

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
- Creating Sample Data
- Start Generating Code from Database Wizard
- Configuring Database
- Selecting Tables
- Configuring Generated Classes Details
- Specifying Code Generation Details
- Using Generated Sample

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.

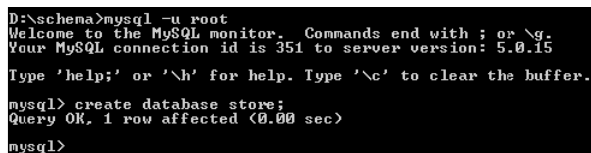
The MySQL 5.0 Community Edition can be downloaded at <http://dev.mysql.com/downloads/mysql/5.0.html>.

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**.

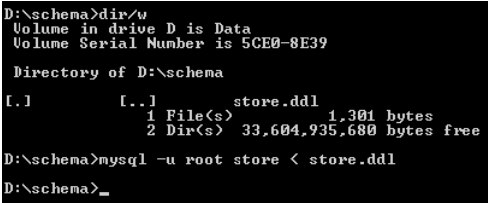
```

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

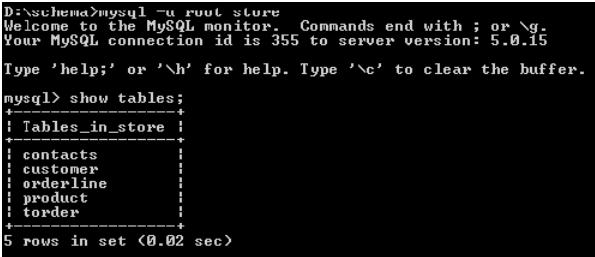
D:\schema>mysql -u root store < store.ddl
D:\schema>_

```

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" .

