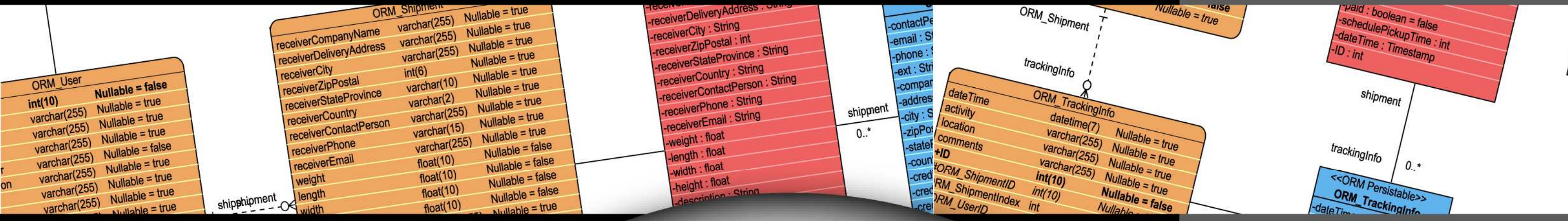
Visual Paradigm

Build Quality Applications Faster, Better and Cheaper

Database Visual ARCHITECT Programmer's Guide for Java

Access database with Object-Oriented technology



DB Visual ARCHITECT 4.0 Programmer's Guide for Java

Acknowledgements

This Product includes software developed by the Apache Software Foundation (http://www.apache.org). Copyright © 1999 The Apache Software Foundation. All rights reserved.

Table of Contents

1 Generating Java, Database and Persistent Library

Chapter 1 - Generating Java, Database and Persistent Library

DB Visual ARCHITECT (DB-VA) can generate Java code, export database schema (DDL) to database and create the persistent library based on what your have designed in the class diagram and entity relationship diagram. DB-VA will generate a high performance O/R Mapping (ORM) layer library ready for you to code and build. The ORM library basically intends to take most of the relational to object-oriented mapping burden off your shoulder. With the generated ORM code and library, you can use the plain Java objects in the application and tell the ORM layer to persist the object for you (e.g. ObjectDAO.save(myObject);) This chapter gives you an introduction to DB-VA, describes how to configure database, generate database and Java code step by step.

In this chapter:

- Introduction
- Configuring Database
- Generating Database
- Generating Java Code
- Selecting Optional Jar for Persistent Library

Introduction

DB Visual ARCHITECT (DB-VA) provides an easy-to-use environment bridging between object model, data model and relational database. You can visually model the system in both logical data design and physical data design, and DB-VA helps automate the mapping between object model and data model.

In this chapter, we assume that you know how to model your system with the class diagram and entity relationship diagram (please refer to the Designer's Guide for more details on modeling with class diagram and entity relationship diagram). Class diagram and entity relationship diagram will be used in this chapter to demonstrate how to make use of the DB-VA to export database schema (DDL) to database and generate Java persistent code.

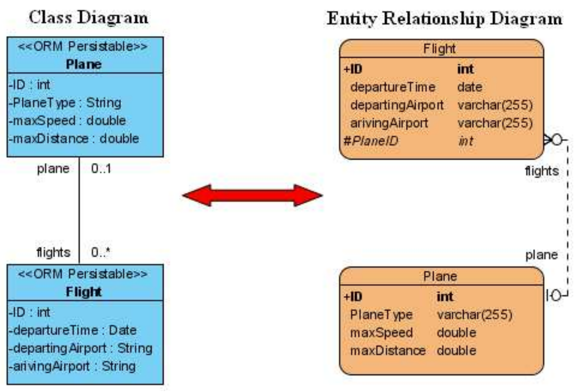


Figure 1.1 - Mapping between object model and data model

Configuring Database

DB-VA covers a wide range of databases in the market. You can check the latest supported databases version from http://www.visual-paradigm.com/product/dbva/

1. Please draw the above class diagram and synchronize it to ERD. (Alternatively, you can simply open the sample project, **flight.vpp**)
2. From the menu, select **Tools** > **Object-Relational Mapping (ORM)** > **Database Configuration...** to open the **Database Configuration** dialog box.

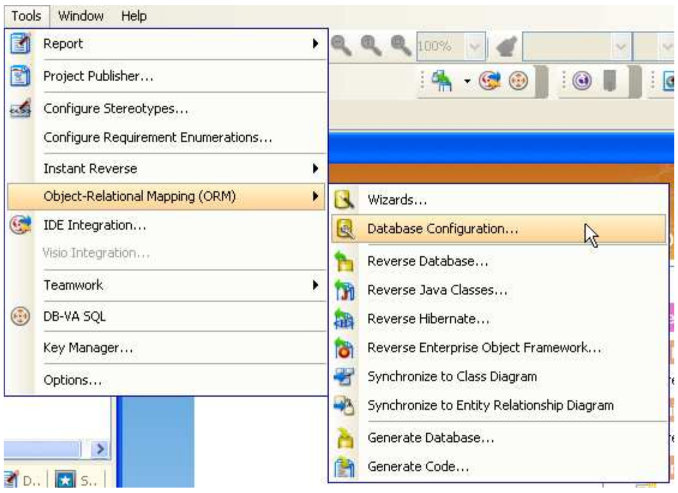
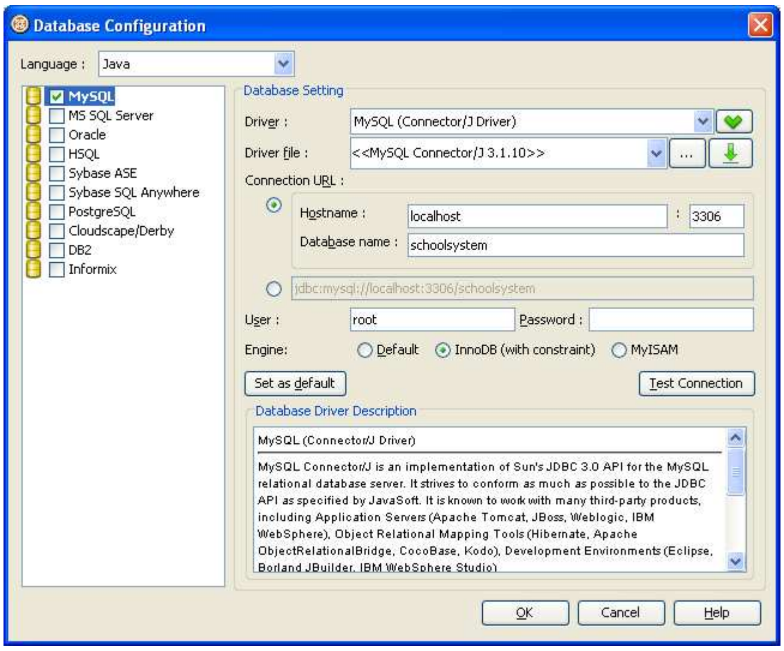


Figure 1.2 - To open the Database Configuration

3. Select **Java** from the drop-down menu of **Language** and select the desired database. MySQL database is used to demonstrate the database configuration.



Language : Java

[✓] MySQL
[ ] MS SQL Server
[ ] Oracle
[ ] HSQL
[ ] Sybase ASE
[ ] Sybase SQL Anywhere
[ ] PostgreSQL
[ ] Cloudscape/Derby
[ ] DB2
[ ] Informix

Database Setting

Driver : MySQL (Connector/J Driver)
Driver file : <<MySQL Connector/J 3.1.10>>
Connection URL :
(●) Hostname : localhost : 3306
Database name : schoolsystem
( ) jdbc:mysql://localhost:3306/schoolsystem
User : root Password :
Engine: ( ) Default (●) InnoDB (with constraint) ( ) MyISAM

Database Driver Description

MySQL (Connector/J Driver)

MySQL Connector/J is an implementation of Sun's JDBC 3.0 API for the MySQL relational database server. It strives to conform as much as possible to the JDBC API as specified by JavaSoft. It is known to work with many third-party products, including Application Servers (Apache Tomcat, JBoss, Weblogic, IBM WebSphere), Object Relational Mapping Tools (Hibernate, Apache ObjectRelationalBridge, CocoBase, Kodo), Development Environments (Eclipse, Borland JBuilder, IBM WebSphere Studio)

Figure 1.3 - Database Configuration dialog

4. Enter database setting for the selected database.

   For **Driver File**, you can press the down arrow button to select **Download**, **Update** or **Default Driver**; DB-VA helps you to download the most up-to-date driver according to the **Driver** field information. You can also select **Browse...** to specify the driver file in your computer.

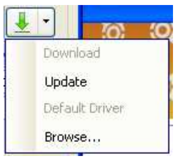
   

   Figure 1.4 - Download Button

   After downloaded the driver file, <<**MySQL Connector/J 3.1.10**>> shown on the **Driver file** indicates that the JDBC driver file is downloaded with the specified version number by DB-VA.

   For **Driver**, select the JDBC Driver from the drop-down menu. The driver's description will be shown in the **Database Driver Description** pane.

   For **Connection URL**, the Connection URL template for different database is shown. Enter the information for connecting the database.

   The default Connection URL template for MySQL is:

   ```
   jdbc:mysql://<host_name>:<port_number>/<database_name>
   ```

   The desired Connection URL MySQL is:

   ```
   jdbc:mysql://localhost:3306/control
   ```

   For **User**, enter the valid username who has the access right to connect database

   For **Password**, enter the password for the this user.
   For **Engine**, select the type of engine used in generating the MySQL database.

   Note The **Engine** option in the **Database Setting** is only provided when configuring **MySQL** database for Java project.

5. Press **Test Connection** button after filling in the database information to test whether the database can be connected.

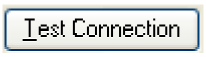
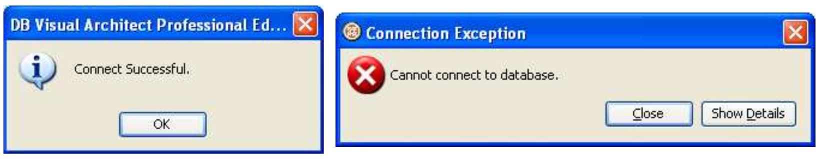
   

   Figure 1.5 - Test Connection button

   If the database can be connected, the **Connection Successful** dialog box will show; otherwise the **Connection Exception** dialog box will be prompted.

   
   DB Visual Architect Professional Ed...
   Connect Successful.

   Connection Exception
   Cannot connect to database.

   Figure 1.6 - Connection successful/failure message

6. Select one database to be the default database connection for generating code and database. To set the default database connection, right click on database and select **Set as default.**

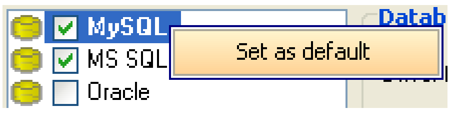
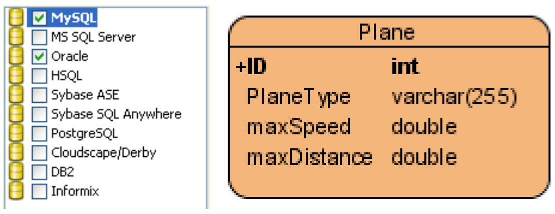
   

   Figure 1.7 - Set the database as default

If you set the default database, the data types used in the entity relationship diagram will be updated automatically. Here shows the difference in data type in entity relationship diagram between MySQL and Oracle:

- When MySQL is selected as the default database:

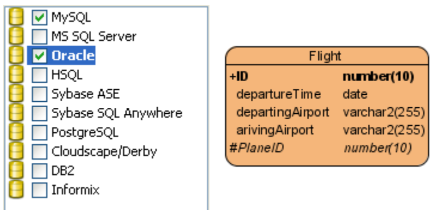


[✓] MySQL
[ ] MS SQL Server
[✓] Oracle
[ ] HSQL
[ ] Sybase ASE
[ ] Sybase SQL Anywhere
[ ] PostgreSQL
[ ] Cloudscape/Derby
[ ] DB2
[ ] Informix

Plane	
+ID	int
PlaneType	varchar(255)
maxSpeed	double
maxDistance	double

Figure 1.8 - Data types for MySQL

- When Oracle is selected as the default database:



[✓] MySQL
[ ] MS SQL Server
[✓] Oracle
[ ] HSQL
[ ] Sybase ASE
[ ] Sybase SQL Anywhere
[ ] PostgreSQL
[ ] Cloudscape/Derby
[ ] DB2
[ ] Informix

Flight	
+ID	number(10)
departureTime	date
departingAirport	varchar2(255)
arivingAirport	varchar2(255)
#PlaneID	number(10)

Figure 1.9 - Data types for Oracle

Generating Database

Now you can export the database schema from the Entity Relationship Diagram to the default database.

1. From the menu, select **Tools > Object-Relational Mapping (ORM) > Generate Database...** to open the **Database Code Generation** dialog box.

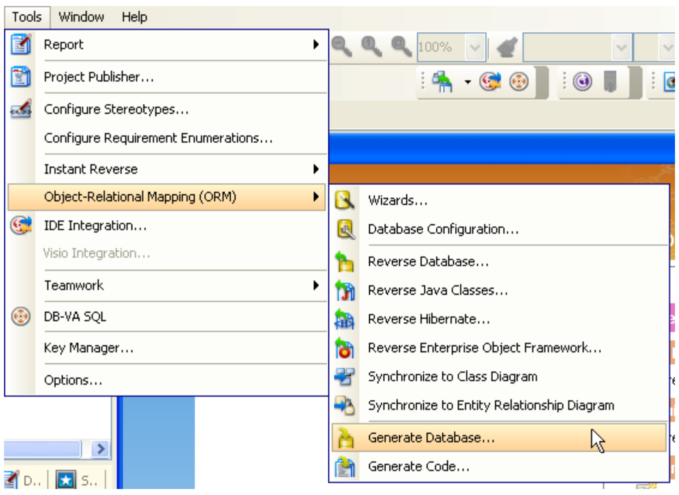
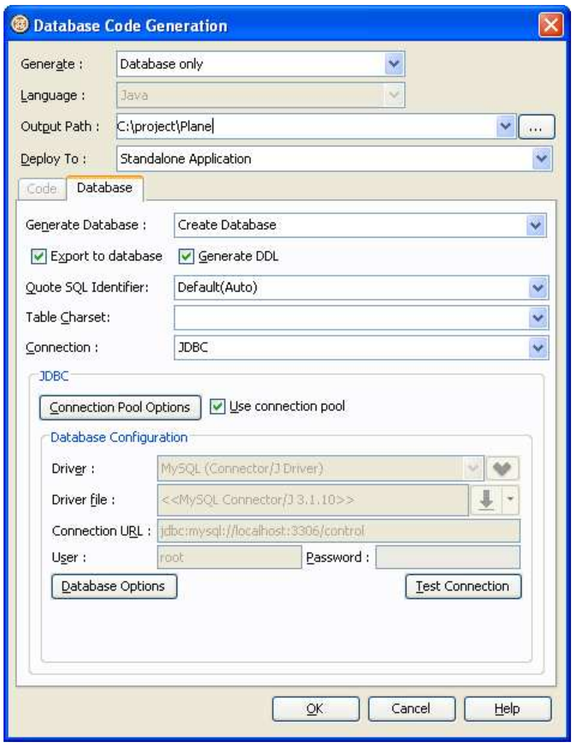


Figure 1.10 - To generate database

The dialog box shows the previous default database setting.


Database Code Generation

Generate : Database only
Language : Java
Output Path : C:\project\Plane
Deploy To : Standalone Application

Code | Database

Generate Database : Create Database
[✓] Export to database [✓] Generate DDL
Quote SQL Identifier: Default(Auto)
Table Charset:
Connection : JDBC

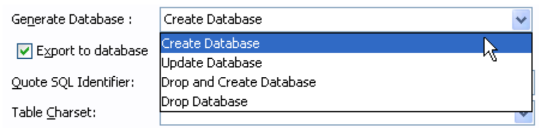
JDBC
Connection Pool Options [✓] Use connection pool

Database Configuration
Driver : MySQL (Connector/J Driver)
Driver file : <<MySQL Connector/J 3.1.10>>
Connection URL : jdbc:mysql://localhost:3306/control
User : root Password :
Database Options Test Connection

OK Cancel Help

Figure 1.11 - Database Code Generation dialog

2. Select **Generate Database** option which specifies the action for the database. Since it is the first time to export database schema, you can select **Create Database** option. DB-VA allows you to select **Create Database**, **Update Database**, **Drop and Create Database** and **Drop Database**.


Generate Database : Create Database
[✓] Export to database
Quote SQL Identifier:
Table Charset:

Create Database
Update Database
Drop and Create Database
Drop Database

Figure 1.12 - Generate Database options

3. Select **Export to database** option to allow altering the database immediately after you click the **OK** button.

[✓] Export to database

Figure 1.13 - Export to database options

4. Select **Generate DDL** option to allow the generation of DDL file.

[✓] Generate DDL

Figure 1.14 - Generate DDL options

5. You may use some reserved words (e.g. Order) in your database design, you can select the **Quote SQL Identifier** to avoid the naming problem in your design with the target database.
   - **Auto** - Only the detected reserved word will be quoted.
   - **Yes** - All table names and column names will be quoted.
   - **No** - No words will be quoted; i.e. reserved words cannot be used in generating the database.

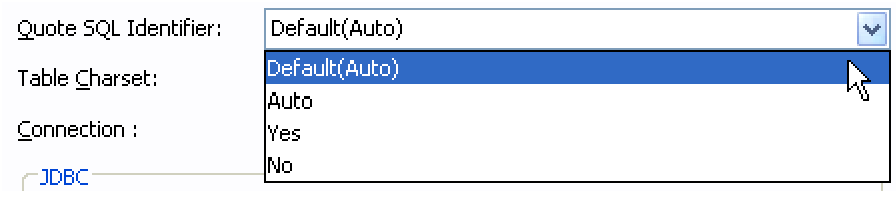


Figure 1.15 - Quote SQL Identifier options

6. Select **Connection** option, either **JDBC** or **Datasource**. If your application server provides a datasource for applications to share the JDBC connection within the application server, you can select **Datasource**. Otherwise, **JDBC** works for all situations.

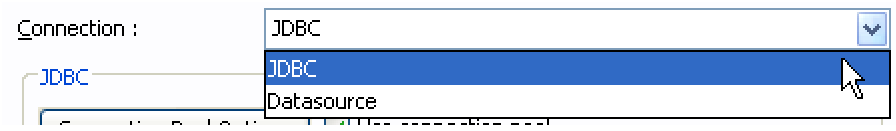


Figure 1.16 - Connection options

7. Click **Database Options** button to reconfigure the database setting if needed before generating the database.
8. Click **Connection Pool Options** button to open the **Connection Pool Options** dialog box to configure the connection pool settings. Connect to and disconnect from the database is an expensive operation, using the connection pool to share the opened connection can dramatically increase the application performance. You can deselect the **Use connection pool** option to disable the use of connection pool.

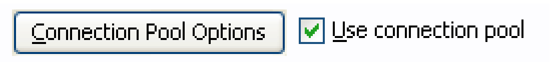
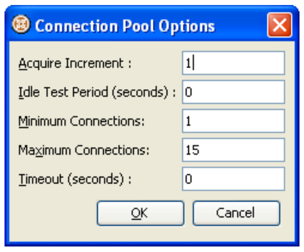


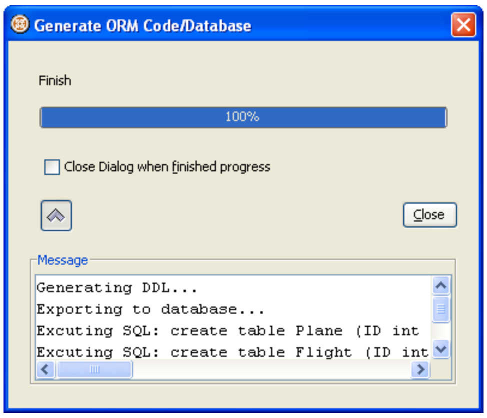
Figure 1.17 - Connection Pool Options



Acquire Increment : 1
Idle Test Period (seconds) : 0
Minimum Connections: 1
Maximum Connections: 15
Timeout (seconds) : 0

Figure 1.18 - Connection Pool Options dialog

9. Click **OK** on the dialog box, DB-VA automatically exports the database schema to the default database and generate the DDL file to the specified output path.


Generate ORM Code/Database
Finish
100%
Close Dialog when finished progress
Close
Message

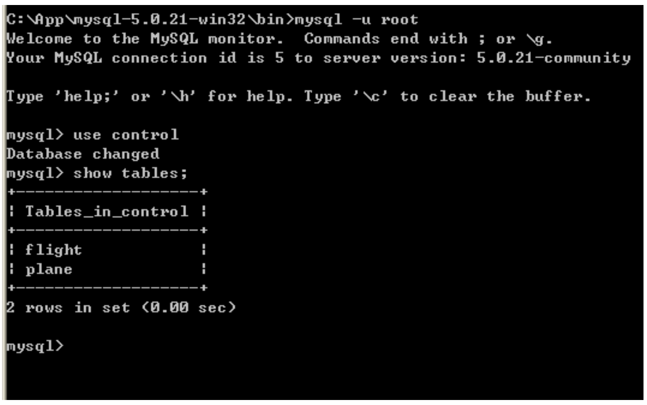
```
Generating DDL...
Exporting to database...
Excuting SQL: create table Plane (ID int
Excuting SQL: create table Flight (ID int
```

Figure 1.19 - Generate ORM Code/Database dialog

The generated DDL file is shown as below.

```
create table Flight (ID int not null auto_increment, departureTime date, departingAirport
varchar(255), arrivingAirport varchar(255), PlaneID int, primary key (ID)) type=InnoDB;
create table Plane (ID int not null auto_increment, PlaneType varchar(255), maxSpeed double
not null, maxDistance double not null, primary key (ID)) type=InnoDB;
alter table Flight add index FK_Flight_1115 (PlaneID), add constraint FK_Flight_1115 foreign
key (PlaneID) references Plane (ID);
```

If you choose to generate DDL only without exporting to database, you can later use the generated DDL to generate database manually.



```
C:\App\mysql-5.0.21-win32\bin>mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5 to server version: 5.0.21-community

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use control
Database changed
mysql> show tables;
+-------------------+
| Tables_in_control |
+-------------------+
| flight            |
| plane             |
+-------------------+
2 rows in set (0.00 sec)

mysql>
```

Figure 1.20 - The tables are generated to database.

Generating Java Code

After generating and exporting the database, you can generate the persistent Java code from the class diagram for developing your application. The following are the steps to generate Java code:

1. From the menu, select **Tools > Object-Relational Mapping (ORM) > Generate Code...** to open **Database Code Generation** dialog box.

