

## Bound and Unbound DataGridView Control for MySQL 5.0/VB.NET 2008 Windows Applications

---

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



**Visual WWW, Inc.**

*'Created by Developers for Developers'*

[www.evisualwww.com](http://www.evisualwww.com) | [info@evisualwww.com](mailto:info@evisualwww.com)

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

## Table of Content

Table of Content.....	2
Introduction .....	2
Required Software .....	2
Bound and Unbound Programming Modes.....	2
Stopwatch - A Code Timer Class .....	2
Bound DataGridView Data Load.....	3
Unbound DataGridView Data Load.....	8
Execution Time Data Load Results.....	10
Conclusions.....	10
Visual WWW Legal Notice .....	10

## Introduction

Understanding and programming MySQL data load into the DataGridView control is a must for Windows Application Developers today. Many times, in real production environment the end-users need to see a result set of data to make daily business decisions. The main question is, how many records need to be seen when the control gets loaded? The answer to this question is defined by the client's business rules. At this point application performance is a critical issue for Application Developers. In this paper I'll show you how load MySQL data into the DataGridView control using VB.NET bound and unbound programming modes. A timer class named Stopwatch for VB.NET will be used to measure the execution time between these two programming modes.

## Required Software

- [MySQL Database Server 5.0.51b](#)
- [MySQL Connector/NET 5.2.1](#)
- [Toad for MySQL Freeware 3.1.1](#)
- [Microsoft Visual Basic 2008 Express Edition](#)

## Bound and Unbound Programming Modes

Bound and unbound programming modes are defined based on how the controls are linked to the database table. A control is called bound if some of its database properties are linked directly to a column(s) in a table. This control displays the contents of the column and when the user edits the data in the control, which overwrites the data in the table. An unbound control is not linked to a column in a table. Data load and update for this control is done by the code behind written by the Application Developers. Before Microsoft .NET framework appeared, Application Developers tried to develop Windows Client/Server applications using bound controls in Visual Basic 6.0. The data load and update using this programming mode was a very slow and unstable process. Because of that we, Application Developers, decided to develop Client/Server applications in unbound mode. Of course, the development time of these applications increased, but they did run fastest and more stable. It was, for us, the only choice at this time. With Visual Studio .NET (2002, 2003, 2005 and 2008) Microsoft continues to improve bound Windows controls. One of these controls is the DataGridView control. This control represents a data table view with standard structure of columns and rows. The DataGridView control can be used to load and edit the data at the same time. In this paper, it will be used to show data load techniques only. For some reasons I have not seen many Windows applications developed with editable DataGridView control. Before looking at the VB.NET programming code, let's look at the code timer class named Stopwatch.

## Stopwatch - A Code Timer Class

Stopwatch represents a high-resolution code timer class for VB.NET using Kernel32 API functions. The main objective of this class is for precisely measuring how long specific operations take to complete. Because one of

my topics in this paper is to compare application performance between bound and unbound modes, I have decided to use this class. The Kernel32 API function QueryPerformanceCounter() contains an excellent finer-grained tick counter. This timer boasts near microsecond (0.000001) resolution and is easily callable from any VB.NET Windows applications. Listing 1 shows how to use the Stopwatch object class (swatch) to measure how long a VB.NET code will run. As you can see the start() and stop() methods are applied to this class and the difference in time gives us the execution time to run the VB.NET code.

```
Private mTimeDouble As Double
Private swatch As New Stopwatch()
swatch.Reset()
swatch.Start()

'VB.NET code

swatch.Stop()
mTimeDouble = swatch.ElapsedMilliseconds * 0.001
```

Listing 1: Stopwatch object class (swatch) implementation in VB.NET code

Let's look at the loading process of MySQL data into the DataGirdView control using VB.NET bound and unbound programming modes.

### Bound DataGridView Data Load

To illustrate how to load MySQL data into the DataGridView control a simple VB.NET 2008 project was developed (Figure 1). As you can see, the form contains a DataGridView control with ten columns and two groups of bound and unbound data load buttons (Load 1 and Load 2). Both groups show the execution time in seconds (s) for 20,000 records of MySQL data load.

