## Super Key

Super key is a set of one or more than one keys that can be used to identify a record uniquely in a table.**Example :**Primary key, Unique key, Alternate key are subset of Super Keys.

## Candidate Key

A Candidate Key is a set of one or more fields/columns that can identify a record uniquely in a table. There can be multiple Candidate Keys in one table. Each Candidate Key can work as Primary Key.

**Example:** In below diagram ID, RollNo and EnrollNo are Candidate Keys since all these three fields can be work as Primary Key.

## Primary Key

Primary key is a set of one or more fields/columns of a table that uniquely identify a record in database table. It can not accept null, duplicate values. Only one Candidate Key can be Primary Key.

## Alternate key

A Alternate key is a key that can be work as a primary key. Basically it is a candidate key that currently is not primary key.

**Example:** In below diagram RollNo and EnrollNo becomes Alternate Keys when we define ID as Primary Key.

## Composite/Compound Key

Composite Key is a combination of more than one fields/columns of a table. It can be a Candidate key, Primary key.

## Unique Key

Uniquekey is a set of one or more fields/columns of a table that uniquely identify a record in database table. It is like Primary key but it can accept only one null value and it can not have duplicate values. For more help refer the article[Difference between primary key and unique key](http://www.dotnet-tricks.com/Tutorial/sqlserver/V2bS260912-Difference-between-Primary-Key-and-Unique-Key.html).

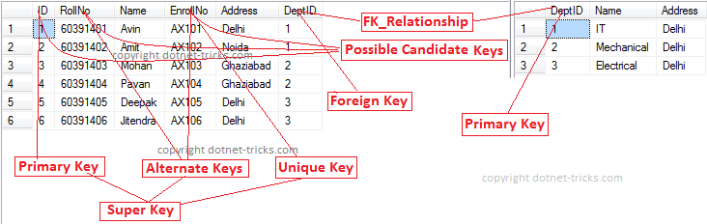
## Foreign Key

Foreign Key is a field in database table that is Primary key in another table. It can accept multiple null, duplicate values. For more help refer the article [Difference between primary key and foreign key](http://www.dotnet-tricks.com/Tutorial/sqlserver/TENc260912-Difference-between-Primary-Key-and-Foreign-Key.html).

**Example :** We can have a DeptID column in the Employee table which is pointing to DeptID column in a department table where it a primary key.

**Defined Keys -**

1. **CREATE TABLE Department**
2. **(**
3. **DeptID int PRIMARY KEY,**
4. **Name varchar (50) NOT NULL,**
5. **Address varchar (200) NOT NULL, )**
6. **CREATE TABLE Student**
7. **(**
8. **ID int PRIMARY KEY,**
9. **RollNo varchar(10) NOT NULL,**
10. **Name varchar(50) NOT NULL,**
11. **EnrollNo varchar(50) UNIQUE,**
12. **Address varchar(200) NOT NULL,**
13. **DeptID int FOREIGN KEY REFERENCES Department(DeptID)**
14. **)**



# Difference between SQL Server 2005 and SQL Server 2008?

In: [**Database Programming**](http://www.answers.com/T/Database_Programming), [**Windows Server 2003**](http://www.answers.com/T/Windows_Server_2003), [**Windows Server 2008**](http://www.answers.com/T/Windows_Server_2008) [[**Edit categories**](javascript:void(0))]

Answer:

**1 : compress**

In sql server 2005,There is no option to compress backup files, but in sql server 2008,there you find it.   
  
storing backup file takes 5 minutes without compression in sqlserver 2005,but it takes only 3 minutes in sql server 2008 for storing backup files with compression.   
  
CPU is used to compress the data before it is written to disk,so less data is written to disk.   
  
  
There are many new features, so it depends on what you need. If you store large files in SQL Server, migrated from Oracle and want to use date and time data types, need to encrypt your databases, etc you will definitely want to look at 2008. The nice thing is that it upgrading should be much easier from 2005 to 2008 than it was from 2000 to 2005.   
  
  
Server 2008 also added CMS which is Central Management Server. It only works with Windows Authentication but it allows you to management multiple SQL Servers at once. If SQL Server systems are popping up like weeds it will appear in the CMS provided that they point to the CMS via SSMS. Its a really cool feature.   
  
PBM Policy-Based Management is another added feature introduced with SQL Server 2008. PBM allows you to define and enforce policies for configuring and managing SQL Server across your enterprise. It goes hand-in-hand with CMS.   
  
  
One of my favorite is that fact that Reporting Services no longer requires IIS as it makes direct calls to HTTP.SYS.

**Difference between Sql server 2005 and sql server 2008**

• Transparent Data Encryption. The ability to encrypt an entire database.  
• Backup Encryption. Executed at backup time to prevent tampering.  
• External Key Management. Storing Keys separate from the data.  
• Auditing. Monitoring of data access.  
• Data Compression. Fact Table size reduction and improved performance.  
• Resource Governor. Restrict users or groups from consuming high levels or resources.  
• Hot Plug CPU. Add CPUs on the fly.  
• Performance Studio. Collection of performance monitoring tools.  
• Installation improvements. Disk images and service pack uninstall options.  
• Dynamic Development. New ADO and Visual Studio options as well as Dot Net 3.  
• Entity Data Services. Line Of Business (LOB) framework and Entity Query Language (eSQL)  
• LINQ. Development query language for access multiple types of data such as SQL and XML.  
• Data Synchronizing. Development of frequently disconnected applications.  
• Large UDT. No size restriction on UDT.  
• Dates and Times. New data types: Date, Time, Date Time Offset.  
• File Stream. New data type VarBinary(Max) FileStream for managing binary data.  
• Table Value Parameters. The ability to pass an entire table to a stored procedure.  
• Spatial Data. Data type for storing Latitude, Longitude, and GPS entries.  
• Full Text Search. Native Indexes, thesaurus as metadata, and backup ability.  
• SQL Server Integration Service. Improved multiprocessor support and faster lookups.  
• MERGE. TSQL command combining Insert, Update, and Delete.  
• SQL Server Analysis Server. Stack improvements, faster block computations.  
• SQL Server Reporting Server. Improved memory management and better rendering.  
• Microsoft Office 2007. Use OFFICE as an SSRS template. SSRS to WORD.

