.NET Framework 4

**Multiple Active Result Sets (MARS)**

Multiple Active Result Sets (MARS) is a feature that allows the execution of multiple batches on a single connection. In previous versions, only one batch could be executed at a time against a single connection. Executing multiple batches with MARS does not imply simultaneous execution of operations.

http://i.msdn.microsoft.com/Global/Images/clear.gif In This Section

[Enabling Multiple Active Result Sets (MARS)](http://msdn.microsoft.com/en-us/library/h32h3abf.aspx)

Discusses how to use MARS with SQL Server 2005.

[Manipulating Data (MARS)](http://msdn.microsoft.com/en-us/library/yf1a7f4f.aspx)

Provides examples of coding MARS applications.

.NET Framework 4

**Enabling Multiple Active Result Sets (MARS)**

Multiple Active Result Sets (MARS) is a feature that works with SQL Server 2005 to allow the execution of multiple batches on a single connection. To access multiple result sets on previous versions of SQL Server using [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx) objects, a separate [SqlConnection](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnection.aspx) object must be used with each [SqlCommand](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.aspx) object. However, when MARS is enabled for use with SQL Server 2005, each command object used adds a session to the connection.

|  |
| --- |
| **NoteNote** |
| A single MARS session opens one logical connection for MARS to use and then one logical connection for each active command. |

http://i.msdn.microsoft.com/Global/Images/clear.gif Enabling and Disabling MARS in the Connection String

|  |
| --- |
| **NoteNote** |
| The following connection strings use the sample **AdventureWorks** database included with SQL Server 2005. The connection strings provided assume that the database is installed on a server named MSSQL1. Modify the connection string as necessary for your environment. |

The MARS feature is disabled by default. It can be enabled by adding the "MultipleActiveResultSets=True" keyword pair to your connection string. "True" is the only valid value for enabling MARS. The following example demonstrates how to connect to an instance of SQL Server and how to specify that MARS should be enabled.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_ctl00_ctl01_code');" \o "Copy Code)

Dim connectionString As String = "Data Source=MSSQL1;" & \_

"Initial Catalog=AdventureWorks;Integrated Security=SSPI" & \_

"MultipleActiveResultSets=True"

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_ctl00_ctl02_code');" \o "Copy Code)

string connectionString = "Data Source=MSSQL1;" +

"Initial Catalog=AdventureWorks;Integrated Security=SSPI" +

"MultipleActiveResultSets=True";

You can disable MARS by adding the "MultipleActiveResultSets=False" keyword pair to your connection string. "False" is the only valid value for disabling MARS. The following connection string demonstrates how to disable MARS.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_ctl00_ctl03_code');" \o "Copy Code)

Dim connectionString As String = "Data Source=MSSQL1;" & \_

"Initial Catalog=AdventureWorks;Integrated Security=SSPI" & \_

"MultipleActiveResultSets=False"

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_ctl00_ctl04_code');" \o "Copy Code)

string connectionString = "Data Source=MSSQL1;" +

"Initial Catalog=AdventureWorks;Integrated Security=SSPI" +

"MultipleActiveResultSets=False";

http://i.msdn.microsoft.com/Global/Images/clear.gif Special Considerations When Using MARS

In general, existing applications should not need modification to use a MARS-enabled connection. However, if you wish to use MARS features in your applications, you should understand the following special considerations.

**Statement Interleaving**

MARS operations execute synchronously on the server. Statement interleaving of SELECT and BULK INSERT statements is allowed. However, data manipulation language (DML) and data definition language (DDL) statements execute atomically. Any statements attempting to execute while an atomic batch is executing are blocked. Parallel execution at the server is not a MARS feature.

If two batches are submitted under a MARS connection, one of them containing a SELECT statement, the other containing a DML statement, the DML can begin execution within execution of the SELECT statement. However, the DML statement must run to completion before the SELECT statement can make progress. If both statements are running under the same transaction, any changes made by a DML statement after the SELECT statement has started execution are not visible to the read operation.

A WAITFOR statement inside a SELECT statement does not yield the transaction while it is waiting, that is, until the first row is produced. This implies that no other batches can execute within the same connection while a WAITFOR statement is waiting.

