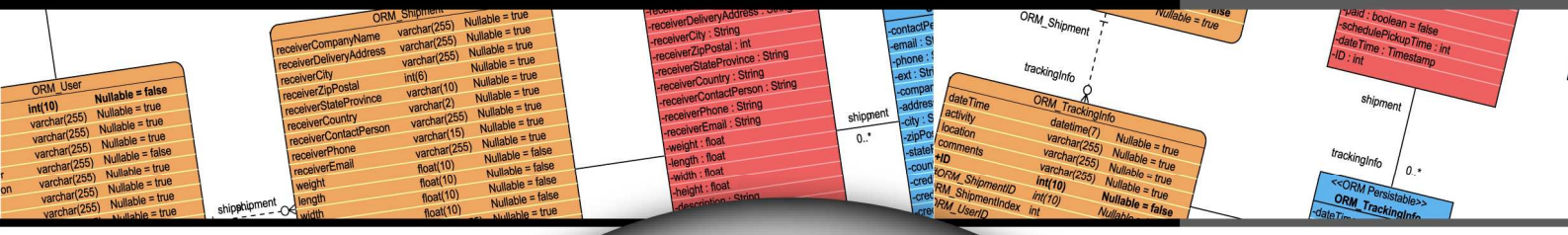
Visual Paradigm

Build Quality Applications Faster, Better and Cheaper

Database Visual ARCHITECT Designer's Guide

Access database with Object-Oriented technology



DB Visual ARCHITECT 4.0 Designer's Guide

The software and documentation are furnished under the DB Visual ARCHITECT license agreement and may be used only in accordance with the terms of the agreement.

DB Visual ARCHITECT License Agreement

THE USE OF THE SOFTWARE LICENSED TO YOU IS SUBJECT TO THE TERMS AND CONDITIONS OF THIS SOFTWARE LICENSE AGREEMENT. BY INSTALLING, COPYING, OR OTHERWISE USING THE SOFTWARE, YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT, UNDERSTAND IT, AND AGREE TO BE BOUNDED BY ALL OF THE TERMS AND CONDITIONS OF THIS SOFTWARE LICENSE AGREEMENT.

1. **Limited License Grant.** Visual Paradigm grants to you ("the Licensee") a personal, non-exclusive, non-transferable, limited, perpetual, revocable license to install and use Visual Paradigm Products ("the Software" or "the Product"). The Licensee must not re-distribute the Software in whole or in part, either separately or included with a product.
2. **Restrictions.** The Software is confidential copyrighted information of Visual Paradigm, and Visual Paradigm and/or its licensors retain title to all copies. The Licensee shall not modify, adapt, decompile, disassemble, decrypt, extract, or otherwise reverse engineer the Software. Software may not be leased, rented, transferred, distributed, assigned, or sublicensed, in whole or in part. The Software contains valuable trade secrets. The Licensee promises not to extract any information or concepts from it as part of an effort to compete with the licensor, nor to assist anyone else in such an effort. The Licensee agrees not to remove, modify, delete or destroy any proprietary right notices of Visual Paradigm and its licensors, including copyright notices, in the Software.
3. **Disclaimer of Warranty.** The software and documentation are provided "AS IS," WITH NO WARRANTIES WHATSOEVER. ALL EXPRESS OR IMPLIED REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. THE ENTIRE RISK AS TO SATISFACTORY QUALITY, PERFORMANCE, ACCURACY AND EFFORT IS WITH THE LICENSEE. THERE IS NO WARRANTY THE DOCUMENTATION, Visual Paradigm's EFFORTS OR THE LICENSED SOFTWARE WILL FULFILL ANY OF LICENSEE'S PARTICULAR PURPOSES OR NEEDS. IF THESE WARRANTIES ARE UNENFORCEABLE UNDER APPLICABLE LAW, THEN Visual Paradigm DISCLAIMS SUCH WARRANTIES TO THE MAXIMUM EXTENT PERMITTED BY SUCH APPLICABLE LAW.
4. **Limitation of Liability.** Visual Paradigm AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY THE LICENSEE OR ANY THIRD PARTY AS A RESULT OF USING OR DISTRIBUTING SOFTWARE. IN NO EVENT WILL Visual Paradigm OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, EXEMPLARY, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF Visual Paradigm HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

5. **Termination.** The Licensee may terminate this License at any time by destroying all copies of Software. Visual Paradigm will not be obligated to refund any License Fees, if any, paid by the Licensee for such termination. This License will terminate immediately without notice from Visual Paradigm if the Licensee fails to comply with any provision of this License. Upon such termination, the Licensee must destroy all copies of the Software. Visual Paradigm reserves all rights to terminate this License.

SPECIFIC DISCLAIMER FOR HIGH-RISK ACTIVITIES. The SOFTWARE is not designed or intended for use in high-risk activities including, without restricting the generality of the foregoing, on-line control of aircraft, air traffic, aircraft navigation or aircraft communications; or in the design, construction, operation or maintenance of any nuclear facility. Visual Paradigm disclaims any express or implied warranty of fitness for such purposes or any other purposes.

NOTICE. The Product is not intended for personal, family or household use; rather, it is intended exclusively for professional use. Its utilization requires skills that differ from those needed to use consumer software products such as word processing or spreadsheet software.

GOVERNMENT RIGHTS. If the Software is licensed by or on behalf of a unit or agency of any government, the Licensee agrees that the Software is "commercial computer software", "commercial computer software documentation" or similar terms and that, in the absence of a written agreement to the contrary, the Licensee's rights with respect to the Software are limited by the terms of this Agreement.

Acknowledgements

Table of Contents

Chapter 6 - Mapping Object Model to Data Model and vice versa

1

Working with DB Visual ARCHITECT

Chapter 1 - Working with DB Visual ARCHITECT

DB Visual ARCHITECT (DB-VA) is an Object-Relational Mapping tool which supports building database application faster, better and cheaper. This chapter gives you an introduction to DB-VA, describes the key benefits of DB-VA and shows you how to connect your database and download the database driver files automatically to facilitate the working environment. The supported database drivers for connecting database to the DB-VA working environment are also described.

In this chapter:

Introduction

Software applications are most likely to be developed with a database such that all data working with the application system can be retained, resulting in information and knowledge. Hence, database application is widely adopted by businesses of all sizes. In order to access and manipulate the relational database, a standard computer language, Structured Query Language (SQL) has to be used. SQL statements play an important role when developing database application.

Taking a trading system as an example, if the end-user wants to update a Sales Order record, the system has to retrieve the corresponding record from the Sales Order table and display to the end-user. After the end-user confirms the modification of data, the system has to update the record accordingly. It is noticeable that a database application requires a lot of coding for handling SQL statements so as to access and manipulate the database.

Hence, it is inevitable that developers spend almost 50% of development time for implementing the code with SQL statement. Moreover, mapping between the persistent code and database table is maintained throughout the development life cycle. Once there is a change in the structure of a database table, SQL statements which related to the modified table have to be re-written. Developers have to keep an eye on every change in the database schema.

DB Visual ARCHITECT (DB-VA) provides a solution to develop database application. DB-VA is an Object-Relational Mapping tool which provides an ease-to-use environment bridging between object model, data model and relational database. DB-VA not only provides you a visual modeling for both logical data design and physical data design, but also automates the mapping between object model and data model.

DB-VA generates not only Java and .NET persistent code, but also a cost-effective, reliable, scalable and high-performance object to relational mapping layer. The generated mapping layer includes the support of transaction, cache and other optimized feature. DB-VA increases the productivity and significantly reduces the risk of developing the mapping layer manually.

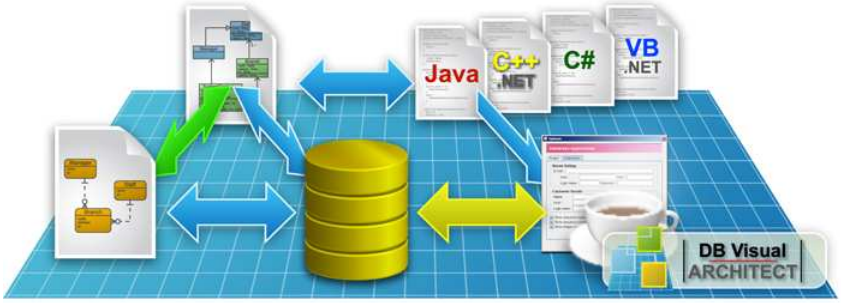


Figure 1.1 - Introduction of Database Visual Architect

The overview of DB-VA

Overview Diagram

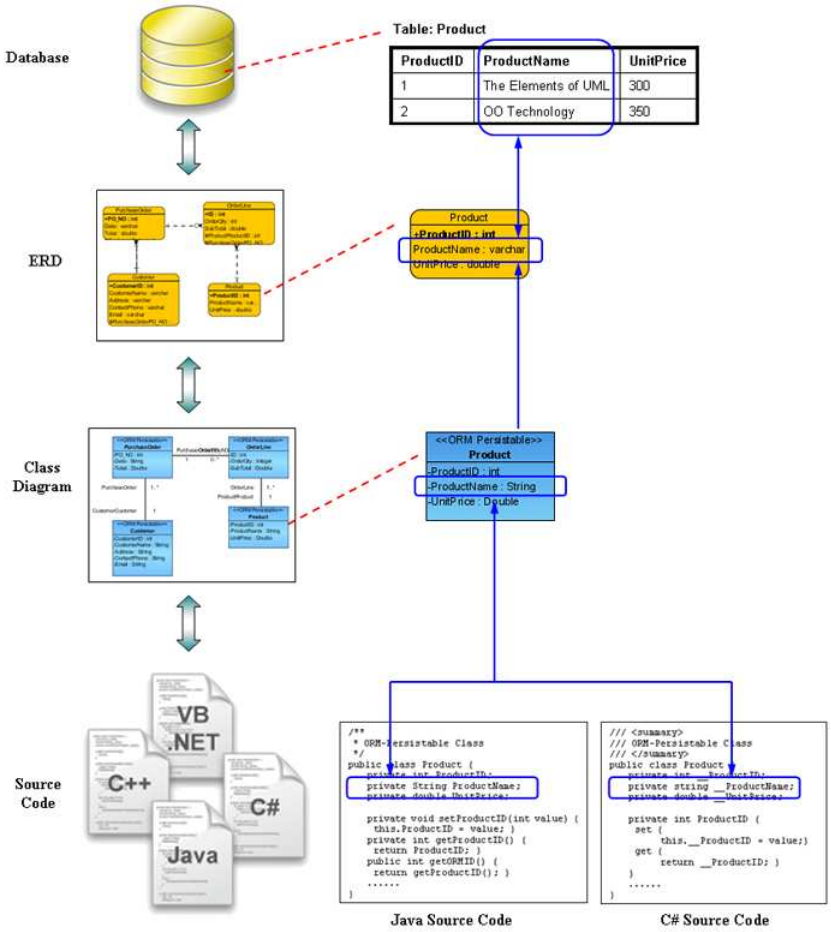


Figure 1.2 - The overview diagram of DB-VA

Key Benefits

DB-VA provides the following key features so as to help you simplify your development:

- Persistence Made Easy

  Traditionally developers spend a lot of effort in saving and loading objects between memory and database which makes the program complicated and difficult to maintain. DB-VA simplifies these tasks by generating a persistence layer between object and data models.

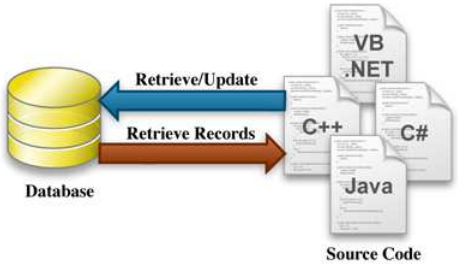


Figure 1.3 - Generate a Persistence Layer for communicate with database

- Sophisticated Object-Relational Mapping Generator

  DB-VA generates object-relational mapping layer which incorporates prime features such as transaction support, pluggable cache layer, connection pool and customizable SQL statement. With this mapping layer, developers can keep away from mundane implementation work and focus on the business requirements.

- Model Driven Development

  DB-VA provides a true model driven platform for application development. DB-VA allows developer not only to start from creating the models by using class diagram or entity relationship diagram to generating the executable persistence layer from the models, but also to modify the entity-relational model which comes from reverse engineering an existing database, transform into object model and generate persistence layer. With the sophisticated model-code generator, the persistent model will be updated automatically according to any modification.

- Extensive Database Coverage

  DB-VA supports a wide range of database, including Oracle, DB2, Cloudscape/Derby, Sybase Adaptive Server Enterprise, Sybase SQL Anywhere, Microsoft SQL Server, PostgreSQL, MySQL and more. DB-VA also promotes an easy migration between databases by enabling the same set of ORM Java objects to work with different databases and transforms the proprietary data type that suit the default database specified.

- Database Reverse Engineering

  DB-VA allows you to reverse engineering an existing database through JDBC into the entity-relational model. Developers can transform the entity-relational model to object model and redesign the database for further development.

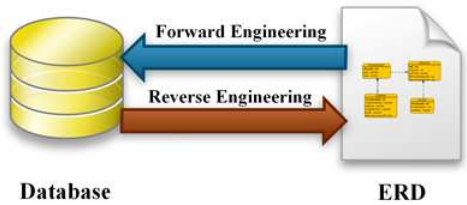


Figure 1.4 - Reverse and Forward Engineering for database

- Class Reverse Engineering

  DB-VA allows you to reverse engineering Java classes and Hibernate models into the persistent object model. Developers can transform the persistent object model to data model and redesign the models for further development.

- IDE Integration

  DB-VA is not only a standalone application, but also integrated to the major Integrated Development Environments (IDEs), including Eclipse/WebSphere®, Borland JBuilder®, NetBeans/Sun™ ONE, IntelliJ IDEA™ Oracle JDeveloper, BEA WebLogic Workshop™and Visual Studio .NET, which results in streamlining the entire model-code-deploy software development process.

Figure 1.5 - The supported IDE

- Visual Modeling for Object and Data Models

  DB-VA inherits the visual modeling environment from Visual Paradigm for UML, a well-known UML CASE Tool, it not only provides an intuitive inline editing for both object and data models, but also adopts the resource-centric interface for assisting frequently performed tasks.

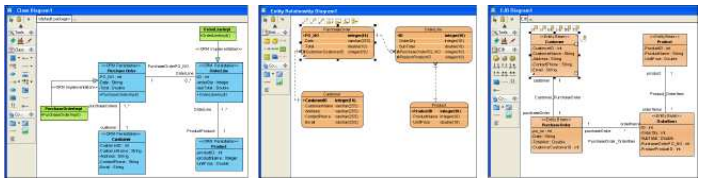


Figure 1.6 - Consistence user interface

Database Configuration

As DB-VA acts as a bridge between object model, data model and relational database, you have to configure the database connection to ensure the environment.

To configure the database connection:

1. From the menu, click **Tools > Object-Relational Mapping (ORM) > Database Configuration...**.

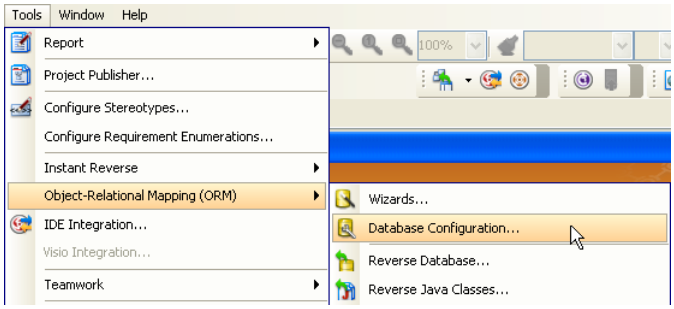


Figure 1.7 - To open the Database Configuration dialog

The **Database Configuration** dialog box is displayed.

2. Select the **Language** for the project to be developed from the drop-down menu. By default, **Java** language is selected.

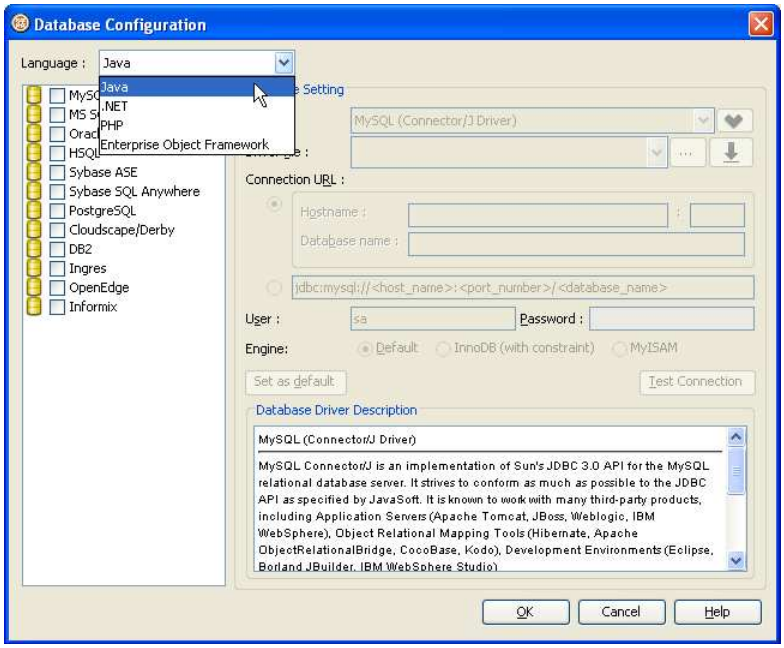


Figure 1.8 - Select the programming language

Database Configuration for Java Project

For Java project development, continue the following steps for configuring the database connection.

1. Place a check mark beside the desired database for configuration.

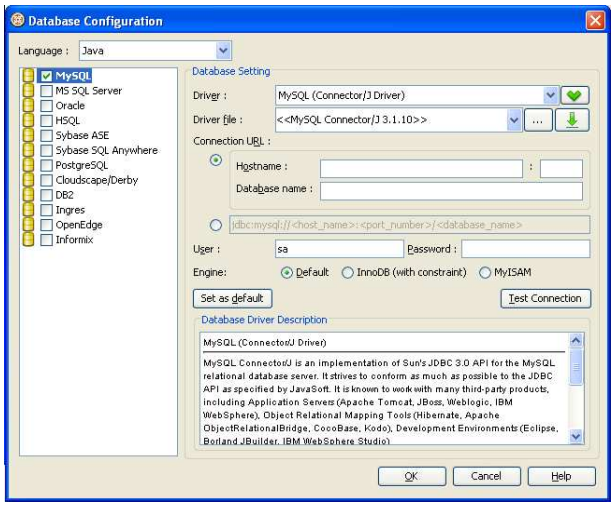


Figure 1.9 - Database Configuration for Java Project

2. Enter the database setting.

   For the **Driver file**, click  button to specify the **Driver file**. The Driver file can be specified either by **Download**, **Update** or **Browse**. For more information, refer to the description of Specifying JDBC Driver File and Supporting Automatic Download of JDBC Driver sections.

   For the **Driver**, select the JDBC Driver from the drop-down menu. The driverï¿½ï¿½s description will be shown in the **Database Driver Description** pane.

   You can press  button to modify the **Driver class** and **Dialect** manually.

   For the **Connection URL**, enter the information required for the JDBC Driver to connect to the database.

   For the **User**, enter the valid username who has the access right to connect to the database.
   For the **Password**, enter the corresponding password for the user to connect to the database.
   For the **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.

3. Click the **Test Connection** button to test whether the database can be connected.

Figure 1.10 - test Connection button

If the database can be connected, you will be prompted by a dialog box showing **Connect Successful**. Otherwise, a **Connection Exception** dialog box will be prompted.

Figure 1.11 - Connection successful/failure message

Configuring Multiple Database Settings

DB-VA supports setting up multiple database configurations in the same project environment. To configure multiple database settings, simply place check marks beside the desired databases and specify the configuration for each database.

Generally, only one database is used for an application. As DB-VA supports multiple database configurations, only one database configuration will be applied as the default database connection for the generation of code and database.

You can set the default database connection in one of the two ways:

- Right-click on the desired database, select **Set as default**.

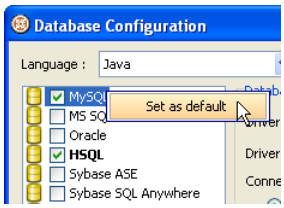


Figure 1.12 - Set the default database type

- Select the desired database, click **Set as default** button on the **Database Setting**.

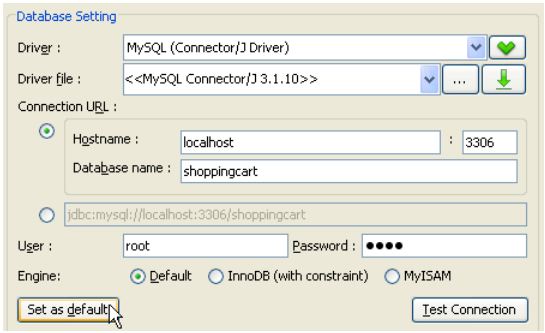


Figure 1.13 - Set the database configure as default

Specifying JDBC Driver File

In order to connect the database successfully, JDBC driver file must be specified for Java project. DB-VA provides three ways to specify the driver files. They are selecting the suitable driver file manually, downloading driver files automatically and updating the driver files automatically.

DB-VA supports the automatic download of JDBC drivers for database connection in Java project development. The drivers downloaded automatically are stored in the %VP_Suite_Installation_Directory%/ormlib/driver directory.

When configuring the database connection for any new projects, DB-VA automatically checks if the desired driver file exists in the driver directory, DB-VA specifies the driver file in the database configuration automatically.

To specify the driver file, click on the  button, either click **Download**, **Update** or **Browse...** from the drop-down menu.

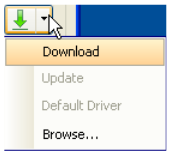


Figure 1.14 - Download JDBC Driver

- **Download**

  If Download is clicked, DB-VA automatically downloads the desired driver file for the desired database. For more information on downloading driver file automatically, refer to the description of Supporting Automatic Download of JDBC Driver section.

- **Update**

  If Update is clicked, DB-VA automatically updates the driver file if there is an update version for the desired database.

- **Browse**

  If Browse is clicked, a File Chooser is shown, specify the location of the driver file.

Note

Update is only available if the driver file is automatically downloaded and stored in the %VP_Suite_Installation_Directory%/ormlib/driver directory in advance.

Supporting Automatic Download of JDBC Driver

As DB-VA supports the automatic download of the driver files for connecting the database, it reduces the effort to find the desired driver file from the Internet manually.

The following steps illustrate the automatic download of JDBC Driver for MySQL database as an example:

1. Click on the  button, click **Download** from the drop-down menu.

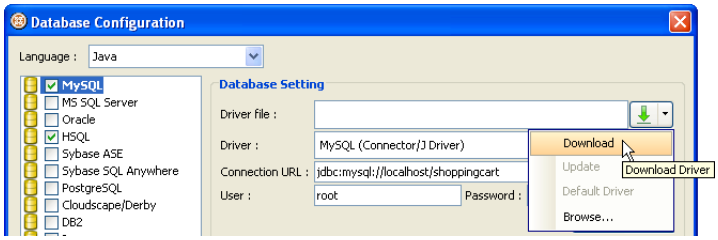


Figure 1.15 - To download the driver

2. A **Download Database Driver** dialog box is shown allowing the proxy setting. To enable proxy for the Internet connection, check the **Use proxy** option, and then fill in the information for proxy setting.

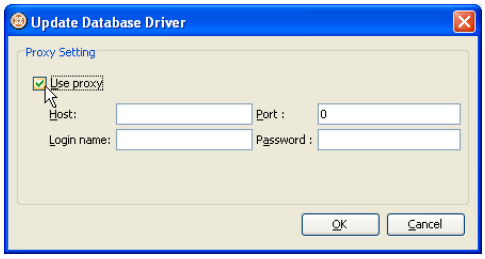


Figure 1.16 - Configure the proxy setting

The **Download** dialog box is shown indicating the download progress.

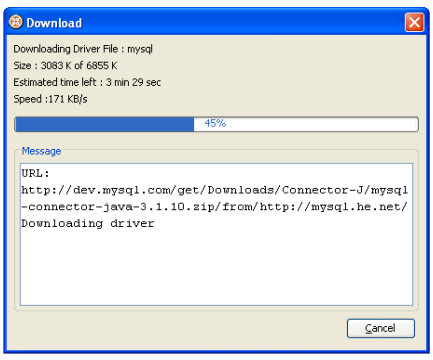


Figure 1.17 - The Download dialog show the download progress

3. Click **Close** when the download is completed.

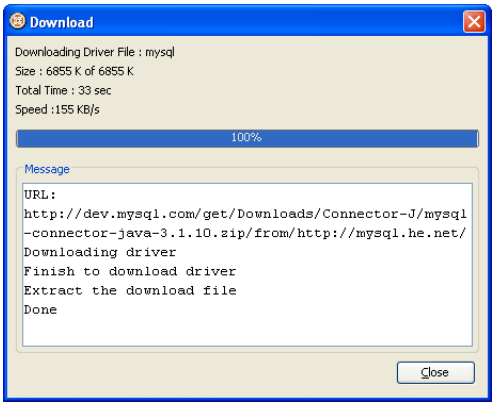


Figure 1.18 - The download complete message show in the download dialog

The driver file is shown on the **Driver file** of the **Database Setting** after download is done.

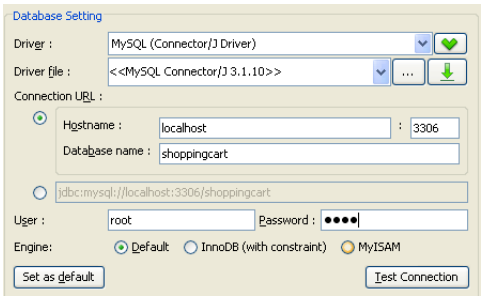


Figure 1.19 - Database setting

Note

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.

Database Configuration for .Net Project

For .Net project development, continue the following steps for configuring the database connection.

1. Place a check mark beside the desired database for configuration.

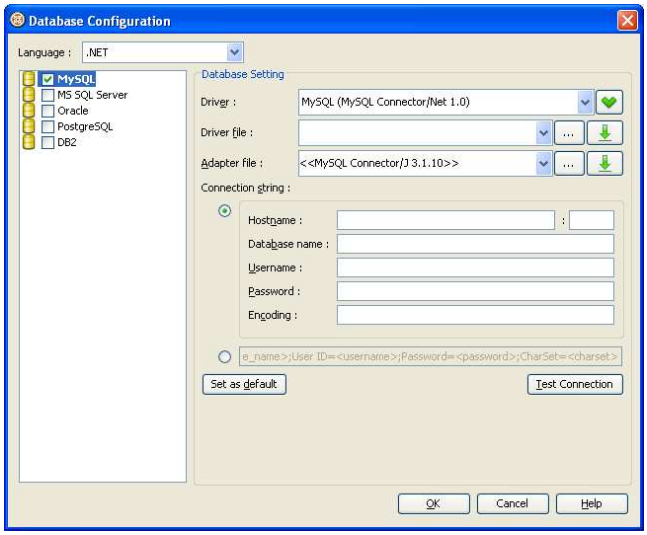


Figure 1.20 - Database Configuration for .Net Project

2. Enter the database setting.

   For the **Driver file**, click  button to specify the **Driver file**. The .NET Driver file can be specified either by Download, Update or Browse. For more information, refer to the description of Specifying .NET Driver and Adapter File and Supporting Automatic Download of .NET Driver and Adapter File section.

   For the **Adapter file**, click  button to specify the **Adapter file**. The Adapter file can be specified either by Download, Update or Browse. For more information, refer to the description of Specifying .NET Driver and Adapter File and Supporting Automatic Download of .NET Driver and Adapter File section.

   For the **Connection String**, enter the information required for the .NET Driver to connect to the database.

   For the **Driver**, select the .NET Driver from the drop-down menu.

   You can press  button to modify the **Driver class** and **Dialect** manually.

3. Click the **Test Connection** button to test whether the database can be connected.

   Figure 1.21 - Test Connection button

   If the database can be connected, you will be prompted by a dialog box showing **Connect Successful**. Otherwise, a **Connection Exception** dialog box will be prompted.

   Figure 1.22 - Connection successful/failure message

Configuring Multiple Database Settings

DB-VA supports setting up multiple database configurations in the same project environment. To configure multiple database settings, simply place check marks beside the desired databases and specify the configuration for each database.

Generally, only one database is used for an application. As DB-VA supports multiple database configurations, only one database configuration will be applied as the default database connection for the generation of code and database.

You can set the default database connection in one of the two ways:

- Right-click on the desired database, select **Set as default**.

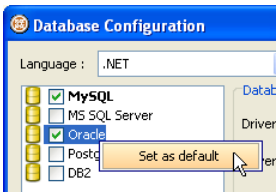
  

  Figure 1.23 - Set the database type as default

- Select the desired database, click **Set as default** button on the **Database Setting**.

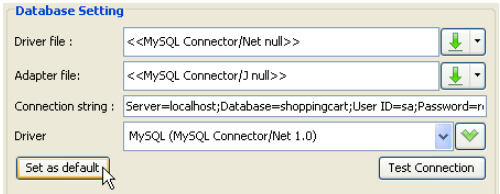
  

  Figure 1.24 - set the database setting as default

Specifying .NET Driver File and Adapter File

In order to connect the database successfully, .NET driver file and Adapter File must be specified for .NET project. DB-VA provides three ways to specify the driver files. They are selecting the suitable driver file manually, downloading driver files automatically and updating the driver files automatically.

DB-VA supports the automatic download of .NET drivers and Adapter File for database connection in .NET project development. The drivers downloaded automatically are stored in the %VP_Suite_Installation_Directory%/ormlib/driver directory.

When configuring the database connection for any new projects, DB-VA automatically checks if the desired driver file exists in the driver directory, DB-VA specifies the driver file in the database configuration automatically.

To specify the driver file, click on the  button, either click **Download**, **Update** or **Browse...**from the drop-down menu.

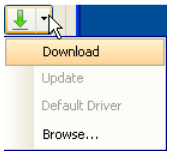


Figure 1.25 - Download button

- **Download**

  If Download is clicked, DB-VA automatically downloads the desired driver file for the desired database. For more information on downloading driver file automatically, refer to the description of Supporting Automatic Download of .NET Driver and Adapter File section.

- **Update**

  If Update is clicked, DB-VA automatically updates the driver file if there is an update version for the desired database.

- **Browse**

  If Browse is clicked, a File Chooser is shown, specify the location of the driver file.

Note

Update is only available if the driver file is automatically downloaded and stored in the %VP_Suite_Installation_Directory%/ormlib/driver directory in advance.

Supporting Automatic Download of .NET Driver and Adapter File

As DB-VA supports the automatic download of the driver files for connecting the database, it reduces the effort to find the desired driver file from the Internet manually.

The following steps illustrate the automatic download of .NET Driver and Adapter File for MySQL database as an example:

1. Click on the  button, click **Download both Driver** from the drop-down menu to download the driver and adapter files at the same time.

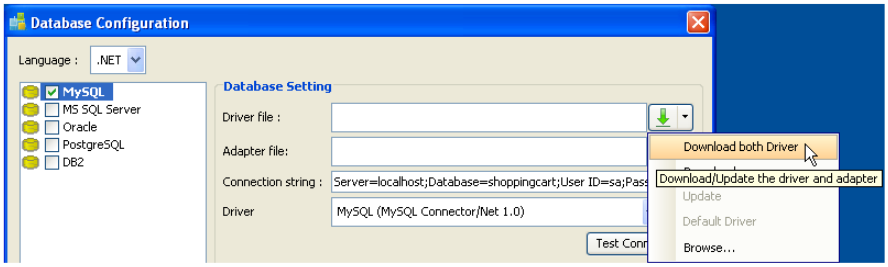


Figure 1.26 - Database Configuration for .NET

Note

The Driver file and Adapter file can be downloaded separately by selecting Download from its drop-down menu respectively.

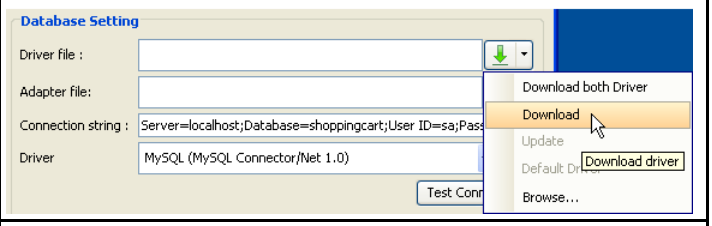
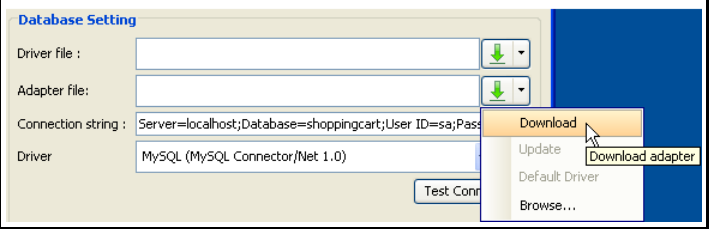
Driver file	
Adapter file	

Table 1.1

2. A **Download Database Driver** dialog box is shown allowing the proxy setting. To enable proxy for the Internet connection, check the **Use proxy** option, and then fill in the information for proxy setting.

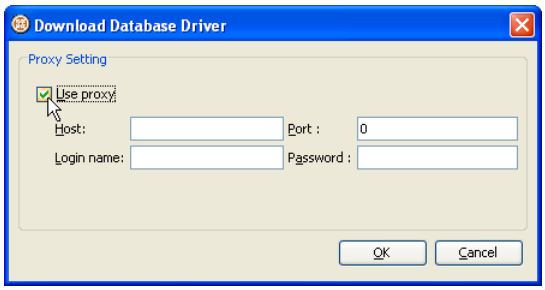


Figure 1.27 - Configure the proxy setting

The **Download** dialog box is shown indicating the download progress.

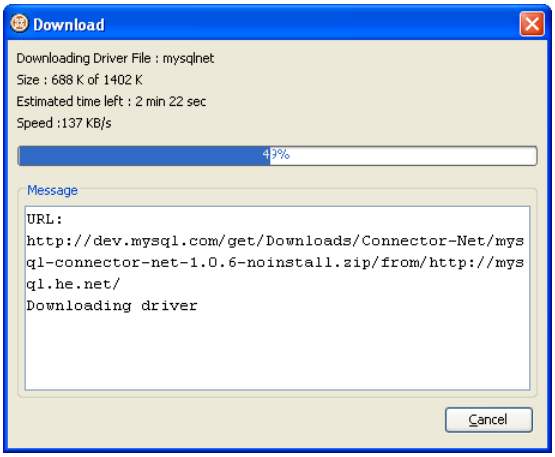


Figure 1.28 - Download dialog show the download progress

3. Click **Close** when the download is complete.

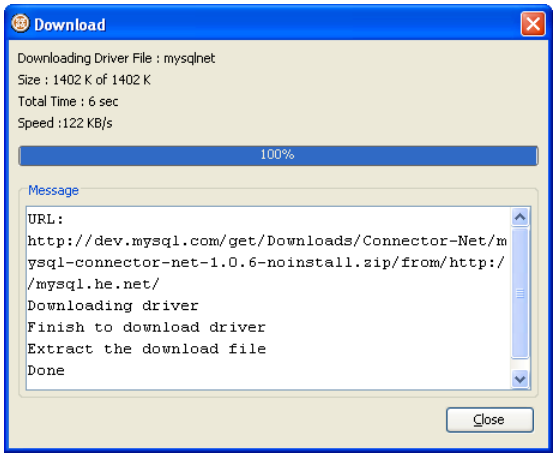


Figure 1.29 - The download complete message show in download dialog

The driver file and adapter file are shown on the **Driver file** and **Adapter file** of the **Database Setting** after download is done.

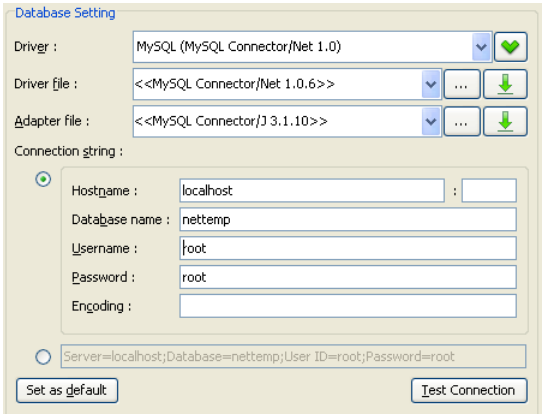


Figure 1.30 - Database Setting

Note

After downloaded the driver file, **<<MySQL Connector/NET 1.0.4>>** shown on the **Driver file** indicates that the .NET driver file is downloaded with the specified version number by DB-VA.

Note

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

Supported Database, JDBC Drivers and .NET Drivers

DB-VA provides an environment for visual modeling the developing system. By connecting the relational database to DB-VA environment, DB-VA can automate the mapping between models and relational database. DB-VA supports the most common relational database, including Oracle, DB2, Microsoft SQL Server, Sybase Adaptive Server Enterprise, Sybase SQL Anywhere, MySQL, HSQLDB, Cloudscape/Derby and PostgreSQL. Their relative JDBC Drivers and .NET Drivers are listed in the following tables.

In order to connect to any of the supported database, the relative JDBC and .NET Drivers are required for configuring the database connection. All of the required JDBC and .Net Drivers will not be bundled with DB-VA. You can get the driver files by using the automatic download facility provided, or download the driver file manually.

Table shows the Supported Database and their relative JDBC Drivers.

Database Name	JDBC Drivers Name
Cloudscape/Derby 10	Cloudscape/Derby (Embedded), Cloudscape/Derby (Server)
DB2 7/8	DB2 (AS/400 Toolbox for Java JDBC Driver) DB2 (App Driver) DB2 (Net Driver)
HSQLDB 1.61-1.8	HSQLDB (In-process) HSQLDB (Server)
IBM Informix	IBM Informix (Client) IBM Informix (Server)
MS SQL Server 2000	MS SQL Server (DataDirect SequeLink Driver) MS SQL Server (JSQL Driver) MS SQL Server (JTURBO Driver) MS SQL Server (Microsoft Driver) MS SQL Server (WebLogic Connect Driver) MS SQL Server (WebSphere Connect Driver) MS SQL Server (jTDS Driver)
MySQL 3/4	MySQL (Connector/J Driver)
Oracle	Oracle (DataDirect SequeLink Driver)
Oracle 8i	Oracle8i (THIN JDBC Driver)
Oracle 9i	Oracle9i
PostgreSQL	PostgreSQL
Sybase Adaptive Server Enterprise 12.5	Sybase Adaptive Server Enterprise (jConnect Driver) Sybase Adaptive Server Enterprise (JDS Driver)
Sybase SQL Anywhere 9	Sybase SQL Anywhere (jConnect Driver)

Table 1.2

Table shows the Supported Database and their relative .NET Drivers.

Database Name	.NET Drivers Name
DB2 7/8	DB2 (DB2 UDB for iSeries .NET Data Provider)
MS SQL Server 2000	MS SQL Server
MySQL 3/4	MySQL (MySQL Connector/Net 1.0)
Oracle	Oracle (.NET Framework Data Provider) Oracle (Oracle Data Provider for .NET)
PostgreSQL	Postgre (Npgsql)

Table 1.3

Supporting Multiple Database

As DB-VA supports multiple databases and allows you to configure multiple database settings, there may be differences in the data type supported by these databases.

Assigning Data Types from Multiple Database

DB-VA allows you to specify the data type of the column in the database by using the drop-down menu of **Type** in the **Column Specification** dialog box. By default, DB-VA provides a list of pre-defined data types which is general to all databases.

