- Part 1 – Connecting to SQL Server
- Part 2 – Creating, altering and dropping a database

Creating and Altering a Database

A SQL Server database can be created, altered and dropped
1. Graphically using SQL Server Management Studio (SSMS) or
2. Using a Query

To Create the database using a query
Create database DatabaseName

Whether, you create a database graphically using the designer or, using a query, the following 2 files gets generated
.MDF file - Data File (Contains actual data)
.LDF file - Transaction Log file (Used to recover the database)

To alter a database, once it's created
Alter database DatabaseName Modify Name = NewDatabaseName

Alternatively, you can also use system stored procedure
Execute sp_renameDB 'OldDatabaseName','NewDatabaseName'

Deleting or Dropping a database

To Delete or Drop a database
Drop Database DatabaseThatYouWantToDrop

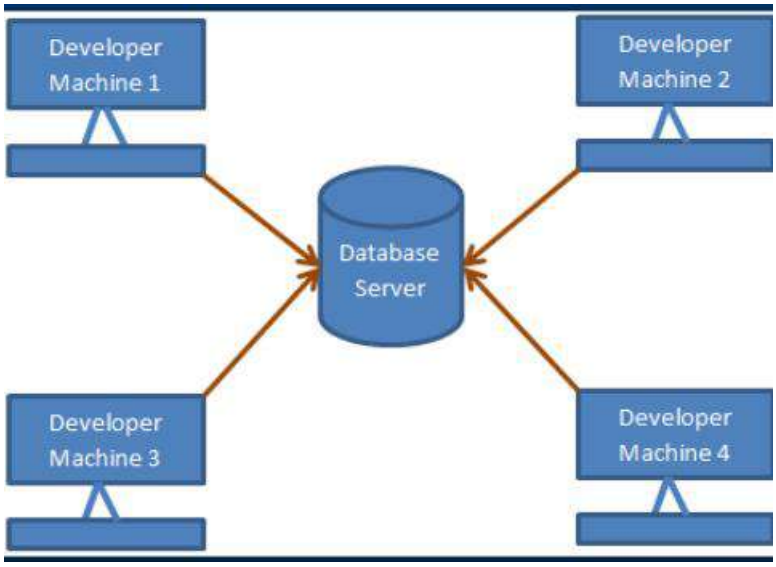
Dropping a database, deletes the LDF and MDF files.

You cannot drop a database, if it is currently in use. You get an error stating - Cannot drop database "NewDatabaseName" because it is currently in use.

So, if other users are connected, you need to put the database in single user mode and then drop the database.
Alter Database DatabaseName Set SINGLE_USER With Rollback Immediate

With Rollback Immediate option, will rollback all incomplete transactions and closes the connection to the database.

Note: System databases cannot be dropped.



Part 3 – Creating and working with tables

Creating Tables

The aim of this session is to create tblPerson and tblGender tables and establish primary key and foreign key constraints.

tblPerson			
ID	Name	Email	GenderID
1	Jade	j@j.com	2
2	Mary	m@m.com	3
3	Martin	ma@ma.com	1
4	Rob	r@r.com	NULL
5	May	may@may.com	2
6	Kristy	k@k.com	NULL

tblGender	
ID	Gender
1	Male
2	Female
3	Unknown

In SQL Server, tables can be created graphically using SQL Server Management Studio (SSMS) or using a query. Foreign key references can be added graphically using SSMS or using a query.

```
Alter table ForeignKeyTable add constraint ForeignKeyTable_ForiegnKeyColumn_FK
FOREIGN KEY (ForiegnKeyColumn) references PrimaryKeyTable (PrimaryKeyColumn)
```

Foreign keys are used to enforce database integrity. In layman's terms, A foreign key in one table points to a primary key in another table. The foreign key constraint prevents invalid data form being inserted into the foreign key column. The values that you enter into the foreign key column, has to be one of the values contained in the table it points to.

Part 4 – Default Constraint

Default Constraint

tblPerson			
ID	Name	Email	GenderID
1	Jade	j@j.com	2
2	Mary	m@m.com	3
3	Martin	ma@ma.com	1
4	Rob	r@r.com	NULL
5	May	may@may.com	2
6	Kristy	k@k.com	NULL

tblGender	
ID	Gender
1	Male
2	Female
3	Unknown

A column default can be specified using Default constraint. The DEFAULT constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified, including NULL.

Altering an existing column to add a default constraint:

```
ALTER TABLE {TABLE_NAME}
ADD CONSTRAINT {CONSTRAINT_NAME}
DEFAULT {DEFAULT_VALUE} FOR {EXISTING_COLUMN_NAME}
```

Adding a new column, with default value, to an existing table:

```
ALTER TABLE {TABLE_NAME}
ADD {COLUMN_NAME} {DATA_TYPE} {NULL | NOT NULL}
CONSTRAINT {CONSTRAINT_NAME} DEFAULT {DEFAULT_VALUE}
```

Dropping a constraint:

```
ALTER TABLE {TABLE_NAME}
DROP CONSTRAINT {CONSTRAINT_NAME}
```

Part 5 – Cascading referential integrity constraint

Cascading referential integrity

tblPerson			
ID	Name	Email	GenderID
1	Jade	j@j.com	2
2	Mary	m@m.com	3
3	Martin	ma@ma.com	1
4	Rob	r@r.com	NULL
5	May	may@may.com	2
6	Kristy	k@k.com	NULL

tblGender	
ID	Gender
1	Male
2	Female
3	Unknown

Cascading referential integrity constraint allows to define the actions Microsoft SQL Server should take when a user attempts to delete or update a key to which an existing foreign keys points.

For example, If you delete row with ID = 1 from tblGender table, then row with ID = 3 from tblPerson table becomes an orphan record. You will not be able to tell the Gender for this row. So, Cascading referential integrity constraint can be used to define actions Microsoft SQL Server should take when this happens. By default, we get an error and the DELETE or UPDATE statement is rolled back.

Cascading referential integrity

Options when setting up Cascading referential integrity constraint:

1. No Action: This is the default behaviour. No Action specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, an error is raised and the DELETE or UPDATE is rolled back.

2. Cascade: Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are also deleted or updated.

3. Set NULL: Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are set to NULL.

4. Set Default: Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are set to default values.

- Part 6 – CHECK constraint

Check Constraint

CHECK constraint is used to limit the range of the values, that can be entered for a column.

The general formula for adding check constraint in SQL Server:

```
ALTER TABLE {TABLE_NAME}
ADD CONSTRAINT {CONSTRAINT_NAME} CHECK (BOOLEAN_EXPRESSION)
```

If the BOOLEAN_EXPRESSION returns true, then the CHECK constraint allows the value, otherwise it doesn't. Since, AGE is a nullable column, it's possible to pass null for this column, when inserting a row. When you pass NULL for the AGE column, the boolean expression evaluates to UNKNOWN, and allows the value.

To drop the CHECK constraint:

```
ALTER TABLE tblPerson
DROP CONSTRAINT CK_tblPerson_Age
```

- Part 7 – Identity Column

Check Constraint

If a column is marked as an identity column, then the values for this column are automatically generated, when you insert a new row into the table.

```
Create Table tblPerson
(
   PersonId int Identity(1,1) Primary Key,
   Name nvarchar(20)
)
```

Note: Seed and Increment values are optional. If you don't specify the identity and seed they both default to 1.

To explicitly supply a value for identity column
1. First turn on identity insert - SET Identity_Insert tblPerson ON
2. In the insert query specify the column list
   Insert into tblPerson(PersonId, Name) values(2, 'John')

If you have deleted all the rows in a table, and you want to reset the identity column value, use DBCC CHECKIDENT command.

```
DBCC CHECKIDENT('tblPerson', RESEED, 0)
```

- Part 8 – Retrieving Identity Column values

Retrieving Identity Column values

Pre Req - Part 7 - Identity Column in SQL Server

From the previous session, we understood that identity column values are auto generated. There are several ways in sql server, to retrieve the last identity value that is generated. The most common way is to use SCOPE_IDENTITY() built in function.

Note: You can also use @@IDENTITY and IDENT_CURRENT('TableName')

Difference:
SCOPE_IDENTITY() - Same session and the same scope.
@@IDENTITY - Same session and across any scope.
IDENT_CURRENT('TableName') - Specific table across any session and any scope.

- Part 9 – Unique key constraint

Unique key constraint

We use UNIQUE constraint to enforce uniqueness of a column i.e the column shouldn't allow any duplicate values. We can add a Unique constraint thru the designer or using a query.

To create the unique key using a query:
Alter Table Table_Name
Add Constraint Constraint_Name Unique(Column_Name)

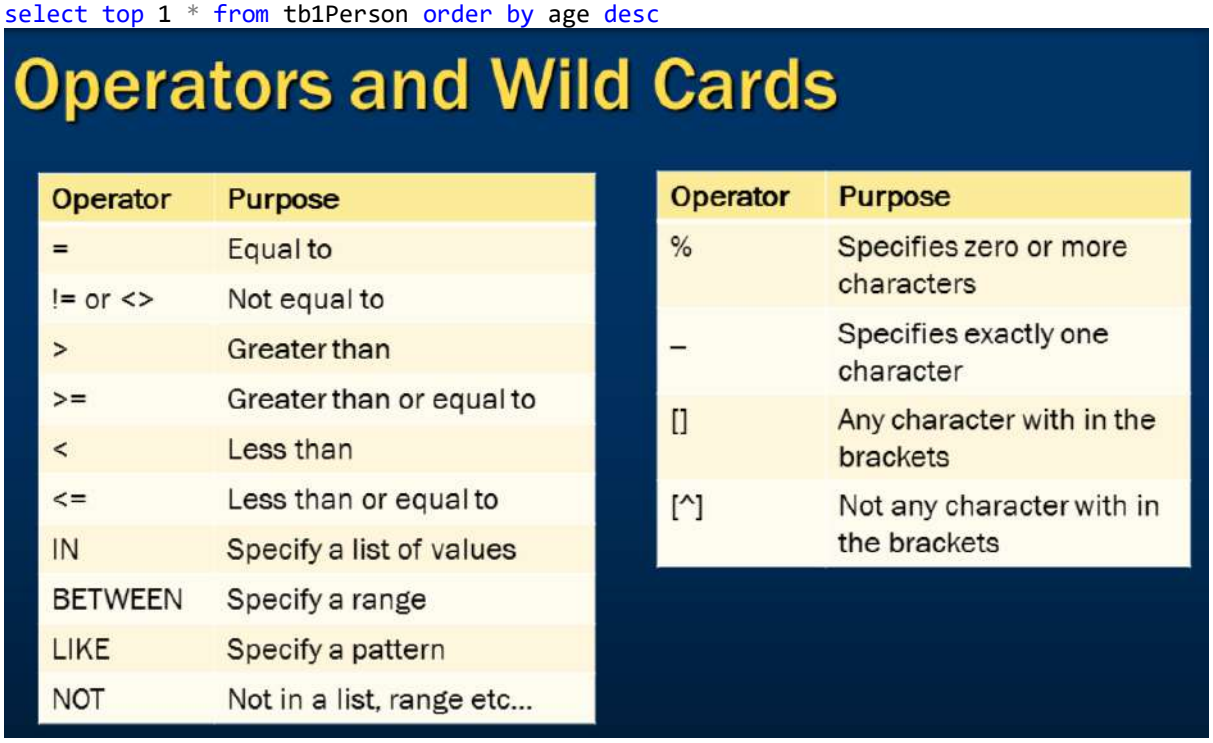
Bot primary key and unique key are used to enforce, the uniqueness of a column. So, when do you choose one over the other?
A table can have, only one primary key. If you want to enforce uniqueness on 2 or more columns, then we use unique key constraint.

What is the difference between Primary key constraint and Unique key constraint?
1. A table can have only one primary key, but more than one unique key
2. Primary key does not allow nulls, where as unique key allows one null

Part 10 – All about "SELECT"

```
--Select statement
--(1) Select specific columns
SELECT [ID]
      ,[Name]
      ,[Email]
      ,[GenderId]
         , [age]
         ,[city]
  FROM [sample2].[dbo].[tb1Person]
  -- here we are giving fully qualified name


insert into tb1Person
values
(4, 'Foram','foram@mail.com',2,21,'Rajkot')


--get distint column values

select distinct name, city from tb1Person

select name from tb1person where city='Florida'
select name from tb1person where city!='Florida'


select name from tb1person where age in (21,25,42)
select name from tb1person where age between 20 and 50


select * from tb1Person where email like '%@%'

--Starting or ending with single character
```

```sql
select * from tb1Person where email like '_@_'

--starting with some character like mst
select * from tb1Person where email like '[mst]%'
--also we can use not in here
select * from tb1Person where email like '[^mst]%'


--sorting and usage if top

select top 1 * from tb1Person order by age desc
```



- [Part 11 – Group By]

GROUP BY

ID	Name	Gender	Salary	City
1	Tom	Male	4000	London
2	Pam	Female	3000	New York
3	John	Male	3500	London
4	Sam	Male	4500	London
5	Todd	Male	2800	Sydney
6	Ben	Male	7000	New York
7	Sara	Female	4800	Sydney
8	Valarie	Female	5500	New York
9	James	Male	6500	London
10	Russell	Male	8800	London

Group by clause is used to group a selected set of rows into a set of summary rows by the values of one or more columns or expressions. It is always used in conjunction with one or more aggregate functions.

City	TotalSalary
London	27300
New York	15500
Sydney	7600

```
SELECT City, SUM(Salary) AS TotalSalary
FROM tblEmployee GROUP BY City
```

Note: If you omit, the GROUP BY clause and try to execute the query, you get an error **Column 'tblEmployee.City' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.**

Filtering Groups

GROUPING BY multiple columns

City	Gender	TotalSalary
London	Male	27300
New York	Female	8500
New York	Male	7000
Sydney	Female	4800
Sydney	Male	2800

Using multiple aggregate functions

City	Gender	TotalSalary	TotalEmployees
London	Male	27300	5
New York	Female	8500	2
New York	Male	7000	1
Sydney	Female	4800	1
Sydney	Male	2800	1

Filtering Groups:

WHERE clause is used to filter rows before aggregation, where as HAVING clause is used to filter groups after aggregations. The following 2 queries produce the same result.

```
Select City, SUM(Salary) as TotalSalary
from tblEmployee
Where City = 'London'
group by City
```

```
Select City, SUM(Salary) as TotalSalary
from tblEmployee
group by City
Having City = 'London'
```

Note: From a performance standpoint, you cannot say that one method is less efficient than the other. Sql server optimizer analyses each statement and selects an efficient way of executing it. As a best practice, use the syntax that clearly describes the desired result. Try to eliminate rows that you wouldn't need, as early as possible.

Difference – WHERE and Having

1. WHERE clause can be used with - Select, Insert, and Update statements, where as HAVING clause can only be used with the Select statement.

2. WHERE filters rows before aggregation (GROUPING), where as, HAVING filters groups, after the aggregations are performed.

3. Aggregate functions cannot be used in the WHERE clause, unless it is in a sub query contained in a HAVING clause, whereas, aggregate functions can be used in Having clause.

Part 12 – JOINS

Joins in SQL Server

Joins in SQL Server are used to retrieve data from 2 or more related tables. In general tables are related to each other using foreign key constraints.

In SQL server, there are different types of JOINS

1. INNER JOIN
2. OUTER JOIN
3. CROSS JOIN

Outer Joins are again divided into

1. Left Join or Left Outer Join
2. Right Join or Right Outer Join
3. Full Join or Full Outer Join

Name	Gender	Salary	DepartmentName
Tom	Male	4000	IT
Pam	Female	3000	HR
John	Male	3500	IT
Sam	Male	4500	Payroll
Todd	Male	2800	Payroll
Ben	Male	7000	IT
Sara	Female	4800	HR
Valarie	Female	5500	IT

ID	Name	Gender	Salary	DepartmentId
1	Tom	Male	4000	1
2	Pam	Female	3000	3
3	John	Male	3500	1
4	Sam	Male	4500	2
5	Todd	Male	2800	2
6	Ben	Male	7000	1
7	Sara	Female	4800	3
8	Valarie	Female	5500	1
9	James	Male	6500	NULL
10	Russell	Male	8800	NULL

Id	DepartmentName	Location	DepartmentHead
1	IT	London	Rick
2	Payroll	Delhi	Ron
3	HR	New York	Christie
4	Other Department	Sydney	Cindrella

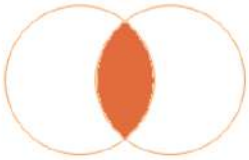
Inner Join or Join

ID	Name	Gender	Salary	DepartmentId
1	Tom	Male	4000	1
2	Pam	Female	3000	3
3	John	Male	3500	1
4	Sam	Male	4500	2
5	Todd	Male	2800	2
6	Ben	Male	7000	1
7	Sara	Female	4800	3
8	Valarie	Female	5500	1
9	James	Male	6500	NULL
10	Russell	Male	8800	NULL

Id	DepartmentName	Location	DepartmentHead
1	IT	London	Rick
2	Payroll	Delhi	Ron
3	HR	New York	Christie
4	Other Department	Sydney	Cindrella

INNER JOIN or JOIN

Matching Rows only – Non matching rows eliminated



Employees - Departments

Name	Gender	Salary	DepartmentName
Tom	Male	4000	IT
Pam	Female	3000	HR
John	Male	3500	IT
Sam	Male	4500	Payroll
Todd	Male	2800	Payroll
Ben	Male	7000	IT
Sara	Female	4800	HR
Valarie	Female	5500	IT

INNER JOIN returns only the matching rows between both the tables. Non matching rows are eliminated.

```
SELECT Name, Gender, Salary, DepartmentName
FROM tblEmployee
INNER JOIN tblDepartment
ON tblEmployee.DepartmentId = tblDepartment.Id
```

OR

```
SELECT Name, Gender, Salary, DepartmentName
FROM tblEmployee
JOIN tblDepartment
ON tblEmployee.DepartmentId = tblDepartment.Id
```

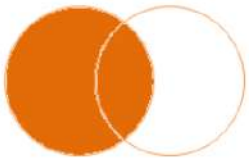

Left Outer Join or Left Join

ID	Name	Gender	Salary	DepartmentId
1	Tom	Male	4000	1
2	Pam	Female	3000	3
3	John	Male	3500	1
4	Sam	Male	4500	2
5	Todd	Male	2800	2
6	Ben	Male	7000	1
7	Sara	Female	4800	3
8	Valarie	Female	5500	1
9	James	Male	6500	NULL
10	Russell	Male	8800	NULL

Id	DepartmentName	Location	DepartmentHead
1	IT	London	Rick
2	Payroll	Delhi	Ron
3	HR	New York	Christie
4	Other Department	Sydney	Cindrella

LEFT OUTER JOIN or LEFT JOIN

Matching Rows + Non matching rows from the left table



Employees - Departments

Name	Gender	Salary	DepartmentName
Name	Gender	Salary	DepartmentName
Tom	Male	4000	IT
Pam	Female	3000	HR
John	Male	3500	IT
Sam	Male	4500	Payroll
Todd	Male	2800	Payroll
Ben	Male	7000	IT
Sara	Female	4800	HR
Valarie	Female	5500	IT
James	Male	6500	NULL
Russell	Male	8800	NULL

LEFT JOIN returns all the matching rows + non matching rows from the left table.

```
SELECT Name, Gender, Salary, DepartmentName
FROM tblEmployee
LEFT OUTER JOIN tblDepartment
ON tblEmployee.DepartmentId = tblDepartment.Id
```

OR

```
SELECT Name, Gender, Salary, DepartmentName
FROM tblEmployee
LEFT JOIN tblDepartment
ON tblEmployee.DepartmentId = tblDepartment.Id
```

Right Outer Join or Right Join

ID	Name	Gender	Salary	DepartmentId
1	Tom	Male	4000	1
2	Pam	Female	3000	3
3	John	Male	3500	1
4	Sam	Male	4500	2
5	Todd	Male	2800	2
6	Ben	Male	7000	1
7	Sara	Female	4800	3
8	Valarie	Female	5500	1
9	James	Male	6500	NULL
10	Russell	Male	8800	NULL

Id	DepartmentName	Location	DepartmentHead
1	IT	London	Rick
2	Payroll	Delhi	Ron
3	HR	New York	Christie
4	Other Department	Sydney	Cindrella

RIGHT OUTER JOIN or RIGHT JOIN

Matching Rows + Non matching rows from the right table

Employees - Departments

Name	Gender	Salary	DepartmentName
Tom	Male	4000	IT
John	Male	3500	IT
Ben	Male	7000	IT
Valarie	Female	5500	IT
Sam	Male	4500	Payroll
Todd	Male	2800	Payroll
Pam	Female	3000	HR
Sara	Female	4800	HR
NULL	NULL	NULL	Other Department

RIGHT JOIN returns all the matching rows + non matching rows from the right table

```
SELECT Name, Gender, Salary, DepartmentName
FROM tblEmployee
RIGHT OUTER JOIN tblDepartment
ON tblEmployee.DepartmentId = tblDepartment.Id
```

OR

```
SELECT Name, Gender, Salary, DepartmentName
FROM tblEmployee
RIGHT JOIN tblDepartment
ON tblEmployee.DepartmentId = tblDepartment.Id
```


Full Outer Join or Full Join

ID	Name	Gender	Salary	DepartmentId
1	Tom	Male	4000	1
2	Pam	Female	3000	3
3	John	Male	3500	1
4	Sam	Male	4500	2
5	Todd	Male	2800	2
6	Ben	Male	7000	1
7	Sara	Female	4800	3
8	Valarie	Female	5500	1
9	James	Male	6500	NULL
10	Russell	Male	8800	NULL

Id	DepartmentName	Location	DepartmentHead
1	IT	London	Rick
2	Payroll	Delhi	Ron
3	HR	New York	Christie
4	Other Department	Sydney	Cindrella

FULL OUTER JOIN or FULL JOIN

Matching Rows + Non matching rows from both the tables

Employees - Departments

Name	Gender	Salary	DepartmentName
Tom	Male	4000	IT
Pam	Female	3000	HR
John	Male	3500	IT
Sam	Male	4500	Payroll
Todd	Male	2800	Payroll
Ben	Male	7000	IT
Sara	Female	4800	HR
Valarie	Female	5500	IT
James	Male	6500	NULL
Russell	Male	8800	NULL
NULL	NULL	NULL	Other Department

FULL JOIN returns all rows from both the left and right tables, including the non matching rows.

```
SELECT Name, Gender, Salary, DepartmentName
FROM tblEmployee
FULL OUTER JOIN tblDepartment
ON tblEmployee.DepartmentId = tblDepartment.Id
```

OR

```
SELECT Name, Gender, Salary, DepartmentName
FROM tblEmployee
FULL JOIN tblDepartment
ON tblEmployee.DepartmentId = tblDepartment.Id
```

Cross Join

CROSS JOIN, produces the Cartesian product of the 2 tables involved in the join. For example, in the Employees table we have 10 rows and in the Departments table we have 4 rows. So, a cross join between these 2 tables produces 40 rows.

NOTE: Cross Join shouldn't have ON clause.

General Formula for Joins:

```
SELECT        COLUMN_LIST
FROM          LEFT_TABLE_NAME
JOIN_TYPE     RIGHT_TABLE_NAME
ON            JOIN_CONDITION
```

Cross Join:

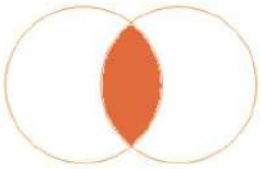
```
SELECT       Name, Gender, Salary, DepartmentName
FROM         tblEmployee
CROSS JOIN   tblDepartment
```

Summary

Join Type	Purpose
Cross Join	Returns Cartesian product of the tables involved in the join
Inner Join	Returns only the matching rows. Non matching rows are eliminated.
Left Join	Returns all the matching rows + non matching rows from the left table
Right Join	Returns all the matching rows + non matching rows from the right table
Full Join	Returns all rows from both tables, including the non-matching rows.

INNER JOIN or JOIN

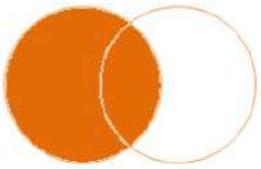
Matching Rows only – Non matching rows eliminated



Employees - Departments

LEFT OUTER JOIN or LEFT JOIN

Matching Rows + Non matching rows from the left table



Employees - Departments

RIGHT OUTER JOIN or RIGHT JOIN

Matching Rows + Non matching rows from the right table

Employees - Departments

FULL OUTER JOIN or FULL JOIN

Matching Rows + Non matching rows from both the tables

Employees - Departments

Part 13 – Advanced JOINS

Advanced or Intelligent Joins

ID	Name	Gender	Salary	DepartmentId
1	Tom	Male	4000	1
2	Pam	Female	3000	3
3	John	Male	3500	1
4	Sam	Male	4500	2
5	Todd	Male	2800	2
6	Ben	Male	7000	1
7	Sara	Female	4800	3
8	Valarie	Female	5500	1
9	James	Male	6500	NULL
10	Russell	Male	8800	NULL

Id	DepartmentName	Location	DepartmentHead
1	IT	London	Rick
2	Payroll	Delhi	Ron
3	HR	New York	Christie
4	Other Department	Sydney	Cindrella

Name	Gender	Salary	DepartmentName
James	Male	6500	NULL
Russell	Male	8800	NULL

```
SELECT      [Column_List]
FROM        tblEmployee E
LEFT JOIN   tblDepartments D
ON          E.DeptId = D.Id
WHERE       D.Id IS NULL
```

Name	Gender	Salary	DepartmentName
NULL	NULL	NULL	Other Department

```
SELECT      [Column_List]
FROM        tblEmployee E
RIGHT JOIN  tblDepartments D
ON          E.DeptId = D.Id
WHERE       E.DeptId IS NULL
```

Name	Gender	Salary	DepartmentName
James	Male	6500	NULL
Russell	Male	8800	NULL
NULL	NULL	NULL	Other Department

```
SELECT      [Column_List]
FROM        tblEmployee E
FULL JOIN   tblDepartments D
ON          E.DeptId = D.Id
WHERE       E.DeptId IS NULL
OR          D.Id IS NULL
```

Part 14 – Self Join

Self Join

In parts 12 and 13, we have seen joining 2 different tables - tblEmployees and tblDepartments. Have you ever thought of a need to join a table with itself.

EmployeeID	Name	ManagerID
1	Mike	3
2	Rob	1
3	Todd	NULL
4	Ben	1
5	Sam	1

Employee	Manager
Mike	Todd
Rob	Mike
Todd	NULL
Ben	Mike
Sam	Mike

```
-- Left Outer Self Join
SELECT      E.Name AS Employee, M.Name AS Manager
FROM        tblEmployee E
LEFT JOIN   tblEmployee M
ON          E.ManagerId = M.EmployeeId
```

```
-- Inner Self Join
SELECT      E.Name AS Employee, M.Name AS Manager
FROM        tblEmployee E
INNER JOIN  tblEmployee M
ON          E.ManagerId = M.EmployeeId
```

```
-- Cross Self Join
SELECT      E.Name AS Employee, M.Name AS Manager
FROM        tblEmployee E
CROSS JOIN  tblEmployee M
```

Joining a table with itself is called as SELF JOIN.

SELF JOIN is not a different type of JOIN.

It can be classified under any type of JOIN:

1. INNER,
2. OUTER (Left, Right, Full)
3. CROSS Joins.

Part 15 – Different ways to replace NULL in SQL Server

Replacing NULL Values

Three ways to replace NULL values - ISNULL() Function, CASE Statement & COALESCE() Function

EmployeeID	Name	ManagerID
1	Mike	3
2	Rob	1
3	Todd	NULL
4	Ben	1
5	Sam	1

Employee	Manager
Mike	Todd
Rob	Mike
Todd	NULL
Ben	Mike
Sam	Mike

Employee	Manager
Mike	Todd
Rob	Mike
Todd	No Manager
Ben	Mike
Sam	Mike

```
SELECT      E.Name as Employee, ISNULL(M.Name, 'No Manager') as Manager
FROM        tblEmployee E
LEFT JOIN   tblEmployee M
ON          E.ManagerID = M.EmployeeID
```

```
SELECT      E.Name as Employee,
            CASE
                WHEN M.Name IS NULL THEN 'No Manager' ELSE M.Name
            END
            as Manager
FROM        tblEmployee E
LEFT JOIN   tblEmployee M
ON          E.ManagerID = M.EmployeeID
```

```
SELECT      E.Name as Employee, COALESCE(M.Name, 'No Manager') as Manager
FROM        tblEmployee E
LEFT JOIN   tblEmployee M
ON          E.ManagerID = M.EmployeeID
```

Part 16 – COALESCE() function in SQL Server

COALESCE

COALESCE() Function – Returns the first NON NULL value

Id	FirstName	MiddleName	LastName
1	Sam	NULL	NULL
2	NULL	Todd	Tanzan
3	NULL	NULL	Sara
4	Ben	Parker	NULL
5	James	Nick	Nancy

Id	Name
1	Sam
2	Todd
3	Sara
4	Ben
5	James

```
SELECT Id, COALESCE(FirstName, MiddleName, LastName) as Name
FROM   tblEmployee
```

Part 17 – UNION and UNION ALL

UNION & UNION ALL

UNION and UNION ALL operators in SQL Server, are used to combine the result-set of two or more SELECT queries.

tblIndiaCustomers

Id	Name	Email
1	Raj	R@R.com
2	Sam	S@S.com

tblUKCustomers

Id	Name	Email
1	Ben	B@B.com
2	Sam	S@S.com

```
SELECT Id, Name, Email FROM tblIndiaCustomers
UNION
SELECT Id, Name, Email FROM tblUKCustomers
```

UNION →

Id	Name	Email
1	Ben	B@B.com
1	Raj	R@R.com
2	Sam	S@S.com

```
SELECT Id, Name, Email FROM tblIndiaCustomers
UNION ALL
SELECT Id, Name, Email FROM tblUKCustomers
```

UNION ALL →

Id	Name	Email
1	Raj	R@R.com
2	Sam	S@S.com
1	Ben	B@B.com
2	Sam	S@S.com

Note: For UNION and UNION ALL to work, the Number, Data types, and the order of the columns in the select statements should be same.

UNION & UNION ALL & JOINS

Difference between UNION and UNION ALL

1. UNION removes duplicate rows, where as UNION ALL does not
2. UNION has to perform distinct sort to remove duplicates, which makes it less faster than UNION ALL

Note: Estimated query execution plan - CTRL + L

Sorting results of a UNION or UNION ALL

ORDER BY clause should be used only on the last SELECT statement in the UNION query

Difference between UNION and JOIN

UNION combines the result-set of two or more select queries into a single result-set which includes all the rows from all the queries in the union, where as JOINS, retrieve data from two or more tables based on logical relationships between the tables.

In short, UNION combines rows from 2 or more tables, where JOINS combine columns from 2 or more table.

Part 18 – Stored Procedures

Stored Procedures

A stored procedure is group of T-SQL (Transact SQL) statements. If you have a situation, where you write the same query over and over again, you can save that specific query as a stored procedure and call it just by it's name.

1. Use CREATE PROCEDURE or CREATE PROC statement to create SP
Note: When naming user defined stored procedures, Microsoft recommends not to use sp_ as a prefix. All system stored procedures, are prefixed with sp_. This avoids any ambiguity between user defined and system stored procedures and any conflicts, with some future system procedure.

To execute the stored procedure
1. spGetEmployees
2. EXEC spGetEmployees
3. Execute spGetEmployees

Note: You can also right click on the procedure name, in object explorer in SQL Server Management Studio and select EXECUTE STORED PROCEDURE.

Stored Procedure with Parameters

Parameters and variables have an @ prefix in their name.

To Execute:
EXECUTE spGetEmployeesByGenderAndDepartment 'Male', 1
EXECUTE spGetEmployeesByGenderAndDepartment @DepartmentId=1, @Gender = 'Male'

To view the text, of the stored procedure
1. Use system stored procedure sp_helptext 'SPName'
OR
2. Right Click the SP in Object explorer -> Scrip Procedure as -> Create To -> New Query Editor Window

To change the stored procedure, use ALTER PROCEDURE statement.

To delete the SP, use DROP PROC 'SPName' or DROP PROCEDURE 'SPName'

To encrypt the text of the SP, use WITH ENCRYPTION option. It is not possible to view the text of an encrypted SP.

Next Session – Creating and Invoking a stored procedure with out put parameters

Part 19 – Stored procedures with output parameters

Stored Procedures – Output Parameters

To create an SP with output parameter, we use the keywords OUT or OUTPUT.

Id	Name	Gender	DepartmentId
1	Sam	Male	1
2	Ram	Male	1
3	Sara	Female	3
4	Todd	Male	2
5	John	Male	3
6	Sana	Female	2
7	James	Male	1
8	Rob	Male	2
9	Steve	Male	1
10	Pam	Female	2

```
Create Procedure spGetEmployeeCountByGender
@Gender nvarchar(20),
@EmployeeCount int Output
as
Begin
    Select @EmployeeCount = COUNT(Id)
    from tblEmployee
    where Gender = @Gender
End
```

To execute the stored procedure with output parameters

```
Declare @EmployeeTotal int
Execute spGetEmployeeCountByGender 'Male', @EmployeeTotal Output
Print   @EmployeeTotal
```

```
Declare @EmployeeTotal int
Execute spGetEmployeeCountByGender @EmployeeCount = @EmployeeTotal OUT, @Gender = 'Male'
Print   @EmployeeTotal
```

If you don't specify the OUTPUT keyword, when executing the stored procedure, the @EmployeeTotal variable will be NULL.

Useful system stored procedures

sp_help procedure_name - View the information about the stored procedure, like parameter names, their datatypes etc. sp_help can be used with any database object, like tables, views, SP's, triggers etc. Alternatively, you can also press ALT+F1, when the name of the object is highlighted.

sp_helptext procedure_name - View the Text of the stored procedure

sp_depends procedure_name - View the dependencies of the stored procedure. This system SP is very useful, especially if you want to check, if there are any stored procedures that are referencing a table that you are abput to drop. sp_depends can also be used with other database objects like table etc.

Part 20 - Stored procedure output parameters or return values

Output Parameters or Return Values

Whenever, you execute a stored procedure, it returns an integer status variable. Usually, zero indicates success, and non-zero indicates failure.

Id	Name	Gender	DepartmentId
1	Sam	Male	1
2	Ram	Male	1
3	Sara	Female	3
4	Todd	Male	2
5	John	Male	3
6	Sana	Female	2
7	James	Male	1
8	Rob	Male	2
9	Steve	Male	1
10	Pam	Female	2

```
Create Procedure spGetTotalCountOfEmployees1
@TotalCount int output
as
Begin
   Select @TotalCount = COUNT(ID) from tblEmployee
End
```

```
Declare @TotalEmployees int
Execute spGetTotalCountOfEmployees @TotalEmployees Output
Select  @TotalEmployees
```

```
Create Procedure spGetTotalCountOfEmployees2
as
Begin
  return (Select COUNT(ID) from Employees)
End
```

```
Declare @TotalEmployees int
Execute @TotalEmployees = spGetTotalCountOfEmployees2
Select  @TotalEmployees
```

So, we are able to achieve what we want, using output parameters as well as return values.

Now, let's look at example, where return status variables cannot be used, but Output parameters can be used.

Output Parameters or Return Values

Id	Name	Gender	DepartmentId
1	Sam	Male	1
2	Ram	Male	1
3	Sara	Female	3
4	Todd	Male	2
5	John	Male	3
6	Sana	Female	2
7	James	Male	1
8	Rob	Male	2
9	Steve	Male	1
10	Pam	Female	2

```
Create Procedure spGetNameById1
@Id int,
@Name nvarchar(20) Output
as
Begin
  Select @Name = Name from tblEmployee Where Id = @Id
End
```

```
Declare @EmployeeName nvarchar(20)
Execute spGetNameById1 3, @EmployeeName out
Print   'Name of the Employee = ' + @EmployeeName
```

```
Create Procedure spGetNameById2
@Id int
as
Begin
  Return (Select Name from tblEmployee Where Id = @Id)
End
```

```
Declare @EmployeeName nvarchar(20)
Execute @EmployeeName = spGetNameById2 1
Print   'Name of the Employee = ' + @EmployeeName
```

Executing spGetNameById2 returns an error stating 'Conversion failed when converting the nvarchar value 'Sam' to data type int.'

Output Parameters or Return Values

Return Staus Value	Output Parameters
Only Integer Datatype	Any Datatype
Only one value	More than value
Use to convey success or failure	Use to return values like name, count etc..

- [Part 21 – Advantages of stored procedures]

Advantages of Stored Procedures

1. Execution plan retention and reusability
2. Reduces network traffic
3. Code reusability and better maintainability
4. Better Security
5. Avoids SQL Injection attack

What is SQL Injection attack and how to prevent it?
http://www.youtube.com/watch?v=uSw0IoSr3Hk

Part 22 – String functions

String functions

Function	Purpose
ASCII(Character_Expression)	Returns the ASCII code of the given character expression.
CHAR(Integer_Expression)	Converts an int ASCII code to a character. The Integer_Expression, should be between 0 and 255.
LTRIM(Character_Expression)	Removes blanks on the left handside of the given character expression.
RTRIM(Character_Expression)	Removes blanks on the right hand side of the given character expression.
LOWER(Character_Expression)	Converts all the characters in the given Character_Expression, to lowercase letters.
UPPER(Character_Expression)	Converts all the characters in the given Character_Expression, to uppercase letters.
REVERSE('Any_String_Expression')	Reverses all the characters in the given string expression.
LEN(String_Expression)	Returns the count of total characters, in the given string expression, excluding the blanks at the end of the expression.