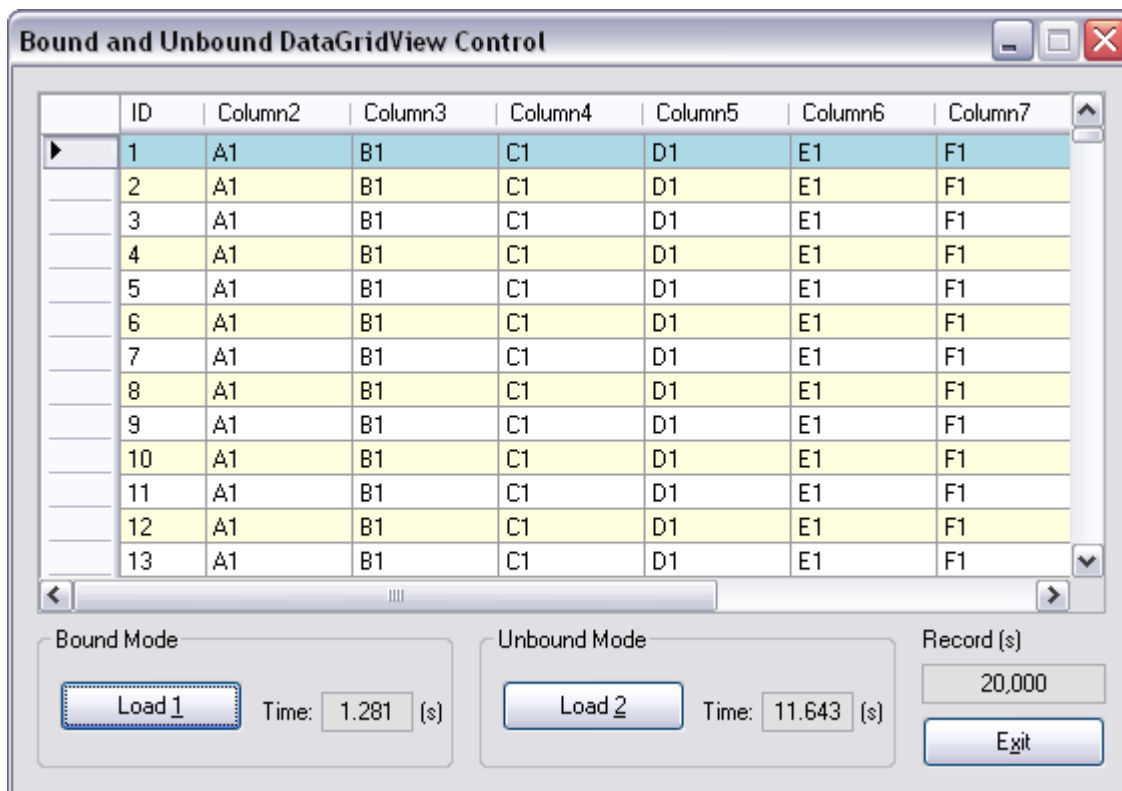


Figure 1: Bound and unbound DataGridView control project

Let's look at the code of the buttons Load 1 (bound mode) and Load 2 (unbound mode). Listing 2 shows the click event code of the bound Load 1 button. The Using statement creates and destroys the instance of a class UnboundClass (Listing 3). The ADO.NET Connection String mMySQLConnectionString has been passed as a constructor to the class. Just a reminder that this Connection String value is defined and stored in the app.config file as I explained in the article "[Define and Store MySQL ADO Connection String in VB.NET 2005](#)". The procedure BoundDataLoading() (Listing 4) is called to load the data. The DataGridView UnboundDataGridView and the TextBox RecordCountTextBox are passed by reference. These procedures is showing below.

```
Private Sub BoundLoadButton_Click(ByVal sender As System.Object, _
                                   ByVal e As System.EventArgs)
    Handles BoundLoadButton.Click

    swatch.Reset()
    swatch.Start()
    Cursor = Cursors.WaitCursor
    Try
        Using BoundObject As New UnboundClass(mMySQLConnectionString)
            Call BoundObject.BoundDataLoading(UnboundDataGridView, _
                                             RecordCountTextBox, _
                                             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, _
                       Me.Text, _
                       MessageBoxButtons.OK, _
                       MessageBoxIcon.Error)
    End Try
    Cursor = Cursors.Default
    swatch.Stop()
    mTimeDouble = swatch.ElapsedMilliseconds * 0.001
    BoundTimeTextBox.Text = mTimeDouble.ToString
End Sub
```