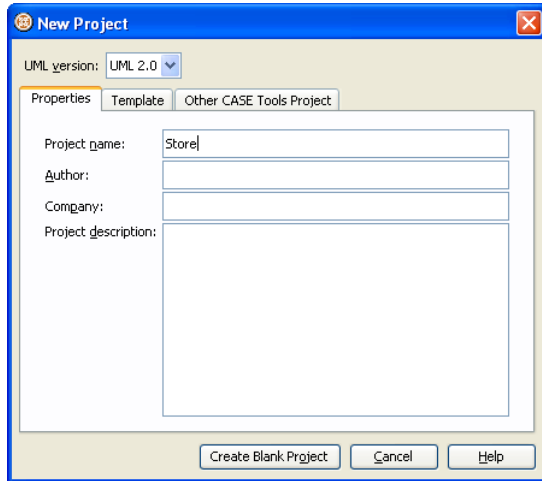


Figure 6.4 - New project dialog

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

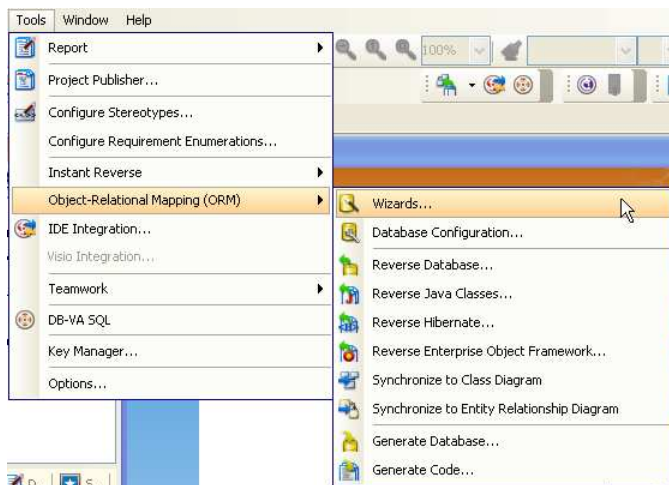


Figure 6.5 - To open the ORM Wizard

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

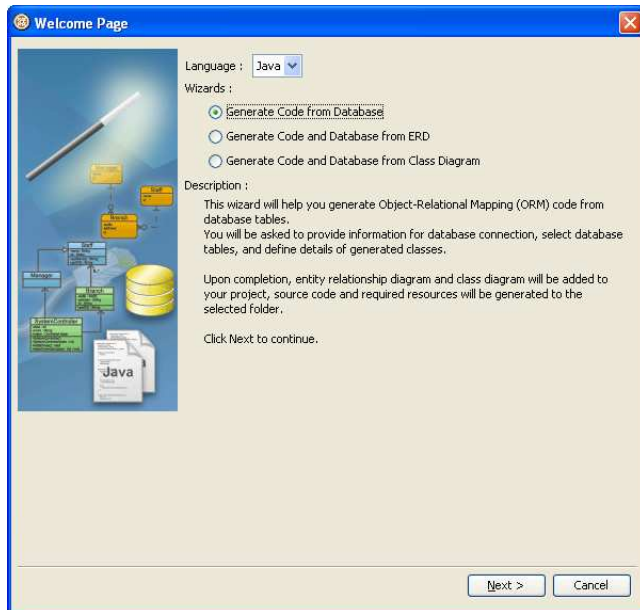


Figure 6.6 - Welcome Page of the ORM Wizard

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.

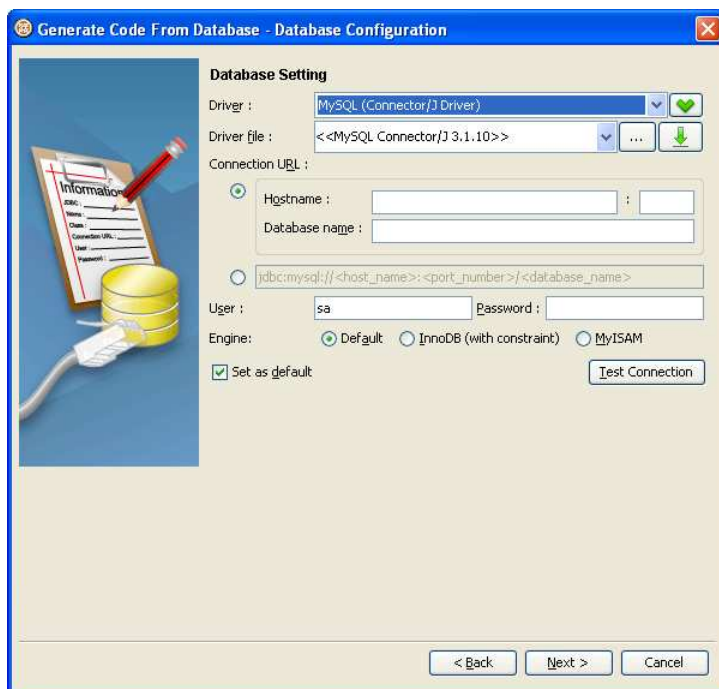


Figure 6.7 - Database Configuration

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

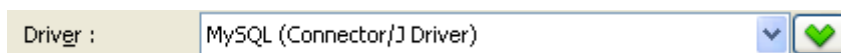


Figure 6.8 - Driver options

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

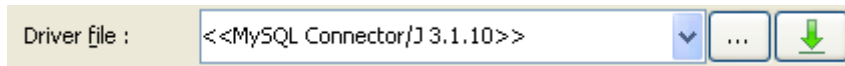


Figure 6.9 - Driver file options

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.

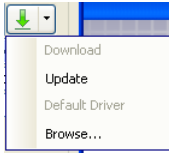


Figure 6.10 - Download driver options

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.

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

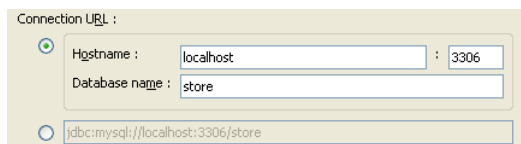


Figure 6.11 - The Connection URL

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.

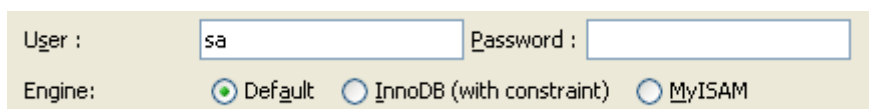


Figure 6.12 - The database engine, username and password



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**.

