Dali Java Persistence Tools
User Guide
Release 1.0.0 for Eclipse

May 2007
## Contents

### 1 Getting started

1.1 Requirements and installation .................................................................................................. 1-1  
1.2 Dali quick start ............................................................................................................................ 1-1  
1.2.1 Creating a new project ........................................................................................................ 1-2  
1.2.2 Creating a Java persistent entity ........................................................................................ 1-3  
1.2.3 Mapping an entity ............................................................................................................... 1-6  
1.3 Dali basic tutorial ........................................................................................................................ 1-7  
1.3.1 Generate the tutorial database schema............................................................................ 1-8  
1.3.1.1 Create a database connection...................................................................................... 1-8  
1.3.2 Create a JPA project........................................................................................................... 1-9  
1.3.3 Create persistent Java entities ........................................................................................ 1-10  
1.3.3.1 Add fields to the entities........................................................................................... 1-11  
1.3.4 Create OR mappings ........................................................................................................ 1-12  
1.3.4.1 Create ID mappings .................................................................................................. 1-12  
1.3.4.2 Create basic mappings ............................................................................................... 1-14  
1.3.4.3 Create one-to-one mappings .................................................................................... 1-16  
1.3.4.4 Create one-to-many mappings ................................................................................ 1-17  
1.3.4.5 Create many-to-one mappings ................................................................................ 1-18  
1.3.4.6 Create version mappings .......................................................................................... 1-19

### 2 Concepts

2.1 Understanding Java persistence .............................................................................................. 2-1  
2.2 Understanding OR mappings ................................................................................................... 2-1  
2.3 Understanding EJB 3.0 Java Persistence API .......................................................................... 2-2  
2.3.1 The persistence.xml file.................................................................................................... 2-2  
2.3.2 The orm.xml file ............................................................................................................... 2-2

### 3 Tasks

3.1 Creating a new JPA project........................................................................................................ 3-1  
3.2 Managing the persistence.xml file........................................................................................ 3-3  
3.2.1 Working with persistence.xml file .................................................................................. 3-4  
3.2.2 Synchronizing classes........................................................................................................ 3-4  
3.3 Managing the orm.xml file ..................................................................................................... 3-5  
3.3.1 Working with orm.xml file ............................................................................................... 3-5
3.4 Adding persistence to a class ................................................................. 3-6
3.4.1 Entity .............................................................................................. 3-6
3.4.2 Embeddable .................................................................................... 3-7
3.4.3 Mapped superclass .......................................................................... 3-8
3.5 Specifying additional tables ................................................................. 3-8
3.6 Specifying entity inheritance ............................................................... 3-9
3.7 Mapping an entity .................................................................................. 3-11
3.7.1 Basic mapping .................................................................................. 3-11
3.7.2 Embedded mapping ......................................................................... 3-12
3.7.3 Embedded ID mapping .................................................................... 3-13
3.7.4 ID mapping ...................................................................................... 3-13
3.7.5 Many-to-many mapping ................................................................. 3-15
3.7.6 Many-to-one mapping ..................................................................... 3-16
3.7.7 One-to-many mapping ................................................................. 3-17
3.7.8 One-to-one mapping ....................................................................... 3-18
3.7.9 Transient mapping .......................................................................... 3-19
3.7.10 Version mapping ........................................................................... 3-19
3.8 Generating entities from tables ........................................................... 3-20
3.9 Validating mappings and reporting problems ..................................... 3-21
3.9.1 Error messages ................................................................................ 3-21
3.9.2 Warning messages .......................................................................... 3-23
3.10 Modifying persistent project properties ............................................. 3-23

4 Reference

4.1 Wizards.................................................................................................. 4-1
4.1.1 Create New JPA Project wizard ...................................................... 4-1
4.1.1.1 New JPA Project page .............................................................. 4-1
4.1.1.2 JPA Facet page ...................................................................... 4-2
4.2 Property pages .................................................................................... 4-2
4.2.1 JPA Details view (for entities) ........................................................ 4-3
4.2.1.1 General information .............................................................. 4-3
4.2.1.2 Secondary table information .................................................. 4-3
4.2.1.3 Inheritance information ......................................................... 4-3
4.2.2 JPA Details view (for attributes) .................................................. 4-4
4.2.2.1 General information .............................................................. 4-4
4.2.2.2 Join Table Information ........................................................... 4-6
4.2.2.3 Join Columns Information ....................................................... 4-7
4.2.2.4 Primary Key Generation information ..................................... 4-7
4.2.3 JPA Details view (for orm.xml) ...................................................... 4-8
4.2.3.1 General information .............................................................. 4-8
4.2.3.2 Persistence Unit information .................................................. 4-8
4.2.4 JPA Structure view ........................................................................ 4-9
4.3 Preferences ......................................................................................... 4-9
4.3.1 Project Properties page – JPA Options ......................................... 4-9
4.4 Dialogs .................................................................................................. 4-10
4.4.1 Generate Entities from Tables dialog ........................................... 4-10
4.4.2 Edit Join Columns Dialog ............................................................. 4-10
4.5  JPA Development perspective ................................................................. 4-11
4.6  Icons and buttons ................................................................................... 4-11
4.6.1 Icons ..................................................................................................... 4-11
4.6.2 Buttons .................................................................................................. 4-12
4.7  Dali Developer Documentation ............................................................... 4-13

5  Tips and tricks

6  What’s new
6.1  Generate Persistent Entities from Tables wizard ...................................... 6-1
6.2  Create and Manage the persistence.xml file ........................................... 6-1
6.3  Create and Manage the orm.xml file ....................................................... 6-2

7  Legal
7.1  About this content ................................................................................... 7-1

Index
This section provides information on getting started with the Java Persistence Tools.

- Requirements and installation
- Dali quick start
- Dali basic tutorial

For additional information, please visit the Dali home page at:

1.1 Requirements and installation

Before installing Dali, ensure that your environment meets the following minimum requirements:

- Eclipse 3.3 (http://www.eclipse.org/downloads)
- Java Runtime Environment (JRE) 1.5 (http://java.com)
- Eclipse Web Tools Platform (WTP) 2.0 (http://www.eclipse.org/webtools)
- Java Persistence API (JPA) for Java EE 5. The reference implementation can be obtained from:
  https://glassfish.dev.java.net/downloads/persistence/JavaPersistence.html

Refer to http://www.eclipse.org/webtools/dali/gettingstarted_main.html for additional installation information.

Review the Dali quick start and Dali basic tutorial to build your first Dali project.

1.2 Dali quick start

This section includes information to help you quickly start using Dali to create relational mappings between Java persistent entities and database tables.

- Creating a new project
- Creating a Java persistent entity
- Mapping an entity

The Dali basic tutorial contains detailed procedures for building your first Dali project.
1.2.1 Creating a new project

This quick start shows how to create a new JPA project.

1. Select **File > New > Project**. The New Project dialog appears.

2. On the New Project dialog, select **JPA > JPA Project** and click **Next**. The Create a Java Project dialog appears.

![New JPA Project](image)

3. On the Create a JPA Project dialog, enter a **Project name** (such as **QuickStart**).

4. Select your **Target Runtime** (such as Apache Tomcat) and click **Next**. The Project Facets dialog appears.

5. Verify that the **Java Persistence 1.0** facet is selected and click **Next**. The JPA Facet dialog appears.

   **Note:** You must configure your project to use Java version 5.0 (or higher). See "Requirements and installation" on page 1-1 for more information.

6. On the JPA Facet dialog, select your vendor-specific JPA platform, database connection (or create a new connection), JPA implementation library (such as TopLink Essentials), define how Dali should manage persistent classes, and click **Finish**.
Eclipse adds the project to the workbench and opens the JPA perspective.

Now that you have created a project with persistence, you can continue with Creating a Java persistent entity.

1.2.2 Creating a Java persistent entity

This quick start shows how to create a new persistent Java entity. We will create an entity to associate with a database table. You will also need to add the ADDRESS table to your database.

1. Right-click the project in the Package Explorer and select New > Class. The New Java Class page appears.

2. On the Java Class page, enter a package name (such as quickstart.demo.model), class name (such as Address), and click Finish.

3. Right-click the Address.java file in the Package Explorer and select Open.
4. Select the Address entity in the JPA Structure view.

5. In the JPA Details view, in the Map As field, select Entity. In the Table field, select the ADDRESS database table.

Eclipse creates the persistent entity and adds the @Entity annotation to the class.
Eclipse also displays the **Address** entity in the JPA Structure view:

**Figure 1–8  Address Entity**

After creating the entity, you must associate it with a database table.

1. Select the **Address** class in the Explorer view.
2. In the Persistence Properties view, notice that Dali has automatically associated the ADDRESS database table with the entity because they are named identically.

---

**Note:** Depending on your database connection type, you may need to specify the **Schema**.

**Figure 1–9  JPA Details View for Address Entity**

After associating the entity with the database table, you must update the persistence.xml file to include this JPA entity.

Right-click the persistence.xml file in the Package Explorer and select **JPA Tools > Synchronize Classes**. Dali adds the following to the persistence.xml file:

```xml
<class>quickstart.demo.model.Address</class>
```
Now that you have created a persistent entity, you can continue with Mapping an entity to map the entity’s fields to columns on the database table.

1.2.3 Mapping an entity

This quick start shows how to map fields in a Java persistent entity. Before beginning, add the following fields to the Address class:

```java
private Long id;
private String city;
private String country;
private String stateOrProvince;
private String postalCode;
private String street;
```

Eclipse updates the Address entity in the JPA Structure view to show its fields:

![Address Entity and Fields](image)

You will also need to add the following columns to the ADDRESS database table:

- `NUMBER(10,0) ADDRESS_ID (primary key)`
- `VARCHAR2(80) PROVINCE`
- `VARCHAR2(80) COUNTRY`
- `VARCHAR2(20) P_CODE`
- `VARCHAR2(80) STREET`
- `VARCHAR2(80) CITY`

Now we are ready to map each field in the Address class to a column in the database table.

1. Select the id field in the JPA Details view.
2. In the JPA Details view:
   - For the Map As field, select Id
   - For the Column field, select ADDRESS_ID.
Getting started

1.3 Dali basic tutorial

In this tutorial, you will use Dali to map the object model of a company’s HR application to track its employees. Figure 1–12 illustrates the object model for the tutorial.