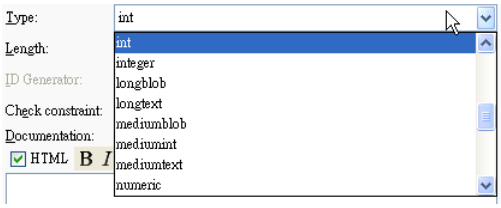


Figure 1.31 - The pre-defined data types for all database

DB-VA also allows you to assign a data type which is database-specific.

1. Place a check mark beside the desired database in the **Database Configuration** dialog box.

Figure 1.32 - Set the database type

The database-specific data types will be automatically added to the list.

2. Select the database-specific data types from the drop-down menu. For example, data type, tinyint is specific to MySQL database.

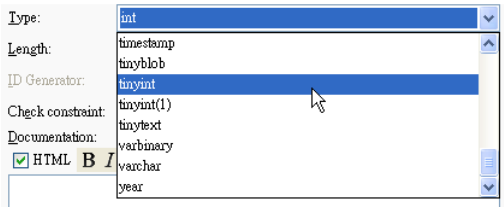


Figure 1.33 - The database-specific data types

Note

If you have checked multiple databases in the **Database Configuration** dialog box, all data types supported by these databases will be added as an option for the drop-down menu.

Displaying Data Type based on Default Database

As DB-VA provides a visual data modeling of the database depicted by the Entity Relationship Diagram (ERD), you are allowed to enable and disable the display of data type for columns of the entities in the ERD. Since the DB-VA environment is configured with a default database, the data type will be displayed according to the data type supported by the default database.

To display the data type for columns of entities in the ERD:

1. Right-click on the background of the ERD, select **Show Column Types**.

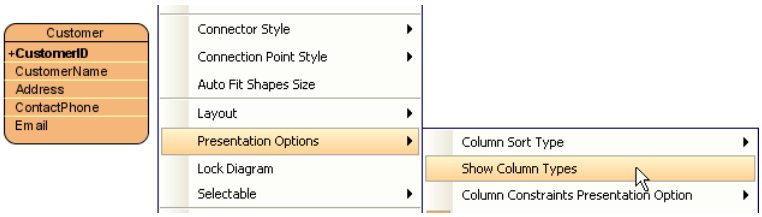


Figure 1.34 - To show column types

The data type for columns is displayed.

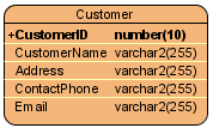


Figure 1.35 - entity with column types

Note

If the default database connection is changed, the data types for all columns will be changed with respect to the new database connection automatically.

Example:

There is an entity, Customer in the ERD. Modify the default database connection from MySQL to Oracle, the data types will be changed automatically.

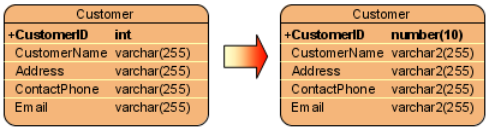


Figure 1.36 - Data type convert automatically

2 Using Wizard

Chapter 2 - Using Wizard

DB Visual ARCHITECT (DB-VA) provides a wizard to help you generate persistent code and/or database either from database, class diagram or entity relationship diagram. This chapter shows you how to use the wizard to generate code and/or database.

In this chapter:

Introduction

Mapping objects to relational database is a complicated and error pound task in the development. As DB-VA acts as a bridge between object model, data model and relational database, it automates the mappings between these models. Apart from these mappings, DB-VA also supports the mapping between object models and persistent code. Hence, the persistent code can thus map to the relational database.

DB-VA supports the synchronization between persistent code, object model, data model and relational database, it reduces the development time for handling these tedious programming jobs between them. Moreover, your document will always keep up-to-date. To support synchronization in-between persistent code and relational database, DB-VA allows you to generate database and persistent code for your development project.

DB-VA provides you a wizard for the generation of persistent code and database. The wizard provides you with three options:

1. **Generate Code from Database**.
2. **Generate Code and Database from Entity Relationship Diagram (ERD.**
3. **Generate Code and Database from Class Diagram**.

To activate the Wizard:

1. On the menu, click **Tools > Object-Relational Mapping (ORM) > Wizards...**.

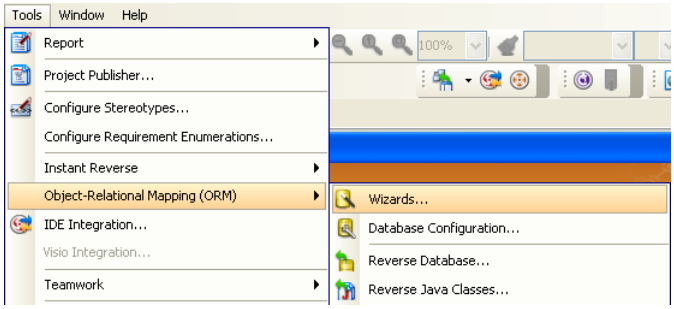


Figure 2.1 - To open the Object-relational Mapping Wizard

2. A Wizard Welcome Page will be shown, select **Language** of the code to be generated from the drop-down menu, either **Java** or **C#.**

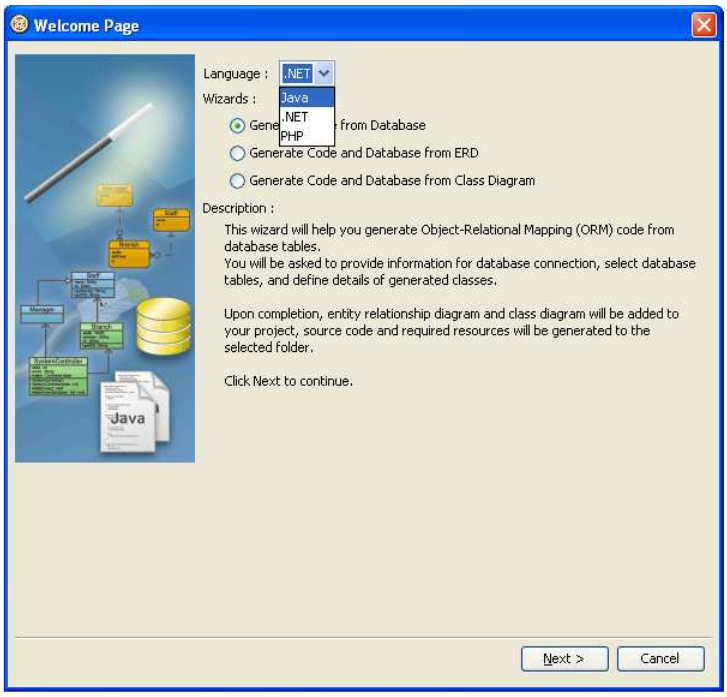


Figure 2.2 - Select the programming language

3. Select one of the wizard options, and then click **Next** to proceed.

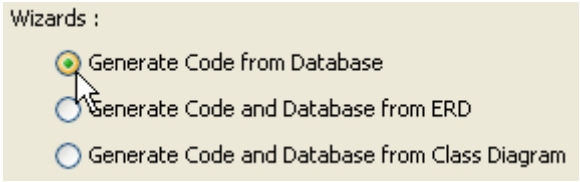


Figure 2.3 - Select the action to perform

Generating Code from Database

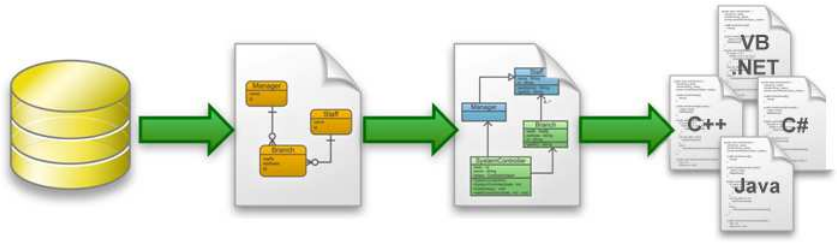


Figure 2.4 - Generate code from database

Upon selecting the option for **Generate Code from Database**, the wizard helps you generate persistent code from database tables according to the requirements you specified throughout the wizard.

Follow the steps of the **Generate Code From Database** wizard:

1. Database Configuration
   - For Java Language selected

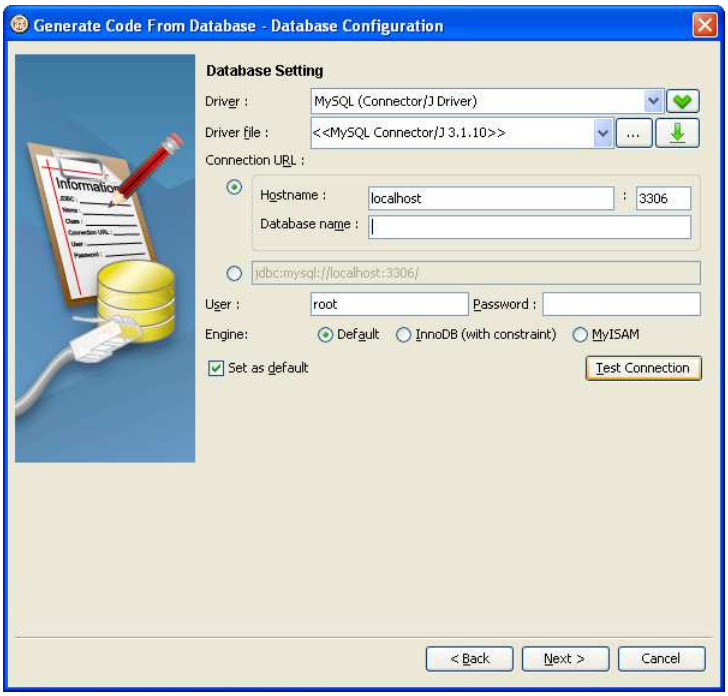


Figure 2.5 - Database Configuration for Java

2. You are asked to define the database configuration. Refer to the descriptions in the Working with DB Visual ARCHITECT chapter for information on how to configure the database in the Database Configuration for Java Project section

   - For C# Language selected

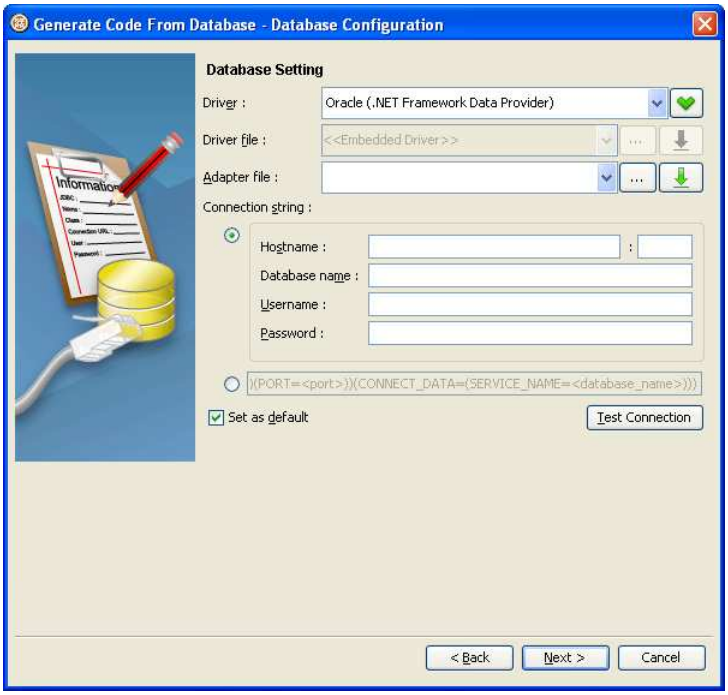


Figure 2.6 - Database Configuration for .NET

You are asked to define the database configuration. Refer to the descriptions in the Working with DB Visual ARCHITECT chapter for information on how to configure the database in the Database Configuration for .Net Project section.

3. Selecting Tables

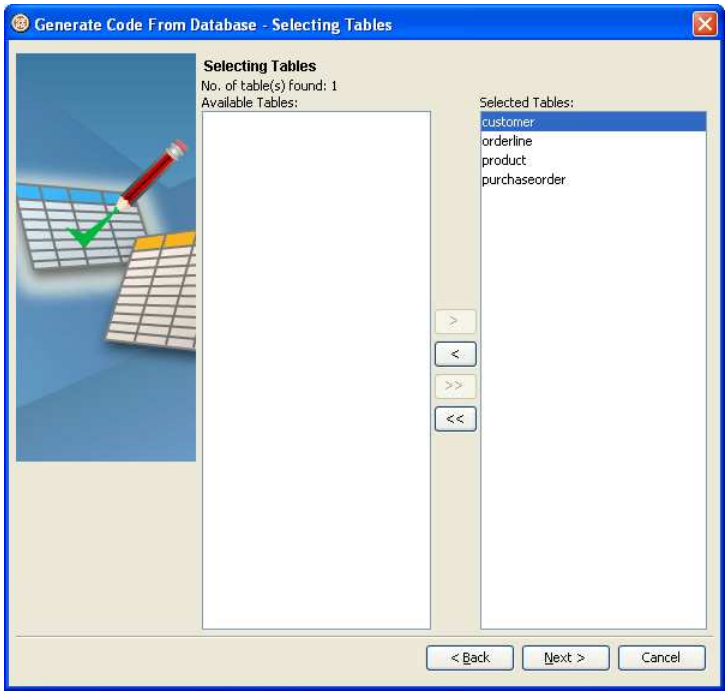


Figure 2.7 - Select the Tables

DB-VA connects your database based on your options in the previous database configuration option pane and reverses all database tables. You are asked to select the database tables which you want to generate persistent class to manipulate those tables. By default, all the database tables are selected for the generation of code shown in the list of **Selected Tables**. You can deselect the table by using the list of buttons between the list of **Available Tables** and **Selected Tables**.

-  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**.

4. Class Details Configuration

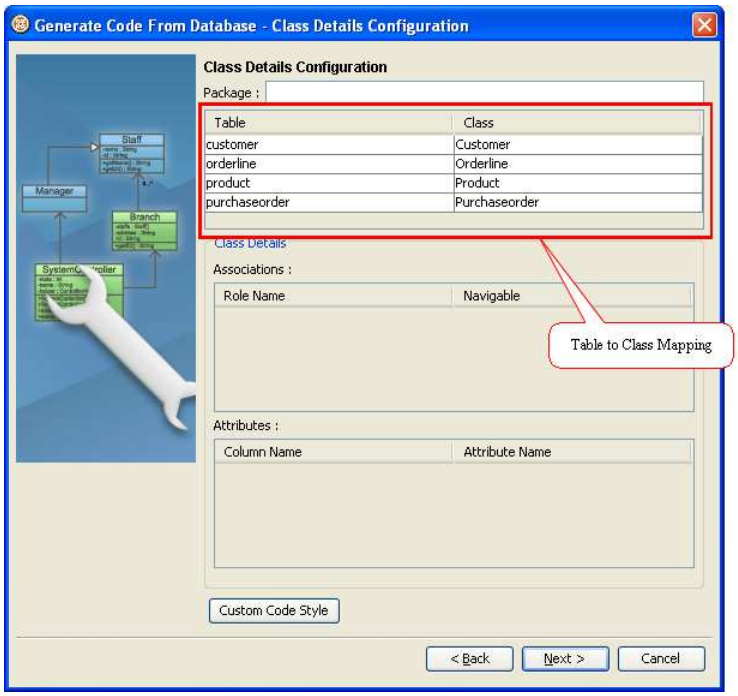


Figure 2.8 - Class mapping

After selecting tables, you will be directed to a Class Details Configuration pane. You are asked to define the Class Details for generating code. DB-VA generates the persistent classes based on the information defined here. You can edit the class details by double-clicking the field.

- **Package**

  Enter the package name. 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 2.9 - Generate classes in default package message

- **Class**

  You can edit the class name which will be used as the name of the generated persistent code for a corresponding table.

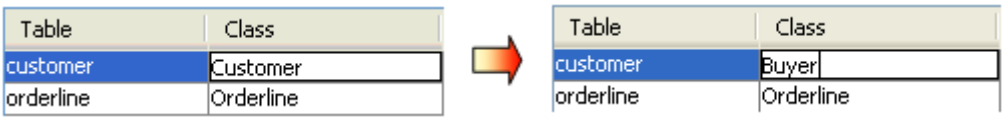


Figure 2.10 - Mapping Classes

- **Associations**

You can edit the role name for a reference in the class.

Figure 2.11 - Mapping Associations

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

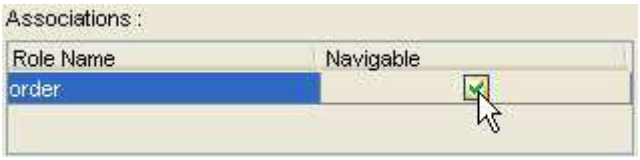


Figure 2.12 - select the Navigable

- **Attributes**

You can edit the attribute name representing the column of the table.

Figure 2.13 - Mapping Attributes

- **Custom Code Style**

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

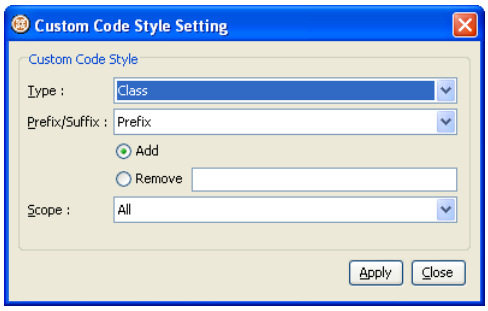


Figure 2.14 - Custom Code Style setting dialog

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

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

Table shows the result of applying 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 2.1

5. Generate Code
    - For Java Language selected

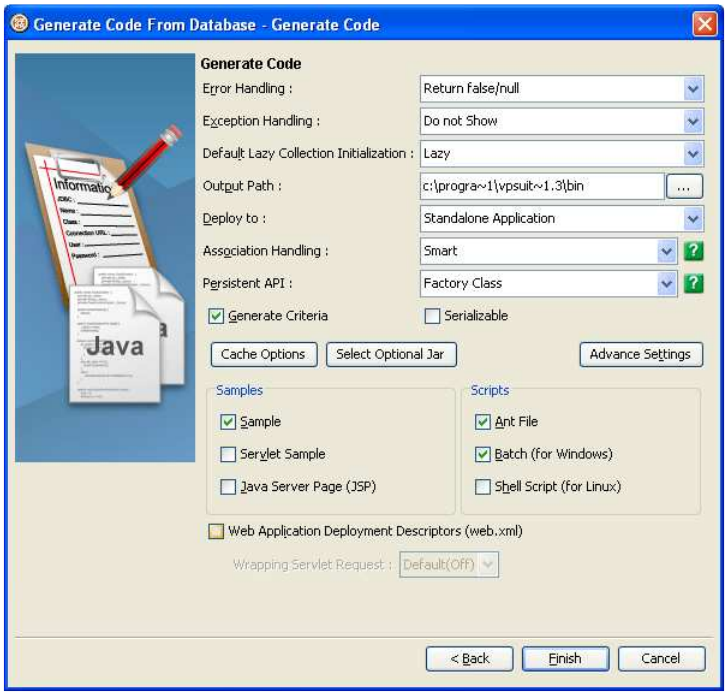


Figure 2.15 - Generate Code options for Java

You are asked to specify the code details. DB-VA generates the Java persistent code based on the information defined here.

For **Error Handling**, select the way to handle errors when they occur.

- **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.
- **Throw RuntimeException** - It throws a RuntimeException which will be handled by the caller.

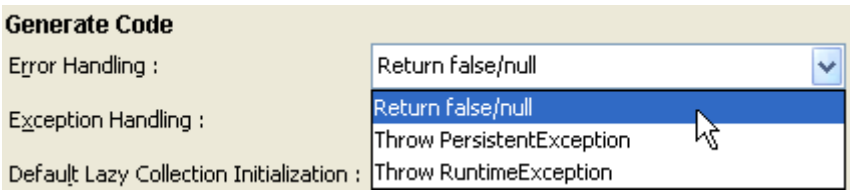


Figure 2.16 - Error handling options

For **Exception Handling**, select how to handle the exception, either Do not Show, Print to Error Stream or Print to log4j.

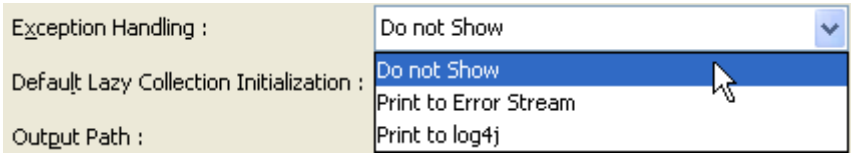


Figure 2.17 - Exception Handling options

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.

For **Output Path**, specify the location to store the generated persistent code source file.

For **Deploy to**, select the type of application that you want to deploy to.

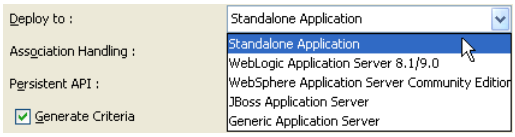


Figure 2.18 - Deployment options

For **Association Handling**, select the type of association handling to be sued, 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.

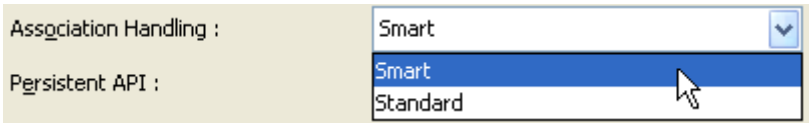


Figure 2.19 - Association Handling

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

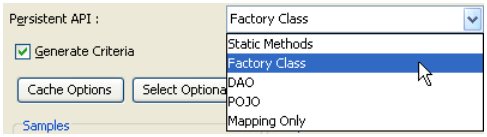


Figure 2.20 - Persistent API options

For **Generate Criteria**, check the option for Generate Criteria to generate the criteria class for each ORM-Persistable class. The criteria class supports querying the database by specifying the searching criteria.

For **Select Optional Jar**, select the libraries and JDBC driver to be included in the generation of the **orm.jar** file. Click on the **Select Optional Jar** button, the **Select Optional Jar** dialog box is displayed. Check the desired libraries to be included in the **orm.jar** which will be built at runtime.

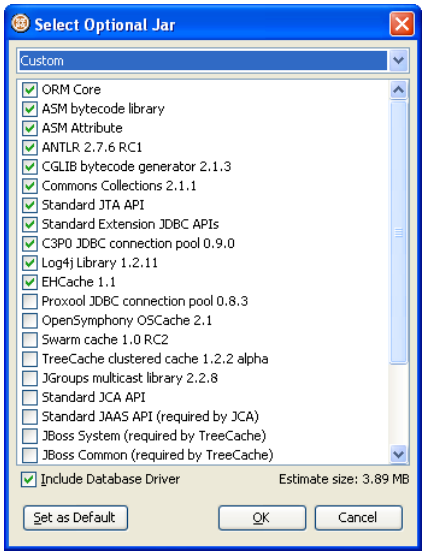


Figure 2.21 - Select Optional Jar dialog

For **Samples**, sample files, including Java application, servlet, servlet filter 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.

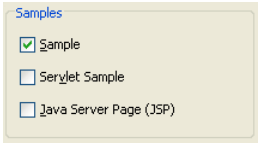


Figure 2.22 - Generate samples options

For **Scripts**, you can check the options to generate the scripts, including Ant File, Batch and Shell Script which allow you to execute the scripts directly.

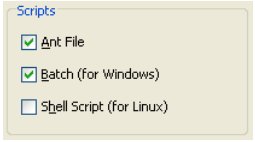


Figure 2.23 - Generate script options

For **Advance Settings**, you can define the Default Order Collection Type, Default Un-Order Collection Type, Override toString Method and Flush Mode. Click on the **Advance Settings** button, select the desired settings in the **Advance Settings** dialog box.

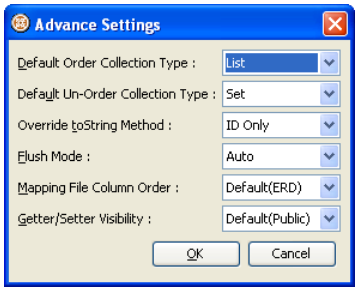


Figure 2.24 - Advance Setting dialog

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

For **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.

For **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.

For **Flush Mode**, select the flush mode, either Auto, Commit, Always or Never.

- For C# Language selected

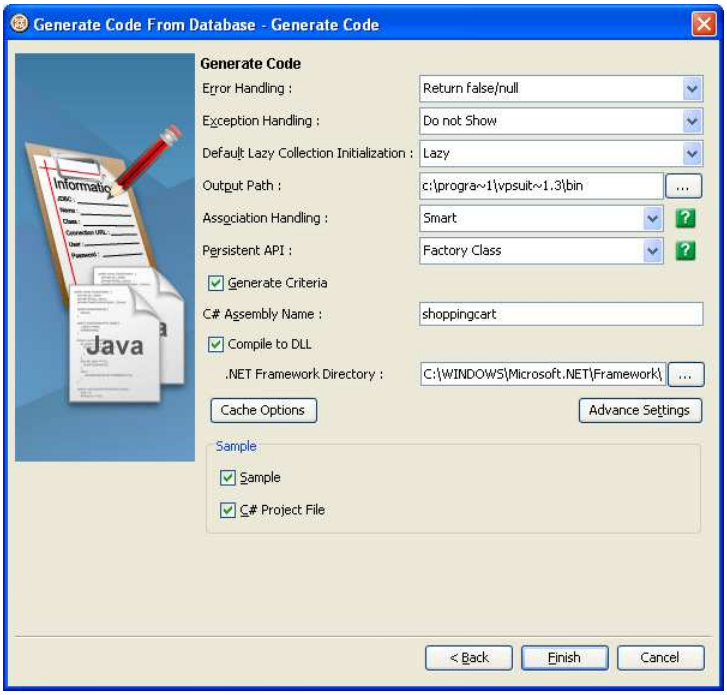


Figure 2.25 - Generate Code options for C#

You are asked to specify the code details. DB-VA generates the .NET persistent code based on the information defined here.

For **Error Handling**, select the way to handle errors when they occur.

- **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.

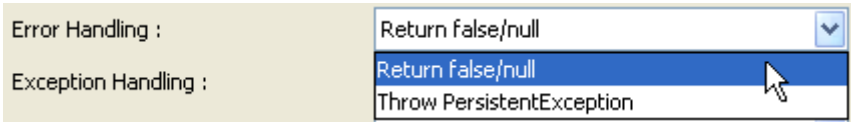


Figure 2.26 - Error Handling options

For **Exception Handling**, select how to handle the exception, either Do not Show, Print to Error Stream or Print to log4net.

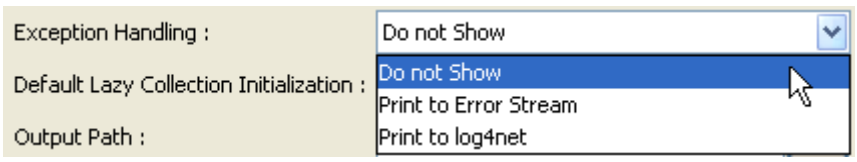


Figure 2.27 - Exception Handling options

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.

For **Output Path**, specify the location to store the generated persistent code source file.

For **Association Handling**, select the type of association handling to be sued, 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.

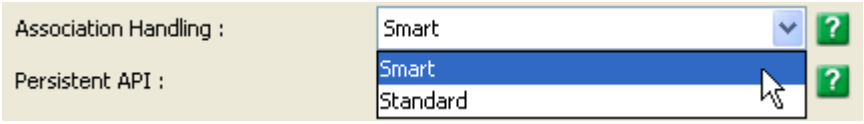


Figure 2.28 - Association Handling options

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

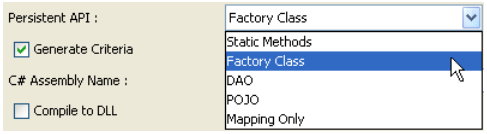


Figure 2.29 - Persistent API options

For **Generate Criteria**, check the option for Generate Criteria to generate the criteria class for each ORM-Persistable class. The criteria class supports querying the database by specifying the searching criteria.

For **C# Assembly Name**, specify the name of the assembly for the .NET application which holds the assembly metadata.

For **Compile to DLL**, check the option for **Compile to DLL**, DB-VA will generate DLL files which can be referenced by .NET projects of language other than C# source.

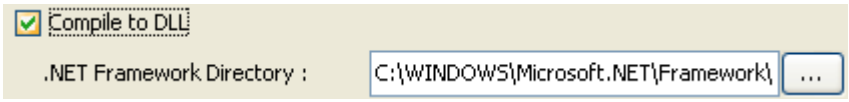


Figure 2.30 - Compile to DLL options

For **Sample**, check the options to generate the sample files and C# project file which guide you through the usage of the .NET persistent classes.

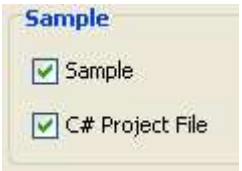


Figure 2.31 - Generate Sample options

For **Advance Settings**, you can define the Default Order Collection Type, Default Un-Order Collection Type and Override toString Method. Click on the **Advance Settings** button, select the desired settings in the **Advance Settings** dialog box.

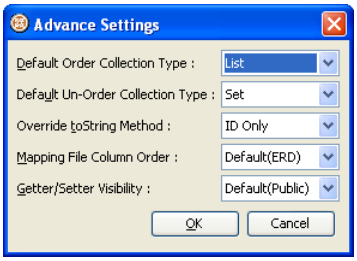


Figure 2.32 - Advance Setting options

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

For **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.

For **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.

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

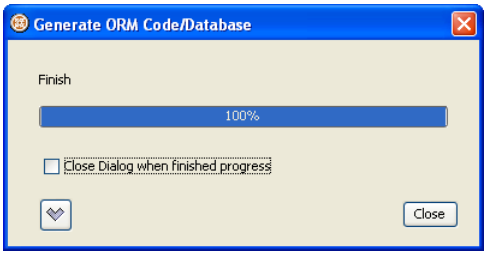


Figure 2.33 - Generate ORM Code/Database dialog

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.

Generating Code and Database from ERD

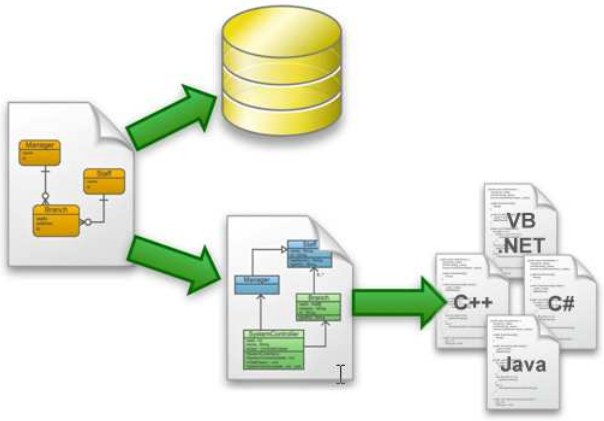


Figure 2.34 - Generating Code and Database from ERD

Upon selecting the option for **Generate Code and Database from ERD**, the wizard helps you generate persistent code and database from ERD with respect to the requirements you specified throughout the wizard.

Take the following ERD as an example:

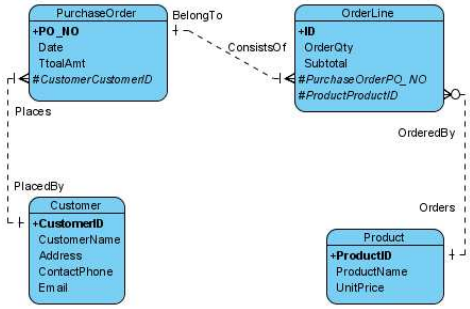


Figure 2.35 - Entities Relationship Diagram

Follow the steps of the **Generate Code and Database From ERD** wizard:

1. Class Details Configuration

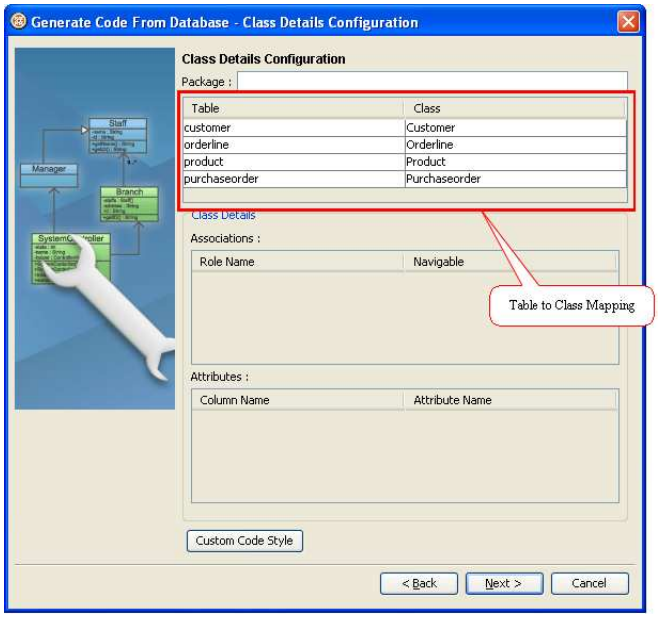


Figure 2.36 - Mapping Tables to Classes

You are asked to define the Class Details for generating code. DB-VA generates persistent classes based on the information defined here. You can edit the class details by double-clicking the field.

- **Package**

  Enter the package name. 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.

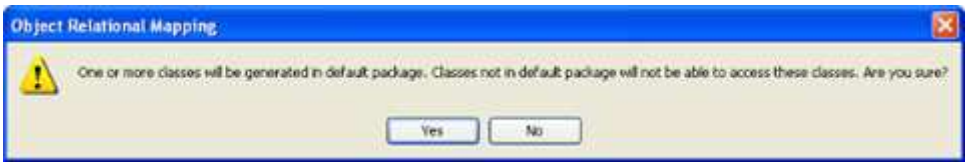


Figure 2.37 - Confirm generate code as default package message

- **Class**

  You can edit the class name which will be used as the name of the generated persistent code for a corresponding table.

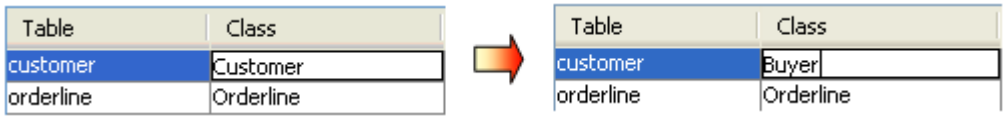


Figure 2.38 - Mapping classes

- **Associations**

  You can edit the role name for a reference in the class.

Figure 2.39 - Mapping associations

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

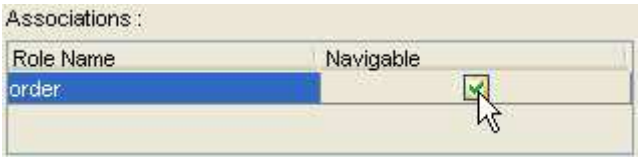


Figure 2.40 - Select Navigable of association

- **Attributes**

  You can edit the attribute name representing the column of the table.

Figure 2.41 - Mapping attributes

- **Custom Code Style**

Click 

button, **Custom Code Style Setting** dialog box will be displayed. You can modify the prefix or suffix of the **Class**, **Attribute** and **Role Name**.

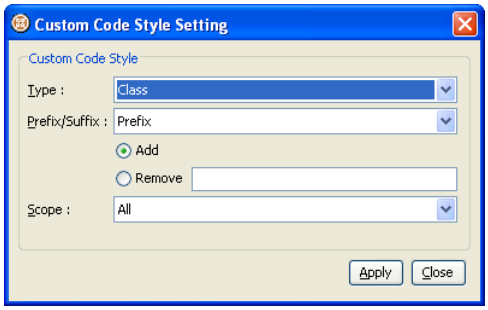


Figure 2.42 - Custom Code Style Setting dialog

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

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

Table shows the result of applying 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 2.2

2. Database Configuration
   - For Java Language selected

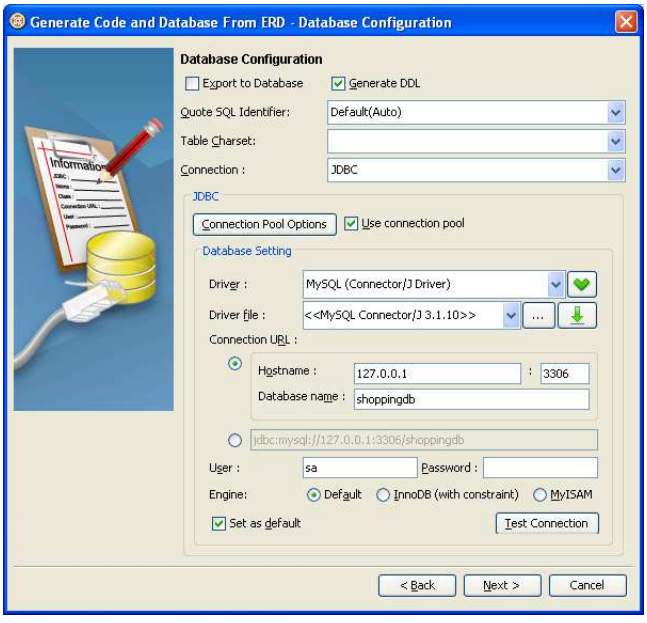


Figure 2.43 - Database Configuration for Java

You are asked to define the database configuration. DB-VA generates the database based on the settings defined here.

For **Export to Database**, check the option to allow altering the database.

For **Generate DDL**, check the option to allow the generation of DDL file.
For **Quote SQL Identifier**, select the option from the drop-down menu to enable using the reserved word. By enabling **Quote SQL Identifier**, the reserved word used in database can be used as ordinary word in generating database.

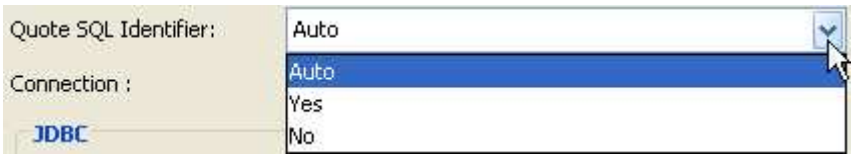


Figure 2.44 - Quote SQL Identifier options

For **Connection**, select the type of connection from the drop-down menu, either **JDBC** or **Datasource**.

Figure 2.45 - Connection options

For **Use connection pool**, check the option to enable using the connection pool for Java project.

Figure 2.46 - Use connection pool option

For **Connection Pool Options**, click the **Connection Pool Options** button to open the **Connection Pool Options** dialog box for configuring the connection pool settings.

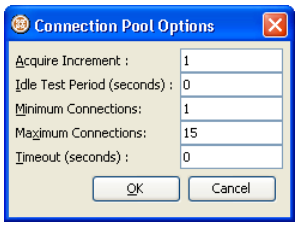


Figure 2.47 - Connection Pool Options dialog

For **Database Setting**, refer to the description on Database Configuration for Java Project section in the Working with DB Visual ARCHITECT chapter for information on how to configure the database settings.

