

# **Dali Object-Relational Mapping Tool**

User Guide

Release 0.5.0 for Eclipse

June 2006

## Dali Object-Relational Mapping Tool User Guide

Copyright © 2006, Oracle. All rights reserved.

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.

---

---

# 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 .....	1-4
1.2.3	Mapping an entity .....	1-6
1.3	Dali basic tutorial .....	1-8
1.3.1	Generate the tutorial database schema .....	1-8
1.3.1.1	Create a database connection .....	1-9
1.3.2	Create a Java project .....	1-9
1.3.2.1	Add persistence to the project .....	1-10
1.3.3	Create persistent Java entities .....	1-11
1.3.3.1	Add fields to the entities .....	1-12
1.3.3.2	Associate the entity with a database table .....	1-12
1.3.4	Create OR mappings .....	1-13
1.3.4.1	Create ID mappings .....	1-13
1.3.4.2	Create basic mappings .....	1-15
1.3.4.3	Create one-to-one mappings .....	1-16
1.3.4.4	Create one-to-many mappings .....	1-18
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 JSR220: EJB 3.0 .....	2-2
2.3.1	The persistence.xml file .....	2-2

## 3 Tasks

3.1	Adding persistence to a Java project .....	3-1
3.2	Managing the persistence.xml file .....	3-2
3.2.1	Working with persistence.xml file .....	3-4
3.2.2	Synchronizing classes .....	3-5
3.3	Adding persistence to a class .....	3-5
3.3.1	Persistent entity .....	3-5

3.3.2	Embeddable .....	3-6
3.3.3	Mapped superclass .....	3-7
3.4	Creating a new Java persistent entity .....	3-8
3.5	Specifying entity inheritance .....	3-9
3.6	Mapping an entity .....	3-10
3.6.1	Basic mapping .....	3-11
3.6.2	Embedded mapping .....	3-12
3.6.3	Embedded ID mapping .....	3-13
3.6.4	ID mapping .....	3-13
3.6.5	Many-to-many mapping .....	3-15
3.6.6	Many-to-one mapping .....	3-16
3.6.7	One-to-many mapping .....	3-17
3.6.8	One-to-one mapping .....	3-18
3.6.9	Transient mapping .....	3-19
3.6.10	Version mapping .....	3-20
3.7	Generating entities from tables .....	3-20
3.8	Generating tables (DDL scripts) from entities .....	3-21
3.9	Validating mappings and reporting problems .....	3-22
3.9.1	Error messages .....	3-22
3.10	Modifying persistent project properties .....	3-24

## 4 Reference

4.1	Wizards .....	4-1
4.1.1	Generate Database DDL from Entities wizard .....	4-1
4.2	Property pages .....	4-1
4.2.1	Persistence Properties view (for entities) .....	4-2
4.2.1.1	General tab .....	4-2
4.2.1.2	Inheritance tab .....	4-2
4.2.2	Persistence Properties view (for attributes) .....	4-3
4.2.2.1	General tab .....	4-3
4.2.2.2	Join Table tab .....	4-5
4.2.2.3	Join Columns tab .....	4-5
4.2.2.4	PK Generation tab .....	4-5
4.2.3	Persistence Outline view .....	4-6
4.3	Preferences .....	4-7
4.3.1	Project Properties page – Persistence Options .....	4-7
4.4	Dialogs .....	4-7
4.4.1	Add Persistence dialog .....	4-7
4.4.2	Generate Entities from Tables dialog .....	4-8
4.4.3	Edit Join Columns Dialog .....	4-8
4.5	Persistence perspective .....	4-9
4.6	Icons and buttons .....	4-9
4.6.1	Icons .....	4-10
4.6.2	Buttons .....	4-10
4.7	Dali Developer Documentation .....	4-11

## 5 Tips and tricks

## 6 What's new

6.1	Generate Persistent Entities from Tables wizard .....	6-1
6.2	Generate DDL from Entities wizard .....	6-1
6.3	Create and Manage the persistence.xml file .....	6-2

## 7 Legal

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

## Index



---

---

# Getting started

This section provides information on getting started with the Dali OR (object-relational) mapping tool.

- [Requirements and installation](#)
- [Dali quick start](#)
- [Dali basic tutorial](#)

For additional information, please visit the Dali home page at: <http://www.eclipse.org/dali>.

## 1.1 Requirements and installation

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

- Eclipse 3.2 (<http://www.eclipse.org/downloads>)
- Java Runtime Environment (JRE) 1.5 (<http://java.com>)
- Eclipse Web Tools Platform (WTP) 1.5 (<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/dali/gettingstarted\\_main.html](http://www.eclipse.org/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](#)

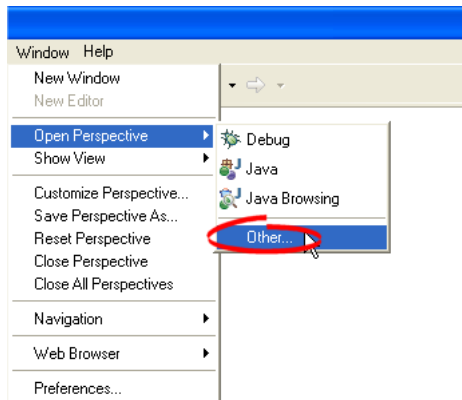
The [Dali basic tutorial](#) contains detailed procedures for building you first Dali project.

## 1.2.1 Creating a new project

This quick start shows how to create a new Java Persistence Entity project. Before creating the project, you should open the new Persistence perspective.

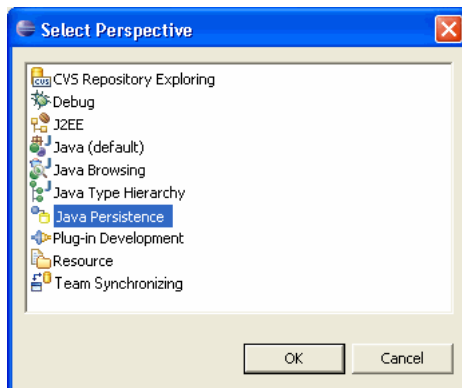
1. Switch to the new **Persistence perspective** to begin working with Java persistence entities. Select **Window > Open Perspective > Other**. The **Select Perspective Dialog** appears.

**Figure 1–1 Selecting Persistence Perspective**



2. On the Select Perspective dialog, select **Java Persistence** and click **OK**. The workbench adds the Persistence Outline and Persistence Properties views.

**Figure 1–2 Select Perspective Dialog**



Now, we will create a new Java project.

1. Select **File > New > Project**. The New Project dialog appears.
2. On the New Project dialog, select **Java > Java Project** and click **Next**. The Create a Java Project dialog appears.
3. On the Create a Java Project dialog, enter a **Project name** (such as `QuickStart`) and click **Next**. The Java Settings page appears.

---



---

**Note:** You must configure your project's JRE to use version 1.5 (or higher). See "[Requirements and installation](#)" on page 1-1 for more information.

---



---



4. On the Java Settings page click **Finish**. Eclipse adds the project to the workbench and opens the Java perspective.

Finally, we will "add persistence" to the project.

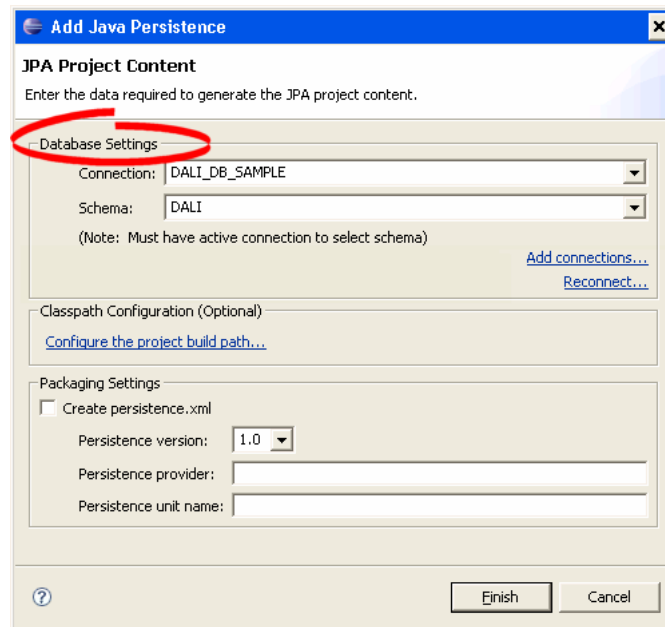
1. Right-click the Java project in the Explorer and select **Java Persistence > Add Java Persistence**. The [Add Java Persistence Dialog](#) appears.
2. On the [Add Persistence dialog](#) select a database connection (or click **Add Connection** to create a new connection).

---

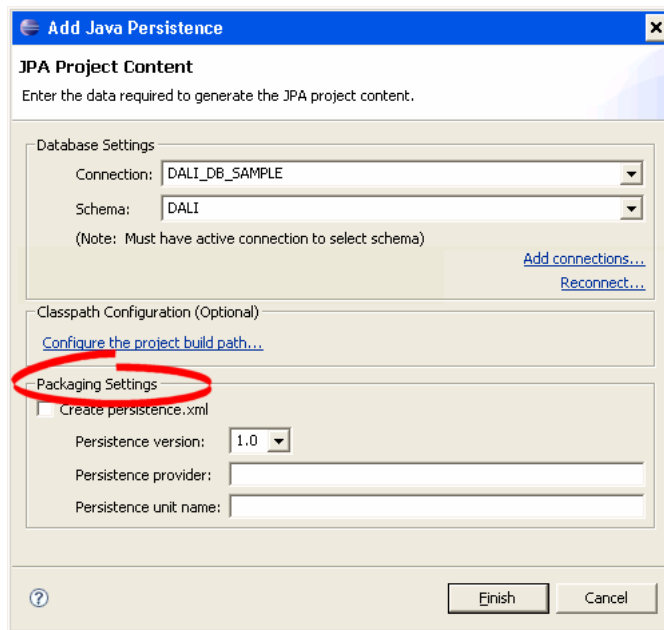
**Note:** You must be connected to the database before adding persistence to the project. You will also need to create a table named ADDRESS (you will add its columns later). Click **Reconnect** to reconnect to an existing database.

---

**Figure 1–3 Add Java Persistence Dialog**

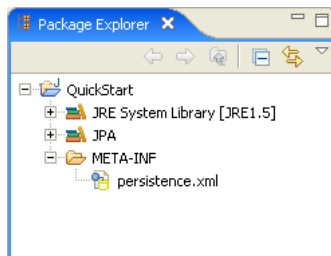


3. Click **Configure the project build path** to add the library or JARs that contain the Java Persistence API (JPA) and entities to the project's Java Build Path.
4. In the Packaging Settings area, enter the necessary information for your persistence provider.

**Figure 1–4 Add Java Persistence Dialog**

For information on using a persistence.xml file for packaging your project, see ["Managing the persistence.xml file"](#) on page 3-2.

5. Click **Finish**. Eclipse adds the persistence information to the project. You can now work in the [Persistence perspective](#).

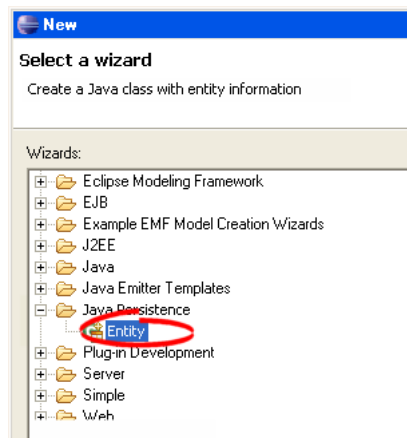
**Figure 1–5 Project in Package Explorer**

Now that you have created a project with persistence, you can continue with [Creating a Java persistent entity](#).

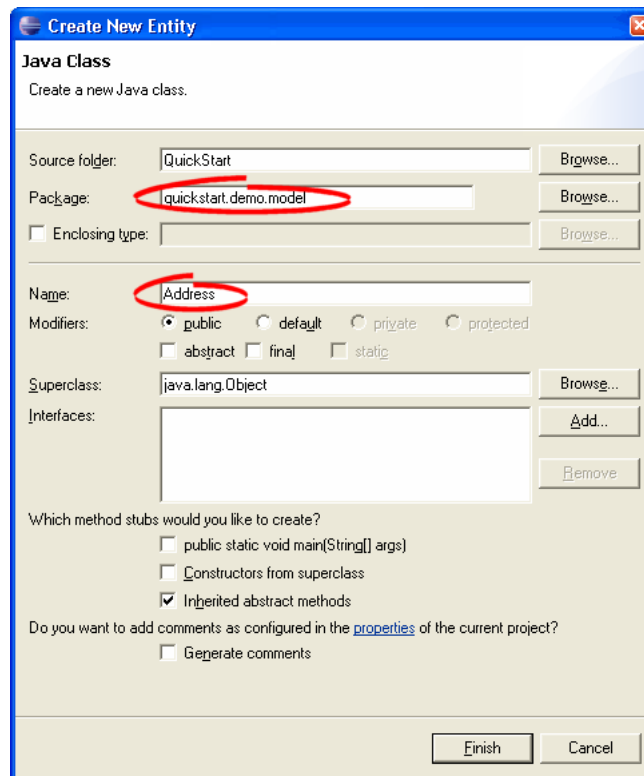
## 1.2.2 Creating a Java persistent entity

This quick start shows how to create a new Java entity with persistence. We will create an entity to associate with a database table.

