**Harbinger System**

16

**SQL Tips and Tricks**

**To write better queries**

**Bhagwat Mane**

Contents

[New to SQL 2012 4](#_Toc499575499)

[Concat 4](#_Toc499575500)

[Example 4](#_Toc499575501)

[Reference Link 4](#_Toc499575502)

[OFFSET and FETCH NEXT 4](#_Toc499575503)

[Example 4](#_Toc499575504)

[Choose() Function 4](#_Toc499575505)

[Best practice for SQL Server 9](#_Toc499575506)

[Stored Procedures Tips 13](#_Toc499575507)

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Created /Modified By | Date | Purpose |
| 1.0 | Bhagwat Mane | 20 May 2016 | Created initial draft. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# New to SQL 2012

## Concat

**CONCAT** takes a variable number of string arguments and concatenates them into a single string. It requires a minimum of two input values; otherwise, an error is raised. All arguments are implicitly converted to string types and then concatenated. Null values are implicitly converted to an empty string. If all the arguments are null, an empty string of type **varchar**(1) is returned. The implicit conversion to strings follows the existing rules for data type conversions.

### Example

1. SELECT CONCAT(1, 2, 3, 4) AS SingleString
2. SELECT CONCAT('One',1, 1.1, GETDATE()) AS SingleString
3. SELECT CONCAT('One',2,NULL) AS SingleString
4. SELECT CONCAT('','','','') AS SingleString
5. SELECT CONCAT(NULL, NULL) AS SingleString

### Reference Link

<https://msdn.microsoft.com/en-in/library/hh231515.aspx>

## OFFSET and FETCH NEXT

Using OFFSET and FETCH clause you can write TSQL code to fetch only a set of rows from the complete result set window. This will help you going forward achieve SQL Server Pagination thereby improving performance when retrieving and displaying large number of records from backed database.

### Example

Select \* from t\_cplIndex Where IndexMode=0

order by PatternCode

OFFSET 25 ROWS

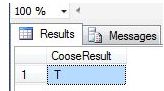
FETCH NEXT 50 ROWS ONLY;

Reference Link

<https://technet.microsoft.com/en-us/library/gg699618%28v=sql.110%29.aspx>

Choose() Function  
This function returns a value out of a list based on its index number. You can think of it as an array kind of thing. The Index number here starts from 1.

1. **DECLARE** @ShowIndex **INT**;
2. **SET** @ShowIndex =5;
3. **Select** Choose(@ShowIndex, 'M','N','H','P','T','L','S','H') **As** ChooseResult

In the preceding example we use index=5. It will start at 1. Choose() returns T as output since T is present at @Index location 5.  
  
**Output  
  
**

**Reference link:**

<http://www.c-sharpcorner.com/UploadFile/rohatash/top-10-exciting-new-features-of-sql-server-2012/>

**Parse Function**  
This function converts a string to Numeric and Date and Time formats. It will raise an error if translation isn't possible. You may still use CAST or CONVERT for general conversions. It depends on the presence of the CLR.

**Syntax**   
  
To demonstrate this new conversion function the following defines the syntax:  
  
PARSE ( string\_value AS data\_type [ USING culture ] )  
  
The Parse Function contains three parameters. The Culture part of the function is optional

**DATEFROMPARTS Function**The DATEFROMPARTS function returns a date value for the specified year, month, and day. The syntax of the DATEFROMPARTS built-in date function is as follows:   
  
**DATEFROMPARTS ( year, month, day )**All three parameters of the DATEFROMPARTS function are required.   
  
year: Integer expression specifying a year.  
month: Integer expression specifying a month, from 1 to 12.   
day: Integer expression specifying a day.

**Example**

1. **Declare** @Year **as** **int**=2013
2. **Declare** @Month **as** **int**=02
3. **Declare** @Day **as** **int**=20
4. **Select** DATEFROMPARTS(@Year, @Month, @Day)

**Output**  
2013-02-20

**TIMEFROMPARTS Function**The TIMEFROMPARTS function returns time values for the specified time and with the specified precision. The syntax of the TIMEFROMPARTS built-in date function is as follows:   
  
TIMEFROMPARTS ( hour, minute, seconds, fractions, precision)   
  
If the arguments are invalid, then an error is raised. If any of the parameters are null, null is returned.   
  
