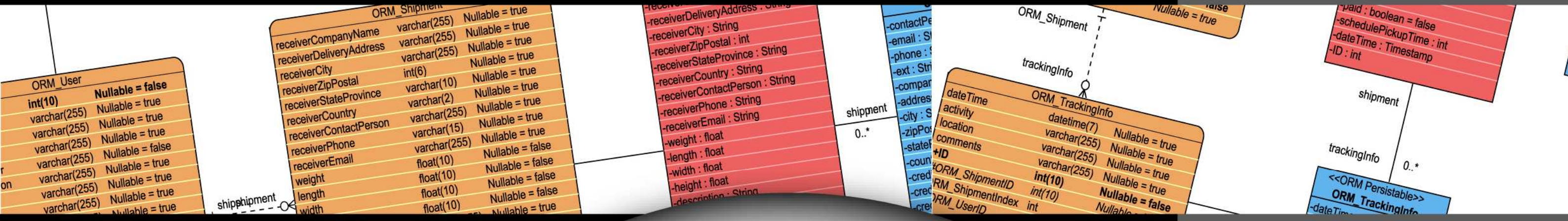
Visual Paradigm

Build Quality Applications Faster, Better and Cheaper

Database Visual ARCHITECT Programmer's Guide for .NET

Access database with Object-Oriented technology



DB Visual ARCHITECT 4.0 Programmer's Guide for .NET

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.

Copyright Information
Copyright © 1999-2007 by Visual Paradigm. All rights reserved.

The material made available by Visual Paradigm in this document is protected under the laws and various international laws and treaties. No portion of this document or the material contained on it may be reproduced in any form or by any means without prior written permission from Visual Paradigm.

Every effort has been made to ensure the accuracy of this document. However, Visual Paradigm makes no warranties with respect to this documentation and disclaims any implied warranties of merchantability and fitness for a particular purpose. The information in this document is subject to change without notice.

All examples with names, company names, or companies that appear in this document are imaginary and do not refer to, or portray, in name or substance, any actual names, companies, entities, or institutions. Any resemblance to any real person, company, entity, or institution is purely coincidental.

Trademark Information

DB Visual ARCHITECT is registered trademark of Visual Paradigm.
Sun, Sun ONE, Java, Java2, J2EE and EJB, NetBeans are all registered trademarks of Sun Microsystems, Inc.
Eclipse is registered trademark of Eclipse.
JBuilder is registered trademark of Borland Corporation.
IntelliJ and IntelliJ IDEA are registered trademarks of JetBrains.
Microsoft, Windows, Windows NT, Visio, and the Windows logo are trademarks or registered trademarks of Microsoft Corporation.
Oracle is a registered trademark, and JDeveloper is a trademark or registered trademark of Oracle Corporation.
BEA is registered trademarks of BEA Systems, Inc.
BEA WebLogic Workshop is trademark of BEA Systems, Inc.
Rational Rose is registered trademark of International Business Machines Corporation.
WinZip is a registered trademark of WinZip Computing, Inc.
Other trademarks or service marks referenced herein are property of their respective owners.

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

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

Table of Contents

1 Generating C# .NET, Database and Persistent Library

Chapter 1 - Generating C# .NET, Database and Persistent Library

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

In this chapter:

- Introduction
- Configuring Database
- Generating Database
- Generating C# .NET Code

Introduction

DB Visual ARCHITECT (DB-VA) provides an easy-to-use environment bridging between object model, data model and relational database. You can use visual modeling for both logical data design and physical data design. It also automates the mapping between object model and data model.

In this chapter, we assume that you know how to use the class diagram and entity relationship diagram to design the model (please refer to the Designer Guide for the design with class diagram and entity relationship diagram in details). Class Diagram and Entity Relationship Diagram will be used in this chapter to demonstrate how to use the -VA to export database schema (DDL) to database and generate C# persistent code.

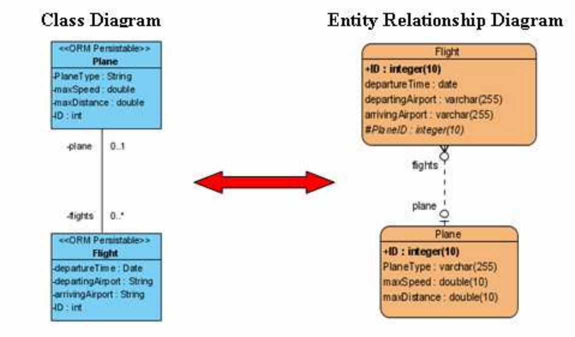


Figure 1.1 - Mapping between classes and entities

Configuring Database

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

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

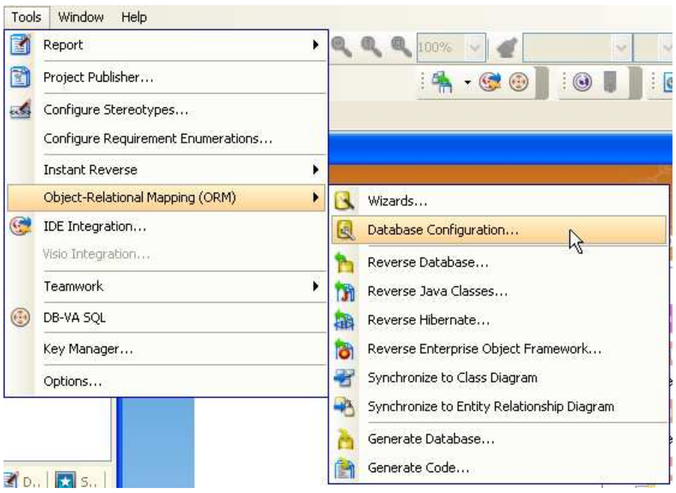
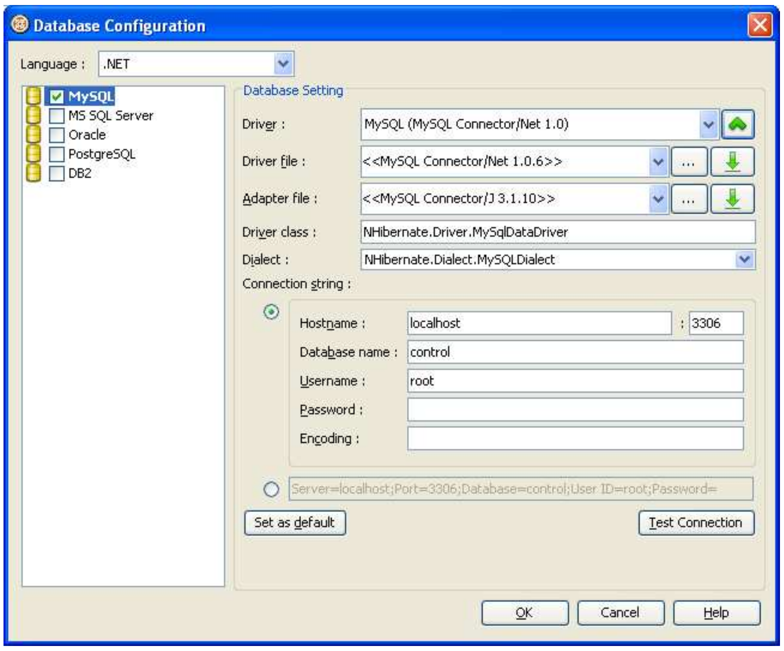


Figure 1.2 - select Database Configuration

3. Select **.NET** in Language in drop down menu, select a database and enter database settings. We will use MySQL database in this example.


Database Configuration

Language : .NET

- [✓] MySQL
- [ ] MS SQL Server
- [ ] Oracle
- [ ] PostgreSQL
- [ ] DB2

Database Setting

Driver : MySQL (MySQL Connector/Net 1.0)
Driver file : <<MySQL Connector/Net 1.0.6>>
Adapter file : <<MySQL Connector/J 3.1.10>>
Driver class : NHibernate.Driver.MySqlDataDriver
Dialect : NHibernate.Dialect.MySQLDialect
Connection string :

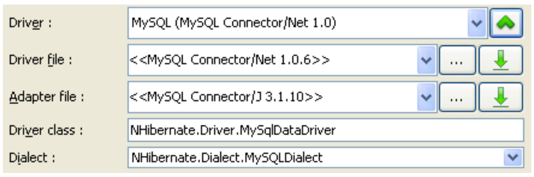
Hostname : localhost : 3306
Database name : control
Username : root
Password :
Encoding :

Server=localhost;Port=3306;Database=control;User ID=root;Password=

Figure 1.3 - Database Configuration dialog

4. Enter database setting

For **Driver**, select a .NET Driver. It contains the default **Driver Class** and **Dialect**. You can click the drop down button to modify its Driver Class and Dialect.



Driver :	MySQL (MySQL Connector/Net 1.0)
Driver file :	<<MySQL Connector/Net 1.0.6>>
Adapter file :	<<MySQL Connector/J 3.1.10>>
Driver class :	NHibernate.Driver.MySqlDataDriver
Dialect :	NHibernate.Dialect.MySQLDialect

Figure 1.4 - The driver configuration

For **Driver** and **Adapter Driver** file, you can click the **Driver** button to select **Download Driver and Adapter**, **Download**, **Update**, or **Default Driver**. DB-VA will help you to download the most up-to-date driver and adapter driver according to the Driver field information. You also can select **Browse...**to select a driver and adapter driver file in your computer.

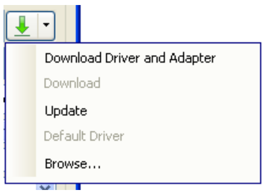


Figure 1.5 - The download button

After downloaded the driver file, <<**MySQL Connector/Net 1.0.6**>> shown on the Driver file indicates that the .NET driver file is downloaded with the specified version number by DB-VA.

For the **Connection String**, It provides the Connection String template for different databases. You need to fill in the information for Connection String to connect database.

The original template for MySQL Connection URL:

```
Server=<host_name>;Database=<database_name>;User ID=<username>;Password=<password>;
CharSet=<charset>
```

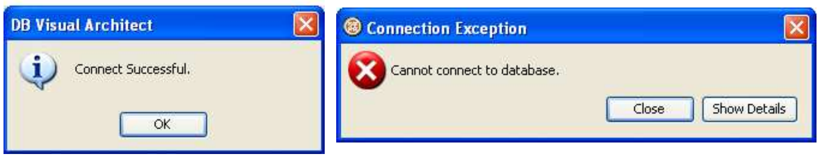
The modified template for MySQL Connection URL:

```
Server=localhost;Database=control;User ID=root;
```

5. Test the database connection by clicking the **Test Connection** button

Figure 1.6 - Test Connection button

If success to connect with database the Connection Successful dialog box will show, otherwise the Connection Exception dialog box will appear.



DB Visual Architect

Connect Successful.

OK

Connection Exception

Cannot connect to database.

Close Show Details

Figure 1.7 - The connection successful/failure message

6. Select a database to be the default database which is the default database connection for generating code and database.
   - Right click on database and select **Set as Default**

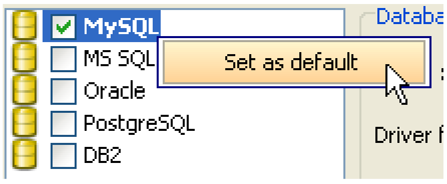


Figure 1.8 - Set the database as default

DB-VA allows you to change the default database anytime, which means you can change to use any database when you are developing application. And you do not need to worry about the database-specific details because DB-VA will take care of them for you. You only need to configure the target database as default.

Generating Database

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

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

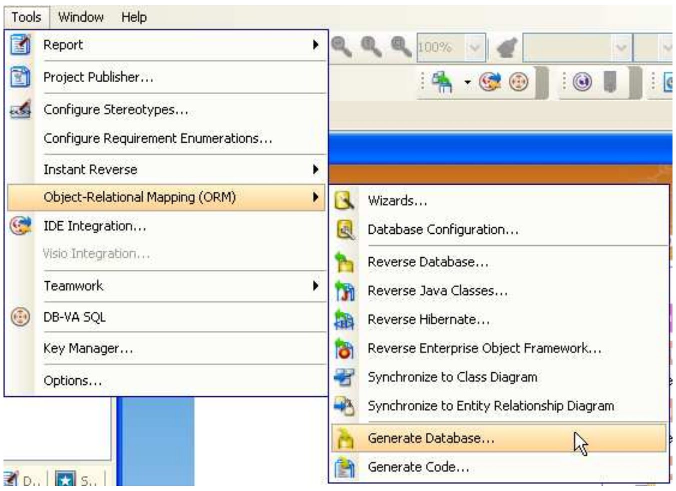


Figure 1.9 - To Generate Database

2. The dialog box shows the previously configured default database setting.

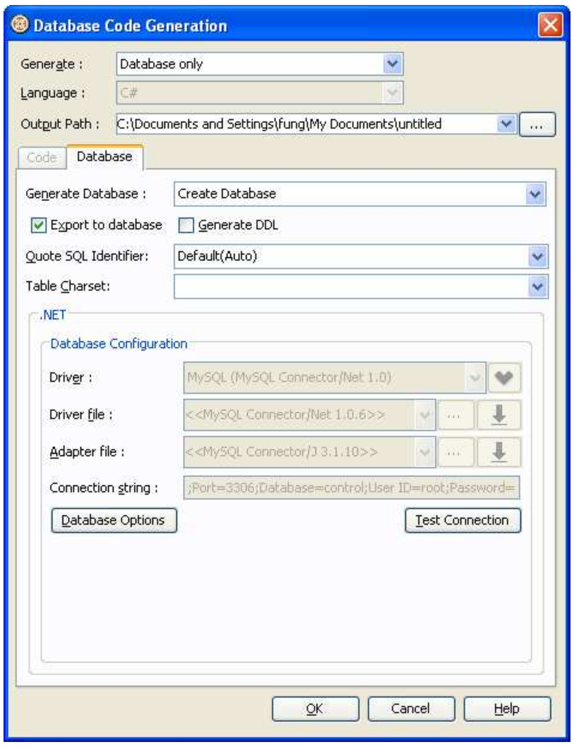

Database Code Generation
Generate : Database only
Language : C#
Output Path : C:\Documents and Settings\fung\My Documents\untitled
Generate Database : Create Database
[✓] Export to database [ ] Generate DDL
Quote SQL Identifier: Default(Auto)
Table Charset:
Driver : MySQL (MySQL Connector/Net 1.0)
Driver file : <<MySQL Connector/Net 1.0.6>>
Adapter file : <<MySQL Connector/J 3.1.10>>
Connection string : ;Port=3306;Database=control;User ID=root;Password=

Figure 1.10 - Database Code Generation dialog

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

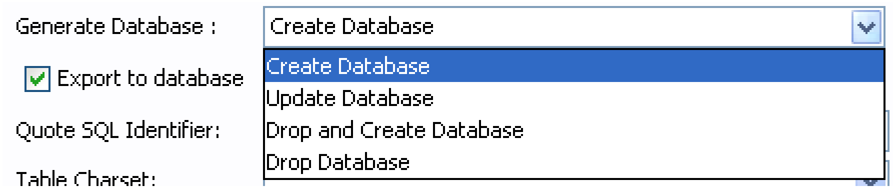

Generate Database : Create Database
[✓] Export to database

Figure 1.11 - Generate Database options

4. Select **Export to database** option allows altering the database immediately after you click the **OK** button.

[✓] Export to database

Figure 1.12 - Export to database option

5. Select **Generate DDL** allows the generation of DDL file.

[✓] Generate DDL

Figure 1.13 - Generate DDL option

6. If you used some reserved word (e.g. Order) in your database design, you can choose the **Quote SQL Identifier** option to avoid the naming problem in your design with the target database. Auto -only quote the detected reserved word. Yes -quote all table or column names. No - donï¿½ï¿½t quote any word(s)

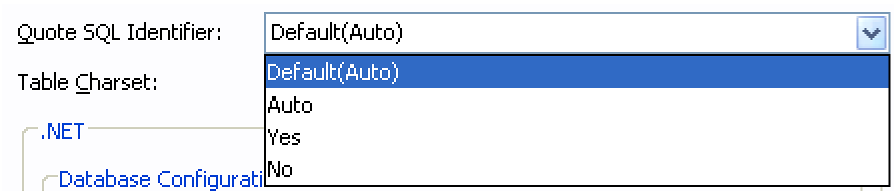
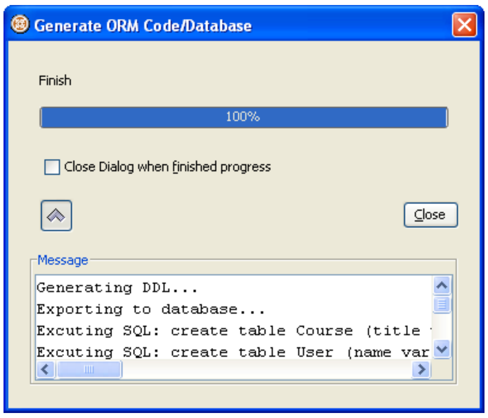


Figure 1.14 - Quote SQL Identifier options

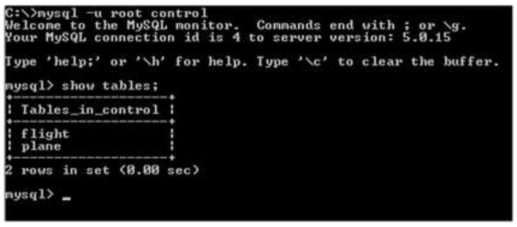
7. Click **Database Options** button to reconfigure the database settings before generating database.
8. Click **OK** on the dialog box then DB-VA will export the database schema to the default database and generate the DDL file to the output path.


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

```
Generating DDL...
Exporting to database...
Excuting SQL: create table Course (title
Excuting SQL: create table User (name var
```

Figure 1.15 - Generate ORM Code/Database dialog

9. Check the tables created in the MySQL database.



```
C:\>mysql -u root control
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 5.0.15

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

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

mysql> _
```

Figure 1.16 - The tables generate in database

10. Read the generated DDL file

```
create table Flight (ID int not null auto_increment, departureTime date,
departingAirport varchar(255), arrivingAirport varchar(255), PlaneID int, primary key
(ID)) type=InnoDB;

create table Plane (ID int not null auto_increment, PlaneType varchar(255), maxSpeed
double not null, maxDistance double not null, primary key (ID)) type=InnoDB;

alter table Flight add index FK_Flight_1115 (PlaneID), add constraint FK_Flight_1115
foreign key (PlaneID) references Plane (ID);
```

Generating C# .NET Code

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

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

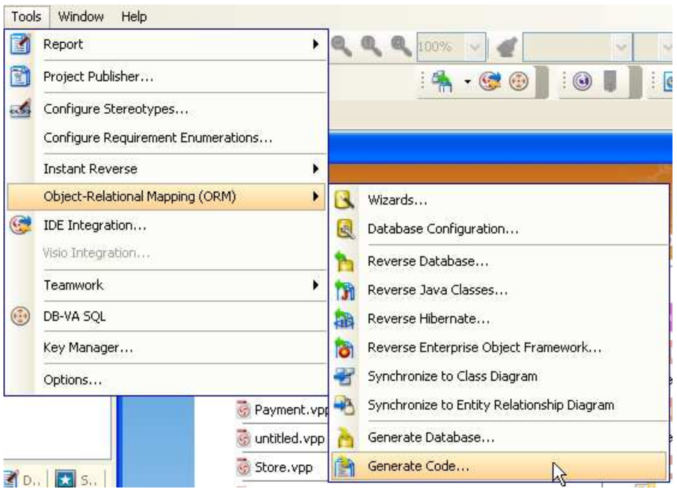
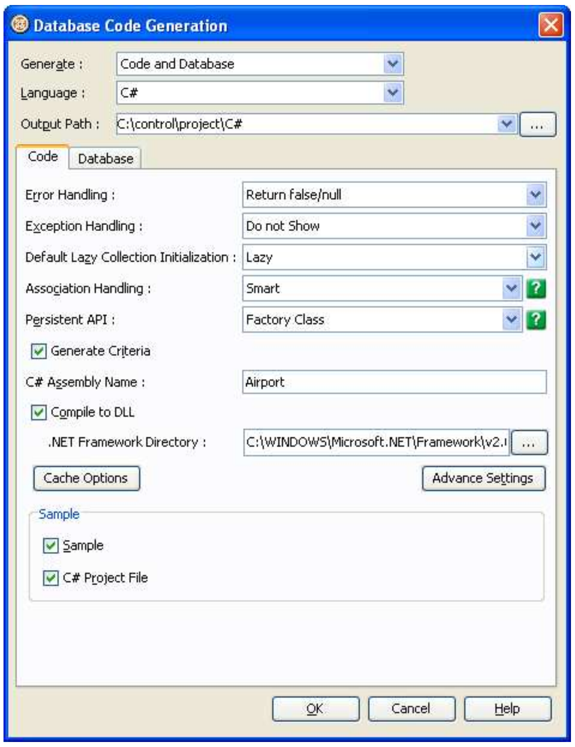


Figure 1.17 - To generate code

2. The Database Code Generation dialog box for C#:



Database Code Generation

Generate : Code and Database
Language : C#
Output Path : C:\control\project\C#

Code | Database

Error Handling : Return false/null
Exception Handling : Do not Show
Default Lazy Collection Initialization : Lazy
Association Handling : Smart
Persistent API : Factory Class
☑ Generate Criteria
C# Assembly Name : Airport
☑ Compile to DLL
.NET Framework Directory : C:\WINDOWS\Microsoft.NET\Framework\v2.
Cache Options | Advance Settings

Sample
☑ Sample
☑ C# Project File

OK | Cancel | Help

Figure 1.18 - Database Code Generation dialog

- **Output Path**

  Specify the location of C# persistent code generation.

- **Error Handling**

  Select the way to handle errors. The possible errors include PersistentException, ADOException.

  - **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.

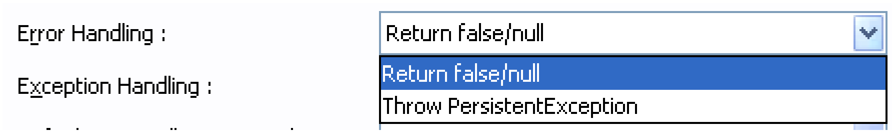


Figure 1.19 - Error Handling options

- **Exception Handling**
  - **Do not Show** - It hides the error message.
  - **Print to Error Stream** -It prints the error message to the error stream.
  - **Print to log4net** - It prints the error message to the log4net library.