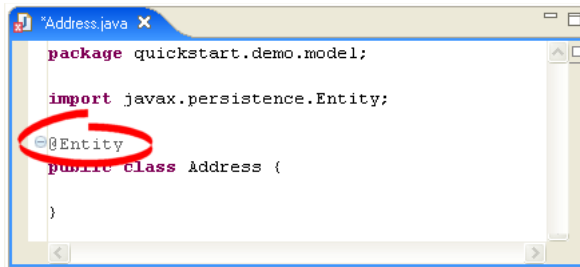
1. Right-click the project in the Package Explorer and select **New > Other**. The Select a Wizard dialog appears.
2. In the Select a Wizard dialog, select **Java Persistence > Entity** and click **Next**. The Java Class page of the Create New Entity wizard appears.

**Figure 1–6 Selecting the Java Persistence Entity Wizard**

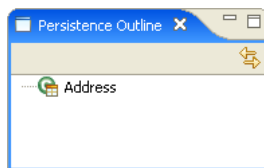
3. On the Java Class page, enter a package name (such as `quickstart.demo.model`), class name (such as `Address`), and click **Finish**.

**Figure 1–7 Creating a Java Class**

Eclipse adds the new entity to the project and adds the `@Entity` annotation to the class.

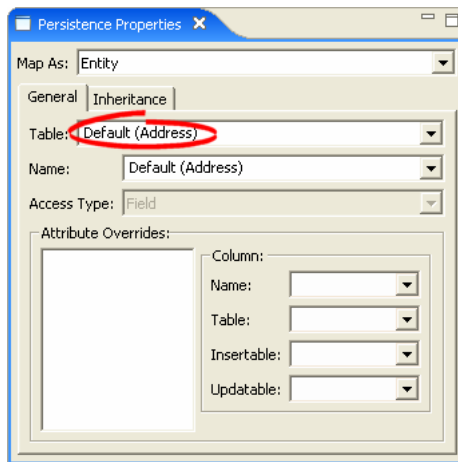
**Figure 1–8 Address Entity**

Eclipse also displays the **Address** entity in the Persistence Outline view:

**Figure 1–9 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.

**Figure 1–10 Persistence Properties View for Address Entity**

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:

```

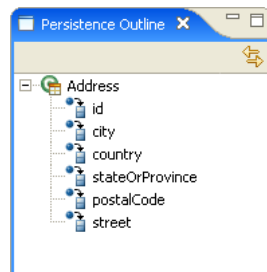
private Long id;
private String city;

```

```
private String country;
private String stateOrProvince;
private String postalCode;
private String street;
```

Eclipse updates the Address entity in the Persistence Outline view to show its fields:

**Figure 1–11 Address Entity and Fields**



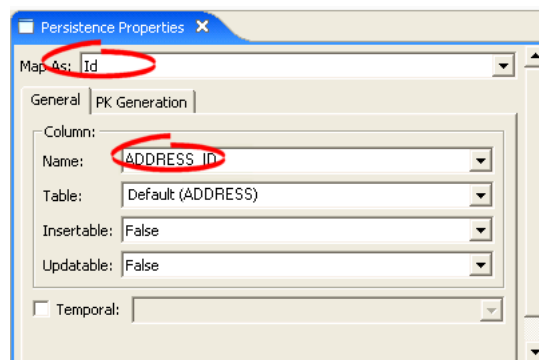
You will also need to add the following columns to the ADDRESS database table:

```
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 fields in the Address class to a column in the database table.

1. Select the **id** field in the Persistence Outline view.
2. In the Persistence Properties view:
  - For the Map As field, select **ID**
  - For the Column field, select **ADDRESS\_ID**.

**Figure 1–12 Persistence Properties View for addressId Field**



Eclipse adds the following annotations to the Address entity:

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

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

Field	Map As	Database Column
city	Basic	CITY
country	Basic	COUNTRY
postalCode	Basic	P_CODE
provinceOrState	Basic	PROVINCE
street	Basic	STREET

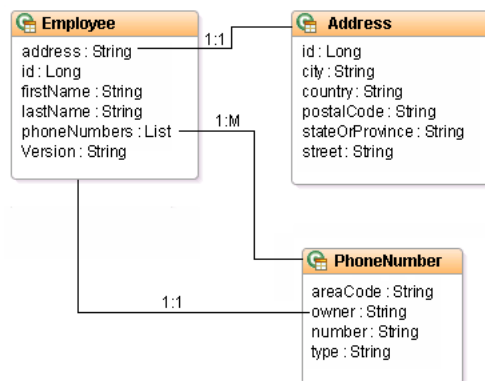
Notice that Dali will automatically map some fields to the correct database column (such as the **city** field to the **CITY** column) if the names are identical.

Refer to the [Dali basic tutorial](#) to map a complete object model using basic and relational mappings.

## 1.3 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 1–13](#) illustrates the object model for the tutorial.

**Figure 1–13 Tutorial Object Model**



### 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/dali/docs/dbscripts.zip>.

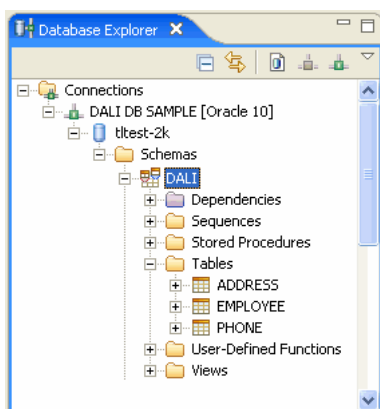
**Table 1–1 Tutorial Database Schema**

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

### 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 1–14 Database Explorer**

Now you're ready to [Create a Java project](#).

## 1.3.2 Create a Java 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 **Java > Java Project** and click **OK**. The Create a Java Project dialog appears.
3. On the Create a Java Project dialog, enter `Employee` as the **Project name** and click **Finish**. Eclipse adds the project to the workbench and opens the Java perspective.

### 1.3.2.1 Add persistence to the project

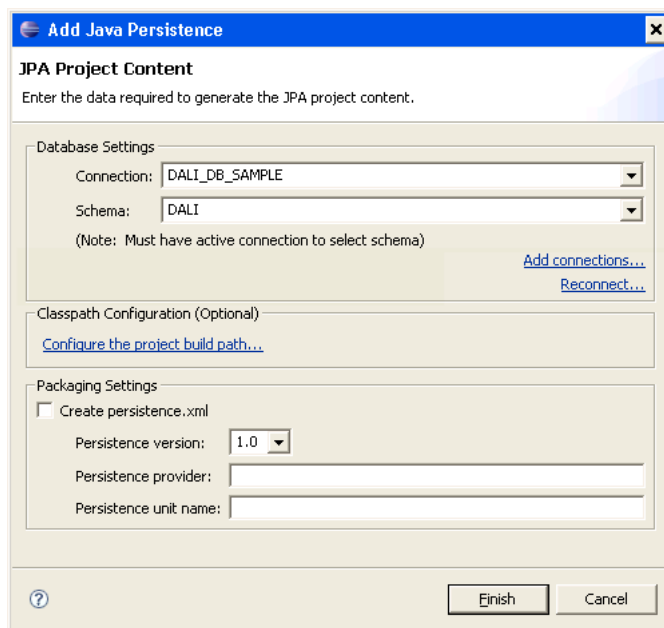
Use this procedure to add persistence to the **Employee** project.

1. Right-click the **Employee** project in the Explorer and select **Java Persistence > Add Java Persistence**. The [Add Java Persistence Dialog](#) appears.
2. In the Database Settings area select a database connection that you created earlier (see "[Create a database connection](#)").

You must be connected to the database before adding persistence to the project.

3. Click **Configure the project build path** to add the library or JARs that contain the Java Persistence API (JPA) and entities to the project's Java Build Path.
4. In the Packaging Settings area select the **Create persistence.xml** option and complete the following fields:
  - Persistence Version: **1.0**
  - Persistence Provider: Enter the name of the JPA provider that you selected in step 3 (such as **TopLink Essentials**).
  - Persistence Unit Name: **Dali Tutorial**

**Figure 1–15 Add Persistence Dialog**

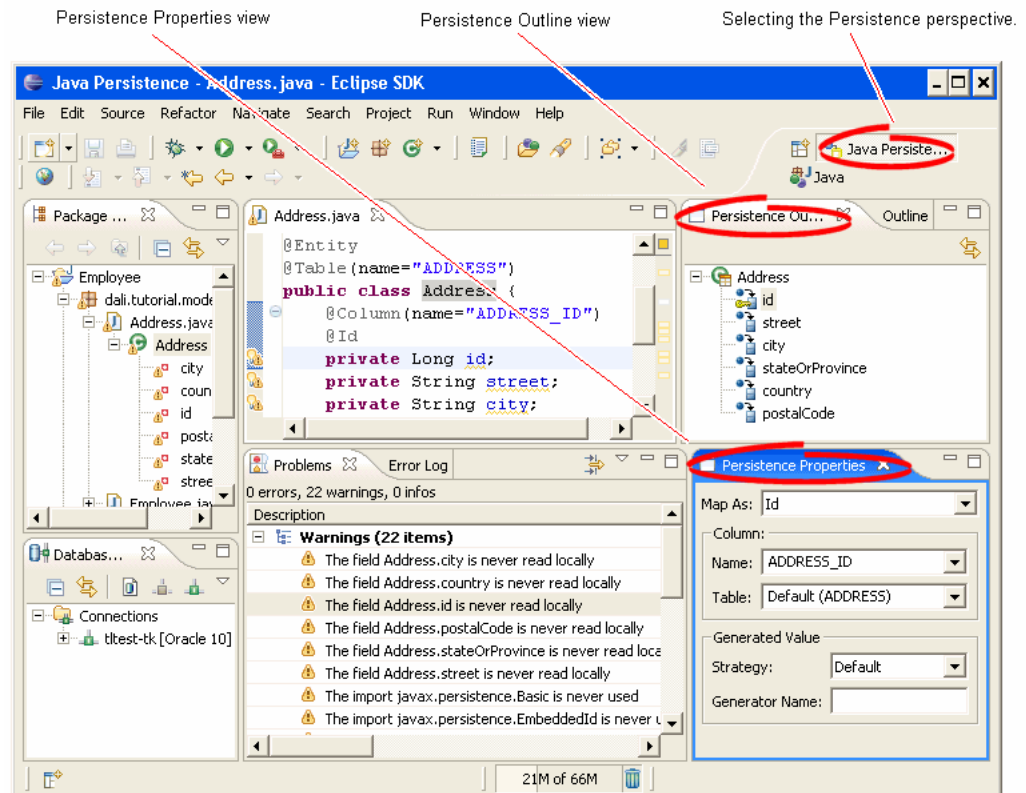


5. Click **Finish**. You can now work in the [Persistence perspective](#).
6. Select **Window > Open Perspective > Other**. The Select Perspective dialog appears.



7. Select **Persistence** and click **OK**. The Persistence Outline and Properties views appear.

**Figure 1–16 Persistence 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 > Other**. The Select a Wizard dialog appears.
2. In the Select a Wizard dialog, select **Java Persistence > Entity** and click **Next**. The Java Class page of the Create New Java Persistence Entity wizard appears.
3. On the Java Class page, 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 and 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:

```
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:

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

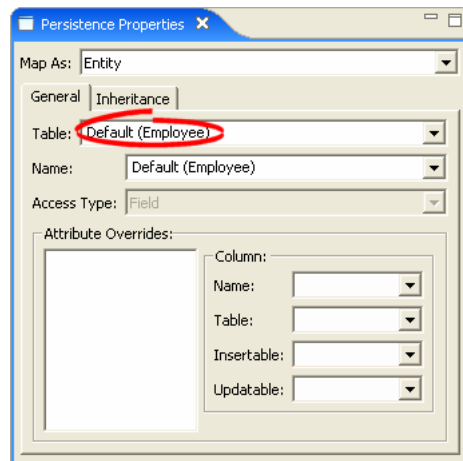
5. Add the following fields to the **PhoneNumber** entity:

```
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 Persistence Properties view, select **General** tab.
3. On the General tab, notice that Dali has automatically selected the EMPLOYEE table as the table name.

**Figure 1–17 Persistence Properties View for the Employee Entity**

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:

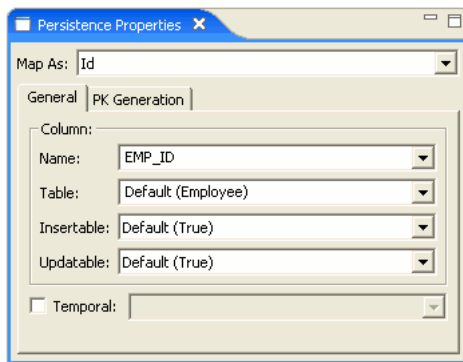
- ID mappings
- Basic mappings
- One-to-one mappings
- Many-to-one mappings
- One-to-many mappings
- Version mappings

#### 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 [Persistence Outline view](#) and select the **id** field. The [Persistence Properties view \(for attributes\)](#) displays the properties for the field.
3. In the Map As field, select **ID**.

**Figure 1–18 ID Mapping for emp\_id Field**

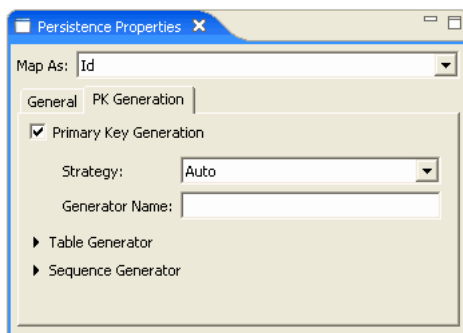


- Use this table to complete the remaining fields on the General tab in the Persistence Properties view.

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

- Leave all other fields on the tab as their defaults. Click the **PK Generation** tab.

**Figure 1–19 Primary Key Generation for emp\_id Field**

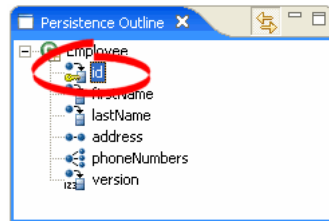


- Use this table to complete the remaining fields on the General tab in the Persistence Properties view.

Property	Description
Generated Value	These fields define how the primary key is generated.
Strategy	For the tutorial project, use the <b>Auto</b> option.
Generator Name	Leave this field blank.

In the Persistence Outline, the `id` field is identified as the primary key by the following icon:

**Figure 1–20 Persistence Outline for Employee Entity**



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

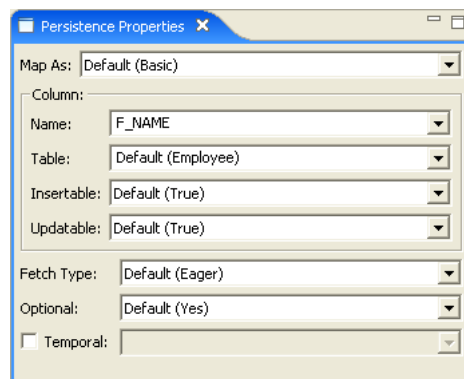
- The `id` field of the **Address** entity to the `ADDRESS_ID` column of the `ADDRESS` table.
- The `number` field of the **PhoneNumber** entity to the `P_NUMBER` column of the `PHONE` table.

### 1.3.4.2 Create basic mappings

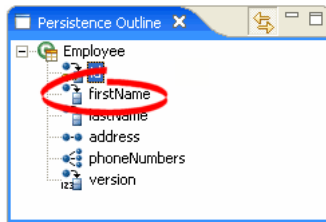
Use a **Basic Mapping** to map an attribute directly to a database column. In the [Tutorial Object Model](#), the `firstName` field of the **Employee** class maps directly to the `F_NAME` column of the `EMPLOYEE` database table.

1. Select the **Employee** entity in the Package Explorer view.
2. In the [Persistence Outline view](#), select the `firstName` field of the **Employee** entity. The [Persistence Properties view \(for attributes\)](#) displays the properties for the field.
3. In the Map As field, select **Basic**. In the Column field, select `F_NAME`.

**Figure 1–21 Basic Mapping for firstName**



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

**Figure 1–22 Persistence Outline 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

---



---

**Note:** Because the **type** field is named identically to its database column, Dali automatically maps the field. No annotation is required.

---

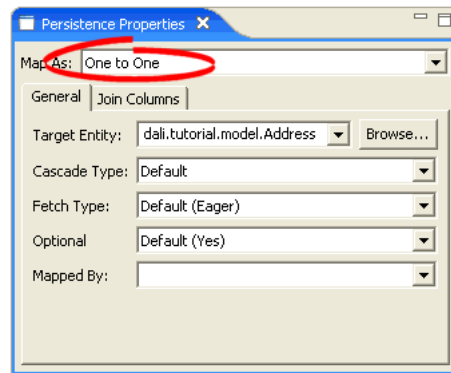


---

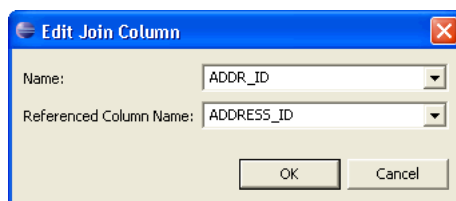
### 1.3.4.3 Create one-to-one mappings

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.

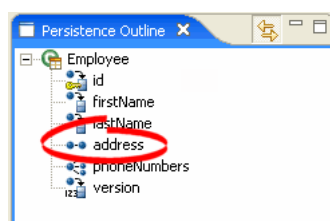
1. Select the **Employee** entity in the Package Explorer view.
2. In the [Persistence Outline view](#), select the **address** field of the **Employee** entity. The [Persistence Properties view \(for attributes\)](#) displays the properties for the field.
3. In the Map As field, select **One-to-One**.

**Figure 1–23 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 **Join Columns** tab specifies 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 **Override Default** option.
7. Select the **address\_ADDRESS\_ID -> ADDRESS\_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: **ADDR\_ID** (from the EMPLOYEE table)
  - Referenced Column Name: **ADDRESS\_ID** (from the ADDRESS table)

**Figure 1–24 Editing Join Column for Address Mapping**

In the Persistence Outline, the **address** field is identified as a one-to-one mapping, as shown in the following figure:

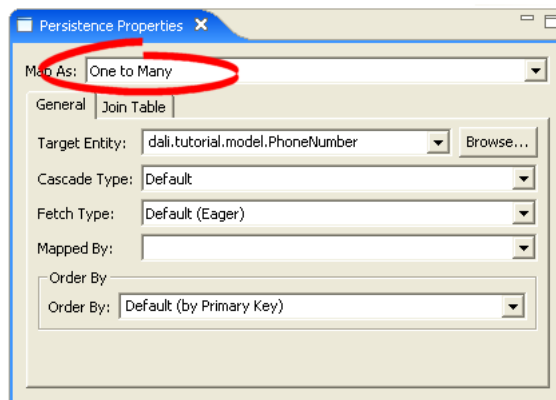
**Figure 1–25 Persistence Outline for Employee Entity**

### 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 [Persistence Outline view](#), select the **phoneNumber** field of the **Employee** entity. The [Persistence Properties view \(for attributes\)](#) displays the properties for the field.
3. In the Map As field, select **One-to-Many**.

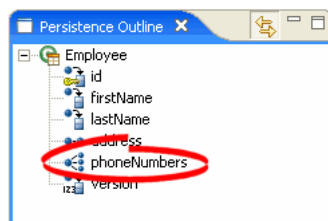
**Figure 1–26 One-to-many Mapping for phoneNumbers**



4. On the General tab, select **PhoneNumber** as the Target Entity. Leave the other fields with their default values.
5. On the Join Table tab, notice that Dali has selected the correct joins, based on the foreign key associations in the database tables.

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

**Figure 1–27 Persistence Outline 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.

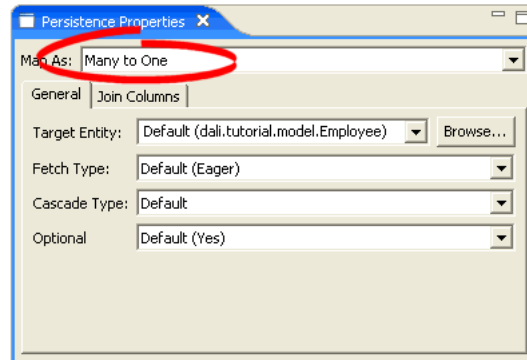
This is the "back mapping" of the one-to-many mapping you previously defined.

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



- In the **Persistence Outline** view, select the **owner** field of the **PhoneNumber** entity. The **Persistence Properties** view (for attributes) displays the properties for the field.
- In the **Map As** field, select **Many to One**.

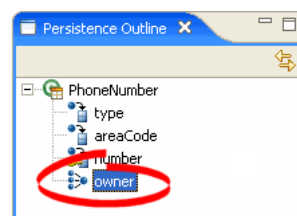
**Figure 1–28 Many to One Mapping for owner**



- Leave the other fields with their default values. Dali correctly completes the information based on the database structure and previously defined mappings.
- Select the **Join Columns** tab specifies 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.
- Select the **Override Default** option.
- Select the **owner\_EMP\_ID -> EMP\_ID** relationship in the **Join Columns** area and click **Edit**.
- 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 **Persistence Outline**, the **owner** field is identified as a many-to-one mapping as shown in the following figure:

**Figure 1–29 Persistence Outline for PhoneNumber Entity**



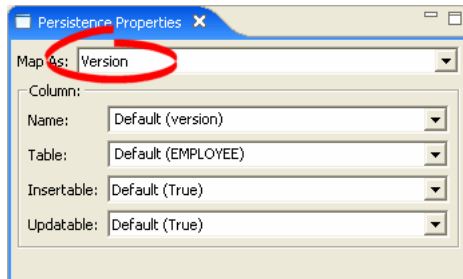
### 1.3.4.6 Create version mappings

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

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

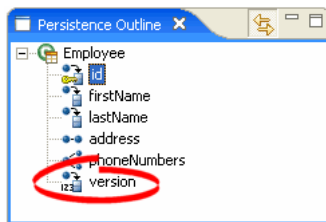
- In the **Persistence Outline view**, select the **version** field of the **Employee** entity. The **Persistence Properties view (for attributes)** displays the properties for the field.
- In the Map As field, select **Version**.

**Figure 1–30 Version Mapping for version**



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

**Figure 1–31 Persistence Outline 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 JSR220: EJB 3.0](#)

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

- Eclipse Dali project: <http://www.eclipse.org/dali>
- Eclipse Web Tools Platform project: <http://www.eclipse.org/webtools>
- JSR 220 EJB 3.0 specification: <http://www.jcp.org/en/jsr/detail?id=220>

## 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 JSR220: EJB 3.0

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 purpose of EJB 3.0 (JSR220) 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 EJB 3.0 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 includes a Persistence XML Editor to help create and maintain this information. See "[Managing the persistence.xml file](#)" on page 3-2 for more information.

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

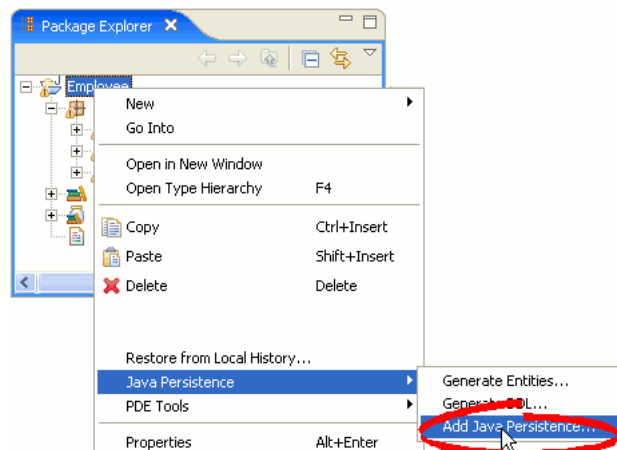
- [Adding persistence to a Java project](#)
- [Managing the persistence.xml file](#)
- [Adding persistence to a class](#)
- [Creating a new Java persistent entity](#)
- [Specifying entity inheritance](#)
- [Mapping an entity](#)
- [Generating entities from tables](#)
- [Generating tables \(DDL scripts\) from entities](#)
- [Validating mappings and reporting problems](#)
- [Modifying persistent project properties](#)

### 3.1 Adding persistence to a Java project

Use this procedure to add persistence to an existing project:

1. Right-click the project in the Package Explorer and select **Java Persistence > Add Java Persistence**.

**Figure 3-1** Adding Persistence to a Project



- Complete the fields in the [Add Persistence dialog](#) to select a database connection and persistence packaging options.

- To create a new database connection, click **Add Connections** use the New Connection wizard. To reconnect to an existing connection, click **Reconnect**.

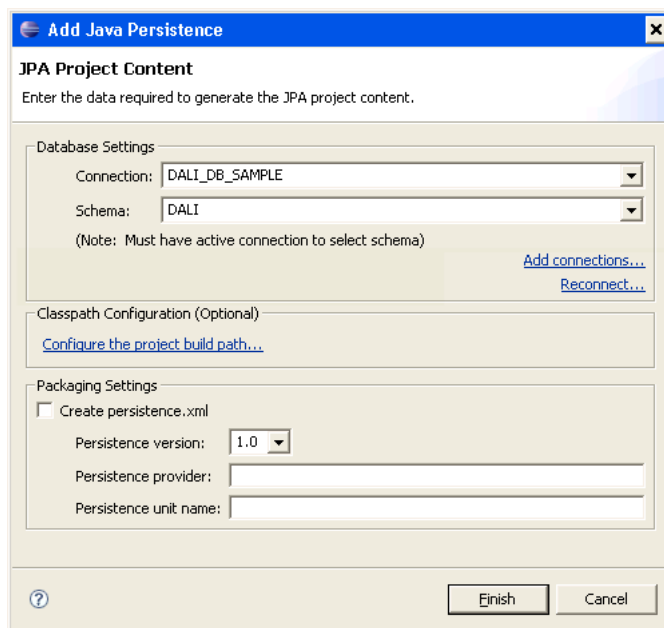
---

**Note:** You must have a defined database connection (and be connected) to add persistence to a project.

---

- To add the libraries that contain the JPA (Java Persistence API), click **Configure the project build path** and use the Java Build Path page of the project's Properties dialog.
- To package your persistent project, select the **Create persistence.xml** option and complete the fields in the **Packaging Settings** area. See "[Managing the persistence.xml file](#)" on page 3-2 for more information.

**Figure 3–2 Add Java Persistence Dialog**



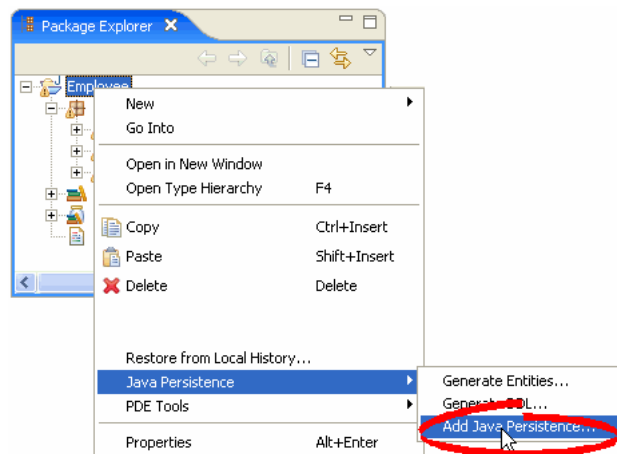
- Click **Finish**. You should now open the [Persistence perspective](#).

## 3.2 Managing the persistence.xml file

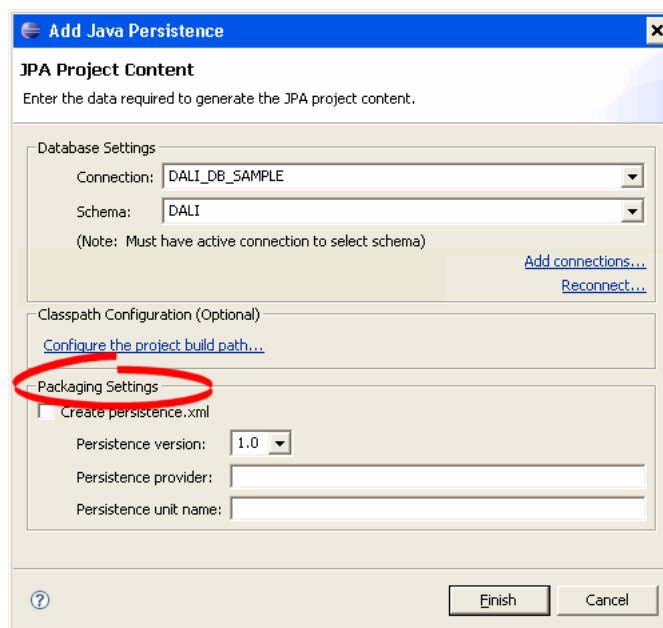
When adding persistence to a project (see "[Adding persistence to a Java project](#)") you can also create the `persistence.xml` file. Complete the **Packaging Settings** fields on the [Add Persistence dialog](#).

Use this procedure to define the `persistence.xml` file:

- Right-click the project in the Package Explorer and select **Java Persistence > Add Java Persistence**.

**Figure 3–3 Adding Persistence to a Project**

2. Complete the **Database Settings** fields on the [Add Persistence dialog](#) to select a database connection to use with the persistent project.
3. Select the **Create persistence.xml** option and complete the fields in the **Packaging Settings** area to create the `persistence.xml` file.
  - Persistence Version – Persistence version to use.
  - Persistence Provider – Name of the persistence provider's `javax.persistence.spi.PersistenceProvider` class.
  - Persistence Unit Name – Name of the persistence unit.

**Figure 3–4 Add Java Persistence Dialog**

4. Click **Finish**. You should now open the [Persistence perspective](#).

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

