Document Name		Confidentiality Level
Test Management System		Only for Recipients' Reference
Template Code	Version	Document Code
BP	1.0	BP

Test Management System

——MFC ADO Programming

Prepared by		Date	
Reviewed by		Date	
Approved by		Date	

Copyright Declaration

Contents

1 Teaching Tips

(1) Learn ADO programming.

(2) Master basic SQL statements.

(3) Master MFC Dialog applications.

(4) Master how to use basic controls.

(5) Master how to use the list control.

(6) Master how to load the menu and the status bar.

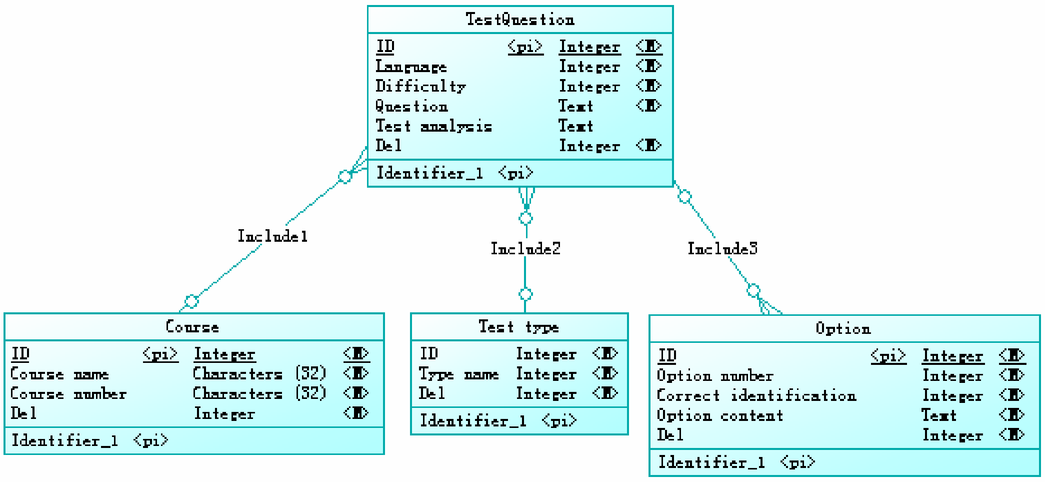
(6) Use above knowledges comprehensively to develop the test management system.

2 Functional Requirements

The test management system is a desktop application system. With MFC Dialog + ADO, it operates the test information stored in SQL Server 2008 database, mainly including the functionalities of adding test, querying test, modifying test and deleting test.

1. Data Information

The system includes four entities: test information, course information, test type information and option information. The relations between the entities are as follows:



There are four tables in the corresponding database TestLibrary:

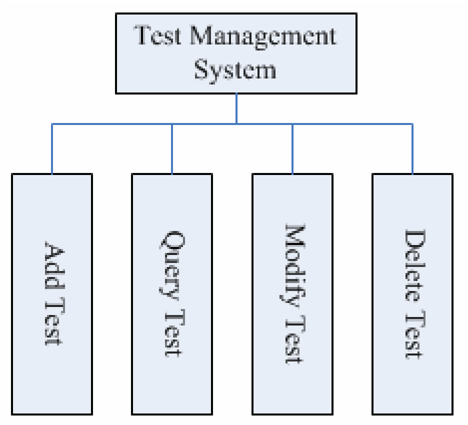
(1) Table_TestQuestion: stores the test question information.

(2) Table_Option: stores the option information.

(3) Table_Course: stores the course information.

(4) Table_TestType: stores the test type information.

2. Functionality Description



(1) Add Test

The user inputs the test information from the test editing interface. Store the test information into database TestLibrary.

(2) Query Test

The user can query the corresponding test information from the database according to the course and the test type. Display in the list.

(3) Modify Test

From the test editing interface, modify the test information in the database and update the interface display.

(4) Delete Test

Select a piece of test from the list, delete this piece of test from the database, and update the interface display.