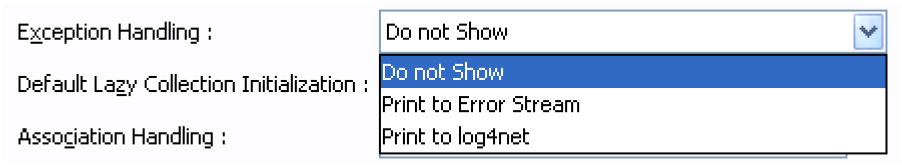


Figure 1.20 - exception Handling options

- **Lazy Collection Initialization**

  Check this option to avoid the associated objects from being loaded when the main object is loaded. Unchecking this option will result in the loading of associated objects when the main object is loaded. If you enabled (checked) the lazy collection initialization, associated objects (1 to many) will not be loaded until you access it (e.g. getFlight(0)). Enabling this option usually reduce more then 80% of the database loading.

- **Association Handling**

  Select the type of association handling to be used, either Smart or Standard.

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

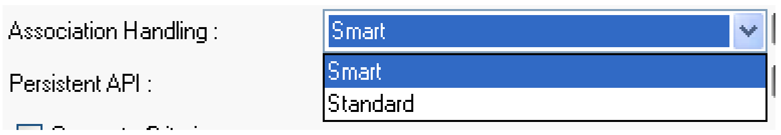


Figure 1.21 - Association Handling options

- **Persistent API**

  Select the type of persistent code to be generated, Static Methods, Factory Class, DAO or POJO.

  - **Static Method** - Client can create, retrieve, and persist the PersistentObject directly.
  - **Factory Class** - Create FactoryObject class for client to create and retrieve PersistentObject. Client still can persist the PersistentObject directly.
  - **DAO** - Client uses the PersistentObjectDAO class to create, retrieve and persist PersistentObject.
  - **POJO** - Client uses the PersistentManager to retrieve and persist PersistentObject

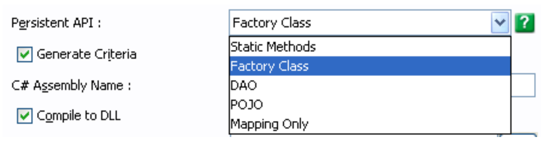


Figure 1.22 - Persistent API options

- **Create Criteria**

  You can check the option for Generate Criteria to generate the criteria class for each ORM Persistable class. The Criteria is used for querying the database in an object-oriented way (please refer to chapter 9 for more details about the criteria)

- **Sample**

  Sample files, including C# application sample and C# project file for Visual Studio .NET 2003 are available for generation. The generated sample files guide you through the usage of the C# persistence class. You can check the options to generate the sample files for reference.

  You need to select to generate the sample and check create C# project because you will modify the sample to execute the generated C# persistence class in Visual Studio .NET 2003.

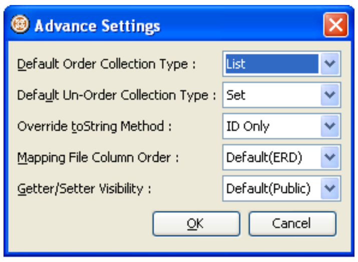
- **C# Assembly name**

  Specify the name of the assembly for the .NET application which holds the assembly metadata.

- **Compile to DLL**

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

- **Advance Setting**
  - **Default Order Collection Type** - Select the type of ordered collection to be used in handling the multiple cardinality relationship, either **List** or **Map**.
  - **Default Un-Order Collection Type** - Select the type of un-ordered collection to be used in handling the multiple cardinality relationship, either **Set** or **Bag**.
  - **Override toString Method** - Select the way that you want to override the toString method of the object. There are three options provided to override the toString method as follows:
    - **ID Only** - the toString method returns the value of the primary key of the object as string.
    - **All Properties** - the toString method returns a string with the pattern "Entity[<column1_name>=<column1_value><column2_name>=<column2_value>...]".
    - **No** - the toString method will not be overridden.



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

Figure 1.23 - Advance Setting dialog

3. Click **OK** to generate the C# persistent code to the **Output Path**.
   - **bin** folder contains the generated code DLL.
   - **src** folder contains the source code of C#.
   - **lib** folder contains the persistent library.

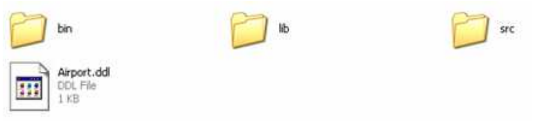


Figure 1.24 - The generated output

2 Configuring Source and Library in Visual Studio .NET 2003

Chapter 2 - Configuring Source and Library in Visual Studio .NET 2003

Microsoft Visual Studio .NET 2003 is an advanced integrated development environment by Microsoft. It lets programmers to create programs that run on Microsoft Windows and the World Wide Web.

In chapter 2, you have generated C# code, exported database schema (DDL) and created persistent library. Now you can create a project for the generated C# code in Visual Studio .NET 2003.

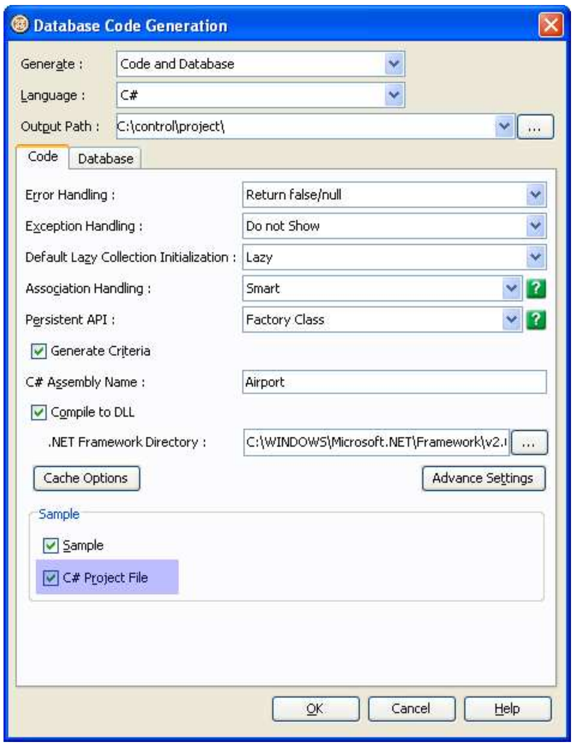
In this chapter:

- Copying Generated Source and Library to Visual Studio .NET Project
- Adding Reference of an Existing Project to DB-VA Generated C# Project
- Testing the Visual Studio .NET Project

Copying Generated Source and Library to Visual Studio .NET Project

You can create Visual Studio .NET project easily because DB-VA supports the generation of C# project file. The project file helps you to configure the generated C# classes, resources files and referenced libraries in Visual Studio .NET.

1. Select C# Project File in the Database Code Generation dialog box


Database Code Generation

Generate : Code and Database
Language : C#
Output Path : C:\control\project\

Code | Database

Error Handling : Return false/null
Exception Handling : Do not Show
Default Lazy Collection Initialization : Lazy
Association Handling : Smart
Persistent API : Factory Class
☑ Generate Criteria
C# Assembly Name : Airport
☑ Compile to DLL
.NET Framework Directory : C:\WINDOWS\Microsoft.NET\Framework\v2.
Cache Options | Advance Settings
Sample
☑ Sample
☑ C# Project File

OK | Cancel | Help

Figure 2.1 - The C# Project File options

2. The C# Project File is created in the **Output Path\src** folder

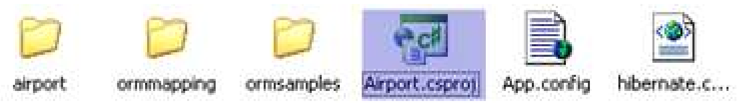


Figure 2.2 - The generated C# Project file

3. Open Visual Studio .NET 2003. From the menu bar, select **Open > Project.**

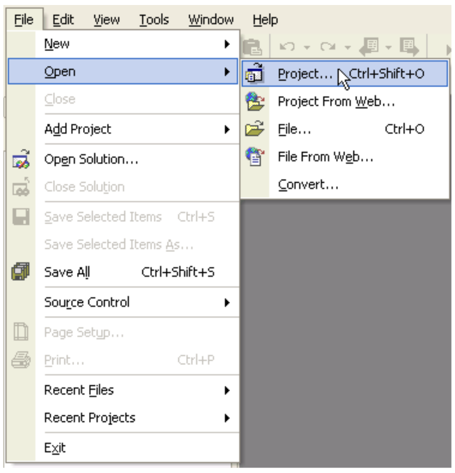


Figure 2.3 - Open a project files

4. Select the **Airport.csproj** and click **Open**

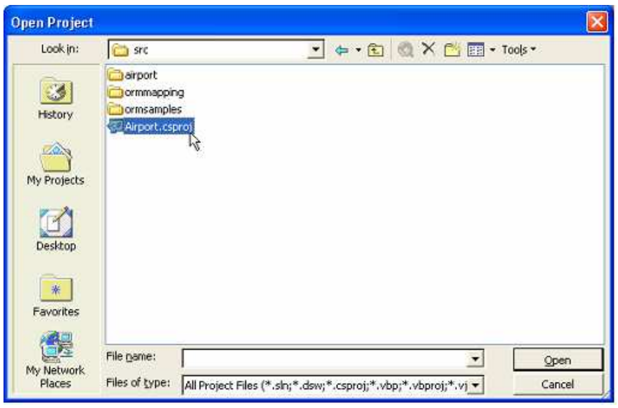
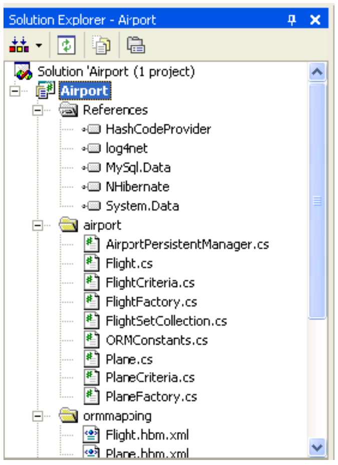


Figure 2.4 - Select a project

5. All Libraries are added to References and C# classes are inside the project.



```
Solution 'Airport (1 project)
  Airport
    References
      HashCodeProvider
      log4net
      MySql.Data
      NHibernate
      System.Data
    airport
      AirportPersistentManager.cs
      Flight.cs
      FlightCriteria.cs
      FlightFactory.cs
      FlightSetCollection.cs
      ORMConstants.cs
      Plane.cs
      PlaneCriteria.cs
      PlaneFactory.cs
    ormmapping
      Flight.hbm.xml
      Plane.hbm.xml
```

Figure 2.5 - The project contains all sources and library

Adding Reference of an Existing Project to DB-VA Generated C# Project

You can also reference the generated C# Visual Studio .NET Project as a library to develop an application.

1. Open Visual Studio .NET Project. Select **File** > **New** > **Project** on menu bar.

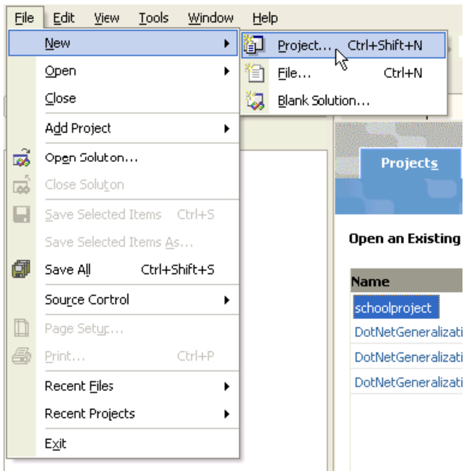
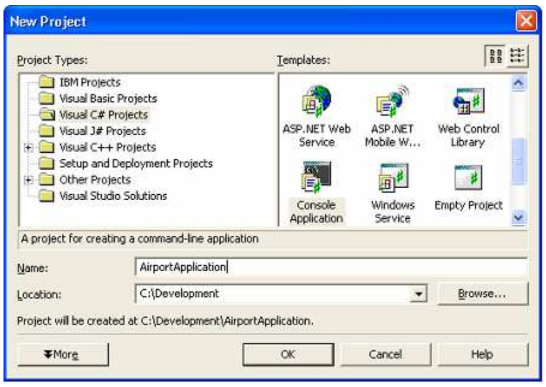


Figure 2.6 - To create a new project

2. Select Project Types as **Visual C# Projects** and Templates as **Console Application**. Enter the Project and Location for the new project and click **OK**.


New Project

Project Types:
- IBM Projects
- Visual Basic Projects
- Visual C# Projects
- Visual J# Projects
- Visual C++ Projects
- Setup and Deployment Projects
- Other Projects
- Visual Studio Solutions

Templates:
- ASP.NET Web Service
- ASP.NET Mobile W...
- Web Control Library
- Console Application
- Windows Service
- Empty Project

A project for creating a command-line application

Name: AirportApplication

Location: C:\Development

Project will be created at C:\Development\AirportApplication.

Figure 2.7 - New Project dialog

3. The **AirportApplication** project is created.

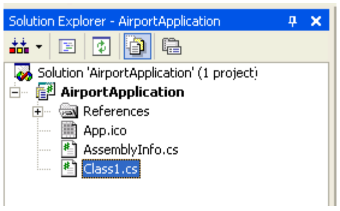
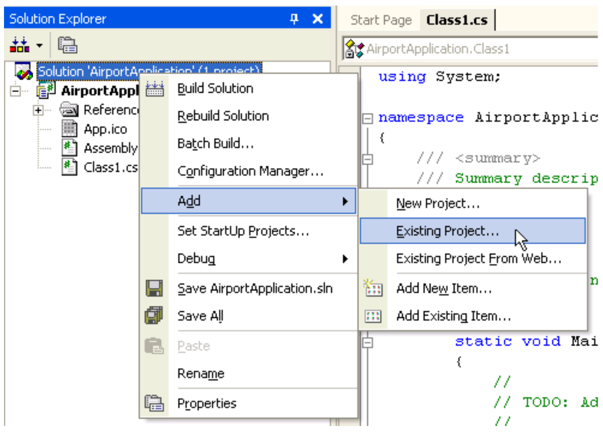

Solution Explorer - AirportApplication
Solution 'AirportApplication' (1 project)
AirportApplication
References
App.ico
AssemblyInfo.cs
Class1.cs

Figure 2.8 - Project created

You have an existing solution called AirportApplication and you may want to use the generated persistent code to develop a database application in AirportApplication.

4. Right click **Solution** > **Add** > **Existing Project...** to select the generated Project



```
using System;

namespace AirportApplic
{
    /// <summary>
    /// Summary descrip
n
        static void Mai
        {
            //
            // TODO: Ad
            //
```

Figure 2.9 - To add and existing project

5. Select the generated C# Project File **(Airport.csproj**) and add it to the existing Solution.

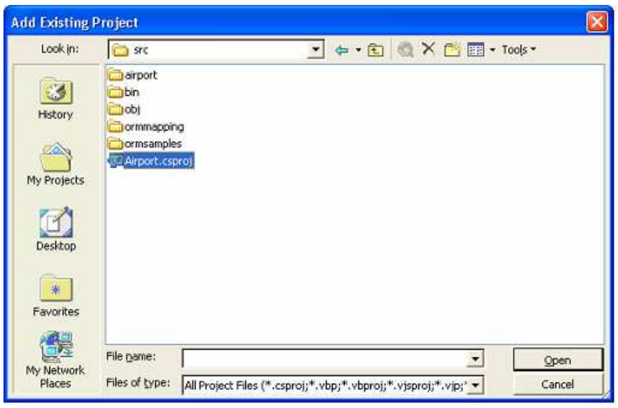


Figure 2.10 - select the project file

6. The generated C# project is added to **Solution**.

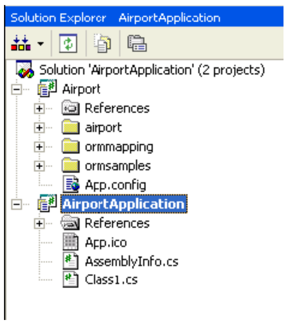


Figure 2.11 - Project imported

7. Right click **Airport** and select **Properties** on menu.

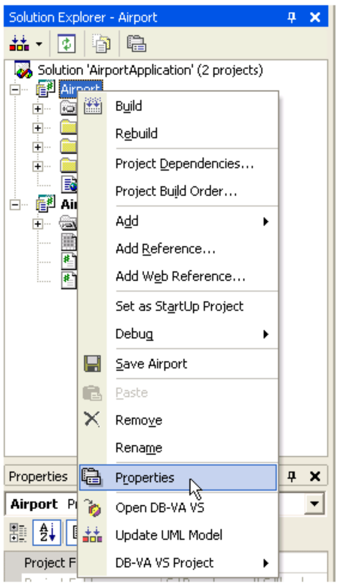
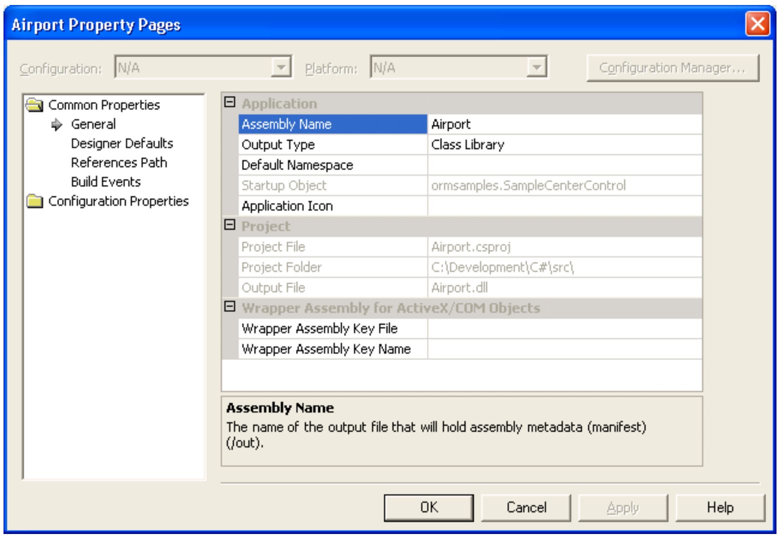


Figure 2.12 - To open the project properties

8. Change the **Output Type** from "**Windows Application**" to "**Class Library**". Click **OK**.



Application	
Assembly Name	Airport
Output Type	Class Library
Default Namespace	
Startup Object	ormsamples.SampleCenterControl
Application Icon	
Project	
Project File	Airport.csproj
Project Folder	C:\Development\C#\src\
Output File	Airport.dll
Wrapper Assembly for ActiveX/COM Objects	
Wrapper Assembly Key File	
Wrapper Assembly Key Name	

Assembly Name
The name of the output file that will hold assembly metadata (manifest) (/out).

Figure 2.13 - Project Property Page

9. From the menu bar, select **Build > Rebuild Solution**. The DLL file is generated.

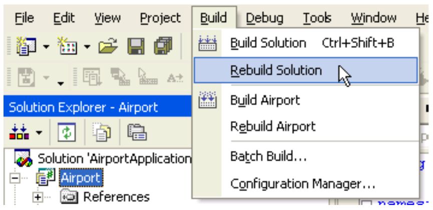


Figure 2.14 - To rebuild the solution

10. Right click the AirportApplication project and select **Add Reference...**on menu.

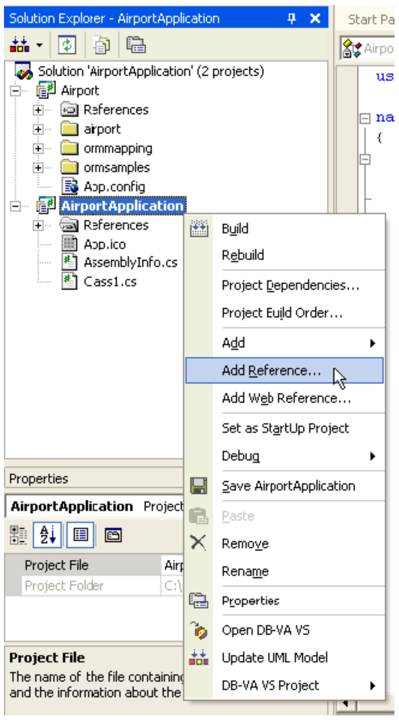
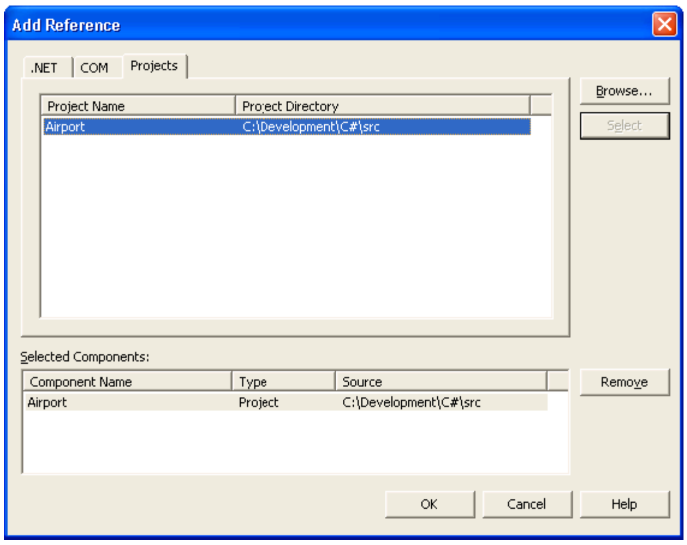


Figure 2.15 - Add project reference

11. Select Airport Project in **Projects** tab.

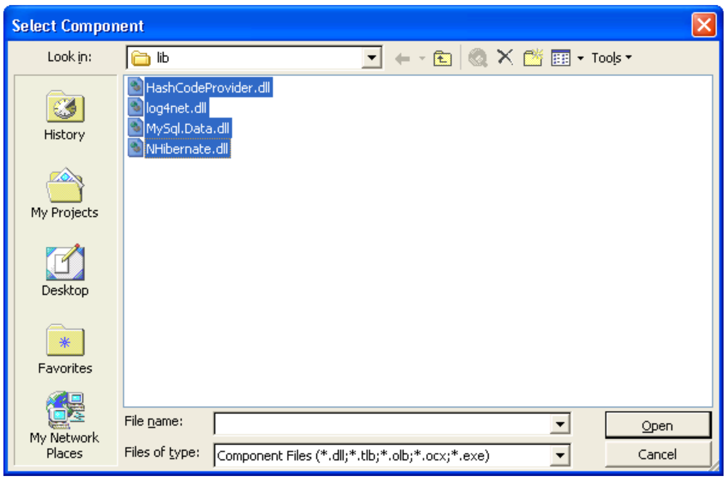


Project Name	Project Directory
Airport	C:\Development\C#\src

Component Name	Type	Source
Airport	Project	C:\Development\C#\src

Figure 2.16 - Add Reference dialog

12. Select **.NET** tab and add all the libraries (**HashCodeProvider.dll, log4net.dll, MySQL.Data.dll, NHibernate.dll**) in Airport project's **lib** folder.