```
<?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>">
    <provider="<PERSISTENCE_PROVIDER>" />
  </persistence-unit>
</persistence>
```

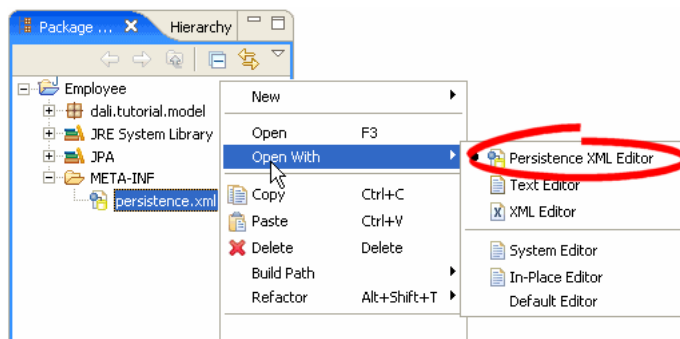
### 3.2.1 Working with persistence.xml file

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

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

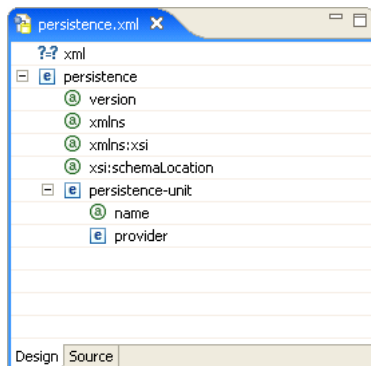
1. Right-click the persistence.xml file in the Package Explorer and select **Open With > Persistence XML Editor**.

**Figure 3–5 Opening the Persistence XML Editor**



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

**Figure 3–6 Persistence XML Editor**





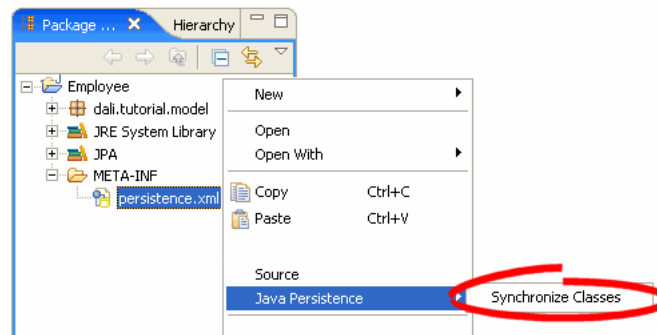
## 3.2.2 Synchronizing classes

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 **Java Persistence > Synchronize Classes**.

**Figure 3-7 Synchronizing the persistence.xml File**



Dali adds the necessary `<class>` elements to the `persistence.xml` file.

2. Use the Persistence XML Editor to continue editing the `persistence.xml` file.

## 3.3 Adding persistence to a class

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

- [Persistent entity](#)
- [Embeddable](#)
- [Mapped superclass](#)

You can also add persistence when creating a new Java class. See "[Creating a new Java persistent entity](#)" on page 3-8 for more information.

### 3.3.1 Persistent 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*:

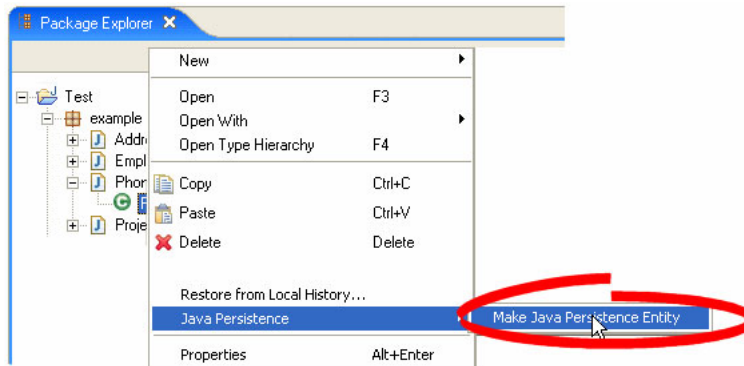
- A no-arg constructor (public or protected); the entity class may have other constructors as well.

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:

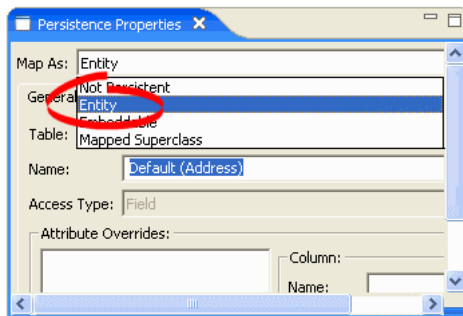
1. Right-click the class in the Package Explorer and select **Java Persistence > Make Java Persistence Entity**.

**Figure 3–8 Adding Persistence to a Class**



2. In the Persistence Properties view, use the **Map As** drop-list to select **Entity**.

**Figure 3–9 Selecting Entity Persistence**



3. Complete the remaining [Persistence Properties view \(for entities\)](#).

### 3.3.2 Embeddable

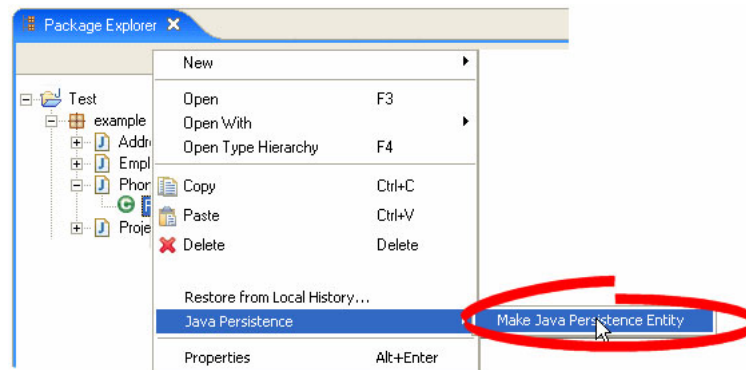
An **Embedded** class is a class whose instances are stored as part of an owning entity; it shares the identity of the owning entity. Each field of the embedded class is mapped to the database table associated with the owning entity.

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

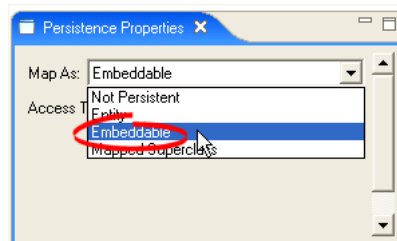
An embeddable entity is identified by the `@Embeddable` annotation.

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

1. Right-click the class in the Package Explorer and select **Java Persistence > Make Java Persistence Entity**.

**Figure 3–10 Adding Persistence to a Class**

2. In the Persistence Properties view, use the **Map As** drop-list to select **Embeddable**.

**Figure 3–11 Selecting Embeddable Persistence**

3. Complete the remaining [Persistence Properties view \(for entities\)](#).

### 3.3.3 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

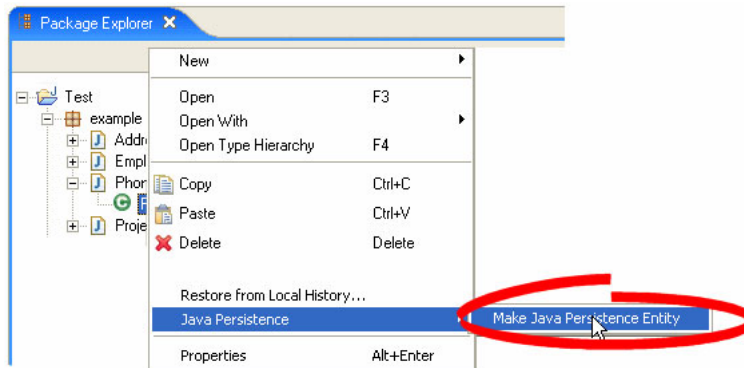
A mapped superclass does not have a defined database table. Instead, its mapping information is derived from its superclass. To override the mapping information for a specific subclass, use the `@AttributeOverride` annotation for that specific class.

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

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

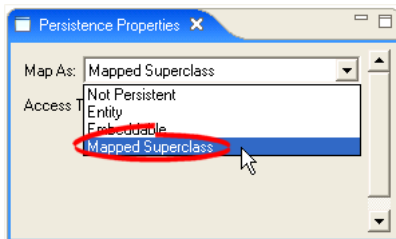
1. Right-click the class in the Package Explorer and select **Persistence > Make Java Persistence Entity**.

**Figure 3–12 Adding Persistence to a Class**



2. In the Persistence Properties view, use the **Map As** drop-list to select **Mapped Superclass**.

**Figure 3–13 Selecting Mapped Superclass Persistence**



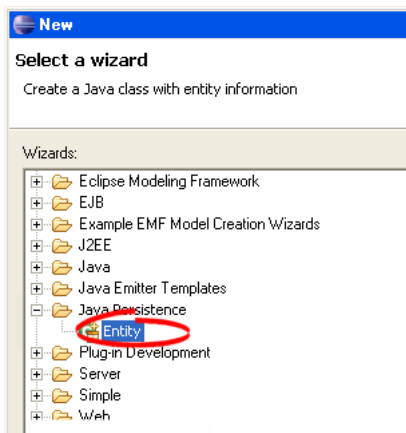
3. Complete the remaining [Persistence Properties view \(for entities\)](#).

### 3.4 Creating a new Java persistent entity

Use this procedure to create a new persistent entity by using the Java Persistent Entity wizard.

1. Right-click the project in the Package Explorer and select **New > Other**.
2. In the Select a Wizard dialog, select **Java Persistence > Entity** and click Next.

**Figure 3–14 Selecting the Java Persistence Entity Wizard**



3. On the Java Class page, complete the information for the new class, and click **Next**.

Eclipse adds the new persistent entity to the project. Use the [Persistence Properties view \(for entities\)](#) to further define the entity.

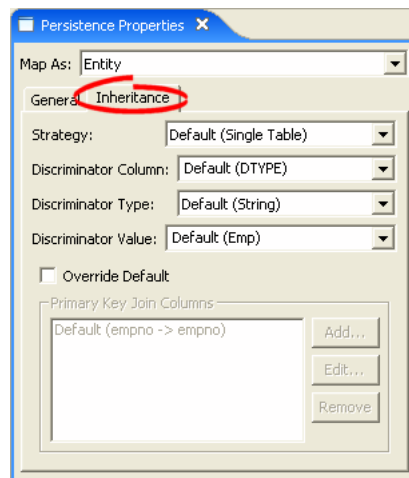
## 3.5 Specifying entity inheritance

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 Persistence Properties view, click the **Inheritance** tab.

**Figure 3–15** *Specifying Inheritance*



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 tab](#)" on page 4-2 for additional details.

Property	Description	Default
Discriminator Column	Name of the discriminator column when using a <b>Single</b> or <b>Joined</b> inheritance strategy.  This field corresponds to the <code>@DiscriminatorColumn</code> annotation.	
Discriminator Type	Set the discriminator type to <code>Char</code> or <code>Integer</code> (instead of its default: <code>String</code> ). The <b>Discriminator Value</b> must conform to this type.	String

Property	Description	Default
Discriminator Value	Specify the discriminator value used to differentiate an entity in this inheritance hierarchy. The value must conform to the specified <b>Discriminator Type</b> .  This field corresponds to the <code>@DiscriminatorValue</code> annotation.	
Override Default	Use this field to specify custom primary key join columns.  This field corresponds to the <code>@PrimaryKeyJoinColumn</code> annotation.	

Eclipse adds the following annotations the entity field:

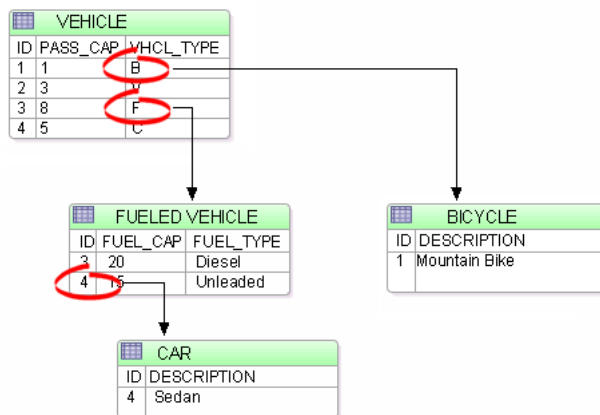
```
@Inheritance(strategy=InheritanceType.<INHERITANCE_STRATEGY>)
@DiscriminatorColumn(name="<DISCRIMINATOR_COLUMN>",
    discriminatorType=<DISCRIMINATOR_TYPE>)
@DiscriminatorValue(value="<DISCRIMINATOR_VALUE>")
@PrimaryKeyJoinColumn(name="<JOIN_COLUMN_NAME>",
    referencedColumnName = "<REFERENCED_COLUMN_NAME>")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