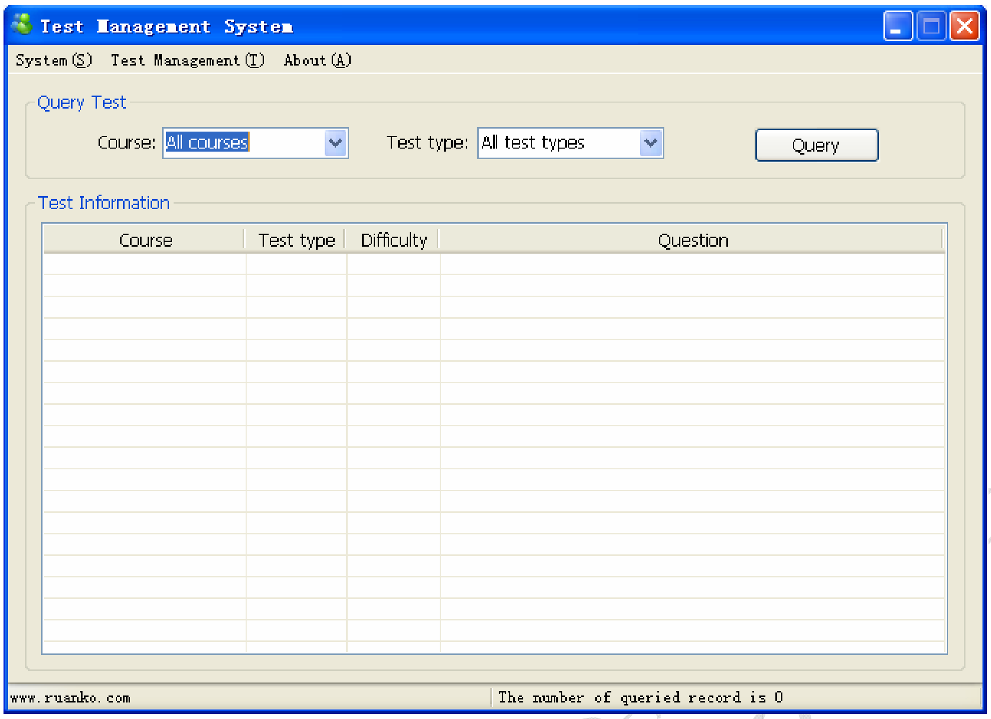
3 Design Ideas

Take SQL Server 2008 as the database management system, and Microsoft Visual Studio 2010 as the development tool. Create a MFC dialog project (TestManager) as the project type of the program. With "MFC Dialog + ADO" technology, implement the functionalities of adding, deleting, modifying and querying.

According to the requirements, the main interfaces in the system include the main interface and the test editing interface.