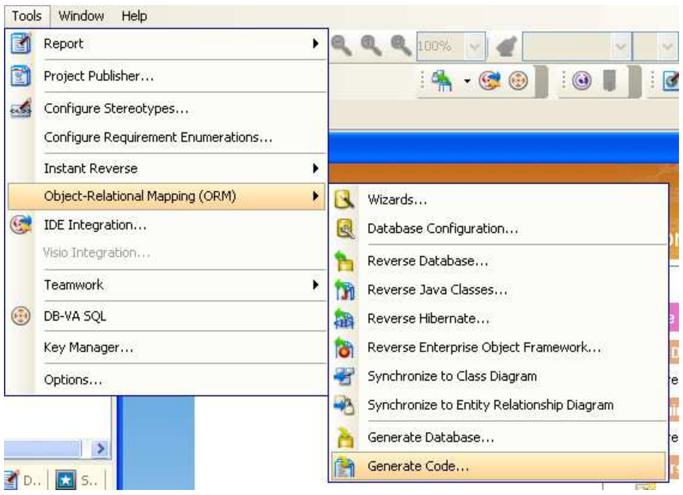
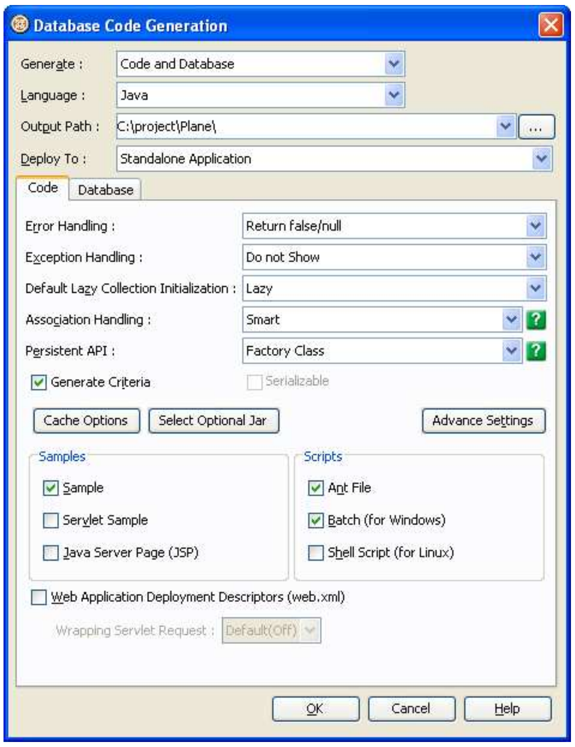


Figure 1.21 - To generate code

2. Specify the setting for Java code generation on the **Database Code Generation** dialog box.



Generate : Code and Database
Language : Java
Output Path : C:\project\Plane\
Deploy To : Standalone Application
Error Handling : Return false/null
Exception Handling : Do not Show
Default Lazy Collection Initialization : Lazy
Association Handling : Smart
Persistent API : Factory Class
[✓] Generate Criteria
[ ] Serializable
[✓] Sample
[ ] Servlet Sample
[ ] Java Server Page (JSP)
[✓] Ant File
[✓] Batch (for Windows)
[ ] Shell Script (for Linux)
[ ] Web Application Deployment Descriptors (web.xml)
Wrapping Servlet Request : Default(Off)

Figure 1.22 - Database Code Generation dialog

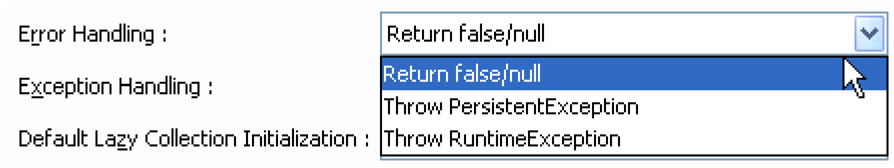
For **Output Path**, specify the location for storing the generated Java persistent code.

For **Deploy To**, specify the template setting for the purpose of the generated code. This setting affects the generated optional Jar and Datasource setting in database connection. You can select **Standalone Application**, **WebLogic Application Server 8.1/9.0**, **WebSphere Application Server Community Edition 1.0**, **JBoss Application Server** and **Generic Application server**.

Standalone Application is selected in this example.

For **Error Handling**, select the way to handle errors. The possible errors include PersistentException, GenericJDBCException, and SQLException.

- **Return false/null** - It returns false/null in the method to terminate its execution.
- **Throw PersistentException** - It throws a PersistentException which will be handled by the caller. A try/catch block has to be implemented to handle the exception.
- **Throw RuntimeException** - It throws a RuntimeException which will be handled by the caller. A try/catch block has not been implemented to handle the exception. The exception will be caught at runtime.



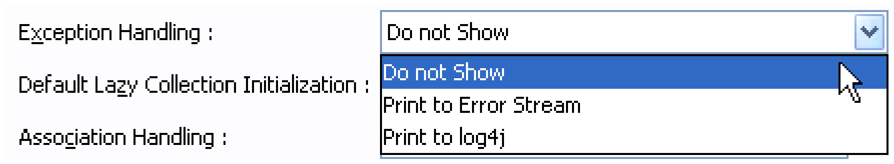
Error Handling : Return false/null

Exception Handling :
Default Lazy Collection Initialization :
Return false/null
Throw PersistentException
Throw RuntimeException

Figure 1.23 - Error Handling options

For **Exception Handling**, select how to handle the exception.

- **Do not Show** - It hides the error message.
- **Print to Error Stream** - It prints the error message to the Error Stream.
- **Print to log4j** - It prints the error message to the log4j library.



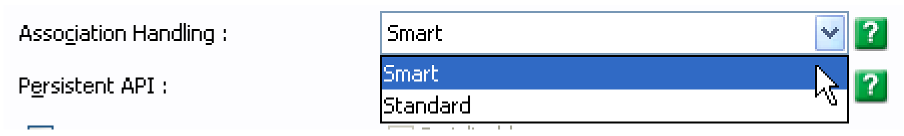
Exception Handling : Do not Show

Figure 1.24 - Exception Handling options

For **Lazy Collection Initialization**, check this option to avoid the associated objects from being loaded when the main object is loaded. Uncheck this option will result in loading the associated objects when the main object is loaded. If you enable (checked) the lazy collection initialization, all associated object (1 to many) will not be loaded until you access it (e.g. getFlight(0)). Enabling this option can usually reduce more than 80% of the database loading.

For **Association Handling**, select the type of association handling to be used, either **Smart** or **Standard**.

- **Smart** - With smart association handling, when you update one end of a bi-directional association, the generated persistent code is able to update the other end automatically. Besides, you do not need to cast the retrieved object(s) into its corresponding persistence class when retrieving object(s) from the collection.
- **Standard** - With standard association handling, you must update both ends of a bi-directional association manually to maintain the consistency of the association. Besides, casting of object(s) to its corresponding persistence class is required when retrieving object(s) from the collection.



Association Handling : Smart

Figure 1.25 - Association Handling options

For **Persistent API**, select the type of persistent code to be generated, either Static Methods, Factory Class, DAO or POJO.

- **Static Method** - Client can create, retrieve and persist with the PersistentObject directly.
- **Factory Class** - FactoryObject class will be generated for client to create and retrieve the PersistentObject. Client can directly persist with the PersistentObject.
- **DAO** - The PersistentObjectDAO class helps client to create, retrieve and persists to PersistentObject.
- **POJO** - The PersistentManager helps client to retrieve and persist with PersistentObject.

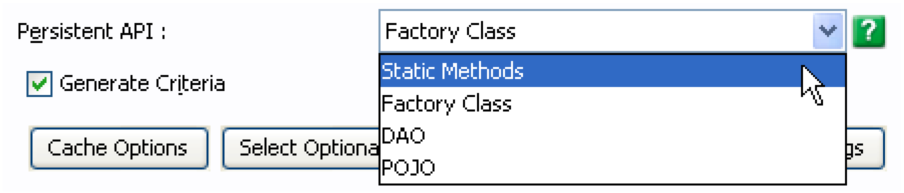


Figure 1.26 - Persistent APi options

For **Generate Criteria**, check the option for Generate Criteria to generate the criteria class for each ORM Persistable class. The criteria class is used for querying the database in object-oriented way (please see Chapter 9 for more details about using criteria)

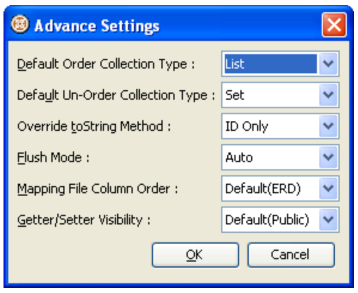
For **Samples**, sample files, including Java application sample, Servlet sample and Java Server Page (JSP) sample are available for generation. The generated sample files guide you through the usage of the Java persistence class. You can check the options to generate the sample files for reference.

You have to select samples for generation in this example so that you can modify the sample file to test and execute the generated Java code.

For **Script**, you can check the options to generate the scripts, including Ant File, Batch and Shell Script which are available for generation. You can execute the generated scripts directly.

For **Advance Setting**, you can define the Default Order Collection Type, Default Un-Order Collection Type, Override toString Method and Flush Mode.

- **Default Order Collection Type** - Select the type of ordered collection to be used in handling the multiple cardinality relationship, either **List** or **Map**.
- **Default Un-Order Collection Type** - Select the type of un-ordered collection to be used in handling the multiple cardinality relationship, either **Set** or **Bag**.
- **Override toString Method** - Select the way that you want to override the toString method of the object. There are three options provided to override the toString method as follows:
  - **ID Only** - the toString method returns the value of the primary key of the object as string.
  - **All Properties** - the toString method returns a string with the pattern "Entity[<column1_name>=<column1_value><column2_name>=(column2_value>...]"
  - **No** - the toString method will not be overridden
- **Flush Mode** - select the Flush Mode to be used in flushing strategy. User can select Auto, Commit, Always and Never.
  - **Auto** - The Session is sometimes flushed before executing query.
  - **Commit** - The Session is flushed when committing Transaction.
  - **Always** - The Session is flushed before every query.
  - **Never** - The Session is never flushed unless the flush method is called.



Advance Settings

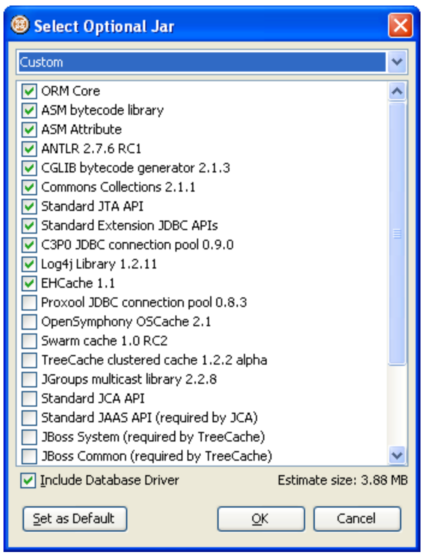
Default Order Collection Type :	List
Default Un-Order Collection Type :	Set
Override toString Method :	ID Only
Flush Mode :	Auto
Mapping File Column Order :	Default(ERD)
Getter/Setter Visibility :	Default(Public)

Figure 1.27 - Advance Setting dialog

Selecting Optional Jar for Persistent Library

DB-VA can generate persistent code and library for you. You are allowed to select the libraries and JDBC driver to be included in the generation of the orm.jar file (Persistent Library).

1. Click **Select Optional Jar** button to open the **Select Optional Jar** dialog box. You can select the **Default** template which contains most of the useful libraries for the orm.jar file.


Select Optional Jar

Custom

- ✓ ORM Core
- ✓ ASM bytecode library
- ✓ ASM Attribute
- ✓ ANTLR 2.7.6 RC1
- ✓ CGLIB bytecode generator 2.1.3
- ✓ Commons Collections 2.1.1
- ✓ Standard JTA API
- ✓ Standard Extension JDBC APIs
- ✓ C3P0 JDBC connection pool 0.9.0
- ✓ Log4j Library 1.2.11
- ✓ EHCache 1.1
- ☐ Proxool JDBC connection pool 0.8.3
- ☐ OpenSymphony OSCache 2.1
- ☐ Swarm cache 1.0 RC2
- ☐ TreeCache clustered cache 1.2.2 alpha
- ☐ JGroups multicast library 2.2.8
- ☐ Standard JCA API
- ☐ Standard JAAS API (required by JCA)
- ☐ JBoss System (required by TreeCache)
- ☐ JBoss Common (required by TreeCache)

✓ Include Database Driver Estimate size: 3.88 MB

Figure 1.28 - Select Optional Jar dialog

2. Select the desired template from the drop-down menu for creating the orm.jar file.

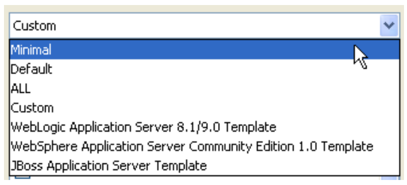

Custom
Minimal
Default
ALL
Custom
WebLogic Application Server 8.1/9.0 Template
WebSphere Application Server Community Edition 1.0 Template
JBoss Application Server Template

Figure 1.29 - Select the desired template

3. Select **Include Database Driver,** the Database Driver (JDBC driver of the default database) will be included in the generated orm.jar file. After configuring the generation of Java code option, press **OK** button to start generating the Java persistent code. The generated **src** folder contains the Java persistent source code and the **lib** folder contains the persistent library.

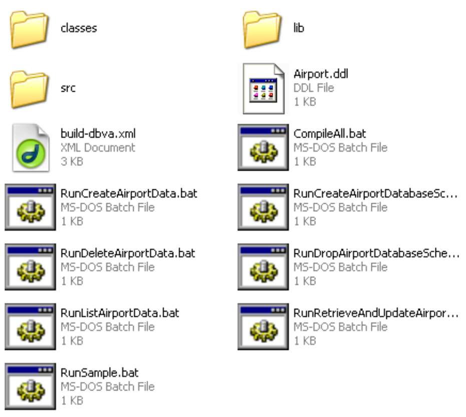

classes
lib
src
Airport.ddl DDL File 1 KB
build-dbva.xml XML Document 3 KB
CompileAll.bat MS-DOS Batch File 1 KB
RunCreateAirportData.bat MS-DOS Batch File 1 KB
RunCreateAirportDatabaseSc... MS-DOS Batch File 1 KB
RunDeleteAirportData.bat MS-DOS Batch File 1 KB
RunDropAirportDatabaseSche... MS-DOS Batch File 1 KB
RunListAirportData.bat MS-DOS Batch File 1 KB
RunRetrieveAndUpdateAirpor... MS-DOS Batch File 1 KB
RunSample.bat MS-DOS Batch File 1 KB

Figure 1.30 - The generated files

2 Configuring Source and Library in Eclipse

Chapter 2 - Configuring Source and Library in Eclipse

In Chapter 1, you have generated Java code, exported database schema (DDL) and persistent library. In this chapter, we will show you how to import the generated code from Chapter 1 to Eclipse. You will create a new Java project and copy the generated persistent code and library to Eclipse, and you can further your development with the generate model efficiently in Eclipse.

In this chapter:

- Copying Generated Source and Library to Eclipse Project
- Configuring the Library
- Modifying the Sample Program to Test the Generated Java Model

Copying Generated Source and Library to Eclipse Project

You can copy the generated persistent code to Eclipse project and develop an application.

1. Open Eclipse and select **File** > **New** > **Project...** from menu

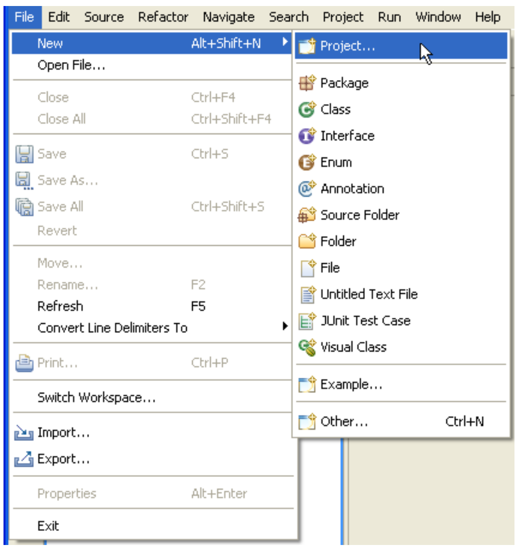


Figure 2.1 - Create a project

2. Select **Java Project**, click **Next >.**

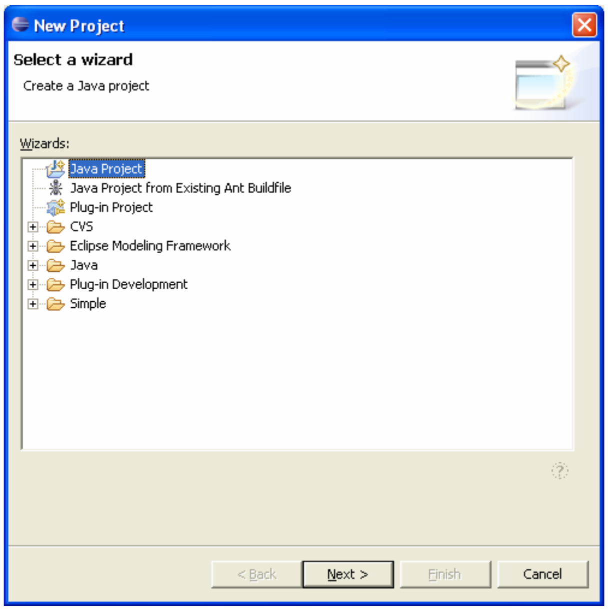
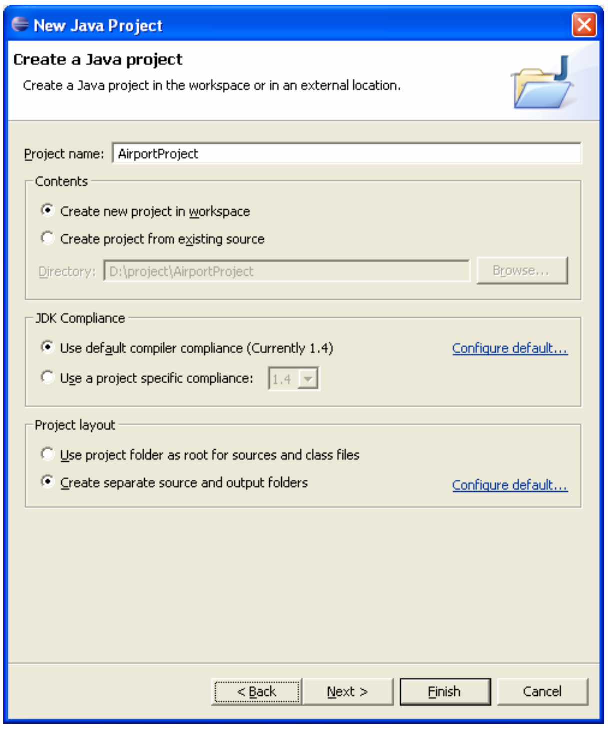


Figure 2.2 - Select the project type as Java Project

3. Enter "**AirportProject"** for the project name and select **Create separate source and output folders** for the **Project layout** option**,** click **Next >.**


New Java Project

Create a Java project

Create a Java project in the workspace or in an external location.

Project name: AirportProject

Contents
- (●) Create new project in workspace
- ( ) Create project from existing source
- Directory: D:\project\AirportProject Browse...

JDK Compliance
- (●) Use default compiler compliance (Currently 1.4) Configure default...
- ( ) Use a project specific compliance: 1.4

Project layout
- ( ) Use project folder as root for sources and class files
- (●) Create separate source and output folders Configure default...

< Back Next > Finish Cancel

Figure 2.3 - Enter the project name

4. Modify the **Default output folder** from "**AirportProjet/bin**" to "**AirportProject/classes**" , click **Finish**.

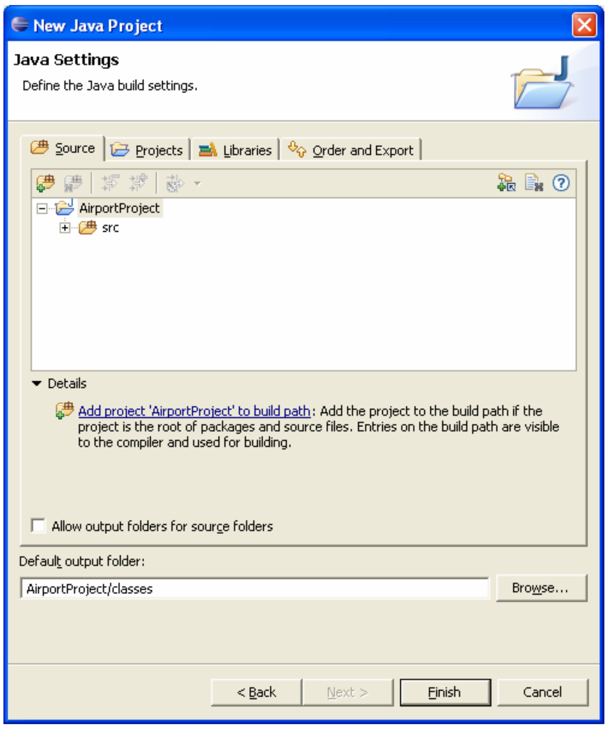

New Java Project
Java Settings
Define the Java build settings.
Source
Projects
Libraries
Order and Export
AirportProject
src
Details
Add project 'AirportProject' to build path: Add the project to the build path if the project is the root of packages and source files. Entries on the build path are visible to the compiler and used for building.
Allow output folders for source folders
Default output folder:
AirportProject/classes
Browse...
< Back
Next >
Finish
Cancel

Figure 2.4 - Configure the project path

5. The AirportProject is created and shown on the **Package Explorer**.

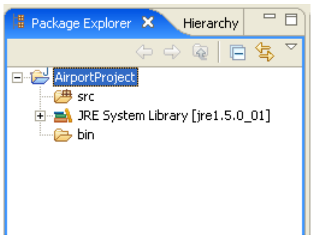


Figure 2.5 - A blank new project created

6. Copy the Java persistent code from the specified output folder.

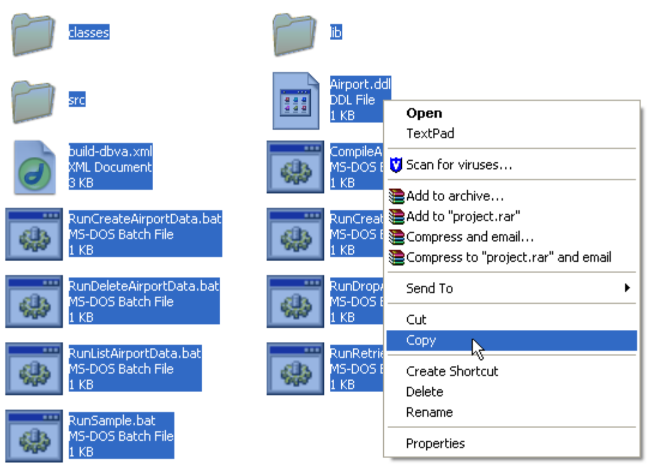


Figure 2.6 - Copy all generated files

7. Paste the Java persistent code on the AirportProject in **Package Explorer**.

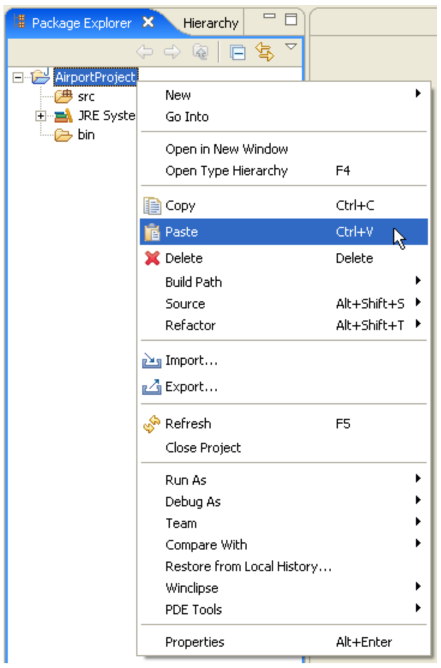


Figure 2.7 - Paste in the Eclipse project

8. All the generated Java persistent codes are copied to the AirportProject but errors exist in the project.

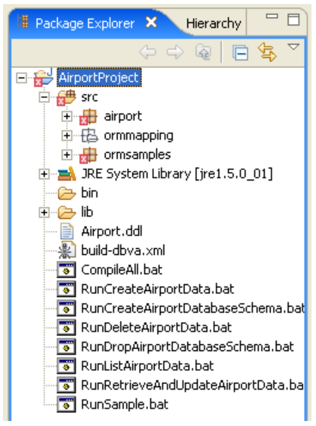
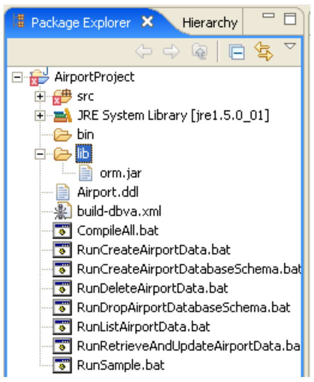


Figure 2.8 - The files is copied to the eclipse project

Configuring the Library

You can observe that the generated Java persistent code in the AirportProject has errors because the persistent library for the AirportProject has not been specified.

