Contents

[1. Brief Introduction 2](#_Toc367195387)

[2. Experiment: Add Record Function Using ADO.NET Classes 3](#_Toc367195388)

[2.1 How It Works – Examine the code in btnCreateRecord\_Click event handler 8](#_Toc367195389)

[3. Modify the Code and Observe the Runtime Error of the Application 9](#_Toc367195390)

[4. Error Handling (Try..Catch..Finally) To Manage User Expectations 11](#_Toc367195391)

[4.1 How does the Try-Catch-Finally Work 12](#_Toc367195392)

[5. Use "Using Block" to Make Computer Happy 13](#_Toc367195393)

**Data Access Topic (Part 1)**

# 1. Brief Introduction

Refer to Fig. 1. When you borrow a book from the school library, the librarian will use a Windows client application to logon to the application server. The application server will authenticate the user first, to make sure that the librarian is a valid user. Then, the Windows client application will capture the barcode information of your book and your admission number. The information is sent to the application server which receives the information and opens a connection to the library database to create a new loan record inside the database. After creating the new load record, the application server sends a confirmation message back to the Windows client program.

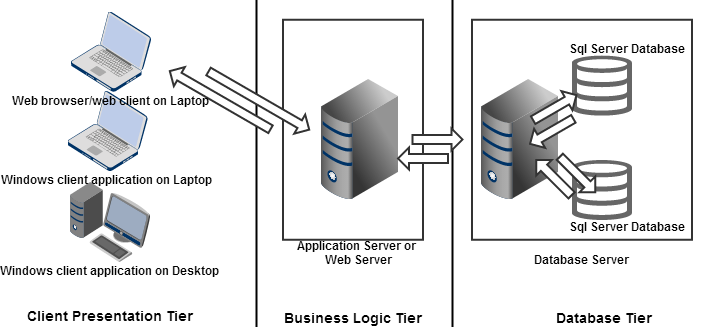


Fig. 1

This is an example of how the 3 Tier Architecture works to service thousands of users. This knowledge is widely discussed among the developer community. You can find a quick reference about 3-tier-architecture here: <http://blog.simcrest.com/what-is-3-tier-architecture-and-why-do-you-need-it/>. This module does not delve much into 3 Tier developments. All good IT developers started experimenting C# coding techniques to work with the database in their own laptop development environment. As a result, this document content aims to expose you to all the frequently applied coding techniques which form a firm foundation for you to build beautiful enterprise wide solutions that benefits thousands of users and your pocket.

# 2. Experiment: Add Record Function Using ADO.NET Classes

There is a Student table in an SQL database, ApplicationDevelopmentDB. Refer to Fig. 2, provide the required interface and C# logic to achieve the following descriptions:

i) When the user clicks the btnCreateRecord Button control, one dummy student record will be created inside the Student table.

ii) After creating the record, the user sees a useful message in the lblMessage Label control.

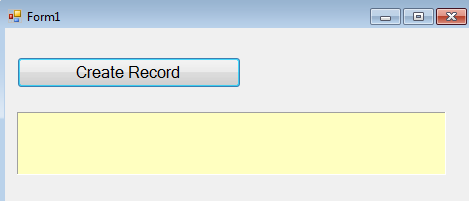


Fig. 2

btnCreateRecord

lblMessage

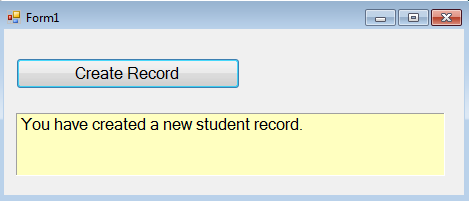


Fig. 3

Displays a simple message to provide assurance that a dummy student record has been created in the Student table successfully.



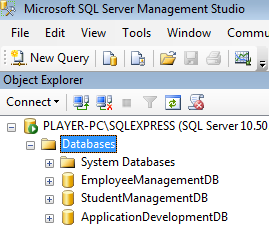
Fig. 4

A dummy student record is created inside the Student table.