**Example**

1. **Declare** @**hour** **as** **int**=58
2. **Declare** @**minute** **as** **int**=46
3. **Declare** @seconds **as** **int**=20
4. **Declare** @fractions **as** **int**=0
5. **Declare** @**precision** **as** **int**=0
7. **Select** TIMEFROMPARTS(@**hour** , @**minute** , @seconds, @fractions , @**precision**)

**Output**58:46:20.0000000

**DATETIMEFROMPARTS Function**The DATETIMEFROMPARTS function returns a DateTime value for the specified date and time. The syntax of the DATETIMEFROMPARTS built-in date function is as follows:   
  
DATETIMEFROMPARTS ( year, month, day, hour, minute, seconds, milliseconds)  
   
If the arguments are invalid, then an error is raised. If any of the parameters are null, null is returned.   
  
**Example**

1. **Declare** @Year **as** **int**=2013
2. **Declare** @Month **as** **int**=12
3. **Declare** @Day **as** **int**=20
4. **Declare** @**hour** **as** **int**=58
5. **Declare** @**minute** **as** **int**=46
6. **Declare** @seconds **as** **int**=0
7. **Declare** @milliseconds **as** **int**=0

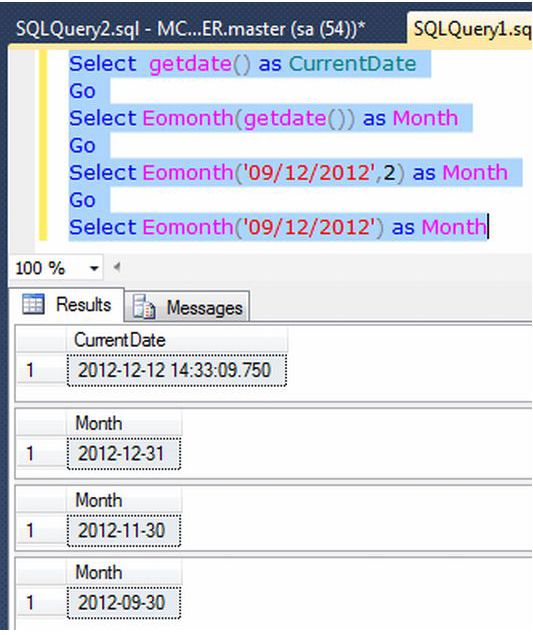
Select DATETIMEFROMPARTS (@Year, @Month, @Day, @hour , @minute , @seconds, @milliseconds)  
  
**Output**2013-12-20 58:59:46.0000000

**The Eomonth Function**The Eomonth function returns the last day of the month that contains the specified date.  
  
**Syntax**The syntax of the "Month" built-in date function is as follows:  
  
MONTH ( startdate [,month\_to\_add ] )   
  
Here:  
The "startdate" parameter can be an expression specifying the date for which to return the last day of the month.   
The "month\_to\_add" is optional.

**Example**

1. **Select** getdate()asCurrentDate
2. Go
3. SelectEomonth(getdate())asMonth
4. Go
5. SelectEomonth('09/12/2012',2)**as** Month
6. Go
7. SelectEomonth('09/12/2012')asMonth

**Output**



**Format Function**The Format() function is used to format how a field is to be displayed.  
  
Format converts the first argument to a specified format and returns the string value.  
  
**Syntax**  
FORMAT(column\_name,format)   
  
where both the field are required.  
  
This function formats the date time. This function is used in the server .NET Framework and CLR. This function will solve many formatting issues for developers.  
  
**Example**

1. **DECLARE** @d DATETIME = '20/03/2011';
2. **SELECT** FORMAT ( @d, 'd', 'en-US' ) **AS** US\_Result;