1. Locate the **orm.jar** in the **lib** folder



```
Package Explorer  Hierarchy
AirportProject
  src
  JRE System Library [jre1.5.0_01]
  bin
  lib
    orm.jar
  Airport.ddl
  build-dbva.xml
  CompileAll.bat
  RunCreateAirportData.bat
  RunCreateAirportDatabaseSchema.bat
  RunDeleteAirportData.bat
  RunDropAirportDatabaseSchema.bat
  RunListAirportData.bat
  RunRetrieveAndUpdateAirportData.ba
  RunSample.bat
```

Figure 2.9 - The orm.jar

2. Right click **orm.jar**, select **Build Path** > **Add to Build Path** from the pop-up menu.

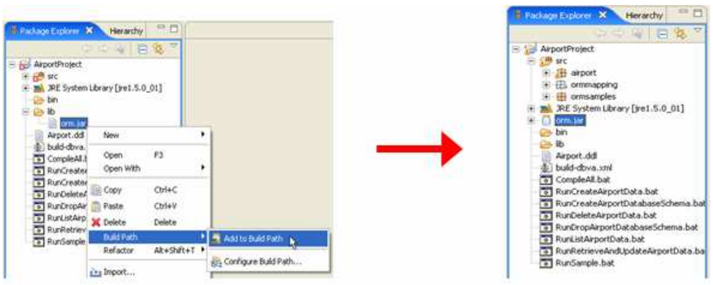


Figure 2.10 - Add the orm.jar to project classpath

As the library has been configured with the persistent library, the errors found in the Java persistent code are cleared.

Modifying the Sample Program to Test the Generated Java Model

Now the AirportProject is able to compile, you can modify the code in generated sample to test the persistent class interact with the created database.

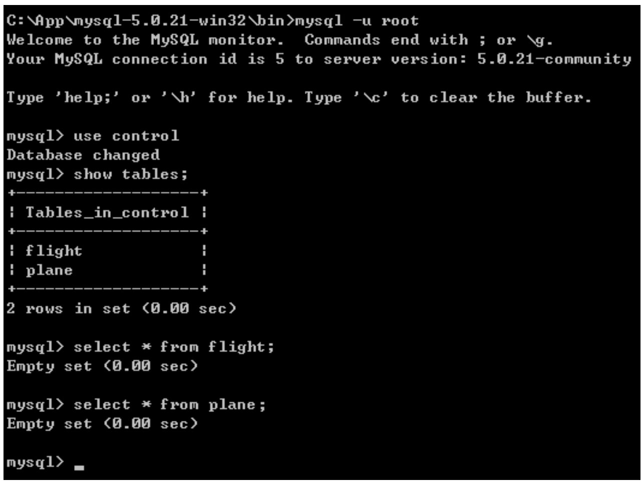
1. Open the **src/ormsamples/CreateAirportData.java** file. The following is the original code

```
public void createTestData() throws PersistentException {
       PersistentTransaction t =
       airport.AirportPersistentManager.instance().getSession().beginTransaction();
       try {
              airport.Flight lairportFlight = airport.FlightFactory.createFlight();
              // Initialize the properties of the persistent object
              lairportFlight.save();

              airport.Plane lairportPlane = airport.PlaneFactory.createPlane();
              // Initialize the properties of the persistent object
              lairportPlane.save();

              t.commit();
       }
       catch (Exception e) {
              e.printStackTrace();
              t.rollback();
       }
}
```

2. Modify to create Flight and Plane instance and create the relationship between them

```
public void createTestData() throws PersistentException {
       PersistentTransaction t =
       airport.AirportPersistentManager.instance().getSession().beginTransaction();
       try {
              System.out.println("Create the Flight");
              airport.Flight lairportFlight = airport.FlightFactory.createFlight();
              // Initialize the properties of the persistent object
              lairportFlight.setArrivingAirport("Hong Kong International Airport");
              lairportFlight.setDepartingAirport("Kansai International Airport");
              lairportFlight.setDepartureTime(new Date());

              System.out.println("Create the Plane");
              airport.Plane lairportPlane = airport.PlaneFactory.createPlane();
              // Initialize the properties of the persistent object
              lairportPlane.setPlaneType("747 plane");
              lairportPlane.setMaxSpeed(967);
              lairportPlane.setMaxDistance(8232);

              System.out.println("Create the relationship between Flight and Plane");
              lairportFlight.setPlane(lairportPlane);

              System.out.println("Save the Plane and the Flight object");
              lairportPlane.save();

              System.out.println("Commit the Transaction");
              t.commit();
       }
       catch (Exception e) {
              e.printStackTrace();
              t.rollback();
       }
}
```

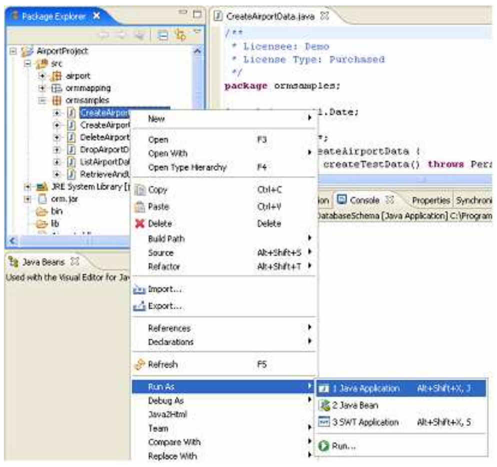
There is no record for the Flight and Plane tables before the execution of the sample application.



```
C:\App\mysql-5.0.21-win32\bin>mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5 to server version: 5.0.21-community

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use control
Database changed
mysql> show tables;
+-------------------+
| Tables_in_control |
+-------------------+
| flight            |
| plane             |
+-------------------+
2 rows in set (0.00 sec)

mysql> select * from flight;
Empty set (0.00 sec)

mysql> select * from plane;
Empty set (0.00 sec)

mysql> _
```

Figure 2.11 - no record in the table

3. Right click the CreateAirportData.java, select **Run As** > **Java Application** from the pop-up menu to execute it.



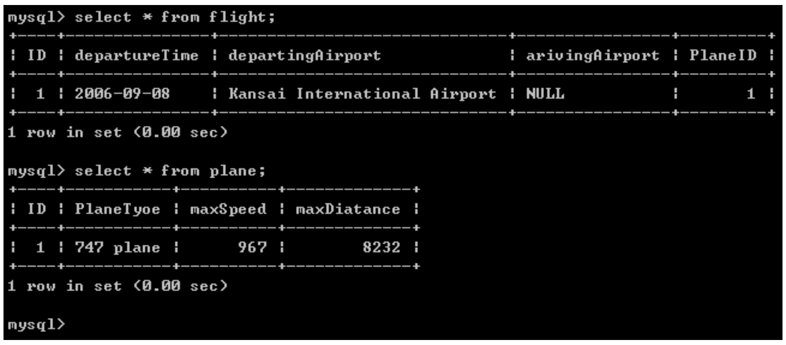
```
/**
 * Licensee: Demo
 * License Type: Purchased
 */
package ormsamples;
```

Figure 2.12 - Run the application

4. The execution result:

```
Create the Flight
Create the Plane
Create the relationship between Flight and Plane
Save the Plane and the Flight object
Commit the Transaction
```

5. After executing the CreateAirportData.java, new records are created in tables.



```
mysql> select * from flight;
+----+---------------+----------------------------+----------------+---------+
| ID | departureTime | departingAirport           | arivingAirport | PlaneID |
+----+---------------+----------------------------+----------------+---------+
|  1 | 2006-09-08    | Kansai International Airport | NULL         |       1 |
+----+---------------+----------------------------+----------------+---------+
1 row in set (0.00 sec)

mysql> select * from plane;
+----+-----------+----------+-------------+
| ID | PlaneTyoe | maxSpeed | maxDiatance |
+----+-----------+----------+-------------+
|  1 | 747 plane |      967 |        8232 |
+----+-----------+----------+-------------+
1 row in set (0.00 sec)

mysql>
```

3 Developing Java Enterprise Web Application

Chapter 3 - Developing Java Enterprise Web Application

With the DB Visual ARCHITECT (DB-VA) you can develop quality Java Enterprise Web Application much faster, better and cheaper. All DB-VA generated code, configuration file and persistent layer library are deployable to most application servers. DB-VA generates all Java code for accessing database. You do not need to write SQL to insert, query, update or delete the record. All code you need to program is plain Java code (e.g. OrderDAO.save(myOrder);). In this chapter we will use a simple "School System" application to show you how to generate Java code, configure your web application, creating, querying, updating and deleting objects. Again you do not need to write a single SQL statement for all the above operations. In the final section we will discuss some issues about the HTTP Session.

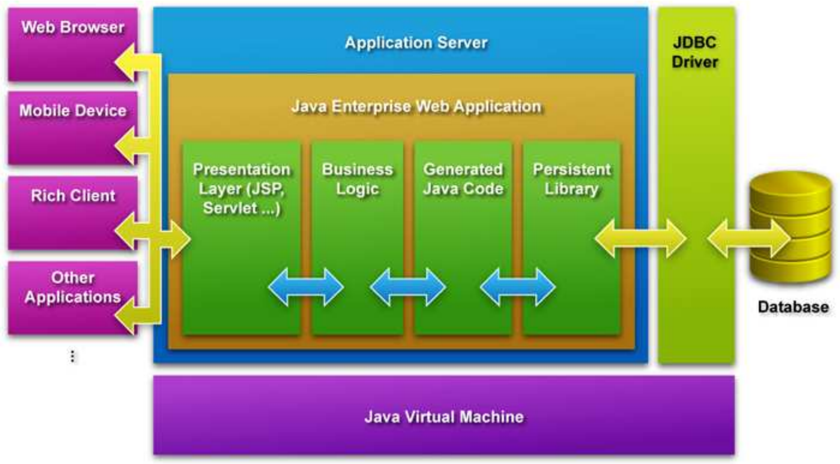


Figure 3.1 - Java Enterprise Web Application

The architecture of Java Enterprise Web Application with DB-VA Persistent Layer

In this chapter:

- Introduction
- Configuring Filter in web.xml
- Creating Object and Saving to Database
- Querying Object from Database
- Updating Object and Saving to Database
- Deleting Object in Database
- Using Persistent Object with HttpSession

Introduction

You will develop a School System.

The School System provides the following functions:

- Create course by teacher
- Enroll course for student
- Cancel course by teacher
- Register for user
- Modify the personal information
- View the Course information (number of students enrolled and teacher information of the course)

Required Software:

- DB Visual ARCHITECT 3.0 Java or Professional Edition (http://www.visual-paradigm.com/download/)
- Sun J2SE 1.4 JDK or above (http://java.sun.com/)
- JBoss Application Server 4.0.2 or above (http://www.jboss.com/products/jbossas/downloads)
- MySQL Server 4.1 or above (http://dev.mysql.com/downloads/)

Please open the SchoolSystem.vpp project file in the Chapter 3 School System.zip file. The project file contains the following diagrams. For the details about how to draw class diagram and entity relationship diagram, please see the Designer's Guide.

Class Diagram of School System:

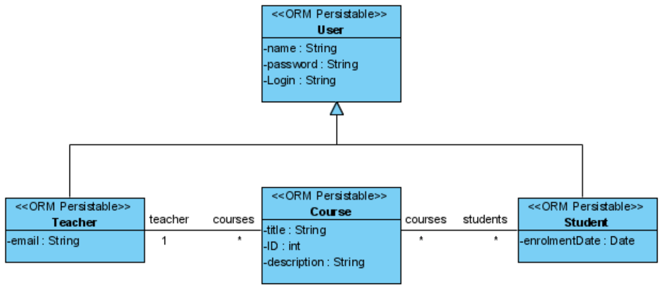


Figure 3.2 - Class Diagram of School System

Entity Relationship Diagram of School System:

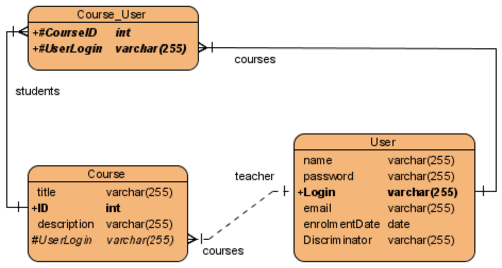


Figure 3.3 - ERD of School System

Configuring Web Application Deployment Descriptor

Each Java Web Application has Web Application Deployment Descriptor (web.xml) to configure the deployment and startup of the web application. Please select "Web Application Deployment Descriptors (web.xml)" option to generate the web.xml for your web application.

1. From menu bar, select **Tools > Object-Relational Mapping (ORM) > Generate Code...** to open the **Database Code Generation** dialog box.

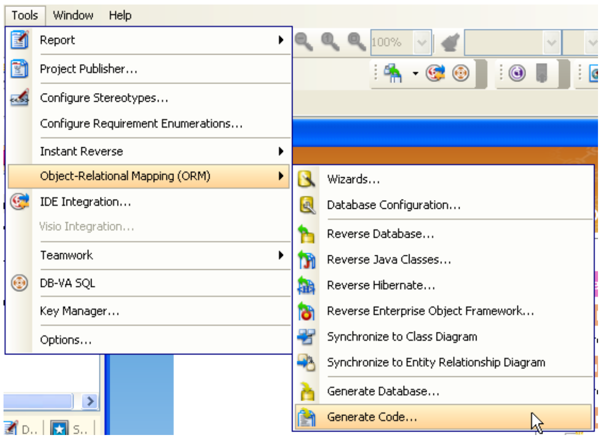


Figure 3.4 - To generate code

2. Fill in code generation information.

Code tab:

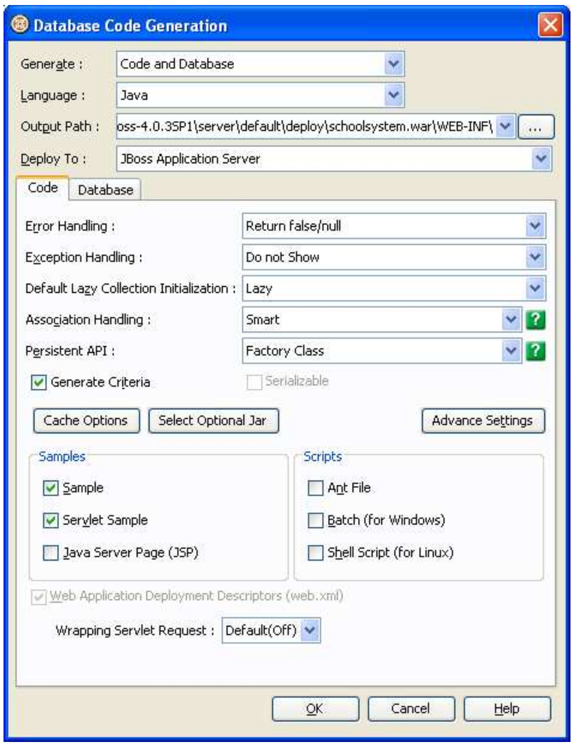
For **Generate**, select **Code and Database** to generate code and create database option.

For **Language**, select **Java** language.
For **Output Path**, select the `JBOSS_HOME_DEPLOY_FOLDER\schoolsystem.war\WEB-INF`

For example:

`C:\DevelopApps\jboss-4.0.3SP1\server\default\deploy\schoolsystem.war\WEB-INF\`

For **Deploy To**, select **JBoss Application Server**, DB-VA will help you to select the optimize option Jar to create orm.jar (persistent library).
For **Association Handling**, select **Smart**.
For **Persistent API**, select **Factory Class**.
For **Sample**, check **Sample** and **Servlet Sample**.
Select **Web Application Deployment Descriptors (web.xml)** to generate web.xml with filter configure.


Database Code Generation

Generate : Code and Database
Language : Java
Output Path : oss-4.0.3SP1\server\default\deploy\schoolsystem.war\WEB-INF\
Deploy To : JBoss Application Server

Code | Database

Error Handling : Return false/null
Exception Handling : Do not Show
Default Lazy Collection Initialization : Lazy
Association Handling : Smart
Persistent API : Factory Class
[✓] Generate Criteria [ ] Serializable

Samples
[✓] Sample
[✓] Servlet Sample
[ ] Java Server Page (JSP)

Scripts
[ ] Ant File
[ ] Batch (for Windows)
[ ] Shell Script (for Linux)

[✓] Web Application Deployment Descriptors (web.xml)
Wrapping Servlet Request : Default(Off)

Figure 3.5 - Generate code configure for generate web application

Database tab:

You can reference Chapter 1 to select export the database schema and configure the default database of JDBC connection.

3. The code is generated to **JBOSS_DEPLOY_FOLDER\schoolsystem.war\.**
4. Start Eclipse Platform, create new Java project

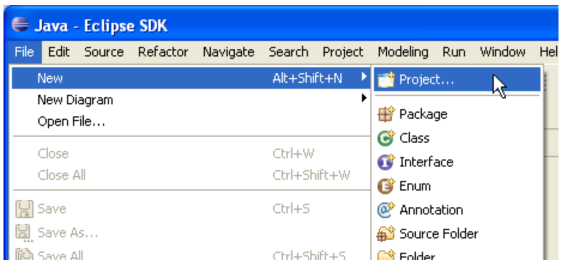


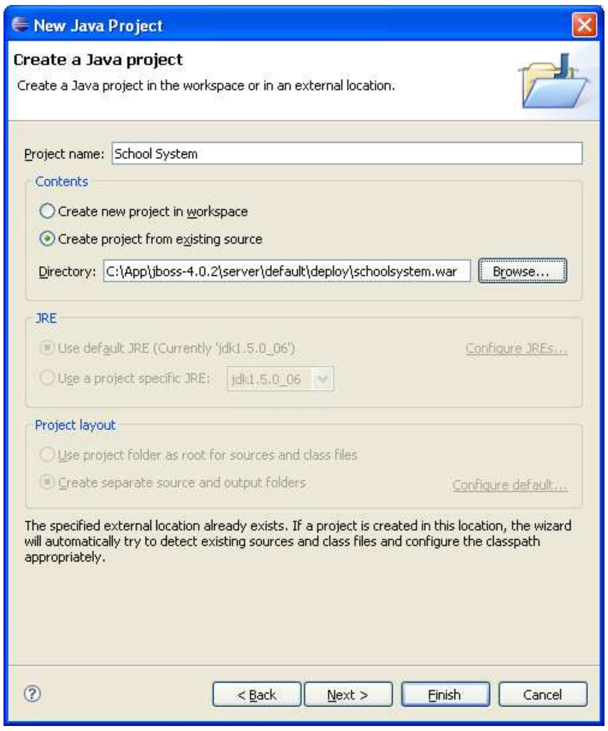
Figure 3.6 - Create a new Eclipse project

5. Enter new project information.

   For **Project name**, Enter "School System"

   Select **Create project from existing source**. In **Directory**, select the **JBOSS_DEPLOY_FOLDER\schoolsystem.war** folder.

   Click **Next >.**


New Java Project

Create a Java project

Create a Java project in the workspace or in an external location.

Project name: School System

Contents

Create new project in workspace [✓ Create project from existing source]

Directory: C:\App\jboss-4.0.2\server\default\deploy\schoolsystem.war

JRE

Use default JRE (Currently 'jdk1.5.0_06')

Use a project specific JRE: jdk1.5.0_06

Project layout

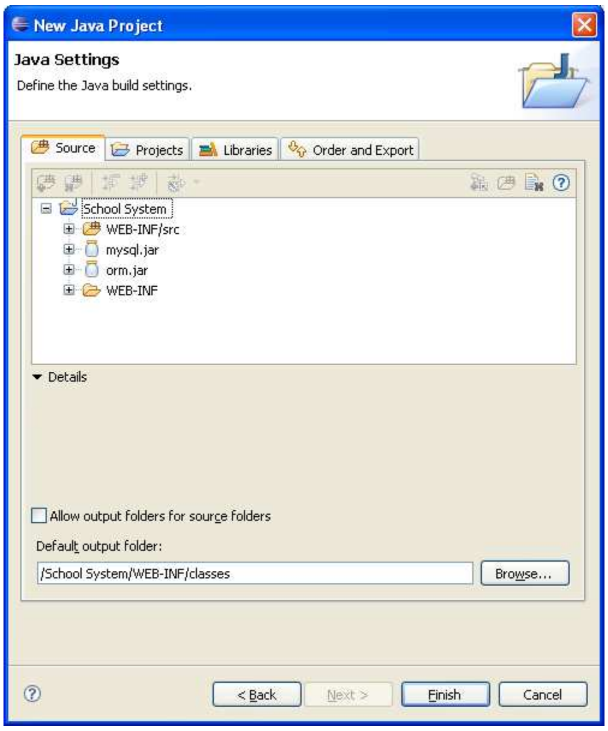
Use project folder as root for sources and class files

Create separate source and output folders

The specified external location already exists. If a project is created in this location, the wizard will automatically try to detect existing sources and class files and configure the classpath appropriately.

Figure 3.7 - Select directory of the generated code

6. Modify the **Default output folder** to **/School System/WEB-INF/classes.**


New Java Project

Java Settings

Define the Java build settings.

School System
- WEB-INF/src
- mysql.jar
- orm.jar
- WEB-INF

Allow output folders for source folders

Default output folder: /School System/WEB-INF/classes

Figure 3.8 - Select the output path and src path

7. Click **Finish** to create the project. In **Package Explorer**, right click **School System** and select **Properties**.