---

**Figure 1–11  JPA Details View for addressId Field**

Eclipse adds the following annotations to the Address entity:

```java
@Id
@Column(name="ADDRESS_ID")
```

3. Map each of the following fields (as Basic mappings) to the appropriate database column:

<table>
<thead>
<tr>
<th>Field</th>
<th>Map As</th>
<th>Database Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>city</td>
<td>Basic</td>
<td>CITY</td>
</tr>
<tr>
<td>country</td>
<td>Basic</td>
<td>COUNTRY</td>
</tr>
<tr>
<td>postalCode</td>
<td>Basic</td>
<td>P_CODE</td>
</tr>
<tr>
<td>provinceOrState</td>
<td>Basic</td>
<td>PROVINCE</td>
</tr>
<tr>
<td>street</td>
<td>Basic</td>
<td>STREET</td>
</tr>
</tbody>
</table>

Notice that Dali will automatically map some fields to the correct database column (such as the city field to the CITY column) if the names are identical.

Refer to the Dali basic tutorial to map a complete object model using basic and relational mappings.
1.3.1 Generate the tutorial database schema

The tutorial application uses three database tables to store each employee’s information: EMPLOYEE, ADDRESS and PHONE. Table 1–1 describes the columns for each table.

You can download SQL scripts to build and populate the database tables with sample data from [http://www.eclipse.org/webtools/dali/docs/dbscripts.zip](http://www.eclipse.org/webtools/dali/docs/dbscripts.zip).

<table>
<thead>
<tr>
<th>Table</th>
<th>Column</th>
<th>Type</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE</td>
<td>EMP_ID</td>
<td>NUMBER(15)</td>
<td>Primary Key</td>
</tr>
<tr>
<td></td>
<td>F_NAME</td>
<td>VARCHAR(40)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>L_NAME</td>
<td>VARCHAR(40)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADDR_ID</td>
<td>NUMBER(15)</td>
<td>Foreign Key, references ADDRESS.ADDRES_ID</td>
</tr>
<tr>
<td></td>
<td>VERSION</td>
<td>NUMBER(15)</td>
<td></td>
</tr>
<tr>
<td>ADDRESS</td>
<td>ADDRESS_ID</td>
<td>NUMBER(15)</td>
<td>Primary Key</td>
</tr>
<tr>
<td></td>
<td>PROVINCE</td>
<td>VARCHAR(80)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COUNTRY</td>
<td>VARCHAR(80)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STREET</td>
<td>VARCHAR(80)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P_CODE</td>
<td>VARCHAR(20)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CITY</td>
<td>VARCHAR(80)</td>
<td></td>
</tr>
<tr>
<td>PHONE</td>
<td>EMP_ID</td>
<td>NUMBER(15)</td>
<td>Foreign Key, reference to EMPLOYEE.EMP_ID</td>
</tr>
<tr>
<td></td>
<td>AREA_CODE</td>
<td>VARCHAR(3)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>P_NUMBER</td>
<td>VARCHAR(7)</td>
<td>Primary key</td>
</tr>
<tr>
<td></td>
<td>TYPE</td>
<td>VARCHAR(15)</td>
<td></td>
</tr>
</tbody>
</table>

1.3.1.1 Create a database connection

After creating the database you will need to create a database connection to use with the tutorial application. An active database connection is required to complete tutorial application.
Use the New Connection wizard to create a database connection.

*Figure 1–13  Database Explorer*

Now you’re ready to **Create a JPA project**.

### 1.3.2 Create a JPA project

In order to begin, you must create a new Java project.

1. Select **File > New > Project**. The New Project dialog appears.
2. On the New Project dialog, select **JPA > JPA Project** and click **OK**. The New JPA Project dialog appears.
3. On the New JPA Project dialog, enter **Employee** as the **Project name** and click **Next**. The Project Facets page appears.
4. Verify that you have selected a Java 5.0 (or higher) and JPA 1.0 facet, and click **Next**. The JPA Facet page appears.
5. Select your vendor-specific platform, database connection, and JPA implementation library, and click **Finish**.

Eclipse adds the project to the workbench and opens the Java perspective.
The next step is to Create persistent Java entities.

### 1.3.3 Create persistent Java entities

The Tutorial Object Model contains three entities: `Employee`, `Address`, and `PhoneNumber`. Use this procedure to add the entities to the project.

1. Right-click the `Employee` project in the Package Explorer and select `New > Class`. The New Java Class dialog appears.

2. On the Java Class dialog, enter a package name (such as `dali.tutorial.model`), class name (such as `Employee`), and click `Finish`. Eclipse adds the Employee entity to the Package Explorer.

3. Select the `Employee` entity in the JPA Structure view.

4. In the JPA Details view, in the Map As field, select `Entity`. In the Table field, select the `EMPLOYEE` database table.
Eclipse adds the @Entity annotation to the class. Repeat this procedure to add the PhoneNumber and Address entities.

Notice that the Problems view reports several errors for each entity. We’ll address these shortly.

1.3.3.1 Add fields to the entities

Before mapping the entities to the database, you must add the necessary fields to each entity.

1. Add the following fields to the Employee entity:

```java
private Long id;
private String firstName;
private String lastName;
private String address;
private List<PhoneNumber> phoneNumbers;
private Long version;
```

2. Import java.util.List.

3. Generate Getters and Setters for each field.

4. Add the following fields to the Address entity:

```java
private Long id;
private String street;
private String city;
private String stateOrProvince;
private String country;
private String postalCode;
```

5. Add the following fields to the PhoneNumber entity:

```java
private String type;
private String areaCode;
private String number;
private Employee owner;
```

1.3.3.2 Associate the entity with a database table

Now you must associate each entity with its primary database table.

1. Select the Employee class in the Explorer view.
2. In the JPA Details view, notice that Dali has automatically selected the EMPLOYEE table as the table name.

![Figure 1–16  JPA Details View for the Employee Entity](image)

By default, Dali attempts to associate each entity with a similarly named database table. Notice that although you have not explicitly associated the Address entity yet, there is no error in the Problems view because the entity name, Address, is identical to the table name (ADDRESS).

For the PhoneNumber entity, however, there is an error. This is because the entity name (PhoneNumber) is different than the database table (PHONE). You must explicitly associate the entity with the PHONE table. Dali adds the @Table(name="PHONE") annotation to the entity.

Now you are ready to Create OR mappings.

### 1.3.4 Create OR mappings

Now you’re ready to map the attributes of each persistent entity to columns in the appropriate database table. For the tutorial application, you will use the following mapping types:

- ID mappings
- Basic mappings
- One-to-one mappings
- Many-to-one mappings
- One-to-many mappings
- Version mappings

#### 1.3.4.1 Create ID mappings

Use an ID Mapping to specify the primary key of an entity. Each persistent entity must have an ID. Notice that the Problems view reports that each entity is missing an ID.

1. Select the Employee entity in the Package Explorer view.
2. Expand the Employee entity in the JPA Structure view and select the id field. The JPA Details view (for attributes) displays the properties for the field.
3. In the Map As field, select ID.
4. Use this table to complete the remaining fields in the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map As</td>
<td>Defines this mapping as an <strong>ID Mapping</strong>. Dali adds the @Id annotation to the entity.</td>
</tr>
<tr>
<td>Column</td>
<td>The database column for the primary key of the table associated with the entity. Select <strong>EMP_ID</strong>. Because the database column (EMP_ID) is named differently than the entity field (id), Dali adds the @Column(name=&quot;EMP_ID&quot;) annotation.</td>
</tr>
</tbody>
</table>

5. Leave all other fields on the tab as their defaults. Expand the **Primary Key Generation** area.

6. Use this table to complete the Primary Key Generation fields in the JPA Details view.
In the JPA Structure view, the id field is identified as the primary key by the following icon:

**Figure 1–19  JPA Structure for Employee Entity**

Repeat this procedure to map the following primary keys (as shown in Table 1–1, "Tutorial Database Schema"):

- The id field of the Address entity to the ADDRESS_ID column of the ADDRESS table.
- The number field of the PhoneNumber entity to the P_NUMBER column of the PHONE table.

### 1.3.4.2 Create basic mappings

Use a Basic Mapping to map an attribute directly to a database column. In the Tutorial Object Model, the firstName field of the Employee class maps directly to the F_NAME column of the EMPLOYEE database table.

1. Select the Employee entity in the Package Explorer view.
2. In the JPA Structure view, select the firstName field of the Employee entity. The JPA Details view (for attributes) displays the properties for the field.
3. In the Map As field, select Basic. In the Column field, select F_NAME.
Dali adds the `@Column(name="P_NAME")` annotation to the entity. In the JPA Structure, the `firstName` field is identified as a basic mapping as shown in the following figure:

**Figure 1–20  Basic Mapping for firstName**

Dali adds the `@Column(name="P_NAME")` annotation to the entity. In the JPA Structure, the `firstName` field is identified as a basic mapping as shown in the following figure:

**Figure 1–21  JPA Structure for Employee Entity**

Repeat this procedure to map each of the following fields as **Basic** mappings:

- **Employee entity**
  - `lastName` field to L_NAME column

- **Address Entity**
  - `city` field to CITY column
  - `country` field to COUNTRY column
  - `postalCode` field to P_CODE column
  - `stateOrProvince` field to PROVINCE column
  - `street` field to STREET column

  **Note:** Because the `city`, `country`, and `street` fields are named identically to their database columns, Dali automatically maps the fields; no annotations are required.

- **Phone Entity**
  - `areaCode` field to AREA_CODE column
  - `type` field to TYPE column
1.3.4.3 Create one-to-one mappings

Use a **One-to-One Mapping** to define a relationship from an attribute to another class, with one-to-one multiplicity to a database column. In the **Tutorial Object Model**, the address field of the Employee class has a one-to-one relationship to the Address class; each employee may have a single address.

1. Select the **Employee** entity in the Package Explorer view.

2. In the **JPA Structure view**, select the address field of the Employee entity. The JPA Details view (for attributes) displays the properties for the field.

3. In the Map As field, select One-to-One.

4. **Figure 1–22 One-to-one Mapping for address**

4. For the Target Entity, click **Browse** and select the Address persistent entity. Dali adds the
  `@OneToOne(targetEntity=dali.tutorial.model.Address.class)`
  entity to the class.

   Leave the other fields with their default values.

5. Select the **Override Default** option to specify the relationship between the Employee and Address entities. Because you had to explicitly define the ID field for the Address entity in its ID mapping, you will need to edit the default join relationship.

