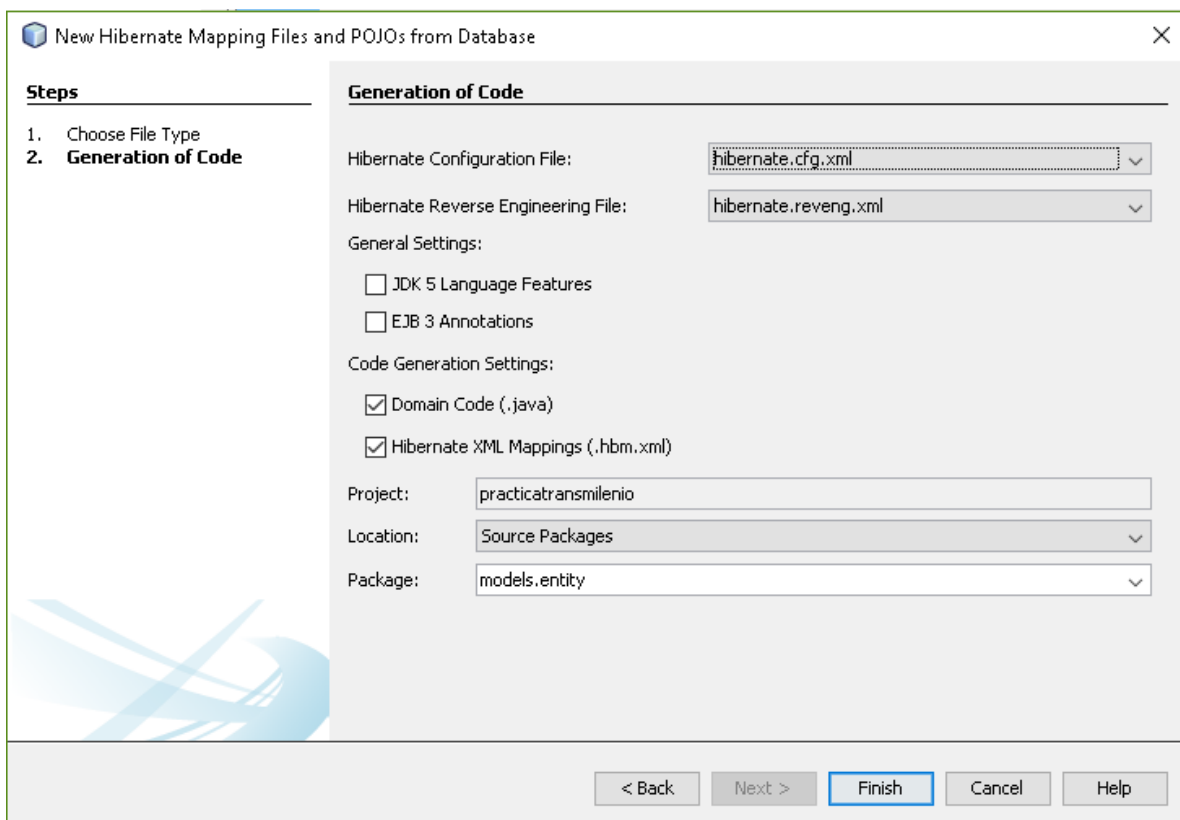




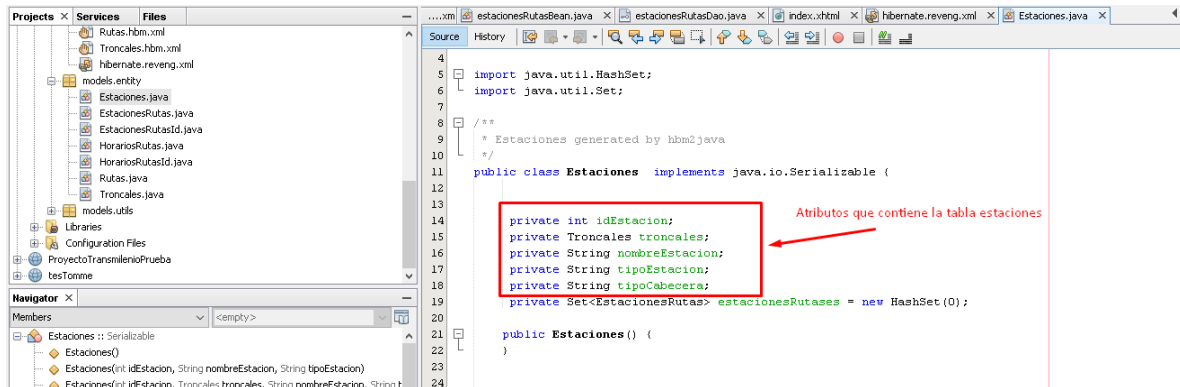
Dentro de esta clase nos arroja una configuración, donde toma el nombre de la base de datos y hace los respectivos filtros necesarios por cada una de las tablas seleccionadas



Ya teniendo esto continuamos con el proceso de generar los pojos de las diferentes tablas, donde nos genera unos archivos .hbm y .java



Dentro de las clases .java nos genera los diferentes atributos, constructores y los métodos set y get para poder acceder a los atributos de las tablas.



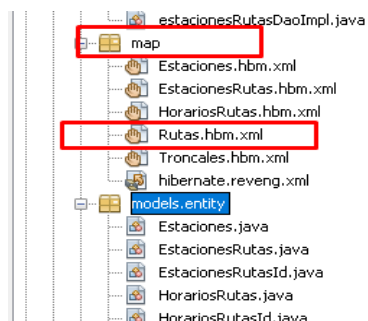

```
19 private Set<EstacionesRutas> estacionesRutas = new HashSet(0);
20
21 public Estaciones() {
22 }
23
24
25 public Estaciones(int idEstacion, String nombreEstacion, String tipoEstacion) {
26     this.idEstacion = idEstacion;
27     this.nombreEstacion = nombreEstacion;
28     this.tipoEstacion = tipoEstacion;
29 }
30
31 public Estaciones(int idEstacion, Troncales troncales, String nombreEstacion, String tipoEstacion,
32     this.idEstacion = idEstacion;
33     this.troncales = troncales;
34     this.nombreEstacion = nombreEstacion;
35     this.tipoEstacion = tipoEstacion;
36     this.tipoCabecera = tipoCabecera;
37     this.estacionesRutas = estacionesRutas;
38 }
```

constructor

```
34 this.tipoEstacion = tipoEstacion;
35 this.tipoCabecera = tipoCabecera;
36 this.estacionesRutas = estacionesRutas;
37 }
38
39 public int getIdEstacion() {
40     return this.idEstacion;
41 }
42
43 public void setIdEstacion(int idEstacion) {
44     this.idEstacion = idEstacion;
45 }
46
47 public Troncales getTroncales() {
48     return this.troncales;
49 }
50
51 public void setTroncales(Troncales troncales) {
52     this.troncales = troncales;
53 }
54
55 public String getNombreEstacion() {
```

metodos set y get

Arrastramos los .hbm al paquete map para tener una mejor distribución



Dentro de estos archivos .hbm contiene los atributos correspondientes de cada tabla y el tipo de dato y de igual forma las relaciones que se generan dentro de cada una de las tablas ya sea de uno a uno, uno a muchos, muchos a uno y de muchos a muchos