**SQL Server 2008 to SQL Server R2**• Backup compression is available in R2 standard (was only in Enterprise)  
• SQL Server 2008 R2 Express Edition database size limit increased to 10 GB (from 4GB)  
• PowerPivot for SharePoint introduced for loading, querying, and managing PowerPivot workbooks that you publish to a SharePoint 2010 server  
• PowerPivot for Excel is used to assemble and create relationships in large amounts of data from different sources, and then use that data as the basis for PivotTables and other data visualization objects that support data analysis in Excel.  
• Utility Explorer introduced for centralized management of managed SQL instances.  
• Master Data Services introduced to manage master (reference) data and maintain an auditable record of that data as it changes over time.  
• Enhanced security to prevent an authentication relay attack.

**SQL Server 2008R2 to 2012**• Inclusion of Visual Studio 2010 (BI edition)  
• AlwaysOn technology – allowing automatic replication and switching to mirrored database/server in the event of problems  
• In most instances, the performance is 10 times faster than the predecessor (Microsoft claim)  
• Buffer rate is high in SQL Server 2012 because of data compression.  
• Data visualization tool is available in SQL Server 2012.This allows snapshots of data.  
• Support for persistent computed columns and extra geographical approach is possible with spatial features in SQL Server 2012.  
• Allows for warm Service Patch patching – meaning SQL or server does not have to be rebooted/restarted to apply patches  
• New ‘Contained Databases’ feature allowing easier movement of database instances between servers  
• New ‘ColumnStore’ indexes – which turn indexes 90 degrees in use (instead of one index per row, you get multiple column indexes per row which has a major impact on performance)  
• New TRY\_CONVERT T-SQL command, which will perform data type conversion, but will NULL rather than error where conversion fails.  
• New OFFSET/FETCH command which allows automatic capture of pages of rows (like SELECT TOP 90 \* from…, but gives a starting row number for the page)  
• Redesign of SQL Server management studio – providing new features, better intellisense, new viewing tools and cleaner interface  
• Custom user server roles – meaning you can define roles with specific access (such as BI data reader) and assign these custom roles to defined users (therefore faster user maintenance)  
• New virtual Windows FileTable commands in T-SQL lets you manage a folder like a table of documents, but still have external control over the contents: UPDATE C:\Docs\\*.\* SET ReadOnly = 1 WHERE Author = ‘Bob’ AND Created < ’20100101′;)  
• NEGATIVE – SQL Server Management Studio will no longer run on XP. Needs Windows Vista, 7 or 8.  
• NEGATIVE – SQL Server is no longer licenced on server, but either CALS or Cores in server CPU which could increase costs for large user connection scenarios.  
• NEGATIVE – AWE is no longer supported, meaning your SQL Server instance on x86 with 32GB of RAM is only going to be able to use 4GB. Note – SQL now designed to run on 64bit hardware

SELECT

    CASE WHEN TRY\_CONVERT(float, 'test') IS NULL

    THEN 'Cast failed'

    ELSE 'Cast succeeded'

END AS Result;

Example 1 Skip first 10 rows from the sorted result set and return the remaining rows.

SELECT First Name + ' ' + Last Name FROM Employees ORDER BY First Name OFFSET 10 ROWS;

Example 2- Skip first 10 rows from the sorted resultset and return next 5 rows.

SELECT First Name + ' ' + Last Name FROM Employees ORDER BY First Name OFFSET 10 ROWS FETCH NEXT 5 ROWS ONLY;

**To Get Even Row From A Table**

SELECT \*

FROM (

SELECT \*, Row\_Number() OVER(ORDER BY userid) AS RowNumber

--Row\_Number() starts with 1

FROM tblEmployee

) t

WHERE t.RowNumber % 2 = 0 --Even

**To Get Odd Row From A Table**

SELECT \*

FROM (

SELECT \*, Row\_Number() OVER(ORDER BY userid) AS RowNumber

--Row\_Number() starts with 1

FROM tblEmployee

) t

WHERE t.RowNumber % 2 = 1 –Odd

select \*, RANK() over (order by userid) as RankNumber from tblEmployee

select \*, RANK() over (order by Age) as RankNumber from tblEmployee

select \*, Dense\_RANK() over (order by Age) as RankNumber from tblEmployee

select \*, Rank() over (partition by Age order by userid) as RankNumber from tblEmployee

CTE is again a temporary result set derived from the underling definition. For syntax of CTE, please refer to MSDN.

