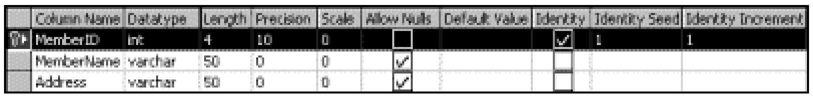
@@IDENTITY crisis

by Manohar Kamath
August 13, 1999

If you have worked with SQL Server, you are probably familiar with identity columns. These are equivalent to the "AutoNumber" columns in Access. The main purpose of these columns is to provide a primary key to the table, when a primary key can not be defined using other fields in the table. One of the common questions in the newsgroups is "How can I get the ID of the inserted record in SQL database?" This translates to "How can I get the value of identity column in the inserted record?" In this article, we will see how we can do this in several ways. Also, we will learn a little bit about the columns themselves.

About identity columns
These columns are like any other column except that their value is not inserted by the user, but by the system itself. The following diagram shows a sample table with an identity column



Column Name	Datatype	Length	Precision	Scale	Allow Nulls	Default Value	Identity	Identity Seed	Identity Increment
MemberID	int	4	10	0	☐		✓	1	1
MemberName	varchar	50	0	0	✓		☐		
Address	varchar	50	0	0	✓		☐		

Sample Table
(Click on the image to view the actual size image)

A few things you need to know about the identity columns

- They should be of datatype int, smallint, tinyint, decimal or numeric with scale 0.
- They can not contain null values
- They can not have any default values
- The identity increment is an integral value (1, -1, 5, etc.) and can not contain decimals. Also, it can not be 0.
- Identity Seed is 1 by default, and so is the Identity Increment. If you leave the seed field empty, it becomes 0.

Enter @@IDENTITY
When a record is inserted into a table with a identity column, the function @@IDENTITY (or the global variable @@IDENTITY as in SQL Server 6.5) returns the last identity value that was inserted in the database. I emphasize the word "last identity value" here because, this maybe different from the identity value of that particular table where the record was inserted.

Why, You may ask. When a record is inserted, any underlying triggers may modify other tables. If a trigger adds a record into another table, which happens to have an identity column, @@IDENTITY will now return this value instead. This is a point to note when retrieving the identity values. Also, if the trigger inserts into a table without any identity column, the @@IDENTITY will be null. For now, let us assume that the tables do not have any triggers and there is just one INSERT statement.

Another point to note is, the @@IDENTITY returns the last identity value on the same open connection to the database. By connection, I mean a physical connection. So, if you connect to the database, do an INSERT, disconnect and connect back again, the @@IDENTITY value will be NULL. On the other hand, if you INSERT into a table, retrieve @@IDENTITY, keep the connection open, get the @@IDENTITY again you will still get the same value as the first retrieval. In summary, you can treat @@IDENTITY as a connection specific global variable.

Let us discuss connections further. Here are a few possibilities of a "same connection" situations

- Within a SQL Server client app such as an ASP page, a VB application etc., the physical connection is held between the Open() and Close() methods of the ADO's Connection object. However, remember that within the connection is "lost" when the Connection object goes out of scope, even though the Close() method is not called explicitly.

  ```
  ...
  Dim loConn
  Set loConn = CreateObject("ADODB.Connection")
  loConn.Open("DSN=myDSN;UID=something;PWD=Something;")
  '==== a physical connection exists here ======
  loConn.Close()
  ....
  ```

- Within an MTS component, from the point the connection is opened and until the connection is closed explicitly or either SetAbort() or SetComplete is called or the method ends.

  Note: When you call an MTS component's method from an ASP page, and the method INSERTs into a table, you can not call another method within the same page to get the identity value. This is because, since MTS pools connections, although a connection maybe open, you will never know if you get the same connection back in the second method.

Now, to business
.. of retrieving the identity value from an INSERTed record. We will assume we are using ADO as our data access method. Of many ways of retrieving the @@IDENTITY, I will discuss three methods that I have tried out. Option 3 is my preferred way of doing it, the other two options are presented here to illustrate some concepts. Note that the code is in VBScript/VB.