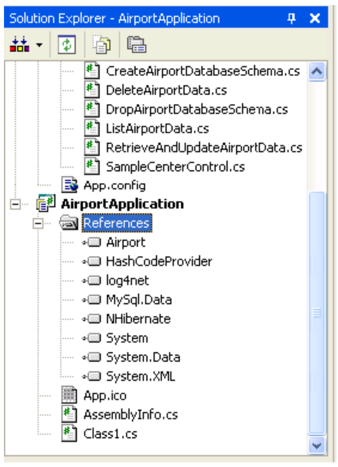


HashCodeProvider.dll
log4net.dll
MySql.Data.dll
NHibernate.dll

File name:
Files of type: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe)

Figure 2.17 - Select the Component

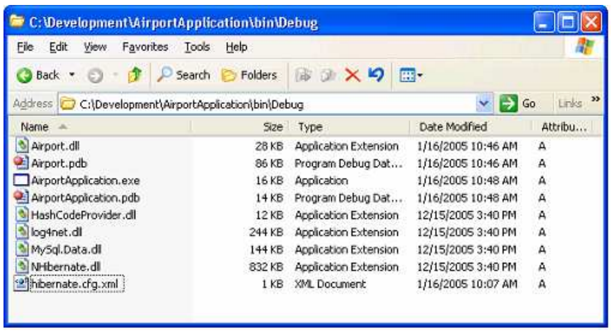
13. The Airport project and libraries are added to the AirportApplication project's Reference. You can develop application to call C# persistent classes in the AirportApplication project.



```
CreateAirportDatabaseSchema.cs
DeleteAirportData.cs
DropAirportDatabaseSchema.cs
ListAirportData.cs
RetrieveAndUpdateAirportData.cs
SampleCenterControl.cs
App.config
AirportApplication
  References
    Airport
    HashCodeProvider
    log4net
    MySql.Data
    NHibernate
    System
    System.Data
    System.XML
  App.ico
  AssemblyInfo.cs
  Class1.cs
```

Figure 2.18 - The reference added

14. Copy the **hibernate.cfg.xml** file from **Airport project\src** to **AirportApplication project\bin\Debug**. The **hibernate.cfg.xml** contains the information of database connections and other settings.



Address: C:\Development\AirportApplication\bin\Debug

```
Name                        Size  Type                   Date Modified        Attribu...
Airport.dll                28 KB  Application Extension  1/16/2005 10:46 AM   A
Airport.pdb                86 KB  Program Debug Dat...   1/16/2005 10:46 AM   A
AirportApplication.exe     16 KB  Application            1/16/2005 10:48 AM   A
AirportApplication.pdb     14 KB  Program Debug Dat...   1/16/2005 10:48 AM   A
HashCodeProvider.dll       12 KB  Application Extension  12/15/2005 3:40 PM   A
log4net.dll               244 KB  Application Extension  12/15/2005 3:40 PM   A
MySql.Data.dll            144 KB  Application Extension  12/15/2005 3:40 PM   A
NHibernate.dll            832 KB  Application Extension  12/15/2005 3:40 PM   A
hibernate.cfg.xml           1 KB  XML Document           1/16/2005 10:07 AM   A
```

Figure 2.19 - Copy the hibernate.cfg.xml to debug folder

Testing the Visual Studio .NET Project

You have created the AirportApplication project and referenced the generated Airport project. You can develop a simple program to test the project.

1. Open the Class1.cs file in AirportApplication.
2. Modify the Class1.cs file.

```
using System;
using airport;
using Orm;

namespace AirportApplication
{
	/// <summary>
	/// Summary description for Class1.
	/// </summary>
	class Class1
	{
		/// <summary>
		/// The main entry point for the application.
		/// </summary>
		[STAThread]
		static void Main(string[] args)
		{
			PersistentTransaction t =
			airport.AirportPersistentManager.Instance().GetSession().BeginTrans
			action();
			try
			{
				airport.Flight lairportFlight =
				airport.FlightFactory.CreateFlight();
				// Initialize the properties of the persistent object
				lairportFlight.ArrivingAirport = "Hong Kong International
				Airport";
				lairportFlight.DepartingAirport = "Kansai International
				Airport";
				lairportFlight.DepartureTime = DateTime.Now;

				airport.Plane lairportPlane =
				airport.PlaneFactory.CreatePlane();
				// Initialize the properties of the persistent object
				lairportPlane.PlaneType = "747 plane";
				lairportPlane.MaxSpeed = 967;
				lairportPlane.MaxDistance = 8232;
				lairportPlane.flights.Add(lairportFlight);
				lairportPlane.Save();
				// lairportPlane.Save();
				t.Commit();
			}
			catch (Exception e)
			{
				t.RollBack();
				Console.WriteLine(e);
			}
		}
	}
}
```

3. From the menu bar, select **Build > Rebuild Solution**.

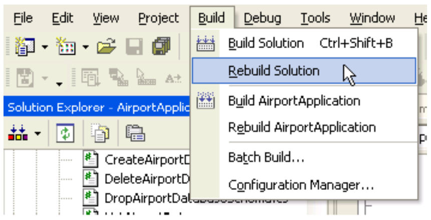


Figure 2.20 - To rebuild solution

4. Select **Debug > Start Without Debugging** to execute Class1.cs.

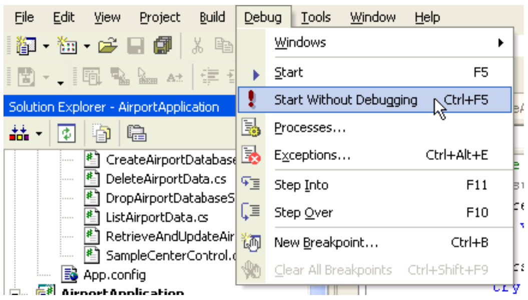
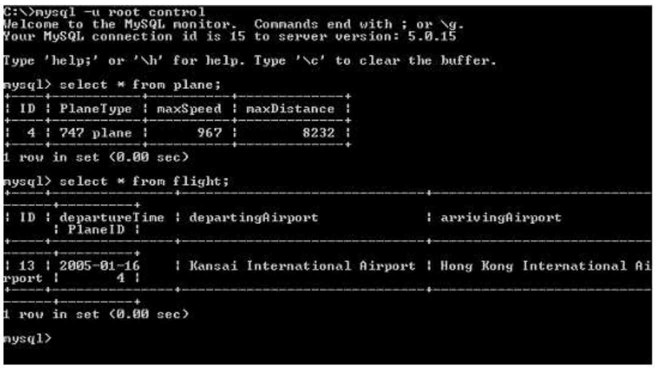


Figure 2.21 - To start without debugging

5. Check the MySQL database. The record is created.



```
C:\>mysql -u root control
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15 to server version: 5.0.15

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

mysql> select * from plane;
+----+-----------+----------+-------------+
| ID | PlaneType | maxSpeed | maxDistance |
+----+-----------+----------+-------------+
|  4 | 747 plane |      967 |        8232 |
+----+-----------+----------+-------------+
1 row in set (0.00 sec)

mysql> select * from flight;
+----+----------------+-----------------------------+------------------------------
-------+----------+
| ID | departureTime  | departingAirport            | arrivingAirport
       | PlaneID |
+----+----------------+-----------------------------+------------------------------
-------+----------+
| 13 | 2005-01-16     | Kansai International Airport | Hong Kong International Ai
rport |        4 |
+----+----------------+-----------------------------+------------------------------
-------+----------+
1 row in set (0.00 sec)

mysql>
```

Figure 2.22 - The record is created in the database

3 Developing ASP.NET Application

Chapter 3 - Developing ASP.NET Application

With DB Visual ARCHITECT (DB-VA) you can develop quality ASP.NET Web Application much faster, better and cheaper. All DB-VA generated C# code, configuration files and persistent layer library are deployable to Internet Information Services (IIS). DB-VA generates all C# code for accessing database. You do not need to write SQL to insert, query, update or delete the record. All code you need to program is plain C# code (e.g. OrderDAO.Save(myOrder);). In this chapter we will use a simple "School System" application to show you how to generate C# code, configure your web application, and create/query/update/ delete objects. Again you do not need to write a single SQL statement for all above operations.

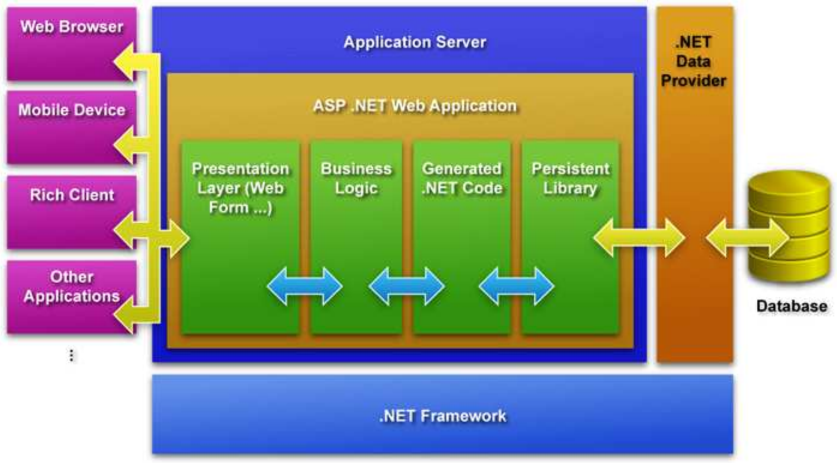


Figure 3.1 - The architecture of ASP .NET application with DB-VA Persistent Layer

In this chapter:

- Introduction
- Creating Object and Saving to Database
- Querying Object from Database
- Updating Object and Saving to Database
- Deleting Object in Database

Introduction

You will develop a School System.

The School System provides the following functions:

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

Required Software:

- DB Visual ARCHITECT 3.0 Java or Professional Edition (http://www.visual-paradigm.com/download/)
- Visual Studio .NET 2003 (http://msdn.microsoft.com/vstudio/previous/2003/)

Please open the SchoolSystem.vpp project file in the Chapter 3 School System.zip file. The project file contains the following diagrams. Please refer to the Designer Guide for details about how to draw class diagram and entity relationship diagram.

Class Diagram of School System:

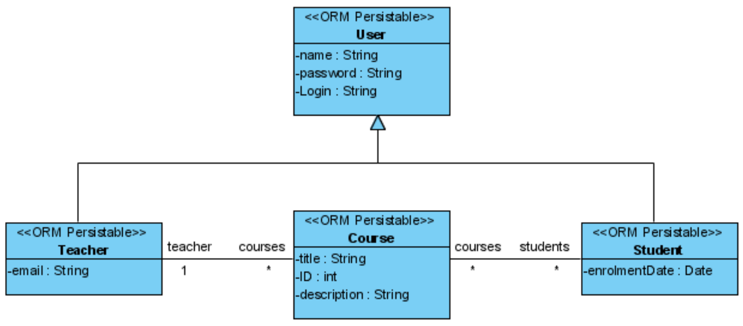


Figure 3.2 - Class Diagram

Entity Relationship Diagram of School System:

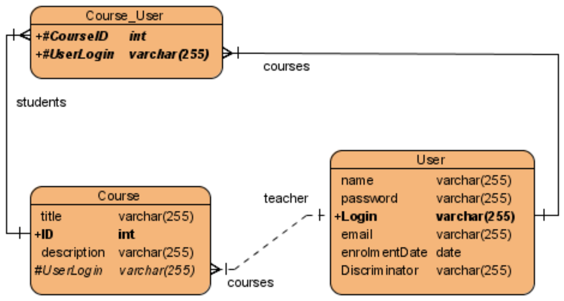


Figure 3.3 - Entity Relationship Diagram (ERD)

Creating Object and Saving to Database

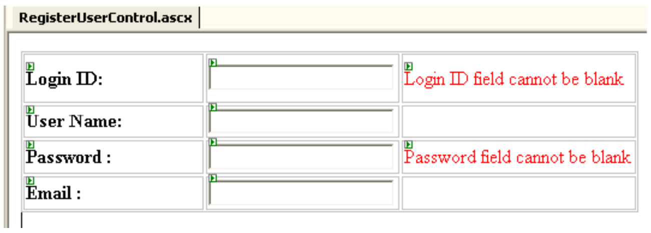
Before writing code to develop the ASP.NET application, you need to:

1. Generate the C# code for accessing database by DB-VA
2. Create the ASP.NET Web Application project.
3. Add generated C# project to Solution which contain the ASP.NET Web Application project.
4. Add the generated C# project and persistent library to the references of the ASP.NET Web Application project.

The details of how to setup the environment to develop the ASP.NET application, please refer to Chapter 2 - Configuring the Source and Library in Visual Studio.

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

1. Create the RegisterUserComponent. It contains Login ID, User Name, Password and Email Field for User to input information. It has RequestedFieldValidators to check the Login ID and Password field so they do not allow blank. It can be reused in teacher and student register page.



RegisterUserControl.ascx

Login ID:		Login ID field cannot be blank
User Name:		
Password :		Password field cannot be blank
Email :		

Figure 3.4 - The RegisterUserComponent

2. Create a Web Form called teacherReg.aspx and drag the RegisterUserControl to it.

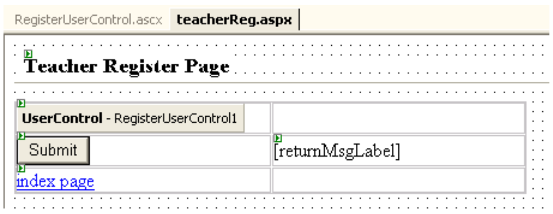


Figure 3.5 - The Web Form

3. Double click the Submit button to create the submitButton_Click method to handle the button click event.

```
private void submitButton_Click(object sender, System.EventArgs e)
{
...
}
```

Source Code : webschoolsystem\teacherReg.aspx.cs

4. Use RegisterUserComponent information to create the Teacher in system.
   - Get the RegisterUserComponent from the Page.

     ```
     userControl = (RegisterUserControl) Page.FindControl("RegisterUserControl1");
     ```

     Source Code : webschoolsystem\teacherReg.aspx.cs

   - Notify Session to begin transaction.

     ```
     PersistentTransaction t =
     SchoolSystemPersistentManager.Instance().GetSession().BeginTransaction();
     ```

     Source Code : webschoolsystem\teacherReg.aspx.cs

   - Create the Teacher instance and set the value to the Teacher instance properties

     ```
     try
     {
             Teacher lTeacher = TeacherFactory.CreateTeacher();
             lTeacher.LoginID = userControl.LoginID;
             lTeacher.Name = userControl.UserName;
             lTeacher.Password = userControl.Password;
     ```

     Source Code : webschoolsystem\teacherReg.aspx.cs

   - Save the Teacher instance and call transaction to commit.

     ```
     lTeacher.Save();
     t.Commit();
     ```

     Source Code : webschoolsystem\teacherReg.aspx.cs

   - Add the Teacher instance to Http Session for checking whether the user has login, and then redirect the user to the login page.

     ```
     Session.Add("user", lTeacher);
     Response.Redirect("login.aspx", true);
     ```

     Source Code : webschoolsystem\teacherReg.aspx.cs

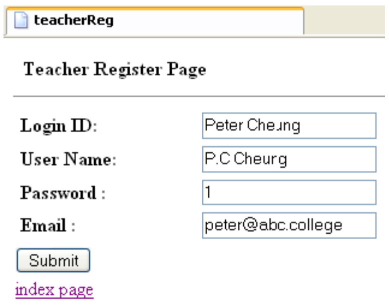
- Add the catch block to handle the exception and transaction can be rollback.

```
catch (Exception ex)
{
        Console.WriteLine(ex.StackTrace);
        t.RollBack();
        returnMsgLabel.Text = "Error to add a new teacher";
}
```

- Close the transaction session

```
SchoolSystemPersistentManager.Instance().GetSession().Close();
```

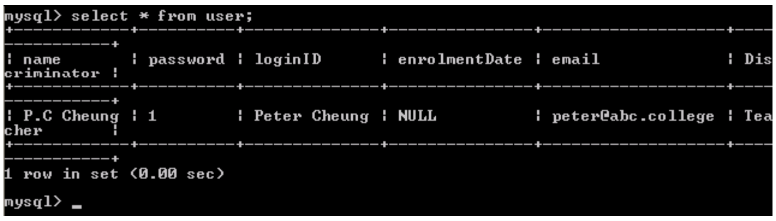
Source Code : webschoolsystem\teacherReg.aspx.cs



Teacher Register Page

Login ID:	Peter Cheung
User Name:	P.C Cheung
Password :	1
Email :	peter@abc.college

Figure 3.6 - Register on the page



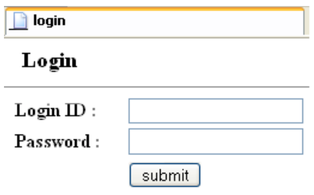
```
mysql> select * from user;
+--------------+-----------+---------------+----------------+---------------------+----
------------+
| name         | password  | loginID       | enrolmentDate  | email               | Dis
criminator |
+--------------+-----------+---------------+----------------+---------------------+----
------------+
| P.C Cheung  | 1          | Peter Cheung | NULL           | peter@abc.college | Tea
cher        |
+--------------+-----------+---------------+----------------+---------------------+----
------------+
1 row in set (0.00 sec)

mysql> _
```

Figure 3.7 - The record is insert to the database

Querying Object from Database

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

1. Create the Login.aspx. It is used for user to login to the school system.



Login

Login ID :	
Password :	

Figure 3.8 - Login page

2. Redirect the user to the student page or the teacher page when user is login. Try to get the User object from http session.

```
private void redirectLoginedUser()
{
        Object lObj = Session.Contents["user"];
        if (lObj != null)
        {
                if (lObj is Teacher)
                {
                        Response.Redirect("teacherPage.aspx", true);
                }
                else if (lObj is Student)
                {
                Response.Redirect("studentPage.aspx", true);
                }
        }
}
```

Source Code : webschoolsystem\login.aspx.cs

3. Get the User object by the user input Login ID and compare the user input password and the object password in the Submit button click event. If the password matches then user can access the system.
   - Get the user input information from the field.

     ```
     String lID = loginIDTextBox.Text;
     String lPassword = passwordTextBox.Text;
     ```

     Source Code : webschoolsystem\login.aspx.cs

   - Load User Object by LoadByUserORMID method.

     ```
     User lUser = UserFactory.LoadUserByORMID(lID);
     ```

     Source Code : webschoolsystem\login.aspx.cs

   - Check User Object is it null and compare the password of user input

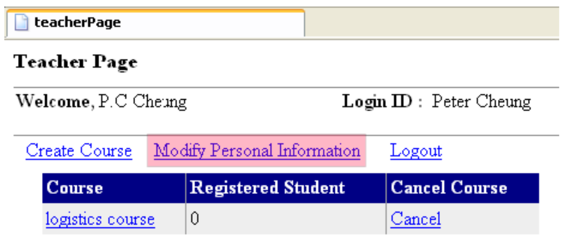
     ```
     if (lUser != null)
     {
             if (lUser.Password == lPassword)
             {
                     Session.Add("user", lUser);
                     redirectLoginedUser();
             }
             else
             {
                     returnMsgLabel.Text = "User ID or Password incorrect";
             }
     ```

     Source Code : webschoolsystem\login.aspx.cs

Updating Object and Saving to Database

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

1. Click Modify Personal Information hyperlink to edit the user information.



teacherPage

Teacher Page

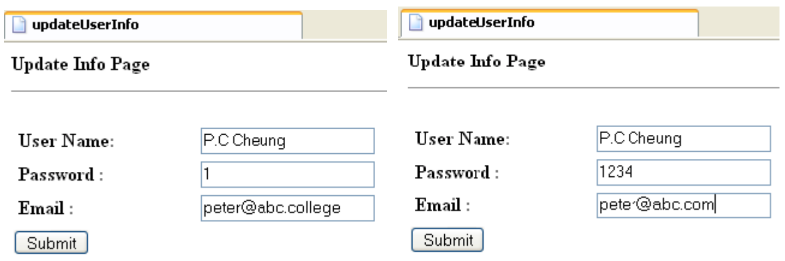
Welcome, P.C Cheung Login ID : Peter Cheung

Create Course Modify Personal Information Logout

Course	Registered Student	Cancel Course
logistics course	0	Cancel

Figure 3.9 - Teaching Page

2. It shows user information and allows you to update the user information except Login ID.



updateUserInfo

Update Info Page

User Name:	P.C Cheung
Password :	1
Email :	peter@abc.college

Submit

updateUserInfo

Update Info Page

User Name:	P.C Cheung
Password :	1234
Email :	pete@abc.com

Submit

Figure 3.10 - To modify the user information

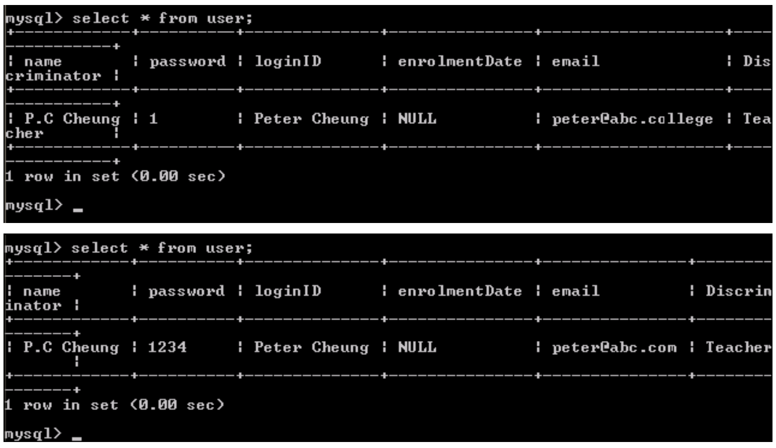
3. Click the submit button and it will set the updated information to the User object from the Http Session.

```
Object lObj = Session.Contents["user"];
if (lObj == null)
{
        Response.Redirect("login.aspx");
}
else
{
        school.SchoolSystemPersistentManager.Instance().GetSession().Lock(lObj,
        NHibernate.LockMode.None);
}
```

4. Update information of User object.

```
user.Name = registerUserControl.UserName;
user.Password = registerUserControl.Password;
if (user is Teacher)
{
        ((Teacher)user).Email = registerUserControl.Email;
}
user.Save();
t.Commit();
```

Source Code : webschoolsystem\login.aspx.cs



```
mysql> select * from user;
+-------------+----------+--------------+---------------+-------------------+----
------------+
| name        | password | loginID      | enrolmentDate | email             | Dis
criminator |
+-------------+----------+--------------+---------------+-------------------+----
------------+
| P.C Cheung | 1        | Peter Cheung | NULL          | peter@abc.college | Tea
cher       |
+-------------+----------+--------------+---------------+-------------------+----
------------+
1 row in set (0.00 sec)

mysql> _
```