## A Simple Illustration: CTE as a Derived Table

We have a simple table Products in our database.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

Select \* from PRODUCTS

ProductID ProductDesc ManufacturingDate ExpiryDate IsSalable Price

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

1 Biscuits 2011-09-01 00:00:00.000 2012-09-01 00:00:00.000 1 20.00

2 Butter 2010-09-01 00:00:00.000 2011-09-01 00:00:00.000 1 30.00

3 Milk 2011-10-01 00:00:00.000 2011-11-01 00:00:00.000 1 46.00

We have created a simple ProductsCTE for displaying all the Products with Price greater than 20.00. Here CTE performs the job of acting as a simple derived table.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

;WITH ProductsCTE(ProdName,Price) AS

( SELECT ProductDesc,Price

FROM PRODUCTS

WHERE Price>20.00

)

SELECT \* FROM ProductsCTE

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

ProdName Price

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

Butter 30.00

Milk 46.00

(2 row(s) affected)

Important point that needs a mention is the SELECT followed by the CTE definition, any operation SELECT, INSERT,UPDATE, DELETE or Merge can be performed immediately after the CTE and the CTE lasts only for a single such operation. When I say that, what do I mean...

It means that the below code is erroneous:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

DECLARE @T INT,@I INT

SET @T = 10

SET @I = 20

;WITH ProductsCTE(ProdName,Price) AS

( SELECT ProductDesc,Price

FROM PRODUCTS

WHERE Price>20.00

)

SELECT @T+@I

SELECT \* FROM ProductsCTE

On executing the code, it results in the below error. Which means I need to execute the ProductsCTE SELECTimmediately after the CTE definition completes.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

Msg 422, Level 16, State 4, Line 10

Common table expression defined but not used.

To rectify the same... I would reinstate the order as:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

DECLARE @T INT,@I INT

SET @T = 10

SET @I = 20

;WITH ProductsCTE(ProdName,Price) AS

( SELECT ProductDesc,Price

FROM PRODUCTS

WHERE Price>20.00

)

SELECT \* FROM ProductsCTE

SELECT @T+@I

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

ProdName Price

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

Butter 30.00

Milk 46.00

(2 row(s) affected)

-----------

30

(1 row(s) affected)

Let’s perform an UPDATE:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

;WITH ProductsCTE(ProdName,Price) AS

( SELECT ProductDesc,Price

FROM PRODUCTS

WHERE Price>20.00

)

UPDATE ProductsCTE SET Price=50 WHERE ProdName='Milk'

SELECT \* FROM ProductsCTE

*-------------------*

(1 row(s) affected)

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

Msg 208, Level 16, State 1, Line 7

Invalid object name 'ProductsCTE'.

The price for Milk gets duly updated to 50 but the next set of select doesn’t work. Always remember, you can hit for a result set once and hit immediately after the CTE definition. Like:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

;WITH ProductsCTE(ProdName,Price) AS

( SELECT ProductDesc,Price

FROM PRODUCTS

WHERE Price>20.00

)

SELECT \* FROM ProductsCTE

UNION

SELECT 'Bread' AS ProdName,MIN(Price) AS PRICE from ProductsCTE

ProdName Price

*-------------------------------------------------- ---------------------*

Bread 30.00

Butter 30.00

Milk 50.00

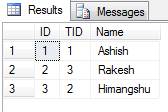
(3 row(s) affected)

Calling Multiple CTEs

We can have multiple CTEs calls from one single query. Let’s have a look at the example. We have 2 tables:

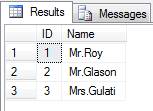
http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

Select \* from Student



http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

Select \* from Teacher



Let’s implement a CTE to get all the respective class teachers for the students.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

;WITH StudCTE(RollNo,StudentName,TeacherID)

AS

(

SELECT ID,Name,TID FROM Student

)

,TeacherCTE(TID,TeacherName)

AS

(

SELECT ID,Name FROM Teacher

)

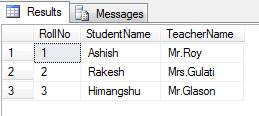
SELECT RollNo,StudentName,TeacherName

FROM StudCTE SC

INNER JOIN

TeacherCTE TC

ON SC.TeacherID=TC.TID



We have called 2 CTEs from a single SELECT and based upon the inner join returned the student-teacher information. That was a simple example to show how multiple CTEs are done.

## Complex Scenarios

So what is the big deal about CTE, the deal is when you need some complex queries or operations, trust me nothing goes as good as CTE. Let’s have a look at one of the most commonly encountered complex issues... Duplicates. We have a sample table (EMP) for the example.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

Select \* from EMP

EID ENAME DEPT

*----------- -------------------- ----------*

1 Sara IT

2 Rick HR

3 Ted IT

4 Sheldon Accounts

5 Sara IT

(5 row(s) affected)

For removing the duplicate employee, i.e. ‘Sara’ from the table, we create a CTE:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

;WITH EliminateDup(Eid,Name,Dept,RowID) AS

(

SELECT Eid,Ename,Dept, ROW\_NUMBER()OVER(PARTITION BY Ename,Dept ORDER BY EID)AS RowID

FROM EMP

)

DELETE FROM EliminateDup WHERE RID>1

The query below creates a temporary result set as :

SELECT Eid,Ename,Dept, ROW\_NUMBER()OVER(PARTITION BY Ename,Dept ORDER BY EID)AS RowID

FROM EMP

Eid Ename Dept RowID

*----------- -------------------- ---------- --------------------*

2 Rick HR 1

1 Sara IT 1

5 Sara IT 2

4 Sheldon Accounts 1

3 Ted IT 1

(5 row(s) affected)

And later, we remove the duplicate with the DELETE. Quite simple, isn’t it.

create table test1

(

A int ,

B int)

insert into test1 values (1,1)

insert into test1 values (1,1)

insert into test1 values (2,3)

select \* from test1

;with cteTest (A, B,ROWID) as (

select \*, Row\_number() over(partition by A,B order by A) as ROWID from test1

)

delete from cteTest where rowid>1

## Recursion

The next and the most important feature is recursion.