In the next video session - Rest of the commonly used built-in string functions

Part 23 – String functions Continued

String functions

Function	Purpose
LEFT(Character_Expression, Integer_Expression)	Returns the specified number of characters from the left hand side of the given character expression.
RIGHT(Character_Expression, Integer_Expression)	Returns the specified number of characters from the right hand side of the given character expression.
CHARINDEX('Expression_To_Find', 'Expression_To_Search', 'Start_Location')	Returns the starting position of the specified expression in a character string
SUBSTRING('Expression', 'Start', 'Length')	Returns substring (part of the string), from the given expression

Id	FirstName	LastName	Email
1	Sam	Sony	Sam@aaa.com
2	Ram	Barber	Ram@aaa.com
3	Sara	Sanosky	Sara@ccc.com
4	Todd	Gartner	Todd@bbb.com
5	John	Grover	John@aaa.com
6	Sana	Lenin	Sana@ccc.com
7	James	Bond	James@bbb.com
8	Rob	Hunter	Rob@ccc.com
9	Steve	Wilson	Steve@aaa.com
10	Pam	Broker	Pam@bbb.com

EmailDomain	Total
aaa.com	4
bbb.com	3
ccc.com	3

```
Select SUBSTRING(Email,CHARINDEX('@',Email)+1,
(LEN(Email) - CHARINDEX('@',Email))) as EmailDomain,
Count(*) as Total
From tblEmployee
Group By SUBSTRING(Email,CHARINDEX('@',Email)+1,
(LEN(Email) - CHARINDEX('@',Email)))
```

Part 24 – String functions Continued

REPLICATE Function

REPLICATE(String_To__Be_Replicated, Number_Of_Times_To_Replicate)
Repeats the given string, for the specified number of times.

FirstName	LastName	Email
Sam	Sony	Sam@aaa.com
Ram	Barber	Ram@aaa.com
Sara	Sanosky	Sara@ccc.com
Todd	Gartner	Todd@bbb.com
John	Grover	John@aaa.com
Sana	Lenin	Sana@ccc.com
James	Bond	James@bbb.com
Rob	Hunter	Rob@ccc.com
Steve	Wilson	Steve@aaa.com
Pam	Broker	Pam@bbb.com

→

FirstName	LastName	Email
Sam	Sony	Sa*****@aaa.com
Ram	Barber	Ra*****@aaa.com
Sara	Sanosky	Sa*****@ccc.com
Todd	Gartner	To*****@bbb.com
John	Grover	Jo*****@aaa.com
Sana	Lenin	Sa*****@ccc.com
James	Bond	Ja*****@bbb.com
Rob	Hunter	Ro*****@ccc.com
Steve	Wilson	St*****@aaa.com
Pam	Broker	Pa*****@bbb.com

Mask the email with 5 * (star) symbols

```
Select  FirstName, LastName,
        SUBSTRING(Email, 1, 2) + REPLICATE('*',5) +
        SUBSTRING(Email, CHARINDEX('@',Email), LEN(Email) - CHARINDEX('@',Email)+1) as Email
from    tblEmployee
```

SPACE Function

SPACE(Number_Of_Spaces)
Returns number of spaces, specified by the Number_Of_Spaces argument.

FirstName	LastName	Email
Sam	Sony	Sam@aaa.com
Ram	Barber	Ram@aaa.com
Sara	Sanosky	Sara@ccc.com
Todd	Gartner	Todd@bbb.com
John	Grover	John@aaa.com
Sana	Lenin	Sana@ccc.com
James	Bond	James@bbb.com
Rob	Hunter	Rob@ccc.com
Steve	Wilson	Steve@aaa.com
Pam	Broker	Pam@bbb.com

→

FullName
Sam Sony
Ram Barber
Sara Sanosky
Todd Gartner
John Grover
Sana Lenin
James Bond
Rob Hunter
Steve Wilson
Pam Broker

The SPACE(5) function, inserts 5 spaces between FirstName and LastName

```
Select  FirstName + SPACE(5) + LastName as FullName
From    tblEmployee
```

PATINDEX Function

PATINDEX('%Pattern%', Expression)
Returns the starting position of the first occurrence of a pattern in a specified expression. It takes two arguments, the pattern to be searched and the expression. PATINDEX() is simial to CHARINDEX(). With CHARINDEX() we cannot use wildcards, where as PATINDEX() provides this capability. If the specified pattern is not found, PATINDEX() returns ZERO.

FirstName	LastName	Email
Sam	Sony	Sam@aaa.com
Ram	Barber	Ram@aaa.com
Sara	Sanosky	Sara@ccc.com
Todd	Gartner	Todd@bbb.com
John	Grover	John@aaa.com
Sana	Lenin	Sana@ccc.com
James	Bond	James@bbb.com
Rob	Hunter	Rob@ccc.com
Steve	Wilson	Steve@aaa.com
Pam	Broker	Pam@bbb.com

→

Email	FirstOccurence
Sam@aaa.com	4
Ram@aaa.com	4
John@aaa.com	5
Steve@aaa.com	6

```
Select   Email, PATINDEX('%@aaa.com', Email) as FirstOccurence
from     tblEmployee
Where    PATINDEX('%@aaa.com', Email) > 0
```

REPLACE Function

REPLACE(String_Expression, Pattern , Replacement_Value)
Replaces all occurrences of a specified string value with another string value.

FirstName	LastName	Email
Sam	Sony	Sam@aaa.com
Ram	Barber	Ram@aaa.com
Sara	Sanosky	Sara@ccc.com
Todd	Gartner	Todd@bbb.com
John	Grover	John@aaa.com
Sana	Lenin	Sana@ccc.com
James	Bond	James@bbb.com
Rob	Hunter	Rob@ccc.com
Steve	Wilson	Steve@aaa.com
Pam	Broker	Pam@bbb.com

→

Email	ConvertedEmail
Sam@aaa.com	Sam@aaa.net
Ram@aaa.com	Ram@aaa.net
Sara@ccc.com	Sara@ccc.net
Todd@bbb.com	Todd@bbb.net
John@aaa.com	John@aaa.net
Sana@ccc.com	Sana@ccc.net
James@bbb.com	James@bbb.net
Rob@ccc.com	Rob@ccc.net
Steve@aaa.com	Steve@aaa.net
Pam@bbb.com	Pam@bbb.net

All .COM strings are replaced with .NET

```
Select   Email, REPLACE(Email, '.com', '.net') as ConvertedEmail
from     tblEmployee
```

STUFF Function

STUFF(Original_Expression, Start, Length, Replacement_expression)
STUFF() function inserts Replacement_expression, at the start position specified, along with removing the charactes specified using Length parameter.

FirstName	LastName	Email
Sam	Sony	Sam@aaa.com
Ram	Barber	Ram@aaa.com
Sara	Sanosky	Sara@ccc.com
Todd	Gartner	Todd@bbb.com
John	Grover	John@aaa.com
Sana	Lenin	Sana@ccc.com
James	Bond	James@bbb.com
Rob	Hunter	Rob@ccc.com
Steve	Wilson	Steve@aaa.com
Pam	Broker	Pam@bbb.com

→

FirstName	LastName	Email	StuffedEmail
Sam	Sony	Sam@aaa.com	S*****aaa.com
Ram	Barber	Ram@aaa.com	R*****aaa.com
Sara	Sanosky	Sara@ccc.com	S*****@ccc.com
Todd	Gartner	Todd@bbb.com	T*****@bbb.com
John	Grover	John@aaa.com	J*****@aaa.com
Sana	Lenin	Sana@ccc.com	S*****@ccc.com
James	Bond	James@bbb.com	J*****s@bbb.com
Rob	Hunter	Rob@ccc.com	R*****ccc.com
Steve	Wilson	Steve@aaa.com	S*****e@aaa.com
Pam	Broker	Pam@bbb.com	P*****bbb.com

```
Select   FirstName, LastName,Email,
         STUFF(Email, 2, 3, '*****') as StuffedEmail
From     tblEmployee
```

Part 25 – DateTime functions in SQL Server

DatePart, DateAdd and DateDiff

DatePart(DatePart, Date) - Returns an integer representing the specified DatePart. This function is simialar to DateName(). DateName() returns nvarchar, where as DatePart() returns an integer.

```
--Examples:
Select DATEPART(weekday, '2012-08-30 19:45:31.793') -- returns 5
SELECt DATENAME(weekday, '2012-08-30 19:45:31.793') -- returns Thursday
```

DATEADD (datepart, NumberToAdd, date) - Returns the DateTime, after adding specified NumberToAdd, to the datepart specified of the given date.

```
Examples:
Select DateAdd(DAY, 20, '2012-08-30 19:45:31.793')
-- Returns 2012-09-19 19:45:31.793
Select DateAdd(DAY, -20, '2012-08-30 19:45:31.793')
-- Returns 2012-08-10 19:45:31.793
```

DATEDIFF(datepart, startdate, enddate) - Returns the count of the specified datepart boundaries crossed between the specified startdate and enddate.

```
--Examples:
Select DATEDIFF(MONTH, '11/30/2005','01/31/2006') -- returns 2
Select DATEDIFF(DAY, '11/30/2005','01/31/2006') -- returns 62
```

DatePart	Abbreviation
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dw
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
microsecond	mcs
nanosecond	ns
TZoffset	tz

Calculating Age

Id	Name	DateOfBirth
1	Sam	1980-12-30 00:00:00.000
2	Pam	1982-09-01 12:02:36.260
3	John	1985-08-22 12:03:30.370
4	Sara	1979-11-29 12:59:30.670

Id	Name	DateOfBirth	DOB
1	Sam	1980-12-30 00:00:00.000	31 Years 8 Months 2 Days old
2	Pam	1982-09-01 12:02:36.260	30 Years 0 Months 0 Days old
3	John	1985-08-22 12:03:30.370	27 Years 0 Months 10 Days old
4	Sara	1979-11-29 12:59:30.670	32 Years 9 Months 3 Days old

```
DECLARE @DOB datetime, @tmpdate datetime, @years int, @months int, @days int
SET @DOB = '10/08/1982'

SELECT @tmpdate = @DOB

SELECT @years = DATEDIFF(YEAR, @tmpdate, GETDATE()) -
                CASE
                    WHEN (MONTH(@DOB) > MONTH(GETDATE())) OR
                    (MONTH(@DOB) = MONTH(GETDATE()) AND DAY(@DOB) > DAY(GETDATE()))
                    THEN 1 ELSE 0
                END
SELECT @tmpdate = DATEADD(YEAR, @years, @tmpdate)

SELECT @months = DATEDIFF(MONTH, @tmpdate, GETDATE()) -
                CASE
                    WHEN DAY(@DOB) > DAY(GETDATE())
                    THEN 1 ELSE 0
                END
SELECT @tmpdate = DATEADD(MONTH, @months, @tmpdate)

SELECT @days = DATEDIFF(DAY, @tmpdate, GETDATE())

SELECT @years as Years, @months as Months, @days as [Days]
```

Part 26 –DateTime functions continued

- [Part 27 –DateTime functions continued]
- [Part 28 –Cast and Convert functions]

CAST & CONVERT

To convert one data type to another, CAST and CONVERT functions can be used.

```
--Syntax of CAST and CONVERT functions from MSDN:
CAST ( expression AS data_type [ ( length ) ] )
CONVERT ( data_type [ ( length ) ] , expression [ , style ] )
```

Id	Name	DateOfBirth
1	Sam	1980-12-30 00:00:00.000
2	Pam	1982-09-01 12:02:36.260
3	John	1985-08-22 12:03:30.370
4	Sara	1979-11-29 12:59:30.670

Id	Name	DateOfBirth	ConvertedDOB
1	Sam	1980-12-30 00:00:00.000	Dec 30 1980 12:00AM
2	Pam	1982-09-01 12:02:36.260	Sep 1 1982 12:02PM
3	John	1985-08-22 12:03:30.370	Aug 22 1985 12:03PM
4	Sara	1979-11-29 12:59:30.670	Nov 29 1979 12:59PM

```
Select Id, Name, DateOfBirth, CAST(DateofBirth as nvarchar) as ConvertedDOB from tblEmployees
Select Id, Name, DateOfBirth, Convert(nvarchar, DateOfBirth) as ConvertedDOB from tblEmployees
```

```
Select Id, Name, DateOfBirth,
Convert(nvarchar, DateOfBirth, 103) as ConvertedDOB
from tblEmployees
```

Id	Name	DateOfBirth	ConvertedDOB
1	Sam	1980-12-30 00:00:00.000	30/12/1980
2	Pam	1982-09-01 12:02:36.260	01/09/1982
3	John	1985-08-22 12:03:30.370	22/08/1985
4	Sara	1979-11-29 12:59:30.670	29/11/1979

Style	DateFormat
101	mm/dd/yyyy
102	yy.mm.dd
103	dd/mm/yyyy
104	dd.mm.yy
105	dd-mm-yy

http://msdn.microsoft.com/en-us/library/ms187928.aspx

DatePart of DateTime

```
--To get just the date part, from DateTime
SELECT CONVERT(VARCHAR(10),GETDATE(),101) -- Returns 09/02/2012

--In SQL Server 2008, Date datatype is introduced, so you can also use
SELECT CAST(GETDATE() as DATE)    -- 2012-09-02
SELECT CONVERT(DATE, GETDATE())   -- 2012-09-02
```

Note: To control the formatting of the Date part, DateTime has to be converted to NVARCHAR using the styles provided. When converting to DATE data type, the CONVERT() function will ignore the style parameter.

```
--Concatenate Id(INT) and Name(NVARCHAR)
Select Id, Name, Name + ' - ' + CAST(Id AS NVARCHAR) AS [Name-Id] FROM tblEmployees
```

Id	Name	Name-Id
1	Sam	Sam - 1
2	Pam	Pam - 2
3	John	John - 3
4	Sara	Sara - 4

Practical Example

Id	Name	Email	RegisteredDate
1	John	j@j.com	2012-08-24 11:04:30.230
2	Sam	s@s.com	2012-08-25 14:04:29.780
3	Todd	t@t.com	2012-08-25 15:04:29.780
4	Mary	m@m.com	2012-08-24 15:04:30.730
5	Sunil	sunil@s.com	2012-08-24 15:05:30.330
6	Mike	mike@m.com	2012-08-26 15:05:30.330

RegistrationDate	TotalRegistrations
2012-08-24	3
2012-08-25	2
2012-08-26	1

```
Select      CAST(RegisteredDate as DATE)as RegistrationDate, COUNT(Id) as TotalRegistrations
from        tblRegistrations
Group BY    CAST(RegisteredDate as DATE)
```

```
Select      RegisteredDate as RegistrationDate, COUNT(Id) as TotalRegistrations
from        tblRegistrations
Group BY    RegisteredDate
```

RegistrationDate	TotalRegistrations
2012-08-24 11:04:30.230	1
2012-08-24 15:04:30.730	1
2012-08-24 15:05:30.330	1
2012-08-25 14:04:29.780	1
2012-08-25 15:04:29.780	1
2012-08-26 15:05:30.330	1

Difference CAST - CONVERT

1. Cast is based on ANSI standard and Convert is specific to SQL Server. So, if portability is a concern and if you want to use the script with other database applications, use Cast().

2. Convert provides more flexibility than Cast. For example, it's possible to control how you want DateTime datatypes to be converted using styles with convert function.

Note: The general guideline is to use CAST(), unless you want to take advantage of the style functionality in CONVERT().

- Part 29 –Mathematical functions

Mathematical Functions

ABS (numeric_expression) - ABS stands for absolute and returns, the absolute (positive) number.

```
Select ABS(-101.5) -- returns 101.5, without the - sign
```

CEILING (numeric_expression) and FLOOR (numeric_expression)

CEILING and FLOOR functions accept a numeric expression as a single parameter. CEILING() returns the smallest integer value greater than or equal to the parameter, whereas FLOOR() returns the largest integer less than or equal to the parameter.

```
Select CEILING(15.2) -- Returns 16
Select CEILING(-15.2) -- Returns -15

Select FLOOR(15.2) -- Returns 15
Select FLOOR(-15.2) -- Returns -16
```

Power(expression, power)

Returns the power value of the specified expression to the specified power.

```
Select POWER(2,3) -- Returns 8
```

SQUARE (Number)

Returns the square of the given number.

```
Select SQUARE(9) -- Returns 81
```

SQRT (Number)

Returns the square root of the given number

```
Select SQRT(81) -- Returns 9
```

RAND() function

RAND([Seed_Value]) - Returns a random float number between 0 and 1. Rand() function takes an optional seed parameter. When seed value is supplied the RADN() function always returns the same value for the same seed.

```
Select RAND(1) -- Always returns the same value
```

Generate a random number between 1 and 100

```
Select FLOOR(RAND() * 100)
```

Prints 10 random numbers between 1 and 100.

```
Declare @Counter INT
Set @Counter = 1
While(@Counter <= 10)
Begin
    Print FLOOR(RAND() * 100)
    Set @Counter = @Counter + 1
End
```

ROUND() function

ROUND (numeric_expression , length [,function]) - Rounds the given numeric expression based on the given length. This function takes 3 parameters.

1. Numeric_Expression is the number that we want to round.
2. Length parameter, specifies the number of the digits that we want to round to. If the length is a positive number, then the rounding is applied for the decimal part, where as if the length is negative, then the rounding is applied to the number before the decimal.
3. The optional function parameter, is used to indicate rounding or truncation operations. 0 indicates rounding, non zero indicates truncation. Default, if not specified is 0.

```
-- Round to 2 places after (to the right) the decimal point
Select ROUND(850.556, 2) -- Returns 850.560

-- Truncate anything after 2 places, after (to the right) the decimal point
Select ROUND(850.556, 2, 1) -- Returns 850.550

-- Round to 1 place after (to the right) the decimal point
Select ROUND(850.556, 1) -- Returns 850.600

-- Truncate anything after 1 place, after (to the right) the decimal point
Select ROUND(850.556, 1, 1) -- Returns 850.500

-- Round the last 2 places before (to the left) the decimal point
Select ROUND(850.556, -2) -- 900.000

-- Round the last 1 place before (to the left) the decimal point
Select ROUND(850.556, -1) -- 850.000
```

- [Part 30 – User defined functions]

SCALAR UDF

From Parts 22 to 29, we have learnt how to use many of the system functions that are available in SQL Server. In this session, we will turn our attention, to creating user defined functions. In short UDF.

In SQL Server there are 3 types of User Defined functions

1. Scalar functions
2. Inline table-valued functions
3. Multi-statement table-valued functions

Scalar functions may or may not have parameters, but always return a single (scalar) value. The returned value can be of any data type, except text, ntext, image, cursor, and timestamp

```
--To create a function, we use the following syntax:
CREATE FUNCTION Function_Name(@Parameter1 DataType, @Parameter2 DataType, ... @ParameterN Datatype)
RETURNS Return_Datatype
AS
BEGIN
   --Function Body
    Return Return_Datatype
END
```

SCALAR UDF

```
CREATE FUNCTION Age(@DOB Date)
RETURNS INT
AS
BEGIN
    DECLARE @Age INT
    SET @Age = DATEDIFF(YEAR, @DOB, GETDATE()) -
                CASE
                  WHEN (MONTH(@DOB) > MONTH(GETDATE())) OR
                       (MONTH(@DOB) = MONTH(GETDATE()) AND DAY(@DOB) > DAY(GETDATE()))
                  THEN 1
                  ELSE 0
                 END
    RETURN @Age
END
```

When calling a scalar user-defined function, you must supply a two-part name, OwnerName.FunctionName. dbo stands for database owner.

```
Select dbo.Age('10/08/1982')
```

You can also invoke it using the complete 3 part name, DatabaseName.OwnerName.FunctionName

```
Select SampleDB.dbo.Age( dbo.Age('10/08/1982')
```

SCALAR UDF

```
--Scalar user defined functions can be used in the Select clause
Select Name, DateOfBirth, dbo.Age(DateOfBirth) as Age from tblEmployees
```

Id	Name	DateOfBirth
1	Sam	1980-12-30 00:00:00.000
2	Pam	1982-09-01 12:02:36.260
3	John	1985-08-22 12:03:30.370
4	Sara	1979-11-29 12:59:30.670

→

Name	DateOfBirth	Age
Sam	1980-12-30 00:00:00.000	31
Pam	1982-09-01 12:02:36.260	30
John	1985-08-22 12:03:30.370	27
Sara	1979-11-29 12:59:30.670	32

```
--Scalar user defined functions can be used in the Select Where clauses
Select Name, DateOfBirth, dbo.Age(DateOfBirth) as Age from tblEmployees
Where dbo.Age(DateOfBirth) > 30
```

Id	Name	DateOfBirth
1	Sam	1980-12-30 00:00:00.000
2	Pam	1982-09-01 12:02:36.260
3	John	1985-08-22 12:03:30.370
4	Sara	1979-11-29 12:59:30.670

→

Name	DateOfBirth	Age
Sam	1980-12-30 00:00:00.000	31
Sara	1979-11-29 12:59:30.670	32

A stored procedure also can accept DateOfBirth and return Age, but you cannot use stored procedures in a select or where clause. This is just one difference between a function and a stored procedure. There are several other differences, which we will talk about in a later session.

To alter a function we use ALTER FUNCTION FuncationName statement and to delete it, we use DROP FUNCTION FuncationName.

Part 31 – Inline Table Valued Functions

Inline Table Valued Functions

In Part 30 of this video series we have seen how to create and call 'scalar user defined functions'. In this part, we will learn about 'Inline Table Valued Functions'

SCALAR FUNCTION – RETURNS A SCALAR VALUE
INLINE TABLE VALUED FUNCTION – RETURNS A TABLE

Id	Name	DateOfBirth	Gender	DepartmentId
1	Sam	1980-12-30 00:00:00.000	Male	1
2	Pam	1982-09-01 12:02:36.260	Female	2
3	John	1985-08-22 12:03:30.370	Male	1
4	Sara	1979-11-29 12:59:30.670	Female	3
5	Todd	1978-11-29 12:59:30.670	Male	1

Id	Name	DateOfBirth	Gender	DepartmentId
1	Sam	1980-12-30 00:00:00.000	Male	1
3	John	1985-08-22 12:03:30.370	Male	1
5	Todd	1978-11-29 12:59:30.670	Male	1

```
CREATE FUNCTION fn_EmployeesByGender(@Gender nvarchar(10))
RETURNS TABLE
AS
RETURN (Select Id, Name, DateOfBirth, Gender, DepartmentId
        from   tblEmployees
        where  Gender = @Gender)
```

1. We specify TABLE as the return type, instead of any scalar data type
2. The function body is not enclosed between BEGIN and END block.
3. The structure of the table that gets returned, is determined by the SELECT statement with in the function.

Inline Table Valued Functions

```
-- To Call the function
Select * from fn_EmployeesByGender('Male')
```

Id	Name	DateOfBirth	Gender	DepartmentId
1	Sam	1980-12-30 00:00:00.000	Male	1
3	John	1985-08-22 12:03:30.370	Male	1
5	Todd	1978-11-29 12:59:30.670	Male	1

Id	DepartmentName	Location	DepartmentHead
1	IT	London	Rick
2	Payroll	Delhi	Ron
3	HR	New York	Christie
4	Other Department	Sydney	Cindrella

Where can we use Inline Table Valued functions

1. Inline Table Valued functions can be used to achieve the functionality of parameterized views. We will talk about views, in a later session.

2. The table returned by the table valued function, can also be used in joins with other tables.

```
SELECT  Name, Gender, DepartmentName
FROM    fn_EmployeesByGender('Male') E
JOIN    tblDepartment D ON D.Id = E.DepartmentId
```

Joins
Part 12: Joins in SQL Server
Part 13: Advanced Joins

Name	Gender	DepartmentName
Sam	Male	IT
John	Male	IT
Todd	Male	IT

Part 32 – Multi-Statement Table Valued Functions

Multi-Statement Table Valued Functions

Multi statement table valued functions are very similar to Inline Table valued functions, with a few differences.

Id	Name	DateOfBirth	Gender	DepartmentId
1	Sam	1980-12-30 00:00:00.000	Male	1
2	Pam	1982-09-01 12:02:36.260	Female	2
3	John	1985-08-22 12:03:30.370	Male	1
4	Sara	1979-11-29 12:59:30.670	Female	3
5	Todd	1978-11-29 12:59:30.670	Male	1

Id	Name	DOB
1	Sam	1980-12-30
2	Pam	1982-09-01
3	John	1985-08-22
4	Sara	1979-11-29
5	Todd	1978-11-29

```
--Inline Table Valued Function
Create Function fn_ILTVF_GetEmployees()
Returns Table
as
Return (Select Id, Name, Cast(DateOfBirth as Date) as DOB From tblEmployees)
```

```
--Multi-statement Table Valued Function
Create Function fn_MSTVF_GetEmployees()
Returns @Table Table (Id int, Name nvarchar(20), DOB Date)
as
Begin
    Insert into @Table
    Select Id, Name, Cast(DateOfBirth as Date) From tblEmployees

    Return
End
```

Differences

1. In an Inline Table Valued function, the RETURNS clause cannot contain the structure of the table, the function returns. Where as, with the multi-statement table valued function, we specify the structure of the table that gets returned

2. Inline Table Valued function cannot have BEGIN and END block, where as the multi-statement function can have.

3. Inline Table valued functions are better for performance, than multi-statement table valued functions. If the given task, can be achieved using an inline table valued function, always prefer to use them, over multi-statement table valued functions.

4. It's possible to update the underlying table, using an inline table valued function, but not possible using multi-statement table valued function.

Reason for improved performance of an inline table valued function:
Internally, SQL Server treats an inline table valued function much like it would a view and treats a multi-statement table valued function similar to how it would a stored procedure.

Part 33 – Functions – Important Concepts

Deterministic and Nondeterministic

Deterministic functions always return the same result any time they are called with a specific set of input values and given the same state of the database.
Examples: Square(), Power(), Sum(), AVG() and Count()

Note: All aggregate functions are deterministic functions.

Nondeterministic functions may return different results each time they are called with a specific set of input values even if the database state that they access remains the same.
Examples: GetDate() and CURRENT_TIMESTAMP

Rand() function is a Non-deterministic function, but if you provide the seed value, the function becomes deterministic, as the same value gets returned for the same seed value.

WITH ENCRYPTION and SCHEMABINDING

Encrypting a function definiton using WITH ENCRYPTION OPTION:
We have learnt how to encrypt Stored procedure text using WITH ENCRYPTION OPTION in Part 18 of this video series. Along the same lines, you can also encrypt a function text. Once, encrypted, you cannot view the text of the function, using sp_helptext system stored procedure. If you try to, you will get a message stating 'The text for object is encrypted.' There are ways to decrypt, which is beyond the scope of this video.

Use WITH ENCRYPTION

Creating a function WITH SCHEMABINDING option:
Schemabinding, specifies that the function is bound to the database objects that it references. When SCHEMABINDING is specified, the base objects cannot be modified in any way that would affect the function definition. The function definition itself must first be modified or dropped to remove dependencies on the object that is to be modified.

Use WITH SCHEMABINDING

Part 34 – Temporary Tables

Temporary Tables

What are Temporary tables?
Temporary tables, are very similar to the permanent tables. Permanent tables get created in the database you specify, and remain in the database permanently, until you delete (drop) them. On the other hand, temporary tables get created in the TempDB and are automatically deleted, when they are no longer used.

Different Types of Temporary tables
1. Local Temporary Tables
2. Global Temporary tables.

```
--Create Local Temporary Table
Create Table #PersonDetails(Id int, Name nvarchar(20))

--Insert Data into the temporary table
Insert into #PersonDetails Values(1, 'Mike')
Insert into #PersonDetails Values(2, 'John')
Insert into #PersonDetails Values(3, 'Todd')

--Select data from the temporary table:
Select * from #PersonDetails
```

Local Temporary Tables

Check if the local temporary table is created:
Temporary tables are created in the TEMPDB. Query the sysobjects system table in TEMPDB. The name of the table, is suffixed with lot of underscores and a random number. For this reason you have to use the LIKE operator in the query.

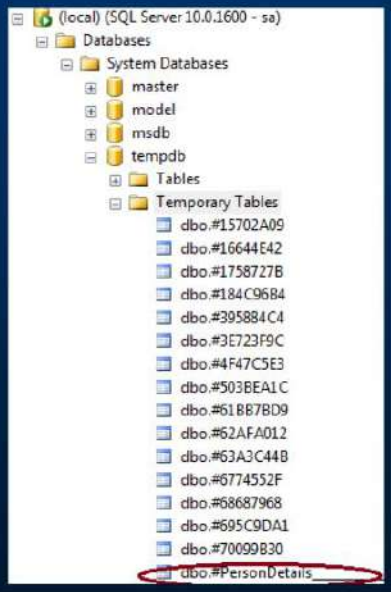
```
Select  name from tempdb..sysobjects
where   name like '#PersonDetails%'
```

A local temporary table is available, **only for the connection that has created the table**

A local temporary table **is automatically dropped**, when the connection that has created the it, is closed.

If the user wants to explicitly drop the temporary table, he can do so using **DROP TABLE #PersonDetails**

You can also check the existence of temporary tables using object explorer.



Local Temporary Tables

If the temporary table, is created inside the stored procedure, it get's dropped automatically upon the completion of stored procedure execution.

```
Create Procedure spCreateLocalTempTable
as
Begin
Create Table #PersonDetails(Id int, Name nvarchar(20))

Insert into #PersonDetails Values(1, 'Mike')
Insert into #PersonDetails Values(2, 'John')
Insert into #PersonDetails Values(3, 'Todd')

Select * from #PersonDetails
End
```

It is also possible for different connections, to create a local temporary table with the same name. For example User1 and User2, both can create a local temporary table with the same name #PersonDetails.

Global Temporary Tables

To create a Global Temporary Table, prefix the name of the table with 2 pound (##) symbols.

```
Create Table ##EmployeeDetails(Id int, Name nvarchar(20))
```

Global temporary tables are visible to all the connections of the sql server, and are only destroyed when the last connection referencing the table is closed.

Multiple users, across multiple connections can have local temporary tables with the same name, but, a global temporary table name has to be unique, and if you inspect the name of the global temp table, in the object explorer, there will be no random numbers suffixed at the end of the table name.

Difference

Difference Between Local and Global Temporary Tables:

1. Local Temp tables are prefixed with single pound (#) symbol, where as gloabl temp tables are prefixed with 2 pound (##) symbols.

2. SQL Server appends some random numbers at the end of the local temp table name, where this is not done for global temp table names.

3. Local temporary tables are only visible to that session of the SQL Server which has created it, where as Global temporary tables are visible to all the SQL server sessions

4. Local temporary tables are automatically dropped, when the session that created the temporary tables is closed, where as Global temporary tables are destroyed when the last connection that is referencing the global temp table is closed.

Part 35 – Indexes

Why Indexes

Indexes are used by queries to find data from tables quickly. Indexes are created on tables and views. Index on a table or a view, is very similar to an index that we find in a book.

If you don't have an index, and I ask you to locate a specific chapter in the book, you will have to look at every page starting from the first page of the book.

On, the other hand, if you have the index, you lookup the page number of the chapter in the index, and then directly go to that page number to locate the chapter.

Obviously, the book index is helping to drastically reduce the time it takes to find the chapter.

In a similar way, Table and View indexes, can help the query to find data quickly.

In fact, the existence of the right indexes, can drastically improve the performance of the query. If there is no index to help the query, then the query engine, checks every row in the table from the beginning to the end. This is called as Table Scan. Table scan is bad for performance.

Index Example

At the moment, the Employees table, does not have an index on SALARY column.

Id	Name	Salary	Gender
1	Sam	2500	Male
2	Pam	6500	Female
3	John	4500	Male
4	Sara	5500	Female
5	Todd	3100	Male

```
Select * from tblEmployee
where Salary > 5000 and Salary < 7000
```

To find all the employees, who has salary greater than 5000 and less than 7000, the query engine has to check each and every row in the table, resulting in a table scan, which can adversely affect the performance, especially if the table is large. Since there is no index, to help the query, the query engine performs an entire table scan.

Creating an Index

```
CREATE Index IX_tblEmployee_Salary
ON tblEmployee (SALARY ASC)
```

The index stores salary of each employee, in the ascending order as shown below. The actual index may look slightly different.

Id	Name	Salary	Gender
1	Sam	2500	Male
2	Pam	6500	Female
3	John	4500	Male
4	Sara	5500	Female
5	Todd	3100	Male

Salary	RowAddress
2500	Row Address
3100	Row Address
4500	Row Address
5500	Row Address
6500	Row Address

Now, when the SQL server has to execute the same query, it has an index on the salary column to help this query. Salaries between the range of 5000 and 7000 are usually present at the bottom, since the salaries are arranged in an ascending order. SQL server picks up the row addresses from the index and directly fetch the records from the table, rather than scanning each row in the table. This is called as Index Seek.

Next Video – Different Types of indexes, Disadvantage etc...

Part 36 - Clustered & NonClustered indexes

Index Types

1. Clustered
2. Nonclustered
3. Unique
4. Filtered
5. XML
6. Full Text
7. Spatial
8. Columnstore
9. Index with included columns
10. Index on computed columns

In this session: Clustered and Nonclustered

Clustered Index

A clustered index determines the physical order of data in a table. For this reason, a table can have only one clustered index.

```
CREATE TABLE [tblEmployee]
(
  [Id] int Primary Key,
  [Name] nvarchar(50),
  [Salary] int,
  [Gender] nvarchar(10),
  [City] nvarchar(50)
)
```

Note that Id column is marked as primary key. Primary key, constraint create clustered indexes automatically if no clustered index already exists on the table.

To confirm: `Execute sp_helpindex tblEmployee`

Note that, the values for Id column are not in a sequential order

```
Insert into tblEmployee Values(3,'John',4500,'Male','New York')
Insert into tblEmployee Values(1,'Sam',2500,'Male','London')
Insert into tblEmployee Values(4,'Sara',5500,'Female','Tokyo')
Insert into tblEmployee Values(5,'Todd',3100,'Male','Toronto')
Insert into tblEmployee Values(2,'Pam',6500,'Female','Sydney')
```

```
Select * from tblEmployee
```

Id	Name	Salary	Gender	City
1	Sam	2500	Male	London
2	Pam	6500	Female	Sydney
3	John	4500	Male	New York
4	Sara	5500	Female	Tokyo
5	Todd	3100	Male	Toronto

Clustered Index

A clustered index is analogous to a telephone directory, where the data is arranged by the last name. We just learnt that, a table can have only one clustered index. However, the index can contain multiple columns (a composite index), like the way a telephone directory is organized by last name and first name.

Create a composite clustered Index on the Gender and Salary columns

```
Create Clustered Index IX_tblEmployee_Gender_Salary
ON tblEmployee(Gender DESC, Salary ASC)
```

```
Select * from tblEmployee
```

Id	Name	Salary	Gender	City
1	Sam	2500	Male	London
5	Todd	3100	Male	Toronto
3	John	4500	Male	New York
4	Sara	5500	Female	Tokyo
2	Pam	6500	Female	Sydney

NonClustered Index

```
Create NonClustered Index IX_tblEmployee_Name
ON tblEmployee(Name)
```

Id	Name	Salary	Gender	City
1	Sam	2500	Male	London
5	Todd	3100	Male	Toronto
3	John	4500	Male	New York
4	Sara	5500	Female	Tokyo
2	Pam	6500	Female	Sydney

Name	Row Address
John	Row Address
Pam	Row Address
Sam	Row Address
Sara	Row Address
Todd	Row Address

A nonclustered index is analogous to an index in a textbook. The data is stored in one place, the index in another place. The index will have pointers to the storage location of the data.

Since, the nonclustered index is stored separately from the actual data, a table can have more than one non clustered index, just like how a book can have an index by Chapters at the beginning and another index by common terms at the end.

In the index itself, the data is stored in an ascending or descending order of the index key, which doesn't in any way influence the storage of data in the table.

Difference

1. Only one clustered index per table, where as you can have more than one non clustered index

2. Clustered index is faster than a non clustered index, because, the clustered index has to refer back to the table, if the selected column is not present in the index.

3. Clustered index determines the storage order of rows in the table, and hence doesn't require additional disk space, but where as a Non Clustered index is stored separately from the table, additional storage space is required.

Part 37 – Unique & Non-Unique indexes

Unique Index

Unique index is used to enforce uniqueness of key values in the index.

```
CREATE TABLE [tblEmployee]
(
        [Id] int Primary Key,
        [FirstName] nvarchar(50),
        [LastName] nvarchar(50),
        [Salary] int,
        [Gender] nvarchar(10),
        [City] nvarchar(50)
)
```

Note: By default, PRIMARY KEY constraint, creates a unique clustered index.

UNIQUENESS is a property of an Index, and both CLUSTERED and NON-CLUSTERED indexes can be UNIQUE.

```
Create Unique NonClustered Index
UIX_tblEmployee_FirstName_LastName
On tblEmployee(FirstName, LastName)
```

Difference between Unique Constraint and Unique Index

There are no major differences between a unique constraint and a unique index. In fact, when you add a unique constraint, a unique index gets created behind the scenes.

Unique Index

When should you be creating a Unique constraint over a unique index?

To make our intentions clear, create a unique constraint, when data integrity is the objective. This makes the objective of the index very clear. In either cases, data is validated in the same manner, and the query optimizer does not differentiate between a unique index created by a unique constraint or manually created.

Useful points to remember:

1. By default, a PRIMARY KEY constraint, creates a unique clustered index, where as a UNIQUE constraint creates a unique Non-Clustered index. These defaults can be changed if you wish to.

2. A UNIQUE constraint or a UNIQUE index cannot be created on an existing table, if the table contains duplicate values in the key columns. Obviously, to solve this, remove the key columns from the index definition or delete or update the duplicate values.

Part 38 – Adv. & Dis-adv. of Indexes

Advantages of Indexes

In Part 35, we have learnt that, Indexes are used by queries to find data quickly.

Id	FirstName	LastName	Salary	Gender	City
1	Mike	Sandoz	4500	Male	New York
2	Sara	Menco	6500	Female	London
3	John	Barber	2500	Male	Sydney
4	Pam	Grove	3500	Female	Toronto
5	James	Mirch	7500	Male	London

Salary	Row Address
2500	Row Address
3500	Row Address
4500	Row Address
6500	Row Address
7500	Row Address

```
--SELECT statement with a WHERE clause
Select * from tblEmployee where Salary > 4000 and Salary < 8000

--DELETE and UPDATE statement
Delete from tblEmployee where Salary = 2500
Update tblEmployee Set Salary = 9000 where Salary = 7500

--ORDER BY ASCENDING
Select * from tblEmployee order by Salary

--ORDER BY DESCENDING
Select * from tblEmployee order by Salary Desc

--GROUP BY
Select Salary, COUNT(Salary) as Total
from tblEmployee
Group By Salary
```

Disadvantages of Indexes

Additional DiskSpace: Clustered Index does not, require any additional storage. Every Non-Clustered index requires additional space as it is stored separately from the table. The amount of space required will depend on the size of the table, and the number and types of columns used in the index.