6. Select the `address_ADDRESS_ID -> ADDRESS_ID` relationship in the Join Columns area and click **Edit**.

7. In the Edit Join Column dialog, select the following options and click **OK**.
   - Name: **ADDR_ID** (from the EMPLOYEE table)
   - Referenced Column Name: **ADDRESS_ID** (from the ADDRESS table)

---

**Note:** Because the type field is named identically to its database column, Dali automatically maps the field. No annotation is required.
In the JPA Structure, the **address** field is identified as a one-to-one mapping, as shown in the following figure:

**Figure 1–24 JPA Structure for Employee Entity**

1.3.4.4 Create one-to-many mappings

Use a **One-to-Many Mapping** to define a relationship from an attribute to another class, with one-to-many multiplicity to a database column. In the **Tutorial Object Model**, the **phoneNumbers** field of the **Employee** class has a one-to-many relationship to the **Phone** class; each employee may have many phone numbers.

1. Select the **Employee** entity in the Package Explorer view.

2. In the JPA Structure view, select the **phoneNumber** field of the **Employee** entity. The JPA Details view (for attributes) displays the properties for the field.

3. In the Map As field, select **One-to-Many**.
4. Select **PhoneNumber** as the Target Entity. Leave the other fields with their default values.

5. In the Join Table area, notice that Dali has selected the correct joins, based on the foreign key associations in the database tables.

In the JPA Structure, the **phoneNumbers** field is identified as a one-to-many mapping as shown in the following figure:

---

**Figure 1–26  JPA Structure for Employee Entity**

---

### 1.3.4.5 Create many-to-one mappings

Use a **May-to-One Mapping** to define a relationship from an attribute to another class, with many-to-one multiplicity to a database column. In the **Tutorial Object Model**, the **owner** field of the **PhoneNumber** class has a one-to-many relationship to the **Employee** class; there are many phone numbers that each employee may have.

This is the “back mapping” of the one-to-many mapping you previously defined.

1. Select the **PhoneNumber** entity in the Package Explorer view.

2. In the **JPA Structure view**, select the **owner** field of the **PhoneNumber** entity. The **JPA Details view** (for attributes) displays the properties for the field.
3. In the Map As field, select **Many to One**.

**Figure 1–27 Many to One Mapping for owner**

4. Leave the other fields with their default values. Dali correctly completes the information based on the database structure and previously defined mappings.

5. Use the **Join Columns** area to specify the relationship between the PhoneNumber and Employee entities. Because you had to explicitly define the ID field for the Employee entity in its ID mapping, you will need to edit the default join relationship.

6. Select the **Override Default** option.

7. Select the **owner_EMP_ID -> EMP_ID** relationship in the Join Columns area and click **Edit**.

8. In the Edit Join Column dialog, select the following options and click **OK**.
   - Name: **EMP_ID** (from the PHONE table)
   - Referenced Column Name: **EMP_ID** (from the EMPLOYEE table)

In the JPA Structure, the **owner** field is identified as a many-to-one mapping as shown in the following figure:

**Figure 1–28 JPA Structure for PhoneNumber Entity**

1.3.4.6 Create version mappings

Use a **Version Mapping** to specify the database field used by a persistent entity for optimistic locking.

1. Select the **Employee** entity in the Package Explorer view.

2. In the **JPA Structure view**, select the version field of the **Employee** entity. The JPA Details view (for attributes) displays the properties for the field.
3. In the Map As field, select Version.

**Figure 1–29  Version Mapping for version**

Dali automatically selects the Version column in the EMPLOYEE database table. In the JPA Structure, the Version field is identified as a version mapping, as shown in the following figure:

**Figure 1–30  JPA Structure for Employee Entity**

Congratulations! All of the entities have been successfully mapped.
This section contains an overview of concepts you should be familiar with when using Dali to create mappings for Java persistent entities.

- Understanding Java persistence
- Understanding OR mappings
- Understanding EJB 3.0 Java Persistence API

In addition to these sections, you should review the following resources for additional information:

- Eclipse Dali project: http://www.eclipse.org/webtools/dali
- Eclipse Web Tools Platform project: http://www.eclipse.org/webtools

2.1 Understanding Java persistence

*Persistence* refers to the ability to store objects in a database and use those objects with transactional integrity. In a J2EE application, data is typically stored and persisted in the data tier, in a relational database.

*Entity beans* are enterprise beans that contain persistent data and that can be saved in various persistent data stores. The entity beans represent data from a database; each entity bean carries its own identity. Entity beans can be deployed using *application-managed persistence or container-managed persistence.*

2.2 Understanding OR mappings

The Dali OR (object-relational) Mapping Tool allows you to describe how your entity objects map to the data source (or other objects). This approach isolates persistence information from the object model–developers are free to design their ideal object model, and DBAs are free to design their ideal schema.

These mappings transform an object data member type to a corresponding relational database data source representation. These OR mappings can also transform object data members that reference other domain objects stored in other tables in the database and are related through foreign keys.

You can use these mappings to map simple data types including primitives (such as int), JDK classes (such as String), and large object (LOB) values. You can also use them to transform object data members that reference other domain objects by way of association where data source representations require object identity maintenance (such as sequencing and back references) and possess various types of multiplicity and
navigability. The appropriate mapping class is chosen primarily by the cardinality of the relationship.

### 2.3 Understanding EJB 3.0 Java Persistence API

The Java 2 Enterprise Edition (J2EE) Enterprise JavaBeans (EJB) are a component architecture that you use to develop and deploy object-oriented, distributed, enterprise-scale applications. An application written according to the Enterprise JavaBeans architecture is scalable, transactional, and secure.

The EJB 3.0 Java Persistence API (JPA) improves the EJB architecture by reducing its complexity through the use of metadata (annotations) and specifying programmatic defaults of that metadata.

#### 2.3.1 The persistence.xml file

The JPA specification requires the use of a persistence.xml file for deployment. This file defines the database and entity manager options, and may contain more than one persistence unit. Dali can use the Eclipse XML Editor to create and maintain this information. See "Managing the persistence.xml file" on page 3-3 for more information.

#### 2.3.2 The orm.xml file

Although the JPA specification emphasizes the use of annotations to specify persistence, you can also the orm.xml file to store this metadata. Dali can use the Eclipse XML Editor to create and maintain this information. The metadata must match the XSD specification of your selected JPA implementation. See "Managing the orm.xml file" on page 3-5 for more information.
Tasks

This section includes detailed step-by-step procedures for accessing the Dali OR mapping tool functionality.

- Creating a new JPA project
- Managing the persistence.xml file
- Managing the orm.xml file
- Adding persistence to a class
- Specifying additional tables
- Specifying entity inheritance
- Mapping an entity
- Generating entities from tables
- Validating mappings and reporting problems
- Modifying persistent project properties

3.1 Creating a new JPA project

Use this procedure to create a new JPA project.

2. On the New Project dialog, select JPA > JPA Project and click Next. The New JPA Project wizard appears.
3. Complete the fields on the New JPA Project page to specify the project name and location, target runtime, and pre-defined configuration.

4. Click Next. The Project Facets page appears.

5. Select the project facets to use to create the project and click Next. The JPA Facet page appears.
3.2 Managing the persistence.xml file

When creating a JPA project, (see "Creating a new JPA project") you can also create the persistence.xml file.

Eclipse creates the META-INF\persistence.xml file in your project’s directory:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
<persistence-unit name="" provider="" />
</persistence-unit>
</persistence>
```

6. Complete the fields on the JPA Facet page to specify your vendor-specific platform, database connection, and JPA implementation library.

**Note:** If the server runtime does not provide a JPA implementation, you must explicitly select a JPA implementation library.

To insure the portability of your application, you must explicitly list the managed persistence classes that are included in the persistence unit. If the server supports EJB 3.0, the persistent classes will be discovered automatically.

7. Click Finish. You should now open the JPA Development perspective.
3.2.1 Working with persistence.xml file

You can work with the persistence.xml by using the XML Editor.

Use this procedure to work with the persistence.xml file:

1. Right-click the persistence.xml file in the Package Explorer and select Open With > XML Editor.

2. Use the Persistence XML Editor to edit the persistence.xml file.

3.2.2 Synchronizing classes

As you work with the classes in your Java project, you will need to update the persistence.xml file to reflect the changes.

Use this procedure to synchronize the persistence.xml file:

1. Right-click the persistence.xml file in the Package Explorer and select JPA Tools > Synchronize Classes.
3.3 Managing the orm.xml file

When creating a JPA project, (see "Creating a new JPA project") you can also create the orm.xml file that defines the mapping metadata and defaults.

Eclipse creates the META-INF/orm.xml file in your project’s directory:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="<PERSISTENCE_VERSION>"
   xmlns="http://java.sun.com/xml/ns/persistence"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
   <persistence-unit name="<PERSISTENCE_UNIT_NAME>"
      providers="<PERSISTENCE_PROVIDER>"
   </persistence-unit>
</persistence>
```

3.3.1 Working with orm.xml file

You can work with the orm.xml by using the JPA Details view.

Use this procedure to work with the orm.xml file:

1. Right-click the orm.xml file in the Package Explorer and select **Open**.
2. In the JPA Structure view, select **EntityMappings**.
3. Use the JPA Details view to configure the entity mapping and persistence unit defaults.
3.4 Adding persistence to a class

You can make a Java class into one of the following persistent types:

- **Entity**
- **Embeddable**
- **Mapped superclass**

3.4.1 Entity

An **Entity** is a persistent domain object.

An entity *can be*:

- Abstract or concrete classes. Entities may also extend non-entity classes as well as entity classes, and non-entity classes may extend entity classes.

An entity *must have*:

- A no-arg constructor (public or protected); the entity class may have other constructors as well.

Each persistent entity must be mapped to a database table and contain a primary key. Persistent entities are identified by the @Entity annotation.

Use this procedure to add persistence to an existing entity:

1. Open the Java class in the Package Explorer.
2. Select the class in the JPA Structure view.
3. In the JPA Details view, use the Map As field to select Entity.
3.4.2 Embeddable

An **Embedded** class is a class whose instances are stored as part of an owning entity; it shares the identity of the owning entity. Each field of the embedded class is mapped to the database table associated with the owning entity.

To override the mapping information for a specific subclass, use the `@AttributeOverride` annotation for that specific class.

An embeddable entity is identified by the `@Embeddable` annotation.

Use this procedure to add embeddable persistence to an existing entity:

1. Open the Java class in the Package Explorer.
2. Select the class in the JPA Structure view.
3. In the JPA Details view, use the Map As drop-list to select **Embeddable**.

