

MySQL Data Loading with Lookup Tables

Written by
Dr. Ernest Bonat, Ph.D.
Visual WWW, Inc.



Visual WWW, Inc.

'Create by Developers for Developers'

www.evisualwww.com | info@evisualwww.com

Copyright @ 2000 - 2007 Visual WWW, Inc. All right reserved

Table of Content

Introduction	2
Required Software	2
LoopUp Table Data Loading	2
Generic ListItemClass Class	4
LoopUp Project Example	5
Getting Selected ID and Name Values	9
Finding ID and Name Values	10
Finding the State and Capital Names by Postal Code	11
Executing Stored Procedures in MySQL 5.0/VB.NET 2005	14
Conclusions	16
Visual WWW Legal Notice	16

Introduction

Lookup tables contain, in general, a fixed list of data. This data doesn't change very often in database business applications. Examples of this data could be a product list, category type, supplier list, state name, zip code, phone area code, etc. In Windows and Internet web business applications, most of these lookup tables are graphically implemented by using ComboBox, ListBox or CheckListBox read-only controls. These controls are loaded with data using two main columns, ID and Name. For example, the USA state table, the ID could be 'CA' and the Name 'California'. Some times, for standard Windows form and Internet web page we need to show data to the end-users from many of these lookup tables. A fast data loading process and defining the main column values for each lookup table is required. In this article I will show you standard lookup data loading procedure and the generic classes object to store and read-only the values of the ID and Name columns from the lookup tables. Selecting and finding the ID and Name values will be provided. Executing stored procedures with input/output parameters in MySQL 5.0/VB.NET 2005 will be covered in detail too.

Required Software

- [MySQL Database Server 5.0.41](#)
- [MySQL Connector/NET 5.0.6](#)
- [Toad for MySQL Freeware 2.0.3](#)
- [Microsoft Visual Basic 2005 Express Edition](#)

LoopUp Table Data Loading

Let's look for a simple way to load the data in a ComboBox from a lookup table. Because we may have many lookup tables, it makes sense to develop a generic class for data loading from these tables. Listing 1 shows the structure of the LookUpClass public class within the LoopUpLibrary Namespace. The MySql.Data.MySqlClient library has been imported to reference the Connector/NET 5.0.6. The standard public class ObjectDisposeClass was included for releasing .NET unmanaged resources. Inside the LookUpClass class body we will develop our custom properties, methods and events.

```
Imports MySql.Data.MySqlClient
Namespace LoopUpLibrary
    Public Class LoopUpClass
        Inherits ObjectDisposeClass
        ' Developed custom properties, methods and events
    End Class
#Region " IDisposable Object ..."
    Public Class ObjectDisposeClass
        Implements IDisposable
        Private disposedValue As Boolean = False
```

```
Public Sub Dispose() Implements IDisposable.Dispose
    Dispose(True)
    GC.SuppressFinalize(Me)
End Sub
Protected Overridable Sub Dispose(ByVal disposing As Boolean)
    If Not Me.disposedValue Then
        If disposing Then
            ' TODO: free unmanaged resources when explicitly called
        End If
        ' TODO: free shared unmanaged resources
    End If
    Me.disposedValue = True
End Sub
End Class
#End Region
End Namespace
```

Listing 1: LookUpClass class structure

The LoopUpDataLoad() subroutine is shown in Listing 2. This subroutine has two main input parameters, the control to be loaded (pComboBox) and the stored procedure (pStoredProcedureString) that contains the SQL Select statement of the Lookup table. Some user stored procedures examples will be explained late in this article (Listing 5 and 13). As we know, MySQL Server 5.0 introduces, for the first time, the capability of development stored procedures, functions, triggers and views database objects. Because of that, instead of passing a dynamic SQL Select statement, the stored procedure name is passed by value to the subroutine. Stored procedures offer several distinct advantages over dynamic (embedding) SQL in your application code. In the future, I'm going write a single paper about stored procedures development and implementation using MySQL 5.0/VB.NET 2005. I can see many Open Source application developers still using dynamic or/and parameterized SQL statements today with MySQL Server 5.0. It seems to me that they don't know how to design and develop stored procedures using SQL:2003 language. Just in case, a very good reference book about this topic was published in 2006 by [O'Reilly Media, Inc.](#), ISBN: 0-596-10089-2, "[MySQL Stored Procedure Programming](#)" by Guy Harrison and Steven Feuerstein. I highly recommend reading this book for any Windows or Internet web application development with MySQL 5.0 database engine.

```
Public Sub LoopUpDataLoad(ByVal pComboBox As ComboBox, _
                        ByVal pStoredProcedureString As String, _
                        ByRef pErrorMsgString As String)
    Dim IDString, NameString As String
    Try
        Using mMySQLConnection As New MySqlConnection(mMySQLConnectionString)
            mMySQLConnection.Open()
            Using mMySQLCommand As New MySqlCommand
                With mMySQLCommand
                    .Connection = mMySQLConnection
                    .CommandType = CommandType.StoredProcedure
                    .CommandText = pStoredProcedureString
                    mMySQLDataReader = .ExecuteReader(CommandBehavior.SingleResult)
                End With
                With pComboBox
                    .Items.Clear()
                    If mMySQLDataReader.HasRows Then
                        .BeginUpdate()
                        While mMySQLDataReader.Read()
                            IDString = mMySQLDataReader.Item(0).ToString()
                            NameString = mMySQLDataReader.Item(1).ToString()
                            If IDString.Length > 0 And NameString.Length > 0 Then
                                Using mListItemClass As New ListItemClass(NameString, IDString)
                                    .Items.Add(mListItemClass)
                                End Using
                            End If
                        End While
                    End If
                End With
            End Using
        End Using
    Catch ex As Exception
        pErrorMsgString = ex.Message
    End Try
End Sub
```

```
                End Using
            End If
        End While
        .EndUpdate()
        .SelectedIndex = 0
    End If
End With
End Using
End Using
Catch exErr As Exception
    pErrorMsgString = exErr.Message
End Try
End Sub
```