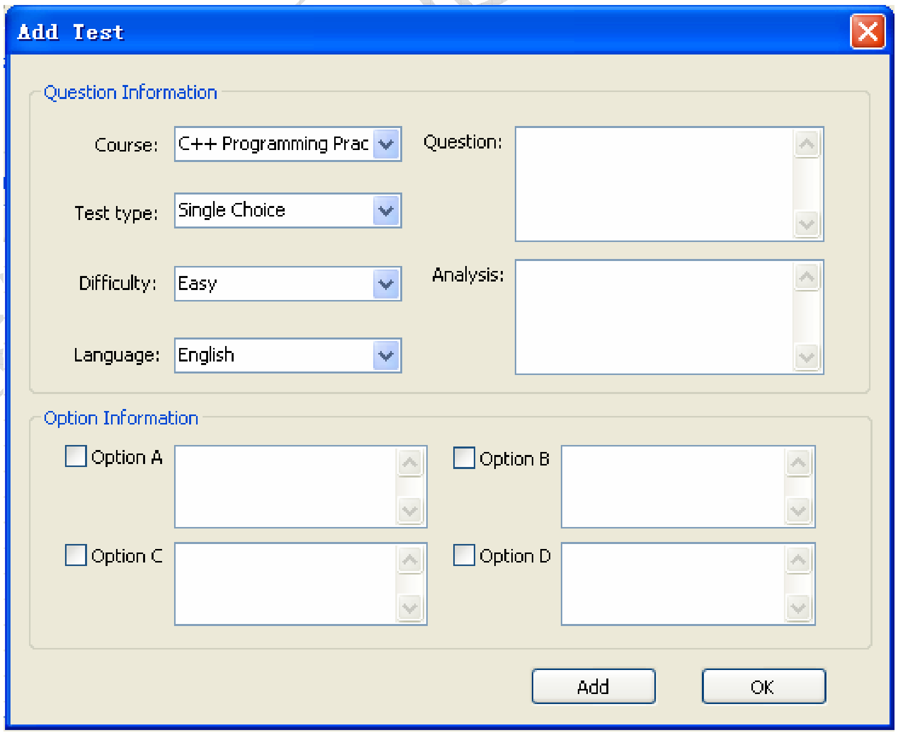
1. Main Interface

The main interface is the window that appears on startup. It is mainly composed of the title bar, the menu bar, the status bar and the client area. The sketch is as follows:



2. Test Editing Interface

When you select "Add Test" or "Modify Test" menu item, test editing interface will pop up. The interface is mainly composed of two parts: the title bar and the client area. The sketch is as follows:



4 Technical Analysis

4.1 ADO Database Programming

ADO (ActiveX Data Object), it is built on OLE DB. ADO is an OLE DB user program. All the applications that use ADO must use OLE DB indirectly. ADO simplifies OLE DB, and supports automation.

1. ADO Object Model

ADO object model mainly has three main objects: Connection object, Command object and Recordset object.

2. MFC ADO Programming

MFC provides the library support for ADO operation. Generally, there is a msado*.dll file under Program Files\Common Files\System\ado\ directory of each Windows operating system. According to different Windows versions, this file can be msado1.dll, msado15.dll and msado2.dll.

In MFC, commonly used operation objects include _ConnectionPtr, _RecordsetPtr and _CommandPtr.

4.1.1 _ConnectionPtr Interface

The smart pointer of Connection object pointing to ADO returns a record set or a null pointer. Generally, it is used to create a database connection, or execute a SQL statement that returns none result, such as a stored procedure.

1. Main Properties of _ConnectionPt Interface

ConnectionString: database connection string. Set or return the details used to establish the connection data source.

ConnectionString conction string can be formed by a series of keywords and values. Each field is separated by the semicolon. It is used to tell the data provider the specific information of the data source to connect.

Connect to SQL Server database:

Implement standard security connection string with OLE DB provider.

Provider=SQLOLEDB.1; Data Source=127.0.0.1; Database=MyDateBaseName; User ID=MyUserName; Password=MyPassword;

Implement integrated authentication connection string with OLE DB provider.

Provider=SQLOLEDB.1;Server=127.0.0.1;Database=MyDateBaseName;Integrated Security=SSPI;

Description:

Data Source: specifies the used data source name.

Provider: specifies the database provider's name.

User: The name of the user that uses the data source.

Password: password.

2.Main Methods

Method	Function
CreateInstance()	Create a Connection object.
Open()	Open an connection.
Execute()	Executes the query, SQL statement, or stored procedure specified in the CommandText or CommandStream property of the Command object.
Close()	Close an connection
Release()	Release the object created by CreateInstance()

4.1.2 _CommandPtr Interface

In MFC, _CommandPtr interface points to Command object of ADO component. This interface returns a record set.

It provides a simple method to execute the stored procedures and SQL statements of the record set.

4.1.3 _RecordsetPtr Interface

In MFC, _RecordsetPtr interface points to Recordset object of ADO component. _CommandPtr interface returns a _RecordsetPtr interface object.

Through _RecordSetPtr interface, we can access the record set. Get the value of each field with GetCollect() function.

4.2 ADO Database Programming Steps

With inserting record as an example, introduce ADO database programming steps

1. Introduce ADO Library File

In stdafx.h header file of the project, import ADO library file with “#import”, so that the compiler can compile correctly.

Method 1: import through the absolute path.

```
#import "c:\program files\common files\system\ado\msado15.dll " no_namespace
rename (_T("EOF"),_T("adoEOF"))
```

Method 2: copy msado15.dll file into the project directory, import according to the relative path.

```
#import "msado15.dll " no_namespace rename(_T("EOF"),_T("adoEOF"))
```

Description:

(1) Its functionality is the same as #include. That is, load ADO dynamic library msado15.dll.

(2) When you compile, the system will generate two C++ header files of msado15.tlh and ado15.tli to define ADO library.

(3) no_namespace: indicates that we don't use the namespace.

