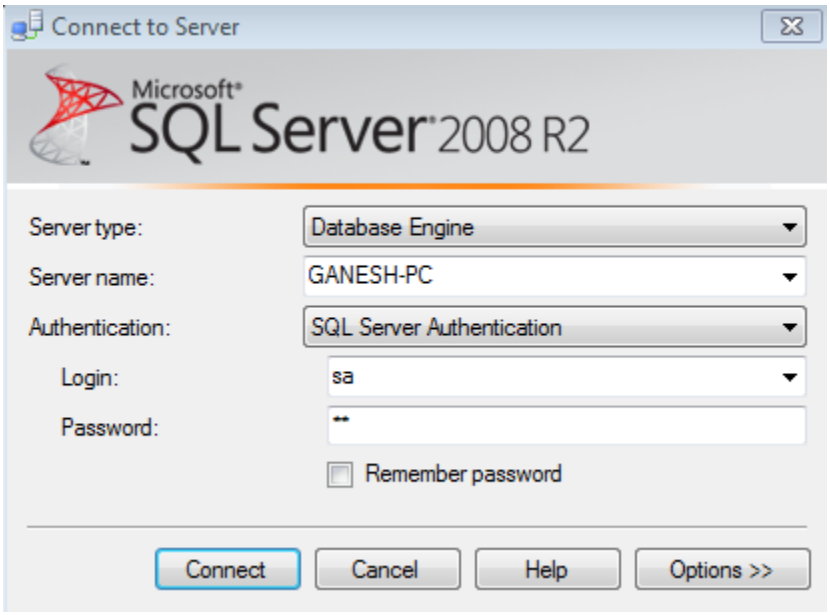
SQL Server:-

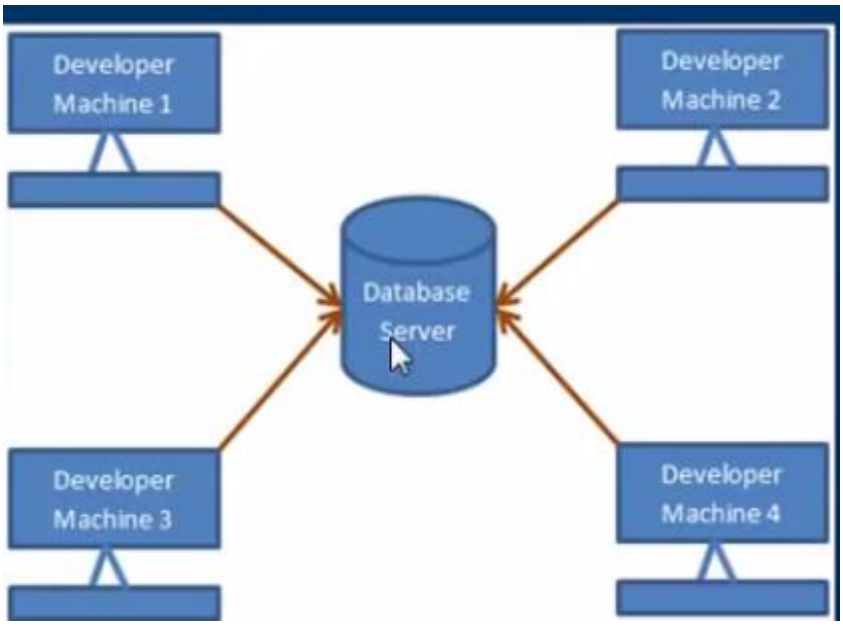
1. Connecting to Sql server:-

Server type=Database Engine
Server Name=Ganesh-pc
Authentication= Windows or Sql Server
In Sql server Authentication
Login
Password



SSMS:-

SSMS is a client tools and not the server by itself



2. Create alter and Drop Database

A sql Server database can be crated, altered and dropped

1. Graphically using Sql Server Management Studio (SSMS) or
2. Using Query

To create a database using Query

Syntax:-

Crate Database Database_Name

Whatever we crate database graphically using the designer or , using a query the following two files gets generated

.Mdf file—Data file (Contains actual data)

. Ldf file—Transaction log file(Use to recover the database).

To alter database once I created

Syntax:- *alter database databaseName Modify Name=NewDatabaseName*

Alternatively, you can also use system store procedure

Execute sp_rename 'OldDatabaseName','NewDatabaseName'

Deleting or Dropping a Database

To delete or Drop a database

Drop Database DatabaseName

Droping a database, deletes the LDF and MDF file

We can not drop a database, if it is currently in use. You get an error stating can not drop database “New Database Name” because it is currently in use. So, if other use are connected, you need to put the database in single use mode and 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 database cannot be dropped.

```
--creating a Database----
create database SQL
----------------------------
--Rename a Database--------------
alter Database SQL1  modify name=SQL
--OR
sp_renameDB 'SQL1','SQL'
---------------------------------------
--To Delete database------------------
drop database SQL
-------------------------------------
```

3. Creating and working with table

The main aim of this lesion is to crate tblPerson and tblGender tables and establish primary key and foreign key constraints. In Sql server, tables can be created graphically using sql Server Management Studio (SSMS) or using query .Foreign key references can be added graphically using SSMS or using a query. Foreign key references can be added graphically using SSMS or using query.

Alter table ForeignKeyTable add constraint ForeignKeyTable_ForeignKeyColumn_FK FOREIGN KEY (FOREIGNKEYCOLUMN) references PrimaryKeyTable(primaryKeyColumn)

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

```
use SQl
create table tblGender
(
ID int Not Null primary key,
Gender varchar(50) not null
)


create table tblPerson
(
ID int Not null primary key,
Name varchar(100) not null,
Email varchar(100),
```

```
Gender int not null
)

alter table tblPerson
add constraint tblPeron_Gender_FK Foreign Key (Gender) references tblGender(ID
```

4 . Defalut Constraint

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

```
Syntax
alter table [TableName]
add constraint [ConstraintName] Default [Value] for [Column Name]
Examples:-
alter table tblPerson
add constraint ConstraintEmail Default 'Ganesh@gmail.com' for Email alter
```

adding a new column,with default value, to an existing table

syntax:-

```
alter table [TableName]
add [ColumnName][ dataType] not null||Null
constraint [ConstraintName] Default [Value]
Example:-
alter table tblPerson
add Status bit not null
constraint default_Status Default 1
```

Droping Constraint

Syntax:-

```
alter table [TableName]
Drop Constraint [ConstraintName]
Examples:-
alter table tblPerson
Drop Constraint default_Status
```

5. Cascading referential integrity constraint

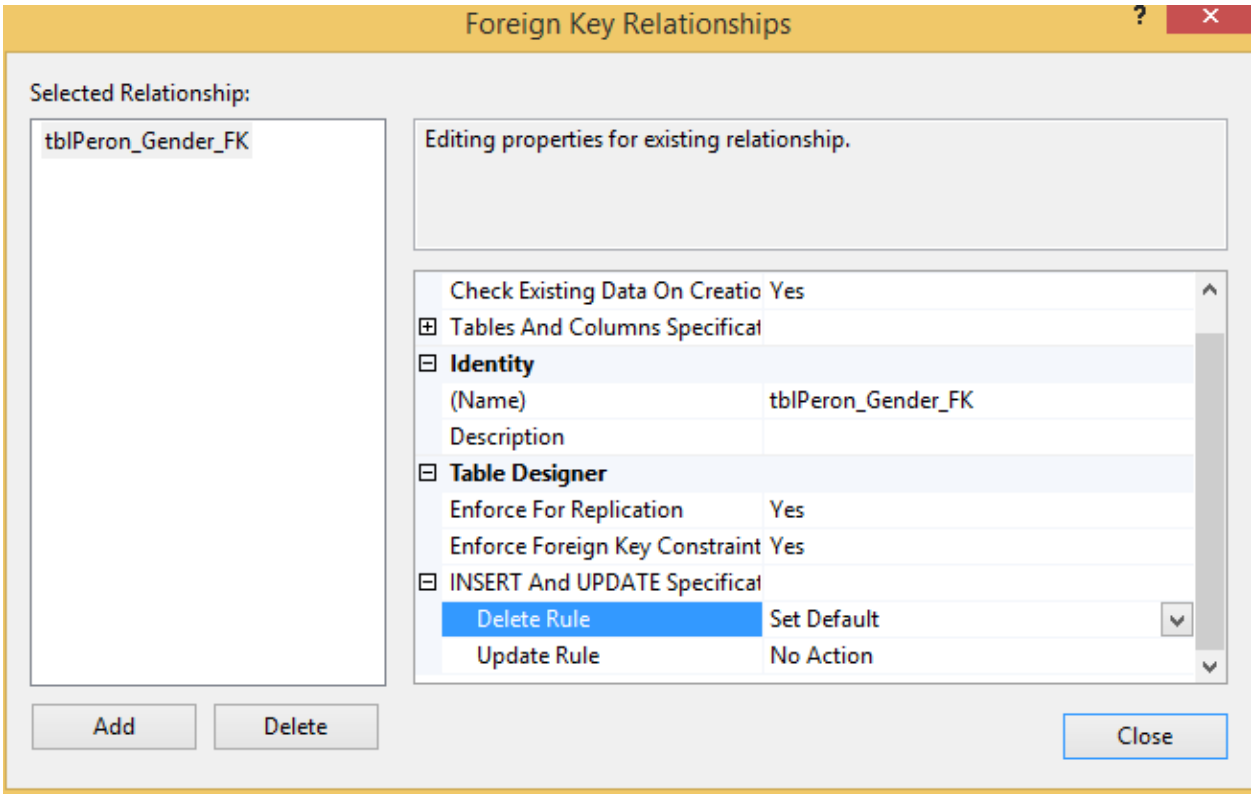
Cascading referential integrity constraint allows to define the action Microsoft SQL server should take when a user attempts to delete or update a key to which an existing foreign key points

For example, if you delete row with ID=1 from tblGender table, then row with ID=3 from tblperson become 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.

Options when setting up cascading referential integrity constraint:

1. No Action:-This is default behavior. 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 update a row with a key referenced by foreign keys in existing rows in other tables, all rows contain 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 references 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 references by foreign keys in existing rows in other tables, all rows containing those foreign keys are said to default values.



6.Adding and check constraint

Check constraint is use to limit the range of the values, that can be entered for a column

To find the infromation about the column inside the table we need to select the table name in query and then press ALT+F1 then we will get the infromation about the table.

the general formula for addinng check constraint in SQL server:-

syntax:-

```
alter table [tableName]
      add constraint [ConstraintName] check  ( boolean expresssion)
```

examples:-

```
alter table tblPerson
      add constraint CK_Age check  ( age <20AND age>0)
```

IF the boolean expression returns true, then the CHECK constraint allows the vlaues, otherwise it doesn't.Since, AGE is nullbale column, It's possible to pass nul for this column, when inserting a row. When you pass null for the age column, the bollean expression evalueates UNKNOWN, and allows the vlaue

To dop the check constraint
Syntax:-

```
alter table [tableName]
drop constraint [ConstraintName]
examples:-
alter table tblPerson
drop constraint CK_Age
```

7.Identity column in SQl Server

Check Constraint:-

If the column marked as identity column,then the vlaue from the column are automatically generated,when you inserted new row into the table.

```
create table tblPerson1
(
personId int identity(1,1) primary key,
Namve varchar(50),
)
```

Note:-Seed and increment value are optional, if you do not specify the identity and seed they both default to 1

To explicitely supply the value for identity column

1.First turn on the idenity insert examples – `Set identity_Insert tblPerson1 ON`

2.In the insert query specify the column list

```
insert into  tblPerson1(PersonId,Namve)values(1,'Ganesh')
```

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

```
DBCC CHECKIDENT('tblPerson1',Reseed,0)
```

8. How to get last generated identity column value in Sql Server.

From the previous session we understand that identiy column values are auto generated.There are several ways in sql server, to retieve the last identity value that is generated.The most common way 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.

```
insert into Test2 values ('xxxx')

select SCOPE_IDENTITY()
Select @@IDENTITY
Select IDENT_CURRENT('test2')
```

9. Unique Key Constraint

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

To create the unique key using query:-

Syntax:-

```
alter table [tableName]
Add Constraint [ConstraintName] Unique(columnName)
```

Examples:-

```
alter table tblPerson
Add Constraint UQ_tblPerson_Email Unique(Email)
```

Both primary key and unique key are use 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 uniquekey constraint.

What is the difference betweeen the primary key constraint and Unique Key constraint?

1.A table can have only one primary key, but more then one unique key

2. primary key donesnot allow null, where as unique key allows one null.

10.Select Statement in Sql Server.