With the UNION ALL, we can make the CTE recursive to formulate a final result. There is an anchor member and a recursive member which may or may not have a terminating condition. Let’s see with an example..

Suppose we have a comma separated string and we wish to extract each word from the string..

Let’s consider the string to be ‘Where,there,is,a,will,there,is,a,way’.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

DECLARE @T VARCHAR(100)='Where,there,is,a,will,there,is,a,way'

SET @T =@T+','

;WITH MyCTE(Start,[End]) AS(

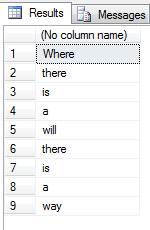
SELECT 1 AS Start,CHARINDEX(',',@T,1) AS [End]

UNION ALL

SELECT [End]+1 AS Start,CHARINDEX(',',@T,[End]+1)AS [End] from MyCTE where [End]<LEN(@T)

)

Select SUBSTRING(@T,Start,[End]-Start)from MyCTE;



Let’s understand what we have done here... We have an anchor in the form of SELECT 1,CHARINDEX(',',@T,1).

For the first pass, the anchor returns the values 1,6 (this value being the CHARINDEX of first comma after the word ‘Where,’) for columns Start & [End].

Next the recursive code returns [End]+1=7 as Start and 12 for CHARINDEX(',',@T,[End]+1, i.e. 7) AS [End], this code recurses unless the terminating condition is met which is [End]<LEN(@T)i.e. 37.

The UNION ALL operator unites all the start & [End], for clarity let's take another look at the values.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

DECLARE @T VARCHAR(100)='Where,there,is,a,will,there,is,a,way'

SET @T =@T+','

;WITH MyCTE(Start,[End]) AS(

SELECT 1 AS Start,CHARINDEX(',',@T,1) AS [End]

UNION ALL

SELECT [End]+1 AS Start,CHARINDEX(',',@T,[End]+1)AS [End] from MyCTE where [End]<LEN(@T)

)

Select Start,[End],SUBSTRING(@T,Start,[End]-Start)AS String from MyCTE;



Hope that makes things clearer. With CTE, we can achieve the same feats of programmability as C# or C++ with respect to generating Fibonacci series, a specific string patterns, etc. The recursion specifically finds an important use while you need a hierarchy to be reported, we will see that in a while. Currently, let’s look into the recursion option.

What if we want the first two values only out of the string?

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

DECLARE @T VARCHAR(100)='Where,there,is,a,will,there,is,a,way'

SET @T =@T+','

;WITH MyCTE(Start,[End]) AS(

SELECT 1 AS Start,CHARINDEX(',',@T,1) AS [End]

UNION ALL

SELECT [End]+1 AS Start,CHARINDEX(',',@T,[End]+1)AS [End] from MyCTE where [End]<LEN(@T)

)

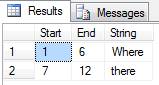
Select Start,[End],SUBSTRING(@T,Start,[End]-Start)AS String from MyCTE

OPTION (MAXRECURSION 1);

The OPTION MAXRECURSION enables the code to recurse only once and terminates as soon as that happens.The self explanatory message flashes and the values returned out on the results pane is:

1. initial anchor value
2. first recursed value





MAXRECURSION value can be between 0 and 32,767. 32,767 is fine but would what a 0 return? 0 enables an infinite recursion hence if the recursive statement does not have a terminating condition, the program loops infinitely. For first hand experience, try the below code?

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

;WITH MyCTE(Val) AS(

SELECT 1 AS Val

UNION ALL

SELECT Val=(Val+1) FROM MyCTE

)

Select Val from MyCTE

OPTION (MAXRECURSION 0);

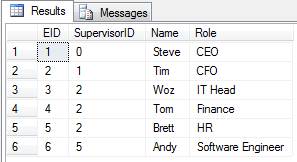
## Fetching Hierarchy

Before we call it a day, let’s look at the final example of fetching the complete hierarchy of a particular organization. In such scenarios, CTE could outperform any complex code both in terms of simplicity and LOC (lines of code) required to derive the result.

We have a table Org as below:

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

Select \* from Org



For fetching the bottom up hierarchy, we pass the eid and get the complete hierarchy for the concerned employee. For example, for Andy’s organizational hierarchy, we pass @T = 6 (His Eid).

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

DECLARE @T INT = 6

;WITH OrgCTE(Eid,Employee,SupervisorID,ReportsTo)AS

(

SELECT @T,O.Name,O2.EID,O2.Name FROM Org O

INNER JOIN Org O2 ON O.SupervisorID=O2.EID

AND O.EID=@T

UNION ALL

SELECT OC.SupervisorID,OC.ReportsTo,O2.EID,O2.Name

FROM OrgCTE OC

INNER JOIN

Org O

ON OC.SupervisorID=O.EID

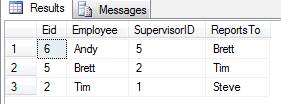
INNER JOIN

Org O2

ON O.SupervisorID=O2.EID

)

SELECT \* FROM OrgCTE



So we have been able to get the hierarchy for Andy.