4. Complete the remaining JPA Details view (for entities).
3.4.3 Mapped superclass

An entities that extend a **Mapped Superclass** class inherit the persistent state and mapping information from a superclass. You should use a mapped superclass to define mapping information that is common to multiple entity classes.

A mapped superclass can be:

- Abstract or concrete classes

A mapped superclass cannot be:

- Be queried or passed as an argument to Entity-Manager or Query operations
- Be the target of a persistent relationship

A mapped superclass does not have a defined database table. Instead, its mapping information is derived from its superclass. To override the mapping information for a specific subclass, use the `@AttributeOverride` annotation for that specific class.

A mapped superclass is identified by the `@MappedSuperclass` annotation.

Use this procedure to add Mapped Superclass persistence to an existing entity:

1. Open the Java class in the Package Explorer.
2. Select the class in the JPA Structure view.
3. In the JPA Details view, use the **Map As** drop-list to select **Mapped Superclass**.

![Figure 3–9 Selecting Mapped Superclass Persistence](image)

4. Complete the remaining **JPA Details view (for entities)**.

3.5 Specifying additional tables

An entity may inherit properties from other entities. You can specify a specific strategy to use for inheritance.

Use this procedure to specify inheritance (`@Inheritance`) for an existing entity (`@Entity`):

1. Select the entity in the Package Explorer.
2. In the JPA Details view, select the **Secondary Tables** information.
Specifying entity inheritance

An entity may inherit properties from other entities. You can specify a specific strategy to use for inheritance.

Use this procedure to specify inheritance [@Inheritance] for an existing entity [@Entity]:

1. Select the entity in the Package Explorer.
2. In the JPA Details view, select the Inheritance information.

3. In the Strategy list, select one of the following the inheritance strategies:
   - A single table (default)
   - Joined table
   - One table per class

4. Use the following table to complete the remaining fields on the tab. See "Inheritance information" on page 4-3 for additional details.
Eclipse adds the following annotations the entity field:

```java
@Inheritance(strategy=InheritanceType.<INHERITANCE_STRATEGY>)
@DiscriminatorColumn(name="<DISCRIMINATOR_COLUMN>", discriminatorType=<DISCRIMINATOR_TYPE>)
@DiscriminatorValue(value="<DISCRIMINATOR_VALUE>")
@PrimaryKeyJoinColumn(name="<JOIN_COLUMN_NAME>", referencedColumnName="<REFERENCED_COLUMN_NAME>")
```

The following figures illustrates the different inheritance strategies.

**Figure 3–12 Single Table Inheritance**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discriminator Column</td>
<td>Name of the discriminator column when using a Single or Joined inheritance strategy. This field corresponds to the @DiscriminatorColumn annotation.</td>
<td></td>
</tr>
<tr>
<td>Discriminator Type</td>
<td>Set the discriminator type to Char or Integer (instead of its default: String). The Discriminator Value must conform to this type.</td>
<td>String</td>
</tr>
<tr>
<td>Discriminator Value</td>
<td>Specify the discriminator value used to differentiate an entity in this inheritance hierarchy. The value must conform to the specified Discriminator Type. This field corresponds to the @DiscriminatorValue annotation.</td>
<td></td>
</tr>
<tr>
<td>Override Default</td>
<td>Use this field to specify custom primary key join columns. This field corresponds to the @PrimaryKeyJoinColumn annotation.</td>
<td></td>
</tr>
</tbody>
</table>
3.7 Mapping an entity

Dali supports the following mapping types for Java persistent entities:

- Basic mapping
- Embedded mapping
- Embedded ID mapping
- ID mapping
- Many-to-many mapping
- Many-to-one mapping
- One-to-many mapping
- One-to-one mapping
- Transient mapping
- Version mapping

3.7.1 Basic mapping

Use a Basic Mapping to map an attribute directly to a database column. Basic mappings may be used only with the following attribute types:

- Java primitive types and wrappers of the primitive types
  - java.lang.String, java.math.BigInteger
  - java.math.BigDecimal
  - java.util.Date
  - java.util.Calendar, java.sql.Date
  - java.sql.Time
  - java.sql.Timestamp
  - byte[]
Mapping an entity

- Byte[]
- char[]
- Character[]
- enums
- any other type that implements Serializable

To create a basic mapping:

1. In the JPA Structure view, select the field to map. The JPA Details view (for attributes) displays the properties for the selected field.
2. In the Map As field, select Basic.
3. Use this table to complete the remaining fields on the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map As</td>
<td>Defines this mapping as a Basic Mapping.</td>
<td>Basic</td>
</tr>
<tr>
<td></td>
<td>This field corresponds to the @Basic annotation.</td>
<td></td>
</tr>
<tr>
<td>Column</td>
<td>The database column mapped to the entity attribute. See “Column” on page 4-5 for details.</td>
<td>By default, the Column is assumed to be named identically to the attribute and always included in the INSERT and UPDATE statements.</td>
</tr>
<tr>
<td>Table</td>
<td>Name of the database table.</td>
<td></td>
</tr>
<tr>
<td>Fetch</td>
<td>Defines how data is loaded from the database. See “Fetch Type” on page 4-5 for details.</td>
<td>Eager</td>
</tr>
<tr>
<td></td>
<td>■ Eager</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Lazy</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>Specifies if this field is can be null.</td>
<td>Yes</td>
</tr>
<tr>
<td>Lob</td>
<td>Specifies if this is a large objects (BLOB or CLOB). See “Lob” on page 4-5 for details.</td>
<td></td>
</tr>
<tr>
<td>Temporal</td>
<td>Specifies the type of data. See “Temporal” on page 4-5 for details.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Date</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Timestamp</td>
<td></td>
</tr>
</tbody>
</table>

Eclipse adds the following annotations to the field:

```java
@Column(name="<COLUMN_NAME>", table="<COLUMN_TABLE>",
         insertable=<INSERTABLE>, updatable=<UPDATABLE>)
@Basic(fetch=<FETCH_TYPE>, <FETCH_TYPE>, optional = <OPTIONAL>)
@Temporal(<TEMPORAL>)
```

3.7.2 Embedded mapping

Use an Embedded Mapping to specify a persistent field or property of an entity whose value is an instance of an embeddable class.
1. In the JPA Structure view, select the field to map. The JPA Details view (for attributes) displays the properties for the selected field.

2. In the Map As field, select Embedded.

3. Use this table to complete the remaining fields on the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map As</td>
<td>Defines this mapping as a Embedded.</td>
<td>Embedded</td>
</tr>
<tr>
<td></td>
<td>This field corresponds to the @Embedded annotation.</td>
<td></td>
</tr>
<tr>
<td>Attribute Overrides</td>
<td>Specity to override the default mapping of an entity’s attribute. Select Override Default.</td>
<td></td>
</tr>
<tr>
<td>Columns</td>
<td>The database column (and its table) mapped to the entity attribute. See “Column” on page 4-5 for details.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Name – Name of the database column.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Table – Name of the database table.</td>
<td></td>
</tr>
</tbody>
</table>

Eclipse adds the following annotations to the field:

```java
@Embedded
@AttributeOverride(column=@Column(table="<COLUMN_TABLE>", name = "<COLUMN_NAME>"))
```

### 3.7.3 Embedded ID mapping

Use an Embedded ID Mapping to specify the primary key of an embedded ID. These mappings may be used with an Embeddable entities.

1. In the JPA Structure view, select the field to map. The JPA Details view (for attributes) displays the properties for the selected field.

2. In the Map As field, select Embedded Id.

3. Use this table to complete the remaining fields on the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map As</td>
<td>Defines this mapping as a Embedded Id.</td>
<td>Embedded Id</td>
</tr>
<tr>
<td></td>
<td>This field corresponds to the @EmbeddedId annotation.</td>
<td></td>
</tr>
</tbody>
</table>

Eclipse adds the following annotations to the field:

```java
@EmbeddedId
```

### 3.7.4 ID mapping

Use an ID Mapping to specify the primary key of an entity. ID mappings may be used with a Entity or Mapped superclass. Each Entity must have an ID mapping.
1. In the JPA Structure view, select the field to map. The JPA Details view (for attributes) displays the properties for the selected.

2. In the Map As field, select ID.

3. Use this table to complete the General information fields in the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map As</td>
<td>Defines this mapping as an ID Mapping.</td>
<td>ID</td>
</tr>
<tr>
<td></td>
<td>This field corresponds to the @Id annotation.</td>
<td></td>
</tr>
<tr>
<td>Column</td>
<td>The database column mapped to the entity attribute. See &quot;Column&quot; on page 4-5 for details.</td>
<td>By default, the Column is assumed to be named identically to the attribute.</td>
</tr>
<tr>
<td>Table</td>
<td>The database table mapped to the entity attribute.</td>
<td>By default, the Table is assumed to be identical to the table associated with the entity.</td>
</tr>
<tr>
<td>Temporal</td>
<td>Specifies the type of data. See &quot;Temporal&quot; on page 4-5 for details.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Date</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Timestamp</td>
<td></td>
</tr>
</tbody>
</table>

4. Use this table to complete the fields in Primary Key Generation information area in the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Key Generation</td>
<td>These fields define how the primary key is generated.</td>
<td></td>
</tr>
<tr>
<td>Strategy</td>
<td>See &quot;Primary Key Generation&quot; on page 4-7 for details.</td>
<td>Auto</td>
</tr>
<tr>
<td></td>
<td>■ Auto</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Sequence</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Identity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Table</td>
<td></td>
</tr>
<tr>
<td>Generator Name</td>
<td>Name of the primary key generator specified in the Strategy</td>
<td></td>
</tr>
</tbody>
</table>

Additional fields will appear in the Primary Key Generation information area, depending on the selected Strategy. See "JPA Details view (for attributes)" on page 4-4 for additional information.

