

Introduction to using the Dali Plugin in Eclipse for JPA

Dominic Duggan, Stevens Institute of Technology

Step 1: Installation

You should download and install Eclipse for JEE, which contains the Dali plugin.

You should also download and install the Glassfish application server on your local machine. You will not actually be running this application server, but Eclipse will need some of the jar files. You will also need to download the latest JDBC driver for Postgresql. You already installed it in your EC instance in the last assignment. This time you must download it to your local machine, so you can make it available to your IDE (Eclipse).

Step 2: Running the Database Server

You will need to run the database server when generating the database schema from your domain model. Start your EC2 instance and connect to it using ssh. Then start the server (remember that you should run it as user postgres):

```
sudo su -  
su postgres -  
/usr/bin/pg_ctl start -D /data -l /data/postgresql.log
```

To stop the database server when you are finished, type:

```
/usr/bin/pg_ctl stop -D /data  
exit  
exit  
sudo shutdown -h now
```

Remember to stop your EC2 instance.

Step 3: Create a New JPA Project

Create a new JPA project, and choose a target runtime:

New JPA Project

Configure JPA project settings.

Project name:

Project location

☒ Use default location

Location:

Target runtime

JPA version

Configuration

The default configuration provides a good starting point. Additional facets can later be installed to add new functionality to the project.

EAR membership

☐ Add project to an EAR

EAR project name:

Choose Glassfish 3.1 as your target runtime. If the adapter for Glassfish is not in your Eclipse environment (by default it is not), you will need to download it by clicking on “Download additional server adapters,” and reboot Eclipse once the Glassfish 3.1 adapter is installed.

New JPA Project

Configure JPA project settings.

Project name:

Project location

☒ Use default location

Location:

Target runtime

JPA version

Configuration

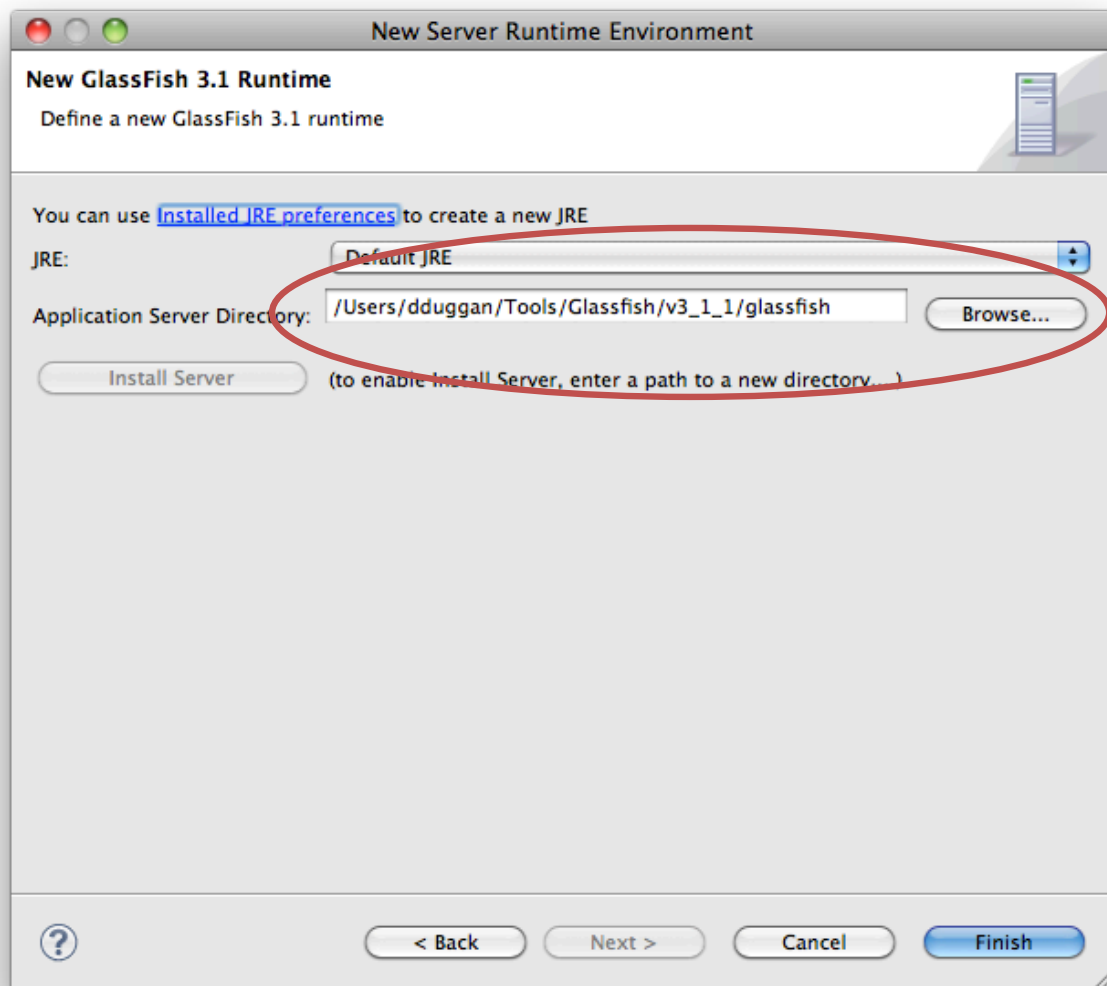
The default configuration provides a good starting point. Additional facets can later be installed to add new functionality to the project.

EAR membership

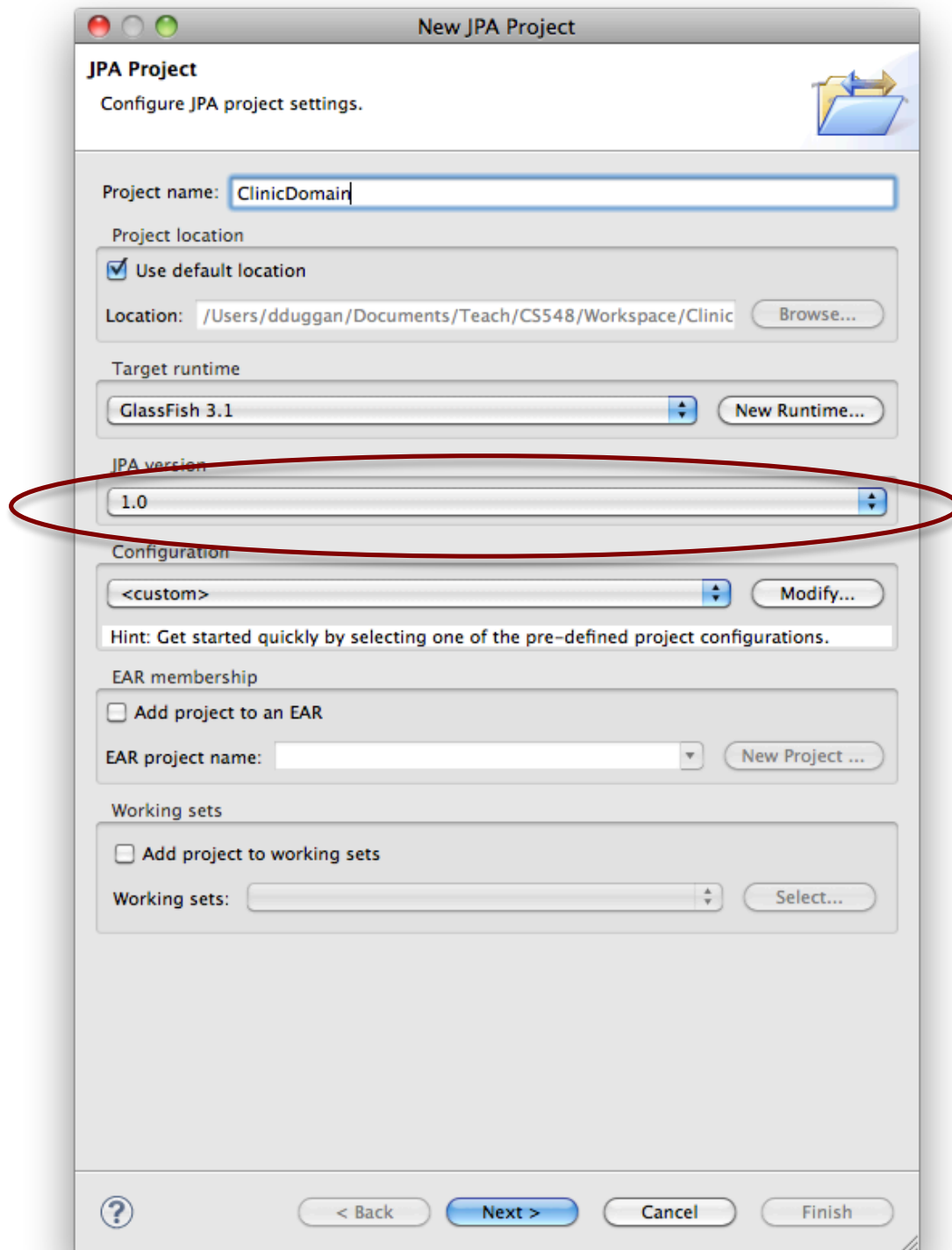
☐ Add project to an EAR

EAR project name:

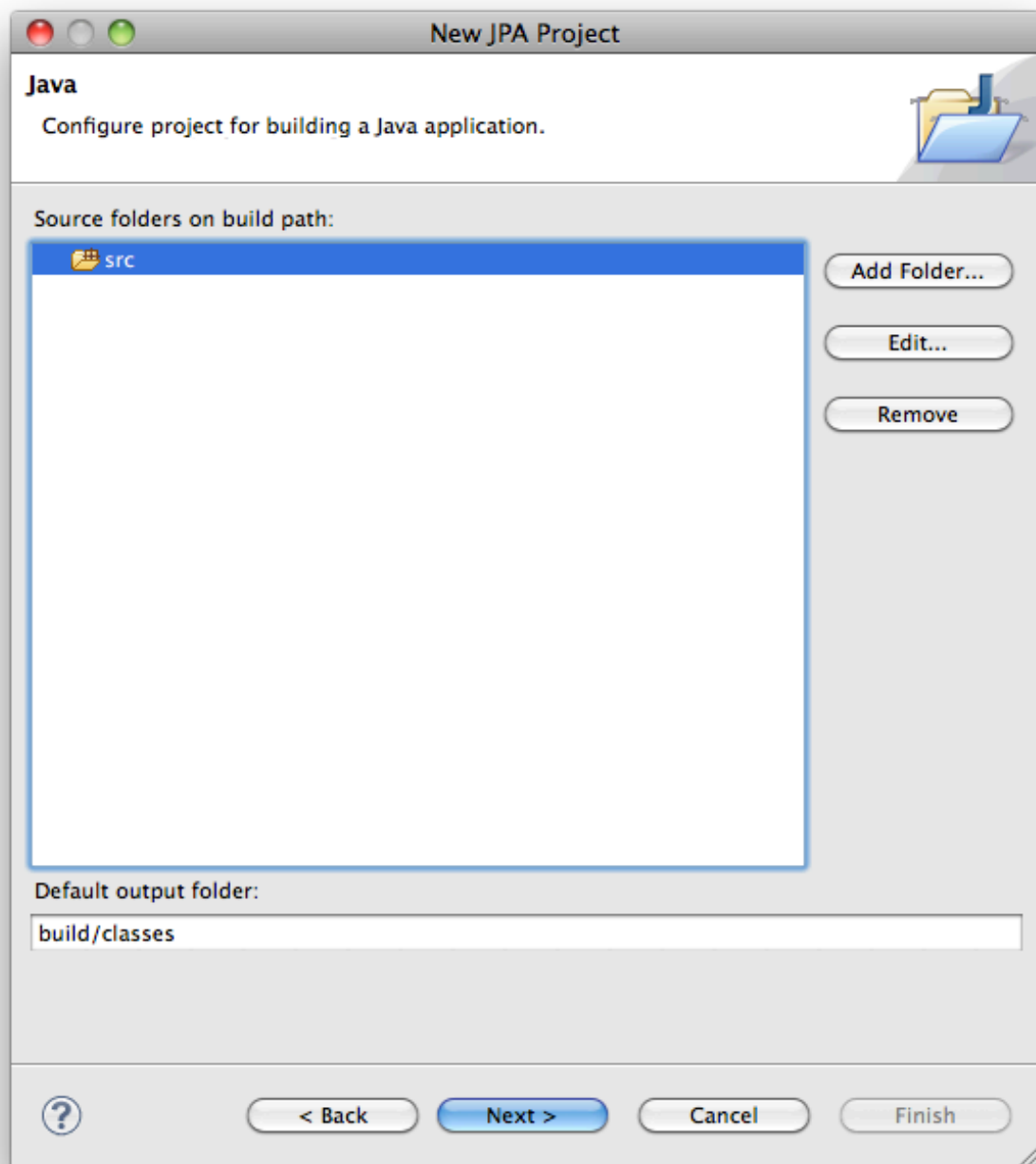
Once you have installed the Glassfish 3.1 adapter, click on “Next” and navigate to the directory where the Glassfish application server is installed on your local machine. Select the subdirectory named “glassfish”, this tells Eclipse where to find the jar file containing the Glassfish classes (needed for compiling Java code in the editor):



Click finish, and return to the wizard for creating a new JPA project. Choose JPA 1.0 as the JPA version, since Eclipse claims that 2.0 is not compatible with Glassfish 3.1:



Click “Next” and choose the source folder:



Click “Next” again and choose the JPA implementation. Choose the EclipseLink platform. Click the button for downloading an ORM runtime:

New JPA Project

JPA Facet

At least one user library must be selected.