Similarly for the top down hierarchy, we can implement the below CTE which gives the level indicating the top down org chart.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/275645/CTE-In-SQL-Server)

;WITH OrgCTE(Eid,SupervisorID,Employee,[Role],[Level])AS

(

SELECT EID,SupervisorID,Name,[Role],0 FROM

Org WHERE SupervisorID=0

UNION ALL

SELECT O.EID,O.SupervisorID,O.Name,O.[Role],[Level]+1

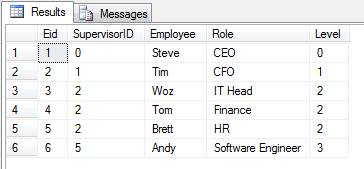
FROM Org O

INNER JOIN OrgCTE OC

ON O.SupervisorID=OC.Eid

)

SELECT \* FROM OrgCTE



SQL programmers find CTE of immense use and thanks to this feature, complexities in programming life have been considerably simplified. I hope I have been able to justify CTE reasonably well in this article.

[AUGUST 31, 2008](http://blog.sqlauthority.com/2008/08/31/sql-server-table-valued-parameters-in-sql-server-2008/) BY [PINAL DAVE](http://blog.sqlauthority.com/author/pinaldave/)

# SQL SERVER – 2008 – Introduction to Table-Valued Parameters with Example

***Table-Valued Parameters*** is a new feature introduced in SQL SERVER 2008. In earlier versions of SQL SERVER it is not possible to pass a table variable in stored procedure as a parameter, but now in SQL SERVER 2008 we can use Table-Valued Parameter to send multiple rows of data to a stored procedure or a function without creating a temporary table or passing so many parameters.

Table-valued parameters are declared using user-defined table types. To use a Table Valued Parameters we need follow steps shown below:

1. Create a table type and define the table structure
2. Declare a stored procedure that has a parameter of table type.
3. Declare a table type variable and reference the table type.
4. Using the INSERT statement and occupy the variable.
5. We can now pass the variable to the procedure.

For Example,

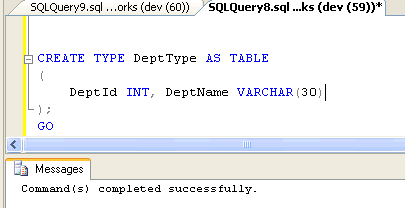
Let’s create a Department Table and pass the table variable to insert data using procedure. In our example we will create Department table and afterward we will query it and see that all the content of table value parameter is inserted into it.

Department:

**CREATE TABLE Department  
(  
DepartmentID INT PRIMARY KEY,  
DepartmentName VARCHAR(30)  
)  
GO**

**1. Create a TABLE TYPE and define the table structure:**

**CREATE TYPE DeptType AS TABLE  
(  
DeptId INT, DeptName VARCHAR(30)  
);  
GO**

  
**2. Declare a STORED PROCEDURE that has a parameter of table type:**

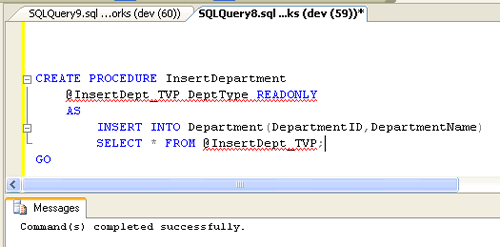
**CREATE PROCEDURE InsertDepartment  
@InsertDept\_TVP DeptType READONLY**  
**AS  
INSERT INTO Department(DepartmentID,DepartmentName)  
SELECT \* FROM @InsertDept\_TVP;**

**GO**

**Important points  to remember :**

-  Table-valued parameters must be passed as READONLY parameters to SQL routines. You cannot perform DML operations like UPDATE, DELETE, or INSERT on a table-valued parameter in the body of a routine.

-  You cannot use a table-valued parameter as target of a SELECT INTO or INSERT EXEC statement. A table-valued parameter can be in the FROM clause of SELECT INTO or in the INSERT EXEC string or stored-procedure.



**3. Declare a table type variable and reference the table type.**

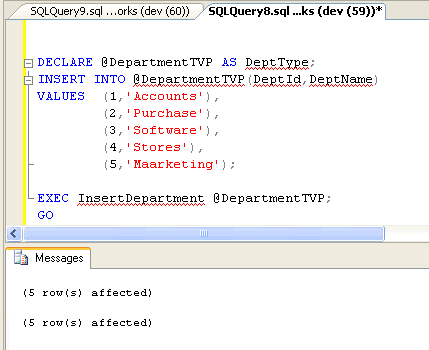
**DECLARE @DepartmentTVP AS DeptType;**

**4. Using the INSERT statement and occupy the variable.**

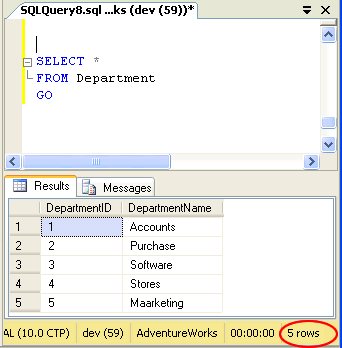
**INSERT INTO @DepartmentTVP(DeptId,DeptName)  
VALUES (1,'Accounts'),  
(2,'Purchase'),  
(3,'Software'),  
(4,'Stores'),  
(5,'Maarketing');**

**5. We can now pass the variable to the procedure and Execute.**

**EXEC InsertDepartment @DepartmentTVP;  
GO**



Let’s see if the Data are inserted in the Department Table



**Conclusion:**

Table-Valued Parameters is a new parameter type in SQL SERVER 2008 that provides efficient way of passing the table type variable than using the temporary table or passing so many parameters. It helps in using complex business logic in single routine. They reduce Round Trips to the server making the performance better.