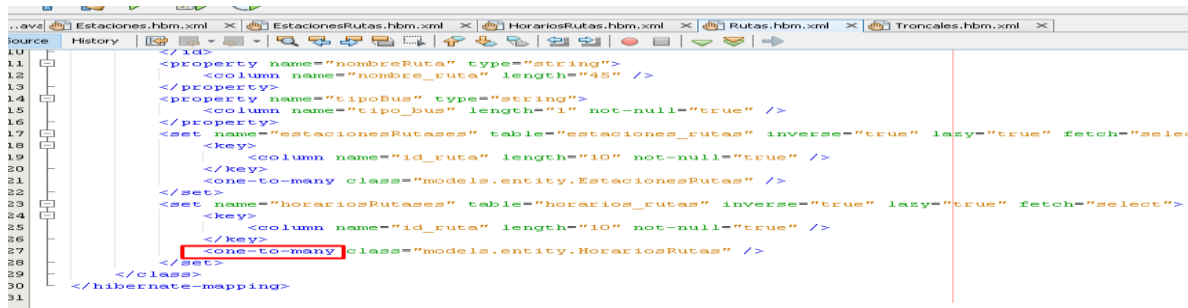
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN" "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
3 <!-- Generated 02-abr-2019 12:32:39 by Hibernate Tools 4.3.1 -->
4 <hibernate-mapping>
5   <class catalog="bdtransmilenioproyecto" name="models.entity.EstacionesRutas" optimistic-lock="version">
6     <composite-id class="models.entity.EstacionesRutasId" name="id">
7       <key-property name="idRuta" type="string">
8         <column length="10" name="id_ruta"/>
9       </key-property>
10      <key-property name="idEstacion" type="int">
11        <column name="id_estacion"/>
12      </key-property>
13    </composite-id>
14    <many-to-one class="models.entity.Estaciones" fetch="join" insert="false" name="estaciones" update="false">
15      <column name="id_estacion" not-null="true"/>
16    </many-to-one>
17    <many-to-one class="models.entity.Rutas" insert="false" fetch="join" name="rutas" update="false">
18      <column length="10" name="id_ruta" not-null="true"/>
19    </many-to-one>
20    <property name="ordenRuta" type="int">
21      <column name="orden_ruta" not-null="true"/>
22    </property>
23  </class>
24 </hibernate-mapping>
```

Relaciones uno a uno: consiste simplemente en que un objeto tenga una referencia a otro objeto de forma que al persistirse el primer objeto también se persista el segundo, en esta relación va a ser unidireccional es decir que la relación uno a uno va a ser en un único sentido.

Relaciones uno a muchos: consiste simplemente en que un objeto padre tenga una lista sin ordenar de otros objetos hijo de forma que al persistirse el objeto principal también se persista la lista de objetos hijo. Esta relación también suele llamarse maestro-detalle o padre-hijo. Dentro de estas relaciones encontramos la de troncales y rutas

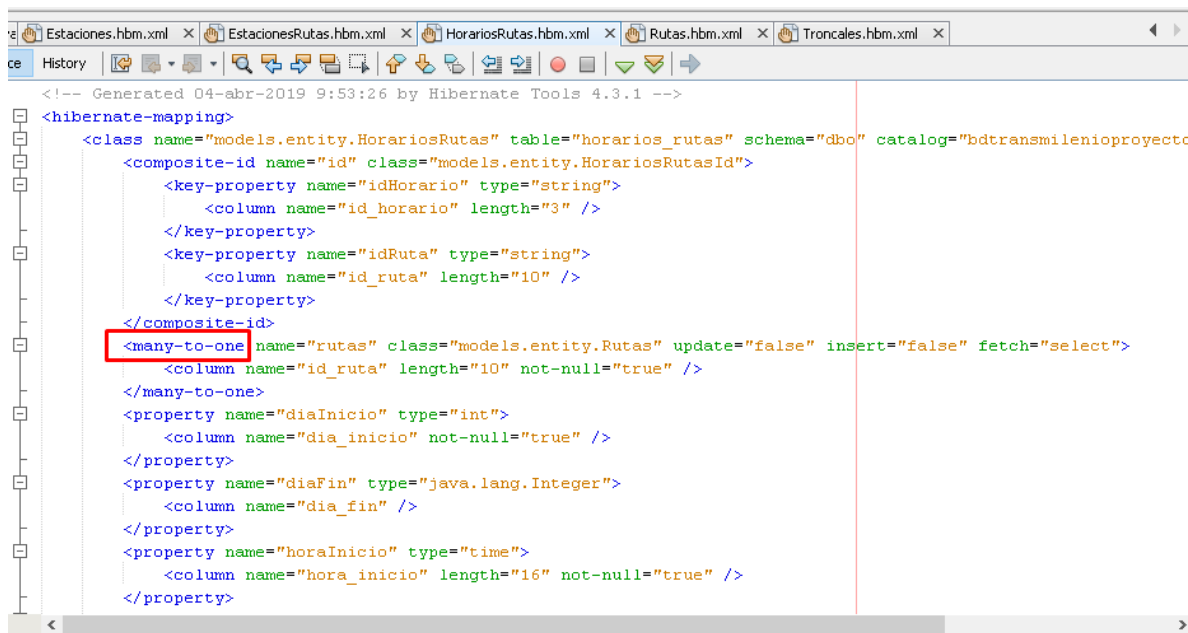
```
1 <class name="models.entity.Troncales" table="troncales" schema="dbo" catalog="bdtransmilenioproyecto" optimistic-lock="version">
2   <id name="idTroncal" type="string">
3     <column name="id_troncal" length="1" />
4     <generator class="assigned" />
5   </id>
6   <property name="nombreTroncal" type="string">
7     <column name="nombre_troncal" length="45" not-null="true" />
8   </property>
9   <property name="longitudTroncal" type="big_decimal">
10    <column name="longitud_troncal" precision="5" not-null="true" />
11  </property>
12  <property name="unidadesLongitud" type="string">
13    <column name="unidades_longitud" length="2" not-null="true" />
14  </property>
15  <set name="estaciones" table="estaciones" inverse="true" lazy="true" fetch="select">
16    <key>
17      <column name="id_troncal" length="1" />
18    </key>
19    <one-to-many class="models.entity.Estaciones" />
20  </set>
21 </class>
22 </hibernate-mapping>
```

Rutas



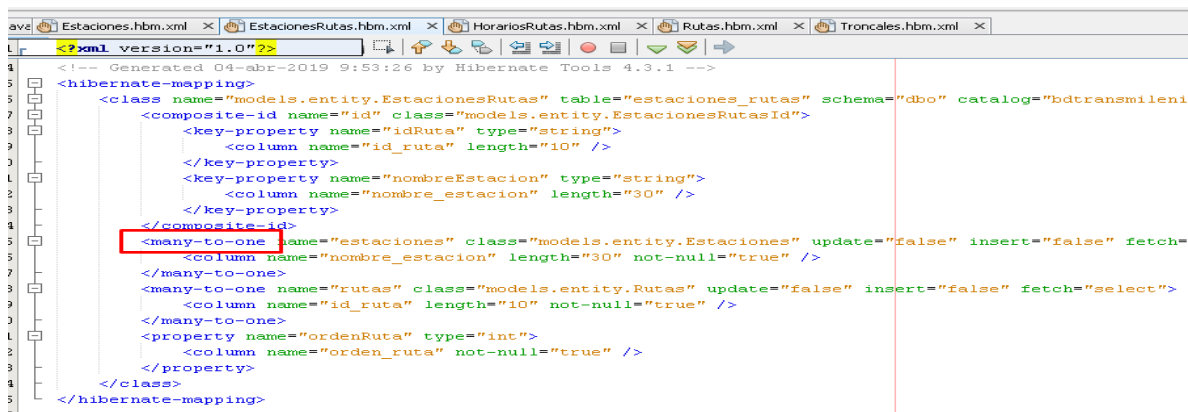
```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated 04-abr-2019 9:53:26 by Hibernate Tools 4.3.1 -->
<hibernate-mapping>
  <class name="models.entity.Rutas" table="rutas" schema="dbo" catalog="bdtransmilenioproyecto">
    <composite-id name="id" class="models.entity.RutasId">
      <key-property name="idHorario" type="string">
        <column name="id_horario" length="3" />
      </key-property>
      <key-property name="idRuta" type="string">
        <column name="id_ruta" length="10" />
      </key-property>
    </composite-id>
    <many-to-one name="estaciones" class="models.entity.Estaciones" update="false" insert="false" fetch="select">
      <column name="id_estacion" length="10" not-null="true" />
    </many-to-one>
    <many-to-one name="horarios" class="models.entity.Horarios" update="false" insert="false" fetch="select">
      <column name="id_horario" length="3" not-null="true" />
    </many-to-one>
    <property name="nombreRuta" type="string">
      <column name="nombre_ruta" length="45" />
    </property>
    <property name="tipoBus" type="string">
      <column name="tipo_bus" length="1" not-null="true" />
    </property>
  </class>
</hibernate-mapping>
```

Relaciones de muchos a uno Estas consiste simplemente en que un objeto padre tenga una lista Ordenar de otros objetos hijo de forma que al persistirse el objeto principal también se persista la lista de objetos hijo. Esta relación también suele llamarse maestro-detalle o padre-hijo. Dentro de estas relaciones encontramos las relaciones que esta entre horarios rutas