Platform: EclipseLink 1.2.x

JPA implementation

Type: User Library

☐ Include libraries with this application

Connection: <None>

[Add connection...](#)

[Connect](#)

☐ Add driver library to build path

Driver:

☐ Override default catalog from connection

Catalog:

☐ Override default schema from connection

Schema:

Persistent class management

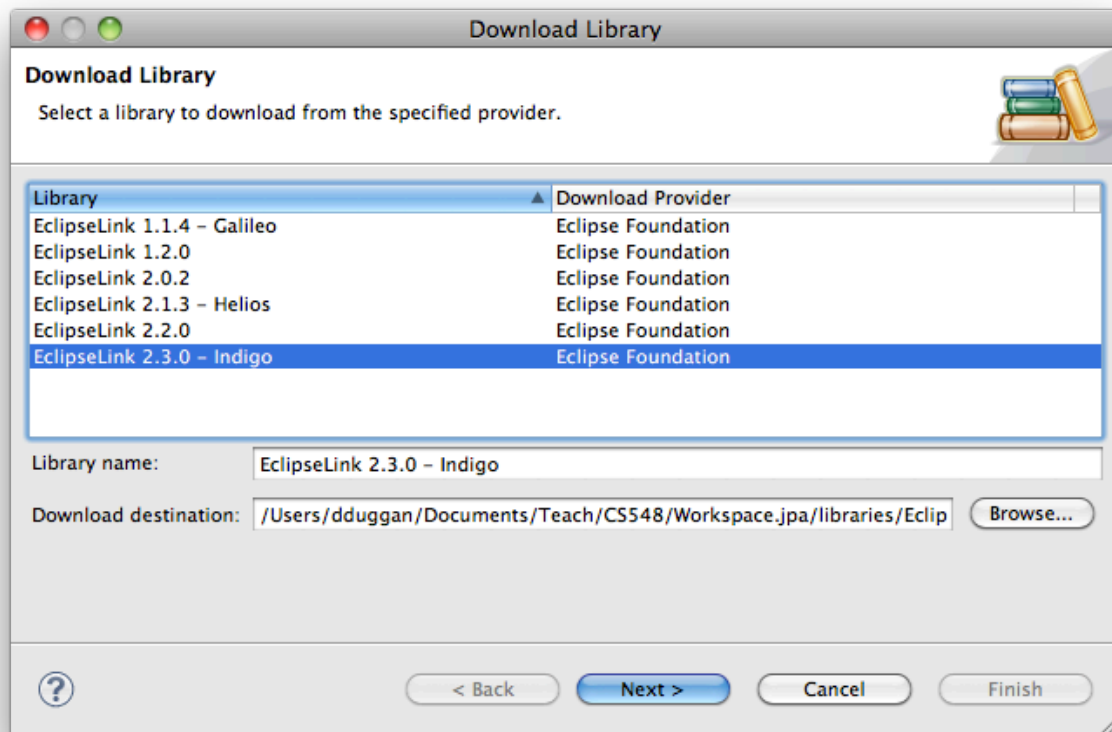
☒ Discover annotated classes automatically

☐ Annotated classes must be listed in persistence.xml

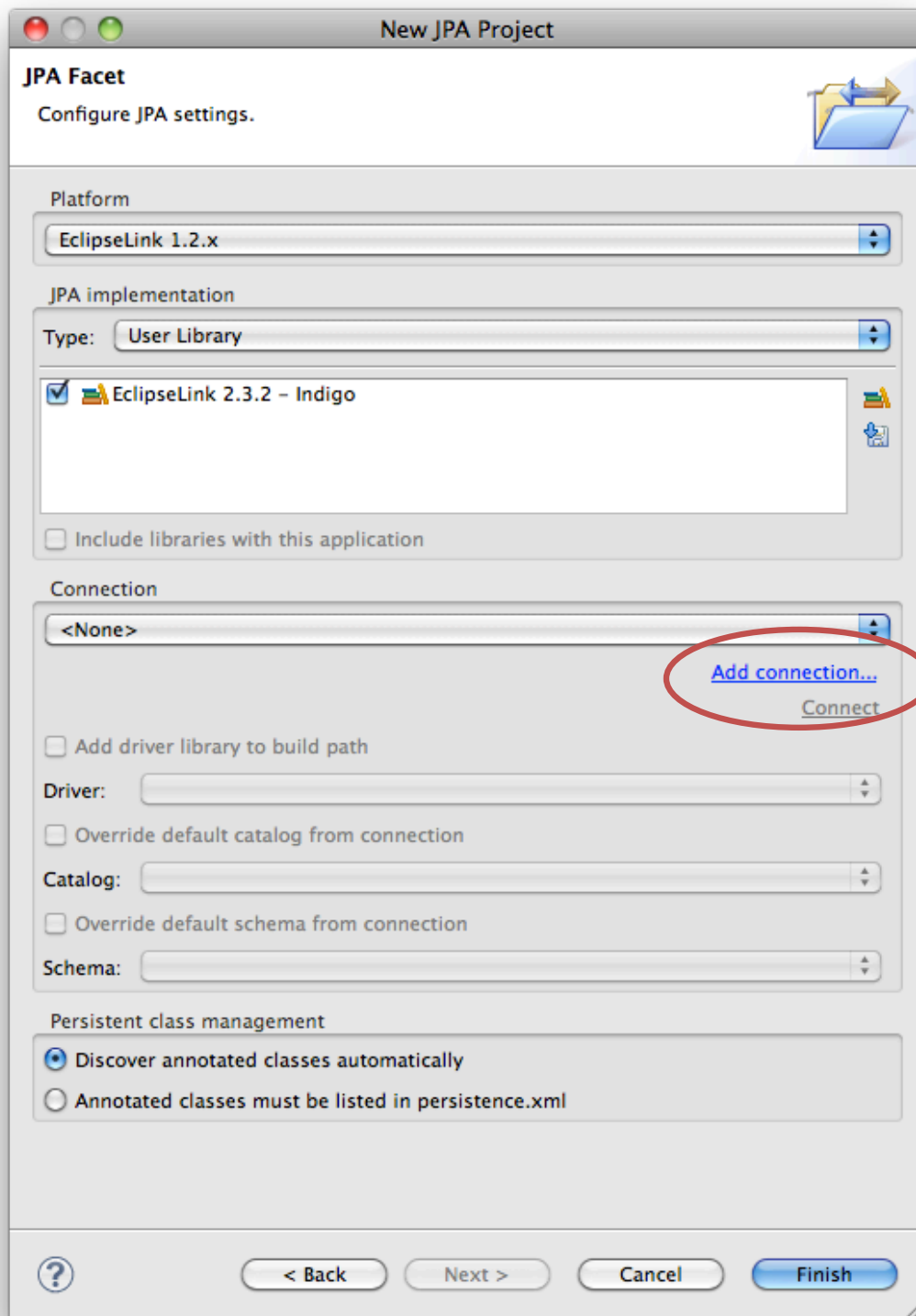
☐ Create mapping file (orm.xml)

< Back Next > Cancel Finish

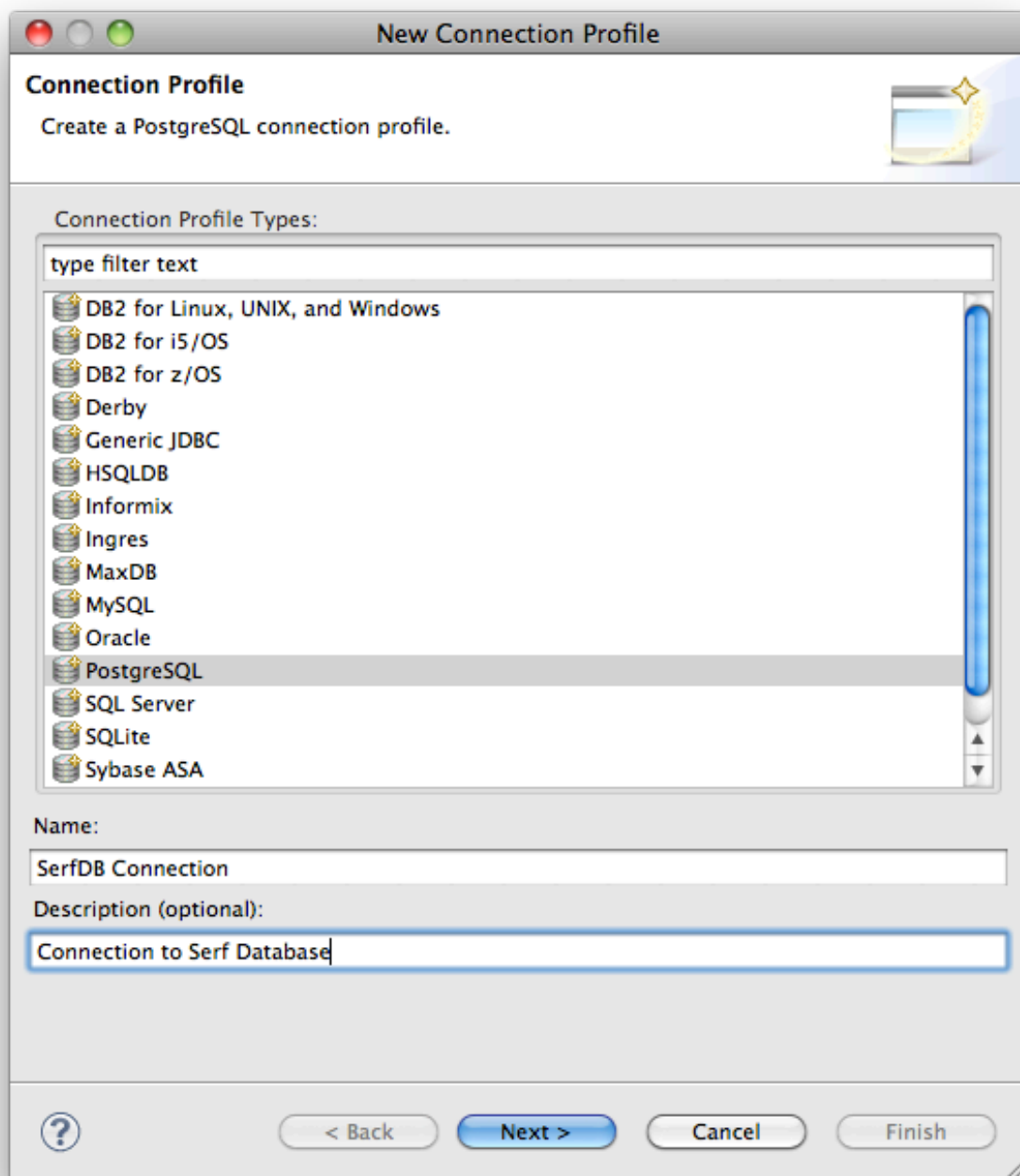
EclipseLink is the reference implementation of JPA. Choose the EclipseLink library for the Eclipse version that you are using:



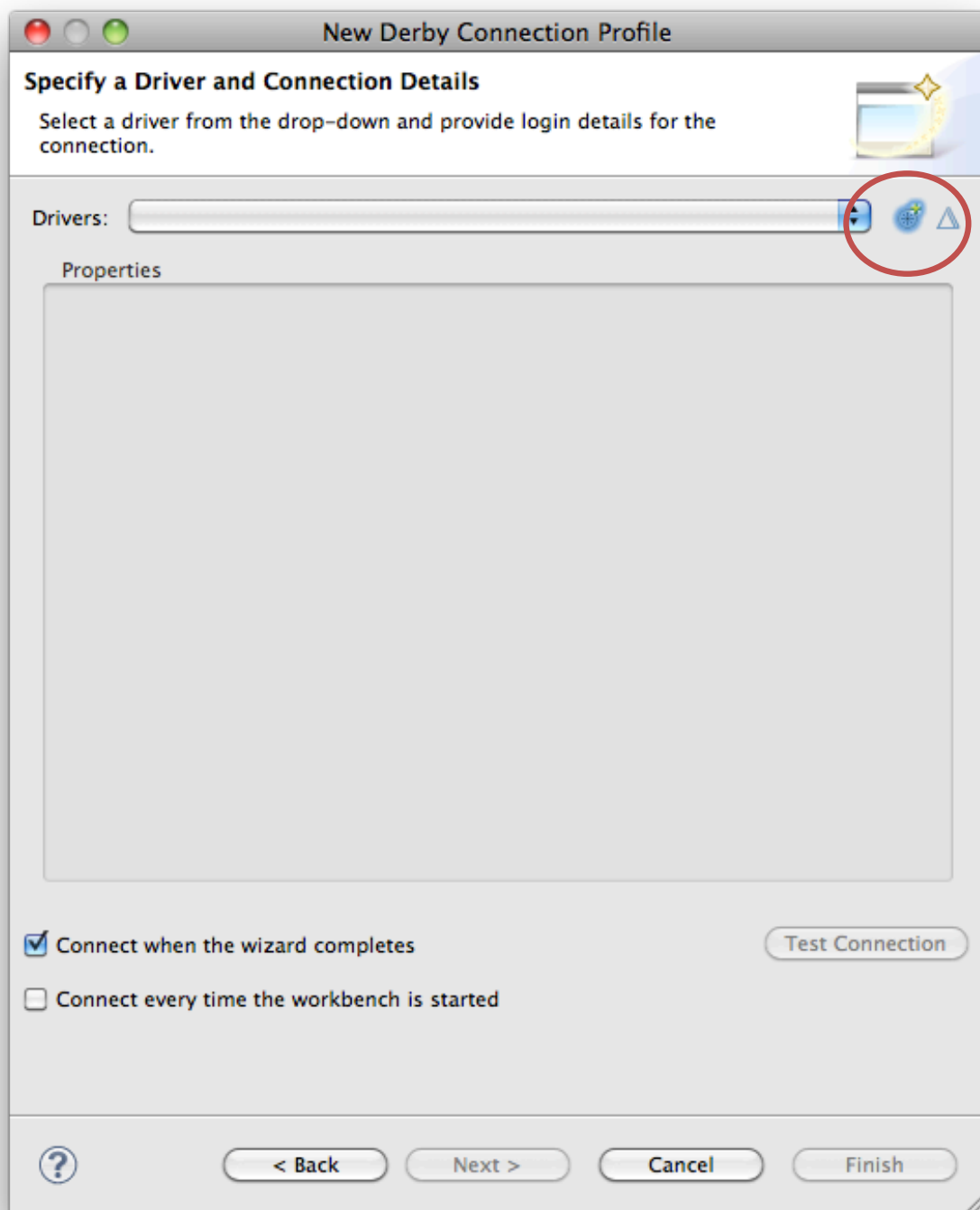
At this point you have installed the EclipseLink ORM runtime in your IDE. The next step is to configure a connection to your running database:



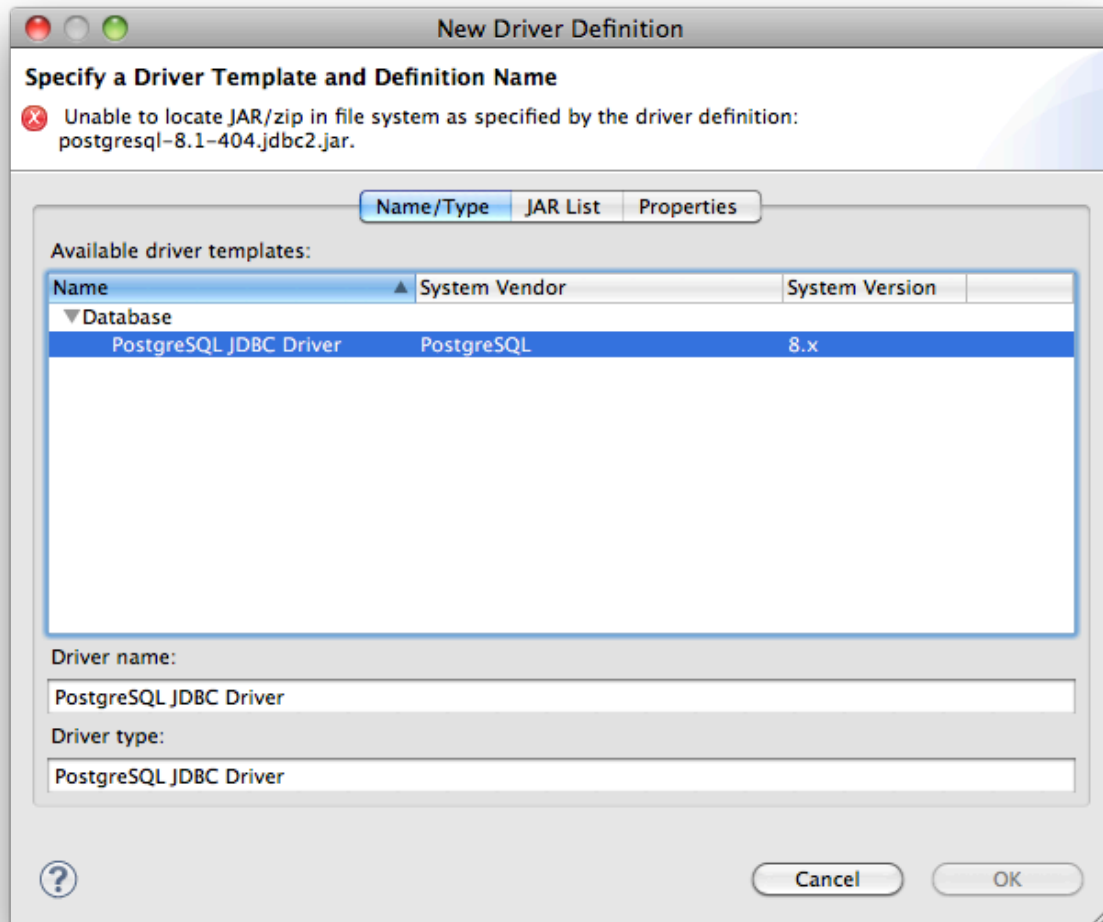
Choose Postgresql as your connection profile:



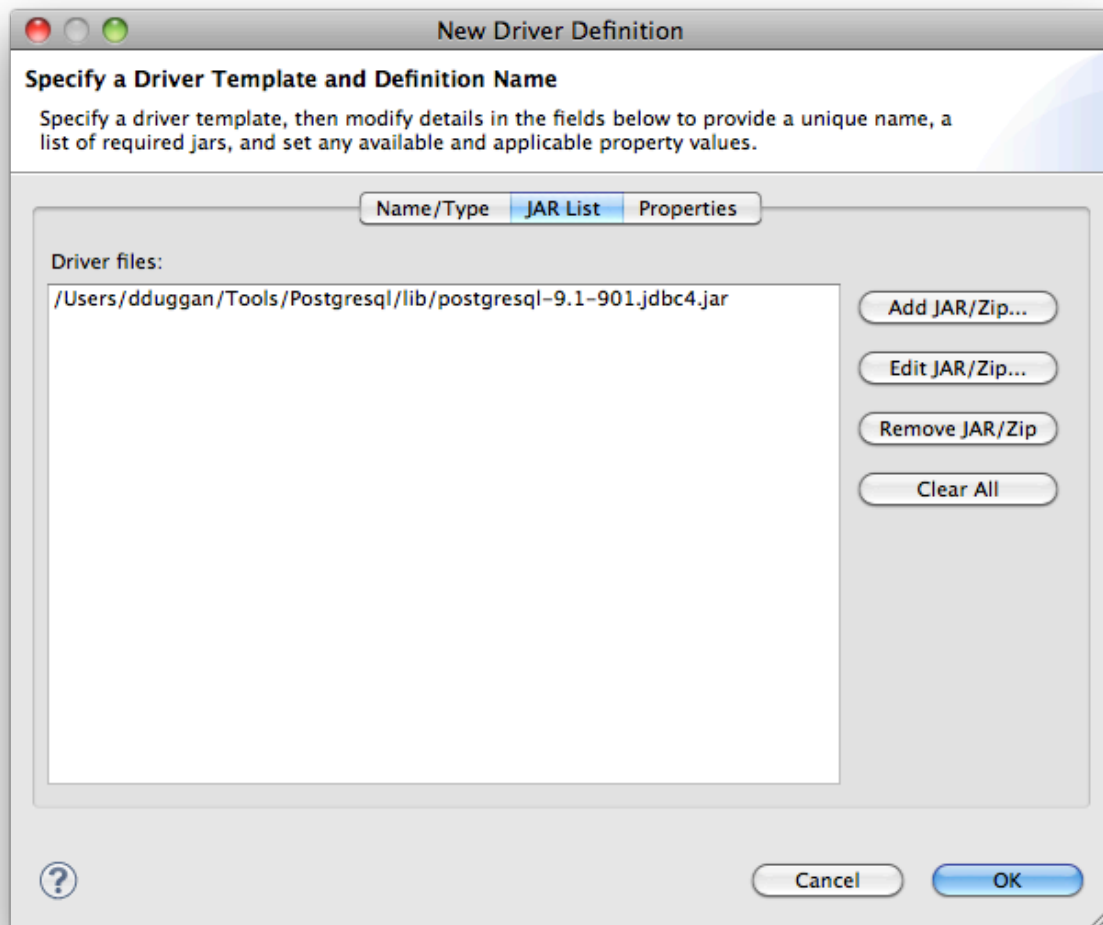
You will need to install the driver for Postgresql. Click the button to download:



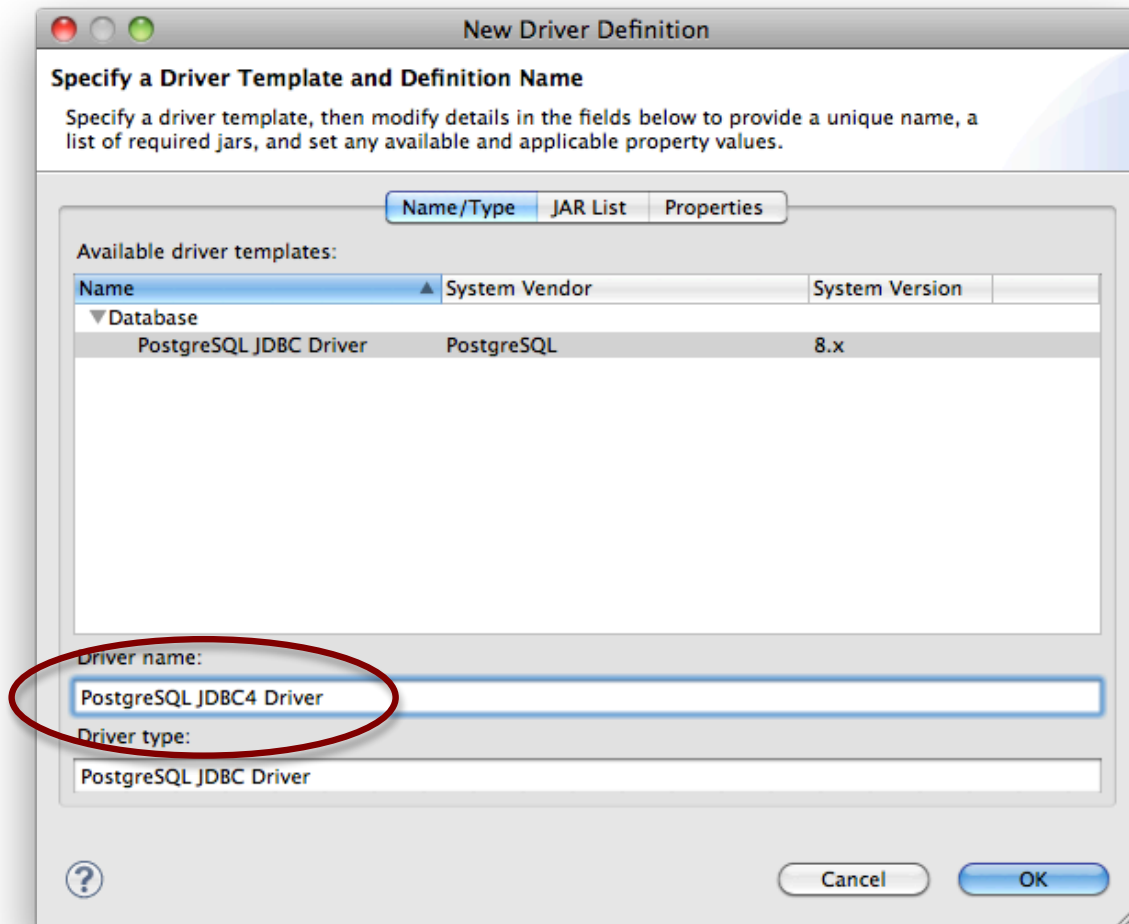
Choose the PostgreSQL JDBC driver type. Ignore its recommendation to download a JDBC2 driver, you should be running the latest JDBC4 driver for Java 6 or Java 7. You downloaded the driver to your EC2 instance, for this exercise you should have also downloaded the JDBC4 driver to your local machine:



On the “JAR List” tab, click “Clear All”, click “Add JAR/Zip...” and navigate to where you have downloaded the latest JDBC4 driver:



Finally give a name to this driver definition:



Click “OK” and set the connection properties. Set the database name, and the credentials to be used for authenticating to the database server. These should be the same as you used when setting up your database in an earlier exercise. Finally, set the URL for the JDBC connection. You may want to install Postgresql on your local machine while developing, but eventually you should be able to run on the database running in the cloud, so “localhost” will be need to be replaced with the public DNS name for your running instance. Click “Test Connection” (with the database server running!) to check your setup.

New JDBC Connection Profile

Specify a Driver and Connection Details

Select a driver from the drop-down and provide login details for the connection.

Drivers: PostgreSQL JDBC4 Driver

Properties

General Optional

Database: serfdb

URL: jdbc:postgresql://localhost:5432/serfdb

User name: serf

Password:

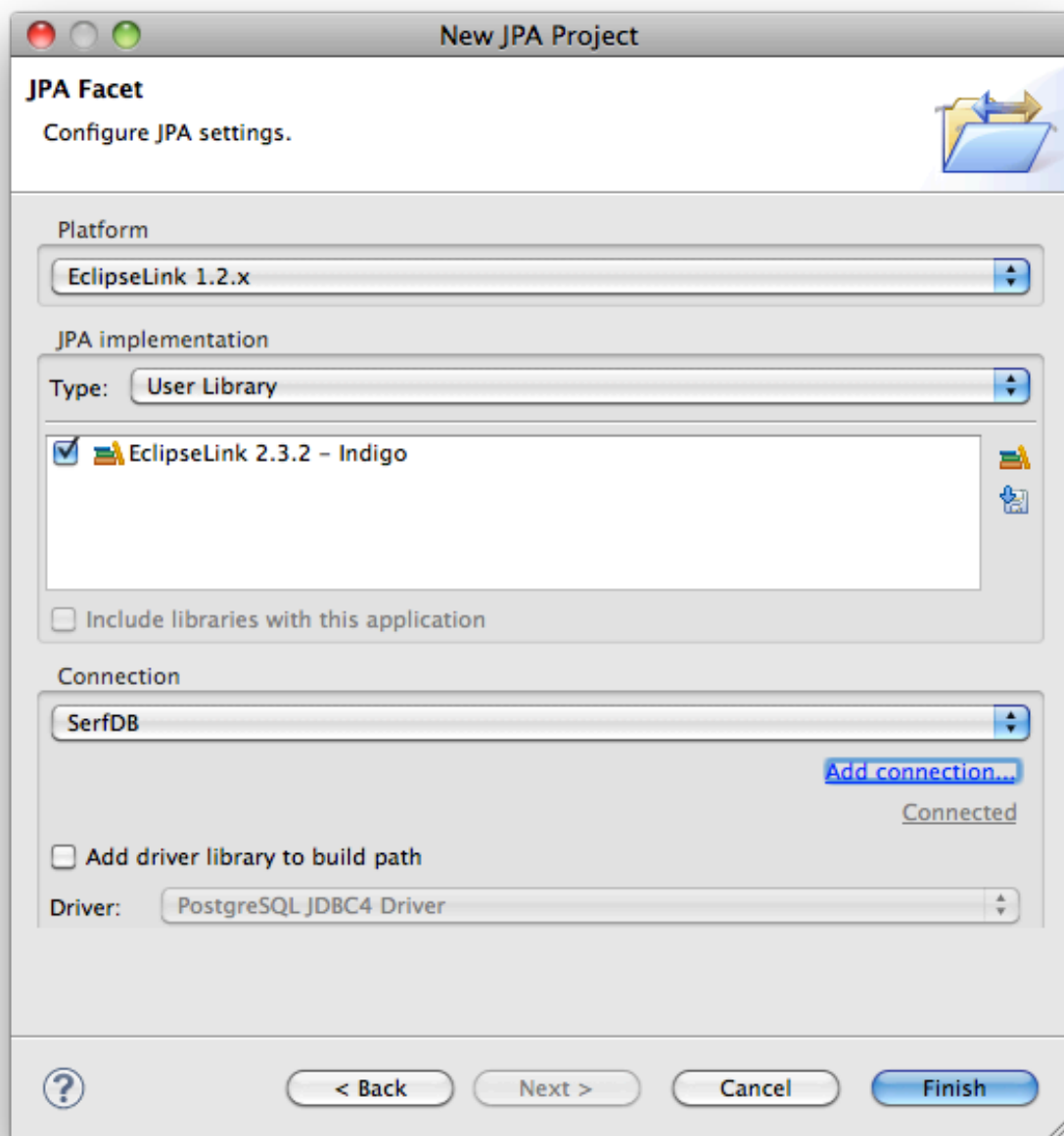
☒ Save password

☒ Connect when the wizard completes Test Connection

☐ Connect every time the workbench is started

? < Back Next > Cancel Finish

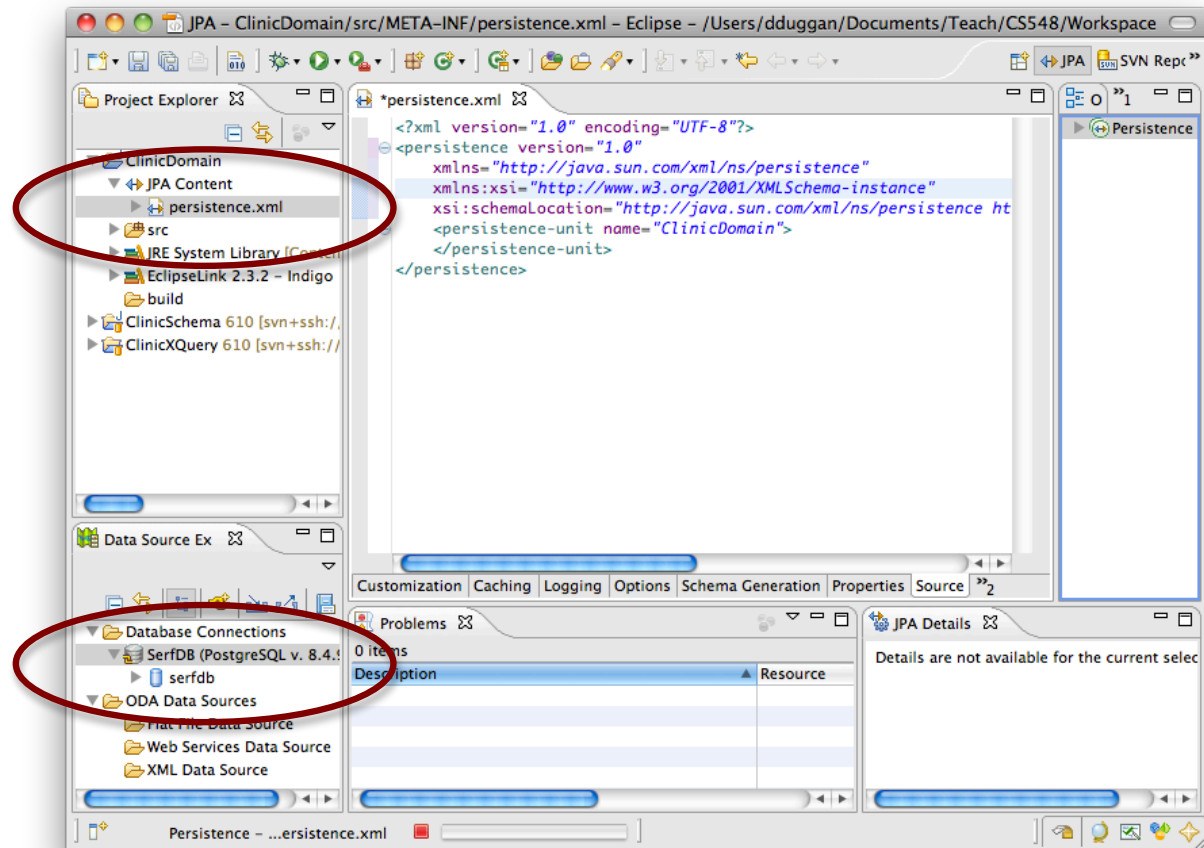
Click “Finish” to return to the JPA Project wizard.



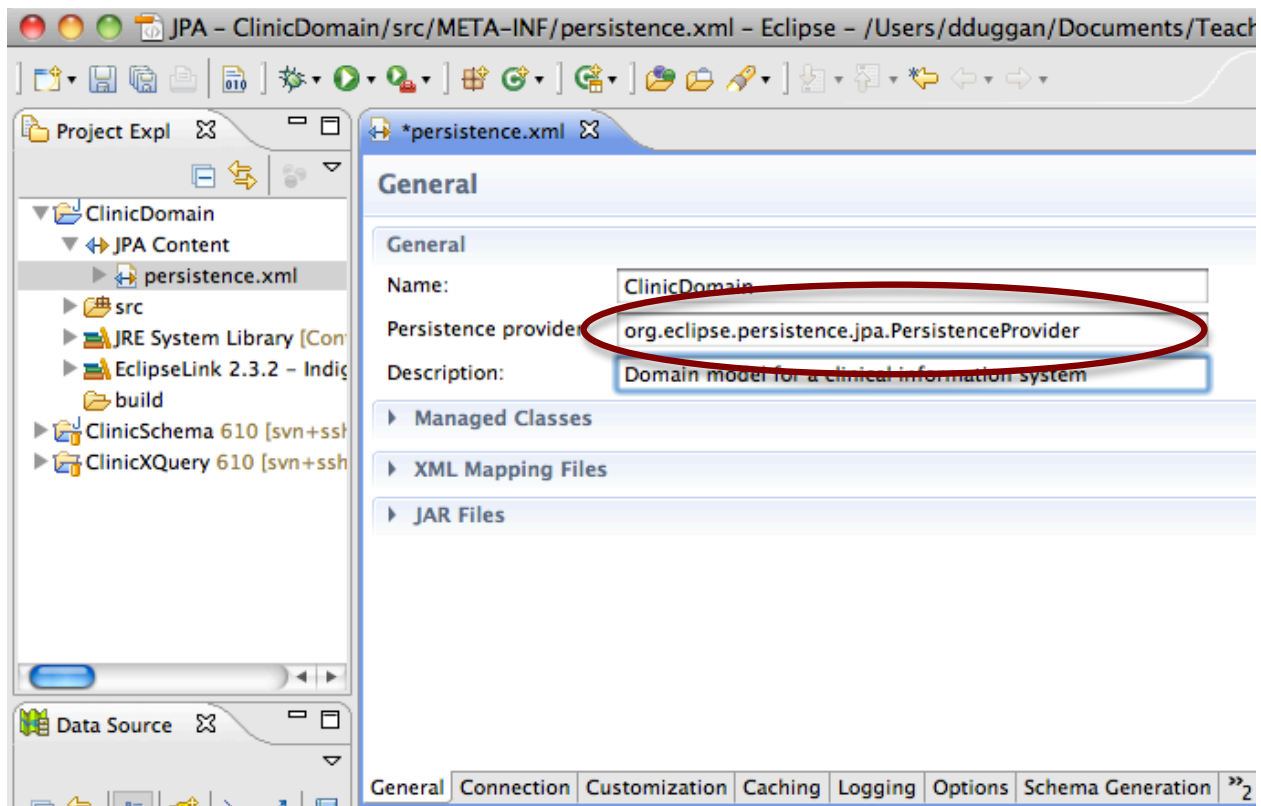
Click “Finish” again to finish the creation of the project, and Eclipse will switch to the JPA perspective.

Step 4: Configure the Persistence Unit

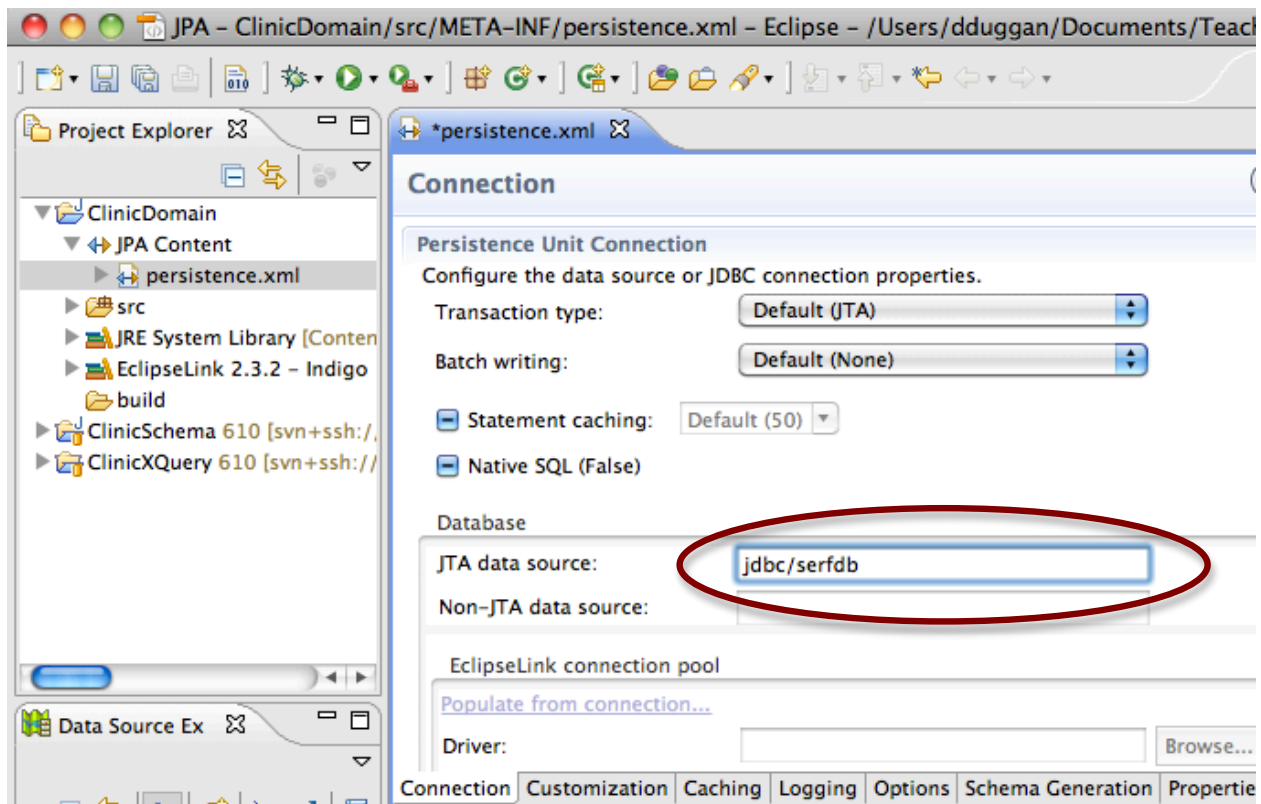
In the JPA perspective, you are able to view your database connections (lower left). The persistence unit in the JPA project provides information about how data is persisted to the database.



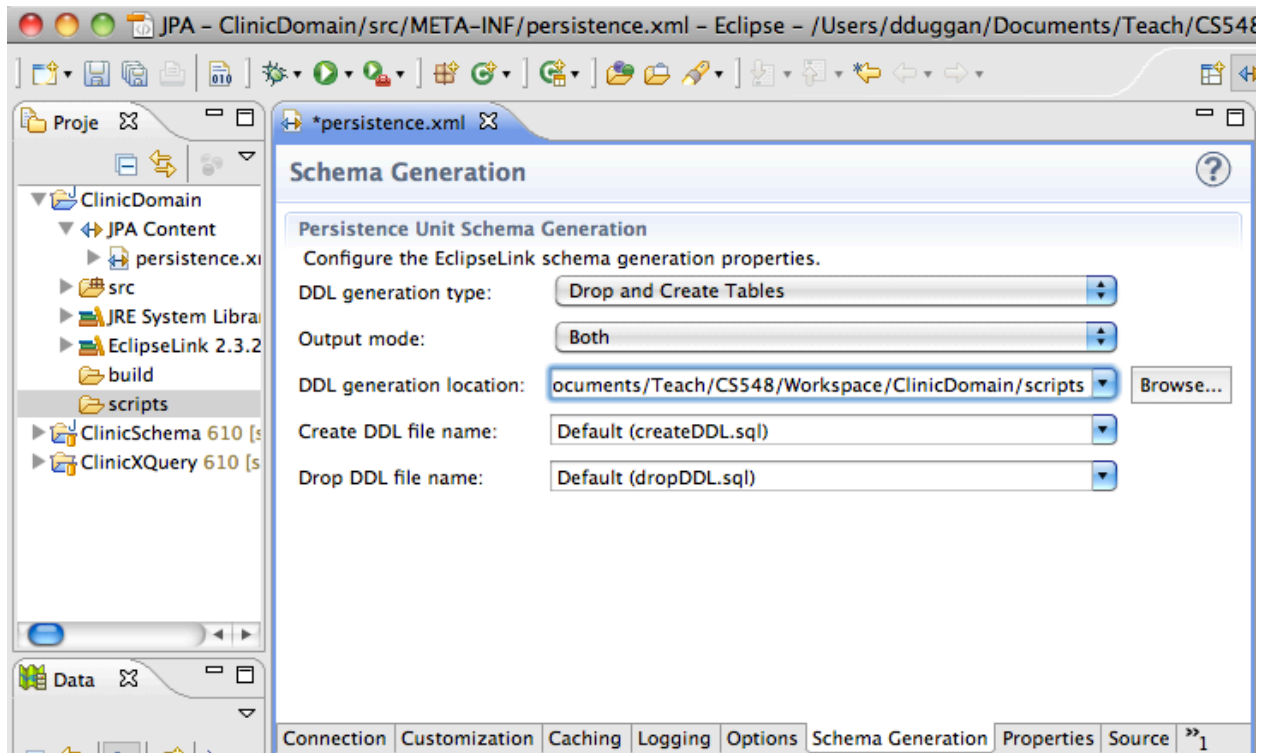
You have various windows for editing properties of the persistence unit. Besides providing a description of the purpose for the persistence unit, you may specify the persistence provider, i.e., the JPA implementation that you are using. This is unnecessary in this case, since we are using EclipseLink, the reference implementation for JPA that is already installed in Glassfish (and you have installed it in your IDE):