# SQL SERVER – Stored Procedure Optimization Tips – Best Practices

We will go over how to optimize Stored Procedure with making simple changes in the code. Please note there are many more other tips, which we will cover in future articles.

* **Include SET NOCOUNT ON statement:** With every SELECT and DML statement, the SQL server returns a message that indicates the number of affected rows by that statement. This information is mostly helpful in debugging the code, but it is useless after that. By setting SET NOCOUNT ON, we can disable the feature of returning this extra information. For stored procedures that contain several statements or contain Transact-SQL loops, setting SET NOCOUNT to ON can provide a significant performance boost because network traffic is greatly reduced.

**CREATE PROC dbo.ProcName  
AS  
SET NOCOUNT ON;  
--Procedure code here  
SELECT column1 FROM dbo.TblTable1  
-- Reset SET NOCOUNT to OFF  
SET NOCOUNT OFF;  
GO**

* **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. This help in directly finding the complied plan instead of searching the objects in other possible schema before finally deciding to use a cached plan, if available. This process of searching and deciding a schema for an object leads to COMPILE lock on stored procedure and decreases the stored procedure’s performance. Therefore, always refer the objects with qualified name in the stored procedure like

**SELECT \* FROM dbo.MyTable -- Preferred method  
-- Instead of  
SELECT \* FROM MyTable -- Avoid this method  
--And finally call the stored procedure with qualified name like:  
EXEC dbo.MyProc -- Preferred method  
--Instead of  
EXEC MyProc -- Avoid this method**

* **Do not use the prefix “sp\_” in the stored procedure name**: If a stored procedure name begins with “SP\_,” then SQL server first searches in the master database and then in the current session database. Searching in the master database causes extra overhead and even a wrong result if another stored procedure with the same name is found in master database.
* **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')**

* **Use the sp\_executesql stored procedure instead of the EXECUTE statement.**  
  The sp\_executesql stored procedure supports parameters. So, using the sp\_executesql stored procedure instead of the EXECUTE statement improve the re-usability of your code. The execution plan of a dynamic statement can be reused only if each and every character, including case, space, comments and parameter, is same for two statements. For example, if we execute the below batch:

**DECLARE @Query VARCHAR(100)  
DECLARE @Age INT  
SET @Age = 25  
SET @Query = 'SELECT \* FROM dbo.tblPerson WHERE Age = ' +CONVERT(VARCHAR(3),@Age)  
EXEC (@Query)**

If we again execute the above batch using different @Age value, then the execution plan for SELECT statement created for @Age =25 would not be reused. However, if we write the above batch as given below,

**DECLARE @Query NVARCHAR(100)  
SET @Query = N'SELECT \* FROM dbo.tblPerson WHERE Age = @Age'  
EXECUTE sp\_executesql @Query, N'@Age int', @Age = 25**

the compiled plan of this SELECT statement will be reused for different value of @Age parameter. The reuse of the existing complied plan will result in improved performance.

* **Try to avoid using SQL Server cursors whenever possible:** Cursor uses a lot of resources for overhead processing to maintain current record position in a recordset and this decreases the performance. If we need to process records one-by-one in a loop, then we should use the WHILE clause. Wherever possible, we should replace the cursor-based approach with SET-based approach. Because the SQL Server engine is designed and optimized to perform SET-based operation very fast. Again, please note cursor is also a kind of WHILE Loop.
* **Keep the Transaction as short as possible:** The length of transaction affects blocking and deadlocking. Exclusive lock is not released until the end of transaction. In higher isolation level, the shared locks are also aged with transaction. Therefore, lengthy transaction means locks for longer time and locks for longer time turns into blocking. In some cases, blocking also converts into deadlocks. So, for faster execution and less blocking, the transaction should be kept as short as possible.
* **Use TRY-Catch for error handling**: Prior to SQL server 2005 version code for error handling, there was a big portion of actual code because an error check statement was written after every t-sql statement. More code always consumes more resources and time. In SQL Server 2005, a new simple way is introduced for the same purpose. The syntax is as follows:

**BEGIN TRY  
--Your t-sql code goes here  
END TRY  
BEGIN CATCH  
--Your error handling code goes here  
END CATCH**

## What is a Trigger

A trigger is a special kind of a store procedure that executes in response to certain action on the table like insertion, deletion or updation of data. It is a database object which is bound to a table and is executed automatically. You can’t explicitly invoke triggers. The only way to do this is by performing the required action no the table that they are assigned to.

## Types Of Triggers

There are three action query types that you use in SQL which are INSERT, UPDATE and DELETE. So, there are three types of triggers and hybrids that come from mixing and matching the events and timings that fire them. Basically, triggers are classified into two main types:

1. After Triggers (For Triggers)
2. Instead Of Triggers

## (i) After Triggers

These triggers run after an insert, update or delete on a table. They are **not supported for views.**   
AFTER TRIGGERS can be classified further into three types as:

1. AFTER INSERT Trigger.
2. AFTER UPDATE Trigger.
3. AFTER DELETE Trigger.

Let’s create After triggers. First of all, let’s create a table and insert some sample data. Then, on this table, I will be attaching several triggers.