```
<!-- Generated 04-abr-2019 9:53:26 by Hibernate Tools 4.3.1 -->
<hibernate-mapping>
  <class name="models.entity.HorariosRutas" table="horarios_rutas" schema="dbo" catalog="bdtransmilenioproyecto">
    <composite-id name="id" class="models.entity.HorariosRutasId">
      <key-property name="idHorario" type="string">
        <column name="id_horario" length="3" />
      </key-property>
      <key-property name="idRuta" type="string">
        <column name="id_ruta" length="10" />
      </key-property>
    </composite-id>
    <many-to-one name="rutas" class="models.entity.Rutas" update="false" insert="false" fetch="select">
      <column name="id_ruta" length="10" not-null="true" />
    </many-to-one>
    <property name="diaInicio" type="int">
      <column name="dia_inicio" not-null="true" />
    </property>
    <property name="diaFin" type="java.lang.Integer">
      <column name="dia_fin" />
    </property>
    <property name="horaInicio" type="time">
      <column name="hora_inicio" length="16" not-null="true" />
    </property>
  </class>
</hibernate-mapping>
```

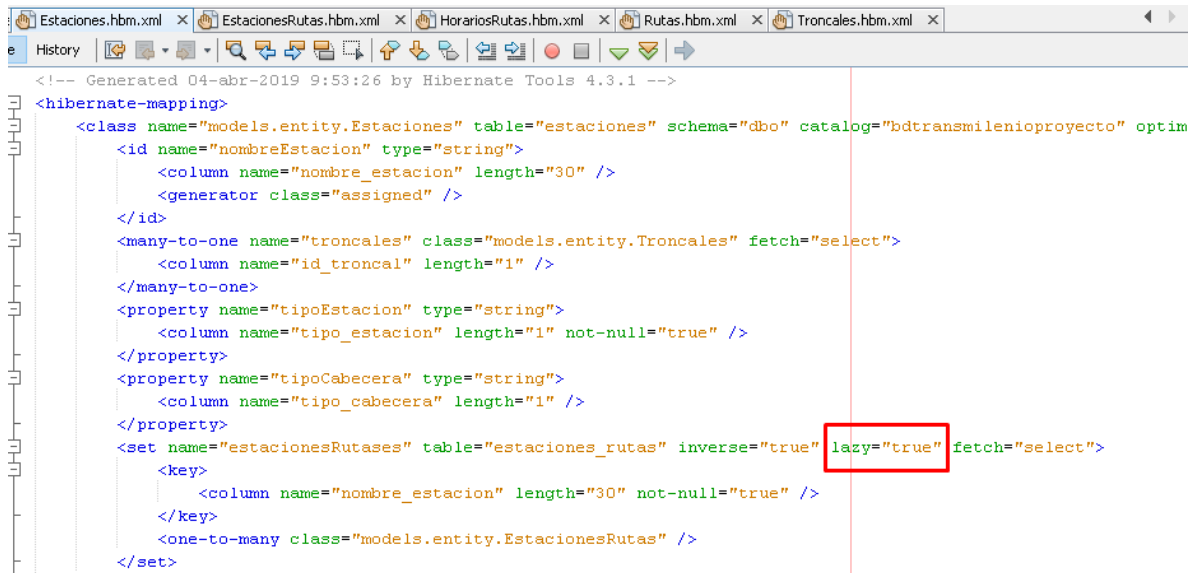
Estaciones rutas



```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Generated 04-abr-2019 9:53:26 by Hibernate Tools 4.3.1 -->
<hibernate-mapping>
  <class name="models.entity.EstacionesRutas" table="estaciones_rutas" schema="dbo" catalog="bdtransmilenioproyecto">
    <composite-id name="id" class="models.entity.EstacionesRutasId">
      <key-property name="idRuta" type="string">
        <column name="id_ruta" length="10" />
      </key-property>
      <key-property name="nombreEstacion" type="string">
        <column name="nombre_estacion" length="30" />
      </key-property>
    </composite-id>
    <many-to-one name="estaciones" class="models.entity.Estaciones" update="false" insert="false" fetch="select">
      <column name="nombre_estacion" length="30" not-null="true" />
    </many-to-one>
    <many-to-one name="rutas" class="models.entity.Rutas" update="false" insert="false" fetch="select">
      <column name="id_ruta" length="10" not-null="true" />
    </many-to-one>
    <property name="ordenRuta" type="int">
      <column name="orden_ruta" not-null="true" />
    </property>
  </class>
</hibernate-mapping>
```

Por último, encontramos **las relaciones muchos a muchos** las cuales consiste en que un objeto A tenga una lista de otros objetos B y también que el objeto B a su vez tenga la lista de objetos A. De forma que al persistirse cualquier objeto también se persista la lista de objetos que posee.

Dentro de los archivos .hbm encontramos los parámetros lazy los cuales Al especificar una relación en un mapeo de Hibernate especificar el “cuando”, por defecto lazy es igual a true esto significa que la colección no se recupera de la base de datos hasta que se hace alguna operación sobre ella.



```
<!-- Generated 04-abr-2019 9:53:26 by Hibernate Tools 4.3.1 -->
<hibernate-mapping>
  <class name="models.entity.Estaciones" table="estaciones" schema="dbo" catalog="bdtransmilenioproyecto" optim
    <id name="nombreEstacion" type="string">
      <column name="nombre_estacion" length="30" />
      <generator class="assigned" />
    </id>
    <many-to-one name="troncales" class="models.entity.Troncales" fetch="select">
      <column name="id_troncal" length="1" />
    </many-to-one>
    <property name="tipoEstacion" type="string">
      <column name="tipo_estacion" length="1" not-null="true" />
    </property>
    <property name="tipoCabecera" type="string">
      <column name="tipo_cabecera" length="1" />
    </property>
    <set name="estacionesRutas" table="estaciones_rutas" inverse="true" lazy="true" fetch="select">
      <key>
        <column name="nombre_estacion" length="30" not-null="true" />
      </key>
      <one-to-many class="models.entity.EstacionesRutas" />
    </set>
  </class>
</hibernate-mapping>
```

Por otro lado, encontramos los atributos fetch estos definen que SQLs se van a lanzar para recuperar la información



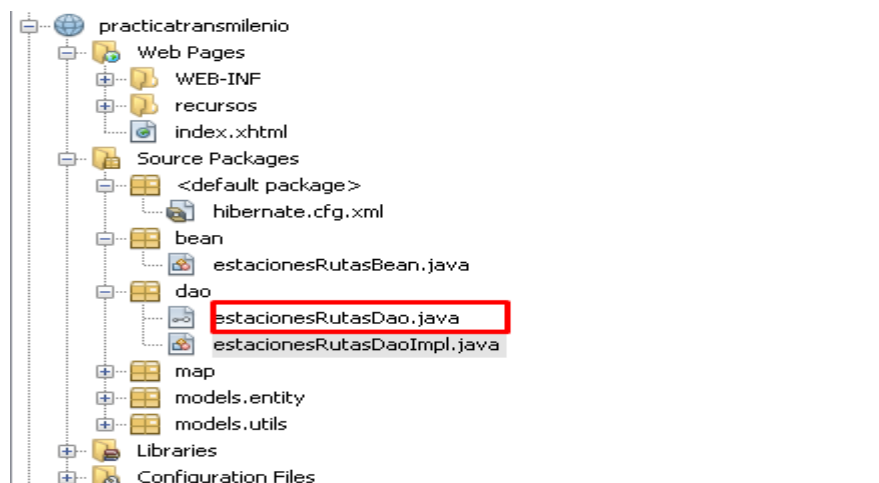
```
<!-- Generated 04-abr-2019 9:53:26 by Hibernate Tools 4.3.1 -->
<hibernate-mapping>
  <class name="models.entity.EstacionesRutas" table="estaciones_rutas" schema="dbo" catalog="bdtransmileniopro
    <composite-id name="id" class="models.entity.EstacionesRutasId">
      <key-property name="idRuta" type="string">
        <column name="id_ruta" length="10" />
      </key-property>
      <key-property name="nombreEstacion" type="string">
        <column name="nombre_estacion" length="30" />
      </key-property>
    </composite-id>
    <many-to-one name="estaciones" class="models.entity.Estaciones" update="false" insert="false" fetch="sel
      <column name="nombre_estacion" length="30" not-null="true" />
    </many-to-one>
    <many-to-one name="rutas" class="models.entity.Rutas" update="false" insert="false" fetch="select">
      <column name="id_ruta" length="10" not-null="true" />
    </many-to-one>
    <property name="ordenRuta" type="int">
      <column name="orden_ruta" not-null="true" />
    </property>
  </class>
</hibernate-mapping>
```

Enseguida dentro del paquete models.utils generamos el archivo hibernate.utils el cual es un archivo de comunicación entre hibernate y la base de datos y las respectivas excepciones.