- For C# Language selected

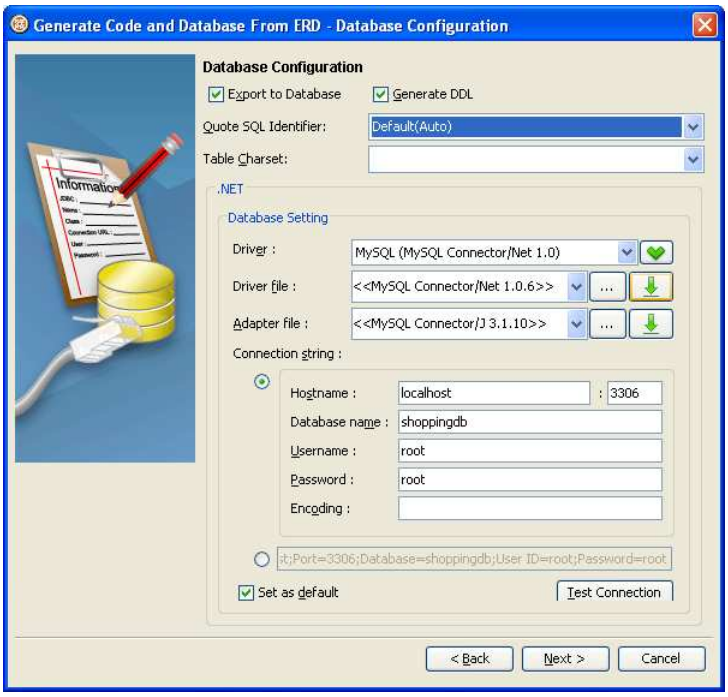


Figure 2.48 - Database Configuration for C#

You are asked to define the database configuration. DB-VA generates the database based on the settings defined here.

For **Export to Database**, check the option to allow altering the database.
For **Generate DDL**, check the option to allow the generation of DDL file.
For **Quote SQL Identifier**, select the option from the drop-down menu to enable using the reserved word. By enabling **Quote SQL Identifier**, the reserved word used in database can be used as ordinary word in generating database.

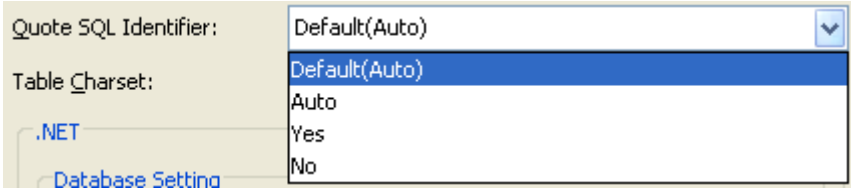


Figure 2.49 - Quote SQL Identifier options

For **Database Setting**, refer to the descriptions on Database Configuration for .Net Project section in the Working with DB Visual ARCHITECT chapter for information on how to configure the database settings.

3. Generate Code
    - For Java Language selected

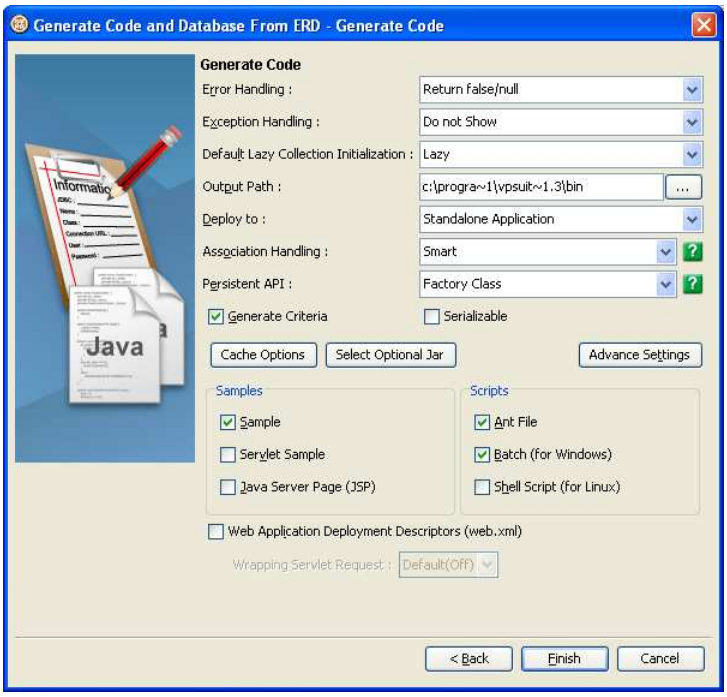


Figure 2.50 - Generate Code setting for Java

You are asked to specify the code details. DB-VA generates the Java persistent code based on the information defined here.

For **Error Handling**, select the way to handle errors when they occur.

- **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.
- **Throw RuntimeException** - It throws a RuntimeException which will be handled by the caller.

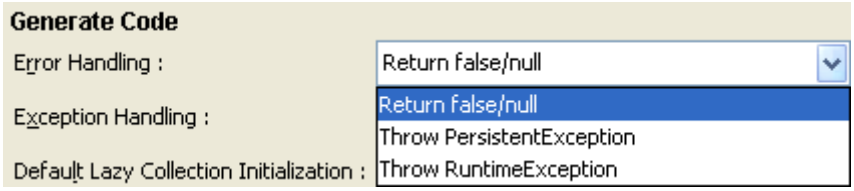


Figure 2.51 - Error Handling options

For **Exception Handling**, select how to handle the exception, either Do not Show, Print to Error Stream or Print to log4j.

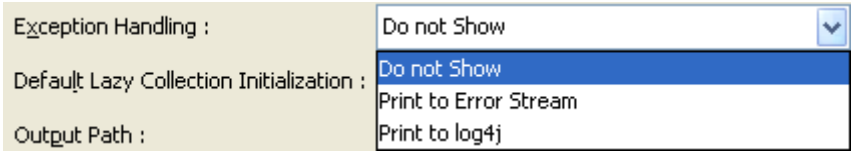


Figure 2.52 - Exception Handling options

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.

For **Output Path**, specify the location to store the generated persistent code source file.

For **Deploy to**, select the type of application that you want to deploy to.

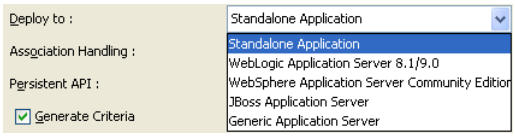


Figure 2.53 - Deployment options

4. For **Association Handling**, select the type of association handling to be sued, 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.

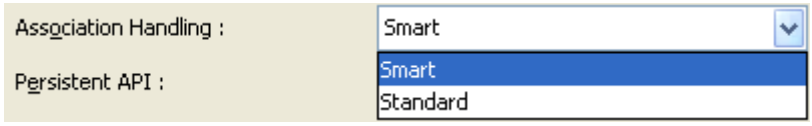


Figure 2.54 - Association Handling options

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

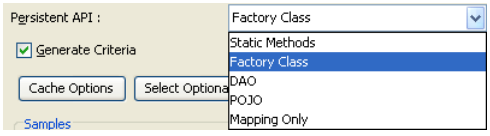


Figure 2.55 - Persistent API options

For **Generate Criteria**, check the option for Generate Criteria to generate the criteria class for each ORM-Persistable class. The criteria class supports querying the database by specifying the searching criteria.

For **Select Optional Jar**, select the libraries and JDBC driver to be included in the generation of the **orm.jar** file. Click on the **Select Optional Jar** button, the **Select Optional Jar** dialog box is displayed. Check the desired libraries to be included in the **orm.jar** which will be built at runtime.

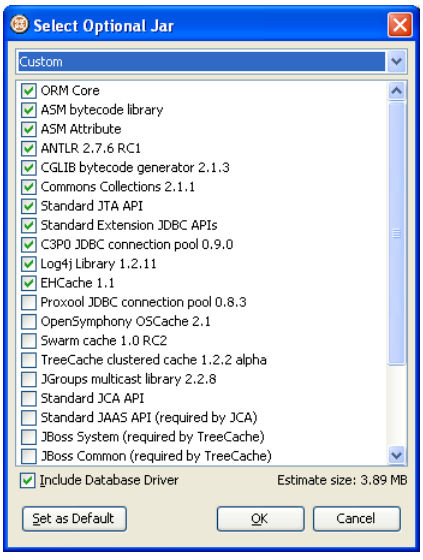


Figure 2.56 - Select Optional Jar options

For **Samples**, sample files, including Java application, servlet, servlet filter 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.

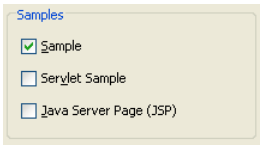


Figure 2.57 - Generate samples options

For **Scripts**, you can check the options to generate the scripts, including Ant File, Batch and Shell Script which allow you to execute the scripts directly.

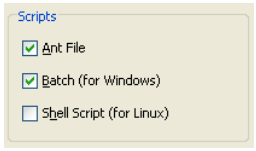


Figure 2.58 - Generate scripts options

For **Advance Settings**, you can define the Default Order Collection Type, Default Un-Order Collection Type, Override toString Method and Flush Mode. Click on the **Advance Settings** button, select the desired settings in the **Advance Settings** dialog box.

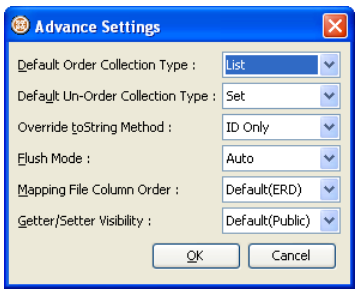


Figure 2.59 - Advance Settings dialog

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

For **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.

For **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.

For **Flush Mode**, select the flush mode, either Auto, Commit, Always or Never.

- For C# Language selected

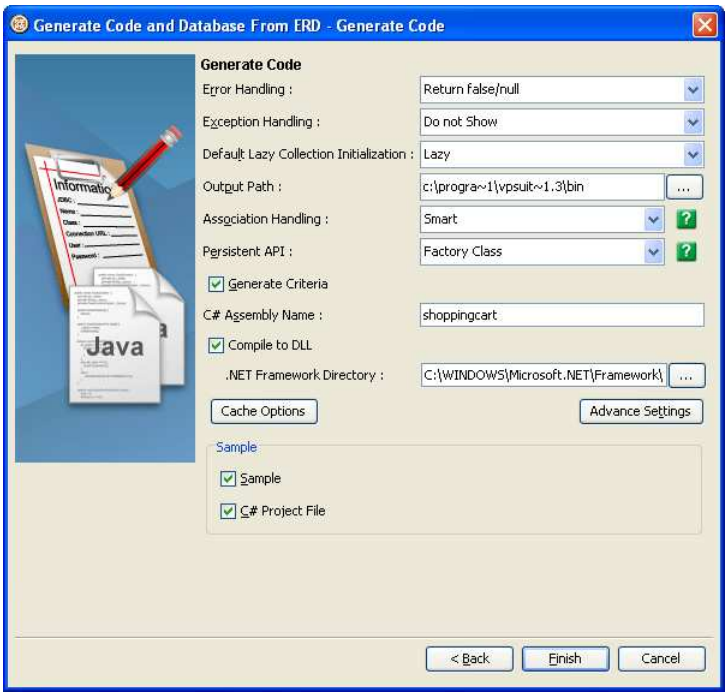


Figure 2.60 - Generate Code setting for C#

You are asked to specify the code details. DB-VA generates the .NET persistent code based on the information defined here.

For **Error Handling**, select the way to handle errors when they occur.

- **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.

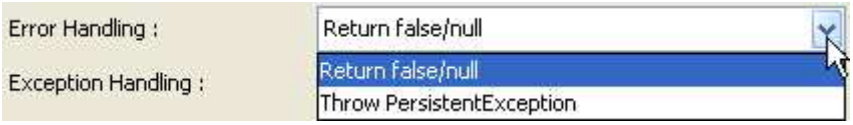


Figure 2.61 - Error Handling options

For **Exception Handling**, select how to handle the exception, either Do not Show, Print to Error Stream or Print to log4net.

Figure 2.62 - 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.

For **Output Path**, specify the location to store the generated persistent code source file.

For **Association Handling**, select the type of association handling to be sued, 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 2.63 - Association Handling options

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

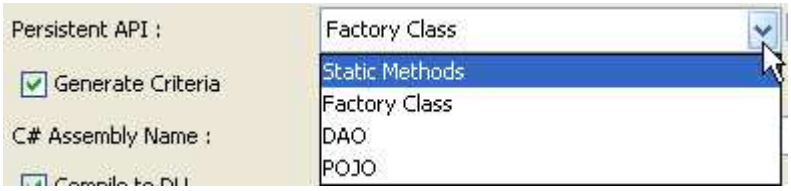


Figure 2.64 - Persistent API options

For **Generate Criteria**, check the option for Generate Criteria to generate the criteria class for each ORM-Persistable class. The criteria class supports querying the database by specifying the searching criteria.

For **C# Assembly Name**, specify the name of the assembly for the .NET application which holds the assembly metadata.

For **Compile to DLL**, check the option for **Compile to DLL**, DB-VA will generate DLL files which can be referenced by .NET projects of language other than C# source.

Figure 2.65 - Compile to DLL options

For **Sample**, check the options to generate the sample files and C# project file which guide you through the usage of the .NET persistent classes.

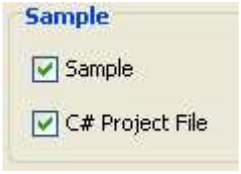


Figure 2.66 - Generate Code sample options

For **Advance Settings**, you can define the Default Order Collection Type, Default Un-Order Collection Type and Override toString Method. Click on the **Advance Settings** button, select the desired settings in the **Advance Settings** dialog box.

Figure 2.67 - Advance Setting dialog

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

For **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.

For **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.

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

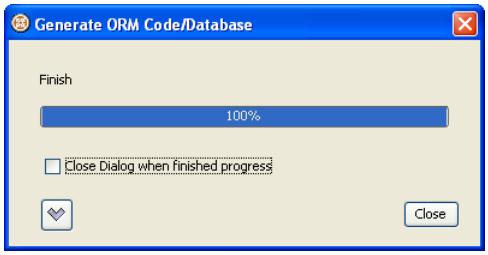


Figure 2.68 - Generate ORM Code/Database dialog

A class 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 and the generated database will be set up to the specified database configuration.

Generating Code and Database from Class Diagram

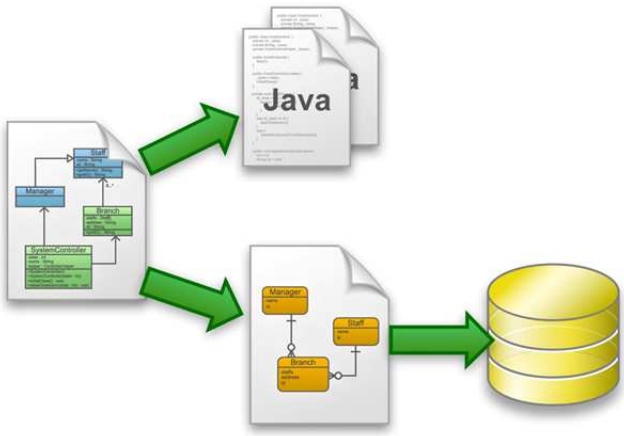


Figure 2.69 - Generate Code from Class Diagram

Upon selecting the option for **Generate Code from Class Diagram**, the wizard helps you generate persistent code and database from class diagram with respect to the requirements you specified throughout the wizard.

Take the following class diagram as an example:

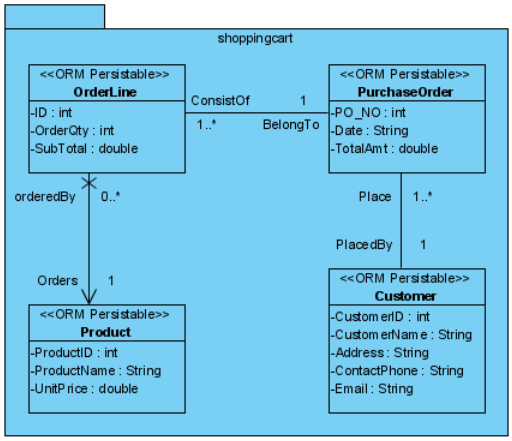


Figure 2.70 - Class Diagram

Follow the steps of the **Generate Code and Database from Class Diagram** wizard:

1. Selecting Classes

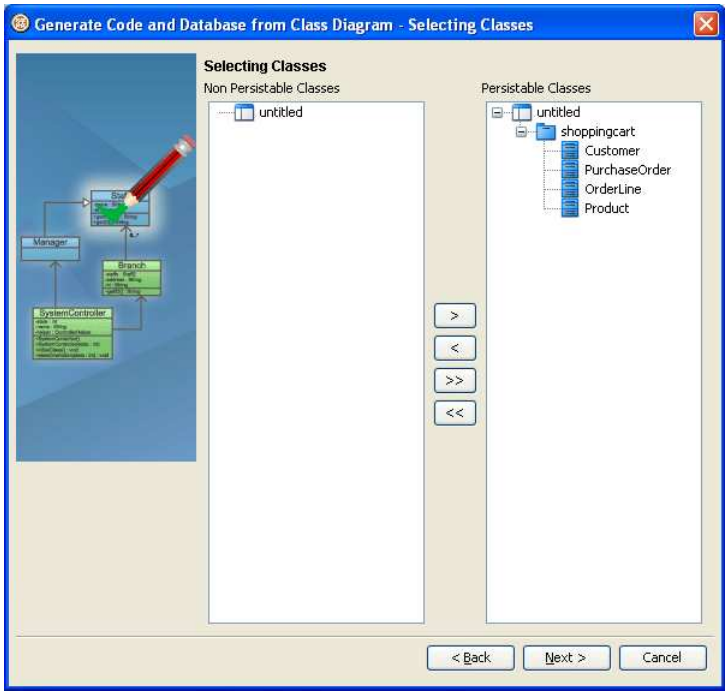


Figure 2.71 - Select the Classes

You are asked to select the classes on the class diagram which you want to generate persistent class to manipulate persistent data. By default, all the classes stereotyped as ORM-Persistable on the class diagram are selected for the generation of code and database shown in the list of **Persistable Classes**. You can deselect the persistable classes by using the list of buttons between the list of **Non Persistable Classes** and **Persistable Classes**.

-  Add Selected

  Add the selected class from **Non Persistable Classes** to **Persistable Classes**.

-  Remove Selected

  Remove the selected class from **Persistable Classes** to **Non Persistable Classes**.

- >> Add All

  Add all classes from **Non Persistable Classes** to **Persistable Classes**.

- << Remove All

  Remove all classes from **Persistable Classes** to **Non Persistable Classes**.

For the classes shown in the list of **Persistable Classes**, they will be stereotyped as ORM-Persistable on the class diagram after the wizard is finished. Meanwhile, for the classes shown in the list of **Non Persistable Classes**, they will not be stereotyped on the class diagram after the wizard is finished.

2. Select Primary Key

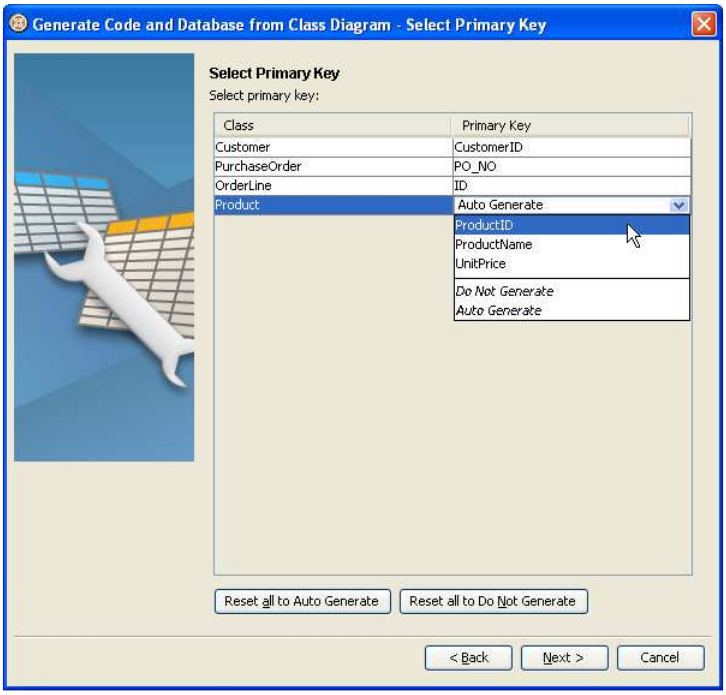


Figure 2.72 - Select the Primary Key for the Table

You are asked to select the primary key for each class being mapped to data model and relational database. You can either select an attribute as the primary key or let DB-VA generate the primary key automatically. Please change the select by using the drop-down menu.

3. Table Details Configuration

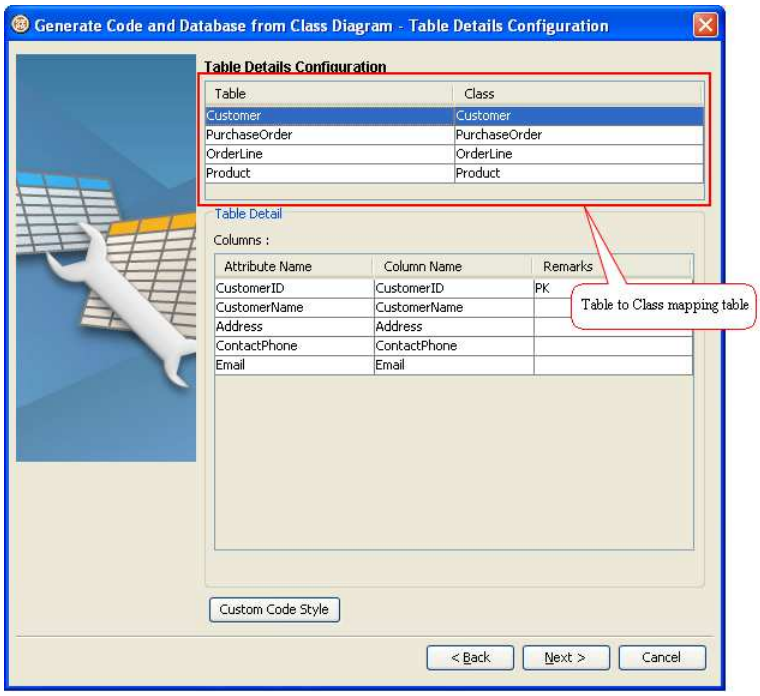


Figure 2.73 - Mapping Tables to Classes

You are asked to define the Table Details for generating database and code. DB-VA generates database and persistent classes based on the information defined here. You can edit the table details by double-clicking the field.

- **Package**

  Enter the package name. 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 2.74 - Confirm generate classes in default package message

- **Table**

  You can edit the table name which will be used as the name of the generated database table.

- **Columns**

  You can edit the column name represented by the class.

Figure 2.75 - Mapping column

- **Custom Code Style**

Click 

button, **Custom Code Style Setting** dialog box will be displayed. You can modify the prefix or suffix of the **Table**, **Column**, **Primary Key** and **Foreign Key**.

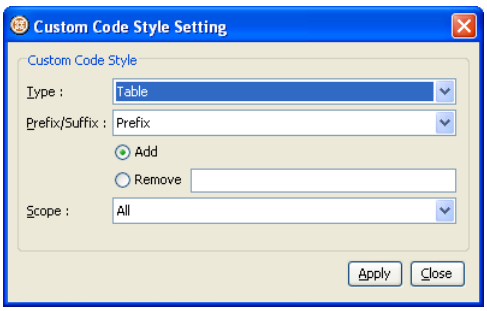


Figure 2.76 - Custom Code Style Setting dialog

For the **Type**, select the type of Table detail, either Table, Column, Primary Key or Foreign Key that you want to apply code style.

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

Table shows the result of applying 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 2.3

4. Database Configuration
   - For Java Language selected

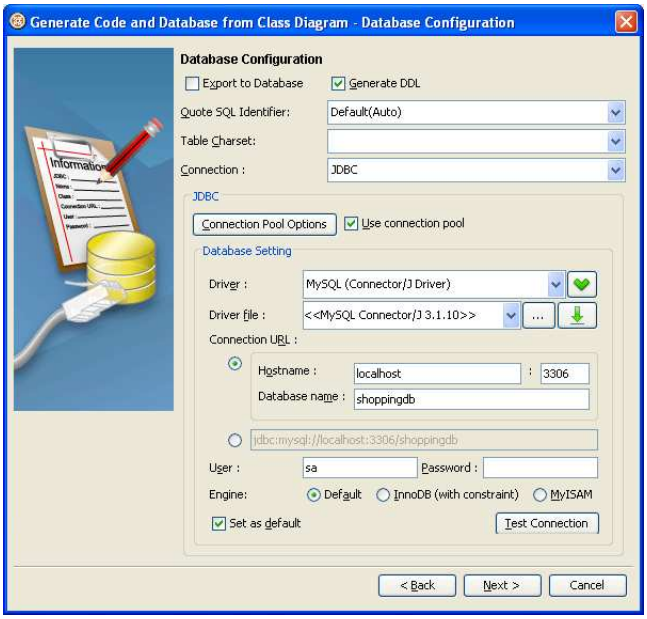


Figure 2.77 - Database Configuration for Java

You are asked to define the database configuration. DB-VA generates the database based on the settings defined here.

For **Export to Database**, check the option to allow altering the database.
For **Generate DDL**, check the option to allow the generation of DDL file.
For **Quote SQL Identifier**, select the option from the drop-down menu to enable using the reserved word. By enabling **Quote SQL Identifier**, the reserved word used in database can be used as ordinary word in generating database.

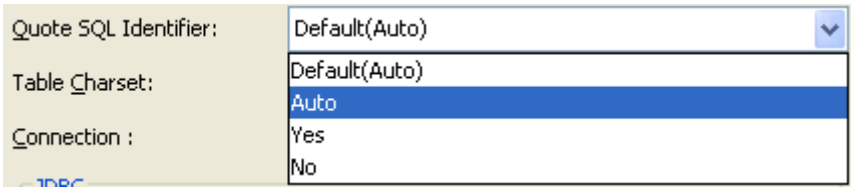


Figure 2.78 - Quote SQL Identifier

For **Connection**, select the type of connection from the drop-down menu, either **JDBC** or **Datasource**.

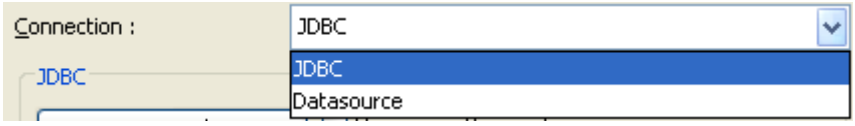


Figure 2.79 - Connection options

For **Use connection pool**, check the option to enable using the connection pool for Java project.

Figure 2.80 - Use connection pool options

For **Connection Pool Options**, click the **Connection Pool Options** button to open the **Connection Pool Options** dialog box for configuring the connection pool settings.

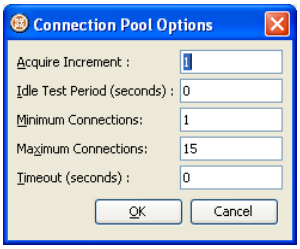


Figure 2.81 - Connection Pool Options dialog

For **Database Setting**, refer to the description on Database Configuration for Java Project section in the Working with DB Visual ARCHITECT chapter for information on how to configure the database settings.

- For C# Language selected

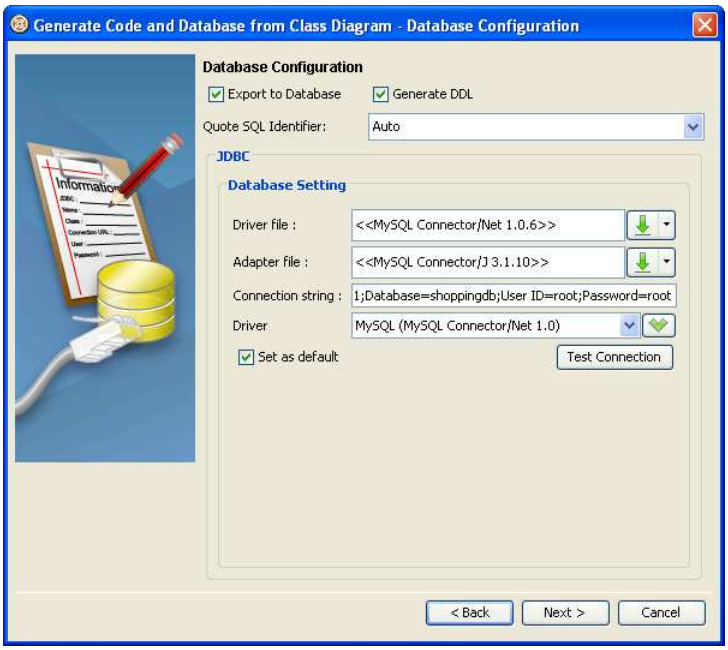


Figure 2.82 - Database Configuration for C#

You are asked to define the database configuration. DB-VA generates the database based on the settings defined here.

For **Export to Database**, check the option to allow altering the database.

For **Generate DDL**, check the option to allow the generation of DDL file.
For **Quote SQL Identifier**, select the option from the drop-down menu to enable using the reserved word. By enabling **Quote SQL Identifier**, the reserved word used in database can be used as ordinary word in generating database.

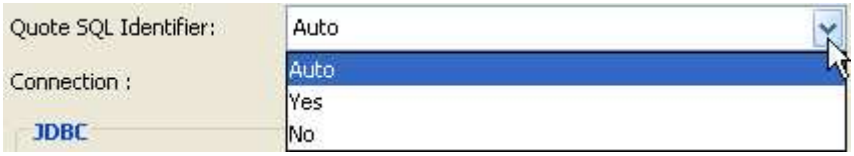


Figure 2.83 - Quote SQL Identifier options

For **Database Setting**, refer to the descriptions on Database Configuration for .Net Project section in the Working with DB Visual ARCHITECT chapter for information on how to configure the database settings.

5. Generate Code
    - For Java Language selected

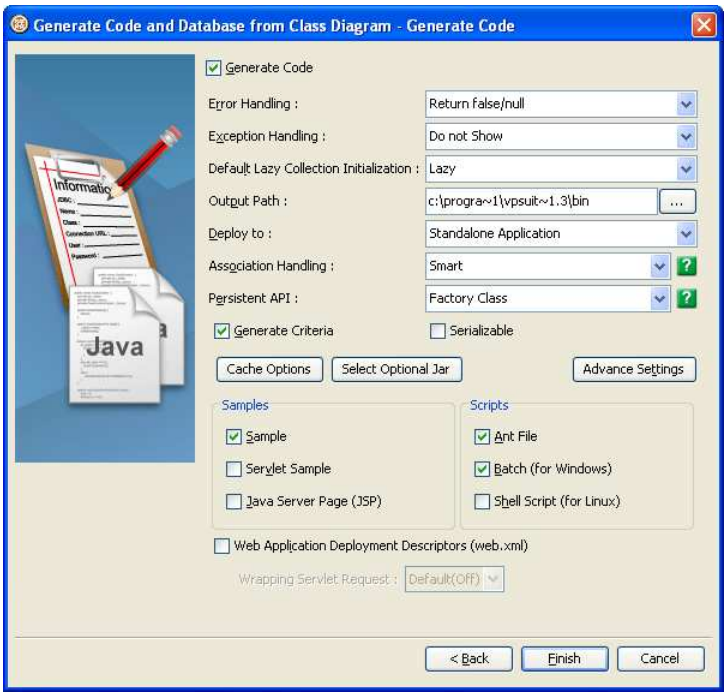


Figure 2.84 - Generate Code setting for Java

You are asked to specify the code details. DB-VA generates the Java persistent code based on the information defined here.

For **Generate Code**, check the option to allow the generation of source code.

For **Error Handling**, select the way to handle errors when they occur.

- **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.
- **Throw RuntimeException** - It throws a RuntimeException which will be handled by the caller.

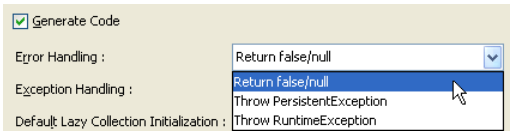


Figure 2.85 - Error Handling options

For **Exception Handling**, select how to handle the exception, either Do not Show, Print to Error Stream or Print to log4j.

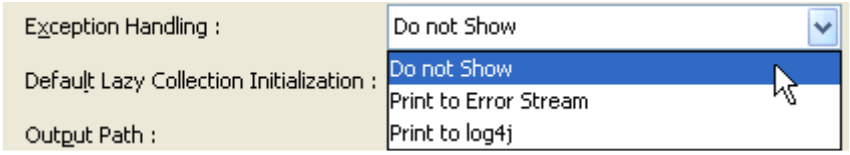


Figure 2.86 - Exception Handling options

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.

For **Output Path**, specify the location to store the generated persistent code source file.

For **Deploy to**, select the type of application that you want to deploy to.

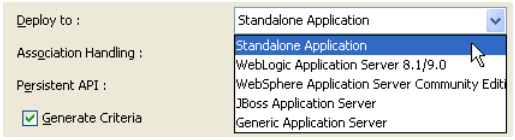


Figure 2.87 - Deployment options

For **Association Handling**, select the type of association handling to be sued, 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.

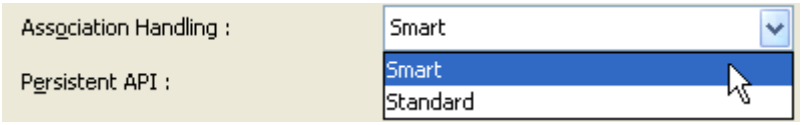


Figure 2.88 - Association Handling options

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

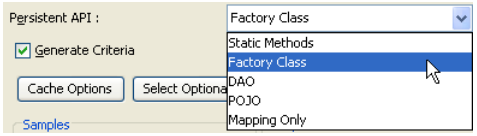


Figure 2.89 - Persistent API options

For **Generate Criteria**, check the option for Generate Criteria to generate the criteria class for each ORM-Persistable class. The criteria class supports querying the database by specifying the searching criteria.

For **Select Optional Jar**, select the libraries and JDBC driver to be included in the generation of the **orm.jar** file. Click on the **Select Optional Jar** button, the **Select Optional Jar** dialog box is displayed. Check the desired libraries to be included in the **orm.jar** which will be built at runtime.

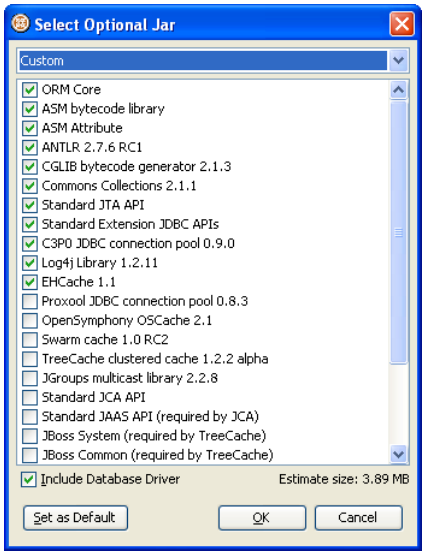


Figure 2.90 - Select Optional Jar dialog

For **Samples**, sample files, including Java application, servlet, servlet filter 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.

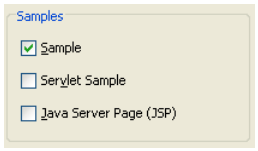


Figure 2.91 - Generate Samples options

For **Scripts**, you can check the options to generate the scripts, including Ant File, Batch and Shell Script which allow you to execute the scripts directly.

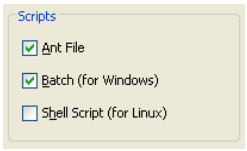


Figure 2.92 - Generate scripts options

For **Advance Settings**, you can define the Default Order Collection Type, Default Un-Order Collection Type, Override toString Method and Flush Mode. Click on the **Advance Settings** button, select the desired settings in the **Advance Settings** dialog box.

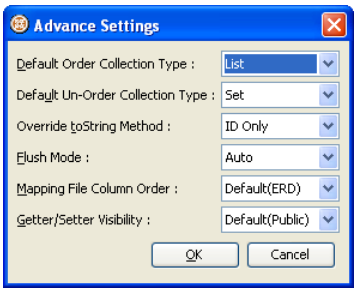


Figure 2.93 - Advance Setting dialog

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

For **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.

For **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.

For **Flush Mode**, select the flush mode, either Auto, Commit, Always or Never.

- For C# Language selected

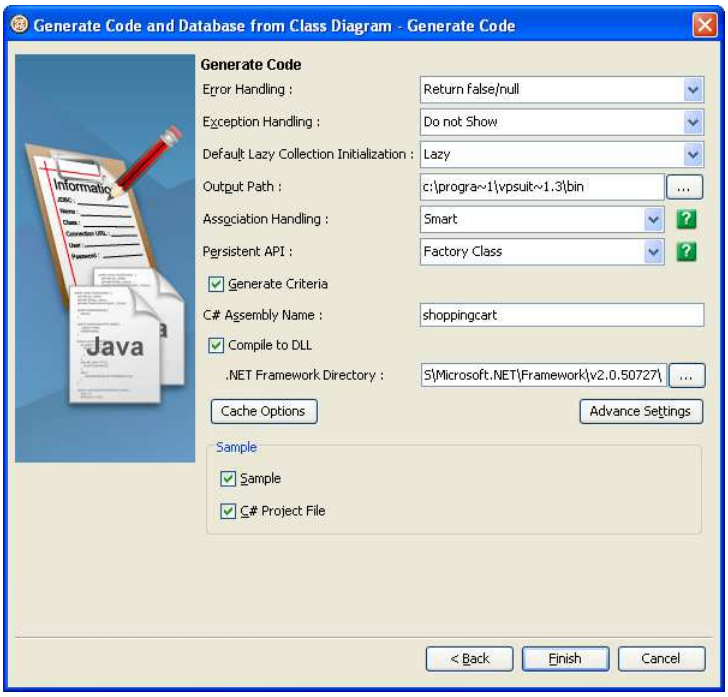


Figure 2.94 - Generate Code Setting for C#

You are asked to specify the code details. DB-VA generates the .NET persistent code based on the information defined here.

For **Generate Code**, check the option to allow the generation of source code.

For **Error Handling**, select the way to handle errors when they occur.

- **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.

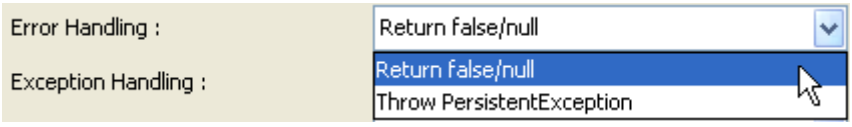


Figure 2.95 - Error Handling options

For **Exception Handling**, select how to handle the exception, either Do not Show, Print to Error Stream or Print to log4net.

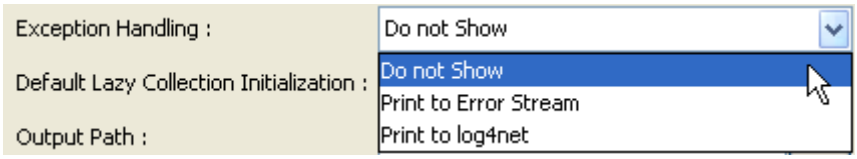


Figure 2.96 - Exception Handling options

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.

For **Output Path**, specify the location to store the generated persistent code source file.

For **Association Handling**, select the type of association handling to be sued, 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.

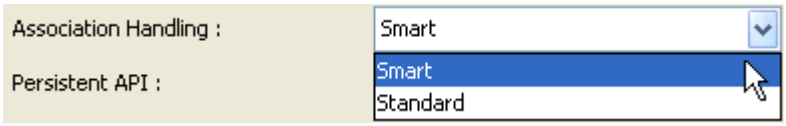


Figure 2.97 - Association Handling options

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

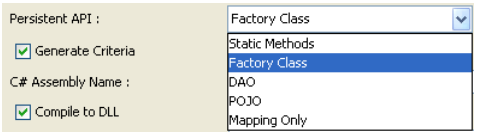


Figure 2.98 - Persistent API options

For **Generate Criteria**, check the option for Generate Criteria to generate the criteria class for each ORM-Persistable class. The criteria class supports querying the database by specifying the searching criteria.

For **C# Assembly Name**, specify the name of the assembly for the .NET application which holds the assembly metadata.

For **Compile to DLL**, check the option for **Compile to DLL**, DB-VA will generate DLL files which can be referenced by .NET projects of language other than C# source.