Hide   Copy Code

CREATE TABLE Employee\_Test

(

Emp\_ID INT Identity,

Emp\_name Varchar(100),

Emp\_Sal Decimal (10,2)

)

INSERT INTO Employee\_Test VALUES ('Anees',1000);

INSERT INTO Employee\_Test VALUES ('Rick',1200);

INSERT INTO Employee\_Test VALUES ('John',1100);

INSERT INTO Employee\_Test VALUES ('Stephen',1300);

INSERT INTO Employee\_Test VALUES ('Maria',1400);

I will be creating an AFTER INSERT TRIGGER which will insert the rows inserted into the table into another audit table. The main purpose of this audit table is to record the changes in the main table. This can be thought of as a generic audit trigger.

Now, create the audit table as:-

Hide   Copy Code

CREATE TABLE Employee\_Test\_Audit

(

Emp\_ID int,

Emp\_name varchar(100),

Emp\_Sal decimal (10,2),

Audit\_Action varchar(100),

Audit\_Timestamp datetime

)

## (a) After Insert Trigger

This trigger is fired after an INSERT on the table. Let’s create the trigger as:

Hide   Copy Code

CREATE TRIGGER trgAfterInsert ON [dbo].[Employee\_Test]

FOR INSERT

AS

declare @empid int;

declare @empname varchar(100);

declare @empsal decimal(10,2);

declare @audit\_action varchar(100);

select @empid=i.Emp\_ID from inserted i;

select @empname=i.Emp\_Name from inserted i;

select @empsal=i.Emp\_Sal from inserted i;

set @audit\_action='Inserted Record -- After Insert Trigger.';

insert into Employee\_Test\_Audit

(Emp\_ID,Emp\_Name,Emp\_Sal,Audit\_Action,Audit\_Timestamp)

values(@empid,@empname,@empsal,@audit\_action,getdate());

PRINT 'AFTER INSERT trigger fired.'

GO

The CREATE TRIGGER statement is used to create the trigger. THE ON clause specifies the table name on which the trigger is to be attached. The FOR INSERT specifies that this is an AFTER INSERT trigger. In place of FOR INSERT, AFTER INSERT can be used. Both of them mean the same.

In the trigger body, table named **inserted**has been used. This table is a logical table and contains the row that has been inserted. I have selected the fields from the logical inserted table from the row that has been inserted into different variables, and finally inserted those values into the Audit table.

To see the newly created trigger in action, lets insert a row into the main table as:

Hide   Copy Code

insert into Employee\_Test values('Chris',1500);

Now, a record has been inserted into the Employee\_Test table. The AFTER INSERT trigger attached to this table has inserted the record into the Employee\_Test\_Audit as:

Hide   Copy Code

6 Chris 1500.00 Inserted Record -- After Insert Trigger. 2008-04-26 12:00:55.700

## (b) AFTER UPDATE Trigger

This trigger is fired after an update on the table. Let’s create the trigger as:

Hide   Copy Code

CREATE TRIGGER trgAfterUpdate ON [dbo].[Employee\_Test]

FOR UPDATE

AS

declare @empid int;

declare @empname varchar(100);

declare @empsal decimal(10,2);

declare @audit\_action varchar(100);

select @empid=i.Emp\_ID from inserted i;

select @empname=i.Emp\_Name from inserted i;

select @empsal=i.Emp\_Sal from inserted i;

if update(Emp\_Name)

set @audit\_action='Updated Record -- After Update Trigger.';

if update(Emp\_Sal)

set @audit\_action='Updated Record -- After Update Trigger.';

insert into Employee\_Test\_Audit(Emp\_ID,Emp\_Name,Emp\_Sal,Audit\_Action,Audit\_Timestamp)

values(@empid,@empname,@empsal,@audit\_action,getdate());

PRINT 'AFTER UPDATE Trigger fired.'

GO

The AFTER UPDATE Trigger is created in which the updated record is inserted into the audit table. There is **no logical table updated like the logical table inserted.** We can obtain the updated value of a field from theupdate(column\_name) function. In our trigger, we have used, if update(Emp\_Name) to check if the column Emp\_Name has been updated. We have similarly checked the column Emp\_Sal for an update.

Let’s update a record column and see what happens.

Hide   Copy Code

update Employee\_Test set Emp\_Sal=1550 where Emp\_ID=6

This inserts the row into the audit table as:

Hide   Copy Code

6 Chris 1550.00 Updated Record -- After Update Trigger. 2008-04-26 12:38:11.843

## (c) AFTER DELETE Trigger

This trigger is fired after a delete on the table. Let’s create the trigger as:

Hide   Copy Code

CREATE TRIGGER trgAfterDelete ON [dbo].[Employee\_Test]

AFTER DELETE

AS

declare @empid int;

declare @empname varchar(100);

declare @empsal decimal(10,2);

declare @audit\_action varchar(100);

select @empid=d.Emp\_ID from deleted d;

select @empname=d.Emp\_Name from deleted d;

select @empsal=d.Emp\_Sal from deleted d;

set @audit\_action='Deleted -- After Delete Trigger.';

insert into Employee\_Test\_Audit

(Emp\_ID,Emp\_Name,Emp\_Sal,Audit\_Action,Audit\_Timestamp)

values(@empid,@empname,@empsal,@audit\_action,getdate());

PRINT 'AFTER DELETE TRIGGER fired.'

GO

In this trigger, the deleted record’s data is picked from the **logical deleted table** and inserted into the audit table. Let’s fire a delete on the main table. A record has been inserted into the audit table as:

Hide   Copy Code

6 Chris 1550.00 Deleted -- After Delete Trigger. 2008-04-26 12:52:13.867

All the triggers can be enabled/disabled on the table using the statement

Hide   Copy Code

ALTER TABLE Employee\_Test {ENABLE|DISBALE} TRIGGER ALL

Specific Triggers can be enabled or disabled as:

Hide   Copy Code

ALTER TABLE Employee\_Test DISABLE TRIGGER trgAfterDelete

This disables the After Delete Trigger named trgAfterDelete on the specified table.