```
13  * object.
14  *
15  * @author BELLARTES
16  */
17  public class HibernateUtil {
18
19      private static final SessionFactory sessionFactory;
20
21      static {
22          try {
23              // Create the SessionFactory from standard (hibernate.cfg.xml)
24              // config file.
25              sessionFactory = new AnnotationConfiguration().configure().buildSessionFactory();
26          } catch (Throwable ex) {
27              // Log the exception.
28              System.err.println("Initial SessionFactory creation failed." + ex);
29              throw new ExceptionInInitializerError(ex);
30          }
31      }
32
33      public static SessionFactory getSessionFactory() {
34          return sessionFactory;
35      }
36
37  }
```

MOSTRAR GESTION ESTACIONES RUTAS

Ahora dentro e paquete dao vamos a generar una nueva clase donde vamos a generar un método donde se pueda mostrar en la página índice.



Dentro de esta clase vamos a generar los respectivos métodos para llenar, mostrar, eliminar y modificar una lista de estaciones rutas el cual le generamos el siguiente código, cabe resaltar que en esta clase se hace el llamado a la entidad estacionesrutas la cual esta en el paquete models.entity

```
Start Page x hibernate.cfg.xml x index.xhtml x HibernateUtil.java x estacionesRutasDaoImpl.java x estacionesRutasDao.java x
Source History
8 import java.util.List;
9 import models.entity.Estaciones;
10 import models.entity.EstacionesRutas;
11 import models.entity.Rutas;
12
13 /**
14  *
15  * @author BELLARTES
16  */
17 public interface estacionesRutasDao {
18     public List<EstacionesRutas> mostrarEstacionesRutas();
19     public void nuevoEstacionesRutas(EstacionesRutas estacionesrutas);
20     public void modificarEstacionRutas(EstacionesRutas estacionesrutas);
21     public int eliminarEstacionRutas(EstacionesRutas estacionesrutas);
22     public List<Estaciones> ListarEstaciones();
23     public List<Rutas> ListarRutas();
24     public List buscarEstacionesRutas(EstacionesRutas estacionesrutas);
25 }
26
```

De igual forma para estaciones

```

*
* @author cesar
*/
public interface estacionesDao {
    public List<Estaciones> mostrarEstaciones();
    public void nuevoEstaciones(Estaciones estaciones);
    public void modificarEstaciones(Estaciones estaciones);
    public int eliminarEstaciones(Estaciones estaciones);
    public List buscarEstaciones(Estaciones estaciones);
    public List<Troncales> ListarTroncales();
}

```

También para rutas

```

*
public interface rutasDao {
    public List<Rutas> mostrarRutas();
    public void nuevoRutas(Rutas rutas);
    public void modificarRutas(Rutas rutas);
    public int eliminarRutas(Rutas rutas);
    public List buscarRutas(Rutas rutas);
}

```

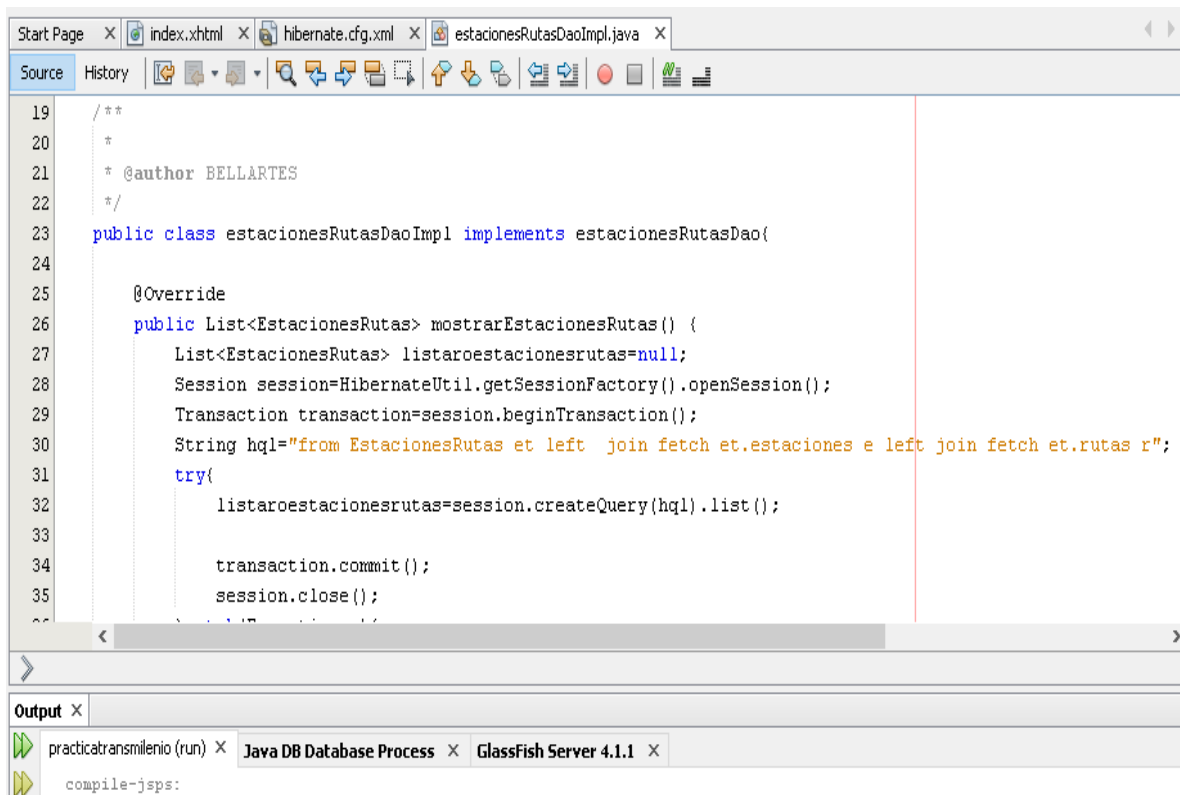
y por ultimo para las troncales

```

public interface troncalesDao {
    public List<Troncales> mostrarTroncales();
    public void nuevoTroncales(Troncales troncales);
    public void modificarTroncales(Troncales troncales);
    public int eliminarTroncales(Troncales troncales);
    public List buscarTroncales(Troncales troncales);
}

```

De igual forma creamos las clase normalita de java esto para que implemente los métodos creados en la clases anterior mente descritos

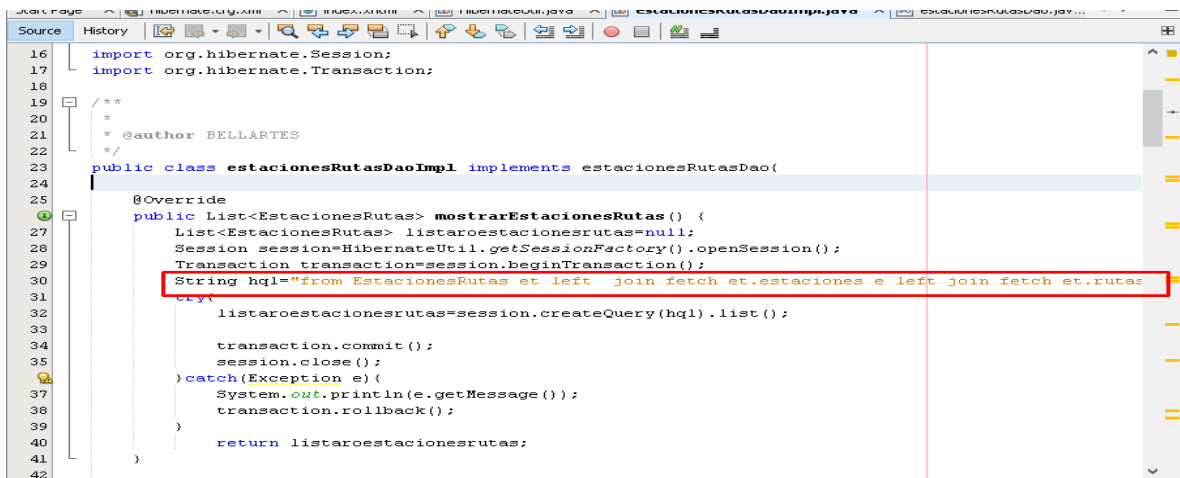


```
19  /**
20   *
21   * @author BELLARTES
22   */
23  public class estacionesRutasDaoImpl implements estacionesRutasDao{
24
25      @Override
26      public List<EstacionesRutas> mostrarEstacionesRutas() {
27          List<EstacionesRutas> listaroestacionesrutas=null;
28          Session session=HibernateUtil.getSessionFactory().openSession();
29          Transaction transaction=session.beginTransaction();
30          String hql="from EstacionesRutas et left join fetch et.estaciones e left join fetch et.rutas r";
31          try{
32              listaroestacionesrutas=session.createQuery(hql).list();
33
34              transaction.commit();
35              session.close();
36          }
37      }
38  }
```

practicatransmilenio (run) x Java DB Database Process x GlassFish Server 4.1.1 x

compile-jsp:s

Para generar el método de mostrar estacionesrutas es necesario generar el siguiente código, dentro de este método se crea una variable hql donde se hace una consulta hql ya que por utilizar el lenguaje hibernate la consulta es diferente a las consultas normalitas de sql