Listing 2: Generic data loading procedure from lookup tables

So far at this point, for the available MySQL Connector/NET 5.0.6, the fastest way to retrieve data from the MySQL Server 5.0 is by using the data reader object `MySqlCommandReader`. XML implementation and data retrieval from MySQL database server is not available at this time. We hope to have these technologies implemented in future releases of MySQL, as Microsoft (SQL Server), Oracle (Oracle Server) and IBM (DB2 Mainframe Server) already have done. As the help file said (C:\Program Files\MySQL\MySQL Connector Net 5.0.6\Documentation\MySql.Data.chm) the data reader object provides a means of reading a forward-only stream of rows from a MySQL database. This object is created by calling the `ExecuteReader()` method of the command object `MySqlCommand` as shown in the code (Listing 2). By reading the `mMySqlCommandReader` object, we can get the postal code (`IDString`) and the state name (`NameString`). After these two values are determined, they need to be passed to the generic class constructor `ListItemClass` (Listing 3) and then the entire object will be adding to the item collection of the `ComboBox`. I would like to mention two new methods implemented in `ComboBox` control in VB.NET 2005, `BeginUpdate()` and `EndUpdate()`. The `BeginUpdate()` method prevents the control from repainting until the `EndUpdate()` method is called. In this case the user will no see the flicker during the drawing of the `ComboBox` when the items are being added to the list.

Generic ListItemClass Class

The `ListItemClass` class (Listing 3) was created to store and read-only the values of the ID and Name columns from the lookup tables. In general the Name column is defined as a character data type. The values of this column will be stored in `mFieldNameString` variable by using the class constructor. The read-only property `FieldName()` will retrieve these values. The ID column could be a character or a numeric data type, depend on the table definition. In this case, the class implements two read-only properties, one for character data type `FieldIDString()` and another for numeric `FieldIDInt32()`. This generic `ListItemClass` class should be included in any Windows or Internet web application project and can be used with any lookup control as `ComboBox`, `ListBox` or `CheckListBox`.

```
Public Class ListItemClass
    Inherits ObjectDisposeClass
    Private mFieldNameString As String
    Private mFieldIDString As String
    Private mFieldIDInt32 As Int32
    Public Sub New(ByVal pFieldNameString As String,
                  ByVal pFieldIDInt32 As Int32)
        mFieldNameString = pFieldNameString
        mFieldIDInt32 = pFieldIDInt32
    End Sub
    Public Sub New(ByVal pFieldNameString As String,
                  ByVal pFieldIDString As String)
        mFieldNameString = pFieldNameString
        mFieldIDString = pFieldIDString
    End Sub
```

```
Public Sub New()  
    mFieldNameString = Nothing  
    mFieldIDString = Nothing  
    mFieldIDInt32 = Nothing  
End Sub  
Public ReadOnly Property FieldName() As String  
    Get  
        Return (mFieldNameString)  
    End Get  
End Property  
Public ReadOnly Property FieldIDInt32() As Int32  
    Get  
        Return (mFieldIDInt32)  
    End Get  
End Property  
Public ReadOnly Property FieldIDString() As String  
    Get  
        Return (mFieldIDString)  
    End Get  
End Property  
Public Overrides Function ToString() As String  
    Return (mFieldNameString)  
End Function  
End Class
```

Listing 3: Generic ListItemClass class

LoopUp Project Example

To show a real example of lookup data loading I created a simple VB.NET 2005 solution project shown in Figure 1. The two ComboBoxes are loaded with USA States and Capitals respectively.