Operators and WildCards:-

Operator	Purpose
=	Equal to
!=or <>	Not equal to
>	Greater then
>=	Greater then or equal to
<	Less then
<=	Less then or Equal to
IN	Specify list of values
Between	Specify range
Like	Specify Patterns
NOT	Not in a list,range etc….
%	Specifies zero and more characters
-	Specifies exectaly one charater
[]	Any character with the brackets
[^]	Not any character with the brackets.

```
select*From tblperson--General conditions
select Distinct Status from tblperson---distinct statement

select *from tblPerson where Gender=2---Where condition in Select statement
select*from tblperson where Gender<>2-- Not Equal to Operator
select*from tblPerson where Age IN( 24,26,27)---IN operator
select*from tblPerson where Age Between 20 and 25---Between Operator
Select*from tblPerson where Name LIKE 'L%'--LiKE operator
select*from tblPerson where Email  Like '_@_.com'
select*from tblPerson where Name Like '[MST]%'--Search with Single character
select*from tblPerson where (Name='John' or Name='Mary') And Age>--multiple and or Or
Conditions
select*from tblPerson Order by Name--order by condition
select top 2 *from tblPerson--Top  Condition
select top 50 Percent *From tblPerson--top condition in percentage.
```

11.Group by In Sql Server

Group by clause is use to group a selected set of rows into a set of summary rows by the value of one or more column or expressions.It always use in conjunction with one or more aggrigate functions.

```
select SUM(salary) from tblEmployee---gives the total sum of the salary column
select MIN(salary) from tblEmployee--Gives the minimum salalry of the salary column
select MAX(salary) from tblEmployee--gives the maximum salary of the salary column
Select City,SUM(salary)  as TotalSalary from tblEmployee group by City--get the total
salary paid by the city

select City,Gender,SUM(salary) as TotalSalary from tblEmployee group by City,Gender--
Group base on multiple column

select Gender,city,SUM(salary) as TotalSalary,COUNT(Name) as [Total Employee] from
tblEmployee group by Gender,City
```

Note:-If you omit, the Goup by clause and try to execute the query you will 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:-

Where clause is used to filter rows before aggregation where as having clause is used to filter group after aggregations. The following 2 query produce the same result.

```
select Gender,City,SUM(salary) as TotalSalary,COUNT(Id) as TotalEmployees from
tblEmployee where Gender='Male' Group by Gender,City—or

select Gender,City,SUM(salary) as TotalSalary,COUNT(Id) as TotalEmployees from
tblEmployee  Group by Gender,City Having gender='Male'
```

Note:-from the performance standpoint, you can not say that one method is less efficient then the other, Sql server optimizer analyses each statement and select an efficient way of executing it As Best practice, use that correctly 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 use with the Select statement.
2. WHERE filter rows before aggregations (GROUPING), whereas, HAVING filter groups, after aggregations and performed.
3. Aggregate function 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.

```
select Gender,City,SUM(salary) as TotalSalary,COUNT(Id) as TotalEmployees from
tblEmployee where Gender='Male' Group by Gender,City Having SUM(salary)>5000
```

12. Join in SQL server

Join in sql server are user to retrieve data form 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 JOIN

1. Inner join
2. Outer Join
3. Cross Join

Outer join are again divided into

1. left join of left outer join
2. Right join or Right outer join
3. Full join or Full outer join

1. INNER JOIN

   Returns only the matching rows between both the tables. Non matching rows are eliminated.

```
select Name,Gender,Salary,DepartmentName from tblEmployee
join tblDepartment
on tblEmployee.DepartmentId=tblDepartment.Id

select Name,Gender,Salary,DepartmentName from tblEmployee
 Inner join tblDepartment
on tblEmployee.DepartmentId=tblDepartment.Id
```

2. Left JOIN

   Returns all the matching rows +non matching rows form the left table.

```
select Name,Gender,Salary,DepartmentName from tblEmployee
 Left join tblDepartment
on tblEmployee.DepartmentId=tblDepartment.Id
or
select Name,Gender,Salary,DepartmentName from tblEmployee
 Left outer join tblDepartment
on tblEmployee.DepartmentId=tblDepartment.Id
```

3. Right JOIN

   Returns all the matching rows + non matching rows from the right table

```
select Name,Gender,Salary,DepartmentName from tblEmployee
 Right join tblDepartment
on tblEmployee.DepartmentId=tblDepartment.Id
or
select Name,Gender,Salary,DepartmentName from tblEmployee
 Right outer join tblDepartment
on tblEmployee.DepartmentId=tblDepartment.Id
```

4. Full Join

   Returns all rows form the both the left and right tables, including the non-matching rows

Cross Join:-

Cross join, Produces the Cartesian product of the 2 tables involved in the join. For example, in the Employees table we have 11 rows and in Department table we have 4 rows so, a cross join between these 2 tables produces 40 rows.

Note:-cross join Should not have ON clause.

General formula for Joins:-

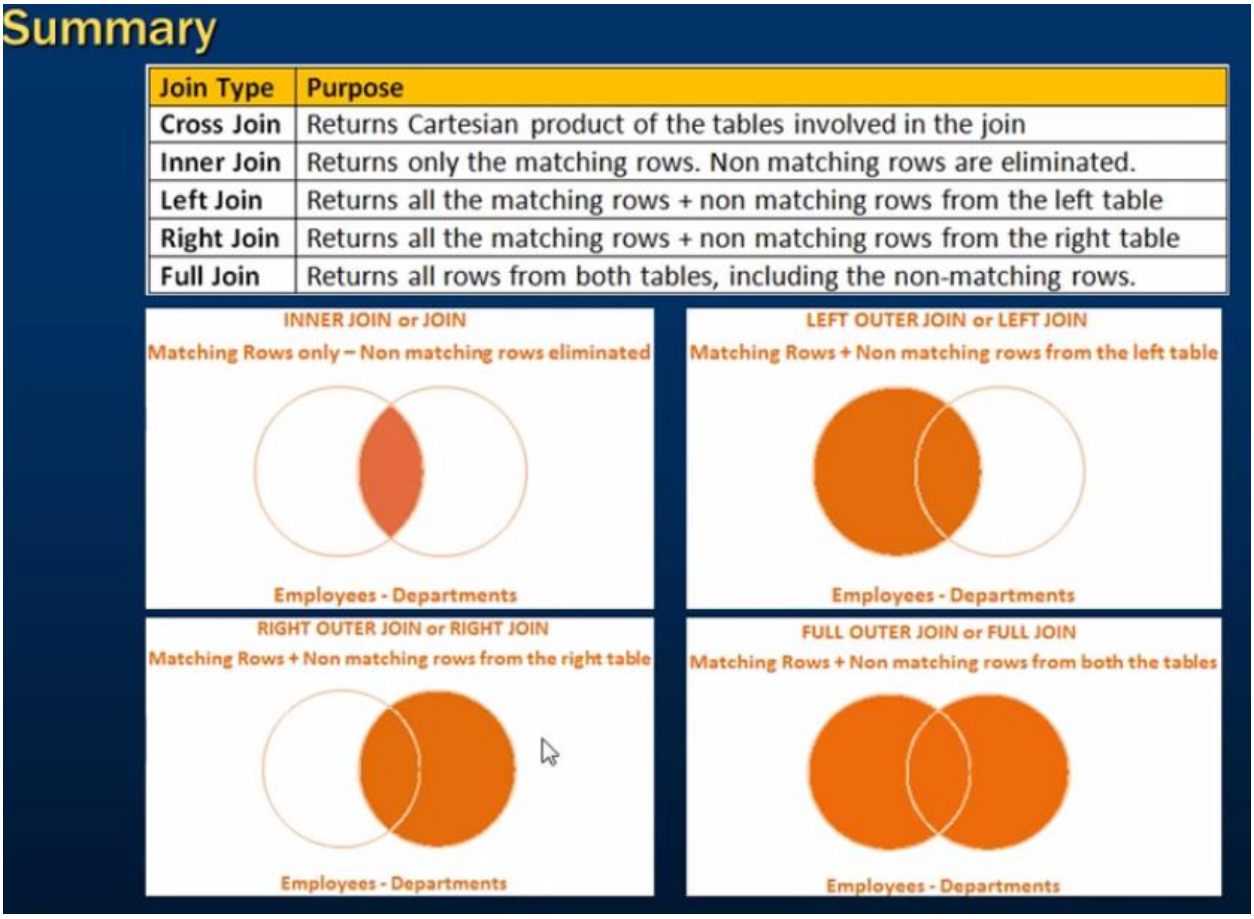
Select *column List*

From *left table Name*

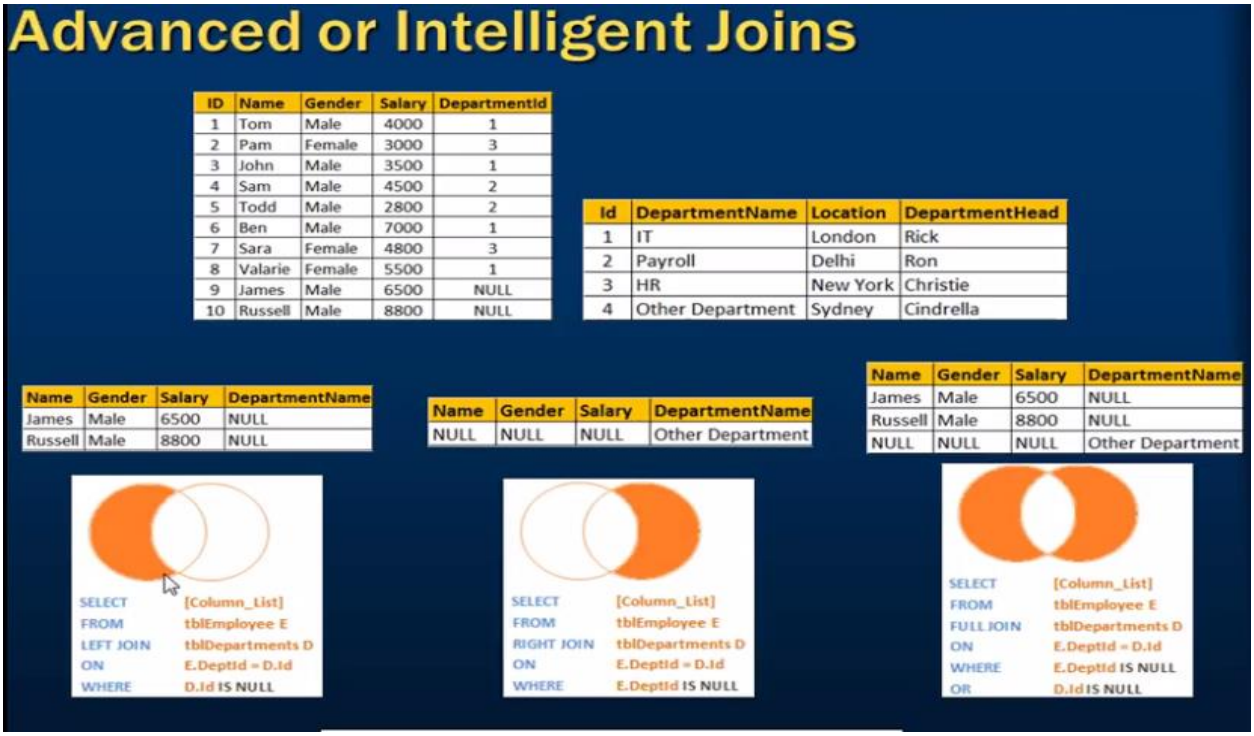
Join Type *Right table name*

On *Join condition*

```
select Name,Gender,Salary,DepartmentName from tblEmployee
 cross join tblDepartment
```



13. Advanced or Intilligent JOIN




```
select Name,Gender,Salary,DepartmentName from tblEmployee
left join tblDepartment
on tblEmployee.DepartmentId=tblDepartment.Id
where tblEmployee.DepartmentId is null
--or
select Name,Gender,Salary,DepartmentName from tblEmployee
left join tblDepartment
on tblEmployee.DepartmentId=tblDepartment.Id
where tblDepartment.Id is null

select Name,Gender,Salary,DepartmentName from tblEmployee
Right join tblDepartment
on tblEmployee.DepartmentId=tblDepartment.Id
where tblEmployee.DepartmentId is null
--or
select Name,Gender,Salary,DepartmentName from tblEmployee
Right join tblDepartment
on tblEmployee.DepartmentId=tblDepartment.Id
where tblDepartment.Id is null

select Name,Gender,Salary,DepartmentName from tblEmployee
Full join tblDepartment
on tblEmployee.DepartmentId=tblDepartment.Id
where tblDepartment.Id is null
or tblEmployee.DepartmentId is null
```

14. Self-Join In Sql Server

Joining the table with itself is called as self join. Self Join is not a different type of the join. It can be classified under any type of join.

1. Inner
2. Outer(left, Right, Full)
3. Cross

```
Select e.Name,m.Name  from tblEmployee e
inner join tblEmployee M
on e.ManagerId=m.id

Select e.Name,m.Name  from tblEmployee e
cross join tblEmployee M

Select e.Name,m.Name  from tblEmployee e
left join tblEmployee M
on e.ManagerId=m.id
```

15. Different ways to replace null in sql server

There are three ways to replace NULL values-ISNULL () function, CASE statement & COALESCE () function.

```
Select ISNULL(NULL,'No Manager') as Manager--Is null function
Select CoalEscE(NULL,'No Manager') as Manager--CoalEscE

--ISNULL FUNCTION--
Select E.Name as Employee,ISNULL(m.Name,'No Manager') as Manager
from tblEmployee E
left join tblEmployee M
on E.ManagerId=m.Id
--COALESCE function
Select CoalEscE(NULL,'No Manager') as Manager--CoalEscE
Select E.Name as Employee,CoalEsce(m.Name,'No Manager') as Manager
from tblEmployee E
left join tblEmployee M
on E.ManagerId=m.Id
--Case  Statement
--Case Wheren Expression THen '' Else '' End
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.Id
```

16. COALESCE () function in SQL Server

COALESCE() function returns the first not null value.

```
Select ID,CoalEscE(FirstNamve,MiddleName,LastNamver) as Name
from employee
```

17. Union and Union All

```
select Id,Name,Email from tblIndiaCustomers
Union
select Id,Name,Email from tblUkCustomer
--UnionAll
select Id,Name,Email from tblIndiaCustomers
Union All
select Id,Name,Email from tblUkCustomer
```

Note:-For Union and Union All to work, the number, data types, and the other of the columns in the select statements should be same.

Difference between Union and Union All

1. UNION remove duplicate rows, whereas UNION ALL does not

2. UNION has to perform distinct sort to remove duplicates, which makes it less faster then 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 Jon

UNION combines the result –set of two or more selected queries into a single result set which include all the rows from all the queries in the union, where as JOINS, retrieves data from two or more tables based on logical relationship between the tables.

In short, UNION combines rows from 2 or more tables, where JOIN combine column from 2 or more tables.

```
select Id,Name,Email from tblIndiaCustomers
Union
select Id,Name,Email from tblUkCustomer
--UnionAll
select Id,Name,Email from tblIndiaCustomers
Union All
select Id,Name,Email from tblUkCustomer
```

18. Store Procedure

A store 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 store procedure and call it just by its name.