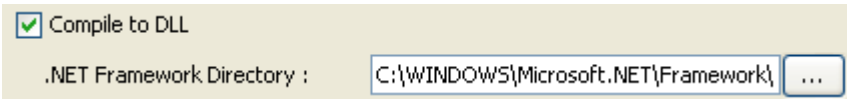


Figure 2.90 - Compile to DLL options

For **Sample**, check the options to generate the sample files and C# project file which guide you through the usage of the .NET persistent classes.

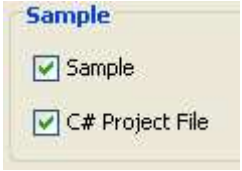


Figure 2.91 - Generate sample options

For **Advance Settings**, you can define the Default Order Collection Type, Default Un-Order Collection Type and Override toString Method. Click on the **Advance Settings** button, select the desired settings in the **Advance Settings** dialog box.

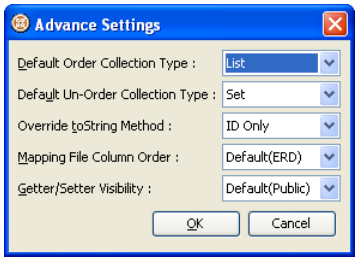


Figure 2.92 - Advance Setting dialog

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

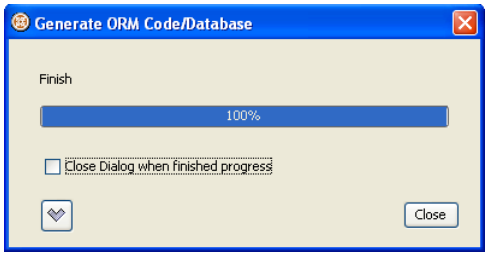
For **Default Un-Order Collection Type,** select the type of un-ordered collection to be used in handling the multiple cardinality relationship, either Set or Bag.

For **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.

Wizard for **Generate Code and Database from Class Diagram** option provides an option of generating code to you. By default, the **Generate Code** option is selected. If you do not want to generate code from class diagram, please deselect the **Generate Code** option. In this case, only database will be generated while persistent code will not be generated.

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



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 and the generated database will be set up to the specified database configuration.

3 Designing Object Model with UML Class Diagram

Chapter 3 - Designing Object Model with UML Class Diagram

DB Visual ARCHITECT (DB-VA) provides you a visual modeling environment for the object model of an application. This chapter shows you how to depict the object models by using a UML Class Diagram.

In this chapter:

Introduction

An object is a self-contained entity with well-defined characteristics and behaviors while the characteristics and behaviors are represented by attributes and operations respectively. A class is a generic definition for a set of similar objects. Hence, an object is an instance of a class. An object model provides a static conceptual view of an application. It shows the key components (objects) and their relationships (associations) within the application system.

DB-VA supports visual modeling for object models, not only by creating a new object model, but also by transforming from a data model. As DB-VA automates object-relational mapping, DB-VA supports the generation of database, code and persistence layer for Java model API and .NET model API, which in turn streamlines the model-code-deploy software development process.

Creating Object Model with Class Diagram

A class diagram can be used to describe the objects and classes inside a system and the relationships between them; and thus, a class diagram is also known as an object model. The class diagram identifies the high-level entities of the system. DB-VA comes with a complete UML 2.0 class diagram for object modeling.

The following section describes how you can depict an object model using the class diagram. DB-VA also supports the generation of persistent code based on the object model, which will be briefly described in the DB-VA Programmer's Guide.

DB-VA provides you with two ways to create a Class Diagram:

1. Drawing a Class Diagram
2. Synchronizing from Data Model to Object Model

Drawing a Class Diagram

You can create a new class diagram in one of the three ways:

- On the menu, click **File > New Diagram > UML Diagrams > Class Diagram**.

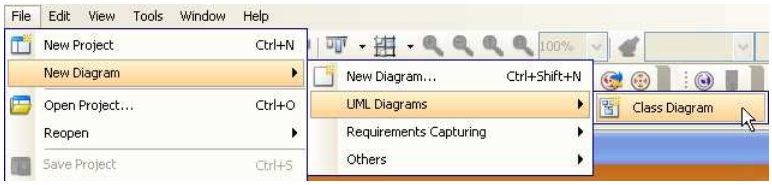


Figure 3.1 - Create a Class Diagram

- On the **Diagram Navigator**, right-click **Class Diagram > Create Class Diagram**.

Figure 3.2 - Create Class Diagram on Diagram Navigator

- On the toolbar, click the **New Class Diagram** icon.

A new class diagram pane is displayed.

Adding Class

1. On the diagram toolbar, click the **Class** shape icon.

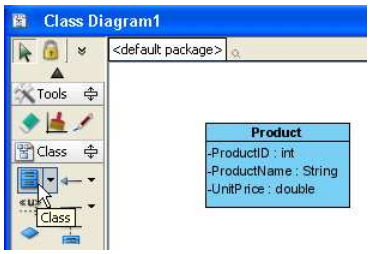


Figure 3.3 - Class shape icon

2. Click a location in the diagram pane.

   DB-VA places an icon representing the class element on the diagram.

3. Type a name for the **Class** element.
   - You can edit the name by double-clicking the name or by pressing the ***F2*** button.

Adding ORM-Persistable Class

ORM-Persistable class is capable of manipulating the persistent data with the relational database. In supporting the generation of persistent code, ORM-Persistable classes should be used in object modeling. Classes added to the class diagram can be stereotyped as ORM-Persistable to manipulate the database. For information on how to specify the stereotype to a class, refer to Specifying Stereotypes section.

DB-VA provides an alternative way to add the ORM-Persistable class easily.

1. On the diagram toolbar, click the **Class** shape icon and hold for a while, a pop-up menu shows.

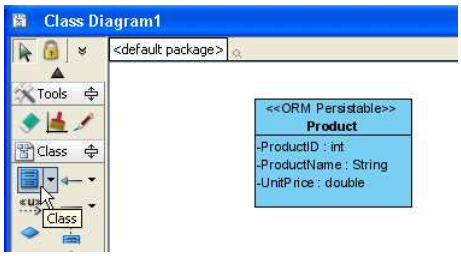


Figure 3.4 - Click on Class shape icon

2. Select **ORM-Persistable Class** from the pop-up menu.

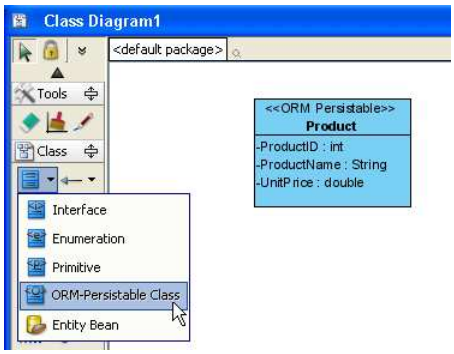


Figure 3.5 - Select ORM-Persistable Class on popup menu

3. Click a location in the diagram pane.

   DB-VA places a class shape icon which is marked with **<<ORM Persistable>>** on the diagram.

Figure 3.6 - ORM Persistable Class

4. Type a name for the **ORM-Persistable Class**.
   - You can edit the name by double-clicking the name or by pressing the ***F2*** button.

Modifying Class Specification

A class specification displays the class properties and relationships.

You can display the Class Specification in one of the two ways:

- Click on a class, click the **Open Specification** resource located at the top-right corner of the class.

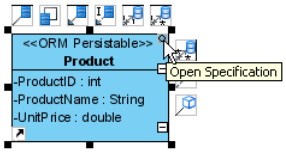


Figure 3.7 - Open Specification resource-centric

- Right-click the class, click **Open Specification...**from the pop-up menu.

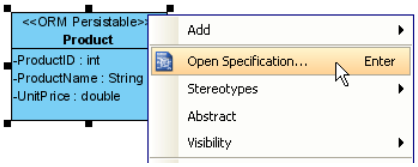


Figure 3.8 - Open specification by click on menu

Class Specification dialog box is displayed, you can modify the class properties and relationships.

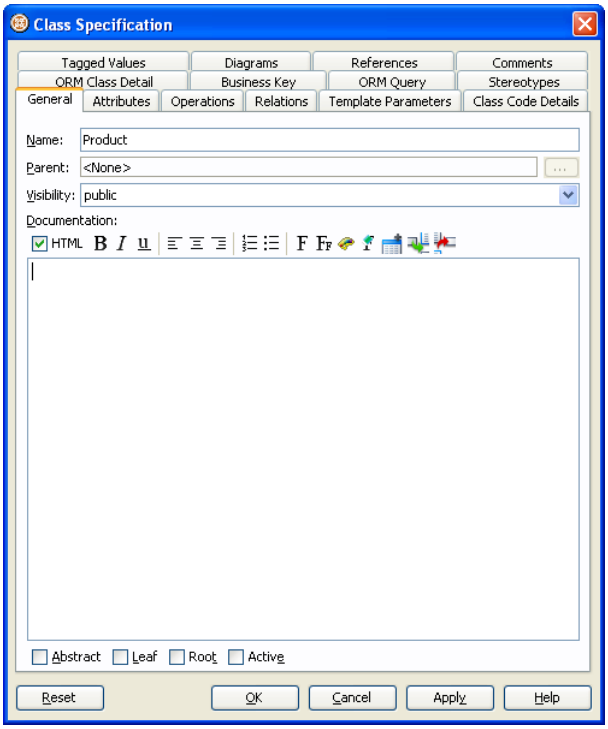


Figure 3.9 - Class Specification dialog

Adding Attribute

An attribute is a property of a class which has a name, a value and also has a type to describe the characteristic of an object.

1. You can add attribute to the class in one of the three ways:
   - Right-click on a class, select **Add > Attribute**.

Figure 3.10 -

2.

A new attribute is added, type the attribute name and type in the form of "**attribute_name: type**". You can also edit the attribute name by double-clicking the attribute name or by pressing the ***F2*** button.

- Click on a class, press the keyboard shortcut - ***Alt + Shift + A***.
- Using Class specification dialog:
  1. Right-click the class element, click **Open Specification...**.
  2. Click the **Attributes** Tab, then click **Add**.

     Attribute Specification dialog box is displayed, you can modify the attribute name and properties, such as type.

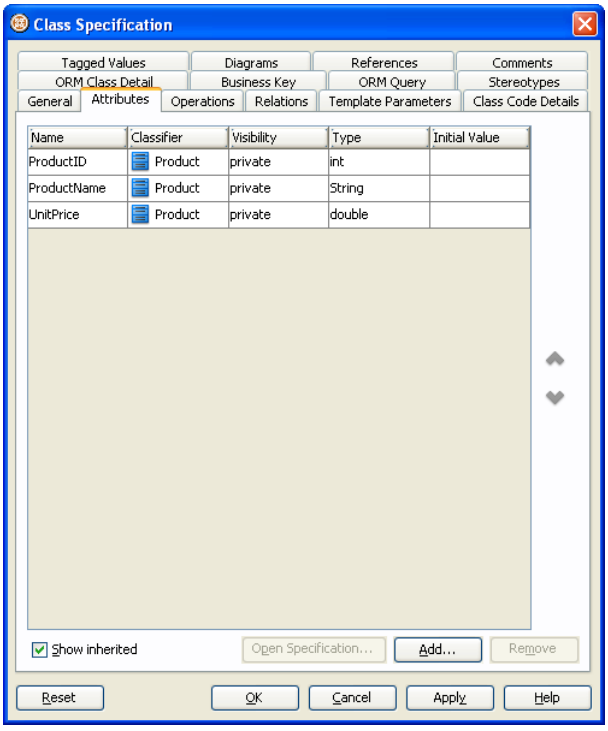


Figure 3.11 - Class Specification dialog

Modifying Attribute Specification

An attribute specification displays the attribute properties, such as name, type, and initial value etc.

To open the Attribute Specification:

1. Right-click the attribute, click **Open Specification...**from the pop-up menu.

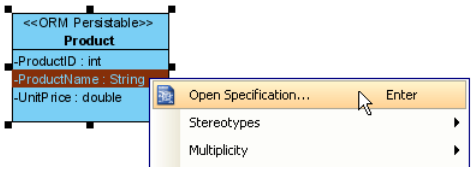


Figure 3.12 - To open attribute specification

The **Attribute Specification** dialog box is displayed, you can modify the attribute properties.

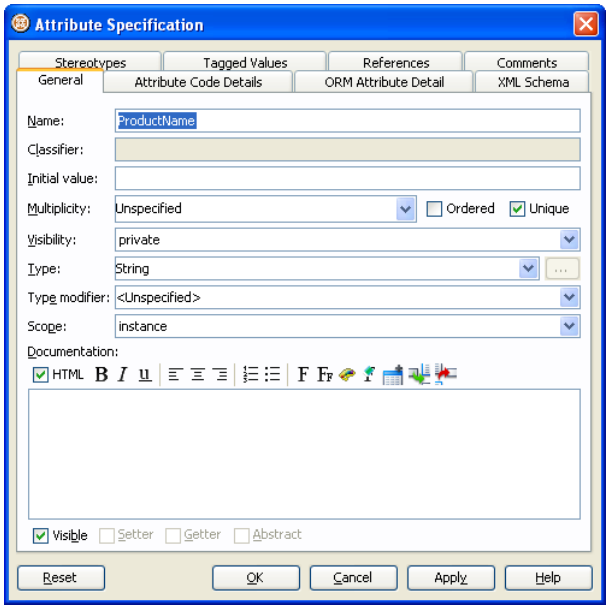


Figure 3.13 - Attribute Specification dialog

Adding Association

An association refers to the relationship specifying the type of link that exists between objects. It shows how the objects are related to each other.

You can add an association to the classes in one of the two ways:

- Using Resource-Centric Interface
    1. Click on a class, a group of valid editing resources is displayed around the shape.

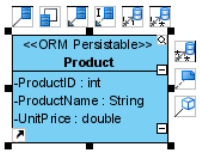


Figure 3.14 - Resource-Centric

2. Mouse over the smart resource of association, select the desired resource, such as "**One-to-Many Bi-directional Association - > Class**".

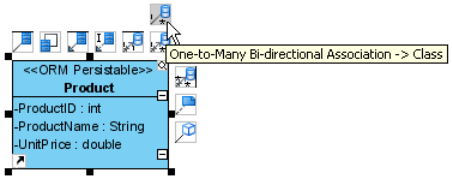


Figure 3.15 - "One-to-Many Bi-directional Association -> Class" resource

3. Drag the resource to the associated class.

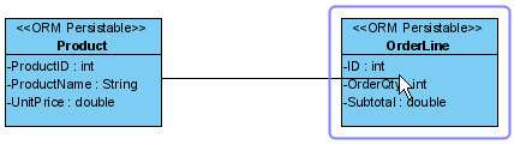


Figure 3.16 - Create an association

Smart resource is a kind of resource which groups the resources of similar purpose together and enables the last selected resource (the default resource) of the group to be visible. To see all the resources, mouse over the default resource to expand it.

- Using Toolbar icon
  1. On the diagram toolbar, click the **Association** icon.

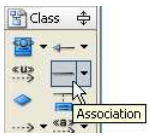


Figure 3.17 - Association icon button

  2. Click on a class and drag to the associated class.

A connector is added between the two classes.

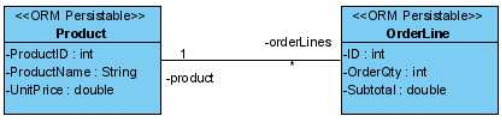


Figure 3.18 - An association are created

DB-VA automatically creates the roles in an association between ORM-Persistable classes.

Modifying Association Specification

You can edit the association specification in one of the three ways:

- Using Open Specification
  1. Right-click on the connection line, click **Open Specification...** from the pop-up menu.

     Association Specification dialog box is displayed, you can modify the association properties, Roles of classes in Association End From and To, Multiplicity and Navigation etc.

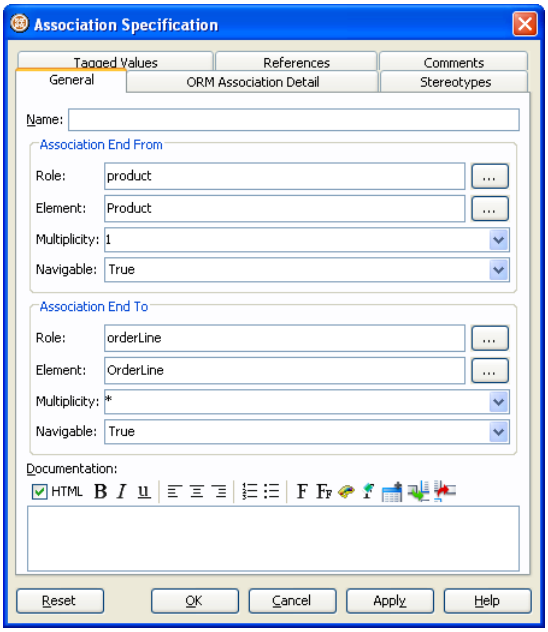


Figure 3.19 - Association Specification dialog

- Using Pop-up Menu
  1. Right-click on the connection line, the property of the association specification is displayed in the pop-up menu, including **Navigable, Multiplicity**, **Visibility**, **Aggregation Kind** and **Edit Role Name...**, **Qualifier...**.

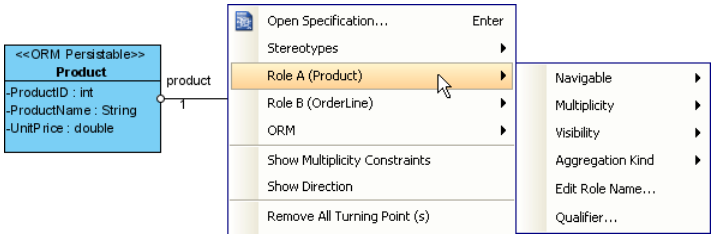


Figure 3.20 - Modify association specification by using popup menu

  2. Select the property that you want to edit, check the desired value.

If you right-click on the connection line towards a class, the pop-up window shows the properties of association specification of the respective class. If you right-click in the middle of the connection line, the pop-up window shows all properties of association specification of both classes.

Role name of the class describes how it acts in the association which will be used in the generation of persistent code. Be sure that you have given the role names to the classes in the association in order to proceed to the generation of code.

- Using Property Pane
    1. On the menu, click **View > Panes > Property**.

       The property pane will be displayed.

    2. Click on the connection line.

       The properties of the association specification are displayed in the property pane. You can edit the property under the property pane.

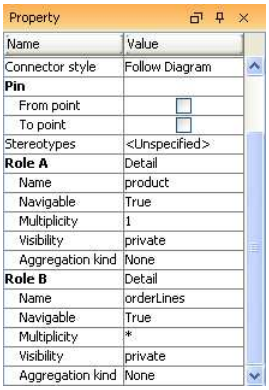


Figure 3.21 - Property Pane

Adding Operation

An operation, also called function or method is the behavior of an object relates to how it acts and reacts.

You can add operation to the class in one of the three ways:

- Right-click on a class, select **Add > Operation**.

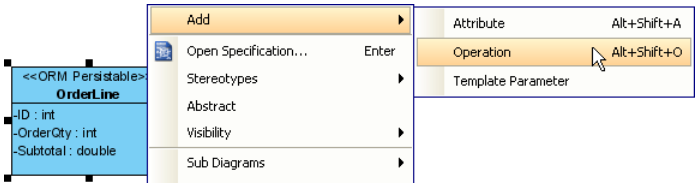


Figure 3.22 - Add an operation

A new operation is added, enter the operation in the form of "**operation_name(parameter_name: type) : return_type**". You can also edit the operation name by double-clicking the operation or by pressing the ***F2*** button.

- Click on a class, press the keyboard shortcut - ***Alt + Shift + O***.
- Using Class Specification dialog:
    1. Right-click the class element, click **Open Specification...**.
    2. Click the **Operations** tab, then click **Add**.

Operation Specification dialog box is displayed, you can modify the operation name and properties, such as return type, parameters.

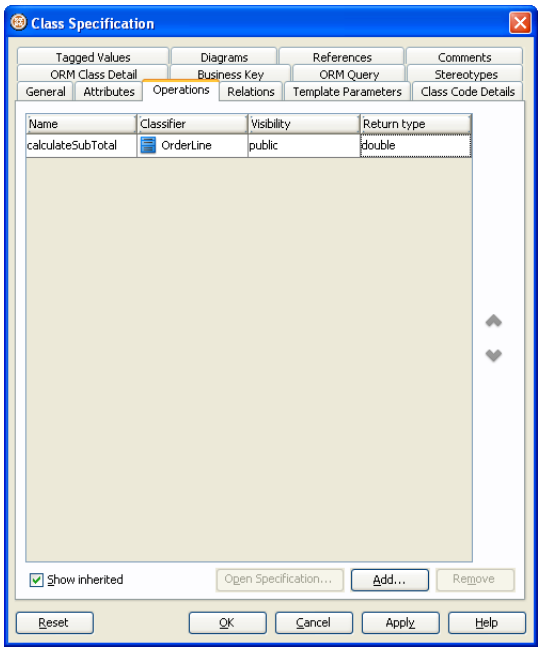


Figure 3.23 - Add operation in Class Specification dialog

Modifying Operation Specification

An operation specification displays the operation properties, such as name, visibility, return type, and parameters etc.

To open the Operation Specification:

1. Right-click the operation, click **Open Specification...** from the pop-up menu.

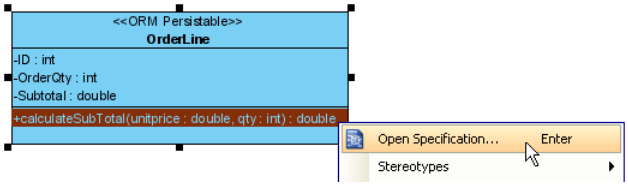


Figure 3.24 - To open operation specification

The **Operation Specification** dialog box is displayed, you can modify the operation properties.

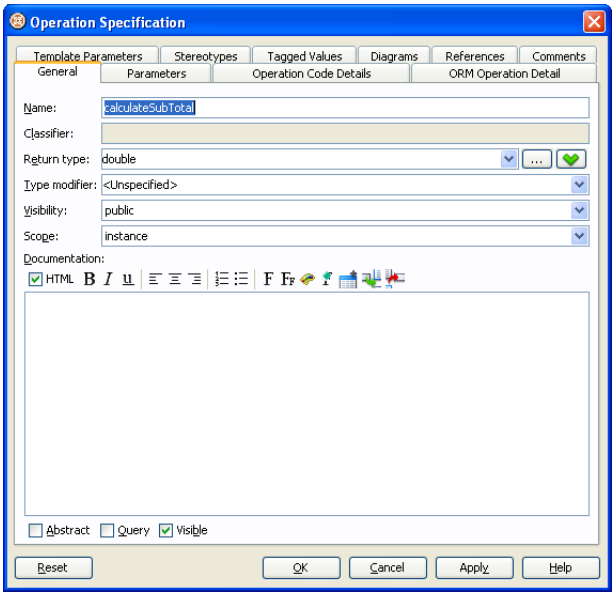


Figure 3.25 - Operation Specification dialog

Adding ORM Implementation Class

ORM-Persistable class is used to generate the persistent class which has the ability to access the database including the basic operations for add, update, delete and search. As the generated code provides the basic operations for manipulating the persistent data, you may find it insufficient and want to add extra logic to it.

DB-VA promotes the use of ORM Implementation Class to add extra logic to the ORM-Persistable class. When generating the persistent code, the ORM implementation class will also be generated, and thus you can implement the logic to the method in the generated implementation class for manipulating the persistent data.

To add an ORM implementation class:

1. Create an ORM-Persistable class with an operation for adding extra logic.

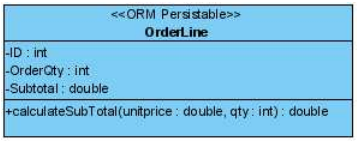


Figure 3.26 - ORM-Persistable class with an operation

2. Mouse over the class and drag the resource of "**Create ORM Implementation Class**" to the diagram pane.

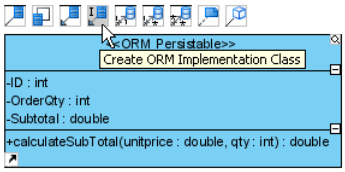


Figure 3.27 - "Create ORM Implementation Class" resource

The implementation class is created and connected to the source class with generalization stereotyped as **<<ORM Implementation>>.** The source class becomes an abstract class with an abstract operation.

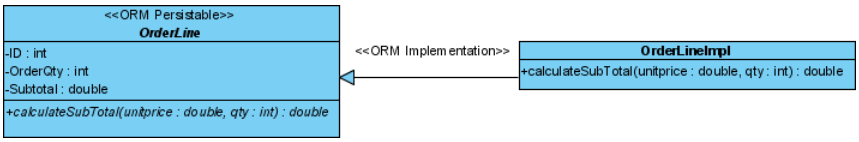


Figure 3.28 - ORM Persistable Class and ORM Implementation Class

Synchronizing from Data Model to Object Model

DB-VA allows you to generate a Class Diagram from an ERD by synchronization if there is an ERD.
You can synchronize the ERD to Class Diagram in one of the three ways:

- On the menu, click **Tools > Object-Relational Mapping (ORM) > Synchronize to Class Diagram**.

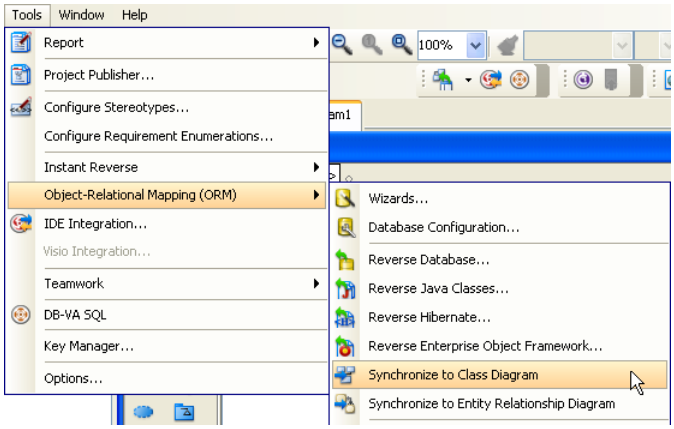


Figure 3.29 - Synchronize ERD to Class Diagram

- Right-click on the ERD, select **Synchronize to Class Diagram**.

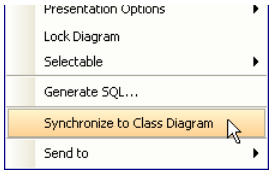


Figure 3.30 - Synchronize to Class Diagram by click on popup menu

- On the ERD, hold down the right-mouse button, move the mouse from right to left to form the gesture. A blue path is shown indicating the gesture.

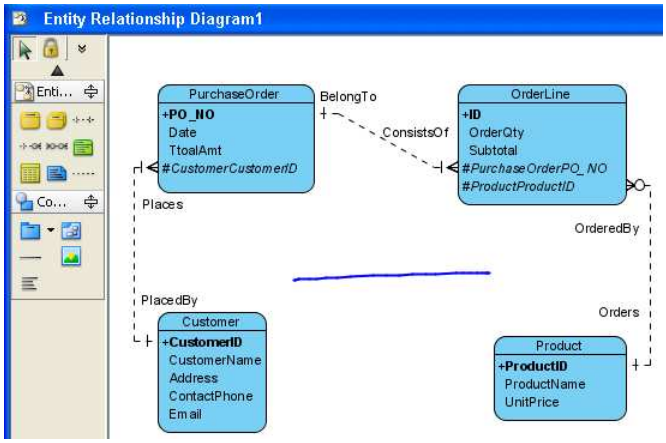


Figure 3.31 - Synchronize to Class Diagram by using Gesture

A class diagram is generated and can be found under the **Diagram Navigator**.

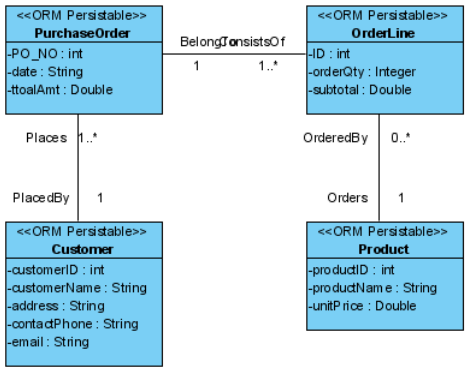


Figure 3.31 - The generated Class Diagram

Defining Package for Classes

DB-VA provides you with two alternative ways to define the packages for the classes.

- Enter the package name to the **<default package>** located at the top-left corner of the class diagram by double-clicking the **<default package>.**

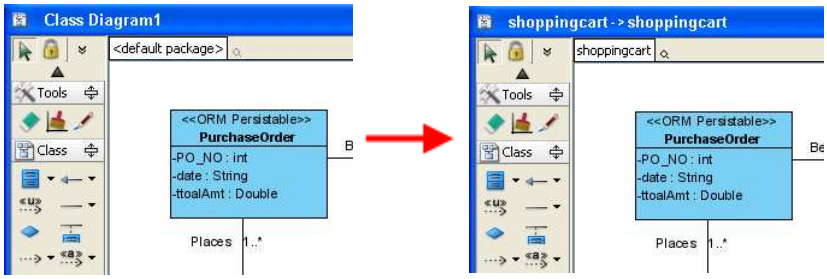


Figure 3.32 - Define the package

- Create a package element on the diagram:
  1. On the diagram toolbar, click the **Package** shape icon.

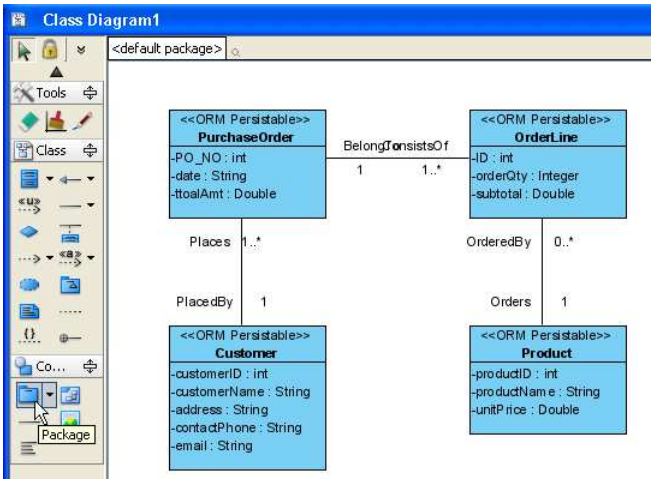


Figure 3.33 - Click on the Package icon

2. Click a location in the diagram pane to create a package element on the diagram.

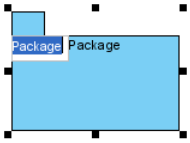


Figure 3.34 - Rename the Package element

3. Type a name for the **Package** element.
4. Move the desired **Class** elements to the package

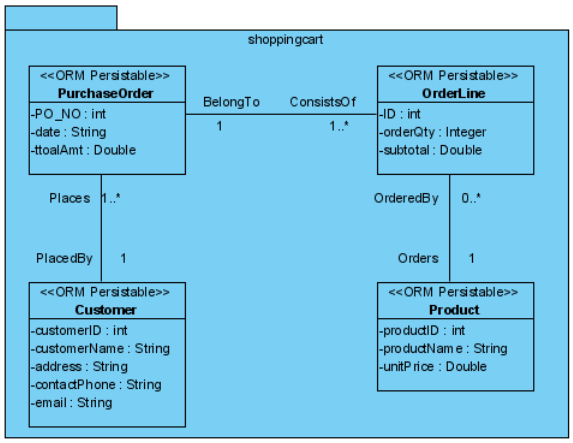


Figure 3.35 - Move the Class element into the Package element

After defining the packages to the classes, the classes are inside the package and depicted on the class repository.

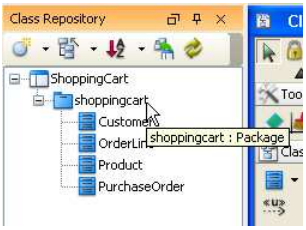


Figure 3.36 - Class Repository show the Classes in the package

Specifying Stereotypes

Stereotype extends the semantics of the UML metamodel. It classifies the element in what respects it behaves as an instance of metamodel. In order to enable the mapping between object model and relational database, the class has to be stereotyped as ORM Persistable.

To specify the stereotypes of a class:

1. Right-click a class, select **Stereotypes > Stereotypes...**.

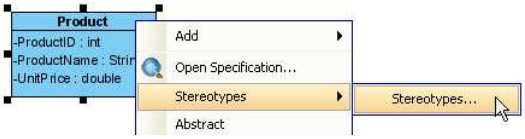


Figure 3.37 - To add/remove stereotypes

The **Class Specification** dialog box is shown with **Stereotypes** Tab

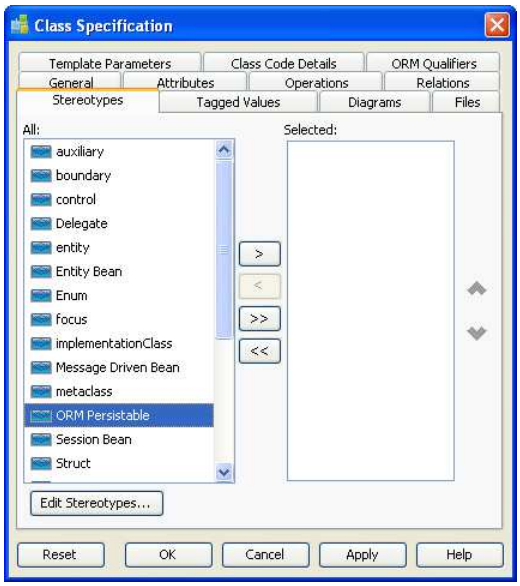


Figure 3.38 - Class Specification dialog (Stereotypes Tab)

2. Select **ORM Persistable**, then > button and **OK**.

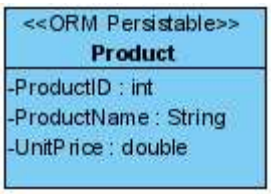


Figure 3.39 - Class with stereotypes

Specifying Inheritance Strategy

In a generalization, the subclass inherits all the features of the superclass. DB-VA provides two inheritance strategies - table per class hierarchy and table per subclass to transform the generalization hierarchy to relational model. By default, table per class hierarchy is used for the generalization.

When transforming generalization into relational model, DB-VA transforms the generalization according to the inheritance strategy applied. For more information on the transformation, refer to the description of Mapping Inheritance/Generalization section.

You can specify the inheritance strategy in one of the two ways:

- Specifying from Superclass
    1. Right-click the superclass, select **ORM > ORM Class Details...**from the pop-up menu. The **Class Specification** dialog showing the **ORM Class Detail** tab is displayed.

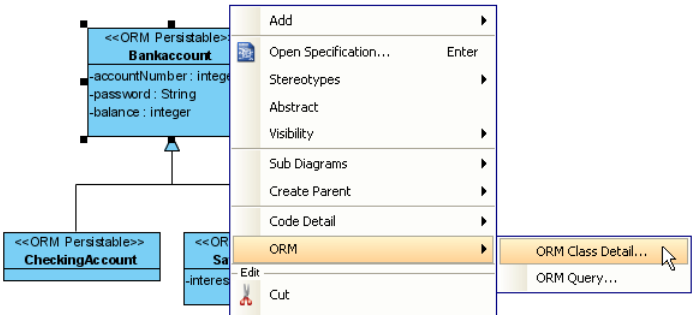


Figure 3.40 - To open ORM Class Detail

    2. Click **Subclasses...** to open the **Inheritance Strategy** dialog box.

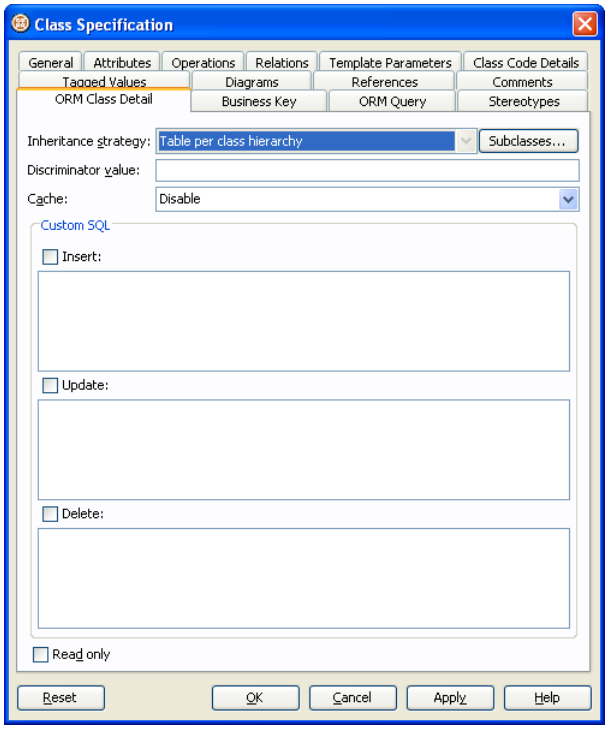


Figure 3.41 - Class Specification (ORM Class Detail Tab)

3. Select the desired subclass from the generalization tree, select the **Inheritance Strategy** from the drop-down menu, and then click **Apply**.

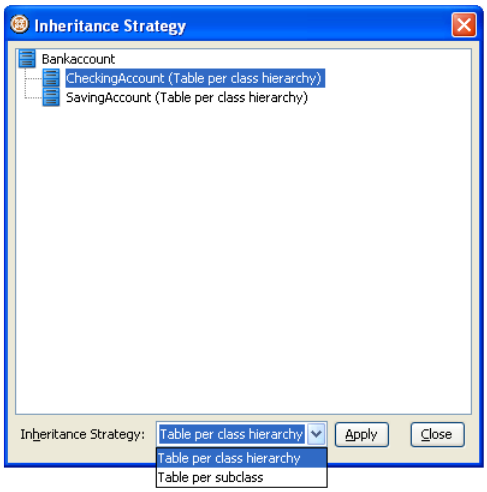


Figure 3.42 - Select the Inheritance Strategy

- Specifying from Subclass
  1. Right-click the subclass, select **ORM > ORM Class Details...** from the pop-up menu. The **Class Specification** dialog box showing the **ORM Class Detail** tab is displayed.

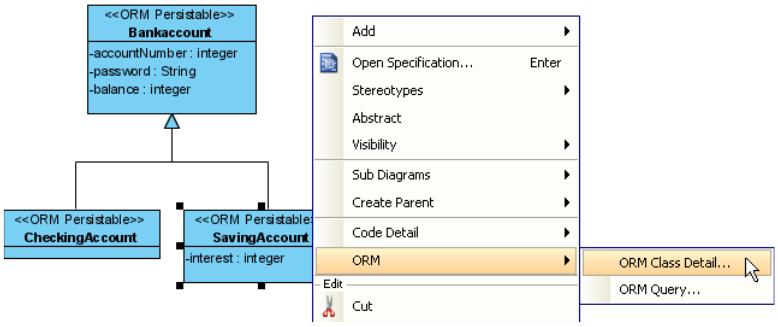


Figure 3.43 - Open ORM Class Detail for the Sub-Class

2. Select the **Inheritance strategy** from the drop-down menu.

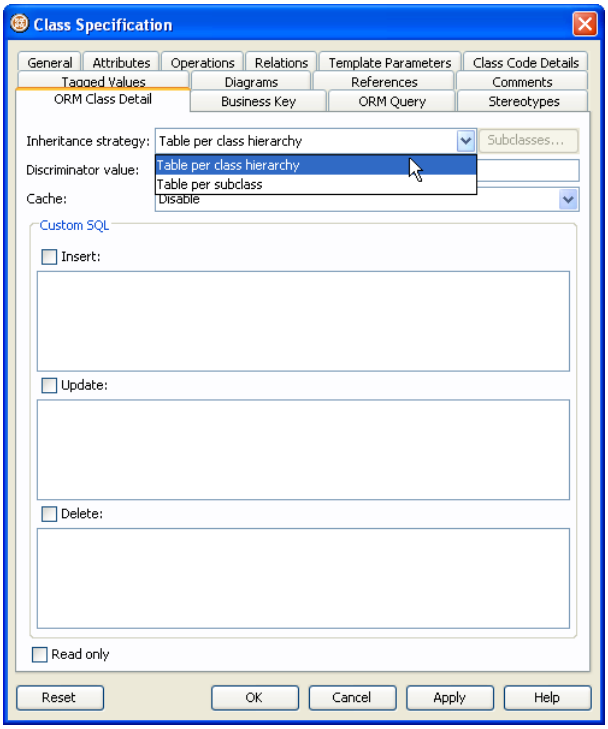


Figure 3.44 - Select the Inheritance Strategy

These two inheritance strategies can be applied to different subclasses within a generalization hierarchy in Java project. Applying two strategies to different subclasses within a generalization in .NET project will result in error when the generation of code and database.