The following figures illustrates the different inheritance strategies.

**Figure 3–16 Single Table Inheritance**

VEHICLE						
ID	PASS_CAP	WHCL_TYPE	FUEL_CAP	FUEL_TYPE	CAR_DESC	BICYCLE_DES
1	1	B				Mountain Bike
2	3	V				
3	8	F	20	Diesel		
4	5	C	15	Unleaded	Sedan	

**Figure 3–17 Joined Table Inheritance**



### 3.6 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

### 3.6.1 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 []`
- `Byte []`
- `char []`
- `Character []`
- `enums`
- any other type that implements `Serializable`

To create a basic mapping:

1. In the [Persistence Outline view](#), select the field to map. The [Persistence Properties 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 Persistence Properties view.

Property	Description	Default
Map As	Defines this mapping as a <b>Basic Mapping</b> . This field corresponds to the <code>@Basic</code> annotation.	Basic

Property	Description	Default
Column	<p>The database column (and its table) mapped to the entity attribute. See <a href="#">"Column"</a> on page 4-3 for details.</p> <ul style="list-style-type: none"> <li>■ Name – Name of the database column.</li> <li>■ Table – Name of the database table.</li> <li>■ Insertable – Specify if the column is always included in SQL INSERT statements.</li> <li>■ Updatable – Specify if the column is always included in SQL UPDATE statements.</li> </ul>	By default, the Column is assumed to be named identically to the attribute and always included in the INSERT and UPDATE statements.
Fetch Type	<p>Defines how data is loaded from the database. See <a href="#">"Fetch Type"</a> on page 4-4 for details.</p> <ul style="list-style-type: none"> <li>■ Eager</li> <li>■ Lazy</li> </ul>	Eager
Optional	Specifies if this field is can be null.	Yes
Temporal	<p>Specifies the type of data. See <a href="#">"Temporal"</a> on page 4-4 for details.</p> <ul style="list-style-type: none"> <li>■ Date</li> <li>■ Time</li> <li>■ Timestamp</li> </ul>	

Eclipse adds the following annotations to the field:

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

### 3.6.2 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 [Persistence Outline view](#), select the field to map. The [Persistence Properties 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 Persistence Properties view.

Property	Description	Default
Map As	<p>Defines this mapping as a <b>Embedded</b>.</p> <p>This field corresponds to the <code>@Embedded</code> annotation.</p>	Embedded
Attribute Overrides	Specify to override the default mapping of an entity's attribute.	



Property	Description	Default
Columns	<p>The database column (and its table) mapped to the entity attribute. See <a href="#">"Column"</a> on page 4-3 for details.</p> <ul style="list-style-type: none"> <li>■ Name – Name of the database column.</li> <li>■ Table – Name of the database table.</li> <li>■ Insertable – Specify if the column is always included in SQL INSERT statements.</li> <li>■ Updatable – Specify if the column is always included in SQL UPDATE statements.</li> </ul>	By default, the Column is assumed to be named identically to the attribute and always included in the INSERT and UPDATE statements.

Eclipse adds the following annotations to the field:

```
@Embedded
```

### 3.6.3 Embedded ID mapping

Use an **Embedded ID Mapping** to specify the primary key of an embedded ID. These mappings may be used with a [Embeddable](#) entities.

1. In the [Persistence Outline view](#), select the field to map. The [Persistence Properties 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 Persistence Properties view.

Property	Description	Default
Map As	<p>Defines this mapping as a <b>Embedded Id</b>.</p> <p>This field corresponds to the <code>@EmbeddedId</code> annotation.</p>	Embedded Id

Eclipse adds the following annotations to the field:

```
@EmbeddedId
```

### 3.6.4 ID mapping

Use an **ID Mapping** to specify the primary key of an entity. ID mappings may be used with a [Persistent entity](#) or [Mapped superclass](#). Each [Persistent entity](#) must have an ID mapping.

1. In the [Persistence Outline view](#), select the field to map. The [Persistence Properties view \(for attributes\)](#) displays the properties for the selected.
2. In the Map As field, select **ID**.
3. Use this table to complete the remaining fields on **General** tab in the Persistence Properties view.

Property	Description	Default
Map As	Defines this mapping as an <b>ID Mapping</b> . This field corresponds to the @Id annotation.	ID
Column	The database column (and its table) mapped to the entity attribute. See " <a href="#">Column</a> " on page 4-3 for details. <ul style="list-style-type: none"> <li>▪ Name – Name of the database column.</li> <li>▪ Table – Name of the database table.</li> <li>▪ Insertable – Specify if the column is always included in SQL INSERT statements.</li> <li>▪ Updatable – Specify if the column is always included in SQL UPDATE statements.</li> </ul>	By default, the Column is assumed to be named identically to the attribute and always included in the INSERT and UPDATE statements.
Temporal	Specifies the type of data. See " <a href="#">Temporal</a> " on page 4-4 for details. <ul style="list-style-type: none"> <li>▪ Date</li> <li>▪ Time</li> <li>▪ Timestamp</li> </ul>	

4. Use this table to complete the fields on **PK Generation** tab in the Persistence Properties view.

Property	Description	Default
Primary Key Generation	These fields define how the primary key is generated.	
Strategy	See " <a href="#">Primary Key Generation</a> " on page 4-6 for details. <ul style="list-style-type: none"> <li>▪ Auto</li> <li>▪ Sequence</li> <li>▪ Identity</li> <li>▪ Table</li> </ul>	Auto
Generator Name	Name of the primary key generator specified in the <b>Strategy</b>	

Additional fields will appear on the **PK Generation** tab, depending on the selected Strategy. See "[Persistence Properties view \(for attributes\)](#)" on page 4-3 for additional information.

Eclipse adds the following annotations to the field:

```
@Id
@Column(name="<COLUMN_NAME>", table="<TABLE_NAME>", insertable=<INSERTABLE>,
        updatable=<UPDATABLE>)
@Temporal(TemporalType.<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>",
pkColumnValue = "<TABLE_GENERATOR_PK_COLUMN_VALUE>"
@SequenceGenerator(name="<SEQUENCE_GENERATOR_NAME>",
sequenceName="<SEQUENCE_GENERATOR_SEQUENCE>")

```

### 3.6.5 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 [Persistence Outline view](#), select the field to map. The [Persistence Properties 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 fields on the [General tab](#) of the Persistence Properties view.

Property	Description	Default
Target Entity	The entity to which this attribute is mapped.	null You do not need to explicitly specify the target entity, since it can be inferred from the type of object being referenced.
Cascade Type	See " <a href="#">Cascade Type</a> " on page 4-4 for details. <ul style="list-style-type: none"> <li>▪ Default</li> <li>▪ All</li> <li>▪ Persist</li> <li>▪ Merge</li> <li>▪ Remove</li> </ul>	Default
Fetch Type	Defines how data is loaded from the database. See " <a href="#">Fetch Type</a> " on page 4-4 for details. <ul style="list-style-type: none"> <li>▪ Eager</li> <li>▪ Lazy</li> </ul>	Eager
Mapped By	The database field that owns the relationship.	
Order By	Specify the default order for objects returned from a query. See " <a href="#">Order By</a> " on page 4-4 for details. <ul style="list-style-type: none"> <li>▪ Primary key</li> </ul>	Primary key

4. Use this table to complete the fields on the [Join Table tab](#) in the Persistence Properties view.

Property	Description	Default
Name	Name of the join table that contains the foreign key column.	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.
Join Columns	Select <b>Override Default</b> , then Add, Edit, or Remove the join columns.	By default, the name is assumed to be the primary tables associated with the entities concatenated with an underscore.
Inverse Join Columns	Select <b>Override Default</b> , then Add, Edit, or Remove the join columns.	By default, the mapping is assumed to have a single join.

- 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:

```
@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>")
```

### 3.6.6 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.

- In the [Persistence Outline view](#), select the field to map. The [Persistence Properties view \(for attributes\)](#) displays the properties for the selected.
- In the Map As field, select **Many-to-One**.
- Use this table to complete the fields on the **General** tab in the Persistence Properties view.

Property	Description	Default
Target Entity	The entity to which this attribute is mapped.	null  You do not need to explicitly specify the target entity, since it can be inferred from the type of object being referenced.
Fetch Type	Defines how data is loaded from the database. See " <a href="#">Fetch Type</a> " on page 4-4 for details. <ul style="list-style-type: none"> <li>▪ Eager</li> <li>▪ Lazy</li> </ul>	Eager

Property	Description	Default
Cascade Type	See " <a href="#">Cascade Type</a> " on page 4-4 for details. <ul style="list-style-type: none"> <li>▪ Default</li> <li>▪ All</li> <li>▪ Persist</li> <li>▪ Merge</li> <li>▪ Remove</li> </ul>	Default
Optional	Specifies if this field is can be null.	Yes

- Use this table to complete the fields on the [Join Columns tab](#) in the Persistence Properties view.

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

Eclipse adds the following annotations to the field:

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

### 3.6.7 One-to-many mapping

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

- In the [Persistence Outline view](#), select the field to map. The [Persistence Properties view \(for attributes\)](#) displays the properties for the selected.
- In the Map As field, select **One-to-many**.
- Use this table to complete the fields on the **General** tab in the Persistence Properties view.