Listing 2: Click event of the bound Load 1 button

```
Imports MySql.Data.MySqlClient
Namespace UnboundLibrary
    Public Class UnboundClass
        Inherits ObjectDisposeClass
        Private mDataView As DataView
        Private mObjectjValue As Object
        Private mMySQLConnectionString As String
        Public Sub New(ByVal pMySQLConnectionString As String)
            mMySQLConnectionString = pMySQLConnectionString
        End Sub
    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 3: Class UnboundClass with IDisposable implementation

The procedure BoundDataLoading() shown in Listing 4 load the data into the DataGridView UnboundDataGridView with the DataView object mDataView. This object contains the data defined in the user stored procedure `usp\_test\_select\_all` (Listing 5). This stored procedure selects all columns and data from table `test` (Listing 6), and was developed and tested using [Toad for MySQL 3.1.1](#) freeware version from [Quest Software, Inc.](#) The table `test` gets loaded with data from the Excel spreadsheet by using the Import Wizard.

The Using statement was applied four times for the following ADO.NET objects: MySqlConnection, MySqlCommand, MySqlDataAdapter and DataSet. This statement allows automatically creating and destroying these unmanaged resources without declaring and initializing these objects. Let's find out how the DataGridView UnboundDataGridView gets formatted and loaded with data.

```
Public Sub BoundDataLoading(ByRef pDataGridViewControl As DataGridView, _
                            ByRef pTextBoxRecordCount As TextBox, _
                            ByRef pErrorMessageString As String)
    Dim RecordCountInt32 As Int32
    Try
        Using mMySqlConnection As New MySqlConnection(mMySQLConnectionString)
            mMySqlConnection.Open()
            Using mMySqlCommand As New MySqlCommand
                With mMySqlCommand
                    .Connection = mMySqlConnection
                    .CommandType = CommandType.StoredProcedure
                    .CommandText = "usp_test_select_all"
                End With
                Using mMySqlDataAdapter As New MySqlDataAdapter(mMySqlCommand)
                    Using mDataSet As New DataSet
                        mDataSet.Clear()
                        mMySqlDataAdapter.Fill(mDataSet, "test")
                        mDataView = New DataView
                        mDataView.Table = mDataSet.Tables("test")
                    End Using
                End Using
            End Using
        End Using
        RecordCountInt32 = mDataView.Count
        pTextBoxRecordCount.Text = FormatNumber(RecordCountInt32.ToString, 0)
        Call DataGridViewBound(pDataGridViewControl)
        pDataGridViewControl.DataSource = mDataView
    Catch exError As Exception
        pErrorMessageString = exError.Message
    End Try
End Sub
```

```
End Try  
End Sub
```

Listing 4: Procedure BoundDataLoading() for data loading into the DataGridView UnboundDataGridView

```
DROP PROCEDURE IF EXISTS `vwww`.`usp_test_select_all`;  
CREATE PROCEDURE `usp_test_select_all`()  
BEGIN  
    SELECT `id`, `column2`, `column3`, `column4`, `column5`,  
           `column6`, `column7`, `column8`, `column9`, `column10`  
    FROM `test`  
    ORDER BY `id`;  
END;
```

Listing 5: User stored procedure `usp\_test\_select\_all` to select all columns and data from table `test`

```
DROP TABLE IF EXISTS `vwww`.`test`;  
CREATE TABLE `test` (  
    `id` int(20) NOT NULL auto_increment,  
    `column2` varchar(50) NOT NULL,  
    `column3` varchar(50) NOT NULL,  
    `column4` varchar(50) NOT NULL,  
    `column5` varchar(50) NOT NULL,  
    `column6` varchar(50) NOT NULL,  
    `column7` varchar(50) NOT NULL,  
    `column8` varchar(50) NOT NULL,  
    `column9` varchar(50) NOT NULL,  
    `column10` varchar(50) NOT NULL,  
    PRIMARY KEY (`id`),  
    KEY `ix1_test` (`column2`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Listing 6: Table `test` definition