**MARS Session Cache**

When a connection is opened with MARS enabled, a logical session is created, which adds additional overhead. To minimize overhead and enhance performance, **SqlClient** caches the MARS session within a connection. The cache contains at most 10 MARS sessions. This value is not user adjustable. If the session limit is reached, a new session is created—an error is not generated. The cache and sessions contained in it are per-connection; they are not shared across connections. When a session is released, it is returned to the pool unless the pool's upper limit has been reached. If the cache pool is full, the session is closed. MARS sessions do not expire. They are only cleaned up when the connection object is disposed. The MARS session cache is not preloaded. It is loaded as the application requires more sessions.

**Thread Safety**

MARS operations are not thread-safe.

**Connection Pooling**

MARS-enabled connections are pooled like any other connection. If an application opens two connections, one with MARS enabled and one with MARS disabled, the two connections are in separate pools. For more information, see [SQL Server Connection Pooling (ADO.NET)](http://msdn.microsoft.com/en-us/library/8xx3tyca.aspx).

**SQL Server Batch Execution Environment**

When a connection is opened, a default environment is defined. This environment is then copied into a logical MARS session.

The batch execution environment includes the following components:

* Set options (for example, ANSI\_NULLS, DATE\_FORMAT, LANGUAGE, TEXTSIZE)
* Security context (user/application role)
* Database context (current database)
* Execution state variables (for example, @@ERROR, @@ROWCOUNT, @@FETCH\_STATUS @@IDENTITY)
* Top-level temporary tables

In SQL Server 2000 and earlier, all the batches that execute under the same connection share the same batch environment. Changes made to the batch environment by a batch are visible to all subsequent batches.

With MARS, a default execution environment is associated to a connection. Every new batch that starts executing under a given connection receives a copy of the default environment. Whenever code is executed under a given batch, all changes made to the environment are scoped to the specific batch. Once execution finishes, the execution settings are copied into the default environment. In the case of a single batch issuing several commands to be executed sequentially under the same transaction, semantics are the same as those exposed by connections involving earlier clients or servers.

**Parallel Execution**

MARS is not designed to remove all requirements for multiple connections in an application. If an application needs true parallel execution of commands against a server, multiple connections should be used.

For example, consider the following scenario. Two command objects are created, one for processing a result set and another for updating data; they share a common connection via MARS. In this scenario, the **Transaction**.**Commit** fails on the update until all the results have been read on the first command object, yielding the following exception:

Message: Transaction context in use by another session.

Source: .Net SqlClient Data Provider

Expected: (null)

Received: System.Data.SqlClient.SqlException

There are three options for handling this scenario:

1. Start the transaction after the reader is created, so that it is not part of the transaction. Every update then becomes its own transaction.
2. Commit all work after the reader is closed. This has the potential for a substantial batch of updates.
3. Don't use MARS; instead use a separate connection for each command object as you would have before MARS.

**Detecting MARS Support**

An application can check for MARS support by reading the **SqlConnection.ServerVersion** value. The major number should be 9 for SQL Server 2005.

http://i.msdn.microsoft.com/Global/Images/clear.gif

.NET Framework 4

**Manipulating Data (MARS)**

Before the introduction of Multiple Active Result Sets (MARS), developers had to use either multiple connections or server-side cursors to solve certain scenarios. In addition, when multiple connections were used in a transactional situation, bound connections (with **sp\_getbindtoken** and **sp\_bindsession**) were required. The following scenarios show how to use a MARS-enabled connection instead of multiple connections.

http://i.msdn.microsoft.com/Global/Images/clear.gif Using Multiple Commands with MARS

The following Console application demonstrates how to use two [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx) objects with two [SqlCommand](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.aspx) objects and a single [SqlConnection](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnection.aspx) object with MARS enabled.

**Example**

The example opens a single connection to the **AdventureWorks** database. Using a [SqlCommand](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.aspx) object, a [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx) is created. As the reader is used, a second [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx) is opened, using data from the first [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx) as input to the WHERE clause for the second reader.

|  |
| --- |
| **NoteNote** |
| The following example uses the sample **AdventureWorks** database included with SQL Server 2005. The connection string provided in the sample code assumes that the database is installed and available on the local computer. Modify the connection string as necessary for your environment. |

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl25_ctl00_ctl08_code');" \o "Copy Code)

Option Strict On

Option Explicit On

Imports System

Imports System.Data

Imports System.Data.SqlClient

Module Module1

Sub Main()

' By default, MARS is disabled when connecting

' to a MARS-enabled host such as SQL Server 2005.