***Step 1 Prepare the database and table***

1. Start the SQL Server Management Studio and create a new database, ApplicationDevelopmentDB.

Fig. 5



Create a database, in the SQL Server Management Studio environment, ApplicationDevelopmentDB

2. Refer to Fig. 6. Create a new table, Student. Ensure that the data type of each field follows the recommended settings provided in Fig. 6.

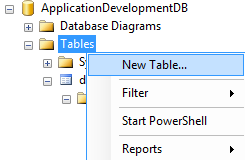
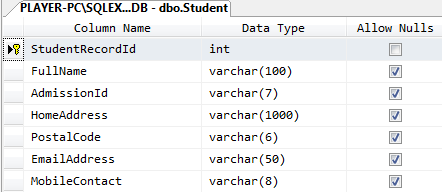


Fig. 6

Use the Column Properties window in the SQL Server Management Studio to set the first field, StudentRecordId as an Identity column.

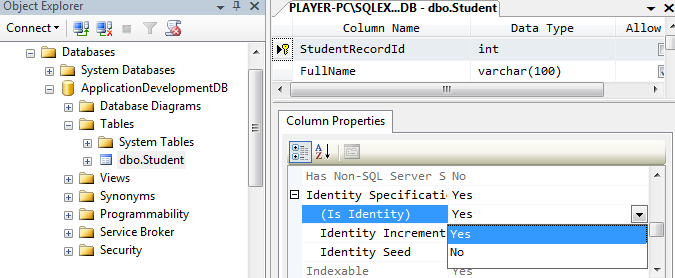


Fig. 7

Note that the table can also be created using SQL statements, in this case:

CREATE TABLE Student

(

StudentRecordId int not null IDENTITY,

FullName varchar(100),

AdmissionId varchar(7),

HomeAddress varchar(1000),

PostalCode varchar(6),

EmailAddress varchar(50),

MobileContact varchar(8),

PRIMARY KEY (StudentRecordId)

)

***Step 2 Design the Form***

1. Create a new Windows Form (Form1) Application project, ExperimentAdoNetObjects.

2. Refer to the Fig. 8 below; provide one Button control and one Label control in Form1.

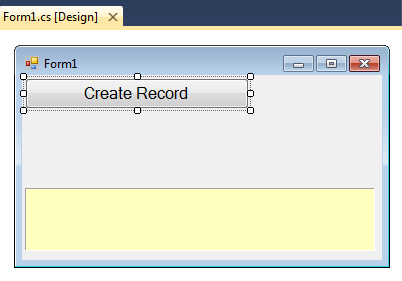


Fig. 8

btnCreateRecord

lblMessage

***Step 3 Edit the Form1.cs code***

1. Add the System.Data.SqlClient namespace in Form1.cs. With this namespace, you can use the necessary classes (e.g. SqlConnection, SqlCommand etc) together with your code which accesses and manages the SQL Server database.

using System.Data.SqlClient;

2. Refer to code listing 1, provide the code in the btnCreateRecord\_Click event handler which will connect to the database and create dummy test data inside the Student table.

| Listing 1 btnCreateRecord\_Click event handler for the Form1 code section. |
| --- |
| 1. private void btnCreateRecord\_Click(object sender, EventArgs e) 2. **{** 3. string userMessage = ""; 4. SqlConnection conn = new SqlConnection(); 5. SqlCommand cmd = new SqlCommand(); 6. conn.ConnectionString = "Data Source=your sqlserver name;" +   Provide your own SqlServer name here.   1. "database=ApplicationDevelopmentDB;" + 2. "integrated security=true"; 3. cmd.Connection = conn; 4. cmd.CommandText = "INSERT INTO Student (FullName, " +   "AdmissionId, HomeAddress, PostalCode, EmailAddress, MobileContact)" +  "Values ('Zen Koh Wai Peng','1234567'," +  "'Blk 111, #06-11 Bukit Panjang Road 3', '566111','zen@hotmail.com','90008888') ";  The SQL statement is hardcoded.   1. conn.Open(); 2. cmd.ExecuteNonQuery(); 3. conn.Close(); 4. userMessage = "You have created a new student record."; 5. lblMessage.Text = userMessage; 6. **}** |