Property	Description	Default
Target Entity	The entity to which this attribute is mapped.	
Cascade Type	See " <a href="#">Cascade Type</a> " on page 4-4 for details. <ul style="list-style-type: none"> <li>▪ Default</li> <li>▪ All</li> <li>▪ Persist</li> <li>▪ Merge</li> <li>▪ Remove</li> </ul>	

Property	Description	Default
Fetch Type	Defines how data is loaded from the database. See " <a href="#">Fetch Type</a> " on page 4-4 for details. <ul style="list-style-type: none"> <li>▪ Eager</li> <li>▪ Lazy</li> </ul>	Eager
Mapped By	The database field that owns the relationship.	
Order By	Specify the default order for objects returned from a query. See " <a href="#">Order By</a> " on page 4-4 for details. <ul style="list-style-type: none"> <li>▪ Primary key</li> </ul>	Primary key

4. Use this table to complete the fields on the **Join Table** tab in the Persistence Properties view.

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

Eclipse adds the following annotations to the field:

```
@OneToMany(targetEntity=<TARGET_ENTITY>)
@Column(name="<COLUMN>")
```

```
@OneToMany(targetEntity=<TARGET_ENTITY>.class, cascade=CascadeType.<CASCADE_TYPE>,
    fetch = FetchType.<FETCH_TYPE>, mappedBy = "<MAPPED_BY>")
@OrderBy("<ORDER_BY>")
@JoinTable(name="<JOIN_TABLE_NAME>", joinColumns=@JoinColumn(name=
    "<JOIN_COLUMN_NAME>", referencedColumnName="<JOIN_COLUMN_REFERENCED_COLUMN>"),
    inverseJoinColumns=@JoinColumn(name="<INVERSE_JOIN_COLUMN_NAME>",
    referencedColumnName="<INVERSE_JOIN_COLUMN_REFERENCED_COLUMN>"))
```

### 3.6.8 One-to-one mapping

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

1. In the [Persistence Outline view](#), select the field to map. The [Persistence Properties 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 remaining fields on the **General** tab in Persistence Properties view.

Property	Description	Default
Target Entity	The entity to which this attribute is mapped.	null You do not need to explicitly specify the target entity, since it can be inferred from the type of object being referenced.
Cascade Type	See " <a href="#">Cascade Type</a> " on page 4-4 for details. <ul style="list-style-type: none"> <li>■ Default</li> <li>■ All</li> <li>■ Persist</li> <li>■ Merge</li> <li>■ Remove</li> </ul>	Default
Fetch Type	Defines how data is loaded from the database. See " <a href="#">Fetch Type</a> " on page 4-4 for details. <ul style="list-style-type: none"> <li>■ Eager</li> <li>■ Lazy</li> </ul>	Eager
Optional	Specifies if this field is can be null.	Yes
Mapped By	The database field that owns the relationship.	

- Use this table to complete the fields on the [Join Columns](#) tab in the Persistence Properties view.

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

Eclipse adds the following annotations to the field:

```
@ManyToOne(targetEntity=<TARGET_ENTITY>, cascade=CascadeType.<CASCADE_TYPE>,
    fetch = FetchType.<FETCH_TYPE>, mappedBy = "<MAPPED_BY>")
@JoinColumn(name="<JOIN_COLUMN_NAME>", referencedColumnName=
    "<JOIN_COLUMN_REFERENCED_COLUMN>", insertable = <INSERTABLE>,
    updatable = <UPDATABLE>)
```

### 3.6.9 Transient mapping

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

To create a version mapping:

- In the [Persistence Outline view](#), select the field to map. The [Persistence Properties view \(for attributes\)](#) displays the properties for the selected.
- In the Map As field, select **Transient**.

Eclipse adds the following annotation to the field:

```
@Transient
```

### 3.6.10 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 [Persistence Outline view](#), select the field to map. The [Persistence Properties 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 on the Persistence Properties view.

Property	Description	Default
Column	<p>The database column (and its table) mapped to the entity attribute. See <a href="#">"Column"</a> on page 4-3 for details.</p> <ul style="list-style-type: none"> <li>■ Name – Name of the database column.</li> <li>■ Table – Name of the database table. This must be the primary table associated with the attribute's entity.</li> <li>■ Insertable – Specify if the column is always included in SQL INSERT statements.</li> <li>■ Updatable – Specify if the column is always included in SQL UPDATE statements.</li> </ul>	By default, the Column is assumed to be named identically to the attribute and always included in the INSERT and UPDATE statements.

Eclipse adds the following annotations to the field:

```
@Version
@Column(table="<COLUMN_TABLE>", name="<COLUMN_NAME>", insertable=false,
        updatable=false)
```

## 3.7 Generating entities from tables

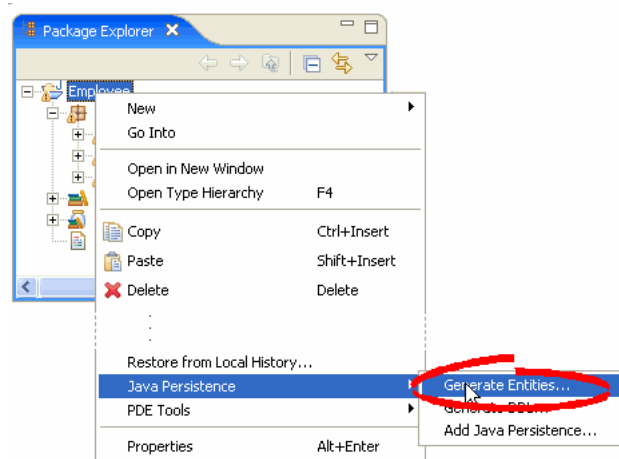
Use this procedure to generate Java persistent entities from database tables. You must add persistence to your project and establish a database connection *before* generating



persistent entities. See ["Adding persistence to a Java project"](#) on page 3-1 for more information.

1. Right-click the persistent project in the Package Explorer and select **Java Persistence > Generate Entities**.

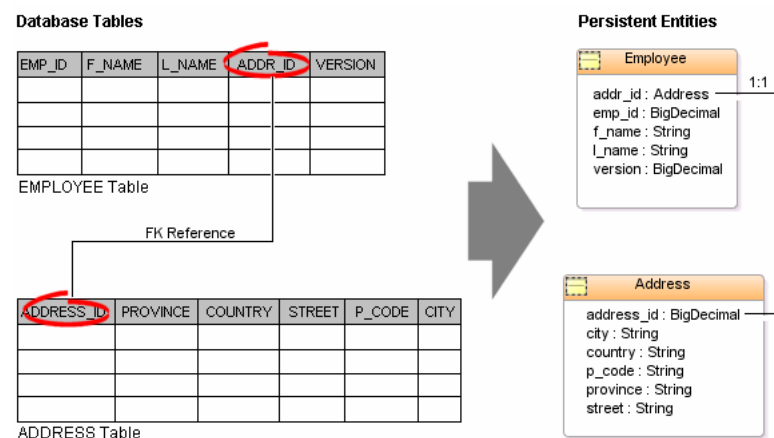
**Figure 3–18 Generating Entities**



2. On the [Generate Entities from Tables dialog](#) 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 3–19](#) illustrates how Eclipse generates entities from tables.

**Figure 3–19 Generating Entities from Tables**

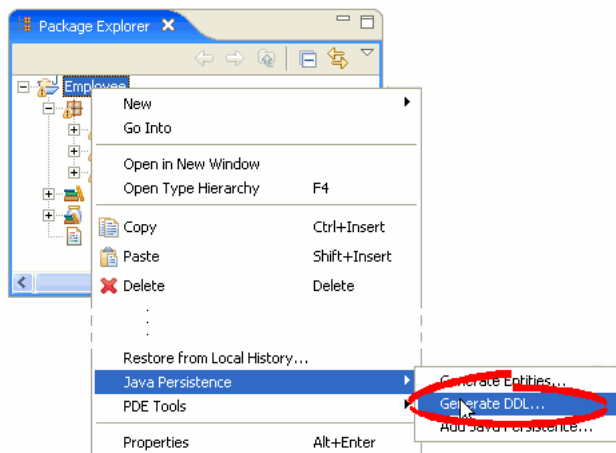


### 3.8 Generating tables (DDL scripts) from entities

Use this procedure to generate DDL script (for database tables) from your persistent entities. You must establish a database connection *before* generating DDL files.

1. Right-click the persistent project in the Package Explorer and select **Java Persistence > Generate DDL**.

**Figure 3–20** *Generating DDL from Entities*



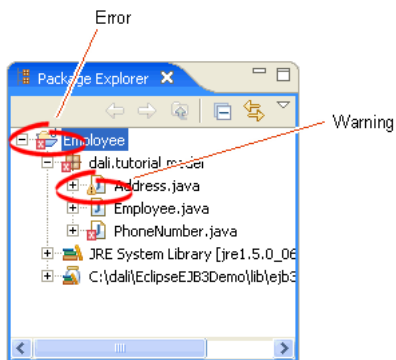
2. Complete the information on each page of the [Generate Database DDL from Entities wizard](#).

Refer to the "Using web tools" section of the *Web Application Development Guide* for additional information.

### 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.

**Figure 3–21** *Sample Errors and Warnings*



This section contains information on the following:

- [Error 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.

**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 3-10.

**Entity does not have an Id.**

You created a persistent entity without identifying its primary key. A persistent entity must have a primary key field designated with an @Id annotation.

Add an ID mapping to the entity as shown in ["ID mapping"](#) on page 3-13.

**MappedBy specified on both sides of the "<MAPPED\_FIELD>" relationship.**

You selected a Mapped By database field on both the source and target of a [Many-to-many mapping](#). This field is required only on the non-owning side of the relationship.

See ["Persistence Properties view \(for attributes\)"](#) on page 4-3 for more information.

**MappedBy specified on "<MAPPED\_FIELD>" is not a valid mapping type.**

You selected an invalid database field as the owner of a [One-to-one mapping](#). On the [General tab](#) of the mapping, select the field in the database table that "owns" the relationship. This field is required only on the non-owning side of the relationship.

See ["Persistence Properties view \(for attributes\)"](#) on page 4-3 for more information.

**Multiple Version properties are not allowed.**

You created two version mappings for an entity. Each persistent entity may have a single [Version mapping](#). Change the mapping type of one of the fields to a different mapping type.

See ["Mapping an entity"](#) on page 3-10 for more information.

**Table "<TABLE\_NAME>" cannot be found on the database.**

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 3-5.

**Target "<CLASS\_NAME>" class is not an entity.**

You created a relationship mapping from a field in an entity to a nonpersistent class. The target of OR relationship must be an entity.

Map the target class as an entity as shown in ["Persistent entity"](#) on page 3-5. See ["Mapping an entity"](#) on page 3-10 for more information.

**The join column "<COLUMN\_NAME>" cannot be found on the table.**

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 tab](#) or create the necessary column on the database table.

See "[Persistence Properties view \(for attributes\)](#)" on page 4-3 for more information.

**The referenced column "<COLUMN\_NAME>" cannot be found on the table.**

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 tab](#) or create the necessary column on the database table.

See "[Persistence Properties view \(for attributes\)](#)" on page 4-3 for more information.

**The table "<TABLE\_NAME>" cannot be found on the database.**

You attempted to associate an entity or mapping to a table that does not exist on the database. Verify that you are connected to the database and create the specific table, if necessary.

See "[Project Properties page – Persistence Options](#)" on page 4-7 for more information.

**The table "<TABLE\_NAME>" is not associated with the owning entity.**

The column that you selected to join a relationship mapping is not associated with the referenced (owning) entity. You must associate each entity with a database table on the entity's [General tab](#) in the Persistence Properties view.

See "[Persistence Properties view \(for entities\)](#)" on page 4-3 for more information.

**Version mapping type must be int, Integer, short, Short, long, Long, or Timestamp.**

For a [Version mapping](#), the attribute must be mapped to a database column of one of a valid type, as listed in "[Version mapping](#)" on page 3-20. See "[Mapping an entity](#)" on page 3-10 for more information.

**Version mappings must be mapped to the primary table.**

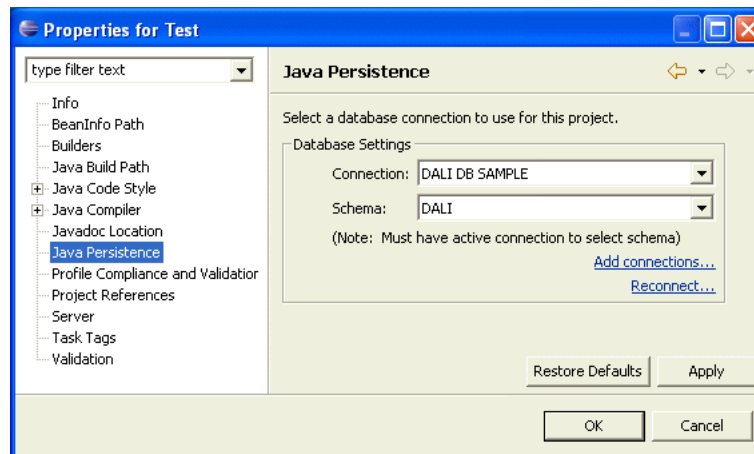
For a [Version mapping](#), the attribute must be mapped to the *primary* table associated with the entity. Confirm that the **Column Name** field of the version mapping is identical to the **Table** field on the entity's [General tab](#). See "[Version mapping](#)" on page 3-20 and "[Mapping an entity](#)" on page 3-10 for more information.