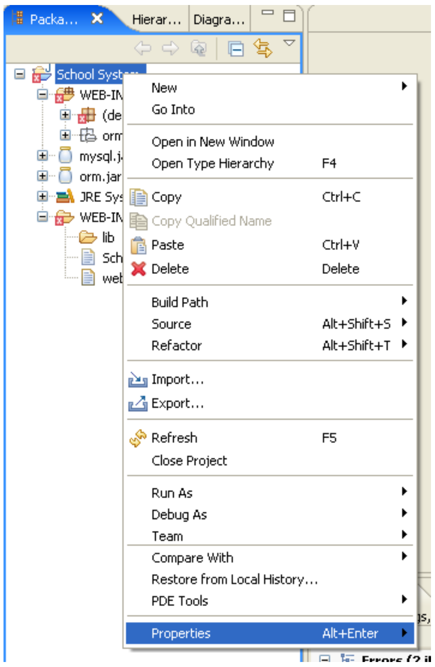


Figure 3.9 - Open the project properties

8. Select **Java Build Path > Libraries > Add External JARs...**

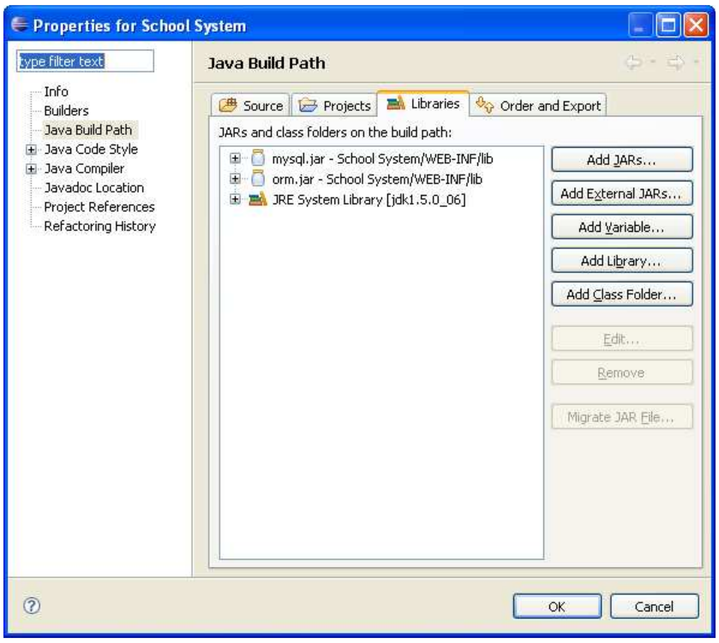
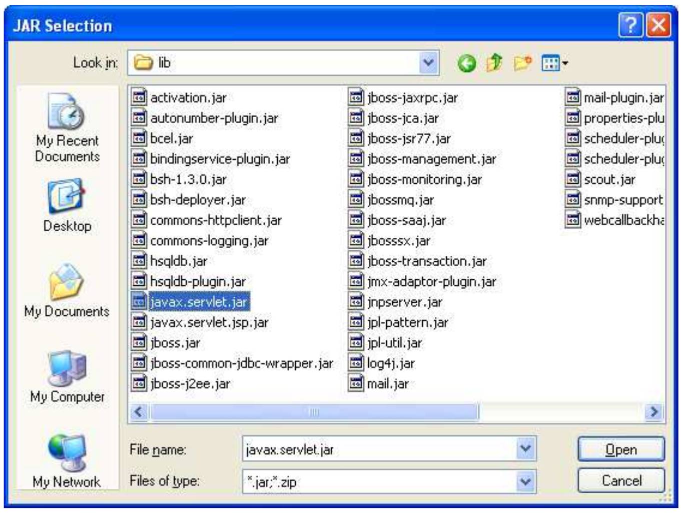


Figure 3.10 - Add an external Jar

9. Select **javax.servlet.jar** in **JBOSS_SERVER\client** folder



File name: javax.servlet.jar

Files of type: *.jar;*.zip

Figure 3.11 - Select the external Jar

10. Right click **WEB-INF\src\WEB-INF\web.xml** to select **Refactor > Move...**

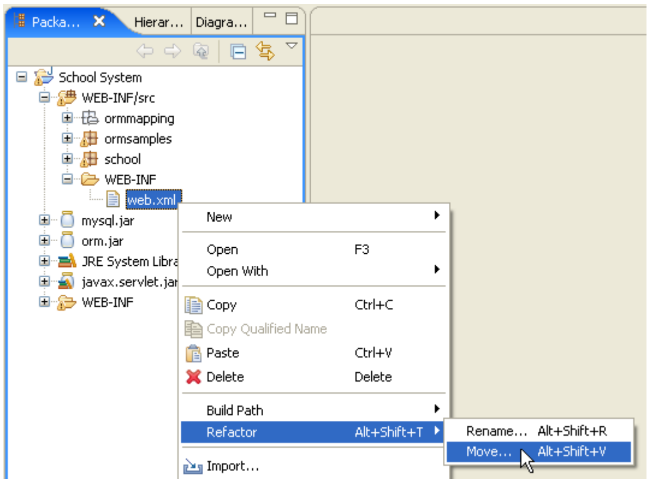
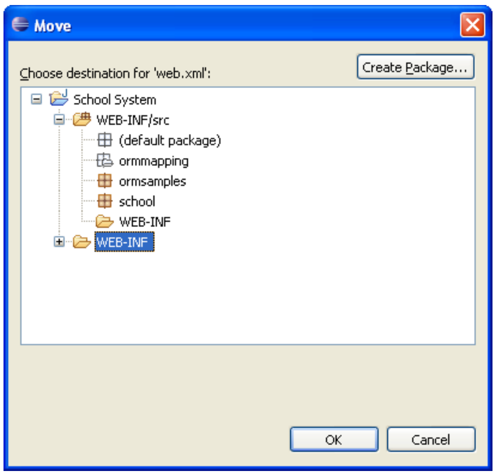


Figure 3.12 - To move the file

11. Select move to **School System\WEB-INF**.



Choose destination for 'web.xml':

Figure 3.13 - Select the destination directory

12. Open the web.xml file which contains the filter information of filter class and filter mapping

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
       <display-name>SchoolSystem</display-name>
       <description>SchoolSystem</description>
       <filter>
              <filter-name>ORMFilter</filter-name>
              <filter-class>school.SchoolSystemFilter</filter-class>
       </filter>
       <filter-mapping>
              <filter-name>ORMFilter</filter-name>
              <url-pattern>/*</url-pattern>
       </filter-mapping>
```

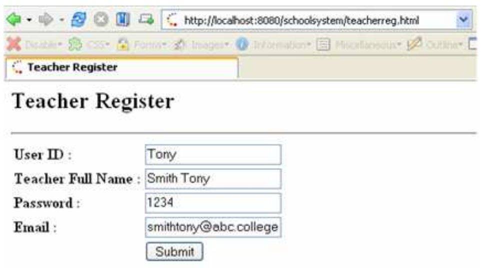
Creating Object and Saving to Database

The school system provides a separate register page for teacher and student to enter their information. The register method of teacher and student method is the same, so we will demonstrate how to create the teacher and save to database.

1. Create "teacherreg.html" for teacher to input their personal information and register to the system. The submit information will be processed by CreateUser Servlet to add new user.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
       <head>
              <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
              <title>Teacher Register</title>
       </head>
       <body>
              <h2>Teacher Register</h2>
              <hr>
              <form action="servlet/createUser" method="post">
                     <table border="0">
                            <tr><td><strong>User ID : </strong></td><td><input
                            name="loginID" type="text" id="loginID"/></td></tr>
                            <tr><td><strong>Teacher Full Name : </strong></td><td><input
                            name="name" type="text" id="name"/></td></tr>
                            <tr><td><strong>Password : </strong></td><td><input
                            name="password" type="text" id="password"/></td></tr>
                            <tr><td><strong>Email : </strong></td><td><input name="email"
                            type="text" id="email"/></td></tr>
                            <tr><td></td><td><input type="submit" name="Submit"
                            value="Submit"></td></tr>
                     </table>
                     <input name="userType" type="hidden" value="teacher"/>
              </form>
       </body>
</html>
```

Source File : School System\teacherreg.html



Teacher Register

User ID :	Tony
Teacher Full Name :	Smith Tony
Password :	1234
Email :	smithtony@abc.college
	Submit

Figure 3.14 - The teacherreg.html view in browser

2. Create the Servlet **CreateUser.java** in **WEB-INF\src** folder in **util** package to create the teacher record in system and provide the unique user ID.
   - The CreateUser.java can identify to create teacher or student by hidden field in "teacherreg.html"

     ```
     <input name="userType" type="hidden" value="teacher"/>
     ```

   - The CreateUser.java contains the addUser() method to create teacher and student. It gets the information from the request

     ```
     String lName = request.getParameter("name");
     String lLoginID = request.getParameter("loginID");
     String lPassword = request.getParameter("password");
     String lUserType = request.getParameter("userType");
     ```

     Source File : School System\WEB-INF\src\util\CreateUser.java

   - Notify the session begin transaction and then use the TeacherFactory class to create Teacher instance.

     ```
     PersistentTransaction t =
     school.SchoolSystemPersistentManager.instance().getSession().beginTransaction();
     String lEmail = request.getParameter("email");
     Teacher lTeacher = TeacherFactory.createTeacher();
     lTeacher.setLoginID(lLoginID);
     lTeacher.setEmail(lEmail);
     lTeacher.setName(lName);
     lTeacher.setPassword(lPassword);
     ```

     Source File : School System\WEB-INF\src\util\CreateUser.java

   - Call save() method of Teacher instance to create the record in database. Add Teacher instance to session to represent the user has login to the system.

     ```
     lTeacher.save();
     lUser = lTeacher;
     request.getSession().setAttribute("user", lUser);
     t.commit();
     ```

     Source File : School System\WEB-INF\src\util\CreateUser.java

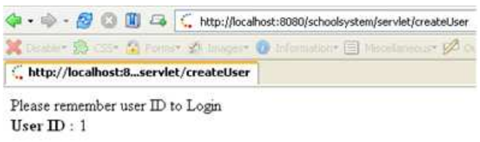
3. Configure the CreateUser.java Servlet in web.xml. After saved the web.xml, you can use /servlet/createUser to call the CreateUser.java Servlet.

   ```
   ...
   <servlet>
          <servlet-name>CreateUser</servlet-name>
          <servlet-class>util.CreateUser</servlet-class>
   </servlet>
   <servlet-mapping>
          <servlet-name>CreateUser</servlet-name>
          <url-pattern>/servlet/createUser</url-pattern>
   </servlet-mapping>
   ...
   ```

   Source File : School System\WEB-INF\web.xml

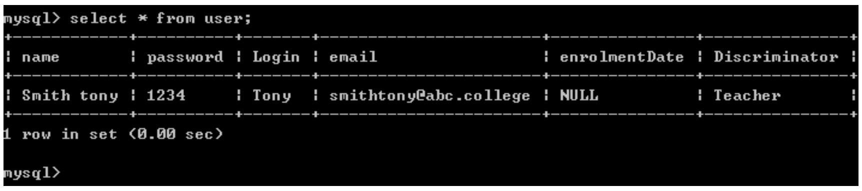
4. When you submit the form the Teacher object will be added to the database. The new user object is added to the Session. If user attribute exists in session, the user is login to the system.

```
request.getSession().setAttribute("user", lUser);
```


http://localhost:8080/schoolsystem/servlet/createUser
http://localhost:8...servlet/createUser

Please remember user ID to Login
User ID : 1

Figure 3.15 - The successful message



```
mysql> select * from user;
+------------+----------+-------+----------------------+---------------+---------------+
| name       | password | Login | email                | enrolmentDate | Discriminator |
+------------+----------+-------+----------------------+---------------+---------------+
| Smith tony | 1234     | Tony  | smithtony@abc.college | NULL         | Teacher       |
+------------+----------+-------+----------------------+---------------+---------------+
1 row in set (0.00 sec)

mysql>
```

Figure 3.16 - The record is added to database

Querying Object from Database

You can retrieve the record in database as object. For example, you need to create the login function for the School System. You will require the user to input the User ID and password to login, the system retrieve the User object from the user id and compare the password to validate the user.

1. Create the Login page (login.jsp). It submits the form to the Login Servlet. If user has login, it will redirect the user to student page or teacher page depends on the user type.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Login Page</title>
</head>
<body>
<%
       school.User lUser = (school.User)request.getSession().getAttribute("user");
       if (lUser == null){
%>
<h2>Login</h2>
<hr>
<form action="servlet/login" method="post">
<table border="0">
<tr><td><strong>Login ID : </strong></td><td><input name="id" type="text"
id="id"/></td></tr>
<tr><td><strong>Password :</strong></td><td><input name="password" type="password"
id="password"/></td></tr>
<tr><td></td><td><input type="submit" name="Submit" value="Submit"></td></tr>
</table>
</form>
<%
       } else if (lUser instanceof school.Student){
              response.sendRedirect("studentPage.jsp");
       } else if (lUser instanceof school.Teacher){
              response.sendRedirect("teacherPage.jsp");
       }
%>
</body>
</html>
```

Source File : School System\login.jsp

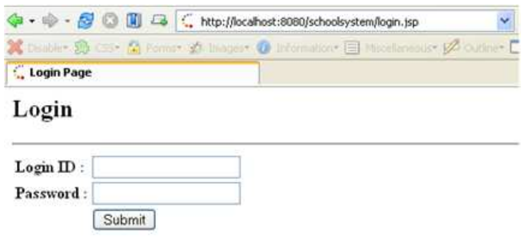


Figure 3.17 - The login.jsp view in browser

2. Use User ID to call loadUserByORMID() method to retrieve the User object in Login Servlet and compare the password

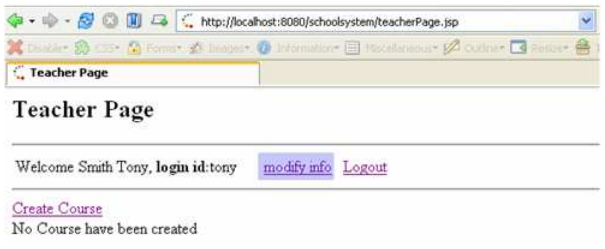
```
if (lUser != null && lUser.getPassword().equals(lPassword)) {
	request.getSession().setAttribute("user", lUser);
	if (lUser instanceof Student) {
		response.sendRedirect("../studentPage.jsp");
	} else {
		response.sendRedirect("../teacherPage.jsp");
	}
}
```

Source File : School System\WEB-INF\src\util\Login.java

Updating Object and Saving to Database

You can modify the teacher information and update the record in database. You get the User object from the session and set the new values for the User object, finally call save() method to update the record in database.

1. Click modify info link on the Teacher Page



Welcome Smith Tony, **login id**:tony

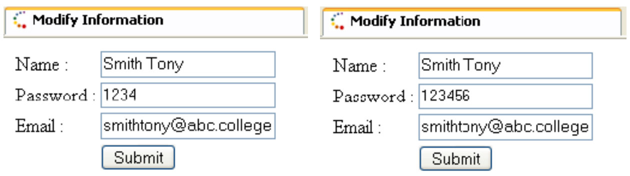
No Course have been created

Figure 3.18 - The teacherPage.jsp

2. Modify the information in modifyuser.jsp. Get the User object from session and set the information to field and allow the user to modify the information, finally submit the form to the ModifyUser Servlet.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"> <html>
<head>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Modify Information</title>
</head>
<body>
<%
	boolean isStudent = false;
	Object lUserObj = request.getSession().getAttribute("user");
	if (lUserObj == null){
		response.sendRedirect("login.jsp");
	} else {
		school.User lUser = (school.User)lUserObj;
%>
<form action="servlet/modifyUser" method="post">
<table border="0">
<tr><td>Name : </td><td><input name="name" type="text" id="studentName" value="<%=
lUser.getName() %>"/></td></tr>
<tr><td>Password :</td><td><input name="password" type="text" id="password" value="<%=
lUser.getPassword() %>"/></td></tr>

<% if (lUser instanceof school.Teacher){ %>

<tr><td>Email :</td><td><input name="email" type="text" id="email" value="<%=
((school.Teacher)lUser).getEmail()%>"/></td></tr>

<%} %>

<tr><td></td><td><input type="submit" name="Submit" value="Submit"></td></tr>
</table>
</form>
<% }%>
</body>
</html>
```

Source File : School System\modifyUser.jsp



Modify Information

Name :	Smith Tony
Password :	1234
Email :	smithtony@abc.college
	Submit

Modify Information

Name :	Smith Tony
Password :	123456
Email :	smithtony@abc.college
	Submit

Figure 3.19 - The password of user changed

3. The ModifyUser Servlet get the modified information from the submitted form and set to the User object and call save() method to update the record in database. It is similar to create user.

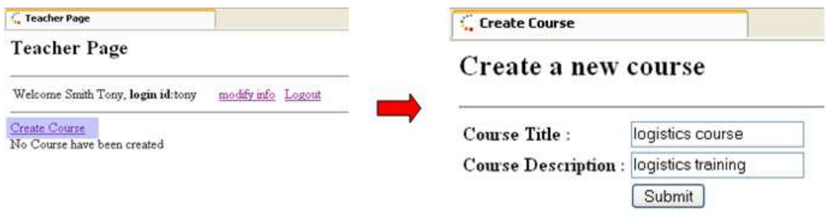
```
private boolean modifyUser(HttpServletRequest request,
HttpServletResponse response, Object aUserObj) throws PersistentException {
        PersistentTransaction t =
        SchoolSystemPersistentManager.instance().getSession().beginTransaction();
        boolean lSuccess = false;
        try {
                String lUserName = request.getParameter("name");
                String lPassword = request.getParameter("password");
                if (lUserName != null && lUserName.length() > 0 && lPassword != null &&
                lPassword.length() > 0) {
                        User lUser = (User) aUserObj;
                        lUser.setName(lUserName);
                        lUser.setPassword(lPassword);
                        if (lUser instanceof Teacher) {
                                ((Teacher) lUser).setEmail(request.getParameter("email"));
                        }
                        lUser.save();
                        lSuccess = true;
                } else {
                        lSuccess = false;
                }
                t.commit();
        } catch (Exception e) {
                e.printStackTrace();
                t.rollback();
        }
        return lSuccess;
}
```

Source File : School System\WEB-INF\src\util\ModifyUser.java

Deleting Object in Database

Teacher can create course for students for registration and they can cancel the course in the system. They only need to click cancel hyperlink of the course then the course information will be deleted in the database and all its relationship with registered students will be removed.

1. Teacher can create the course by selecting Create Course hyperlink in teacher page. Fill in Course name and Description to create Course.



Teacher Page

Welcome Smith Tony, login id:tony modify info Logout

Create Course

No Course have been created

Create a new course

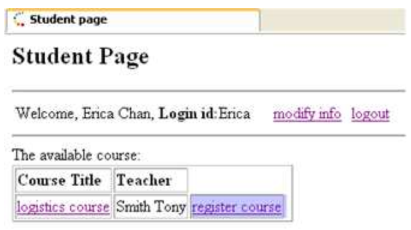
Course Title : logistics course

Course Description : logistics training

Submit

Figure 3.20 - Create course

2. Student can register the course in the student page.



Student Page

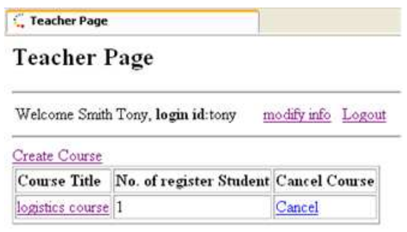
Welcome, Erica Chan, Login id:Erica modify info logout

The available course:

Course Title	Teacher	
logistics course	Smith Tony	register course

Figure 3.21 - Student Page

3. The teacher can view how many students have registered his course and he can cancel the course.



Teacher Page

Welcome Smith Tony, login id:tony modify info Logout

Create Course

Course Title	No. of register Student	Cancel Course
logistics course	1	Cancel

4. Click Cancel of a Course then it will pass the course id to the DeleteCourse Servlet. The DeleteCourse Servlet contains dropCourse method which uses the Course ID to retrieve to Course Object.

```
private void dropCourse(HttpServletRequest request)throws PersistentException{
        String lID = request.getParameter("courseID");
        PersistentTransaction t = SchoolSystemPersistentManager.instance().getSession().beginTransaction();
        try{
                Course lCourse = CourseFactory.loadCourseByORMID(Integer.parseInt(lID));
                ...
```

Source File : School System\WEB-INF\src\util\DeleteCourse.java

5. Call deleteAndDissociate() method to delete the course object and remove the relationships of student and teacher with the course.

```
                lCourse.deleteAndDissociate();
                t.commit();
        }catch(Exception e){
                e.printStackTrace();
                t.rollback();
        }
}
```

Source File : School System\WEB-INF\src\util\DeleteCourse.java

Using Persistent Object with HttpSession

here are mainly two kinds of Object in Web Application (Persistent Object and HttpSession Object). Persistent Object is used to store and retrieve data to and from the database (long term persistent media). HttpSession Object is used to store the information when the user is using the web application (usually store in RAM for short term persistent). If you need to use the Persistent Object as HttpSession Object (the lUser in the example code below), please make sure you have used the generated web.xml or configured your web application according to our generated web.xml. For more details about how to use the web.xml, please refer to previous section.

In School System, you have put the User persistent object in HttpSession to determine the user is login or not.

Set User Object to Session:

```
request.getSession().setAttribute("user", lUser);
```

Get User Object From Session:

```
Object lUserObj = request.getSession().getAttribute("user");
```

4 Developing Standalone Java Application

Chapter 4 - Developing Standalone Java Application

With DB Visual ARCHITECT (DB-VA) you can develop quality Standalone Java Application much faster, better and cheaper. DB-VA generates all Java code for accessing database. You do not need to write SQL to insert, query, update or delete the record. All code you need to program is plain Java code (e.g. OrderDAO.save(myOrder);). In this chapter we will use a simple "School System" application to show you how to generate Java code, create standalone Java application, and create/query/update/delete objects. Again you do not need to write a single SQL statement for all the above operations.

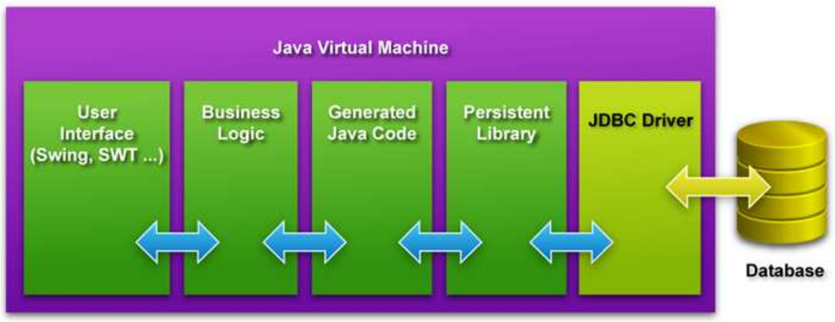


Figure 4.1 - Develop Standalone Java Application with DB Visual ARCHITECT

The architecture of Standalone Java Application with DB-VA Persistent Layer

In this chapter:

- Introduction
- Generating Java Source and Database
- Using PersistentManager and Transaction
- Creating Object and Saving to Database
- Querying Object from Database
- Updating Object and Saving to Database
- Deleting Object in Database

Introduction

You will develop a School System.

The School System provides the following functions:

- Create course by teacher
- Enroll course for student
- Cancel course by teacher
- Register for user
- Modify the personal information
- View the Course information (number of students enrolled and teacher information of the course)

Required Software:

- DB Visual ARCHITECT 3.0 Java or Professional Edition (http://www.visual-paradigm.com/download/)
- Sun J2SE 1.4 JDK or above (http://java.sun.com/)
- MySQL Server 4.1 or above (http://dev.mysql.com/downloads/)

Please open the SchoolSystem.vpp project file in the "Chapter 4 Standalone Java School System.zip" file. The project file contains the following diagrams. For the details about how to draw class diagram and entity relationship diagram, please see the Designer's Guide.