Insert Update and Delete statements can become slow: When DML (Data Manipulation Language) statements (INSERT, UPDATE, DELETE) modifies data in a table, the data in all the indexes also needs to be updated. Indexes can help, to search and locate the rows, that we want to delete, but too many indexes to update can actually hurt the performance of data modifications.

What is a covering query- If all the columns that you have requested in the SELECT clause of query, are present in the index, then there is no need to lookup in the table again. The requested columns data can simply be returned from the index.

A clustered index, always covers a query, since it contains all of the data in a table. A composite index is an index on two or more columns. Both clustered and non-clustered indexes can be composite indexes. To a certain extent, a composite index, can cover a query.

Part 39 – Views in SQL Server

Views

What is a View?

A view is nothing more than a saved SQL query. A view can also be considered as a virtual table

DeptId	DeptName
1	IT
2	Payroll
3	HR
4	Admin

Id	Name	Salary	Gender	DepartmentId
1	John	5000	Male	3
2	Mike	3400	Male	2
3	Pam	6000	Female	1
4	Todd	4800	Male	4
5	Sara	3200	Female	1
6	Ben	4800	Male	3

Id	Name	Salary	Gender	DeptName
1	John	5000	Male	HR
2	Mike	3400	Male	Payroll
3	Pam	6000	Female	IT
4	Todd	4800	Male	Admin
5	Sara	3200	Female	IT
6	Ben	4800	Male	HR

```
Select Id, Name, Salary, Gender, DeptName
from tblEmployee
join tblDepartment
on tblEmployee.DepartmentId = tblDepartment.DeptId
```

```
Create View vWEmployeesByDepartment
as
Select Id, Name, Salary, Gender, DeptName
from tblEmployee
join tblDepartment
on tblEmployee.DepartmentId = tblDepartment.DeptId
```

Part 40 – Updateable Views

Updateable Views

Id	Name	Salary	Gender	DepartmentId
1	John	5000	Male	3
2	Mike	3400	Male	2
3	Pam	6000	Female	1
4	Todd	4800	Male	4
5	Sara	3200	Female	1
6	Ben	4800	Male	3

```
Create view vWEmployeesDataExceptSalary
as
Select Id, Name, Gender, DepartmentId
from tblEmployee
```

Id	Name	Gender	DepartmentId
1	John	Male	3
2	Mike	Male	2
3	Pam	Female	1
4	Todd	Male	4
5	Sara	Female	1
6	Ben	Male	3

```
Is it possible to INSERT, UPDATE & DELETE
From the base table tblEmployee thru the
View?

Select * from vWEmployeesDataExceptSalary

Update vWEmployeesDataExceptSalary ?
Insert into vWEmployeesDataExceptSalary ?
Delete from vWEmployeesDataExceptSalary ?
```

Updateable Views

Id	Name	Salary	Gender	DepartmentId
1	John	5000	Male	3
2	Mikey	NULL	Male	2
3	Pam	6000	Female	1
4	Todd	4800	Male	4
5	Sara	3200	Female	1
6	Ben	4800	Male	3

DeptId	DeptName
1	IT
2	Payroll
3	HR
4	Admin

```
Create view vwEmployeeDetailsByDepartment
as
Select Id, Name, Salary, Gender, DeptName
from tblEmployee
join tblDepartment
on tblEmployee.DepartmentId = tblDepartment.DeptId
```

Id	Name	Salary	Gender	DeptName
1	John	5000	Male	HR
2	Mikey	NULL	Male	Payroll
3	Pam	6000	Female	IT
4	Todd	4800	Male	Admin
5	Sara	3200	Female	IT
6	Ben	4800	Male	HR

Updateable Views

```
Update vwEmployeeDetailsByDepartment
set DeptName='IT' where Name = 'John'
```

Id	Name	Salary	Gender	DeptName
1	John	5000	Male	IT
2	Mikey	NULL	Male	Payroll
3	Pam	6000	Female	IT
4	Todd	4800	Male	Admin
5	Sara	3200	Female	IT
6	Ben	4800	Male	IT

DeptId	DeptName
1	IT
2	Payroll
3	IT
4	Admin

Id	Name	Salary	Gender	DepartmentId
1	John	5000	Male	3
2	Mikey	NULL	Male	2
3	Pam	6000	Female	1
4	Todd	4800	Male	4
5	Sara	3200	Female	1
6	Ben	4800	Male	3

Conclusion - If a view is based on multiple tables, and if you update the view, it may not update the underlying base tables correctly. To correctly update a view, that is based on multiple table, **INSTEAD OF** triggers are used.

We will discuss about triggers and correctly updating a view that is based on multiple tables, in a later video session.

Part 41 – Indexed Views

Indexed Views

What is an Indexed View?
or
What happens when you create an Index on a view?

A standard or Non-indexed view, is just a stored SQL query. When, we try to retrieve data from the view, the data is actually retrieved from the underlying base tables.

So, a view is just a virtual table it does not store any data, by default.

However, when you create an index, on a view, the view gets materialized. This means, the view is now, capable of storing data.

In SQL server, we call them Indexed views and in Oracle, Materialized views.

ndexed Views - Example

ProductId	Name	UnitPrice
1	Books	20
2	Pens	14
3	Pencils	11
4	Clips	10

ProductId	QuantitySold
1	10
3	23
4	21
2	12
1	13
3	12
4	13
1	11
2	12
1	14

Name	TotalSales	TotalTransactions
Books	960	4
Clips	340	2
Pencils	385	2
Pens	336	2

```
Create view vWTotalSalesByProduct
With SchemaBinding
as
Select Name,
SUM(ISNULL((QuantitySold * UnitPrice), 0)) as TotalSales,
COUNT_BIG(*) as TotalTransactions
from dbo.tblProductSales
join dbo.tblProduct
on dbo.tblProduct.ProductId = dbo.tblProductSales.ProductId
group by Name
```

Guidelines for creating Indexed Views

```
Create view vWTotalSalesByProduct
With SchemaBinding
as
Select Name,
SUM(ISNULL((QuantitySold * UnitPrice), 0)) as TotalSales,
COUNT_BIG(*) as TotalTransactions
from dbo.tblProductSales
join dbo.tblProduct
on dbo.tblProduct.ProductId = dbo.tblProductSales.ProductId
group by Name
```

Name	TotalSales	TotalTransactions
Books	960	4
Clips	340	2
Pencils	385	2
Pens	336	2

1. The view should be created with SchemaBinding option
2. If an Aggregate function in the SELECT LIST, references an expression, and if there is a possibility for that expression to become NULL, then, a replacement value should be specified.
3. If GROUP BY is specified, the view select list must contain a COUNT_BIG(*) expression
4. The base tables in the view, should be referenced with 2 part name.

```
Create Unique Clustered Index UIX_vWTotalSalesByProduct_Name
on vWTotalSalesByProduct(Name)
```

Part 42 –View Limitations

View Limitations

1. You cannot pass parameters to a view. Table Valued functions are an excellent replacement for parameterized views.

2. Rules and Defaults cannot be associated with views.

3. The ORDER BY clause is invalid in views unless TOP or FOR XML is also specified.

4. Views cannot be based on temporary tables.

View Limitations

```
Create View vWEmployeeDetails
as
Select Id, Name, Gender, DepartmentId
from   tblEmployee
```

```
Create View vWEmployeeDetails
@Gender nvarchar(20)
as
Select Id, Name, Gender, DepartmentId
from   tblEmployee
Where  Gender = @Gender
```

```
--You can filter Gender in the WHERE clause
Select * from vWEmployeeDetails where Gender = 'Male'
```

```
-- Inline Table valued function as a replacement for
-- Parameterized views
Create function fnEmployeeDetails(@Gender nvarchar(20))
Returns Table
as
Return
(Select Id, Name, Gender, DepartmentId
from tblEmployee where Gender = @Gender)


Select * from dbo.fnEmployeeDetails('Male')
```

Cannot create views on Temp Tables

```
Create Table ##TestTempTable
(Id int, Name nvarchar(20), Gender nvarchar(10))


Insert into ##TestTempTable values(101, 'Martin', 'Male')
Insert into ##TestTempTable values(102, 'Joe', 'Female')
Insert into ##TestTempTable values(103, 'Pam', 'Female')
Insert into ##TestTempTable values(104, 'James', 'Male')


Create View vwOnTempTable
as
Select Id, Name, Gender
from ##TestTempTable
```

Part 43 –DML Triggers

Triggers

In SQL server there are 3 types of triggers

1. DML triggers
2. DDL triggers
3. Logon trigger

DML triggers are fired automatically in response to DML events (INSERT, UPDATE & DELETE)

DML triggers can be again classified into 2 types

1. After triggers (Sometimes called as FOR triggers)
2. Instead of triggers

After triggers, fires after the triggering action. The INSERT, UPDATE, and DELETE statements, causes an after trigger to fire after the respective statements complete execution.

INSTEAD of triggers, fires instead of the triggering action. The INSERT, UPDATE, and DELETE statements, causes an INSTEAD OF trigger to fire INSTEAD OF the respective statement execution.

After Triggers

Id	Name	Salary	Gender	DepartmentId
1	John	5000	Male	3
2	Mike	3400	Male	2
3	Pam	6000	Female	1
4	Todd	4800	Male	4
5	Sara	3200	Female	1
6	Ben	4800	Male	3

Id	AuditData
1	New employee with Id = 8 is added at Sep 17 2012 8:04PM
2	An existing employee with Id = 8 is deleted at Sep 17 2012 8:29PM
3	An existing employee with Id = 1 is deleted at Sep 17 2012 8:31PM
4	An existing employee with Id = 3 is deleted at Sep 17 2012 8:31PM
5	An existing employee with Id = 4 is deleted at Sep 17 2012 8:31PM

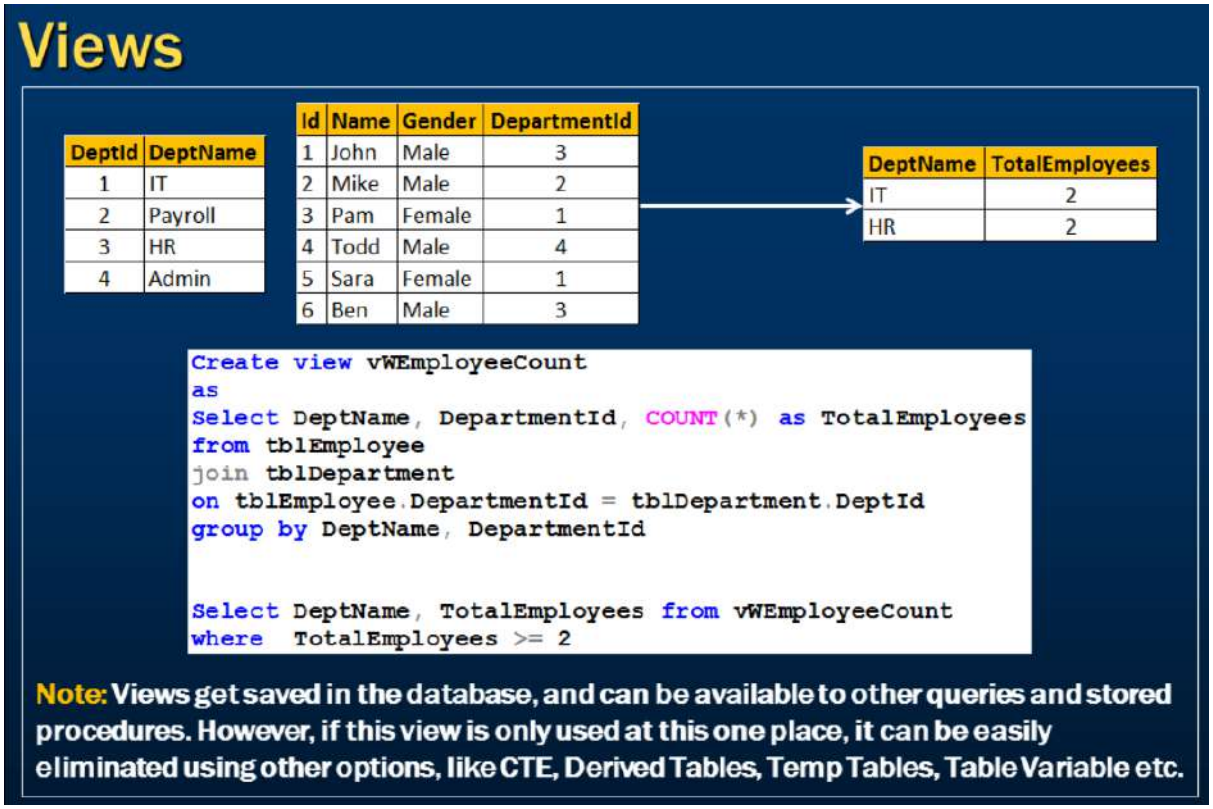
```
CREATE TRIGGER tr_tblEmployee_ForInsert
ON tblEmployee
FOR INSERT
AS
BEGIN
        Declare @Id int
        Select @Id = Id from inserted

        insert into tblEmployeeAudit
        values ('New employee with Id  = ' +
                Cast(@Id as nvarchar(5)) +
                ' is added at ' +
                cast(Getdate() as nvarchar(20))
               )
END
```

```
Insert into tblEmployee values
(8,'Ben', 4800, 'Male', 3)
```

```
CREATE TRIGGER tr_tblEmployee_ForDelete
ON tblEmployee
FOR DELETE
AS
BEGIN
        Declare @Id int
        Select @Id = Id from deleted

        insert into tblEmployeeAudit
        values('An existing employee with Id  = ' +
                Cast(@Id as nvarchar(5)) +
                ' is deleted at ' +
                cast(Getdate() as nvarchar(20))
               )
END
```

```
Delete from tblEmployee where Id = 3
```

- [Part 44 –DML Triggers]
- [Part 45 –Instead Of Insert Trigger]
- [Part 46 –Instead Of Update Trigger]
- [Part 47 –Instead Of Delete Trigger]
- [Part 48 –Derived Tables & CTE]



Temp Tables

DeptId	DeptName
1	IT
2	Payroll
3	HR
4	Admin

Id	Name	Gender	DepartmentId
1	John	Male	3
2	Mike	Male	2
3	Pam	Female	1
4	Todd	Male	4
5	Sara	Female	1
6	Ben	Male	3

DeptName	TotalEmployees
IT	2
HR	2

```
Select DeptName, DepartmentId, COUNT(*) as TotalEmployees
into #TempEmployeeCount
from tblEmployee
join tblDepartment
on tblEmployee.DepartmentId = tblDepartment.DeptId
group by DeptName, DepartmentId

Select DeptName, TotalEmployees
From #TempEmployeeCount
where  TotalEmployees >= 2

Drop Table #TempEmployeeCount
```

Notes: Temporary tables are stored in TempDB. Local temporary tables are visible only in the current session, and can be shared between nested stored procedure calls. Global temporary tables are visible to other sessions and are destroyed, when the last connection referencing the table is closed.

Table Variable

DeptId	DeptName
1	IT
2	Payroll
3	HR
4	Admin

Id	Name	Gender	DepartmentId
1	John	Male	3
2	Mike	Male	2
3	Pam	Female	1
4	Todd	Male	4
5	Sara	Female	1
6	Ben	Male	3

DeptName	TotalEmployees
IT	2
HR	2

```
Declare @tblEmployeeCount table(DeptName nvarchar(20),DepartmentId int,
TotalEmployees int)

Insert @tblEmployeeCount
Select DeptName, DepartmentId, COUNT(*) as TotalEmployees
from tblEmployee
join tblDepartment
on tblEmployee.DepartmentId = tblDepartment.DeptId
group by DeptName, DepartmentId

Select DeptName, TotalEmployees
From @tblEmployeeCount
where  TotalEmployees >= 2
```

Note: Just like TempTables, a table variable is also created in TempDB. The scope of a table variable is the batch, stored procedure, or statement block in which it is declared. They can be passed as parameters between procedures.

Derived Tables

DeptId	DeptName
1	IT
2	Payroll
3	HR
4	Admin

Id	Name	Gender	DepartmentId
1	John	Male	3
2	Mike	Male	2
3	Pam	Female	1
4	Todd	Male	4
5	Sara	Female	1
6	Ben	Male	3

DeptName	TotalEmployees
IT	2
HR	2

```
Select DeptName, TotalEmployees
from
        (
                Select DeptName, DepartmentId, COUNT(*) as TotalEmployees
                from tblEmployee
                join tblDepartment
                on tblEmployee.DepartmentId = tblDepartment.DeptId
                group by DeptName, DepartmentId
        )
as EmployeeCount
where TotalEmployees >= 2

EmployeeCount is the derived table
```

Note: Derived tables are available only in the context of the current query.

CTE

DeptId	DeptName
1	IT
2	Payroll
3	HR
4	Admin

Id	Name	Gender	DepartmentId
1	John	Male	3
2	Mike	Male	2
3	Pam	Female	1
4	Todd	Male	4
5	Sara	Female	1
6	Ben	Male	3

DeptName	TotalEmployees
IT	2
HR	2

```
With EmployeeCount(DeptName, DepartmentId, TotalEmployees)
as
(
        Select DeptName, DepartmentId, COUNT(*) as TotalEmployees
        from tblEmployee
        join tblDepartment
        on tblEmployee.DepartmentId = tblDepartment.DeptId
        group by DeptName, DepartmentId
)

Select DeptName, TotalEmployees
from   EmployeeCount
where  TotalEmployees >= 2
```

Note: A CTE can be thought of as a temporary result set that is defined within the execution scope of a single SELECT, INSERT, UPDATE, DELETE, or CREATE VIEW statement. A CTE is similar to a derived table in that it is not stored as an object and lasts only for the duration of the query.

Part 49 – CTE

CTE

Common table expression (CTE) is introduced in SQL server 2005. A CTE is a temporary result set, that can be referenced within a SELECT, INSERT, UPDATE, or DELETE statement, that immediately follows the CTE.

Id	Name	Gender	DepartmentId
1	John	Male	3
2	Mike	Male	2
3	Pam	Female	1
4	Todd	Male	4
5	Sara	Female	1
6	Ben	Male	3

```
--Syntax
WITH cte_name (Column1, Column2, ..)
AS
( CTE_query )
```

DeptId	DeptName
1	IT
2	Payroll
3	HR
4	Admin

DeptName	TotalEmployees
Payroll	1
Admin	1
IT	2
HR	2

```
With EmployeeCount(DepartmentId, TotalEmployees)
as
(
        Select DepartmentId, COUNT(*) as TotalEmployees
        from tblEmployee
        group by DepartmentId
)

Select DeptName, TotalEmployees
from tblDepartment
join EmployeeCount
on tblDepartment.DeptId = EmployeeCount.DepartmentId
order by TotalEmployees
```

CTE

```
With EmployeeCount(DepartmentId, TotalEmployees)
as
(
        Select DepartmentId, COUNT(*) as TotalEmployees
        from tblEmployee
        group by DepartmentId
)

Select 'Hello'

Select DeptName, TotalEmployees
from tblDepartment
join EmployeeCount
on tblDepartment.DeptId = EmployeeCount.DepartmentId
order by TotalEmployees
```

A CTE can only be referenced by a SELECT, INSERT, UPDATE, or DELETE statement, that immediately follows the CTE.

CTE

```sql
--Creating multiple CTE's using a single WITH clause
With EmployeesCountBy_Payroll_IT_Dept(DepartmentName, Total)
as
(
        Select DeptName, COUNT(Id) as TotalEmployees
        from tblEmployee
        join tblDepartment
        on tblEmployee.DepartmentId = tblDepartment.DeptId
        where DeptName IN ('Payroll','IT')
        group by DeptName
),
EmployeesCountBy_HR_Admin_Dept(DepartmentName, Total)
as
(
        Select DeptName, COUNT(Id) as TotalEmployees
        from tblEmployee
        join tblDepartment
        on tblEmployee.DepartmentId = tblDepartment.DeptId
        where DeptName IN ('HR','Admin')
        group by DeptName
)
Select * from EmployeesCountBy_HR_Admin_Dept
UNION
Select * from EmployeesCountBy_Payroll_IT_Dept
```

Part 50 – Updatable CTEs

CTE on one base table

Is it possible to UPDATE a CTE – Yes & No

Id	Name	Gender	DepartmentId
1	John	Male	3
2	Mike	Male	2
3	Pam	Female	1
4	Todd	Male	4
5	Sara	Female	1
6	Ben	Male	3

```sql
--CTE with one base table
With Employees_Name_Gender
as
(
        Select Id, Name, Gender
        from tblEmployee
)
Select * from Employees_Name_Gender
```

Id	Name	Gender
1	John	Male
2	Mike	Male
3	Pam	Female
4	Todd	Male
5	Sara	Female
6	Ben	Male

```sql
With Employees_Name_Gender
as
(
        Select Id, Name, Gender from tblEmployee
)
Update Employees_Name_Gender
Set Gender = 'Female' where Id = 1
```

Id	Name	Gender	DepartmentId
1	John	Female	3
2	Mike	Male	2
3	Pam	Female	1
4	Todd	Male	4
5	Sara	Female	1
6	Ben	Male	3

So, if a CTE is created on one base table, then it is possible to UPDATE the CTE, which in turn will update the underlying base table.

In this case, UPDATING Employees_Name_Gender CTE, updates tblEmployee table.

CTE on 2 base tables

Id	Name	Gender	DepartmentId
1	John	Male	3
2	Mike	Male	2
3	Pam	Female	1
4	Todd	Male	4
5	Sara	Female	1
6	Ben	Male	3

DeptId	DeptName
1	IT
2	Payroll
3	HR
4	Admin

Id	Name	Gender	DeptName
1	John	Female	HR
2	Mike	Male	Payroll
3	Pam	Female	IT
4	Todd	Male	Admin
5	Sara	Female	IT
6	Ben	Male	HR

```
--CTE on 2 base tables, Update affecting only one base table
With EmployeesByDepartment
as
(
        Select Id, Name, Gender, DeptName
        from tblEmployee
        join tblDepartment
        on tblDepartment.DeptId = tblEmployee.DepartmentId
)
Update EmployeesByDepartment set Gender = 'Male' where Id = 1
```

Note: If a CTE is based on more than one table, and if the UPDATE affects only one base table, then the UPDATE is allowed.

CTE on 2 base tables

Id	Name	Gender	DepartmentId
1	John	Male	3
2	Mike	Male	2
3	Pam	Female	1
4	Todd	Male	4
5	Sara	Female	1
6	Ben	Male	3

DeptId	DeptName
1	IT
2	Payroll
3	HR
4	Admin

Id	Name	Gender	DeptName
1	John	Female	HR
2	Mike	Male	Payroll
3	Pam	Female	IT
4	Todd	Male	Admin
5	Sara	Female	IT
6	Ben	Male	HR

```
--CTE on 2 base tables, Update affecting more than one base table
With EmployeesByDepartment
as
(
        Select Id, Name, Gender, DeptName
        from tblEmployee
        join tblDepartment
        on tblDepartment.DeptId = tblEmployee.DepartmentId
)
Update EmployeesByDepartment set Gender = 'Female', DeptName = 'IT'
where Id = 1
```

Note: If a CTE is based on multiple tables, and if the UPDATE statement affects more than 1 base table, then the UPDATE is not allowed.

CTE on 2 base tables

Id	Name	Gender	DepartmentId
1	John	Male	3
2	Mike	Male	2
3	Pam	Female	1
4	Todd	Male	4
5	Sara	Female	1
6	Ben	Male	3

DeptId	DeptName
1	IT
2	Payroll
3	HR
4	Admin

Id	Name	Gender	DeptName
1	John	Female	HR
2	Mike	Male	Payroll
3	Pam	Female	IT
4	Todd	Male	Admin
5	Sara	Female	IT
6	Ben	Male	HR

```
With EmployeesByDepartment
as
(
        Select Id, Name, Gender, DeptName
        from tblEmployee
        join tblDepartment
        on tblDepartment.DeptId = tblEmployee.DepartmentId
)
Update EmployeesByDepartment set
DeptName = 'IT' where Id = 1
```

Id	Name	Gender	DeptName
1	John	Male	IT
2	Mike	Male	Payroll
3	Pam	Female	IT
4	Todd	Male	Admin
5	Sara	Female	IT
6	Ben	Male	IT

DeptId	DeptName
1	IT
2	Payroll
3	IT
4	Admin

Note: A CTE is based on more than one base table, and if the UPDATE affects only one base table, the UPDATE succeeds (but not as expected always)

Updatable CTE

1. If a CTE is based on a single base table, then the UPDATE succeeds and works as expected.

2. If a CTE is based on more than one base table, and if the UPDATE affects multiple base tables, the update is not allowed and the statement terminates with an error.

3. If a CTE is based on more than one base table, and if the UPDATE affects only one base table, the UPDATE succeeds(but not as expected always)

Part 51 – Recursive CTEs

Recursive CTE

EmployeeId	Name	ManagerId
1	Tom	2
2	Josh	NULL
3	Mike	2
4	John	3
5	Pam	1
6	Mary	3
7	James	1
8	Sam	5
9	Simon	1

Employee Name	Manager Name
Tom	Josh
Josh	Super Boss
Mike	Josh
John	Mike
Pam	Tom
Mary	Mike
James	Tom
Sam	Pam
Simon	Tom

```
-- A simple self join
Select Employee.Name as [Employee Name],
IsNull(Manager.Name, 'Super Boss') as [Manager Name]
from tblEmployee Employee
left join tblEmployee Manager
on Employee.ManagerId = Manager.EmployeeId
```

Joining a table with itself is called as self join

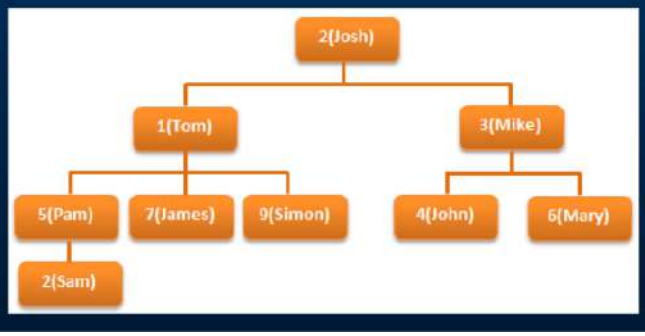
Part 14 – Self Join
Part 15 – Different ways to replace NULL values

Recursive CTE

Along with Employee and their Manager name, we also want to display their level in the organization

EmployeeId	Name	ManagerId
1	Tom	2
2	Josh	NULL
3	Mike	2
4	John	3
5	Pam	1
6	Mary	3
7	James	1
8	Sam	5
9	Simon	1

Employee	Manager	Level
Josh	Super Boss	1
Tom	Josh	2
Mike	Josh	2
John	Mike	3
Mary	Mike	3
Pam	Tom	3
James	Tom	3
Simon	Tom	3
Sam	Pam	4



Recursive CTE

```
With
  EmployeesCTE (EmployeeId, Name, ManagerId, [Level])
  as
  (
    Select EmployeeId, Name, ManagerId, 1
    from tblEmployee
    where ManagerId is null

    union all

    Select tblEmployee.EmployeeId, tblEmployee.Name,
    tblEmployee.ManagerId, EmployeesCTE.[Level] + 1
    from tblEmployee
    join EmployeesCTE
    on tblEmployee.ManagerID = EmployeesCTE.EmployeeId
  )
Select EmpCTE.Name as Employee, Isnull(MgrCTE.Name, 'Super Boss') as Manager,
EmpCTE.[Level]
from EmployeesCTE EmpCTE
left join EmployeesCTE MgrCTE
on EmpCTE.ManagerId = MgrCTE.EmployeeId
```

Part 52 – Normalization

What is Normalization

Database normalization is the process of organizing data to minimize data redundancy (data duplication), which in turn ensures data consistency.

EmployeeName	Gender	Salary	DeptName	DeptHead	DeptLocation
Sam	Male	4500	IT	John	London
Pam	Female	2300	HR	Mike	Sydney
Simon	Male	1345	IT	John	London
Mary	Female	2567	HR	Mike	Sydney
Todd	Male	6890	IT	John	London

Problems of Data Redundancy

1. Disk Space Wastage
2. Data Inconsistency
3. DML queries can become slow

Normalized Table Design

DeptId	DeptName	DeptHead	DeptLocation
1	IT	John	London
2	HR	Mike	Sydney

EmployeeId	EmployeeName	Gender	Salary	DeptId
1	Sam	Male	4500	1
2	Pam	Female	2300	2
3	Simon	Male	1345	1
4	Mary	Female	2567	2
5	Todd	Male	6890	1

Database normalization is a step by step process. There are 6 normal forms, First Normal form (1NF) thru Sixth Normal Form (6NF). Most databases are in **third normal form (3NF).** There are certain rules, that each normal form should follow.

First Normal Form (1NF)

A table is said to be in 1NF, if

1. The data in each column should be atomic. No multiple values, separated by comma
2. The table does not contain any repeating column groups
3. Identify each record uniquely using primary key

Non Atomic Employee Column

DeptName	Employee
IT	Sam, Mike, Shan
HR	Pam

Problems of Non Atomic Columns

It is not possible to SELECT, INSERT, UPDATE and DELETE just one employee

No Repeating Column Groups

DeptName	Employee1	Employee2	Employee3
IT	Sam	Mike	Shan
HR	Pam		

Problems of Repeating Column Groups

More than 3 employees - Table structure change required
Less than 3 employees - Wasted disk space

First Normal Form 1NF

Primary Key

DeptId	DeptName
1	IT
2	HR

Foreign Key

DeptId	Employee
1	Sam
1	Mike
1	Shan
2	Pam

First Normal Form 1NF

Part 53 – 2NF & 3NF

Second Normal Form (2NF)

A table is said to be in 2NF, if

1. The table meets all the conditions of 1NF
2. Move redundant data to a separate table
3. Create relationship between these tables using foreign keys.

EmpId	EmployeeName	Gender	Salary	DeptName	DeptHead	DeptLocation
1	Sam	Male	4500	IT	John	London
2	Pam	Female	2300	HR	Mike	Sydney
3	Simon	Male	1345	IT	John	London
4	Mary	Female	2567	HR	Mike	Sydney
5	Todd	Male	6890	IT	John	London

Problems of Data Redundancy

1. Disk Space Wastage
2. Data Inconsistency
3. DML queries can become slow

Table Design in Second Normal Form

DeptId	DeptName	DeptHead	DeptLocation
1	IT	John	London
2	HR	Mike	Sydney

EmpId	EmployeeName	Gender	Salary	DeptId
1	Sam	Male	4500	1
2	Pam	Female	2300	2
3	Simon	Male	1345	1
4	Mary	Female	2567	2
5	Todd	Male	6890	1

Third Normal Form (3NF)

A table is said to be in 3NF, if the table

1. Meets all the conditions of 1NF and 2NF
2. Does not contain columns (attributes) that are not fully dependent upon the primary key

EmpId	EmployeeName	Gender	Salary	AnnualSalary	DeptId
1	Sam	Male	4500	~~54000~~	1
2	Pam	Female	2300	~~27600~~	2
3	Simon	Male	1345	~~16140~~	1
4	Mary	Female	2567	~~30804~~	2
5	Todd	Male	6890	~~82680~~	1

EmpId	EmployeeName	Gender	Salary	DeptName	DeptHead
1	Sam	Male	4500	IT	John
2	Pam	Female	2300	HR	Mike
3	Simon	Male	1345	IT	John
4	Mary	Female	2567	HR	Mike
5	Todd	Male	6890	IT	John

EmpId	EmployeeName	Gender	Salary	DeptId
1	Sam	Male	4500	1
2	Pam	Female	2300	2
3	Simon	Male	1345	1
4	Mary	Female	2567	2
5	Todd	Male	6890	1

DeptId	DeptName	DeptHead
1	IT	John
2	HR	Mike

Part 54 – Pivot in SQL Server 2008

PIVOT Operator

Pivot is a sql server operator that can be used to turn unique values from one column, into multiple columns in the output, there by effectively rotating a table.

SalesAgent	SalesCountry	SalesAmount
Tom	UK	200
John	US	180
John	UK	260
David	India	450
Tom	India	350
David	US	200
Tom	US	130
John	India	540
John	UK	120
David	UK	220
John	UK	420
David	US	320
Tom	US	340
Tom	UK	660
John	India	430
David	India	230
David	India	280
Tom	UK	480
John	US	360
David	UK	140

SalesCountry	SalesAgent	Total
India	David	960
India	John	970
India	Tom	350
UK	David	360
UK	John	800
UK	Tom	1340
US	David	520
US	John	540
US	Tom	470

```
--Group By Query
SELECT SalesCountry, SalesAgent,
SUM(SalesAmount) AS Total
FROM tblProductSales
GROUP BY SalesCountry, SalesAgent
ORDER BY SalesCountry, SalesAgent
```

PIVOT Operator

SalesAgent	SalesCountry	SalesAmount
Tom	UK	200
John	US	180
John	UK	260
David	India	450
Tom	India	350
David	US	200
Tom	US	130
John	India	540
John	UK	120
David	UK	220
John	UK	420
David	US	320
Tom	US	340
Tom	UK	660
John	India	430
David	India	230
David	India	280
Tom	UK	480
John	US	360
David	UK	140

SalesCountry	SalesAgent	Total
India	David	960
India	John	970
India	Tom	350
UK	David	360
UK	John	800
UK	Tom	1340
US	David	520
US	John	540
US	Tom	470

```
--Group By Query
SELECT SalesCountry, SalesAgent,
SUM(SalesAmount) AS Total
FROM tblProductSales
GROUP BY SalesCountry, SalesAgent
ORDER BY SalesCountry, SalesAgent
```

SalesAgent	India	US	UK
David	960	520	360
John	970	540	800
Tom	350	470	1340

```
--Query using Pivot operator
SELECT SalesAgent, India, US, UK
FROM tblProductSales
PIVOT
(
        SUM(SalesAmount)
        FOR SalesCountry
        IN ([India],[US],[UK])
)
AS PivotTable
```

PIVOT Operator

Id	SalesAgent	SalesCountry	SalesAmount
1	Tom	UK	200
2	John	US	180
3	John	UK	260
4	David	India	450
5	Tom	India	350
6	David	US	200
7	Tom	US	130
8	John	India	540
9	John	UK	120
10	David	UK	220
11	John	UK	420
12	David	US	320
13	Tom	US	340
14	Tom	UK	660
15	John	India	430
16	David	India	230
17	David	India	280
18	Tom	UK	480
19	John	US	360
20	David	UK	140

```
--Query using Pivot operator
SELECT SalesAgent, India, US, UK
FROM tblProductSales
PIVOT
(
        SUM(SalesAmount)
        FOR SalesCountry
        IN ([India],[US],[UK])
)
AS PivotTable
```

```
Select SalesAgent,India,US,UK
from
(
        Select SalesAgent,
        SalesCountry,SalesAmount
        from tblProductsSale
) as SourceTable
Pivot
(
        Sum(SalesAmount)
        for SalesCountry in
        (India, US, UK)
) as PivotTable
```

SalesAgent	India	US	UK
Tom	NULL	NULL	200
John	NULL	180	NULL
John	NULL	NULL	260
David	450	NULL	NULL
Tom	350	NULL	NULL
David	NULL	200	NULL
Tom	NULL	130	NULL
John	540	NULL	NULL
John	NULL	NULL	120
David	NULL	NULL	220
John	NULL	NULL	420
David	NULL	320	NULL
Tom	NULL	340	NULL
Tom	NULL	NULL	660
John	430	NULL	NULL
David	230	NULL	NULL
David	280	NULL	NULL
Tom	NULL	NULL	480
John	NULL	360	NULL
David	NULL	NULL	140

PIVOT Operator

```
--Syntax from MSDN
SELECT <non-pivoted column>,
    [first pivoted column] AS <column name>,
    [second pivoted column] AS <column name>,

    ...

    [last pivoted column] AS <column name>
FROM
    (<SELECT query that produces the data>)
    AS <alias for the source query>
PIVOT
(
    <aggregation function>(<column being aggregated>)
FOR
    [<column that contains the values that will become column headers>]
    IN ( [first pivoted column], [second pivoted column], ... [last pivoted
column])
)
AS <alias for the pivot table>
<optional ORDER BY clause>
```

```
Select SalesAgent,India,US,UK
from
(
        Select SalesAgent,
        SalesCountry,SalesAmount
        from tblProductsSale
) as SourceTable
Pivot
(
        Sum(SalesAmount)
        for SalesCountry in
        (India, US, UK)
) as PivotTable
```

Part 55 – Error Handling in SQL Server 2000

Error handling

With the introduction of Try/Catch blocks in SQL Server 2005, error handling in sql server, is now similar to programming languages like C#, and java.

Error Handling in SQL Server 2000 - @@Error
Error Handling in SQL Server 2005 & later - Try...Catch

Note: Sometimes, system functions that begin with two at signs (@@), are called as global variables. They are not variables and do not have the same behaviours as variables, instead they are very similar to functions.

tblProduct

ProductId	Name	UnitPrice	QtyAvailable
1	Laptops	2340	90
2	Desktops	3467	50

tblProductSales

ProductSalesId	ProductId	QuantitySold
1	1	10
2	1	10

RAISERROR('Error Message', ErrorSeverity, ErrorState)
Create and return custom errors
Severity level = 16 (indicates general errors that can be corrected by the user)
State = Number between 1 & 255. RAISERROR only generates errors with state from 1 through 127.

@@Error System function

@@Error returns a NON-ZERO value, if there is an error, otherwise ZERO, indicating that the previous SQL statement encountered no errors.

Note: @@ERROR is cleared and reset on each statement execution. Check it immediately following the statement being verified, or save it to a local variable that can be checked later.

```
Insert into tblProduct values(2, 'Mobile Phone', 1500, 100)
if(@@ERROR <> 0)
        Print 'Error Occured'
Else
        Print 'No Errors'
```

```
Insert into tblProduct values(2, 'Mobile Phone', 1500, 100)
--At this point @@ERROR will have a NON ZERO value
Select * from tblProduct
--At this point @@ERROR gets reset to ZERO, because the select
--statement successfully executed
if(@@ERROR <> 0)
        Print 'Error Occured'
Else
        Print 'No Errors'
```

@@Error System function

```
Declare @Error int
Insert into tblProduct values(2, 'Mobile Phone', 1500, 100)
Set @Error = @@ERROR
Select * from tblProduct
if(@Error <> 0)
        Print 'Error Occured'
Else
        Print 'No Errors'
```

Part 56 – Error Handling in SQL Server 2005 & Later versions

Error handling

With the introduction of Try/Catch blocks in SQL Server 2005, error handling in sql server, is now similar to programming languages like C#, and java.