1. Use CREATE PROCEDURE or CREATE PROC statements 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 store procedure

1. SpGetEmployees
2. Exec spGetEmployees
3. Execute SpgetEmployees.

Note:-you can also right click on the procedures name, in object explorer in sql server Management studio and select EXECUTE STORE PROCEDURE.
Parameters and variables have an @ prefix in their name.
To Execute:-
EXECUTE spGetEmployeesGetBYGenderAndDepartment 'Male',1
EXECUTE spGetEmployeesByGenderAndDepartment@DepartmentId=1,@Gender='Male'
To view the text, of the store procedure
1.Use system store procedure sp_helptext'spName'
Or
2.Right Click the SP in object explorer->Script Procedure as->Create to->New query Editor window
To change the store 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.

```
alter proc spGetEmployeeByGenderAndDepartment
@Gender nvarchar(20),
@DepartmentId int
with encryption
as
Begin
Select Name,Gender,DepartmentId from tblEmployee where gender=@Gender and
DepartmentId=@DepartmentId
End
--to see the text inside the store procedures
sp_helptext SpGetEmployees
--To encrypt store procedure
alter proc spGetEmployeeByGenderAndDepartment
@Gender nvarchar(20),
@DepartmentId int
with encryption
as
Begin
Select Name,Gender,DepartmentId from tblEmployee where gender=@Gender and
DepartmentId=@DepartmentId
End
```

19.Store Procedure with Output Parameters

```
create procedure spGetEmployeeCoutbyGender
@Gender nvarchar(20),
@EmployeeCount int output
as
Begin
select @EmployeeCount=COUNT(Id) from tblEmployee where Gender=@Gender
End
--To Execute the stored procedur with output parameters
Declare @TotalCount int
Execute spGetEmployeeCoutbyGender 'Male' ,@TotalCount out
print @TotalCount
--Or we can use this conditions as well
Declare @TotalCount int
Execute spGetEmployeeCoutbyGender 'Male' ,@TotalCount out
If(@TotalCount is null)
      print '@totalCount is null'
Else
```

```
      print '@TotalCount is not null'
```

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

Useful system storeprocedure

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 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 about to drop. Sp_depends can also be used with other database object like table etc.

```
--made new procedure with output parameters
create procedure spGetEmployeeCoutbyGender
@Gender nvarchar(20),
@EmployeeCount int output
as
Begin
select @EmployeeCount=COUNT(Id) from tblEmployee where Gender=@Gender
End
--To Execute the stored procedur with output parameters
Declare @TotalCount int
Execute spGetEmployeeCoutbyGender 'Male' ,@TotalCount out
print @TotalCount
--Or we can use this conditions as well
Declare @TotalCount int
Execute spGetEmployeeCoutbyGender 'Male' ,@TotalCount out
If(@TotalCount is null)
      print '@totalCount is null'
Else
      print '@TotalCount is not null'
--system store procedure
--to see help store procedure
sp_help spGetEmployeeCoutbyGender
sp_help tblEmployee
--to see the text of the stored procedure\
sp_helptext spGetEmployeeCoutbyGender
--TOE SEE THE DEPENDENICIES
sp_depends spGetEmployeeCoutbyGender
```

20-Stored procedure output parameters or return values

Whenever,you execute astored procedure, it return an integer status variable.Usually, zero indicates success, and non-zero indicates failture

```
create procedure spGettotalCOuntOfEmployees1
@TotalCount int output
```

```
as
Begin
Select @TotalCount=Count(id) from tblEmployee
End

--Execute store procedure with output parameters
Declare @total int
Execute spGettotalCOuntOfEmployees1 @Total OUt
print @Total
--create procedure with Returns values
create proc spGetTotalEmployee@
as
Begin
       return (select COUNT(Id) from tblEmployee)
End
--execute store procedure with retruns values
Declare @Total int
Execute @Total=spGetTotalEmployee@
print @total
```

so , we are able to achieve what we want, using output parameters as well as return values. Now lets look at example, where returns status variables cannot be used, But output parameter can be used.

```
create procedure spGetNameById1
@Id int,
@Name nvarchar(20) output
as
Begin
select @Name=Name from tblEmployee where ID=@Id
End

--execute sp
Declare @Name nvarchar(20)
Execute spGetNameById1 1,@Name Out
print 'Name='+@Name

create proc spGetNamebyId2
@id int
as
begin
return(select Name from tbleEmployee where ID=@id)
End

---execute store procedure
Declare @Name nvarcharint
Execute @Name=spGetNamebyId2 1
print 'Name='+@Name
```

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

Returns Values	Output Parameters
Only Returns DataType	Andy DataType
Only one values	More then value
Use to convey success or Failture	use to Return values like name,count etc..

21.Advantaget of StoreProcedure

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 attact

22.Stirng Functions

Function	Purpose
ASCII(Character_Expression)	Returns the ASCII code of the given Character Expression
CHAR(Integer_Expression)	Conver an int ASCII code to a character. The Integer Expression,should be between 0 to 255
LTRIM(Character _Expression)	Removes blank on the left handside of the given character expression.
RTRIM(Character_Expression)	Removes blank on the right hand side of the given character expression.
LOWER(Character_Expression)	Convert all the character in the given character_Expression
UPPER(Character Expression)	Convert all the character in the given character _Expression to Uppercase letters.
REVERSE('Any_String_Expression')	Reverse all the characters in the given string expression,
LEN(String_Expression)	Returns the count of toal characters, in the give en string expression, excluding the blanks at th the end of the expression.

```
--ASCII function
select ASCII('A')
select ASCII('BC')
-----
select CHAR(65)
------------------print A to X
declare @Start int
Set @Start=65
while(@Start<=90)
Begin
print CHAR(@Start)
set @Start=@Start+1
End
--print small alphabets a to z----
declare @Start int
Set @Start=97
while(@Start<=122)
Begin
print CHAR(@Start)
set @Start=@Start+1
End
```

```
--print the value form 0 to 9----------------
declare @Start int
Set @Start=48
while(@Start<=57)
Begin
print CHAR(@Start)
set @Start=@Start+1
End
--Ltrim Function
Select  LTrim('   Hello')
Select LTRIM(Name) from tblEmployee
--Rtirm function
select Ltrim(RTRIM(Name)) from tblEmployee

--Lower function\
select LOWER(Name) from tblEmployee
--UPPEr function
Select UPPER(Name) from tblEmployee
--ReverseFunction
Select REVERSE(Name) from tblEmployee
--Lenght FUnction
Select LEN(Name) as TotalCharacter from tblEmployee
```

23.Left,Right CharIndex and SUBSTRING function

Function	Purpose
LEFT(Character_Expression,int_exp)	Returns the specified number of characters from the left hand side of the given character expression
RIGHT(Character_Exp,Int_Exp)	Returns the specified number of characters from the right hand side fo the give character expression.
CHARINDEX(“Expression_To_Find’, Expression_to_Search’,’Start_Location’)	Returns the starting postion of the specified expression in a character string.
SUBSTRING(‘Expression’,’Start’,’Length’)	Returns substring (Part of the string), from the given expression.

```
--Left function
'ABCDEF'
select LEFT('ABCDEF',3)
--Right Function
select RIGHT('ABCDEF',3)
--CHARINDEX function
select CHARINDEX('@','sara@aaa.com')
--Substring function---------------
Select SUBSTRING('sara@aaa.com',6,7)
Select SUBSTRING('sara@aaa.com',CHARINDEX('@','sara@aaa.com'),7)
Select
SUBSTRING('sara@aaa.com',CHARINDEX('@','sara@aaa.com')+1,len('sara@aaa
.com')-CHARINDEX('@','sara@aaa.com'))
select*from tblEmployee
```

```
select*from tblPerson
select  SUBSTRING(Email,CharINdex('@',Email)+1,LEN(Email)-
Charindex('@',Email)) as EmailDomain,Count(Email) as Total
from tblPerson
Group by SUBSTRING(email,CharIndex('@',Email)+1,
Len(Email)-CHARINDEX('@',Email))
```

24. Replicate Space, PatIndex, Replace and Stuff String Function

Replicate Function

REPLICATE(String to be Replicated,No.Of Times to be Replicate) Repeates a Given string ,For the specified Number of times

```
--Replicate Function
Select*from tblPerson
select
Name,Gender,
Substring(Email,1,2)+Replicate('*',5)+
Substring (Email,CharINDEX('@',Email),Len(email)-
Charindex('@',Email)+1) as Email
From
tblPerson
```

Space Function:-

SPACE(Number of Spaces)

Returns the number of spaces ,Specifies the number of specifies arguments

```
--Space Function
select
Name+SPACE(5)+Email
from
tblPerson
```

PatIndexFunction

PatIndex('%Pattern%',Expression)

Returns the string position of the first Occurance of the Pattern in specified expression.It takes a two Arguments.The Pattern to be search and the expression.PATINDEX() is similar to CHARINDEX().With CHARINDEX() we can use the wildcards, Where as patindex provides this capibility.If the specified pattern is not found PATINDEX() return zero

```
--PatIndexFunction
select Email,PATINDEX('%@a.com',Email) as FirstOccurance
from tblPerson
where PATINDEX('%@a.com',Email)>0
```

Replace Function:-

Replace(String Expression,Pattern,ReplacementValue)

Replace All occurance of a specified string value with another string value.

```
--Replace Function
select
Email,REPLACE(Email,'.com','.net') as ReplacedEmail
From
tblPerson
```

Stuff Function:-

Stuff(Original Expression,Start,Length,Replacement Expression)

Stuff function() insert Replacement Expression, at the strat position specified , along with removing the character specified using length parameter.

25.Date time function In sql Server.

Data type	Format	Range	Accuracy	Storage size(bytes)
Time	hh:mm:ss[.nnnnnnn]	00:00:00.0000000 through 23:59:59.9999999	100 nanoseconds	3 to 5
Date	YYY-MM-DD	0001-91-01 through 9999-12-31	1 day	3
Small date time	YYY-MM-DD Hh:mmm:ss	1900-01-01 through 2079-06-06	1minute	4
Datetime	YYY-MM-DD hh:mmm:ss[.nnn]	1753-01-01 through 9999-12-31	0.003333 second	8
Datetime2	YYYY-MM-DD Hh:mmm:ss[.nnnnnnn]	0001-01-01-00:00:00.0000000 through 9999-12-31	100 nanoseconds	6 to 8
datetimeOffset	YYYY-MM-DD Hh:mm.ss[.nnnnnnn] [+|-]hh:mm	01-01-01 00:00:00.0000000 through 9999-12-31 01-01-02 23:59:59.9999999(in Utc)	100 nanosecond	8 to 10

Utc stands for Coordinated Universal time, based on which , the word regulates clocks and time. There are slights difference between GMT and UTC, but for the most common purposes, UTC is synonymous with GMT.

Function	DatetimeFormat	Description
GATEDATE()	2012-08-31 20:15:04.543	Commonely Used
CURRENT_TIMESTAMP	2012-08-31 20:15:04.543	ANSI SQl equivalent to GETDATE
SYSDATETIME()	2012-08-31 20:15:04.5380028	More fractional second Precision
SYSDATETIMEOFFSET()	2012-08-31 20:15:04. 5380028 +01:00	More fractional second Precession+time zone offset
GETUTCDATE()	2012-08-31 19:15:04.543	Utc Date and Time
SYSUTCDATETIME()	2012-08-31 19:15:04.5380028	Utc Date and Time, with more fractional second Precession

```
create table tblDateTime
(
c_time time(7) null,
c_date Date null,
c_smallDateTime smalldatetime null,
c_datetime datetime null,
c_datetime2 datetime2(7) null,
c_datetimeOffset datetimeoffset(7) null
)
insert into tblDateTime
values(GETDATE(),GETDATE(),GETDATE(),GETDATE(),GETDATE(),GETDATE())
```

```
update tblDateTime set c_datetimeOffset='2016-06-18 17:03:26.0870000
+10:00' where c_datetimeOffset='2016-06-18 17:03:26.0870000 +00:00'
select GETDATE()
select SYSDATETIMEOFFSET()
select CURRENT_TIMESTAMP()
select GETUTCDATE()
select SYSUTCDATETIME()
```

26.Is Date,Day,Mont,Year and DateName DateTime Function

ISDate():-

Checks the given value, is valid date, time, or datetime. Returns 1 for success zero for failure.

```
--IsDate() function
select ISDATE('Ganesh')--returns zero
select ISDATE(GetDate())--returns 1
Select ISDATE('2016-06-18 17:21:35.160')--returns 1
select ISDATE('2016-06-18 17:03:26.0870000')--returns 0
```

Day() function:-Returns the 'Day of the Month ' of the given date

```
--Day() function
select DAY(GetDate())--returns the current date day number of the month
select DAY('01/31/2012')--Returns 31
select* from tblDateTime
```

Month() function:-Returns the 'month number of the year of given date'

```
--Month() function
select MONTH(GETDATE())--returns the current date, month number of the year.
Select MONTH('01/31/2012')--returns 1
```

Year() function():- Returns the 'Year number of the given date'

```
--Year() funciton
Select YEAR(Getdate())--Reetuns the curret year of the given date
Select YEAR('01/31/2012')-- retun 2012
```

DateName() Function:-

DateName(Date Part,Date)-Returns a string ,that represent the part of the given date. This function takes two parameters. The first parameter Date Part specifies, the part of the date, we want. The second parameter, is the actual date, from which we want the part of the date.

```
--Examples-----------------------------
--DateName() Function:-
Select DATENAME(Day,'2012-09-30 12:43:46.8376')--retuns 30
SELECT DATENAME(WEEKDAY,'2012-09-30 12:43:46.8376')--Retuns Sunday
Select DATENAME(Month,'2012-09-30 12:43:46.8376')--retuns september
create table tblEmployeenew
(
Id int identity(1,1) not null,
Name varchar(100) not null,
DateOfBirth Datetime not null
)
insert into tblEmployeenew values('Sam','1980-12-30 00:00:00.000')
insert into tblEmployeenew values('Pam','1982-09-01 12:02:36.260')
insert into tblEmployeenew values('John','1985-08-22 12:03:30.370')
insert into tblEmployeenew values('Sara','1979-11-29 12:59:30.670')
Select
Name,DateOfBirth,DATENAME(WEEKDAY,DateOfBirth) as [Day],MONTH(DateOfBirth) as
MonthNumber,
DATENAME(MONTH,DateOfBirth) as [MonthName],
YEAR(DateofBirth) as [Year],
YEAR(DateOfBirth) as Year
From tblEmployeenew
```

27.DatePart,DateAdd and DateDiff function in Sql Server

DatePart(DatePart,Date)-Returns an integer repeasenting the specified date part. This function is similar to DateName().DateName() returns the nvarchar, where as Datepart() return an integer.

```
--DatePart
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 give date.

