PL/SQL (or) TSQL:

Procedural Language / SQL (or) Transact SQL:

The SQL Language what we are using till now has few drawbacks

1. It doesn't provide any mechanism for step by step Execution of the code.
2. It doesn't provide conditional branching and conditional looping statement to execute the logic.
3. It doesn't provide any error handling mechanism.

To overcome the above problems SQL has been added with procedural capabilities to provide the above features and introduced as PL/SQL (or) TSQL.

PLSQL is all about programming where a program is a block of code.

PL/SQL programs are of two types.

→ Anonymous Blocks.
→ Sub-Programs.
  - Stored procedures
  - Stored functions
  - Triggers.

ANONYMOUS BLOCKS: It is a block of code which is executed at a point of time. with a session scope.

SUB-PROGRAMS: It is a named block of code that is stored on a server, and can be executed anytime from anywhere also. A sub-program can be defined in three ways.

→ Stored procedure
→ Stored function
→ Trigger.

Declaring Variables in pl/sql programs:

Syntax:- Declare @<var> [AS] &datatype> [size] [,---n]

Ex:- Declare (@) int
Declare @M Money
Declare @s varchar(50), @d DateTime.

Note:- Every variable should be prefixed with @ symbol.

Assigning values to variables:

We can Assign either a static value to a variable (or) load values from tables into variables also.

Assigning static values:-

Syntax Set @<var> = <value>
set @x = 100;

- Note: We need to set use set keyword to assign a static value to a variable.

Loading values from tables into variables:

Syntax:- Select @<var> = <colname> [,...] From <Tname>
[<clauses>]

Eg:- Select @s = Ename, @d = Hiredate. From EMP
Where Empno = 1001

Printing values of variables:

Syntax:- Print @<var>

Ex:- Print @x

Program:- Declare @x int
① Set @x = 100
Print @x

o/p - 100.

② Declare @Ename varchar(50), @sal Decimal(7,2)
Select @Ename=Ename, @sal = sal From Emp Where
Empno = 1001
Print @Ename
Print @sal

Monday
15/07/2013

CONDITIONAL STATEMENTS:

It is a block of code, that executes basing on a condition
They are of two types

1. Conditional branching
2. conditional looping.

Conditional branching

- IF ELSE IF ELSE
- SWITCHCASE (We write select case) but the behaviour is like switch.

conditional looping

- While loop.

Syntax:- If <condition>
Begin
-stmts
End
Else IF (condition>
Begin
-stmts
End
--<multiple else if blocks if required>--
Else
Begin
-stmts
End

Write a program to print the weekday on which the employee as joined.

Sol:- Declare @ Empno int
Declare @ weekDay int
Set @ Empno = 1001
Select @ weekDay = Datepart (dw, HireDate) From
Emp where Empno = @Empno
If @weekDay = 1
Print 'Sunday'
Else If @weekDay = 2
Print 'monday'
Else If @weekday = 3
Print 'Tuesday'
Else If @weekday = 4
Print 'Wednesday'

```
Else If @weekday = 5
  Print 'Thursday'
Else If @weekday = 6
  Print 'Friday'
Else @weekday = 7
  Print 'Saturday'
```

using Select case.

```
Declare @Empno Int
Declare @weekday int
Set @Empno = 1001
Select @weekDay = Datepart(dw, HireDate) From
 Emp where Empno = @Empno
 Select case @weekDay
  When 1 Then 'sunday'
  When 2 Then 'monday'
  When 3 Then 'Tuesday'
  When 4 Then 'Wednesday'
  When 5 Then 'Thursday'
  When 6 Then 'Friday'
  When 7 Then
  Else 'saturday'
```

Conditional looping:

-While loop

Syntax :- While <condition>
Begin
-Statements
End.

Every loop requires 3 things in common.

1. Initialization which sets the starting point of the loop.
2. condition which sets the ending point of the loop
3. Iteration which takes you to the next level of the cycle, either in forward direction (or) backward direction, basing on positive (or) negative Iteration.

Eg:- Declare @x Int
Set @x=0 -- Initialization
While (@x<10) -- condition
Begin
Set @x+=1 -- Iteration
Print @x
End

o/p - 1 2 3 4 5 6 7 8 9

Break statement under a loop:

We can break the execution of a loop by using a break statement

Eg:- Declare @x Int
Set @x=0
While (@x<10)
Begin
If @x=5 break
Set @x+=1
Print @x
End

o/p - 1 2 3 4

Write a program For incrementing the salary of a given Employee basing on his job.