```
16  import org.hibernate.Session;
17  import org.hibernate.Transaction;
18
19  /**
20   *
21   * @author BELLARTES
22   */
23  public class estacionesRutasDaoImpl implements estacionesRutasDao{
24
25      @Override
26      public List<EstacionesRutas> mostrarEstacionesRutas() {
27          List<EstacionesRutas> listaroestacionesrutas=null;
28          Session session=HibernateUtil.getSessionFactory().openSession();
29          Transaction transaction=session.beginTransaction();
30          String hql="from EstacionesRutas et left join fetch et.estaciones e left join fetch et.rutas r";
31          try{
32              listaroestacionesrutas=session.createQuery(hql).list();
33
34              transaction.commit();
35              session.close();
36          } catch (Exception e) {
37              System.out.println(e.getMessage());
38              transaction.rollback();
39          }
40          return listaroestacionesrutas;
41      }
42  }
```

De igual forma dentro del método insertar podemos ver que se crea una session con nuestra base de datos para ello utilizamos el hibernate entro del método insertar podemos ver que se crea una session con nuestra base de datos para ello utilizamos el hibernate para nuestro caso estamos utilizando un pojo que tiene las llaves primarias de estaciones y rutas por ello creamos instancias de los pojoes rutas

y estaciones donde se verifico que los datos que estamos ingresando existen en los registros para poder modificarlos luego mediante la session hacemos la respectiva modificaciones

```
public class estacionesRutasDaoImpl implements estacionesRutasDao{

    @Override
    public List<EstacionesRutas> mostrarEstacionesRutas() {
        List<EstacionesRutas> listaroestacionesrutas=null;
        Session session=HibernateUtil.getSessionFactory().openSession();
        Transaction transaction=session.beginTransaction();
        String hql="from EstacionesRutas et left join fetch et.estaciones e left join fetch et.rutas r";
        try{
            listaroestacionesrutas=session.createQuery(hql).list();

            transaction.commit();
            session.close();
        }catch(Exception e){
            System.out.println(e.getMessage());
            transaction.rollback();
        }

        return listaroestacionesrutas;
    }
}
```

Enseguida vamos a realizar el método de nueva estación ruta la cual podrá registrar una nueva inserción a la base de datos, para ello debemos traer los atributos necesarios para llevar esta inserción, este lo atrapamos con los métodos get que están dentro de los pojos estaciones rutas.

```
@Override
public void nuevoEstacionesRutas(EstacionesRutas estacionesrutas) {
    Session session=null;
    try{
        session=HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        Rutas r=(Rutas) session.get(Rutas.class, estacionesrutas.getRutas().getIdRuta());
        Estaciones e=(Estaciones) session.get(Estaciones.class, estacionesrutas.getEstaciones().getIdEstacion());
        EstacionesRutasId eid=new EstacionesRutasId(r.getIdRuta(), e.getIdEstacion());
        EstacionesRutas er=new EstacionesRutas(eid, e, r, estacionesrutas.getOrdenRuta());
        session.save(er);
        session.getTransaction().commit();
    }catch(Exception e){
        System.out.println(e.getMessage());
        session.getTransaction().rollback();
    }finally{
        if(session!=null){
            session.close();
        }
    }
}
```

Después creamos el método que se pueda eliminar un registro de las estaciones rutas y a la vez se actualice dentro de la base de datos

```
@Override
public int eliminarEstacionRutas(EstacionesRutas estacionesrutas) {
    Session session=null;
    int result=0;
    try{
        session=HibernateUtil.getSessionFactory().openSession();
        Query query = session.createQuery("delete from EstacionesRutas et where et.id.idEstacion=:idestacion and et");

        session.beginTransaction();
        //session.delete(operadores);
    }
}
```


Enseguida creamos el método de listar estaciones en donde se crea una sesión que atrape los atributos de los pojos correspondientes de la clase estaciones, y dentro de este método también es necesario implementar una consulta hql para que nos muestre todas las estaciones

```
@Override
public List<Estaciones> ListarEstaciones() {
    List<Estaciones> listarestaciones=null;
    Session session=HibernateUtil.getSessionFactory().openSession();
    Transaction transaction=session.beginTransaction();
    String hql="from Estaciones e";
    try{
        listarestaciones=session.createQuery(hql).list();

        transaction.commit();
        session.close();
    }catch(Exception e){
        System.out.println(e.getMessage());
        transaction.rollback();
    }

    return listarestaciones;
}
```

Enseguida dentro de esta misma clase generamos una lista para que nos atrape todas las rutas que están insertadas en la base de datos y nos la muestre, para ello es necesario implementar una consulta hql para que nos traiga todos los datos de esta tabla.

```
@Override
public List<Rutas> ListarRutas() {
    List<Rutas> listarrutas=null;
    Session session=HibernateUtil.getSessionFactory().openSession();
    Transaction transaction=session.beginTransaction();
    String hql="from Rutas";
    try{
        listarrutas=session.createQuery(hql).list();

        transaction.commit();
        session.close();
    }catch(Exception e){
        System.out.println(e.getMessage());
        transaction.rollback();
    }

    return listarrutas;
}
```

Por ultimo creamos el método de buscar estaciones rutas para ello es necesario crear una sesión en donde nos traiga los atributos correspondientes a la tabla estacionesrutas esto se hace a través de los métodos get, de igual forma se debe

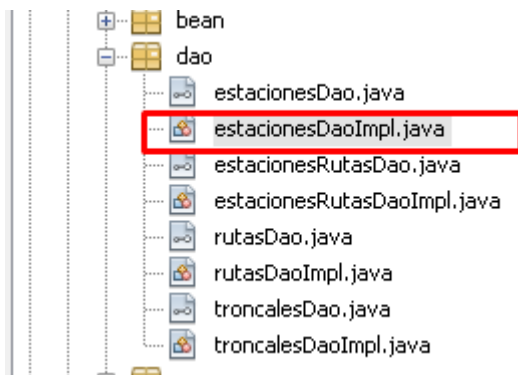
generar una consulta para que nos traiga los datos tanto de las estaciones como de las rutas

```
@Override
public List buscarEstacionesRutas(EstacionesRutas estacionesrutas) {
    Session session=null;
    List result=null;
    try{
        session=HibernateUtil.getSessionFactory().openSession();
        String hql="select er.id from EstacionesRutas er where er.id.nombreEstacion=nombreEstacion and er";
        session.beginTransaction();
        //session.delete(operadores);
        result=(List) session.createQuery(hql)
            .setParameter("idruta",estacionesrutas.getRutas().getIdRuta())
            .setParameter("nombreEstacion", estacionesrutas.getEstaciones().getNombreEstacion())
            .list();

        session.getTransaction().commit();
    }catch(Exception e){
        System.out.println(e.getMessage());
    }
```

MOSTRAR GESTION ESTACIONES

Dentro del paquete dao creamos una nueva clase llamada estaciones.dao para generar los métodos de listar, mostrar, buscar, modificar y eliminar



Enseguida vamos a desarrollar el método de mostrar estaciones en donde es necesario implementar un método llamado List para me muestre la lista de las estaciones y crear una sesión para que me traiga los métodos correspondientes a los atributos de esta clase que se encuentran en los pojos creados anteriormente, enseguida creamos una consulta hql para que me traiga todas las estaciones

```

@Override
public List<Estaciones> mostrarEstaciones() {
    List<Estaciones> listarestaciones=null;
    Session session=HibernateUtil.getSessionFactory().openSession();
    Transaction transaction=session.beginTransaction();
    String hql="from Estaciones e left join fetch e.troncales t";
    try{
        listarestaciones=session.createQuery(hql).list();

        transaction.commit();
        session.close();
    }catch(Exception e){
        System.out.println(e.getMessage());
        transaction.rollback();
    }
}

```

Enseguida configuramos el método de nueva estación para ello creamos una sesión para que me atrape los nuevos registros dentro de la tabla estaciones

```

@Override
public void nuevoEstaciones(Estaciones estaciones) {
    Session session=null;
    try{
        session=HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        session.save(estaciones);
        session.getTransaction().commit();
    }catch(Exception e){
        System.out.println(e.getMessage());
        session.getTransaction().rollback();
    }finally{
        if(session!=null){
            session.close();
        }
    }
}

```

Ahora creamos el método de modificar estación la cual le colocamos un parámetro que atrape todas las estaciones insertadas en la base de datos, enseguida creamos una sesión la cual va modificar dichos datos utilizando el parámetro update, después de ello guardamos los cambios correspondientes con un commit, en caso que no quiera restablecer a las inserciones anteriores utilizamos el parámetro rollback

```

@Override
public void modificarEstaciones(Estaciones estaciones) {
    Session session=null;
    try{
        session=HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        session.update(estaciones);
        session.getTransaction().commit();
    }catch(Exception e){
        System.out.println(e.getMessage());
        session.getTransaction().rollback();
    }finally{
        if(session!=null){
            session.close();
        }
    }
}

```

Dentro de esta misma clase generamos el método de eliminar estación la cual es necesario generar un Query para hacer una consulta la cual tendrá el parámetro de delete y el parámetro por el cual queremos borrar es por el nombre de la estación que sea igual al id y esta al hacer eliminado se actualice dentro de la base de datos.