## (ii) Instead Of Triggers

These can be used as an interceptor for anything that anyone tried to do on our table or view. If you define anInstead Of trigger on a table for the Delete operation, they try to delete rows, and they will not actually get deleted (unless you issue another delete instruction from within the trigger)

INSTEAD OF TRIGGERS can be classified further into three types as:

1. INSTEAD OF INSERT Trigger.
2. INSTEAD OF UPDATE Trigger.
3. INSTEAD OF DELETE Trigger.

Let’s create an Instead Of Delete Trigger as:

Hide   Copy Code

CREATE TRIGGER trgInsteadOfDelete ON [dbo].[Employee\_Test]

INSTEAD OF DELETE

AS

declare @emp\_id int;

declare @emp\_name varchar(100);

declare @emp\_sal int;

select @emp\_id=d.Emp\_ID from deleted d;

select @emp\_name=d.Emp\_Name from deleted d;

select @emp\_sal=d.Emp\_Sal from deleted d;

BEGIN

if(@emp\_sal>1200)

begin

RAISERROR('Cannot delete where salary > 1200',16,1);

ROLLBACK;

end

else

begin

delete from Employee\_Test where Emp\_ID=@emp\_id;

COMMIT;

insert into Employee\_Test\_Audit(Emp\_ID,Emp\_Name,Emp\_Sal,Audit\_Action,Audit\_Timestamp)

values(@emp\_id,@emp\_name,@emp\_sal,'Deleted -- Instead Of Delete Trigger.',getdate());

PRINT 'Record Deleted -- Instead Of Delete Trigger.'

end

END

GO

This trigger will prevent the deletion of records from the table where Emp\_Sal > 1200. If such a record is deleted, the Instead Of Trigger will rollback the transaction, otherwise the transaction will be committed. Now, let’s try to delete a record with the Emp\_Sal >1200 as:

Hide   Copy Code

delete from Employee\_Test where Emp\_ID=4

This will print an error message as defined in the RAISE ERROR statement as:

Hide   Copy Code

Server: Msg 50000, Level 16, State 1, Procedure trgInsteadOfDelete, Line 15

Cannot delete where salary > 1200

And this record will not be deleted.

In a similar way, you can code Instead of Insert and Instead Of Update triggers on your tables.

## Conclusion

In this article, I took a brief introduction of triggers, explained the various kinds of triggers – After Triggers and Instead Of Triggers along with their variants and explained how each of them works. I hope you will get a clear understanding about the Triggers in SQL Server and their usage.

## Introduction

In most of our application scenario, we need to get latest inserted row information through SQL query. And for that, we have multiple options like:

* @@IDENTITY
* SCOPE\_IDENTITY
* IDENT\_CURRENT

All three functions return last-generated identity values. However, the scope and session on which last is defined in each of these functions differ.

## Compare

### @@IDENTITY

It returns the last identity value generated for any table in the current session, across all scopes.

Let me explain this... suppose we create an insert trigger on table which inserts a row in another table with generate an identity column, then @@IDENTITY returns that identity record which is created by trigger.

### SCOPE\_IDENTITY

It returns the last identity value generated for any table in the current session and the current scope.

Let me explain this... suppose we create an insert trigger on table which inserts a row in another table with generate an identity column, then SCOPE\_IDENTITY result is not affected but if a trigger or a user defined function is affected on the same table that produced the value returns that identity record thenSCOPE\_IDENTITY returns that identity record which is created by trigger or a user defined function.

### IDENT\_CURRENT

It returns the last identity value generated for a specific table in any session and any scope.

In other words, we can say it is not affected by scope and session, it only depends on a particular table and returns that table related identity value which is generated in any session or scope.

## SQL Query

I am explaining the above process with the help of some sample query, hope it helps:

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

CREATE TABLE Parent(id int IDENTITY);

CREATE TABLE Child(id int IDENTITY(100,1));

GO

CREATE TRIGGER Parentins ON Parent FOR INSERT

AS

BEGIN

INSERT Child DEFAULT VALUES

END;

GO

*--End of trigger definition*

SELECT id FROM Parent;

*--id is empty.*

SELECT id FROM Child;

*--ID is empty.*

*--Do the following in Session 1*

INSERT Parent DEFAULT VALUES;

SELECT @@IDENTITY;

*/\*Returns the value 100. This was inserted by the trigger.\*/*

SELECT SCOPE\_IDENTITY();

*/\* Returns the value 1. This was inserted by the*

*INSERT statement two statements before this query.\*/*

SELECT IDENT\_CURRENT('Child');

*/\* Returns value inserted into Child, that is in the trigger.\*/*

SELECT IDENT\_CURRENT('Parent');

*/\* Returns value inserted into Parent.*

*This was the INSERT statement four statements before this query.\*/*

*-- Do the following in Session 2.*

SELECT @@IDENTITY;

*/\* Returns NULL because there has been no INSERT action*

*up to this point in this session.\*/*

SELECT SCOPE\_IDENTITY();

*/\* Returns NULL because there has been no INSERT action*

*up to this point in this scope in this session.\*/*

SELECT IDENT\_CURRENT('Child');

*/\* Returns the last value inserted into Child.\*/*