(4) rename("EOF","adoEOF"): indicates that we will change EOF used in ADO to adoEOF, so as to prevent naming conflicts.

Notice: this code needs to be completed in a line. If you write in two or more lines, you must add "\" symbol at the end of the line. It represents that these lines are regarded as a line.

2. Initialize COM Library (Initialize)

ADO library is a group of COM dynamic libraries. Before ADO is called, you must initialize OLE/COM library environment.

In the MFC-based applications, a better location to initialize OLE/COM library environment is in InitInstance member function of the application class. Initialize OLE/COM library environment with ::CoInitialize().

```
CXxxApp::InitInstance()
{
        ......;
        ::CoInitialize(NULL);
        ......;
}
```

3. Create _ConnectionPtr Object

By calling CreatInstance() function of _ConnectionPtr object, create _ConnectionPtr object.

```
_ConnectionPtr  pConnection  =  NULL;
pConnection.CreateInstance(__uuidof(Connection));
```

Notice: when method CreateInstance of _ConnectionPtr interface pointer, "." is used, but not "->".

4. Open Data Source, Establish Connection with Data Source

Through the database connection string, call Open() function of _ConnectionPtr pointer object, establish the connection with the data source.

```
// Set the connection string
CString  strConnect  =  Provider=SQLOLEDB.1;Data  Source=127.0.0.1;  Database=M
```

```
yDateBaseName;User  ID=MyUserName;Password=MyPassword;

// Connect to the database
pConnection->Open((_bstr_t)strConnect, _T(""), _T(""), adModeUnknown);
```

5. Execute Insert Operation to Database

(1) Create _CommadPtr Object

By calling CreateInstance() function of _CommandPtr object, create _CommandPtr object.

```
_CommandPtr pCommand = NULL;
pCommand.CreateInstance(__uuidof(Command));
```

(2) Set SQL Statement

The value of _CommandPtr's CommandText property can be directly set as the insert statement you want to execute.

```
pCommand->CommandText = _T("insert into Table_Option (Test_ID, Num) values(1,
50)");
```

You can also set SQL statement by passing the parameter.

```
pCommand->CommandText = _T("insert into Table_Option (Test_ID, Num) values
(?,?)");

// Set the parameter
_ParameterPtr pTestID = pCommand->CreateParameter( "Test_ID", adInteger, adP
aramInput, -1, option.GetTestID());
_ParameterPtr pNum = pCommand->CreateParameter( "Num", adInteger, adPara
mInput, -1, option.GetNum());

// Add the parameter
pCommand->Parameters->Append(pTestID);
pCommand->Parameters->Append(pNum);
```

(3) Get Database Connection Object

Assign pConnection to ActiveConnection property of pCommand.

```
pCommand->ActiveConnection = pConnection;
```

(4) Execute SQL Statement

Call Execute() function of _CommandPtr pointer object, execute SQL statement.

```
pCommand->Execute(&vRecordsAffected, NULL, adCmdText);
```

(5) Release _pCommandPt Object

By calling Release() function of _CommandPtr object, release _CommandPtr object.

6. Close Connection

Disconnect from the database with Close() method of _ConnectionPtr object. This method corresponds to Open() method.

By calling Close() function of _ConnectionPtr pointer object, close the connection with the database.

```
pConnection->Close()
```

7. Release Object

After the database connection is executed, you also need to release created _ConnectionPtr object with Release() method. This method corresponds to CreateInstance() method.

By calling Release() function of _ConnectionPtr object, release _ConnectionPtr object.

```
pConnection.Release()
```

8. Release COM Library (Uninitialize)

In CXxxApp ExitInitInstance(), release used COM resource with ::CoUninitialize().

```
CXxxApp::ExitInstance()
{
        ......;
        ::CoUninitialize();
        ......;
}
```

4.3 MFC Dialog Programming

In this program, we will use MFC standard form and some common controls: button, edit box, combo box, check box and list control.

5 Implementation Idea

5.1 Create Project

1.Launch Microsoft Visual Studio 2010. Through MFC application wizard, create MFC Dialog project TestManager.