Error Handling in SQL Server 2000 - @@Error
Error Handling in SQL Server 2005 & later - Try...Catch

tblProduct

ProductId	Name	UnitPrice	QtyAvailable
1	Laptops	2340	90
2	Desktops	3467	50

tblProductSales

ProductSalesId	ProductId	QuantitySold
1	1	10
2	1	10

RAISERROR('Error Message', ErrorSeverity, ErrorState)
Create and return custom errors
Severity level = 16 (indicates general errors that can be corrected by the user)
State = Number between 1 & 255. RAISERROR only generates errors with state from 1 through 127.

Try/Catch Syntax

```
--Syntax:
BEGIN TRY
     { Any set of SQL statements }
END TRY
BEGIN CATCH
     [ Optional: Any set of SQL statements ]
END CATCH
[Optional: Any other SQL Statements]
```

```
--System Functions to retrieve
--error information:
ERROR_NUMBER()
ERROR_MESSAGE()
ERROR_PROCEDURE()
ERROR_STATE()
ERROR_SEVERITY()
ERROR_LINE()
```

Any set of SQL statements, that can possibly throw an exception are wrapped between BEGIN TRY and END TRY blocks. If there is an exception in the TRY block, the control immediately, jumps to the CATCH block. If there is no exception, CATCH block will be skipped, and the statements, after the CATCH block are executed.

Errors trapped by a CATCH block are not returned to the calling application. If any part of the error information must be returned to the application, the code in the CATCH block must do so by using RAISERROR() function.

In the scope of the CATCH block, there are several system functions, that are used to retrieve more information about the error that occurred. These functions return NULL if they are executed outside the scope of the CATCH block. TRY/CATCH cannot be used in a user-defined functions.

Part 57 – Transactions in SQL Server

Transactions

What is a Transaction?

A transaction is a group of commands that change the data stored in a database. A transaction, is treated as a single unit. A transaction ensures that, either all of the commands succeed, or none of them. If one of the commands in the transaction fails, all of the commands fail, and any data that was modified in the database is rolled back. In this way, transactions maintain the integrity of data in a database.

Transaction processing follows these steps:

1. Begin a transaction.
2. Process database commands.
3. Check for errors.
   If errors occurred,
      rollback the transaction,
   else,
      commit the transaction

Note: NOT able to see the un-committed changes

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
```

Isolation Levels : Covered in a later session

Transactions

tblPhysicalAddress

AddressId	EmployeeNumber	HouseNumber	StreetAddress	City	PostalCode
1	101	#10	King Street	LONDOON	CR27DW

tblMailingAddress

AddressId	EmployeeNumber	HouseNumber	StreetAddress	City	PostalCode
1	101	#10	King Street	LONDOON	CR27DW

```
Create Procedure spUpdateAddress
as
Begin
  Begin Try
    Begin Transaction
      Update tblMailingAddress set City = 'LONDON'
      where AddressId = 1 and EmployeeNumber = 101

      Update tblPhysicalAddress set City = 'LONDON'
      where AddressId = 1 and EmployeeNumber = 101
    Commit Transaction
  End Try
  Begin Catch
    Rollback Transaction
  End Catch
End
```

Transactions

tblPhysicalAddress

AddressId	EmployeeNumber	HouseNumber	StreetAddress	City	PostalCode
1	101	#10	King Street	LONDOON	CR27DW

tblMailingAddress

AddressId	EmployeeNumber	HouseNumber	StreetAddress	City	PostalCode
1	101	#10	King Street	LONDOON	CR27DW

```
Create Procedure spUpdateAddress
as
Begin
  Begin Try
    Begin Transaction
      Update tblMailingAddress set City = 'LONDON'
      where AddressId = 1 and EmployeeNumber = 101

      Update tblPhysicalAddress set City = 'LONDON LONDON'
      where AddressId = 1 and EmployeeNumber = 101
    Commit Transaction
  End Try
  Begin Catch
    Rollback Transaction
  End Catch
End
```

Next Session: ACID Test

Part 58 – Transaction ACID Test

Transaction ACID Test

A transaction is a group of database commands that are treated as a single unit. A successful transaction must pass the "ACID" test, that is, it must be

Atomic- All statements in the transaction either completed successfully or they were all rolled back. The task that the set of operations represents is either accomplished or not, but in any case not left half-done.

Consistent – All data touched by the transaction is left in a logically consistent state. For example, if stock available numbers are decremented from tblProductTable, then, there has to be a related entry in tblProductSales table. The inventory can't just disappear.

tblProduct

ProductId	Name	UnitPrice	QtyAvailable
1	Laptops	2340	90
2	Desktops	3467	50

tblProductSales

ProductSalesId	ProductId	QuantitySold
1	1	10
2	1	10

```
Create Procedure spUpdateInventory_and_Sell
as
Begin
 Begin Try
  Begin Transaction
   Update tblProduct
   set QtyAvailable = (QtyAvailable - 10)
   where ProductId = 1

   Insert into tblProductSales values(3,1,10)
  Commit Transaction
 End Try
 Begin Catch
   Rollback Transaction
 End Catch
End
```

Transaction ACID Test

Isolated : The transaction must affect data without interfering with other concurrent transactions, or being interfered with by them. This prevents transactions from making changes to data based on uncommitted information, for example changes to a record that are subsequently rolled back. Most databases use locking to maintain transaction isolation.

Durable : Once a change is made, it is permanent. If a system error or power failure occurs before a set of commands is complete, those commands are undone and the data is restored to its original state once the system begins running again.

Next Session : Isolation Levels in SQL Server

Want to receive email alerts when new videos are uploaded, subscribe to my youtube channel: **www.YouTube.com/kudvenkat**

Part 59 – SubQueries in SQL Server

Subqueries in sql server

Let us understand subqueris with an example.

Example 1:
Write a query to retrieve products that are not at all sold?
Example 2:
Write a query to retrieve the NAME and TOTALQUANTITY sold?

From these examples, it should be very clear that, a subquery is simply a select statement, that returns a single value and can be nested inside a SELECT, UPDATE, INSERT, or DELETE statement. It is also possible to nest a subquery inside another subquery. According to MSDN, subqueries can be nested upto 32 levels.

Subqueries are always encolsed in paranthesis and are also called as inner queries, and the query containing the subquery is called as outer query.The columns from a table that is present only inside a subquery, cannot be used in the SELECT list of the outer query.

Next Video:
What to choose for performance? Queries that involve a subquery or a join

Part 60 – Co related SubQueries

Correlated subquery

If the subquery depends on the outer query for its values, then that sub query is called as correlated subquery.

In the where clause of the subquery below, "ProductId" column get it's value from tblProducts table that is present in the outer query.

```
Select [Name],
(Select SUM(QuantitySold) from tblProductSales
where ProductId = tblProducts.Id) as TotalQuantity
from tblProducts
order by Name
```

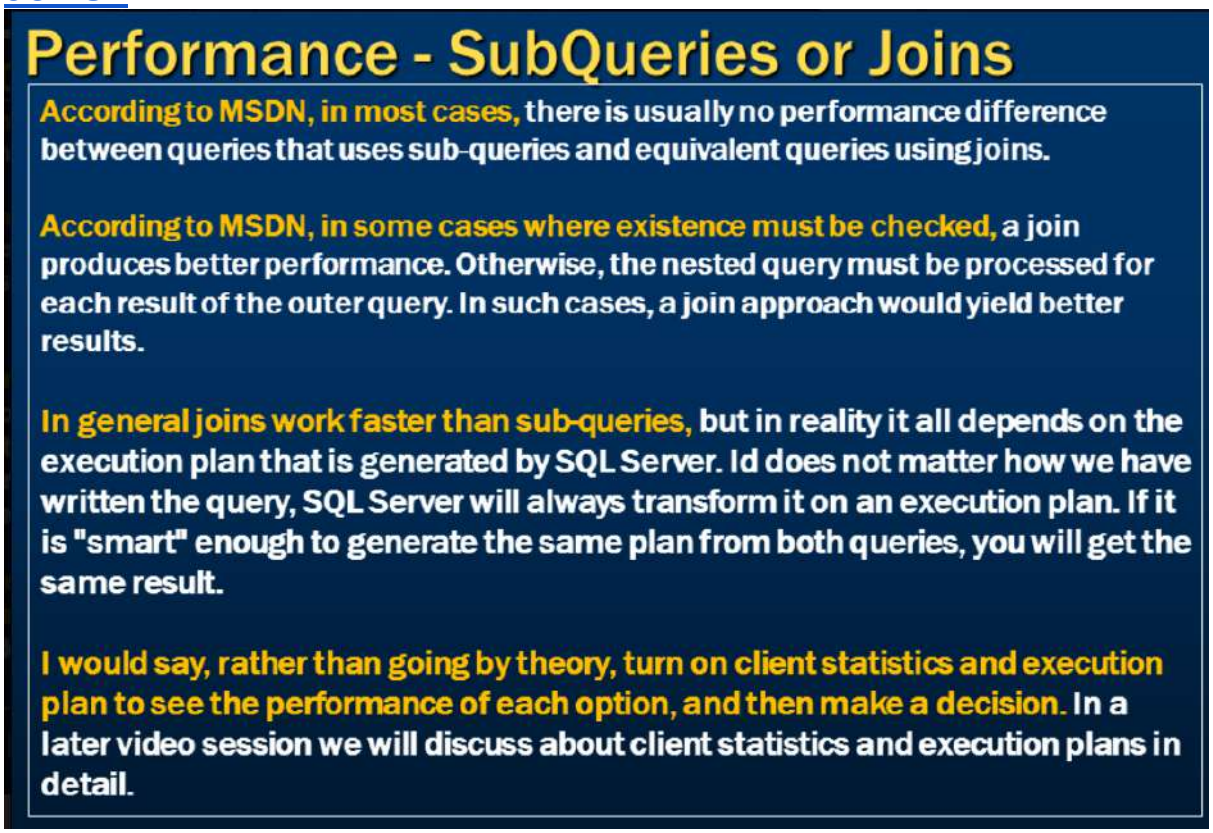
So, here the subquery is dependent on the outer query for it's value, hence this subquery is a correlated subquery.

Correlated subqueries get executed, once for every row that is selected by the outer query.

Corelated subquery, cannot be executed independently of the outer query.

- Part 61 – Creating a large table with random data for performance testing

- Part 62 – What to choose for performance - SubQueries or Joins?



Part 63 – Cursors in SQL Server

Cursors in SQL Server

Relational Database Management Systems, including sql server are very good at handling data in SETS. For example, the following "UPDATE" query, updates a set of rows that matches the condition in the "WHERE" clause at the same time.

```
Update tblProductSales Set UnitPrice = 50 where ProductId = 101
```

tblProducts

Id	Name	Description
1	Product - 1	Product - 1 Description
2	Product - 2	Product - 2 Description
3	Product - 3	Product - 3 Description
4	Product - 4	Product - 4 Description
5	Product - 5	Product - 5 Description

tblProductSales

Id	ProductId	UnitPrice	QuantitySold
1	5	5	3
2	4	23	4
3	3	31	2
4	4	93	9
5	5	72	5

However, if there is ever a need to process the rows, on a row-by-row basis, then cursors are your choice. Cursors are very bad for performance, and should be avoided always. Most of the time, cursors can be very easily replaced using joins.

Cursors in SQL Server

There are different types of cursors in sql server as listed below. We will talk about the differences between these cursor types in a later video session.

1. Forward-Only
2. Static
3. Keyset
4. Dynamic

DEMO

The cursor will loop thru each row in tblProductSales table. As there are 600,000 rows, to be processed on a row-by-row basis, it takes around 40 to 45 seconds on my machine. We can achieve this very easily using a join, and this will significantly increase the performance. We will discuss about this in our next video session.

Part 64 – Replacing cursors using joins

Replacing cursors using joins

In Part 63, we have discussed about cursors. The example, in Part 63, took around 45 seconds on my machine. Please watch Part 63, before proceeding with this video. In this video we will re-write the example, using a join.

```
Update tblProductSales
set UnitPrice =
    Case
        When Name = 'Product - 55' Then 155
        When Name = 'Product - 65' Then 165
        When Name like 'Product - 100%' Then 10001
    End
from tblProductSales
join tblProducts
on tblProducts.Id = tblProductSales.ProductId
Where Name = 'Product - 55' or Name = 'Product - 65' or
Name like 'Product - 100%'
```

When I executed this query, on my machine it took less than a second. Where as the same thing using a cursor took 45 seconds.

Just imagine the amount of impact cursors have on performance. Cursors should be used as your last option.

Most of the time cursors can be very easily replaced using joins.

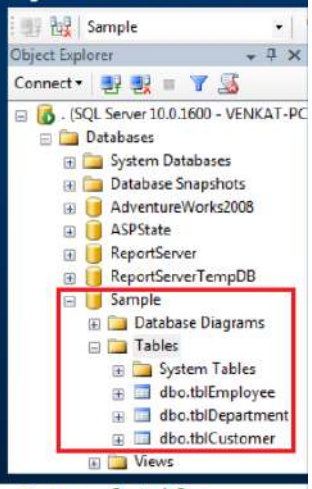
Part 65 – List all tables in a sql server database using a query

T-SQL query to list all tables

Object explorer with in SQL Server management studio can be used to get the list of tables in a specific database. However, if we have to write a query to achieve the same, there are 3 system views that we can use.

1. SYSOBJECTS - SQL Server 2000, 2005 & 2008
2. SYS.TABLES - SQL Server 2005 & 2008
3. INFORMATION_SCHEMA.TABLES - SQL Server 2005 & 2008

```
-- Gets the list of tables only
Select * from SYSOBJECTS where XTYPE='U'

-- Gets the list of tables only
Select * from SYS.TABLES

-- Gets the list of tables and views
select * from INFORMATION_SCHEMA.TABLES
```



To get the list of different object types (XTYPE) in a database

```
Select Distinct XTYPE from SYSOBJECTS
```

```
IT - Internal table
P  - Stored procedure
PK - PRIMARY KEY constraint
S  - System table
U  - User table
V  - View
```

MSDN link for all possible XTYPE column values

http://msdn.microsoft.com/en-us/library/ms177596.aspx

Part 66 – Writing re-runnable SQL Server scripts

Re-runnable SQL Server Scripts

What is a re-runnable SQL Script?

A re-runnable script is a script, that, when run more than once, will not throw errors.

```
-- This script is not re-runnable
USE [Sample]
Create table tblEmployee
(
    ID int identity primary key,
    Name nvarchar(100),
    Gender nvarchar(10),
    DateOfBirth DateTime
)
```

To make this script re-runnable

1. Check for the existence of the table
2. Create the table if it does not exist
3. Else print a message stating, the table already exists

```
Use [Sample]
If not exists (select * from information_schema.tables where table_name = 'tblEmployee')
Begin
    Create table tblEmployee
    (
        ID int identity primary key,
        Name nvarchar(100),
        Gender nvarchar(10),
        DateOfBirth DateTime
    )
    Print 'Table tblEmployee successfully created'
End
Else
Begin
    Print 'Table tblEmployee already exists'
End
```

Re-runnable SQL Server Scripts

Sql server built-in function OBJECT_ID(), can also be used to check for the existence

```
IF OBJECT_ID('tblEmployee') IS NULL
Begin
   -- Create Table Script
   Print 'Table tblEmployee created'
End
Else
Begin
   Print 'Table tblEmployee already exists'
End
```

Drop (if the table already exists) and re-create

```
Use [Sample]
IF OBJECT_ID('tblEmployee') IS NOT NULL
Begin
    Drop Table tblEmployee
End
Create table tblEmployee
(
    ID int identity primary key,
    Name nvarchar(100),
    Gender nvarchar(10),
    DateOfBirth DateTime
)
```

Re-runnable SQL Server Scripts

This script is not re-runnable because, if the column exists we get a script error.

```
Use [Sample]
ALTER TABLE tblEmployee
ADD EmailAddress nvarchar(50)
```

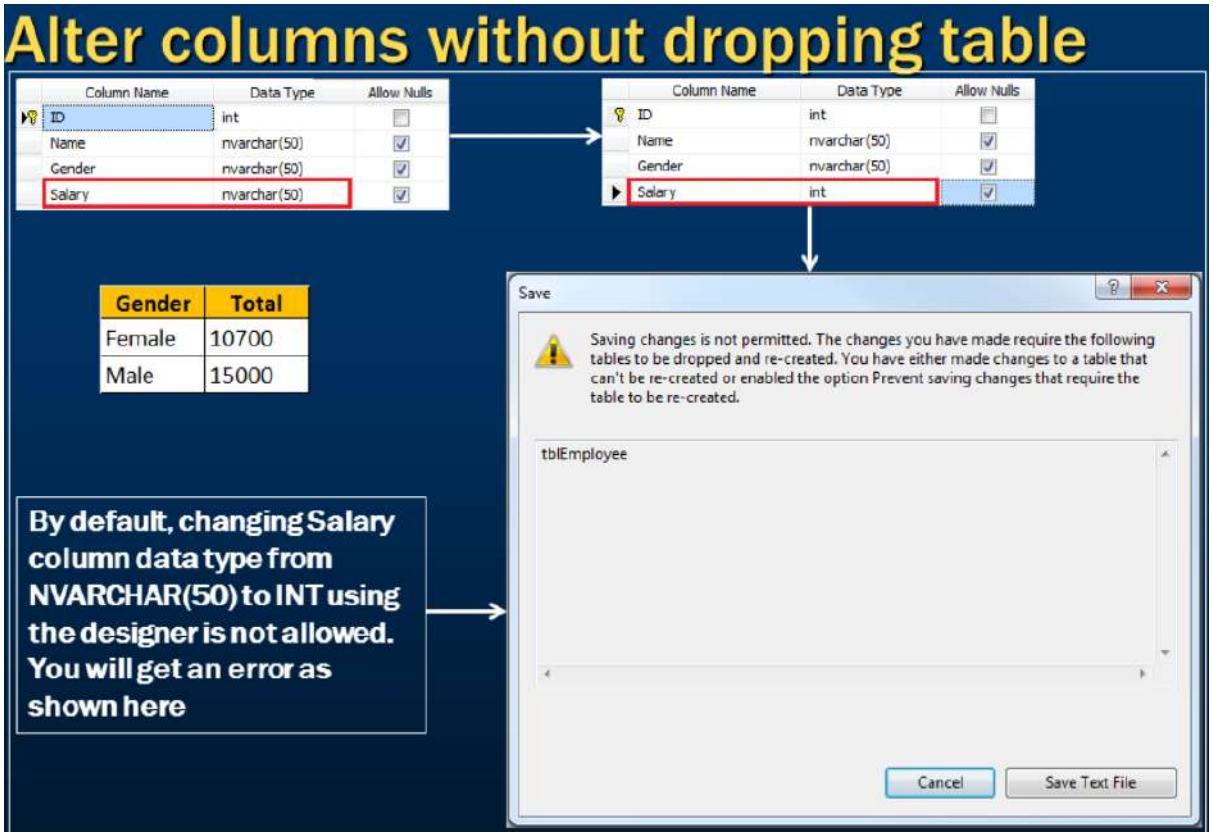
To make this script re-runnable, check for the column existence

```
Use [Sample]
if not exists(Select * from INFORMATION_SCHEMA.COLUMNS where COLUMN_NAME='EmailAddress'
and TABLE_NAME = 'tblEmployee' and TABLE_SCHEMA='dbo')
Begin
    ALTER TABLE tblEmployee
    ADD EmailAddress nvarchar(50)
End
Else
BEgin
    Print 'Column EmailAddress already exists'
End
```

Col_length() function can also be used to check for the existence of a column

```
If col_length('tblEmployee','EmailAddress') is not null
Begin
    Print 'Column already exists'
End
Else
Begin
    Print 'Column does not exist'
End
```

Part 67 – Alter database table columns without dropping table



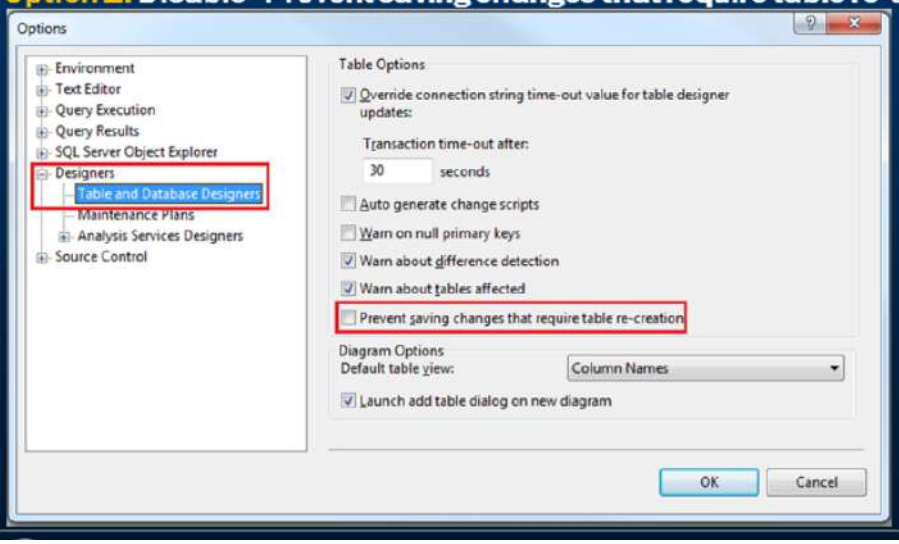
Alter columns without dropping table

How to alter the database table definition without the need to drop, re-create and again populate the table with data?

Option 1: Use a SQL query to alter the column as shown below

```
ALTER TABLE tblEmployee
ALTER COLUMN Salary INT
```

Option 2: Disable "Prevent saving changes that require table re-creation" option



[Part 68 - Optional parameters in sql server stored procedures]

Optional Stored Procedures Parameters

Parameters of a SQL Server stored procedure can be made optional by specifying default values.

Name, Email, Age and Gender parameters of spSearchEmployees stored procedure are optional. We have set defaults for all the parameters, and in the "WHERE" clause we are checking if the respective parameter IS NULL.

```
Create Proc spSearchEmployees
@Name nvarchar(50) = NULL,
@Email nvarchar(50) = NULL,
@Age int = NULL,
@Gender nvarchar(50) = NULL
as
Begin
    Select * from tblEmployee where
    (Name = @Name OR @Name IS NULL) AND
    (Email = @Email OR @Email IS NULL) AND
    (Age = @Age OR @Age IS NULL) AND
    (Gender = @Gender OR @Gender IS NULL)
End
```

Testing the stored procedure

```
Execute spSearchEmployees -- Return all the rows
```

```
Execute spSearchEmployees @Gender = 'Male'
```

Id	Name	Email	Age	Gender	HireDate
2	James Histo	James.Histo@test.com	33	Male	2008-07-13
4	Paul Sensit	Paul.Sensit@test.com	29	Male	2007-10-23

```
Execute spSearchEmployees @Gender = 'Male', @Age = 29
```

Id	Name	Email	Age	Gender	HireDate
4	Paul Sensit	Paul.Sensit@test.com	29	Male	2007-10-23

Part 69 - Merge in SQL Server

Merge in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

What is the use of MERGE statement in SQL Server

Merge statement introduced in SQL Server 2008 allows us to perform Inserts, Updates and Deletes in one statement. This means we no longer have to use multiple statements for performing Insert, Update and Delete

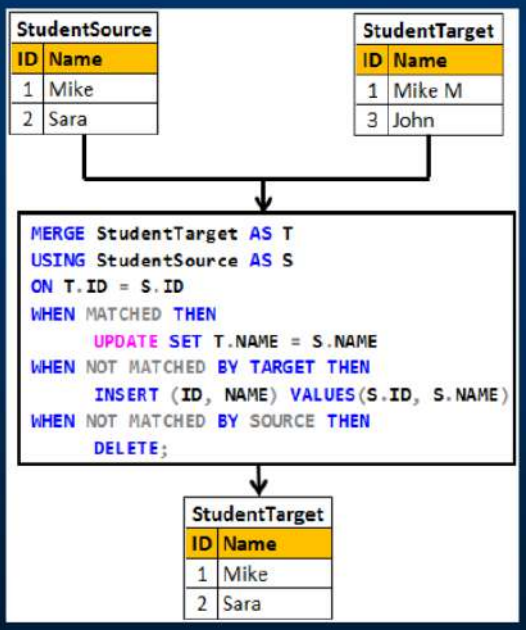
With merge statement we require 2 tables

1. Source Table - Contains the changes that needs to be applied to the target table
2. Target Table - The table that require changes (Inserts, Updates and Deletes)

Merge statement joins the target table to the source table by using a common column in both the tables. Based on how the rows match up, we can perform insert, update, and delete on the target table

Merge in SQL Server - Example 1

```
MERGE [TARGET] AS T
USING [SOURCE] AS S
   ON [JOIN_CONDITIONS]
 WHEN MATCHED THEN
      [UPDATE STATEMENT]
 WHEN NOT MATCHED BY TARGET THEN
      [INSERT STATEMENT]
 WHEN NOT MATCHED BY SOURCE THEN
      [DELETE STATEMENT]
```



Part 70 - SQL Server concurrent transactions

SQL Server Concurrent Transactions

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

What is a transaction

A transaction is a group of commands that change the data stored in a database. A transaction, is treated as a single unit of work

Id	AccountName	Balance
1	Mark	1000
2	Mary	1000

```
-- Transfer $100 from Mark to Mary Account
BEGIN TRY
     BEGIN TRANSACTION
          UPDATE Accounts SET Balance = Balance - 100 WHERE Id = 1
          UPDATE Accounts SET Balance = Balance + 100 WHERE Id = 2
     COMMIT TRANSACTION
     PRINT 'Transaction Committed'
END TRY
BEGIN CATCH
     ROLLBACK TRANSACTION
     PRINT 'Transaction Rolled back'
END CATCH
```

SQL Server Concurrent Transactions

Common concurrency problems

- Dirty Reads
- Lost Updates
- Nonrepeatable Reads
- Phantom Reads

SQL Server Transaction Isolation Levels

- Read Uncommitted
- Read Committed
- Repeatable Read
- Snapshot
- Serializable

Isolation Level	Dirty Reads	Lost Update	Nonrepeatable Reads	Phantom Reads
Read Uncommitted	Yes	Yes	Yes	Yes
Read Committed	No	Yes	Yes	Yes
Repeatable Read	No	No	No	Yes
Snapshot	No	No	No	No
Serializable	No	No	No	No

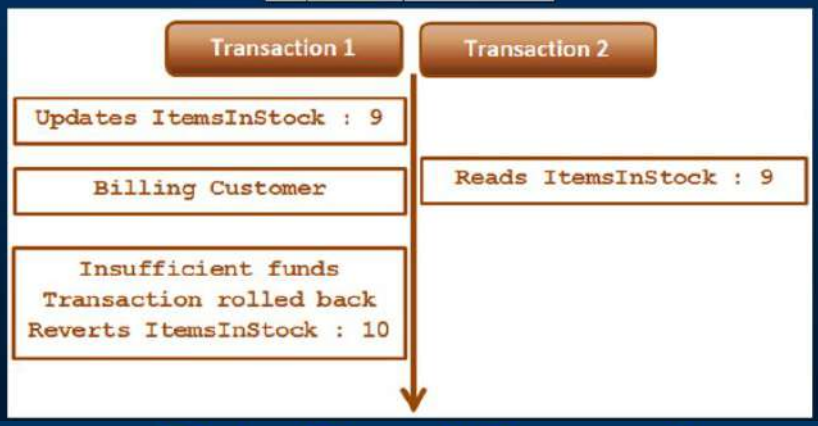
Part 71 - SQL Server dirty read example

SQL Server Dirty Read Example

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

A dirty read happens when one transaction is permitted to read data that has been modified by another transaction that has not yet been committed. In most cases this would not cause a problem. However, if the first transaction is rolled back after the second reads the data, the second transaction has dirty data that does not exist anymore

Id	Product	ItemsInStock
1	iPhone	10



SQL Server Dirty Read Example

Id	Product	ItemsInStock
1	iPhone	10

```
--Transaction 1 :
Begin Tran
Update tblInventory set ItemsInStock = 9 where Id=1

-- Billing the customer
Waitfor Delay '00:00:15'
-- Insufficient Funds. Rollback transaction
Rollback Transaction
```

```
--Transaction 2 :
Set Transaction Isolation Level Read Uncommitted
Select * from tblInventory where Id=1
```

Read Uncommitted transaction isolation level is the only isolation level that has dirty read side effect. This is the least restrictive of all the isolation levels. When this transaction isolation level is set, it is possible to read uncommitted or dirty data. Another option to read dirty data is by using NOLOCK table hint. The query below is equivalent to the query in Transaction 2.

```
Select * from tblInventory (NOLOCK) where Id=1
```

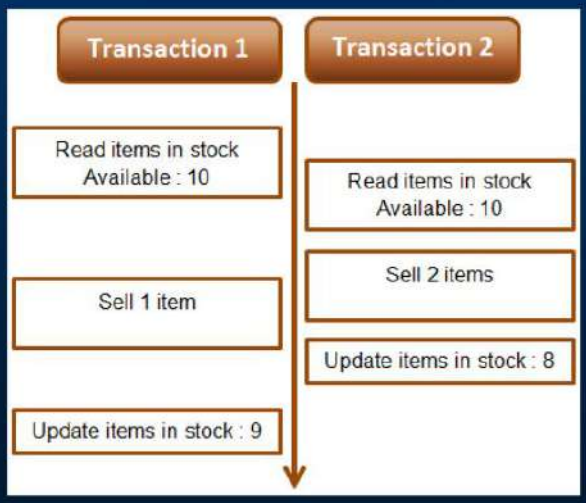
Part 72 - SQL Server lost update problem

SQL Server Lost Update Problem

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Lost update problem happens when 2 transactions read and update the same data

Id	Product	ItemsInStock
1	iPhone	10



SQL Server Lost Update Problem

```
-- Transaction 1
Begin Tran
Declare @ItemsInStock int

Select @ItemsInStock = ItemsInStock
from tblInventory where Id=1

-- Transaction takes 10 seconds
Waitfor Delay '00:00:10'
Set @ItemsInStock = @ItemsInStock - 1

Update tblInventory
Set ItemsInStock = @ItemsInStock
where Id=1

Print @ItemsInStock
Commit Transaction
```

```
-- Transaction 2
Begin Tran
Declare @ItemsInStock int

Select @ItemsInStock = ItemsInStock
from tblInventory where Id=1

-- Transaction takes 1 second
Waitfor Delay '00:00:1'
Set @ItemsInStock = @ItemsInStock - 2

Update tblInventory
Set ItemsInStock = @ItemsInStock
where Id=1

Print @ItemsInStock
Commit Transaction
```

Id	Product	ItemsInStock
1	iPhone	10

Transaction 1 is processing an order for 1 iPhone, while Transaction 2 is processing an order for 2 iPhones. At the end of both the transactions ItemsInStock must be 7, but we have a value of 9. This is because Transaction 1 silently overwrites the update of Transaction 2. This is called the lost update problem

SQL Server Lost Update Problem

Isolation Level	Dirty Reads	Lost Update	Nonrepeatable Reads	Phantom Reads
Read Uncommitted	Yes	Yes	Yes	Yes
Read Committed	No	Yes	Yes	Yes
Repeatable Read	No	No	No	Yes
Snapshot	No	No	No	No
Serializable	No	No	No	No

Both Read Uncommitted and Read Committed transaction isolation levels have the lost update side effect.

Repeatable Read, Snapshot, and Serializable isolation levels does not have this side effect.

The repeatable read isolation level uses additional locking on rows that are read by the current transaction, and prevents them from being updated or deleted elsewhere. This solves the lost update problem.

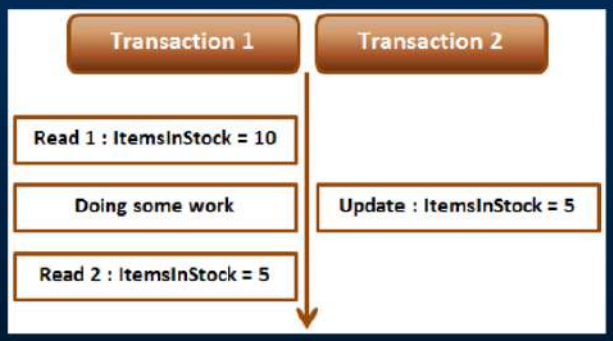
Part 73 - Non repeatable read example in sql server

Non repeatable read example

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Non repeatable read happens when one transaction reads the same data twice and another transaction updates that data in between the first and second read of transaction one.

Id	Product	ItemsInStock
1	iPhone	10



Non repeatable read example

```
-- Transaction 1
Begin Transaction
Select ItemsInStock
from tblInventory where Id = 1

-- Do Some work
waitfor delay '00:00:10'

Select ItemsInStock
from tblInventory where Id = 1
Commit Transaction
```

```
-- Transaction 2
Update tblInventory
set ItemsInStock = 5 where Id = 1
```

Repeatable read or any other higher isolation level should solve the non-repeatable read problem. To fix the non-repeatable read problem, set transaction isolation level of Transaction 1 to repeatable read. This will ensure that the data that Transaction 1 has read, will be prevented from being updated or deleted elsewhere. This solves the non-repeatable read problem

Isolation Level	Dirty Reads	Lost Update	Nonrepeatable Reads	Phantom Reads
Read Uncommitted	Yes	Yes	Yes	Yes
Read Committed	No	Yes	Yes	Yes
Repeatable Read	No	No	No	Yes
Snapshot	No	No	No	No
Serializable	No	No	No	No

Part 74 - Phantom reads example in sql server

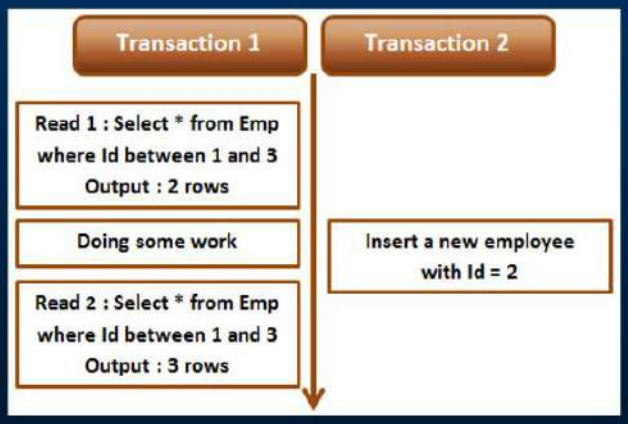
Phantom reads example in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Phantom read happens when one transaction executes a query twice and it gets a different number of rows in the result set each time. This happens when a second transaction inserts a new row that matches the WHERE clause of the query executed by the first transaction.

Employees Table

Id	Name
1	Mark
3	Sara
100	Mary



Phantom reads example in SQL Server

```
-- Transaction 1
Begin Transaction

Select * from tblEmployees where Id between 1 and 3

-- Do Some work
waitfor delay '00:00:10'

Select * from tblEmployees where Id between 1 and 3
Commit Transaction
```

```
-- Transaction 2
Insert into tblEmployees
values(2, 'Marcus')
```

To fix the phantom read problem, set transaction isolation level of Transaction 1 to serializable. This will place a range lock on the rows between 1 and 3, which prevents any other transaction from inserting new rows with in that range.

Isolation Level	Dirty Reads	Lost Update	Nonrepeatable Reads	Phantom Reads
Read Uncommitted	Yes	Yes	Yes	Yes
Read Committed	No	Yes	Yes	Yes
Repeatable Read	No	No	No	Yes
Snapshot	No	No	No	No
Serializable	No	No	No	No

Repeatable Read v/s Serializable

Repeatable read prevents only non-repeatable read. Repeatable read isolation level ensures that the data that one transaction has read, will be prevented from being updated or deleted by any other transaction, but it doe not prevent new rows from being inserted by other transactions resulting in phantom read concurrency problem.

Serializable prevents both non-repeatable read and phantom read problems. Serializable isolation level ensures that the data that one transaction has read, will be prevented from being updated or deleted by any other transaction. It also prevents new rows from being inserted by other transactions, so this isolation level prevents both non-repeatable read and phantom read problems.

Part 75 - Snapshot isolation level in sql server

Snapshot Isolation Level in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Just like serializable isolation level, snapshot isolation level does not have any concurrency side effects

Isolation Level	Dirty Reads	Lost Update	Nonrepeatable Reads	Phantom Reads
Read Uncommitted	Yes	Yes	Yes	Yes
Read Committed	No	Yes	Yes	Yes
Repeatable Read	No	No	No	Yes
Snapshot	No	No	No	No
Serializable	No	No	No	No

Difference between serializable and snapshot isolation levels

Serializable isolation is implemented by acquiring locks which means the resources are locked for the duration of the current transaction. This isolation level does not have any concurrency side effects but at the cost of significant reduction in concurrency.

Snapshot isolation doesn't acquire locks, it maintains versioning in Tempdb. Since, snapshot isolation does not lock resources, it can significantly increase the number of concurrent transactions while providing the same level of data consistency as serializable isolation does.

Snapshot Isolation Level in SQL Server

Example : Reading data with Snapshot Isolation level

```
--Transaction 1
Set transaction isolation level serializable
Begin Transaction
Update tblInventory set ItemsInStock = 5 where Id = 1
waitfor delay '00:00:10'
Commit Transaction
```

```
-- Transaction 2
-- Enable snapshot isloation for the database
Alter database SampleDB SET ALLOW_SNAPSHOT_ISOLATION ON
-- Set the transaction isolation level to snapshot
Set transaction isolation level snapshot

Select ItemsInStock from tblInventory where Id = 1
```

From the first window execute Transaction 1 code and from the second window, execute Transaction 2 code. Notice that Transaction 2 is not blocked and returns the data from the database as it was before Transaction 1 has started

Snapshot Isolation Level in SQL Server

Example : Modifying data with Snapshot Isolation level

```
--Transaction 1
Set transaction isolation level serializable
Begin Transaction
Update tblInventory set ItemsInStock = 5 where Id = 1
waitfor delay '00:00:10'
Commit Transaction
```

```
-- Transaction 2
-- Enable snapshot isloation for the database
Alter database SampleDB SET ALLOW_SNAPSHOT_ISOLATION ON
-- Set the transaction isolation level to snapshot
Set transaction isolation level snapshot