***Step 4 Test the Form1 and observe the behavior of the Student table***

1. Click the btnCreateRecord Button control 4 times, there should be 4 test records created inside the Student table.

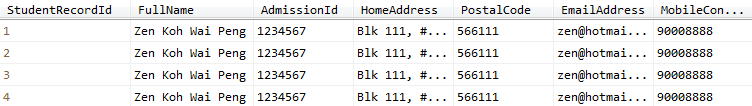


Fig. 9

The first column of the Student table, StudentRecordId is a primary key column. This column is also an Identity column (<http://en.wikipedia.org/wiki/Identity_column>). Examine the Student table in the SQL Server Management Studio environment, the StudentRecordId column is not editable. When a new record is created in the Student table, the database will provide a new unique value in this field.

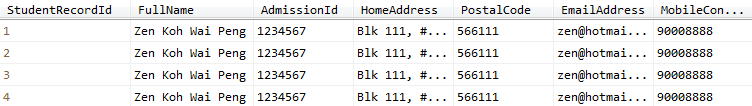


Fig. 10

The StudentRecordId column's value is not updateable by you. The database will provide a new value when a new record is created.

2. Within the SQL Server Management Studio environment, manually delete two student records in the Student table. Then, add a new student record by clicking the btnCreateRecord Button control in the Windows form. Observe that the new record's StudentRecordId value will not reuse the deleted records' StudentRecordId value. The Fig. 11 below shows that 2 records (which have record IDs 2 and 3) have been deleted. A new record will have a new student record ID value of 5.

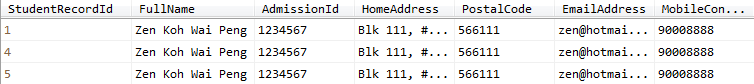


Fig. 11

This record is added after deleting 2 records which were created earlier.

## 2.1 How It Works – Examine the code in btnCreateRecord\_Click event handler

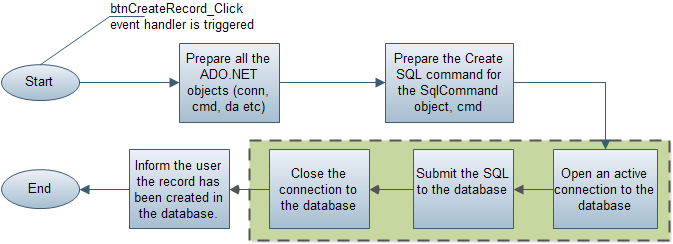


Fig. 12

Refer to Listing 1, the following statements at line 4 and 5:

SqlConnection conn = new SqlConnection();

SqlCommand cmd = new SqlCommand();

will create one SqlConnection object, conn, and one SqlCommand object, cmd. This SqlConnection class object is responsible for connecting to the SQL Server database. Without this object, your code cannot communicate with the database at all. The SqlCommand object, cmd is responsible in holding the necessary SQL statements, such as INSERT, UPDATE, DELETE, and SELECT SQL statements, which are to be submitted to the database for execution. In this section, you have used the INSERT SQL statements for the SqlCommand object to create a dummy student record.