' It must be enabled in the connection string.

Dim connectionString As String = GetConnectionString()

Dim vendorID As Integer

Dim vendorCmd As SqlCommand

Dim productCmd As SqlCommand

Dim productReader As SqlDataReader

Dim vendorSQL As String = & \_

"SELECT VendorId, Name FROM Purchasing.Vendor"

Dim productSQL As String = \_

"SELECT Production.Product.Name FROM Production.Product " & \_

"INNER JOIN Purchasing.ProductVendor " & \_

"ON Production.Product.ProductID = " & \_

"Purchasing.ProductVendor.ProductID " & \_

"WHERE Purchasing.ProductVendor.VendorID = @VendorId"

Using awConnection As New SqlConnection(connectionString)

vendorCmd = New SqlCommand(vendorSQL, awConnection)

productCmd = New SqlCommand(productSQL, awConnection)

productCmd.Parameters.Add("@VendorId", SqlDbType.Int)

awConnection.Open()

Using vendorReader As SqlDataReader = vendorCmd.ExecuteReader()

While vendorReader.Read()

Console.WriteLine(vendorReader("Name"))

vendorID = CInt(vendorReader("VendorId"))

productCmd.Parameters("@VendorId").Value = vendorID

' The following line of code requires

' a MARS-enabled connection.

productReader = productCmd.ExecuteReader()

Using productReader

While productReader.Read()

Console.WriteLine(" " & CStr(productReader("Name")))

End While

End Using

End While

End Using

End Using

Console.WriteLine("Press any key to continue")

Console.ReadLine()

End Sub

Function GetConnectionString() As String

' To avoid storing the connection string in your code,

' you can retrive it from a configuration file.

Return "Data Source=(local);Integrated Security=SSPI;" & \_

"Initial Catalog=AdventureWorks; MultipleActiveResultSets=True"

End Function

End Module

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl25_ctl00_ctl09_code');" \o "Copy Code)

using System;

using System.Data;

using System.Data.SqlClient;

class Class1

{

static void Main()

{

// By default, MARS is disabled when connecting

// to a MARS-enabled host such as SQL Server 2005.

// It must be enabled in the connection string.

string connectionString = GetConnectionString();

int vendorID;

SqlDataReader productReader = null;

string vendorSQL =

"SELECT VendorId, Name FROM Purchasing.Vendor";

string productSQL =

"SELECT Production.Product.Name FROM Production.Product " +

"INNER JOIN Purchasing.ProductVendor " +

"ON Production.Product.ProductID = " +

"Purchasing.ProductVendor.ProductID " +

"WHERE Purchasing.ProductVendor.VendorID = @VendorId";

using (SqlConnection awConnection =

new SqlConnection(connectionString))

{

SqlCommand vendorCmd = new SqlCommand(vendorSQL, awConnection);

SqlCommand productCmd =

new SqlCommand(productSQL, awConnection);

productCmd.Parameters.Add("@VendorId", SqlDbType.Int);

awConnection.Open();

using (SqlDataReader vendorReader = vendorCmd.ExecuteReader())

{

while (vendorReader.Read())

{

Console.WriteLine(vendorReader["Name"]);

vendorID = (int)vendorReader["VendorId"];

productCmd.Parameters["@VendorId"].Value = vendorID;

// The following line of code requires

// a MARS-enabled connection.

productReader = productCmd.ExecuteReader();

using (productReader)

{

while (productReader.Read())

{

Console.WriteLine(" " +

productReader["Name"].ToString());

}

}

}

}

Console.WriteLine("Press any key to continue");

Console.ReadLine();

}

}

private static string GetConnectionString()

{

// To avoid storing the connection string in your code,

// you can retrive it from a configuration file.

return "Data Source=(local);Integrated Security=SSPI;" +

"Initial Catalog=AdventureWorks;MultipleActiveResultSets=True";

}

}

http://i.msdn.microsoft.com/Global/Images/clear.gif Reading and Updating Data with MARS

MARS allows a connection to be used for both read operations and data manipulation language (DML) operations with more than one pending operation. This feature eliminates the need for an application to deal with connection-busy errors. In addition, MARS can replace the user of server-side cursors, which generally consume more resources. Finally, because multiple operations can operate on a single connection, they can share the same transaction context, eliminating the need to use **sp\_getbindtoken** and **sp\_bindsession** system stored procedures.