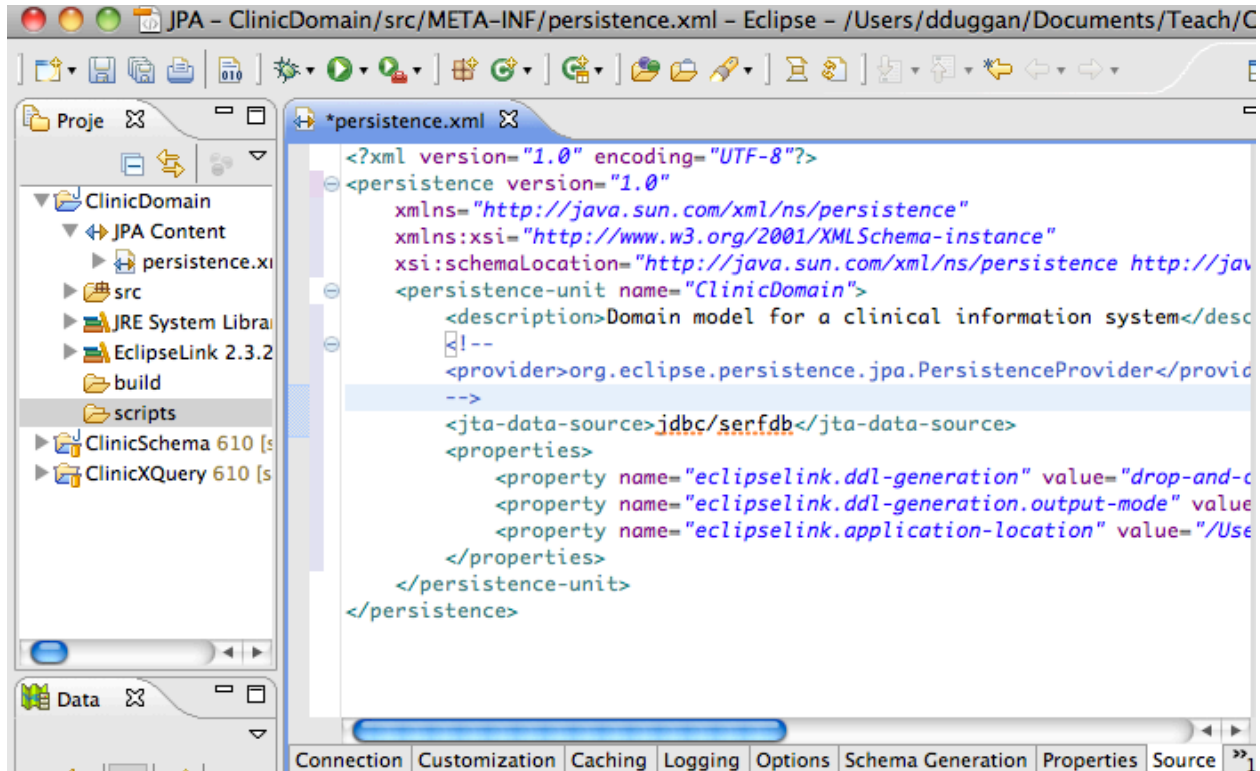
You can set properties of the database connection. In particular, the database transaction type is either JTA (Java Transaction API), which means transactions will be managed by an application server such as Glassfish, or Resource Local (the application will manage database transactions programmatically). The default is JTA. You should also specify the data source, i.e., the JNDI name for looking up the connection in the application. Use the same JNDI name as you specified when setting up a JNDI name for the JDBC resource in the application server:



We'll also configure the generation of database schemas from entity classes. Eclipse will automatically generate SQL scripts for deleting and recreating database tables, and both send them to the database for execution, and save them locally in the location I have specified. I have created a "scripts" folder in the project for this purpose:

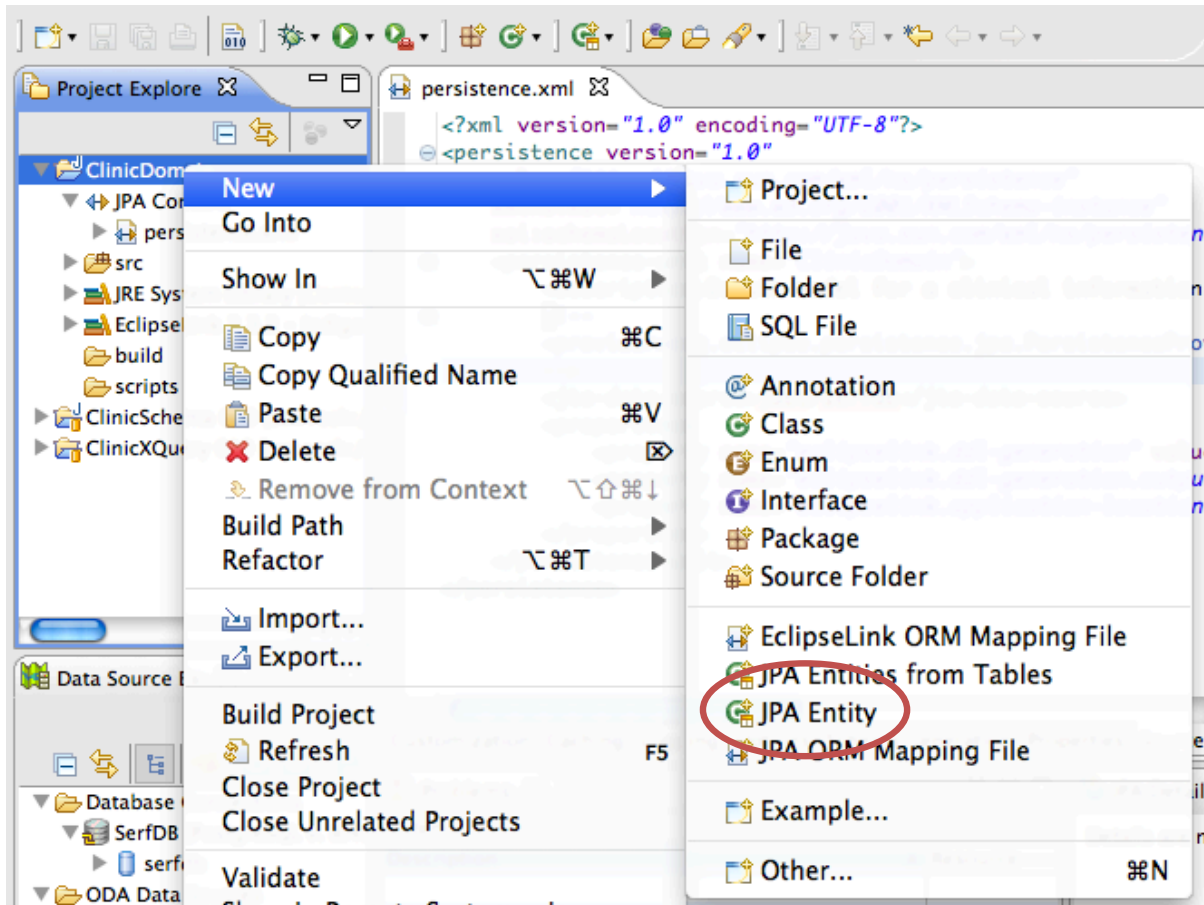


Here is what the persistent unit descriptor looks like after these changes. The provider specification is commented out since its specification is not necessary, given that we are deploying in Glassfish. We could also add other properties of the persistence unit, such as the connection URL (property name: `javax.persistence.jdbc.url`), database user (property name: `javax.persistence.jdbc.user`) and database password (property name: `javax.persistence.jdbc.password`). Instead we have defined these properties separately in the IDE (Eclipse) and the application server (Glassfish):

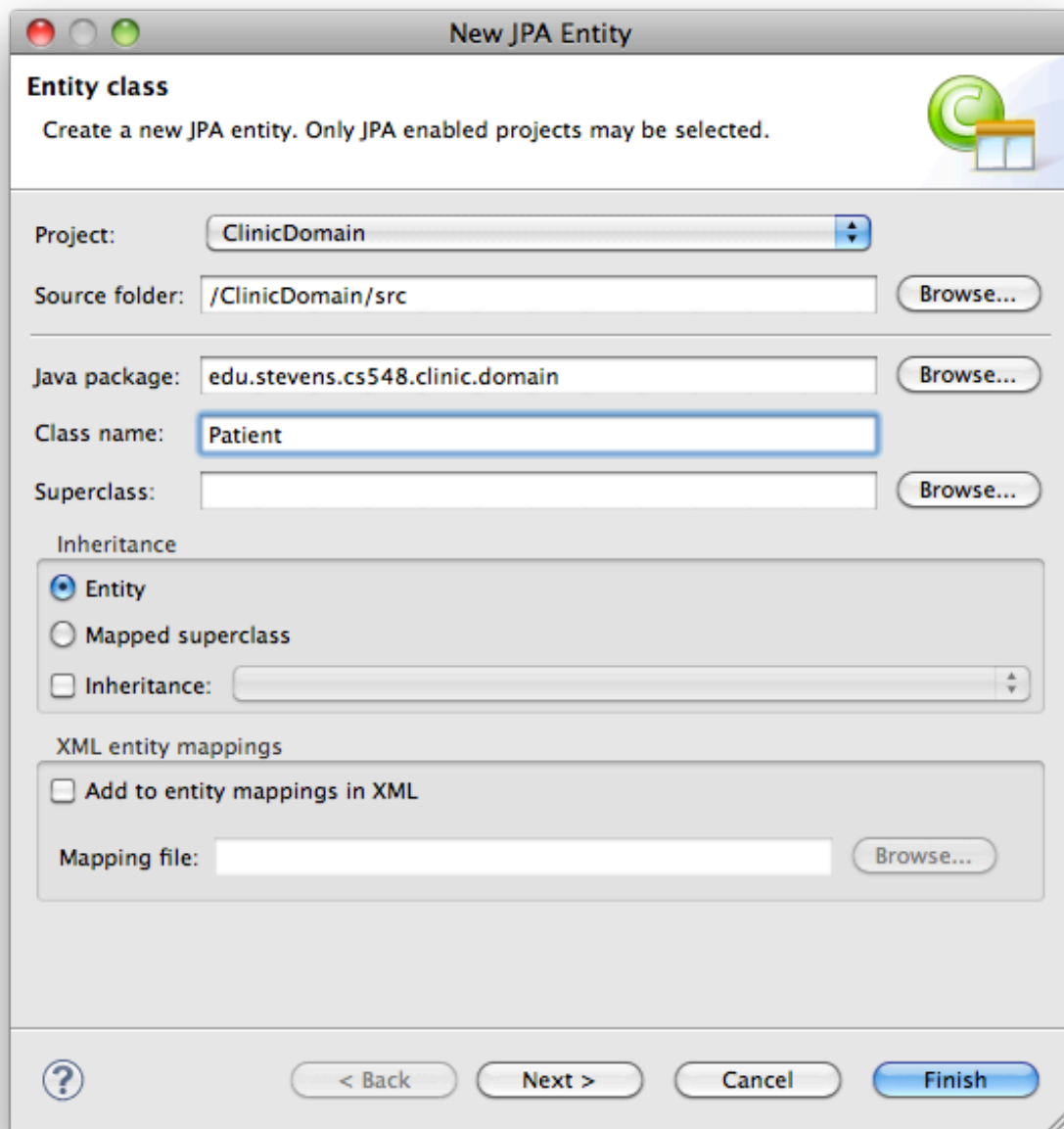


Step 4: Creating Entity Classes

In the JPA perspective, you have the option of creating new entity classes from scratch, generating entity classes from existing database schemas, or adapting existing Java classes to be entity classes. We will generate database tables from entity classes, so you will want to create new entity classes for the domain entities:



This is an example of the dialogue with the wizard:



New JPA Entity

Entity class
Create a new JPA entity. Only JPA enabled projects may be selected.

Project: ClinicDomain

Source folder: /ClinicDomain/src [Browse...](#)

Java package: edu.stevens.cs548.clinic.domain [Browse...](#)

Class name: Patient

Superclass: [Browse...](#)

Inheritance

☒ Entity

☐ Mapped superclass

☐ Inheritance: [Browse...](#)