The Connection object, conn, needs to know the following information so that it can connect successfully to a database server.

i) Server name (e.g. laptopname\sqlexpress)

ii) The database name (e.g. ApplicationDevelopmentDB)

The above information is given to the SqlConnection object, conn, via the ConnectionString property (line 6).

conn.ConnectionString = "Data Source=laptopname\\sqlexpress;" +

"database=ApplicationDevelopmentDB;" +

"integrated security=true";

You can find a good beginner tutorial on database connection techniques in various situations with this reference:

<http://www.codeproject.com/Articles/4416/Beginners-guide-to-accessing-SQL-Server-through-C>

The statement at line 9, cmd.Connection = conn; is telling the SqlCommand object, cmd to use the SqlConnection object, conn. This statement is assigning the SqlConnection object, conn into the Connection property of the SqlCommand object, cmd.

Without this statement, the SqlCommand object, cmd, does not know the destination server which is supposed to receive its SQL.

The SqlCommand object, cmd, will need to hold an SQL statement such as

INSERT INTO Student (FullName, AdmissionId, HomeAddress, PostalCode, EmailAddress, MobileContact) Values ('Zen Koh Wai Peng','1234567','Blk 111, #06-11 Bukit Panjang Road 3', '566111','zen@hotmail.com','90008888')

The database will recognize the above SQL statement as a valid INSERT SQL command to create a record in the Student table.

Refer to line 10:

cmd.CommandText = "INSERT INTO Student (FullName, " +

"AdmissionId, HomeAddress, PostalCode, EmailAddress, MobileContact)" +

"Values ('Zen Koh Wai Peng','1234567'," +

"'Blk 111, #06-11 Bukit Panjang Road 3', '566111','zen@hotmail.com','90008888') ";

The SQL command is assigned as a string of characters into the CommandText property of the SqlCommand object, cmd.

Before the SQL can be sent to the database, you need the statement at line 11, conn.Open(); to tell the SqlConnection object, conn to begin communicating with the database server by calling its Open() method.

At line 12, the following statement:

cmd.ExecuteNonQuery();

calls the ExecuteNonQuery() method of the SqlCommand object, cmd. As a result, the cmd will send the SQL command to the database through the SqlConnection object, conn.

After submitting the SQL to the database server, the database server will recognize the command to create a new dummy student record. At line 13, you will need to close the connection by calling the Close() method which belongs to the SqlConnection class object, conn.

conn.Close();

Line 14 and 15 are just some statements to update the Label control, lblMessage, to show that the code in the btnCreateRecord\_Click event handler has successfully completed its task.

# 3. Modify the Code and Observe the Runtime Error of the Application

The next topic is about error handling using the Try-Catch-Finally technique. To appreciate the next section better, it is best to try the following 4 actions to intentionally create a runtime error (runtime exceptions).

Action 1. Comment out the command, conn.Open() at line 11 and see what happens during runtime. // conn.Open()

Action 2. Change the connection string information (e.g. purposely give an inappropriate server name) and see what happens during runtime.

conn.ConnectionString = "Data Source=omgExamIsComing\\SqlExpress;" +

"database=ApplicationDevelopmentDB;" +

"integrated security=true";

Action 3. Purposely make the INSERT command SQL wrong and see what happens during runtime. cmd.CommandText = "INSERT INTO StudentABC (FullName,...."

Action 4. Delete the cmd.Connection = conn and see what happens during runtime.

The notes here will not discuss all 4 actions. Here is a description on what will happen if you try out the first action. If conn.Open() is commented out, the SqlConnection object, conn, will not be able to open a connection to the database. As a result, a runtime error will occur when the SqlCommand object, cmd, tries to send the SQL statement to the database.

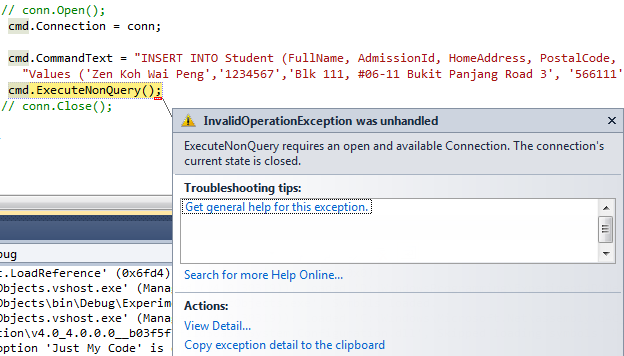


Fig. 13

Commenting out the conn.Open() statement will cause a runtime error when cmd.ExecuteNonQuery() is executed.