Update tblInventory set ItemsInStock = 8 where Id = 1
```

Transaction 2 is blocked while Transaction 1 is in progress. When Transaction 1 completes, Transaction 2 fails with an error.

[Part 76 - Read committed snapshot isolation level in sql server]

Read committed snapshot isolation level

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Read committed snapshot isolation level is not a different isolation level. It is a different way of implementing Read committed isolation level. One problem we have with Read Committed isolation level is that, it blocks the transaction if it is trying to read the data, that another transaction is updating at the same time.

To use READ_COMMITTED_SNAPSHOT isolation, enable it at the database level

```
Alter database SampleDB SET READ_COMMITTED_SNAPSHOT ON
```

```
--Transaction 1
Set transaction isolation level Read Committed
Begin Transaction
Update tblInventory set ItemsInStock = 5 where Id = 1
waitfor delay '00:00:10'
Commit Transaction

-- Transaction 2
Set transaction isolation level read committed
Begin Transaction
Select ItemsInStock from tblInventory where Id = 1
Commit Transaction
```

Id	Name	ItemsInStock
1	iPhone	10

[Part 77 - Difference between snapshot isolation and read committed snapshot]

Snapshot v/s Read Committed Snapshot

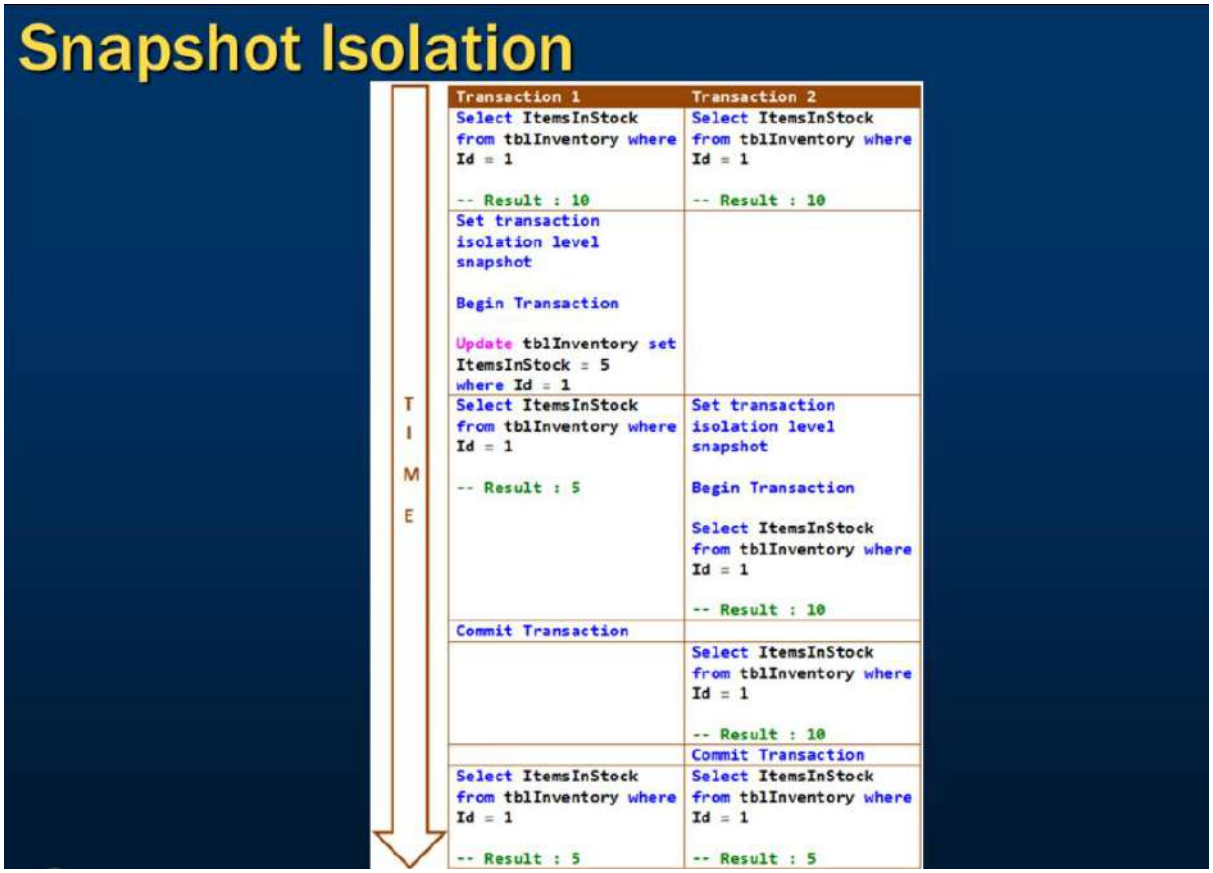
If you are in need of the DVD with all the videos and PPT's, please visit
http://pragimtech.com/order.aspx

Read Committed Snapshot Isolation	Snapshot Isolation
No update conflicts	Vulnerable to update conflicts
Works with existing applications without requiring any change to the application	Application change may be required to use with an existing application
Can be used with distributed transactions	Cannot be used with distributed transactions
Provides statement-level read consistency	Provides transaction-level read consistency

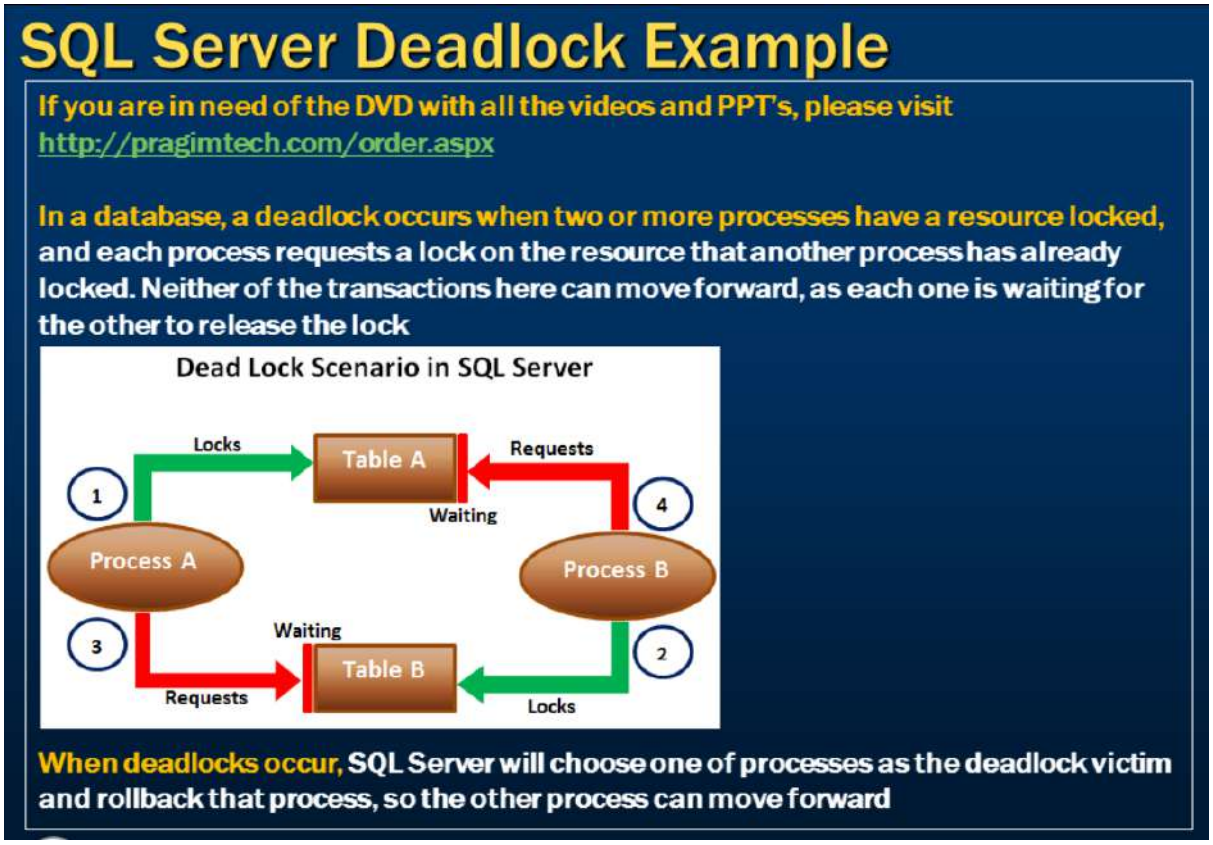
Read Committed Snapshot Isolation

TIME

Transaction 1	Transaction 2
Select ItemsInStock from tblInventory where Id = 1 -- Result : 10	Select ItemsInStock from tblInventory where Id = 1 -- Result : 10
Set transaction isolation level read committed Begin Transaction Update tblInventory set ItemsInStock = 5 where Id = 1	
Select ItemsInStock from tblInventory where Id = 1 -- Result : 5	Set transaction isolation level read committed Begin Transaction Select ItemsInStock from tblInventory where Id = 1 -- Result : 10
Commit Transaction	
	Select ItemsInStock from tblInventory where Id = 1 -- Result : 5
	Commit Transaction
Select ItemsInStock from tblInventory where Id = 1 -- Result : 5	Select ItemsInStock from tblInventory where Id = 1 -- Result : 5



- [Part 78 -SQL Server deadlock example]



SQL Server Deadlock Example

Table A	
Id	Name
1	Mark

Table B	
Id	Name
1	Mary

```
-- Transaction 1
Begin Tran
Update TableA Set Name = 'Mark Transaction 1' where Id = 1

-- From Transaction 2 window execute the first update statement

Update TableB Set Name = 'Mary Transaction 1' where Id = 1

-- From Transaction 2 window execute the second update statement
Commit Transaction
```

```
-- Transaction 2
Begin Tran
Update TableB Set Name = 'Mark Transaction 2' where Id = 1

-- From Transaction 1 window execute the second update statement

Update TableA Set Name = 'Mary Transaction 2' where Id = 1

-- After a few seconds notice that one of the transactions complete
-- successfully while the other transaction is made the deadlock victim
Commit Transaction
```

Next video: Criteria SQL Server uses to choose a deadlock victim

- [Part 79 - SQL Server deadlock victim selection]

SQL Server deadlock victim selection

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

How SQL Server detects deadlocks

Lock monitor thread in SQL Server, runs every 5 seconds by default to detect if there are any deadlocks. If the lock monitor thread finds deadlocks, the deadlock detection interval will drop from 5 seconds to as low as 100 milliseconds depending on the frequency of deadlocks. If the lock monitor thread stops finding deadlocks, the Database Engine increases the intervals between searches to 5 seconds

What happens when a deadlock is detected

When a deadlock is detected, the Database Engine ends the deadlock by choosing one of the threads as the deadlock victim. The deadlock victim's transaction is then rolled back and returns a 1205 error to the application. Rolling back the transaction of the deadlock victim releases all locks held by that transaction. This allows the other transactions to become unblocked and move forward.

What is DEADLOCK_PRIORITY

What is DEADLOCK_PRIORITY

By default, SQL Server chooses a transaction as the deadlock victim that is least expensive to roll back. However, a user can specify the priority of sessions in a deadlock situation using the SET DEADLOCK_PRIORITY statement. The session with the lowest deadlock priority is chosen as the deadlock victim

Example : `SET DEADLOCK_PRIORITY NORMAL`

DEADLOCK_PRIORITY

1. The default is Normal
2. Can be set to LOW, NORMAL, or HIGH
3. Can also be set to a integer value in the range of -10 to 10
   - LOW : -5
   - NORMAL : 0
   - HIGH : 5

Deadlock Victim Selection Criteria

What is the deadlock victim selection criteria

1. If the DEADLOCK_PRIORITY is different, the session with the lowest priority is selected as the victim

2. If both the sessions have the same priority, the transaction that is least expensive to rollback is selected as the victim

3. If both the sessions have the same deadlock priority and the same cost, a victim is chosen randomly

Deadlock Victim Selection - Example 1

```sql
-- Transaction 1
Begin Tran
Update TableA Set Name = Name + ' Transaction 1' where Id IN (1, 2, 3, 4, 5)

-- From Transaction 2 window execute the first update statement

Update TableB Set Name = Name + ' Transaction 1' where Id = 1

-- From Transaction 2 window execute the second update statement
Commit Transaction
```

```sql
-- Transaction 2
Begin Tran
Update TableB Set Name = Name + ' Transaction 2' where Id = 1

-- From Transaction 1 window execute the second update statement

Update TableA Set Name = Name + ' Transaction 2' where Id IN (1, 2, 3, 4, 5)

-- After a few seconds notice that this transaction will be chosen as the deadlock
-- victim as it is less expensive to rollback this transaction than Transaction 1
Commit Transaction
```

Deadlock Victim Selection - Example 2

```sql
-- Transaction 1
Begin Tran
Update TableA Set Name = Name + ' Transaction 1' where Id IN (1, 2, 3, 4, 5)

-- From Transaction 2 window execute the first update statement

Update TableB Set Name = Name + ' Transaction 1' where Id = 1

-- From Transaction 2 window execute the second update statement
Commit Transaction
```

```sql
-- Transaction 2
SET DEADLOCK_PRIORITY HIGH
GO
Begin Tran
Update TableB Set Name = Name + ' Transaction 2' where Id = 1

-- From Transaction 1 window execute the second update statement

Update TableA Set Name = Name + ' Transaction 2' where Id IN (1, 2, 3, 4, 5)

-- After a few seconds notice that Transaction 2 will be chosen as the deadlock
-- victim as it's DEADLOCK_PRIORITY (Normal) is lower than the DEADLOCK_PRIORITY
-- this transaction (HIGH)
Commit Transaction
```

Part 80 - Logging deadlocks in sql server

Logging deadlocks in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

When deadlocks occur, SQL Server chooses one of the transactions as the deadlock victim and rolls it back. There are several ways in SQL Server to track down the queries that are causing deadlocks. One of the options is to use SQL Server trace flag 1222 to write the deadlock information to the SQL Server error log.

```
-- Set SQL Server trace flag 1222
DBCC Traceon(1222, -1)

-- Check the status of the trace flag
DBCC TraceStatus(1222, -1)

-- Turn off the trace flag
DBCC Traceoff(1222, -1)
```

```
-- To read the error log
execute sp_readerrorlog
```

Please note: -1 parameter indicates that the trace flag must be set at the global level. If you omit -1 parameter the trace flag will be set only at the session level

Part 81 - SQL Server deadlock analysis and prevention

Deadlock analysis and prevention

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

The deadlock information in the error log has three sections

Section	Description
Deadlock Victim	Contains the ID of the process that was selected as the deadlock victim and killed by SQL Server
Process List	Contains the list of the processes that participated in the deadlock
Resource List	Contains the list of the resources (database objects) owned by the processes involved in the deadlock

Process List Section :

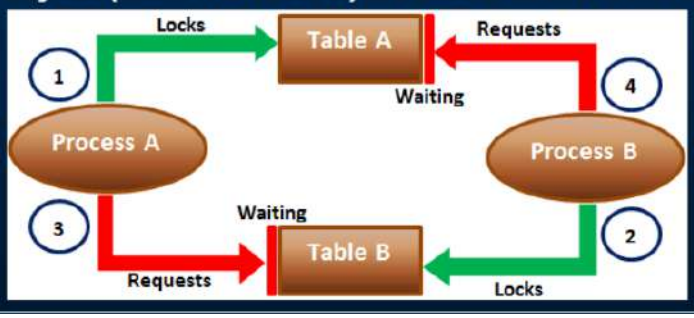
Node	Description
loginname	The loginname associated with the process
isolationlevel	What isolation level is used
procname	The stored procedure name
Inputbuf	The code the process is executing when the deadlock occured

Deadlock analysis and prevention

Resource List Section

Node	Description
objectname	Fully qualified name of the resource involved in the deadlock
owner-list	Contains (owner id) the id of the owning process and the lock mode it has acquired on the resource. lock mode determines how the resource can be accessed by concurrent transactions. S for Shared lock, U for Update lock, X for Exclusive lock etc
waiter-list	Contains (waiter id) the id of the process that wants to acquire a lock on the resource and the lock mode it is requesting

To prevent the deadlock that we have in our case, we need to ensure that database objects (Table A & Table B) are accessed in the same order every time



Part 82 - Capturing deadlocks in sql profiler

Analyzing Deadlock Graph

1. The oval on the graph, with the blue cross, represents the transaction that was chosen as the deadlock victim by SQL Server.

2. The oval on the graph without blue cross represents the transaction that completed successfully.

3. When you move the mouse pointer over the oval, you can see the SQL code that was running that caused the deadlock.

4. The oval symbols represent the process nodes

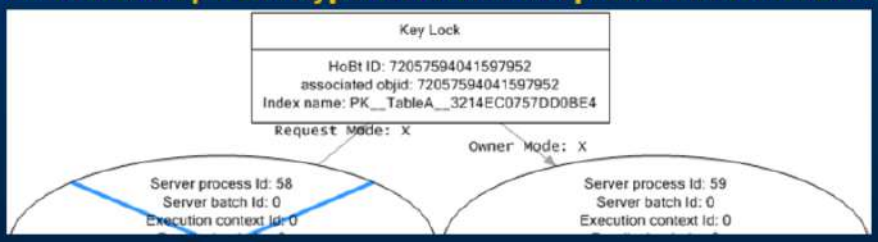
- Server Process Id : If you are using SQL Server Management Studio you can see the server process id on information bar at the bottom
- Deadlock Priority : If you have not set DEADLOCK PRIORITY explicitly using SET DEADLOCK PRIORITY statement, then both the processes should have the same default deadlock priority NORMAL (0)
- Log Used : The transaction log space used. If a transaction has used a lot of log space then the cost to roll it back is also more. So the transaction that has used the least log space is killed and rolled back

Analyzing Deadlock Graph

5. The rectangles represent the resource nodes.
   a) HoBt ID : Heap Or Binary Tree ID. Using this ID query sys.partitions view to find the database objects involved in the deadlock.

```
SELECT object_name([object_id])
FROM sys.partitions
WHERE hobt_id = 72057594041663488
```

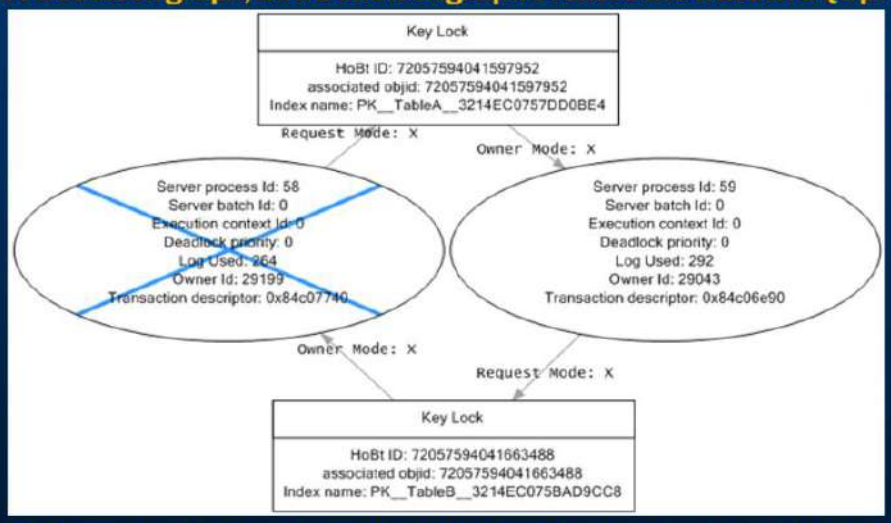
6. The arrows represent types of locks each process has on each resource node.



Capturing Deadlocks in SQL Profiler

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

To capture deadlock graph, add Deadlock graph event to the trace in SQL profiler

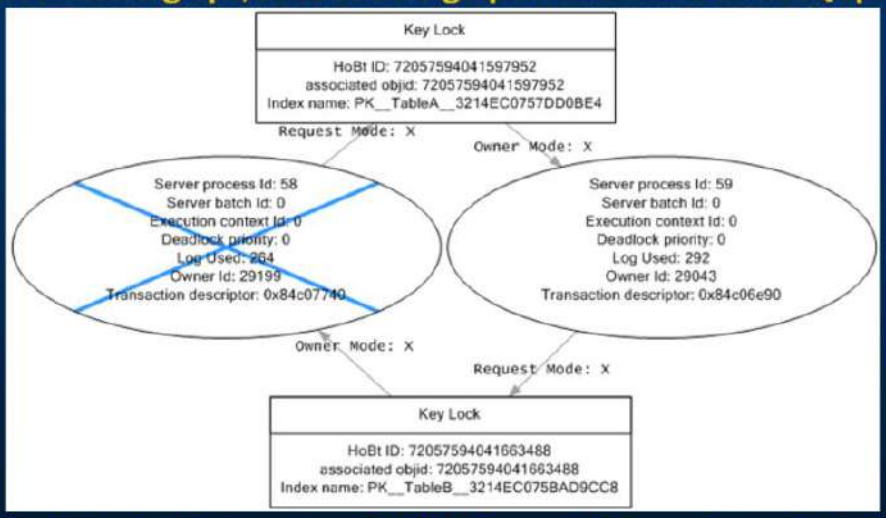


The deadlock graph data is captured in XML format which can be extracted to a physical file for later analysis. This is similar to data captured using trace flag 1222.

Capturing Deadlocks in SQL Profiler

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

To capture deadlock graph, add Deadlock graph event to the trace in SQL profiler



The deadlock graph data is captured in XML format which can be extracted to a physical file for later analysis. This is similar to data captured using trace flag 1222.

Analyzing Deadlock Graph

1. The oval on the graph, with the blue cross, represents the transaction that was chosen as the deadlock victim by SQL Server.

2. The oval on the graph without blue cross represents the transaction that completed successfully.

3. When you move the mouse pointer over the oval, you can see the SQL code that was running that caused the deadlock.

4. The oval symbols represent the process nodes

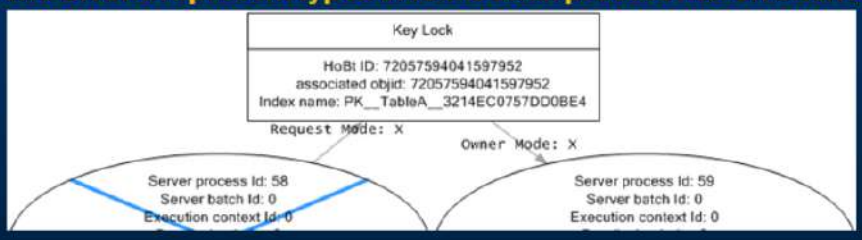
- Server Process Id : If you are using SQL Server Management Studio you can see the server process id on information bar at the bottom
- Deadlock Priority : If you have not set DEADLOCK PRIORITY explicitly using SET DEADLOCK PRIORITY statement, then both the processes should have the same default deadlock priority NORMAL (0)
- Log Used : The transaction log space used. If a transaction has used a lot of log space then the cost to roll it back is also more. So the transaction that has used the least log space is killed and rolled back

Analyzing Deadlock Graph

5. The rectangles represent the resource nodes.
   a) HoBt ID : Heap Or Binary Tree ID. Using this ID query sys.partitions view to find the database objects involved in the deadlock.

```
SELECT object_name([object_id])
FROM sys.partitions
WHERE hobt_id = 72057594041663488
```

6. The arrows represent types of locks each process has on each resource node.



- [Part 83 - SQL Server deadlock error handling]

SQL Server Deadlock Error Handling

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

```
Alter procedure spTransaction1
as
Begin
    Begin Tran
    Begin Try
        Update TableA Set Name = 'Mark Transaction 1' where Id = 1
        Waitfor delay '00:00:05'
        Update TableB Set Name = 'Mary Transaction 1' where Id = 1
        -- If both the update statements succeeded.
        -- No Deadlock occurred. So commit the transaction.
        Commit Transaction
        Select 'Transaction Successful'
    End Try
    Begin Catch
        -- Check if the error is deadlock error
        If(ERROR_NUMBER() = 1205)
        Begin
            Select 'Deadlock. Transaction failed. Please retry'
        End
        -- Rollback the transaction
        Rollback
    End Catch
End
```

- //Dotnet topics Part 84 - Handling deadlocks in ado.net
- Part 85 - Retry logic for deadlock exceptions

Part 86 - How to find blocking queries in sql server

How to find blocking queries in sql server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Blocking occurs if there are open transactions

DBCC OpenTran will display only the oldest active transaction. It is not going to show you all the open transactions

SQL Server process can be killed using

1. SQL Server Activity Monitor
2. Using SQL command `KILL Process_ID`

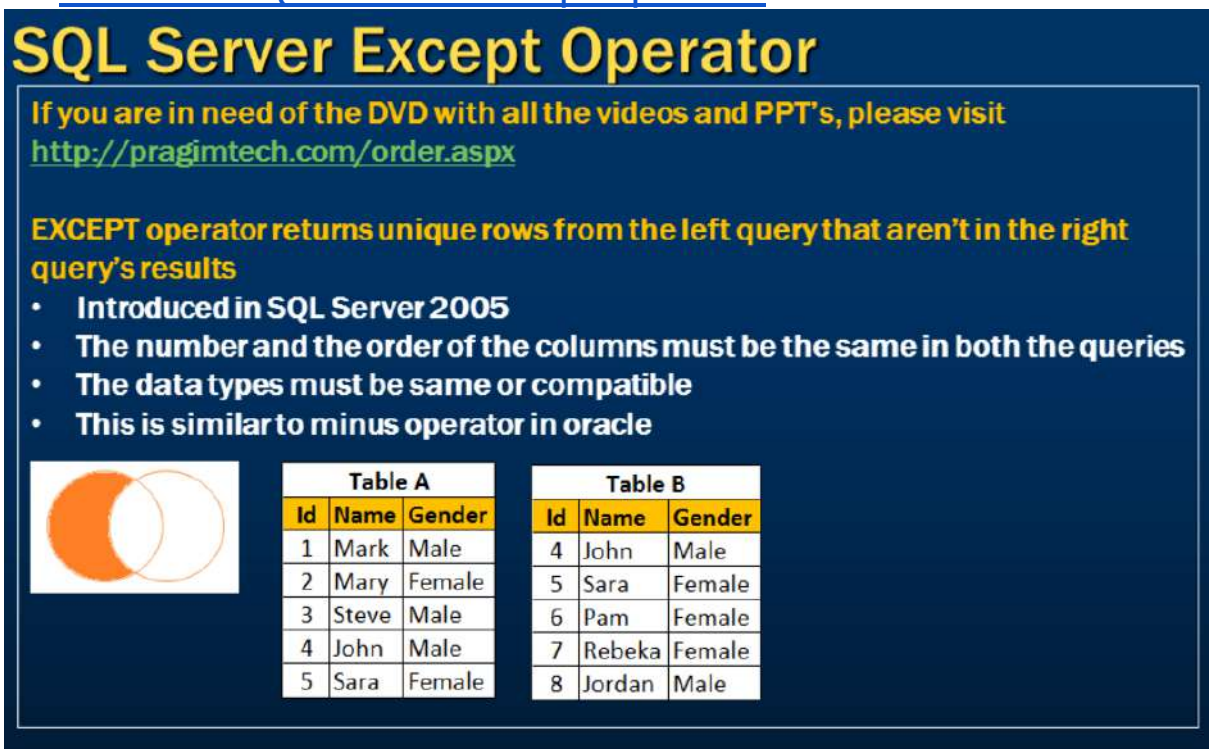
What happens when you kill a session

All the work that the transaction has done will be rolled back. The database must be put back in the state it was in, before the transaction started

```
--script to find all active transaction
SELECT
    [s_tst].[session_id],
    [s_es].[login_name] AS [Login Name],
    DB_NAME (s_tdt.database_id) AS [Database],
    [s_tdt].[database_transaction_begin_time] AS [Begin Time],
    [s_tdt].[database_transaction_log_bytes_used] AS [Log Bytes],
    [s_tdt].[database_transaction_log_bytes_reserved] AS [Log Rsvd],
    [s_est].text AS [Last T-SQL Text],
    [s_eqp].[query_plan] AS [Last Plan]
FROM
    sys.dm_tran_database_transactions [s_tdt]
JOIN
    sys.dm_tran_session_transactions [s_tst]
ON
    [s_tst].[transaction_id] = [s_tdt].[transaction_id]
JOIN
    sys.[dm_exec_sessions] [s_es]
ON
    [s_es].[session_id] = [s_tst].[session_id]
JOIN
    sys.dm_exec_connections [s_ec]
ON
    [s_ec].[session_id] = [s_tst].[session_id]
LEFT OUTER JOIN
```

```
    sys.dm_exec_requests [s_er]
ON
    [s_er].[session_id] = [s_tst].[session_id]
CROSS APPLY
    sys.dm_exec_sql_text ([s_ec].[most_recent_sql_handle]) AS [s_est]
OUTER APPLY
    sys.dm_exec_query_plan ([s_er].[plan_handle]) AS [s_eqp]
ORDER BY
    [Begin Time] ASC;
G
```

Part 87 - SQL Server except operator



SQL Server Except Operator

Table A

Id	Name	Gender
1	Mark	Male
2	Mary	Female
3	Steve	Male
4	John	Male
5	Sara	Female

Table B

Id	Name	Gender
4	John	Male
5	Sara	Female
6	Pam	Female
7	Rebeka	Female
8	Jordan	Male

```
Select Id, Name, Gender
From TableA
Except
Select Id, Name, Gender
From TableB
```

Result

Id	Name	Gender
1	Mark	Male
2	Mary	Female
3	Steve	Male

```
Select Id, Name, Gender
From TableB
Except
Select Id, Name, Gender
From TableA
```

Result

Id	Name	Gender
6	Pam	Female
7	Rebeka	Female
8	Jordan	Male

SQL Server Except Operator

You can also use Except operator on a single table

tblEmployees

Id	Name	Gender	Salary
1	Mark	Male	52000
2	Mary	Female	55000
3	Steve	Male	45000
4	John	Male	40000
5	Sara	Female	48000
6	Pam	Female	60000
7	Tom	Male	58000
8	George	Male	65000
9	Tina	Female	67000
10	Ben	Male	80000

```
Select Id, Name, Gender, Salary
From tblEmployees
Where Salary >= 50000
Except
Select Id, Name, Gender, Salary
From tblEmployees
Where Salary >= 60000
```

Result

Id	Name	Gender	Salary
1	Mark	Male	52000
2	Mary	Female	55000
7	Tom	Male	58000

Order By clause should be used only once after the right query

```
Select Id, Name, Gender, Salary
From tblEmployees
Where Salary >= 50000
Except
Select Id, Name, Gender, Salary
From tblEmployees
Where Salary >= 60000
Order By Id Desc
```

Result

Id	Name	Gender	Salary
7	Tom	Male	58000
2	Mary	Female	55000
1	Mark	Male	52000

- [Part 88 - Difference between except and not in sql server]

Difference between EXCEPT and NOT IN

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

EXCEPT operator returns all the rows from the left query that aren't in the right query's results. NOT IN operator also does the same.

Table A

Id	Name	Gender
1	Mark	Male
2	Mary	Female
3	Steve	Male

Table B

Id	Name	Gender
2	Mary	Female
3	Steve	Male

```
Select Id, Name, Gender From TableA
Except
Select Id, Name, Gender From TableB
```

```
Select Id, Name, Gender From TableA
Where Id NOT IN
(Select Id from TableB)
```

Result

Id	Name	Gender
1	Mark	Male

Difference between EXCEPT and NOT IN

Except filters duplicates and returns only DISTINCT rows from the left query that aren't in the right query's results, where as NOT IN does not filter the duplicates

EXCEPT operator expects the same number of columns in both the queries, where as NOT IN, compares a single column from the outer query with a single column from the sub-query

Part 89 - Intersect operator in sql server

INTERSECT Operator in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Intersect operator retrieves the common records from both the left and the right query of the Intersect operator

- Introduced in SQL Server 2005
- The number and the order of the columns must be same in both the queries
- The data types must be same or at least compatible

Table A

Id	Name	Gender
1	Mark	Male
2	Mary	Female
3	Steve	Male

Table B

Id	Name	Gender
2	Mary	Female
3	Steve	Male

```
Select Id, Name, Gender from TableA
Intersect
Select Id, Name, Gender from TableB
```

```
Select TableA.Id, TableA.Name, TableA.Gender
From TableA Inner Join TableB
On TableA.Id = TableB.Id
```

Result

Id	Name	Gender
2	Mary	Female
3	Steve	Male

INTERSECT v/s INNER JOIN

INTERSECT filters duplicates and returns only DISTINCT rows that are common between the LEFT and Right Query, where as INNER JOIN does not filter the duplicates

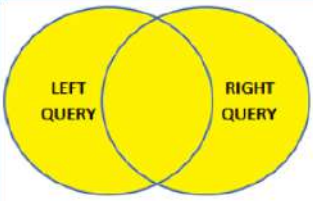
To make INNER JOIN behave like INTERSECT operator use the DISTINCT operator

INNER JOIN treats two NULLS as two different values. So if you are joining two tables based on a nullable column and if both tables have NULLs in that joining column then, INNER JOIN will not include those rows in the result-set, where as INTERSECT treats two NULLs as a same value and it returns all matching rows

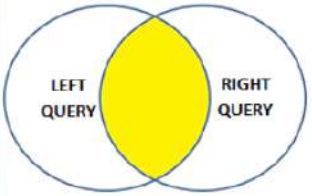
Part 90 - Difference between union intersect and except in sql server

UNION v/s INTERSECT v/s EXCEPT

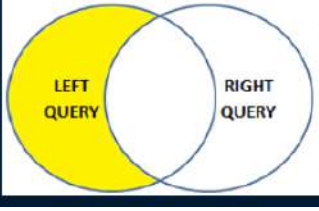
If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx



UNION operator returns all the unique rows from both the left and the right query. UNION ALL includes the duplicates as well



INTERSECT operator retrieves the common unique rows from both the left and the right query



EXCEPT operator returns unique rows from the left query that aren't in the right query's results

Table A

Id	Name	Gender
1	Mark	Male
2	Mary	Female
3	Steve	Male
3	Steve	Male

Table B

Id	Name	Gender
2	Mary	Female
3	Steve	Male
4	John	Male

UNION Result

Id	Name	Gender
1	Mark	Male
2	Mary	Female
3	Steve	Male
4	John	Male

UNION ALL Result

Id	Name	Gender
1	Mark	Male
2	Mary	Female
3	Steve	Male
3	Steve	Male
2	Mary	Female
3	Steve	Male
4	John	Male

INTERSECT Result

Id	Name	Gender
2	Mary	Female
3	Steve	Male

EXCEPT Result

Id	Name	Gender
1	Mark	Male

Part 91 - Cross apply and outer apply in sql server

Cross apply & Outer apply in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Department Table

Id	DepartmentName
1	IT
2	HR
3	Payroll
4	Administration
5	Sales

Employee Table

Id	Name	Gender	Salary	DepartmentId
1	Mark	Male	50000	1
2	Mary	Female	60000	3
3	Steve	Male	45000	2
4	John	Male	56000	1
5	Sara	Female	39000	2

DepartmentName	Name	Gender	Salary
IT	Mark	Male	50000
Payroll	Mary	Female	60000
HR	Steve	Male	45000
IT	John	Male	56000
HR	Sara	Female	39000

DepartmentName	Name	Gender	Salary
IT	Mark	Male	50000
IT	John	Male	56000
HR	Steve	Male	45000
HR	Sara	Female	39000
Payroll	Mary	Female	60000
Administration	NULL	NULL	NULL
Sales	NULL	NULL	NULL

- The APPLY operator introduced in SQL Server 2005, is used to join a table to a table-valued function
- The Table Valued Function on the right hand side of the APPLY operator gets called for each row from the left (also called outer table) table
- Cross Apply returns only matching rows (semantically equivalent to Inner Join)
- Outer Apply returns matching + non-matching rows (semantically equivalent to Left Outer Join). The unmatched columns of the table valued function will be set to NULL

Cross Apply Example

Department Table	
Id	DepartmentName
1	IT
2	HR
3	Payroll
4	Administration
5	Sales

Employee Table				
Id	Name	Gender	Salary	DepartmentId
1	Mark	Male	50000	1
2	Mary	Female	60000	3
3	Steve	Male	45000	2
4	John	Male	56000	1
5	Sara	Female	39000	2

```
Select D.DepartmentName, E.Name, E.Gender, E.Salary
from Department D
Cross Apply fn_GetEmployeesByDepartmentId(D.Id) E
```

DepartmentName	Name	Gender	Salary
IT	Mark	Male	50000
Payroll	Mary	Female	60000
HR	Steve	Male	45000
IT	John	Male	56000
HR	Sara	Female	39000

Outer Apply Example

Department Table	
Id	DepartmentName
1	IT
2	HR
3	Payroll
4	Administration
5	Sales

Employee Table				
Id	Name	Gender	Salary	DepartmentId
1	Mark	Male	50000	1
2	Mary	Female	60000	3
3	Steve	Male	45000	2
4	John	Male	56000	1
5	Sara	Female	39000	2

```
Select D.DepartmentName, E.Name, E.Gender, E.Salary
from Department D
Outer Apply fn_GetEmployeesByDepartmentId(D.Id) E
```

DepartmentName	Name	Gender	Salary
IT	Mark	Male	50000
IT	John	Male	56000
HR	Steve	Male	45000
HR	Sara	Female	39000
Payroll	Mary	Female	60000
Administration	NULL	NULL	NULL
Sales	NULL	NULL	NULL

Part 92 - DDL Triggers in sql server

DDL Triggers in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

In SQL Server there are 4 types of triggers

1. DML Triggers - Data Manipulation Language. Parts 43 to 47 (SQL Server Tutorial)
2. DDL Triggers - Data Definition Language
3. CLR triggers - Common Language Runtime
4. Logon triggers

What are DDL triggers

DDL triggers fire in response to DDL events - CREATE, ALTER, and DROP (Table, Function, Index, Stored Procedure etc...). For the list of all DDL events please visit https://msdn.microsoft.com/en-us/library/bb522542.aspx

Certain system stored procedures that perform DDL-like operations can also fire DDL triggers.

Example - sp_rename system stored procedure

DDL Triggers in SQL Server

What is the use of DDL triggers

- If you want to execute some code in response to a specific DDL event
- To prevent certain changes to your database schema
- Audit the changes that the users are making to the database structure

DDL trigger Syntax