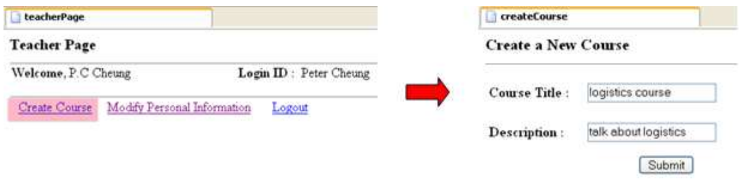
```
mysql> select * from user;
+-------------+----------+--------------+---------------+---------------+--------
-------+
| name        | password | loginID      | enrolmentDate | email         | Discrin
inator |
+-------------+----------+--------------+---------------+---------------+--------
-------+
| P.C Cheung | 1234     | Peter Cheung | NULL          | peter@abc.com | Teacher
       |
+-------------+----------+--------------+---------------+---------------+--------
-------+
1 row in set (0.00 sec)

mysql> _
```

Figure 3.11 - The record in database are modified

Deleting Object in Database

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

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



teacherPage

Teacher Page

Welcome, P.C Cheung Login ID : Peter Cheung

Create Course Modify Personal Information Logout

createCourse

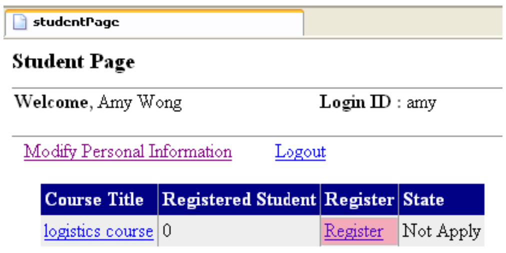
Create a New Course

Course Title : logistics course

Description : talk about logistics

Figure 3.12 - Delete record

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



studentPage

Student Page

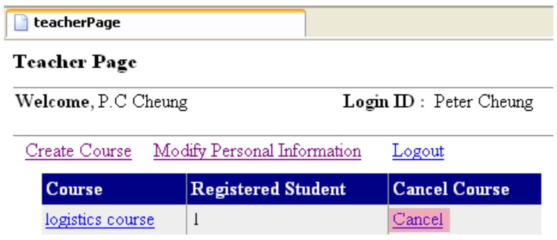
Welcome, Amy Wong Login ID : amy

Modify Personal Information Logout

Course Title	Registered Student	Register	State
logistics course	0	Register	Not Apply

Figure 3.13 - Student Page

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



teacherPage

Teacher Page

Welcome, P.C Cheung Login ID : Peter Cheung

Create Course Modify Personal Information Logout

Course	Registered Student	Cancel Course
logistics course	1	Cancel

Figure 3.14 - Teacher Page

4. Click Cancel of a Course then it will pass the course id to the function:
   - Use course id to load the Course object in teacherPage.aspx - deleteCourse() method

```
string delCourseID = Request.Params["delCourseID"];
PersistentTransaction t =
SchoolSystemPersistentManager.Instance().GetSession().BeginTransaction();
try
{
Course lCourse = CourseFactory.LoadCourseByORMID(int.Parse(delCourseID));
```

   - Call deleteAndDissociate() method to delete the course object and remove the relationship of student and teacher.

```
       lCourse.DeleteAndDissociate();
       t.Commit();
}
```

4 Deevloper Standalone .NET Application

Chapter 4 - Developing Standalone .NET Application

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

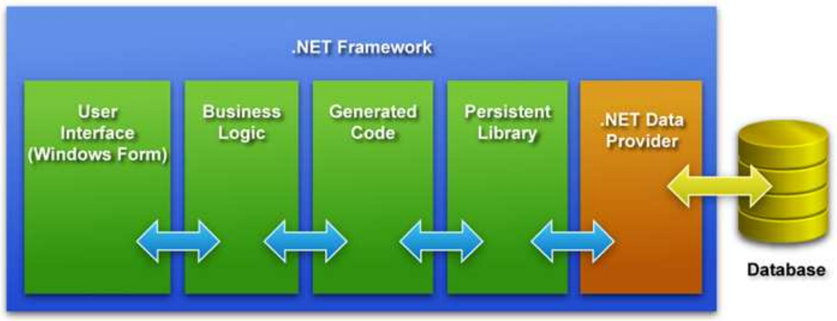


Figure 4.1 - The architecture of Standalone .NET Application with DB-VA Persistent Layer

In this chapter:

- Using the PersistentManager
- Creating Object and Saving to Database
- Querying Object from Database
- Updating Object and Saving to Database
- Deleting Object in Database

Introduction

You will develop a School System.
The School System provides the following functions:

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

Required Software:

- DB Visual ARCHITECT 3.0 Java or Professional Edition (http://www.visual-paradigm.com/download/)
- Visual Studio .NET 2003 (http://msdn.microsoft.com/vstudio/previous/2003/)

Class Diagram of School System:

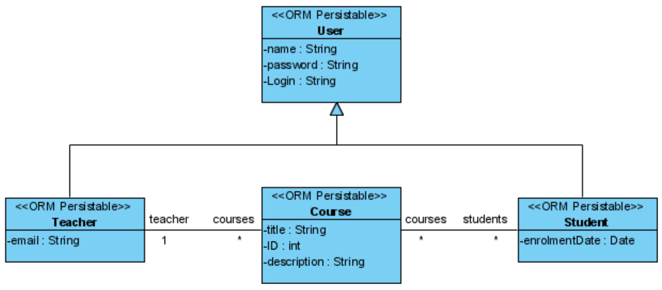


Figure 4.2 - Class Diagram

Entity Relationship Diagram of School System:

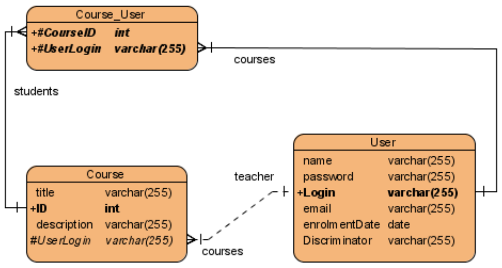


Figure 4.3 - Entity Relationship Diagram (ERD)

Using the PersistentManager

In DB-VA generated code there is a PersistentManager class. The PersistentManager can manage the database connection information and states of the persistent objects. When you create or update the persistent object, you can request the PersistentManager to get session and notify begin transaction.

Sample:

```
private void okButton_Click(object sender, System.EventArgs e)
{
        DialogResult = DialogResult.OK;
        if (titleTextBox.Text.Length > 0)
        {
                PersistentTransaction t =
                SchoolSystemPersistentManager.Instance().GetSession().BeginTransaction();
                try
                {
                        Course lCourse = CourseFactory.CreateCourse();
                        lCourse.Title = titleTextBox.Text;
                        lCourse.Description = descriptionTextBox.Text;
                        lCourse.Teacher = _teacher;
                        lCourse.Save();
                        t.Commit();
                        CreatedCourse = lCourse;
                        Close();
                }
                catch(Exception ex)
                {
                        Console.WriteLine(ex.InnerException.Message);
                        t.RollBack();
                }
        }
        else
        {
                MessageBox.Show("Missing Title");
        }
}
```

Source File : Standalone School System\CreateCourseDialog.cs

Creating Object and Saving to Database

In developing C# application, you need to generate code by DB-VA and configure the source code and library. The following example is a standalone .NET application, so you need to pay attention to the following settings when configure to generate C# code.

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

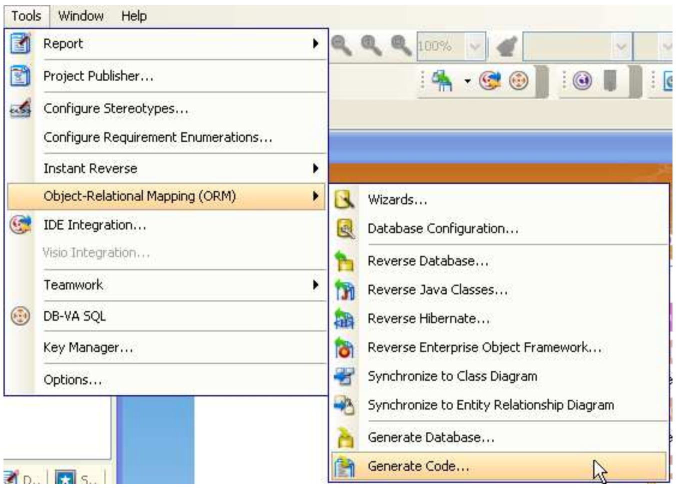
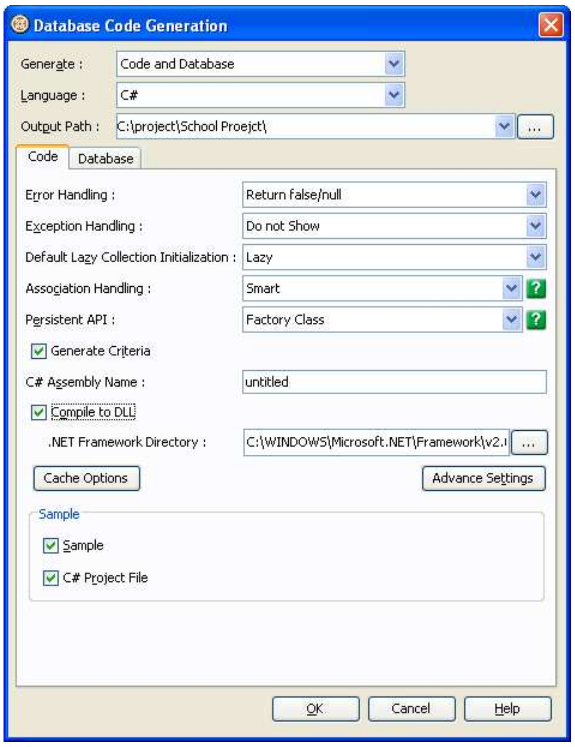


Figure 4.4 - Select generate code

2. Fill in code generation information.

For Language, select C#.
For Association Handling, select Smart.
For Persistent API, select Factory Class.



Database Code Generation

Generate : Code and Database
Language : C#
Output Path : C:\project\School Proejct\

Code | Database

Error Handling : Return false/null
Exception Handling : Do not Show
Default Lazy Collection Initialization : Lazy
Association Handling : Smart
Persistent API : Factory Class
☑ Generate Criteria
C# Assembly Name : untitled
☑ Compile to DLL
.NET Framework Directory : C:\WINDOWS\Microsoft.NET\Framework\v2.

Sample
☑ Sample
☑ C# Project File

Figure 4.5 - Database Code Generation dialog

For the other configuration details, please refer to Chapter 1 - Generating .NET Code, Database Schema (DDL) and Persistent Library and Chapter 2 - Configuring Source and Library in Visual Studio.

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

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

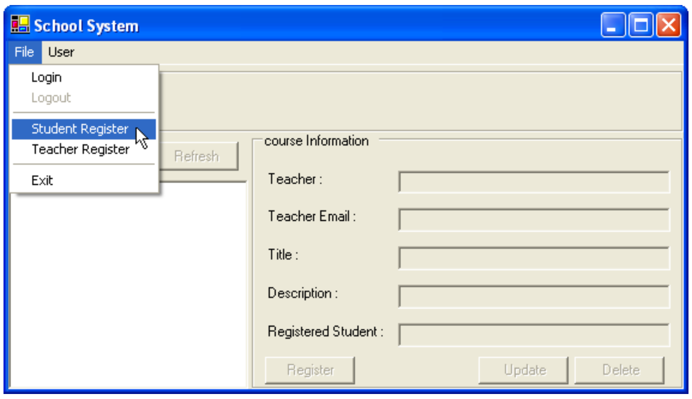
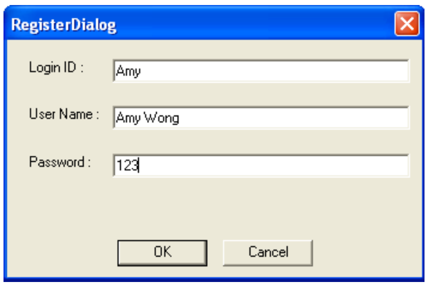


Figure 4.6 - School System

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



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

Figure 4.7 - Register dialog

5. After click OK, the system creates the new Student Persistent object.

```
Private void okButton_Click(object sender, System.EventArgs e)
{
        ...
        PersistentTransaction t =
        SchoolSystemPersistentManager.Instance().GetSession().BeginTransaction();
        try
        {
                if (_userType == CREATE_TEACHER)
                {
                        User = TeacherFactory.CreateTeacher();
                }
                else if (_userType == CREATE_STUDENT)
                {
                        User = StudentFactory.CreateStudent();
                }
```

Source File : Standalone School System\RegisterDialog.cs

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

```
User.Name = userNameTextBox.Text;
User.Password = passwordTextBox.Text;
User.LoginID = loginIDTextBox.Text;
if (User is Teacher)
{
        ((Teacher)User).Email = emailTextBox.Text;
}
```

Source File : Standalone School System\RegisterDialog.cs

7. Call Save() method of Student Persistent Object and call Commit() method of PersistentTransaction., then the new Student object will be saved in database. If the transaction has error occurred during the transaction, you can call the rollback() method to cancel the proposed changes in a pending database transaction.

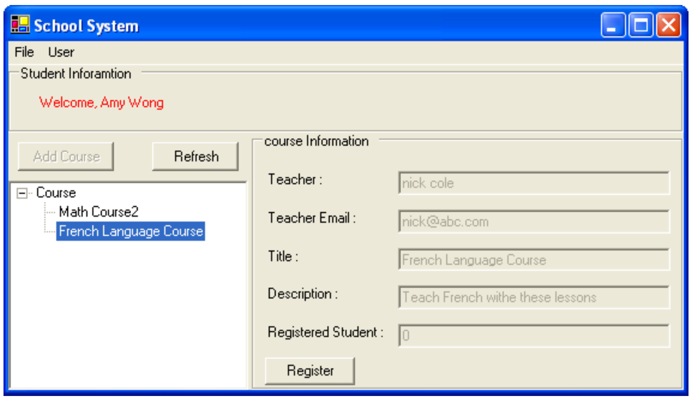
```
        User.Save();
        t.Commit();
        Close(); //close dialog
}
catch (Exception ex)
{
        Console.WriteLine(ex.InnerException.Message);
        t.RollBack();
}
```

Source File : Standalone School System\RegisterDialog.cs

Querying Object from Database

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

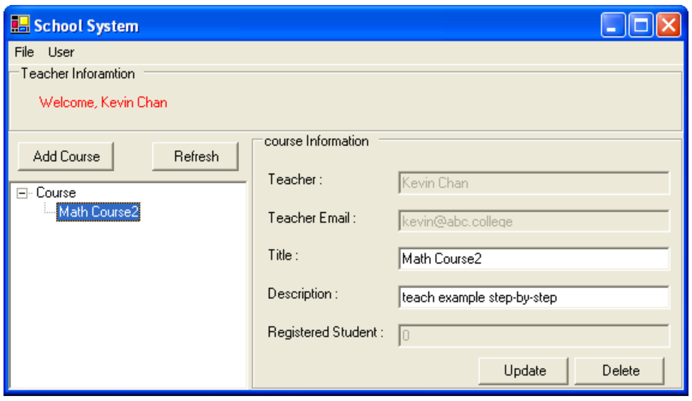
Student Login:



Teacher : nick cole
Teacher Email : nick@abc.com
Title : French Language Course
Description : Teach French withe these lessons
Registered Student : 0

Figure 4.8 - Student login to the system

Teacher Login:



Teacher Inforamtion

Welcome, Kevin Chan

Course
Math Course2

course Information

Teacher : Kevin Chan

Teacher Email : kevin@abc.college

Title : Math Course2

Description : teach example step-by-step

Registered Student : 0

Figure 4.9 - Teacher login to the system

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

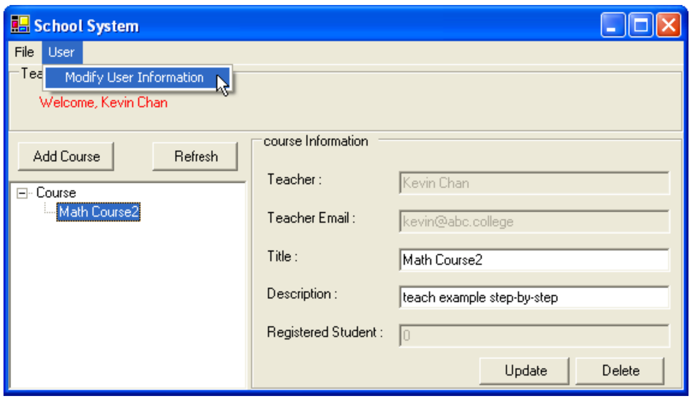
```
private void UpdateTreeView()
{
Course[] lCourses = null;
if (CurrentUser is Student)
{
lCourses = CourseFactory.ListCourseByQuery(null, null);
}
else if (CurrentUser is Teacher)
{
lCourses = ((Teacher)CurrentUser).courses.ToArray();
}
```

Source File : Standalone School System\SchoolSystemForm.cs

Updating Object and Saving to Database

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

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



Tea

Welcome, Kevin Chan

Course
Math Course2

course Information

Teacher : Kevin Chan

Teacher Email : kevin@abc.college

Title : Math Course2

Description : teach example step-by-step

Registered Student : 0

Figure 4.10 - To modify user information

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

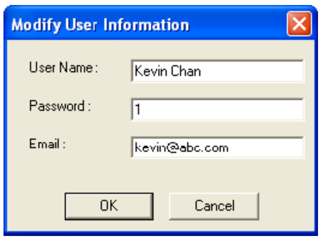


Figure 4.11 - Modify User Information dialog

3. Update the information for the User object includes password, name and email address. This operation is just as simple as the create User object process.

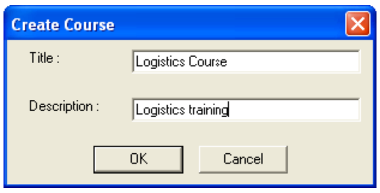
```
private void okButton_Click(object sender, System.EventArgs e)
{
        if (nameTextBox.Text.Length == 0 || passwordTextBox.Text.Length == 0)
        {
                MessageBox.Show("Missing user name or password");
        }
        else
        {
                PersistentTransaction t =
                SchoolSystemPersistentManager.Instance().GetSession().BeginTransaction();
                try
                {
                        _user.Name = nameTextBox.Text;
                        _user.Password = passwordTextBox.Text;
                        if(_user is Teacher)
                        {
                                ((Teacher)_user).Email = emailTextBox.Text;
                        }
                        _user.Save();
                        DialogResult = DialogResult.OK;
                        t.Commit();
                }
                catch (Exception)
                {
                        DialogResult = DialogResult.Cancel;
                        t.RollBack();
                }
                Close();
        }
}
```

Source File : Standalone School System\ModifyUserDialog.cs

Deleting Object in Database

Teachers can create courses for students to register and they can cancel courses in the system. They only need to click **delete** button of the course then the course information will be deleted in the database, and its relationships with registered students will be removed.

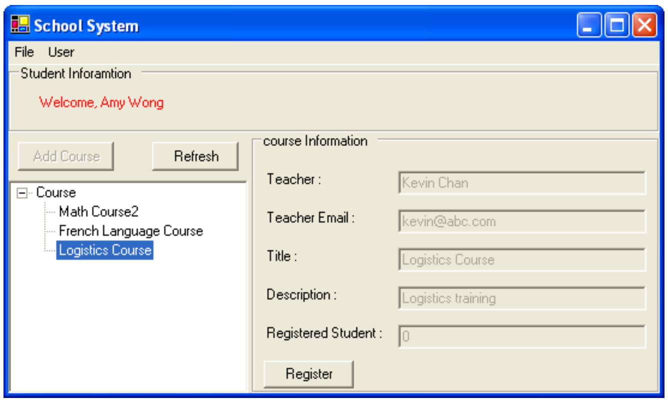
1. Teacher can create the course by clicking the **Add Course** button, fill in Course name and Description to create Course



Title : Logistics Course
Description : Logistics training

Figure 4.12 - Create Course dialog

2. Student can register the course by clicking the Register button.



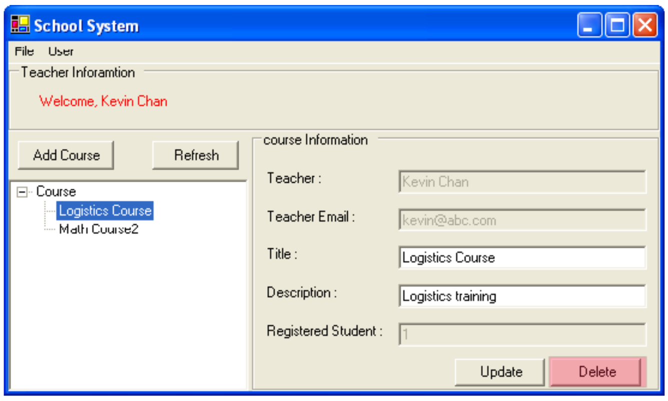
Student Inforamtion
Welcome, Amy Wong

Course
- Math Course2
- French Language Course
- Logistics Course

course Information
Teacher : Kevin Chan
Teacher Email : kevin@abc.com
Title : Logistics Course
Description : Logistics training
Registered Student : 0

Figure 4.13 - Course are created

3. The teacher can view how many students registered his course, and he can delete the course in system.



Teacher Inforamtion
Welcome, Kevin Chan

Course
- Logistics Course
- Math Course2

course Information
Teacher : Kevin Chan
Teacher Email : kevin@abc.com
Title : Logistics Course
Description : Logistics training
Registered Student : 1

Figure 4.14 - Delete a course

4. Click Delete of a Course then it will trigger the delButton_Click() method .

```
private void delButton_Click(object sender, System.EventArgs e)
{
	if (MessageBox.Show("Delete", "Delete", MessageBoxButtons.OKCancel) ==
	DialogResult.OK)
	{
```

Source File : Standalone School System\CoursePanel.cs

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

```
Try
{
	_courseNode.Course.DeleteAndDissociate();
	_courseNode.Remove();
	t.Commit();
}
catch (Exception ex)
{
	Console.WriteLine(ex.InnerException.Message);
	t.RollBack();
}
```

Source File : Standalone School System\CoursePanel.cs

5 Querying Database

Chapter 5 - Querying Database

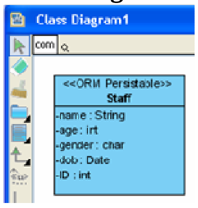
DB Visual ARCHITECT(DB-VA) provides two features to query database. You can use the ORM Qualifier and Criteria to define you requirements to retrieve the data from database. They provide a much simpler way to retrieve records from the database than SQL query.