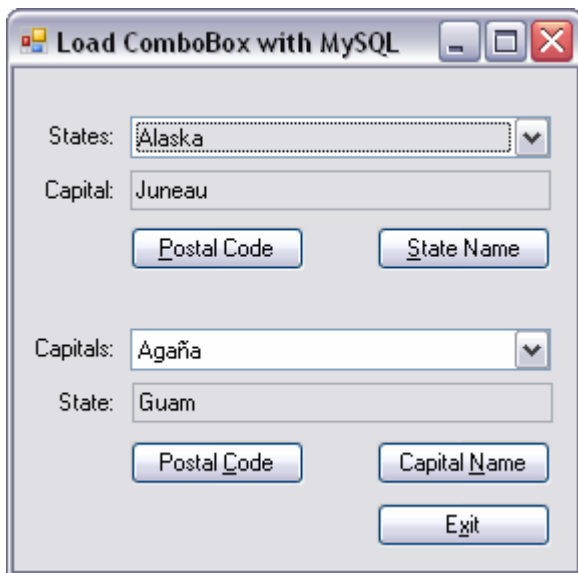


Figure 1: ComboBox data loading for USA States and Capitals

The load event of the VB.NET Form is shown in Listing 4. As you can see, both ComboBoxes, the States (StatesComboBox) and Capitals (CapitalsComboBox) are loaded using the same LoopUpDataLoad() subroutine shown above in Listing 2. If an error occurred, it gets stored in mErrorMsgString variable and shown to the end-users. It's a very good programming practice to provide error handling capability in all your application code. Many published VB.NET papers do not provide this important implementation code today.

```
Private Sub MySQLComboBoxForm_Load(ByVal sender As System.Object, _  
                                   ByVal e As System.EventArgs) _  
    Handles MyBase.Load  
    Cursor = Cursors.WaitCursor  
    Try  
        Using LoopUpObject As New LoopUpClass(mMySQLConnectionString)  
            Call LoopUpObject.LoopUpDataLoad(StatesComboBox, _  
                                             "usp_states_select_postal_name", _  
                                             mErrorMsgString)  
            If Not IsNothing(mErrorMsgString) Then  
                Cursor = Cursors.Default  
                MessageBox.Show(mErrorMsgString, _  
                                Me.Text, _  
                                MessageBoxButtons.OK, _  
                                MessageBoxIcon.Error)  
            End If  
            Call LoopUpObject.LoopUpDataLoad(CapitalsComboBox, _  
                                             "usp_states_select_postal_capital", _  
                                             mErrorMsgString)  
            If Not IsNothing(mErrorMsgString) Then  
                Cursor = Cursors.Default  
                MessageBox.Show(mErrorMsgString, _  
                                Me.Text, _  
                                MessageBoxButtons.OK, _  
                                MessageBoxIcon.Error)  
            End If  
        End Using  
    Catch exError As Exception  
        MessageBox.Show(exError.Message)  
    End Try  
    Cursor = Cursors.Default  
End Sub
```

Listing 4: Load event of the form "Load ComboBox with MySQL"

The user stored procedures `usp_states_select_postal_name` and `usp_states_select_postal_capital` are passed by value and shown in Listing 5. Both procedures was developed based on the USA state table ('state') with data (Listing 6).

```
DROP PROCEDURE IF EXISTS `usp_states_select_postal_name`;  
CREATE PROCEDURE `usp_states_select_postal_name`()  
BEGIN  
    SELECT `states`.`postal`, `states`.`statename`  
    FROM `states`  
    ORDER BY `states`.`postal`;  
END;  
  
DROP PROCEDURE IF EXISTS `usp_states_select_postal_capital`;  
CREATE PROCEDURE `usp_states_select_postal_capital`()  
BEGIN  
    SELECT `states`.`postal`, `states`.`capital`  
    FROM `states`  
    ORDER BY `states`.`capital`;  
END;
```