Specifying Collection Type

If one end of an association contains a multiplicity of many, a collection class will be generated for handling the multiple instances. DB-VA allows you to specify the type of collection, including set, bag, list and map.

Set is an unordered collection that does not allow duplication of objects. Bag is an unordered collection that may contain duplicate objects. List is an ordered collection that allows duplication of objects. Map is an ordered collection that maps keys to values while each key can map to exactly one value.

You can specify the collection type in one of the two ways:

- Right-click on the connection line, select the desired collection type from the pop-up menu of **ORM > Collection Type**.

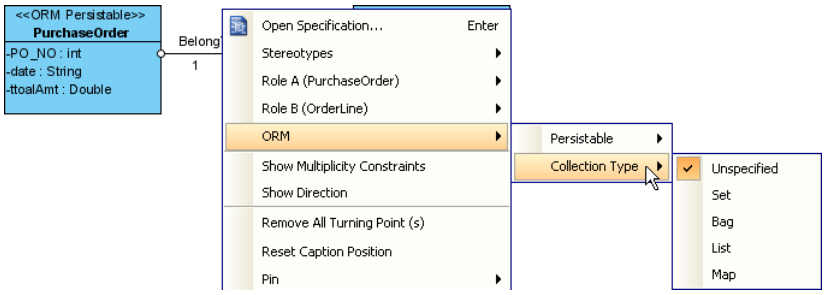


Figure 3.45 - Set the Collection Type in popup menu

- Using Association Specification dialog:
  1. Right-click on the connection line, click **Open Specification...**from the pop-up menu.

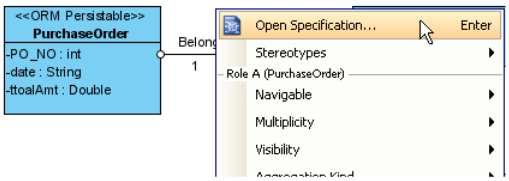


Figure 3.46 - To open the association specification

  2. Click the **ORM Association Detail** tab, select the **Collection Type** from the drop-down menu.

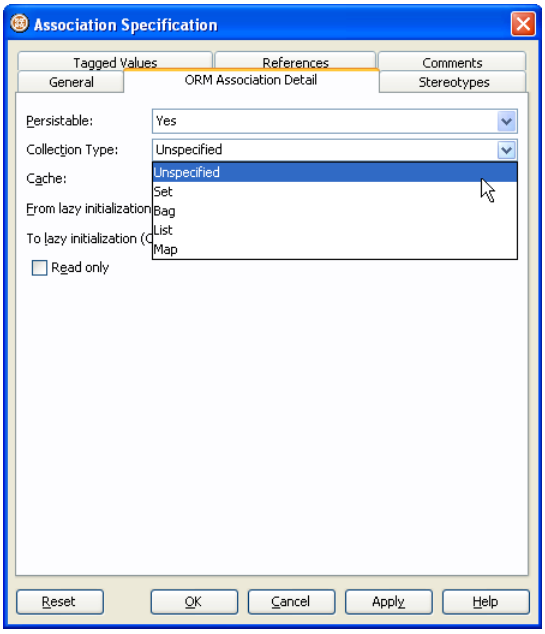


Figure 3.47 - Association Specification (ORM Association Detail Tab)

Defining ORM Qualifier

ORM Qualifier is used to specify the extra retrieval rules of the generated persistent class for querying the database. DB-VA allows you to define the ORM Qualifiers of the classes in the class diagram before the generation of persistent code. For more information on the usage of ORM Qualifier, refer to the DB-VA Programmer's Guide.

1. Open the ORM Qualifier tab inside the Class Specification dialog box in one of the two ways:
   - Right-click on the ORM-Persistable class, select **ORM > ORM Query...** from the pop-up menu. The **Class Specification** dialog box showing the **ORM Query** tab is displayed.

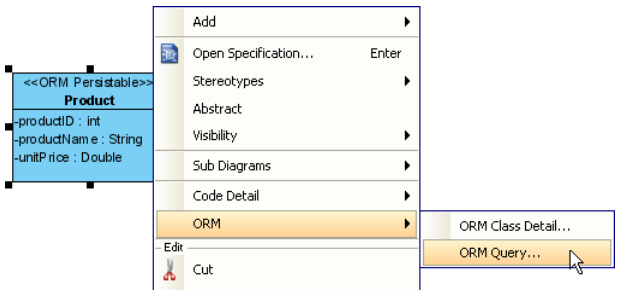


Figure 3.48 - To edit the ORM Qualifier

- Edit ORM Qualifier in Class Specification dialog:
  1. Right-click on an ORM-Persistable class that you want to add extra retrieval rules, click **Open Specification**.

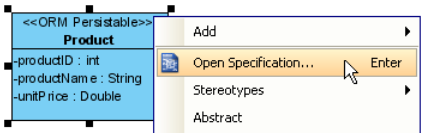


Figure 3.49 - To open the Class Specification dialog

  2. Click the **ORM Query** tab.

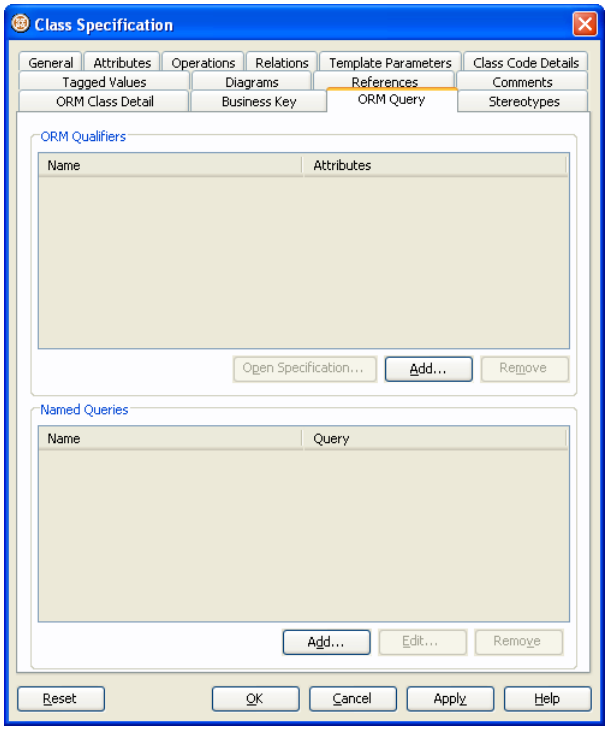


Figure 3.50 - Class Specification dialog (ORM Query Tab)

2. Click **Add**, the **ORM Qualifier Specification** dialog box is displayed with a list of attributes of the selected class.
3. Enter the name of the ORM Qualifier, place a check mark for the **Key** column of the attribute that will be used in querying the database.

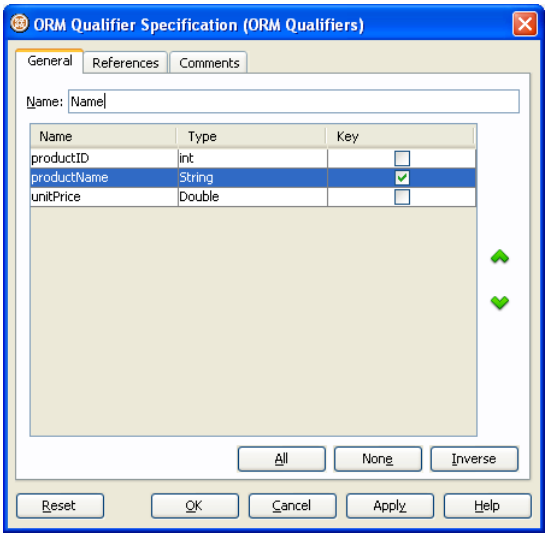


Figure 3.51 - ORM Qualifier Specification dialog

4. Click **OK** to confirm adding the ORM Qualifier. The newly added ORM Qualifier is listed on the **ORM Query** tab.

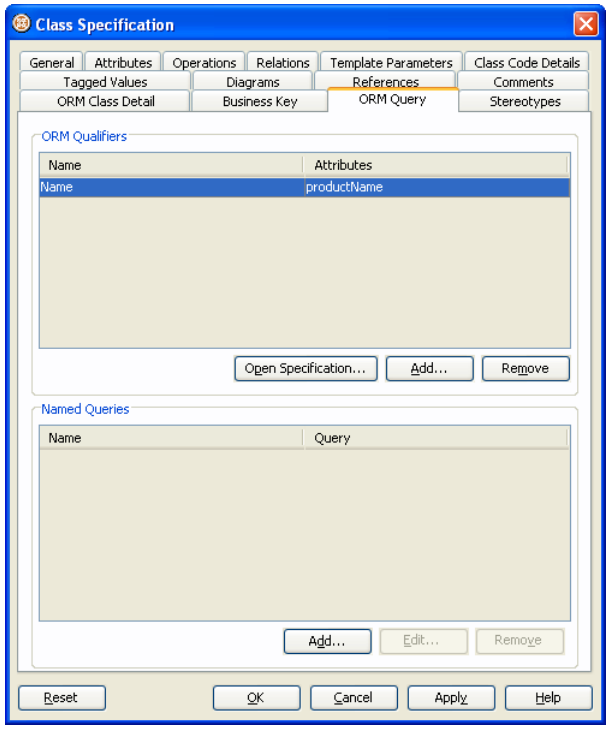


Figure 3.52 - ORM Qualifier added in the ORM Qualifiers list

Customizing SQL

DB-VA generates SQL statements which are ready-to-use for accessing the database. In some cases, you may find the generated SQL statements not appropriate for your needs. DB-VA allows you to override the generated SQL statements, including the Insert, Update and Delete statements whenever you want to.

To customize the generated SQL statements:

1. Right-click on an ORM-Persistable class that you want to customize the SQL statements, select **ORM > ORM Class Details...**from the pop-up menu. The **Class Specification** dialog box showing the **ORM Class Detail** tab is displayed.

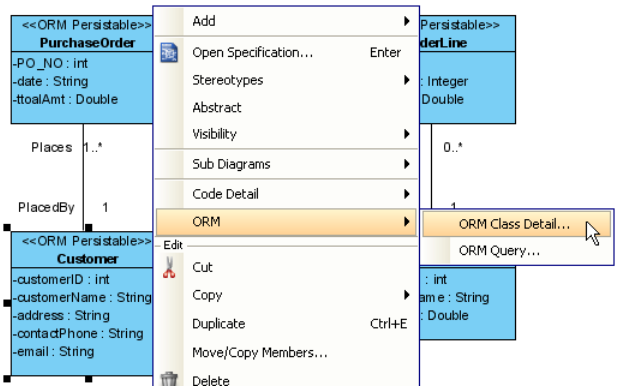


Figure 3.53 - Open the ORM Class Detail

2. Select the type of SQL statement that you want to customize.

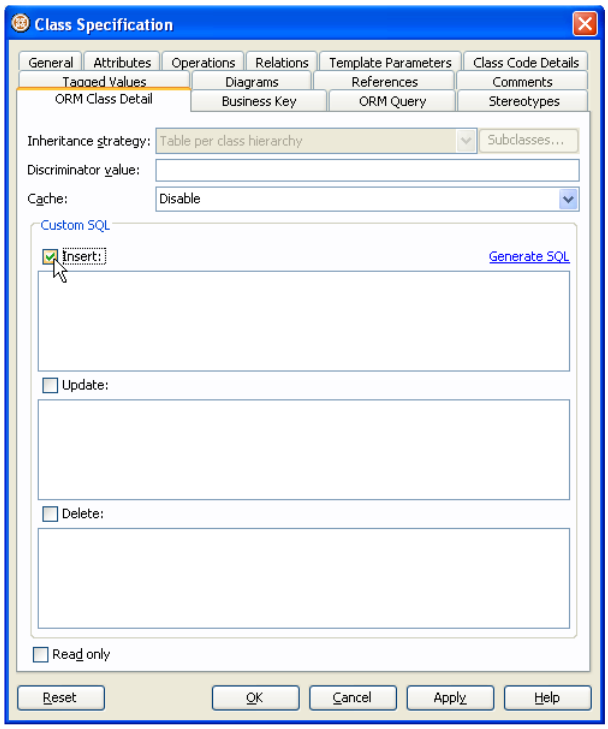


Figure 3.54 - Class Specification dialog (ORM Class Detail)

3. Click **Generate SQL** to generate the ready-to-use SQL statement.

Figure 3.55 - Generate Insert SQL statement

The SQL statement is generated based on the property of class.

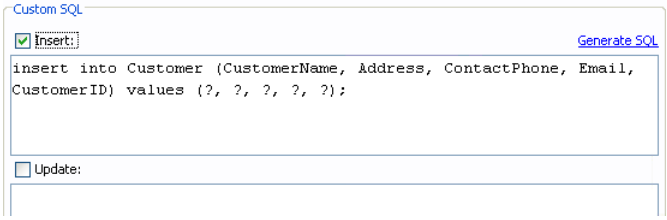


Figure 3.56 - The generated Insert SQL statement

4. Modify the SQL statement to the desired one.

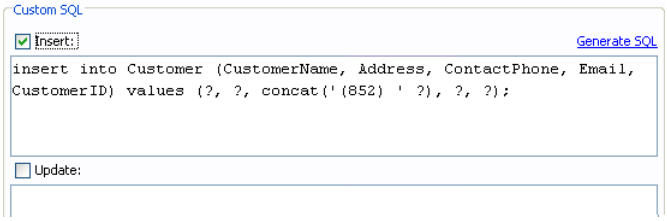


Figure 3.57 - The modified SQL statement

4

Designing Data Model by Entity Relationship Diagram

Chapter 4 - Designing Data Model by Entity Relationship Diagram

DB Visual ARCHITECT (DB-VA) provides you a visual modeling environment for the object model of an application. This chapter shows you how to depict the object models by using Entity Relationship Diagram.

In this chapter:

- Introduction
- Creating Data Model by Entity Relationship Diagram
- Creating Array Table in Data Model
- Creating Partial Table in Data Model
- Copying SQL Statements

Introduction

An entity is an object in the business or system with well-defined characteristics which are represented by columns showing what information can be stored. In relational databases, an entity refers to a record structure, i.e. table.

A data model provides the lower-level detail of a relational database of an application. It shows the physical database models and their relationships in an application. An entity relationship diagram can be used to describe the entities inside a system and their relationships with each other; the entity relationship diagram is also known as a data model.

DB-VA supports visual modeling for data models, not only by creating a new data model, but also by transforming from an object model. As DB-VA automates object-relational mapping, DB-VA supports the generation of database, code and persistence layer for Java model API and .NET model API, which in turn streamlines the model-code-deploy software development process.

Creating Data Model by Entity Relationship Diagram

Entity relationship diagram is a graphical representation of a data model of an application. It acts as the basis for mapping the application to the relational database.

The following section describes how you can depict the data model using the entity relationship diagram.

DB-VA provides you with two ways to create a Class Diagram:

1. Drawing an Entity Relationship Diagram (ERD)
2. Synchronizing from Object Model to Data Model

Drawing an Entity Relationship Diagram

You can create a new ERD in one of the three ways:

- On the menu, click **File > New Diagram > Others > Entity Relationship Diagram**.

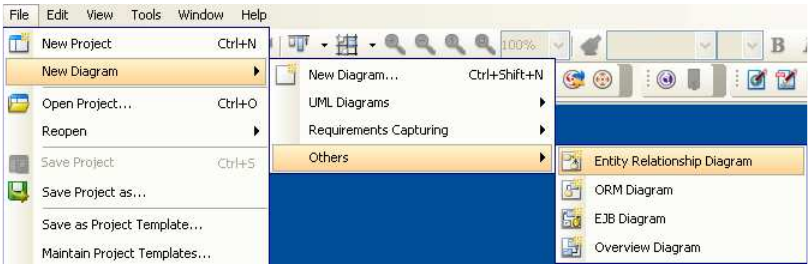


Figure 4.1 - Create an ERD by click on menu

- On the **Diagram Navigator**, right-click **Entity Relationship Diagram > Create Entity Relationship Diagram**.

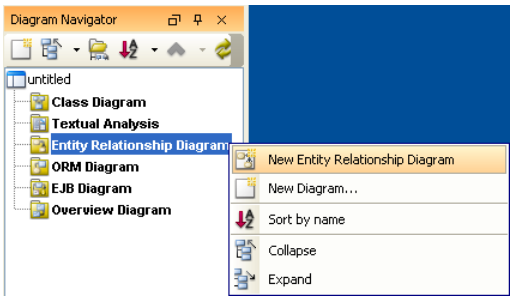


Figure 4.2 - Create ERD by click on Diagram Navigator

- On the toolbar, click the **New Entity Relationship Diagram** icon.

A new Entity Relationship Diagram pane is displayed.

Adding Entity

1. On the diagram toolbar, click the **Entity** shape icon.

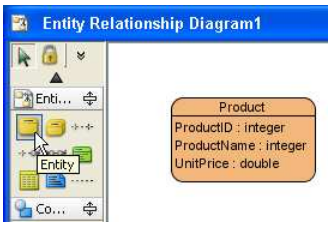


Figure 4.3 - Create an Entity by using the Entity icon

2. Click a location in the diagram pane.

   DB-VA places an icon representing the entity element on the diagram.

3. Type a name for the **Entity** element.
   - You can edit the name by double-clicking the name or by pressing the ***F2*** button.

Modifying Entity Specification

An entity specification displays the entity properties and constraints.
You can display the Entity Specification in one of the two ways:

- Click on an entity class, click the **Open Specification** resource located at the top-right corner of the entity.

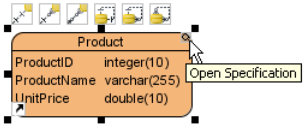


Figure 4.4 - "Open Specification" resource

- Right-click the entity, click **Open Specification...**from the pop-up menu.

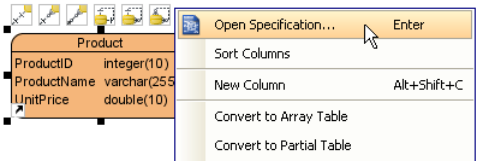


Figure 4.5 - Open specification by click on popup menu

Entity Specification dialog box is displayed, you can modify the entity properties and constraints.

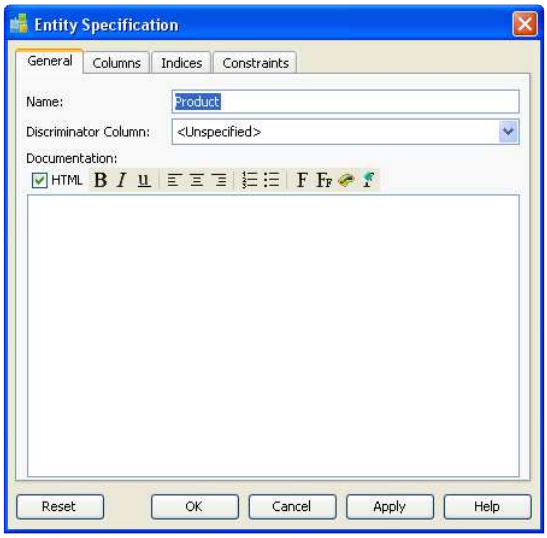


Figure 4.6 - Entity Specification dialog

Adding Column

You can add a new column to the entity in one of the three ways:

- Right-click on an entity, select **New Column**.

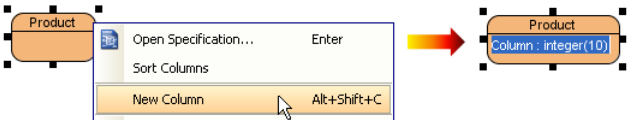


Figure 4.7 - Add column by click on popup menu

A new column is added, type the column name and type in the form of "**column_name: type**". You can also edit the column name by double-clicking the column name or by pressing the ***F2*** button.

- Click on an entity, press the keyboard shortcut - ***Alt + Shift + C***.
- Add column in **Entity Specification** dialog:
    1. Right-click the entity element, click **Open Specification**.
    2. Click the **Columns** tab, then click **Add**.

Column Specification dialog box is displayed, you can modify the column name and properties, such as type.

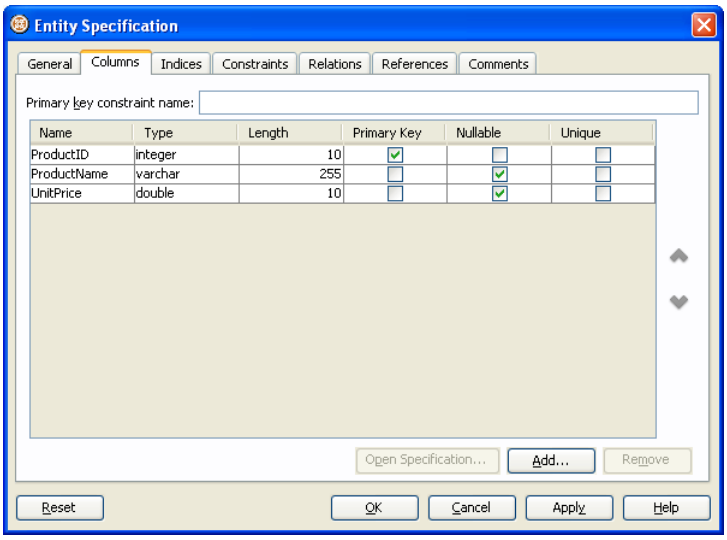


Figure 4.8 - Entity Specification dialog

Modifying Column Specification

To open the Column Specification:

1. Right-click the column, click **Open Specification...**from the pop-up menu.

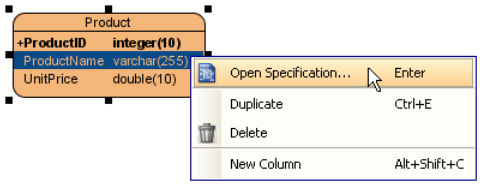


Figure 4.9 - To open the column specification

The **Column Specification** dialog box is displayed, you can modify the column properties.

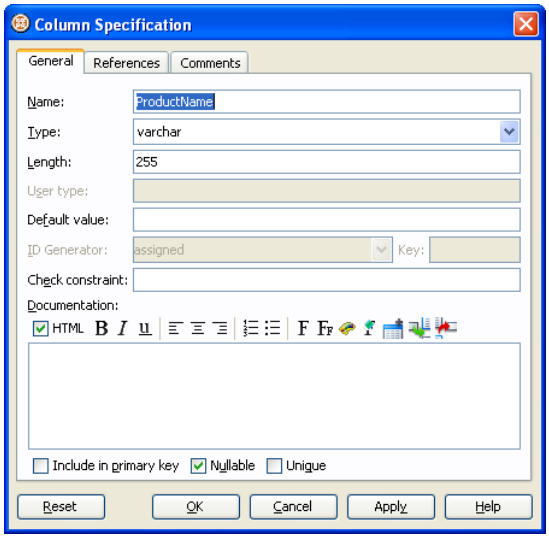


Figure 4.10 - Column Specification dialog

Adding Relationship

Relationship shows how the entities are related to each other.

You can add a relationship to the entities in one of the two ways:

- Using Resource-Centric Interface
    1. Click on an entity, a group of valid editing resources is displayed around the shape.
    2. Mouse over the resource, select the desired resource, such as "**One-to-One Relationship - > Entity**".

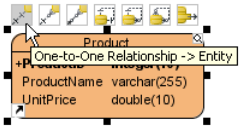


Figure 4.11 - "One-to-one Relationship -> Entity" resource

3. Drag the resource to the related entity.

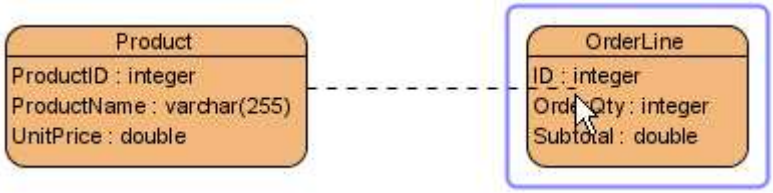


Figure 4.12 - Drag the resource to the related entity

- Using Toolbar icon
    1. On the diagram toolbar, click the **Relationship** icon.
        - One-to-One Relationship
        - One-to-Many Relationship
        - Many-to-Many Relationship
    2. Click on an entity and drag to the related entity.

A connector is added between the two entities.

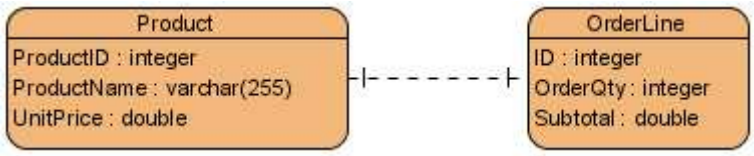


Figure 4.13 - an one-to-one relationship created

Modifying Relationship Specification

The relationship specification displays the relationship properties, such as name, phrase, and cardinality of the related entities.
To open the relationship specification dialog box:

1. Right-click on the connection line, click **Open Specification...** from the pop-up menu.

   Relationship Specification dialog box is displayed, you have to modify the relationship properties, Phrase and Cardinality.

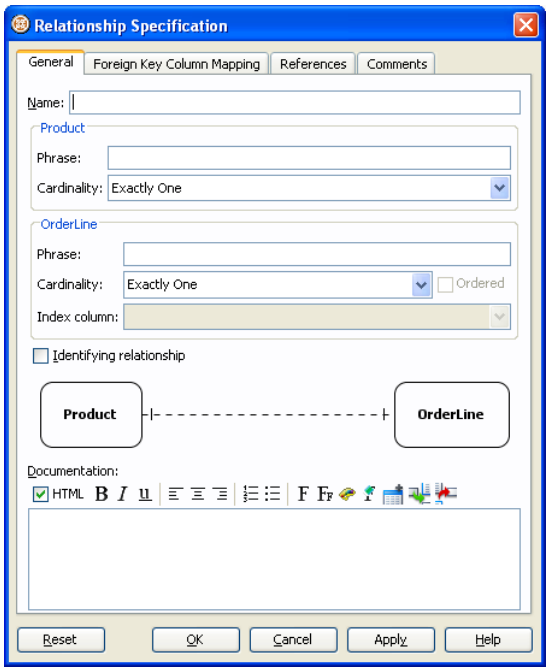


Figure 4.14 - Relationship Specification dialog

Synchronizing from Object Model to Data Model

DB-VA allows you to generate the ERD from a class diagram by synchronization as if there is a class diagram.
You can synchronize the Class Diagram to ERD in one of the three methods:

- On the menu, click **Tools > Object-Relational Mapping (ORM) > Synchronize to Entity Relationship Diagram**.

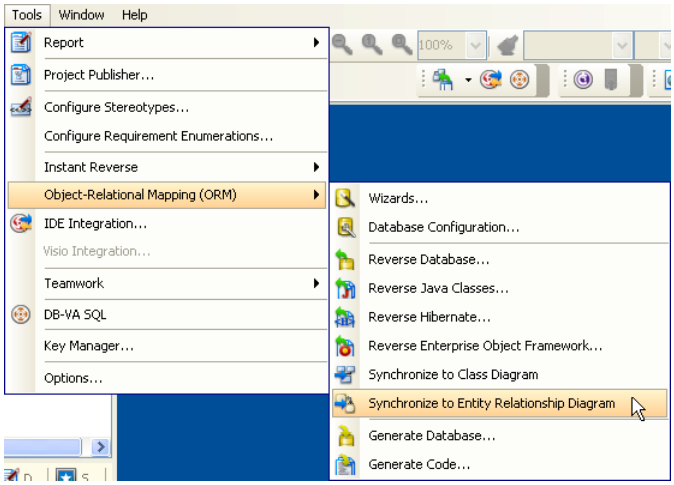


Figure 4.15 - Synchronize to ERD

- Right-click on the class diagram, select **Synchronize to Entity Relationship Diagram**.

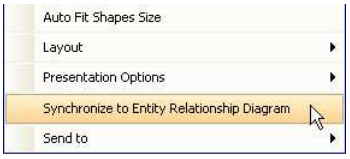


Figure 4.16 - Synchronize to ERD by click on popup menu

- On the class diagram, hold down the right-mouse button, move the mouse from left to right to form the gesture. A blue path is shown indicating the gesture.

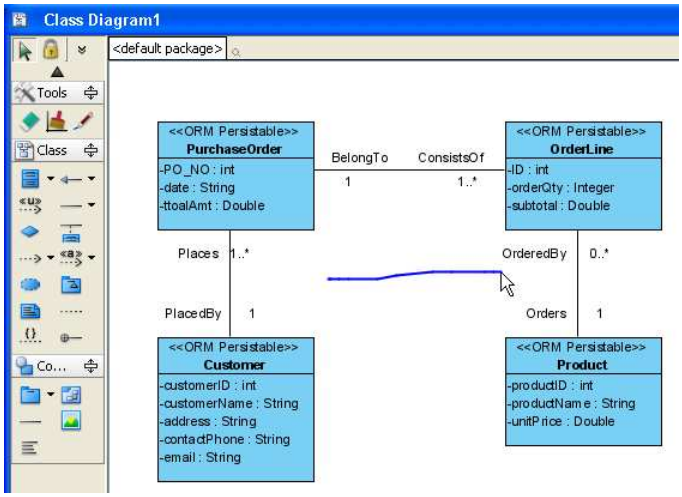


Figure 4.17 - Synchronize to ERD by using Gesture

An entity relationship diagram is generated and can be found under the **Diagram Navigator**.

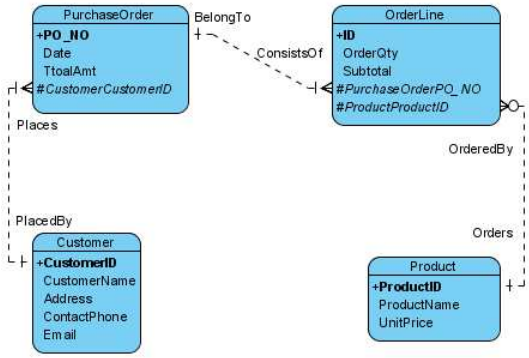


Figure 4.18 - The generated ERD

Specifying Primary Key

You can specify the column to be included in the primary key in one of the two ways:

- Specify Primary Key in **Entity Specification** dialog:
  1. Right-click on the entity, click **Open Specification...** to open the **Entity Specification** dialog box.
  2. Click the **Columns** tab, check the **Primary Key** option for the column that will be included in the primary key.

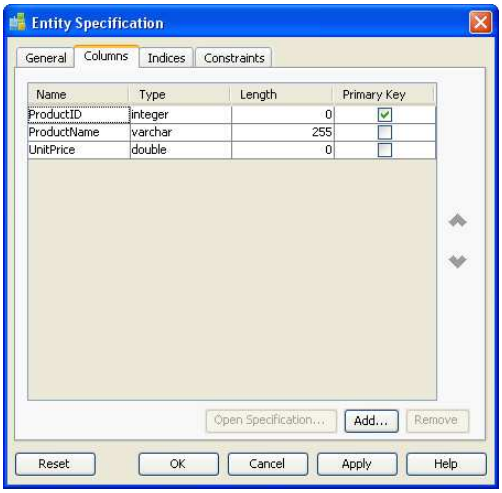


Figure 4.19 - Entity Specification dialog

- Specify Primary Key in **Column Specification** dialog:
  1. Right-click the column, click **Open Specification...** to open the **Column Specification** dialog box.
  2. Check the **Include in primary key** option.

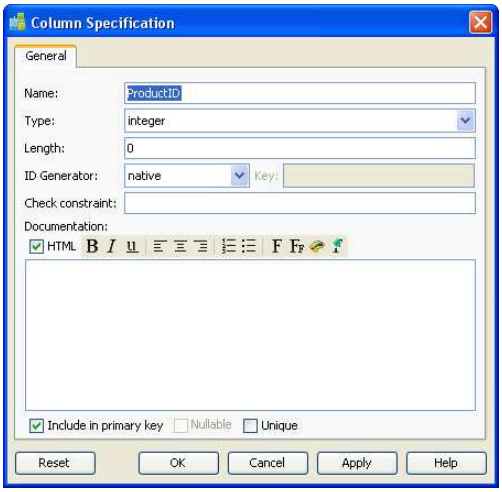


Figure 4.20 - Column Specification dialog

Note

If you assign a primary key to a column of an entity, DB-VA will automatically add a foreign key column to the related entities.

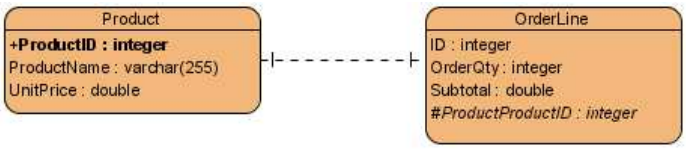


Figure 4.21 - The foreign key will be added automatically

Specifying Index Column

If a relationship with cardinality of many at one end, a corresponding collection class will be used for handling its multiple cardinality. DB-VA allows you to specify an index column to sort the collection.

1. Right-click on the connection line, click **Open Specification** from the pop-up menu.

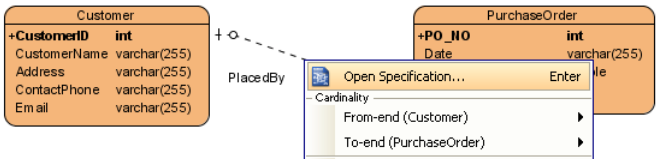


Figure 4.22 - Open relationship specification

Relationship Specification dialog box is displayed.

2. Check the option for **Ordered**.
3. Select the index column from the drop-down menu of **Index column**, click **OK**.

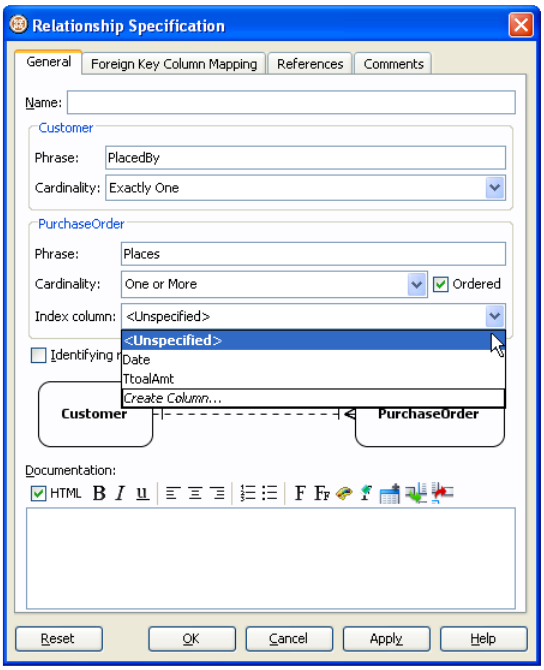


Figure 4.23 - Select the index column

Note

You can select **Create Column** from the drop-down menu to create a new index column for sorting.

Using the ID Generator

As the primary key is unique, DB-VA provides you with the generation of primary key. The ID Generator is specialized for generating a primary key value at runtime.

1. Right-click on the primary key of an entity, select **Open Specification** from the pop-up menu.

Figure 4.24 - Open the column specification

Column Specification of the primary key is displayed.

2. Select the ID generator from the drop-down menu of **ID Generator**, click **OK** to confirm setting.

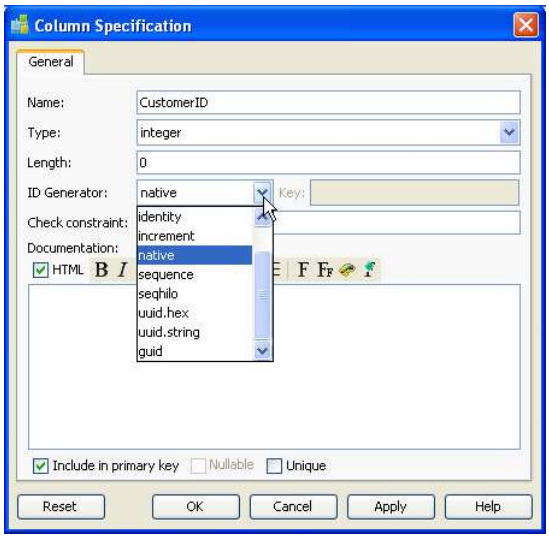


Figure 4.25 - Select the ID Generator

Note

If the ID Generator is specified as either sequence, seqhilo or hilo, you have to enter the key for the sequence/table name.

Defining Discriminator

In generalization, the superclass distributes its commonalities to a group of similar subclasses. The subclass inherits all superclassï¿½ï¿½s attributes and it may contain specific attributes. DB-VA combines the entities within the hierarchy into one single entity containing all the attributes and a discriminator column for using table per class hierarchy as the inheritance strategy. The discriminator contains a unique value which is used for identifying the entity which hierarchy it belongs to.

DB-VA allows you to define the discriminator in the entity and the discriminator value in the classes.

Defining Discriminator Column for Entity

You can add a new column acting as the discriminator column for an entity.

1. Right-click on an entity, select **New Column**.

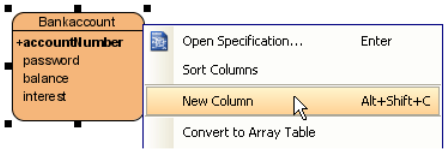


Figure 4.26 - Create a column

2. Enter the name and type for the discriminator in the form of "**discriminator_name: type**".

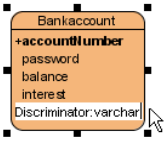


Figure 4.27 - Enter the column name and data type

3. Right-click on the entity, select **Open Specification...**.

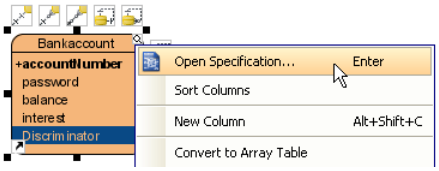


Figure 4.28 - Open the column specification

Entity Specification dialog box is displayed.

4. Select the desired column from the drop-down menu of **Discriminator Column**, click **OK** to confirm setting.

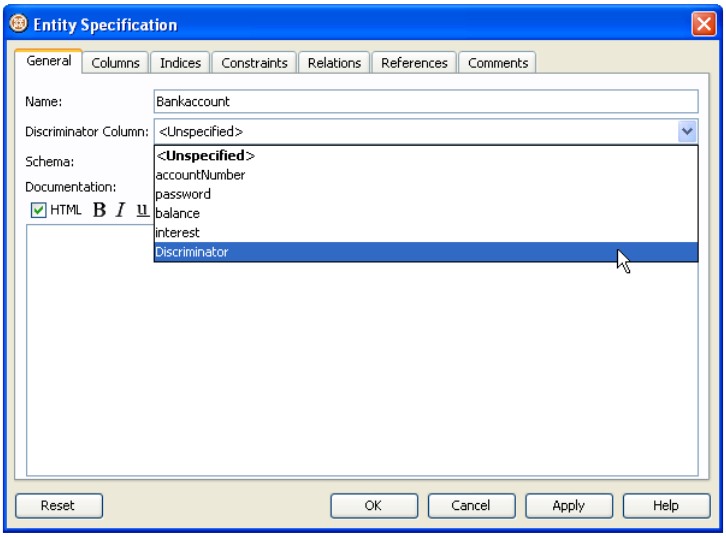


Figure 4.29 - Select the Discriminator Column

Defining Discriminator Value for Class

You can specify the discriminator value for each sub-class.