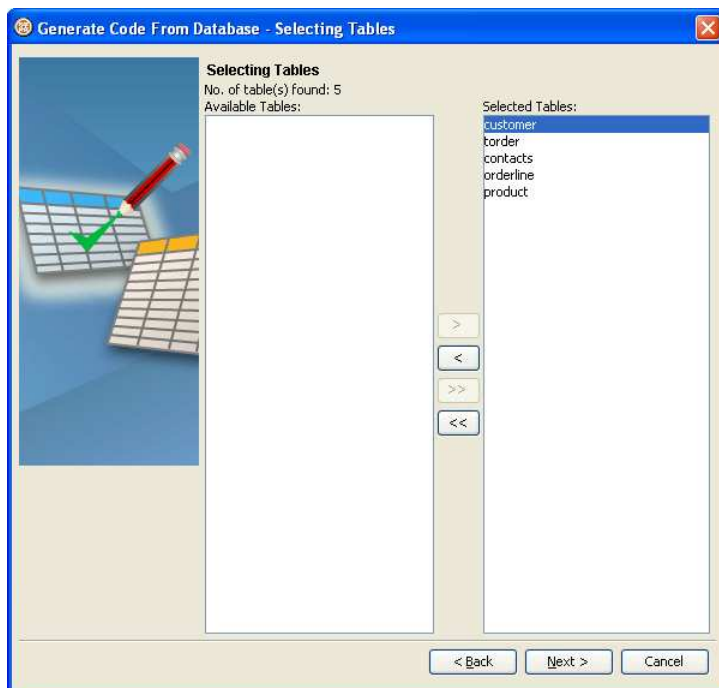


Figure 6.15 - Selecting Table

-  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...

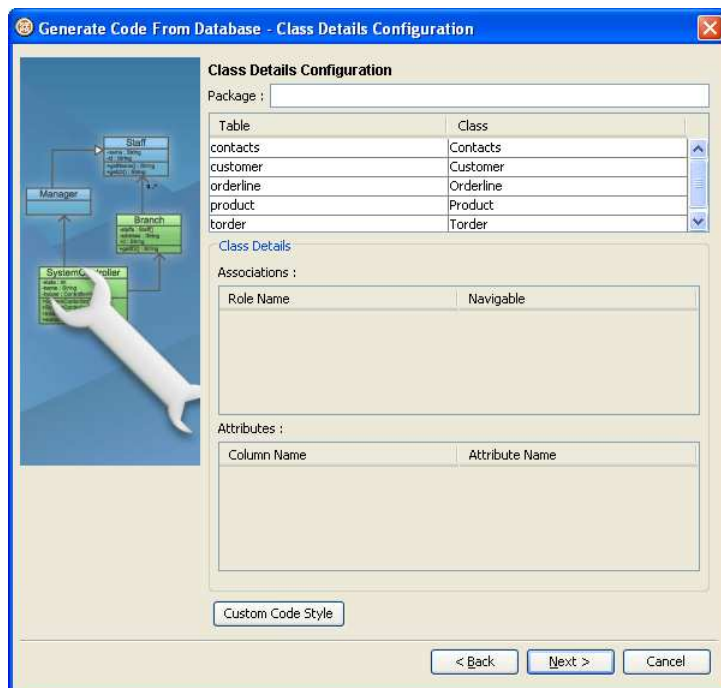


Figure 6.16 - Configuring Generated Class Details

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.



Figure 6.18 - Mapping Classes

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



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.

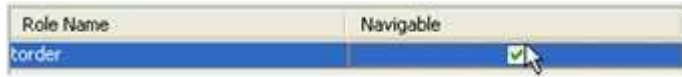


Figure 6.20 - Select the Navigable

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



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**.