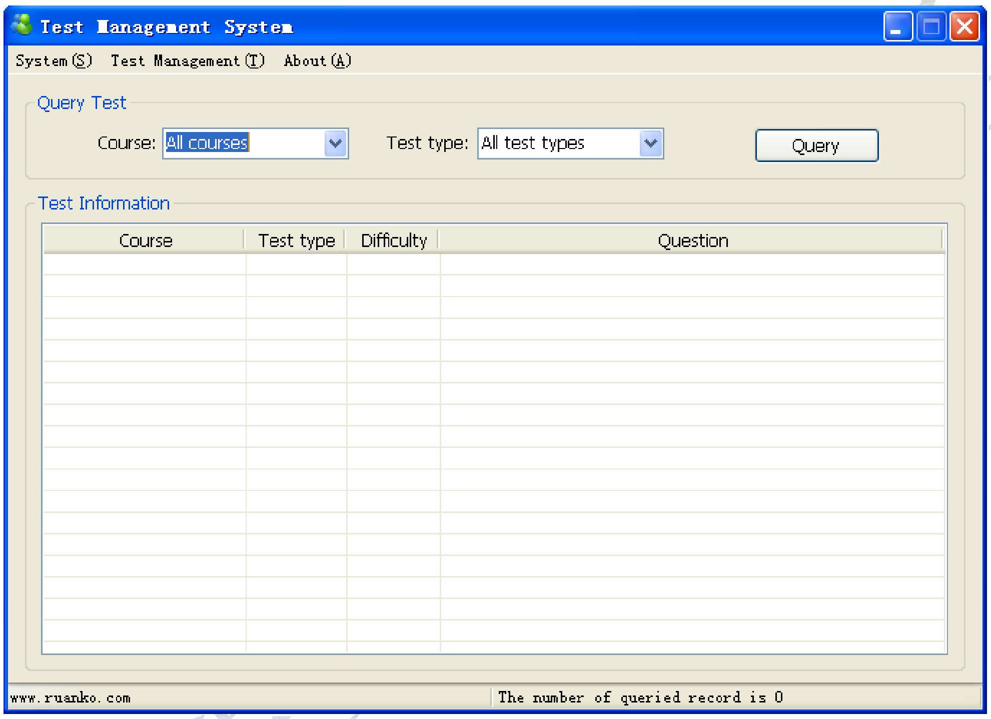
2.Compile and run the program.

5.2 Interface Layout

According to the requirements, design the interface of "Test Management System". The main interfaces in the system include Main interface and Edit Test interface.

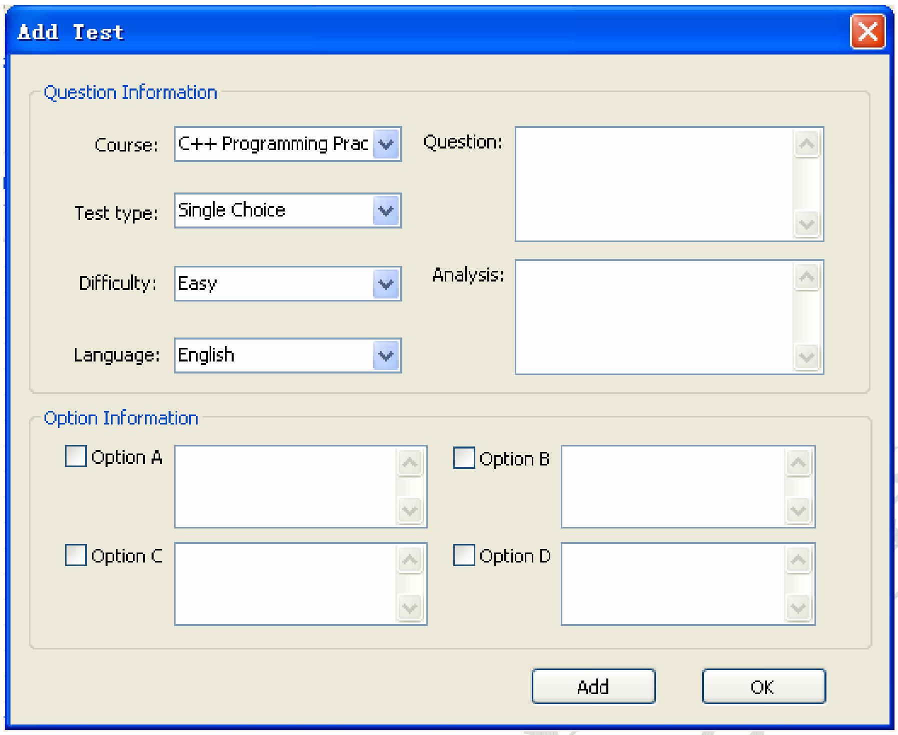
1. Modify Main Interface

With VS2010 development tool, add controls for Main interface, and add the menu bar a nd the status bar. The effect is as follows:



2. Add Edit Test Interface

Create edit test dialog resource. Create dialog class CAddTestDlg for the dialog box. And finish the interface layout. The effect is as follows:



Course: C++ Programming Prac
Test type: Single Choice
Difficulty: Easy
Language: English

5.3 Import ADO Library File

1. Import ADO Library File

(1) Find msado15.dll file in the system disk and copy it into the project directory.

(2) In stdafx.h file, import ADO library file with "#import" statement.

```
#import "msado15.dll " no_namespace rename(_T("EOF"),_T("adoEOF"))
```

2. Initialize ADO Library

In CTestManagerApp::InitInstance() function, add code and initialize OLE/COM library environment with ::CoInitialize(NULL).

```
BOOL  CTestManagerApp::InitInstance()
{
      ::CoInitialize(NULL);
}
```

3. Release ADO Library

Add ExitInstance() virtual function for CTestManagerApp class. In the function, release the environment with ::CoUninitialize().

```
int  CTestManagerApp::ExitInstance()
{
      ::CoUninitialize();
}
```

5.4 Add Course

When you add the test, you need to select the course and the test type that the test belongs to. As the basic data of the system, the course and the test type that the test belongs to can be input on the interface and saved into the tables of the database, and also can be directly inserted into the corresponding tables with SQL statements in the database.

In CAddTestDlg::OnOnInitDialog() function, set a piece of course information, save into Table_Course table of the database.

1.Set Course Information

In CAddTestDlg::OnOnInitDialog() function, set a piece of fixed course information.

```
CAddTestDlg::OnInitDialog()
{
      CString CourseName = _T("Database Application");
      CString CourseID = _T("0902");
}
```

2. Connect to Database

In CAddTestDlg::OnInitDialog() function, create _ConnectionPtr object, and connect to the database.

```
CAddTestDlg::OnInitDialog()
{
      // Set course information
      ……;
      // Set the connection string
      CString szConnect =  _T("Provider=SQLOLEDB.1;Data Source=127.0.0.1; Dat
abase=TestLibrary; User ID=sa;Password=123456");

      // Create the _ConnectionPtr object
      _ConnectionPtr pConnection = NULL;
      pConnection.CreateInstance(_uuidof(Connection));

      // Connect to the database
      pConnection->Open((_bstr_t)szConnect, _T(""), _T(""), adModeUnknown);
}
```

3. Set SQL Insert Statement

(1) Create _CommandPtr Object

In CAddDlg::OnInitDialog() function, create _CommandPtr object.

```
_CommandPtr  pCommand  =  NULL;
pCommand.CreateInstance(__uuidof(Command));
```

(2) Set SQL Statement

```
CString  strSql;
strSql.Fromat  =  _T("insert  into  Table_  Course  (CourseName  ,  CourseID)  values
(%s,  %s)");
pCommand->CommandText = (_bstr_t)strSql;
```

4.执 Execute SQL Insert Statement

(1) Get Database Connection Object

```
pCommand-> ActiveConnection = pConnection;
```

(2) Execute SQL Insert Statement

```
pCommand->Execute(NULL, NULL, adCmdText);
```

5. Release _CommandPtr Object

```
pCommand.Release ();
```

6. Close Database Connection

```
pConnection->Close();
```

7. Release _ConnectionPtr Object

```
pConnection.Release();
```

8. Compile and Run Program

The reader can implement adding the basic test type data according to his own idea, and add the test type into Table_TestType table.

During the development process, generally, these basic data are in the database, and directly inserted into the corresponding basic data tables with SQL statements.The program will get the basic data through query operations.