**Output**20/03/2011

# Best practice for SQL Server

* Keep primary key of lesser chars or integer. It is easier to process small width keys.
* Store image paths or URLs in database instead of images. It has less overhead.
* Use proper database types for the fields. If StartDate is database filed use date time as datatypes instead of VARCHAR (20).
* Specify column names instead of using \* in SELECT statement.
* Use LIKE clause properly. If you are looking for exact match use “=” instead.
* Using JOIN is better for performance than using sub queries or nested queries.
* Use stored procedures. They are faster and help in maintainability as well security of the database.
* User comments for readability as well as guidelines for the next developer who comes to modify the same code. Proper documentation of application will also aid help too.
* Proper indexing will improve the speed of operations in the database : <http://blog.sqlauthority.com/2008/04/06/sql-server-15-best-practices-for-better-database-performance/>
* **Avoid server-side Cursors** as much as possible, instead use SELECT statement. If you need to use cursor then replace it next suggestion.
* Avoid using spaces within the name of database objects; this may create issues with front-end data access tools and applications. If you need spaces in your database object name then will accessing it surround the database object name with square brackets.
* Do not use **reserved words** for naming database objects, as that can lead to some unpredictable situations.
* Do not use **wild card characters** at the beginning of word while search using LIKE keyword as it results in Index scan
* While using **JOINs** in your SQL query always **prefix column name** with the table name. ([Example](http://blog.sqlauthority.com/2008/08/02/sql-server-effect-of-order-of-join-in-query/)). If additionally require then prefix Table name with ServerName, DatabaseName, DatabaseOwner. ([Example](http://blog.sqlauthority.com/2007/06/26/sql-server-explanation-and-example-four-part-name/))

*SELECT he.EmployeeID, he.Title, hd.Name, hd.GroupName, hdh.StartDate  
FROM HumanResources.Employee he  
LEFT JOIN HumanResources.EmployeeDepartmentHistory hdh  
ON he.EmployeeID = hdh.EmployeeID  
RIGHT JOIN HumanResources.Department hd  
ON hd.DepartmentID = hdh.DepartmentID*

<http://blog.sqlauthority.com/2008/09/23/sql-server-coding-standards-guidelines-part-1/>

* Use transactions : Specially on long-running queries. This will save you when things get wrong. Working with data for some time you'll soon discover some unexpected situation which will make your stored procured crash.
* Don’t create too many indexes : Create the indexes it’s really useful otherwise don’t. I have seen a table has 5 non clustered indexes with mostly same column definitions for a particular table/SP and the table size is 4 GB and the index size are nearly 9 GB.
* Don’t use wildcard characters at the beginning of the word

Always write seekable codes.

*SELECT ID FROM <table\_name> WHERE NAME LIKE '%kumar'*

*-- use below*

*SELECT ID FROM <table\_name> WHERE NAME LIKE 'muthu%'*

* Use “order by” and “distinct” – sorting, really when there is a need : If you have a doubt on that, you can test run the code with/without sorting, you come to know, how it perform well.
* Use NO LOCK hint if you have no problem with dirty reads.
* Create a clustered index (Primary key) for all mostly used tables :

<http://www.sqlserverblogforum.com/tag/t-sql-server-best-practices/>

* Always use table aliases when your SQL statement involves more than one source : If more than one table is involved in a from clause, each column name must be qualified using either the complete table name or an alias. The alias is preferred.

It is more human readable to use aliases instead of writing columns with no table information.

* Do not use column numbers in the ORDER BY clause : Always use column names in an order by clause. Avoid positional references. Consider the following example in which the second query is more readable than the first one:

SELECT OrderID, OrderDate FROM Orders ORDER BY 2

SELECT OrderID, OrderDate FROM Orders ORDER BY OrderDate

* Always use a column list in your INSERT statements: Always specify the target columns when executing an insert command. This helps in avoiding problems when the table structure changes (like adding or dropping a column).

Consider the following table:

*CREATE TABLE EUROPEANCOUNTRIES  
  (  
     Countryid   INT PRIMARY KEY,  
     Countryname VARCHAR(25)  
  )*

Here’s an INSERT statement without a column list , that works perfectly:

INSERT INTO EuropeanCountries VALUES (1, ‘Ireland’)

Now, let’s add a new column to this table:

ALTER TABLE EuropeanCountries ADD EuroSupport bit

Now run the above INSERT statement. You get the following error from SQL Server:

Server: Msg 213, Level 16, State 4, Line 1

Insert Error: Column name or number of supplied values does not match table definition.

This problem can be avoided by writing an INSERT statement with a column list as shown below:

INSERT INTO EuropeanCountries(CountryID, CountryName) VALUES (1, ‘England’)

* Avoid doing an ORDER BY on a large data set, especially if the response time is important.
* Try not to use "OR" in a query. Instead use "UNION" to combine the result set of two distinguished queries. This will improve query performance.
* Better use UNION ALL if a distinguished result is not required. UNION ALL is faster than UNION as it does not have to sort the result set to find out the distinguished values.
* Dynamic Queries - Try to minimize the usage of dynamic queries.