Class Diagram of School System:

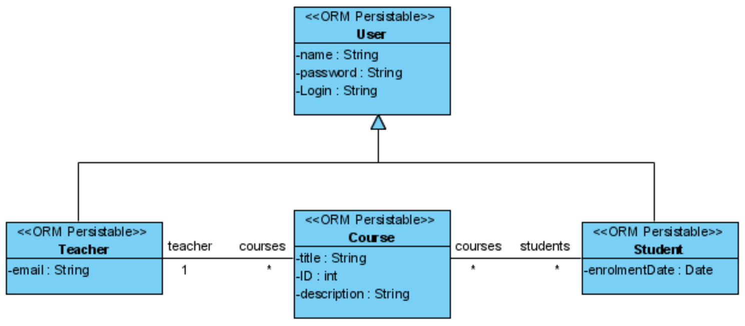


Figure 4.2 - Class Diagram of School System

Entity Relationship Diagram of School System:

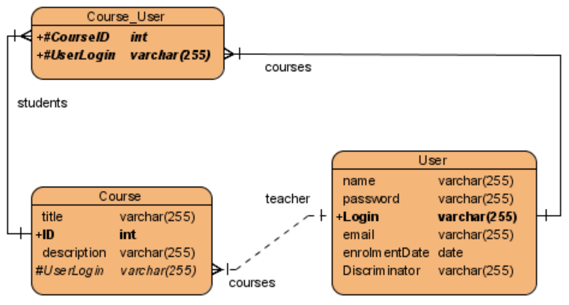


Figure 4.3 - ERD of School System

Generating Java Source and Database

In order to develop Java application, you need to generate code by DB-VA and configure the source code and library. The following example is a standalone Java application, so please pay attention to the following settings when configure to generate Java code.

1. From menu bar, select **Tools** > **Object-Relational Mapping (ORM)** > **Generate Code...** to open the **Database Code Generation** dialog box.

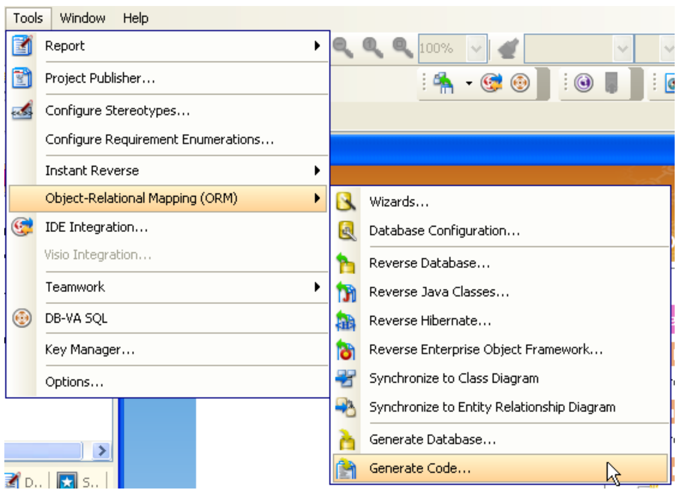
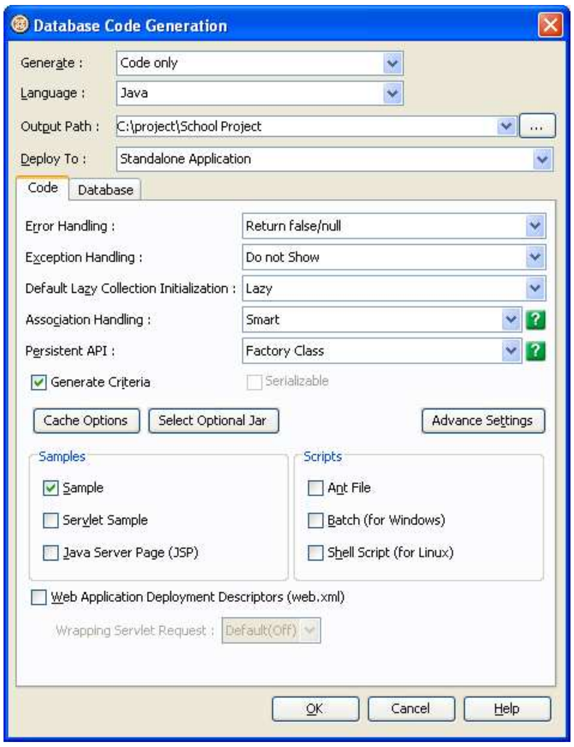


Figure 4.4 - To generate the code

2. Fill in code generation information.

   For **Language**, select **Java**.
   For **Association Handling**, select **Smart**.
   For **Persistent API**, select **Factory Class**.



Generate : Code only
Language : Java
Output Path : C:\project\School Project
Deploy To : Standalone Application
Error Handling : Return false/null
Exception Handling : Do not Show
Default Lazy Collection Initialization : Lazy
Association Handling : Smart
Persistent API : Factory Class
[✓] Generate Criteria
[ ] Serializable
[✓] Sample
[ ] Servlet Sample
[ ] Java Server Page (JSP)
[ ] Ant File
[ ] Batch (for Windows)
[ ] Shell Script (for Linux)
[ ] Web Application Deployment Descriptors (web.xml)
Wrapping Servlet Request : Default(Off)

Figure 4.5 - Database Code Generation dialog

For the other configuration details, please refer to Chapter 1 "Generate Java Code, Database Schema (DDL) and persistent Library" and Chapter 2 "Configure Source and Library in Eclipse" .

Using PersistentManager and Transaction

There is a PersistentManager (Class) in DB-VA generated code. The PersistentManager is used to manage the database connection, state of the persistent objects and transaction when the application is running.

A database transaction is a unit for the application. The database transaction is used to keep integrity of your data in database. DB-VA Persistent Library provides a transaction API for you to create, commit and rollback the transaction for your application.

Start Transaction

When you need to start a transaction, you just need to get the session from the PersistentManager and call the beginTransaction function (PersistentManager.instance().getSession().beginTransaction();).

```
PersistentTransaction t = PersistentManager.instance().getSession().beginTransaction();
```

Commit Transaction

After you inserted/updated or deleted the data, you can call the commit function of the transaction object (t.commit()) to commit your changes to Database.

```
t.commit();
```

Rollback Transaction

In case there are any exception, you can call the rollback function to cancel all the operation (t.rollback());

```
t.rollback();
```

The following is an example of using the transaction API.

Sample:

```
private Course fireOK() throws PersistentException {
        PersistentTransaction t =
        SchoolSystemPersistentManager.instance().getSession().beginTransaction();
        try {
                Course lCourse = CourseFactory.createCourse();
                lCourse.setTitle(getTitleTextField().getText());
                lCourse.setDescription(getDescriptionTextField().getText());
                lCourse.setTeacher(_teacher);
                _teacher.save();
                t.commit();
                return lCourse;
        } catch (Exception e) {
                e.printStackTrace();
                t.rollback();
        }
        return null;
}
```

Source File : School Project\system\dialog\AddCourseDialog.java

Creating Object and Saving to Database

In this chapter we will use a simple Swing application called "School System" to illustrate the application of the persistent API. If you are interested in this application, please browse the source code inside the "Chapter 4 Standalone Java School System.zip" zip file.

The school system provides the register function for teacher and student. They need to enter login id, password etc... information for system. The registration process is same for teacher and student, so we will only demonstrate how to create the student and save to database.

1. From menu bar, select **File** > **Register Student** to open the Add Student dialog box.

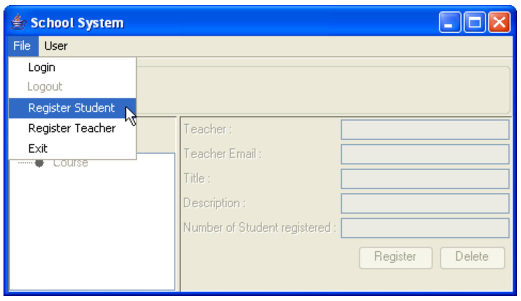
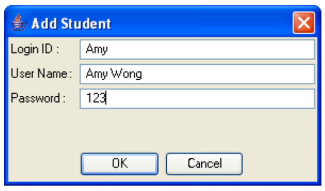


Figure 4.6 - school system

2. Enter the Student information; click **OK** to create the new Student record in School System.



Login ID : Amy
User Name : Amy Wong
Password : 123

Figure 4.7 - Add Student dialog

3. After click **OK**, the system create the new Student Persistent object

```
private void fireOK() throws PersistentException {
        ...
        PersistentTransaction t =
        SchoolSystemPersistentManager.instance().getSession().beginTransaction();
        _user = StudentFactory.createStudent();
```

Source File : School Project\system\dialog\AddCourseDialog.java

4. Set the student information from the text fields value to the Student Object

```
        ((Student) _user).setEnrolmentDate(new Date());
        _user.setLoginID(_loginIDField.getText());
        _user.setName(_userNameField.getText());
        _user.setPassword(_passwordField.getText());
```

Source File : School Project\system\dialog\AddCourseDialog.java

5. Call `save()` method of Student Persistent Object and `commit()` method of PersistentTransaction., then the new Student object will be recorded in database. If any error occurred during the transaction, you can call the `rollback()` method to cancel all proposed changes.

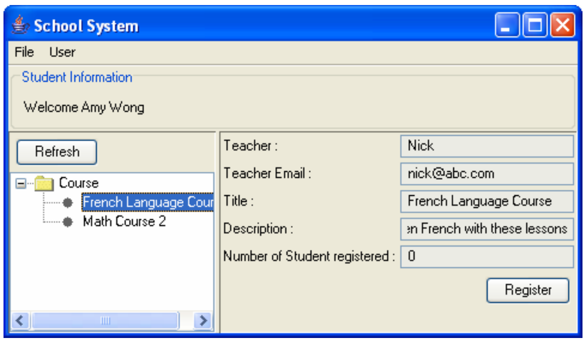
```
        _user.save();
        t.commit();
}catch (Exception e) {
        e.printStackTrace();
        t.rollback();
}
```

Source File : School Project\system\dialog\AddCourseDialog.java

Querying Object from Database

After the user login to the School System, the system queries different Course objects from the database according to user role. If the user is a student, the system shows all the available courses. The student can select and register the course. If the user is teacher, the system shows courses that are created by that teacher. The teacher can update or delete the course information in system.

Student Login:



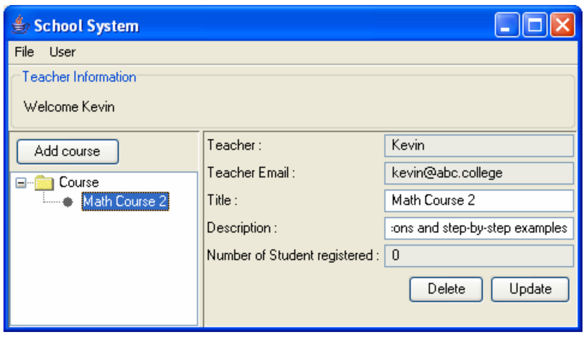
Welcome Amy Wong

Course
- French Language Cour
- Math Course 2

Teacher : Nick
Teacher Email : nick@abc.com
Title : French Language Course
Description : n French with these lessons
Number of Student registered : 0

Figure 4.8 - Student login to the system

Teacher Login:



Welcome Kevin

Teacher : Kevin
Teacher Email : kevin@abc.college
Title : Math Course 2
Description : :ons and step-by-step examples
Number of Student registered : 0

Figure 4.9 - Teacher login to the system

1. Query the course objects when user login. When Student login, the system will call **listCourseByQuery()** method in **CouseFactory** to get all available courses. When Teacher login, the system will call **courses** collection variable in Teacher object.

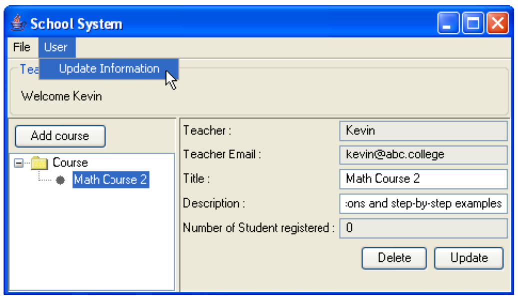
```
private void updateCourseTree() throws PersistentException{
        try {
                Course[] courses = null;
                if (_currentUser instanceof Student) {
                        courses = CourseFactory.listCourseByQuery(null, null);
                } else if (_currentUser instanceof Teacher) {
                        courses = ((Teacher) _currentUser).courses.toArray();
                }
                ...
        }
```

Source File : School Project\system\SchoolSystemFrame.java

Updating Object and Save to Database

You can modify the user information and update the record in database. After the user login, the User object is stored in the application, so you can set new information in the user object and update the database record.

1. From menu bar, select **User** > **Update Information** to open the Update User Information dialog box.

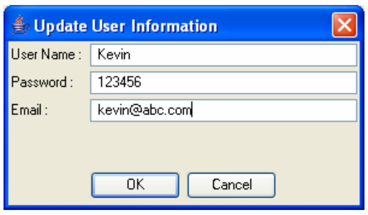


Welcome Kevin

Teacher : Kevin
Teacher Email : kevin@abc.college
Title : Math Course 2
Description : :ons and step-by-step examples
Number of Student registered : 0

Figure 4.10 - To Update Information

2. Enter new user information and click **OK** to update the User record.



User Name : Kevin
Password : 123456
Email : kevin@abc.com

Figure 4.11 - Update User Information

3. Update the information for the User object includes set password, name and email address. The process is simple with the create User object process.

```
private void fireOK() throws PersistentException {
        ...
        PersistentTransaction t =
        SchoolSystemPersistentManager.instance().getSession().beginTransaction();
        try {
                if (_user != null) {
                        _user.setName(_userNameField.getText());
                        _user.setPassword(_passwordField.getText());
                        if (_user instanceof Teacher) {
                                ((Teacher) _user).setEmail(_emailField.getText());
                        }
                        _user.save();
                }
                ...
                t.commit();
        } catch (Exception e) {
                e.printStackTrace();
                t.rollback();
        } finally {
                this.setVisible(false);
        }
}
```

Deleting Object in Database

Teacher can create course for students to registration and they can cancel the course in the system. They only need to click Delete then the course information will be deleted in the database and all its relationships with register students will be removed.

1. Teacher can create the course by clicking Add Course. Fill in Course name and Description to create Course.

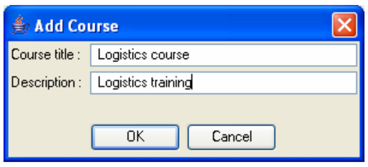

Course title : Logistics course
Description : Logistics training

Figure 4.12 - Add Course dialog

2. Student can register the course by clicking Register.

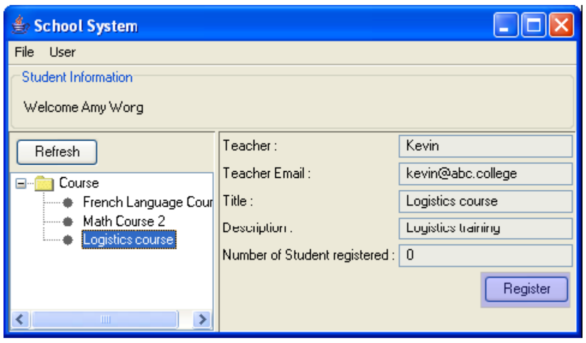
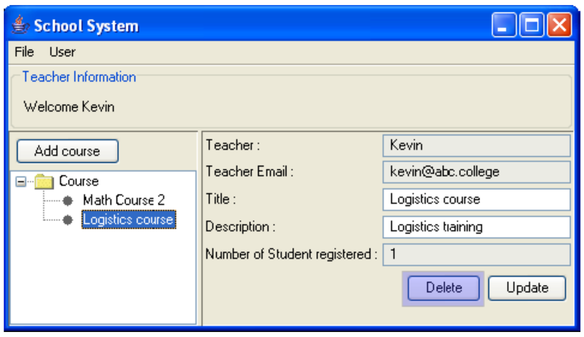

Welcome Amy Worg
Teacher : Kevin
Teacher Email : kevin@abc.college
Title : Logistics course
Description : Logistics training
Number of Student registered : 0

Figure 4.13 - Register a course

3. After the student is registered, the teacher can view how many students have registered his course and he can delete the course in system.



Welcome Kevin

Teacher : Kevin
Teacher Email : kevin@abc.college
Title : Logistics course
Description : Logistics training
Number of Student registered : 1

4. Click Delete then it will pass the Course object to the fireDeleteCourse() method .

```
private void fireDeleteCourse() throws PersistentException {

        PersistentTransaction t = SchoolSystemPersistentManager.instance()

        .getSession().beginTransaction();
```

Source File : School System\src\system\courseInformationPane.java
Call deleteAndDissociate() method to delete the course object and remove the relationship of student and teacher.

```
try {

        _currentCourse.deleteAndDissociate();

        _courseImpl.fireDeleteCourse();

        t.commit();

} catch (Exception e) {

        e.printStackTrace();

        t.rollback();

}
```

Source File : School System\src\system\courseInformationPane.java

5 Querying Database

Chapter 5 - Querying Database

DB Visual ARCHITECT (DB-VA) provides two features for querying database. You can use the ORM Qualifier and Criteria to define you requirements to retrieve the data from database. They provide a much simpler way to retrieve records from the database than SQL query.

In this chapter:

- Introduction
- Creating Test Data
- Using ORM Qualifier
- Defining ORM Qualifier
- Retrieving from ORM Qualifier
- Using Criteria
- Configuring Criteria Class Generation
- Description of Criteria Class
- Comparing the Criteria Class and SQL Query

Introductions

When you want to provide the search function and make an easy way for retrieving record from the database, then you can use **ORMQualifier** or **Generate Criteria** in DB-VA. In this chapter, we use an example to teach you how to use the criteria class to search the persistent objects from a user-defined condition. First of all, you need to create the following Class Diagram and synchronize to Entity Relationship Diagram.

Class Diagram:

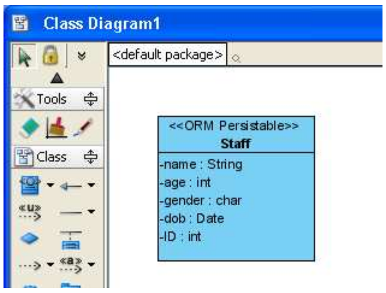


Figure 5.1 - Class Diagram

Entity Relationship Diagram:

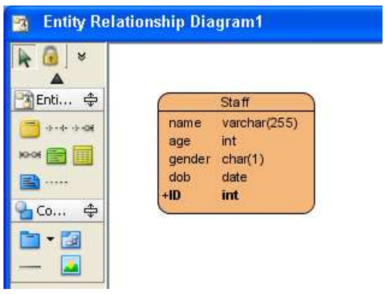


Figure 5.2 - Entity Relationship Diagram

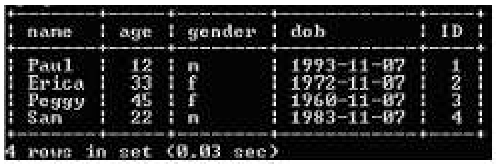
Creating Test Data

Select generate Sample when generate Persistent Java code. Then you can create the staff to the database and test the criteria class to help you to get the record from database.

1. Open the CreateUntitledData.java and modify the code of the following:

```
public class CreateUntitledData {
	public void createTestData() throws PersistentException {
		PersistentTransaction t =
		com.UntitledPersistentManager.instance().getSession().beginTransaction();
		try {
			com.Staff lcomStaff = com.StaffFactory.createStaff();
			// Initialize the properties of the persistent object
			lcomStaff.setName("Paul");
			lcomStaff.setAge(12);
			java.util.Calendar c = java.util.Calendar.getInstance();
			c.set(1993, 11, 7);
			lcomStaff.setDob(c.getTime());
			lcomStaff.setGender('m');
			lcomStaff.save();

			lcomStaff = com.StaffFactory.createStaff();
			lcomStaff.setName("Erica");
			lcomStaff.setAge(33);
			c.set(1972, 11, 7);
			lcomStaff.setDob(c.getTime());
			lcomStaff.setGender('f');
			lcomStaff.save();

			lcomStaff = com.StaffFactory.createStaff();
			lcomStaff.setName("Peggy");
			lcomStaff.setAge(45);
			c.set(1960, 11, 7);
			lcomStaff.setDob(c.getTime());
			lcomStaff.setGender('f');
			lcomStaff.save();

			lcomStaff = com.StaffFactory.createStaff();
			lcomStaff.setName("Sam");
			lcomStaff.setAge(22);
			c.set(1983, 11, 7);
			lcomStaff.setDob(c.getTime());
			lcomStaff.setGender('m');
			lcomStaff.save();
			t.commit();
		}
		catch (Exception e) {
			t.rollback();
		}
	}
	...
}
```

2. Execute the CreateUntitledData class. It creates the specified records in database.



```
+--------+-------+---------+-------------+----+
| name   | age   | gender  | dob         | ID |
+--------+-------+---------+-------------+----+
| Paul   |    12 | m       | 1993-11-07  |  1 |
| Erica  |    33 | f       | 1972-11-07  |  2 |
| Peggy  |    45 | f       | 1960-11-07  |  3 |
| Sam    |    22 | m       | 1983-11-07  |  4 |
+--------+-------+---------+-------------+----+
4 rows in set (0.03 sec)
```

Figure 5.3 - The executed result

Using ORM Qualifier

ORM Qualifier is an additional feature of DB-VA allowing you to specify the extra data retrieval rules apart from the system pre-defined rules. The ORM Qualifier can be defined to generate persistence code.

Defining ORM Qualifier

1. Right-click on the **Staff** class, select **Open Specification**.

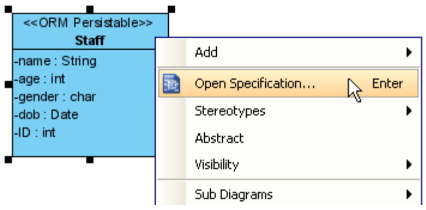


Figure 5.4 - To open class specification

2. Click the **ORM Qualifiers** Tab, then click **Add**.

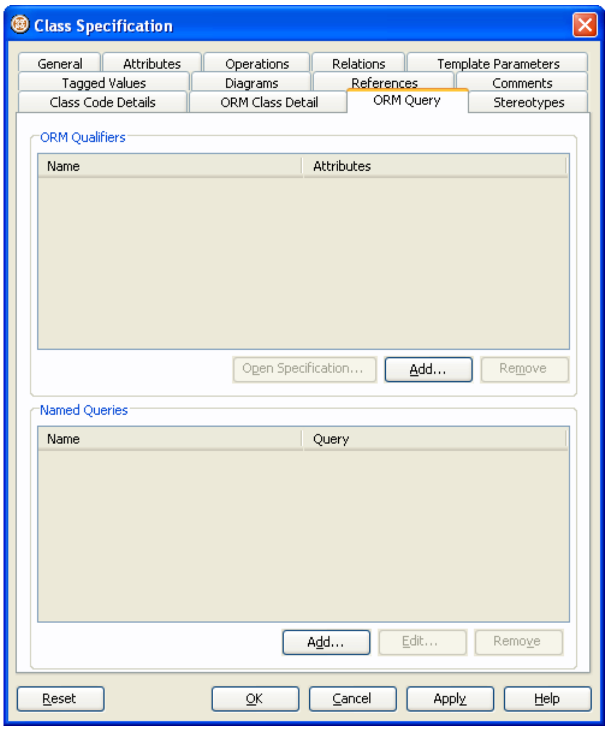
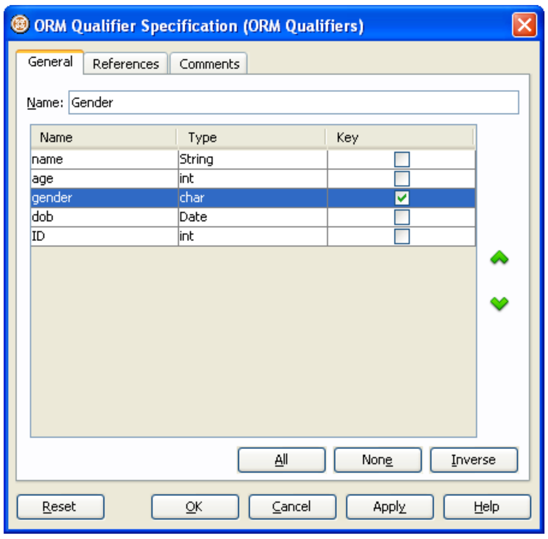


Figure 5.5 - Class Specification dialog

3. **ORM Qualifier Specification** dialog box is displayed with a list of attributes of the Staff class. Enter the name as **Gender** and select attribute **gender**.