President	10%
Manager	8%
Analayst	6%
others	5%

```
Declare @Empno Int
Declare @Job varchar(50)
Set @Empno=1005
Select @Job=Job From Emp Where Empno=@Empno
IF @Job='president'
Update Emp set sal+=sal*0.10 where
Empno=@Empno
Else IF @Job='manager'
Update Emp set sal+=sal*0.08 where
Empno=@Empno
ElseIF @Job='Analayst'
Update Emp set sal+=sal*0.06 where
Empno=@Empno
Else @Job='other'
Update Emp set sal+=sal*0.05 where
Empno=@Empno
```

16/07/13 Tuesday

CURSOR:- It is a memory location for storing the result of a query which is designed for storing resultsets that is multiple rows at a time.

We use these cursors under PL/SQL programs for processing multirow select statement i.e, a statement which gets multiple rows of data at a time for processing.

MANAGING A CURSOR: Cursor management involves of five steps.

1. Declaring a cursor: In this process we define a cursor by specifying various attributes related to the behaviour of a cursor.

Syntax:- DECLARE cursor_name CURSOR
[LOCAL | GLOBAL]
[FORWARD_ONLY | SCROLL]
[STATIC | DYNAMIC | FASTFORWARD]
FOR <select statement>

2. OPENING A CURSOR: When we open a cursor it will internally execute the select statement that is associated with the cursor declaration and loads data into the cursor.

Syntax:- open cursor_name

After loading data into a cursor it will also provide a pointer for accessing the data from the cursor.

3. FETCHING DATA FROM THE CURSOR: In this process, we access row by row from the cursor for processing of the data

Syntax:- fetch first | last | prior | next | absolute n | relative n from cursor_name into <variables>

Using an appropriate fetch method we can move to the desired location and once we move to an appropriate location to identify the fetch statement is successful or not we can make use of an implicit variable that is @@fetch_status

@@fetch_status is an implicit attribute which doesn't required to be declared. It returns the status of last cursor fetch statement which can be any of the following values

0	The fetch statement was successful
-1	The fetch statement failed or the row was beyond the result set
-2	The row fetched is missing.

4. CLOSING A CURSOR: In this process, it releases the current result set of the cursor leaving the datastructure available for re-opening.

Syntax: close cursor_name

5. DEALLOCATING A CURSOR: In this process, it removes the cursor reference and deallocates it by destroying the data structure.

Syntax: deallocate cursor_name

```
While @@Fetch_status=0
 Begin
 IF @Job = 'president'
 update Emp Set sal += sal * 0.08 Where
        Empno =@Empno
 Else If @Job = 'Manager'
 Update Emp set sal += sal * 0.08 Where
        Empno = @Empno
 Else If @Job = 'Analyst'
 update Emp set sal += sal * 0.06 Where
        Empno = @Empno
 Else
 update Emp set sal += sal * 0.05 Where
        Empno = @Empno
 Fetch Next from Empcur into @Empno, @Job
 End
 Close Empcur
 Deallocate Empcur
```

LOCAL / GLOBAL CURSORS: IF a cursor is declared as Local the scope of the cursor is only "within the program" under which the cursor is declared. Once the program execution is completed we cannot use that cursor anymore. Even if it is not deallocated.

If a cursor is declared as Global under a program, we can use that cursor in other programs also "within the connection" without declaring again. When we want a cursor as global the program should not contain a deallocate statement in it. The cursor gets deallocated once the connection is closed.

```
Declare @Empno Int
Declare Empcur cursor Global for Select Empno From Emp
open Empcur
Fetch Next From Empcur Into @Empno
While @@Fetch_Status = 0
Begin
Print @Empno
Fetch Next from Empcur into @Empno
End
Close Empcur
```

→ Write the above program by opening a New Query window (New connection)

After executing the program for the first time, comment the declare cursor statement and execute again. still the program executes because the cursor is declared as "Global"

→ Now write the same code in another query window change Global as Local and run the program. After completing the Execution comment the Declare cursor statement and Exe run the program again. Where we will get an "error" as the cursor is "Local"