```
CREATE TRIGGER [Trigger_Name]
ON  [Scope (Server|Database)]
FOR [EventType1, EventType2, EventType3, ...],
AS
BEGIN
   -- Trigger Body
END
```

DDL triggers scope : DDL triggers can be created in a specific database or at the server level

DDL Triggers Example

This trigger fires whenever a table is created

```
-- Trigger that fires in response to a single DDL event
CREATE TRIGGER trMyFirstTrigger
ON Database
FOR CREATE_TABLE
AS
BEGIN
   Print 'New table created'
END
```

This trigger fires whenever a table is created, altered or dropped

```
-- Trigger that fires in response to a multiple DDL
-- events
ALTER TRIGGER trMyFirstTrigger
ON Database
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
   Print 'You just created, modified or deleted a table'
END
```

DDL Triggers in SQL Server

```
-- DDL trigger to prevent users from creating,
-- altering or dropping tables
ALTER TRIGGER trMyFirstTrigger
ON Database
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
   Rollback
   Print 'You cannot create, alter or drop a table'
END
```

DDL Triggers in SQL Server

Certain system stored procedures that perform DDL-like operations can also fire DDL triggers

```
CREATE TRIGGER trRenameTable
ON DATABASE
FOR RENAME
AS
BEGIN
    Print 'You just renamed something'
END
```

The trigger will be fired when ever you rename a database object using sp_rename system stored procedure

```
-- Renaming table - Fires the trigger
sp_rename 'TestTable', 'NewTestTable'

-- Renaming table column - Fires the trigger
sp_rename 'NewTestTable.Id' , 'NewId', 'column'
```

Part 93 - Server-scoped DDL triggers

Server-Scoped DDL Triggers

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

```
-- Database scoped trigger
CREATE TRIGGER tr_DatabaseScopeTrigger
ON DATABASE
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
    ROLLBACK
    Print 'You cannot create, alter or drop a
           table in the current database'
END
```

```
-- Server scoped trigger
CREATE TRIGGER tr_ServerScopeTrigger
ON ALL SERVER
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
    ROLLBACK
    Print 'You cannot create, alter or drop a table
           in any database on the server'
END
```

- [Part 94 - SQL Server trigger execution order]

SQL Server Trigger Execution Order

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Server scoped triggers will always fire before any of the database scoped triggers

Using the sp_settriggerorder stored procedure, you can set the execution order of server-scoped or database-scoped triggers

Parameter	Description
@triggername	Name of the trigger
@order	Value can be First, Last or None. When set to None, trigger is fired in random order
@stmttype	SQL statement that fires the trigger. Can be INSERT, UPDATE, DELETE or any DDL event
@namespace	Scope of the trigger. Value can be DATABASE, SERVER, or NULL

```
EXEC sp_settriggerorder
@triggername = 'tr_DatabaseScopeTrigger1',
@order = 'none',
@stmttype = 'CREATE_TABLE',
@namespace = 'DATABASE'
GO
```

SQL Server Trigger Execution Order

If you have a database-scoped and a server-scoped trigger handling the same event, and if you have set the execution order at both the levels. Here is the execution order of the triggers

1. The server-scope trigger marked First
2. Other server-scope triggers
3. The server-scope trigger marked Last
4. The database-scope trigger marked First
5. Other database-scope triggers
6. The database-scope trigger marked Last

Part 95 - Audit table changes in sql server

Audit Table Changes in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

DatabaseName	TableName	EventType	LoginName	SQLCommand	AuditDateTime
SampleDB	MyTable	CREATE_TABLE	VENKAT-PC\Tan	Create Table MyTa	2015-09-11 16:02:24.240
SampleDB	MyTable	DROP_TABLE	VENKAT-PC\Tan	Drop Table MyTabl	2015-09-11 16:02:57.430
SampleDB	MyTable	CREATE_TABLE	VENKAT-PC\Tan	Create Table MyTa	2015-09-11 16:06:07.037
SampleDB	MyTable	ALTER_TABLE	VENKAT-PC\Tan	Alter Table MyTabl	2015-09-11 16:06:11.610

EventData() function returns event data in XML format

```
<EVENT_INSTANCE>
  <EventType>CREATE_TABLE</EventType>
  <PostTime>2015-09-11T16:12:49.417</PostTime>
  <ServerName>VENKAT-PC</ServerName>
  <LoginName>VENKAT-PC\Tan</LoginName>
  <DatabaseName>SampleDB</DatabaseName>
  <ObjectName>MyTable</ObjectName>
  <TSQLCommand>
    <CommandText>
      Create Table MyTable(Id int,Name nvarchar(50),Gender nvarchar(50))
    </CommandText>
  </TSQLCommand>
</EVENT_INSTANCE>
```

Audit Table Changes in SQL Server

```
CREATE TRIGGER tr_AuditTableChanges
ON ALL SERVER
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
    DECLARE @EventData XML
    SELECT @EventData = EVENTDATA()

    INSERT INTO SampleDB.dbo.TableChanges
    (DatabaseName, TableName, EventType, LoginName, SQLCommand, AuditDateTime)
    VALUES
    (
        @EventData.value('(/EVENT_INSTANCE/DatabaseName)[1]', 'varchar(250)'),
        @EventData.value('(/EVENT_INSTANCE/ObjectName)[1]', 'varchar(250)'),
        @EventData.value('(/EVENT_INSTANCE/EventType)[1]', 'nvarchar(250)'),
        @EventData.value('(/EVENT_INSTANCE/LoginName)[1]', 'varchar(250)'),
        @EventData.value('(/EVENT_INSTANCE/TSQLCommand)[1]', 'nvarchar(2500)'),
        GetDate()
    )
END
```

Part 96 - Logon triggers in sql server

Logon Triggers in SQL Server

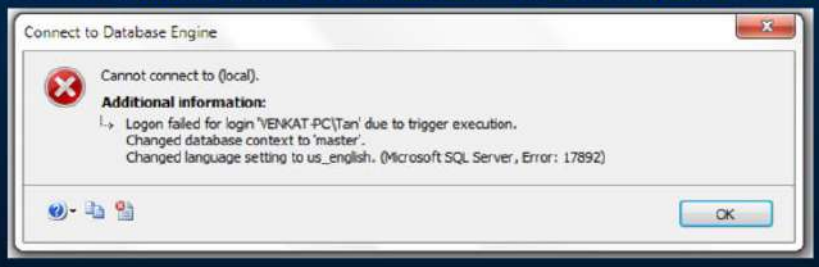
If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

As the name implies Logon triggers fire in response to a LOGON event. Logon triggers fire after the authentication phase of logging in finishes, but before the user session is actually established

Logon triggers can be used for

1. Tracking login activity
2. Restricting logins to SQL Server
3. Limiting the number of sessions for a specific login

Write a trigger to limit the maximum number of open connections for a user to 3



Logon Triggers in SQL Server

```
CREATE TRIGGER tr_LogonAuditTrigger
ON ALL SERVER
FOR LOGON
AS
BEGIN
    DECLARE @LoginName NVARCHAR(100)

    Set @LoginName = ORIGINAL_LOGIN()

    IF (SELECT COUNT(*) FROM sys.dm_exec_sessions
        WHERE is_user_process = 1
        AND original_login_name = @LoginName) > 3
    BEGIN
        Print 'Fourth connection of ' + @LoginName + ' blocked'
        ROLLBACK
    END
END
```

The trigger error message will be written to the error log

`Execute sp_readerrorlog` to read the error log

LogData	ProcessInfo	Text
13/09/2015	spid54	Fourth connection of VENKAT-PC\Tan blocked
13/09/2015	spid54	Error: 3609, Severity: 16, State: 2.
13/09/2015	spid54	The transaction ended in the trigger. The batc

Part 97 - Select into in sql server

SELECT INTO in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Departments Table

DepartmentId	DepartmentName
1	IT
2	HR
3	Payroll

Employees Table

Id	Name	Gender	Salary	DeptId
1	Mark	Male	50000	1
2	Sara	Female	65000	2
3	Mike	Male	48000	3
4	Pam	Female	70000	1
5	John	Male	55000	2

The SELECT INTO statement in SQL Server, selects data from one table and inserts it into a new table

SELECT INTO statement in SQL Server can do the following

1. Copy all rows and columns from an existing table into a new table. This is extremely useful when you want to make a backup copy of the existing table

```
SELECT * INTO EmployeesBackup FROM Employees
```

SELECT INTO in SQL Server

2. Copy all rows and columns from an existing table into a new table in an external database

```
SELECT * INTO HRDB.dbo.EmployeesBackup FROM Employees
```

3. Copy only selected columns into a new table

```
SELECT Id, Name, Gender INTO EmployeesBackup FROM Employees
```

4. Copy only selected rows into a new table

```
SELECT * INTO EmployeesBackup FROM Employees WHERE DeptId = 1
```

5. Copy columns from 2 or more table into a new table

```
SELECT * INTO EmployeesBackup FROM Employees
INNER JOIN Departments
ON Employees.DeptId = Departments.DepartmentId
```

6. Create a new table whose columns and data types match with an existing table

```
SELECT * INTO EmployeesBackup FROM Employees WHERE 1 <> 1
```

SELECT INTO in SQL Server

7. Copy all rows and columns from an existing table into a new table on a different SQL Server instance. For this, create a linked server and use the 4 part naming convention

```
SELECT * INTO TargetTable
FROM [SourceServer].[SourceDB].[dbo].[SourceTable]
```

Please note : You cannot use SELECT INTO statement to select data into an existing table. For this you will have to use INSERT INTO statement.

```
INSERT INTO ExistingTable (ColumnList)
SELECT ColumnList FROM SourceTable
```

Part 98 - Difference between where and having in sql server

Difference between WHERE and HAVING

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Sales	
Product	**SaleAmount**
iPhone	500
Laptop	800
iPhone	1000
Speakers	400
Laptop	600

Calculate total sales by product

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY Product
```

Product	TotalSales
iPhone	1500
Laptop	1400
Speakers	400

Difference between WHERE and HAVING

Find only those products where the total sales amount is greater than $1000

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY Product
HAVING SUM(SaleAmount) > 1000
```

Product	TotalSales
iPhone	1500
Laptop	1400

If we use WHERE clause instead of HAVING clause, we will get a syntax error. This is because the WHERE clause doesn't work with aggregate functions like sum, min, max, avg, etc.

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY Product
WHERE SUM(SaleAmount) > 1000
```

```
Msg 156, Level 15, State 1, Line 4
Incorrect syntax near the keyword 'WHERE'.
```

So in short, the difference is WHERE clause cannot be used with aggregates where as HAVING can

Difference between WHERE and HAVING

WHERE clause filters rows before aggregate calculations are performed where as HAVING clause filters rows after aggregate calculations are performed

Total sales of iPhone and Speakers can be calculated by using either WHERE or HAVING clause

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
WHERE Product in ('iPhone', 'Speakers')
GROUP BY Product
```

Product	TotalSales
iPhone	1500
Speakers	400

```
SELECT Product, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY Product
HAVING Product in ('iPhone', 'Speakers')
```

WHERE clause filters rows before aggregate calculations are performed where as HAVING clause filters rows after aggregate calculations are performed. So from a performance standpoint, HAVING is slower than WHERE and should be avoided when possible.

Difference between WHERE and HAVING

WHERE clause cannot be used with aggregates where as HAVING can. This means WHERE clause is used for filtering individual rows where as HAVING clause is used to filter groups.

WHERE comes before GROUP BY. This means WHERE clause filters rows before aggregate calculations are performed. HAVING comes after GROUP BY. This means HAVING clause filters rows after aggregate calculations are performed. So from a performance standpoint, HAVING is slower than WHERE and should be avoided when possible.

WHERE and HAVING can be used together in a SELECT query. In this case WHERE clause is applied first to filter individual rows. The rows are then grouped and aggregate calculations are performed, and then the HAVING clause filter the groups.

- Part 99 - Table valued parameters in SQL Server

Table Valued Parameters in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Table Valued Parameter is a new feature introduced in SQL SERVER 2008. Table Valued Parameter allows a table (i.e multiple rows of data) to be passed as a parameter to a stored procedure from T-SQL code or from an application. Prior to SQL SERVER 2008, it is not possible to pass a table variable to a stored procedure.

Please note : Table valued parameters must be passed as read-only to stored procedures, functions etc. This means you cannot perform DML operations like INSERT, UPDATE or DELETE on a table-valued parameter in the body of a function, stored procedure etc.

3 steps to pass multiple rows to a stored procedure using Table Valued Parameter
Step 1 : Create User-defined Table Type
Step 2 : Use the User-defined Table Type as a parameter in the stored procedure.
Step 3 : Declare a table variable, insert the data rows and then pass the table variable as a parameter to the stored procedure

Table Valued Parameters in SQL Server

Step 1 : Create User-defined Table Type

```
Create Type EmpTableType as Table
(
    Id int primary key,
    Name nvarchar(50),
    Gender nvarchar(10)
)
```

Step 2 : Use the User-defined Table Type as a parameter in the stored procedure

```
CREATE PROCEDURE spInsertEmployees
@EmpTableType EmpTableType READONLY
AS
BEGIN
    INSERT INTO Employees
    SELECT * FROM @EmpTableType
END
```

Table Valued Parameters in SQL Server

Step 3 : Declare a table variable, insert the data and then pass the table variable as a parameter to the stored procedure

```
-- Declare the Table Variable
DECLARE @EmployeeTableType EmpTableType

-- Insert rows into the Table variable that you want to send to the SP
INSERT INTO @EmployeeTableType VALUES (1, 'Mark', 'Male')
INSERT INTO @EmployeeTableType VALUES (2, 'Mary', 'Female')
INSERT INTO @EmployeeTableType VALUES (3, 'John', 'Male')
INSERT INTO @EmployeeTableType VALUES (4, 'Sara', 'Female')
INSERT INTO @EmployeeTableType VALUES (5, 'Rob', 'Male')

-- Pass the table variable as a parameter to the stored procedure
EXECUTE spInsertEmployees @EmployeeTableType
```

Next Video : How to pass table as a parameter to the stored procedure from an ADO.NET application

- [Part 100 - Send datatable as parameter to stored procedure]
- [Part 101 - Grouping Sets in SQL Server]

Grouping Sets in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Grouping sets is a new feature introduced in SQL Server 2008

Employees Table

Id	Name	Gender	Salary	Country
1	Mark	Male	5000	USA
2	John	Male	4500	India
3	Pam	Female	5500	USA
4	Sara	Female	4000	India
5	Todd	Male	3500	India
6	Mary	Female	5000	UK
7	Ben	Male	6500	UK
8	Elizabeth	Female	7000	USA
9	Tom	Male	5500	UK
10	Ron	Male	5000	USA

Sum of Salary by Country and Gender

Country	Gender	TotalSalary
India	Female	4000
UK	Female	5000
USA	Female	12500
India	Male	8000
UK	Male	12000
USA	Male	10000

Group By query to calculate Sum of Salary by Country and Gender

```
Select Country, Gender, Sum(Salary) as TotalSalary
From Employees
Group By Country, Gender
```

Grouping Sets in SQL Server

Along with Sum of Salary by Country and Gender, we also want Sum of Salary just by Country

Employees Table

Id	Name	Gender	Salary	Country
1	Mark	Male	5000	USA
2	John	Male	4500	India
3	Pam	Female	5500	USA
4	Sara	Female	4000	India
5	Todd	Male	3500	India
6	Mary	Female	5000	UK
7	Ben	Male	6500	UK
8	Elizabeth	Female	7000	USA
9	Tom	Male	5500	UK
10	Ron	Male	5000	USA

Country	Gender	TotalSalary
India	Female	4000
UK	Female	5000
USA	Female	12500
India	Male	8000
UK	Male	12000
USA	Male	10000
India	NULL	12000
UK	NULL	17000
USA	NULL	22500

Sum of Salary by Country

```
Select Country, Gender, Sum(Salary) as TotalSalary
From Employees
Group By Country, Gender

UNION ALL

Select Country, NULL, Sum(Salary) as TotalSalary
From Employees
Group By Country
```

Grouping Sets in SQL Server

Along with Sum of Salary by Country and Gender, we also want Sum of Salary just by Country, and also Sum of Salary just by Gender

Country	Gender	TotalSalary
India	Female	4000
UK	Female	5000
USA	Female	12500
India	Male	8000
UK	Male	12000
USA	Male	10000
India	NULL	12000
UK	NULL	17000
USA	NULL	22500
NULL	Female	21500
NULL	Male	30000

Sum of Salary by Country

Sum of Salary by Gender

Grouping Sets in SQL Server

Query to compute

Sum of Salary by Country and Gender,
Sum of Salary just by Country, and
Sum of Salary just by Gender

```
Select Country, Gender, Sum(Salary) as TotalSalary
From Employees
Group By Country, Gender

UNION ALL

Select Country, NULL, Sum(Salary) as TotalSalary
From Employees
Group By Country

UNION ALL

Select NULL, Gender, Sum(Salary) as TotalSalary
From Employees
Group By Gender
```

Grouping Sets in SQL Server

Along with Sum of Salary by Country and Gender, we also want Sum of Salary just by Country, and also Sum of Salary just by Gender and finally grand total

Country	Gender	TotalSalary
India	Female	4000
UK	Female	5000
USA	Female	12500
India	Male	8000
UK	Male	12000
USA	Male	10000
India	NULL	12000
UK	NULL	17000
USA	NULL	22500
NULL	Female	21500
NULL	Male	30000
NULL	NULL	51500

Sum of Salary by Country

Sum of Salary by Gender

Grand Total

Grouping Sets in SQL Server

Query to compute

Sum of Salary by Country and Gender,
Sum of Salary just by Country,
Sum of Salary just by Gender,
Grand Total

```
Select Country, Gender, Sum(Salary) as TotalSalary
From Employees
Group By Country, Gender

UNION ALL

Select Country, NULL, Sum(Salary) as TotalSalary
From Employees
Group By Country

UNION ALL

Select NULL, Gender, Sum(Salary) as TotalSalary
From Employees
Group By Gender

UNION ALL

Select NULL, NULL, Sum(Salary) as TotalSalary
From Employees
```

Grouping Sets in SQL Server

With Grouping Sets feature introduced in SQL Server 2008, the amount of T-SQL code that you have to write will be greatly reduced

```
Select Country, Gender, Sum(Salary) TotalSalary
From Employees
Group BY
 GROUPING SETS
 (
   (Country, Gender), -- Sum of Salary by Country & Gender
   (Country),         -- Sum of Salary by Country
   (Gender) ,         -- Sum of Salary by Gender
   ()                 -- Grand Total
 )
```

Grouping Sets in SQL Server

The order of the rows in the result set is not the same as in the case of UNION ALL query. To control the order use ORDER BY

```
Select Country, Gender, Sum(Salary) TotalSalary
From Employees
Group BY
 GROUPING SETS
 (
   (Country, Gender), -- Sum of Salary by Country & Gender
   (Country),         -- Sum of Salary by Country
   (Gender) ,         -- Sum of Salary by Gender
   ()                 -- Grand Total
 )
Order By Grouping(Country), Grouping(Gender), Gender
```

Part 102 - Rollup in SQL Server

Rollup in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

ROLLUP is used to do aggregate operation on multiple levels in a hierarchy

Employees Table

Id	Name	Gender	Salary	Country
1	Mark	Male	5000	USA
2	John	Male	4500	India
3	Pam	Female	5500	USA
4	Sara	Female	4000	India
5	Todd	Male	3500	India
6	Mary	Female	5000	UK
7	Ben	Male	6500	UK
8	Elizabeth	Female	7000	USA
9	Tom	Male	5500	UK
10	Ron	Male	5000	USA

Retrieve Salary by country along with grand total

Country	TotalSalary
India	12000
UK	17000
USA	22500
NULL	51500

Retrieve Salary by country along with grand total

```
SELECT Country, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY ROLLUP(Country)
```

Rollup in SQL Server

Other ways to retrieve Salary by Country along with Grand Total

```
-- Using ROLLUP clause with GROUP BY
SELECT Country, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country WITH ROLLUP
```

```
-- Using UNION ALL operator along with GROUP BY
SELECT Country, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country

UNION ALL

SELECT NULL, SUM(Salary) AS TotalSalary
FROM Employees
```

```
--Using GROUPING SETS
SELECT Country, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY GROUPING SETS
(
    (Country),
    ()
)
```

Rollup in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Grouping sets is a new feature introduced in SQL Server 2008

Employees Table

Id	Name	Gender	Salary	Country
1	Mark	Male	5000	USA
2	John	Male	4500	India
3	Pam	Female	5500	USA
4	Sara	Female	4000	India
5	Todd	Male	3500	India
6	Mary	Female	5000	UK
7	Ben	Male	6500	UK
8	Elizabeth	Female	7000	USA
9	Tom	Male	5500	UK
10	Ron	Male	5000	USA

Group Salary by Country and Gender. Also compute the Subtotal at Country level and Grand Total.

Country	Gender	TotalSalary
India	Female	4000
India	Male	8000
India	NULL	12000
UK	Female	5000
UK	Male	12000
UK	NULL	17000
USA	Female	12500
USA	Male	10000
USA	NULL	22500
NULL	NULL	51500

```
SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY ROLLUP(Country, Gender)
```

Rollup in SQL Server

Other ways to group Salary by Country and Gender and also to compute the Subtotal at Country level and Grand Total

```
-- Using ROLLUP clause with GROUP BY
SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country, Gender WITH ROLLUP
```

```
--Using UNION ALL with GROUP BY
SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country, Gender

UNION ALL

SELECT Country, NULL, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country

UNION ALL

SELECT NULL, NULL, SUM(Salary) AS TotalSalary
FROM Employees
```

Rollup in SQL Server

Other ways to group Salary by Country and Gender and also to compute the Subtotal at Country level and Grand Total

```
--Using GROUPING SETS
SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY GROUPING SETS
(
    (Country, Gender),
    (Country),
    ()
)
```

- Part 103 - Cube in SQL Server

Cube in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Cube() in SQL Server produces the result set by generating all combinations of columns specified in GROUP BY CUBE()

Write a query to retrieve Sum of Salary grouped by all combinations of the 2 columns (Country & Gender) as well as Grand Total.

Employees Table

Id	Name	Gender	Salary	Country
1	Mark	Male	5000	USA
2	John	Male	4500	India
3	Pam	Female	5500	USA
4	Sara	Female	4000	India
5	Todd	Male	3500	India
6	Mary	Female	5000	UK
7	Ben	Male	6500	UK
8	Elizabeth	Female	7000	USA
9	Tom	Male	5500	UK
10	Ron	Male	5000	USA

Country	Gender	TotalSalary
India	Female	4000
UK	Female	5000
USA	Female	12500
NULL	Female	21500
India	Male	8000
UK	Male	12000
USA	Male	10000
NULL	Male	30000
NULL	NULL	51500
India	NULL	12000
UK	NULL	17000
USA	NULL	22500

Rollup in SQL Server

```sql
-- Using Cube with Group By
SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Cube(Country, Gender)

-- OR

SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country, Gender with Cube
```

```sql
-- The above query is equivalent to
-- the following Grouping Sets query

SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY GROUPING SETS
    (
        (Country, Gender),
        (Country),
        (Gender),
        ()
    )
```

Rollup in SQL Server

The queries on the previous are equivalent to the following UNION ALL query. While the data in the result set is the same, the ordering is not. Use ORDER BY to control the ordering of rows in the result set

```sql
SELECT Country, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country, Gender

UNION ALL

SELECT Country, NULL, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Country

UNION ALL

SELECT NULL, Gender, SUM(Salary) AS TotalSalary
FROM Employees
GROUP BY Gender

UNION ALL

SELECT NULL, NULL, SUM(Salary) AS TotalSalary
FROM Employees
```

Part 104 - Difference between cube and rollup in SQL Server

Difference between CUBE and ROLLUP

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

CUBE generates a result set that shows aggregates for all combinations of values in the selected columns, where as ROLLUP generates a result set that shows aggregates for a hierarchy of values in the selected columns

Continent	Country	City	SaleAmount
Asia	India	Bangalore	1000
Asia	India	Chennai	2000
Asia	Japan	Tokyo	4000
Asia	Japan	Hiroshima	5000
Europe	United Kingdom	London	1000
Europe	United Kingdom	Manchester	2000
Europe	France	Paris	4000
Europe	France	Cannes	5000

ROLLUP(Continent, Country, City)

Continent, Country, City
Continent, Country,
Continent
()

CUBE(Continent, Country, City)

Continent, Country, City
Continent, Country,
Continent, City
Continent
Country, City
Country,
City
()

Difference between CUBE and ROLLUP

You won't see any difference when you use ROLLUP and CUBE on a single column

```
SELECT Continent, Sum(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ROLLUP(Continent)

-- OR

SELECT Continent, SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY CUBE(Continent)
```

Continent	TotalSales
Asia	12000
Europe	12000
NULL	24000

Part 105 - Grouping function in SQL Server

Grouping function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

What is Grouping function

Grouping(Column) indicates whether the column in a GROUP BY list is aggregated or not. Grouping returns 1 for aggregated or 0 for not aggregated in the result set.

Continent	Country	City	TotalSales	GP_Continent	GP_Country	GP_City
Asia	India	Bangalore	1000	0	0	0
Asia	India	Chennai	2000	0	0	0
Asia	India	NULL	3000	0	0	1
Asia	Japan	Hiroshima	5000	0	0	0
Asia	Japan	Tokyo	4000	0	0	0
Asia	Japan	NULL	9000	0	0	1
Asia	NULL	NULL	12000	0	1	1
Europe	France	Cannes	5000	0	0	0
Europe	France	Paris	4000	0	0	0
Europe	France	NULL	9000	0	0	1
Europe	United Kingdom	London	1000	0	0	0
Europe	United Kingdom	Manchester	2000	0	0	0
Europe	United Kingdom	NULL	3000	0	0	1
Europe	NULL	NULL	12000	0	1	1
NULL	NULL	NULL	24000	1	1	1

Grouping function in SQL Server

The following query returns 1 for aggregated or 0 for not aggregated in the result set

```
SELECT   Continent, Country, City, SUM(SaleAmount) AS TotalSales,
         GROUPING(Continent) AS GP_Continent,
         GROUPING(Country) AS GP_Country,
         GROUPING(City) AS GP_City
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
```

Continent	Country	City	TotalSales	GP_Continent	GP_Country	GP_City
Asia	India	Bangalore	1000	0	0	0
Asia	India	Chennai	2000	0	0	0
Asia	India	NULL	3000	0	0	1
Asia	Japan	Hiroshima	5000	0	0	0
Asia	Japan	Tokyo	4000	0	0	0
Asia	Japan	NULL	9000	0	0	1
Asia	NULL	NULL	12000	0	1	1
Europe	France	Cannes	5000	0	0	0
Europe	France	Paris	4000	0	0	0
Europe	France	NULL	9000	0	0	1
Europe	United Kingdom	London	1000	0	0	0
Europe	United Kingdom	Manchester	2000	0	0	0
Europe	United Kingdom	NULL	3000	0	0	1
Europe	NULL	NULL	12000	0	1	1
NULL	NULL	NULL	24000	1	1	1

Grouping function in SQL Server

What is the use of Grouping function in real world

When a column is aggregated in the resultset, the column will have a NULL value. If you want to replace NULL with All then this GROUPING function is very handy

Continent	Country	City	TotalSales
Asia	India	Bangalore	1000
Asia	India	Chennai	2000
Asia	India	All	3000
Asia	Japan	Hiroshima	5000
Asia	Japan	Tokyo	4000
Asia	Japan	All	9000
Asia	All	All	12000
Europe	France	Cannes	5000
Europe	France	Paris	4000
Europe	France	All	9000
Europe	United Kingdom	London	1000
Europe	United Kingdom	Manchester	2000
Europe	United Kingdom	All	3000
Europe	All	All	12000
All	All	All	24000

Grouping function in SQL Server

```
SELECT   CASE WHEN
             GROUPING(Continent) = 1 THEN 'All' ELSE ISNULL(Continent, 'Unknown')
         END AS Continent,
         CASE WHEN
             GROUPING(Country) = 1 THEN 'All' ELSE ISNULL(Country, 'Unknown')
         END AS Country,
         CASE
             WHEN GROUPING(City) = 1 THEN 'All' ELSE ISNULL(City, 'Unknown')
         END AS City,
         SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
```

Continent	Country	City	TotalSales
Asia	India	Bangalore	1000
Asia	India	Chennai	2000
Asia	India	All	3000
Asia	Japan	Hiroshima	5000
Asia	Japan	Tokyo	4000
Asia	Japan	All	9000
Asia	All	All	12000
Europe	France	Cannes	5000
Europe	France	Paris	4000
Europe	France	All	9000
Europe	United Kingdom	London	1000
Europe	United Kingdom	Manchester	2000
Europe	United Kingdom	All	3000
Europe	All	All	12000
All	All	All	24000

Grouping function in SQL Server

Can't I use ISNULL function instead of GROUPING function to achieve the same

You can, but only if your data does not contain NULL values

```
SELECT   ISNULL(Continent, 'All') AS Continent,
         ISNULL(Country, 'All') AS Country,
         ISNULL(City, 'All') AS City,
         SUM(SaleAmount) AS TotalSales
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
```

Please note : Grouping function can be used with Rollup, Cube and Grouping Sets

Part 106 - GROUPING_ID function in SQL Server

GROUPING_ID function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

GROUPING_ID function computes the level of grouping

Difference between GROUPING and GROUPING_ID

Syntax : GROUPING function is used on single column, where as the column list for GROUPING_ID function must match with GROUP BY column list.

```
GROUPING(Col1)

GROUPING_ID(Col1, Col2, Col3,...)
```

GROUPING indicates whether the column in a GROUP BY list is aggregated or not. Grouping returns 1 for aggregated or 0 for not aggregated in the result set.

GROUPING_ID() concatenates all the GOUPING() functions, perform the binary to decimal conversion, and returns the equivalent integer.

In short

```
GROUPING_ID(A, B, C) = GROUPING(A) + GROUPING(B) + GROUPING(C)
```

GROUPING_ID function in SQL Server

```
SELECT   Continent, Country, City, SUM(SaleAmount) AS TotalSales,
         CAST(GROUPING(Continent) AS NVARCHAR(1)) +
         CAST(GROUPING(Country) AS NVARCHAR(1)) +
         CAST(GROUPING(City) AS NVARCHAR(1)) AS Groupings,
         GROUPING_ID(Continent, Country, City) AS GPID
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
```

#	Continent	Country	City	TotalSales	Groupings	GPID
1	Asia	India	Bangalore	1000	000	0
2	Asia	India	Chennai	2000	000	0
3	Asia	India	NULL	3000	001	1
4	Asia	Japan	Hiroshima	5000	000	0
5	Asia	Japan	Tokyo	4000	000	0
6	Asia	Japan	NULL	9000	001	1
7	Asia	NULL	NULL	12000	011	3
8	Europe	France	Cannes	5000	000	0
9	Europe	France	Paris	4000	000	0
10	Europe	France	NULL	9000	001	1
11	Europe	United Kingdom	London	1000	000	0
12	Europe	United Kingdom	Manchester	2000	000	0
13	Europe	United Kingdom	NULL	3000	001	1
14	Europe	NULL	NULL	12000	011	3
15	NULL	NULL	NULL	24000	111	7

Sorting by Level of Grouping

```
SELECT   Continent, Country, City, SUM(SaleAmount) AS TotalSales,
         GROUPING_ID(Continent, Country, City) AS GPID
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
ORDER BY GPID
```

Continent	Country	City	TotalSales	GPID
Asia	Japan	Hiroshima	5000	0
Asia	Japan	Tokyo	4000	0
Asia	India	Bangalore	1000	0
Asia	India	Chennai	2000	0
Europe	France	Cannes	5000	0
Europe	France	Paris	4000	0
Europe	United Kingdom	London	1000	0
Europe	United Kingdom	Manchester	2000	0
Europe	United Kingdom	NULL	3000	1
Europe	France	NULL	9000	1
Asia	India	NULL	3000	1
Asia	Japan	NULL	9000	1
Asia	NULL	NULL	12000	3
Europe	NULL	NULL	12000	3
NULL	NULL	NULL	24000	7

Filter by Level of Grouping

```
SELECT   Continent, Country, City, SUM(SaleAmount) AS TotalSales,
         GROUPING_ID(Continent, Country, City) AS GPID
FROM Sales
GROUP BY ROLLUP(Continent, Country, City)
HAVING GROUPING_ID(Continent, Country, City) = 3
```

Retrieves only CONTINENT level aggregated data

Continent	Country	City	TotalSales	GPID
Asia	NULL	NULL	12000	3
Europe	NULL	NULL	12000	3

Part 107 - Debugging sql server stored procedures

Debugging SQL Server stored procedures

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Setting up the Debugger in SSMS : For the debugger to work use your computer name to connect to SQL Server

Debugging SQL Server stored procedures

Start Debugging	ALT + F5
Stepping thru the code	
Step Over	F10
Step Into	F11
Step Out	SHIFT + F11
Stop Debugging	SHIFT + F5
Show Next Statement	Shows the next statement that the debugger is about to execute
Run to Cursor	Executes all the statements in a batch up to the current cursor position
Locals Window	Displays the current values of variables and parameters
Watch Window	Watch the values of variables
Call Stack	Allows you to navigate up and down the call stack to see what values your application is storing at different levels. It's an invaluable tool for determining why your code is doing what it's doing.
Immediate Window	Very helpful during debugging to evaluate expressions, and print variable values

Debugging SQL Server stored procedures

There are 2 ways to set a breakpoint in SSMS

1. **By clicking on the grey margin on the left hand side in SSMS (to remove click again)**
2. **By pressing F9 (to remove press F9 again)**

From the debug menu you can enable, disable or delete all breakpoints

Conditional Breakpoint: Conditional Breakpoints are hit only when the specified condition is met. These are extremely useful when you have some kind of a loop and you want to break, only when the loop variable has a specific value (For example loop variable = 100)

Part 108 - Over clause in SQL Server

Over clause in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit
http://pragimtech.com/order.aspx

The OVER clause combined with PARTITION BY is used to break up data into partitions. The specified function operates for each partition

Syntax: `function (...) OVER (PARTITION BY col1, Col2, ...)`

For example: `COUNT(Gender) OVER (PARTITION BY Gender)` will partition the data by GENDER i.e there will 2 partitions (Male and Female) and then the COUNT() function is applied over each partition

Any of the following functions can be used
COUNT(), AVG(), SUM(), MIN(), MAX(), ROW_NUMBER(), RANK(), DENSE_RANK() etc

Over clause in SQL Server

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

```
SELECT Gender, COUNT(*) AS GenderTotal, AVG(Salary) AS AvgSal,
       MIN(Salary) AS MinSal, MAX(Salary) AS MaxSal
FROM Employees
GROUP BY Gender
```

Gender	GenderTotal	AvgSal	MinSal	MaxSal
Female	4	5375	4000	7000
Male	6	5000	3500	6500

Over clause in SQL Server

What if we want non-aggregated values (like employee Name and Salary) in result set along with aggregated values

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Salary	Gender	GenderTotals	AvgSal	MinSal	MaxSal
Pam	5500	Female	4	5375	4000	7000
Sara	4000	Female	4	5375	4000	7000
Mary	5000	Female	4	5375	4000	7000
Jodi	7000	Female	4	5375	4000	7000
Tom	5500	Male	6	5000	3500	6500
Ron	5000	Male	6	5000	3500	6500
Ben	6500	Male	6	5000	3500	6500
Todd	3500	Male	6	5000	3500	6500
Mark	5000	Male	6	5000	3500	6500
John	4500	Male	6	5000	3500	6500

```
SELECT Name, Salary, Gender, COUNT(*) AS GenderTotal, AVG(Salary) AS AvgSal,
       MIN(Salary) AS MinSal, MAX(Salary) AS MaxSal
FROM Employees
GROUP BY Gender
```

```
Column 'Employees.Name' is invalid in the select list because it is
not contained in either an aggregate function or the GROUP BY clause
```

Over clause in SQL Server

One way to achieve this is by including the aggregations in a sub query and then JOINING it with the main query

```
SELECT Name, Salary, Employees.Gender, Genders.GenderTotals,
       Genders.AvgSal, Genders.MinSal, Genders.MaxSal
FROM Employees
INNER JOIN
(SELECT Gender, COUNT(*) AS GenderTotals,
        AVG(Salary) AS AvgSal,
        MIN(Salary) AS MinSal, MAX(Salary) AS MaxSal
FROM Employees
GROUP BY Gender) AS Genders
ON Genders.Gender = Employees.Gender
```

Better way of doing this is by using the OVER clause combined with PARTITION BY

```
SELECT Name, Salary, Gender,
       COUNT(Gender) OVER(PARTITION BY Gender) AS GenderTotals,
       AVG(Salary) OVER(PARTITION BY Gender) AS AvgSal,
       MIN(Salary) OVER(PARTITION BY Gender) AS MinSal,
       MAX(Salary) OVER(PARTITION BY Gender) AS MaxSal
FROM Employees
```

Part 109 - Row_Number function in SQL Server

ROW_NUMBER function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Row_Number function

- Introduced in SQL Server 2005
- Returns the sequential number of a row starting at 1
- ORDER BY clause is required
- PARTITION BY clause is optional
- When the data is partitioned, row number is reset to 1 when the partition changes

Syntax:

```
ROW_NUMBER() OVER (ORDER BY Col1, Col2)
```

ROW_NUMBER function in SQL Server

Row_Number function without PARTITION BY : In this example, data is not partitioned, so ROW_NUMBER will provide a consecutive numbering for all the rows in the table based on the order of rows imposed by the ORDER BY clause.

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Gender	Salary	RowNumber
Pam	Female	5500	1
Sara	Female	4000	2
Mary	Female	5000	3
Jodi	Female	7000	4
Tom	Male	5500	5
Ron	Male	5000	6
Ben	Male	6500	7
Todd	Male	3500	8
Mark	Male	5000	9
John	Male	4500	10

```
SELECT Name, Gender, Salary,
       ROW_NUMBER() OVER (ORDER BY Gender) AS RowNumber
FROM   Employees
```

Please note : If ORDER BY clause is not specified you will get the following error

```
The function 'ROW_NUMBER' must have an OVER clause with ORDER BY
```

ROW_NUMBER function in SQL Server

Row_Number function with PARTITION BY : In this example, data is partitioned by Gender, so ROW_NUMBER will provide a consecutive numbering only for the rows with in a parttion. When the partition changes the row number is reset to 1

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Gender	Salary	RowNumber
Pam	Female	5500	1
Sara	Female	4000	2
Mary	Female	5000	3
Jodi	Female	7000	4
Tom	Male	5500	1
Ron	Male	5000	2
Ben	Male	6500	3
Todd	Male	3500	4
Mark	Male	5000	5
John	Male	4500	6

```
SELECT Name, Gender, Salary,
       ROW_NUMBER() OVER (PARTITION BY Gender ORDER BY Gender) AS RowNumber
FROM   Employees
```

Use case for Row_Number function : Deleting all duplicate rows except one

Part 4 of SQL Server Interview Questions and Answers video series

Part 110 - Rank and Dense_Rank in SQL Server

RANK and DENSE_RANK in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit
http://pragimtech.com/order.aspx

RANK and DENSE_RANK functions

- Introduced in SQL Server 2005
- Return a rank starting at 1 based on the ordering of rows imposed by the ORDER BY clause
- ORDER BY clause is required
- PARTITION BY clause is optional
- When the data is partitioned, rank is reset to 1 when the partition changes

Difference between RANK and DENSE_RANK functions

RANK function skips ranking(s) if there is a tie where as DENSE_RANK will not

For example : If you have 2 rows at rank 1 and you have 5 rows in total

```
RANK() returns - 1, 1, 3, 4, 5
DENSE_RANK returns - 1, 1, 2, 3, 4
```

Syntax :

```
RANK() OVER (ORDER BY Col1, Col2, ...)
DENSE_RANK() OVER (ORDER BY Col1, Col2, ...)
```

RANK and DENSE_RANK in SQL Server

RANK() and DENSE_RANK() functions without PARTITION BY clause : In this example, data is not partitioned, so RANK() function provides a consecutive numbering except when there is a tie. Rank 2 is skipped as there are 2 rows at rank 1. The third row gets rank 3. DENSE_RANK() on the other hand will not skip ranks if there is a tie. The first 2 rows get rank 1. Third row gets rank 2.