Listing 5: User stored procedures to select USA States and Capitals

```
DROP TABLE IF EXISTS `states`;
```

```
CREATE TABLE `states` (  
  `statename` varchar(20) NOT NULL,  
  `abbrev` varchar(10) NOT NULL,  
  `postal` char(2) NOT NULL,  
  `capital` varchar(20) NOT NULL,  
  PRIMARY KEY (`postal`),  
  KEY `statename` (`statename`),  
  KEY `capital` (`capital`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;  
  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Alaska','Alaska','AK','Juneau');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Alabama','Ala.','AL','Montgomery');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Arkansas','Ark.','AR','Little Rock');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Arizona','Ariz.','AZ','Phoenix');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('California','Calif.','CA','Sacramento');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Colorado','Colo.','CO','Denver');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Connecticut','Conn.','CT','Hartford');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Dist. of Columbia','D.C.','DC','Washington');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Delaware','Del.','DE','Dover');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Florida','Fla.','FL','Tallahassee');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Georgia','Ga.','GA','Atlanta');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Guam','Guam','GU','Agaña');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Hawaii','Hawaii','HI','Honolulu');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Iowa','Iowa','IA','Des Moines');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Idaho','Idaho','ID','Boise');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Illinois','Ill.','IL','Springfield');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Indiana','Ind.','IN','Indianapolis');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Kansas','Kans.','KS','Topeka');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Kentucky','Ky.','KY','Frankfort');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Louisiana','La.','LA','Baton Rouge');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Massachusetts','Mass.','MA','Boston');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Maryland','Md.','MD','Annapolis');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Maine','Maine','ME','Augusta');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
```

```
('Michigan','Mich.','MI','Lansing');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Minnesota','Minn.','MN','St Paul');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Missouri','Mo.','MO','Jefferson City');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Mississippi','Miss.','MS','Jackson');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Montana','Mont.','MT','Helena');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('North Carolina','N.C.','NC','Raleigh Durham');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('North Dakota','N.D.','ND','Bismarck');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Nebraska','Nebr.','NE','Lincoln');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('New Hampshire','N.H.','NH','Concord');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('New Jersey','N.J.','NJ','Trenton');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('New Mexico','N.M.','NM','Santa Fe');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Nevada','Nev.','NV','Carson City');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('New York','N.Y.','NY','Albany');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Ohio','Ohio','OH','Columbus');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Oklahoma','Okla.','OK','Oklahoma City');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Oregon','Ore.','OR','Salem');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Pennsylvania','Pa.','PA','Harrisburg');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Puerto Rico','P.R.','PR','San Juan');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Rhode Island','R.I.','RI','Providence');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('South Carolina','S.C.','SC','Columbia');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('South Dakota','S.D.','SD','Pierre');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Tennessee','Tenn.','TN','Nashville');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Texas','Tex.','TX','Austin');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Utah','Utah','UT','Salt Lake City');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Virginia','Va.','VA','Richmond');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Virgin Islands','V.I.','VI','Charlotte Amalie');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Vermont','Vt.','VT','Montpelier');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Washington','Wash.','WA','Olympia');
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values
('Wisconsin','Wis.','WI','Madison');
```

```
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('West Virginia','W.Va.','WV','Charleston');  
insert into `states`(`statename`,`abbrev`,`postal`,`capital`) values  
('Wyoming','Wyo.','WY','Cheyenne');
```

Listing 6: State table definition and SQL insert data statements

Getting Selected ID and Name Values

After the ComboBoxes are loaded with data, the users can choose any items (States and Capitals) by clicking and selecting on them. The selected ID and Name values must be known by the program for any possible selection. In general this ID represents a foreign key in the master table to keep data integrity, inserting and updating records. The Name values are showing to the user and some times it requires data validation depending on application business rules. In the last couple of years I have gotten a lot of requests from many users to load the combination of the ID and Name ([ID] – [Name]) as a general shown Name field in the ComboBox, ListBox and/or CheckListBox read-only controls. It seems to me that this combination sometimes gives the end-users more business meaning about the select data than a single Name only. I guess they may start understanding the purpose of the ID value as a primary key of a table in business applications development. Let's look at the code (Listing 7) of the Postal Code button in Figure 1. This code is calling the LoopUpGetIDString() function (Listing 8) to retrieve the value of the postal code by the selected state. For example, for the 'Alaska' state we got 'AK' postal code. If an error does not occur the postal code is shown to the user by using the Show() method of the MessageBox class object.

```
Private Sub PostalCodeButton1_Click(ByVal sender As System.Object, _  
                                   ByVal e As System.EventArgs)  
    Handles PostalCodeButton1.Click  
  
    Cursor = Cursors.WaitCursor  
    Dim PostalCodeString As String  
    Using LoopUpObject As New LoopUpClass()  
        PostalCodeString = LoopUpObject.LoopUpGetIDString(StatesComboBox, _  
                                                         mErrorMsgString)  
  
        If Not IsNothing(mErrorMsgString) Then  
            Cursor = Cursors.Default  
            MessageBox.Show(mErrorMsgString, _  
                            Me.Text, _  
                            MessageBoxButtons.OK, _  
                            MessageBoxIcon.Error)  
        Else  
            Cursor = Cursors.Default  
            MessageBox.Show("Postal Code: " & PostalCodeString, _  
                            Me.Text, _  
                            MessageBoxButtons.OK, _  
                            MessageBoxIcon.Information)  
        End If  
    End Using  
    Cursor = Cursors.Default  
End Sub
```