Name: Gender

Name	Type	Key
name	String	☐
age	int	☐
gender	char	✓
dob	Date	☐
ID	int	☐

Figure 5.6 - ORM Qualifier Specification (ORM Qualifiers) dialog

4. Generate Persistent Java code, the ORM Qualifier methods will be generated in Persistent class according to the selected Persistent API. For example, if you selected Factory class as Persistent API, then the following methods will be generated in the **StaffFactory** class.

Return Type	**Method Name**	**Sample**	**Description**
Class	loadByORMQualifier(DataType attribute)	`loadByGender(char gender)`	Retrieve the first record that matches the specified value with the attribute defined in the ORM Qualifier.
Class	loadByORMQualifier (PersistentSession session, DataType attribute)	`loadByGender(PersistentSession session, char gender)`	Retrieve the first record that matches the specified value with the attribute defined in the ORM Qualifier and specified session.
Class[]	listByORMQualifier (DataType attribute)	`listByGender(char gender)`	Retrieve the records that match the specified value with the attribute defined in the ORM Qualifier.
Class[]	listByORMQualifier (PersistentSession session, DataType attribute)	`listByGender(PersistentSession session, char gender)`	Retrieve the records that match the specified value with the attribute defined in the ORM Qualifier and specified session.

Table 5.1

Retrieving from ORM Qualifier

Having created the "**Gender**" qualifier, you can use the "Gender" qualifier to load or list the Staff data from database. The following examples show how to load or list ORM qualifier with the generated sample code.

By Load method:

```
System.out.println(com.PersonFactory.loadByGender('m'));
```

Result: After executing the code the first occurrence of 'm' gender column in the Staff table will be loaded to the object identified as Staff.

```
Retrieving staff by gender...
Person[ Name=Paul Age=12 Gender=m Dob=1993-11-07 ID=1 ]
```

By List method:

```
System.out.println("Retrieving Staffs by gender...");
com.Staff[] staffs = com.StaffFactory.listByGender('f');
for (int i = 0; i < staffs.length; i++){
	System.out.println(staffs[i]);
}
```

Result: After executing the code, all rows which contain 'f' in the gender column in the Staff table will be retrieved and stored in an array of Staff object.

```
Retrieving Staffs by gender...
Staff[ Name=Erica Age=33 Gender=f Dob=1972-11-07 ID=2 ]
Staff[ Name=Peggy Age=45 Gender=f Dob=1960-11-07 ID=3 ]
```

Using Criteria

When generating the persistence class for each ORM-Persistable class defined in the object model, the corresponding criteria class can also be generated.

Configuring Criteria Class Generation

After you have created the Class Diagram and ER Diagram, setup the database and generate code configuration. If you want to use the Criteria Class, you must select the Generate Criteria option in the Database Code Generation Dialog.

1. From the menu, select **Tools > Object-Relational Mapping (ORM) > Generate Code...** to open **Database Code Generation** dialog box.

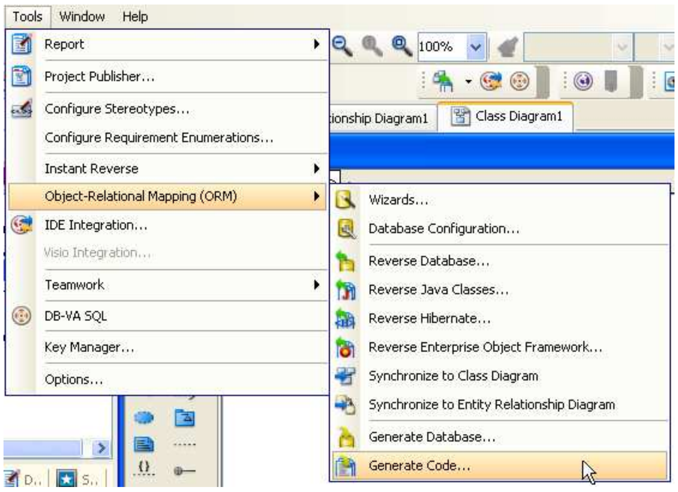
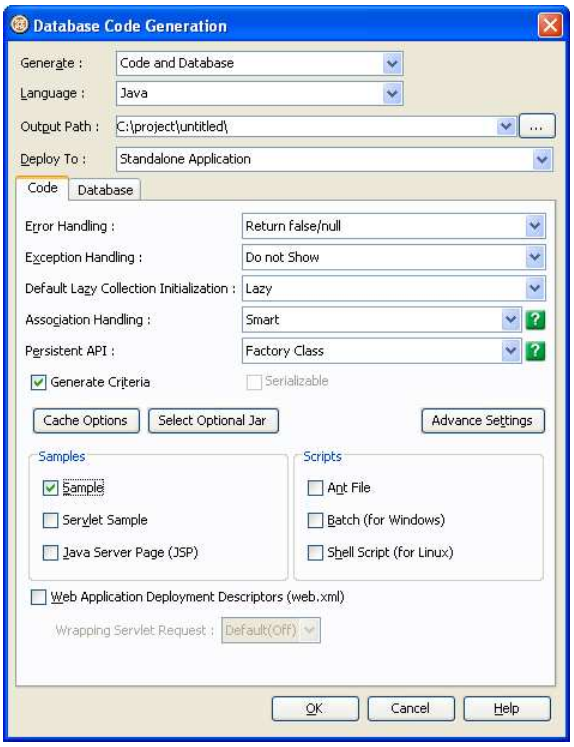


Figure 5.7 - To generate code

2. Select **Generate Criteria** option and the other settings can be set to follow the picture below.

   Check **Sample** so you can use the sample to test the criteria class.



Database Code Generation

Generate : Code and Database
Language : Java
Output Path : C:\project\untitled\
Deploy To : Standalone Application

Code | Database

Error Handling : Return false/null
Exception Handling : Do not Show
Default Lazy Collection Initialization : Lazy
Association Handling : Smart
Persistent API : Factory Class
[✓] Generate Criteria [ ] Serializable

Cache Options | Select Optional Jar | Advance Settings

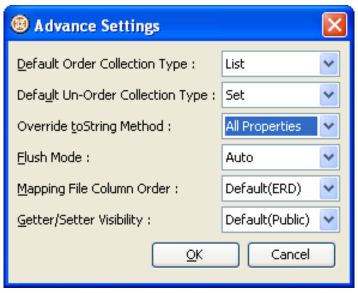
Samples: [✓] Sample, [ ] Servlet Sample, [ ] Java Server Page (JSP)
Scripts: [ ] Ant File, [ ] Batch (for Windows), [ ] Shell Script (for Linux)

[ ] Web Application Deployment Descriptors (web.xml)
Wrapping Servlet Request : Default(Off)

OK | Cancel | Help

Figure 5.8 - Database Code Generation dialog

Open the **Advance Settings** dialog box and set the **Override toString Method** to **All Properties**. It can help you to easily print out the persistent object data. Click **OK** to generate the Java Code with Criteria Class.



Advance Settings

Default Order Collection Type : List
Default Un-Order Collection Type : Set
Override toString Method : All Properties
Flush Mode : Auto
Mapping File Column Order : Default(ERD)
Getter/Setter Visibility : Default(Public)

OK | Cancel

Figure 5.9 - Advance Settings dialog

For more details of configuring the database and generating code, you can refer to Chapter 1 - Generate Java, Database and Persistent Library.

Description of Criteria Class

The following is the criteria class of the previous generated Staff class called "**StaffCriteria**".

```
public class StaffCriteria extends AbstractORMCriteria {
	public final StringExpression name;
	public final IntegerExpression age;
	public final CharacterExpression gender;
	public final DateExpression dob;
	public final IntegerExpression ID;

	public StaffCriteria(PersistentSession session) {
		super(session.createCriteria(Staff.class));
		name = new StringExpression("name", this);
		age = new IntegerExpression("age", this);
		gender = new CharacterExpression("gender", this);
		dob = new DateExpression("dob", this);
		ID = new IntegerExpression("ID", this);
	}

	public StaffCriteria() throws PersistentException {
		this(com.UntitledPersistentManager.instance().getSession());
	}

	public Staff uniqueStaff() {
		return (Staff) super.uniqueResult();
	}

	public Staff[] listStaff() {
		return (Staff[]) super.list().toArray(new Staff[super.list().size()]);
	}
}
```

The StaffCriteria class is generated with attribute, which are defined in the object model, with type of one of Expression with respect to the type of attribute defined in the object model and two operations for specifying the type of record retrieval.

To apply the restriction to the property, implement the following code template:

```
criteria.property.expression(parameter);
```

where criteria is the instance of the criteria class; property is the property of the criteria; expression is the expression to be applied on the property; parameter is the parameter(s) of the expression.

The table below shows the expression that can be used for specifying the condition for query.

Expression	Description
eq(value)	The value of the property is equal to the specified value.
ne(value)	The value of the property is not equal to the specified value.
gt(value)	The value of the property is greater than to the specified value.
ge(value)	The value of the property is greater than or equal to the specified value.
lt(value)	The value of the property is less than the specified value.
le(value)	The value of the property is less than or equal to the specified value.
isEmpty()	The value of the property is empty.
isNotEmpty()	The value of the property is not empty.
isNull()	The value of the property is NULL.
isNotNull()	The value of the property is not NULL.
in(values)	The value of the property contains the specified values in the array.
between(value1, value2)	The value of the property is between the two specified values, value1 and value2.
like(value)	The value of the property matches the string pattern of value; use % in value for wildcard.
ilike(value)	The value of the property matches the string pattern of value, ignoring case differences.

Table 5.2

For example:

```
staffCriteria.age.ge(13);
```

There are two types of ordering to sort the retrieved records, that is, ascending and descending order.

To sort the retrieved records with respect to the property, implement the following code template:

```
criteria.property.order(ascending_order);
```

where the value of `ascending_order` is either true or false. True refers to sort the property in ascending order while false refers to sort the property in descending order.

For example:

```
staffCriteria.age.order(true);
```

To set the range of the number of records to be retrieved by using one of the two methods:

- setFirstResult(int i) - Retrieve the i-th record from the results as the first result.
- setMaxResult(int i) - Set the maximum number of retrieved records by specified value, i.

For example:

```
staffCriteria.setMaxResults(100);
```

The StaffCriteria class contains two methods to load the retrieved record(s) to an object or array.

- uniqueClass() - Retrieve a single record matching the specified condition(s) for the criteria; Exception will be thrown if the number of retrieved record is not equal to 1.
- listClass() - Retrieve the records matched with the specified condition(s) for the criteria.

For example:

```
com.Staff[] lcomStaffs = staffCriteria.listStaff();
```

Comparing the Criteria Class and SQL Query

The SQL Query can help you to find the record from the database and Criteria Class also can provide the same function to get the persistent object from the database.

The SQL Query is very long and easy to have syntax mistake but with the Criteria Class it is easy to set the condition for each property of the persistent class. The Criteria Class can directly get the Persistent Objects but SQL Query only get the individual data in database.

- By Specifying one criteria

Retrieve a staff record whose name is "Paul":

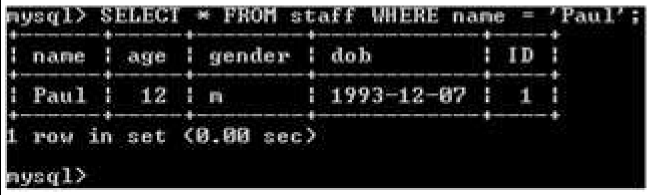
Criteria Class	SQL Query

`com.StaffCriteria staffCriteria = new com.StaffCriteria();`
`staffCriteria.name.eq("Paul");`
`com.Staff[] lcomStaffs = staffCriteria.listStaff();`
`int length = (lcomStaffs == null) ? 0 :`
`Math.min(lcomStaffs.length, 100);`
`for (int i = 0; i < length; i++) {`
`       System.out.println(lcomStaffs[i]);`
`}`
`System.out.println(length + " Staff record(s)`
`retrieved.");`
	`SELECT * FROM staff WHERE`
	`name = 'Paul';`

Table 5.3

The Result:

Criteria Class	SQL Query

`Staff[ Name=Paul Age=12 Gender=m`
`Dob=1993-12-07 ID=1 ]`
`1 Staff record(s) retrieved.`
	

```
mysql> SELECT * FROM staff WHERE name = 'Paul';
+------+-----+--------+------------+----+
| name | age | gender | dob        | ID |
+------+-----+--------+------------+----+
| Paul |  12 | m      | 1993-12-07 |  1 |
+------+-----+--------+------------+----+
1 row in set (0.00 sec)

mysql>
```

Table 5.4

Retrieving all staff records whose date of birth is between 1/1/1970 and 31/12/1985:

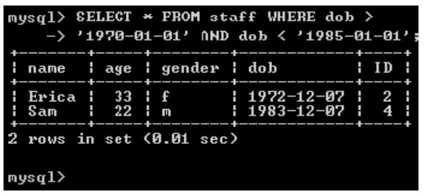
Criteria Class	SQL Query

`com.StaffCriteria staffCriteria = new com.StaffCriteria();`
`staffCriteria.dob.between(new GregorianCalendar(1970, 1, 1).getTime(), new GregorianCalendar(1985, 12, 31).getTime());`
`com.Staff[] lcomStaffs = staffCriteria.listStaff();`
`int length = (lcomStaffs == null) ? 0 : Math.min(lcomStaffs.length, 100);`
`for (int i = 0; i < length; i++) {`
`      System.out.println(lcomStaffs[i]);`
`}`
`System.out.println(length + " Staff record(s) retrieved.");`
	`SELECT * FROM staff WHERE dob > '1970-01-01' AND dob < '1985-01-01';`

Table 5.5

The Result:

Criteria Class	SQL Query

`Listing Staff by Criteria...`
`Staff[ Name=Erica Age=33 Gender=g Dob=1972-12-07 ID=2 ]`
`Staff[ Name=Sam Age=22 Gender=m Dob=1983-12-07 ID=4 ]`
`2 Staff record(s) retrieved.`
	

```
mysql> SELECT * FROM staff WHERE dob >
    -> '1970-01-01' AND dob < '1985-01-01';
+-------+-----+--------+------------+----+
| name  | age | gender | dob        | ID |
+-------+-----+--------+------------+----+
| Erica |  33 | f      | 1972-12-07 |  2 |
| Sam   |  22 | m      | 1983-12-07 |  4 |
+-------+-----+--------+------------+----+
2 rows in set (0.01 sec)

mysql>
```

Table 5.6

- By specifying more than one criteria

Retrieve all male staff records whose age is between 18 and 22:

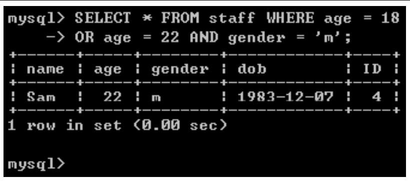
Criteria Class	SQL Query

`com.StaffCriteria staffCriteria = new com.StaffCriteria();`
`staffCriteria.age.in(new int[]{18, 22});`
`staffCriteria.gender.eq('m');`
`com.Staff[] lcomStaffs = staffCriteria.listStaff();`
`int length = (lcomStaffs == null)? 0 : Math.min(lcomStaffs.length, 100);`
`for (int i = 0; i < length; i++) {`
`      System.out.println(lcomStaffs[i]);`
`}`
`System.out.println(length + " Staff record(s) retrieved.");`
	`SELECT * FROM staff WHERE age = 18 OR age = 22 AND gender = 'm';`

Table 5.7

The Result:

Criteria Class	SQL Query

`Staff[ Name=Sam Age=22 Gender=m Dob=1983-12-07 ID=4 ]`
`1 Staff record(s) retrieved.`
	

```
mysql> SELECT * FROM staff WHERE age = 18
    -> OR age = 22 AND gender = 'm';
+------+-----+--------+------------+----+
| name | age | gender | dob        | ID |
+------+-----+--------+------------+----+
| Sam  |  22 | m      | 1983-12-07 |  4 |
+------+-----+--------+------------+----+
1 row in set (0.00 sec)

mysql>
```

Table 5.8

Retrieve all staff records whose name starts with "P" and age is lesser than 50, ordering by the name:

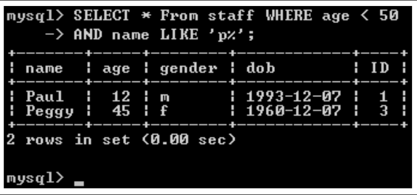
Criteria Class	SQL Query

`com.StaffCriteria staffCriteria = new`
`com.StaffCriteria();`
`staffCriteria.name.like("P%");`
`staffCriteria.age.lt(50);`
`staffCriteria.name.order(true);`
`com.Staff[] lcomStaffs = staffCriteria.listStaff();`
`int length = (lcomStaffs == null)? 0 :`
`Math.min(lcomStaffs.length, 100);`
`for (int i = 0; i < length; i++) {`
`       System.out.println(lcomStaffs[i]);`
`}`
`System.out.println(length + " Staff record(s)`
`retrieved.");`
	`SELECT * From staff WHERE`
	`age < 50 AND name LIKE 'p%';`

Table 5.9

The Result:

Criteria Class	SQL Query

`Staff[ Name=Paul Age=12 Gender=m Dob=1993-12-07 ID=1 ]`
`Staff[ Name=Peggy Age=45 Gender=g Dob=1960-12-07 ID=3 ]`
`2 Staff record(s) retrieved.`
	

```
mysql> SELECT * From staff WHERE age < 50
    -> AND name LIKE 'p%';
+--------+-----+--------+------------+----+
| name   | age | gender | dob        | ID |
+--------+-----+--------+------------+----+
| Paul   |  12 | m      | 1993-12-07 |  1 |
| Peggy  |  45 | f      | 1960-12-07 |  3 |
+--------+-----+--------+------------+----+
2 rows in set (0.00 sec)

mysql> _
```

Table 5.10

6 Generating Object-Oriented Java Source from Relational Database

Chapter 6 - Generating Object-Oriented Java Source from Relational Database

With DB Visual ARCHITECH (DB-VA), you can easily reverse the relational database schema to Object-Oriented Java Source. This feature can help user to develop a new application for existing data in relational database. You do not need to write SQL to insert, query, update or delete for relational database. In this chapter, you will focus on using the wizards in DB-VA to reverse the relational database schema to Object-Oriented Java Source.

In this chapter:

Introduction

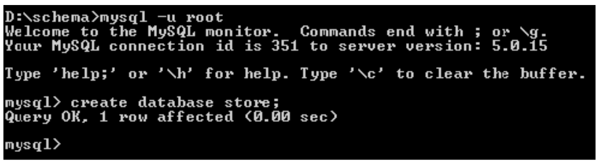
In this chapter, we will use a MySQL database. You need to import schema to the MySQL database. The schema is used for simulating an existing database schema. You will reverse that schema in DB-VA and generate Object-Oriented Java source. Before start reversing the schema, you need to have the MySQL 5.0 Community Edition installed.

The MySQL 5.0 Community Edition can be downloaded at http://dev.mysql.com/downloads/mysql/5.0.html.

Creating Simulate Data

1. Open the Command Prompt, log on MySql database and create database called "store".

```
mysql -u root

create database store;
exit;
```



```
D:\schema>mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 351 to server version: 5.0.15

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create database store;
Query OK, 1 row affected (0.00 sec)

mysql>
```

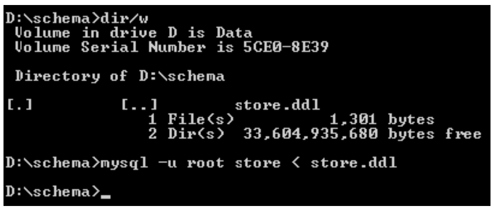
Figure 6.1 - The excute result of the command

2. Change to the directory that contains the sample schema. The following is the content of the sample schema in **store.ddl**.

```
Create table contacts (TOrderIndex int not null unique, contact varchar(255), TOrderID
int not null, primary key (TOrderIndex, TOrderID)) type=InnoDB;
create table customer (ID varchar(255) not null unique, name varchar(255), discount
double, primary key (ID)) type=InnoDB;
create table orderline (ID int not null auto_increment unique, quantity int not null,
OrderID int not null, primary key (ID)) type=InnoDB;
create table product (ID int not null auto_increment unique, name varchar(255),
OrderLineID int not null, primary key (ID)) type=InnoDB;
create table torder (ID int not null auto_increment unique, OrderDate date, CustomerID
varchar(255) not null, primary key (ID)) type=InnoDB;
alter table contacts add index FK_Contacts_7690 (TOrderID), add constraint
FK_Contacts_7690 foreign key (TOrderID) references torder (ID);
alter table orderline add index FK_OrderLine_9672 (OrderID), add constraint
FK_OrderLine_9672 foreign key (OrderID) references torder (ID);
alter table product add index FK_Product_5274 (OrderLineID), add constraint
FK_Product_5274 foreign key (OrderLineID) references orderline (ID);
alter table torder add index FK_TOrder_4869 (CustomerID), add constraint FK_TOrder_4869
foreign key (CustomerID) references customer (ID);
```

3. Type the following command to import the schema to the store database.

```
mysql - u root store < store.ddl
```

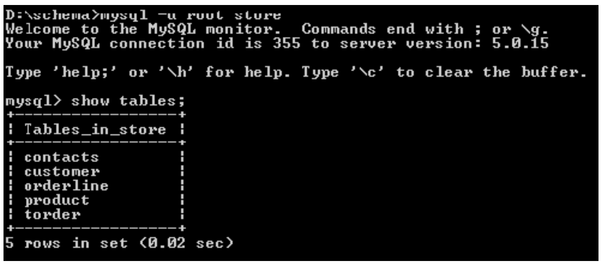


```
D:\schema>dir/w
 Volume in drive D is Data
 Volume Serial Number is 5CE0-8E39

 Directory of D:\schema

[.]          [..]         store.ddl
               1 File(s)          1,301 bytes
               2 Dir(s)  33,604,935,680 bytes free

D:\schema>mysql -u root store < store.ddl

D:\schema>_
```

Figure 6.2 - Import the DDL script

4. Log on MySql database and show tables in the store database.

```
mysql - u root store
show tables;
```



```
D:\schema>mysql -u root store
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 355 to server version: 5.0.15

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show tables;
+-----------------+
| Tables_in_store |
+-----------------+
| contacts        |
| customer        |
| orderline       |
| product         |
| torder          |
+-----------------+
5 rows in set (0.02 sec)
```

Figure 6.3 - Show the tables in the database

Start Generating Code from Database Wizard

The following steps show how you can use DB-VA Wizard to generate code from database.