```
SELECT Name, Salary, Gender,
RANK() OVER (ORDER BY Salary DESC) AS [Rank],
DENSE_RANK() OVER (ORDER BY Salary DESC) AS DenseRank
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	8000
2	John	Male	8000
3	Pam	Female	5000
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	6000
7	Ben	Male	6500
8	Jodi	Female	4500
9	Tom	Male	7000
10	Ron	Male	6800

Name	Salary	Gender	Rank	DenseRank
Mark	8000	Male	1	1
John	8000	Male	1	1
Tom	7000	Male	3	2
Ron	6800	Male	4	3
Ben	6500	Male	5	4
Mary	6000	Female	6	5
Pam	5000	Female	7	6
Jodi	4500	Female	8	7
Sara	4000	Female	9	8
Todd	3500	Male	10	9

RANK and DENSE_RANK in SQL Server

RANK() and DENSE_RANK() functions with PARTITION BY clause : Notice when the partition changes from Female to Male Rank is reset to 1

```
SELECT Name, Salary, Gender,
RANK() OVER (PARTITION BY Gender ORDER BY Salary DESC) AS [Rank],
DENSE_RANK() OVER (PARTITION BY Gender ORDER BY Salary DESC) AS DenseRank
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	8000
2	John	Male	8000
3	Pam	Female	5000
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	6000
7	Ben	Male	6500
8	Jodi	Female	4500
9	Tom	Male	7000
10	Ron	Male	6800

Name	Salary	Gender	Rank	DenseRank
Mary	6000	Female	1	1
Pam	5000	Female	2	2
Jodi	4500	Female	3	3
Sara	4000	Female	4	4
Mark	8000	Male	1	1
John	8000	Male	1	1
Tom	7000	Male	3	2
Ron	6800	Male	4	3
Ben	6500	Male	5	4
Todd	3500	Male	6	5

RANK and DENSE_RANK in SQL Server

Use case for RANK and DENSE_RANK functions : Both these functions can be used to find Nth highest salary. However, which function to use depends on what you want to do when there is a tie.

Since we have 2 Employees with the FIRST highest salary. Rank() function will not return any rows for the SECOND highest Salary.

```
WITH Result AS
(
    SELECT Salary, RANK() OVER (ORDER BY Salary DESC) AS Salary_Rank
    FROM Employees
)
SELECT TOP 1 Salary FROM Result WHERE Salary_Rank = 2
```

Though we have 2 Employees with the FIRST highest salary. Dense_Rank() function returns, the next Salary after the tied rows as the SECOND highest Salary

```
WITH Result AS
(
    SELECT Salary, DENSE_RANK() OVER (ORDER BY Salary DESC) AS Salary_Rank
    FROM Employees
)
SELECT TOP 1 Salary FROM Result WHERE Salary_Rank = 2
```

RANK and DENSE_RANK in SQL Server

You can also use RANK and DENSE_RANK functions to find the Nth highest Salary among Male or Female employee groups.

The following query finds the 3rd highest salary amount paid among the Female employees group

```
WITH Result AS
(
    SELECT Salary, Gender,
           DENSE_RANK() OVER (PARTITION BY Gender ORDER BY Salary DESC) AS Salary_Rank
    FROM Employees
)
SELECT TOP 1 Salary FROM Result WHERE Salary_Rank = 3 AND Gender = 'Female'
```

Part 111 - Difference between rank dense_rank and row_number in SQL

RANK, DENSE_RANK & ROW_NUMBER

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Similarities between RANK, DENSE_RANK and ROW_NUMBER functions

- Returns an increasing integer value starting at 1 based on the ordering of rows imposed by the ORDER BY clause (if there are no ties)
- ORDER BY clause is required
- PARTITION BY clause is optional
- When the data is partitioned, the integer value is reset to 1 when the partition changes

Id	Name	Gender	Salary
1	Mark	Male	6000
2	John	Male	8000
3	Pam	Female	4000
4	Sara	Female	5000
5	Todd	Male	3000

Name	Salary	Gender	RowNumber	Rank	DenseRank
John	8000	Male	1	1	1
Mark	6000	Male	2	2	2
Sara	5000	Female	3	3	3
Pam	4000	Female	4	4	4
Todd	3000	Male	5	5	5

```
SELECT Name, Salary, Gender,
ROW_NUMBER() OVER (ORDER BY Salary DESC) AS RowNumber,
RANK() OVER (ORDER BY Salary DESC) AS [Rank],
DENSE_RANK() OVER (ORDER BY Salary DESC) AS DenseRank
FROM Employees
```

RANK, DENSE_RANK & ROW_NUMBER

You will only see the difference when there are ties

Id	Name	Gender	Salary
1	Mark	Male	8000
2	John	Male	8000
3	Pam	Female	8000
4	Sara	Female	4000
5	Todd	Male	3500

Name	Salary	Gender	RowNumber	Rank	DenseRank
Mark	8000	Male	1	1	1
John	8000	Male	2	1	1
Pam	8000	Female	3	1	1
Sara	4000	Female	4	4	2
Todd	3500	Male	5	5	3

Difference between RANK, DENSE_RANK and ROW_NUMBER functions

- **ROW_NUMBER :** Returns an increasing unique number for each row starting at 1, even if there are duplicates.
- **RANK :** Returns an increasing unique number for each row starting at 1. When there are duplicates, same rank is assigned to all the duplicate rows, but the next row after the duplicate rows will have the rank it would have been assigned if there had been no duplicates. So RANK function skips rankings if there are duplicates.
- **DENSE_RANK :** Returns an increasing unique number for each row starting at 1. When there are duplicates, same rank is assigned to all the duplicate rows but the DENSE_RANK function will not skip any ranks. This means the next row after the duplicate rows will have the next rank in the sequence.

Part 112 - Calculate running total in SQL Server 2012

Running Total in SQL Server 2012

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

SQL Query to compute running total without partitions

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Gender	Salary	RunningTotal
Mark	Male	5000	5000
John	Male	4500	9500
Pam	Female	5500	15000
Sara	Female	4000	19000
Todd	Male	3500	22500
Mary	Female	5000	27500
Ben	Male	6500	34000
Jodi	Female	7000	41000
Tom	Male	5500	46500
Ron	Male	5000	51500

```
SELECT Name, Gender, Salary,
       SUM(Salary) OVER (ORDER BY ID) AS RunningTotal
FROM Employees
```

Running Total in SQL Server 2012

SQL Query to compute running total with partitions

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Gender	Salary	RunningTotal
Pam	Female	5500	5500
Sara	Female	4000	9500
Mary	Female	5000	14500
Jodi	Female	7000	21500
Mark	Male	5000	5000
John	Male	4500	9500
Todd	Male	3500	13000
Ben	Male	6500	19500
Tom	Male	5500	25000
Ron	Male	5000	30000

```
SELECT Name, Gender, Salary,
       SUM(Salary) OVER (PARTITION BY Gender ORDER BY ID) AS RunningTotal
FROM Employees
```

Running Total in SQL Server 2012

What happens if I use order by on Salary column

If you have duplicate values in the Salary column, all the duplicate values will be added to the running total at once. In the example below notice that we have 5000 repeated 3 times. So 15000 (i.e 5000 + 5000 + 5000) is added to the running total at once

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Gender	Salary	RunningTotal
Todd	Male	3500	3500
Sara	Female	4000	7500
John	Male	4500	12000
Mark	Male	5000	27000
Mary	Female	5000	27000
Ron	Male	5000	27000
Tom	Male	5500	38000
Pam	Female	5500	38000
Ben	Male	6500	44500
Jodi	Female	7000	51500

```
SELECT Name, Gender, Salary,
       SUM(Salary) OVER (ORDER BY Salary) AS RunningTotal
FROM Employees
```

So when computing running total, it is better to use a column that has unique data in the ORDER BY clause.

Part 113 - NTILE function in SQL Server

NTILE function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

NTILE function

- Introduced in SQL Server 2005
- ORDER BY Clause is required
- PARTITION BY clause is optional
- Distributes the rows into a specified number of groups
- If the number of rows is not divisible by number of groups, you may have groups of two different sizes.
- Larger groups come before smaller groups
- For example,

  NTILE(2) of 10 rows divides the rows in 2 Groups (5 in each group)

  NTILE(3) of 10 rows divides the rows in 3 Groups (4 in first group, 3 in 2nd & 3rd group)

Syntax:

```
NTILE (Number_of_Groups) OVER (ORDER BY Col1, Col2, ...)
```

NTILE function in SQL Server

NTILE function without PARTITION BY clause : Divides the 10 rows into 3 groups. 4 rows in first group, 3 rows in the 2nd & 3rd group

```
SELECT Name, Gender, Salary,
NTILE(3) OVER (ORDER BY Salary) AS [Ntile]
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Gender	Salary	Ntile
Todd	Male	3500	1
Sara	Female	4000	1
John	Male	4500	1
Mark	Male	5000	1
Mary	Female	5000	2
Ron	Male	5000	2
Tom	Male	5500	2
Pam	Female	5500	3
Ben	Male	6500	3
Jodi	Female	7000	3

NTILE function in SQL Server

What if the specified number of groups is GREATER THAN the number of rows
It will try to create as many groups as possible with one row in each group.

With 10 rows in the table, NTILE(11) will create 10 groups with 1 row in each group.

```
SELECT Name, Gender, Salary,
NTILE(11) OVER (ORDER BY Salary) AS [Ntile]
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Gender	Salary	Ntile
Todd	Male	3500	1
Sara	Female	4000	2
John	Male	4500	3
Mark	Male	5000	4
Mary	Female	5000	5
Ron	Male	5000	6
Tom	Male	5500	7
Pam	Female	5500	8
Ben	Male	6500	9
Jodi	Female	7000	10

NTILE function in SQL Server

NTILE function with PARTITION BY clause : When the data is partitioned, NTILE function creates the specified number of groups with in each partition.

The following query partitions the data into 2 partitions (Male & Female). NTILE(3) creates 3 groups in each of the partitions.

```
SELECT Name, Gender, Salary,
NTILE(3) OVER (PARTITION BY GENDER ORDER BY Salary) AS [Ntile]
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	5000
2	John	Male	4500
3	Pam	Female	5500
4	Sara	Female	4000
5	Todd	Male	3500
6	Mary	Female	5000
7	Ben	Male	6500
8	Jodi	Female	7000
9	Tom	Male	5500
10	Ron	Male	5000

Name	Gender	Salary	Ntile
Sara	Female	4000	1
Mary	Female	5000	1
Pam	Female	5500	2
Jodi	Female	7000	3
Todd	Male	3500	1
John	Male	4500	1
Mark	Male	5000	2
Ron	Male	5000	2
Tom	Male	5500	3
Ben	Male	6500	3

Part 114 - Lead and Lag functions in SQL Server 2012

Lead & Lag functions in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Lead and Lag functions

- Introduced in SQL Server 2012
- Lead function is used to access subsequent row data along with current row data
- Lag function is used to access previous row data along with current row data
- ORDER BY clause is required
- PARTITION BY clause is optional

Syntax :

```
LEAD(Column_Name, Offset, Default_Value) OVER (ORDER BY Col1, Col2, ...)
LAG (Column_Name, Offset, Default_Value) OVER (ORDER BY Col1, Col2, ...)
```

Offset	Number of rows to lead or lag
Default_Value	The default value to return if the number of rows to lead or lag goes beyond first row or last row in a table or partition. If default value is not specified NULL is returned.

Lead & Lag functions in SQL Server

Lead and Lag functions example WITHOUT partitions : This example Leads 2 rows and Lags 1 row from the current row

```
SELECT Name, Gender, Salary,
       LEAD(Salary, 2, -1) OVER (ORDER BY Salary) AS Lead_2,
       LAG(Salary, 1, -1) OVER (ORDER BY Salary) AS Lag_1
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	1000
2	John	Male	2000
3	Pam	Female	3000
4	Sara	Female	4000
5	Todd	Male	5000
6	Mary	Female	6000
7	Ben	Male	7000
8	Jodi	Female	8000
9	Tom	Male	9000
10	Ron	Male	9500

Name	Gender	Salary	Lead_2	Lag_1
Mark	Male	1000	3000	-1
John	Male	2000	4000	1000
Pam	Female	3000	5000	2000
Sara	Female	4000	6000	3000
Todd	Male	5000	7000	4000
Mary	Female	6000	8000	5000
Ben	Male	7000	9000	6000
Jodi	Female	8000	9500	7000
Tom	Male	9000	-1	8000
Ron	Male	9500	-1	9000

Lead & Lag functions in SQL Server

Lead and Lag functions example WITH partitions : Notice that in this example, Lead and Lag functions return default value if the number of rows to lead or lag goes beyond first row or last row in the partition.

```
SELECT Name, Gender, Salary,
       LEAD(Salary, 2, -1) OVER (PARTITION By Gender ORDER BY Salary) AS Lead_2,
       LAG(Salary, 1, -1) OVER (PARTITION By Gender ORDER BY Salary) AS Lag_1
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	1000
2	John	Male	2000
3	Pam	Female	3000
4	Sara	Female	4000
5	Todd	Male	5000
6	Mary	Female	6000
7	Ben	Male	7000
8	Jodi	Female	8000
9	Tom	Male	9000
10	Ron	Male	9500

Name	Gender	Salary	Lead_2	Lag_1
Pam	Female	3000	6000	-1
Sara	Female	4000	8000	3000
Mary	Female	6000	-1	4000
Jodi	Female	8000	-1	6000
Mark	Male	1000	5000	-1
John	Male	2000	7000	1000
Todd	Male	5000	9000	2000
Ben	Male	7000	9500	5000
Tom	Male	9000	-1	7000
Ron	Male	9500	-1	9000

Part 115 - FIRST_VALUE function in SQL Server

FIRST_VALUE function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

FIRST_VALUE function

- Introduced in SQL Server 2012
- Retrieves the first value from the specified column
- ORDER BY clause is required
- PARTITION BY clause is optional

Syntax :

```
FIRST_VALUE(Column_Name) OVER (ORDER BY Col1, Col2, ...)
```

FIRST_VALUE function in SQL Server

FIRST_VALUE function example WITHOUT partitions : Returns the name of the lowest paid employee from the entire table

```
SELECT Name, Gender, Salary,
FIRST_VALUE(Name) OVER (ORDER BY Salary) AS FirstValue
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	1000
2	John	Male	2000
3	Pam	Female	3000
4	Sara	Female	4000
5	Todd	Male	5000
6	Mary	Female	6000
7	Ben	Male	7000
8	Jodi	Female	8000
9	Tom	Male	9000
10	Ron	Male	9500

→

Name	Gender	Salary	FirstValue
Mark	Male	1000	Mark
John	Male	2000	Mark
Pam	Female	3000	Mark
Sara	Female	4000	Mark
Todd	Male	5000	Mark
Mary	Female	6000	Mark
Ben	Male	7000	Mark
Jodi	Female	8000	Mark
Tom	Male	9000	Mark
Ron	Male	9500	Mark

FIRST_VALUE function in SQL Server

FIRST_VALUE function example WITH partitions : Returns the name of the lowest paid employee from the respective partition

```
SELECT Name, Gender, Salary,
FIRST_VALUE(Name) OVER (PARTITION BY Gender ORDER BY Salary) AS FirstValue
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	1000
2	John	Male	2000
3	Pam	Female	3000
4	Sara	Female	4000
5	Todd	Male	5000
6	Mary	Female	6000
7	Ben	Male	7000
8	Jodi	Female	8000
9	Tom	Male	9000
10	Ron	Male	9500

→

Name	Gender	Salary	FirstValue
Pam	Female	3000	Pam
Sara	Female	4000	Pam
Mary	Female	6000	Pam
Jodi	Female	8000	Pam
Mark	Male	1000	Mark
John	Male	2000	Mark
Todd	Male	5000	Mark
Ben	Male	7000	Mark
Tom	Male	9000	Mark
Ron	Male	9500	Mark

Part 116 - Window functions in SQL Server

Window functions in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Different categories of window functions

- Aggregate functions - AVG, SUM, COUNT, MIN, MAX etc..
- Ranking functions - RANK, DENSE_RANK, ROW_NUMBER etc..
- Analytic functions - LEAD, LAG, FIRST_VALUE, LAST_VALUE etc...

OVER Clause defines the partitioning and ordering of rows (i.e a window) for the above functions to operate on. Hence these functions are called window functions. The OVER clause accepts the following three arguments to define a window for these functions to operate on

- ORDER BY : Defines the logical order of the rows
- PARTITION BY : Divides the query result set into partitions. The window function is applied to each partition separately.
- ROWS or RANGE clause : Further limits the rows within the partition by specifying start and end points within the partition.

The default for ROWS or RANGE clause is

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

Window functions in SQL Server

Compute average salary and display it against every employee row

```
SELECT Name, Gender, Salary,
       AVG(Salary) OVER(ORDER BY Salary
           ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS Average
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	1000
2	John	Male	2000
3	Pam	Female	3000
4	Sara	Female	4000
5	Todd	Male	5000
6	Mary	Female	6000
7	Ben	Male	7000
8	Jodi	Female	8000
9	Tom	Male	9000
10	Ron	Male	9500

→

Name	Gender	Salary	Average
Mark	Male	1000	5450
John	Male	2000	5450
Pam	Female	3000	5450
Sara	Female	4000	5450
Todd	Male	5000	5450
Mary	Female	6000	5450
Ben	Male	7000	5450
Jodi	Female	8000	5450
Tom	Male	9000	5450
Ron	Male	9500	5450

Window functions in SQL Server

Compute the average salary of

1. **The current row**
2. **One row PRECEDING the current row and**
3. **One row FOLLOWING the current row**

```
SELECT Name, Gender, Salary,
       AVG(Salary) OVER(ORDER BY Salary
                        ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS Average
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	1000
2	John	Male	2000
3	Pam	Female	3000
4	Sara	Female	4000
5	Todd	Male	5000
6	Mary	Female	6000
7	Ben	Male	7000
8	Jodi	Female	8000
9	Tom	Male	9000
10	Ron	Male	9500

→

Name	Gender	Salary	Average
Mark	Male	1000	1500
John	Male	2000	2000
Pam	Female	3000	3000
Sara	Female	4000	4000
Todd	Male	5000	5000
Mary	Female	6000	6000
Ben	Male	7000	7000
Jodi	Female	8000	8000
Tom	Male	9000	8833
Ron	Male	9500	9250

Part 117 - Difference between rows and range

Difference between ROWS and RANGE

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Calculate the running total of Salary and display it against every employee row

Id	Name	Salary
1	Mark	1000
2	John	2000
3	Pam	3000
4	Sara	4000
5	Todd	3000

Name	Salary	RunningTotal
Mark	1000	1000
John	2000	3000
Pam	3000	6000
Sara	4000	10000
Todd	5000	15000

ROWS or RANGE clause is not specified. It's default value is used.

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary) AS RunningTotal
FROM Employees
```

The above query can be re-written using an explicit value for ROWS or RANGE clause

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary
       RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM Employees
```

Difference between ROWS and RANGE

Calculate the running total of Salary and display it against every employee row

Id	Name	Salary
1	Mark	1000
2	John	2000
3	Pam	3000
4	Sara	4000
5	Todd	3000

Name	Salary	RunningTotal
Mark	1000	1000
John	2000	3000
Pam	3000	6000
Sara	4000	10000
Todd	5000	15000

We can also achieve the same result, by replacing RANGE with ROWS

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary
       ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM Employees
```

What is the difference between ROWS and RANGE

The difference is in the way duplicate rows are treated. ROWS treat duplicates as distinct values, where as RANGE treats them as a single entity.

Difference between ROWS and RANGE

Calculate the running total of Salary and display it against every employee row

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary
       ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM Employees
```

Id	Name	Salary
1	Mark	1000
2	John	1000
3	Pam	3000
4	Sara	3000
5	Todd	3000

→

Name	Salary	RunningTotal
Mark	1000	1000
John	1000	2000
Pam	3000	5000
Sara	3000	8000
Todd	5000	13000

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary
       RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS RunningTotal
FROM Employees
```

Id	Name	Salary
1	Mark	1000
2	John	1000
3	Pam	3000
4	Sara	3000
5	Todd	3000

→

Name	Salary	RunningTotal
Mark	1000	2000
John	1000	2000
Pam	3000	8000
Sara	3000	8000
Todd	5000	13000

Difference between ROWS and RANGE

All together side by side. The following query shows how running total changes

1. **When no value is specified for ROWS or RANGE clause**
2. **When RANGE clause is used explicitly with it's default value**
3. **When ROWS clause is used instead of RANGE clause**

```
SELECT Name, Salary,
       SUM(Salary) OVER(ORDER BY Salary) AS [Default],
       SUM(Salary) OVER(ORDER BY Salary
                        RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS [Range],
       SUM(Salary) OVER(ORDER BY Salary
                        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS [Rows]
FROM Employees
```

Id	Name	Salary
1	Mark	1000
2	John	1000
3	Pam	3000
4	Sara	3000
5	Todd	3000

→

Name	Salary	Default	Range	Rows
Mark	1000	2000	2000	1000
John	1000	2000	2000	2000
Pam	3000	8000	8000	5000
Sara	3000	8000	8000	8000
Todd	5000	13000	13000	13000

Part 118 - LAST_VALUE function in SQL Server

LAST_VALUE function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

LAST_VALUE function

- Introduced in SQL Server 2012
- Retrieves the last value from the specified column
- ORDER BY clause is required
- PARTITION BY clause is optional
- ROWS or RANGE clause is optional, but for it to work correctly you may have to explicitly specify a value

Syntax:

```
LAST_VALUE(Column_Name) OVER (ORDER BY Col1, Col2, ...)
```

LAST_VALUE function in SQL Server

LAST_VALUE function not working as expected : In the following example, LAST_VALUE function does not return the name of the highest paid employee. This is because we have not specified an explicit value for ROWS or RANGE clause. As a result it is using it's default value `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`

```
SELECT Name, Gender, Salary,
       LAST_VALUE(Name) OVER (ORDER BY Salary) AS LastValue
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	1000
2	John	Male	2000
3	Pam	Female	3000
4	Sara	Female	4000
5	Todd	Male	5000
6	Mary	Female	6000
7	Ben	Male	7000
8	Jodi	Female	8000
9	Tom	Male	9000
10	Ron	Male	9500

Name	Gender	Salary	LastValue
Mark	Male	1000	Mark
John	Male	2000	John
Pam	Female	3000	Pam
Sara	Female	4000	Sara
Todd	Male	5000	Todd
Mary	Female	6000	Mary
Ben	Male	7000	Ben
Jodi	Female	8000	Jodi
Tom	Male	9000	Tom
Ron	Male	9500	Ron

LAST_VALUE function in SQL Server

LAST_VALUE function working as expected : LAST_VALUE function in this case returns the name of the highest paid employee as expected, as we have set an explicit value for ROWS or RANGE clause to `ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING`

This tells the LAST_VALUE function that it's window starts at the first row and ends at the last row in the result set

```
SELECT Name, Gender, Salary,
       LAST_VALUE(Name) OVER (ORDER BY Salary ROWS BETWEEN UNBOUNDED PRECEDING
                              AND UNBOUNDED FOLLOWING) AS LastValue
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	1000
2	John	Male	2000
3	Pam	Female	3000
4	Sara	Female	4000
5	Todd	Male	5000
6	Mary	Female	6000
7	Ben	Male	7000
8	Jodi	Female	8000
9	Tom	Male	9000
10	Ron	Male	9500

Name	Gender	Salary	LastValue
Mark	Male	1000	Ron
John	Male	2000	Ron
Pam	Female	3000	Ron
Sara	Female	4000	Ron
Todd	Male	5000	Ron
Mary	Female	6000	Ron
Ben	Male	7000	Ron
Jodi	Female	8000	Ron
Tom	Male	9000	Ron
Ron	Male	9500	Ron

LAST_VALUE function in SQL Server

LAST_VALUE function example with partitions : In the following example, LAST_VALUE function returns the name of the highest paid employee from the respective partition

```
SELECT Name, Gender, Salary,
       LAST_VALUE(Name) OVER (PARTITION BY Gender ORDER BY Salary ROWS BETWEEN
                              UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS LastValue
FROM Employees
```

Id	Name	Gender	Salary
1	Mark	Male	1000
2	John	Male	2000
3	Pam	Female	3000
4	Sara	Female	4000
5	Todd	Male	5000
6	Mary	Female	6000
7	Ben	Male	7000
8	Jodi	Female	8000
9	Tom	Male	9000
10	Ron	Male	9500

Name	Gender	Salary	LastValue
Pam	Female	3000	Jodi
Sara	Female	4000	Jodi
Mary	Female	6000	Jodi
Jodi	Female	8000	Jodi
Mark	Male	1000	Ron
John	Male	2000	Ron
Todd	Male	5000	Ron
Ben	Male	7000	Ron
Tom	Male	9000	Ron
Ron	Male	9500	Ron

Part 119 - UNPIVOT in SQL Server

UNPIVOT in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

PIVOT operator turns ROWS into COLUMNS, where as UNPIVOT turns COLUMNS into ROWS

Part 53 - PIVOT Operator

SalesAgent	India	US	UK
David	960	520	360
John	970	540	800

SalesAgent	Country	SalesAmount
David	India	960
David	US	520
David	UK	360
John	India	970
John	US	540
John	UK	800

```
SELECT SalesAgent, Country, SalesAmount
FROM tblProductSales
UNPIVOT
(
        SalesAmount
        FOR Country IN (India, US ,UK)
) AS UnpivotExample
```

Part 120 - Reverse PIVOT table in SQL Server

Reverse PIVOT table in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Is it always possible to reverse what PIVOT operator has done using UNPIVOT operator

No, not always. If the PIVOT operator has not aggregated the data, you can get your original data back using the UNPIVOT operator but not if the data is aggregated.

PIVOT operator turns ROWS into COLUMNS

SalesAgent	Country	SalesAmount
David	India	960
David	US	520
John	India	970
John	US	540

```
SELECT SalesAgent, India, US
FROM tblProductSales
PIVOT
(
    SUM(SalesAmount)
    FOR Country IN (India, US)
) AS PivotTable
```

SalesAgent	India	US
David	960	520
John	970	540

Reverse PIVOT table in SQL Server

UNPIVOT operator reverses what PIVOT operator has done (turn COLUMNS into ROWS)

SalesAgent	India	US
David	960	520
John	970	540

```
SELECT SalesAgent, Country, SalesAmount
FROM
(SELECT SalesAgent, India, US
FROM tblProductSales
PIVOT
(
    SUM(SalesAmount)
    FOR Country IN (India, US)
) AS PivotTable) P
UNPIVOT
(
    SalesAmount
    FOR Country IN (India, US)
) AS UnpivotTable
```

We are able to get the original data back, because the SUM aggregate function that we used with the PIVOT operator did not perform any aggregation

SalesAgent	Country	SalesAmount
David	India	960
David	US	520
John	India	970
John	US	540

Reverse PIVOT table in SQL Server

Notice that for SalesAgent - David and Country - India we have 2 rows. These rows are aggregated in the result-set.

SalesAgent	Country	SalesAmount
David	India	960
David	US	520
John	India	970
John	US	540
David	India	100

→

```
SELECT SalesAgent, India, US
FROM tblProductSales
PIVOT
(
    SUM(SalesAmount)
    FOR Country IN (India, US)
) AS PivotTable
```

→

SalesAgent	India	US
David	1060	520
John	970	540

Reverse PIVOT table in SQL Server

UNPIVOT operator reverses what PIVOT operator has done

SalesAgent	India	US
David	1060	520
John	970	540

```
SELECT SalesAgent, Country, SalesAmount
FROM
(SELECT SalesAgent, India, US
FROM tblProductSales
PIVOT
(
    SUM(SalesAmount)
    FOR Country IN (India, US)
) AS PivotTable) P
UNPIVOT
(
    SalesAmount
    FOR Country IN (India, US)
) AS UnpivotTable
```

We are unable to get the original data back, because the SUM aggregate function that we used with the PIVOT operator has aggregated the data, and there is no way for SQL Server to know how to undo the aggregation

SalesAgent	Country	SalesAmount
David	India	1060
David	US	520
John	India	970
John	US	540

Part 121 - Choose function in SQL Server

Choose function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Choose function

- Introduced in SQL Server 2012
- Returns the item at the specified index from the list of available values
- The index position starts at 1 and NOT 0 (ZERO)

Syntax:

```
CHOOSE ( index, val_1, val_2, ... )
```

Example: Returns the item at index position 2

```
SELECT CHOOSE(2, 'India','US', 'UK') AS Country
```

Country
US

Choose function in SQL Server

Example : Using CASE statement

```sql
SELECT Name, DateOfBirth,
       CASE DATEPART(MM, DateOfBirth)
          WHEN 1 THEN 'JAN'
          WHEN 2 THEN 'FEB'
          WHEN 3 THEN 'MAR'
          WHEN 4 THEN 'APR'
          WHEN 5 THEN 'MAY'
          WHEN 6 THEN 'JUN'
          WHEN 7 THEN 'JUL'
          WHEN 8 THEN 'AUG'
          WHEN 9 THEN 'SEP'
          WHEN 10 THEN 'OCT'
          WHEN 11 THEN 'NOV'
          WHEN 12 THEN 'DEC'
       END
       AS [MONTH]
FROM Employees
```

Id	Name	DateOfBirth
1	Mark	1980-01-11
2	John	1981-12-12
3	Amy	1979-11-21
4	Ben	1978-05-14
5	Sara	1970-03-17
6	David	1978-04-05

Name	DateOfBirth	MONTH
Mark	1980-01-11	JAN
John	1981-12-12	DEC
Amy	1979-11-21	NOV
Ben	1978-05-14	MAY
Sara	1970-03-17	MAR
David	1978-04-05	APR

Choose function in SQL Server

Example : Using CHOOSE function

```sql
SELECT Name, DateOfBirth,CHOOSE(DATEPART(MM, DateOfBirth),
       'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG',
       'SEP', 'OCT', 'NOV', 'DEC') AS [MONTH]
FROM Employees
```

Id	Name	DateOfBirth
1	Mark	1980-01-11
2	John	1981-12-12
3	Amy	1979-11-21
4	Ben	1978-05-14
5	Sara	1970-03-17
6	David	1978-04-05

Name	DateOfBirth	MONTH
Mark	1980-01-11	JAN
John	1981-12-12	DEC
Amy	1979-11-21	NOV
Ben	1978-05-14	MAY
Sara	1970-03-17	MAR
David	1978-04-05	APR

Part 122 - IIF function in SQL Server

IIF function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

IIF function

- Introduced in SQL Server 2012
- Returns one of two the values, depending on whether the Boolean expression evaluates to true or false
- IIF is a shorthand way for writing a CASE expression

Syntax :

```
IIF ( boolean_expression, true_value, false_value )
```

Example : Returns Male as the boolean expression evaluates to TRUE

```
DECLARE @Gender INT
SET @Gender = 1
SELECT IIF( @Gender = 1, 'Male', 'Femlae') AS Gender
```

Gender
Male

IIF function in SQL Server

Example : Using IIF() function with table data

Id	Name	GenderId
1	Mark	1
2	John	1
3	Amy	2
4	Ben	1
5	Sara	2
6	David	1

Name	GenderId	Gender
Mark	1	Male
John	1	Male
Amy	2	Female
Ben	1	Male
Sara	2	Female
David	1	Male

Using CASE statement

```
SELECT Name, GenderId,
       CASE WHEN GenderId = 1
            THEN 'Male'
            ELSE 'Female'
       END AS Gender
FROM Employees
```

Using IIF function

```
SELECT Name, GenderId,
       IIF(GenderId = 1, 'Male', 'Female') AS Gender
FROM Employees
```

Part 123 - TRY_PARSE function in SQL Server 2012

TRY_PARSE function in SQL Server 2012

Use CASE statement or IIF function to provide a meaningful error message instead of NULL when the conversion fails

Example : Using CASE statement to provide a meaningful error message when the conversion fails

```
SELECT
CASE WHEN TRY_PARSE('ABC' AS INT) IS NULL
     THEN 'Conversion Failed'
     ELSE 'Conversion Successful'
END AS Result
```

Result
Conversion Failed

Example : Using IIF function to provide a meaningful error message when the conversion fails

```
SELECT IIF(TRY_PARSE('ABC' AS INT) IS NULL,
           'Conversion Failed',
           'Conversion Successful') AS Result
```

Result
Conversion Failed

TRY_PARSE function in SQL Server 2012

What is the difference between PARSE and TRY_PARSE

PARSE will result in an error if the conversion fails, where as TRY_PARSE will return NULL instead of an error

Since ABC cannot be converted to INT, PARSE will return an error

```
SELECT PARSE('ABC' AS INT) AS Result
```

↓

```
Msg 9819, Level 16, State 1, Line 1
Error converting string value 'ABC' into data type int using culture ''.
```

Since ABC cannot be converted to INT, TRY_PARSE will return NULL instead of an error

```
SELECT TRY_PARSE('ABC' AS INT) AS Result
```

↓

Result
NULL

TRY_PARSE function in SQL Server 2012

Example : Using TRY_PARSE() function with table data

```
SELECT Name, TRY_PARSE(Age AS INT) AS Age
FROM Employees
```

Id	Name	Age
1	Mark	40
2	John	20
3	Amy	THIRTY
4	Ben	21
5	Sara	FIFTY
6	David	25

Name	Age
Mark	40
John	20
Amy	NULL
Ben	21
Sara	NULL
David	25

If you use PARSE instead of TRY_PARSE, the query fails with an error

```
SELECT NAME, PARSE(Age AS INT) AS Age
FROM Employees
```

```
Msg 9819, Level 16, State 1, Line 1
Error converting string value 'THIRTY' into data type int using culture ''.
```

TRY_PARSE function in SQL Server 2012

If you are in need of the DVD with all the videos and PPT's, please visit
http://pragimtech.com/order.aspx

TRY_PARSE function

- Introduced in SQL Server 2012
- Converts a string to Date/Time or Numeric type
- Returns NULL if the provided string cannot be converted to the specified data type
- Requires .NET Framework Common Language Runtime (CLR)

Syntax :

```
TRY_PARSE ( string_value AS data_type )
```

Example : As the string can be converted to INT, the result will be 99

```
SELECT TRY_PARSE('99' AS INT) AS Result
```

Result
99

Example : As the string cannot be converted to INT, TRY_PARSE returns NULL

```
SELECT TRY_PARSE('ABC' AS INT) AS Result
```

Result
NULL

Part 124 - TRY_CONVERT function in SQL Server 2012

TRY_CONVERT function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit
http://pragimtech.com/order.aspx

TRY_CONVERT function

- Introduced in SQL Server 2012
- Converts a value to the specified data type
- Returns NULL if the provided value cannot be converted to the specified data type
- If you request a conversion that is explicitly not permitted, then TRY_CONVERT fails with an error

Syntax :

```
TRY_CONVERT ( data_type, value, [style] )
```

Style parameter is optional. The range of acceptable values is determined by the target data_type
https://msdn.microsoft.com/en-us/library/ms187928.aspx

Example : As the string can be converted to INT, the result will be 99

```
SELECT TRY_CONVERT(INT, '99') AS Result
```

Result
99

TRY_CONVERT function in SQL Server

Example : As the string cannot be converted to INT, TRY_CONVERT returns NULL

```
SELECT TRY_CONVERT(INT, 'ABC') AS Result
```

Result
NULL

Example : Converting an integer to XML is not explicitly permitted. so in this case TRY_CONVERT fails with an error

```
SELECT TRY_CONVERT(XML, 10) AS Result
```

CASE statement or IIF function can be used to provide a meaningful error message instead of NULL when the conversion fails

Using CASE statement to provide a meaningful error message when the conversion fails

```
SELECT
CASE WHEN TRY_CONVERT(INT, 'ABC') IS NULL
     THEN 'Conversion Failed'
     ELSE 'Conversion Successful'
END AS Result
```

Result
Conversion Failed

Using IIF function to provide a meaningful error message when the conversion fails

```
SELECT IIF(TRY_CONVERT(INT, 'ABC') IS NULL, 'Conversion Failed',
           'Conversion Successful') AS Result
```

Result
Conversion Failed

TRY_CONVERT function in SQL Server

What is the difference between CONVERT and TRY_CONVERT

CONVERT will result in an error if the conversion fails, where as TRY_CONVERT will return NULL instead of an error

Since ABC cannot be converted to INT, CONVERT will return an error

```
SELECT CONVERT(INT, 'ABC') AS Result
```

Since ABC cannot be converted to INT, TRY_CONVERT will return NULL instead of an error