Eclipse adds the following annotations to the field:

```java
@Id
@Column(name="<COLUMN_NAME>", table="<TABLE_NAME>", insertable=<INSERTABLE>, updatable=<UPDATABLE>)
@Temporal(<TEMPORAL>)
@GeneratedValue(strategy=GeneratorType.<STRATEGY>, generator="<GENERATOR_NAME>")
@TableGenerator(name="<TABLE_GENERATOR_NAME>", table = "<TABLE_GENERATOR_TABLE>", pkColumnName = "<TABLE_GENERATOR_PK>", valueColumnName = "<TABLE_GENERATOR_VALUE_COLUMN>"),```
3.7.5 Many-to-many mapping

Use a Many-to-Many Mapping to define a many-valued association with many-to-many multiplicity. A many-to-many mapping has two sides: the **owning side** and **non-owning side**. You must specify the join table on the owning side. For bidirectional mappings, either side may be the owning side.

1. In the JPA Structure view, select the field to map. The JPA Details view (for attributes) displays the properties for the selected.

2. In the Map As field, select **Many-to-Many**.

3. Use this table to complete the **General information** fields of the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map As</td>
<td>Defines this mapping as a Many to Many Mapping.</td>
<td>Many to Many</td>
</tr>
<tr>
<td></td>
<td>This field corresponds to the @ManyToMany annotation.</td>
<td></td>
</tr>
<tr>
<td>Target Entity</td>
<td>The entity to which this attribute is mapped.</td>
<td>null</td>
</tr>
<tr>
<td></td>
<td>You do not need to explicitly specify the target entity, since it can be</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inferred from the type of object being referenced.</td>
<td></td>
</tr>
<tr>
<td>Fetch</td>
<td>Defines how data is loaded from the database. See &quot;Fetch Type&quot; on page 4-5 for details.</td>
<td>Lazy</td>
</tr>
<tr>
<td></td>
<td>■ Eager</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Lazy</td>
<td></td>
</tr>
<tr>
<td>Mapped By</td>
<td>The database field that owns the relationship.</td>
<td></td>
</tr>
<tr>
<td>Order By</td>
<td>Specify the default order for objects returned from a query. See &quot;Order By&quot; on page 4-6 for details.</td>
<td>No ordering</td>
</tr>
<tr>
<td></td>
<td>■ No ordering</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Primary key</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Custom</td>
<td></td>
</tr>
</tbody>
</table>

4. Use this table to complete the fields in the **Join Table Information** area in the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the join table that contains the foreign key column.</td>
<td>You must specify the join table on the owning side. By default, the name is assumed to be the primary tables associated with the entities concatenated with an underscore.</td>
</tr>
</tbody>
</table>
5. To add a new Join or Inverse Join Column, click Add.
   To edit an existing Join or Inverse Join Column, select the field to and click Edit.

Eclipse adds the following annotations to the field:

```java
@JoinTable(joinColumns=@JoinColumn(name="<JOIN_COLUMN>",
  name = "<JOIN_TABLE_NAME>"),
@ManyToMany(cascade=CascadeType.<CASCADE_TYPE>, fetch=FetchType.<FETCH_TYPE>,
  targetEntity=<TARGET_ENTITY>, mappedBy = "<MAPPED_BY>"),
@OrderBy("<ORDER_BY>")
```

### 3.7.6 Many-to-one mapping

Use a Many-to-One mapping to defines a single-valued association to another entity class that has many-to-one multiplicity.

1. In the JPA Structure view, select the field to map. The JPA Details view (for attributes) displays the properties for the selected.
2. In the Map As field, select Many-to-One.
3. Use this table to complete the General information fields JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Entity</td>
<td>The entity to which this attribute is mapped.</td>
<td>null</td>
</tr>
<tr>
<td>Fetch Type</td>
<td>Defines how data is loaded from the database. See &quot;Fetch Type&quot; on page 4-5 for details.</td>
<td>Eager</td>
</tr>
<tr>
<td></td>
<td>■ Eager</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Lazy</td>
<td></td>
</tr>
</tbody>
</table>

4. Use this table to complete the fields on the Join Columns Information tab in the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join Column</td>
<td>Specify a mapped column for joining an entity association. This field corresponds to the @JoinColumn attribute. Select Override Default, then Add, Edit, or Remove the join columns.</td>
<td>By default, the mapping is assumed to have a single join.</td>
</tr>
</tbody>
</table>
Eclipse adds the following annotations to the field:

```java
@JoinTable(joinColumns=@JoinColumn(name="<JOIN_COLUMN>",
    name = "<JOIN_TABLE_NAME>"),
@ManyToOne(targetEntity=<TARGET_ENTITY>, fetch=<FETCH_TYPE>,
cascade=<CASCADE_TYPE>)
```

### 3.7.7 One-to-many mapping

Use a **One-to-Many Mapping** to define a relationship with one-to-many multiplicity.

1. In the JPA Structure view, select the field to map. The JPA Details view (for attributes) displays the properties for the selected.

2. In the Map As field, select **One-to-many**.

3. Use this table to complete the **General information** fields JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Entity</td>
<td>The entity to which this attribute is mapped.</td>
<td></td>
</tr>
<tr>
<td>Fetch Type</td>
<td>Defines how data is loaded from the database. See &quot;Fetch Type&quot; on page 4-5 for details.</td>
<td>Eager</td>
</tr>
<tr>
<td></td>
<td>■ Eager</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Lazy</td>
<td></td>
</tr>
<tr>
<td>Mapped By</td>
<td>The database field that owns the relationship.</td>
<td></td>
</tr>
<tr>
<td>Order By</td>
<td>Specify the default order for objects returned from a query. See &quot;Order By&quot; on page 4-6 for details.</td>
<td>No ordering</td>
</tr>
<tr>
<td></td>
<td>■ No ordering</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Primary key</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Custom</td>
<td></td>
</tr>
</tbody>
</table>

4. Use this table to complete the **Join Table Information** fields in the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the join table</td>
<td>By default, the name is assumed to be the primary tables associated with the entities concatenated with an underscore.</td>
</tr>
<tr>
<td>Join Columns</td>
<td>Specify two or more join columns (that is, a primary key).</td>
<td></td>
</tr>
<tr>
<td>Inverse Join Columns</td>
<td>The join column on the owned (or inverse) side of the association: the owned entity’s primary key column.</td>
<td></td>
</tr>
</tbody>
</table>

Eclipse adds the following annotations to the field:

```java
@OneToMany(targetEntity=<TARGET_ENTITY>)
@Column(name="<COLUMN>")
```
### One-to-one mapping

Use a **One-to-One Mapping** to define a relationship with one-to-many multiplicity.

1. In the **JPA Structure view**, select the field to map. The **JPA Details view** (for attributes) displays the properties for the selected.

2. In the **Map As field**, select **One-to-one**.

3. Use this table to complete the **General information** fields in the **JPA Details view**.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Entity</td>
<td>The entity to which this attribute is mapped.</td>
<td>null</td>
</tr>
<tr>
<td></td>
<td>You do not need to explicitly specify the target entity, since it can be inferred from the type of object being referenced.</td>
<td></td>
</tr>
<tr>
<td>Fetch Type</td>
<td>Defines how data is loaded from the database. See “Fetch Type” on page 4-5 for details.</td>
<td>Eager</td>
</tr>
<tr>
<td></td>
<td>■ Eager</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Lazy</td>
<td></td>
</tr>
<tr>
<td>Mapped By</td>
<td>The database field that owns the relationship.</td>
<td></td>
</tr>
</tbody>
</table>

4. Use this table to complete the **Join Columns Information** fields in the **JPA Details view**.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join Column</td>
<td>Specify a mapped column for joining an entity association. This field corresponds to the @JoinColumn attribute.</td>
<td>By default, the mapping is assumed to have a single join.</td>
</tr>
<tr>
<td></td>
<td>Select <strong>Override Default</strong>, then Add, Edit, or Remove the join columns.</td>
<td></td>
</tr>
</tbody>
</table>

Eclipse adds the following annotations to the field:

```java
@OneToOne(targetEntity=<TARGET_ENTITY>.class, cascade=CascadeType.<CASCADE_TYPE>,
 fetch = FetchType.<FETCH_TYPE>, mappedBy = "<MAPPED_BY>")
@JoinColumn(name="<JOIN_COLUMN_NAME>", referencedColumnName="<JOIN_COLUMNREFERENCED_COLUMN>",
inverseJoinColumn=name="<INVERSE_JOIN_COLUMN_NAME>",
referencedColumnName="<INVERSE_JOIN_COLUMN_REFERENCED_COLUMN>")
```
3.7.9 Transient mapping

Use the Transient Mapping to specify a or field of the entity class that is not persistent.

To create a version mapping:
1. In the JPA Structure view, select the field to map. The JPA Details view (for attributes) displays the properties for the selected.
2. In the Map As field, select Transient.

Eclipse adds the following annotation to the field:

@Transient

3.7.10 Version mapping

Use a Version Mapping to specify the field used for optimistic locking. If the entity is associated with multiple tables, you should use a version mapping only with the primary table. You should have only a single version mapping per persistent entity. Version mappings may be used only with the following attribute types:

- int
- Integer
- short, Short
- long, Long
- Timestamp

To create a version mapping:
1. In the JPA Structure view, select the field to map. The JPA Details view (for attributes) displays the properties for the selected.
2. In the Map As field, select Version.
3. Use this table to complete the remaining fields in the JPA Details view.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
<td>The database column mapped to the entity attribute. See &quot;Column&quot; on page 4-5 for details.</td>
<td>By default, the Column is assumed to be named identically to the attribute and always included in the INSERT and UPDATE statements.</td>
</tr>
<tr>
<td>Table</td>
<td>Name of the database table. This must be the primary table associated with the attribute’s entity.</td>
<td></td>
</tr>
<tr>
<td>Temporal</td>
<td>Specifies the type of data. See &quot;Temporal&quot; on page 4-5 for details.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Date</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Timestamp</td>
<td></td>
</tr>
</tbody>
</table>

Eclipse adds the following annotations to the field:

@Version
@Column(table="<COLUMN_TABLE>", name="<COLUMN_NAME>")
3.8 Generating entities from tables

Use this procedure to generate Java persistent entities from database tables. You must create a JPA project and establish a database connection before generating persistent entities. See "Creating a new JPA project" on page 3-1 for more information.

1. Right-click the JPA project in the Package Explorer and select JPA Tools > Generate Entities.

2. If you are not currently connected to the database, the Database Connection page appears. Select your database connection and schema, and click Reconnect.

   To create a new database connection, click Add connection.

   After connecting to the database, click Next.

3. On the Generate Entities from Tables dialog, select the tables from which to generate Java persistent entities and click Finish.

   Eclipse creates a Java persistent entity for each database table. Each entity contains fields based on the table’s columns. Eclipse will also generate entity relationships (such as one-to-one) based on the table constraints. Figure 3–15 illustrates how Eclipse generates entities from tables.
3.9 Validating mappings and reporting problems

Errors and warnings on persistent entities and mappings are indicated with a red error or yellow warning next to the resource with the error, as well as the parent containers up to the project.

This section contains information on the following:

- Error messages
- Warning messages

3.9.1 Error messages

This section contains information on error messages (including how to resolve the issue) you may encounter while working with Dali.

Attribute "<ATTRIBUTE__NAME>" has invalid mapping type in this context

The mapped attribute is invalid. Either change the mapping type or change the entity type.
See "Mapping an entity" on page 3-11 for more information.

**Attribute "<ATTRIBUTE_NAME>" cannot be resolved.**
Dali cannot map the attribute to a database table and column. Verify that you database connection information is correct.
See "Creating a new JPA project" on page 3-1 for more information.

**Class "<CLASS_NAME>" is not annotated as a persistent class.**
The class has not been identified as a persistent class. Configure the class as an Entity, Mapped Superclass, or Embeddable persistent entity.
See "Adding persistence to a class" on page 3-6.

**Column "<COLUMN_NAME>" cannot be resolved.**
You mapped an entity’s field to an incorrect or invalid column in the database table. By default, Dali will attempt to map each field in the entity with an identically named row in the database table. If the field’s name differs from the row’s name, you must explicitly create the mapping.
Map the field to a valid row in the database table as shown in "Mapping an entity" on page 3-11.

**Duplicate class "<CLASS_NAME>".**
You created persistence classes with the same name. Each Java class must have a unique name. See "Adding persistence to a class" on page 3-6 for more information.

**Entity does not have an Id or Embedded Id.**
You created a persistent entity without identifying its primary key. A persistent entity must have a primary key field designated with an @Id or @EmbeddedId annotation.
Add an ID mapping to the entity as shown in "ID mapping" on page 3-13 or "Embedded ID mapping" on page 3-13.

**Multiple persistence.xml files in project.**
You created a JPA project with more than one persistence.xml file. Each JPA project must contain a single persistence.xml file.
See "Managing the persistence.xml file" on page 3-3 for more information.

**No generator named "<GENERATOR_NAME>" is defined in persistence unit.**
You created a persistence entity that uses sequencing, but did not define the sequence generator in the persistence.xml file. Synchronize the persistence.xml file with your current project.
See "Synchronizing classes" on page 3-4 for more information.

**No persistence unit defined.**
There is no persistence-unit-metadata information in the orm.xml file. Add the default persistence unit information.
See "Managing the orm.xml file" on page 3-5 for more information.

**No persistence.xml file in project.**
You created a JPA project without a persistence.xml file. Each JPA project must contain a single persistence.xml file.
See "Managing the persistence.xml file" on page 3-3 for more information.

**Referenced column "<COLUMN_NAME>" in join column "<COLUMN_NAME>" cannot be resolved.**
The column that you selected to join a relationship mapping does not exist on the database table. Either select a different column on the Join Table Information or create the necessary column on the database table.

See "JPA Details view (for attributes)" on page 4-4 for more information.

**Schema "<SCHEMA_NAME>" cannot be resolved for table/join table "<TABLE_NAME>".**
Define the default database schema information in the persistence unit.

See "Managing the orm.xml file" on page 3-5 for more information.

**Table "<TABLE_NAME>" cannot be resolved.**
You associated a persistent entity to an incorrect or invalid database table. By default, Dali will attempt to associate each persistent entity with an identically named database table. If the entity’s name differs from the table’s name, you must explicitly create the association.

Associate the entity with a valid database table as shown in "Adding persistence to a class" on page 3-6.

### 3.9.2 Warning messages

This section contains information on warning messages (including how to resolve the issue) you may encounter while working with Dali.

**Connection "<CONNECTION_NAME>" is not active. No validation will be done against the data source.**
The database connection you specified to use with the JPA project is not active. The JPA project requires an active connection.

**No connection specified for the project. No data-specific validation will be performed.**
You created a JPA project without specifying a database connection. The JPA project requires an active connection.

See "Creating a new JPA project" on page 3-1 or "Modifying persistent project properties" on page 3-23 for information on specifying a database connection.

### 3.10 Modifying persistent project properties

Each persistent project must be associated with a database connection. To create a new database connection, click Database Connection use the New Connection wizard.
Use this procedure to modify the vendor-specific platform and database connection associated with your JPA project.

1. Right-click the project in the Explorer view and select Properties. The Properties page appears.

![Properties – Persistence Page](image)

**Figure 3–17 Properties – Persistence Page**

2. Use this table to complete the remaining fields on the Properties – JPA page and click OK.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>Select the vendor-specific platform for the JPA implementation.</td>
</tr>
<tr>
<td>Database Connection</td>
<td>Database connection to use to store the persistent entities. To create a new connection, click Add Connection.</td>
</tr>
</tbody>
</table>

To create a new connection, click Add connections.
This section includes detailed help information for each of the following elements in the Dali OR Mapping Tool:

- Wizards
- Property pages
- Preferences
- Dialogs
- JPA Development perspective
- Icons and buttons
- Dali Developer Documentation

4.1 Wizards

This section includes information on the following wizards:

- Create New JPA Project wizard

4.1.1 Create New JPA Project wizard

The Create New JPA Project wizard allows you to create a new Java project using JPA. The wizard consists of the following pages:

- New JPA Project page
- Project Facets page
- JPA Facet page

4.1.1.1 New JPA Project page

This table lists the properties available on the New JPA Project page of the Create New JPA Project wizard.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project name</td>
<td>Name of the Eclipse JPA project.</td>
<td></td>
</tr>
<tr>
<td>Project contents</td>
<td>Location of the workspace in which to save the project. Unselect The Use Default option and click Browse to select a new location.</td>
<td>Current workspace</td>
</tr>
</tbody>
</table>
### 4.1.1.2 JPA Facet page

This table lists the properties available on the JPA Facet page of the Create New JPA Project wizard.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target runtime</td>
<td>Select a pre-defined target for the project. Click <strong>New</strong> to create a new environment with the New Server Runtime wizard.</td>
<td></td>
</tr>
<tr>
<td>Configurations</td>
<td>Select a project configuration with pre-defined facets. Select <strong>&lt;custom&gt;</strong> to manually select the facets for this project.</td>
<td>Utility JPA project with Java 5.0</td>
</tr>
<tr>
<td>EAR membership</td>
<td>Specify if this project should be included in an EAR file for deployment. Select the <strong>EAR Project Name</strong>, or click <strong>New</strong> to create a new EAR project.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform</td>
<td>Vendor-specific JPA implementation.</td>
<td>Generic</td>
</tr>
<tr>
<td>Connection</td>
<td>Select the database connection to use with the project. Dali requires an active database connection to use and validate the persistent entities and mappings. Click <strong>Add connection</strong> to create a new database connection.</td>
<td></td>
</tr>
<tr>
<td>JPA Implementation</td>
<td>Select to use the <strong>JPA implementation provided by the server at runtime</strong>, or select a specific <strong>implementation library</strong> that contain the Java Persistence API (JPA) and entities to be added to the project’s Java Build Path. Click <strong>Configure default JPA implementation library</strong> to create a default library for the project or click <strong>Configure user libraries</strong> to define additional libraries.</td>
<td>Determined by server.</td>
</tr>
<tr>
<td>Persistent class</td>
<td>Specify if Dali will <strong>discover annotated classes automatically</strong>, or if the <strong>annotated classes must be listed in the persistence.xml file</strong>. <strong>Note</strong>: To insure application portability, you should explicitly list the managed persistence classes that are included in the persistence unit.</td>
<td>Determined by server.</td>
</tr>
<tr>
<td>management</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create orm.xml</td>
<td>Specify if Dali should create a default <strong>orm.xml</strong> file for your entity mappings and persistence unit defaults.</td>
<td>Selected</td>
</tr>
</tbody>
</table>

### 4.2 Property pages

This section includes information on the following property pages:
4.2.1 JPA Details view (for entities)

The JPA Details view displays the persistence information for the currently selected entity and contains the following tabs:

- General information
- Secondary table information
- Inheritance information

4.2.1.1 General information

This table lists the General information fields available in the JPA Details view for each entity type.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