The SQL statements are as follows:

```
insert into Table_Course (CourseName, CourseID) values ('C++ Programming Practice',
'0901')
insert into Table_Course (CourseName, CourseID) values ('Database Application', '0902')
insert into Table_Course (CourseName, CourseID) values ('MFC Programming Practice',
'0903')

insert into Table_TestType(TypeName) values ('Single Choice');
insert into Table_TestType(TypeName) values ('Multiple Choice');
insert into Table_TestType(TypeName) values ('Short Answer');
```

5.5 Query Course

After you add the basic data into the tables in the database with SQL script, modify CAddTestDlg::OnInitDialog() function, read the course name and the test type name from Table_ Course table and Table_TestType table, then display them on the interface.

In this document, with Query Test as an example, we will introduce how to execute database query operation. The reader should finish Test Type Query functionality.

1. Connect to Database

In CAddTestDlg::OnInitDialog() function, create _ConnectionPtr object pConnection, and connect to the database with it.

2. Set SQL Query Statement

In CAddTestDlg::OnInitDialog() function, create _CommandPtr object pCommand, and insert SQL query statement into pCommand object with it.

```
CString strSql;
strSql.Fromat = _T("select ID, CourseName from Table_Course where Del = 0");
pCommand->CommandText = (_bstr_t)strSql;
```

3. Execute SQL Statement, Return Record.

(1) Define _RecordsetPtr Object

```
_RecordsetPtr pRecordset = NULL;
pRecordset.CreateInstance(__uuidof(Recordset));
```

(2) Execute SQL Statement with pCommand Object. Receive Execution Results with pRecordset Object.

```
pRecordset = pCommand->Execute(NULL, NULL, adCmdText);
```

4. Get Data in Record Set

Get data from the record set, and display into the course combo box on the interface.

```
While(!pRecordset -> adoEOF)
{
        int i= 0;
        // Get data from the record set
        int nId = pRecordset->GetCollect("ID");
        CString strCourse = pRecordset->GetCollect("CourseName");

        // Display the data into the combo box
        m_cbCourse.InsertString(i, strCourse);
        m_cbCourse.SetItemData(i, nId);
        i++;
}
```

5. Compile and Run Program

5.6 Add Test

After the course information and the test type information are successfully queried in the database and displayed on the interface, when you add the test, on the interface, you can select the course and the test type that the test belongs to. Then, implement saving the test question information of the test information into Table_TestQuestion table of the database, saving the option information into Table_Option table.

In Add Button message response function CAddTestDlg::OnBtnAdd(), there are two steps to save the test information into the database.

1. Save Test Question Information

Save the test question information into Table_TestQuestion table. After the test question information is saved successfully, get ID of this test question information item.

```
// Set the SQL statement of inserting question
pCommand->CommandText = _T("insert into Table_TestQuestion (Course_ID,
Type_ID, Language, TestContent, Difficulty, Analysis, Del) values(1, 1, 0,
m_szContent, 0, m_szAnalysis, 0)");

// Get ID of the record after Insert it successfully
pCommand->CommandText = _T("select @@identity as ID");
pRecordset = pCommand->Execute(&vRecordsAffected, NULL, adCmdText);
int nId = pRecordset->GetCollect(_T("ID"));
```

2. Save Option Information

Save the option information into Table_Optio table.

```
// Set the SQL statement of inserting option information
```

```
pCommand->CommandText = _T("insert into Table_Option (Test_ID, Num,
OptionContent, IsTrue, Del) values(nId, 0, m_szOptionA , 0, 0)");
```