In this chapter:

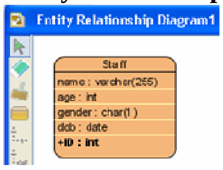
Introduction

If you want to provide the search function and make an easy way to retrieve records from the database, then you can use ORM Qualifier or Generate Criteria in DB-VA, in this chapter, we use an example to tell you that how to use the criteria class to search the persistent objects with your defined condition. First of all, you need to create the Class Diagram and synchronizes to Entity Relationship Diagram.

Class Diagram:



Entity Relationship Diagram:



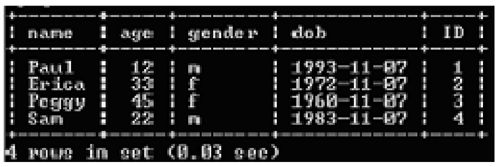
Creating Test Data

You can create a staff record in the database and test how the criteria class can help you to retrieve the record.

1. Open CreateUntitledData.cs and modify the code with the following content:

```
private void CreateData() {
            PersistentTransaction t = 
com.UntitledPersistentManager.Instance().GetSession().BeginTransaction();
            try {
                com.Staff lcomStaff = com.StaffFactory.CreateStaff();
                // Initialize the properties of the persistent object
                lcomStaff.Name = "Paul";
                lcomStaff.Age = 12;
                lcomStaff.Dob = new DateTime(1993, 11, 7);
                lcomStaff.Gender = 'm';
                lcomStaff.Save();

                lcomStaff = com.StaffFactory.CreateStaff();
                lcomStaff.Name = "Erica";
                lcomStaff.Age = 33;
                lcomStaff.Dob = new DateTime(1972, 11, 7);
                lcomStaff.Gender = 'f';
                lcomStaff.Save();

                lcomStaff = com.StaffFactory.CreateStaff();
                lcomStaff.Name = "Peggy";
                lcomStaff.Age = 45;
                lcomStaff.Dob = new DateTime(1960, 11, 7);
                lcomStaff.Gender = 'f';
                lcomStaff.Save();

                lcomStaff = com.StaffFactory.CreateStaff();
                lcomStaff.Name = "Sam";
                lcomStaff.Age = 22;
                lcomStaff.Dob = new DateTime(1983, 11, 7);
                lcomStaff.Gender = 'm';
                lcomStaff.Save();
                t.Commit();
            }
            catch(Exception e) {
                t.RollBack();
                Console.WriteLine(e);
            }
…
}
```

2. Execute the CreateUntitledData.class. The new record is added.



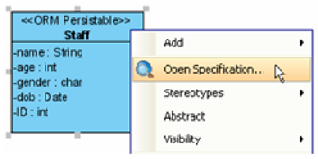
```
+--------+-----+--------+------------+----+
| name   | age | gender | dob        | ID |
+--------+-----+--------+------------+----+
| Paul   |  12 | m      | 1993-11-07 |  1 |
| Erica  |  33 | f      | 1972-11-07 |  2 |
| Peggy  |  45 | f      | 1960-11-07 |  3 |
| Sam    |  22 | m      | 1983-11-07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

Using ORM Qualifier

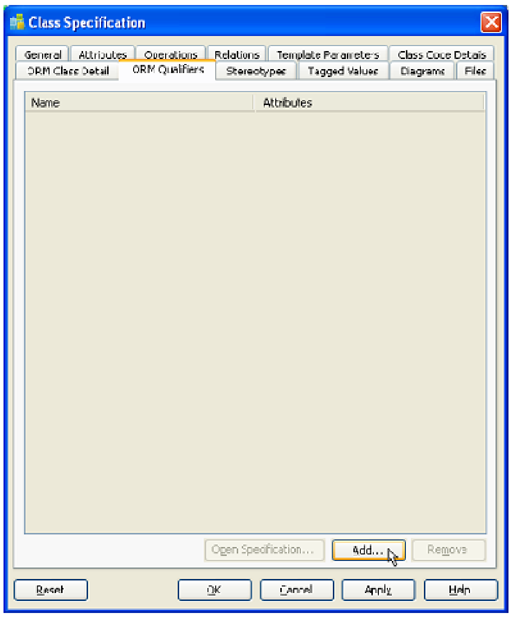
ORM Qualifier is an additional feature of DB-VA allowing you to specify the extra data retrieval rules apart from the system pre-defined rules. ORM Qualifier can be defined when you generate persistence code.

Defining ORM Qualifier

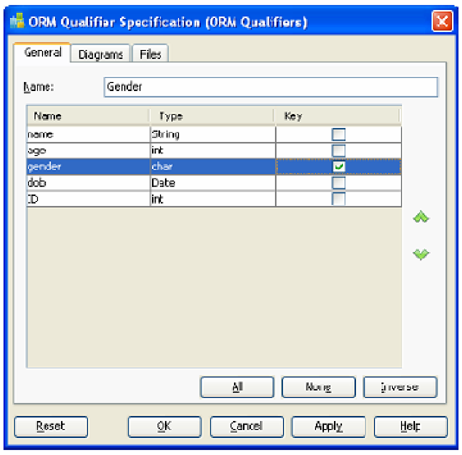
1. Right-click on the **Staff** class, select **Open Specification…**.



2. Click the **ORM Qualifiers** tab, then click **Add…**.



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



Name: Gender

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

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

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

Retrieving From ORM Qualifier

After you have created the "**Gender**" qualifier, you can use it to load or list the Staff data from database. The following are examples to load or list records by ORM qualifier with the generated sample code.

After you have created the "**Gender**" qualifier, you can use it to load or list the Staff data from database. The following are examples to load or list records by ORM qualifier with the generated sample code.

By Load method:

```
System.Console.WriteLine("Retrieving Staff by gender...");

System.Console.WriteLine(com.StaffFactory.LoadByGender('m'));
```

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

```
Retrieving Staff by gender...
Staff[ Name=Paul Age=12 Gender=m Dob=12/7/1993 12:00:00 AM ID=1 ]
```

By List method:

```
System.Console.WriteLine("Retrieving Staffs by gender...");
foreach(com.Staff lStaff in com.StaffFactory.ListByGender('f'))
{
    System.Console.WriteLine(lStaff);

}
```

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

```
Retrieving Staffs by gender...
Staff[ Name=Erica Age=33 Gender=f Dob=12/7/1972 12:00:00 AM ID=2 ]
Staff[ Name=Peggy Age=45 Gender=f Dob=12/7/1960 12:00:00 AM ID=3 ]
```

Using Criteria

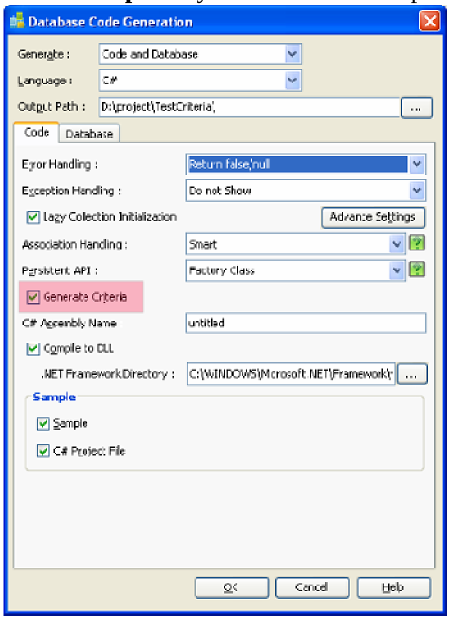
When generating the persistence class for each ORM-Persistable class defined in the object model, the corresponding criteria class can also be generated.

Configuring Criteria Class Generation

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

1. From the menu, select **Tools** > **Object-Relational Mapping (ORM)** > **Generate Code…** to open the **Database Code Generation** dialog box.
2. Select the **Generate Criteria** option. The other settings can be set to follow the picture below.

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


Database Code Generation

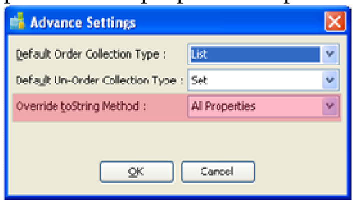
Generate : Code and Database
Language : C#
Output Path : D:\project\TestCriteria\

Code | Database

Error Handling : Return false/null
Exception Handling : Do not Show
[✓] Lazy Collection Initialization
Association Handling : Smart
Persistent API : Factory Class
[✓] Generate Criteria
C# Assembly Name : untitled
[✓] Compile to DLL
.NET Framework Directory : C:\WINDOWS\Microsoft.NET\Framework\

Sample
[✓] Sample
[✓] C# Project File

Open the Advance Settings dialog box and set the **Override toString Method** to **All Properties**. It can help you to print out the properties of persistent objects. Click OK to generate the code with Criteria Class.


Advance Settings

Default Order Collection Type : List

Default Un-Order Collection Type : Set

Override toString Method : All Properties

For more details of how to configure the database and generate code, please refer to Chapter 1 **"Generate C# .NET code, Database Schema (DDL) and Persistent Library"**.

Description of Criteria Class

The following is the generated Staff class's Criteria Class call "**StaffCriteria**".

```
namespace com {
    public class StaffCriteria : AbstractORMCriteria {
        private StringExpression _name;
        public StringExpression Name {
            get {
                return  _name;
            }

        }

        private Int32Expression _age;
        public Int32Expression Age {
            get {
                return  _age;
            }

        }

        private CharExpression _gender;
        public CharExpression Gender {
            get {
                return  _gender;
            }

        }

        private DateTimeExpression _dob;
        public DateTimeExpression Dob {
            get {
                return  _dob;
            }

        }

        private Int32Expression _ID;
        public Int32Expression ID {
            get {
                return  _ID;
            }

        }

        public StaffCriteria(PersistentSession session) :
base(session.CreateCriteria(typeof(Staff))) {
            _name =  new StringExpression("Name", this);
            _age =  new Int32Expression("Age", this);
            _gender =  new CharExpression("Gender", this);
            _dob =  new DateTimeExpression("Dob", this);
            _ID =  new Int32Expression("ID", this);
        }

        public StaffCriteria() :
this(com.UntitledPersistentManager.Instance().GetSession()) {
        }

        public Staff UniqueStaff() {
            return (Staff)base.UniqueResult();
        }

        public Staff[] ListStaff() {
            IList lList = base.List();
            Staff[] lValues = new Staff[lList.Count];
            lList.CopyTo(lValues, 0);
            return lValues;
        }

}
```

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

To apply the restriction to the property, call the method:

criteria.property.expression(parameter);

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

The table below shows the expressions used for specifying the condition for query.

Expression	**Description**
`Eq(value)`	The value of the property is equal to the specified value.
`Ne(value)`	The value of the property is not equal to the specified value.
`Gt(value)`	The value of the property is greater than the specified value.

`Ge(value)`	The value of the property is greater than or equal to the specified value.
`Lt(value)`	The value of the property is less than the specified value.
`Le(value)`	The value of the property is less than or equal to the specified value.
`IsEmpty()`	The value of the property is empty.
`IsNotEmpty()`	The value of the property is not empty.
`IsNull()`	The value of the property is NULL.
`IsNotNull()`	The value of the property is not NULL.
`In(values)`	The value of the property contains the specified values in the array.
`Between(value1, value2)`	The value of the property is between the two specified values, value1 and value2.
`Like(value)`	The value of the property matches the string pattern of value; use % in value for wildcard.
`Ilike(value)`	The value of the property matches the string pattern of value, ignoring case differences.

For example:

```
staffCriteria.Age.Ge(13);
```

There are two types of ordering to sort the retrieved records - ascending and descending.

To sort the retrieved records with respect to the property, call the method:

criteria.property.Order(ascending_order);

where the value of ascending_order is either true or false. Pass true to sort the property in ascending order, or pass false to sort the property in descending order.

For example:

```
staffCriteria.Age.Order(true);
```

To set the range of the number of records to be retrieved, use one of these two methods:

SetFirstResult(int i) – Retrieve the i-th record from the results as the first result.

SetMaxResult(int i) – Set the maximum number of retrieved records by the specified value i.

For example:

```
staffCriteria.SetMaxResults(100);
```

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

UniqueClass() – Retrieve a single record matching the specified condition(s) for the criteria; Exception will be thrown if the number of retrieved record is not 1.

ListClass() – Retrieve the records matched with the specified condition(s) for the criteria.

For example:

```
com.Staff[] staffs = staffCriteria.ListStaff();
```

Comparing Criteria Class and SQL Query

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

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

- By Specifying one criteria

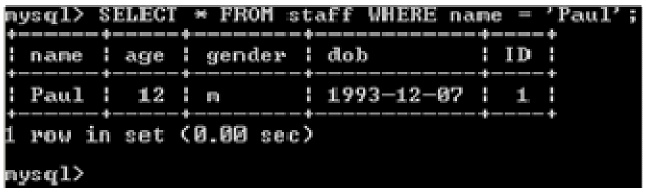
Retrieve a staff record whose name is "Paul":

Criteria Class	**SQL Query**

`com.StaffCriteria staffCriteria = new com.StaffCriteria();`
`staffCriteria.Name.Eq("Paul");`
`staffCriteria.SetMaxResults(ROW_COUNT);`
`com.Staff[] staffs = staffCriteria.ListStaff();`
`int length =staffs== null ? 0 : Math.Min(staffs.Length, 100);`
 	
`for (int i = 0; i < length; i++) {`
`    System.Console.WriteLine(staffs[i]);`
`}`
`System.Console.WriteLine(length + " Staff record(s) retrieved.");`
	SELECT * FROM staff WHERE name = 'Paul';

The Result:

Criteria Class	**SQL Query**

`Staff[ Name=Paul Age=12 Gender=m Dob=11/7/1993 12:00:00 AM ID=1 ]`
`1 Staff record(s) retrieved.`
	

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

mysql>
```

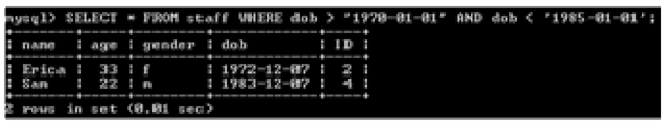
Retrieve all staff records whose date of birth is between 1/1/1970 and 31/12/1985:

Criteria Class	SQL Query

`com.StaffCriteria staffCriteria = new com.StaffCriteria();`
`staffCriteria.Dob.Between(new DateTime(1970,1,1), new DateTime(1985,12,31));`
`staffCriteria.SetMaxResults(ROW_COUNT);`
`com.Staff[] staffs = staffCriteria.ListStaff();`
`int length =staffs== null ? 0 : Math.Min(staffs.Length, ROW_COUNT);`
`for (int i = 0; i < length; i++) {`
`    System.Console.WriteLine(staffs[i]);`
`}`
`System.Console.WriteLine(length + " Staff record(s) retrieved.");`
	SELECT * FROM staff WHERE dob > '1970-01-01' AND dob < '1985-01-01';

The Result:

Criteria Class	SQL Query

`Staff[ Name=Erica Age=33 Gender=f Dob=11/7/1972 12:00:00 AM ID=2 ]`
`Staff[ Name=Sam Age=22 Gender=m Dob=11/7/1983 12:00:00 AM ID=4 ]`
`2 Staff record(s) retrieved.`
	

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

- By Specifying more than one criteria

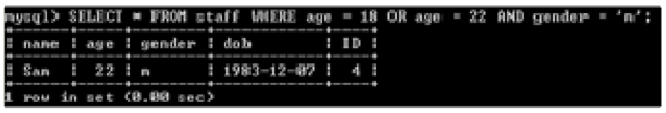
Retrieve all male staff records whose age is between 18 and 22:

Criteria Class	SQL Query

`com.StaffCriteria staffCriteria = new com.StaffCriteria();`
`staffCriteria.Age.In(new int[]{18, 22});`
`staffCriteria.Gender.Eq('m');`
`staffCriteria.SetMaxResults(ROW_COUNT);`
`com.Staff[] staffs = staffCriteria.ListStaff();`
`int length =staffs== null ? 0 : Math.Min(staffs.Length, ROW_COUNT);`
`for (int i = 0; i < length; i++) {`
`    System.Console.WriteLine(staffs[i]);`
`}`
`System.Console.WriteLine(length + " Staff record(s) retrieved.");`
	SELECT * FROM staff WHERE age = 18 OR age = 22 AND gender = 'm';

The Result:

Criteria Class	**SQL Query**

`Staff[ Name=Sam Age=22 Gender=m Dob=11/7/1983 12:00:00 AM ID=4 ]`
`1 Staff record(s) retrieved.`
	

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

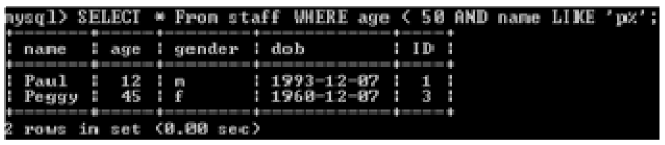
Retrieve all staff records whose name starts with "P" and age is less than 50, ordering by the name:

Criteria Class	**SQL Query**

`com.StaffCriteria staffCriteria = new com.StaffCriteria();`
`staffCriteria.Name.Like("P%");`
`staffCriteria.Age.Lt(50);`
`staffCriteria.Name.Order(true);`
`staffCriteria.SetMaxResults(ROW_COUNT);`
`com.Staff[] staffs = staffCriteria.ListStaff();`
`int length =staffs== null ? 0 : Math.Min(staffs.Length, ROW_COUNT);`
`for (int i = 0; i < length; i++) {`
`    System.Console.WriteLine(staffs[i]);`
`}`
`System.Console.WriteLine(length + " Staff record(s) retrieved.");`
	SELECT * From staff WHERE age < 50 AND name LIKE 'p%' ORDER BY name;

The Result:

Criteria Class	**SQL Query**

`Staff[ Name=Paul Age=12 Gender=m Dob=11/7/1993 12:00:00 AM ID=1 ]`
`Staff[ Name=Peggy Age=45 Gender=f Dob=11/7/1960 12:00:00 AM ID=3 ]`
`2 Staff record(s) retrieved.`
	

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

6 Generating Object-oriented .NET Source from Relational Database

Chapter 6 - Generating Object-Oriented .NET Source from Relational Database

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

In this chapter:

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

Introduction

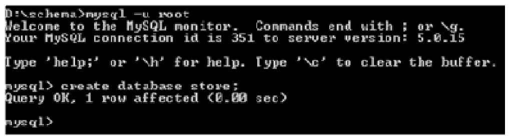
In this chapter, we will use a MySQL database. You need to import schema to the MySQL database. This schema is used for simulating an existing database schema that you will reverse in DB-VA into Class Diagram and Entity Relationship Diagram. And you will generate Object-Oriented .NET source from the reversed model. Before you start to reverse the schema, you should make sure you have the MySQL 5.0 Community Edition installed.

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

Creating Sample Data

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

```
mysql –u root

create database store;

exit;
```



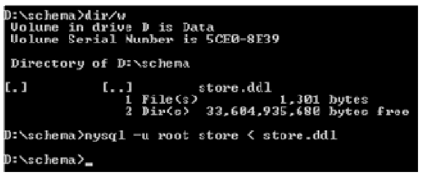
```
D:\schema>mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 351 to server version: 5.0.15

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

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

mysql>
```

2. Use Command Prompt to change to the directory that contains the sample schema. The following is content of the sample schema (store.ddl).

```
create table contacts (TOrderIndex int not null unique, contact varchar(255), TOrderID int not null, primary key (TOrderIndex, TOrderID)) type=InnoDB;

create table customer (ID varchar(255) not null unique, name varchar(255), discount double, primary key (ID)) type=InnoDB;

create table orderline (ID int not null auto_increment unique, quantity int not null, OrderID int not null, primary key (ID)) type=InnoDB;

create table product (ID int not null auto_increment unique, name varchar(255), OrderLineID int not null, primary key (ID)) type=InnoDB;

create table torder (ID int not null auto_increment unique, OrderDate date, CustomerID varchar(255) not null, primary key (ID)) type=InnoDB;

alter table contacts add index FK_Contacts_7690 (TOrderID), add constraint FK_Contacts_7690 foreign key (TOrderID) references torder (ID);

alter table orderline add index FK_OrderLine_9672 (OrderID), add constraint FK_OrderLine_9672 foreign key (OrderID) references torder (ID);

alter table product add index FK_Product_5274 (OrderLineID), add constraint FK_Product_5274 foreign key (OrderLineID) references orderline (ID);

alter table torder add index FK_TOrder_4869 (CustomerID), add constraint FK_TOrder_4869 foreign key (CustomerID) references customer (ID);
```