<th>Available for Entity Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map As</td>
<td>Specify the type of entity: Entity, Mapped Superclass, Embeddable.</td>
<td>Entity</td>
<td>Entity, Embeddable, and Mapped superclass</td>
</tr>
<tr>
<td>Name</td>
<td>The name of this entity. By default, the class name is used as the entity name.</td>
<td></td>
<td>Entity</td>
</tr>
<tr>
<td>Table</td>
<td>The primary database table associated with the entity.</td>
<td></td>
<td>Entity</td>
</tr>
<tr>
<td>Catalog</td>
<td>The database catalog that contains the Table.</td>
<td>As defined in orm.xml</td>
<td>Entity</td>
</tr>
<tr>
<td>Schema</td>
<td>The database schema that contains the Table.</td>
<td>As defined in orm.xml</td>
<td>Entity</td>
</tr>
<tr>
<td>Attribute Overrides</td>
<td>Specify a property or field to be overridden (from the default mappings). Select Override Default.</td>
<td></td>
<td>Entity</td>
</tr>
<tr>
<td>Column</td>
<td>The database column (from the Table Name) mapped to the entity.</td>
<td></td>
<td>Entity</td>
</tr>
<tr>
<td>Table</td>
<td>Name of the database table that contains the selected column.</td>
<td></td>
<td>Entity</td>
</tr>
</tbody>
</table>

4.2.1.2 Secondary table information

Use the Secondary Tables area in the Java Details view to associate additional tables with an entity. Use this area if the data associated with an entity is spread across multiple tables.

Refer to "Specifying additional tables" on page 3-9 for additional information.

4.2.1.3 Inheritance information

This table lists the fields available on the Inheritance area in the Java Details view for each entity type.
Refer to "Specifying entity inheritance" on page 3-9 for additional information.

### 4.2.2 JPA Details view (for attributes)

The JPA Details view displays the persistence information for the currently selected mapped attribute and contains the following areas:

- **General information**
- **Join Table Information**
- **Join Columns Information**
- **Primary Key Generation information**

See "Mapping an entity" on page 3-11 for more information.

#### 4.2.2.1 General information

This table lists the General properties available in the Java Details view for each mapping type.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
<th>Available for Mapping Type</th>
</tr>
</thead>
</table>
| Strategy                     | Specify the strategy to use when mapping a class or class hierarchy:  
  - Single table – All classes in the hierarchy are mapped to a single table.  
  - Joined – The root of the hierarchy is mapped to a single table; each child maps to its own table.  
  - Table per class – Each class is mapped to a separate table. | Single table | All mapping types          |
| Discriminator Column         | Use to specify the name of the discriminator column when using a Single or Joined inheritance strategy. |         |                             |
| Discriminator Type           | Set this field to set the discriminator type to Char or Integer (instead of its default: String). The Discriminator Value must conform to this type. | String  |                             |
| Discriminator Value          | Specify the discriminator value used to differentiate an entity in this inheritance hierarchy. The value must conform to the specified Discriminator Type. |         |                             |
| Primary Key Join Columns      | Select Override Default.  
  This field corresponds with @PrimaryKeyJoinColumn annotation. |         |                             |
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
<th>Available for Mapping Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
<td>The database column that contains the value for the attribute. This field corresponds to the @Column annotation.</td>
<td>By default, the Column is assumed to be named identically to the attribute.</td>
<td>Basic mapping, Embedded mapping, ID mapping, Version mapping</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the database column. This field corresponds to the @Column annotation.</td>
<td></td>
<td>Basic mapping, Embedded mapping, ID mapping</td>
</tr>
<tr>
<td>Table</td>
<td>Name of the database table that contains the selected column.</td>
<td></td>
<td>Basic mapping, Embedded mapping, ID mapping</td>
</tr>
<tr>
<td>Fetch Type</td>
<td>Defines how data is loaded from the database:</td>
<td>Eager</td>
<td>Basic mapping, One-to-one mapping</td>
</tr>
<tr>
<td></td>
<td>■ Eager – Data is loaded in before it is actually needed.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Lazy – Data is loaded only when required by the transaction.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>Specifies if this field is can be null.</td>
<td>Yes</td>
<td>Basic mapping, One-to-one mapping</td>
</tr>
<tr>
<td>Lob</td>
<td>Specify if the field is mapped to java.sql.Clob or java.sql.Blob.</td>
<td></td>
<td>Basic mapping</td>
</tr>
<tr>
<td></td>
<td>This field corresponds to the @Lob annotation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temporal</td>
<td>Specifies if this field is one of the following:</td>
<td></td>
<td>Basic mapping, ID mapping</td>
</tr>
<tr>
<td></td>
<td>■ Date – java.sql.Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Time – java.sql.Time</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ Timestamp – java.sql.Timestamp</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This field corresponds to the @Temporal annotation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enumerated</td>
<td>Specify how to persist enumerated constraints if the String value suits your application requirements or to match an existing database schema.</td>
<td>Ordinal</td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ ordinal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>■ String</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This field corresponds to the @Enumerated annotation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Target Entity</td>
<td>The persistent entity to which the attribute is mapped.</td>
<td></td>
<td>One-to-one mapping, One-to-many mapping, Many-to-many mapping, Many-to-one mapping</td>
</tr>
</tbody>
</table>
### 4.2.2.2 Join Table Information

Use area to specify a mapped column for joining an entity association. By default, the mapping is assumed to have a single join.

This table lists the fields available on the Join Table area in the JPA Details view for **One-to-many mapping** and **Many-to-many mapping** mapping types.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
<th>Available for Mapping Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascade Type</td>
<td>Specify which operations are propagated throughout the entity.</td>
<td></td>
<td>One-to-one mapping</td>
</tr>
<tr>
<td></td>
<td>• All – All operations</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Persist</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Merge</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Move</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mapped By</td>
<td>The field in the database table that &quot;owns&quot; the relationship. This field</td>
<td></td>
<td>One-to-one mapping, One-to-many mapping</td>
</tr>
<tr>
<td></td>
<td>is required only on the non-owning side of the relationship.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Order By</td>
<td>Specify the default order for objects returned from a query:</td>
<td>Primary key</td>
<td>One-to-many mapping, Many-to-many mapping, Many-to-one mapping</td>
</tr>
<tr>
<td></td>
<td>• No ordering</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Primary key</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Custom ordering</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This field corresponds to the @OrderBy annotation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute Overrides</td>
<td>Overrides the column mappings from the mapped, entity tabled. (for</td>
<td>Embedded mapping</td>
<td>Embedded mapping</td>
</tr>
<tr>
<td></td>
<td>example, if the inherited column name is incompatible with a pre-existing</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>data model, or invalid as a column name in your database).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
<th>Available for Mapping Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the join table that contains the foreign key column.</td>
<td>By default, the name is assumed to be the primary tables associated with the entities concatenated with an underscore.</td>
<td></td>
</tr>
<tr>
<td>Join Columns</td>
<td>Specify a mapped column for joining an entity association. This field</td>
<td>By default, the mapping is assumed to have a single join.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>corresponds to the @JoinColumn attribute.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Select Override Default, then Add,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Edit, or Remove the join columns.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inverse Join Columns</td>
<td>Select Override Default, then Add,</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Edit, or Remove the join columns.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 4.2.2.3 Join Columns Information

This table lists the fields available in the Join Table area in JPA Details view for Many-to-one mapping and One-to-one mapping mapping types.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join Column</td>
<td>Specify a mapped column for joining an entity association. This field corresponds to the @JoinColumn attribute. Select Override Default, then Add, Edit, or Remove the join columns.</td>
<td>By default, the mapping is assumed to have a single join.</td>
</tr>
</tbody>
</table>

### 4.2.2.4 Primary Key Generation Information

This table lists the fields available in the Primary Key Generation area in JPA Details view for ID mapping types.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Key Generation</td>
<td>These fields define how the primary key is generated. These fields correspond to the @GeneratedValue annotation.</td>
<td>Generated Value</td>
</tr>
</tbody>
</table>
| Strategy         | ■ Auto  
■ Identity – Values are assigned by the database’s Identity column.  
■ Sequence – Values are assigned by a sequence table (see Sequence Generator).  
■ Table – Values are assigned by a database table (see Table Generator). | Auto                                         |
| Generator Name   | Unique name of the generated value.                                                                                                                                                                         |                                              |
| Table Generator  | These fields define the database table used for generating the primary key and correspond to the @TableGenerator annotation. These fields apply only when Strategy = Table. |                                              |
| Name             | Unique name of the generator.                                                                                                                                                                              |                                              |
| Table            | Database table that stores the generated ID values.                                                                                                                                                       |                                              |
| Primary Key Column | The column in the table generator’s Table that contains the primary key.                                                                            |                                              |
| Value Column     | The column that stores the generated ID values.                                                                                                                                                           |                                              |
| Primary Key Column Value | The value for the Primary Key Column in the generator table.                                                                                     |                                              |
4.2.3 JPA Details view (for orm.xml)

The JPA Details view displays the default mapping and persistence information for the project and contains the following areas:

- **General information**
- **Persistence Unit information**

These defaults can be overridden by the settings on a specific entity or mapping.

### 4.2.3.1 General information

This table lists the General information fields available in the JPA Details view for each entity type.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>The Java package that contains the persistent entities. Click <strong>Browse</strong> and select the package.</td>
<td></td>
</tr>
<tr>
<td>Schema</td>
<td>The database schema that contains the <strong>Table</strong>. This field corresponds to the <code>&lt;schema&gt;</code> element in the <code>orm.xml</code> file.</td>
<td></td>
</tr>
<tr>
<td>Catalog</td>
<td>The database catalog that contains the <strong>Table</strong>. This field corresponds to the <code>&lt;catalog&gt;</code> element in the <code>orm.xml</code> file.</td>
<td></td>
</tr>
</tbody>
</table>
| Access         | Specify the default access method for the variables in the project:  