# Stored Procedures Tips

Many of our projects do use databases. I am sure many of us use stored procedures commonly. Performance optimization is always a concern for many projects. With the projects I have worked, there are some interesting optimization tips I have come across to improve performance.

I would be sharing some of my experiences in next few blogs.

Today, I would like to share few things to keep in mind when you create stored procedures:

1. Use stored procedures instead of heavy-duty queries.
2. Include the SET NOCOUNT ON statement into your stored procedures to stop the message indicating the number of rows affected by a Transact-SQL statement.
3. Call stored procedure using its fully qualified name.
4. Consider returning the integer value as an RETURN statement instead of an integer value as part of a recordset.
5. Don't use the prefix "sp\_" in the stored procedure name if you need to create a stored procedure to run in a database other than the master database.
6. Use the sp\_executesql stored procedure instead of the EXECUTE statement
7. Use sp\_executesql stored procedure instead of temporary stored procedures.
8. If you have a very large stored procedure, try to break down this stored procedure into several sub-procedures, and call them from a controlling stored procedure.
9. Try to avoid using temporary tables inside your stored procedure.
10. Try to avoid using DDL (Data Definition Language) statements inside your stored procedure.
11. Use WITH ENCRYPTION: You can also use ‘WITH ENCRYPTION’ option while creating SP to hide the code.

What are some tips you use to improve the performance? Do let us know through the comments below.

**Keep in mind when you create stored procedure**

1. Use stored procedures instead of heavy-duty queries.
2. Include the SET NOCOUNT ON statement into your stored procedures to stop the message indicating the number of rows affected by a Transact-SQL statement.
3. Call stored procedure using its fully qualified name.
4. Consider returning the integer value as an RETURN statement instead of an integer value as part of a recordset.
5. Don't use the prefix "sp\_" in the stored procedure name if you need to create a stored procedure to run in a database other than the master database.
6. Use the sp\_executesql stored procedure instead of the EXECUTE statement
7. Use sp\_executesql stored procedure instead of temporary stored procedures.
8. If you have a very large stored procedure, try to break down this stored procedure into several sub-procedures, and call them from a controlling stored procedure.
9. Try to avoid using temporary tables inside your stored procedure.
10. Try to avoid using DDL (Data Definition Language) statements inside your stored procedure.
11. Do not use the RECOMPILE option for Stored Procedure unless there is specific requirements.
12. Try to avoid the temp table in the stored procedure. Stored procedures usually use a cached execution plan to increase the performance. When you use the temp table it will do the compilation every time
13. It is not recommended to use more variables in the procedure. It will occupy more space in the memory.
14. Use the ORDER BY and DISTINCT, TOP only when requires. The SQL Server engine will get the result first and it will do again the query execution for these operations.
15. It is recommended to use a Table variable when the result set is small. It is always in the memory and when the limit exceeds it will be created as a table in the temp. But the temp table will be created on the temp database and that makes it slower.
16. Use the proper indexing to the columns in the table. Do not create an index on the columns that are not used anywhere in the where clause. It will require an extra roundtrip to query the result.
17. Do not use SP parameters directly within the WHERE clause of SQL statements. This may cause the case of Prameter Sniffing. To avod this assign the parameter values to local variables and then use them with SQL queries.
18. Avoid using CURSORS: This will make your SP slow and degrade the performance. Try using more effecient SET based approaches instead of using CURSORS.
19. Use schema name with object name: The object name is qualified if used with schema name. Schema name should be used with the stored procedure name and with all objects referenced inside the stored procedure.
20. Use IF EXISTS (SELECT 1) instead of (SELECT \*): To check the existence of a record in another table, we uses the IF EXISTS clause. The IF EXISTS clause returns True if any value is returned from an internal statement, either a single value “1” or all columns of a record or complete recordset. The output of the internal statement is not used. Hence, to minimize the data for processing and network transferring, we should use “1” in the SELECT clause of an internal statement, as shown below:

IF EXISTS (SELECT 1 FROM sysobjects  
WHERE name = 'MyTable' AND type = 'U')