The procedure DataGridViewBound() formats the DataGridView UnboundDataGridView as shown in Listing 7. First the grid is formatted itself and then column by column. This standard code is used to format the DataGridView control before it gets loaded with data.

```
Private Sub DataGridViewBound(ByRef pDataGridViewControl As DataGridView)  
    With pDataGridViewControl  
        .AlternatingRowsDefaultCellStyle.BackColor = Color.LightYellow  
        .DefaultCellStyle.SelectionBackColor = Color.LightBlue  
        .SelectionMode = DataGridViewSelectionMode.FullRowSelect  
        .RowTemplate.Height = 17  
        .AllowUserToOrderColumns = True  
        .AllowUserToDeleteRows = False  
        .AllowUserToAddRows = False  
        .ReadOnly = True  
        .MultiSelect = False  
        .Columns.Clear()  
  
        Dim ColumnID As New DataGridViewTextBoxColumn  
        With ColumnID  
            .DataPropertyName = "id"  
            .Name = "ID"  
            .HeaderText = "ID"  
            .AutoSizeMode = DataGridViewAutoSizeColumnMode.ColumnHeader  
            .DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleLeft  
        End With  
        .Columns.Add(ColumnID)
```

```
Dim Column2 As New DataGridViewTextBoxColumn  
With Column2  
    .DataPropertyName = "Column2"  
    .Name = "Column2"  
    .HeaderText = "Column2"  
    .AutoSizeMode = DataGridViewAutoSizeColumnMode.ColumnHeader  
    .DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleLeft  
End With  
.Columns.Add(Column2)
```

```
Dim Column3 As New DataGridViewTextBoxColumn  
With Column3  
    .DataPropertyName = "Column3"  
    .Name = "Column3"  
    .HeaderText = "Column3"  
    .AutoSizeMode = DataGridViewAutoSizeColumnMode.ColumnHeader  
    .DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleLeft  
End With  
.Columns.Add(Column3)
```

```
Dim Column4 As New DataGridViewTextBoxColumn  
With Column4  
    .DataPropertyName = "Column4"  
    .Name = "Column4"  
    .HeaderText = "Column4"  
    .AutoSizeMode = DataGridViewAutoSizeColumnMode.ColumnHeader  
    .DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleLeft  
End With  
.Columns.Add(Column4)
```

```
Dim Column5 As New DataGridViewTextBoxColumn  
With Column5  
    .DataPropertyName = "Column5"  
    .Name = "Column5"  
    .HeaderText = "Column5"  
    .AutoSizeMode = DataGridViewAutoSizeColumnMode.ColumnHeader  
    .DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleLeft  
End With  
.Columns.Add(Column5)
```

```
Dim Column6 As New DataGridViewTextBoxColumn  
With Column6  
    .DataPropertyName = "Column6"  
    .Name = "Column6"  
    .HeaderText = "Column6"  
    .AutoSizeMode = DataGridViewAutoSizeColumnMode.ColumnHeader  
    .DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleLeft  
End With  
.Columns.Add(Column6)
```

```
Dim Column7 As New DataGridViewTextBoxColumn  
With Column7  
    .DataPropertyName = "Column7"  
    .Name = "Column7"  
    .HeaderText = "Column7"  
    .AutoSizeMode = DataGridViewAutoSizeColumnMode.ColumnHeader
```

```
        .DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleLeft
    End With
    .Columns.Add(Column7)

    Dim Column8 As New DataGridViewTextBoxColumn
    With Column8
        .DataPropertyName = "Column8"
        .Name = "Column8"
        .HeaderText = "Column8"
        .AutoSizeMode = DataGridViewAutoSizeColumnMode.ColumnHeader
        .DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleLeft
    End With
    .Columns.Add(Column8)

    Dim Column9 As New DataGridViewTextBoxColumn
    With Column9
        .DataPropertyName = "Column9"
        .Name = "Column9"
        .HeaderText = "Column9"
        .AutoSizeMode = DataGridViewAutoSizeColumnMode.ColumnHeader
        .DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleLeft
    End With
    .Columns.Add(Column9)

    Dim Column10 As New DataGridViewTextBoxColumn
    With Column10
        .DataPropertyName = "Column10"
        .Name = "Column10"
        .HeaderText = "Column10"
        .AutoSizeMode = DataGridViewAutoSizeColumnMode.ColumnHeader
        .DefaultCellStyle.Alignment = DataGridViewContentAlignment.MiddleLeft
    End With
    .Columns.Add(Column10)
End With
End Sub
```

Listing 7: Procedure DataGridViewBound() for formatting the DataGridView UnboundDataGridView in unbound mode