```
--DateAdd
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
```

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

```
--DateDiff
Select DATEDIFF(Month,'11/30/2005','01/31/2006')--returns 2
select DATEDIFF(Day,'11/30/2005','01/31/2006')--retuns 62
```

Create new function and Calculate the age:-

```
--Calculating Age-- create new function
Select dbo.fnComputeAge('11/30/2005')
select Name,DateofBirth, dbo.fnComputeAge(DateofBirth) as Age from tblEmployeenew
create function fnComputeAge(@DOB as Datetime)
returns nvarchar(50)
as
Begin
Declare @TempDate datetime,@years int,@months int,@days int
Select @TempDate=@DOB
Select @years= DateDiff(Year,@TempDate,GetDate())-
                            Case
                                  When
                                  (Month(@DOB)>Month(GetDate())) Or
                                   (Month(@DOB)=Month(Getdate()) And
Day(@DOB)>Day(Getdate()))
                                   Then 1
                                   Else
                                   0
                                   End
Select @TempDate=DATEADD(Year,@Years,@tempdate)
Select @months=DATEDIFF(Month,@tempDate,GetDate())-
                            Case
                            When DAY(@DOB)>DAY(GETDATE())
                            Then 1 Else 0
                            END
Select @TempDate=DATEADD(MONTH,@months,@TempDate)
Select @days=DATEDIFF(Day,@Tempdate,GetDate())
Declare @Age nvarchar(50)
```

```
set @Age=Cast(@years as nvarchar(4))+' Years'+Cast(@months as Nvarchar(2))+'
Months'+Cast(@days as nvarchar(20))+' Days Old'
return @Age
End
```

28.Cast and Convert Function in Sql Server

To convert one datatype to another data type Cast and convert function can be used.

Syntax of Cast and Convert function from MSDN

CAST(Expression as Datatyp [(Length)]

Convert(datatype[(length)],Expression [,Style])

```
select ID,Name,DateOfBirth,CAST(DateOfBirth as nvarchar) as ConvertedDOB from
tblEmployeenew
Select ID,Name,DateOfbirth, CONVERT(nvarchar,DateofBirth) as ConverteDOB from
tblEmployeenew
```

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

```
Select ID,Name,DateofBirth, CONVERT(NVARCHAR,DateOfBirth,103) as
ConvertedDOB from tblEmployeenew
```

Date part of DateTime

```
--To get the just date part form datetime
Select CONVERT(varchar(10),GetDate(),101)---return today date 06/18/2016
--In sql server 2008,Date datatype is introduce, so you can also use
Select CAST(GETDATE() as Date)--gives date only 2016-06-18
select CONVERT(Date,GETDATE())--give date part only 2016-06-18
```

Note:-To control the formatting of the datepart, Datetim has been converted to nvarchar style provided. When converting to Date Datatype, the convert() function will ignore the style parameter.

```
Select Id,Name,Name+'-'+Cast(ID as Nvarchar) as [Name-Id from]  from tblEmployeene
--Create new table tblRegistrationc
create table tblRegistration
(
Id int identity(1,1) not null,
Name varchar(100) not null,
Email nvarchar(100) not null,
RegisteredDate Datetime
)
select cast(RegisteredDate as DATE)     ,COUNT(ID) as Total from tblRegistration
Group by     cast(RegisteredDate as DATE)
```

Difference Cast-Convert

1. Cast is based on ANSI standard as Convert is Specific to Sql Server. So , if probability is concern and if you want to use the script with other database applications, use Cast()
2. Convert provides more flexibility then cast. For Example, it is possible to control how you want datetime data type to be converted using style with convert function.

Note:-the general Guideline is used Cast(), unless you want to take advantage of the style functionality in Convert()

29.Mathematical Function in Sql Server

Abs

Ceiling

Floor

Power

Rand
Square
Sqrt
Round

Abs

Abs(Numberic_Expression): Abs stands for Absolute and returns. The absolute (positive) number.

```
--Abs function
select ABS(-101.5)--returns 101.5,without the -sign  and
```

Ceiling and Floor:-

Ceilling(Numeric_expression) and Floor(numeric_expression)

Ceiling and Floor accept a numeric expression as a single parameter Ceiling() returns a smallest integer value greater than or equal to parameter whereas floor() returns the smallest integer less than or equal to parameter

```
--Ceiling and Floor function
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

Power(Expression,power)

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

```
--power function
select POWER(2,3)--returns 8
```

Square(number)

Returns the square of the given number.

```
--Square function
select SQUARE(9)--returns 81
```

SQURT(Number)

Returns square root of the given number.

```
--Sqrt funciton
select SQRT(81)--returns 9
```

Rand([Seed_value)]—returns the random float number between 0 and 1. Rand function takes an optional seed parameter. When see value is supplied the RAND() function always returns the same value for the same seed.

```
select Rand(1)--always retruns the same values
Select RAND()--Returns random values
--Generate Random between 1 and 1000
select FLOOR(Rand()*1000)
--Prints 10 random number betwwwn 1 and 100
Declare @Counter Int
Set @Counter=1
while (@Counter<=10)
Begin
```

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 of the digit that we want to round

2**. Length** parameters specifies the number of digits that we want to round to. If the length is the positive number, then the rounding is applied for the positive part, where as if the length is negative, then the rounding is applied to the number before the decimal.

3. The optional function parameter, used to rounding or truncation operations.0 indicates rounding, non-zero indicates truncation. Default, if not specified is 0.

```
-Round Function
--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
--Rount to 1 places after (to the right) the decimal point
Select ROUND(850.556,1)--Return 850.600
--truncate everything after 1 places, after(to the right) the decimal point
Select ROUND(850.566,1,1)--850.500
--Round the last 2 places before(to the left) the decimal point
Select ROUND(850.566,-2)--900.000
--Round the late 1 places before to the left the decimal point
Select ROUND(850.566,-1)--850.00
```

30.Scalar User Define function

From parts 22 to 29,we have learnt how to use many of the **System function** that are available in the **Sql Server.** In this session, we will turn out attention, to creating user define functions. In short UDF.
Ins Sql there are 3 types of User Define functions

1. Scalar functions
2. Inline table-valued function
3. Multi-Statement table-valued functions

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

```
Create function Function_Name(@Parameter1 datatype,@Parameter2 Datatype,@Parameter3
Datatype)
Returns Return_Datatype
as
Begin
--function Body
Return Return_DatatYpe
End
```

```
--create user define function that compute the age of the person.
select  dbo.CalculateAge('2015-7-25 17:38:42.013')
Create function CalculateAge(@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.CalculateAge('2015-7-25 ')
```

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

```
select  Sql.dbo.CalculateAge(dbo.Age('2015-7-25 '))
--Scalare user defnined function can be used int select clause
  select Name,DateOfBirth,dbo.CalculateAge(DateOfBirth) as Age from
tblEmployeenew
  --Scalar user defined function can be used in the select where clause
  Select Name,DateOfBirth,dbo.CalculateAge(DateOfBirth) as Age from
tblEmployeenew
  where dbo.CalculateAge(DateOfBirth)>30
```

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

To alter function we use **Alter function** FunctionName Statement and to delete it, we use **Drop Function** FuncitonName.

31. Inline Table Valued Function

In **part 30.** Of this video series we have seen how to create and call **'scalar user define functions'** In this lesion we will learn about **'Inline table values functions'**

SCALAR FUNCTION-Returns a **Scalar** value

Inline table value function –returns a table

```
create function fn_EmployeeByGender(@Gender nvarchar(10))
Returns table
as
Return
(select Id,Name,gender,DepartmentId from tblEmployee where gender=@Gender)
```

1. We specify table as the return type, instead of any scalar data type
2. The function body part is not enclosed between Begin and End Block.
3. The structure of the table that gets returned, in determined by the select statement with in the function.

```
--to call this function
Select* from fn_EmployeeByGender('Male')
```

Where can we use Inline table valued functions?

1. Inline table valued function 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 join with other tables.

```
Select Name,Gender,DepartmentName from fn_EmployeeByGender('Male') E
join tblDepartment D on D.Id=E.DepartmentId where Name='Ganesh'
```

32. Multi-Statement table valued functions

Multi-Statement table valued function are very similar to inline table valued functions, with a few differences.

```
--Inline table valuef function
create function fn_ITVF_GetEmployee()
Returns Table
as
Return (select Id,Name,CAST(DateOfBirth  as Date) as Dob from tblEmployeeNew
--MulitStatement Table Valued function
create function fn_MSTVF_GetEmployee()
Returns @Table table(id int,Name nvarchar(20),DOB date)
as
Begin
Insert into @Table
```

```
Select Id,Name,CAST(dateOfBirth as Date) from tblEmployeenew
Return
End
```

Differences between Inline table valued function and Multi-Statement table valued function

1. In an inline table valued function, the returns clause cannot contain the structure of the table, the function returns. whereas, with the multi-statement table valued function, we specify the structure of the tables that gets returned
2. Inline table valued function can not have begin and End block ,where as the multi-statement function can have.
3. Inline table valued functions are better for performance, then 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 functions.

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.

```
-- we can update data in inline table value function
select * from fn_ITVF_GetEmployee()
update fn_ITVF_GetEmployee() set Name='Sam1' where ID=1
---we can not update data in multiline table value funciton
select*from fn_MSTVF_GetEmployee()
update fn_MSTVF_GetEmployee() set Name='Sam' where id=1
```

33. Important Concepts Related to Functions.

Deterministic Function:- It always returns 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 function

Nondeterministic Functions :- It may return different result each time they are called with a specific set of input values event if the database state that they access remains the same

Example:-GetDate() and CURRENT_TIMESTAMP

Rand() function :- It 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.

Encyrypting a Funciton Definition using with encryption option

We have learnt how to encrypt stored procedure text using with ENCRYPTION OPTION in Part 18. Along the same line you can also encrypt the function text. Once encrypted you can not view text of the function, sp_helptext system stored procedure. If you Try to, you will get message stating 'The text for object is encrypted'. There are ways to decrypt, which is beyond the scope of this video.

Use WITHENCRYPTION

Creating a Function with SCHAMABINDING Option:-

SchamaBinding specifies that the function is beyond the database object that it references ,When SCHEMABINDING is specified, base object can not be modified in any way that would affect the function definition. The function definition itself must be modified or droped to remove dependencies on the object that it to be modified

```
--deterministic Function
select COUNT(*) from tblEmployeenew
select SQUARE(3)

--Non deterministic funciton
```

```
select GETDATE()
select CURRENT_TIMESTAMP
--Rand functon is both determinestic function and Non Deterministic function
select RAND(1)
select RAND()
--Create simple functon to do encryption
alter function  fn_GetNameByID(@ID int)
Returns nvarchar(30)
with Encryption
as
Begin
return (select name from tblEmployeenew where Id=3)
End
--SchamaBinding Option
alter function  fn_GetNameByID(@ID int)
Returns nvarchar(30)
with SchemaBinding
as
Begin
return (select name from dbo.tblEmployeenew where Id=3)
End
--Now reference table can not be deleted
drop table tblEmployeenew
```

34. Temporary Table In Sql Server

What is Temporary Tables?

Temporary tables are very similar to the permanent tables, permanent table get created in the database you specify and remain the database permanently, until you delete(drop) them .On other hand temporary table 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

Local Temporary tables

```
--34.Temporary tables in sql Server
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
```

Check if the local temporary table is Created:-

Temporary table are created in TempDB. Query the sysObject system table in TempDB. The name of the table, is suffix with a lot of underscores and the random numbers. 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 , the connection that has created it, is closed.

If the user want to explicitly want to drop the temporary tables he can use using ***Drop table #PersonDetails***

If the temporary table is created inside the stored procedure , its get dropped automatically upon the completion of the stored procedure execution

```
--create stored procedure using temporary table
```

```
execute spCreateLocalTempTables
create procedure spCreateLocalTempTables
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 connectons,to create local temporary table with the same name. for example User 1 and User 2 both can create a local temporary tables with the same name #PersonDetails

Global Temporary tables

To create a global temporary tables, prefix the name of the table with 2 pounds symbols (##).

```
create table ##EmployeeDetails(Id int,Name nvarchar(20))
insert into ##EmployeeDetails values(1,'Mike')
insert into ##EmployeeDetails values(2,'John')
insert into ##EmployeeDetails values(3,'Todd')
```

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 table with the same name but global temporary table has to be unique, and if you inspect the name of global temp table, in the object explorer, there will be no random numbers, suffixed at the end of the table name

Differences between local temporary table and Global Temporary tables

1. Local temp tables are prefixed with single pound(#) whereas global temporary table are prefixed with double pound(##) symbols.
2. Sql server appends some random numbers at the end of the local temporary tables whereas this is not done for the global temp table name
3. Local temporary table are only visible to the sql server which has create it, whereas glo
4. bal temporary table are visible to all sql server sessions.
5. Local temporary tables are automatically dropped , when the session that created the temporary table is closed, whereas global temporary tables are destroyed when the last connection that is referencing the global temp table is closed

35. Indexes in Sql Server

Why Indexes?

Indexes are used by queries to find data from table quickly, indexes are created in table and views . Indexes on table or a view, is very similar to index that we find in a book. If you don't have an index , and I ask you to locate specific chapter in the book you will have to look every page of the book.

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

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

In the similar ways table and View index 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 query engine ,check every row in the table from the beginning to the end. This is called the table scan .Table scan is bad for the performance

Index Examples:-

At the moment, employee table does not have the index in salary column

```
select*From tblEmployee where salary>5000 and salary<20000
```

To find all the employee who has salary greate then 500 and less then 20000, the query engine has to check each and every row in the table, resulting in the table scan, which can adversely affect the performance specialy if the table is large .Since there is no index , to help the query , the query engine performs an entiere table scan.

Creating and Index:-

```
---Create Index
create index IX_tblEmployee_Salary
on  tblEmployee  (salary asc)
--to see all the index
sp_helpIndex tblEmployee
--to drop index
drop index tblEmployee.IX_tblEmployee_Salary
```

When the sql server has to execute some query , it has an index on the salary column to help this query. Salary between the range 5000 and 7000 usually present in the bottom, Since the salary are arranged in the ascending order sql server picks the row address form index and directly fetch the record from the table , rather than scanning each row in the table. This is called an index seek

36.Claustered and nonClustered Index

Index type:-

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

Clustered Index:-

A clustered index determines the physical order of the data in a table. For his reason, as table can have only once clustered index

PK__tblEmplo__3214EC071367E606 clustered, unique, primary key located on PRIMARY Id

When creating the table Primary key constraint create closeted indexes automatically if no clustered index already exist in the table.

To confirm:- `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

A clustered index is analogous to a telephone directory ,where the data is arranged by the last name. we just learned that a table can have only one clustered index, however index can contains multiple

columns(a complete index),like the way is telephone directory is organized by the last name or first name.
Create the composite clustered index on the gender and salary columns