Examine the SQL that search and view student record(s)

Build a windows form that allows the user to do the same thing using a Textbox control and the GridView control, grdData.

The observations should hintthat all the 4 actions will lead to runtime errors when the runtime system executes either the statement, conn.open() or the statement, cmd.ExecuteNonQuery().

# 4. Error Handling (Try-Catch-Finally) To Manage Expectations

In the previous *Section* *3. Modify the Code and Observe the Runtime Error of the Application*, anything can happen when your application attempts to connect to a database server. For example:

1. If the database server is broken, conn.Open() will throw an exception.
2. If the SQL command is operating on a table that does not exist, your cmd.ExecuteNonQuery() will be in deep trouble and throws an exception.

Each of these scenarios will throw an exception. To allow an application to handle problems gracefully or recover from a runtime error beautifully, the Try-Catch-Finally is a neat way of letting you take control of unpredictable situations. You can apply the Try-Catch-Finally in the code within the btnCreateRecord\_Click event handler as shown in the listing below:

| Listing 2 btnCreateRecord\_Click event handler (Using Try-Catch-Finally) |
| --- |
| 1. private void btnCreateRecord\_Click(object sender, EventArgs e)   The statement at line 20, conn.Close() will always execute because it is in the Finally block. There is a long story why a connection to the database must be closed when no longer in use.<http://stackoverflow.com/questions/4111594/why-always-close-database-connection>   1. **{** 2. SqlConnection conn = new SqlConnection(); 3. SqlCommand cmd = new SqlCommand(); 4. string userMessage = ""; 5. conn.ConnectionString = "Data Source=xyz;" + 6. "database=ApplicationDevelopmentDB;" + 7. "integrated security=true"; 8. cmd.Connection = conn; 9. try**{**   Wrap the code from line 12 to line 16 with the Try block so that if anything goes wrong, the code at the Catch block (line 18 to 19) will execute to avoid giving users a heart attack.   1. conn.Open(); 2. cmd.CommandText = "INSERT INTO Student (FullName, AdmissionId, HomeAddress, PostalCode, EmailAddress, MobileContact)" + 3. "VALUES ('Zen Koh Wai Peng','1234567','Blk 111, #06-11 Bukit Panjang Road 3', '566111','zen@hotmail.com','90008888') "; 4. cmd.ExecuteNonQuery(); 5. userMessage = "You have created a new student record."; 6. **}**catch(Exception ex)**{** 7. Console.WriteLine(ex.Message); *//This statement is just for debugging purpose* 8. userMessage = "Something wrong has occurred."; 9. **}**finally**{** 10. conn.Close(); 11. Console.WriteLine("Closed the connection");*//for debugging and observing purpose* 12. **}** 13. lblMessage.Text = userMessage; 14. **}** |

## 4.1 How does the Try-Catch-Finally Work

Error handling is also known as exception handling. A good reference can be found at <http://www.dotnetperls.com/exception>. In this section, you will analyze code listing 2 to find out how the Try-Catch-Finally block helps the code which does the insertion of the dummy student record into the Student table.

The syntax structure of the Try-Catch-Finally control structure is as follows:

**try**

try-block

**catch (exception-type name)**

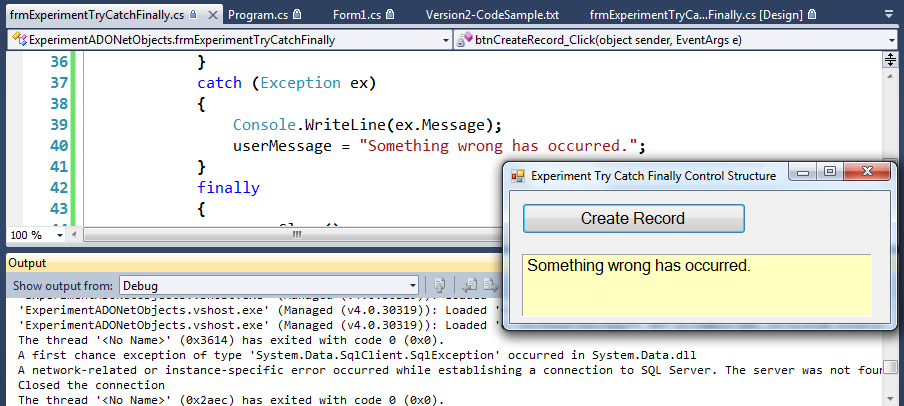
catch-block

**finally**

finally-block

The statements from line 11 to line 15 are wrapped in the Try block. If the conn.Open() statement at line 11 and the statement, cmd.ExecuteNonQuery() at line 14 throw an exception (runtime error has occurred), the Catch block will execute.

Within the Catch block, the Console.WriteLine(...) statement at line 17 is used often by developers so that they can verify that the code at the Catch block is indeed running when the runtime error happens. In this case, the Console.WriteLine() is displaying the description of the error inside the Output window in the Visual Studio IDE. Fig. 14 below describes what has happened when the statement conn.Open() throws an exception due to a wrong database server name.



This message is contributed by the Console.WriteLine() statement in the Finally block.

Fig. 14

Console.WriteLine(...) writes information to the Output window when running the application in Debug mode. The Message property of the Exception class object, ex describes the error that occurred

The code inside the Finally clause will be executed in all cases, both in case of errors and in case of error-free execution of Try part. Therefore, the statement, conn.Close() at line 20 will execute to close the database connection. The Console.WriteLine(...) at line 21 inside the Finally block is used only for observation purpose to provide assurance that the code in the Finally block has executed (Fig. 14).

# 5. Use "Using Block" to Make Computer Happy

The next section will introduce an important keyword called ‘Using’ (<http://www.dotnetperls.com/using>). The Using block is necessary so that your program has better computer memory management. There are many articles talking about it, such as <http://stackoverflow.com/questions/212198/what-is-the-c-sharp-using-block-and-why-should-i-use-it>. Applying the Using block does not require significant change to the code. Refer to Listing 3 and provide the necessary changes to the btnCreateRecord\_Click event handler. You will notice that your program will perform the same functionality.

| Listing 3 btnCreateRecord\_Click event handler for the Form1 code section |
| --- |
| 1. private void btnCreateRecord\_Click(object sender, EventArgs e) 2. **{**string userMessage = ""; 3. using (SqlConnection conn = new SqlConnection())**{**   Provide your own SqlServer name here.   1. using (SqlCommand cmd = new SqlCommand())**{** 2. try**{** 3. conn.ConnectionString = "Data Source=player-pc\\sqlexpress;" + 4. "database=ApplicationDevelopmentDB;" + 5. "integrated security=true"; 6. conn.Open(); 7. cmd.Connection = conn; 8. cmd.CommandText = "INSERT INTO Student (FullName, AdmissionId, HomeAddress, PostalCode, EmailAddress, MobileContact)" + 9. "Values ('Zen Koh Wai Peng','3234567','Blk 111, #06-11 Bukit Panjang Road 3', '566111','zen@hotmail.com','90008888') "; 10. cmd.ExecuteNonQuery(); 11. userMessage = "You have created a new student record."; 12. **}**catch(Exception ex)**{** 13. Console.WriteLine(ex.Message); 14. userMessage = "Something wrong has occurred."; 15. **}**finally**{** 16. conn.Close(); 17. Console.WriteLine("Closed the connection"); 18. **}** 19. **}***//end of using cmd* 20. **}***//end of using conn* 21. lblMessage.Text = userMessage; **}** |

End of Part 1