## Unbound DataGridView Data Load

The unbound programming mode of developing business database applications has been the most popular for Windows Application Developers. In .NET Framework, as I explained before, Microsoft tremendously improved the Windows and Internet bound controls. The main idea behind this effort was to decrease development time and increase application stability and flexibility. We already know from previous years that using unbound application development in VB 6.0 increases performance and stability. Here is the real question: Can we get the same results by using bound controls in VB.NET 2008? Unfortunately I don't have enough development references from developers and friends at this point. I can show you that the bound DataGridView is faster than the unbound. Let's look at the code and execution time. Listing 8 shows the click event code of the unbound Load 2 button. This code is similar to the code of the bound Load 1 button (Listing 2). The only difference is that the procedure UnboundDataLoading() (Listing 9) gets called at this time.

```
Private Sub UnboundLoadButton_Click(ByVal sender As System.Object,
                                     ByVal e As System.EventArgs)
    Handles UnboundLoadButton.Click
    swatch.Reset()
    swatch.Start()
    Cursor = Cursors.WaitCursor
```

```
Try
    Using UnboundObject As New UnboundClass(mMySQLConnectionString)
        Call UnboundObject.UnboundDataLoading(UnboundDataGridView, _
                                                RecordCountTextBox, _
                                                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, _
                    Me.Text, _
                    MessageBoxButtons.OK, _
                    MessageBoxIcon.Error)

End Try
Cursor = Cursors.Default
swatch.Stop()
mTimeDouble = swatch.ElapsedMilliseconds * 0.001
UnboundTimeTextBox.Text = mTimeDouble.ToString
End Sub
```

Listing 8: Click event of the bound Load 2 button

Listing 9 shows the procedure UnboundDataLoading() for loading the data into the DataGridView UnboundDataGridView by using the same user stored procedure `usp\_test\_select\_all` (Listing 5). In this case the ADO.NET mMySqlDataReader object was created to loop through it and get data loaded into the DataGridView row by row. First of all, in unbound mode, to format the DataGridView (Listing 7) the property DataPropertyName is not required at all. So the procedure DataGridViewUnbound() is the same as DataGridViewBound() without setting the property DataPropertyName. The one-dimensional array CellsArrayObject() is used with Object data type to locally store any data from the MySQL data reader mMySqlDataReader by using the GetValues() method. The entire row is added to the DataGridView control with the Add method of the property Rows. The variable RecordCountInt32 counts how many rows have been added to the DataGridView control.

```
Public Sub UnboundDataLoading(ByRef pDataGridViewControl As DataGridView, _
                              ByRef pTextBoxRecordCount As TextBox, _
                              ByRef pErrorMessageString As String)
    Dim CellsArrayObject() As Object
    Dim RecordCountInt32 As Int32
    Try
        Using mMySQLConnection As New MySqlConnection(mMySQLConnectionString)
            mMySQLConnection.Open()
            Using mMySQLCommand As New MySqlCommand
                With mMySQLCommand
                    .Connection = mMySQLConnection
                    .CommandType = CommandType.StoredProcedure
                    .CommandText = "usp_test_select_all"
                End With
                Using mMySqlDataReader As MySqlDataReader = _
                    mMySQLCommand.ExecuteReader(CommandBehavior.SingleResult)
                    With mMySqlDataReader
                        If .HasRows Then
                            Call DataGridViewUnbound(pDataGridViewControl)
                        End If
                    End With
                End Using
            End Using
        End Using
    Catch
    End Try
End Sub
```

```
ReDim CellsArrayObject(.FieldCount - 1)
While .Read
    .GetValues(CellsArrayObject)
    pDataGridViewControl.Rows.Add(CellsArrayObject)
    RecordCountInt32 += 1
End While
pTextBoxRecordCount.Text = FormatNumber(RecordCountInt32.ToString, 0)
End If
End With
End Using
End Using
End Using
Catch exError As Exception
    pErrorMessageString = exError.Message
End Try
End Sub
```

Listing 9: Procedure UnboundDataLoading() for data loading into the DataGridView UnboundDataGridView

### Execution Time Data Load Results

Table 1 shows the execution time data load results for bound and unbound DataGridView control. The shown values are measured in seconds using the code timer class Stopwatch. As you can see the bound mode offers a much better performance than the unbound for different amounts of records.

Modes / Records	1,000	3,000	5,000	10,000	20,000
Bound	0.116	0.209	0.317	0.622	1.281
Unbound	0.708	1.796	2.946	5.894	11.643

Table 1: Timing data load results in seconds

### Conclusions

Based on the execution time data load results provided in this paper for bound and unbound DataGridView control, a simple conclusion can be made: Bound DataGridView control implementation provides a good performance in MySQL/VB.NET Windows Applications development. Unbound DataGridView control is not necessary to be implemented.

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.