1. Use TRY-Catch for error handling

**Other SQL Tips**

* sp\_dropuser: It is used to drop the specified user those user exist in current data base

e.g. exec sp\_dropuser username

* sp\_helpdb: It is used to display the information about specified database or all database

e.g. exec sp\_helpdb or exec sp\_hyelpdb database name

* session\_user: It returns the current session name

e.g. select session\_user it gives "dbo"

* Set datefirst: It used to set the first week day whatever u want. bydefault monday is 1, tuesday 2 wednasday is 3..............
* Datename: it returns the date information in text format e.g. select datename(dw,getdate()) it returns wednesday and if select datename(month,getdate()) then it returns is December
* DB\_name(): It returns current database name
* sp\_detach\_db: It is used for data migration e.g. EXEC sp\_detach\_db 'MyTest', 'true' where MyTest is the database name
* sp\_attach\_db: It is used to attached database in SQL Server e.g.

EXEC sp\_attach\_db @dbname = N'MyTest', @filename1 = N'c:\Program Files\Microsoft SQL Server\MSSQL\Data\Mytest.mdf',

@filename2 = N'c:\Program Files\Microsoft SQL Server\MSSQL\Data\Mytest\_log.ldf'

where MyTest is the database name which database u want to connect.

* Those are not allowed in creating view when you want to create index on view

count(\*), rowset function, derived table self join, distinct, stdev, variance, avg, float \*, text, ntext, image column, subquery, full-text predicates, freetext,

sum on nullable expression, min, max, top, join, union.

* sp\_addtype: It is used to create a user defined data type e.g. EXEC sp\_addtype birthday, datetime, 'NULL' where birthday is a datatype name
* More than one column can’t altered at a time but if we want to required then we can use ALTER TABLE bbb ALTER COLUMN a CHAR(30) ALTER TABLE bbb ALTER COLUMN b CHAR(30) this way actually it look like two query but run simaltaneously
* For XML auto or raw command used with select query, it will automatically return data in xml format.

For query optimization need to use always create table then insert data instead of select col name into new table name from table name.

(A) CREATE TABLE #TempTable  
(spid int)

INSERT INTO #TempTable  
SELECT spid  
FROM sys.objects

(B) SELECT spid  
INTO #TempTable  
FROM sys.objects

In both Example (A) is much faster than (B)

SELECT @@VERSION It display the current version of Server

SQL Server Version (6.0, sp1, sp2, sp3, sp3a, 6.5, sp1, sp2, sp3, sp4,sp5,sp5a,7.0,sp1,sp2,sp3,sp4, 2000, sp1,sp2,sp3,sp4,2005

* These are objects type

C = CHECK constraint  
D = Default or DEFAULT constraint  
F = FOREIGN KEY constraint  
L = Log  
FN = Scalar function  
IF = Inlined table-function  
P = Stored procedure  
PK = PRIMARY KEY constraint (type is K)  
RF = Replication filter stored procedure   
S = System table  
TF = Table function  
TR = Trigger  
U = User table  
UQ = UNIQUE constraint (type is K)  
V = View  
X = Extended stored procedure

* Declare @TestScores table (StudentID int, Score int)

insert @TestScores (StudentID, Score) Values (1, 20)

insert @TestScores (StudentID, Score) Values (2, 03)

select \* from @testscores

This example for declare a variable as table type insert is used for insert the value and select statement for display the data from @TestScores variables

* insert into Orders

select 1,'ABC', '2007-01-01', 40 union all

select 2,'ABC', '2007-01-02', 30 union all

select 3,'ABC', '2007-01-03', 25 union all

select 4,'DEF', '2007-01-02', 10

it is another way to insert record in table

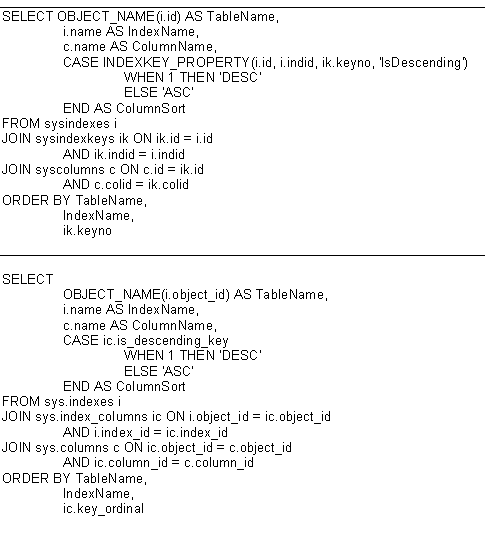
* These statements are used to display the information of specified database as well as tables in 2005