STATIC / DYNAMIC: If a cursor is declared as static after opening the cursor any modifications that are performed to the data in the table will not be reflected into a cursor. So the cursor Contains old values only in it.

```
Declare @Sal Money
Declare Empcur cursor static for Select sal From Emp
Where Empno = 1001
```

```
Open Empcur
Update Emp set sal=10000 where Empno=1001
Fetch Next from Empcur into @sal
Print @sal
Close Empcur
Deallocate Empcur
```

Before Executing the above program, verify the salary of Employee 1001 and then execute the program. Even if, the Program is updating the salary in the table the fetch statement will still display as the old value of the table only but not the new value.

If we want the changes made on the table to be reflected into the cursor after opening the cursor declare the cursor as dynamic

FORWARD ONLY / SCROLL CURSORS:

If a cursor is declared as forward only it allows you to navigate only to the next records in a sequential order and moreover it supports only a single fetch method that is fetch Next where as a scroll cursor allows you to navigate bidirectionally that is top - bottom (or) bottom - top also. and it supports six different fetch methods.

Fetch Next / Fetch prior, Fetch First, Fetch last, Fetch Absolute #, Fetch Relative #

Eg:-
```
Declare @Empno int
Declare Empcur cursor Scroll for select Empno
From Emp
Open Empcur
```

```
Fetch Next from Empcur into @Empno                    o/p
Print @Empno                                          1001
Fetch last from Empcur into @Empno
Print @Empno                                          1015
Fetch prior from Empcur into @Empno
Print @Empno                                          1014
Fetch Absolute 5 from Empcur into @Empno
Print @Empno                                          1005
Fetch Relative -2 from Empcur into @Empno
Print @Empno                                          1008
Fetch first from Empcur into @Empno
Print @Empno                                          1001
close Empcur
Deallocate Empcur
```

SUB-PROGRAMS:

It is a named block of code that is directly saved on the server and can be executed when and where it is required under databases a subprogram is refered as a "stored Procedures" or a "Stored function".

STORED PROCEDURES:

It's a block of code designed to perform an action when called.

Syntax:- Create [Alter] procedure <name> [(
@<param> <datype> [size] - [default] [out|output],
...n)]
[With <procedure attributes>]

```
As
  Begin
    -stmts
  End
```

A procedure is very similar to a function in our C, C++ Languages (or) a method in Java or .Net Languages.

A procedure definition contains two parts in it

1. Procedure header
2. procedure body.

Ex:- The content above AS is known as "procedure header" and the content below AS is known as "procedure body"

If required we can pass parameters to a procedure to make the procedures more dynamic

Calling a stored procedure.

Syntax:- Exec <procedure name> [<list of param values>]

Ex1:-
```
Create procedure Test1
    As
     Begin
       Print 'My First stored procedure'
     End
```

Once the procedure is created, on database it is physically saved on the server as a "database object" which can be called from anywhere connecting to the server.

Wed 17/07/2013

STORED PROCEDURES

Calling the above procedure :-

We can call the above procedure from anywhere that is in a new query window (or) from any application that is developed using JAVA (or) .NET Languages also.

→ Calling the procedure in query window

```
Exec Test1
```

o/p - My First Stored procedure

Procedure With Parameters:

Ex:-

```
Create Procedure Test2 (@x int, @y int)
As
Begin
Declare @z int
Set @z = @x + @y
Print 'sum of the 2 no's is :' + cast(@z varchar)
End.
```

Calling procedure with Parameters

```
Exec Test2 100, 50
```

o/p - Sum of the 2 no's is : 150

(or)

```
Exec Test2. @x = 100, @y = 50. (or)
Exec Test2 @y = 50, @x = 100.
```

Defining a procedure With default values to Parameters:

```
Create Procedure Test3 (@x int=100, @y int)
As
  Begin
  Declare @z int
  set @z = @x + @y
  Print 'sum of the anos is :' + cast (@z as varchar)
  End
```

If a default value is given to a parameter of a Procedure, while calling the procedure passing values to that parameter is only optional.