**Example**

The following Console application demonstrates how to use two [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx) objects with three [SqlCommand](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.aspx) objects and a single [SqlConnection](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnection.aspx) object with MARS enabled. The first command object retrieves a list of vendors whose credit rating is 5. The second command object uses the vendor ID provided from a [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx) to load the second [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx) with all of the products for the particular vendor. Each product record is visited by the second [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx). A calculation is performed to determine what the new **OnOrderQty** should be. The third command object is then used to update the **ProductVendor** table with the new value. This entire process takes place within a single transaction, which is rolled back at the end.

|  |
| --- |
| **NoteNote** |
| The following example uses the sample **AdventureWorks** database included with SQL Server 2005. The connection string provided in the sample code assumes that the database is installed and available on the local computer. Modify the connection string as necessary for your environment. |

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl26_ctl00_ctl07_code');" \o "Copy Code)

Option Strict On

Option Explicit On

Imports System

Imports System.Data

Imports System.Data.SqlClient

Module Module1

Sub Main()

' By default, MARS is disabled when connecting

' to a MARS-enabled host such as SQL Server 2005.

' It must be enabled in the connection string.

Dim connectionString As String = GetConnectionString()

Dim updateTx As SqlTransaction

Dim vendorCmd As SqlCommand

Dim prodVendCmd As SqlCommand

Dim updateCmd As SqlCommand

Dim prodVendReader As SqlDataReader

Dim vendorID As Integer

Dim productID As Integer

Dim minOrderQty As Integer

Dim maxOrderQty As Integer

Dim onOrderQty As Integer

Dim recordsUpdated As Integer

Dim totalRecordsUpdated As Integer

Dim vendorSQL As String = \_

"SELECT VendorID, Name FROM Purchasing.Vendor " & \_

"WHERE CreditRating = 5"

Dim prodVendSQL As String = \_

"SELECT ProductID, MaxOrderQty, MinOrderQty, OnOrderQty " & \_

"FROM Purchasing.ProductVendor " & \_

"WHERE VendorID = @VendorID"

Dim updateSQL As String = \_

"UPDATE Purchasing.ProductVendor " & \_

"SET OnOrderQty = @OrderQty " & \_

"WHERE ProductID = @ProductID AND VendorID = @VendorID"

Using awConnection As New SqlConnection(connectionString)

awConnection.Open()

updateTx = awConnection.BeginTransaction()

vendorCmd = New SqlCommand(vendorSQL, awConnection)

vendorCmd.Transaction = updateTx

prodVendCmd = New SqlCommand(prodVendSQL, awConnection)

prodVendCmd.Transaction = updateTx

prodVendCmd.Parameters.Add("@VendorId", SqlDbType.Int)

updateCmd = New SqlCommand(updateSQL, awConnection)

updateCmd.Transaction = updateTx

updateCmd.Parameters.Add("@OrderQty", SqlDbType.Int)

updateCmd.Parameters.Add("@ProductID", SqlDbType.Int)

updateCmd.Parameters.Add("@VendorID", SqlDbType.Int)

Using vendorReader As SqlDataReader = vendorCmd.ExecuteReader()

While vendorReader.Read()

Console.WriteLine(vendorReader("Name"))

vendorID = CInt(vendorReader("VendorID"))

prodVendCmd.Parameters("@VendorID").Value = vendorID

prodVendReader = prodVendCmd.ExecuteReader()

Using prodVendReader

While (prodVendReader.Read)

productID = CInt(prodVendReader("ProductID"))

If IsDBNull(prodVendReader("OnOrderQty")) Then

minOrderQty = CInt(prodVendReader("MinOrderQty"))

onOrderQty = minOrderQty

Else

maxOrderQty = CInt(prodVendReader("MaxOrderQty"))

onOrderQty = CInt(maxOrderQty / 2)

End If

updateCmd.Parameters("@OrderQty").Value = onOrderQty

updateCmd.Parameters("@ProductID").Value = productID

updateCmd.Parameters("@VendorID").Value = vendorID

recordsUpdated = updateCmd.ExecuteNonQuery()

totalRecordsUpdated += recordsUpdated

End While

End Using

End While

End Using

Console.WriteLine("Total Records Updated: " & \_

CStr(totalRecordsUpdated))

updateTx.Rollback()

Console.WriteLine("Transaction Rolled Back")

End Using

Console.WriteLine("Press any key to continue")

Console.ReadLine()

End Sub

Function GetConnectionString() As String

' To avoid storing the connection string in your code,

' you can retrive it from a configuration file.