## 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 database associated with your persistent project.

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

**Figure 3–22 Properties – Persistence Page**

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

Property	Description
Database Connection	Database connection to use to store the persistent entities. To reconnect to the selected connection, click <b>Reconnect</b> .
Database Schema	Schema of the <b>Database Connection</b> to use. <b>Note:</b> You must be connected to the database before selecting the schema.

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](#)
- [Persistence perspective](#)
- [Icons and buttons](#)
- [Dali Developer Documentation](#)

## 4.1 Wizards

This section includes information on the following wizards:

- [Generate Database DDL from Entities wizard](#)

### 4.1.1 Generate Database DDL from Entities wizard

Use the Generate DDL wizard to generate DDL files from your Java persistence entities. You must have a defined database connection (and be connected) before using this wizard.

The wizard contains the following pages:

- Options page
- Objects page
- Save and Run DDL page
- Summary page

Refer to the "Using data tools" section of the Web Application Development Guide for additional information.

## 4.2 Property pages

This section includes information on the following property pages:

- [Persistence Properties view \(for entities\)](#)
- [Persistence Properties view \(for attributes\)](#)
- [Persistence Outline view](#)

## 4.2.1 Persistence Properties view (for entities)

The Persistence Properties view displays the persistence information for the currently selected entity and contains the following tabs:

- [General tab](#)
- [Inheritance tab](#)

### 4.2.1.1 General tab

This table lists the fields available on the General tab in the Persistence Properties view for each entity type.

Property	Description	Default	Available for Entity Type
Access Type	Specify how the entity its access instance variables. <ul style="list-style-type: none"> <li>■ Property – Persistent state accessed through the property accessor methods. The property accessor methods must be <b>public</b> or <b>private</b>.</li> <li>■ Field – Instance variables are accessed directly. All non-transient instance variables are persistent.</li> </ul> <p><b>Note:</b> This field is for display only, based on the properties in the <code>orm.xml</code>: If only the methods of the class are annotated, <b>property</b> access type is used. In all other cases, <b>field</b> access type is used.</p>	Property	<a href="#">Persistent entity</a> , <a href="#">Embeddable</a> , and <a href="#">Mapped superclass</a>
Name	The name of this entity. By default, the class name is used as the entity name.		<a href="#">Persistent entity</a>
Table Name	The primary database table associated with the entity.		<a href="#">Persistent entity</a>
Attribute Overrides	Specify a property or field to be overridden (from the default mappings).		<a href="#">Persistent entity</a>
Column	The database column (from the <b>Table Name</b> ) mapped to the entity.		<a href="#">Persistent entity</a>
Name	Name of the database column.		<a href="#">Persistent entity</a>
Table	Name of the database table that contains the selected column.		<a href="#">Persistent entity</a>
Insertable	Specifies if the column is always included in SQL INSERT statements.	True	<a href="#">Persistent entity</a>
Updatable	Specifies if this column is always included in SQL UPDATE statements.	True	<a href="#">Persistent entity</a>

### 4.2.1.2 Inheritance tab

This table lists the fields available on the Inheritance tab in the Persistence Properties view for each entity type.



Property	Description	Default
Strategy	Specify the strategy to use when mapping a class or class hierarchy: <ul style="list-style-type: none"> <li>Single table – All classes in the hierarchy are mapped to a single table.</li> <li>Joined – The root of the hierarchy is mapped to a single table; each child maps to its own table.</li> <li>Table per class – Each class is mapped to a separate table.</li> </ul>	Single table
Discriminator Column	Use to specify the name of the discriminator column when using a <b>Single</b> or <b>Joined</b> inheritance strategy.	
Discriminator Type	Use this field to set the discriminator type to <code>Char</code> or <code>Integer</code> (instead of its default: <code>String</code> ). The <b>Discriminator Value</b> must conform to this type.	
Discriminator Value	Specify the discriminator value used to differentiate an entity in this inheritance hierarchy. The value must conform to the specified <b>Discriminator Type</b> .	

Refer to ["Specifying entity inheritance"](#) on page 3-9 for additional information.

## 4.2.2 Persistence Properties view (for attributes)

The Persistence Properties view displays the persistence information for the currently selected mapped attribute and contains the following tabs:

- General tab
- Join Table tab
- Join Columns tab
- PK Generation tab

See ["Mapping an entity"](#) on page 3-10 for more information.

### 4.2.2.1 General tab

This table lists the properties available on the General tab in the Persistence Properties view for each mapping type.

Property	Description	Default	Available for Mapping Type
Map As	Define the mapping type for the attribute	Basic	All mapping types
Column	The database column that contains the value for the attribute. This property corresponds to the <code>@Column</code> annotation.	By default, the Column is assumed to be named identically to the attribute.	<a href="#">Basic mapping</a> , <a href="#">Embedded mapping</a> , <a href="#">ID mapping</a> , <a href="#">Version mapping</a>

Property	Description	Default	Available for Mapping Type
Name	Name of the database column.		Basic mapping, Embedded mapping, ID mapping
Table	Name of the database table that contains the selected column.		Basic mapping, Embedded mapping, ID mapping
Insertable	Specifies if the column is always included in SQL INSERT statements.	True	Basic mapping, Embedded mapping, ID mapping
Updatable	Specifies if this column is always included in SQL UPDATE statements.	True	Basic mapping, Embedded mapping, ID mapping
Fetch Type	Defines how data is loaded from the database: <ul style="list-style-type: none"> <li>▪ Eager – Data is loaded in before it is actually needed.</li> <li>▪ Lazy – Data is loaded only when required by the transaction.</li> </ul>	Eager	Basic mapping, One-to-one mapping
Optional	Specifies if this field is can be null.	Yes	Basic mapping, One-to-one mapping
Temporal	Specifies if this field is one of the following: <ul style="list-style-type: none"> <li>▪ Date – <code>java.sql.Date</code></li> <li>▪ Time – <code>java.sql.Time</code></li> <li>▪ Timestamp – <code>java.sql.Timestamp</code></li> </ul> This field corresponds to the <code>@Temporal</code> annotation.		Basic mapping, ID mapping
Target Entity	The persistent entity to which the attribute is mapped.		One-to-one mapping
Cascade Type	Specify which operations are propagated throughout the entity. <ul style="list-style-type: none"> <li>▪ All – All operations</li> <li>▪ Persist</li> <li>▪ Merge</li> <li>▪ Move</li> </ul>		One-to-one mapping
Mapped By	The field in the database table that "owns" the relationship. This field is required only on the non-owning side of the relationship.		One-to-one mapping
Order By	Specify the default order for objects returned from a query: <ul style="list-style-type: none"> <li>▪ Primary key</li> </ul> This field corresponds to the <code>@OrderBy</code> annotation.	Primary key	One-to-many mapping, Many-to-many mapping, Many-to-one mapping

Property	Description	Default	Available for Mapping Type
Attribute Overrides	Overrides the column mappings from the mapped, entity tabled. (for example, if the inherited column name is incompatible with a pre-existing data model, or invalid as a column name in your database).		<a href="#">Embedded mapping</a> <a href="#">Embedded mapping</a>

#### 4.2.2.2 Join Table tab

Use this tab 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 tab in Properties view for [One-to-many mapping](#) and [Many-to-many mapping](#) mapping types.

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

#### 4.2.2.3 Join Columns tab

This table lists the fields available on the Join Table tab in Properties view for [Many-to-one mapping](#) and [One-to-one mapping](#) mapping types.

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

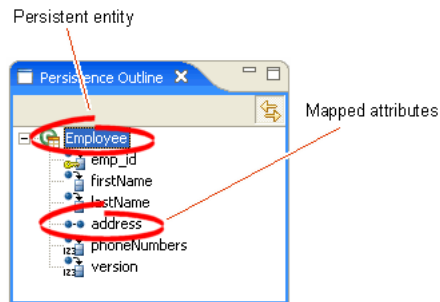
#### 4.2.2.4 PK Generation tab

This table lists the fields available on the PK Generation tab in Properties view for [ID mapping](#) types.

Property	Description	Default
Primary Key Generation	These fields define how the primary key is generated. These fields correspond to the <code>@GeneratedValue</code> annotation.	Generated Value
Strategy	<ul style="list-style-type: none"> <li>■ Auto</li> <li>■ Sequence – Values are assigned by a sequence table (see <a href="#">Sequence Generator</a>).</li> <li>■ Identity – Values are assigned by the database’s <b>Identity</b> column.</li> <li>■ Table – Values are assigned by a database table (see <a href="#">Table Generator</a>).</li> </ul>	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 <code>@TableGenerator</code> annotation.  These fields apply only when <b>Strategy = Table</b> .	
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 <b>Table</b> that contains the primary key.	
Value Column	The column that stores the generated ID values.	
Primary Key Column Value	The value for the <b>Primary Key Column</b> in the generator table.	
Sequence Generator	These fields define the database table used for generating the primary key and correspond to the <code>@SequenceGenerator</code> annotation. These fields apply only when <b>Strategy = Sequence</b> .	
Name	Name of the sequence table to use for defining primary key values.	
Sequence	Unique name of the sequence.	

### 4.2.3 Persistence Outline view

The Persistence Outline 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 Persistence Outline View**

## 4.3 Preferences

This section includes information on the following preference pages:

- [Project Properties page – Persistence Options](#)

### 4.3.1 Project Properties page – Persistence Options

Use the Persistence 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.

Property	Description
Database Connection	The database connection used to map the persistent entities. <ul style="list-style-type: none"> <li>■ To create a new connection, click <b>Add Connections</b>.</li> <li>■ To reconnect to an existing connection, click <b>Reconnect</b>.</li> </ul>
Database Schema	The database schema used to map the persistent entities. You must be selected to the database before selecting a schema.

See "[Modifying persistent project properties](#)" on page 3-24 for additional information.

## 4.4 Dialogs

This section includes information on the following preference pages:

- [Add Persistence dialog](#)
- [Generate Entities from Tables dialog](#)

### 4.4.1 Add Persistence dialog

Use the Add Persistence dialog to define the database connection used to store the persistence entities and to create the `persistence.xml` file.

This table lists the properties available in the Add Persistence dialog.

Property	Description
Database Settings	Use these fields to define the database connection used to store the persistent entities.
Connection	The database connection used to map the persistent entities.
Schema	The database schema used to map the persistent entities. You must be selected to the database before selecting a schema.
Classpath Configuration	Add libraries or JARs that contain the Java Persistence API (JPA) and entities to the project's Java Build Path.
Packaging Settings	Use these fields to create the <code>persistence.xml</code> file.
Create persistence.xml	Specify if Dali should create the <code>persistence.xml</code> file.
Persistence Version	Select the version of the persistence provider.
Persistence Provider	The name of the JPA provider's <code>javax.persistence.spi.PersistenceProvider</code> class.
Persistence Unit Name	The name used to identify the persistence unit. You can use this name when referencing the persistence unit by the <code>PersistenceContext</code> and <code>PersistenceUnit</code> annotations or when creating an entity manager factor programmatically.

See ["Adding persistence to a Java project"](#) on page 3-1 for more information.

#### 4.4.2 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.

Property	Description
Source Folder	Enter a project folder name in which to generate the Java persistent entities, or click <b>Browse</b> to select an existing folder.
Package	Enter a package name in which to generate the Java persistent entities, or click <b>Browse</b> to select an existing package.
Tables	Select the tables from which to create Java persistent entities. The tables shown are determined by the database connection that you defined in the <a href="#">Project Properties page – Persistence Options</a> .

See ["Generating entities from tables"](#) on page 3-20 for more information.