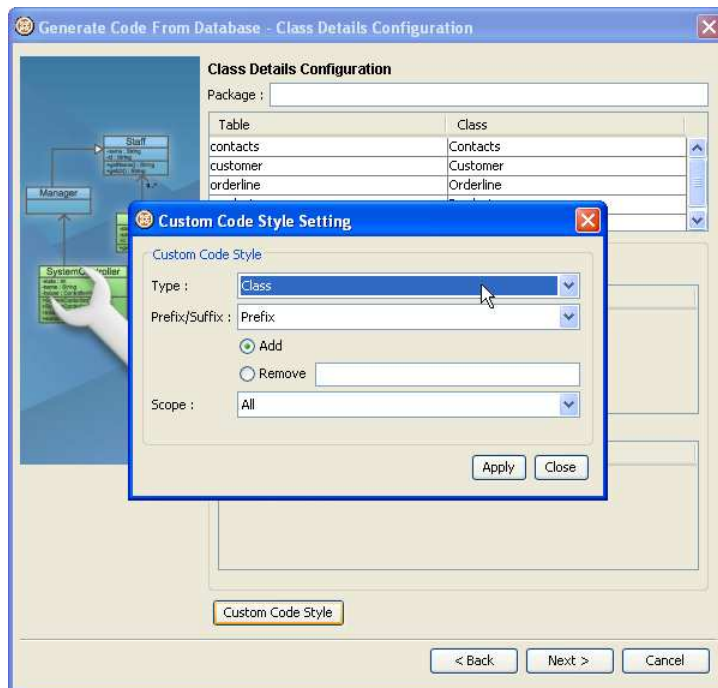


Figure 6.22 - Custom Code Style Setting

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.

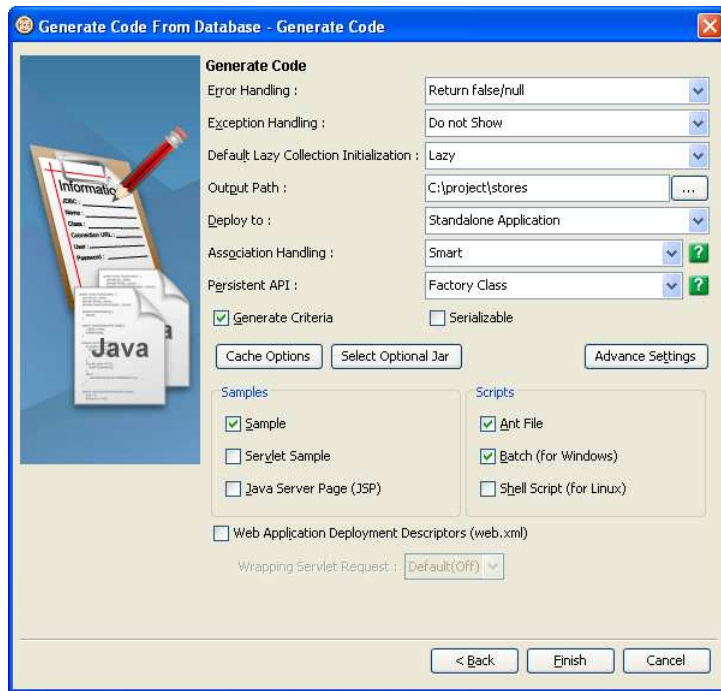


Figure 6.23 - Generate code options

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.

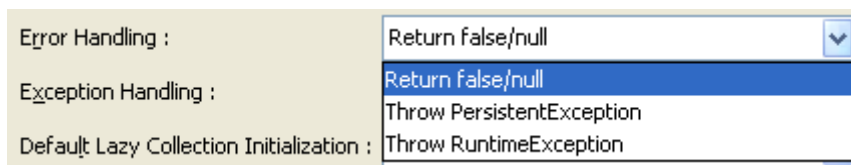


Figure 6.24 - Error Handling options

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.

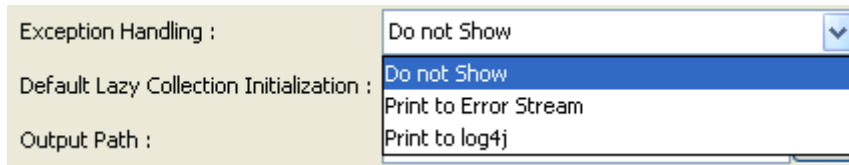


Figure 6.25 - Exception Handling options

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.