Return "Data Source=(local);Integrated Security=SSPI;" & \_

"Initial Catalog=AdventureWorks;MultipleActiveResultSets=True"

End Function

End Module

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl26_ctl00_ctl08_code');" \o "Copy Code)

using System;

using System.Collections.Generic;

using System.Text;

using System.Data;

using System.Data.SqlClient;

class Program

{

static void Main()

{

// By default, MARS is disabled when connecting

// to a MARS-enabled host such as SQL Server 2005.

// It must be enabled in the connection string.

string connectionString = GetConnectionString();

SqlTransaction updateTx = null;

SqlCommand vendorCmd = null;

SqlCommand prodVendCmd = null;

SqlCommand updateCmd = null;

SqlDataReader prodVendReader = null;

int vendorID = 0;

int productID = 0;

int minOrderQty = 0;

int maxOrderQty = 0;

int onOrderQty = 0;

int recordsUpdated = 0;

int totalRecordsUpdated = 0;

string vendorSQL =

"SELECT VendorID, Name FROM Purchasing.Vendor " +

"WHERE CreditRating = 5";

string prodVendSQL =

"SELECT ProductID, MaxOrderQty, MinOrderQty, OnOrderQty " +

"FROM Purchasing.ProductVendor " +

"WHERE VendorID = @VendorID";

string updateSQL =

"UPDATE Purchasing.ProductVendor " +

"SET OnOrderQty = @OrderQty " +

"WHERE ProductID = @ProductID AND VendorID = @VendorID";

using (SqlConnection awConnection =

new SqlConnection(connectionString))

{

awConnection.Open();

updateTx = awConnection.BeginTransaction();

vendorCmd = new SqlCommand(vendorSQL, awConnection);

vendorCmd.Transaction = updateTx;

prodVendCmd = new SqlCommand(prodVendSQL, awConnection);

prodVendCmd.Transaction = updateTx;

prodVendCmd.Parameters.Add("@VendorId", SqlDbType.Int);

updateCmd = new SqlCommand(updateSQL, awConnection);

updateCmd.Transaction = updateTx;

updateCmd.Parameters.Add("@OrderQty", SqlDbType.Int);

updateCmd.Parameters.Add("@ProductID", SqlDbType.Int);

updateCmd.Parameters.Add("@VendorID", SqlDbType.Int);

using (SqlDataReader vendorReader = vendorCmd.ExecuteReader())

{

while (vendorReader.Read())

{

Console.WriteLine(vendorReader["Name"]);

vendorID = (int) vendorReader["VendorID"];

prodVendCmd.Parameters["@VendorID"].Value = vendorID;

prodVendReader = prodVendCmd.ExecuteReader();

using (prodVendReader)

{

while (prodVendReader.Read())

{

productID = (int) prodVendReader["ProductID"];

if (prodVendReader["OnOrderQty"] == DBNull.Value)

{

minOrderQty = (int) prodVendReader["MinOrderQty"];

onOrderQty = minOrderQty;

}

else

{

maxOrderQty = (int) prodVendReader["MaxOrderQty"];

onOrderQty = (int)(maxOrderQty / 2);

}

updateCmd.Parameters["@OrderQty"].Value = onOrderQty;

updateCmd.Parameters["@ProductID"].Value = productID;

updateCmd.Parameters["@VendorID"].Value = vendorID;

recordsUpdated = updateCmd.ExecuteNonQuery();

totalRecordsUpdated += recordsUpdated;

}

}

}

}

Console.WriteLine("Total Records Updated: " +

totalRecordsUpdated.ToString());

updateTx.Rollback();

Console.WriteLine("Transaction Rolled Back");

}

Console.WriteLine("Press any key to continue");

Console.ReadLine();

}

private static string GetConnectionString()

{

// To avoid storing the connection string in your code,

// you can retrive it from a configuration file.

return "Data Source=(local);Integrated Security=SSPI;" +

"Initial Catalog=AdventureWorks;" +

"MultipleActiveResultSets=True";

}

}