Listing 7: Click event of the Postal Code button in Figure 1

The LoopUpGetIDString() and LoopUpGetIDInt32() functions (Listing 8) return the ID value for a character and a numeric data type respectively. The LoopUpGetNameString() function (Listing 8) returns the Name as a character data type only. These functions use the SelectedIndex() property to position the item number and the CType() explicit conversion function converts the selected ComboBox item object into ListItemClass class (Listing 3). Using the read-only properties FieldIDString, FieldIDInt32 and FieldName (Listing 3) the ID and the Name values can be retrieved.

```
Public Function LoopUpGetIDString(ByVal pComboBox As ComboBox, _
```

```

        ByRef pErrorMsgString As String) As String
Dim IndexNumberInt32 As Int32, GetIDString As String
Try
    IndexNumberInt32 = pComboBox.SelectedIndex()
    GetIDString = CType(pComboBox.Items(IndexNumberInt32), ListItemClass).FieldIDString
    Return (GetIDString)
Catch exErr As Exception
    Return (Nothing)
    pErrorMsgString = exErr.Message
End Try
End Function

Public Function LoopUpGetIDInt32(ByVal pComboBox As ComboBox, _
        ByRef pErrorMsgString As String) As Int32
Dim IndexNumberInt32 As Int32, GetIDInt32 As Int32
Try
    IndexNumberInt32 = pComboBox.SelectedIndex()
    GetIDInt32 = CType(pComboBox.Items(IndexNumberInt32), ListItemClass).FieldIDInt32
    Return (GetIDInt32)
Catch exErr As Exception
    pErrorMsgString = exErr.Message
    Return (Nothing)
End Try
End Function

Public Function LoopUpGetNameString(ByVal pComboBox As ComboBox, _
        ByRef pErrorMsgString As String) As String
Dim IndexNumberInt32 As Int32, GetNameString As String
Try
    IndexNumberInt32 = pComboBox.SelectedIndex()
    GetNameString = CType(pComboBox.Items(IndexNumberInt32), ListItemClass).FieldName
    Return (GetNameString)
Catch exErr As Exception
    pErrorMsgString = exErr.Message
    Return (Nothing)
End Try
End Function

```

Listing 8: Lookup ID functions for a character and a numeric data type

Finding ID and Name Values

As I explained above, the ListItemClass generic class (Listing 3) was created to store and read-only the values of the ID and Name columns from the lookup tables. In business applications with lookup tables in the database server finding the ID and Name values is a must. To find a string Name item, for example, in a ComboBox, the FindString() method is the easiest way to do so. With two lines of code (Listing 9) the string Name can be found by returning the Index number from the FindString() method and the Index position by using the SelectedIndex() property.

```

Public Sub LoopUpFindNameString(ByVal pComboBox As ComboBox, _
        ByVal pFindNameString As String, _
        ByRef pErrorMsgString As String)
Dim IndexNumberInt32 As Int32
Try
    IndexNumberInt32 = pComboBox.FindString(pFindNameString)
    pComboBox.SelectedIndex = IndexNumberInt32
Catch exErr As Exception
    pErrorMsgString = exErr.Message

```

```
End Try
End Sub
```

Listing 9: Lookup find string Name subroutine

Finding the ID value in a ComboBox requires more codes. I did not find any ID method implemented in the ComboBox control. In this particular case we need to loop through the entire Item collection and compare two string values as shown in Listing 10. The Compare() method of the String class object compares two strings and returns a Int32 zero value if both strings are equal. If the strings are equal the SelectedIndex() property position the Index in the ComboBox as I explained before. If the strings are not equal, the ComboBox does not show any data by setting SelectedIndex() equal -1.

```
Public Sub LoopUpFindIDString(ByVal pComboBox As ComboBox, _
                             ByVal pFindIDString As String, _
                             ByRef pErrorMsgString As String)
    Dim LoopInt32, CompareInt32 As Int32
    Dim GetIDString As String
    Dim IsFoundBoolean As Boolean = False
    Try
        For LoopInt32 = 0 To pComboBox.Items.Count - 1
            GetIDString = CType(pComboBox.Items(LoopInt32), ListItemClass).FieldIDString
            CompareInt32 = String.Compare(GetIDString, pFindIDString)
            If CompareInt32 = 0 Then
                pComboBox.SelectedIndex() = LoopInt32
                IsFoundBoolean = True
                Exit For
            End If
        Next LoopInt32
        If Not IsFoundBoolean Then
            pComboBox.SelectedIndex() = -1
        End If
    Catch exErr As Exception
        pErrorMsgString = exErr.Message
    End Try
End Sub
```