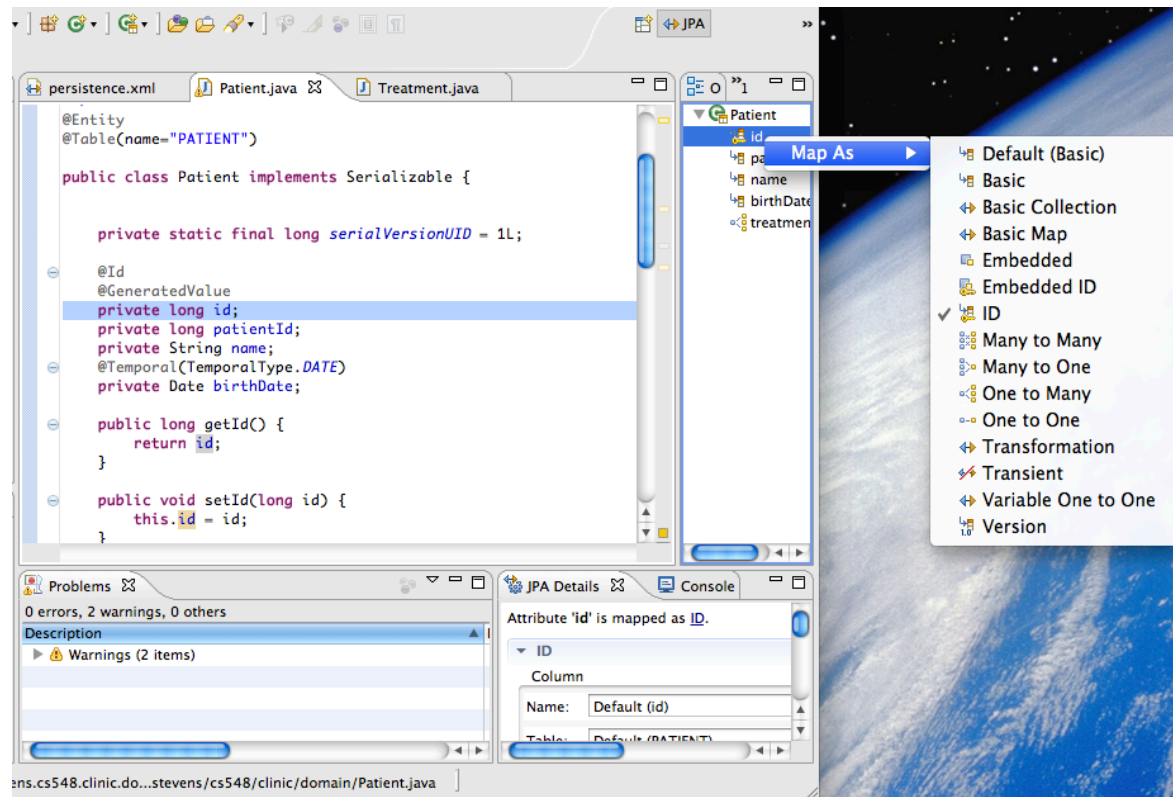
XML entity mappings

☐ Add to entity mappings in XML

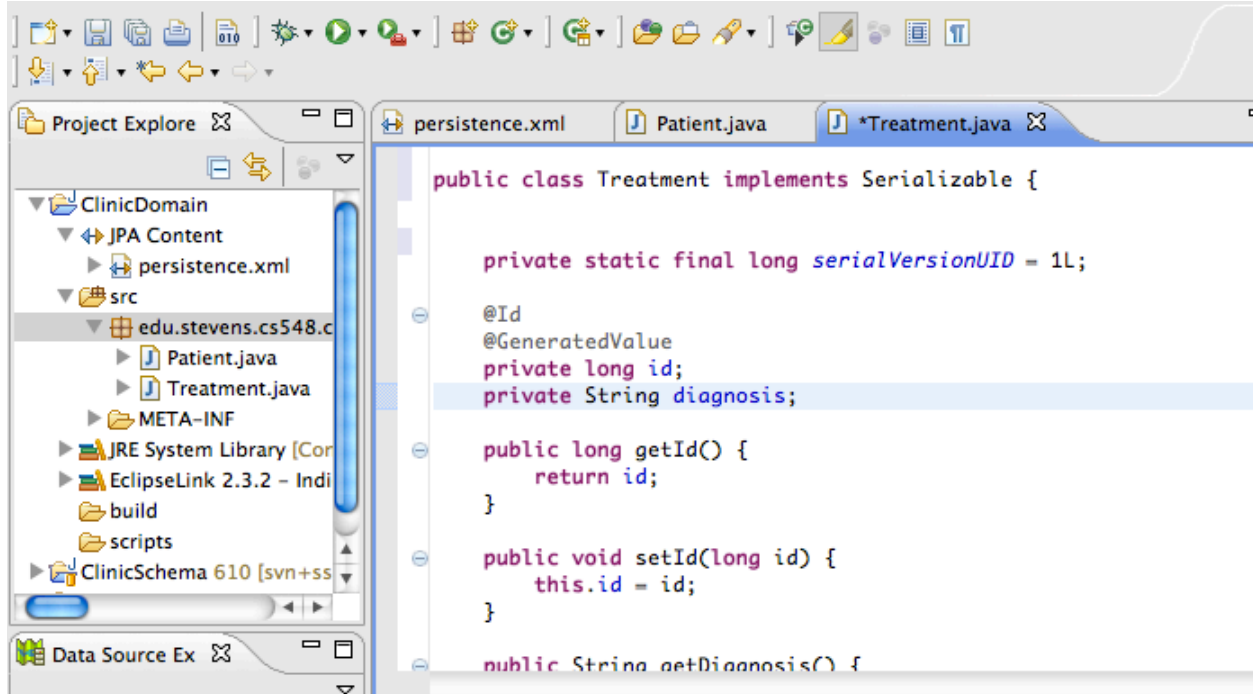
Mapping file: [Browse...](#)

[?](#) [< Back](#) [Next >](#) [Cancel](#) [Finish](#)

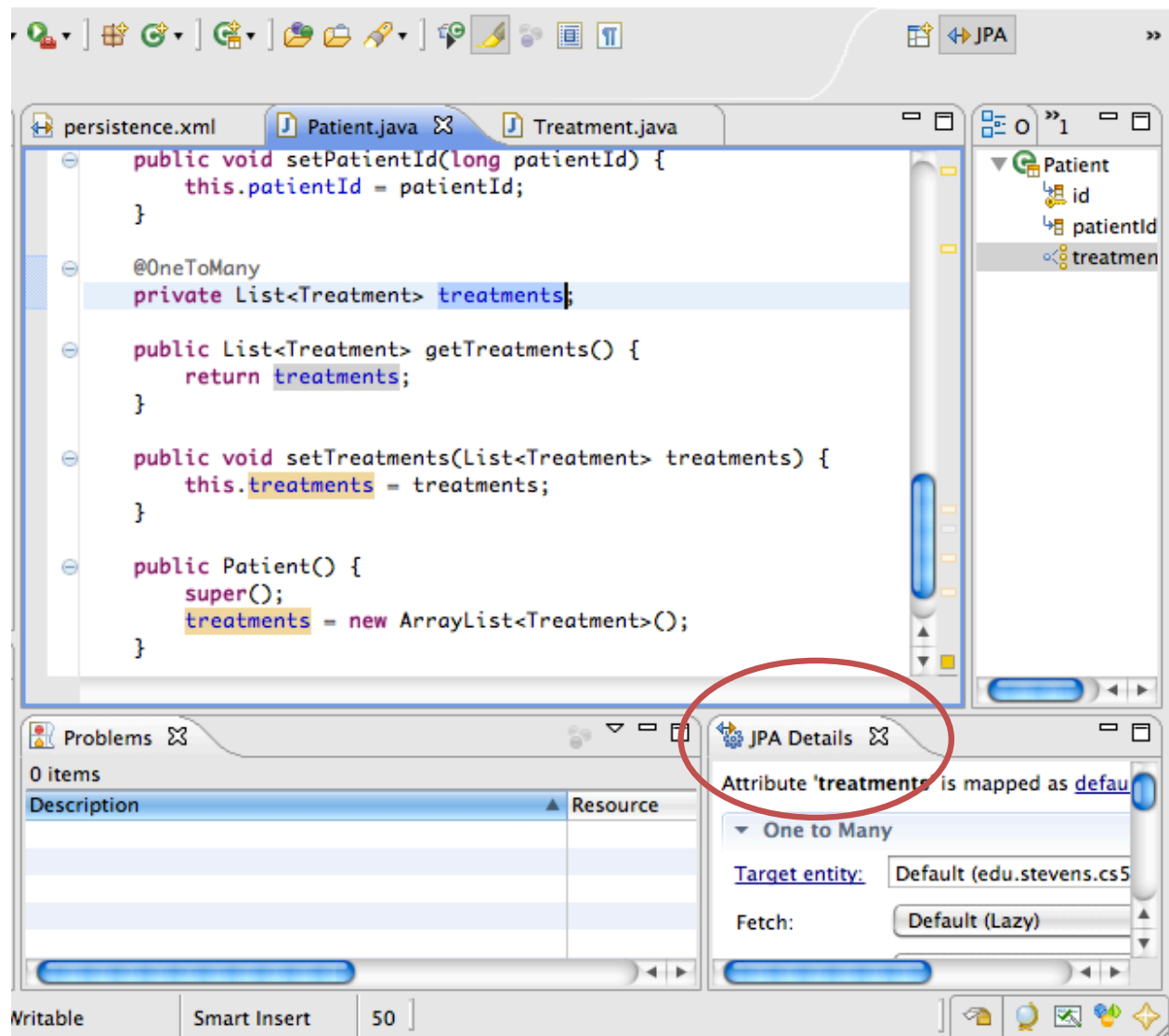
The JPA Perspective allows me to define fields as primary keys, and to annotate fields that represent one-to-many or many-to-one relationships:



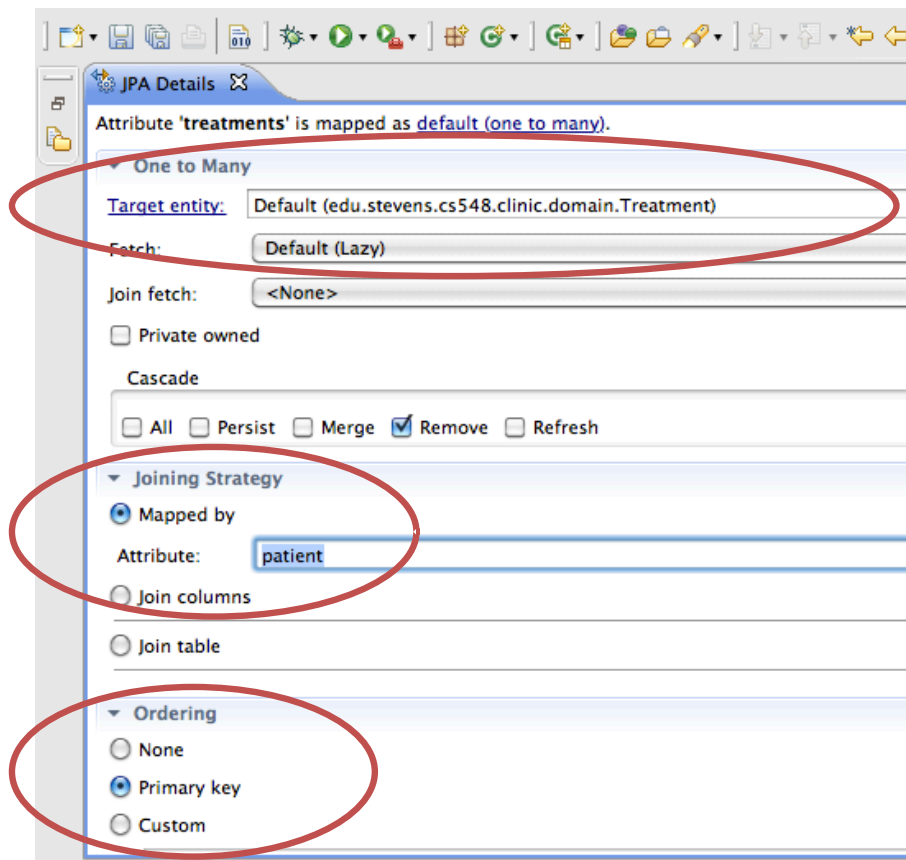
I also define a class for treatment entities, that will be related to patients:



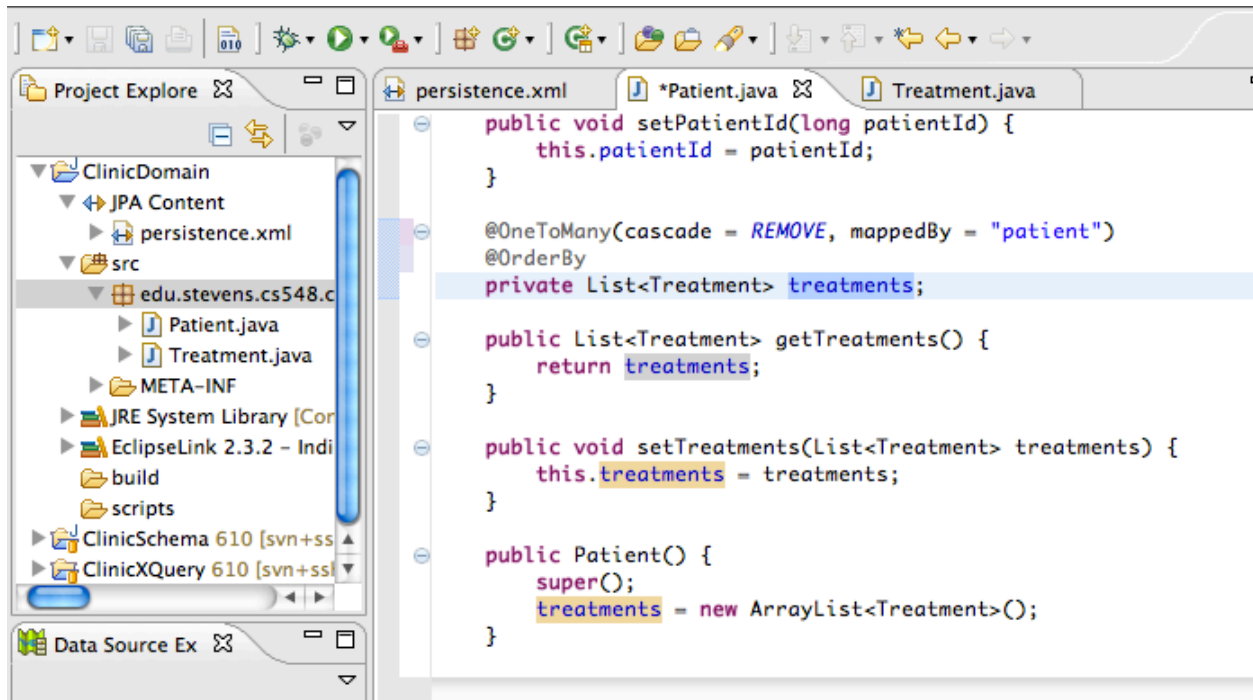
Add the @OneToMany attribute to the “treatments” field in the Patient class. When this field is selected, the JPA Details tab provides more information about this field:



Click “Maximize” to fill up the editor pane with JPA details about this field. The target entity (the entity that “owns” the relationship) is identified as the Treatment class. The attribute in the target entity class that maps this relationship is the “patient” field. Various operations on a patient entity may “cascade” to the treatment entities to which they are related. In this case, the “remove” operation cascades, so if a patient entity is deleted, then the related treatment entities are also deleted. Finally, the fact that the collection of treatments in the patient entity is a list requires that an order be specified on treatments. In this case, treatments are ordered by primary key:



Restore the JPA details pane to return to the normal JPA perspective. The customization of the relationship is reflected in the annotations:



In the Treatment class, annotate the patient field with @ManyToOne. Select this field to focus the “JPA Details” pane on this field.