```

@Override
public int eliminarEstaciones(Estaciones estaciones) {
    Session session=null;
    int result=0;
    try{
        session=HibernateUtil.getSessionFactory().openSession();
        Query query = session.createQuery("delete from Estaciones where nombreEstacion=:id");

        session.beginTransaction();
        //session.delete(operadores);
        query.setString("id",estaciones.getNombreEstacion());
        result = query.executeUpdate();
        session.getTransaction().commit();

    }catch(Exception e){
        System.out.println(e.getMessage());
        session.getTransaction().rollback();
    }finally{
    }
}

```

Enseguida creamos el método de buscar en donde se crea una consulta hql donde nos mostrara los nombres de todas las estaciones que están insertadas hasta el momento en la base de datos y como resultado nos traerá la lista por nombre de la estación.

```

@Override
public List buscarEstaciones(Estaciones estaciones) {
    Session session=null;
    List<String> result=null;
    try{
        session=HibernateUtil.getSessionFactory().openSession();
        String hql={"from Estaciones e where e.nombreEstacion=:nombreestacion"};

        session.beginTransaction();
        //session.delete(operadores);
        result=(List) session.createQuery(hql)
            .setParameter("nombreestacion", estaciones.getNombreEstacion())
            .list();

        session.getTransaction().commit();

    }catch(Exception e){
        System.out.println(e.getMessage());
        session.getTransaction().rollback();
    }
}

```

Y por último creamos el método de listar troncales en donde se crea una sesión que atrape los atributos de los pojos correspondientes de la clase troncales, dentro de este método también es necesario implementar una consulta hql para que nos muestre todas las troncales

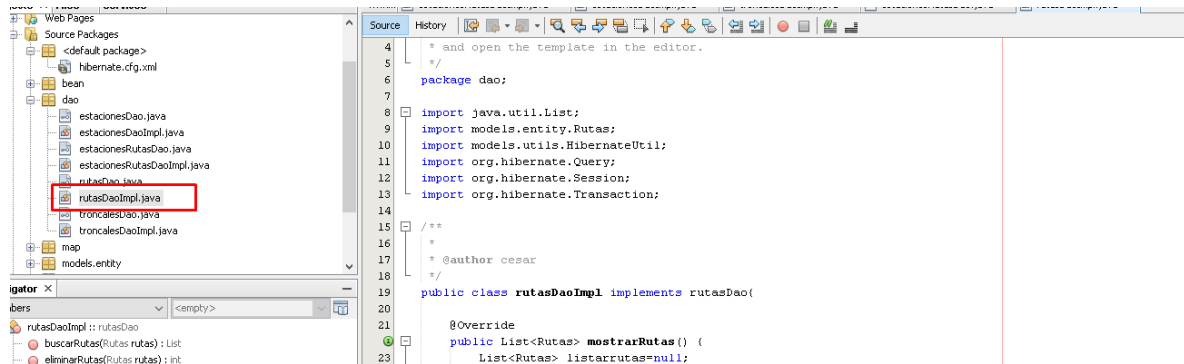
```
@Override
public List<Troncales> ListarTroncales() {
    List<Troncales> listartroncales=null;
    Session session=HibernateUtil.getSessionFactory().openSession();
    Transaction transaction=session.beginTransaction();
    String hql="from Troncales t";
    try{
        listartroncales=session.createQuery(hql).list();

        transaction.commit();
        session.close();
    }catch(Exception e){
        System.out.println(e.getMessage());
        transaction.rollback();
    }

    return listartroncales;
}
```

MOSTRAR GESTION RUTAS

Creamos una nueva clase la cual llamaremos rutasdaoimplement para generar los diferentes métodos de mostrar rutas, nueva ruta, modificar ruta, eliminar ruta, y buscar rutas



Enseguida vamos a desarrollar el método de mostrar rutas en donde es necesario implementar un método llamado List para que me muestre la lista de las rutas y crear una sesión para que me traiga los métodos correspondientes a los atributos de esta clase que se encuentran en los pojos creados anteriormente, enseguida creamos una consulta hql para que me traiga todas las rutas

```

public class rutasDaoImpl implements rutasDao{

    @Override
    public List<Rutas> mostrarRutas() {
        List<Rutas> listarrutas=null;
        Session session=HibernateUtil.getSessionFactory().openSession();
        Transaction transaction=session.beginTransaction();
        String hql="from Rutas r";
        try{
            listarrutas=session.createQuery(hql).list();

            transaction.commit();
            session.close();
        }catch(Exception e){
            System.out.println(e.getMessage());
            transaction.rollback();
        }

        return listarrutas;
    }
}

```

Enseguida configuramos el método de nueva ruta el cual cuenta con un parámetro de rutas para ello creamos una sesión para que me atrape los nuevos registros dentro de la tabla rutas

```

public void nuevoRutas(Rutas rutas) {
    Session session=null;
    try{
        session=HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        session.save(rutas);
        session.getTransaction().commit();
    }catch(Exception e){
        System.out.println(e.getMessage());
        session.getTransaction().rollback();
    }finally{
        if(session!=null){
            session.close();
        }
    }
}

```

Ahora creamos el método de modificar rutas la cual le colocamos un parámetro que atrape todas las rutas insertadas en la base de datos, enseguida creamos una sesión la cual va modificar dichos datos utilizando el parámetro update, después de ello guardamos los cambios correspondientes con un commit, en caso que no quiera restablecer a las inserciones anteriores utilizamos el parámetro rollback

```

@Override
public void modificarRutas (Rutas rutas) {
    Session session=null;
    try{
        session=HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        session.update(rutas);
        session.getTransaction().commit();
    }catch(Exception e){
        System.out.println(e.getMessage());
        session.getTransaction().rollback();
    }finally{
        if(session!=null){
            session.close();
        }
    }
}

```

Dentro de esta misma clase generamos el método de eliminar rutas la cual es necesario generar un Query para hacer una consulta la cual tendrá el parámetro de delete y el parámetro por el cual queremos borrar en este caso es por el id de la ruta, esta al haber sido eliminado debe actualizar los registros dentro de la base de datos

```

public int eliminarRutas (Rutas rutas) {
    Session session=null;
    int result=0;
    try{
        session=HibernateUtil.getSessionFactory().openSession();
        Query query = session.createQuery("delete from Rutas where idRuta=:id");

        session.beginTransaction();
        //session.delete(operadores);
        query.setString("id",rutas.getIdRuta());
        result = query.executeUpdate();
        session.getTransaction().commit();

    }catch(Exception e){
        System.out.println(e.getMessage());
        session.getTransaction().rollback();
    }finally{
        if(session!=null){
            session.close();
        }
    }
}

```

Enseguida creamos el método de buscar en donde en él se crea una consulta hql donde nos mostrara los id de las rutas que están insertadas hasta el momento en la base de datos y como resultado nos arroja dentro de la consulta los id.

