@@IDENTITY Is a system function that returns the last-inserted identity value

SCOPE\_IDENTITY

Returns the last identity value inserted into an identity column in the same scope. A scope is a module: a stored procedure, trigger, function, or batch. Therefore, if two statements are in the same stored procedure, function, or batch, they are in the same scope.

Use a stored procedure.  
a function cannot change data. No DML operations allowed, no dynamic SQL and a lot of other restrictions.

Primary Defenses:

* **Option 1: Use of Prepared Statements (with Parameterized Queries)**
* **Option 2: Use of Stored Procedures**
* **Option 3: White List Input Validation**
* **Option 4: Escaping All User Supplied Input**

Additional Defenses:

* **Also: Enforcing Least Privilege**
* **Also: Performing White List Input Validation as a Secondary Defense**

<https://searchsqlserver.techtarget.com/feature/Writing-efficient-queries>

**Clustered and Nonclustered Indexes Described**

An index is an on-disk structure associated with a table or view that speeds retrieval of rows from the table or view. An index contains keys built from one or more columns in the table or view. These keys are stored in a structure (B-tree) that enables SQL Server to find the row or rows associated with the key values quickly and efficiently.

A table or view can contain the following types of indexes:

* Clustered
  + Clustered indexes sort and store the data rows in the table or view based on their key values. These are the columns included in the index definition. There can be only one clustered index per table, because the data rows themselves can be stored in only one order.
  + The only time the data rows in a table are stored in sorted order is when the table contains a clustered index. When a table has a clustered index, the table is called a clustered table. If a table has no clustered index, its data rows are stored in an unordered structure called a heap.
* Nonclustered
  + Nonclustered indexes have a structure separate from the data rows. A nonclustered index contains the nonclustered index key values and each key value entry has a pointer to the data row that contains the key value.
  + The pointer from an index row in a nonclustered index to a data row is called a row locator. The structure of the row locator depends on whether the data pages are stored in a heap or a clustered table. For a heap, a row locator is a pointer to the row. For a clustered table, the row locator is the clustered index key.
  + You can add nonkey columns to the leaf level of the nonclustered index to by-pass existing index key limits, and execute fully covered, indexed, queries. For more information, see [Create Indexes with Included Columns](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/create-indexes-with-included-columns?view=sql-server-2017). For details about index key limits see [Maximum Capacity Specifications for SQL Server](https://docs.microsoft.com/en-us/sql/sql-server/maximum-capacity-specifications-for-sql-server?view=sql-server-2017).

Both clustered and nonclustered indexes can be unique. This means no two rows can have the same value for the index key. Otherwise, the index is not unique and multiple rows can share the same key value. For more information, see [Create Unique Indexes](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/create-unique-indexes?view=sql-server-2017).

Indexes are automatically maintained for a table or view whenever the table data is modified.

See [Indexes](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/indexes?view=sql-server-2017) for additional types of special purpose indexes.

### Single Indexes

In a relational database, an index is a data structure that increases retrieval speed at the expense of decreasing write speed as well as using more storage space.

Like a single index, a composite index is also a data structure of records sorted on something. But unlike a single index, that something is not a field, but a concatenation of multiple fields.

--------------------

A covering index is one which can satisfy all requested columns in a query without performing a further lookup into the clustered index.

If **all the columns** requested in the select list of query, are **available in the index**, then the query engine doesn't have to lookup the table again which can significantly increase the performance of the query. Since all the requested columns are available with in the index, the index is covering the query. So, the query is called a covering query and the index is a covering index.

<https://www.red-gate.com/simple-talk/sql/learn-sql-server/using-covering-indexes-to-improve-query-performance/>

**WHAT IS CURSOR?**

A cursor is a database object which is used to retrieve data from resultset one row at a time.The cursor can be used when the data needs to be updated row by row.

**CURSOR LIMITATIONS**

A cursor is a memory resident set of pointers -- meaning it occupies memory from your system that may be available for other processes.

Cursors can be faster than a while loop but they do have more overhead.

Another factor affecting cursor speed is the number of rows and columns brought into the cursor. Time how long it takes to open your cursor and fetch statements.

Too many columns being dragged around in memory, which are never referenced in the subsequent cursor operations, can slow things down.

The cursors are slower because they update tables row by row.

# @@ROWCOUNT

Returns the number of rows affected by the last statement.

DROP TABLE #tmp

SELECT TOP (1000) n = CONVERT(INT, ROW\_NUMBER() OVER (ORDER BY s1.[object\_id]))

INTO #tmp

FROM sys.all\_objects AS s1 CROSS JOIN sys.all\_objects AS s2

OPTION (MAXDOP 1);

select n from #tmp t

where n % 2 = 0

and n % 3 = 0

SELECT ID, Employee, ManagerID,

DENSE\_RANK() OVER (ORDER BY ManagerID) AS HLevel

FROM Employee

WITH CTE AS(

SELECT RN = ROW\_NUMBER()OVER(PARTITION BY colD ORDER BY colD)

FROM dbo.Duplicate

)

DELETE FROM CTE WHERE RN > 1

This is an example of web.config file inheritance. From [MSDN](http://msdn.microsoft.com/en-us/library/vstudio/ms178685(v=vs.100).aspx)

You can distribute ASP.NET configuration files throughout your application directories to configure ASP.NET applications in an inheritance hierarchy. This structure allows you to achieve the level of configuration detail that your applications require at the appropriate directory levels without affecting configuration settings at higher directory levels.

Specifically, for MVC projects, the web.config in the View subdirectory is used to tailor the .cshtml / .aspx files. You can use web.config files in subfolders to extend, override, and remove settings inherited from the app's own root, and further up the hierarchy, e.g. up to machine.config

Common configurations in the /Views/web.config include:

* Blocking requests attempting to access razor and aspx views directly (these need to be served from Controllers via the appropriate routes). A 404 response is configured for such direct requests, e.g.

<add path="\*" verb="\*" type="System.Web.HttpNotFoundHandler"/>

* To setup [default import namespaces](https://stackoverflow.com/a/6723046/314291) for the view pages, which would otherwise have to be explicitly added via using. You can add namespaces for your common custom assemblies here (e.g. custom html helper extensions) e.g.

SQL Questions