1. Right-click on the relative sub-class for adding discriminator, select **ORM > ORM Class Details...**from the pop-up menu. The **Class Specification** dialog box showing the **ORM Class Detail** tab is displayed.

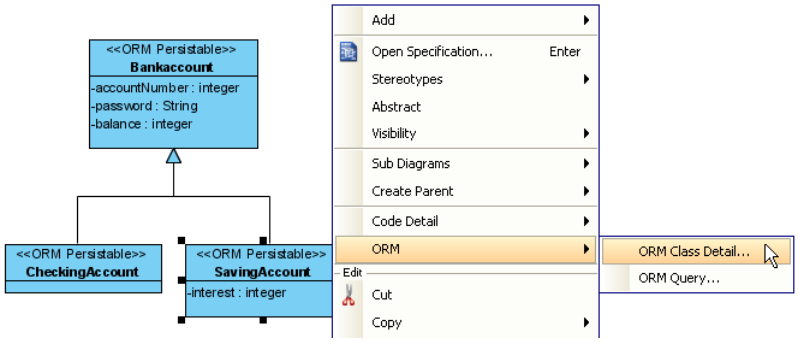


Figure 4.30 - Open the ORM Class Detail

2. Enter the discriminator value for identifying the sub-class.

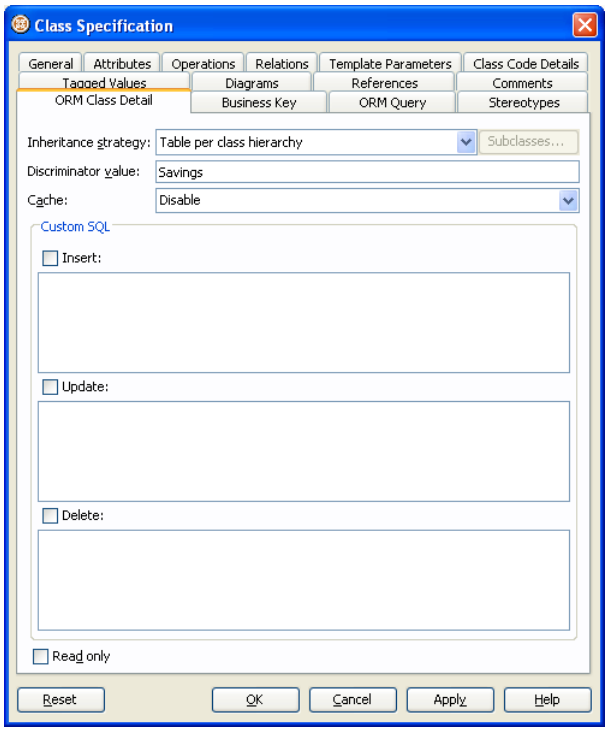


Figure 4.31 - Class Specification dialog (ORM Class Detail Tab)

Creating an Array Table

In a one-to-many relationship, a collection is used for handling the multiple objects such that it is simpler to retrieve each object from the collection one by one.

DB-VA promotes the idea of Array Table which allows users to retrieve objects in the form of primitive array, instead of a collection when handling a data column with cardinality of many.

DB-VA allows you to create an array table in the entity and define an array type in the classes.

Defining an Array Table

You can create an Array Table for the Entity with a column containing more than one instance of data.

1. Create a one-to-many relationship between the entity and one of its columns that may contain more than one instance of data.

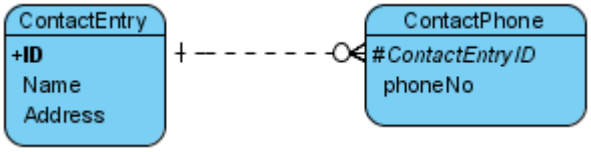


Figure 4.32 - Entities with One-to-many relationship

In the above case, the phonebook has a contact entry for each contact person. Each contact person may have more than one phone numbers. A one-to-many relationship between contact entry and contact phone can be built.

2. Right-click on the entity for the data column with cardinality of many, select **Convert to Array Table** from the pop-up menu.

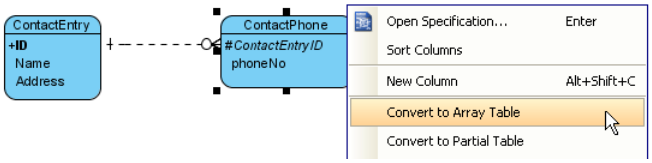


Figure 4.33 - Convert to Array Table

3. A warning message will be displayed, showing that the listed constraints are not satisfied for converting to array table. Click **Yes** to let DB-VA to resolve the constraints automatically. Click **No** to cancel the conversion to array table.

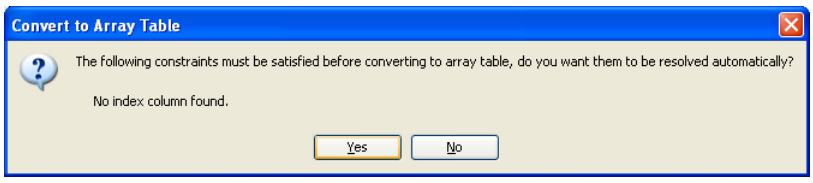


Figure 4.34 - Warning message for no index column

The conversion to Array Table is completed and the entity for the data column is stereotyped as Array Table.

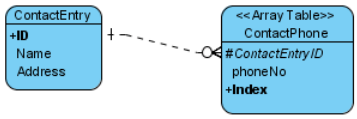


Figure 4.35 - Array Table created

Defining an Array Type for Attribute in Class

A class with an attribute of array type modifier means that the attribute may contain more than one data; thus it implies the idea of Array Table.

You can define the array type for the attribute in one of the two ways:

- Using Inline Editing
    1. Right-click on a class, click **Add > Attribute**.

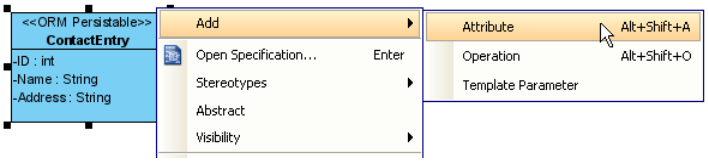


Figure 4.36 - Add an attribute by click on popup menu

    2. Enter the name and type for the attribute in the form of "**attribute_name :type[]",** the sign, **"[ ]"** indicates the attribute is an array.

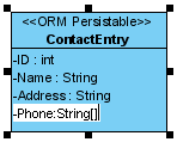


Figure 4.37 - Enter the attribute name and data types

- Using **Class Specification** dialog box
    1. Right-click on a class, click **Open Specification**.

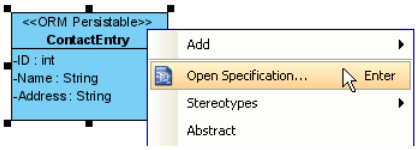


Figure 4.38 - Open the class specification

    - The **Class Specification** dialog box is displayed

    1. Click **Attribute** Tab, click **Add**.

       Attribute Specification is displayed.

2. Enter attribute name and type, select [] from the drop-down menu of **Type modifier**, then click **OK** to confirm setting.

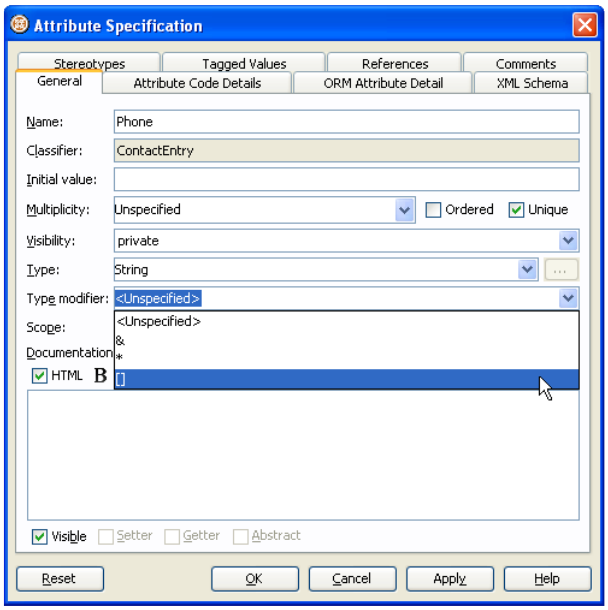


Figure 4.39 - Select the Type modifier

Creating a Partial Table

In a one-to-one identifying relationship, an entity may be a subordinate of the related entity; that is, the subordinate entity has columns which also belong to its superior entity in the real world situation.

DB-VA promotes the idea of Split Table with stereotype of Partial which allows developers to optimize the size of database, and minimizes the redundant persistent classes for handling one-to-one identifying relationship. In order to reduce the risk of appending a new column to an existing database table, Split table supports developers to add new columns to the partial table with a one-to-one identifying relationship linked to the existing table.

DB-VA allows you to split the entity into two and convert the subordinate entity to be a Partial Table in a one-to-one identifying relationship.

Splitting Table

You can split an entity into two associated with a one-to-one identifying relationship.

1. You can activate the **Split Table** dialog box in one of the two ways:
   - Using Pop-up Menu
     1. Right-click an entity, select **Split Table**.

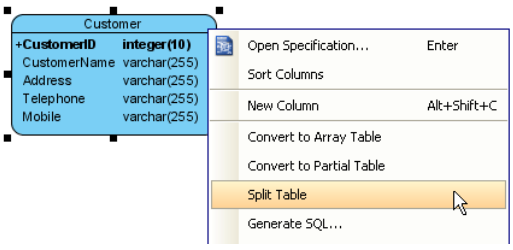


Figure 4.40 - Select "Split Table" in popup menu

- Using Resource-Centric Interface
    1. Click on an entity, a group of valid editing resources are displayed around the entity.
    2. Click the resource of "**One-to-One Relationship - > Partial Table**".

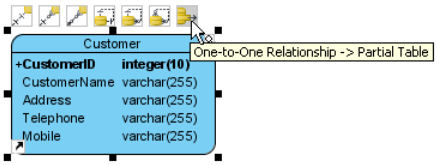


Figure 4.41 - Click on "One-to-one Relationship -> Partial Table" resource

2. **Split Table** dialog box is displayed.

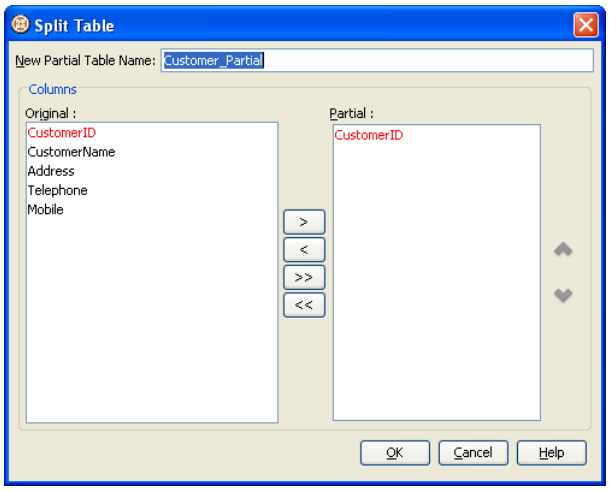


Figure 4.42 - Split Table dialog

3. Edit the **New Partial Table Name,** select the columns from the list of **Original** to **Partial**, and click **OK**.

   An entity stereotyped as **<<Partial>>** is created.

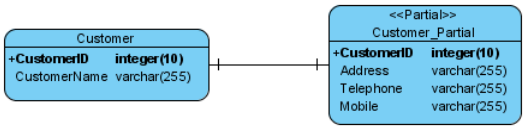


Figure 4.43 - Partial Table is created

Converting to a Partial Table

You can convert an entity to a Partial Table in a one-to-one identifying relationship.

1. Right-click on the entity, select **Convert to Partial Table** from the pop-up menu.

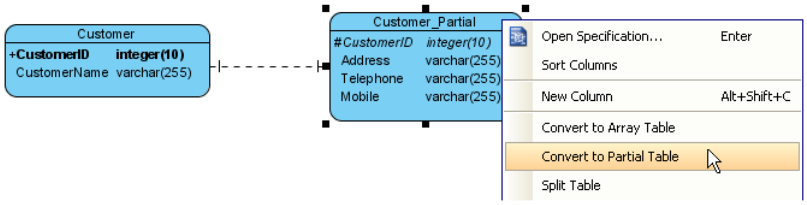


Figure 4.44 - Convert to Partial Table

The entity is stereotyped as **<<Partial>>.**

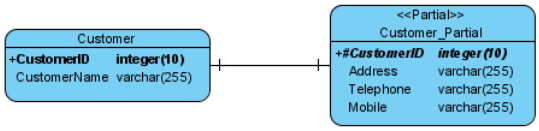


Figure 4.45 - Converted to a Partial Table

Copying SQL Statements from Tables

DB-VA provides function of copying SQL statements from the ERD entities. It allows the developers to copy the SQL statements from the entity relationship diagram easily such that developers can use and modify the SQL statement on the database directly.

In order to copy the SQL statement, you must configure the database setting in advance as DB-VA will generate the SQL statements according to the default database server type.

1. To configure database connection, on the menu, click **Tools > Object-Relational Mapping (ORM) > Database Configuration...**.

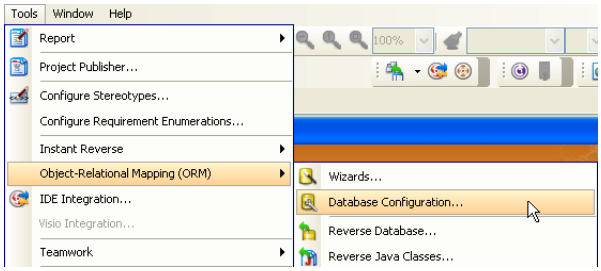


Figure 4.46 - To open the Database Configuration

Database Configuration dialog box will be displayed. To configure the database, refer to the descriptions of the Database Configuration section in the Working with DB Visual ARCHITECT chapter for more information.

Note

In DB-VA for IDE, you are allowed to configure the database connection by using **Modeling > ORM > Database Configuration** to display the **Database Configuration** dialog box.

Note

DB-VA will only provide you the function of copying SQL if the default database connection is set.

Note

If there are multiple database settings, DB-VA will generate SQL statements for all these database servers.

Example:

There are two database settings selected in the DB-VA environment.

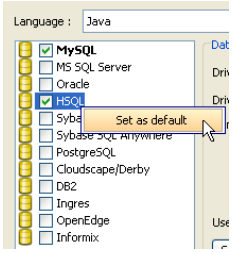


Figure 4.47 - Two database settings Selected

2. Right-click on the ERD, select **Generate SQL...** from the pop-up menu.

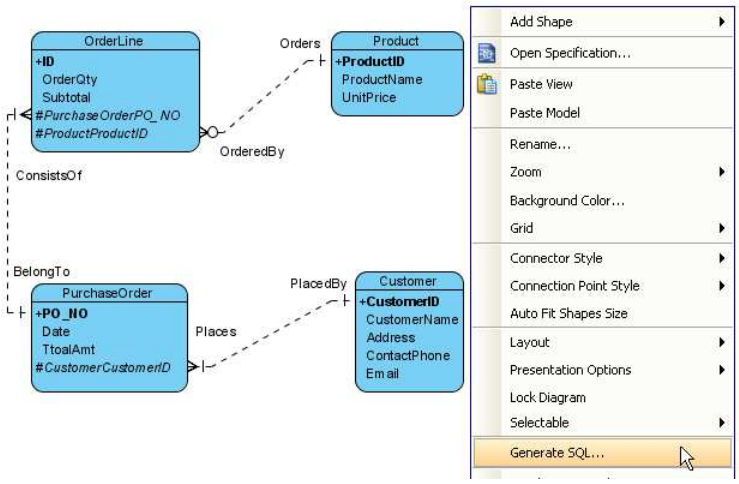


Figure 4.48 - Select generate SQL in popup menu

3. **Generate SQL** dialog box is displayed, select the database server from the drop-down menu of **Database**, the corresponding SQL statements will be displayed accordingly.

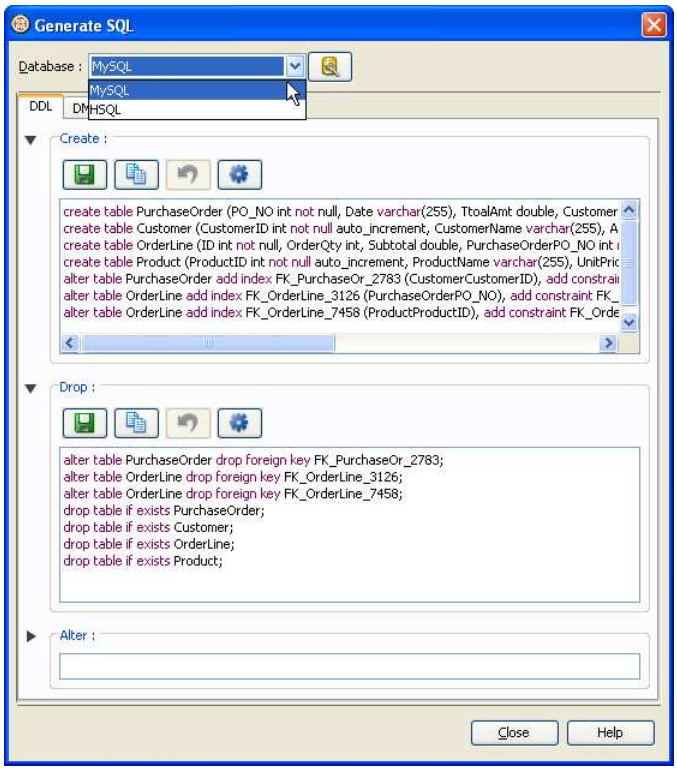


Figure 4.49 - Generate SQL dialog

You are allowed to copy the SQL statements from the **Generate SQL** dialog box.

Note

You can select **Create Table(s), Drop Table(s), Select**, **Insert**, **Update** and **Delete** from the **Generate SQL** dialog to directly copy the SQL statements to clipboard.

Copying SQL Statements from Specified Scope

You can specify the scope on the ERD for DB-VA to generate the SQL statements.

You can specify one of the three scopes:

- All entities on the ERD
    1. Click on an entity.
    2. On the menu, select **Edit > Select All of Same Type**.
    3. Right-click on the entity, select **Generate SQL...**.

    As copying SQL without specifying scope, SQL statements will be generated for all components including both entities and relationships on the ERD.

    Example:

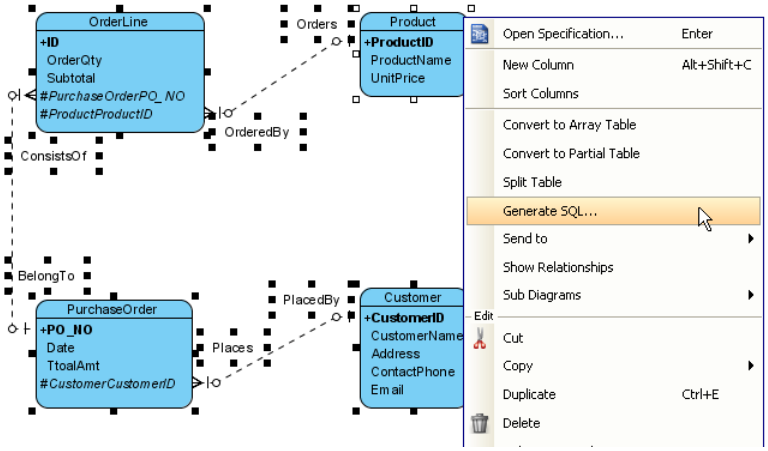


Figure 4.50 - Generate SQL for all entity

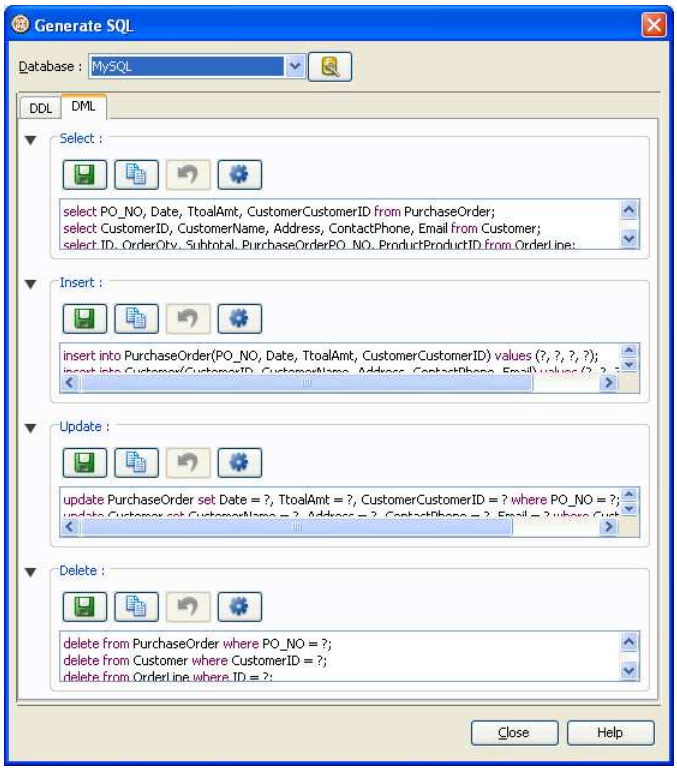


Figure 4.51 - The generated SQL for all entities

- Multiple entities and connection lines on the ERD
    1. Select several entities and relationships on the ERD, right-click on the diagram pane, select **Copy SQL** from the pop-up menu.

As **generate SQL** with specifying a particular scope, SQL statements will be generated only for the components included in the specified scope.

Example:

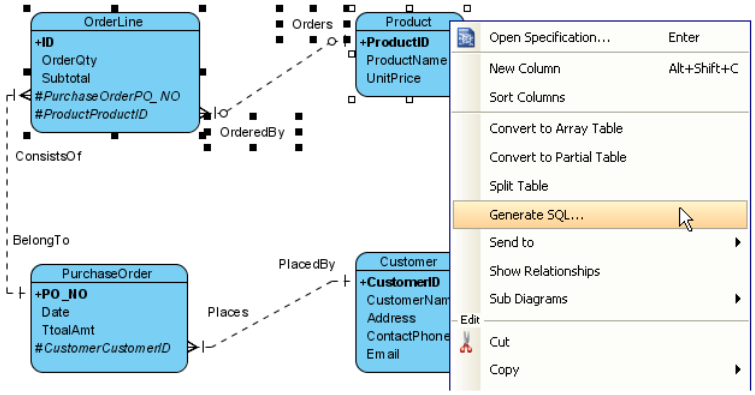


Figure 4.52 - Generate SQL for selected entities

- Connection lines on the ERD
    1. Select connection line, right-click on the diagram pane, select **Generate SQL...** from the pop-up menu.

As **Generate SQL...** with connection lines, SQL statements for Create Constraint(s) and Drop Constraints(s) will be generated.

Example:

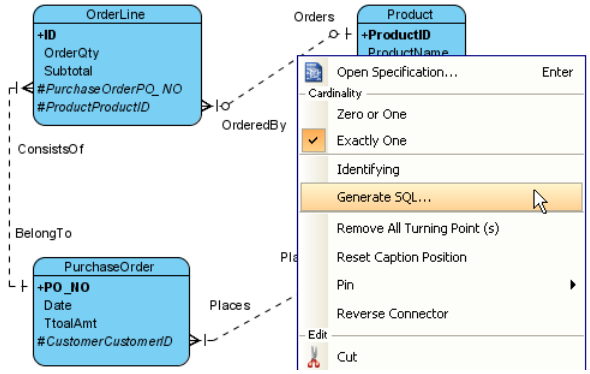


Figure 4.53 - Generate connection line's constraint

Create Constraint :

```
alter table 'OrderLine' add index 'FK_OrderLine_1969'
('PurchaseOrderPO_NO'), add constraint 'FK_OrderLine_1969' foreign key
('PurchaseOrderPO_NO') references 'PurchaseOrder' ('PO_NO');
```

Drop Constraint :

```
alter table 'OrderLine' drop foreign key 'FK_OrderLine_1969';
```

5 Reverse Engineering Classes and Database

Chapter 5 - Reverse Engineering Classes and Databases

DB Visual ARCHITECT (DB-VA) allows you not only to reverse engineer the existing classes, including Java classes and Hibernate models into object models, but also to reverse engineer the existing database into data models. This chapter shows you how to reverse engineer the existing classes and database by the reverse facility and the ORM pane.

In this chapter:

Introduction

Apart from generating persistent classes and database, DB-VA also supports reverse engineering the existing classes and database into object models and data models respectively.

By supporting reverse engineering, object and data models will be generated which assists you in re-designing the application by visual modeling. The following sections show you how to reverse engineer existing classes and database to object and data models respectively.

Reverse Engineering Classes

DB-VA allows you to reverse engineer the existing classes, including Java classes and Hibernate models into object models. The reversed object models are stereotyped with ORM-Persistable. You can thus further model your application with the reversed models and generate the persistent classes and persistence layer after modification.

Reverse Engineering Java Classes to Object Model

DB-VA allows you to reverse engineer the Java classes into object model with ORM-Persistable stereotyped.

To reverse engineer Java classes:

1. On the menu, click **Tools > Object-Relational Mapping (ORM) > Reverse Java Classes...**.

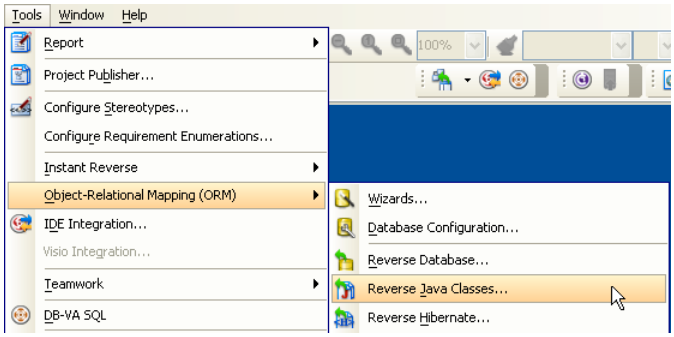


Figure 5.1 - To reverse Java Classes

The **Reverse Java Classes** dialog box is displayed.

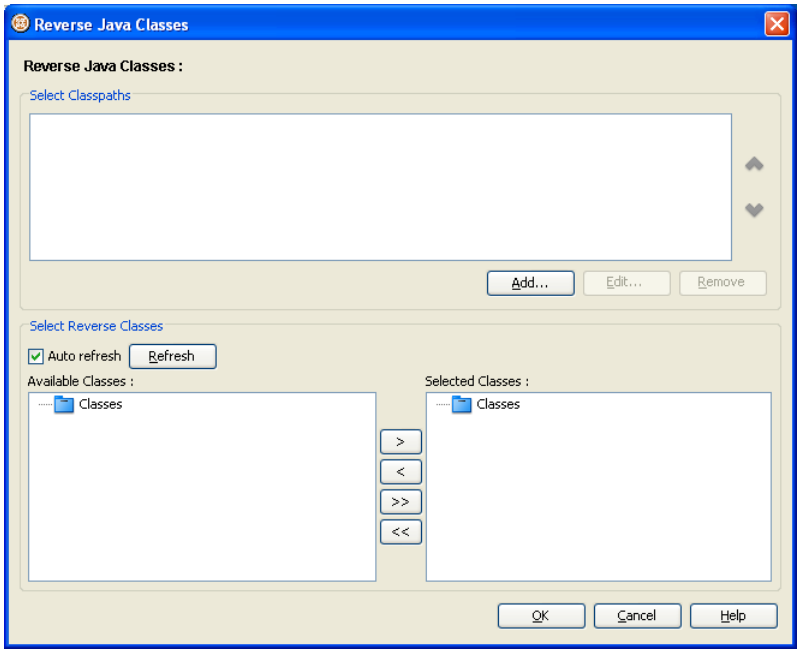


Figure 5.2 - Reverse Java Classes dialog

2. Click **Add...** to select the classpath of the Java classes to be reversed. The classpath can be a folder, zip file or jar file. After finished selecting the classpath, the classpath is added to the list of **Select Classpaths**, and the classes identified from the classpath are shown in the list of **Available Classes**.

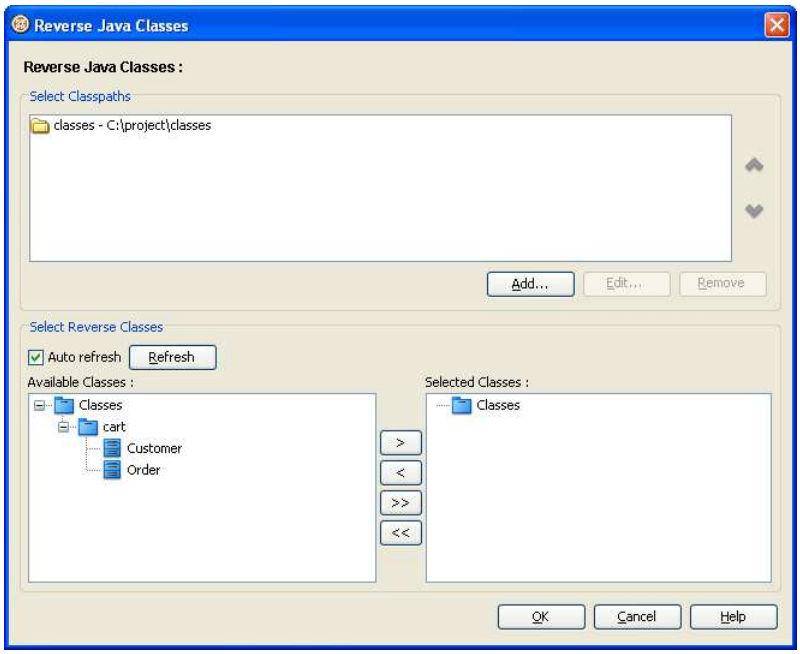


Figure 5.3 - The classes in the selected path are shown in the dialog

3. Select the desired classes by using the list of buttons between the list of **Available Classes** and **Selected Classes**.

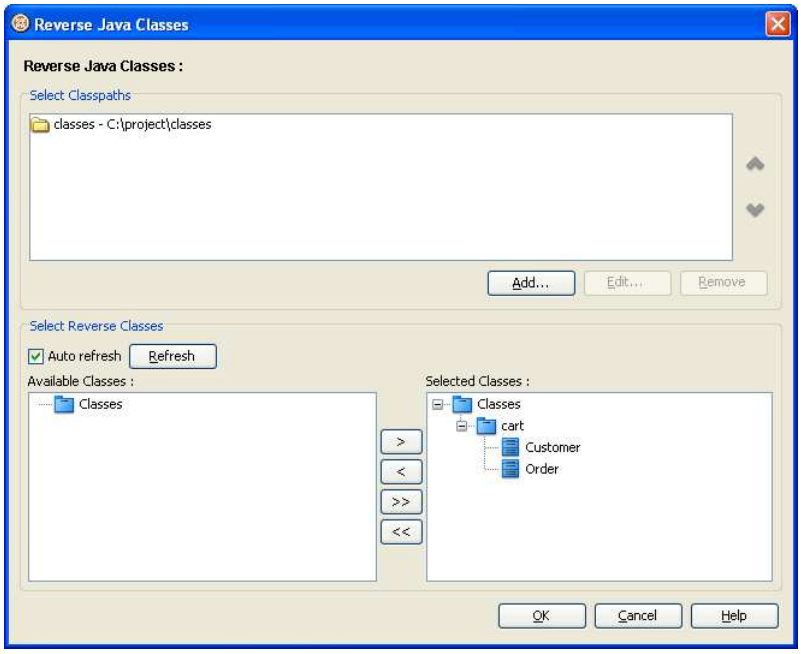


Figure 5.4 - Select the classes for reverse

4. Click **OK**. The selected classes are reverse engineered to class models which can be found under the **Model** tree.

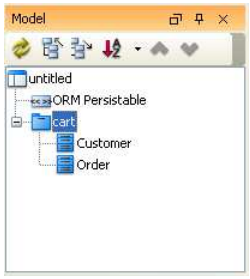


Figure 5.5 - The reversed classes model show in Model pane

To work on the reversed class models, simply add the reversed models to the class diagram:

1. Create a new class diagram by using the **New Class Diagram** icon.
2. Select the classes from the **Model** tree, drag the classes to the newly created class diagram.

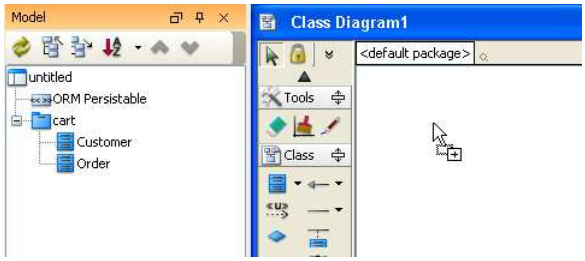


Figure 5.6 - Drag the classes to Class Diagram

The classes are added to the class diagram accordingly. The classes shown on the class diagram are stereotyped as ORM Persistable; meaning that the Java classes are reversed engineered to ORM Persistable classes supporting the object relational mapping.

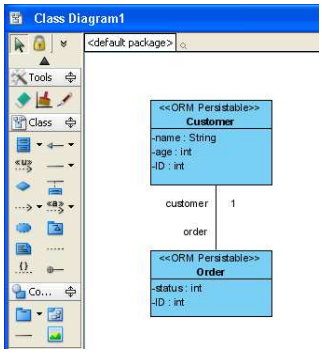


Figure 5.7 - The reversed classes are shown on the Class Diagram

Reverse Engineering Hibernate Model to Object Model

DB-VA not only allows you to reverse engineer the Java classes, but also the hibernate model to object model with ORM-Persistable stereotyped. DB-VA also reverse engineer the database configuration as the database setting is defined in the hibernate model.

To reverse engineer Hibernate model:

1. On the menu, click **Tools > Object-Relational Mapping (ORM) > Reverse Hibernate...**.

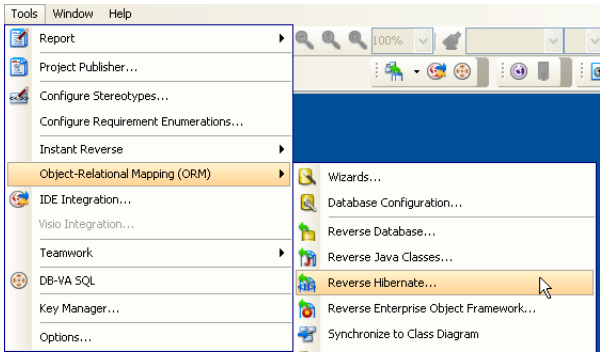


Figure 5.8 - Reverse Hibernat

The **Reverse Hibernate Model** dialog box is displayed.

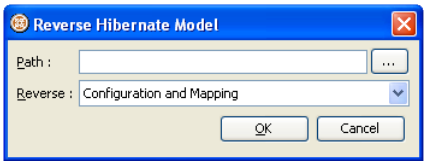


Figure 5.9 - Reverse Hibernate Model dialog

2. Select the path of the Hibernate xml files by using the ... button.
3. Select the type of reverse engineering to be performed from the drop-down menu of Reverse, either **Configuration and Mapping**, **Configuration only** or **Mapping only**.

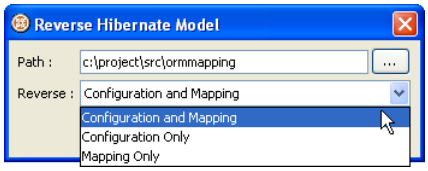


Figure 5.10 - Select the type of reverse engineering

4. Click **OK**. The hibernate model are reverse engineered to class models and entities which can be found under the **Model** tree.

Figure 5.11 - The reversed model is shown in Model pane

After reverse engineered the hibernate model, you can use an ORM diagram to view the mapping between the reversed classes and entities. For more information on ORM diagram, refer to the description of Showing Mapping between Object and Data Models by ORM Diagram section.

1. Create a new ORM diagram by using the **New ORM Diagram** icon.
2. Add the reversed classes to the ORM diagram by dragging the class models from the **Model** tree to the ORM diagram.

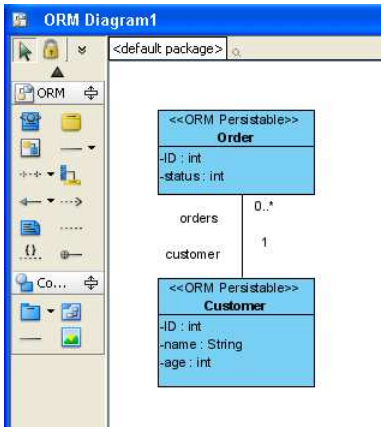


Figure 5.12 - Drag the Class Model to ORM Diagram

3. Drag the entities from the **Model** tree to the ORM diagram to add the entities to the ORM diagram.

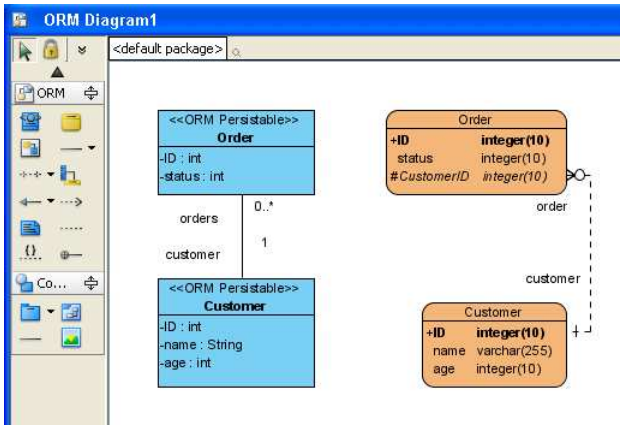


Figure 5.13 - Drag the entities to ORM Diagram

4. Right-click the ORM diagram, select **View > Attribute Mapping** from the pop-up menu.

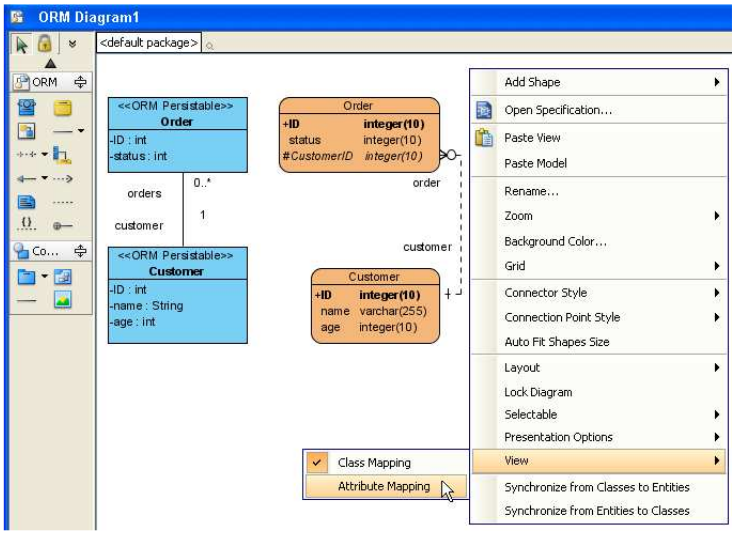


Figure 5.14 - Switch to Attribute mapping view

The mapping between the attributes of class models and columns of entities are shown.