```

@Override
public List buscarRutas(Rutas rutas) {
    Session session=null;
    List result=null;
    try{
        session=HibernateUtil.getSessionFactory().openSession();
        String hql=("from Rutas r where r.idRuta=:idruta");

        session.beginTransaction();
        //session.delete(operadores);
        result=(List) session.createQuery(hql)
        .setParameter("idruta", rutas.getIdRuta())
        .list();

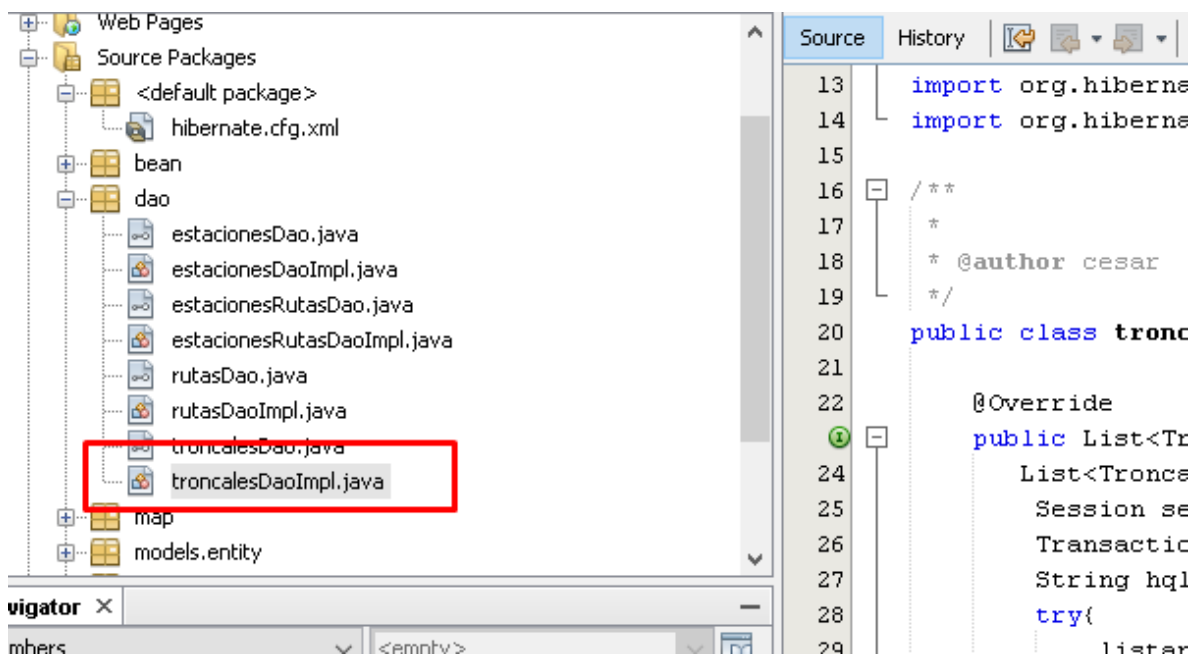
        session.getTransaction().commit();

    }catch(Exception e){
        System.out.println(e.getMessage());
        session.getTransaction().rollback();
    }finally{

```

MOSTRAR GESTION TRONCALES

Creamos una nueva clase la cual llamaremos troncalesdaoimplement para generar los diferentes métodos de mostrar tronca , nueva troncal, modificar troncal, eliminar troncal, y buscar troncal



Enseguida vamos a desarrollar el método de mostrar troncal en donde es necesario implementar un método llamado List para que me muestre la lista de las troncales y crear una sesión para que me traiga los métodos correspondientes a los atributos de esta clase que se encuentran en los pojos creados anteriormente, enseguida creamos una consulta hql para que me traiga todas las troncales


```

@Override
public List<Troncales> mostrarTroncales() {
    List<Troncales> listartroncales=null;
    Session session=HibernateUtil.getSessionFactory().openSession();
    Transaction transaction=session.beginTransaction();
    String hql="from Troncales t";
    try{
        listartroncales=session.createQuery(hql).list();

        transaction.commit();
        session.close();
    }catch(Exception e){
        System.out.println(e.getMessage());
        transaction.rollback();
    }

    return listartroncales;
}

```

Enseguida configuramos el método de nueva troncal el cual cuenta con un parámetro de troncales para ello creamos una sesión para que me atrape los nuevos registros dentro de la tabla troncales

```

-----
public void nuevoTroncales(Troncales troncales) {
    Session session=null;
    try{
        session=HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        session.save(troncales);
        session.getTransaction().commit();
    }catch(Exception e){
        System.out.println(e.getMessage());
        session.getTransaction().rollback();
    }finally{
        if(session!=null){
            session.close();
        }
    }
}

```

Ahora creamos el método de modificar troncales la cual le colocamos un parámetro que atrape todas las troncales insertadas en la base de datos, enseguida creamos una sesión la cual va modificar dichos datos utilizando el parámetro update, después de ello guardamos los cambios correspondientes con un commit, en caso que no quiera restablecer a las inserciones anteriores utilizamos el parámetro rollback

```

@Override
public void modificarTroncales(Troncales troncales) {
    Session session=null;
    try{
        session=HibernateUtil.getSessionFactory().openSession();
        session.beginTransaction();
        session.update(troncales);
        session.getTransaction().commit();
    }catch(Exception e){
        System.out.println(e.getMessage());
        session.getTransaction().rollback();
    }finally{
        if(session!=null){
            session.close();
        }
    }
}

```

Dentro de esta misma clase generamos el método de eliminar troncales la cual es necesario generar un Query para hacer una consulta la cual tendrá el parámetro de delete y el parámetro por el cual queremos borrar en este caso es por el id de la troncal, y esta al haber sido eliminado debe actualizar los registros dentro de la base de datos

```

public int eliminarTroncales(Troncales troncales) {
    Session session=null;
    int result=0;
    try{
        session=HibernateUtil.getSessionFactory().openSession();
        Query query = session.createQuery("delete from Troncales where idTroncal=:id");

        session.beginTransaction();
        //session.delete(operadores);
        query.setString("id", troncales.getIdTroncal());
        result = query.executeUpdate();
        session.getTransaction().commit();

    }catch(Exception e){
        System.out.println(e.getMessage());
        session.getTransaction().rollback();
    }finally{
    }
}

```

Enseguida creamos el método de buscar en donde en él se crea una consulta hql donde nos mostrara los id de las troncales que están insertadas hasta el momento en la base de datos y como resultado nos arrojará dentro de la consulta los id.

```

@Override
public List buscarTroncales(Troncales troncales) {
    Session session=null;
    List result=null;
    try{
        session=HibernateUtil.getSessionFactory().openSession();
        String hql=("from Troncales t where t.idTroncal=:idtroncal");

        session.beginTransaction();
        //session.delete(operadores);
        result=(List) session.createQuery(hql)
            .setParameter("idtroncal", troncales.getIdTroncal())
            .list();

        session.getTransaction().commit();

    }catch(Exception e){
        System.out.println(e.getMessage());
        session.getTransaction().rollback();
    }finally{
        if(session!=null){
    }
}

```

Para poder controlar los métodos creados en las clases anteriores es necesario crear dentro del paquete bean los archivos java server para que controle todos los métodos que se hicieron anterior mente dentro del paquete dao,los cuales son implementados dentro de los objetos es decir los pojos, con la funcionalidad de insertar, modificar, consultar, listar y eliminar

```

public class estacionesRutasBean {
    private List<EstacionesRutas> listarestacionesrutas;
    private List<SelectItem> listarestaciones;
    private List<SelectItem> listarrutas;
    private EstacionesRutas estacionesrutas;

    /**
     * Creates a new instance of estacionesRutasBean
     */
    public estacionesRutasBean() {
        estacionesrutas=new EstacionesRutas();
    }

    public List<EstacionesRutas> getListarestacionesrutas() {
        estacionesRutasDao etDao=new estacionesRutasDaoImpl();
        listarestacionesrutas=etDao.mostrarEstacionesRutas();
        return listarestacionesrutas;
    }

    public List<SelectItem> getListarestaciones() {
        this.listarestaciones=new ArrayList<SelectItem>();
        estacionesRutasDao etDao=new estacionesRutasDaoImpl();
        List<Estaciones> e=etDao.ListarEstaciones();
    }
}

```

Esto se debe hacer por cada una de las clases creadas tanto para estaciones como troncales y rutas

```

    }

    public List<SelectItem> getListarTroncales() {
        this.listartroncales=new ArrayList<SelectItem>();
        estacionesDao eDao=new estacionesDaoImpl();
        List<Troncales> tr=eDao.ListarTroncales();

        for(Troncales tronc:tr){
            SelectItem troncalesitem=new SelectItem(tronc.getIdTroncal(),tronc.getNombreTroncal());
            this.listartroncales.add(troncalesitem);
        }
        return listartroncales;
    }

    public void insertarEstaciones(){
        this.resultnombrestacion=new ArrayList<String>();
        estacionesDao eDao=new estacionesDaoImpl();
        resultnombrestacion= eDao.buscarEstaciones(estaciones);
        if(resultnombrestacion.size()==0){
            eDao.nuevoEstaciones(estaciones);

            FacesContext.getCurrentInstance().addMessage(null,new FacesMessage(FacesMessage.SEVERITY_INFO,"Co")
        )else{
    }
}

```

Para la clase ruta