Listing 10: Lookup find string ID subroutine

Finding the State and Capital Names by Postal Code

When the user selects an item in the ComboBox the ID is determined by using any of the functions in Listing 8 as we discussed before (LoopUpGetIDString() and LoopUpGetIDInt32()). Often, based on the selected ID, we need to find and load with data another control from different table. In my solution project example (Figure 1), when the user select a State the Capital TextBox control is loaded data. As you can see for 'Alaska' state the capital is 'Juneau', and for the capital 'Agaña' the state is 'Guam'. In both case the Postal Code represents the selected ID as the primary key in `state` table (Listing 6). How to implement this approach in Windows environment? One of the easiest and fastest ways in MySQL 5.0/VB.NET 2005 is by using the SelectedIndexChanged() event of the ComboBox and MySQL stored procedures. I remember in the old days of VB 6.0 we used to use the Click() event and dynamic SQL statements with MDAC technology. Listing 11 and 12 shows the SelectedIndexChanged() events of the Sates and Capitals ComboBoxes. As you can see, in both events, the Postal Code ID (PostalCodeString) is determined first by using the LoopUpGetIDString() function.

```
Private Sub StatesComboBox_SelectedIndexChanged(ByVal sender As System.Object, _
                                             ByVal e As System.EventArgs) _
    Handles StatesComboBox.SelectedIndexChanged
    Dim PostalCodeString As String
    Dim CapitalNameString As String = Nothing
```

```
Using LoopUpObject As New LoopUpClass(mMySQLConnectionString)
    PostalCodeString = LoopUpObject.LoopUpGetIDString(StatesComboBox, _
                                                mErrorMsgString)

    Call LoopUpObject.LoopUpGetCapitalName(PostalCodeString, _
                                                "usp_states_capital_by_postal", _
                                                CapitalNameString, _
                                                mErrorMsgString)

    CapitalTextBox.Text = CapitalNameString
End Using
End Sub
```

Listing 11: SelectedIndexChanged() event of the States ComboBox

```
Private Sub CapitalsComboBox_SelectedIndexChanged(ByVal sender As System.Object, _
                                                ByVal e As System.EventArgs) _
    Handles CapitalsComboBox.SelectedIndexChanged

    Dim PostalCodeString As String
    Dim StateNameString As String = Nothing
    Using LoopUpObject As New LoopUpClass(mMySQLConnectionString)
        PostalCodeString = LoopUpObject.LoopUpGetIDString(CapitalsComboBox, _
                                                        mErrorMsgString)

        Call LoopUpObject.LoopUpGetStateName(PostalCodeString, _
                                                "usp_states_name_by_postal", _
                                                StateNameString, _
                                                mErrorMsgString)

        StateTextBox.Text = StateNameString
    End Using
End Sub
```

Listing 12: SelectedIndexChanged() event of the Capitals ComboBox

The user stored procedures shown in Listing 13 ('usp_states_capital_by_postal' and 'usp_states_name_by_postal') have Postal Code as input parameter (par_postalcode) and Capital and State (par_capital and par_statename) as output parameters respectively. Both procedures were developed and tested using [Toad for MySQL 2.0.3](#) freeware version from [Quest Software, Inc.](#)

```
DROP PROCEDURE IF EXISTS `usp_states_capital_by_postal`;
CREATE PROCEDURE `usp_states_capital_by_postal` (
    IN par_postalcode CHAR(2),
    OUT par_capital VARCHAR(20)
)
BEGIN
    SELECT `states`.`capital` INTO par_capital
    FROM `states`
    WHERE `postal` = par_postalcode;
END;

DROP PROCEDURE IF EXISTS `usp_states_name_by_postal`;
CREATE PROCEDURE `usp_states_name_by_postal` (
    IN par_postalcode CHAR(2),
    OUT par_statename VARCHAR(20)
)
BEGIN
    SELECT `states`.`statename` INTO par_statename
    FROM `states`
    WHERE `postal` = par_postalcode;
END;
```

Listing 13: User stored procedures to retrieve State and Capital name by Postal Code