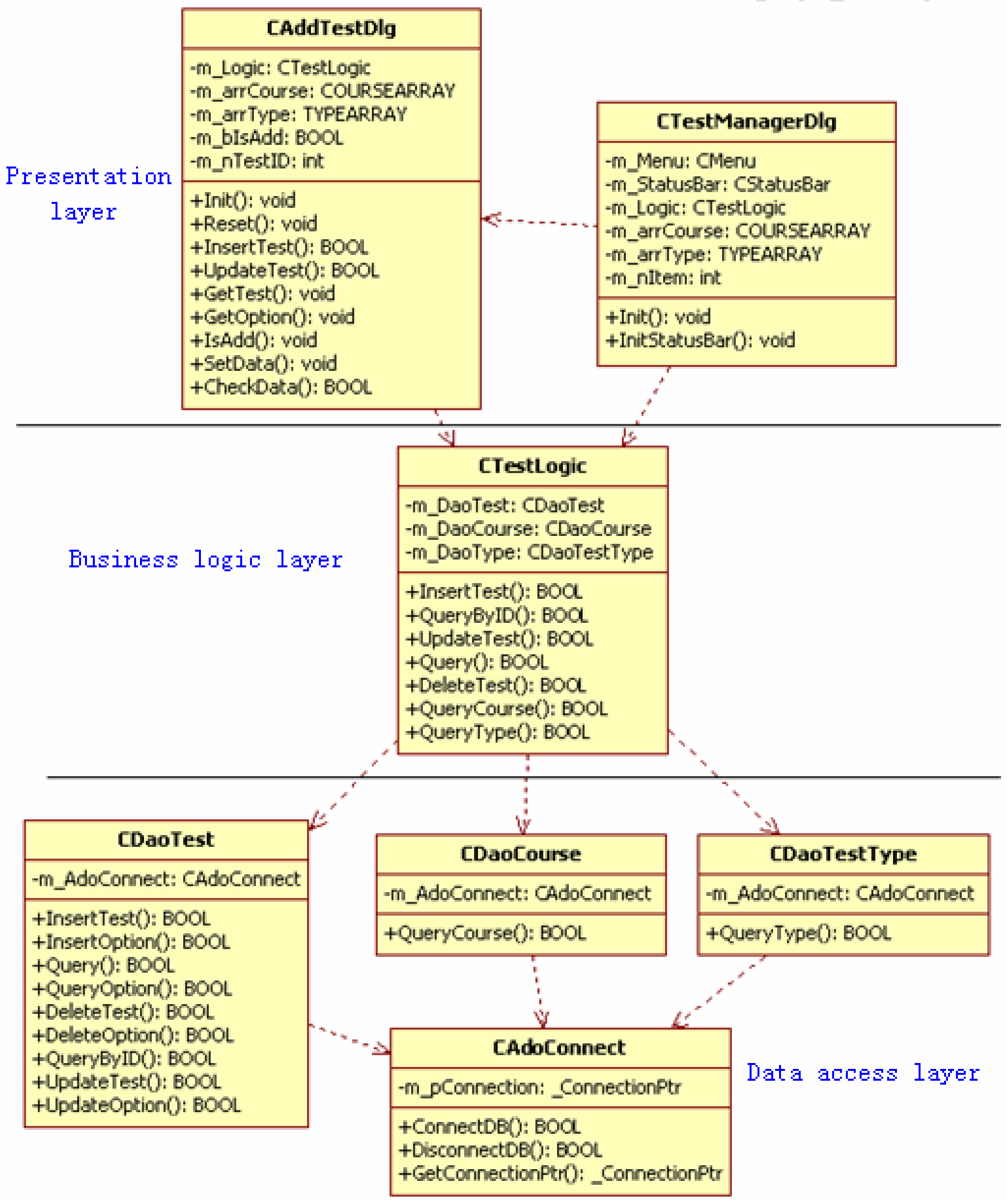
The reader himself can finish the functionalities of Query Test, Modify Test and Delete Test according to the above contents.

5.7 Program Optimization

1.Structure Optimization

Optimize the program structure. According to the three-layer structure in the architecture, the program can be layered into the presentation layer, the business logic layer, and the data access layer. Pass the entity object between the layers.

The optimized program structure is as follows:



2. Functionality Optimization

When you set the SQL statement, in order to avoid SQL injection, you can add the parameters in SQL statements into _CommandPt by passing the parameters.

```
// Set the SQL statement
pCommand->CommandText = _T("select * from Table_TestQuestion where Course
_ID = ? and Type_ID = ? and Del = 0");

// Set the parameter
_ParameterPtr pCourseID = pCommand->CreateParameter("Course_ID",adInteger,
adParamInput,-1,nCourseID);
_ParameterPtr pTypeID = pCommand->CreateParameter("Type_ID",adInteger,adPa
ramInput,-1,nTypeID);
// Add the parameter
pCommand->Parameters->Append(pCourseID);
pCommand->Parameters->Append(pTypeID);
```