#### 4.4.3 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.

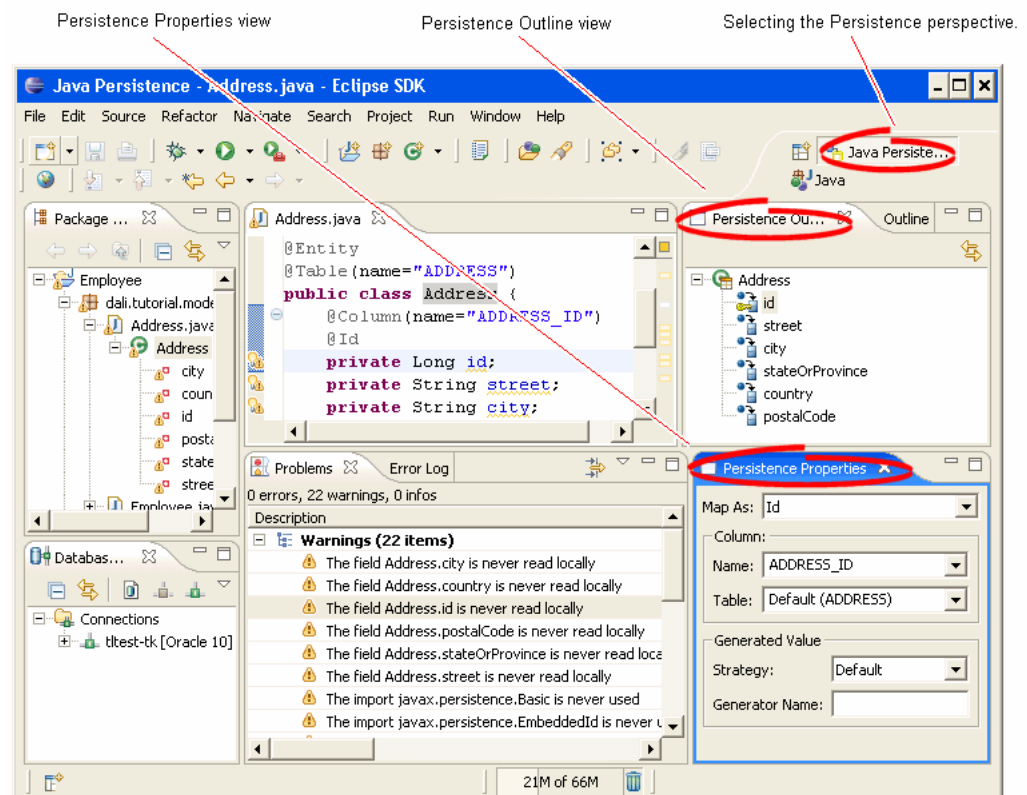
Property	Description
Name	Name of the joint table column that contains the foreign key column.
Referenced Column Name	Name of the database column that contains the foreign key reference for the entity relationship.

## 4.5 Persistence perspective

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

- [Persistence Outline view](#)
- [Persistence Properties view \(for entities\)](#)

**Figure 4–2 Sample Persistence Perspective**



## 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

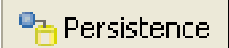
The following icons are used throughout the Dali OR Mapping Tool.

Icon	Description
	Nonpersistent class
	Persistent entity
	Embeddable entity
	Mapped superclass
	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

### 4.6.2 Buttons

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



Icon	Description
 Persistence	Persistence perspective

## 4.7 Dali Developer Documentation

Additional Dali documentation is available online at:

[http://wiki.eclipse.org/index.php/Dali\\_Developer\\_Documentation](http://wiki.eclipse.org/index.php/Dali_Developer_Documentation)

This developer documentation includes information about:

- Dali architecture
- Plugins that comprise the Dali JPA Eclipse feature
- Extension points



---

---

## Tips and tricks

The following tips and tricks give some helpful ideas for increasing your productivity.

- [Database Connections](#)
- [Schema-based persistence.xml](#)

Tip	Description
<b>Database Connections</b>	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.
<b>Schema-based persistence.xml</b>	If you are behind a firewall, you may need to configure your Eclipse workspace proxy in the Preferences dialog ( <b>Preferences &gt; Internet &gt; Proxy Settings</b> ) to properly validate a schema-based <code>persistence.xml</code> file.



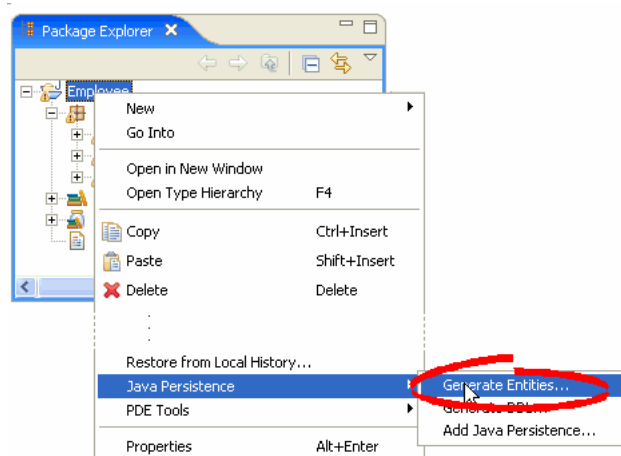
This section contains descriptions of the following new feature and significant changes made to the Dali OR Mapping Tool for Release 0.5.0:

- [Generate Persistent Entities from Tables wizard](#)
- [Generate DDL from Entities wizard](#)
- [Create and Manage the persistence.xml file](#)

## 6.1 Generate Persistent Entities from Tables wizard

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

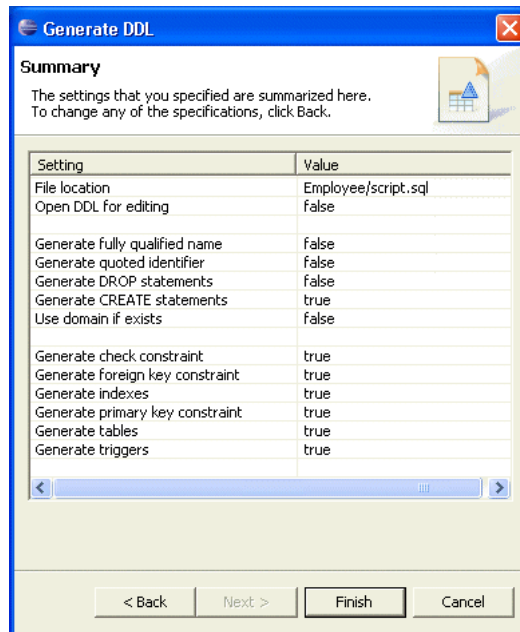
*Figure 6–1 Generating Entities*



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

## 6.2 Generate DDL from Entities wizard

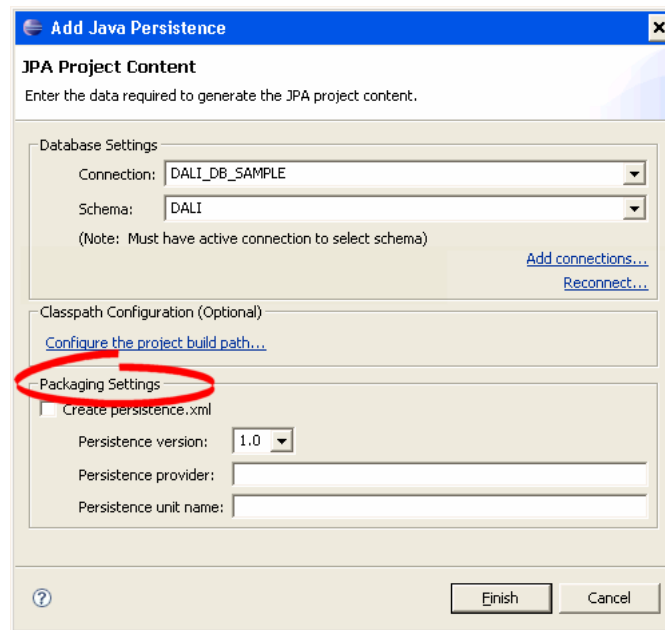
Use the Generate DDL wizard to quickly create DDL scripts from your persistent entities.

**Figure 6–2 Generating DDL**

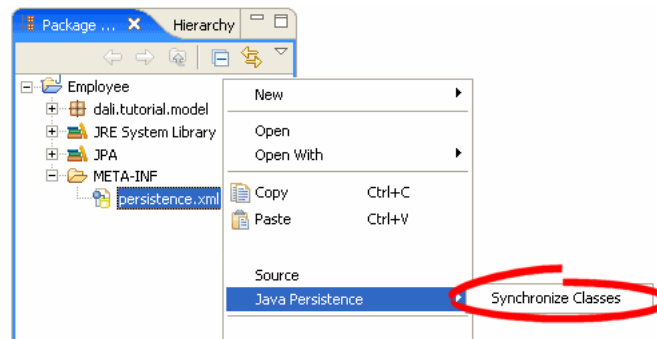
Dali automatically creates the necessary primary and foreign keys, based on your entity mappings.

## 6.3 Create and Manage the persistence.xml file

When adding persistence to a Java project, you can also create the `persistence.xml` file.

**Figure 6–3 Add Java Persistence Dialog**

Use the Packaging Settings options to create the `persistence.xml` file. After adding your persistent classes, use the **Java Persistence > Synchronize Classes** option to add the classes to the `persistence.xml` file.

**Figure 6–4 Synchronizing the persistence.xml File.**





The material in this guide is copyright © 2006, by Oracle.

## 7.1 About this content

Terms and conditions regarding the use of this guide.

May 2, 2006

### **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, 4-3  
@DiscriminatorColumn, 3-9  
@DiscriminatorValue, 3-10  
@Embeddable, 3-6  
@Embedded, 3-12  
@EmbeddedId, 3-13  
@Entity, 3-5  
@GeneratedValue, 4-6  
@Id, 3-13  
@Inheritance, 3-9  
@JoinColumn, 3-17, 3-19, 4-5  
@ManyToMany, 3-15  
@ManyToOne, 3-16  
@MappedSuperclass, 3-7  
@OneToMany, 3-17  
@OneToOne, 3-18  
@SequenceGenerator, 4-6  
@Transient, 3-19  
@Version, 3-20

## A

---

Add Persistence dialog, 3-2, 3-3, 4-1, 4-7  
adding persistence to projects, 3-1  
@OrderBy, 4-4  
@Temporal, 4-4  
annotations. *See specific annotation.*  
architecture of Dali feature, 4-11  
attributes  
    mapping, 2-1  
    Persistence Properties view, 4-3

## B

---

basic mapping  
    @Basic, 3-11  
    about, 3-11

## C

---

classes  
    adding persistence to, 3-5  
    embeddable, 3-6  
    entity, 3-5

    synchronizing, 3-5  
classpath configuration, persistence, 4-8  
columns  
    discriminator, 3-9  
    join, 3-17, 3-19, 4-5  
    mapping to, 4-3  
    value, 3-10  
connections, database, 3-2

## D

---

database tables  
    generating entities from, 3-21  
    generating from entities, 3-21  
database, persistence  
    connection, 3-2, 4-7, 4-8  
    schema, 4-7, 4-8  
developer documentation, 4-11

## E

---

eager fetch, 4-4  
EJB. *see* persistent entities  
embeddable class  
    @Embeddable, 3-6  
    about, 3-6  
embedded ID mapping  
    @EmbeddedId, 3-13  
    about, 3-13  
embedded mapping  
    @Embedded, 3-12  
    about, 3-12  
entities  
    @Entity annotation, 3-5  
    about, 2-1  
    creating tables from, 3-21  
    embeddable, 3-6  
    from tables, 3-20, 4-8  
    mapped superclass, 3-7  
    mapping, 1-6  
    persistence, 1-4  
    Persistence Properties view, 4-2  
    persistent, 3-5  
error messages, Dali, 3-22  
extension points, Dali feature, 4-11

## F

---

fetch type, 4-4

## G

---

Generate Entities from Tables dialog, 3-21, 4-8

generated values

  ID mappings, 4-6

  sequence, 4-6

## I

---

ID mapping

  @Id, 3-13

  about, 3-13

inheritance

  entity, 3-9, 4-2

  joined tables, 3-10

  single table, 3-10

Inheritance tab, 4-2

installation, Dali, 1-1

## J

---

Java Persistence API (JPA), adding to project, 3-2

joined tables, inheritance, 3-10

## L

---

lazy fetch, 4-4

## M

---

many-to-many mapping

  @ManyToMany, 3-15

  about, 3-15

many-to-one mapping

  @ManyToOne, 3-16

  about, 3-16

mapped superclass

  @MappedSuperclass, 3-7

  about, 3-7

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, 3-22

  transient, 3-19

  version, 3-20

## N

---

nonpersistent

classes, 3-5

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

ordering, 4-4

outline, persistence, 4-6

## P

---

persistence

  about, 2-1

  adding to projects, 3-1

  database connection, 4-7, 4-8

  database schema, 4-7, 4-8

  entity class, 3-5

  options, 4-7

  projects, 4-1, 4-7

Persistence Outline view, 4-6

Persistence perspective, 4-9

Persistence Properties view

  attributes, 4-3

  entities, 4-2

Persistence XML Editor, 3-4

persistence.xml file

  about, 2-2

  creating, 3-2

  editor, 3-4

  managing, 3-2

  sample, 3-3

  synchronizing with classes, 3-5

persistent entity, 3-5

perspective, persistence, 4-9

problems, 3-22

projects

  creating new, 1-2

  options, 4-7

  persistence, 4-1, 4-7

  persistence, adding, 3-1

## Q

---

quick start, 1-1

## R

---

requirements

  persistent entities, 3-5

requirements, Dali, 1-1

## S

---

schema, database, 4-7, 4-8

single table inheritance, 3-10

superclass, 3-7

## **T**

---

tables

- creating entities from, 3-20, 4-8

- from entities, 3-21

- inheritance, 3-10

temporal, 4-4

transient mapping

- @Transient, 3-19

- about, 3-19

tutorial, Dali, 1-8

## **V**

---

version mapping

- @Version, 3-20

- about, 3-20

views

- Persistence Outline view, 4-6

- Persistence Properties view, 4-2, 4-3

## **W**

---

warning messages, Dali, 3-22