                | - **Property**  
                | - **Field**  
                | This field corresponds to the `<access>` element in the `orm.xml` file. |         |

### 4.2.3.2 Persistence Unit information

This table lists the Persistence Unit information fields available in the JPA Details view for each entity type. These fields are contained in the `<persistence-unit-metadata>` element in the `orm.xml` file.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Generator</td>
<td>These fields define the specific sequence used for generating the primary key and correspond to the @SequenceGenerator annotation. These fields apply only when <strong>Strategy</strong> = <strong>Sequence</strong>.</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Name of the sequence table to use for defining primary key values.</td>
<td></td>
</tr>
<tr>
<td>Sequence</td>
<td>Unique name of the sequence.</td>
<td></td>
</tr>
</tbody>
</table>
4.2.4 JPA Structure view

The JPA Structure view displays an outline of the structure (its attributes and mappings) of the entity that is currently selected or opened in the editor. The structural elements shown in the outline are the entity and its fields.

Figure 4–1  Sample JPA Structure View

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Mapping Data Complete</td>
<td>Specifies that the Java classes in this persistence unit are fully specified by their metadata. Any annotations will be ignored. This field corresponds to the <code>&lt;xml-mapping-metadata-complete&gt;</code> element in the orm.xml file.</td>
<td></td>
</tr>
<tr>
<td>Package</td>
<td>The Java package that contains the persistent entities for this persistence unit. Click Browse and select the package.</td>
<td></td>
</tr>
<tr>
<td>Schema</td>
<td>The database schema that contains the Table. This field corresponds to the <code>&lt;schema&gt;</code> element in the orm.xml file.</td>
<td></td>
</tr>
<tr>
<td>Catalog</td>
<td>The database catalog that contains the Table. This field corresponds to the <code>&lt;catalog&gt;</code> element in the orm.xml file.</td>
<td></td>
</tr>
<tr>
<td>Cascade Persist</td>
<td>Adds cascade-persist to the set of cascade options in entity relationships of the persistence unit. This field corresponds to the <code>&lt;cascade-persist&gt;</code> element in the orm.xml file.</td>
<td></td>
</tr>
</tbody>
</table>

4.3 Preferences

This section includes information on the following preference pages:

- Project Properties page – JPA Options

4.3.1 Project Properties page – JPA Options

Use the JPA options on the Properties page to select the database connection to use with the project.

This table lists the properties available in the Persistence Properties page.
### 4.4 Dialogs

This section includes information on the following preference pages:

- **Generate Entities from Tables dialog**
- **Edit Join Columns Dialog**

#### 4.4.1 Generate Entities from Tables dialog

Use the Generate Entities dialog to create Java persistent entities from your database tables and columns.

This table lists the properties available in the Generate Entities dialog.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Folder</td>
<td>Enter a project folder name in which to generate the Java persistent entities, or click Browse to select an existing folder.</td>
</tr>
<tr>
<td>Package</td>
<td>Enter a package name in which to generate the Java persistent entities, or click Browse to select an existing package.</td>
</tr>
<tr>
<td>Tables</td>
<td>Select the tables from which to create Java persistent entities. The tables shown are determined by the database connection that you defined in the Project Properties page – JPA Options.</td>
</tr>
</tbody>
</table>

See "Generating entities from tables" on page 3-20 for more information.

#### 4.4.2 Edit Join Columns Dialog

Use the Join Columns dialog to create or modify the join tables and columns in relationship mappings.

This table lists the properties available in the Join Columns dialog.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the joint table column that contains the foreign key column.</td>
</tr>
<tr>
<td>Referenced Column Name</td>
<td>Name of the database column that contains the foreign key reference for the entity relationship.</td>
</tr>
</tbody>
</table>
4.5 JPA Development perspective

The JPA Development perspective defines the initial set and layout of views in the Workbench window when using Dali. By default, the JPA Development perspective includes the following views:

- JPA Structure view
- JPA Details view (for entities)
- JPA Details view (for attributes)
- JPA Details view (for orm.xml)

![Sample JPA Development Perspective](image)

4.6 Icons and buttons

This section includes information on each of the icons and buttons used in the Dali OR Mapping Tool.

- Icons
- Buttons

4.6.1 Icons

The following icons are used throughout the Dali OR Mapping Tool.
### 4.6.2 Buttons

The following buttons are used throughout the Dali OR Mapping Tool.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Icon" /></td>
<td>Nonpersistent class</td>
</tr>
<tr>
<td><img src="image2.png" alt="Icon" /></td>
<td>Entity</td>
</tr>
<tr>
<td><img src="image3.png" alt="Icon" /></td>
<td>Embeddable entity</td>
</tr>
<tr>
<td><img src="image4.png" alt="Icon" /></td>
<td>Mapped superclass</td>
</tr>
<tr>
<td><img src="image5.png" alt="Icon" /></td>
<td>Basic mapping</td>
</tr>
<tr>
<td><img src="image6.png" alt="Icon" /></td>
<td>Embedded mapping</td>
</tr>
<tr>
<td><img src="image7.png" alt="Icon" /></td>
<td>Embedded ID mapping</td>
</tr>
<tr>
<td><img src="image8.png" alt="Icon" /></td>
<td>ID mapping</td>
</tr>
<tr>
<td><img src="image9.png" alt="Icon" /></td>
<td>Many-to-many mapping</td>
</tr>
<tr>
<td><img src="image10.png" alt="Icon" /></td>
<td>Many-to-one mapping</td>
</tr>
<tr>
<td><img src="image11.png" alt="Icon" /></td>
<td>One-to-many mapping</td>
</tr>
<tr>
<td><img src="image12.png" alt="Icon" /></td>
<td>One-to-one mapping</td>
</tr>
<tr>
<td><img src="image13.png" alt="Icon" /></td>
<td>Transient mapping</td>
</tr>
<tr>
<td><img src="image14.png" alt="Icon" /></td>
<td>Version mapping</td>
</tr>
<tr>
<td><img src="image15.png" alt="Icon" /></td>
<td>JPA Development perspective</td>
</tr>
</tbody>
</table>
4.7 Dali Developer Documentation

Additional Dali documentation is available online at:


This developer documentation includes information about:

- Dali architecture
- Plugins that comprise the Dali JPA Eclipse feature
- Extension points
The following tips and tricks give some helpful ideas for increasing your productivity.

- **Database Connections**
- **Schema-based persistence.xml**

<table>
<thead>
<tr>
<th>Tip</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Database Connections</strong></td>
<td>When starting a new workbench session, be sure to reconnect to your database (if you are working online). This allows Dali to provide database-related mapping assistance and validation.</td>
</tr>
<tr>
<td><strong>Schema-based persistence.xml</strong></td>
<td>If you are behind a firewall, you may need to configure your Eclipse workspace proxy in the Preferences dialog (Preferences &gt; Internet &gt; Proxy Settings) to properly validate a schema-based persistence.xml file.</td>
</tr>
</tbody>
</table>
This section contains descriptions of the following new feature and significant changes made to the Dali OR Mapping Tool for Release 1.0.0:

- Generate Persistent Entities from Tables wizard
- Create and Manage the persistence.xml file
- Create and Manage the orm.xml file

6.1 Generate Persistent Entities from Tables wizard

Use the Generate Entities from Tables wizard to quickly create JPA entities from your database tables.

![Generating Entities](image)

Dali automatically creates the necessary OR mappings, based on your database table constraints.

6.2 Create and Manage the persistence.xml file

When creating a JPA project, Dali automatically creates the persistence.xml file.
Use the XML editor to edit the persistence.xml file.

After adding your JPA entities, use the Java Persistence > Synchronize Classes option to add the classes to the persistence.xml file.

6.3 Create and Manage the orm.xml file

When creating a JPA project, you can also create the orm.xml file. Select the Create orm.xml option on the JPA Facet page page of the Create New JPA Project wizard.
Use the `orm.xml` file to define the project and persistence unit defaults.
The material in this guide is copyright © 2006, 2007 by Oracle.

7.1 About this content

Terms and conditions regarding the use of this guide.

May 5, 2007

License

The Eclipse Foundation makes available all content in this plug-in ("Content"). Unless otherwise indicated below, the Content is provided to you under the terms and conditions of the Eclipse Public License Version 1.0 ("EPL"). A copy of the EPL is available at http://www.eclipse.org/legal/epl-v10.html. For purposes of the EPL, "Program" will mean the Content.

If you did not receive this Content directly from the Eclipse Foundation, the Content is being redistributed by another party ("Redistributor") and different terms and conditions may apply to your use of any object code in the Content. Check the Redistributor's license that was provided with the Content. If no such license exists, contact the Redistributor. Unless otherwise indicated below, the terms and conditions of the EPL still apply to any source code in the Content and such source code may be obtained at http://www.eclipse.org.
Annotations
@Basic, 3-11
@Column, 4-5
@DiscriminatorColumn, 3-10
@DiscriminatorValue, 3-10
@Embeddable, 3-7
@Embedded, 3-12
@EmbeddedId, 3-13
@Entity, 3-6
@Enumerated, 4-5
@GeneratedValue, 4-7
@Id, 3-13
@Inheritance, 3-8, 3-9
@JoinColumn, 3-16, 3-18, 4-6, 4-7
@Lob, 4-5
@ManyToMany, 3-15
@ManyToOne, 3-16
@MappedSuperclass, 3-8
@OneToMany, 3-17
@OneToOne, 3-18
@OrderBy, 4-6
@SequenceGenerator, 4-8
@Temporal, 4-5
@Transient, 3-19
@Version, 3-19

A
annotations. See specific annotation.
architecture of Dali feature, 4-13
attributes
  JPA Details view, 4-4
  mapping, 2-1

B
basic mapping
  @Basic, 3-11
  about, 3-11
  See also mappings

C
classes
  adding persistence to, 3-6
  embeddable, 3-7

D
database tables
  generating entities from, 3-20
database, persistence
  connection, 4-10
  schema, 4-10
developer documentation, Dali, 4-13

E
eager fetch, 4-5
EJB. see persistent entities
embeddable class
  @Embeddable, 3-7
    about, 3-7
embedded ID mapping
  @EmbeddedId, 3-13
    about, 3-13
embedded mapping
  @Embedded, 3-12
    about, 3-12
entities
  @Entity annotation, 3-6
    about, 2-1
  embeddable, 3-7
  from tables, 3-20, 4-10
  JPA Details view, 4-3
  mapped superclass, 3-8
  mapping, 1-6
  persistence, 1-3
  persistent, 3-6
  secondary tables, 4-3
  @Enumerated, 4-5
  enumerated, 4-5
  error messages, Dali, 3-21
  extension points, Dali feature, 4-13
**F**
fetch type, 4-5

**G**
Generate Entities from Tables dialog, 3-20, 4-10
generated values
   ID mappings, 4-7
   sequence, 4-8

**I**
ID mapping
   @Id, 3-13
   about, 3-13
inheritance
   entity, 3-8, 3-9, 4-3
   joined tables, 3-11
   single table, 3-10
Inheritance, in Java Details view, 4-3
installation, Dali, 1-1

**J**
joined tables, inheritance, 3-11
JPA Details view
   attributes, 4-4
   entities, 4-3
JPA Development perspective, 4-11
JPA Facet dialog, 3-3
JPA project
   creating new, 3-1
   platform, 4-10
JPA Project dialog, 3-2
JPA Structure view, 4-9

**L**
lazy fetch, 4-5

**M**
many-to-many mapping
   @ManyToMany, 3-15
   about, 3-15
many-to-one mapping
   @ManyToOne, 3-16
   about, 3-16
mapped superclass
   @MappedSuperclass, 3-8
   about, 3-8
mapping entities, 1-6
mappings
   about, 2-1
   basic, 3-11
   embedded, 3-12
   embedded ID, 3-13
   ID, 3-13
   many-to-many, 3-15
   many-to-one, 3-16
one-to-many, 3-17
one-to-one, 3-18
problems, 3-21
transient, 3-19
version, 3-19

**N**
New JPA Project wizard, 3-1
nonpersistent
   classes, 3-6
   fields. See transient

**O**
one-to-many mapping
   @OneToMany, 3-17
   about, 3-17
one-to-one mapping
   @OneToOne, 3-18
   about, 3-18
OR (object-relational) mappings. See mappings
   @OrderBy, 4-6
ordering, 4-6
orm.xml file
   about, 2-2
   managing, 3-5
   sample, 3-5
outline, persistence. See JPA Structure view

**P**
persistence
   about, 2-1
   database connection, 4-10
   database schema, 4-10
   entity class, 3-6
   options, 4-9
Persistence XML Editor, 3-4
persistence.xml file
   about, 2-2
   editor, 3-4
   managing, 3-3, 3-4, 3-5
   sample, 3-3
   synchronizing with classes, 3-4
persistent entity, 3-6
perspective, JPA Development, 4-11
platform, JPA, 4-10
problems, 3-21
projects, JPA
   creating new, 1-2, 3-1
   options, 4-9

**Q**
quick start, Dali, 1-1

**R**
requirements
   Dali Java Persistence Tools, 1-1
persistent entities, 3-6

S
schema, database, 4-10
secondary tables, 4-3
Secondary Tables, in Java Details view, 4-3
single table inheritance, 3-10
superclass, 3-8

T
tables
creating entities from, 3-20, 4-10
inheritance, 3-10
secondary, 4-3
@Temporal, 4-5
temporal, 4-5
transient mapping
@Transient, 3-19
about, 3-19
tutorial, Dali, 1-7

V
version mapping
@Version, 3-19
about, 3-19
views
JPA Details view, 4-3, 4-4
JPA Structure view, 4-9

W
warning messages, Dali, 3-21
wizards
New JPA Project wizard, 3-1

X
XML editor, 3-4, 3-5