```
--First drop the existing clustered index
drop index tblEmployee.PK__Employee__3214EC071B0907CE--also can delete from
object explorer
--create new clustered Index
create Clustered index IX_tbleEmployee_Gender_Salary
on tblEmployee (Gender desc,Salary asc)
```

Non Clustered Index:-

```
create NonClustered index IX_tblEmployee_Name on
tblEmployee(Name)
```

A nonclustered index is analogous to an index in a textbox. The data is stored in one places, the index in another places .the index will have pointer to the storage location of the data.

Since the nonclustered index is stored separately from the actual data , table can have more than one nonclustered index. Just like how a book can have a index by chapters at the beginning and another index by common term at the end .
In the index itself, the data is stored in ascending or descending order of the index key which does not in any way influence the storage of data in the table.

Differences between clustered and nonclustered index

1. Only one clustered index per table, where a you can have more than one non clustered index
2. Clustered index is faster than a nonclustered 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, an hence doesn't require additional disk space, but whereas a non-clustered index is stored separately from the table, additional storage space is required.

37. Unique and Non-Unique Indexes

Unique index :-

Unique index is used to enforce uniqueness of key values in the index.
Note:- by default, primary key constraint, created unique clustered index.
Uniqueness is a property of an index, and both clustered and non-clustered indexes can be unique

```
--create unique non-clustered index
create unique nonclustered index
UIX_tblEmployee_firstName_lastName
on tblEmployee(Name)
```

Differences 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 crated behind the scenes.

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

To make our intension 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 point to remember:-

1. **By default**, a PRIMARYKEY constraint, creates a unique clustered index, whereas 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** can not be created on an existing table, if the table contains duplicate values in the key columns. Obviously, to solve this, remove the key column form the index definition or delete or update the duplicate values.

38. Advantages and Disadvantages of Index:-

Advantages:-

Index are used by queries to find data quickly

Disadvantages of Index:-

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

Insert Update Delete statements can become slow:-When DML(Data manipulation language) statement(INSERT,UPDATE,DELETE) modifies data in a table, the data in all the indexes also needs to be updated. Index can help, to sear and locate the rows, that we want to delete, but too many indexes to update can actually hurt the performances of data manipulations.

What is a conversing query- If all the column that you have requested in the select clause of query, are present in the index, then there is no need to look up 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.

39. Views on Sql Server:-

What is View?

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

Advantages of Views:-

Views can be used to reduce the complexity of the database schema

Views can be used as a mechanism to implement row and column level security.

Views can be used to present aggregated data and hide detailed data.

To modify a view:- Alter view statement

To drop a view:- Drop view vWName

40. Updatable Views in Sql Server

```
create view vWEmployeeDataExceptSalary
as
Select ID,Name,Gender,DepartmentId from tblEmployee
select *from vWEmployeeDataExceptSalary
```

Is it possible to insert, update & delete from the base table tblEmployee thru the view?

```
Yes We can update data thrugh view also
update vWEmployeeDataExceptSalary set Name='Mikey' where Id=2

--create views using multiple tables
create view vwEmployeeDetailsbyDepartment
as
Select tblEmployee.Id,Name,Salary,Gender,DepartMentName from tblEmployee
join tblDepartment on tblDepartment.Id=tblEmployee.DepartmentId
select*from vwEmployeeDetailsbyDepartment
update vwEmployeeDetailsbyDepartment set DepartmentName='IT' where
Name='Dhurmush'
```

Conclusion:- IF a view based on multiple tables, and if you update the view, it may not update the underline table 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 table in later video session.

41. Index views in sql server

What is an Index views?

Or

What happens when you create a index on a views?

A standard or non-indexed view, is just stored Sql query. When you try to retrieve data form the view , the data is actually retrieve from the underline base table.

So view is just a virtual table. It does not stored 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 in oracle, Materialized views

```
--create new product table
create table tblProduct
(
Id int identity(1,1) primary key not null,
Name varchar(100) not null,
UnitPrice int not null
)
--insert into product table
insert into tblProduct values('Books',20)
insert into tblProduct values('Pens',14)
insert into tblProduct values('Pencils',20)
insert into tblProduct values('Clips',20)
--create one more table  of product sales
create table tblProductSales
(
 productId int not null,
 QuantitySold int not null,
 constraint fk_tblProductSales_Productid   foreign key (productId) references
tblProduct(id)
 )
 insert into tblProductSales values(1,10)
 insert into tblProductSales values(3,23)
 insert into tblProductSales values(4,21)
 insert into tblProductSales values(2,12)
 insert into tblProductSales values(1,13)
 insert into tblProductSales values(3,12)
 insert into tblProductSales values(4,13)
 insert into tblProductSales values(1,11)
 insert into tblProductSales values(2,12)
 insert into tblProductSales values(1,14)

 ---Now create a 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.Id=dbo.tblProductSales .productId
 group by Name
```

```
select *from vwTotalSalesByProduct
```

1. The view should be created with schema binding option
2. If an aggregate function in the select list, reference an expression, and if there is a possibility for that expression to become null, then, a replacement values should be specified.
3. If group by is specified , the view select must be contain a COUNT_BIG(*) expression
4. The base table in the view , should be references with 2 part name

```
--create index in view
 create unique clustered index UIX_vwTotalSalesByProduct_Name on
vwTotalSalesByProduct(Name)
```

42. View limitations

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

```
-- we can create one views like this
 create view vwEmployeeDetails
 as
 Select ID,Name,Gender,DepartMentId from tblEmployee
 --but we can not create view like this
 create view vwEmployeeDettails
 @Gender nvarchar(20)
 as
 select ID,Name,Gender,DepartmentId from tblEmployee
 where Gender=@Gender
 ---we can filter gender in where clause
 select *from vwEmployeeDetails where gender='Male'
 --inline table valued funtion to replace parameterized views
 create function fnEmployeeDetails(@Gender nvarchar(20))
 Returns table
 as
 Return
 (select Id,Name,gender,DepartmentId  from tblEmployee where gender=@Gender)
 select * from fnEmployeeDetails('Male')
```

    2. Rules and Default cannot be associated with views.
    3. The order by clause is invalid in views unless TOP or FOR XML is also specified
    4. View cannot be based on temporary tables.

43. DML 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 triggering action. The Insert, Update, and Delete statements, causes an INSTEAD of Triggers to fire instead of the respective statement execution.

```
--create triggers
 create Trigger tr_tblEmployee_ForInsert
 on tblEmployee
 for Insert
 as
 Begin
 select *from inserted
 End
--trigger goint to fire while goint to insert data
 insert into tblEmployee values('Kalpana','Female',10000,'Lalbandi',1,3)
 --create tblEmployeeAudit table-----------------------------------------
 create table tblEmployeeAudit
 (
 Id int identity(1,1) not null,
 AuditData nvarchar(max) not null,
 )
 create trigger tr_tblEmployeeAudit_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('Karuna','Female',10000,'Bardibash',1,3)
 select*from tblEmployeeAudit
 --Delete triggers
 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 Id='+CAST(@ID as
nvarchar(5))+'is deleted at '+CAST(GetDate() as nvarchar(20)))
 End
 delete from tblEmployee where Name='Karuna'
  select*from tblEmployeeAudit
```

44. after update trigger in sql server

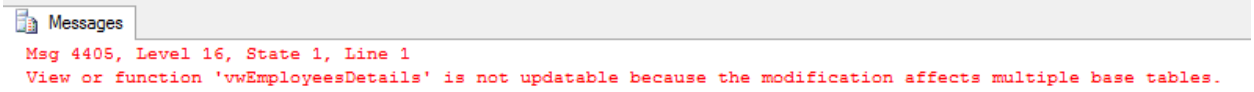
```
-------------------------------------------------------------------------------
---------------------------------------
 --44.After Update triggers
 create trigger tr_tblEmployee_ForUPdate
 on tblEmployee
```

```
 for Update
 as
 Begin
select *From deleted
select *From inserted
End
 select*from tblEmployee
 update tblEmployee set Name='Parsuram',Salary=14000,City='Biratnager' where
Id=5

 -----
 alter trigger tr_tblEmployee_ForUPdates
 on tblEmployee
 for Update
 as
 Begin
declare @Id int
declare @OldName nvarchar(20),@NewName nvarchar(20)
declare @OldSalary int ,@NewSalary int
declare @OldGender nvarchar(20),@NewGender nvarchar(20)
declare @OldDepartmentId int,@NewDepartmentId int
declare @AutoString nvarchar(1000)
select * into #TempTable from inserted
while(Exists(select ID from #TempTable))
Begin
Set @AutoString=''
select top 1
@Id=ID,@NewName=Name,@NewGender=Gender,@NewSalary=salary,@NewDepartmentId=Dep
artmentId from #TempTable
select
@OldName=Name,@OldGender=gender,@OldSalary=salary,@OldDepartmentId=department
Id from deleted where @Id=id
set @AutoString='Employee with id'+CAST(@Id as nvarchar(4))+'Changed'
if(@OldName<>@NewName)
set @AutoString=@AutoString+'Name from '+@OldName+'to '+@NewName
if(@OldGender<>@NewGender)
set @AutoString=@AutoString+'Gender from '+@OldGender+'to '+@NewGender
if(@OldSalary<>@NewSalary)
set @AutoString=@AutoString+'Salary from '+Cast(@OldSalary as
nvarchar(10))+'to '+cast(@NewSalary as nvarchar(10))
if(@OldDepartmentId<>@NewDepartmentId)
set @AutoString=@AutoString+'DepartmentId from '+Cast(@OldDepartmentId as
nvarchar(10))+'to '+cast(@NewDepartmentId as nvarchar(10))
insert into tblEmployeeAudit values(@AutoString)
delete from #TempTable where @Id=Id
End
End
 select*from tblEmployee
 update tblEmployee set
Name='Krishma',Salary=16500,City='Lalbandi',DepartmentId=2 where  Id=5

 select*From tblEmployeeAudit
```

Note: the after trigger for update event, make use of both **inserted** and **deleted** tables. The inserted table contain the updated data and deleted table contain the old data

45. Instead of Insert trigger.

```
insert into vwEmployeesDetails values ('Usha','Male',2)
```




```
--create view--
 create view vwEmployeesDetails
 as
 select tblEmployee.Id,Name,Gender,DepartmentName from tblEmployee
 join tblDepartment on tblDepartment.Id=tblEmployee.DepartmentId
select*from  vwEmployeesDetails
 insert into vwEmployeesDetails values ('Usha','Male',2)
 --create new triggers
 create trigger tr_vwEmployeesDetails_InsteadOfInsert
 on
 vwEmployeesDetails
 Instead of Insert
 as
 Begin
 select * from inserted
 select* from deleted
 End
  insert into vwEmployeesDetails values (26,'Usha','Male','IT')
 ---------create trigger
 create trigger tr_vwEmployeesDetails_InsteadofInsert1
 on vwEmployeesDetails
 instead of Insert
as
begin
declare @DepartmentId int
--chek if there is valid deparmtnet for the given deparmetmenet name
select @DepartmentId=tblDepartment.Id from tblDepartment join inserted on
inserted.DepartmentName=tblDepartment.DepartmentName
-- if depatment is null throw an error and  stop processing
if(@DepartmentId IS NULL)
BEGIN
RaisError('Invalid Department Name .Statment Terminated',16,1)
return
End
--finally insert into  tblEmployee table
insert into tblEmployee(Id,Name,gender,DepartmentId)
select ID,Name,gender,@DepartmentId from inserted
End
```

46. Instead of Update trigger

```
--update affecting just one basetable
select*from tblDepartment
update vwEmployeesDetails set DepartMentName='Admin' where Id=4
select*from vwEmployeesDetails
```

```
---create trigger
create trigger tr_vwEmployeeDetails_InsteadOfUpdate
on vwEmployeesDetails
instead of update
as Begin
--if employee is updated
if(UPDATE(ID))
Begin
Raiserror('Id can not be changed',16,1)
Return
End
--If department is updated
if(UPDATE(departmentName))
 declare @DeparetmentId int
 select @DeparetmentId=tbldepartment.Id from tblDepartment join inserted on
inserted.DepartmentName=tblDepartment.DepartmentName
 if(@DeparetmentId is null)
 begin
 Raiserror('Invalid departmnet Name',6,1)
 return
 end
 --if gender is updated
 if(UPDATE(gender))
 begin
 update tblEmployee set gender=inserted.gender from inserted join
 tblEmployee on tblEmployee.id=inserted.id
 End
 --if name is updated
 if(UPDATE(Name))
 Begin
 update tblEmployee set Name=inserted.name from inserted
 join tblEmployee on tblEmployee.id=inserted.id
 End
 End
update vwEmployeesDetails set
Name='Johny',Gender='Female',DepartmentName='Payroll' where ID=1
```

event

46. Instead of Delete trigger

```
delete from vwEmployeesDetails where ID=1
```

Messages

```
Msg 4405, Level 16, State 1, Line 1
View or function 'vwEmployeesDetails' is not updatable because the modification affects multiple base tables.
```

```
elete from vwEmployeesDetails where ID=1
--create a triger
alter trigger tr_vwEmployeesDetails_InsteadOfDelete
on vwEmployeesDetails
instead of delete
as begin
--delete  tblEmployee from  tblEmployee join deleted on
```

```
--tblEmployee.Id=deleted.id
--subquery
delete from tblEmployee where ID in(select id from deleted)
End
delete from vwEmployeesDetails where id=2
select *from vwEmployeesDetails
```

Note:-In most case join are faster then sub queries. Howerver, in cases when you only need subset of records from a table that you are joining with , sub queries can be faster.

48. Derived tables and CTE

```
create view vwEmaployeeCount
 as
 select
 DepartMentName,DepartmentId,COUNT(*) as TotalEmployee from tblEmployee
 join tblDepartment on tblDepartment.Id=tblEmployee.DepartmentId
 group by DepartmentName,DepartmentId

 select*from vwEmaployeeCount

 select DepartmentName,TotalEmployee from vwEmaployeeCount
 where TotalEmployee>=2
 select DepartmentName,DepartmentId,COUNT(*) as TotalEmployee into
 #tblemployeeCount1
 from tblEmployee join
 tblDepartment on tblDepartment.Id=tblEmployee.DepartmentId
 group by DepartmentName,DepartmentId
 select DepartmentName,TotalEmployee from #tblemployeeCount1 where
totalEmployee>=2
 drop table #tblemployeeCount1
```

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

Table Variables