SELECT \* FROM INFORMATION\_SCHEMA.TABLES

SELECT \* FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = 'MyTable'

* Check below image -



In 2000

SELECT name FROM sysindexes

WHERE id = OBJECT\_ID('YourTable')

AND indid < 0 AND root < 0x0

In 2005

SELECT OBJECT\_DEFINITION(OBJECT\_ID('sysindexes'))

# and ## both option used for create temporary table but if u create with # this option it will not show in database as well as after close session as well as any other session but if u create with ## this option then it show another session still those session doesn’t goes to closed.

The main difference between UNION ALL and UNION is that, while UNION only selects distinct values, UNION ALL selects all values (including duplicates).

DBCC checkident(‘table name’) it tell the current identity value

DBCC CHECKIDENT('test',RESEED,29) it is used to set forcefully identity value

DBCC CHECKIDENT('test') WITH NO\_INFOMSGS it suppress all messages

* Insert into dbo.stgimage (Personid,Photo\_image) select '80000501',(SELECT \* FROM OPENROWSET(BULK 'D:\PMITestData\SendPilat10042008\TMS\_80000501.bmp', SINGLE\_BLOB)AS x)

update dbo.stgimage set Photo\_image =(SELECT \* FROM OPENROWSET(BULK 'D:\PMITestData\SendPilat10042008\TMS\_80000501.bmp', SINGLE\_BLOB)AS x) WHERE Personid ='80000501'

SELECT IDENTITY(int, 1,1) AS ID\_Num

Stored Procedure for upload image in data base

Alter Procedure TestImage as

begin

declare @idPerson as int,@filename as varchar(100),@filenamewithPath as varchar(200),@ssql as varchar(1000)

declare Image\_Cursor cursor FAST\_FORWARD for idPerson,filename from Mod\_PersonDocument

open Image\_Cursor

Fetch Next from Image\_Cursor into @idPerson,@filename

While(@@FETCH\_STATUS=0)

BEGIN

set @filenamewithPath='D:\indra\PilatImage\'+@filename

SET @ssql='Update Mod\_PersonDocument

set Document =(SELECT \* FROM OPENROWSET(BULK '''+@filenamewithPath+''', SINGLE\_BLOB)AS x )

WHERE idPerson ='+cast(@idPerson as varchar)+''

Exec(@ssql)

Fetch Next from Image\_Cursor into @idPerson,@filename

END

close Image\_Cursor

deallocate Image\_Cursor

End

* Use for Empty Log File

backup log [databasename] with truncate\_only

go

DBCC SHRINKDATABASE ([databasename], 10, TRUNCATEONLY)

go

* It is used to Identify that how much memory use ur sql server

run DBCC MEMORYSTATUS

* Used to convert string to date in SSIS package (DT\_DATE)(SUBSTRING(BEGDA,5,2) + "-" + SUBSTRING(BEGDA,7,2) + "-" + SUBSTRING(BEGDA,1,4))

(DT\_STR, 4, 1252) DATEPART("yyyy", @[System::StartTime]) + "-" +

(DT\_STR, 4, 1252) DATEPART("mm", @[System::StartTime]) + "-" +

(DT\_STR, 4, 1252) DATEPART("dd", @[System::StartTime])

* Delete duplicate records without using any extra table or extra column

WITH numbered AS

(

SELECT

ROW\_NUMBER() OVER (PARTITION BY keyvalue ORDER BY keyvalue) AS rowno,

keyvalue

FROM

#SampleTestData

)

DELETE FROM numbered

WHERE rowno > 1;

* New option in SQL 2005 CROSS APPLY

Let's start at the TOP

Along the way to making this solution work we'll discuss a few new features of SQL Server 2005. The first is that the TOP clause can now take a variable:

DECLARE @Rows INT

SET @Rows = 10

SELECT TOP ( @Rows ) \*

FROM Sales.SalesOrderHeader

This will return the top 10 rows from SalesOrderHeader. You can also replace @Rows with anything that evaluates to a number. The following query looks odd but runs just fine:

SELECT TOP (

SELECT COUNT(\*)

FROM Sales.Customer

) \*

FROM Sales.SalesOrderHeader

There are 19,185 rows in the Customer table and this query returns the top 19,185 rows from SalesOrderHeader. You can also use the TOP clause for INSERT, UPDATE and DELETE statements. If you wanted to DELETE in batches of 500 you can now do that using the TOP clause.