1. Create a new Project in DB-VA called "**Store**" .

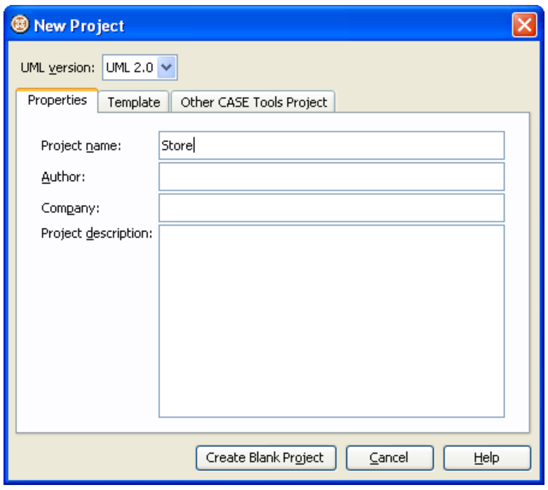


Figure 6.4 - New project dialog

2. On the menu, click **Tools** > **Object-Relational Mapping (ORM)** > **Wizard...** to open Wizard dialog box.

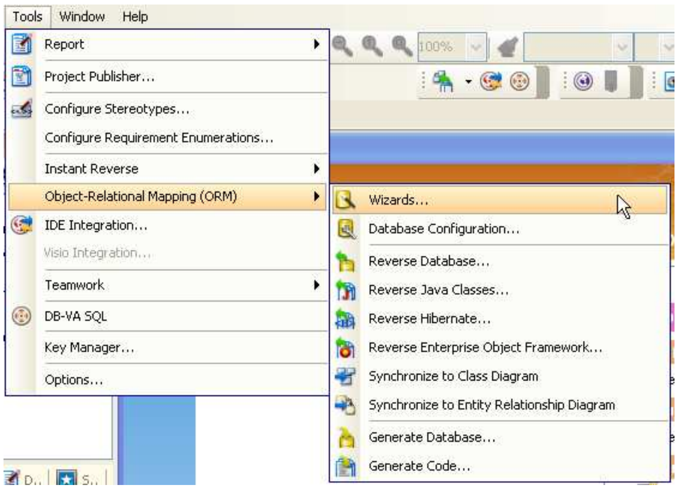
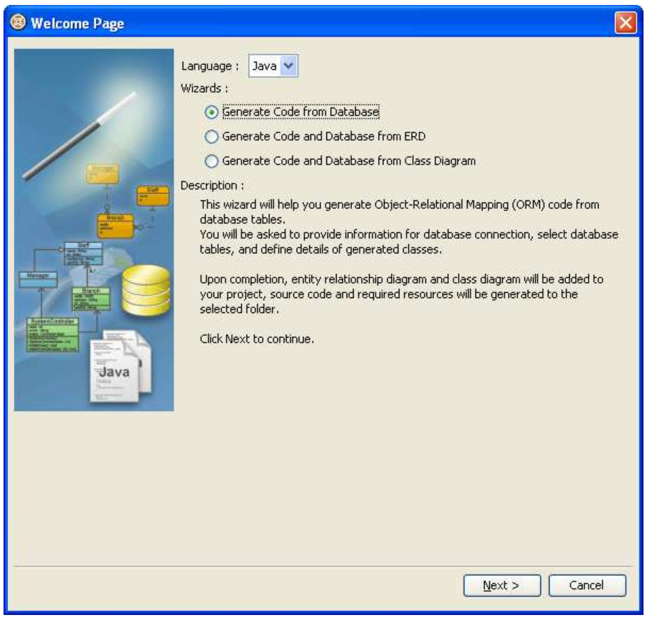


Figure 6.5 - To open the ORM Wizard

3. Select **Java** Language and **Generate Code from Database** on the Wizard Welcome page and then click **Next >.**



Welcome Page

Language : Java

Wizards :

- [✓ Generate Code from Database]
- Generate Code and Database from ERD
- Generate Code and Database from Class Diagram

Description :

This wizard will help you generate Object-Relational Mapping (ORM) code from database tables.
You will be asked to provide information for database connection, select database tables, and define details of generated classes.

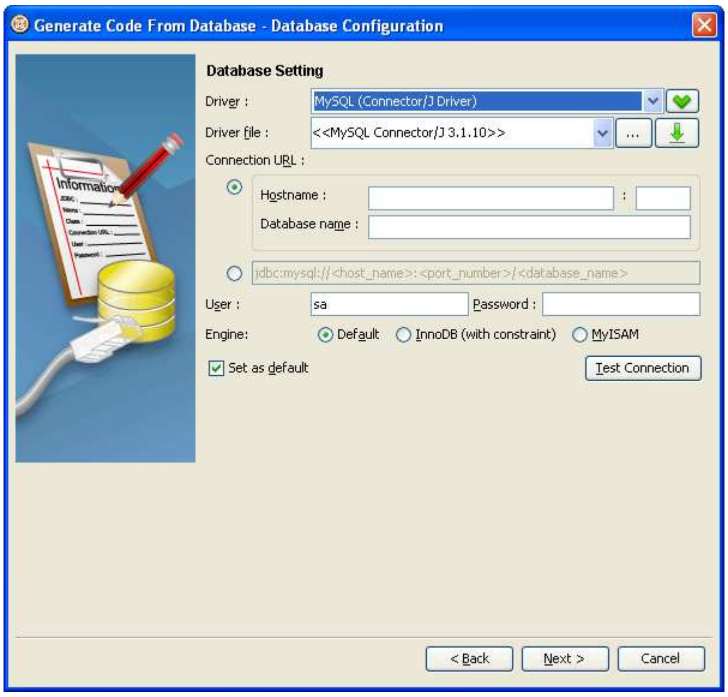
Upon completion, entity relationship diagram and class diagram will be added to your project, source code and required resources will be generated to the selected folder.

Click Next to continue.

Figure 6.6 - Welcome Page of the ORM Wizard

Configuring Database

In this part, you need to select the database and enter the database information. DB-VA will use the information you entered to get the database schema information to reverse.



Generate Code From Database - Database Configuration

Database Setting

Driver : MySQL (Connector/J Driver)

Driver file : <<MySQL Connector/J 3.1.10>>

Connection URL :

[✓] Hostname : :
Database name :

jdbc:mysql://<host_name>:<port_number>/<database_name>

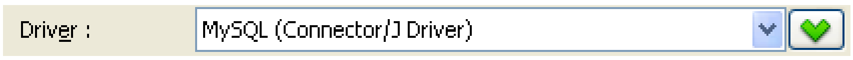
User : sa Password :

Engine: [✓ Default] InnoDB (with constraint) MyISAM

[✓] Set as default

Figure 6.7 - Database Configuration

1. Select **MySQL (Connector/J Driver)** for Driver option.



Driver : MySQL (Connector/J Driver)

Figure 6.8 - Driver options

2. Download or browse the suitable JDBC Driver file for your selected Driver option.

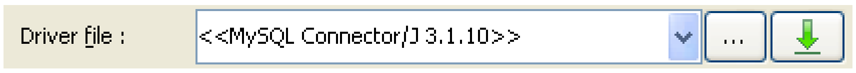

Driver file : <<MySQL Connector/J 3.1.10>>

Figure 6.9 - Driver file options

For **Driver File**, you can press the down arrow button to select **Download**, **Update** and **Default Driver**; DB-VA helps you to download the most up-to-date driver according to the **Driver** field information. You can also select **Browse...** to specify the driver file in your computer.

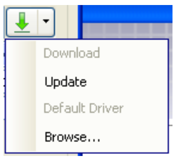


Figure 6.10 - Download driver options

After downloaded the driver file, **<<MySQL Connector/J 3.1.10>>** shown on the **Driver file** indicates that the JDBC driver file is downloaded with the specified version number by DB-VA.

3. Fill in the **Connection URL** information of your database.

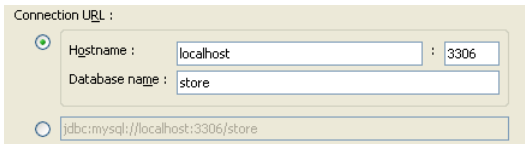

Connection URL :
Hostname : localhost : 3306
Database name : store
jdbc:mysql://localhost:3306/store

Figure 6.11 - The Connection URL

For the **Connection URL**, the Connection URL template for different database is shown; enter the information for connecting the database.

The default Connection URL template for MySQL is:

```
jdbc:mysql://<host_name>:<port_number>/<database_name>
```

The desired Connection URL of MySQL is:

```
jdbc:mysql://localhost:3306/store
```

For **User**, enter the valid username who has the access right to connect database

For **Password**, enter the password for this user.
For **Engine**, select the type of engine used to reverse the MySQL database. You need to select InnoDB otherwise the relationship between models will be lost.

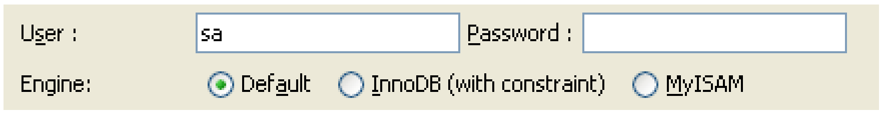

User : sa Password :
Engine: [✓ Default] InnoDB (with constraint) MyISAM

Figure 6.12 - The database engine, username and password

The **Engine** option in the **Database Setting** is only provided when configuring **MySQL** database for Java project.

4. Press **Test Connection** button after filling in the database information to test whether the database can be connected.

Figure 6.13 - Test Connection button

If the database can be connected, the **Connection Successful** dialog box will show; otherwise the **Connection Exception** dialog box will be prompted.

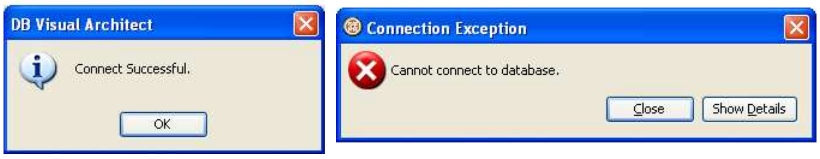


Figure 6.14 - Connection successful and failure message

5. Click **Next >** to Selecting Tables.

Selecting Table

DB-VA uses your configured database setting to connect to database. You can select the database tables which you want to generate persistent class to manipulate those tables. In this sample, you need to select all the database tables to reverse. You can deselect the table by using the buttons between the list of **Available Tables** and **Selected Tables**.

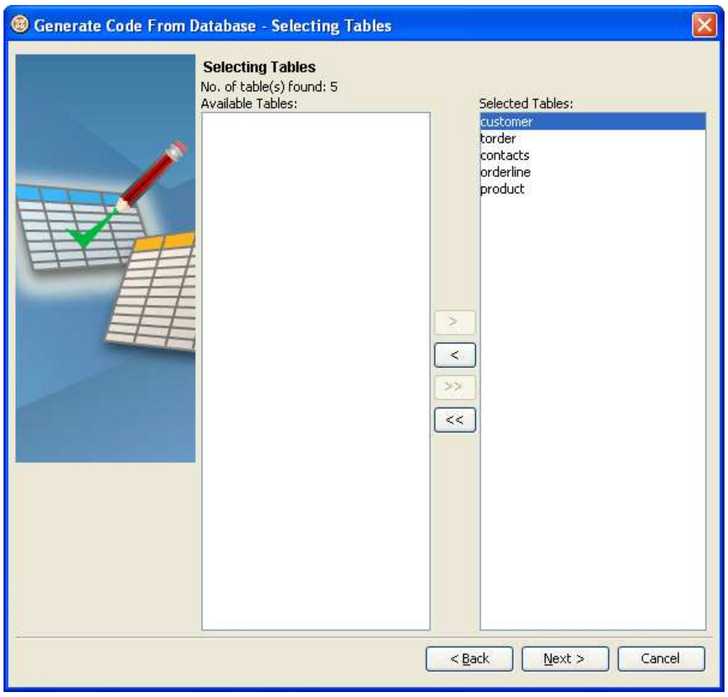


Figure 6.15 - Selecting Table

- Add Selected

  Add the selected table from **Available Tables** to **Selected Tables**.

- Remove Selected

  Remove the selected table from **Selected Tables** to **Available Tables**.

- >> Add All

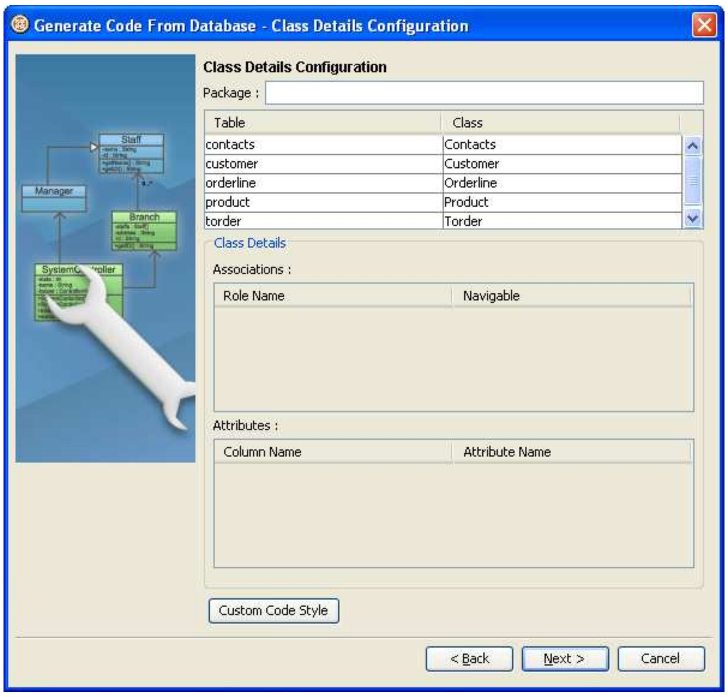
  Add all tables from **Available Tables** to **Selected Tables**.

- << Remove All

  Remove all tables from **Selected Tables** to **Available Tables**.

Configuring Generated Class Details

After selecting tables, you will be directed to a Class Details Configuration pane. In this pane, you can define the Class Details for generating code. DB-VA generates persistent classes based on the information define here. You can edit the class details by double-clicking the field. The following is a sample to modify the Customer class's class name, association role name and attribute, etc...



Generate Code From Database - Class Details Configuration

Class Details Configuration

Package :

Table	Class
contacts	Contacts
customer	Customer
orderline	Orderline
product	Product
torder	Torder

Class Details

Associations :

Role Name	Navigable

Attributes :

Column Name	Attribute Name

Figure 6.16 - Configuring Generated Class Details

1. Type the package name call "store". A package will be created to store the generated persistent code. If the package name was not defined, you will be prompted by a dialog box warning you the classes will be generated in default package.

Figure 6.17 - Enter the package name

2. Change Customer Class name to Buyer. You can edit the class name which will be used as the name of the generated persistent code for a corresponding table.

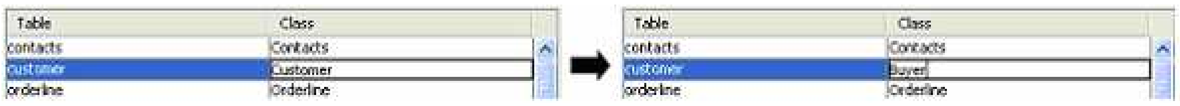
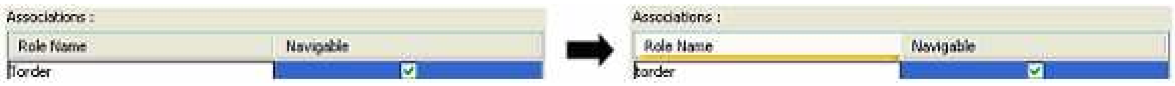


Table	Class
contacts	Contacts
customer	Customer
orderline	Orderline

Table	Class
contacts	Contacts
customer	Buyer
orderline	Orderline

Figure 6.18 - Mapping Classes

3. Change the first character of the Buyer class's Association Role Name from upper case to lower case.



Associations :

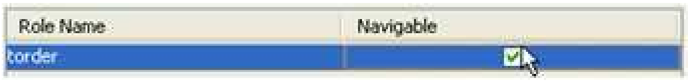
Role Name	Navigable
Torder	✓

Associations :

Role Name	Navigable
torder	✓

Figure 6.19 - Change the association name

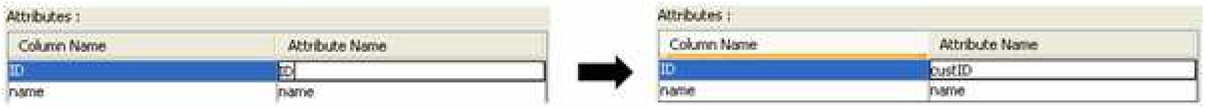
You can deselect navigable for an association such that the reference for the target role will not be created.



Role Name	Navigable
torder	✓

Figure 6.20 - Select the Navigable

4. Modify the Buyer class's Attribute from ID to custID.



Attributes :

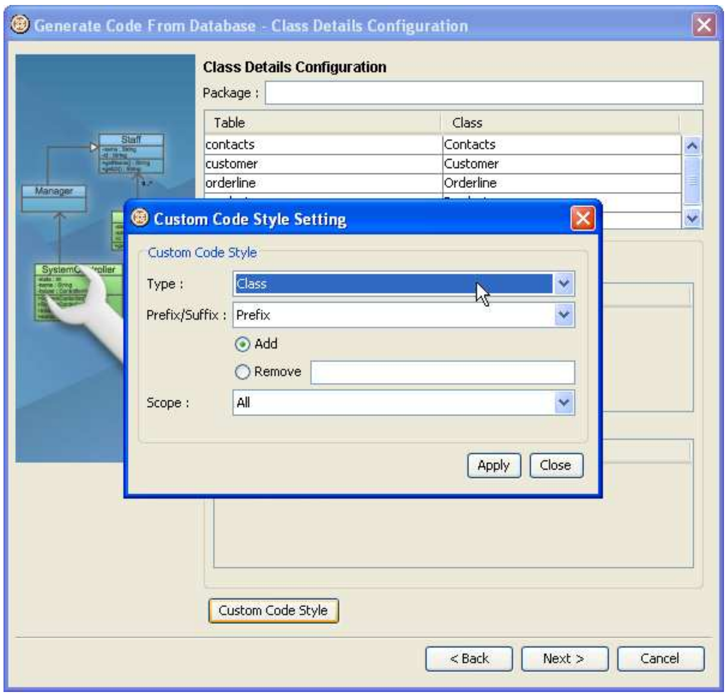
Column Name	Attribute Name
ID	ID
name	name

Attributes :

Column Name	Attribute Name
ID	custID
name	name

Figure 6.21 - Mapping attributes

5. Click **Custom Code Style** button to open **Custom Code Style Setting** dialog box. You can modify the prefix or suffix of the **Class**, **Attribute** and **Role Name**.



Generate Code From Database - Class Details Configuration

Class Details Configuration

Package :

Table	Class
contacts	Contacts
customer	Customer
orderline	Orderline

Custom Code Style Setting

Custom Code Style

Type : Class

Prefix/Suffix : Prefix

[✓ Add]

Remove

Scope : All

Figure 6.22 - Custom Code Style Setting

For **Type**, select the type - either Class, Attribute or Role Name (PK) that you want to apply code style.

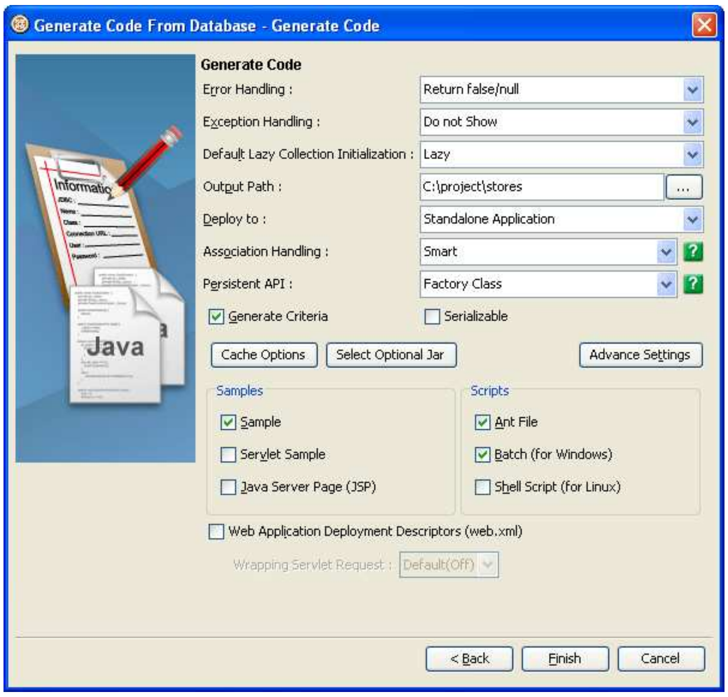
For **Prefix/Suffix**, select either Prefix or Suffix to be added or removed.
For **Add/Remove** option, select the option for the action of code style to be applied.
For **Textbox**, enter the word for either prefix or suffix.
For **Scope**, select the scope of the code style to be applied to, either All or Selected.
The table below shows the result of applying the code style.

Code Style	Before Applying	After Applying
Add Prefix (E.g. pre_)	Item	pre_Item
Remove Prefix (E.g. pre_)	pre_Item	Item
Add Suffix (E.g. _suf)	Item	Item_suf
Remove (E.g. _suf)	Item_suf	Item

Table 6.1

Specifying Code Generation Details

This is the final step to specify Java code generation details. You can select the location of the code generation, the type of generation code deploy to, etc... accord your requirement.


Generate Code From Database - Generate Code

Generate Code
Error Handling : Return false/null
Exception Handling : Do not Show
Default Lazy Collection Initialization : Lazy
Output Path : C:\project\stores
Deploy to : Standalone Application
Association Handling : Smart
Persistent API : Factory Class
[✓] Generate Criteria
[ ] Serializable

Samples
[✓] Sample
[ ] Servlet Sample
[ ] Java Server Page (JSP)

Scripts
[✓] Ant File
[✓] Batch (for Windows)
[ ] Shell Script (for Linux)

[ ] Web Application Deployment Descriptors (web.xml)
Wrapping Servlet Request : Default(Off)

Figure 6.23 - Generate code options

For **Error Handling**, select the way to handle errors. The possible errors include PersistentException, GenericJDBCException, and SQLException.

- **Return false/null** - It returns false/null in the method to terminate its execution.
- **Throw PersistentException** - It throws a PersistentException which will be handled by the caller. A try/catch block has to be implemented to handle the exception.
- **Throw RuntimeException - It throws a RuntimeException which will be handled by the caller. A try/catch block has not been implemented to handle the exception. The exception will be caught in runtime.**

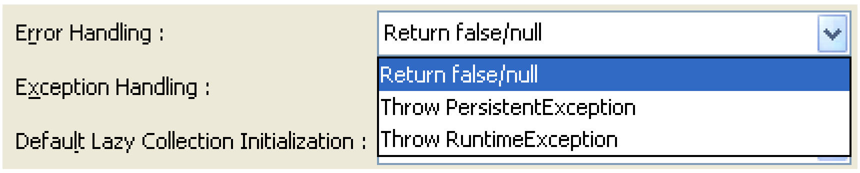


Figure 6.24 - Error Handling options

For **Exception Handling**, select how to handle the exception.