The subroutines LoopUpGetCapitalName() and LoopUpGetStateName() are shown in Listing 14 and 15 respectively.

```
Public Sub LoopUpGetCapitalName(ByVal pPostalCodeString As String, _
                               ByVal pStoredProcedureString As String, _
                               ByRef pCapitalNameString As String, _
                               ByRef pErrorMsgString As String)
    Dim MySqlParameterPostal As New MySqlParameter
    Dim MySqlParameterCapital As New MySqlParameter
    Try
        Using mMySqlConnection As New MySqlConnection(mMySQLConnectionString)
            mMySqlConnection.Open()
            Using mMySqlCommand As New MySqlCommand
                With mMySqlCommand
                    .Connection = mMySqlConnection
                    .CommandType = CommandType.StoredProcedure
                    .CommandText = pStoredProcedureString
                    With MySqlParameterPostal
                        .ParameterName = "?par_postalcode"
                        .Direction = ParameterDirection.Input
                        .MySqlDbType = MySqlDbType.VarChar
                        .Size = 2
                        .Value = pPostalCodeString
                    End With
                    .Parameters.Add(MySqlParameterPostal)
                    With MySqlParameterCapital
                        .ParameterName = "?par_capital"
                        .Direction = ParameterDirection.Output
                        .MySqlDbType = MySqlDbType.VarChar
                        .Size = 20
                        .Value = pCapitalNameString
                    End With
                    .Parameters.Add(MySqlParameterCapital)
                    .ExecuteNonQuery()
                    mObjectjValue = .Parameters("?par_capital").Value
                    If Not IsDBNull(mObjectjValue) Then
                        pCapitalNameString = mObjectjValue.ToString
                    Else
                        pCapitalNameString = String.Empty
                    End If
                End With
            End Using
        End Using
    Catch exErr As Exception
        pErrorMsgString = exErr.Message
    Finally
        If Not IsNothing(MySqlParameterPostal) Then
            MySqlParameterPostal = Nothing
        End If
        If Not IsNothing(MySqlParameterCapital) Then
            MySqlParameterCapital = Nothing
        End If
    End Try
End Sub
```

Figure 14: Retrieve Capital name subroutine by Postal Code

```
Public Sub LoopUpGetStateName(ByVal pPostalCodeString As String, _
                             ByVal pStoredProcedureString As String, _
                             ByRef pStateNameString As String, _
                             ByRef pErrorMsgString As String)
    Dim MySqlParameterPostal As New MySqlParameter
    Dim MySqlParameterState As New MySqlParameter
    Try
        Using mMySqlConnection As New MySqlConnection(mMySQLConnectionString)
            mMySqlConnection.Open()
            Using mMySqlCommand As New MySqlCommand
                With mMySqlCommand
                    .Connection = mMySqlConnection
                    .CommandType = CommandType.StoredProcedure
                    .CommandText = pStoredProcedureString
                    With MySqlParameterPostal
                        .ParameterName = "?par_postalcode"
                        .Direction = ParameterDirection.Input
                        .MySqlDbType = MySqlDbType.VarChar
                        .Size = 2
                        .Value = pPostalCodeString
                    End With
                    .Parameters.Add(MySqlParameterPostal)
                    With MySqlParameterState
                        .ParameterName = "?par_statename"
                        .Direction = ParameterDirection.Output
                        .MySqlDbType = MySqlDbType.VarChar
                        .Size = 20
                        .Value = pStateNameString
                    End With
                    .Parameters.Add(MySqlParameterState)
                    .ExecuteNonQuery()
                    mObjectjValue = .Parameters("?par_statename").Value
                    If Not IsDBNull(mObjectjValue) Then
                        pStateNameString = mObjectjValue.ToString
                    Else
                        pStateNameString = String.Empty
                    End If
                End With
            End Using
        End Using
    Catch exErr As Exception
        pErrorMsgString = exErr.Message
    Finally
        If Not IsNothing(MySqlParameterPostal) Then
            MySqlParameterPostal = Nothing
        End If
        If Not IsNothing(MySqlParameterState) Then
            MySqlParameterState = Nothing
        End If
    End Try
End Sub
```

Figure 15: Retrieve State name subroutine by Postal Code

Executing Stored Procedures in MySQL 5.0/VB.NET 2005

After the paper "[Define and Store MySQL ADO Connection String in VB.NET 2005](#)" was published online, I got many questions from many Open Source application developers around the world. One of the main questions