My Function

Next we need a function to return the TOP X rows from SalesOrderHeader based on the total sales amount. That function looks like this:

CREATE FUNCTION dbo.fn\_GetTopOrders(@custid AS int, @n AS INT)

RETURNS TABLE

AS

RETURN

SELECT TOP(@n) \*

FROM Sales.SalesOrderHeader

WHERE CustomerID = @custid

ORDER BY TotalDue DESC

GO

Notice that it accepts the number of orders to return as a parameter. Also notice that I'm using SELECT \* all over the place. I really encourage you to explicitly list out the columns when you write this for real. If you call this with a CustomerID and a number of rows it will return that many rows ordered by the total amount of the order in descending order. Also notice that there is an ORDER BY clause in this function. We'll talk about that in a minute.

Apply the APPLY Clause

The real magic happens when you use SQL Server 2005's new APPLY clause. The APPLY clause let's you join a table to a table-valued-function. That let's you write a query like this:

SELECT C.CustomerID,

O.SalesOrderID,

O.TotalDue

FROM

AdventureWorks.Sales.Customer AS C

CROSS APPLY

AdventureWorks.dbo.fn\_GetTopOrders(C.CustomerID, 3) AS O

ORDER BY

CustomerID ASC, TotalDue DESC

which results in this...

CustomerID SalesOrderID TotalDue

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

1 45283 37643.1378

1 46042 34722.9906

1 44501 26128.8674

2 46976 10184.0774

2 47997 5469.5941

2 57044 4537.8484

3 53616 92196.9738

3 47439 78578.9054

3 48378 56574.3871

4 47658 132199.8023

. . .

The APPLY clause acts like a JOIN without the ON clause comes in two flavors: CROSS and OUTER. The OUTER APPLY clause returns all the rows on the left side (Customers) whether they return any rows in the table-valued-function or not. The columns that the table-valued-function returns are null if no rows are returned. The CROSS APPLY only returns rows from the left side (Customers) if the table-valued-function returns rows.

* A table has only one column with identity

Create table Test (col1 int identity (1,1))

Insert test default values

* Find the size of any database

USE master;

go

SELECT name AS 'File',

CAST(CAST(SIZE\*1.0/128 AS DECIMAL(9,2)) AS VARCHAR(12)) + ' Mb' AS 'File Size',

CASE max\_size WHEN 0 THEN 'Off'

WHEN -1 THEN 'On'

ELSE 'Will grow to 2 Tb'

END AS 'Auto Growth',

growth AS 'Growth',

CASE WHEN growth = 0 THEN 'size is fixed and will not grow'

WHEN growth > 0 and is\_percent\_growth = 0 THEN 'Growth in 8Kb pages'

ELSE '%'

END AS 'Increment'

FROM tempdb.sys.database\_files;

GO

* Update the database size

USE master;

GO

ALTER DATABASE tempdb

MODIFY FILE (NAME = 'tempdev', SIZE = 434MB);

* Reduced database size see URL

http://support.microsoft.com/kb/307487

* select count(\*) from Mod\_Person

We can use this

SELECT rows FROM sysindexes WHERE id = OBJECT\_ID('Mod\_Person') AND indid < 2

* this is for identity

SELECT table\_name, column\_name

FROM INFORMATION\_SCHEMA.COLUMNS

where COLUMNPROPERTY (OBJECT\_ID(Table\_Name),Column\_Name,'IsIdentity') = 1

order by 1

<http://kashifzeeshan.spaces.live.com/Blog/cns!E76A8E802C3F7743!445.entry>

For database shrink in sql 2000 server

* How to run SQL Profiler without sysadmin rights?

In SQL Server 2000, to run SQL Profiler the sysadmin needs to grant permissions to the user running the trace. In SQL Server 2005 and 2008, the option exists of granting permissions to Profiler so that users do not need to be a member of the sysadmin group by using the code below:  
USE master   
GO   
GRANT ALTER TRACE TO username;   
GO  
If the permission needs to be revoked, use the code below:  
USE master   
GO   
DENY ALTER TRACE TO username;   
GO