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

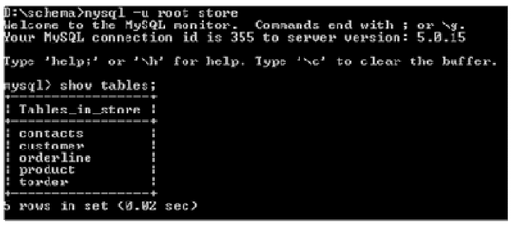
```
mysql –u root store < store.ddl
```



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

 Directory of D:\schema

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

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

D:\schema>_
```

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

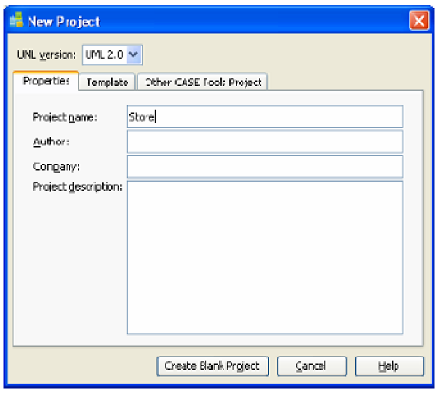
```
mysql -u root store

show tables;
```



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

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

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

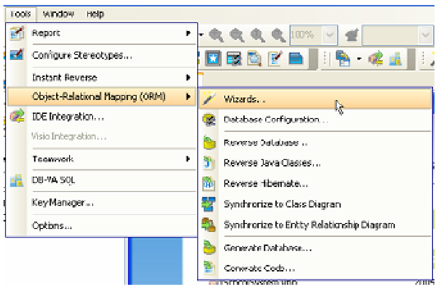
Generating Code from Database Wizard

1. New Project in DB-VA called "Store".

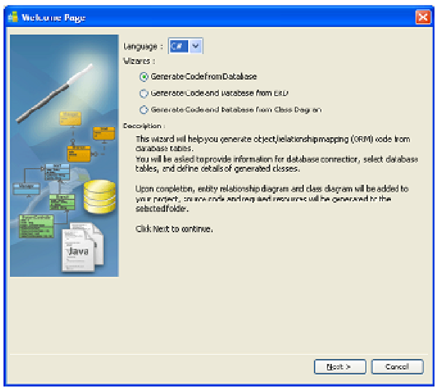


Project name: Store

2. From the menu, select **Tools** > **Object-Relational Mapping (ORM)** > **Wizard…** to open the Wizard.



3. Select **C#** in Language and select **Generate Code from Database** on the Wizard Welcome page and then click **Next** >.


Welcome Page
Wizards :
Generate Code and Database from ERD
Generate Code and Database from Class Diagram
Next >
Cancel

Language : C#

[✓ Generate Code from Database]

Description :

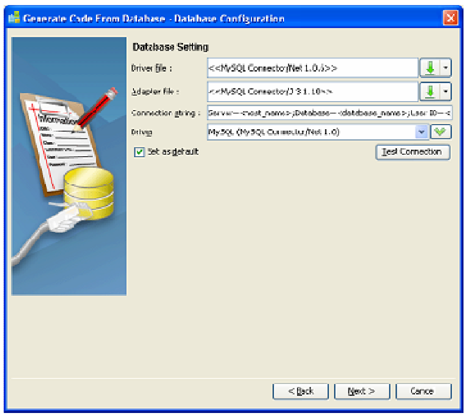
This wizard will help you generate object/relationship mapping (ORM) code from database tables.
You will be asked to provide information for database connection, select database tables, and define details of generated classes.

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

Click Next to continue.

Configuring Database

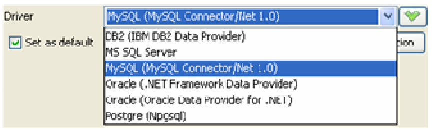
In this step, you need to select the database and enter the database information, then DB-VA will use this information to get the database schema to reverse.


Generate Code From Database - Database Configuration

Database Setting

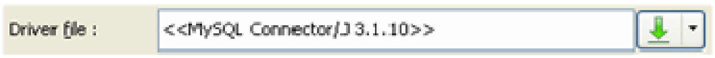
Driver file : <<MySQL Connector/Net 1.0.6>>
Adapter file : <<MySQL Connector/J 3.1.10>>
Connection string : Server=<host_name>;Database=<database_name>;User ID=<
Driver : MySQL (MySQL Connector/Net 1.0)
[✓] Set as default

1. Select **MySQL (MySQL Connector/Net 1.0)** for Driver option.

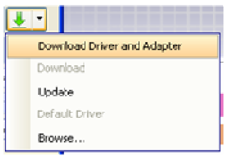

Driver : MySQL (MySQL Connector/Net 1.0)
[✓] Set as default

DB2 (IBM DB2 Data Provider)
MS SQL Server
MySQL (MySQL Connector/Net 1.0)
Oracle (.NET Framework Data Provider)
Oracle (Oracle Data Provider for .NET)
Postgre (Npgsql)

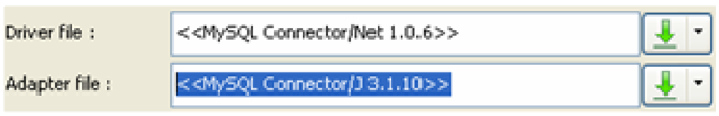
2. Download or browse the suitable **Driver** and **Adapter** file for your selected driver.


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

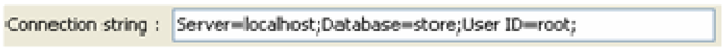
For **Driver** and **Adapter Driver** file, you can click the **Driver** button to select **Download Driver and Adapter**, **Download**, **Update**, and **Default Driver**, DB-VA helps you to download the most up-to-date driver and adapter driver according to the Driver field information. You can also select **Browse…** to select a driver and adapter driver file in your computer.



After downloaded the driver file, **<<MySQL Connector/Net 1.0.6>>** will be shown on **Driver file** and <<MySQL Connector/J 3.1.10>> will be shown on **Adapter file**.


Driver file : <<MySQL Connector/Net 1.0.6>>
Adapter file : <<MySQL Connector/J 3.1.10>>

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


Connection string : Server=localhost;Database=store;User ID=root;

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

The default Connection URL template for MySQL is:

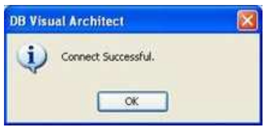
```
Server=<host_name>;Database=<database_name>;User
ID=<username>;Password=<password>;CharSet=<charset>
```

A sample Connection URL for MySQL is:

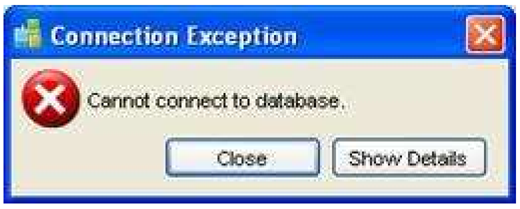
```
Server=localhost;Database=store;User ID=root;
```

4. Click **Test Connection** to test the database connection.

If success to connect with database the Connection Successful dialog box will show, otherwise the Connection Exception dialog will show.


DB Visual Architect

Connect Successful.

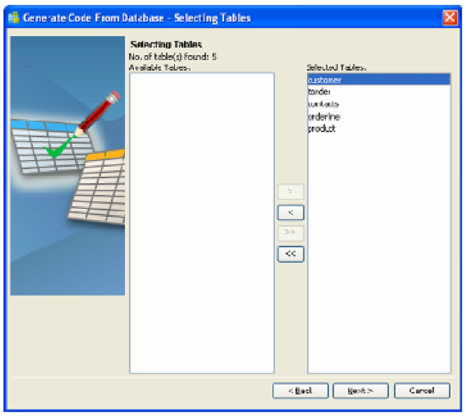

Connection Exception

Cannot connect to database.

5. Click **Next >** to select tables.

Selecting Table

DB-VA uses your previously configured database settings to connect to the database. You can select the database tables which you want to generate persistent classes for. In this example, we will select all the database tables to reverse. You can deselect tables by using the list of buttons between the list of **Available Tables** and **Selected Tables**.


Generate Code From Database - Selecting Tables

Selecting Tables

No. of table(s) found: 5

Available Tables:

Selected Tables:

customer
torder
contacts
orderline
product

- > Add Selected

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

- < Remove Selected

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

- >> Add All

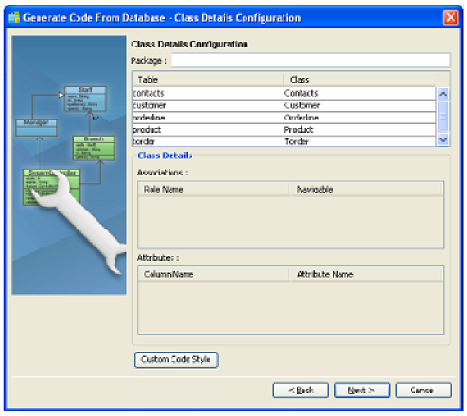
  Add all tables from **Available Tables** to **Selected Tables**.

- << Remove All

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

Configuring Generated Class Details

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


Generate Code From Database - Class Details Configuration

Class Details Configuration

Package :

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

Class Details

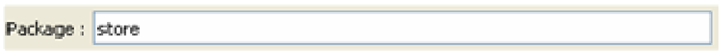
Associations :

Role Name	Navigable

Attributes :

ColumnName	Attribute Name

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


Package : store

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

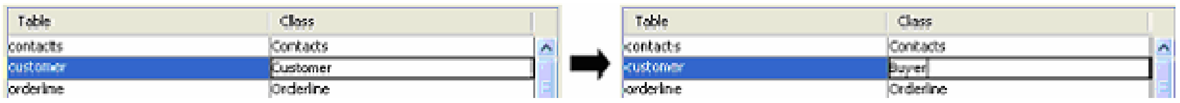
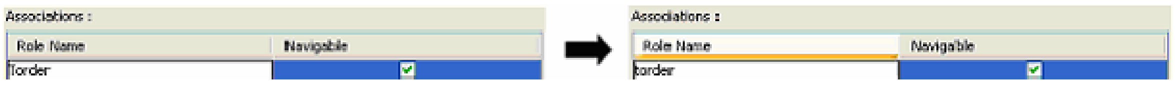


Table	Class
contacts	Contacts
customer	Customer
orderline	Orderline

Table	Class
contacts	Contacts
customer	Buyer
orderline	Orderline

3. Change the first character of the Buyer class's Association Role Name from upper case to lower case. You can edit the role name for a reference in the class.

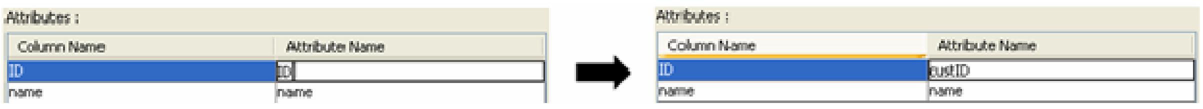

Associations :

Role Name	Navigable
Torder	✓

Associations :

Role Name	Navigable
torder	✓

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



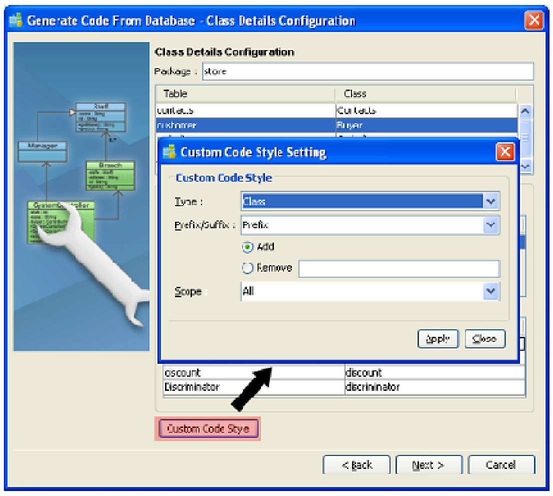
Attributes :

Column Name	Attribute Name
ID	ID
name	name

Attributes :

Column Name	Attribute Name
ID	custID
name	name

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



Type : Class
Prefix/Suffix : Prefix
(● Add) Remove
Scope : All

Generate Code From Database - Class Details Configuration
Class Details Configuration
Package : store
Custom Code Style Setting
Custom Code Style
Apply
Close
Custom Code Style
< Back
Next >
Cancel

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

For **Prefix/Suffix**, select either Prefix or Suffix to be added or removed.

For **Add/Remove** option, select the option for the action of code style to be applied.

For the textbox, enter the word for either prefix or suffix.

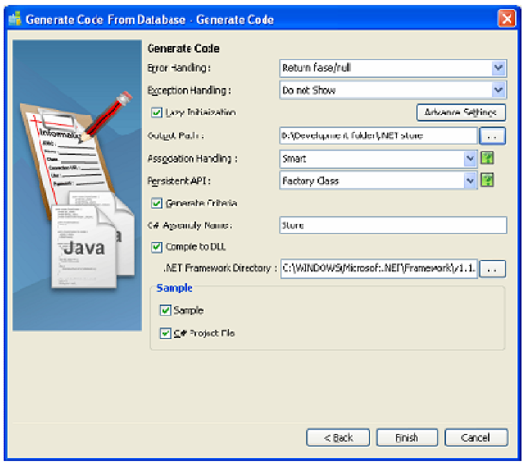
For **Scope**, select the scope of the code style to be applied to, either All or Selected.

The table below shows the result of applying code styles.

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

Specifying Code Generation Details

This is the final step to specify .NET code generation details, you can select the location of the code generation, C# Assembly Name, etc… according to your requirements.



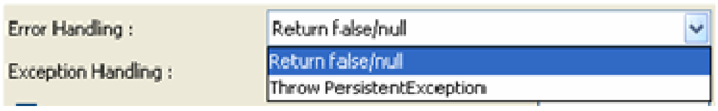
Generate Code From Database - Generate Code

Generate Code
Error Handling: Return false/null
Exception Handling: Do not Show
✓ Lazy Initialization
Output Path: D:\Development Folder\.NET store
Association Handling: Smart
Persistent API: Factory Class
✓ Generate Criteria
C# Assembly Name: Store
✓ Compile to DLL
.NET Framework Directory: C:\WINDOWS\Microsoft.NET\Framework\v1.1.
Sample
✓ Sample
✓ C# Project File

- **Error Handling**

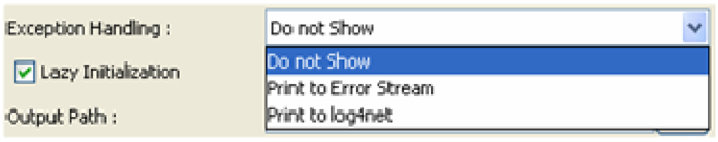
  Select the way to handle errors. The possible errors include PersistentException, ADOException.

  - **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.



Error Handling: Return false/null
Exception Handling:

- **Exception Handling**
  - **Do not Show** - It hides the error message.
  - **Print to Error Stream** – It prints the error message to the Error Stream.
  - **Print to log4net** - It prints the error message to the log4net library.



Exception Handling: Do not Show
✓ Lazy Initialization
Output Path:

- **Lazy Collection Initialization.**

  Check this option to avoid the associated objects from being loaded when the main object is loaded. Unchecking this option will result in loading the associated objects when the main object is loaded. If you enabled (checked) lazy collection initialization, associated objects (1 to many) will not be loaded until you access them (e.g. getFlight(0)). Enabling this option can usually reduce more then 80% of the database loading.

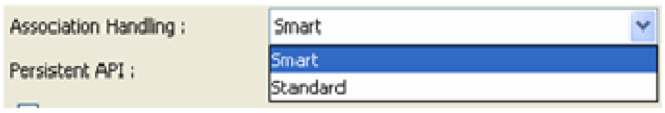
- **Output Path**

  Specify the location of C# persistent code generation.

- **Association Handling**

  Select the type of association handling to be used, either Smart or Standard.

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

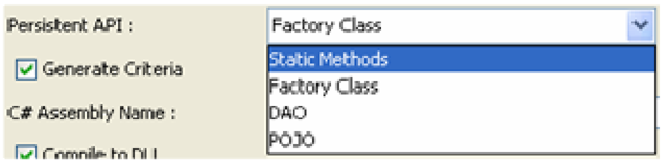

Association Handling : Smart

Persistent API :

- **Persistent API**

  Select the type of persistent code to be generated, either Static Methods, Factory Class, DAO or POJO.

  - **Static Method** - Client can create, retrieve, and persist the PersistentObject directly.
  - **Factory Class** - Create FactoryObject class for client to create and retrieve PersistentObject. Client still can persist the PersistentObject directly.
  - **DAO** - Client uses the PersistentObjectDAO class to create, retrieve and persist the PersistentObject.
  - **POJO** - Client uses the PersistentManager to retrieve and persist the PersistentObject.


Persistent API : Factory Class

Generate Criteria

C# Assembly Name :

Compile to DLL

- **Create Criteria**

  You can check the Generate Criteria option to generate the criteria class for each ORM Persistable class. The Criteria is used for querying the database in object-oriented way (please refer to chapter 9 for more details about the criteria)

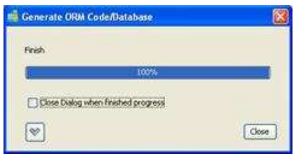
- **Sample**

  Sample files, including C# application sample and C# project file for Visual Studio .NET 2003 are available for generation. The generated sample files guide you through the usage of the C# persistence class. You can check the options to generate the sample files for reference.

  You need to generate the sample and check the create C# project option so that you can modify the sample to execute the generated C# persistence class in Visual Studio .NET 2003.

You have to select All Properties of Override toString Method when you execute the list data sample so that you can read the persistent object information.

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


Generate ORM Code/Database

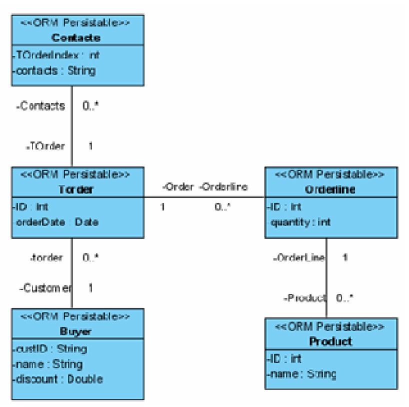
Finish

100%

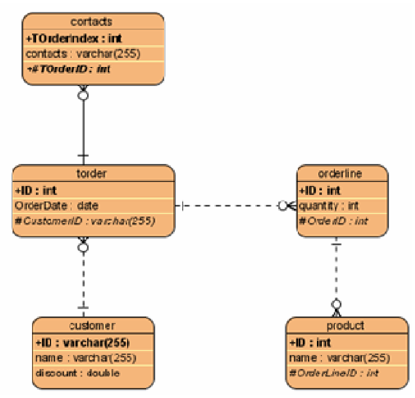
Close Dialog when finished progress

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

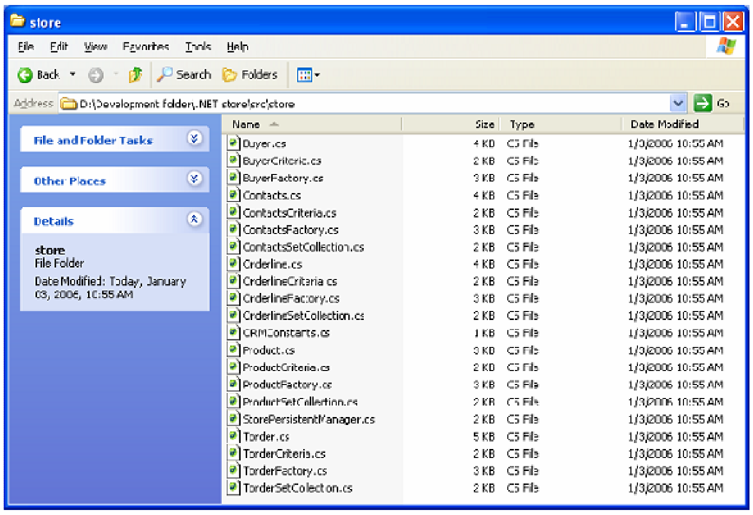
The Class Diagram:



The ER Diagram:



Generated C# code:



store
File Edit View Favorites Tools Help
Back Search Folders
Address D:\Development folder\.NET store\src\store
Go
File and Folder Tasks
Other Places
Details
store
File Folder
Date Modified: Today, January 03, 2006, 10:55 AM

Name	Size	Type	Date Modified
Buyer.cs	4 KB	CS File	1/3/2006 10:55 AM
BuyerCriteria.cs	2 KB	CS File	1/3/2006 10:55 AM
BuyerFactory.cs	3 KB	CS File	1/3/2006 10:55 AM
Contacts.cs	4 KB	CS File	1/3/2006 10:55 AM
ContactsCriteria.cs	2 KB	CS File	1/3/2006 10:55 AM
ContactsFactory.cs	3 KB	CS File	1/3/2006 10:55 AM
ContactsSetCollection.cs	2 KB	CS File	1/3/2006 10:55 AM
Orderline.cs	4 KB	CS File	1/3/2006 10:55 AM
OrderlineCriteria.cs	2 KB	CS File	1/3/2006 10:55 AM
OrderlineFactory.cs	3 KB	CS File	1/3/2006 10:55 AM
OrderlineSetCollection.cs	2 KB	CS File	1/3/2006 10:55 AM
ORMConstants.cs	1 KB	CS File	1/3/2006 10:55 AM
Product.cs	3 KB	CS File	1/3/2006 10:55 AM
ProductCriteria.cs	2 KB	CS File	1/3/2006 10:55 AM
ProductFactory.cs	3 KB	CS File	1/3/2006 10:55 AM
ProductSetCollection.cs	2 KB	CS File	1/3/2006 10:55 AM
StorePersistentManager.cs	2 KB	CS File	1/3/2006 10:55 AM
Torder.cs	5 KB	CS File	1/3/2006 10:55 AM
TorderCriteria.cs	2 KB	CS File	1/3/2006 10:55 AM
TorderFactory.cs	3 KB	CS File	1/3/2006 10:55 AM
TorderSetCollection.cs	2 KB	CS File	1/3/2006 10:55 AM

Using the Generated Sample

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

Class File	**Function**
CreateStoreData.cs	Create persistent objects and save objects to database.
CreateStoreDatabaseSchema.cs	Export the schema to database.
DeleteStoreData.cs	Delete persistent objects from the database.
DropStoreDatabaseSchema.cs	Remove the schema in the database.
ListStoreData.cs	List all the persistent objects in database.
RetrieveAndUpdateStoreData.cs	Get the persistent object from the database and modify the object attributes.

We will demonstrate how to modify the CreateStoreData.cs to create the persistent objects and relationships from the generated C# code and save the persistent object to database.

The Original CreateStoreData.cs file, CreateData() method:

```
private void CreateData() {
    PersistentTransaction t = store.StorePersistentManager.Instance().GetSession().BeginTransaction();
    try {
        store.Contacts lstoreContacts = store.ContactsFactory.CreateContacts();
        // Initialize the properties of the persistent object
        lstoreContacts.Save();
        store.Buyer lstoreBuyer = store.BuyerFactory.CreateBuyer();
        // Initialize the properties of the persistent object
        lstoreBuyer.Save();
        store.Orderline lstoreOrderline = store.OrderlineFactory.CreateOrderline();
        // Initialize the properties of the persistent object
        lstoreOrderline.Save();
        store.Product lstoreProduct = store.ProductFactory.CreateProduct();
        // Initialize the properties of the persistent object
        lstoreProduct.Save();
        store.Torder lstoreTorder = store.TorderFactory.CreateTorder();
        // Initialize the properties of the persistent object
        lstoreTorder.Save();
        t.Commit();
    }
    catch(Exception e) {
        t.RollBack();
        Console.WriteLine(e);
    }
}
```

The modified CreateStoreData.cs file, CreateData() method:

```
public class CreateStoreData {
	private void CreateData() {
		PersistentTransaction t = store.StorePersistentManager.Instance().GetSession().BeginTransaction();
		try {
			//create persistent object instance
			//create Contacts
			Console.WriteLine("Create persistent objects.");
			store.Contacts lstoreContacts = store.ContactsFactory.CreateContacts();
				lstoreContacts.Contact = "contact : 12345678";
				lstoreContacts.TOrderIndex = 1;

			//create Buyer
			store.Buyer lstoreBuyer = store.BuyerFactory.CreateBuyer();
			lstoreBuyer.Discount = 0.9;
			lstoreBuyer.Name = "Judy";
			lstoreBuyer.CustID = "judy";

			//create Torder
			store.Torder lstoreTorder = store.TorderFactory.CreateTorder();
			lstoreTorder.OrderDate = DateTime.Now;
			//create Orderline
			store.Orderline lstoreOrderline = store.OrderlineFactory.CreateOrderline();
			lstoreOrderline.Quantity = 10000;

			//create Product
			store.Product lstoreProduct = store.ProductFactory.CreateProduct();
			lstoreProduct.Name = "Chocolate";

			//create relationship
			Console.WriteLine("Create the relationships between persistent objects.");
			lstoreTorder.Customer = lstoreBuyer;
			lstoreContacts.TOrder = lstoreTorder;
			lstoreOrderline.Order = lstoreTorder;
			lstoreProduct.OrderLine = lstoreOrderline;
			//save the persistent objects
			Console.WriteLine("Save the persistent objects.");
			lstoreBuyer.Save();
			t.Commit();
		}
		catch(Exception e) {
			t.RollBack();
			Console.WriteLine(e);
		}
	}
```

Uncomment the line **CreateStoreData.Main(args);** in SampleCenterControl.cs and modify it to execute the generated sample. The persistent objects will be created in database.ã€€ You can execute ListStoreData.java to show the information of all the created persistent objects.

The ListStoreData.cs file, listTestData() method:

```
public void ListData() {
        System.Console.WriteLine("Listing Contacts...");
        store.Contacts[] lstoreContactss = store.ContactsFactory.ListContactsByQuery(null, null);
        int length = Math.Min(lstoreContactss.Length, ROW_COUNT);
        for (int i = 0; i < length; i++) {
                System.Console.WriteLine(lstoreContactss[i]);
        }
        System.Console.WriteLine(length + " record(s) retrieved.");
        System.Console.WriteLine("Listing Buyer...");
        store.Buyer[] lstoreBuyers = store.BuyerFactory.ListBuyerByQuery(null, null);
        length = Math.Min(lstoreBuyers.Length, ROW_COUNT);
        for (int i = 0; i < length; i++) {
                System.Console.WriteLine(lstoreBuyers[i]);
        }
        System.Console.WriteLine(length + " record(s) retrieved.");
        System.Console.WriteLine("Listing Orderline...");
        store.Orderline[] lstoreOrderlines = store.OrderlineFactory.ListOrderlineByQuery(null, null);
        length = Math.Min(lstoreOrderlines.Length, ROW_COUNT);
        for (int i = 0; i < length; i++) {
                System.Console.WriteLine(lstoreOrderlines[i]);
        }
        System.Console.WriteLine(length + " record(s) retrieved.");
        System.Console.WriteLine("Listing Product...");
        store.Product[] lstoreProducts = store.ProductFactory.ListProductByQuery(null, null);
        length = Math.Min(lstoreProducts.Length, ROW_COUNT);
        for (int i = 0; i < length; i++) {
                System.Console.WriteLine(lstoreProducts[i]);
        }
        System.Console.WriteLine(length + " record(s) retrieved.");
        System.Console.WriteLine("Listing Torder...");
        store.Torder[] lstoreTorders = store.TorderFactory.ListTorderByQuery(null, null);
        length = Math.Min(lstoreTorders.Length, ROW_COUNT);
        for (int i = 0; i < length; i++) {
                System.Console.WriteLine(lstoreTorders[i]);
        }

        System.Console.WriteLine(length + " record(s) retrieved.");

}
```

Uncomment the **ListStoreData.Main(args);** in SampleCenterControl.cs and execute the sample.

The result of execute ListStoreData.cs:

```
Listing Contacts...