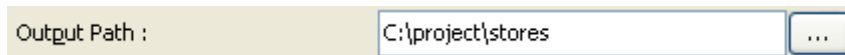


Figure 6.26 - Output path

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**.

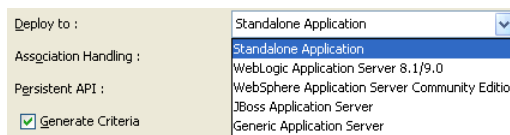


Figure 6.27 - The deployment options

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.

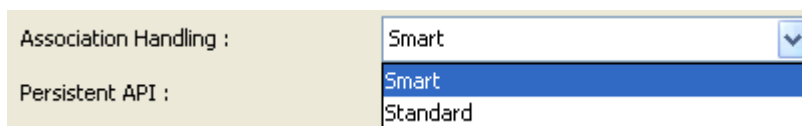


Figure 6.28 - Association Handling options

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.

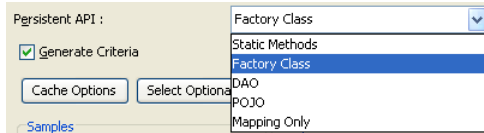



Figure 6.29 - Persistent API options

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.

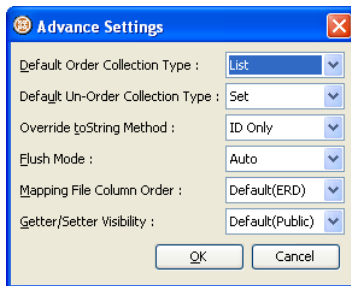


Figure 6.30 - Advance Settings options

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**.

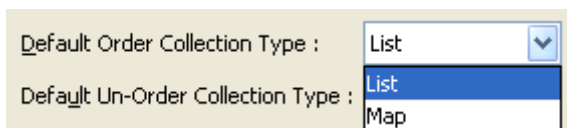


Figure 6.31 - Default Order Collection Type options

- **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**.

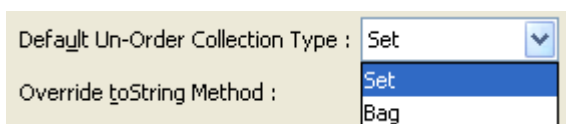


Figure 6.32 - Default Un-Order Collection Type options

- **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

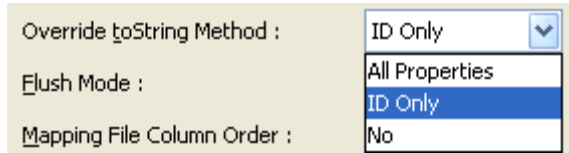


Figure 6.33 - Override toString Method options



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.

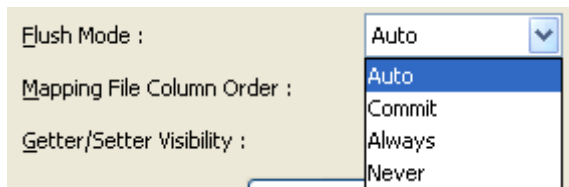


Figure 6.34 - Flush Mode options

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).