- **Do not Show** - It hides the error message.
- **Print to Error Stream** -It prints the error message to the Error Stream.
- **Print to log4j** - It prints the error message to the log4j library.

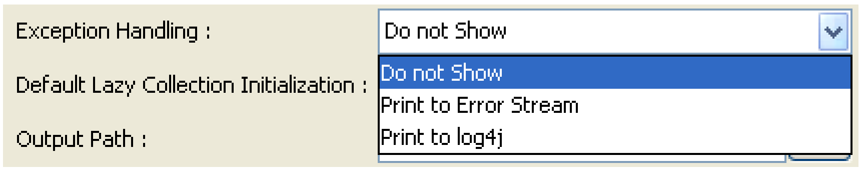

Exception Handling : Do not Show

Figure 6.25 - Exception Handling options

For **Lazy Initialization**, check the option to avoid the associated objects from being loaded when the main object is loaded. Uncheck the option will result in loading the associated objects when the main object is loaded. If you enable (checked) lazy initialization, all associated object (1 to many) will not load until you access it (e.g. getFlight(0)). Enabling this option can usually reduce more than 80% of the database loading.

For **Output Path**, specify the location for storing the generated Java persistent code.

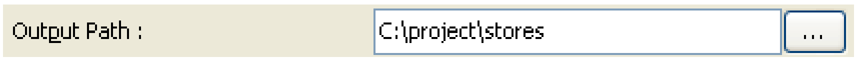
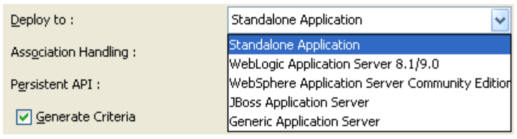

Output Path : C:\project\stores

Figure 6.26 - Output path

For **Deploy To**, specify the template setting for the purpose of the generated code. This setting affects the generated optional Jar and Datasource setting in database connection. You can select **Standalone Application**, **WebLogic Application Server 8.1/9.0**, **WebSphere Application Server Community Edition 1.0**, **JBoss Application Server** and **Generic Application server**.


Deploy to : Standalone Application

Association Handling :
Persistent API :
Generate Criteria
Standalone Application
WebLogic Application Server 8.1/9.0
WebSphere Application Server Community Editio
JBoss Application Server
Generic Application Server

Figure 6.27 - The deployment options

For **Association Handling**, select the type of association handling to be used, either **Smart** or **Standard**.

- **Smart** - With smart association handling, when you update one end of a bi-directional association, the generated persistent code is able to update the other end automatically. Besides, you do not need to cast the retrieved object(s) into its corresponding persistence class when retrieving object(s) from the collection.
- **Standard** - With standard association handling, you must update both ends of a bi-directional association manually to maintain the consistency of association. Besides, casting of object(s) to its corresponding persistence class is required when retrieving object(s) from the collection.

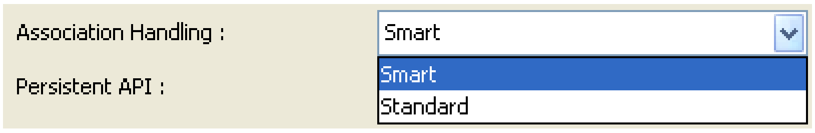

Association Handling : Smart

Figure 6.28 - Association Handling options

For **Persistent API**, select the type of persistent code to be generated, either Static Methods, Factory Class, DAO or POJO.

- **Static Method** -Client can create, retrieve and persist with the PersistentObject directly.
- **Factory Class** -FactoryObject class will be generated for client to create and retrieve the PersistentObject. Client can directly persist with the PersistentObject.
- **DAO** -The PersistentObjectDAO class helps client to create, retrieve and persists the PersistentObject.
- **POJO** -The PersistentManager helps client to retrieve and persist the PersistentObject.

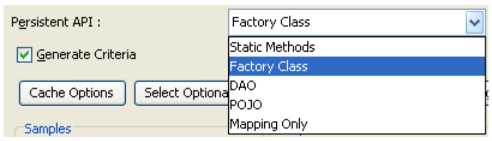


Figure 6.29 - Persistent API options

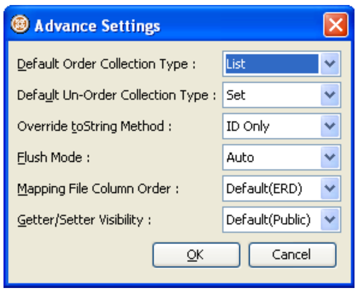
For **Generate Criteria**, check this option to generate the criteria class for each ORM Persistable class. The criteria class is used for querying the database in an object-oriented way (please refer to Chapter 9 for more details about using criteria)

For **Samples**, sample files, including Java application sample, Servlet sample and Java Server Page (JSP) sample are available for generation. The generated sample files guide you through the usage of the Java persistence class. You can check the options to generate the sample files for reference.

You have to select samples for generation in this example so that you can modify the sample file to test and execute the generated Java code.

For **Script**, you can check the options to generate the scripts, including Ant File, Batch and Shell Script which are available for generation. You can execute the generated scripts directly.

For **Web Application Deployment Descriptors (web.xml),** you can check this option to generate web.xml file for application server and it contains the filter class information for servlet.


Advance Settings

Default Order Collection Type : List
Default Un-Order Collection Type : Set
Override toString Method : ID Only
Flush Mode : Auto
Mapping File Column Order : Default(ERD)
Getter/Setter Visibility : Default(Public)

Figure 6.30 - Advance Settings options

For **Advance Setting**, you can define the Default Order Collection Type, Default Un-Order Collection Type, Override toString Method and Flush Mode.

- **Default Order Collection Type** -Select the type of ordered collection to be used in handling multiple cardinality relationship, either **List** or **Map**.

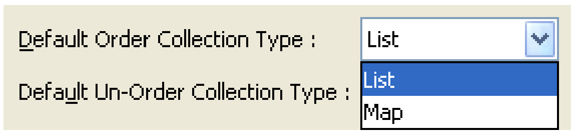

Default Order Collection Type :
List
Default Un-Order Collection Type :
List
Map

Figure 6.31 - Default Order Collection Type options

- **Default Un-Order Collection Type** -Select the type of un-ordered collection to be used in handling the multiple cardinality relationship, either **Set** or **Bag**.

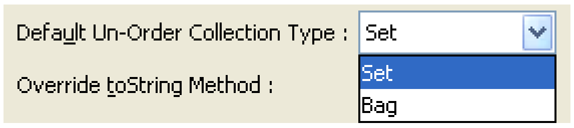
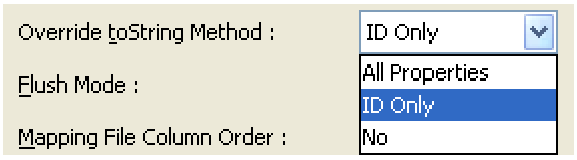

Default Un-Order Collection Type :
Set
Override toString Method :
Set
Bag

Figure 6.32 - Default Un-Order Collection Type options

- **Override toString Method** -Select the way that you want to override the toString method of the object. There are three options provided to override the toString method as follows:
  - **ID Only** -the toString method returns the value of the primary key of the object as string.
  - **All Properties** -the toString method returns a string with the pattern "Entity[<column1_name>=<column1_value><column2_name>=(column2_value>...]"
  - **No** -the toString method will not be overridden

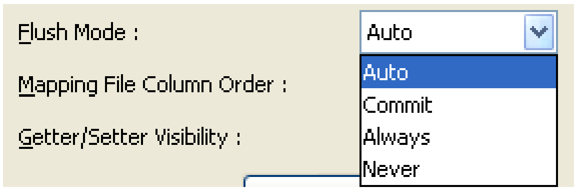


Override toString Method : ID Only
Flush Mode :
Mapping File Column Order :

Figure 6.33 - Override toString Method options

You have to select All Properties of Override toString Method when you execute the list data sample, so you can easily print out the persistent object information.

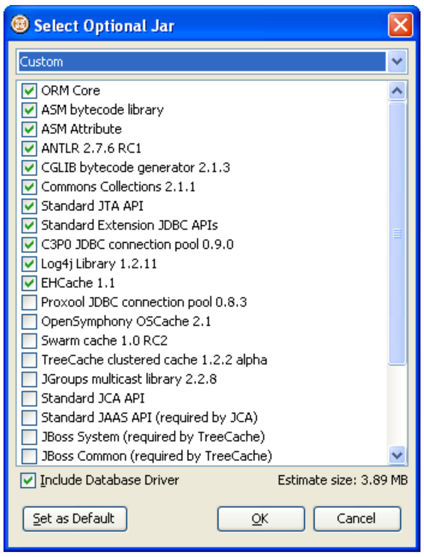
- **Flush Mode** -Select the Flush Mode to be used in flushing strategy. User can select Auto, Commit, Always and Never.
  - **Auto** -The Session is sometimes flushed before executing query.
  - **Commit** -The Session is flushed when committing Transaction.
  - **Always** -The Session is flushed before every query.
  - **Never** -The Session is never flushed unless the flush method is called.



Flush Mode : Auto
Mapping File Column Order :
Getter/Setter Visibility :

Figure 6.34 - Flush Mode options

For **Select Optional Jar**, you can select the libraries and JDBC driver to be included in the generation of the **orm.jar** file (Persistent Library).



Select Optional Jar

Custom

- [✓] ORM Core
- [✓] ASM bytecode library
- [✓] ASM Attribute
- [✓] ANTLR 2.7.6 RC1
- [✓] CGLIB bytecode generator 2.1.3
- [✓] Commons Collections 2.1.1
- [✓] Standard JTA API
- [✓] Standard Extension JDBC APIs
- [✓] C3P0 JDBC connection pool 0.9.0
- [✓] Log4j Library 1.2.11
- [✓] EHCache 1.1
- [ ] Proxool JDBC connection pool 0.8.3
- [ ] OpenSymphony OSCache 2.1
- [ ] Swarm cache 1.0 RC2
- [ ] TreeCache clustered cache 1.2.2 alpha
- [ ] JGroups multicast library 2.2.8
- [ ] Standard JCA API
- [ ] Standard JAAS API (required by JCA)
- [ ] JBoss System (required by TreeCache)
- [ ] JBoss Common (required by TreeCache)

[✓] Include Database Driver Estimate size: 3.89 MB

Figure 6.35 - Select Optional Jar dialog

- Select the desired template from the drop-down menu for creating the orm.jar file.

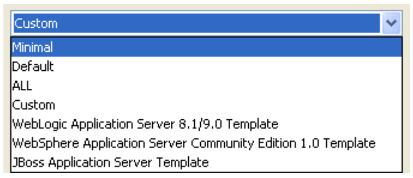

Custom
Minimal
Default
ALL
Custom
WebLogic Application Server 8.1/9.0 Template
WebSphere Application Server Community Edition 1.0 Template
JBoss Application Server Template

Figure 6.36 - Select the desired template

Click **Finish**, the **Generate ORM Code/Database** dialog box appears showing the progress of code generation. Click **Close** when the generation is complete.

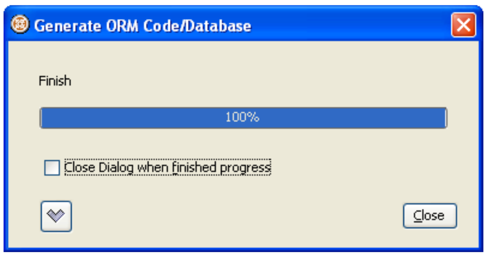

Generate ORM Code/Database
Finish
100%
Close Dialog when finished progress
Close

Figure 6.37 - Generate ORM Code/Database dialog

A class diagram and an entity relationship diagram will be generated automatically and added to your project. The generated persistent code and required resources will be generated to the specified output path.

Class Diagram:

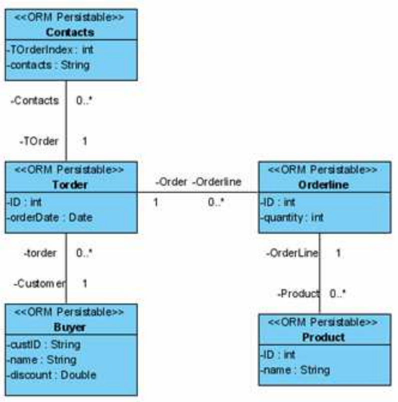


Figure 6.38 - The Class Diagram mapping with the reversed ERD

ER Diagram:

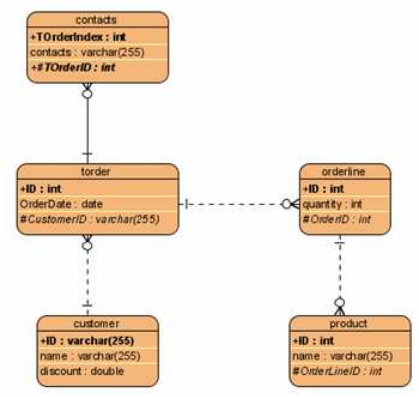
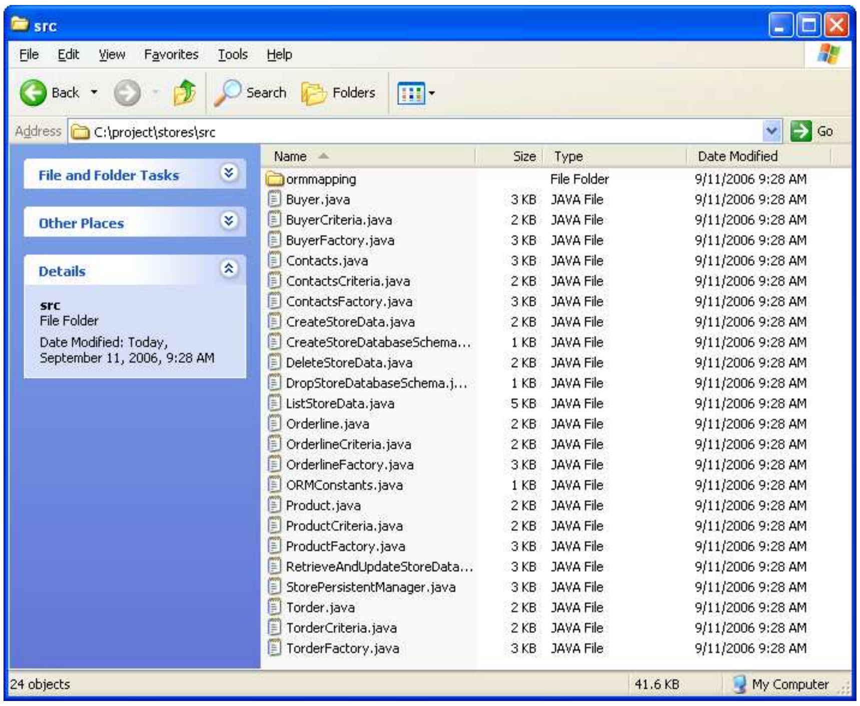


Figure 6.39 - The reversed ERD

Generated Code:



C:\project\stores\src

src
File Folder
Date Modified: Today, September 11, 2006, 9:28 AM

Name	Size	Type	Date Modified
ormmapping		File Folder	9/11/2006 9:28 AM
Buyer.java	3 KB	JAVA File	9/11/2006 9:28 AM
BuyerCriteria.java	2 KB	JAVA File	9/11/2006 9:28 AM
BuyerFactory.java	3 KB	JAVA File	9/11/2006 9:28 AM
Contacts.java	3 KB	JAVA File	9/11/2006 9:28 AM
ContactsCriteria.java	2 KB	JAVA File	9/11/2006 9:28 AM
ContactsFactory.java	3 KB	JAVA File	9/11/2006 9:28 AM
CreateStoreData.java	2 KB	JAVA File	9/11/2006 9:28 AM
CreateStoreDatabaseSchema...	1 KB	JAVA File	9/11/2006 9:28 AM
DeleteStoreData.java	2 KB	JAVA File	9/11/2006 9:28 AM
DropStoreDatabaseSchema.j...	1 KB	JAVA File	9/11/2006 9:28 AM
ListStoreData.java	5 KB	JAVA File	9/11/2006 9:28 AM
Orderline.java	2 KB	JAVA File	9/11/2006 9:28 AM
OrderlineCriteria.java	2 KB	JAVA File	9/11/2006 9:28 AM
OrderlineFactory.java	3 KB	JAVA File	9/11/2006 9:28 AM
ORMConstants.java	1 KB	JAVA File	9/11/2006 9:28 AM
Product.java	2 KB	JAVA File	9/11/2006 9:28 AM
ProductCriteria.java	2 KB	JAVA File	9/11/2006 9:28 AM
ProductFactory.java	3 KB	JAVA File	9/11/2006 9:28 AM
RetrieveAndUpdateStoreData...	3 KB	JAVA File	9/11/2006 9:28 AM
StorePersistentManager.java	3 KB	JAVA File	9/11/2006 9:28 AM
Torder.java	2 KB	JAVA File	9/11/2006 9:28 AM
TorderCriteria.java	2 KB	JAVA File	9/11/2006 9:28 AM
TorderFactory.java	3 KB	JAVA File	9/11/2006 9:28 AM

24 objects 41.6 KB

Figure 6.40 - The generated code

Using Generated Sample

You have selected to generate the sample for the persistent code, so you can modify the Java sample code slightly to test and execute the Java code. If you have entered the package name for the generated Java code, the sample code will be generated in the **ormsamples** package. The ormsamples package contains the following files:

Class File	Function
CreateStoreData.java	Create Persistent Objects and Save objects to database.
CreateStoreDatabaseSchema.java	Export the schema to database
DeleteStoreData.java	Delete Persistent Objects from the database.
DropStoreDatabaseSchema.java	Remove the Schema from the database.
ListStoreData.java	List all the Persistent Object in database.
RetrieveAndUpdateStoreData.java	Get the persistent object from the database and modify the object attributes.

Table 6.2

We will demonstrate how to modify the CreateStoreData.java to create the persistent object and relationship from the generated Java Code and save the Persistent object to database.

The original CreateTestData() method of CreateStoreData.java file:

```
public void createTestData() throws PersistentException {
      PersistentTransaction t =
      store.StorePersistentManager.instance().getSession().beginTransaction();
      try {
            store.Contacts lstoreContacts = store.ContactsFactory.createContacts();
            // Initialize the properties of the persistent object
            lstoreContacts.save();
            store.Buyer lstoreBuyer = store.BuyerFactory.createBuyer();
            // Initialize the properties of the persistent object
            lstoreBuyer.save();
            store.Orderline lstoreOrderline = store.OrderlineFactory.createOrderline();
            // Initialize the properties of the persistent object
            lstoreOrderline.save();
            store.Product lstoreProduct = store.ProductFactory.createProduct();
            // Initialize the properties of the persistent object
            lstoreProduct.save();
            store.Torder lstoreTorder = store.TorderFactory.createTorder();
            // Initialize the properties of the persistent object
            lstoreTorder.save();
            t.commit();
      }
      catch (Exception e) {
            t.rollback();
      }
}
```

The modified CreateTestData() method of CreateStoreData.java file:

```
public void createTestData() throws PersistentException {
      PersistentTransaction t =
      store.StorePersistentManager.instance().getSession().beginTransaction();
      try {
            //create persistent object instance
            //create Contacts
            System.out.print("Create persistent objects.");
            store.Contacts lstoreContacts = store.ContactsFactory.createContacts();
            lstoreContacts.setContact("contact : 12345678");
            lstoreContacts.setTOrderIndex(1);

            //create Buyer
            store.Buyer lstoreBuyer = store.BuyerFactory.createBuyer();
            lstoreBuyer.setDiscount(0.8);
            lstoreBuyer.setName("Judy");
            lstoreBuyer.setCustID("july");

            //create Torder
            store.Torder lstoreTorder = store.TorderFactory.createTorder();
            lstoreTorder.setOrderDate(new Date());

            //create Orderline
            store.Orderline lstoreOrderline = store.OrderlineFactory.createOrderline();
            lstoreOrderline.setQuantity(10);

            //create Product
            store.Product lstoreProduct = store.ProductFactory.createProduct();
            lstoreProduct.setName("Chocolate");

            //create relationship
            System.out.println("Create the relationships between persistent objects.");
            lstoreTorder.setCustomer(lstoreBuyer);
            lstoreContacts.setTOrder(lstoreTorder);
            lstoreOrderline.setOrder(lstoreTorder);
            lstoreProduct.setOrderLine(lstoreOrderline);

            //save the persistent object
            System.out.println("Save the persistent objects.");
            lstoreBuyer.save();
            t.commit();
      }
      catch (Exception e) {
            t.rollback();
      }
}
```

Execute the modified CreateStoreData.java. The persistent objects will be saved to the database. You can execute the ListStoreData.java to show all the created persistent object information.

The ListStoreData.java file's listTestData() method:

```
public void listTestData() throws PersistentException {
      System.out.println("Listing Contacts...");
      store.Contacts[] lstoreContactss = store.ContactsFactory.listContactsByQuery(null,
      null);
      int length = Math.min(lstoreContactss.length, ROW_COUNT);
      for (int i = 0; i < length; i++) {
            System.out.println(lstoreContactss[i]);
      }
      System.out.println(length + " record(s) retrieved.");
```

```
        System.out.println("Listing Buyer...");
        store.Buyer[] lstoreBuyers = store.BuyerFactory.listBuyerByQuery(null, null);
        length = Math.min(lstoreBuyers.length, ROW_COUNT);
        for (int i = 0; i < length; i++) {
              System.out.println(lstoreBuyers[i]);
        }
        System.out.println(length + " record(s) retrieved.");

        System.out.println("Listing Orderline...");
        store.Orderline[] lstoreOrderlines = store.OrderlineFactory.listOrderlineByQuery(null,
        null);
        length = Math.min(lstoreOrderlines.length, ROW_COUNT);
        for (int i = 0; i < length; i++) {
              System.out.println(lstoreOrderlines[i]);
        }
        System.out.println(length + " record(s) retrieved.");

        System.out.println("Listing Product...");
        store.Product[] lstoreProducts = store.ProductFactory.listProductByQuery(null, null);
        length = Math.min(lstoreProducts.length, ROW_COUNT);
        for (int i = 0; i < length; i++) {
              System.out.println(lstoreProducts[i]);
        }
        System.out.println(length + " record(s) retrieved.");

        System.out.println("Listing Torder...");
        store.Torder[] lstoreTorders = store.TorderFactory.listTorderByQuery(null, null);
        length = Math.min(lstoreTorders.length, ROW_COUNT);
        for (int i = 0; i < length; i++) {
              System.out.println(lstoreTorders[i]);
        }
        System.out.println(length + " record(s) retrieved.");
}
```

The result of executing ListStoreData.java:

```
Listing Contacts...
Contacts[ TOrderIndex=1 Contacts=contact : 12345678 TOrder.Persist_ID=1 ]
1 record(s) retrieved.
Listing Buyer...
Buyer[ CustID=july Name=Judy Discount=0.8 torder.size=1 ]
1 record(s) retrieved.
Listing Orderline...
Orderline[ ID=1 Quantity=10000 Order.Persist_ID=1 Product.size=1 ]
1 record(s) retrieved.
Listing Product...
Product[ ID=1 Name=Chocolate OrderLine.Persist_ID=1 ]
1 record(s) retrieved.
Listing Torder...
Torder[ ID=1 OrderDate=2006-01-03 Customer.Persist_ID=july Contacts.size=1 Orderline.size=1 ]
1 record(s) retrieved.
```

You can also modify the other sample file to test the generated Java code.