Contacts[ TOrderIndex=1 Contacts=contact : 12345678 TOrder.Persist_ID=1 ]

1 record(s) retrieved.

Listing Buyer...

Buyer[ CustID=judy Name=Judy Discount=0.9 torder.size=1 ]

1 record(s) retrieved.

Listing Orderline...

Orderline[ ID=1 Quantity=10000 Order.Persist_ID=1 Product.size=1 ]

1 record(s) retrieved.

Listing Product...

Product[ ID=1 Name=Chocolate OrderLine.Persist_ID=1 ]

1 record(s) retrieved.

Listing Torder...

Torder[ ID=1 OrderDate=2006/1/3 ä¸Šå ˆ 12:00:00 Customer.Persist_ID=judy Contacts.

size=1 Orderline.size=1 ]

1 record(s) retrieved.
```

7 Programming in VB.NET

Chapter 7 - Programming in VB.NET

DB Visual ARCHITECT (DB-VA) can generate C#.NET source code so you can implement your application by C# programming language directly but you also can choose another language (VB.NET or C++) based on your taste in the .NET Framework. DB-VA generates DLL files and persistent libraries that can be referenced by .NET projects of language other than C#.

In this chapter:

- Introduction
- Generating DLL file
- Creating VB.NET Project
- Adding Referenced Project
- Working with the Generated Code and Persistent Library
- Running the Application

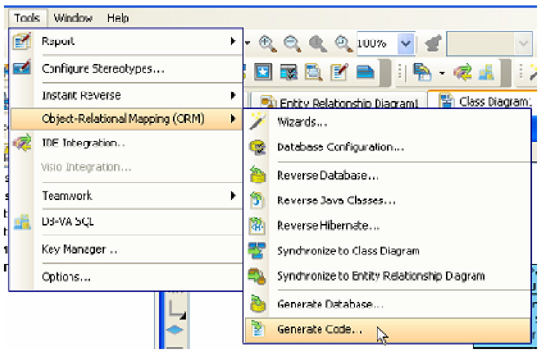
Introduction

Visual Basic .NET (VB.NET) is an Object-Oriented computer language that can be viewed as an evolution of Microsoft’s Visual Basic (VB) implemented on the Microsoft .NET framework. The .NET Framework contains a virtual machine called Common Intermediate Language (CIL). Simply put, programs are compiled to produce CIL and the CIL is distributed to user to run on a virtual machine. VB.NET, C++, C# compilers are available from Microsoft for creating CIL. In DB-VA you can generate C# persistent source code and DLL files, so you can reference the DLL file and Persistent Library in Visual Studio .NET 2003 and develop the VB.NET application.

In the following section, you will develop a VB.NET application. The application is exactly the same as the one in Chapter 4 – Developing Standalone .NET Application sample but this time you use VB.NET instead of C# for development. You need to download Chapter 4 sample application because it contains the DLL file and persistent libraries for your VB.NET project.

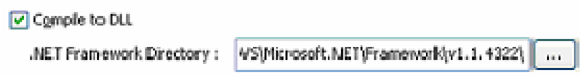
Generating DLL File

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



2. Check the **Compile to DLL** option to generate the DLL file.
   - **Compile to DLL**

     By checking this option, DB-VA will generate DLL files which can be referenced by .NET projects of language other than C#. DB-VA generates a batch file for this DLL file at the same time. You can modify the configure file (hibernate.cfg.xml) manually and use the batch file to recompile and build an up-to-date DLL file for referenced project.

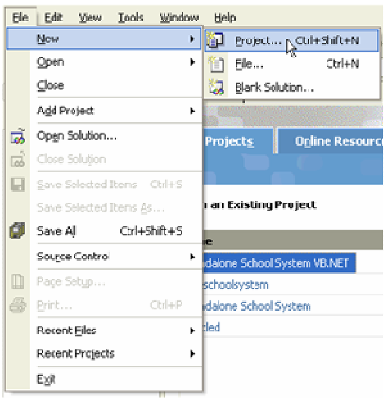


[✓] Compile to DLL

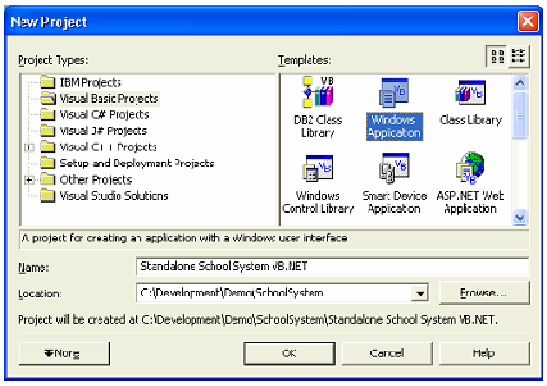
.NET Framework Directory : WS\Microsoft.NET\Framework\v1.1.4322\ ...

Creating VB.NET Project

1. Open Microsoft Visual Studio .NET 2003.
2. Select **File** > **New** > **Project …** from the menu.



3. Select the Project Types as Visual Basic Projects and Templates as Windows Application and enter the Name and Location for the new application.



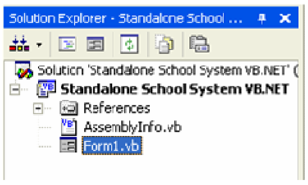
A project for creating an application with a Windows user interface

Name: Standalone School System VB.NET

Location: C:\Development\Demo\SchoolSystem

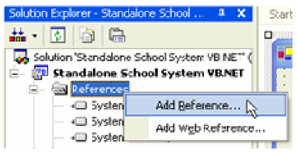
Project will be created at C:\Development\Demo\SchoolSystem\Standalone School System VB.NET.

4. The School System Project is created.

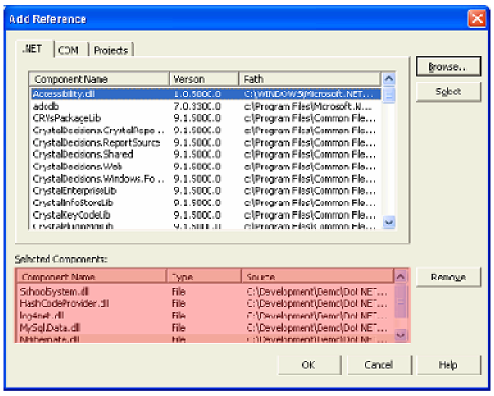

Solution Explorer - Standalone School ...
Solution 'Standalone School System VB.NET' (
Standalone School System VB.NET
References
AssemblyInfo.vb
Form1.vb

Adding Referenced Project

1. Right click References under the Standalone School System VB.NET project and select **Add Reference…** to add the C# example DDL file and persistent libraries.

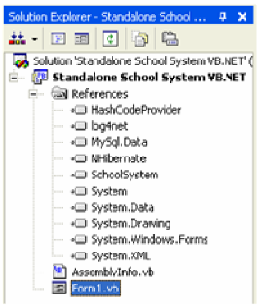

Solution Explorer - Standalone School ...
Start
Solution 'Standalone School System VB.NET' (
Standalone School System VB.NET
References
System
System
System
Add Reference...
Add Web Reference...

2. Click **Browse…** on the Add Reference dialog box to select the folder of the downloaded C# standalone application sample. Select **C# sample folder/bin/SchoolSystem.dll** and **all libraries in C# sample folder/lib**.



Component Name	Version	Path
Accessibility.dll	1.0.5000.0	C:\WINDOWS\Microsoft.NET…
adodb	7.0.3300.0	c:\Program Files\Microsoft.N…
CRVsPackageLib	9.1.5000.0	c:\Program Files\Common File…
CrystalDecisions.CrystalRepo…	9.1.5000.0	c:\Program Files\Common File…
CrystalDecisions.ReportSource	9.1.5000.0	c:\Program Files\Common File…
CrystalDecisions.Shared	9.1.5000.0	c:\Program Files\Common File…
CrystalDecisions.Web	9.1.5000.0	c:\Program Files\Common File…
CrystalDecisions.Windows.Fo…	9.1.5000.0	c:\Program Files\Common File…
CrystalEnterpriseLib	9.1.5000.0	c:\Program Files\Common File…
CrystalInfoStoreLib	9.1.5000.0	c:\Program Files\Common File…
CrystalKeyCodeLib	9.1.5000.0	c:\Program Files\Common File…

Component Name	Type	Source
SchoolSystem.dll	File	C:\Development\Demo\Dot NET…
HashCodeProvider.dll	File	C:\Development\Demo\Dot NET…
log4net.dll	File	C:\Development\Demo\Dot NET…
MySql.Data.dll	File	C:\Development\Demo\Dot NET…
NHibernate.dll	File	C:\Development\Demo\Dot NET…



```
Solution 'Standalone School System VB.NET' (
  Standalone School System VB.NET
    References
      HashCodeProvider
      log4net
      MySql.Data
      NHibernate
      SchoolSystem
      System
      System.Data
      System.Drawing
      System.Windows.Forms
      System.XML
    AssemblyInfo.vb
    Form1.vb
```

Working with the Generated Code and Persistent Library

Both the C# and VB.NET languages are built on the .NET framework. Both languages accomplish the same thing using different language syntax. In this section, we will point out some examples of using the generated code and persistent library.

- Importing DLL and Persistent Libraries namespace for the Class.

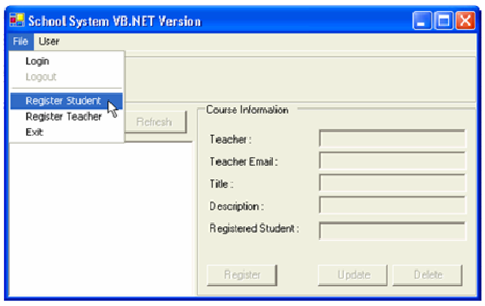
VB.NET	**C#**

`Imports school`
`Imports Orm`
	`using school;`
	`using Orm;`

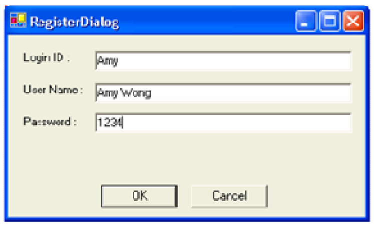
- Creating PersistentTransaction when manipulating the database.

VB.NET	**C#**
`Dim t As PersistentTransaction = SchoolSystemPersistentManager.Instance.GetSession.BeginTransaction`	`PersistentTransaction t = SchoolSystemPersistentManager.Instance().GetSession().BeginTransaction();`

- Creating Object and Save to Database
  1. From menu bar, select **File** > **Register Student** to open the Add Student dialog box.



  2. Fill in the student information. Click OK to create the new Student record in School System.


RegisterDialog

Login ID : Amy
User Name : Amy Wong
Password : 1234

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

```
Private Sub okButton_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles okButton.Click
    If (loginIDTextBox.Text.Length = 0 Or
passwordTextBox.Text.Length = 0) Then
        MessageBox.Show("Login id or password missing")
        Return
    End If
    Dim t As PersistentTransaction =
SchoolSystemPersistentManager.Instance.GetSession.BeginTransaction
    Try
        If (_userType = CREATE_TEACHER) Then
            User = TeacherFactory.CreateTeacher()
        ElseIf (_userType = CREATE_STUDENT) Then
            User = StudentFactory.CreateStudent()
        End If
```

Source File : Standalone School System VB.NET\RegisterDialog.vb

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

```
User.Password = passwordTextBox.Text
User.LoginID = loginIDTextBox.Text
If (TypeOf User Is Teacher) Then
    CType(User, Teacher).Email = emailTextBox.Text
End If
```

Source File : Standalone School System VB.NET\RegisterDialog.vb

5. Call Save() method of Student Persistent Object and Commit() method of PersistentTransaction., then the new Student object will be saved to the database. If any error occurred during the transaction, you can call the Rollback() method to cancel the proposed changes in a pending database transaction

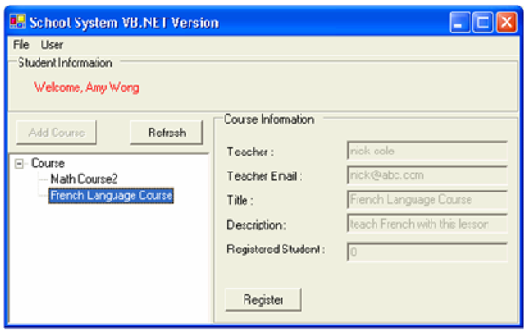
```
    User.Save()
    t.Commit()
    DialogResult = DialogResult.OK
    Close()
Catch ex As Exception
    Console.WriteLine(ex.InnerException.Message)
    DialogResult = DialogResult.Cancel
    t.RollBack()
End Try
```

Source File : Standalone School System VB.NET\RegisterDialog.vb

- Query Object from Database

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

Student Login:



Student Information

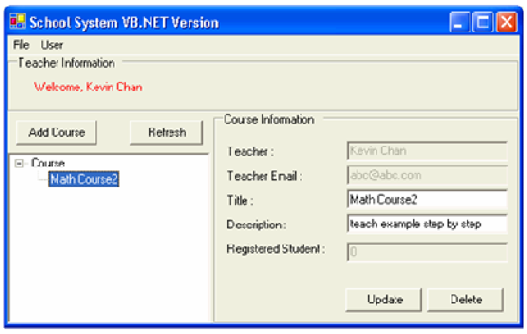
Welcome, Amy Wong

Course
- Math Course2
- French Language Course

Course Information

Teacher : nick cole
Teacher Email : nick@abc.com
Title : French Language Course
Description : teach French with this lesson
Registered Student : 0

Teacher Login:



Teacher Information

Welcome, Kevin Chan

Course
- Math Course2

Course Information

Teacher : Kevin Chan
Teacher Email : abc@abc.com
Title : Math Course2
Description : teach example step by step
Registered Student : 0

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

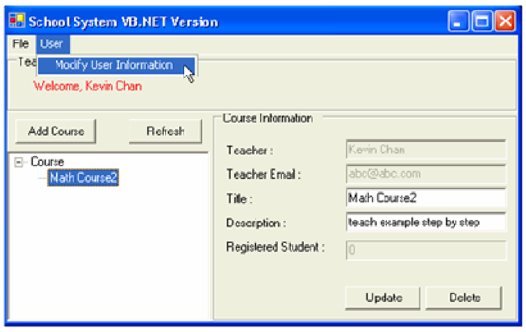
```
Private Sub UpdateTreeView()
    Dim lCourses() As Course
    If (TypeOf CurrentUser Is Student) Then
        lCourses = CourseFactory.ListCourseByQuery(Nothing, Nothing)
    ElseIf (TypeOf CurrentUser Is Teacher) Then
        lCourses = CType(CurrentUser, Teacher).courses.ToArray()
    End If
    ...
End Sub
```

Source File : Standalone School System VB.NET \SchoolSystemForm.vb

- Update Object and Save to Database

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

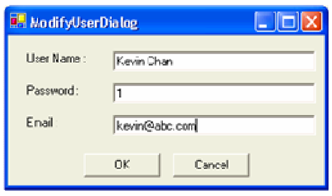
1. From the menu bar, select **User** > **Modify User Information** to open the Modify User Information dialog box.



Welcome, Kevin Chan

Teacher : Kevin Chan
Teacher Email : abc@abc.com
Title : Math Course2
Description : teach example step by step
Registered Student : 0

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



User Name : Kevin Chan
Password : 1
Email kevin@abc.com

3. Update the information for the User object by setting password, name and email address.

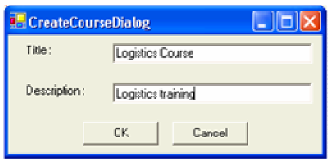
```
Private Sub okButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles okButton.Click
    If (nameTextBox.Text.Length = 0 Or passwordTextBox.Text.Length =
0) Then
        MessageBox.Show("Missing login id and password")
    Else
        Dim t As PersistentTransaction =
SchoolSystemPersistentManager.Instance.GetSession.BeginTransaction
        Try
            _user.Name = nameTextBox.Text
            _user.Password = passwordTextBox.Text
            If (TypeOf _user Is Teacher) Then
                CType(_user, Teacher).Email = emailTextBox.Text
            End If
            _user.Save()
            DialogResult = DialogResult.OK
            t.Commit()
        Catch ex As Exception
            DialogResult = DialogResult.Cancel
            t.RollBack()
        End Try
    End If
    Close()
End Sub
```

Source File : Standalone School System VB.NET\ModifyUserDialog.cs

- Delete Object in Database

Teacher can create course for students to register, and they can cancel courses in the system. They only need to click delete button of the course then the course information will be deleted in the database and all its relationships with registered students will be removed.

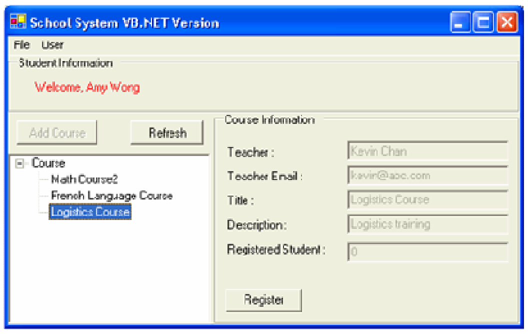
1. Teacher can create the course by clicking the **Add Course** button, fill in Course name and Description.


CreateCourseDialog

Title: Logistics Course

Description: Logistics training

2. Student can register the course by clicking the **Register** button.



Student Information

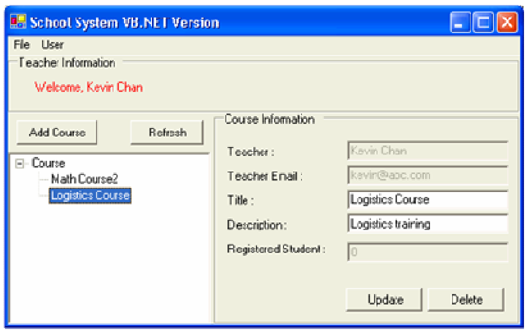
Welcome, Amy Wong

Course
- Math Course2
- French Language Course
- Logistics Course

Course Information

Teacher : Kevin Chan
Teacher Email : kevin@abc.com
Title : Logistics Course
Description: Logistics training
Registered Student : 0

3. The teacher can view how many students registered his course, and he can delete courses in the system.



Teacher Information

Welcome, Kevin Chan

Course
- Math Course2
- Logistics Course

Course Information

Teacher : Kevin Chan
Teacher Email : kevin@abc.com
Title : Logistics Course
Description: Logistics training
Registered Student : 0

4. Click Delete of a course then it will trigger the **delButton_Click()** method .