```
declare @tableEmployeeCount table(DepartmentName nvarchar(20),DeparetmentId
int,totAlEmployee int)
 insert @tableEmployeeCount
 select DepartmentName,DepartmentId,COUNT(*) as totalEmployee   from
tblEmployee
 join tblDepartment on tblDepartment.Id=tblEmployee.DepartmentId
 group by DepartmentName,DepartmentId
 select DepartmentName,totAlEmployee from @tableEmployeeCount where
totAlEmployee >=2
```

Note:-just like the temptables, a talbe varible also created in tempDB. The scope of a table variable is batch, Stored procedure , or statement block in which it is declared. They can passed as parameter between procedure

Derived Tables:-

```
Select DepartmentName,TotalEmployee
 from (
            select DepartmentName,DepartmentId, COUNT(*) as totalEmployee
from tblEmployee join
            tblDepartment on tblDepartment.Id=tblEmployee.DepartmentId
            group by DepartmentName,DepartmentId
      ) as EmployeeCount where totalEmployee>=2
```

Note:- Derived table available only in the context of the current query.

Common table Expression(CTE)

```
with EmployeeCount(DepartmentName,DepartmentId,totalEmployee)
as
(
Select DepartmentName,DepartmentId, COUNT(*) as totalEmployee  from
tblEmployee join
tblDepartment on tblDepartment.Id=tblEmployee.DepartmentId
group by DepartmentName,DepartmentId
)
select DepartmentName,totalEmployee from EmployeeCount where totalEmployee>=2
```

A CTE can be thought of as a temporary result that is defined within the execution scope of a single Select, Insert, Delete, or create view statement. A CTE is similar to a derived table in that it is not stored as an object and only for the duration of the query.

49. Common table expression(CTE):-

It is introduced in sql server 2005, a CTE is temporary result set, that can be referenced within the select , Insert, Update or Delete statement that immediately follows the CTE

Syntax:-

```
with cte_name(column1,column2,.............)
as
(Cte_Query)
```

```
with EmployeeCount(DepartmentName,DepartmentId,tot)
as
(
Select DepartmentName,DepartmentId, COUNT(*) as totalEmployee  from
tblEmployee join
tblDepartment on tblDepartment.Id=tblEmployee.DepartmentId
group by DepartmentName,DepartmentId
)

select DepartmentName,tot from EmployeeCount where tot>=2
```

```
---Get error if we immidetley not used cte
with cte_name(column1,column2,.............)
as
(Cte_Query)
--
with EmployeeCount(DepartmentName,DepartmentId,tot)
as
(
Select DepartmentName,DepartmentId, COUNT(*) as totalEmployee  from
tblEmployee join
tblDepartment on tblDepartment.Id=tblEmployee.DepartmentId
group by DepartmentName,DepartmentId
)
select 'Hello'
select DepartmentName,tot from EmployeeCount where tot>=2
```

A CTE only by a referenced by a Select, Insert, Update or Delete statement that immediately follows the CTE.

```
with EmployeeCountBy_Payroll_It_department(DepartmentName,total)
as
```

```
(
 select DepartmentName, COUNT(tblEmployee.Id) as Total from tblEmployee join
tblDepartment on tblDepartment.Id=tblEmployee.DepartmentId where
DepartmentName in('Payroll','It')
 group by DepartmentName
),
EmployeeCountby_Hr_Admin_department(DepartmentName, total)
as
(
select DepartmentName, COUNT(tblEmployee.Id) as Total from tblEmployee join
tblDepartment on tblDepartment.Id=tblEmployee.DepartmentId where
DepartmentName in('Hr','Admin')
group by DepartmentName
)
select *from EmployeeCountBy_Payroll_It_department
union
select *from EmployeeCountby_Hr_Admin_department
```

50. Updatable CTE'S

Is it possible to update a CTE?

```
--Cte with one base table
with Employee_Name_Gender
as
(
Select ID,Name,Gender from tblEmployee
)
select*From tblEmployee
```

```
with Employee_Name_Gender
as
(
Select ID,Name,Gender from tblEmployee
)
update Employee_Name_Gender set Gender='Female' where id= 3
```

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

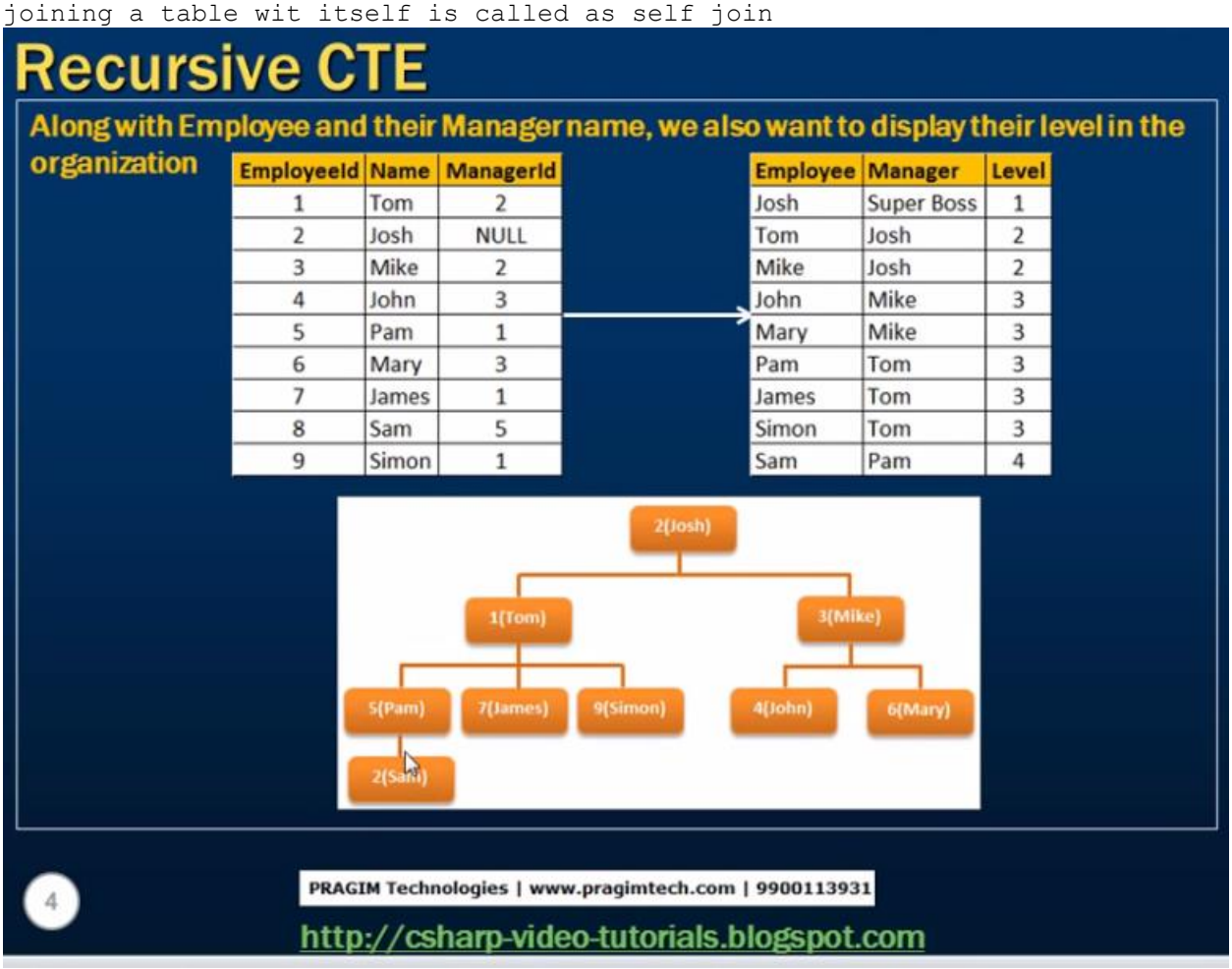
In this case , Updating Employee_name_gender CTE, updates tblEmployee tables

```
--Cte on 2 base tables, update affecting only one base table
with EmployeeDepartments
as
( select tblEmployee.Id,Name,gender,DepartmentName From tblEmployee join
tblDepartment on tblDepartment.Id=tblEmployee.DepartmentId
)
UPDATE EmployeeDepartments SET gender='Male' where ID=3

with EmployeeDepartments
as
( select tblEmployee.Id,Name,gender,DepartmentName From tblEmployee join
tblDepartment on tblDepartment.Id=tblEmployee.DepartmentId
)
UPDATE EmployeeDepartments SET DepartmentName='IT' where ID=3
```

Note- if cte is based on more then one table, and if update affects only one base table, then the update is allowed.

51. Recursive CTE

```
--A simple self join
select Employee.Name as [Employee Name],
ISNULL(Manager.Name, 'Super Boss') as [Manager Name]
 from tblEmployee as Employee
left join tblEmployee as Manager
on Manager.Id=Employee.ManagerId
joining a table wit itself is called as self join
```



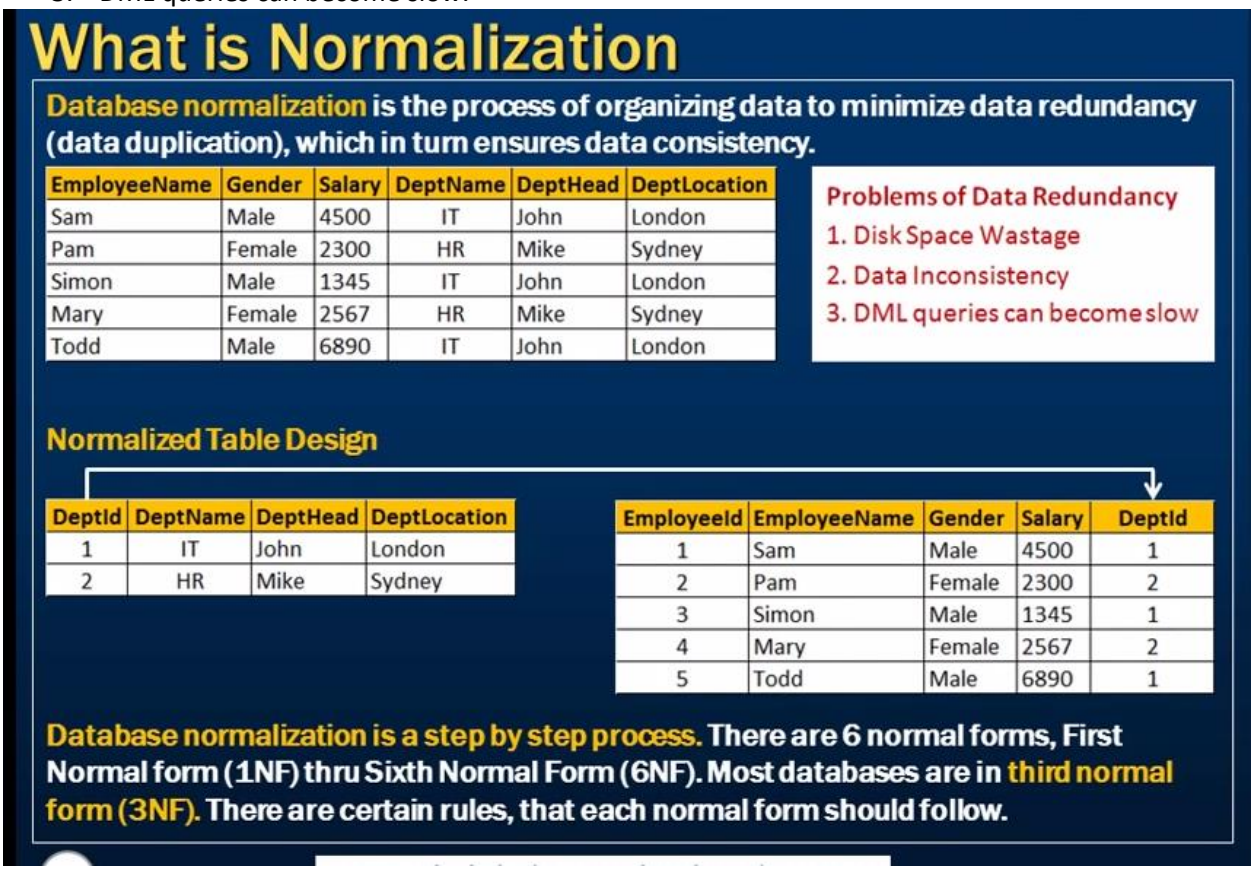