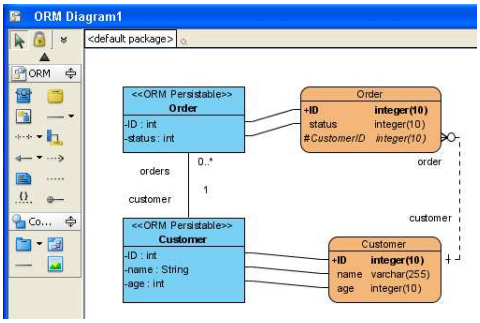


Figure 5.15 - ORM Diagram in attribute mapping view

You can also check the reversed engineered database connection by the following steps:

1. Click the **Database Configuration** icon, to open the **Database Configuration** dialog box.

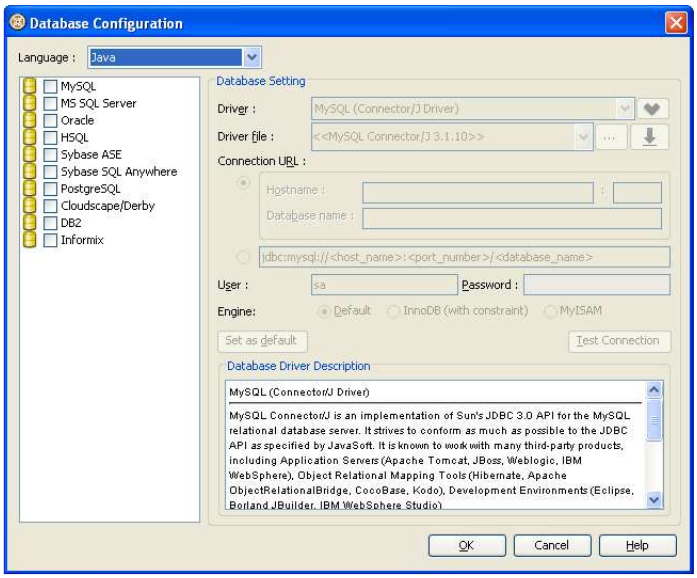


Figure 5.16 - Database Configuration dialog

2. Select the connected database from the **Database Configuration** dialog box, the **Database Setting** is shown which shows the database configuration has been reversed successfully.

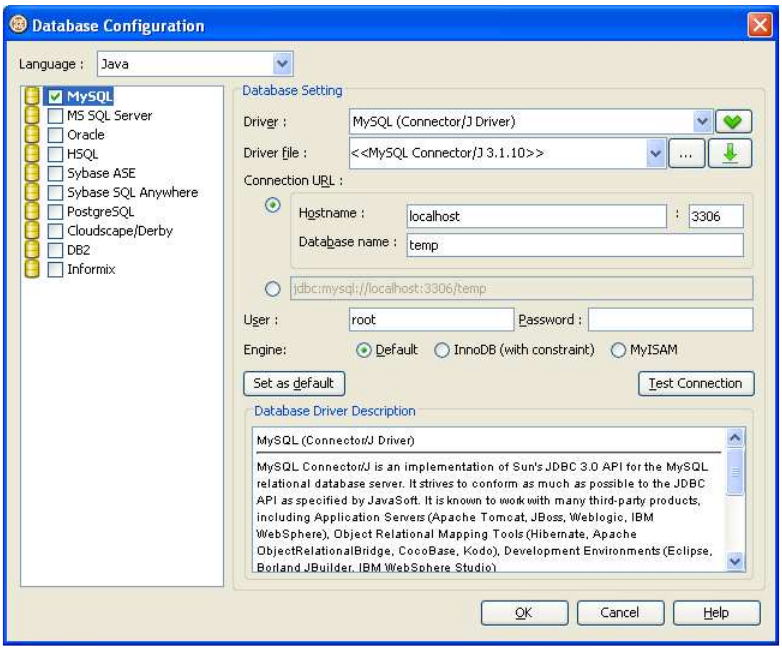


Figure 5.17 - The database configuration reversed successfully

Using ORM Pane

DB-VA provides you an ORM pane to generate persistent model and entity from an existing object model in Java classes and database respectively. Using the ORM pane, the existing object model and database will be transformed to ORM-Persistable class and entity; you can further develop the ORM-Persistable classes and entities by adding the desired classes and entities to the class diagram and entity relationship diagram respectively.

The ORM pane provides two views, including Class View and Database View. The class view allows you to transform the existing object model to class model while the database view allows you to transform the existing database to entity.

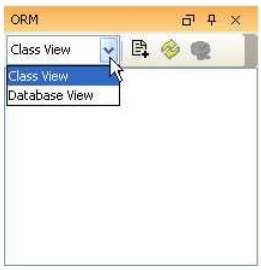


Figure 5.18 - ORM Pane

Reverse Engineering Java Classes by Class View

As the class view of the ORM pane supports the transformation of the existing object model into ORM-Persistable class, you are allowed to further your development based on the transformed object model.

1. Select the **Class View** of the **ORM** pane.
2. Click the **Classpath Configuration** icon.

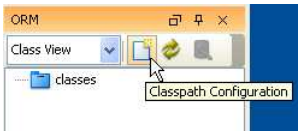


Figure 5.19 - To open the classpath configuration

The **Select Classpaths** dialog box is displayed.

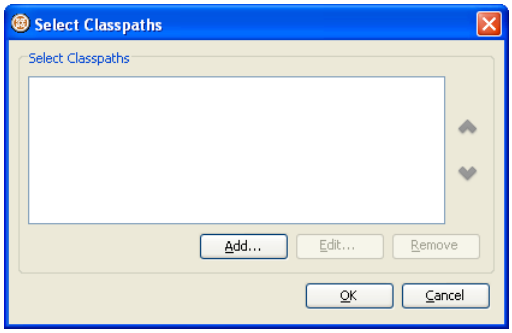


Figure 5.20 - Select Classpaths dialog

3. Click **Add...** button to select the desired classpath.

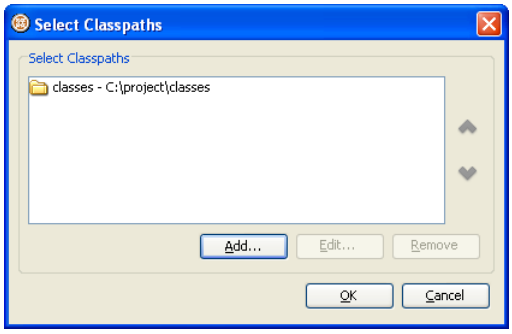


Figure 5.21 - Added classpath to the dialog

All the available classes found from the specified classpath(s) are transformed and shown on the **ORM** pane.

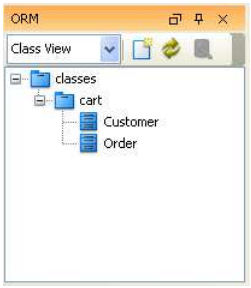


Figure 5.22 - The classes in the classpath will shown in ORM pane

4. Create a new class diagram by using the **New Class Diagram** icon.
5. Select the desired classes and drag to the class diagram.

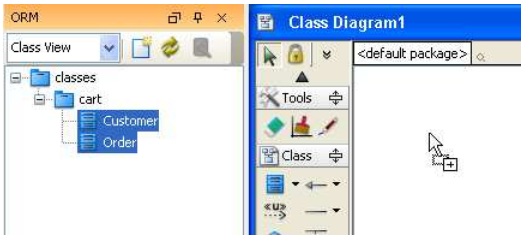


Figure 5.23 - Drag the classes to Class Diagram

The classes are added to the class diagram such that you can further develop the model by using the visual modeling feature.

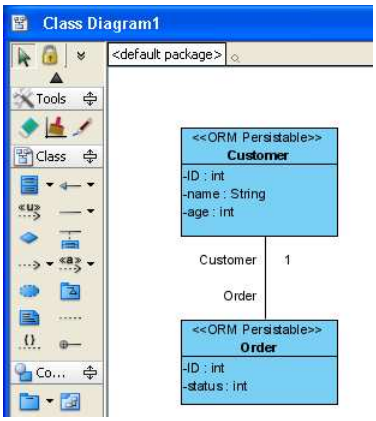


Figure 5.24 - The reversed classes shown in Class Diagram

Reverse Engineering Relational Database

DB-VA supports reverse engineering the existing database to data models. As DB-VA automates object-relational mapping, object model can thus be generated from the data model reverse engineered from the existing database.

Using Reverse Database Facility

You can create an Entity Relationship Diagram by reverse engineering an existing relational database.

- On the menu, click **Tools > Object-Relational Mapping (ORM) > Reverse Database...**.

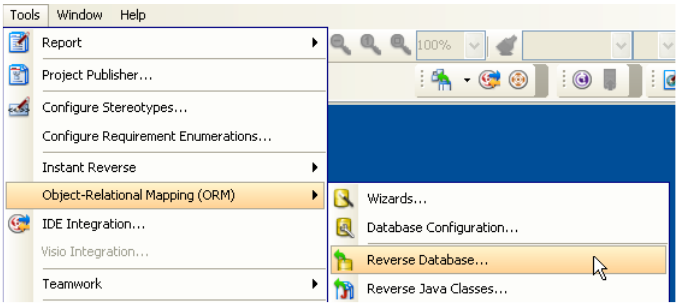


Figure 5.25 - Reverse Database

The **Database to Data Model** dialog box is displayed.

Step 1: Select Language

Select the language of the project from the drop-down menu, either **Java**, **C#**, **PHP** or **Enterprise Object Framework**, and then click **Next >** to proceed to Step 2.

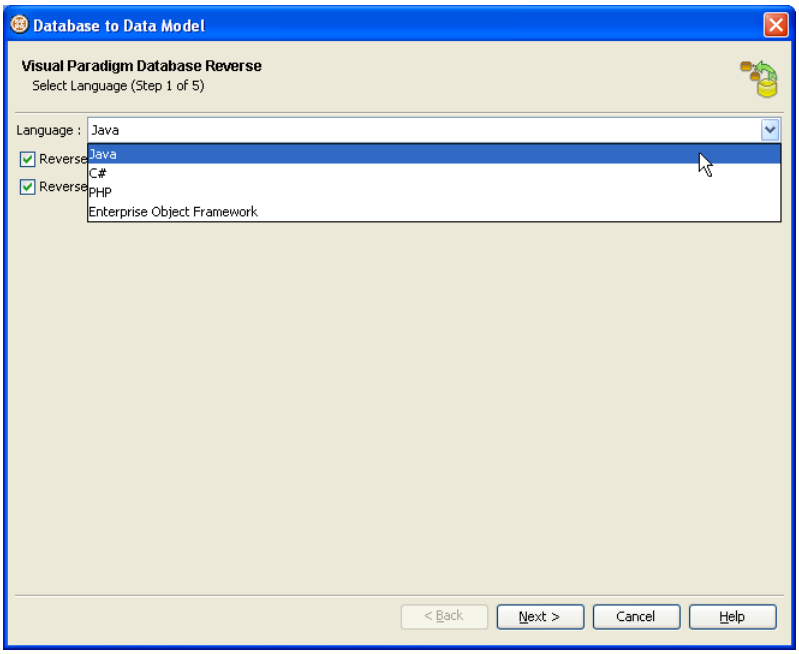


Figure 5.26 - Select the language of the project

Step 2: Database Configuration

You can configure the database connection for the desired database to be reversed.

1. You are asked to define the database configuration. To configure the database settings, refer to the descriptions of the Database Configuration section in the Working with DB Visual ARCHITECT chapter.

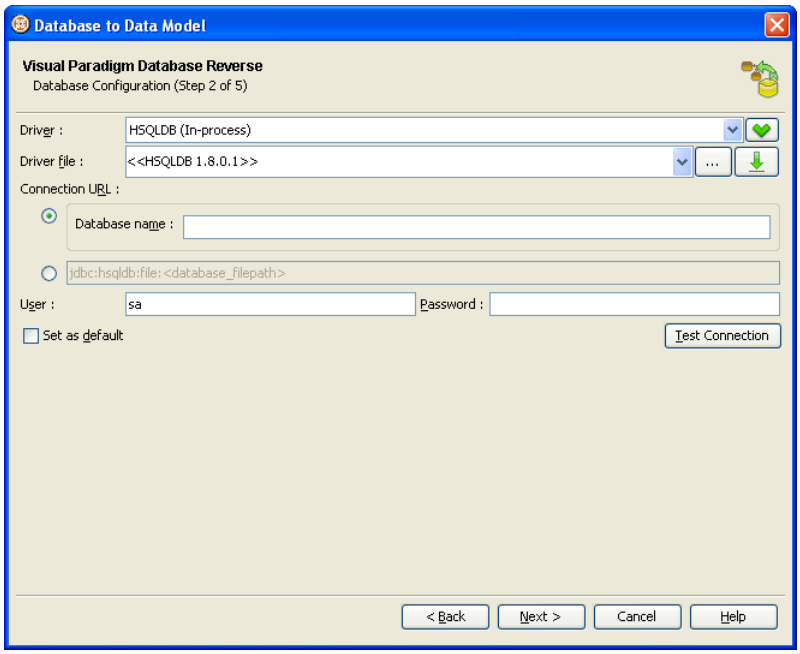


Figure 5.27 - Database Configuration

2. Click **Next>,** go to Step 3 of Reverse Database.

Step 3: Selecting Tables

All the available tables found from the connected database are listed.

1. Select the tables that you want to reverse to Data Model.
2. Click **Finish**.

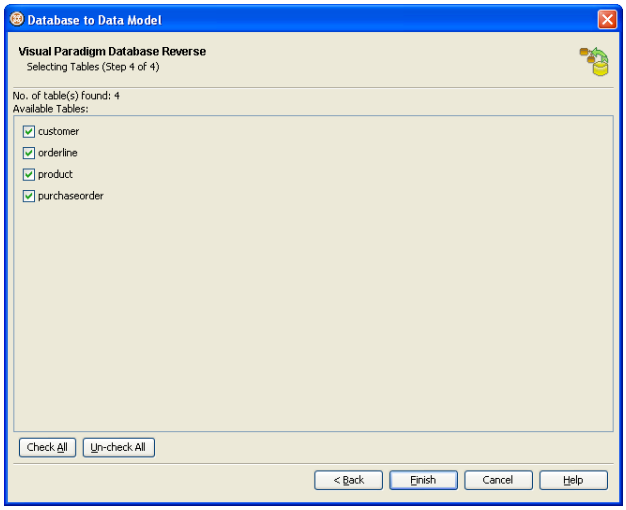


Figure 5.28 - The available tables in the database

An Entity Relationship Diagram is automatically generated and displayed. It can be found under the **Diagram Navigator**.

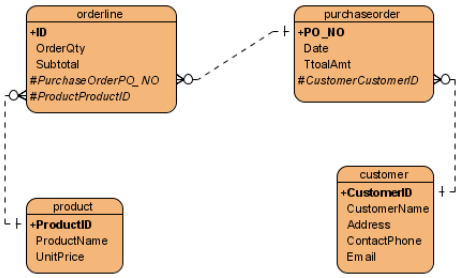


Figure 5.29 - The reversed ERD

Using ORM Pane

DB-VA provides you an ORM pane to generate persistent model and entity from an existing object model in Java classes and database respectively. Using the ORM pane, the existing object model and database will be transformed to ORM-Persistable class and entity; you can further develop the ORM-Persistable classes and entities by adding the desired classes and entities to the class diagram and entity relationship diagram respectively.

The ORM pane provides two views, including Class View and Database View. The class view allows you to transform the existing object model to class model while the database view allows you to transform the existing database to entity.

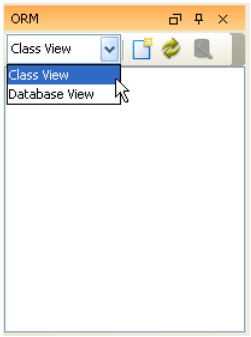


Figure 5.30 - ORM Pane

Reverse Engineering Database by Database View

As the database view of the ORM pane supports the transformation of the existing database into entity, you are allowed to alter the database schema by modeling with the entity relationship diagram and exporting to the existing database.

1. Select the **Database View** of the **ORM** pane.

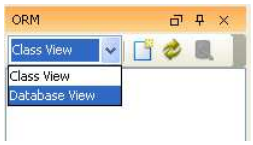


Figure 5.31 - Switch to Database View

2. Click the **Database Configuration** icon.

Figure 5.32 - To open the Database Configuration

The **Database Configuration** dialog box is displayed.

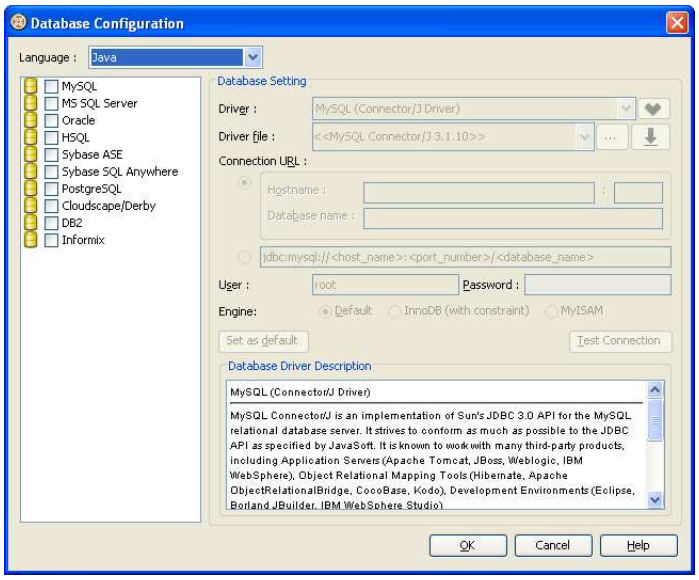


Figure 5.33 - Database Configuration dialog

3. Configure the database connection by using the **Database Configuration** dialog box, refer to the descriptions of the Database Configuration section in the Working with DB Visual ARCHITECT chapter for more information. If the database is successfully connected, the tables of the connected database are transformed into entities and shown on the **ORM** pane.

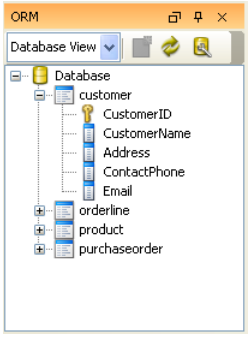


Figure 5.34 - The reversed entity shown in ORM Pane

4. Create a new entity relationship diagram by using the **New Entity Relationship Diagram** icon.
5. Select the desired entities from the **ORM** pane, drag to the entity relationship diagram.

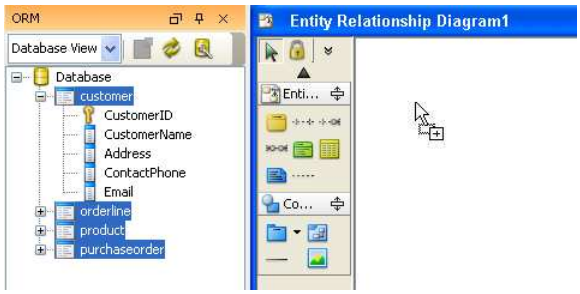
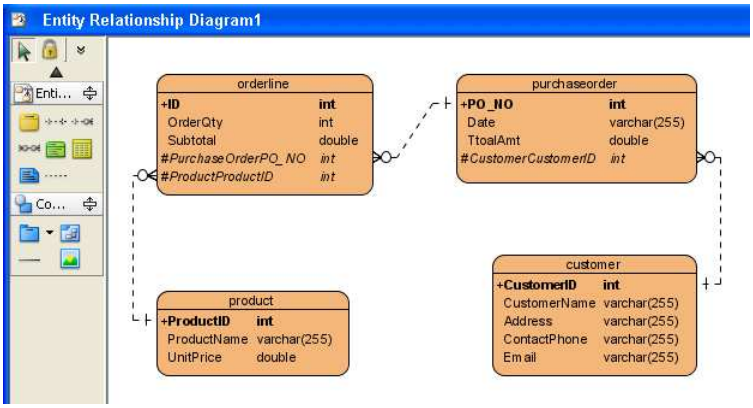


Figure 5.35 - Drag the entities to ERD

The selected entities are added to the class diagram allowing you alter the database schema by visual modeling.



6 Mapping Object Model to Data Model and vice versa

Chapter 6 - Mapping Object Model to Data Model and vice versa

Being an Object-Relational Mapping tool, DB Visual ARCHITECT (DB-VA) automates the mapping between object and data models. This chapter describes how DB-VA maps the object model to data model and vise versa, and introduces the usage of ORM diagram.

In this chapter:

- Introduction
- Mapping Object Model to Data Model
- Mapping Data Model to Object Model
- Showing Mapping by ORM Diagram

Introduction

DB Visual ARCHITECT (DB-VA) supports Object Relational Mapping (ORM) which maps object models to entity relational models and vice versa.

DB-VA helps mapping between Java objects to relational database. It not only preserves the data, but also the state, foreign/primary key mapping, difference in data type and business logic. Thus, you are not required to handle those tedious tasks during software development.

Mapping Object Model to Data Model

This section shows you how DB-VA maps object models to data model.

Mapping Classes to Entities

Object Model can map to Data Model due to the persistent nature of classes. Persistent classes can act as persistent data storage during the application is running. Hence, all persistent classes can map to entities using a one-to-one mapping.

Example:

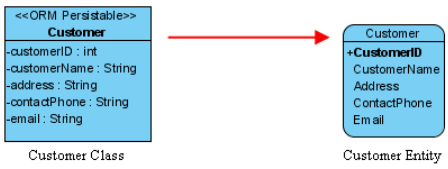


Figure 6.1 - Mapping Classes to Entities

In the above example, the Customer Class is mapped with the Customer Entity as the Customer instance can store the customer information from the Customer Entity.

Mapping Attributes to Columns

Since the persistent classes map to the entities, persistent attributes map to columns accordingly. DB-VA ignores all non-persistent attributes such as derived values.

Example:

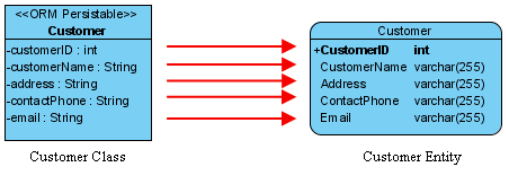


Figure 6.2 - Mapping Attributes to Columns

In the above example, the following table shows the mapping between the attributes of the Customer Class and the columns of the Customer Entity.

Customer Class	Customer Entity
CustomerID	CustomerID
CustomerName	CustomerName
Address	Address
ContactPhone	ContactPhone
Email	Email

Table 6.1

Mapping Data Type

DB-VA automatically maps the persistent attribute type to an appropriate column data type of the database you desired.

Example:

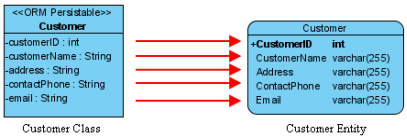


Figure 6.3 - Mapping Data Type

In the above example, the following table shows the mapping between data types

Customer Class	Customer Entity
int	int (10)
String	varchar(255)

Table6.2

A table shows the data type mapping between object model and data model.

Object Model	Data Model
Boolean	Bit (1)
Byte	Tinyint (3)
Byte[]	Binary(1000)
Blob	Blob
Char	Char(1)
Character	Char(1)
String	varchar(255)
Int	Integer(10)
Integer	Integer(10)
Double	Double(10)
Decimal	Integer
Bigdecimal	Decimal(19)
Float	Float(10)
Long	Bigint(19)
Short	Smallint(5)
Date	Date
Time	Time(7)
Timestamp	Timestamp(7)

Table 6.3

Mapping Primary Key

You can map an attribute to a primary key column. When you synchronize the ORM-Persistable Class to the ERD, you will be prompted by a dialog box to select primary key.

- You can select an attribute as the primary key.
- You can let DB-VA generate the primary key automatically.

Example:

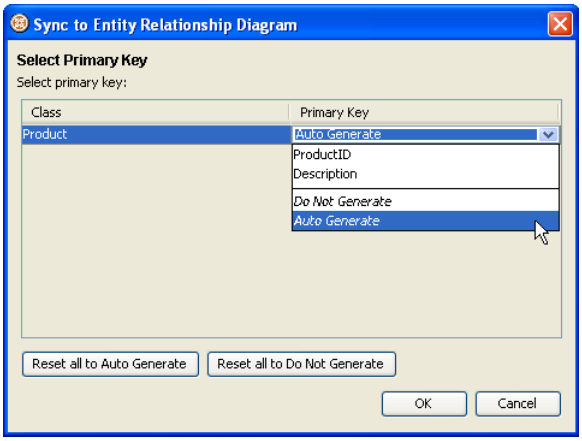


Figure 6.4 - Sync to Entity Relationship Diagram dialog

In the above example, when synchronizing the class of Product to entity relationship diagram, the above dialog box is shown to prompt you to select the primary key of the Product class.

Under the drop-down menu, you can select either one of the attributes of the Product class to be the primary key, or assign DB-VA to generate the primary key automatically, or select "**Do Not Generate**" to leave the generated entity without primary key.

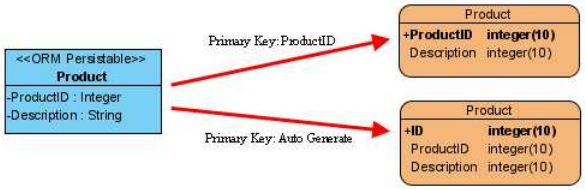


Figure 6.5 - Different between specify a primary key and auto generated

The above diagram shows if you assign ProductID as primary key, the ProductID of the generated entity, Product will become bold; whereas if you select "**Auto Generate**" for the primary key, DB-VA generates an additional attribute ID as the primary key of the Product entity.

Mapping Association

Association represents a binary relationship among classes. Each class of an association has a role. A role name is attached at the end of an association line. DB-VA maps the role name to a phrase of relationship in the data model.

Mapping Aggregation

Aggregation is a stronger form of association. It represents the "has-a" or "part-of" relationship.

Example:

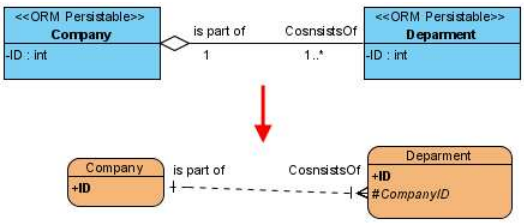


Figure 6.6 - Mapping Aggregation

In the above example, it shows that a company consists of one or more department while a department is a part of the company.

Note

You have to give the role names, "ConsistsOf" and "is Part Of" to the classes, Company and Department in the association respectively in order to proceed to the generation of code.

Mapping Composite Aggregation

Composite aggregation implies exclusive ownership of the "part-of" classes by the "whole" class. It means that parts may be created after a composite is created, meanwhile such parts will be explicitly removed before the destruction of the composite.

Example:

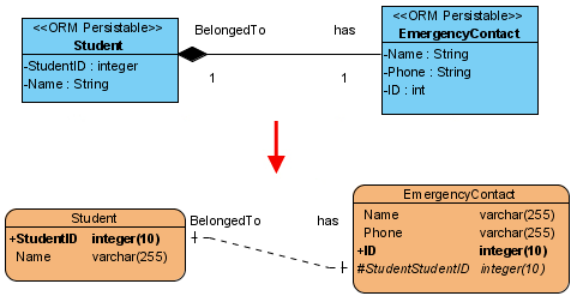


Figure 6.7 - Mapping Composite Aggregation

In the above example, DB-VA performs the Primary/Foreign Key Column Mapping automatically. StudentID of the student entity is added to the entity, EmergencyContact as primary and foreign key column.

Mapping Multiplicity

Multiplicity refers to the number of objects associated with a given object. There are six types of multiplicity commonly found in the association. The following table shows the syntax to express the Multiplicity.

Table shows the syntax expressing the Multiplicity

Type of Multiplicity	Description
0	Zero instance
0..1	Zero or one instances
0..*	Zero or more instances
1	Exactly one instance
1..*	One or more instances
*	Unlimited number of instances

Table 6.4

Example:

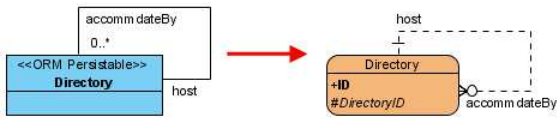


Figure 6.8 - Mapping Multiplicity

In the above example, it shows that a parent directory (role: host) contains zero or more subdirectories (role: accommodated by).

When DB-VA transforms a class with multiplicity of zero, the foreign key of parent entity can be nullable in the child entity. It is illustrated by the DirectoryID.

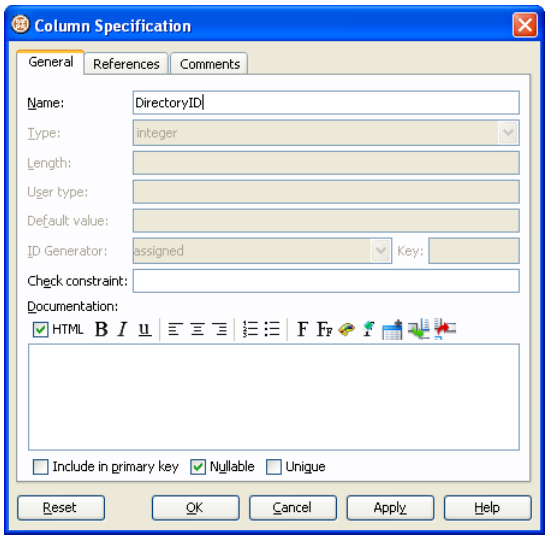


Figure 6.9 - Column Specification of the foreign key

Table shows the typical mapping between Class Diagram and Entity Relationship Diagram.

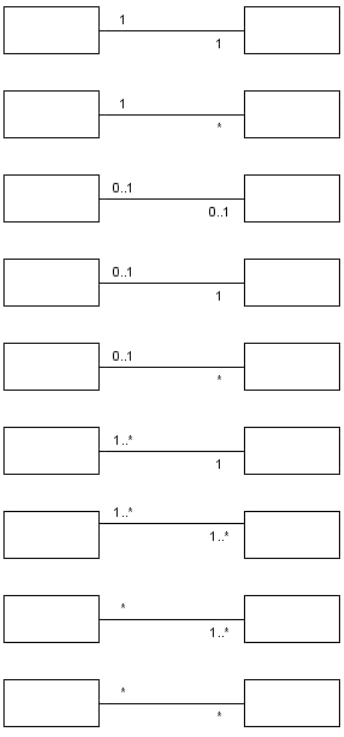
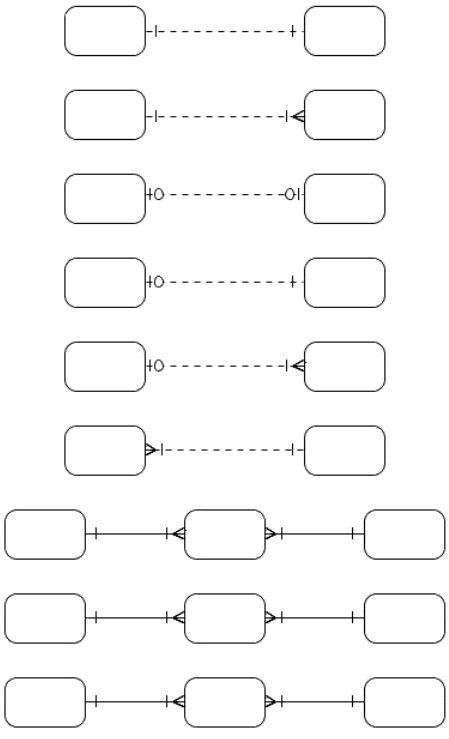
Class Diagram	Entity Relationship Diagram
	

Table 6.5

Mapping Many-to-Many Association

For a many-to-many association between two classes, DB-VA will generate a Link Entity to form two one-to-many relationships in-between two generated entities. The primary keys of the two entities will migrate to the link entity as the primary/foreign keys.

Example:

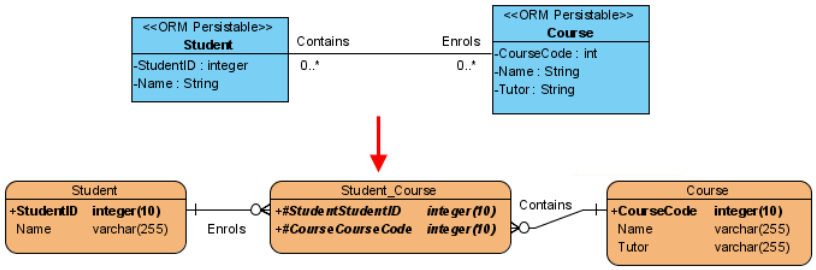


Figure 6.10 - Mapping Many-to-Many Association

In the above example, DB-VA generates the link entity, Student_Course between entities of Student and Course when transforming the many-to-many association.

Mapping Inheritance/Generalization

Generalization distributes the commonalities from the superclass among a group of similar subclasses. The subclass inherits all the superclass's attributes and it may contain specific attributes.

DB-VA provides two strategies for transforming the generalization hierarchy to relational model. The two strategies for transformation are table per class hierarchy and table per subclass.

Using Table per Class Hierarchy Strategy

Transforming generalization hierarchy to relational model with the table per class hierarchy strategy, DB-VA not only combines all the classes within the hierarchy into one single entity containing all the attributes, but also generates a discriminator column to the entity. The discriminator is a unique value identifying the entity which hierarchy it belongs to.

By using the table per class hierarchy strategy, the time used for reading and writing objects can be saved. However, more memory is used for storing data. It is useful if the class will be loaded frequently.

Example:

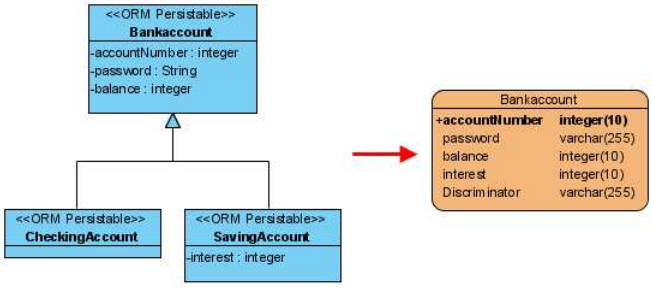


Figure 6.11 - Mapping Inheritance/Generalization using Table per Class Hierarchy Strategy

In the above example, it shows how DB-VA transforms the generalization with CheckingAccount, SavingsAccount and their superclass, BankAccount applying the table per class hierarchy strategy.

Using Table per Subclass Strategy

When DB-VA transforms a generalization hierarchy using the table per subclass strategy to relational model, each subclass will be transformed to an entity with a one-to-one identifying relationship with the entity of the superclass.

By using the table per subclass strategy, it can save memory for storing data. However, it takes time for reading and writing an object among several tables which slows down the speed for accessing the database. It is useful if the class, which contains a large amount of data, is not used frequently.

Example:

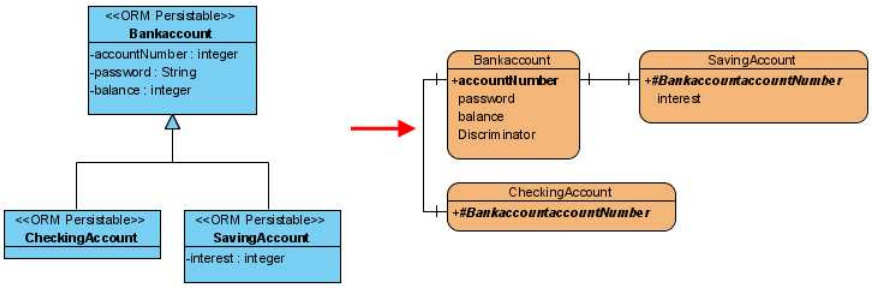


Figure 6.12 - Mapping Inheritance/Generalization using Table per Subclass Strategy

In the above example, it shows how DB-VA transforms the generalization which applies the table per subclass hierarchy strategy to the CheckingAccount and SavingsAccount.

Using Mixed Strategies

DB-VA allows applying the two inheritance strategies to different subclasses within a generalization hierarchy in Java project. By applying different strategies to different subclasses within a generalization hierarchy, DB-VA transforms the generalization hierarchy with respect to the specified inheritance strategies.

Example:

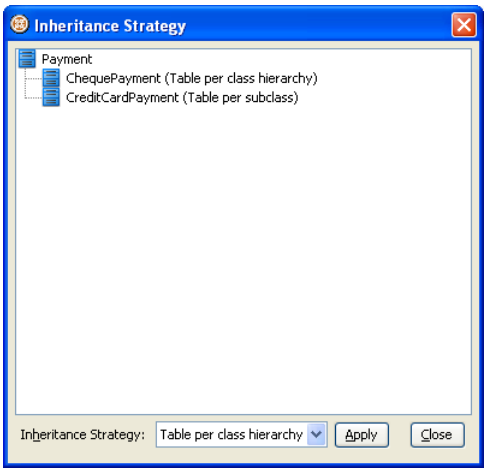


Figure 6.13 - Using Mixed Strategies

Applying the above inheritance strategy to the generalization with the ChequePayment and CreditCardPayment and their superclass, Payment, DB-VA transforms it into two entities as shown below.

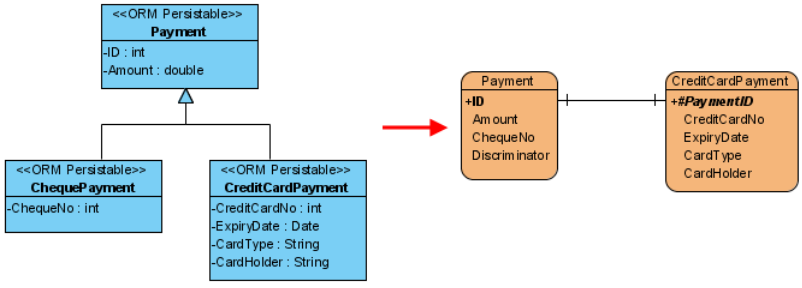


Figure 6.14 - Mapping with mixed Strategies

In the above example, it shows that applying table per hierarchy inheritance strategy will result in combining the attributes of the superclass and subclass into one single entity while table per subclass will result in forming an entity of subclass and a one-to-one identifying relationship between entities of superclass and subclass.

Note

Applying two inheritance strategies to different subclasses within a generalization hierarchy is only available in Java project. If mixed strategies are applied to the generalization hierarchy in .NET project, it will result in error when the generation of code and database.

Mapping Collection of Objects to Array Table

For a persistent class acting as persistent data storage, it may consist of a persistent data containing a collection of objects. DB-VA promotes the use of Array Table to allow users retrieve objects in the form of primitive array.

When DB-VA transforms a class with an attribute of array type modifier, this attribute will be converted to an Array Table automatically. The generated entity and the array table form a one-to-many relationship.

Example:

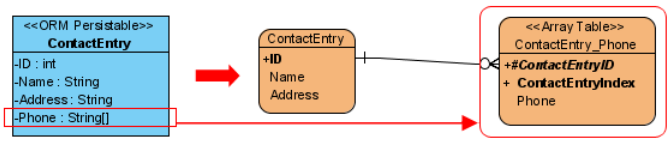


Figure 6.15 - Mapping Collection of Objects to Array Table

In the above example, the phonebook has a contact entry for each contact person. Each contact person may have more than one phone numbers. In order to ease the retrieval of a collection of phone objects, DB-VA converts the phone attribute into a ContactEntry_Phone array table.

Mapping Object Model Terminology

Table shows the shift from object model to data model terminology.

Object Model Term	Data Model Term
Class	Entity
Object	Instance of an entity
Association	Relationship
Generalization	Supertype/subtype
Attribute	Column
Role	Phrase
Multiplicity	Cardinality

Table 6.6

Mapping Data Model to Object Model

This section shows you how DB-VA maps the data model to object model.