```
Private Sub deleteButton_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles deleteButton.Click
    If (MessageBox.Show("Delete", "Delete",
MessageBoxButtons.OKCancel).Equals(DialogResult.OK)) Then
```

Source File : Standalone School System VB.NET\CoursePanel.vb

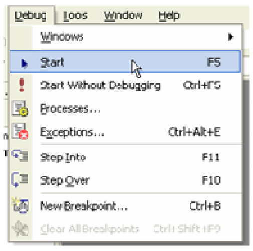
5. Call **deleteAndDissociate()** method to delete the course object and remove the relationships of students and teachers with it.

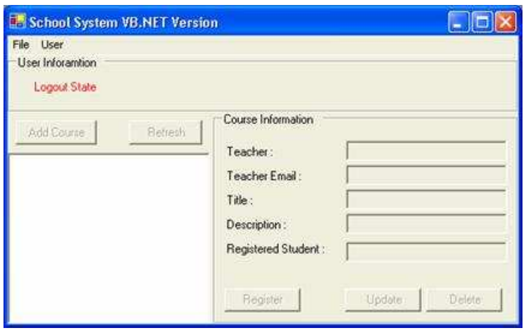
```
        Dim t As PersistentTransaction =
SchoolSystemPersistentManager.Instance().GetSession().BeginTransactio
n()
        Try
            _CourseNode.Course().DeleteAndDissociate()
            _CourseNode.Remove()
            t.Commit()
        Catch ex As Exception
            Console.WriteLine(ex.InnerException.Message)
            t.RollBack()
        End Try
    End If
End Sub
```

Source File : Standalone School System VB.NET\CoursePanel.vb

Running Application

The VB.NET project is implemented in Visual Studio .NET 2003. To execute the VB.NET Application, select **Debug** > **Start (F5)** from the menu bar.





8 Programming in C++.NET

Chapter 8 - Programming in C++ .NET

DB Visual ARCHITECT (DB-VA) can generate C#.NET source code so you can implement your application by C# programming language directly but you can also choose another language (VB.NET or C++) based on your taste in the .NET Framework. DB-VA generates DLL file and persistent libraries that can be referenced by .NET projects of language other than C#.

In this chapter:

- Introduction
- Generating DLL file
- Creating C++ Project
- Adding Referenced Project
- Working with the Generating Code and Persistent Library
- Running the Application

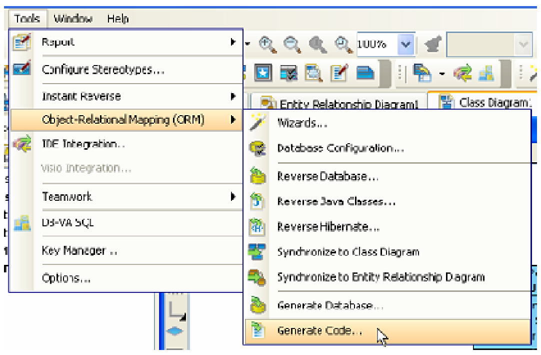
Introduction

C++ is an Object-Oriented Programming (OOP) language that is viewed by many as the best language for creating large-scale applications. The .NET Framework contains a virtual machine called Common Intermediate Language (CIL). Simply put, programs are compiled to produce CIL and the CIL is distributed to user to run on a virtual machine. C++, VB.NET, C# compilers are available from Microsoft for creating CIL. In DB-VA you can generate C# persistent source code and DLL file, so you can reference the DLL file and persistent library in Visual Studio .NET 2003 and develop the C++ application.

In the following section, you will develop a C++ application. The application is exactly same as the one in Chapter 4 – Developing Standalone .NET Application sample, but this time you use C++ instead of C# for development. You need to download the Chapter 4 sample application because it contains DLL file and persistent libraries for your C++ project.

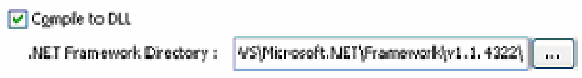
Generating DLL File

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



2. Check the **Compile to DLL** option to generate the DLL file.
   - **Compile to DLL**

     By checking this option, DB-VA will generate DLL files which can be referenced by .NET projects of language other than C#. DB-VA generates batch file for the DLL file at the same time. You can modify the configuration file (hibernate.cfg.xml) manually and use the batch file to recompile and build an up-to-date DLL file for referenced project.

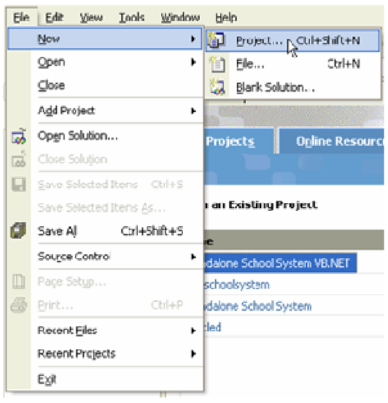


Compile to DLL ✓

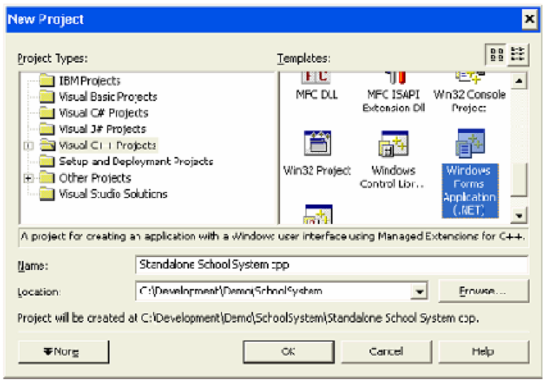
.NET Framework Directory : WS\Microsoft.NET\Framework\v1.1.4322\

Creating C++ Project

1. Open Microsoft Visual Studio .NET 2003.
2. Select **File** > **New** > **Project …** from the menu.



3. Select Project Types as Visual C++ Projects and Templates as Windows Form Application (.NET) and Location for the new application.



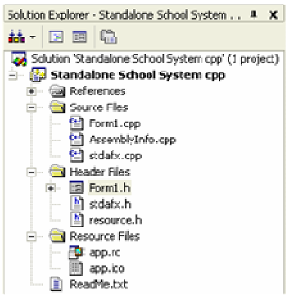
A project for creating an application with a Windows user interface using Managed Extensions for C++.

Name: Standalone School System cpp

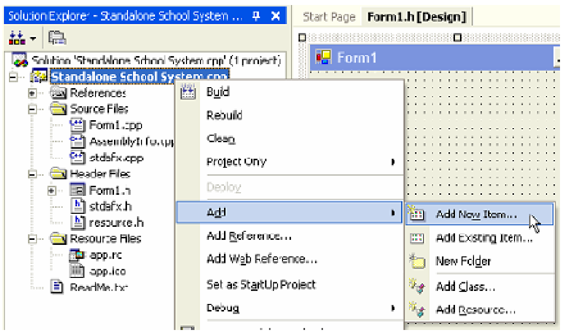
Location: C:\Development\Demo\SchoolSystem

Project will be created at C:\Development\Demo\SchoolSystem\Standalone School System cpp.

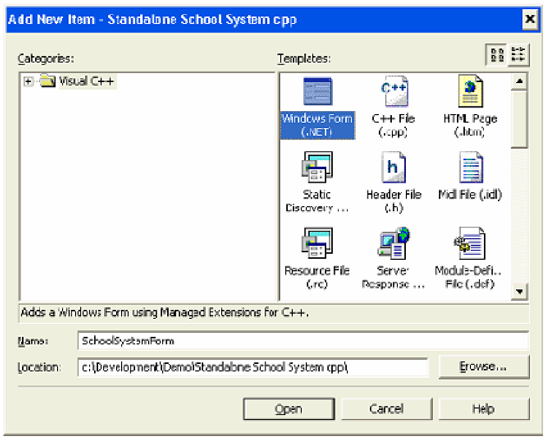
4. The School System Project is created.



5. Right click Standalone School System cpp Project, select **Add** > **Add New Item…** from the popup menu.



6. Select Category as Visual C++, Template Windows Form (.NET) and enter the name for the form called “SchoolSystemForm”. This is the start point for the application.



Adds a Windows Form using Managed Extensions for C++.

Name: SchoolSystemForm

Location: c:\Development\Demo\Standalone School System cpp\

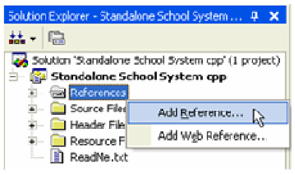
7. Append the following content to the SchoolSystemForm.cpp file (Source files/SchoolSystemForm.cpp). This is the main method for C++ Application to execute.

```
#include <windows.h>

using namespace StandaloneSchoolSystemcpp;

int APIENTRY _tWinMain(HINSTANCE hInstance,
                                    HINSTANCE hPrevInstance,
                                    LPTSTR lpCmdLine,
                                    int nCmdShow)
{
    System::Threading::Thread::CurrentThread->ApartmentState =
System::Threading::ApartmentState::STA;
    Application::Run(new SchoolSystemForm());
    return 0;
}
```

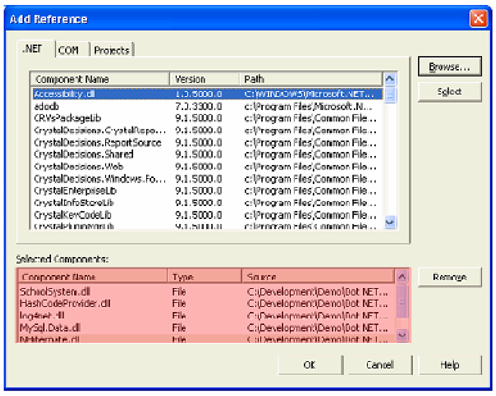
8. Remove the Form1.cpp and Form1.h files.

Adding Referenced Project

1. Right click References under the Standalone School System C++ project and select **Add Reference…**. Reference the C# example DDL file and persistent libraries for developing the C++ application.

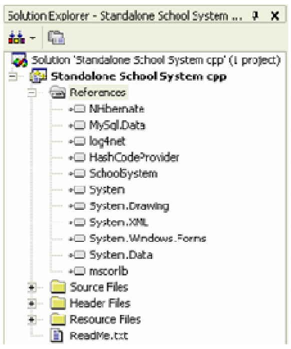

Solution Explorer - Standalone School System ...
Solution 'Standalone School System cpp' (1 project)
Standalone School System cpp
References
Source Files
Header Files
Resource F
ReadMe.txt
Add Reference...
Add Web Reference...

2. Click **Browse…** on the Add Reference dialog box to select the folder of the downloaded C# standalone application sample. Select **C# sample folder/bin/SchoolSystem.dll** and **all libraries in C# sample folder/lib**.



Component Name	Version	Path
Accessibility.dll	1.0.5000.0	C:\WINDOWS\Microsoft.NET...
adodb	7.0.3300.0	c:\Program Files\Microsoft.N...
CRVsPackageLib	9.1.5000.0	c:\Program Files\Common File...
CrystalDecisions.CrystalRepo...	9.1.5000.0	c:\Program Files\Common File...
CrystalDecisions.ReportSource	9.1.5000.0	c:\Program Files\Common File...
CrystalDecisions.Shared	9.1.5000.0	c:\Program Files\Common File...
CrystalDecisions.Web	9.1.5000.0	c:\Program Files\Common File...
CrystalDecisions.Windows.Fo...	9.1.5000.0	c:\Program Files\Common File...
CrystalEnterpriseLib	9.1.5000.0	c:\Program Files\Common File...
CrystalInfoStoreLib	9.1.5000.0	c:\Program Files\Common File...
CrystalKeyCodeLib	9.1.5000.0	c:\Program Files\Common File...
CrystalPluginMgrLib	9.1.5000.0	c:\Program Files\Common File...

Component Name	Type	Source
SchoolSystem.dll	File	C:\Development\Demo\Dot NET...
HashCodeProvider.dll	File	C:\Development\Demo\Dot NET...
log4net.dll	File	C:\Development\Demo\Dot NET...
MySql.Data.dll	File	C:\Development\Demo\Dot NET...
NHibernate.dll	File	C:\Development\Demo\Dot NET...

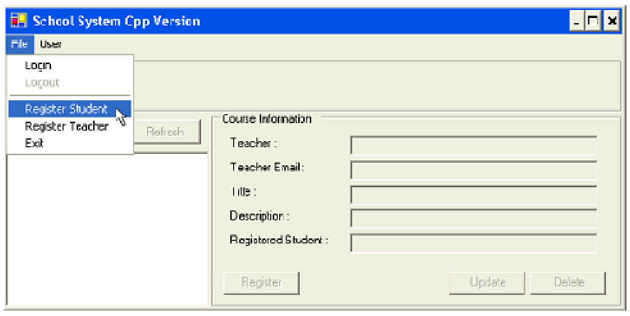


```
Solution 'Standalone School System cpp' (1 project)
  Standalone School System cpp
    References
      NHibernate
      MySql.Data
      log4net
      HashCodeProvider
      SchoolSystem
      System
      System.Drawing
      System.XML
      System.Windows.Forms
      System.Data
      mscorlib
    Source Files
    Header Files
    Resource Files
    ReadMe.txt
```

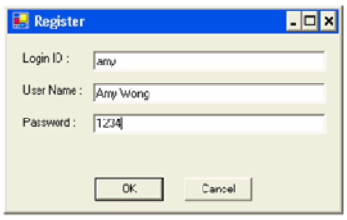
Working with the Generating Code and Persistent Library

C# and C++ are both languages that built on the .NET framework and they both accomplish the same thing, just using different language syntax. In this section, you will learn how to work with Generate Code and Persistent Library with C++ language.

- Creating Object and Save to Database
  1. From the menu bar, select **File** > **Register Student** to open the Add Student dialog box.



  2. Fill in the student information. Click OK to create the new Student record in School System.



Login ID : amy
User Name : Amy Wong
Password : 1234

3. After that, the system creates the new Student Persistent object.

```
private: System::Void okButton_Click(System::Object *sender,
System::EventArgs *e)
{
    if(loginIDTextBox->Text->Length == 0 || passwordTextBox->Text-
>Length == 0){
        MessageBox::Show("Login id or password missing");
        return;
    }
    PersistentTransaction *t =
SchoolSystemPersistentManager::Instance()->GetSession()-
>BeginTransaction();
    try{ã€€ã€€ã€€ ã€€ã€€ã€€
        if(userType == CREATE_TEACHER){
            createdUser = TeacherFactory::CreateTeacher();
    }else{
        createdUser = StudentFactory::CreateStudent();

    }
```

Source File : Standalone School System cpp \RegisterDialog.h

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

```
createdUser->set_Name(userNameTextBox->Text);
createdUser->set_Password(passwordTextBox->Text);
createdUser->set_LoginID(loginIDTextBox->Text);
if(dynamic_cast<Teacher*>(createdUser)){
    dynamic_cast<Teacher*>(createdUser)->set_Email(emailTextBox->Text);

}
```

Source File : Standalone School System cpp \RegisterDialog.h

5. Call Save() method of Student Persistent Object and Commit() method of PersistentTransaction, then the new Student object will be saved in database. If there are any errors occurred during the transaction, you can call the Rollback() method to cancels the proposed changes in a pending database transaction

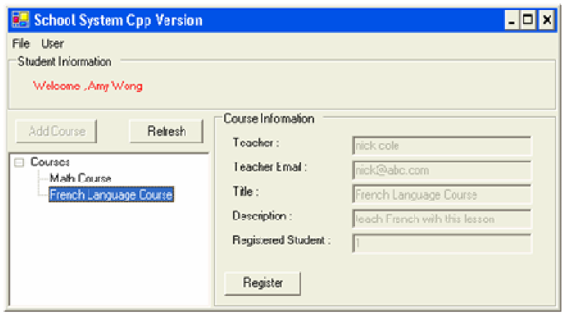
```
    createdUser->Save();
    this->set_CreatedUser(createdUser);
    DialogResult = DialogResult::OK;
    t->Commit();
    Close();
}catch(Exception *ex){
    Console::WriteLine(ex->InnerException->Message);
    DialogResult = DialogResult::Cancel;
    t->RollBack();
}
```

Source File : Standalone School System cpp \RegisterDialog.h

- Querying Object from Database

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

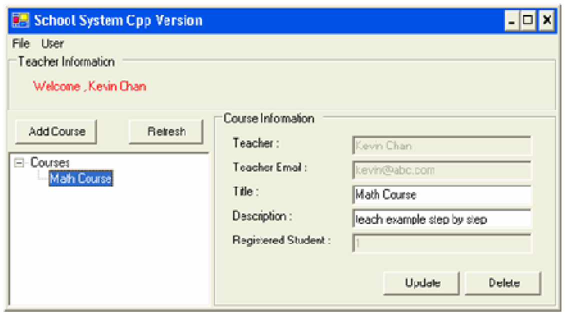
Student Login:



Welcome , Amy Wong

Teacher : nick cole
Teacher Email : nick@abc.com
Title : French Language Course
Description : teach French with this lesson
Registered Student : 1

Teacher Login:



Welcome , Kevin Chan

Teacher : Kevin Chan
Teacher Email : kevin@abc.com
Title : Math Course
Description : teach example step by step
Registered Student : 1

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

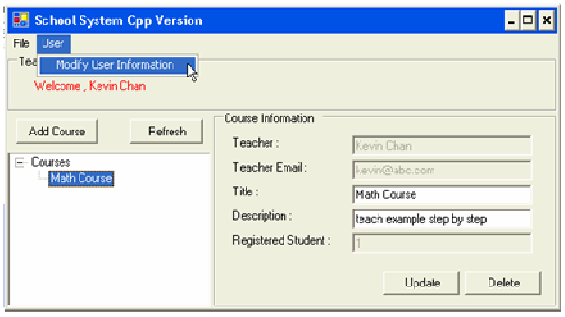
```
void updateTreeView(void){
    Course *courses[];
    if(dynamic_cast<Student*>(currentUser)){
        courses = CourseFactory::ListCourseByQuery(NULL, NULL);
    }else{
        courses = dynamic_cast<Teacher*>(currentUser)->courses-
>ToArray();
    }
    ...
}
```

Source File : Standalone School System cpp \SchoolSystemForm.h

- Updating Object and Saving to Database

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

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

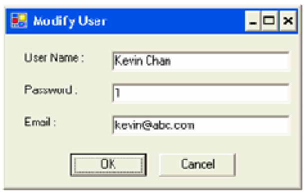


Welcome , Kevin Chan

Courses
Math Course

Course Information
Teacher : Kevin Chan
Teacher Email : kevin@abc.com
Title : Math Course
Description : teach example step by step
Registered Student : 1

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



User Name : Kevin Chan
Password : 1
Email : kevin@abc.com

3. Update the information for the User object includes password, name and email address.

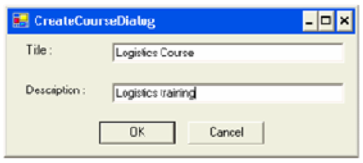
```
private: System::Void okButton_Click(System::Object *sender,
System::EventArgs *e)
{
    if(nameTextBox->Text->Length == 0 || passwordTextBox->Text->Length ==
0){
        MessageBox::Show("Missing name or password");
        return;
    }else{
        PersistentTransaction *t =
SchoolSystemPersistentManager::Instance()->GetSession()-
>BeginTransaction();
        try{
            user->set_Name(nameTextBox->Text);
            user->set_Password(passwordTextBox->Text);
            if(dynamic_cast<Teacher*>(user)){
                (dynamic_cast<Teacher*>(user))->set_Email(emailTextBox-
>Text);
            }
            user->Save();
            DialogResult = DialogResult::OK;
            t->Commit();
        }catch(Exception *ex){
            DialogResult = DialogResult::Cancel;
            t->RollBack();
        }
    }
}
```

Source File : Standalone School System cpp\ModifyUserDialog.h

- Deleting Object in Database

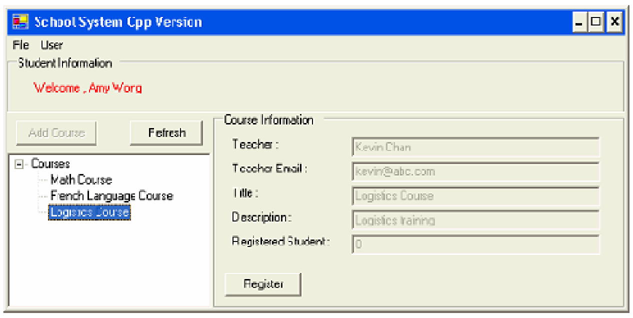
Teacher can create courses for students to register and they can cancel the course in the system. They only need to click delete button of the course then the course information will be deleted in the database and all its relationships with register students will be removed.

1. Teacher can create the course by clicking the **Add Course** button, fill in Course name and Description.



Title : Logistics Course
Description : Logistics training

2. Student can register the course by clicking the **Register** button.

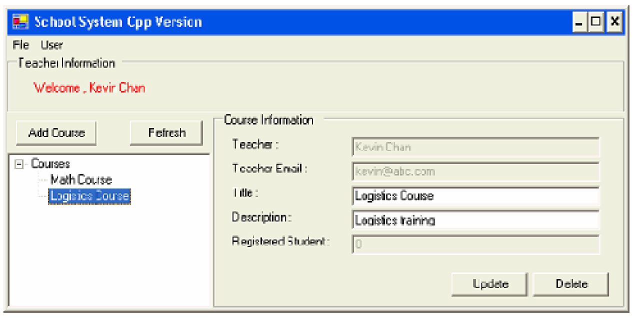


Welcome , Amy Wong

Courses
Math Course
French Language Course
Logistics Course

Teacher : Kevin Chan
Teacher Email : kevin@abc.com
Title : Logistics Course
Description : Logistics training
Registered Student : 0

3. The teacher can view how many students registered his course, and he can delete the course in the system.



Welcome , Kevir Chan

Courses
Math Course
Logistics Course

Teacher : Kevin Chan
Teacher Email : kevin@abc.com
Title : Logistics Course
Description : Logistics training
Registered Student : 0

4. Click Delete of a Course then it will trigger the **deleteButton_Click()** method.

```
private: System::Void deleteButton_Click(System::Object *sender,
System::EventArgs *e)
{
    if(MessageBox::Show("Delete", "Delete", MessageBoxButtons::OKCancel)
== DialogResult::OK){
```

Source File : Standalone School System cpp\SchoolSystemform.h

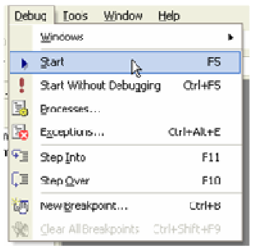
5. Call **deleteAndDissociate()** method to delete the course object and remove the relationships of student and teacher with it.

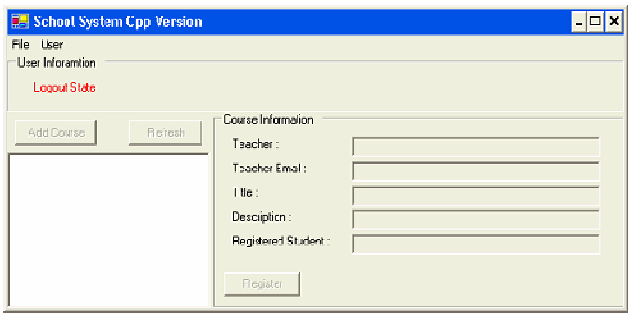
```
        PersistentTransaction *t =
SchoolSystemPersistentManager::Instance()->GetSession()-
>BeginTransaction();
        try{
           selectedNode->get_Course()->DeleteAndDissociate();
           selectedNode->Remove();
            t->Commit();
       }catch(Exception *ex){
            Console::WriteLine(ex->InnerException->Message);
           t->RollBack();
       }
    }
}
```

Source File : Standalone School System cpp\SchoolSystemForm.h

Running the Application

To execute the C++ application, select **Debug** > **Start (F5)** on the menu bar Visual Studio .NET 2003.


Debug
Tools
Window
Help
Windows
Start F5
Start Without Debugging Ctrl+F5
Processes...
Exceptions... Ctrl+Alt+E
Step Into F11
Step Over F10
New Breakpoint... Ctrl+B
Clear All Breakpoints Ctrl+Shift+F9


School System Cpp Version
File
User
User Inforamtion
Logout State
Add Course
Refresh
Course Information
Teacher :
Teacher Email :
Title :
Desciption :
Registered Student :
Register