```
with EmployeeCTE(EmployeeId,Name,ManagerId,[Level])
as
(
Select tblEmployee.Id,Name,managerid,1 from
tblEmployee where managerid IS NULL
union all
select
tblEmployee.Id,tblEmployee.Name,tblEmployee.ManagerId,EmployeeCTE.[Level]+1
from tblEmployee
join EmployeeCTE on EmployeeCTE.EmployeeId=tblemployee.ManagerId
)
select empCTE.Name as Employee,ISNULL(mgrCTE.Name,'Super Boss') as
Manager,empCTE.[Level] from EmployeeCTE empCTE
left join EmployeeCTE mgrCTE
on empCTE.ManagerId=mgrCte.EmployeeID
```

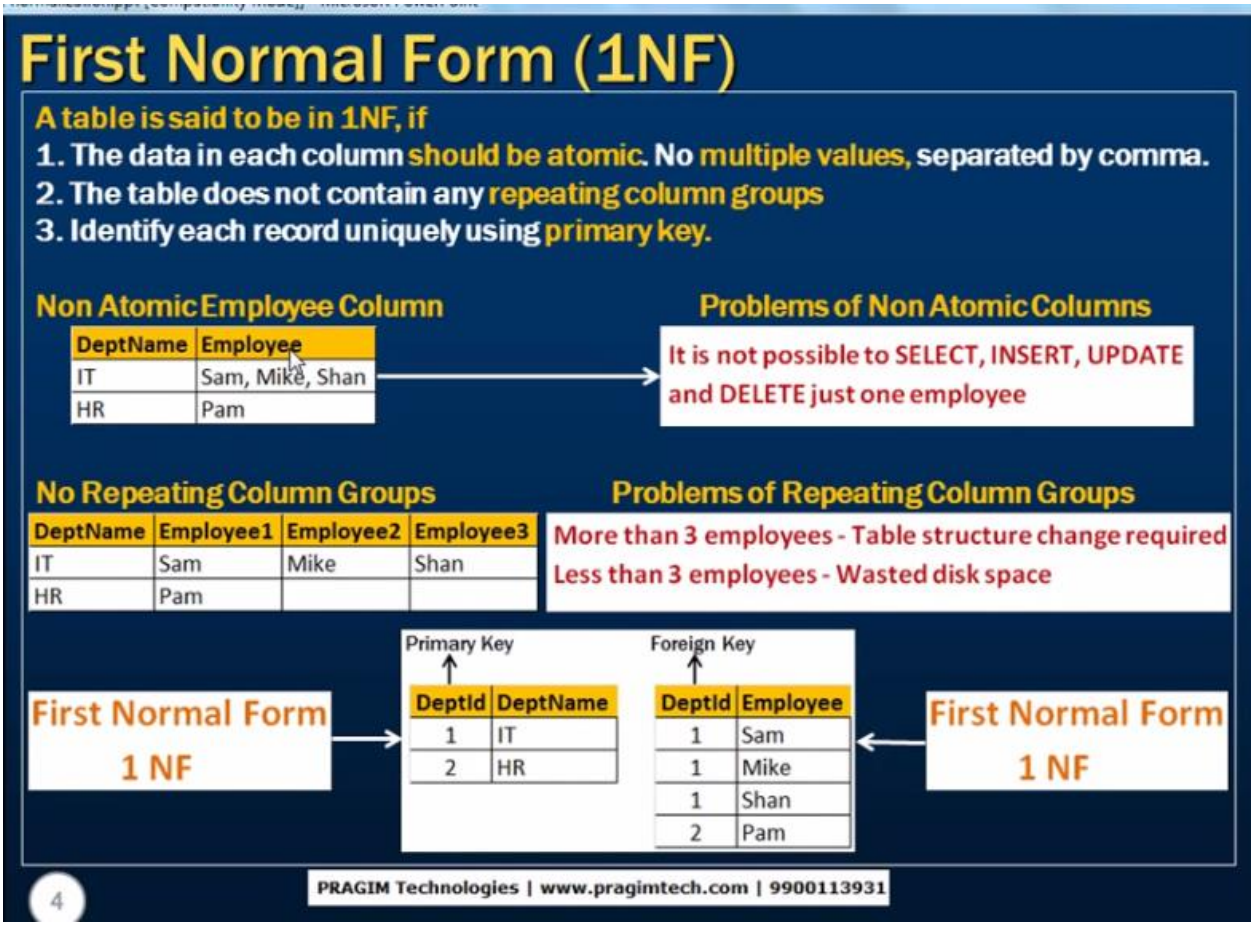
52. Database Normalization

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

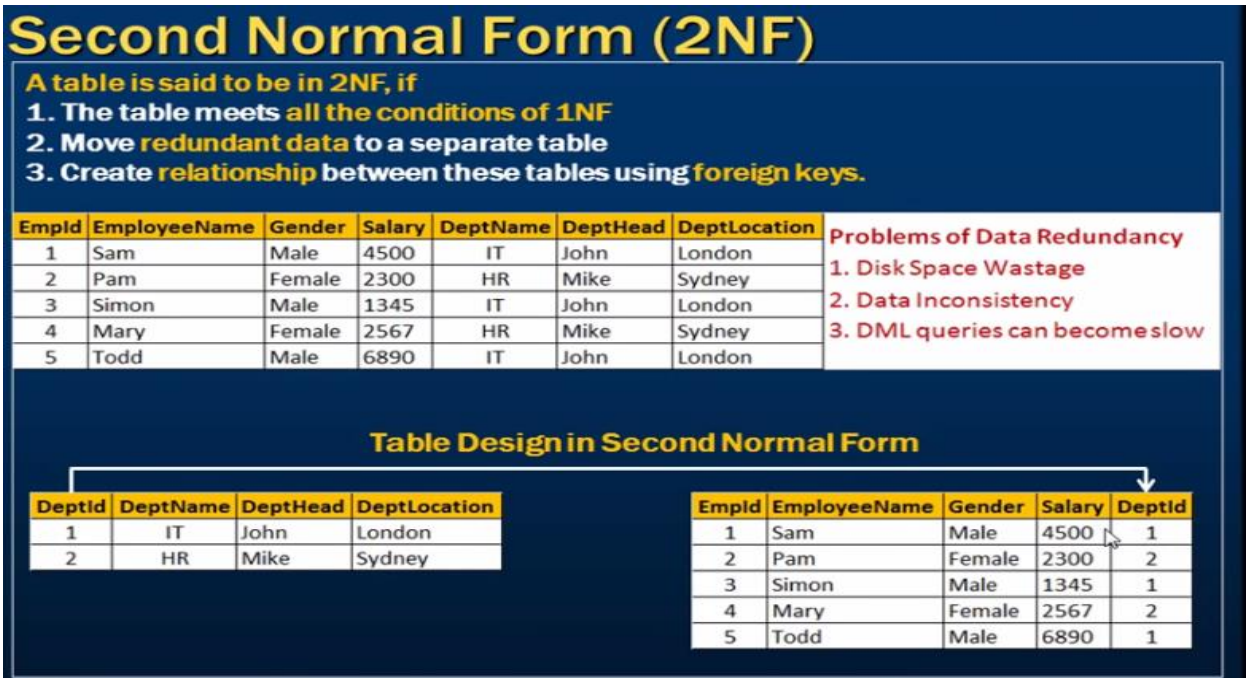
Problems of Data Redundancy

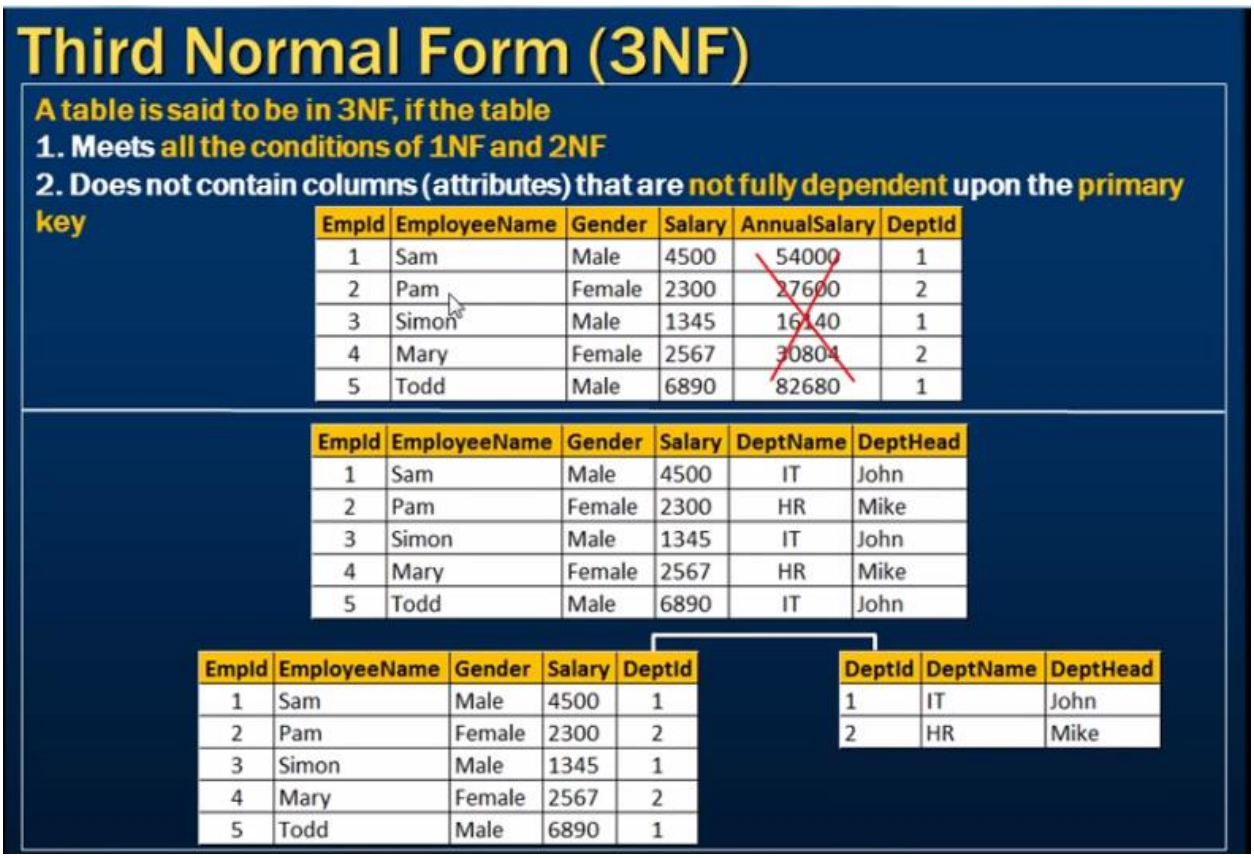
1. Disk space Wastage
2. Data Inconsistency
3. DML queries can become slow.





53. Second Normal Form and Third Normal Form

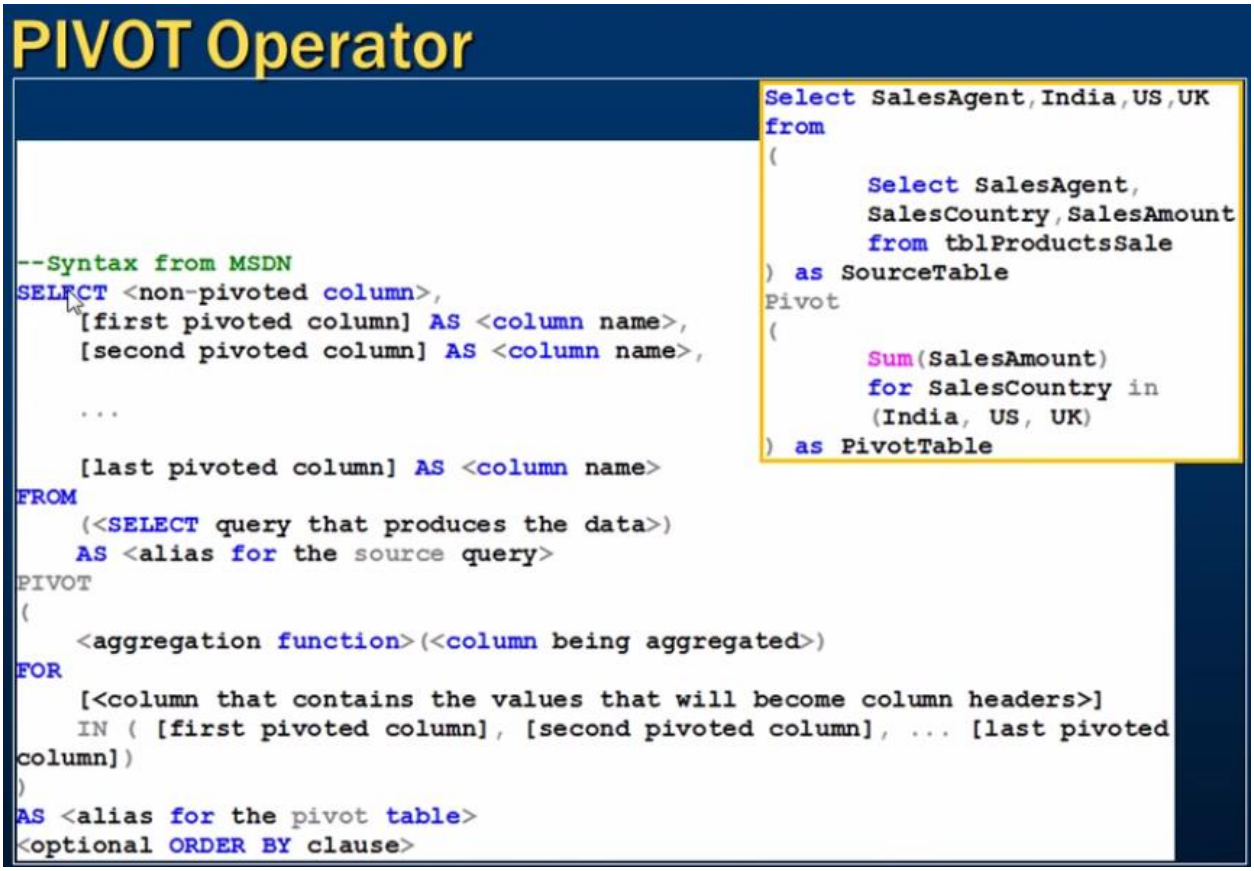




54. PIVOT OPERATOR

Pivot is sql server operator that can be used to turn unique value form one column into multiple column in output, there by effectively rotating table.