The screenshot shows an IDE with the following components:

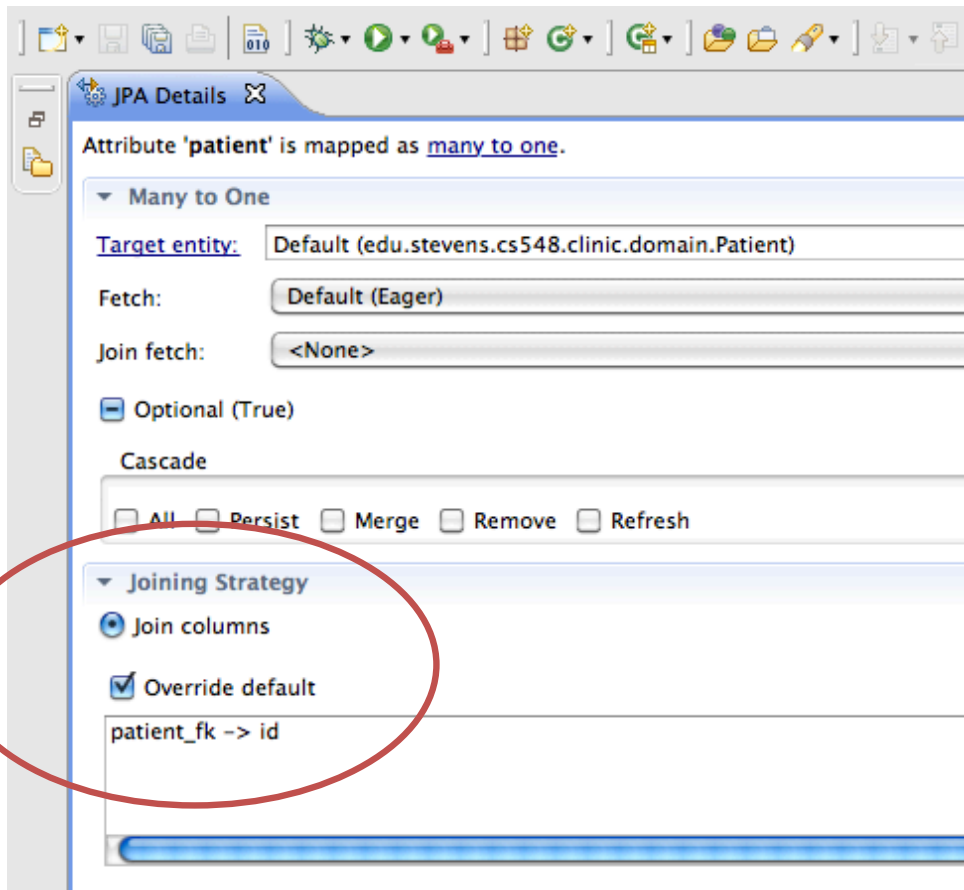
- Top Bar:** Persistence.xml, Patient.java, Treatment.java (selected), JPA.
- Editor:** Treatment.java code with the following content:

```
public void setId(long id) {  
    this.id = id;  
}  
  
@ManyToOne  
private Patient patient;  
  
public Patient getPatient() {  
    return patient;  
}  
  
public void setPatient(Patient patient) {  
    this.patient = patient;  
}  
  
public Treatment() {  
    super();  
}
```

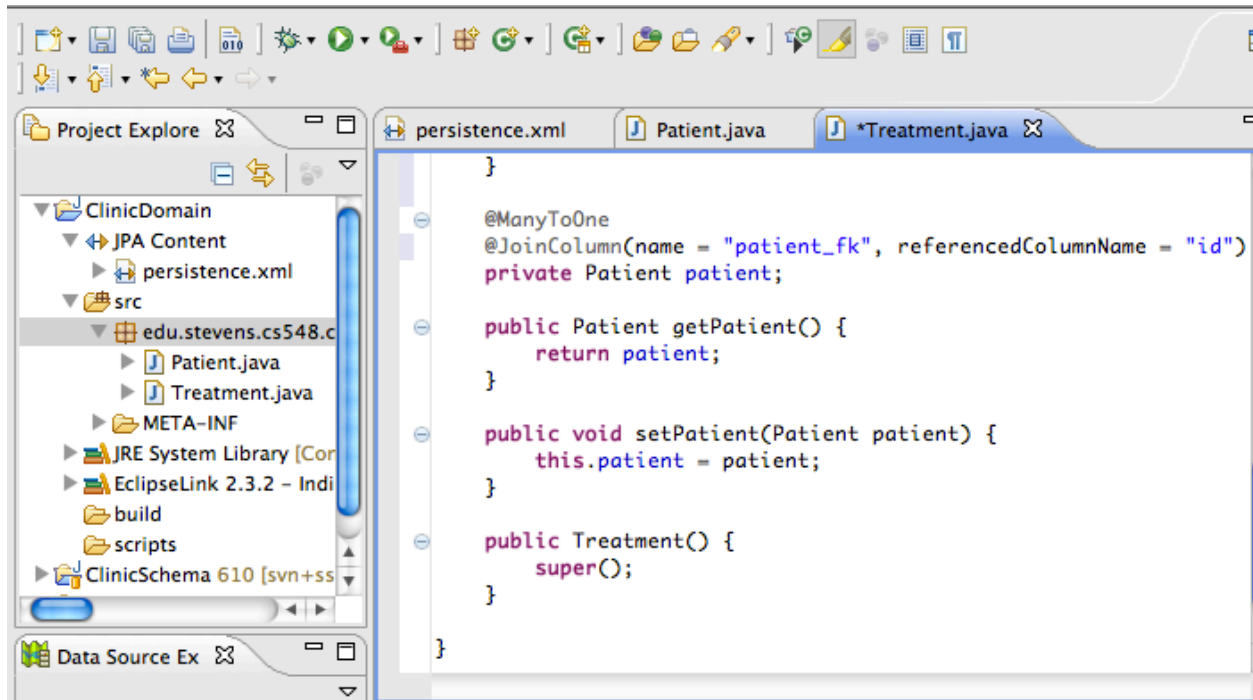
The line `@ManyToOne` is highlighted with a red oval.
- Right Panel:** A tree view showing the `Treatment` entity with fields `id` and `patient`. The `patient` field is selected.
- Bottom Panel:**
 - Problems:** 0 items.
 - JPA Details:** Attribute 'patient' is mapped as many to one.
 - Many to One:**
 - Target entity:** Default (edu.stevens.cs5)
 - Fetch:** Default (Eager)

At the bottom of the IDE, there is a status bar with the text "Writable", "Smart Insert", and "30".

To specify how the relationship between patients and treatments is mapped, you can again maximize the “JPA Details” panel. I have selected the “Join Columns” joining strategy, which means that a treatment entity has a foreign key reference to the corresponding patient entity. I have overridden the default name for this column in the treatment table to be “patient_fk”.

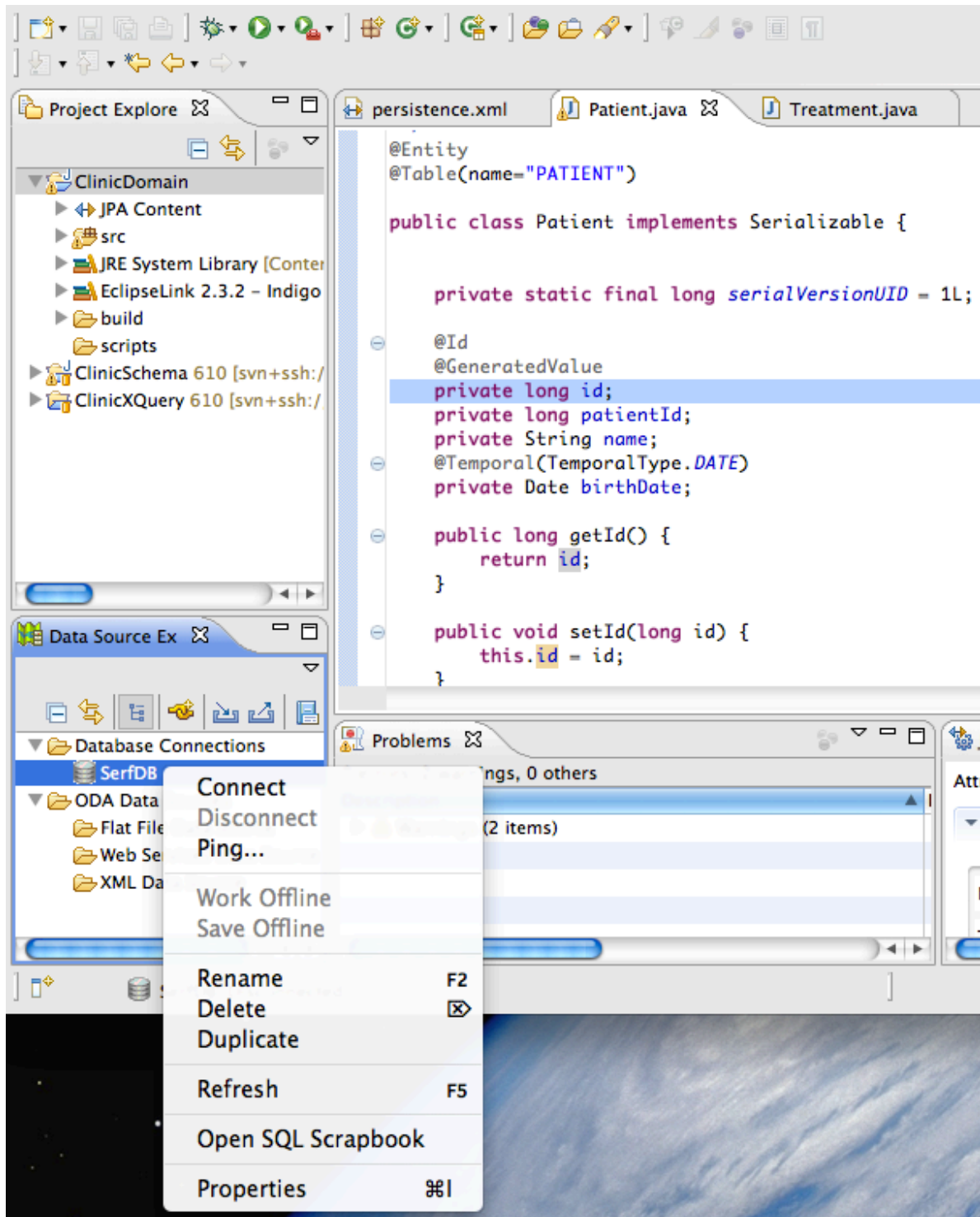


This results in the `@JoinColumn` annotation on the `patient` field in the `treatment` class:

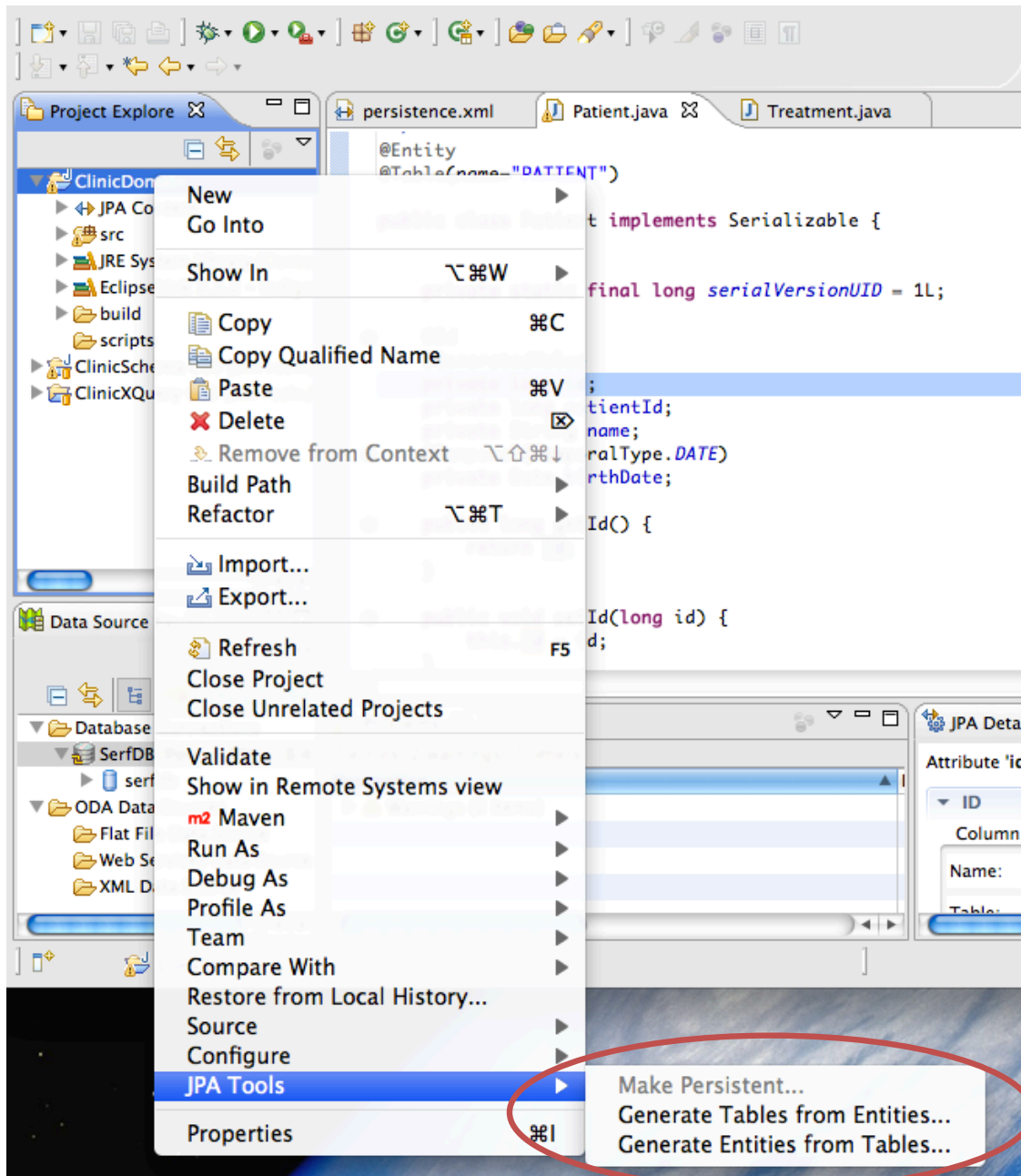


Step 5: Connecting to the Database

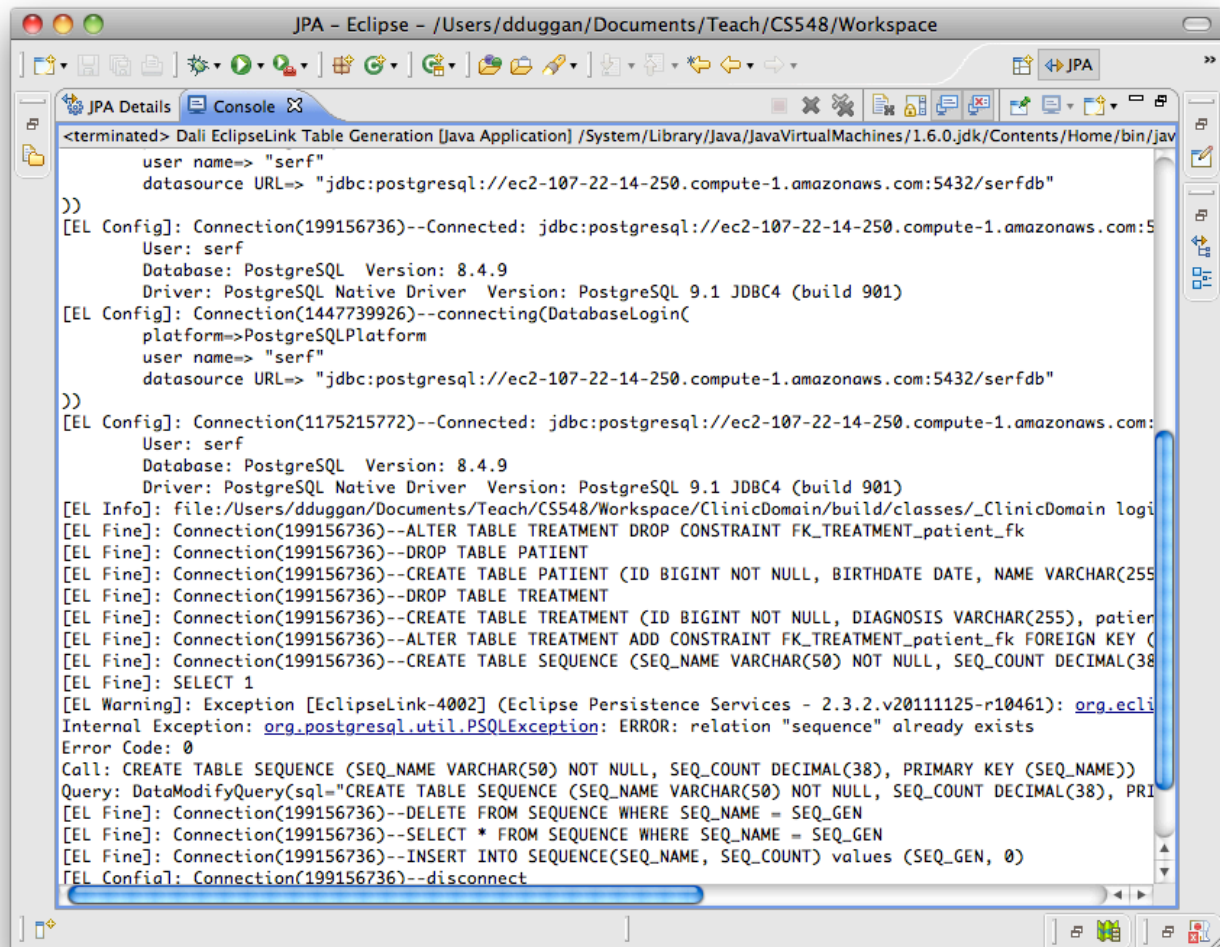
Back in the JPA Perspective, right-click on the database for which you have defined a JDBC connection, in the Data Source Explorer subwindow, and choose Connect to open a connection to the database:



With a connection to the database open, right-click on the project and select the JPA Tools submenu. This offers the option of generating database tables from JPA entities:



When you run this, you can see the result of generating tables in the console window, in the lower right. Maximize this pane to see what happened:

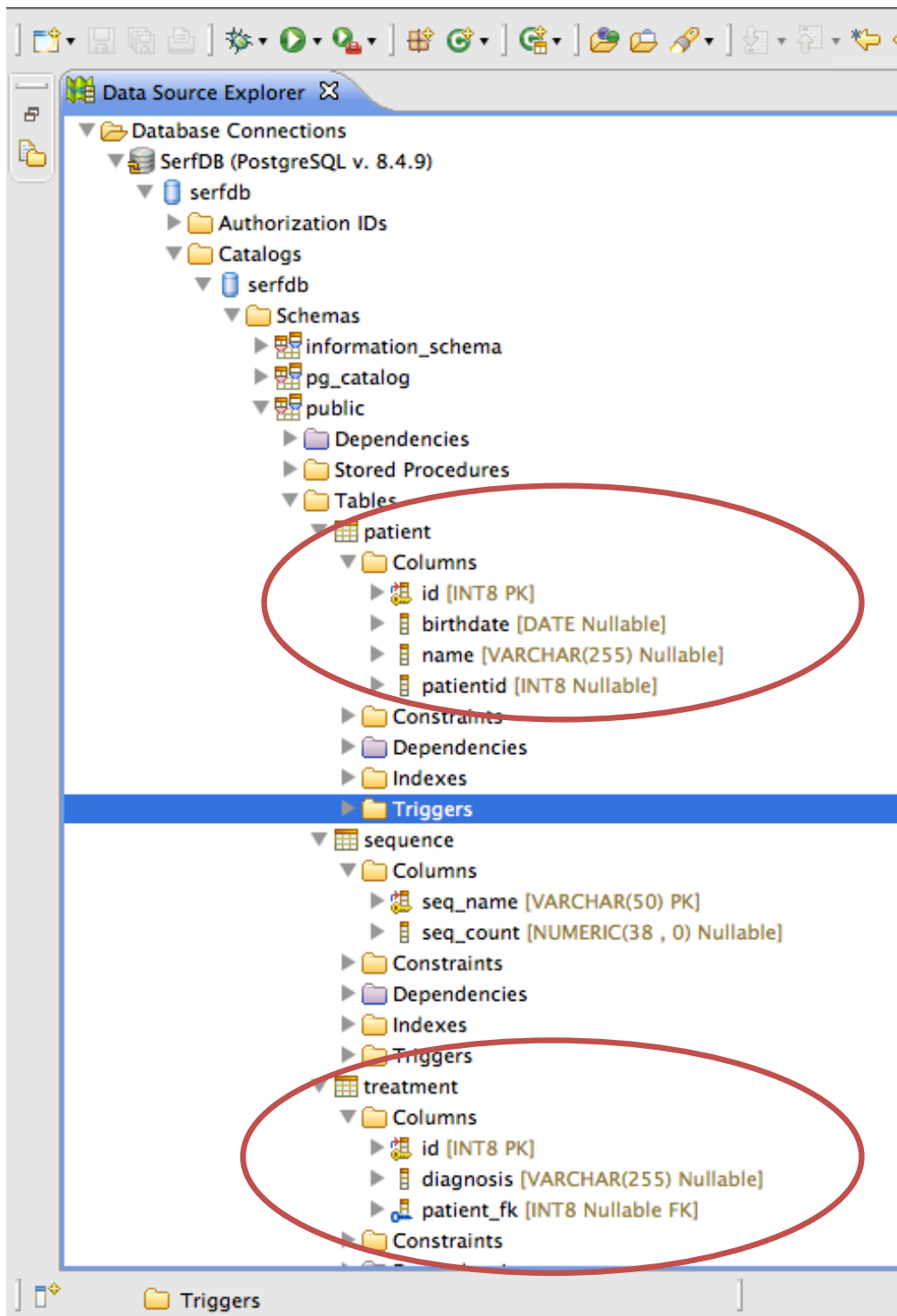


The screenshot shows the Eclipse IDE with the 'Console' tab selected. The console output displays the execution of a JPA application that generates database tables. The logs include connection details for a PostgreSQL database, table creation and alteration commands, and a warning about a duplicate sequence name.

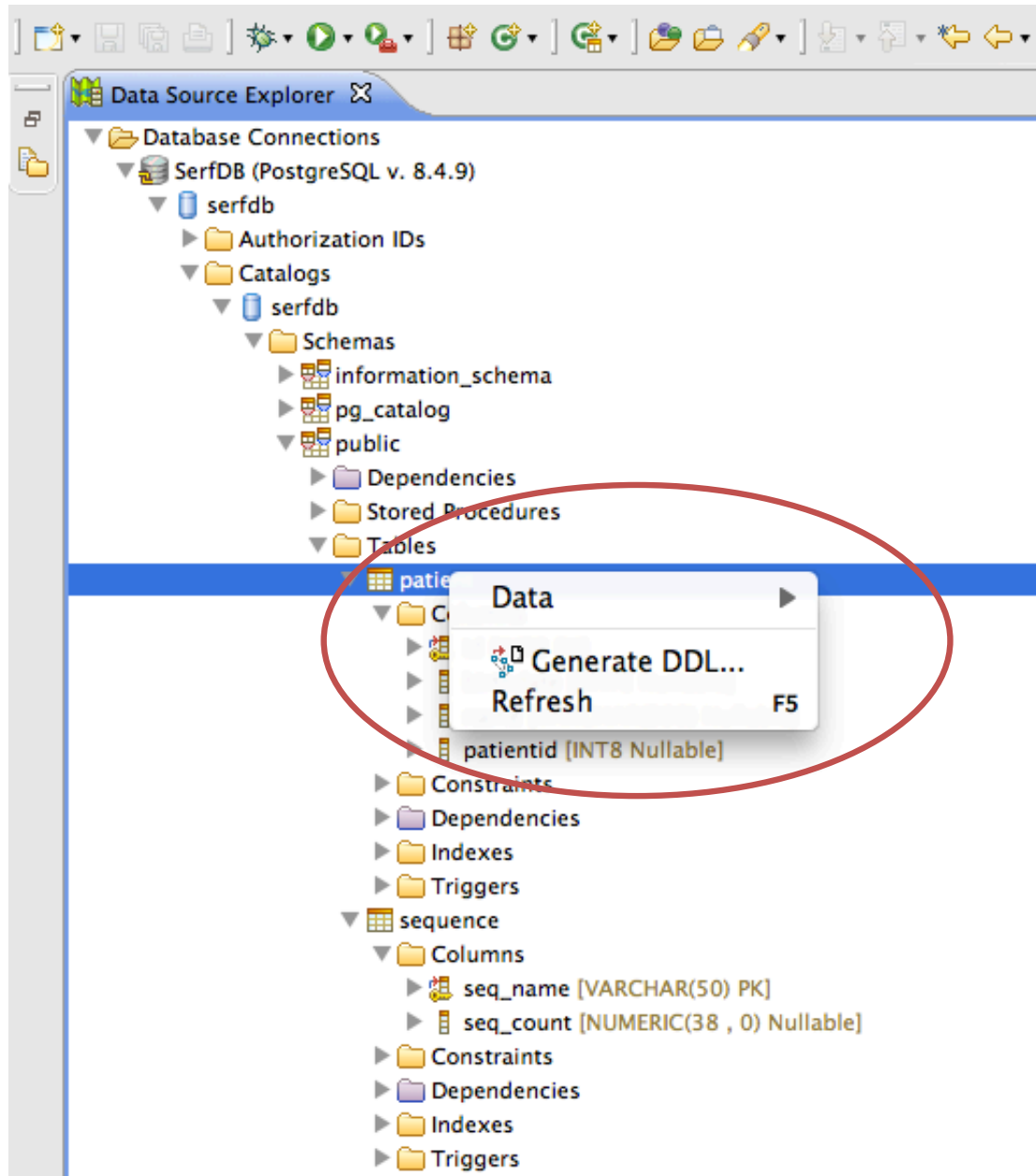
```
<terminated> Dali EclipseLink Table Generation [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/jav
    user name=> "serf"
    datasource URL=> "jdbc:postgresql://ec2-107-22-14-250.compute-1.amazonaws.com:5432/serfdb"
))
[EL Config]: Connection(199156736)--Connected: jdbc:postgresql://ec2-107-22-14-250.compute-1.amazonaws.com:5
    User: serf
    Database: PostgreSQL Version: 8.4.9
    Driver: PostgreSQL Native Driver Version: PostgreSQL 9.1 JDBC4 (build 901)
[EL Config]: Connection(1447739926)--connecting(DatabaseLogin(
    platform=>PostgreSQLPlatform
    user name=> "serf"
    datasource URL=> "jdbc:postgresql://ec2-107-22-14-250.compute-1.amazonaws.com:5432/serfdb"
))
[EL Config]: Connection(1175215772)--Connected: jdbc:postgresql://ec2-107-22-14-250.compute-1.amazonaws.com:
    User: serf
    Database: PostgreSQL Version: 8.4.9
    Driver: PostgreSQL Native Driver Version: PostgreSQL 9.1 JDBC4 (build 901)
[EL Info]: file:/Users/dduggan/Documents/Teach/CS548/Workspace/ClinicDomain/build/classes/_ClinicDomain logi
[EL Fine]: Connection(199156736)--ALTER TABLE TREATMENT DROP CONSTRAINT FK_TREATMENT_patient_fk
[EL Fine]: Connection(199156736)--DROP TABLE PATIENT
[EL Fine]: Connection(199156736)--CREATE TABLE PATIENT (ID BIGINT NOT NULL, BIRTHDATE DATE, NAME VARCHAR(255
[EL Fine]: Connection(199156736)--DROP TABLE TREATMENT
[EL Fine]: Connection(199156736)--CREATE TABLE TREATMENT (ID BIGINT NOT NULL, DIAGNOSIS VARCHAR(255), patier
[EL Fine]: Connection(199156736)--ALTER TABLE TREATMENT ADD CONSTRAINT FK_TREATMENT_patient_fk FOREIGN KEY (
[EL Fine]: Connection(199156736)--CREATE TABLE SEQUENCE (SEQ_NAME VARCHAR(50) NOT NULL, SEQ_COUNT DECIMAL(38
[EL Fine]: SELECT 1
[EL Warning]: Exception [EclipseLink-4002] (Eclipse Persistence Services - 2.3.2.v20111125-r10461): org.ecli
Internal Exception: org.postgresql.util.PSQLException: ERROR: relation "sequence" already exists
Error Code: 0
Call: CREATE TABLE SEQUENCE (SEQ_NAME VARCHAR(50) NOT NULL, SEQ_COUNT DECIMAL(38), PRIMARY KEY (SEQ_NAME))
Query: DataModifyQuery(sql="CREATE TABLE SEQUENCE (SEQ_NAME VARCHAR(50) NOT NULL, SEQ_COUNT DECIMAL(38), PRI
[EL Fine]: Connection(199156736)--DELETE FROM SEQUENCE WHERE SEQ_NAME = SEQ_GEN
[EL Fine]: Connection(199156736)--SELECT * FROM SEQUENCE WHERE SEQ_NAME = SEQ_GEN
[EL Fine]: Connection(199156736)--INSERT INTO SEQUENCE(SEQ_NAME, SEQ_COUNT) values (SEQ_GEN, 0)
[EL Config]: Connection(199156736)--disconnect
```

You may need to clean the project (Project | Clean...) to get rid of some confusing diagnostics.

Restore the console to normal size. Back in the JPA Perspective, maximize the Data Source Explorer pane to browse your database. This shows an example of browsing the schemas for the database tables that have been generated from the JPA entities:




Right-clicking on a table in this perspective provides several options, and one is to generate the Data Definition Language (DDL) for creating a table:



Generate DDL

Save and Run DDL



Specify a path to save the generated DDL script. You can run the DDL script by providing your database connection information.

Folder:

/ClinicDomain/scripts

Browse...

File name:

Patient.sql

Preview DDL

--<ScriptOptions statementTerminator=";" />

CREATE TABLE patient (
 id INT8 NOT NULL,
 birthdate DATE,
 name VARCHAR(255),
 patientid INT8
);

CREATE UNIQUE INDEX patient_pkey ON patient (id ASC);

ALTER TABLE patient ADD CONSTRAINT patient_pkey PRIMARY KEY (id);


Statement terminator:

;

Apply

☐ Run DDL on server

☐ Open DDL file for editing



< Back

Next >

Cancel

Finish