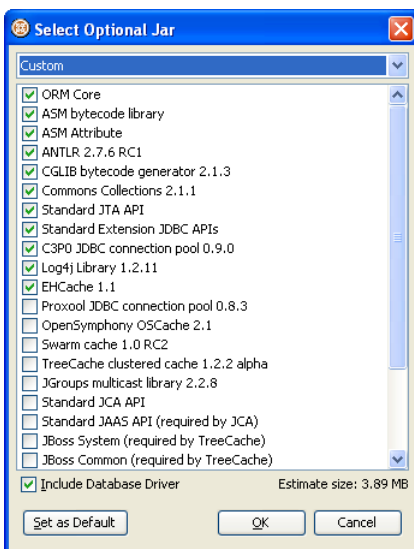


Figure 6.35 - Select Optional Jar dialog

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

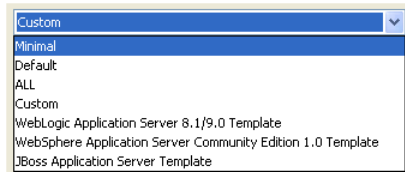


Figure 6.36 - Select the desired template

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

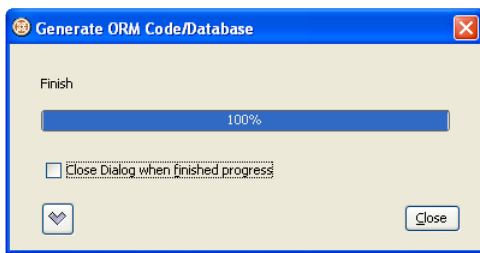


Figure 6.37 - Generate ORM Code/Database dialog

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:

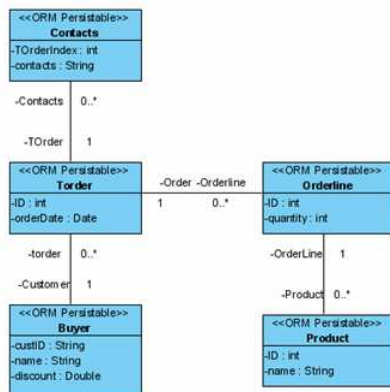


Figure 6.38 - The Class Diagram mapping with the reversed ERD

ER Diagram:

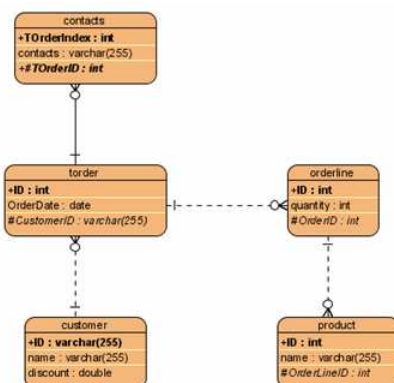


Figure 6.39 - The reversed ERD

Generated Code:

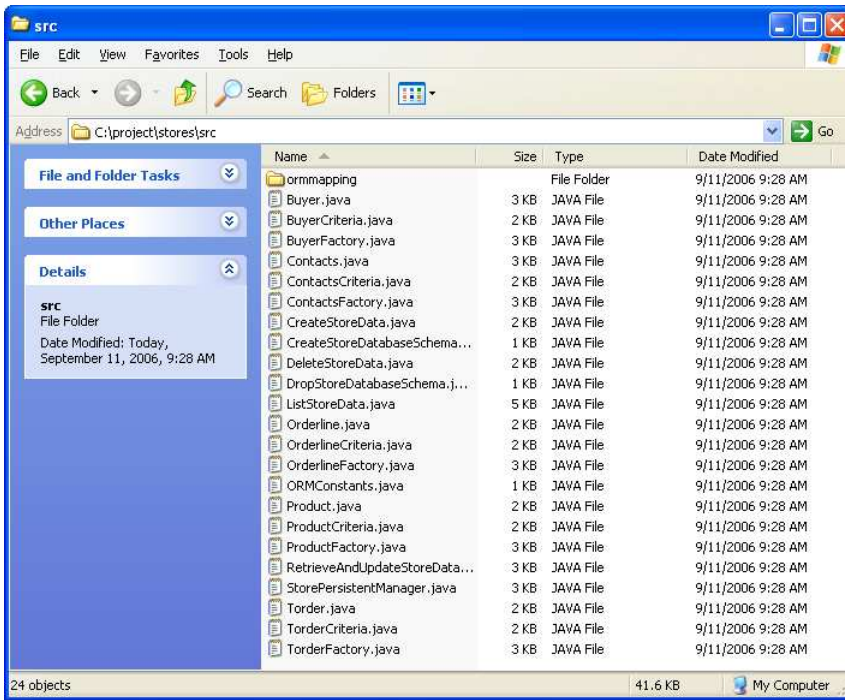


Figure 6.40 - The generated code

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.println("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.