```
--54.PIVOT operator in SQL server-----------------------
--Gorup by Query
select SalesCountry,SalesAgent, SUM(SalesAmount) as Total  from productSales
group by SalesCountry,SalesAgent
Order by SalesCountry,SalesAgent
--Query Using PIVOT operator
select SalesAgent,India,US,UK from ProductSales
PIVOT
(Sum(salesAmount)
for SalesCountry
IN([INDIA],[US],[UK])
)
AS PivotTables
```



55. Error Handling in sql server

Error Handling

With the introduction of Try/Catch block in Sql server 2005, error handling in sql server , is now similar to programming language like c# and Java

Error handling in sql server sql server 2000--@@Error

Error handling in sql server 2005-try.. Catch

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

```
--frist create two tables
select*from tblProduct
create table product
(
ProductID  int identity(1,1) not null,
Name varchar(100) not null,
UnitPrice int not null,
QtyAvailabe int not null
)

create table productsell
(
productsalesId int identity(1,1) not null,
productId int not null,
QuantitySold int not null
)
```

```
alter procedure spSellProduct 1,10
@ProductId int,
@QuantityToSell int
as
Begin
--Chek the stock availabe , for the product that we want to sell
Declare @StockAvailable int
select @StockAvailable= QtyAvailabe from  product
where ProductID=@ProductId
--throw an error  to the calling application, if enought stock is not
availabe
If(@StockAvailable<@QuantityToSell)
Begin
Raiserror('Not enough stock availabe',16,1)
End
--if enough stock availabe
Else
Begin
Begin Tran
--first reduce the quantiy available
update product set  QtyAvailabe=(QtyAvailabe-@QuantityToSell)
where ProductID=@ProductId
Declare @MaxProductSalesId int
select @MaxProductSalesId=case  when
                                        max(ProductsalesId) is null
                                        then 0
                                        else
                                        max(ProductsalesId) End
                                        from productsell
--increment @MaxProductSalesId by 1
Set @MaxProductSalesId=@MaxProductSalesId+1
insert into productsell values(@ProductId,@QuantityToSell)
if(@@ERROR<>0)
begin
rollback tran
print 'Transaction roolback'
End
else
Begin
Commit Tran
print 'transaction commit'
End
End
End



select *from product
select*From productsell
```

RaisError('Error Message',ErrorSeverity,ErrorState)
Create and return custom errors
Severity Level=16(indicates general error that can be corrected by the user)
State:- Number between 1 & 255.RAISERROR only generates errors with state from 1 through 127.
@@Error returns a NON-Zero value, if there is an error , otherwise zero , indicating that the previous Sql statement encounter no errrors.

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

```
----@@ERROR system function
 insert into product values ('Mobile phone',1500,100)
 if(@@ERROR<>0)
 print 'Error Occured'
 else
 Print 'No error'
```

```
insert into product values('IPhone',1500,200)
--At this point  @@ERROR will have  non zero value
select * from product
--At this point @@ERROR gets reset  to zero, because the select statement
sucessefully  executed
If(@@ERROR <>0)
print 'Error occured'
else
 print 'No error'
```

```
Declare @Error int
insert into product values('IPhone',1500,200)
set @Error=@@ERROR
select *From  product
if(@@ERROR<>0)
print 'Error occured'
Else
print 'No Error'
```

56. Error Handling in Sql Server

Try/Catch 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 function to retrive error information:

```
ERROR_NUMBER() as ErrorNumaber,
Error_Message() as ErrorMessage,
ERROR_PROCEDURE() as ErorrProcedure,
Error_state() as ErrorState,
ERROR_SEVERITY() as ErrorSeverity,
ERROR_LINE() as ErrorLine
```

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 retruned 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 function

```
create procedure spSellProduct --2,10
@ProductId int,
@QuantityToSell int
as
Begin
--Chek the stock availabe , for the product that we want to sell
Declare @StockAvailable int
select @StockAvailable= QtyAvailabe from  product
where ProductID=@ProductId
--throw an error  to the calling application, if enought stock is not
availabe
If(@StockAvailable<@QuantityToSell)
Begin
Raiserror('Not enough stock availabe',16,1)
End
--if enough stock availabe
Else
Begin
Begin Try
Begin tran
--first reduce the quantiy available
update product set  QtyAvailabe=(QtyAvailabe-@QuantityToSell)
where ProductID=@ProductId
Declare @MaxProductSalesId int
select @MaxProductSalesId=case  when
                                        max(ProductsalesId) is null
                                        then 0
                                        else
                                        max(ProductsalesId) End
                                        from productsell
--increment @MaxProductSalesId by 1
Set @MaxProductSalesId=@MaxProductSalesId+1
insert into productsell values(@ProductId,@QuantityToSell)
Commit Tran
End try
begin Catch
Rollback tran
select
ERROR_NUMBER() as ErrorNumaber,
Error_Message() as ErrorMessage,
ERROR_PROCEDURE() as ErorrProcedure,
Error_state() as ErrorState,
ERROR_SEVERITY() as ErrorSeverity,
ERROR_LINE() as ErrorLine
End Catch

End
End
```

57. Transactions in Sql server

What is Transactions?

A transactions is a group of commands that changes 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 command 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 transaction
2. Process database commands
3. Check for errors.
   If
   error occurred,
   Rollback the transaction
   Else
   Commit the transaction

Note:-NOT able to see the un-committed changes

```
Set transaction isolation level read uncommitted

Begin transaction
Update product set QtyAvailabe=200 where ProductID=1
Commit transaction
```

```
-create two table to see the transaction demo
 create table tblPhysicalAddress
 (
 AddressId int identity(1,1),
 MeployeeNumber nvarchar(5),
 HouseNumber nvarchar(5),
 StreetAddres nvarchar(20),
 City nvarchar(20),
 PostalCode nvarchar(20)
 )

 create table tblMailingAddress
 (

 AddressId int identity(1,1),
 MeployeeNumber nvarchar(5),
 HouseNumber nvarchar(5),
 StreetAddres nvarchar(20),
 City nvarchar(20),
 PostalCode nvarchar(20)
 )
 insert into tblPhysicalAddress values('101','#10','King
Street','LONDOON','CR27DW')
  insert into tblMailingAddress values('101','#10','King
Street','LONDOON','CR27DW')
```

```
--now create stored procedure
alter procedure spUpdateAddress
as
Begin
begin try
Begin tran
update tblMailingAddress set City='London' where AddressId=1 and
MeployeeNumber='101'
update tblPhysicalAddress set City='London' where AddressId=1 and
MeployeeNumber='101'
commit tran
print 'Commit transaction'
End try
Begin Catch
Rollback tran
print 'RollBack transaction'
End Catch
End
```

58. Transaction in sql server and ACID test.

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

ATOMIC:-All the statement 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 left- half-done.

Consistent:- All data touched by the transactions is left in a logically consistent state . For example, if stock available number are decremented from product table then there have to be related entry in tblproductsales table. The inventory cannot be just disappear.

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

Durable:- Once a changes Is made , it is permanent, if system error or power Failure occurred before a set of command is complete , these commands are undone and data is restored to its original state once the system begins running again

```
--create stored procedure
create procedure spUpdateInventory_and_Sell
as
Begin
Begin try
Begin transaction
update product set QtyAvailabe=(QtyAvailabe-10) where ProductID=1
insert into productsell values(1,10)
Commit transaction
End try
Begin Catch
Rollback transaction
End Catch
End
```

59. Sub Queries in Sql server

Let us understand sub query with example

Example 1:-

Write query to retrieve product that are not at all sold?

```
Select ProductID,Name from product where  ProductID not in( select distinct
ProductId from productsell)
Or
Select productsell.productId,Name from product
left join productsell on
product.ProductID=productsell.productId where productsell.ProductID is null
```

Example 2:-

Write a query to retrieve a name and total Quantity sold?

```
select  Name,(select SUM(QuantitySold) from productsell where
ProductID=product.ProductID) as QtySold  from product
order by Name
or
Select Name,SUM(QuantitySold)  as QtySold from product
left join productsell on
product.ProductID=productsell.productId
group by Name
```

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

Sub queries are always enclosed in parenthesis and are also called as inner queries, and the query containing the sub query is called an outer query. The column form a table that is present only inside a sub query cannot be used in the select list of the outer query.

60. Correlated Subqueries

If the Subqueries depends on the outer query for its value, then that subquery called as correlated subquery.

In the where clause of the subquery below, 'ProductId' column gets its value from product table that is present in outer query

```
Select [Name],(Select SUM(QuantitySold) from productsell where
productId=product.ProductID) as TotalQuantity from product
order by Name
```

So, here the Subqueries is dependent on the outer query for it's value, subquery is correlative subquery

Correlative subquery get executed , once for every row that is select by the outer query .

Correlative subquery can not be executed independently of the outer query

61. Creating a large table with random data for performance testing

```
--61. Creating a large table with random data for performance testing
--If the table exist drop and recreate
IF (exists(select*from INFORMATION_SCHEMA.TABLES where
TABLE_NAME='tblProductSales1'))
begin
Drop table tblProductSales1
end
if (exists(select *from INFORMATION_SCHEMA.TABLES  where
TABLE_NAME='tblProduct1'))
begin
drop table tblProduct1
```

```
end


create table tblProduct1
(
Id int identity primary key,
Name nvarchar(50),
[Description] nvarchar(250)
)
create table tblProductSales1
(
Id int primary key identity,
productId int foreign key references tblProduct1(Id),
UnitPrice int,
QuantitySold int
)

--insert sample data into tblproducttable
Declare @Id int
Set @ID=1
while(@ID<=10000)
begin
insert into tblProduct1 values('product- '+CAST(@Id AS nvarchar(20)),'Product
- '+CAST(@ID as nvarchar(20))+' Description')
print @ID
set @id=@Id+1
End
select *from tblProduct1

---insert random data in tblProductSales1 table
--declare variable to hold random productId, unit price and quantity sold
declare @RandomProductId int
declare @RanmdomUnitPrice int
declare @RandomQuantitySold int

--decalre and set variable to generate a random productId  between 1 and
10000
declare @upperLimitForProductId int
declare @LowerLimitForProductId int
set @LowerLimitForProductId=1
set @upperLimitForProductId=8500
--declare and set variable to  generate  random unit price between 1 and 100
declare @upperLimitforUnitPrice int
declare @lowerLimitforUnitPrice int
set @lowerLimitforUnitPrice=1
set @upperLimitforUnitPrice=100
--declare and set variable to generatge random QuantitySold between 1 and 10
declare @upperLimitforQuantitySold int
declare @lowerLimitForQuantitySold int
set @lowerLimitForQuantitySold=1
set @upperLimitforQuantitySold=10
--insert sample data into tblproductSales1 table
declare @Counter int
set @Counter=1
while(@Counter<=15000)
Begin
```

```
select @RandomProductId=ROUND(((@upperLimitForProductId-
@LowerLimitForProductId)*Rand()+@LowerLimitForProductId),0)
select @RanmdomUnitPrice=ROUND(((@upperLimitforUnitPrice-
@lowerLimitforUnitPrice)*Rand()+@lowerLimitforUnitPrice),0)
select @RandomQuantitySold=ROUND(((@upperLimitforQuantitySold-
@lowerLimitForQuantitySold)*Rand()+@lowerLimitForQuantitySold),0)
insert into tblProductSales1
values(@RandomProductId,@RanmdomUnitPrice,@RandomQuantitySold)
print @Counter
set @Counter=@Counter+1
End
select *from tblProductSales1
```

62. What to choose for performance subquery or Join

According to MSDN, In most cases, there is usually no performance difference queries that uses Subqueries and equivalent queries that uses joins

According to MSDN, in some cases where existence must be checked, A join produces better performances. Otherwise nested query must be processed for each result for outer query, In such cases, a join approach would yield better results.

In general join work fast then sub-queries, but in reality it all depends on the execution plan that is generated by SQL server. It does not matter how we have written the query, Sql server will always transform it on an execution plan. If it is "smart" enough to generate the same plan from both queries, you will get the same result.

It would say, rater then going by theory, turn on client statistics and execution plan to see the performance of each option, and then make a decision. In later video session we will discuss about client statics and execution plan

details.

```
-here no difference time taken to execute the query
select Id,Name,[Description]  from tblProduct1
where Id in(select productId from tblProductSales1)


select distinct tblProduct1.Id,Name,Description from tblProduct1 inner join
tblProductSales1
on tblProduct1.Id=tblProductSales1.productId
--Select the product
select Id,Name,Description from tblProduct1 where not Exists(select *From
tblProductSales1 where productId=tblProduct1.Id)

select tblProduct1.Id,Name,Description from tblProduct1
left join tblProductSales1
on tblProduct1.Id=tblProductSales1.productId
where tblProductSales1.productId is null
```

63. 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 tblProductSales1 set Unit Price=50 where Product ID=101

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

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

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

Demo:-

The cursor will loop through each row in tblProductSales table. As there are 600000 rows, the 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 increases the performance. We will discuss about this in out next video session.

```
DECLARE @ProductId int
DECLARE @Name nvarchar(30)
DECLARE ProductCursor CURSOR FOR
SELECT
  Id,
  Name
FROM tblProduct1
WHERE id <= 10000
OPEN ProductCursor
FETCH NEXT FROM productCursor INTO @ProductId, @Name
WHILE (@@FETCH_STATUS = 0)
BEGIN
  PRINT 'Id= ' + CAST(@ProductId AS nvarchar(10)) + ' Name= ' + @Name
  FETCH NEXT FROM productCursor INTO @ProductId, @Name
END
CLOSE ProductCursor
DEALLOCATE ProductCursor
select*From tblProduct1
--create another cursor
DECLARE @ProductId int
DECLARE @Name nvarchar(30)
DECLARE ProductCursor CURSOR FOR
SELECT
  ProductId
FROM tblproductSales1
OPEN ProductCursor
FETCH NEXT FROM productCursor INTO @ProductId
WHILE (@@FETCH_STATUS = 0)
BEGIN

  SELECT
    @Name = Name
  FROM tblProduct1
  WHERE Id = @ProductId
  IF (@Name = 'Product-55')
  BEGIN
    UPDATE tblProductSales1
    SET UnitPrice = 55
    WHERE productId = @ProductId
```

```
  END
  ELSE
  IF (@Name = 'Product-65')
  BEGIN
    UPDATE tblProductSales1
    SET UnitPrice = 65
    WHERE productId = @ProductId
  END
  ELSE IF (@Name LIKE 'Product-100%')
  BEGIN
    UPDATE tblProductSales1
    SET UnitPrice = 1000
    WHERE productId = @ProductId
  END
FETCH NEXT FROM productCursor INTO @ProductId

END
CLOSE productCursor
DEALLOCATE ProductCursor
```

64. Replacing Cursor 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 tblProductSales1
SET UnitPrice =
               CASE
                 WHEN Name = 'Product-55' THEN 155
                 WHEN Name = 'Product-65' THEN 165
                 WHEN Name LIKE 'Product-100%' THEN 10001
               END
FROM tblProductSales1
JOIN tblProduct1
  ON tblProduct1.Id = tblProductSales1.productId
WHERE Name = 'Product-55'
OR Name = 'Product-65'
OR Name LIKE 'Product-100%'

When I executed this query, on my mechin it took less then a second. Whereas
the same thing using a cursor took 45 seconds.
Just imageing that amount of impact cursor have on performance. Cursor should
be used as your last option.
```

65. list all table in sql server database using query

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 same, there are 3 system views that we can use

1. SYSOBJECTS-sql server 2000,2005 and 2008
2. SYS.TABLES-Sql server 2005 &2008
3. INFORMATION_SCHEMA.TABLES- sql server 2005 and 2008

```
--gets the list of the tables
select *from sysobjects where xtype='U'
--gets other in database
select *from sysobjects where xtype='FN'
select *from sysobjects where xtype='V'
```

```
select distinct xtype from sysobjects

--gets the list of the tables only
select *from sys.tables

select *from sys.views
--gets the list of the tables and views
select *from INFORMATION_SCHEMA.TABLES
select *from INFORMATION_SCHEMA.VIEWS
```

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

66. Writing re-runnable sql server script

What is re