was how to execute a MySQL 5.0 stored procedure in VB.NET 2005 with input and output parameters. They were looking for a simple code implementation and a good explanation using the latest Connector/NET 5.0.6. In quality of example, let's look carefully at the subroutine LoopUpGetCapitalName() in Listing 14. Looking for a better explanation, I have decided to divide this code in five main blocks from Listing 14.1 to Listing 14.5. The first Listing 14.1 uses the Using statement to create the Connection object mMySQLConnection and initialize it. The variable mMySQLConnectionString represents the MySQL Connection String defined in the app.config file as Server=xxx;Database=xxx;Uid=xxx;Pwd=xxx. After the connection is open with the Open() method the Command object mMySQLCommand is created and initialized with the Using statement. Three main properties of the Command object needs to be determined: 1.Connection property sets to Connection object, 2. CommandType property sets to stored procedure type (CommandType.StoredProcedure) and 3. CommandText property sets to the stored procedure object name developed in the MySQL database server.

```
Using mMySQLConnection As New MySqlConnection(mMySQLConnectionString)
    mMySQLConnection.Open()
    Using mMySQLCommand As New MySqlCommand
        With mMySQLCommand
            .Connection = mMySQLConnection
            .CommandType = CommandType.StoredProcedure
            .CommandText = pStoredProcedureString
        End With
    End Using
End Using
```

Listing 14.1

Because the user stored procedure `usp_states_capital_by_postal` (Listing 13) has two parameters we need to create, initialize, set and add these parameters to the Command object parameters collection. Listing 14.2 and 14.3 shows the code for Postal Code (MySQLParameterPostal) and Capital Name (MySQLParameterCapital) MySQL parameters. As you can see, for the Connector/NET 5.0.6, the question sign (?) must be included before the parameter name. The Postal Code parameter direction is setup to Input and the Capital Name is setup to Output as required from the user stored procedure `usp_states_capital_by_postal` definition.

```
Dim MySQLParameterPostal As New MySQLParameter

With MySQLParameterPostal
    .ParameterName = "?par_postalcode"
    .Direction = ParameterDirection.Input
    .MySQLDbType = MySQLDbType.VarChar
    .Size = 2
    .Value = PostalCodeString
End With
.Parameters.Add(MySQLParameterPostal)
```

Listing 14.2

```
Dim MySQLParameterCapital As New MySQLParameter

With MySQLParameterCapital
    .ParameterName = "?par_capital"
    .Direction = ParameterDirection.Output
    .MySQLDbType = MySQLDbType.VarChar
    .Size = 20
    .Value = pCapitalNameString
End With
.Parameters.Add(MySQLParameterCapital)
```

Listing 14.3

After the Command object is executed with ExecuteNonQuery() method the output parameter value Capital Name can be retrieved as shown in Listing 14.4. The object variable mObjectjValue stores the Capital Name

and it needs to be checked for Null database value. To do that the IsDBNull() function was used. If the Capital Name value is not Null, it's gets stored in the pCapitalNameString function by reference parameter. On the other hand, if the value is Null the pCapitalNameString is set to empty string (String.Empty) using the String class object.

```
.ExecuteNonQuery()  
mObjectjValue = .Parameters("?par_capital").Value  
If Not IsDBNull(mObjectjValue) Then  
    pCapitalNameString = mObjectjValue.ToString  
Else  
    pCapitalNameString = String.Empty  
End If
```

Listing 14.4

After everything is done we need to release .NET used unmanaged resource as required. The Using statement takes care of releasing the resources for the mMySQLConnection Connection object and mMySQLCommand command object. The MySQL parameter objects MySQLParameterPostal and MySQLParameterCapital needs to be destroyed by setting than to Nothing in the Finally block (Listing 14.5). Setting to Nothing these objects will enable the Garbage Collection (GC) to release them.

```
Finally  
    If Not IsNothing(MySqlParameterPostal) Then  
        MySQLParameterPostal = Nothing  
    End If  
    If Not IsNothing(MySqlParameterCapital) Then  
        MySQLParameterCapital = Nothing  
    End If  
End Try
```

Listing 14.5

Conclusions

From this paper you can find the following main conclusions:

- Lookup table data loading process should be fast and well defined to maintain the ID and Name column values.
- The generic ListItemClass class provides a clear and well done development code to store and read-only the values of the ID and Name columns. This class should also be included in any VB.NET 2005 solution project for developing database business applications.
- The fastest way to retrieve data from MySQL 5.0 using VB.NET 2005 (Connector/NET 5.0.6) today is by using stored procedures/functions and MySQL data reader object (MySQLDataReader).

To download the source codes and a PDF format for this article go to [Visual WWW Downloads](#).

Visual WWW Legal Notice

This document contains freeware information. The information described in this document may be used or copied or transmitted for computer programming development purposes only without the written permission of Visual WWW, Inc. No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose.

The information contained in this document is subject to change without notice. Visual WWW makes no warranty of any kind with respect to this information. VISUAL WWW SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTY OF THE MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Visual WWW

shall not be liable for any direct, indirect, incidental, consequential, or other damage alleged in connection with the furnishing or use of this information.

Visual WWW, Inc. and Visual WWW logo are registered trademarks of Visual WWW, Inc. Other trademarks and registered trademarks used in this document are property of their respective owners.