Mapping Entities to Classes

All entities map one-to-one to persistent classes in an object model.

Example:

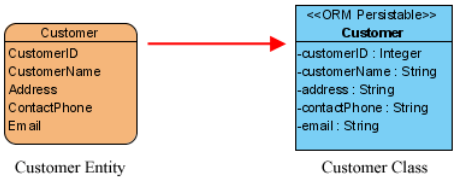


Figure 6.16 - Mapping Entities to Classes

In the above example, the Customer Entity map one-to-one the Customer Class as the Customer instance can store the customer information from the Customer Entity.

Mapping Columns to Attributes

Since all entities map one-to-one to persistent classes in an object model, columns in turn map to attributes in a one-to-one mapping. DB-VA ignores all specialty columns such as computed columns and foreign key columns.

Example:

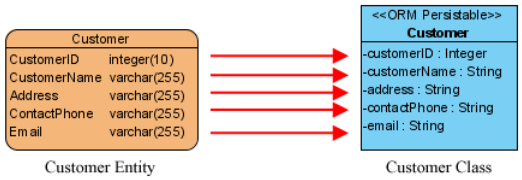


Figure 6.17 - Mapping Columns to Attributes

In the above example, the following table shows the mapping between the columns of the Customer Entity and the attributes of the Customer Class.

Customer Entity	Customer Class
CustomerID	CustomerID
CustomerName	CustomerName
Address	Address
ContactPhone	ContactPhone
Email	Email

Table 6.7

Mapping Data Type

DB-VA automatically maps the column data type to an appropriate attribute type of object model.

Example:

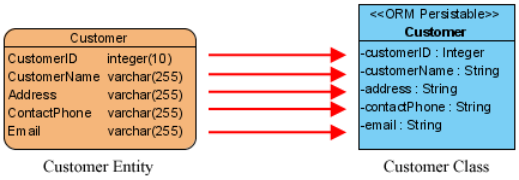


Figure 6.18 - Mapping Data Type

In the above example, the following table shows the mapping between data types

Customer Entity	Customer Class
int (10)	int
varchar(255)	String

Table 6.8

A table shows the data type mapping between Object model and Data model.

Data Model	Object Model
Bigint	Long
Binary	Byte[]
Bit	Boolean
Blob	Blob
Varchar	String
Char	Character
Char(1)	Character
Clob	String
Date	Date
Decimal	BigDecimal
Double	Double
Float	Float
Integer	Integer
Numeric	BigDecimal
Real	Float
Time	Time
Timestamp	Timestamp
Tinyint	Byte
Smallint	Short
Varbinary	Byte[]

Table 6.9

Mapping Primary Key

As the columns map to attributes in a one-to-one mapping, primary key columns in the entity map to attributes as a part of a class.

Example:

Figure 6.19 - Mapping Primary Key

In the example, the primary key of entity Product, **ProductID** maps to an attribute ProductID of the class Product.

Mapping Relationship

Relationship represents the correlation among entities. Each entity of a relationship has a role, called Phrase describing how the entity acts in it. The phrase is attached at the end of a relationship connection line. DB-VA maps the phrase to role name of association in the object model.

There are two types of relationships in data model mapping to object model - identifying and non-identifying.

Identifying relationship specifies the part-of-whole relationship. It means that the child instance cannot exist without the parent instance. Once the parent instance is destroyed, the child instance becomes meaningless.

Non-identifying relationship implies weak dependency relationship between parent and child entities. There are two kinds of non-identifying relationships, including optional and mandatory. The necessity of the parent entity is "exactly one" and "zero or one" in the mandatory and optional non-identifying relationship respectively.

Mapping Identifying Relationship

Since the identifying relationship specifies the part-of-whole relationship, it maps to composite aggregations which implies that the part cannot exist without its corresponding whole.

Example:

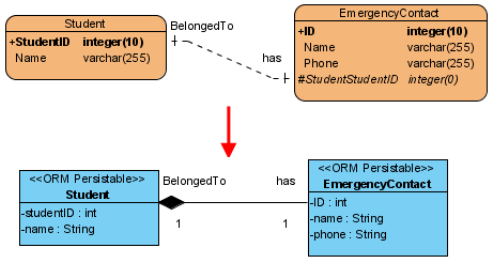


Figure 6.20 - Mapping Identifying Relationship

In the above example, DB-VA maps the identifying relationship between the entities of EmergencyContact and Student to composition aggregation.

Mapping Non-identifying Relationship

Since non-identifying relationship implies weak relationship between entities, DB-VA maps it to association.

Example:

Figure 6.21 - Mapping Non-identifying Relationship

In the above example, non-identifying relationship between entities Owner and Property maps to association between Classes of Owner and Property.

Mapping Cardinality

Cardinality refers to the number of possible instances of an entity relate to one instance of another entity. The following table shows the syntax to express the Cardinality.

Table shows the syntax expressing the Cardinality

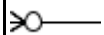
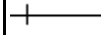
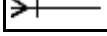
Type of Cardinality	Description
	Zero or one instance
	Zero or more instances
	Exactly one instance
	One or more instances

Table 6.10

Table shows the typical mapping between Entity Relationship Diagram and Class Diagram.

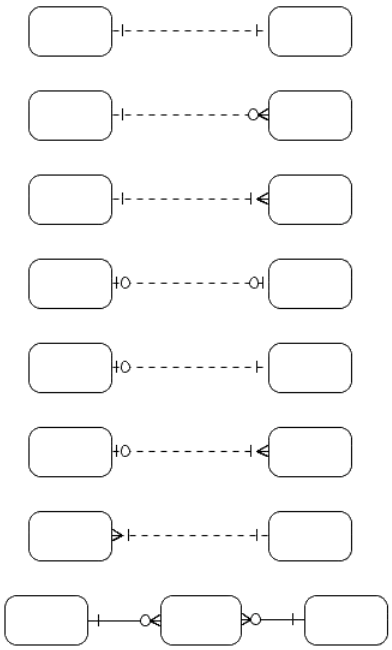
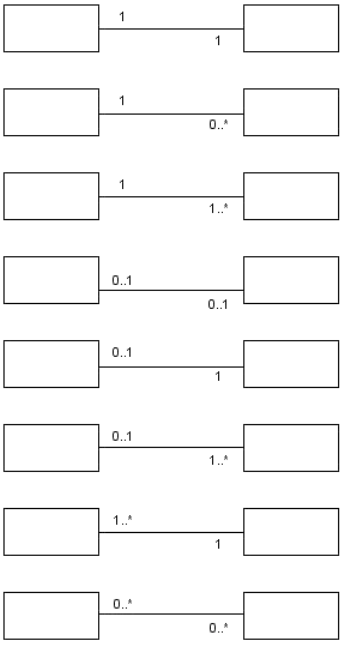
Entity Relationship Diagram	Class Diagram
	

Table 6.11

Mapping Many-to-Many Relationship

For each many-to-many relationship between entities, DB-VA generates a Link Entity to form two one-to-many relationships in between. The primary keys of the two entities will automatically migrate to the link entity to form the primary/foreign keys.

Example:

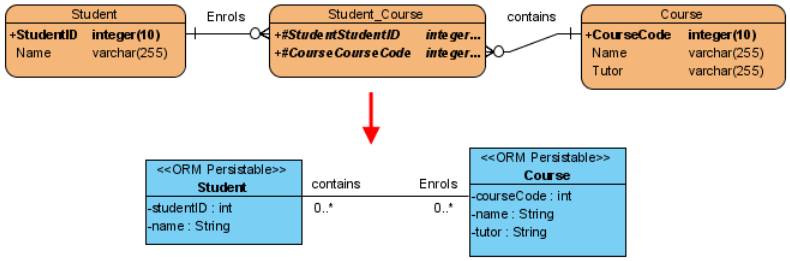


Figure 6.22 - Mapping Many-to-Many Relationship

In the above example, DB-VA generates the link entity once a many-to-many relationship is setup between two entities. To transform the many-to-many relationship, DB-VA maps the many-to-many relationship to many-to-many association.

Mapping Array Table to Collection of Objects

DB-VA promotes the use of Array Table to allow users retrieve objects in the form of primitive array.

When DB-VA transforms an array table, the array table will map to an attribute with array type modifier.

Example:

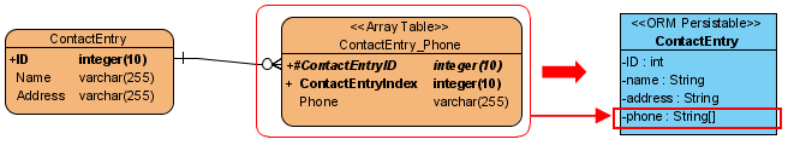


Figure 6.23 - Mapping Array Table to Collection of Objects

In the above example, the phonebook has a contact entry for each contact person. Each contact person may have more than one phone numbers. The array table of ContactEntry_Phone maps into the phone attribute with array type modifier in the ContactEntry class.

Mapping Data Model Terminology

The following table shows the shift from data model to object model terminology.

Data Model Term	Object Model Term
Entity	Class
Instance of an entity	Object
Relationship	Association
Supertype/subtype	Generalization
Column	Attribute
Phrase	Role
Cardinality	Multiplicity

Table 6.12

Showing Mapping by ORM Diagram

In order to identify the mapping between the object and data models clearly, DB-VA provides you an ORM diagram to show the mappings between the ORM-Persistable class and its corresponding entity.

As DB-VA allows you to name the ORM-Persistable class and its corresponding entity differently, and also the attributes and columns as well, you may find it difficult to identify the mappings not only between the ORM-Persistable classes and the corresponding entities, but also the attributes and columns. Taking the advantage of ORM diagram, the mappings between ORM-Persistable classes and entities and between attributes and columns can be clearly identified.

DB-VA provides you two ways to create the ORM diagram for showing the mapping between the object and data models:

1. Creating an ORM diagram from the existing class diagram and/or ERD.
2. Drawing an ORM Diagram

Creating an ORM Diagram from Existing Diagrams

The following example shows you how to show the mapping between the existing object and data models by the ORM diagram.

Let us assume the following class diagram has been created and synchronized to the entity relationship diagram (ERD).

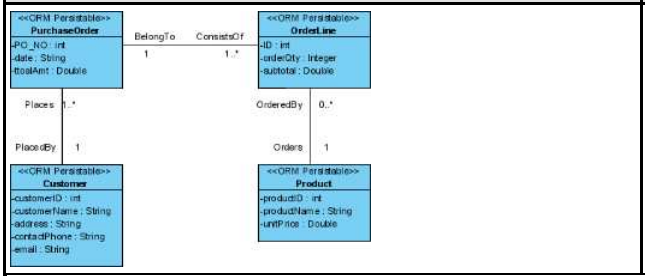
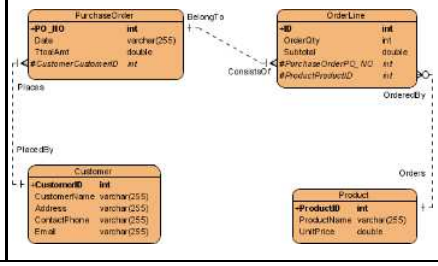
Class Diagram	Synchronized ERD
	

Table 6.13

You can create an ORM diagram by either the object model, data model, or both. The following steps show you how to create the ORM diagram by using the object model.

1. Right-click on the class diagram, select **Send to > ORM Diagram > New ORM Diagram** from the pop-up menu.

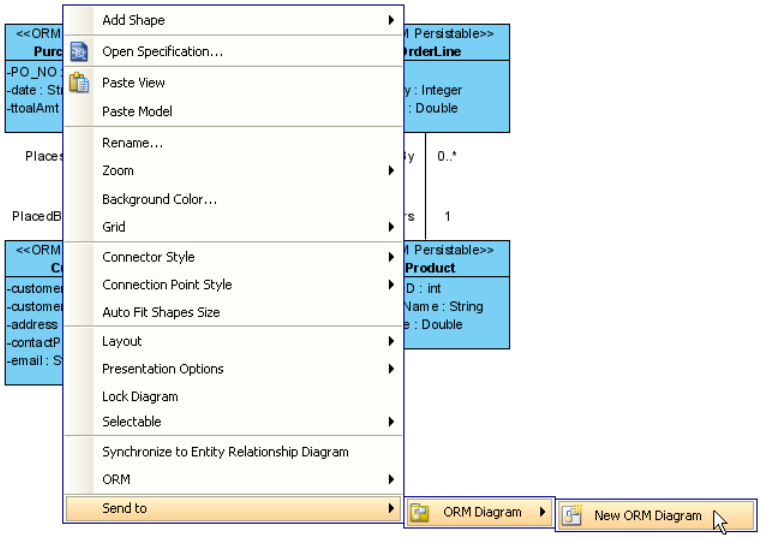


Figure 6.24 - Send to a blank new ORM Diagram

A new ORM diagram is created which displays the ORM-Persistable classes.

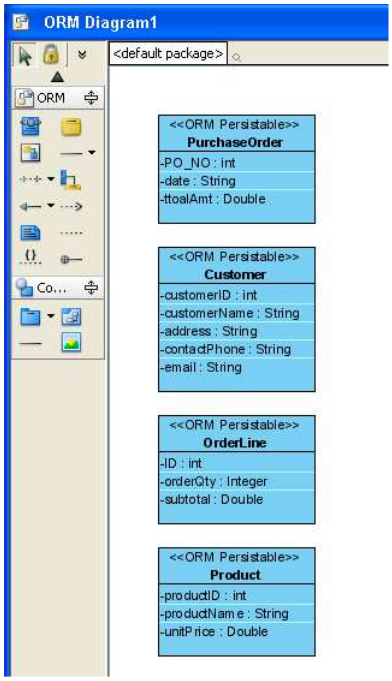


Figure 6.25 - The classes in ORM Diagram

Note

Alternatively, you can create the ORM diagram from the data model by right-clicking on the ERD, and selecting **Send to > ORM Diagram > New ORM Diagram** from the pop-up menu.

2. Mouse over the ORM Persistable class, click the **Class-Entity Mapping - > Entity** resource.

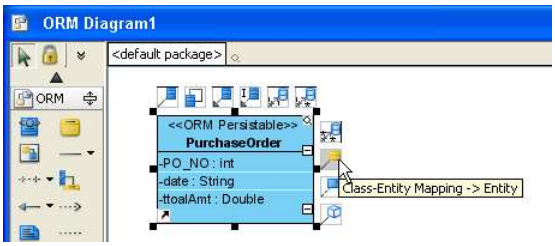


Figure 6.26 - "Class-Entity Mapping - > Entity" resource

3. Drag the resource on the ORM diagram to show the corresponding entity associated in the mapping.

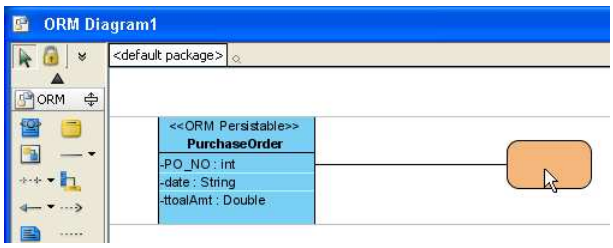


Figure 6.27 - Drag the resource to the diagram

A class-to-entity mapping is shown on the diagram.

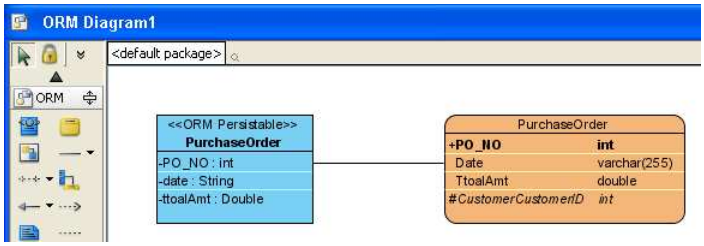


Figure 6.28 - The Mapping Entity will be created

4. By repeating steps 2 and 3, all class-to-entity mappings for all classes can be shown.

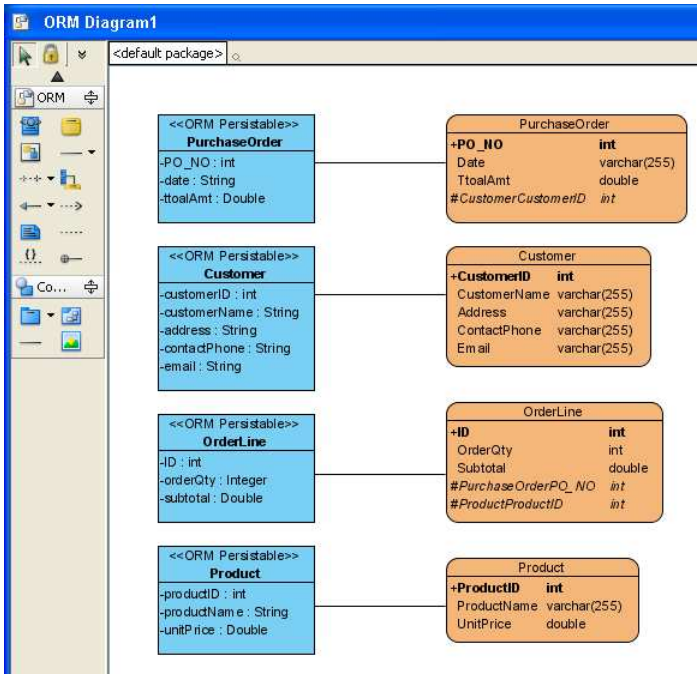


Figure 6.29 - Mapping entities for all classes

Note

If you have created the ORM diagram from the data model, you can show the class-to-entity mapping by dragging the **Class-Entity Mapping - > Class** resource.

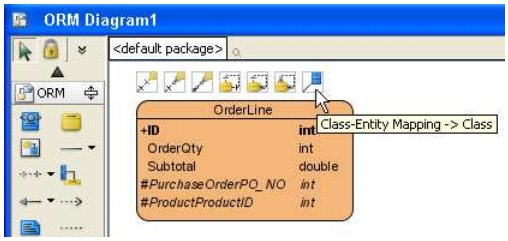


Figure 6.30 - "Class-Entity Mapping - > Class" resource on Entity

Drawing an ORM Diagram

You can create a new ORM diagram in one of the three ways:

- On the menu, click **File > New Diagram > Others > ORM Diagram**.

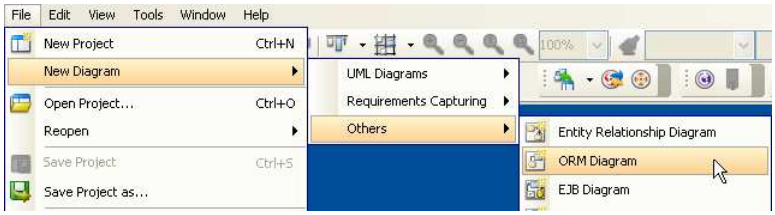


Figure 6.31 - Create an ORM Diagram

- On the **Diagram Navigator,** right-click **ORM Diagram > Create ORM Diagram**.

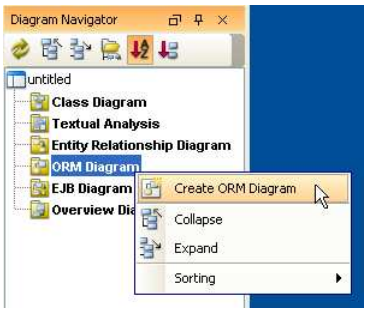


Figure 6.32 - Create ORM Diagram on Diagram Navigator

- On the toolbar, click the **New ORM Diagram** icon.

A new ORM diagram is displayed.

Creating ORM-Persistable Class and Mapping Entity to the ORM Diagram

After created a new ORM diagram, you can create ORM-Persistable class and its mapping entity on the ORM diagram.
To create an ORM-Persistable class on ORM diagram:

1. On the diagram toolbar, click **ORM-Persistable Class** shape icon.

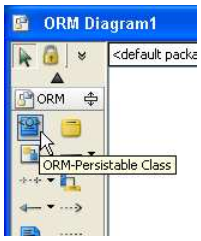


Figure 6.33 - ORM Persistable Class icon

2. Click a location in the diagram pane.

   DB-VA places a class shape icon which is marked with **<<ORM-Persistable>>** on the diagram.

3. Type the name for the **ORM-Persistable Class**.
   - You can edit the name by double-clicking the name or by pressing the ***F2*** button.

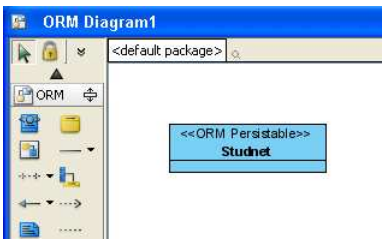


Figure 6.34 - ORM Persistable Class

Creating Associated ORM-Persistable Class to the ORM Diagram

You can create an associated ORM-Persistable class by using the association resources of an ORM-Persistable class.

1. Mouse over the ORM-Persistable class, drag the **One-to-One Association - > Class** resource to the diagram to create another ORM-Persistable class with a one-to-one directional association.

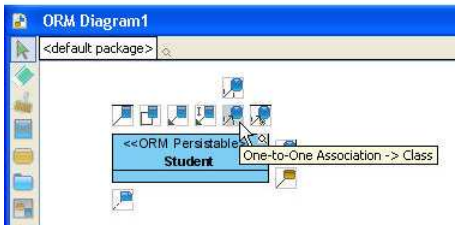


Figure 6.35 - "One-to-One Association - > Class" resource

2. Enter the name to the newly created class.

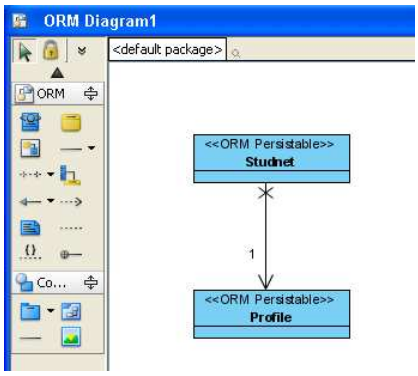


Figure 6.36 - An ORM Persistable Class are created

Creating Mapping Entity to the ORM Diagram

To create the mapping entity of the ORM-Persistable class:

1. Mouse over the ORM-Persistable class, click and drag the **Class-Entity Mapping - > Entity** resource to the diagram.

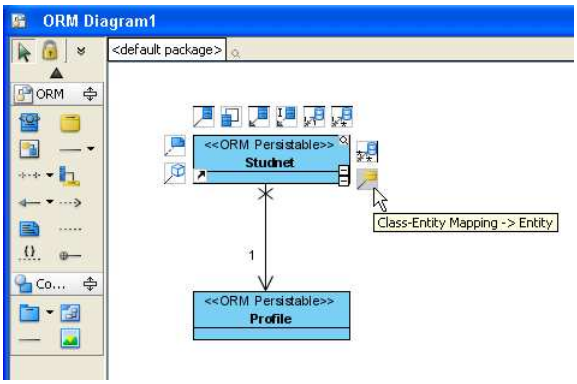


Figure 6.37 - "Class-Entity Mapping - > Entity" resource

The corresponding mapping entity, **Student** is created automatically.

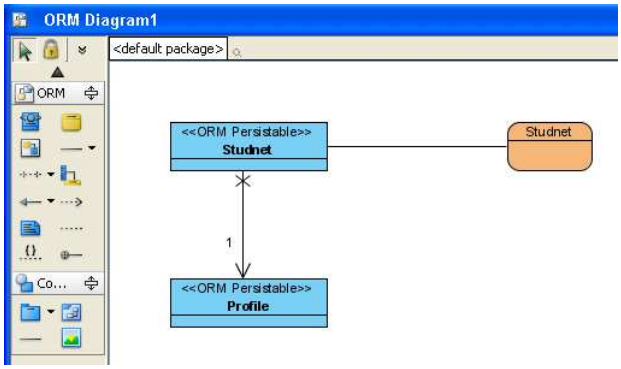


Figure 6.38 - Mapping Entity are created

2. Create the mapping entity of Profile class by using the **Class-Entity Mapping - > Entity** resource.

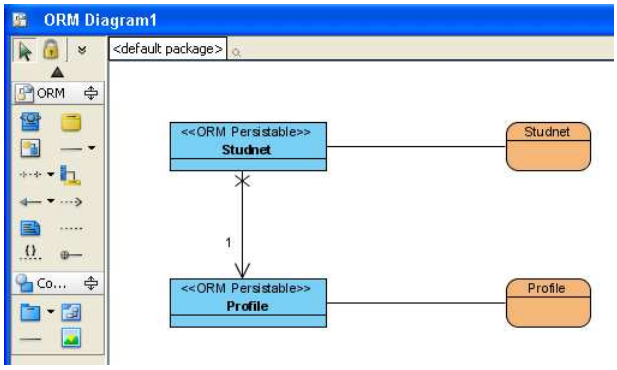


Figure 6.39 - Mapping Entities for the Classes

Note

You can create the Entity and the mapping ORM-Persistable class using the same approach of Creating ORM-Persistable Class and Mapping Entity by using the **Entity** icon on the diagram toolbar and the **Class-Entity Mapping - > Class** resource.

Showing Attribute Mapping

The object-relational mapping exists not only between the ORM-Persistable class and entity, but also the attributes and columns. You can investigate the mapping between the attributes and columns by using the Attribute Mapping feature.

To view the attribute mapping:

1. Right-click on the ORM diagram, select **View > Attribute Mapping** from the pop-up menu.

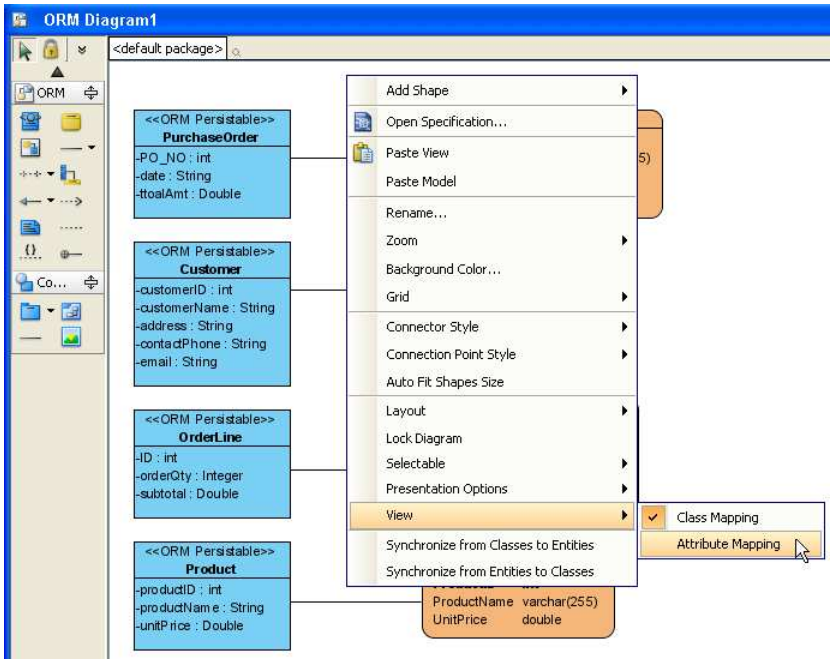


Figure 6.40 - Switch to Attribute Mapping view

The class-to-entity mapping shown on the ORM diagram is changed to attribute-to-column mapping automatically.

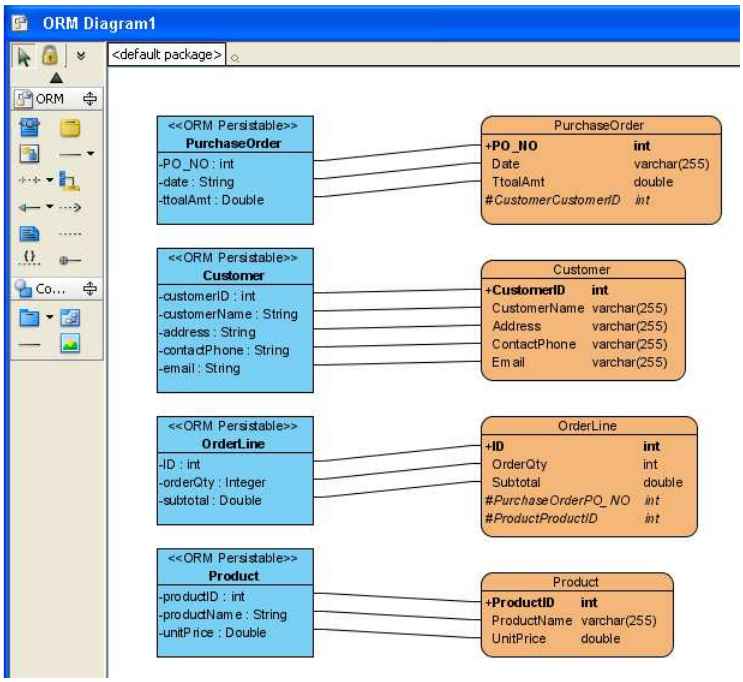


Figure 6.41 - ORM Diagram in Attribute Mapping view

Supporting Real-time Synchronization

ORM diagram supports real-time synchronization; that is, any change in the class diagram, entity relationship diagram and/or ORM diagram will automatically synchronize to each other.

Let us take the ORM diagram created in the Drawing ORM Diagram section as an example to modify the ORM-Persistable Class and Entity.

Forming a Class Diagram

You can create a class diagram from the existing models.

1. Create a new class diagram by using **New Class Diagram** icon.
2. Select the classes from the **Class Repository**, drag to the newly created class diagram.

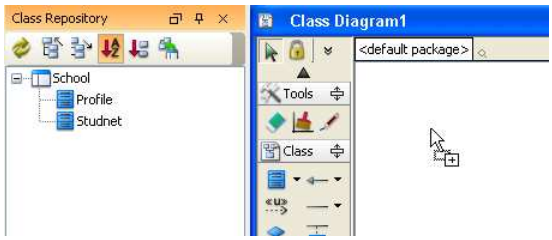


Figure 6.42 - Drag the classes from Class Repository

The following class diagram is created.

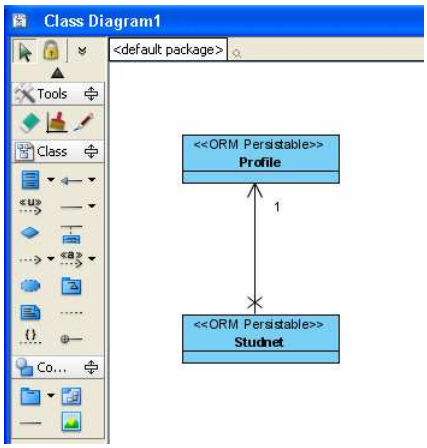


Figure 6.43 - Class Diagram form by the model in Class Repository

Modifying ORM-Persistable Class

You can modify the ORM-Persistable class such as renaming the class name and adding attributes to the class.

1. Right-click the **Student** class, select **Add > Attribute** from the pop-up menu.

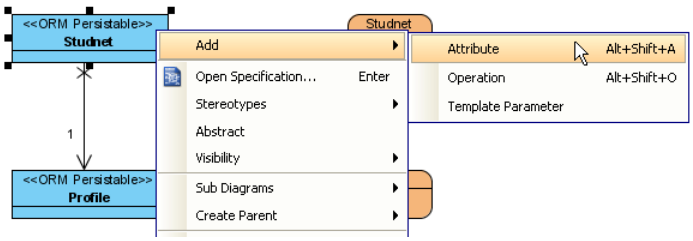


Figure 6.44 - Add an attribute in Class

An attribute is added to the Student class, and the mapping attribute is added to the mapping Student entity automatically.

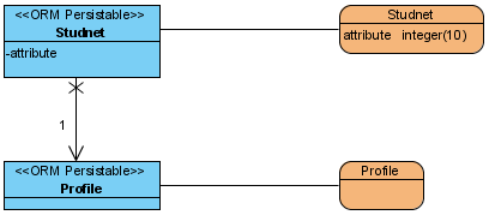


Figure 6.45 - The column will added automatically

2. Enter "**StudentID : String**" to the attribute of Student by double-clicking on the attribute. The type of the mapping column is mapped to varchar(255) automatically.

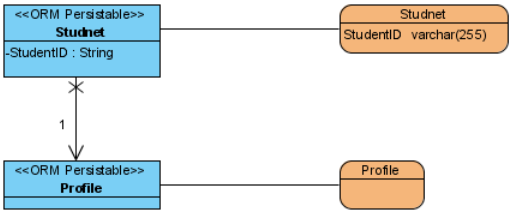


Figure 6.46 - The column will update when the mapping attribute modified

Modifying Entity

You can modify the entity such as renaming the entity name and adding columns to the entity.

1. Rename the column of **Student** entity from **attribute** to **ID** by double-clicking on it.

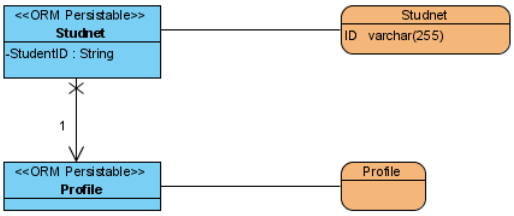


Figure 6.47 - Rename the column name and select not update to attrbiute

2. Right-click the **Student** entity, select **New Column** from the pop-up menu.

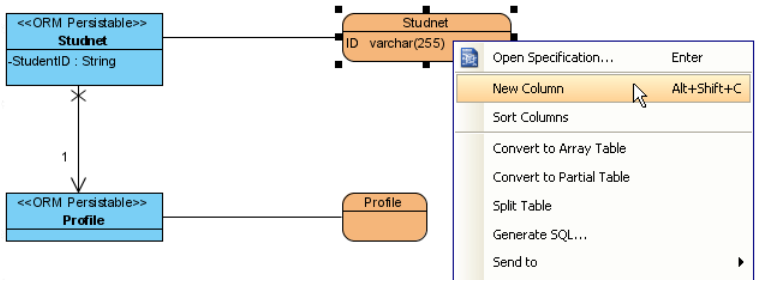


Figure 6.48 - Add a column to an Entity

A new column is added to the Student entity and the corresponding attribute is added to the Student ORM-Persistable class automatically.

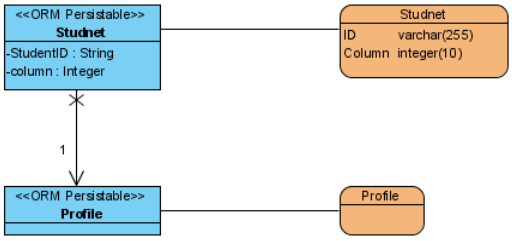


Figure 6.49 - Attribute will be added automatically

3. Enter "**Name : varchar(255)**" to the column by double-clicking on it. The type of the mapping attribute is mapped to String automatically.

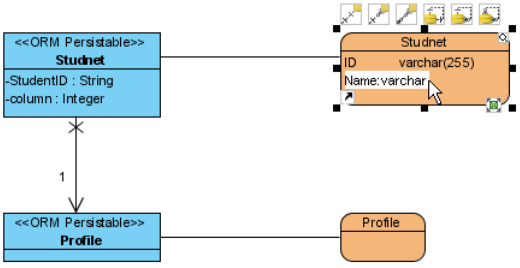


Figure 6.50 - Rename the column and data type

4. Double-click the column attribute of the Student class, rename from **column** to **Name**.

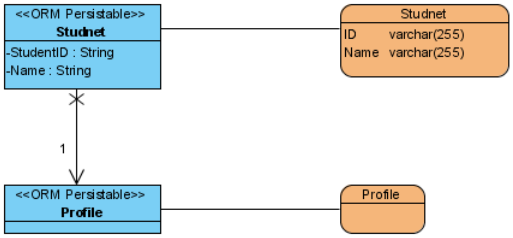


Figure 6.51 - The attribute changed

5. Modify the classes and entities on the ORM diagram as shown below:

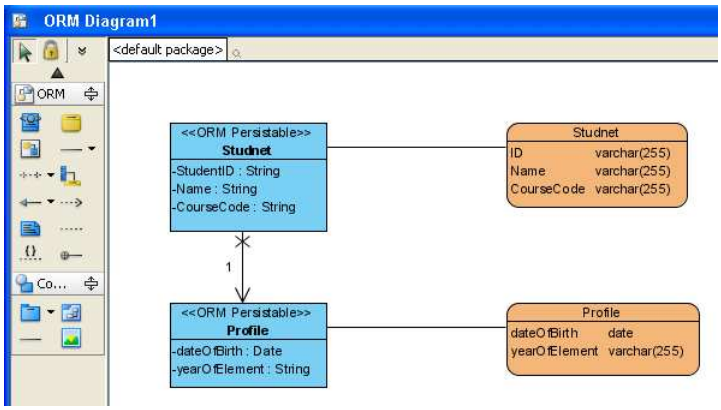


Figure 6.52 - Modify the Class in ORM Diagram

6. Navigate to the class diagram, the class diagram is updated automatically.

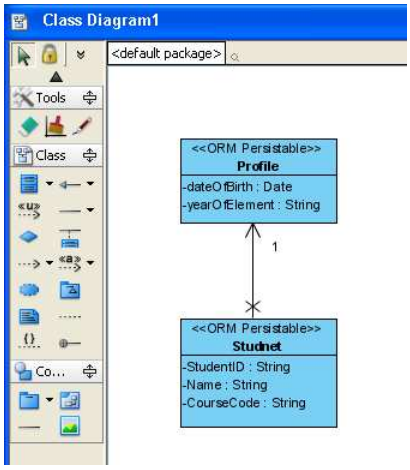


Figure 6.53 - The classes in Class Diagram is updated automatically

Switching the View of Mapping

As ORM diagram provides two views of mapping, including the mapping between ORM-Persistable class and entity (called Class Mapping), and the mapping between attributes and columns (called Attribute Mapping).

To change the view of mapping, right-click on the ORM-diagram, select the desired view from the sub-menu of **View**.

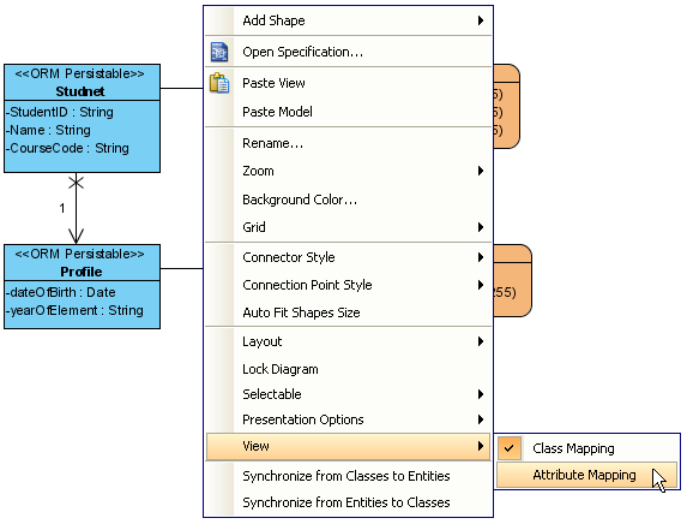


Figure 6.54 - Switch to Attribute mapping view

By selecting the **View > Attribute Mapping** from the pop-up menu, The class-to-entity mapping shown on the ORM diagram is changed to attribute-to-column mapping automatically.

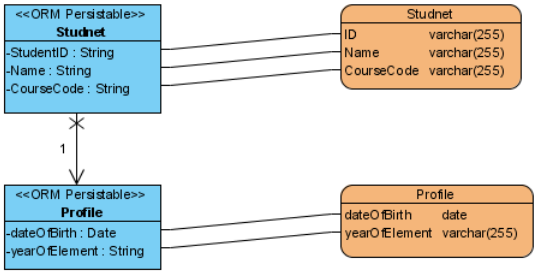


Figure 6.55 - The ORM Diagram in Attribute Mapping