```
SELECT TRY_CONVERT(INT, 'ABC') AS Result
```

→

Result
NULL

Example : Using TRY_CONVERT() function with table data

```
SELECT Name, TRY_CONVERT(INT, Age) AS Age
FROM Employees
```

→

Id	Name	Age
1	Mark	40
2	John	20
3	Amy	THIRTY
4	Ben	21
5	Sara	FIFTY
6	David	25

→

Name	Age
Mark	40
John	20
Amy	NULL
Ben	21
Sara	NULL
David	25

TRY_CONVERT function in SQL Server

If you use CONVERT instead of TRY_CONVERT, the query fails with an error

```
SELECT NAME, CONVERT(INT, Age) AS Age
FROM Employees
```

↓

```
Conversion failed when converting the nvarchar value 'THIRTY' to data type int
```

Difference between TRY_PARSE and TRY_CONVERT functions

TRY_PARSE can only be used for converting from string to date/time or number data types where as TRY_CONVERT can be used for any general type conversions.

For example, you can use TRY_CONVERT to convert a string to XML data type, where as you can do the same using TRY_PARSE

```
SELECT TRY_CONVERT(XML, '<root><child/></root>') AS [XML]
```

→

XML
<root><child /></root>

```
SELECT TRY_PARSE('<root><child/></root>' AS XML) AS [XML]
```

↓

```
Invalid data type xml in function TRY_PARSE
```

Another difference is TRY_PARSE relies on the presence of .the .NET Framework Common Language Runtime (CLR) where as TRY_CONVERT does not.

Part 125 - EOMONTH function in SQL Server 2012

EOMONTH function in SQL Server 2012

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

EOMONTH function

- Introduced in SQL Server 2012
- Returns the last day of the month of the specified date

Syntax :

```
EOMONTH ( start_date [, month_to_add ] )
```

Parameter	Description
start_date	The date for which to return the last day of the month
month_to_add	Optional. Number of months to add to the start_date. EOMONTH adds the specified number of months to start_date, and then returns the last day of the month for the resulting date.

Example : Returns last day of the November month

```
SELECT EOMONTH('11/20/2015') AS LastDay
```

→

LastDay
30/11/2015

EOMONTH function in SQL Server 2012

Example : Returns last day of the month of February from a NON-LEAP year

```
SELECT EOMONTH('2/20/2015') AS LastDay
```

→

LastDay
28/02/2015

Example : Returns last day of the month of February from a LEAP year

```
SELECT EOMONTH('2/20/2016') AS LastDay
```

→

LastDay
29/02/2016

month_to_add optional parameter can be used to add or subtract a specified number of months from the start_date, and then return the last day of the month from the resulting date.

Adds 2 months to the start_date and returns the last day of the month from the resulting date

```
SELECT EOMONTH('3/20/2016', 2) AS LastDay
```

→

LastDay
31/05/2016

Subtracts 1 month from the start_date and returns the last day of the month from the resulting date

```
SELECT EOMONTH('3/20/2016', -1) AS LastDay
```

→

LastDay
29/02/2016

EOMONTH function in SQL Server 2012

Using EOMONTH function with table data : Returns the last day of the month from the DateOfBirth of every employee

```
SELECT Name, DateOfBirth, EOMONTH(DateOfBirth) AS LastDay
FROM Employees
```

Id	Name	DateOfBirth
1	Mark	11/01/1980
2	John	12/12/1981
3	Amy	21/11/1979
4	Ben	14/05/1978
5	Sara	17/03/1970
6	David	05/04/1978

Name	DateOfBirth	LastDay
Mark	11/01/1980	31/01/1980
John	12/12/1981	31/12/1981
Amy	21/11/1979	30/11/1979
Ben	14/05/1978	31/05/1978
Sara	17/03/1970	31/03/1970
David	05/04/1978	30/04/1978

EOMONTH function in SQL Server 2012

Use DATEPART function to retrieve just the last day instead of the full date

```
SELECT Name, DateOfBirth, DATEPART(DD,EOMONTH(DateOfBirth)) AS LastDay
FROM Employees
```

Id	Name	DateOfBirth
1	Mark	11/01/1980
2	John	12/12/1981
3	Amy	21/11/1979
4	Ben	14/05/1978
5	Sara	17/03/1970
6	David	05/04/1978

Name	DateOfBirth	LastDay
Mark	11/01/1980	31
John	12/12/1981	31
Amy	21/11/1979	30
Ben	14/05/1978	31
Sara	17/03/1970	31
David	05/04/1978	30

Part 126 - DATEFROMPARTS function

DATEFROMPARTS function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

DATEFROMPARTS function

- Introduced in SQL Server 2012
- Returns a date value for the specified year, month, and day
- The data type of all the 3 parameters (year, month, and day) is integer
- If invalid argument values are specified, the function returns an error
- If any of the arguments are NULL, the function returns null

Syntax:

```
DATEFROMPARTS ( year, month, day )
```

Example : All the arguments have valid values, so DATEFROMPARTS function returns the expected date

```
SELECT DATEFROMPARTS ( 2015, 10, 25) AS [Date]
```

Date
25/10/2015

DATEFROMPARTS function in SQL Server

Example : Invalid value specified for month parameter, so the function returns an error

```
SELECT DATEFROMPARTS ( 2015, 15, 25) AS [Date]
```

```
Cannot construct data type date, some of the
arguments have values which are not valid
```

Example : NULL specified for month parameter, so the function returns NULL

```
SELECT DATEFROMPARTS ( 2015, NULL, 25) AS [Date]
```

Date
NULL

DATEFROMPARTS function in SQL Server

```
New date and time functions - SQL Server 2012
-------------------------------------------------------------
EOMONTH (Discussed in Part 125 of SQL Server tutorial)

DATETIMEFROMPARTS : Returns DateTime
    Syntax : DATETIMEFROMPARTS (year,month,day,hour,minute,seconds,milliseconds)

SMALLDATETIMEFROMPARTS : Returns SmallDateTime
    Syntax : SMALLDATETIMEFROMPARTS (year,month,day,hour,minute)


Next Video
-------------------------------------------------------------
Difference between DateTime and SmallDateTime


Later Video
-------------------------------------------------------------
TIMEFROMPARTS
DATETIME2FROMPARTS
DATETIMEOFFSETFROMPARTS
```

[Part 127 - Difference between DateTime and SmallDateTime in SQL Server]

DateTime v/s SmallDateTime

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Attribute	SmallDateTime	DateTime
Date Range	January 1, 1900, through June 6, 2079	January 1, 1753, through December 31, 9999
Time Range	00:00:00 through 23:59:59	00:00:00 through 23:59:59.997
Accuracy	1 Minute	3.33 Milli-seconds
Size	4 Bytes	8 Bytes
Default value	1900-01-01 00:00:00	1900-01-01 00:00:00

Part 128 - DateTime2FromParts function in SQL Server 2012

DateTime2FromParts function

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

DateTime2FromParts function

- Introduced in SQL Server 2012
- Returns DateTime2
- The data type of all the parameters is integer
- If invalid argument values are specified, the function returns an error
- If any of the required arguments are NULL, the function returns null
- If the precision argument is null, the function returns an error

Syntax : `DATETIME2FROMPARTS(year,month,day,hour,minute,seconds,fractions,precision)`

Example : All the function arguments have valid values, so DATETIME2FROMPARTS returns DATETIME2 value as expected

```
SELECT DATETIME2FROMPARTS (2015, 11, 15, 20, 55, 55, 0, 0) AS [DateTime2]
```

DateTime2
2015-11-15 20:55:55

DateTime2FromParts function

Example : Invalid value specified for month parameter, so the function returns an error

```
SELECT DATETIME2FROMPARTS ( 2015, 15, 15, 20, 55, 55, 0, 0 ) AS [DateTime2]
```

```
Cannot construct data type datetime2, some of the
arguments have values which are not valid
```

Example : If any of the required arguments are NULL, the function returns null. NULL specified for month parameter, so the function returns NULL

```
SELECT DATETIME2FROMPARTS ( 2015, NULL, 15, 20, 55, 55, 0, 0 ) AS [DateTime2]
```

DateTime2
NULL

Example : If the precision argument is null, the function returns an error

```
SELECT DATETIME2FROMPARTS ( 2015, 15, 15, 20, 55, 55, 0, NULL ) AS [DateTime2]
```

```
Scale argument is not valid. Valid expressions for data
type datetime2 scale argument are integer constants and
integer constant expressions
```

DateTime2FromParts function

TIMEFROMPARTS : Returns time value

Syntax : `TIMEFROMPARTS (hour,minute,seconds,fractions,precision)`

Next video : Difference between DateTime and DateTime2

Part 129 - Difference between DateTime and DateTime2 in SQL Server

DateTime v/s DateTime2

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Differences between DateTime and DateTime2

Attribute	DateTime	DateTime2
Date Range	January 1, 1753, through December 31, 9999	January 1, 0001, through December 31, 9999
Time Range	00:00:00 through 23:59:59.997	00:00:00 through 23:59:59.9999999
Accuracy	3.33 Milli-seconds	100 nanoseconds
Size	8 Bytes	6 to 8 Bytes (Depends on the precision)
Default Value	1900-01-01 00:00:00	1900-01-01 00:00:00

DATETIME2 has a bigger date range than DATETIME. Also, DATETIME2 is more accurate than DATETIME

So I would recommend using DATETIME2 over DATETIME when possible

I think the only reason for using DATETIME over DATETIME2 is for backward compatibility

DateTime v/s DateTime2

DateTime2 Syntax: `DATETIME2 [ (fractional seconds precision) ]`

With DateTime2

- Optional fractional seconds precision can be specified
- The precision scale is from 0 to 7 digits
- The default precision is 7 digits
- For precision 0,1 and 2, storage size is 6 bytes
- For precision 3 and 4, storage size is 7 bytes
- For precision 5, 6 and 7, storage size is 8 bytes

Notice as the precision increases the storage size also increases

Precision	DateValue	StorageSize
Precision - 0	2015-10-20 15:09:12.0000000	6
Precision - 1	2015-10-20 15:09:12.1000000	6
Precision - 2	2015-10-20 15:09:12.1200000	6
Precision - 3	2015-10-20 15:09:12.1230000	7
Precision - 4	2015-10-20 15:09:12.1235000	7
Precision - 5	2015-10-20 15:09:12.1234600	8
Precision - 6	2015-10-20 15:09:12.1234570	8
Precision - 7	2015-10-20 15:09:12.1234567	8

Part 130 - Offset fetch next in SQL Server 2012

Offset fetch next in SQL Server 2012

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

SQL Server 2012 OFFSET FETCH Clause makes it very easy to implement paging

OFFSET FETCH Clause

- Introduced in SQL Server 2012
- Returns a page of results from the result set
- ORDER BY clause is required

OFFSET FETCH Syntax:

```
SELECT * FROM Table_Name
ORDER BY Column_List
OFFSET Rows_To_Skip ROWS
FETCH NEXT Rows_To_Fetch ROWS ONLY
```

Id	Name	Description	Price
1	Product - 1	Product Description - 1	10
2	Product - 2	Product Description - 2	20
3	Product - 3	Product Description - 3	30
4	Product - 4	Product Description - 4	40
5	Product - 5	Product Description - 5	50
6	Product - 6	Product Description - 6	60
7	Product - 7	Product Description - 7	70
8	Product - 8	Product Description - 8	80
9	Product - 9	Product Description - 9	90
10	Product - 10	Product Description - 10	100

Offset fetch next in SQL Server 2012

The following SQL query

- **Sorts the table data by Id column**
- **Skips the first 10 rows and**
- **Fetches the next 10 rows**

```
SELECT * FROM tblProducts
ORDER BY Id
OFFSET 10 ROWS
FETCH NEXT 10 ROWS ONLY
```

Id	Name	Description	Price
1	Product - 1	Product Description - 1	10
2	Product - 2	Product Description - 2	20
3	Product - 3	Product Description - 3	30
4	Product - 4	Product Description - 4	40
5	Product - 5	Product Description - 5	50
6	Product - 6	Product Description - 6	60
7	Product - 7	Product Description - 7	70
8	Product - 8	Product Description - 8	80
9	Product - 9	Product Description - 9	90
10	Product - 10	Product Description - 10	100

Id	Name	Description	Price
11	Product - 11	Product Description - 11	110
12	Product - 12	Product Description - 12	120
13	Product - 13	Product Description - 13	130
14	Product - 14	Product Description - 14	140
15	Product - 15	Product Description - 15	150
16	Product - 16	Product Description - 16	160
17	Product - 17	Product Description - 17	170
18	Product - 18	Product Description - 18	180
19	Product - 19	Product Description - 19	190
20	Product - 20	Product Description - 20	200

Offset fetch next in SQL Server 2012

This stored procedure accepts PAGE NUMBER and the PAGE SIZE as parameters and returns the correct set of rows

```
CREATE PROCEDURE spGetRowsByPageNumberAndSize
@PageNumber INT,
@PageSize INT
AS
BEGIN
    SELECT * FROM tblProducts
    ORDER BY Id
    OFFSET (@PageNumber - 1) * @PageSize ROWS
    FETCH NEXT @PageSize ROWS ONLY
END
```

```
EXECUTE spGetRowsByPageNumberAndSize 3, 10
```

Id	Name	Description	Price
21	Product - 21	Product Description - 21	210
22	Product - 22	Product Description - 22	220
23	Product - 23	Product Description - 23	230
24	Product - 24	Product Description - 24	240
25	Product - 25	Product Description - 25	250
26	Product - 26	Product Description - 26	260
27	Product - 27	Product Description - 27	270
28	Product - 28	Product Description - 28	280
29	Product - 29	Product Description - 29	290
30	Product - 30	Product Description - 30	300

Part 131 - Identifying object dependencies in SQL Server

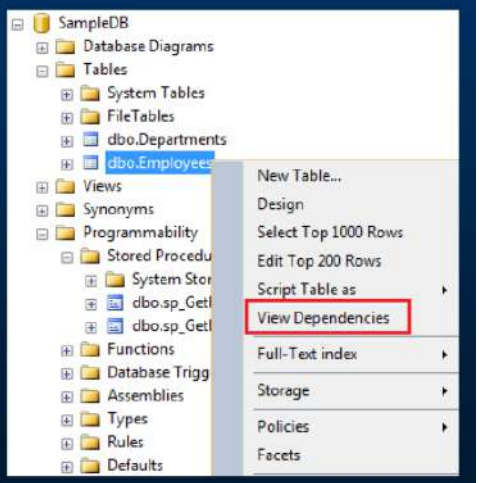
Identifying Object Dependencies

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

How to find dependencies using SQL Server Management Studio

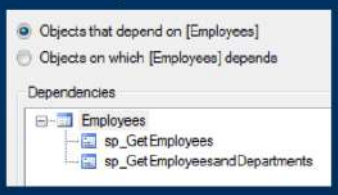
Use View Dependencies option in SQL Server Management studio to find the object dependencies

For example : To find the dependencies on the **Employees** table, right click on it and select **View Dependencies** from the context menu



Identifying Object Dependencies

In the Object Dependencies window, depending on the radio button you select, you can find the objects that depend on Employees table and the objects on which Employees table depends on



Identifying object dependencies is important especially when you intend to modify or delete an object upon which other objects depend. Otherwise you may risk breaking the functionality

Next Video : Other ways for finding object dependencies in SQL Server

Part 132 - sys.dm_sql_referencing_entities in SQL Server

sys.dm_sql_referencing_entities

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Returns all the objects that depend on Employees table

```
Select * from sys.dm_sql_referencing_entities('dbo.Employees','Object')
```

Difference between referencing entity and referenced entity

A dependency is created between two objects when one object appears by name inside a SQL statement stored in another object. The object which is appearing inside the SQL expression is known as REFERENCED ENTITY and the object which has the SQL expression is known as a REFERENCING ENTITY

Referencing entities	sys.dm_sql_referencing_entities
Referenced entities	sys.dm_sql_referenced_entities

```
Create view VwEmployees
as
Select * from Employees
```

→ REFERENCING ENTITY

→ REFERENCED ENTITY

sys.dm_sql_referencing_entities

The following query returns all the REFERENCED ENTITIES of the stored procedure sp_GetEmployeesandDepartments

```
Select * from sys.dm_sql_referenced_entities('dbo.sp_GetEmployeesandDepartments','Object')
```

Please note : For both these dynamic management functions to work we need to specify the schema name as well

Difference between Schema-bound dependency and Non-schema-bound dependency

Schema-bound dependency : Schema-bound dependency prevents referenced objects from being dropped or modified as long as the referencing object exists

Example : A view created with SCHEMABINDING, or a table created with foreign key constraint.

Non-schema-bound dependency : A non-schema-bound dependency doesn't prevent the referenced object from being dropped or modified.

Part 133 - sp_depends in SQL Server

sp_depends in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

There are several ways to find object dependencies in SQL Server

1. View Dependencies feature in SQL Server Management Studio - Part 131
2. SQL Server dynamic management functions - Part 132
    1. sys.dm_sql_referencing_entities
    2. sys.dm_sql_referenced_entities
3. sp_depends system stored procedure - This video

sp_depends

1. A system stored procedure that returns object dependencies
2. For example,
    1. If you specify a table name as the argument, then the views and procedures that depend on the specified table are displayed
    2. If you specify a view or a procedure name as the argument, then the tables and views on which the specified view or procedure depends are displayed

Syntax: `Execute sp_depends 'ObjectName'`

sp_depends in SQL Server

Returns the stored procedure that depends on table Employees

```
sp_depends 'Employees'
```

name	type
dbo.sp_GetEmployees	stored procedure

Returns the name of the table and the respective column names on which the stored procedure sp_GetEmployees depends

```
sp_depends 'sp_GetEmployees'
```

name	type	updated	selected	column
dbo.Employees	user table	no	yes	Id
dbo.Employees	user table	no	yes	Name
dbo.Employees	user table	no	yes	Gender

sp_depends does not report dependencies correctly, if the table on which the stored procedure depends is deleted and recreated.

sp_depends is on the deprecation path. This might be removed from the future versions of SQL server.

Part 134 - Sequence object in SQL Server 2012

Sequence object in SQL Server 2012

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Sequence object

- Introduced in SQL Server 2012
- Generates sequence of numeric values in an ascending or descending order

Syntax :

```
CREATE SEQUENCE [schema_name . ] sequence_name
    [ AS [ built_in_integer_type | user-defined_integer_type ] ]
    [ START WITH <constant> ]
    [ INCREMENT BY <constant> ]
    [ { MINVALUE [ <constant> ] } | { NO MINVALUE } ]
    [ { MAXVALUE [ <constant> ] } | { NO MAXVALUE } ]
    [ CYCLE | { NO CYCLE } ]
    [ { CACHE [ <constant> ] } | { NO CACHE } ]
    [ ; ]
```

Sequence object in SQL Server 2012

```
CREATE SEQUENCE [schema_name . ] sequence_name
    [ AS [ built_in_integer_type | user-defined_integer_type ] ]
    [ START WITH <constant> ]
    [ INCREMENT BY <constant> ]
    [ { MINVALUE [ <constant> ] } | { NO MINVALUE } ]
    [ { MAXVALUE [ <constant> ] } | { NO MAXVALUE } ]
    [ CYCLE | { NO CYCLE } ]
    [ { CACHE [ <constant> ] } | { NO CACHE } ]
    [ ; ]
```

Property	Description
DataType	Built-in integer type (tinyint , smallint, int, bigint, decimal etc...) or user-defined integer type. Default bigint.
START WITH	The first value returned by the sequence object
INCREMENT BY	The value to increment or decrement by. The value will be decremented if a negative value is specified.
MINVALUE	Minimum value for the sequence object
MAXVALUE	Maximum value for the sequence object
CYCLE	Specifies whether the sequence object should restart when the max value (for incrementing sequence object) or min value (for decrementing sequence object) is reached. Default is NO CYCLE, which throws an error when minimum or maximum value is exceeded.
CACHE	Cache sequence values for performance. Default value is CACHE.

Sequence object in SQL Server 2012

Creating an Incrementing Sequence

```
CREATE SEQUENCE [dbo].[SequenceObject]
AS INT
START WITH 1
INCREMENT BY 1
```

Generating the Next Sequence Value

```
SELECT NEXT VALUE FOR [dbo].[SequenceObject]
```

Retrieving the current sequence value

```
SELECT current_value FROM sys.sequences WHERE name = 'SequenceObject'
```

Reset the sequence value

```
ALTER SEQUENCE [SequenceObject] RESTART WITH 1
```

Using sequence value in an INSERT query

```
INSERT INTO Employees VALUES (NEXT VALUE for [dbo].[SequenceObject], 'Ben', 'Male')
INSERT INTO Employees VALUES (NEXT VALUE for [dbo].[SequenceObject], 'Sara', 'Female')
```

Sequence object in SQL Server 2012

Creating a decrementing Sequence :

```
CREATE SEQUENCE [dbo].[SequenceObject]
AS INT
START WITH 100
INCREMENT BY -1
```

Specifying MIN and MAX values for the sequence

```
CREATE  SEQUENCE [dbo].[SequenceObject]
        START WITH 100
        INCREMENT BY 10
        MINVALUE 100
        MAXVALUE 150
```

If you call NEXT VALUE, when the value reaches 150 (MAXVALUE)

```
The sequence object 'SequenceObject' has reached its minimum or maximum value.
Restart the sequence object to allow new values to be generated.
```

Sequence object in SQL Server 2012

Recycling Sequence values : When the sequence object has reached it's maximum value, and if you want to restart from the minimum value, set CYCLE option

```
ALTER   SEQUENCE [dbo].[SequenceObject]
        INCREMENT BY 10
        MINVALUE 100
        MAXVALUE 150
        CYCLE
```

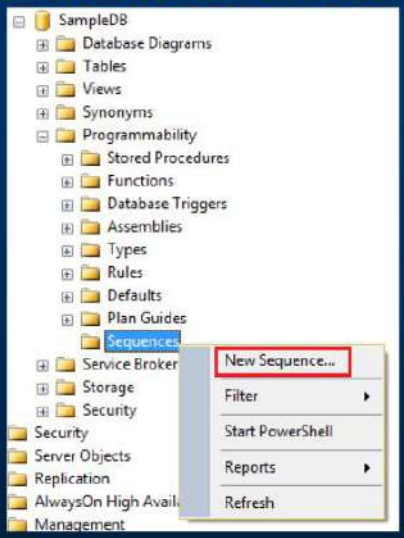
To improve performance, the Sequence object values can be cached using the CACHE option.

```
CREATE  SEQUENCE [dbo].[SequenceObject]
        START WITH 1
        INCREMENT BY 1
        CACHE 10
```

Sequence object in SQL Server 2012

Using SQL Server Graphical User Interface (GUI) to create the sequence object :

1. Expand the database folder
2. Expand Programmability folder
3. Right click on Sequences folder
4. Select New Sequence



Next video : Difference between SEQUENCE and IDENTITY in SQL Server

Part 135 - Difference between sequence and identity in SQL Server

How to check GUID is null or empty

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Use IS NULL keywords to check if a GUID is NULL

```
Declare @MyGuid Uniqueidentifier

If( @MyGuid IS NULL )
Begin
    Print 'Guid is null'
End
Else
Begin
    Print 'Guid is not null'
End
```

Difference between Sequence & Identity

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Identity	Sequence
Identity property is a table column property meaning it is tied to the table	Sequence is a user-defined database object and is not tied to any specific table meaning it's value can be shared by multiple tables.
To generate the next identity value, a row has to be insetred into the table	With sequence object there is no need to insert a row into the table to generate the next sequence value. You can use NEXT VALUE FOR clause to generate the next sequence value.
Maximum value for the identity property cannot be specified. The maximum value will be the maximum value of the correspoding column data type.	With the sequence object you can use the MAXVALUE option to specify the maximum value. If the MAXVALUE option is not specified for the sequence object, then the maximum value will be the maximum value of it's data type.
Identity property does not have any option to automatically restart the identity values	With the Sequence object CYCLE option can be used to specify whether the sequence should restart automatically when the max value (for incrementing sequence object) or min value (for decrementing sequence object) is reached.

Part 136 - Guid in SQL Server

Guid in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

What is a GUID in SQL Server

A GUID is a 16 byte binary data type that is globally unique. GUID stands for Global Unique Identifier. The terms GUID and UNIQUEIDENTIFIER are used interchangeably

```
Declare @ID UNIQUEIDENTIFIER
```

How to create a GUID

To create a GUID in SQL Server use NEWID() function

For example, SELECT NEWID() creates a GUID that is guaranteed to be unique across tables, databases, and servers.

Example GUID : 215D55D8-1683-42D8-927C-B151DBE4EE0B

Guid in SQL Server

Advantages :

- A GUID is unique across tables, databases and servers
- Useful if you're consolidating records from multiple SQL Servers into a single table

Disadvantages :

- Size is 16 bytes, where as INT is only 4 bytes
- One of the largest datatypes in SQL Server
- An Index built on a GUID is larger and slower
- Hard to read compared to INT

Summary : Only use a GUID when you really need a globally unique identifier. In all other cases it is better to use an INT data type

Part 137 - How to check GUID is null or empty in SQL Server

How to check GUID is null or empty

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Use IS NULL keywords to check if a GUID is NULL

```
Declare @MyGuid Uniqueidentifier

If( @MyGuid IS NULL )
Begin
    Print 'Guid is null'
End
Else
Begin
    Print 'Guid is not null'
End
```

How to check GUID is null or empty

If a GUID variable is NULL, then initialise it with a new GUID value, else use the existing value

```
Declare @MyGuid UniqueIdentifier

If(@MyGuid IS NULL)
Begin
    Set @MyGuid = NEWID()
End

Select @MyGuid
```

OR

```
Declare @MyGuid UniqueIdentifier

Select ISNULL(@MyGuid, NewID())
```

How to check GUID is null or empty

What is an empty GUID : An empty GUID is a GUID with all ZEROS

00000000-0000-0000-0000-000000000000

To create an EMPTY GUID

```
SELECT CAST(CAST(0 AS BINARY) AS UNIQUEIDENTIFIER)
OR
SELECT CAST(0x0 AS UNIQUEIDENTIFIER)
```

Two options to check if a GUID is an empty GUID

Option 1 :
```
If(@MyGuid = '00000000-0000-0000-0000-000000000000')
```

Option 2 :
```
If(@MyGuid = Cast(0x0 as Uniqueidentifier))
```

Part 138 - Dynamic SQL in SQL Server

Dynamic SQL in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit
http://pragimtech.com/order.aspx

What is Dynamic SQL
Dynamic SQL is a SQL built from strings at runtime

Employee Search Form

First Name	
Last Name	
Gender	
Salary	

Search Clear

```
Create Procedure spSearchEmployees
@FirstName nvarchar(100),
@LastName nvarchar(100),
@Gender nvarchar(50),
@Salary int
As
Begin

    Select * from Employees where
    (FirstName = @FirstName OR @FirstName IS NULL) AND
    (LastName  = @LastName  OR @LastName  IS NULL) AND
    (Gender    = @Gender    OR @Gender    IS NULL) AND
    (Salary    = @Salary    OR @Salary    IS NULL)
End
```

Dynamic SQL in SQL Server

```
Declare @sql nvarchar(1000)
Declare @params nvarchar(1000)

Set @sql = 'Select * from Employees where FirstName=@FirstName and LastName=@LastName'
Set @params = '@FirstName nvarchar(100), @LastName nvarchar(100)'

Execute sp_executesql @sql, @params, @FirstName='Ben',@LastName='Hoskins'
```

To execute the Dynamic SQL use system stored procedure sp_executesql. It takes two pre-defined parameters and any number of user-defined parameters

Parameter	Description
@statement	The is the first parameter which is mandatory, and contains the SQL statements to execute
@params	This is the second parameter and is optional. This is used to declare parameters specified in @statement

The rest of the parameters are the parameters that you declared in @params, and you pass them the same way as you pass parameters to a stored procedure

Part 139 - Implement search web page using ASP.NET and Stored Procedure

Implementing a search web page

```
Create Procedure spSearchEmployees
@FirstName nvarchar(100) = NULL,
@LastName nvarchar(100) = NULL,
@Gender nvarchar(50) = NULL,
@Salary int = NULL
As
Begin

    Select * from Employees where
    (FirstName = @FirstName OR @FirstName IS NULL) AND
    (LastName  = @LastName  OR @LastName  IS NULL) AND
    (Gender    = @Gender    OR @Gender    IS NULL) AND
    (Salary    = @Salary    OR @Salary    IS NULL)
End
Go
```

Part 140 - Implement search web page using ASP.NET and Dynamic SQL

Part 141 - Prevent sql injection with dynamic sql

Prevent SQL Injection with Dynamic SQL

- Dynamic SQL provides great flexibility when implementing complicated logic with lot of permutations and combinations
- Always use parameters to build dynamic sql statements, which prevents SQL Injection

```
if (inputFirstname.Value.Trim() != "")
{
    sbCommand.Append(" AND FirstName=@FirstName");
    SqlParameter param = new SqlParameter("@FirstName", inputFirstname.Value);
    cmd.Parameters.Add(param);
}
```

- Never rely on concatenating user input values to build dynamic sql statements. It open doors for SQL Injection

```
if (inputFirstname.Value.Trim() != "")
{
    sbCommand.Append(" AND FirstName = '" + inputFirstname.Value + "'");
}
```

Prevent SQL Injection with Dynamic SQL

- Stored procedures are also prone to SQL Injection, if you are building dynamic sql statements in stored procedures by concatenating user input values instead of using parameters
  Example in next video
- Using parameters to build dynamic sql statements not only prevents SQL injection, but also increases performance by reusing the cached query plans.
  Example in a later video

Part 142 - Dynamic SQL in Stored Procedure

Dynamic SQL in Stored Procedure

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Whether you are creating Dynamic SQL in a client application (like a web application) or in a stored procedure always use parameters instead of concatenating strings. Using parameters to create dynamic SQL statements prevents SQL injection

Dynamic SQL in Stored Procedure

```
-- Stored procedure with Dynamic SQL that is prone to SQL Injection
Create Procedure spSearchEmployeesBadDynamicSQL
@FirstName nvarchar(100) = NULL,
@LastName nvarchar(100) = NULL,
@Gender nvarchar(50) = NULL,
@Salary int = NULL
As
Begin

    Declare @sql nvarchar(max)

    Set @sql = 'Select * from Employees where 1 = 1'

    if(@FirstName is not null)
        Set @sql = @sql + ' and FirstName=''' + @FirstName + ''''
    if(@LastName is not null)
        Set @sql = @sql + ' and LastName=''' + @LastName + ''''
    if(@Gender is not null)
        Set @sql = @sql + ' and Gender=''' + @Gender + ''''
    if(@Salary is not null)
        Set @sql = @sql + ' and Salary=''' + @Salary + ''''

    Execute sp_executesql @sql
End
Go
```

Dynamic SQL in Stored Procedure

```sql
-- Stored procedure with Dynamic SQL that prevents SQL Injection
Create Procedure spSearchEmployeesGoodDynamicSQL
@FirstName nvarchar(100) = NULL,
@LastName nvarchar(100) = NULL,
@Gender nvarchar(50) = NULL,
@Salary int = NULL
As
Begin

    Declare @sql nvarchar(max)
    Declare @sqlParams nvarchar(max)

    Set @sql = 'Select * from Employees where 1 = 1'

    if(@FirstName is not null)
        Set @sql = @sql + ' and FirstName=@FN'
    if(@LastName is not null)
        Set @sql = @sql + ' and LastName=@LN'
    if(@Gender is not null)
        Set @sql = @sql + ' and Gender=@Gen'
    if(@Salary is not null)
        Set @sql = @sql + ' and Salary=@Sal'

    Execute sp_executesql @sql,
    N'@FN nvarchar(50), @LN nvarchar(50), @Gen nvarchar(50), @sal int',
    @FN=@FirstName, @LN=@LastName, @Gen=@Gender, @Sal=@Salary
End
Go
```

Part 143 - Sql server query plan cache

Sql server query plan cache

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

What happens when a query is issued to SQL Server

In SQL Server, every query requires a query plan before it is executed. When you run a query the first time, the query gets compiled and a query plan is generated. This query plan is then saved in sql server query plan cache.

How to check what is in SQL Server plan cache

- sys.dm_exec_cached_plans
- sys.dm_exec_sql_text
- sys.dm_exec_query_plan

CROSS APPLY discussed in Part 91 of SQL Server Tutorial

```
SELECT cp.usecounts, cp.cacheobjtype, cp.objtype, st.text, qp.query_plan
FROM sys.dm_exec_cached_plans AS cp
CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
ORDER BY cp.usecounts DESC
```

To remove all elements from the plan cache

```
DBCC FREEPROCCACHE
```

Sql server query plan cache

- In older versions of SQL Server up to 6.5 only stored procedure plans are cached
- With SQL Server 7, and later versions the query plans for Ad hoc sql statements and dynamic sql statements are also cached
- The query plan cache lookup is by a hash value computed from the query text. If the query text changes even slightly, sql server will not be able to reuse the existing plan
- With parameterised queries, sql server will not treat parameter values as part of the query text. So when you change the parameters values, sql server can still reuse the cached query plan

```
Declare @FirstName nvarchar(50)
Set @FirstName = 'Steve'
Execute sp_executesql N'Select * from Employees where FirstName=@FN',
                      N'@FN nvarchar(50)', @FirstName
```

- **Summary :** You should never ever concatenate user input values with strings to build dynamic sql statements. You should always use parameterised queries which not only promotes cached query plans reuse but also prevent sql injection attacks.

Part 144 - exec vs sp_executesql in sql server

Exec vs SP_ExecuteSQL in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Two options to execute Dynamic SQL

- Exec/Execute
- sp_executesql (Discussed in Part 138 of SQL Server tutorial)

Many articles on the web says using exec over sp_executesql will have 2 problems

- It open doors for sql injection attacks (QUOTENAME function can prevent this)
- Cached query plans may not be reused and leads to poor performance (With auto-parameterisation capability this may not be an issue)

What is exec() in SQL Server

Exec() or Execute() function is used to execute dynamic sql and has only one parameter i.e the dynamic sql statement you want to execute.

Exec vs SP_ExecuteSQL in SQL Server

Summary

- If you use QUOTENAME() function, you can prevent sql injection while using Exec()
- Cached query plan reusability is also not an issue while using Exec(), as SQL server automatically parameterizes queries
- It is better to use sp_executesql over exec() as we can explicitly parameterise queries instead of relying on sql server auto-parameterisation feature or QUOTENAME() function. Use Exec() only in throw away scripts rather than in production code

Part 145 - Dynamic sql table name variable

Dynamic SQL table name variable

```
Create procedure spDynamicTableName
@TableName nvarchar(100)
As
Begin
    Declare @sql nvarchar(max)
    Set @sql = 'Select * from ' + @TableName
    Execute sp_executesql @sql
End
```

```
Create procedure spDynamicTableName
@TableName nvarchar(100)
As
Begin
    Declare @sql nvarchar(max)
    Set @sql = 'Select * from ' + QUOTENAME(@TableName)
    Execute sp_executesql @sql
End
```

Part 146 - Quotename function in SQL Server

Quotename function in SQL Server

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

QUOTENAME() function

- Takes two parameters - the first is a string, and the second is a delimiter that you want SQL server to use to wrap the string in
- The delimiter can be a left or right bracket ([]), a single quotation mark ('), or a double quotation mark (")
- The default for the second parameter is []

```
Declare @sql nvarchar(max)
Declare @tableName nvarchar(50)
Set @tableName = 'USA Customers'
Set @sql = 'Select * from ' + QUOTENAME(@tableName)
Execute sp_executesql @sql
```

Part 147 - Dynamic SQL vs Stored Procedure

Dynamic SQL vs Stored Procedure

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

Advantages and disadvantages of Dynamic SQL and Stored Procedures

- Separating database logic from business logic
- Network traffic
- SQL Injection Attacks
- Cached query plans reuse
- Maintenance
- Implementing flexible logic

Part 148 - Dynamic sql output parameter

Dynamic SQL Output Parameter

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

```
--Dynamic SQL with INPUT & OUTPUT Parameters
Declare @sql nvarchar(max)
Declare @gender nvarchar(10)
Declare @count int
Set @gender = 'Male'
Set @sql = 'Select @count = Count(*) from Employees where Gender=@gender'

Execute sp_executesql @sql,
        N'@gender nvarchar(10), @count int OUTPUT',
        @gender, @count OUTPUT

Select @count
```

Please note : The OUTPUT parameter will not work as expected, if you forget to use OUTPUT keyword

Part 149 - Temp tables in dynamic sql

Temp Tables in Dynamic SQL

If you are in need of the DVD with all the videos and PPT's, please visit http://pragimtech.com/order.aspx

```
Create procedure spTempTableInDynamicSQL
as
Begin
    Declare @sql nvarchar(max)
    Set @sql = 'Create Table #Test(Id int)
                insert into #Test values (101)
                Select * from #Test'
    Execute sp_executesql @sql
End
```

- Creating, Inserting & Selecting from Temp Table all happens in the dynamic SQL code block, so we are able to access the temp table without any issue

Temp Tables in Dynamic SQL

```
Alter procedure spTempTableInDynamicSQL
as
Begin
    Declare @sql nvarchar(max)
    Set @sql = 'Create Table #Test(Id int)
                insert into #Test values (101)'
    Execute sp_executesql @sql
    Select * from #Test
End
```

- Creating and Inserting into Temp Table happen in the dynamic SQL code block, but selecting data is outside of the dynamic sql code block, so we get the error - Invalid object name '#Test'
- This proves the point temp tables created by dynamic SQL are not accessible from the calling procedure. They are dropped when the dynamic SQL block in the stored procedure completes execution

Temp Tables in Dynamic SQL

```sql
Alter procedure spTempTableInDynamicSQL
as
Begin
    Create Table #Test(Id int)
    insert into #Test values (101)
    Declare @sql nvarchar(max)
    Set @sql = 'Select * from #Test'
    Execute sp_executesql @sql
End
```

- The calling procedure is Creating and Inserting into Temp Table, and the dynamic sql block is selecting data
- So in this case we are able to access temp table, which proves that that dynamic SQL block can access temp tables created by the calling stored procedure

Temp Tables in Dynamic SQL

Summary

- Temp tables created by dynamic SQL are not accessible from the calling procedure
- They are dropped when the dynamic SQL block in the stored procedure completes execution
- On the other hand, dynamic SQL block can access temp tables created by the calling stored procedure