Option 1: For all ADO versions
Essentially, we will first insert a record into the table using a SQL statement. Once the record is inserted, within the same connection context, we will issue another SQL statement to retrieve the identity value. The following code describes how we can do this in either ASP or Visual Basic.

```
Dim loConn, lsSQL, loRs
Set loConn = CreateObject("ADODB.Connection")
' Open a connection to the database
loConn.Open("DSN=myDSN;UID=something;PWD=Something;")

' Insert a new record into the table
lsSQL = "INSERT INTO tMembers (MemberName) VALUES ('Manohar')"

' Execute the SQL statement
loConn.Execute(lsSQL)

' Get the @@IDENTITY. The key statement is the one below
' llID will contain the new identity value inserted
lsSQL = "SELECT @@IDENTITY AS NewID"
Set loRs = loConn.Execute(lsSQL)
llID = loRs.Fields("NewID").value

' Close the connection
loConn.Close()
Set loConn = Nothing
```

Option 2: For ADO versions 2 and above
ADO 2 introduced a new method for the RecordSet object - NextRecordSet(). Using this method, you can execute have more than one SQL statement, and then use NextRecordSet() to get resultsets from individual SQL statements.

In option 1, we issued two SQL statements. We will combine this into one and issue it at the same time. The first resultset will contain the result from the INSERT (an empty recordset). The second will contain the identity value. The following code will explain:

```
Dim loConn, lsSQL, loRs
Set loConn = CreateObject("ADODB.Connection")

' Open a connection to the database
loConn.Open("DSN=myDSN;UID=something;PWD=Something;")

' Insert a new record into the table
lsSQL = "INSERT INTO tMembers (MemberName) VALUES ('Manohar');" &_
        "SELECT @@IDENTITY AS NewID;"

' Execute the SQL statement
Set loRs = loConn.Execute(lsSQL)

' Get the second resultset into a RecordSet object
Set loRs = loRs.NextRecordSet()

' Get the inserted ID
```

```
llID = loRs.Fields("NewID").value

' Close the connection
loConn.Close()
Set loConn = Nothing
```

Note: The above method is just to illustrate the NextRecordSet() function. The above can be done without using it, and in fact in a more efficient manner by suppressing the first resultset. The modified code now can be used by any version of ADO.

Option 3: PREFERRED - For all ADO versions

```
Dim loConn, lsSQL, loRs
Set loConn = CreateObject("ADODB.Connection")

' Open a connection to the database
loConn.Open("DSN=myDSN;UID=something;PWD=Something;")

' Insert a new record into the table
lsSQL = "SET NOCOUNT ON;" &_
        "INSERT INTO tMembers (MemberName) VALUES ('Manohar');" &_
        "SELECT @@IDENTITY AS NewID;"

' Execute the SQL statement
Set loRs = loConn.Execute(lsSQL)

' Get the inserted ID
llID = loRs.Fields("NewID").value

' Close the connection
loConn.Close()
Set loConn = Nothing
```

The SET NOCOUNT ON suppresses messages like "1 Row(s) affected" being sent back. These messages are sent back every time SQL statement is executed.

TIP: As a rule of thumb, the SET NOCOUNT ON suppresses all the resultsets from non-recordset returning statements like DELETE, INSERT and UPDATE. So, if you have more than one record to insert, do a SELECT @@IDENTITY AS NewID; after each INSERT and use the NextRecordSet() method to get the new IDs this inserted. The code is as shown below:

```
'Create SQL to insert more than one record.
'Select the identity value after each insert
lsSQL = "SET NOCOUNT ON;" &_
        "INSERT INTO tMembers (MemberName) VALUES ('John'); " &_
        "SELECT @@IDENTITY AS NewID;" &_
        "INSERT INTO tMembers (MemberName) VALUES ('Jane'); " &_
        "SELECT @@IDENTITY AS NewID;"

' Execute the SQL statement
Set loRs = loConn.Execute(lsSQL)

' Get the first inserted ID
llID1 = loRs.Fields("NewID").value

' Get the second inserted ID
Set loRs = loRs.NextRecordSet()
llID2 = loRs.Fields("NewID").value
```

IMPORTANT: All the above examples were tested using ASP and SQL Server 7. Please let us know if you had any problems when you used it with SQL 6.5. Thanks!