```

private List<Rutas> listarrutas;
/**
 * Creates a new instance of rutasBean
 */
public rutasBean() {
    rutas=new Rutas();
}

public List<Rutas> getListarrutas() {
    rutasDao rDao=new rutasDaoImpl();
    listarrutas=rDao.mostrarRutas();
    return listarrutas;
}

public void insertarRutas(){
    rutasDao rDao=new rutasDaoImpl();
    if(rDao.buscarRutas(rutas).size()==0){
        rDao.nuevoRutas(rutas);

        FacesContext.getCurrentInstance().addMessage(null,new FacesMessage(FacesMessage.SEVERITY_INFO,"Co

```

Troncales

dentro del proceso de remover estación se hace una comparación si existe dicho registro lo elimine si es así que lo elimine de aunó

```

tDao.nuevoTroncales(troncales);

FacesContext.getCurrentInstance().addMessage(null,new FacesMessage(FacesMessage.SEVERITY_INFO,"Corr
    )else{
        FacesContext.getCurrentInstance().addMessage(null,new FacesMessage(FacesMessage.SEVERITY_ER
    )
    troncales=new Troncales();
}

public void modificarTroncales(){
    troncalesDao tDao=new troncalesDaoImpl();
    tDao.modificarTroncales(troncales);
    troncales=new Troncales();
}

public void removerTroncales(){
    troncalesDao tDao=new troncalesDaoImpl();
    if(tDao.eliminarTroncales(troncales)==1){
        FacesContext.getCurrentInstance().addMessage(null,new FacesMessage(FacesMessage.SEVERITY_INFO,"Corr

    }else{
        FacesContext.getCurrentInstance().addMessage(null,new FacesMessage(FacesMessage.SEVERITY_ERROR,"E
    )
    troncales=new Troncales();
}

```

Enseguida vamos a codificar el index de nuestra pagina donde desarrollamos un formulario para cada una de las gestiones de rutas, troncales, estaciones y paradas donde creamos diferentes form para llamar los respectivos métodos creados,

```

</style>
<title>GESTION ESTACIONES</title>
</h:head>
<h:body>
<p:growl id="mensajeGeneral" autoUpdate="true" showDetail="true" life="3000"/>
<h:form id="formMenuTransmilenio">
<p:dock position="bottom">
<p:menuItem value="INICIO" icon="recursos/img/iconoinicio.png" url="index.xhtml"/>
<p:menuItem value="TRONCAL" icon="recursos/img/iconotroncal.png" url="paginatroncales.xhtml">
<p:menuItem value="ESTACIONES" icon="recursos/img/iconoestaciones.png" url="paginaestaciones
<p:menuItem value="RUTAS" icon="recursos/img/iconoruta.png" url="paginarutas.xhtml"/>
<p:menuItem value="ESTACIONES RUTAS" icon="recursos/img/iconoestrutas.png" url="paginaestaci
</p:dock>
</h:form>
<h:form id="formInsertarMostrarEstaciones">
<p:panelGrid columns="2" style="margin:0 auto">
<p:outputLabel value="Nombre de la estacion"/>
<p:inputText value="#(estacionesBean.estaciones.nombreEstacion)"
    required="true" requiredMessage="nombre de la estacion es obligatorio"
<p:outputLabel value="id de la troncal"/>
<p:selectOneMenu id="idtroncale" value="#(estacionesBean.estaciones.troncales.idTroncal"
    required="true" requiredMessage="la troncal es obligatorio">
<f:selectItem itemLabel="Seleccione una troncal" itemValue="" />
<f:selectItems value="#(estacionesBean.listarTroncales)" />
<f:ajax:execute id="troncale"/>

```

y de igual forma vamos a insertar dentro de estos formularios una tabla para que nos muestre los datos que este contiene insertados dentro de cada tabla de la base de dato.

```
h:form id="formMostrarEstaciones">
  <p:dataTable id="tablaMostrarEstaciones" var="est" styleClass="myTable" rowStyleClass="highlight"
    emptyMessage="No hay datos de Troncales">
```

dentro de estos mismos vamos a insertar unas cajas de texto donde se pueda rellenar los datos correspondientes para que se pueda insertar datos dentro de cada una de las gestiones del Transmilenio, este a la ves contendrá dos botones con funcionalidad de guardar y cancelar respectivamente.

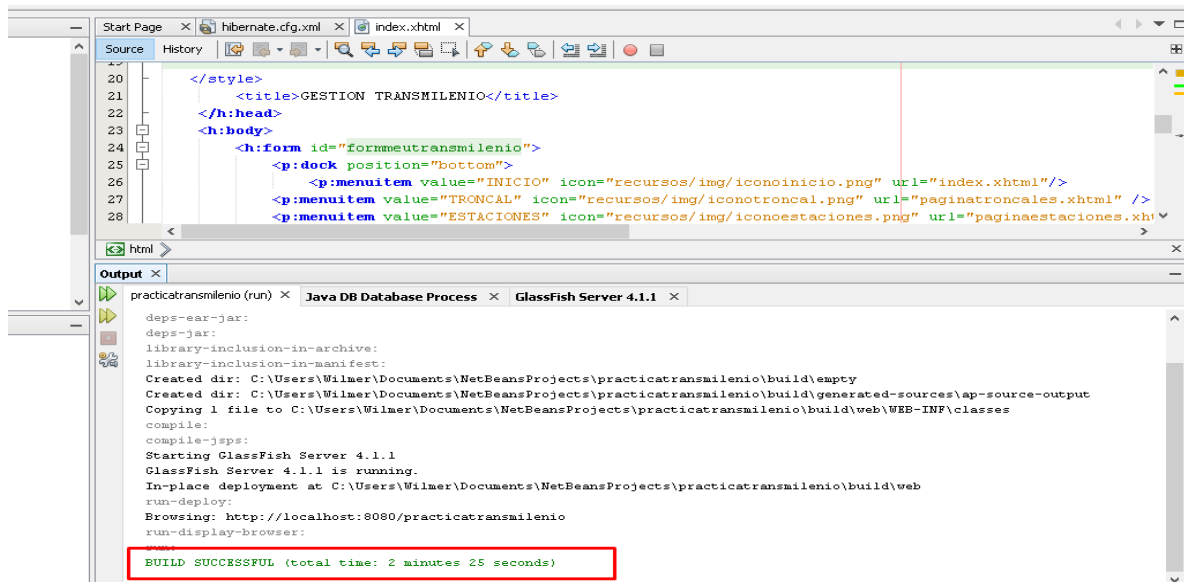
```
<p:outputLabel id="tipob" value="Tipo de Bus"/>
<p:selectOneMenu id="idtipob" value="#{rutasBean.rutas.tipoBus}"
  required="true" requiredMessage="la troncal es obligatorio">
  <f:selectItem itemLabel="Seleccione un tipo de bus" itemValue=""/>
  <f:selectItem itemLabel="Transmilenio" itemValue="T"/>
  <f:selectItem itemLabel="Alimentador" itemValue="A"/>
  <f:selectItem itemLabel="Urbano" itemValue="U"/>
  <f:ajax execute="idtipob"/>
</p:selectOneMenu>

<p:commandButton value="Guardar" actionListener="#{rutasBean.insertarRutas()}"
  update=":formMostrarRutas:tablaMostrarRutas :formInsertarMostrarRutas"/>
<p:commandButton value="Cancelar" action="#{rutasBean.cancelarRutas()}"
```

Dentro de las gestiones de rutas, troncales, estaciones se encontrara el botón de modificar y eliminar respectivamente dentro de la tabla

```
<f:setPropertyActionListener target="#{estacionesBean.estaciones}" value="#{est}" />
</p:column>
<p:column headerText="Modificar">
  <p:commandButton value="Modificar" oncomplete="PF('dlgModificar').show();"
    update=":formModificar" style="background:#0033ff;color: #FFFFFF;font"
    <f:setPropertyActionListener target="#{estacionesBean.estaciones}" value="#{est}"
  </p:commandButton>
</p:column>
<p:column headerText="Eliminar">
  <p:commandButton value="Eliminar" oncomplete="PF('dlgEliminar').show();"
    update=":formEliminar" style="background:#E30D34;color: #FFFFFF;font"
    <f:setPropertyActionListener target="#{estacionesBean.estaciones}" value="#{est}"
  </p:commandButton>
</p:column>
</p:dataTable>
```

Ejecutamos nuestro proyecto



Vamos a nuestro navegador y verificamos que nos esta mostrando la pagina de inicio donde se puede evidenciar la gestión troncal, gestión estaciones gestión rutas, gestión paradas rutas