→ Calling the above procedure

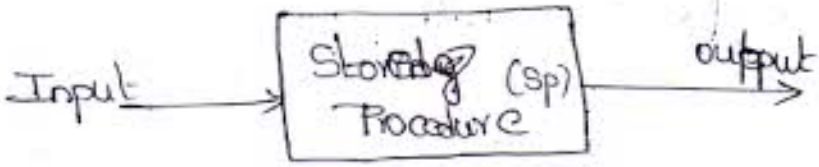
```
Exec Test3  200, 250          o/p
                              450
Exec Test3  default, 150      250
Exec Test3  @x=default, @y=300  400
Exec Test3  @y=250            350
```

A PROCEDURE WITH OUTPUT PARAMETERS:

Parameters of a procedure can be of two types

1. Input parameters
2. ouput parameters

Input parameters are used for bringing a value into the procedure for execution where as output parameters are used for carrying a value out of the procedure after Execution.



Ex:- Create procedure Test4 (@a int, @b int, @c int out,
@d int output)

As
Begin
Set @c = @a + @b
Set @d = @a * @b
End

If a parameter is declared as output we only require to assign a value to the parameter inside the procedure. So that, Procedure will send that value out in the end of procedures Execution.

An output parameter can be declared either by using out / output keyword also.

→ Calling the above procedure

Declare @x int, @y int
Exec Test4 200, 50, @x out; @y output
Print @x
Print @y

Note:- While calling a procedure with output parameters we need to declare variables first and substitute in the place of the parameter list, so that the result comes and sits in those variable, but here also we need use out/output keywords.

18/01/2013 – Thursday

Sp- Help Text: It is a predefined system procedure using which we can view the content of a stored Procedure Sub-program.

Ex:- Sp_Help Text Test4

Note:- Whenever a procedure is created the content of the procedure is saved under the syscomments table just like views are saved.

Ex:- Select * From Syscomments Where object Name (Id) = 'Testy'

In this table we have a column text under which the complete create procedure statement gets saved. The Sp_Help text System procedure will retrive the data from the syscomments table that is the text columns data and displays it.

PROCEDURE ATTRIBUTES: There are two types of attributes

1. With Encryption
2. With Recompile

1. With Encryption: If this attribute is used on the Procedure the text of this procedure is encrypted and will not be shown to anyone in the text column of the syscomments table. So, noone will be having an option to view the content of it.

To test this do the following.

Alter procedure Test4 (@a int, @b int, @c int out, @d int output)

With Encryption
As
Begin
Set @c = @a + @b
Set @d = @a * @b
End

After Altering the procedure again go and verify the text from the syscomments table (or) by using sp-Helptext which will not show the code of the procedure

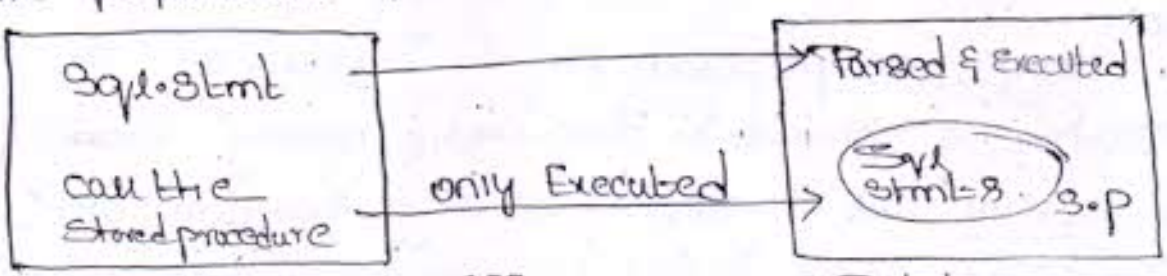
Sp-Helptext Testy

Note:- When an Application is developed for a client at the time of installing this application on client system we will be using the with encryption option on all the views, procedures, functions, triggers etc and install on the clients machine. so that they will not be having the chance viewing the source code or Altering the source code.

→ Why do We create Procedures?

Sol:- Whenever we want to execute an sql statement from an application the sql statement what we send from an application will be first parsed (compiled) for execution Where the process of parsing is time Consuming because parsing occurs each and every time we execute the statement.

To overcome the above problem we write sql statements under a stored procedure and execute, because a stored Procedure is a Precompiled block of code without Parsing the statement gets executed whenever the Procedures are called which can drastically increase the performance of database Server.



2. With Recompile Attribute :- Whenever a procedure is compiled for the first time it prepares a best query plan according to the current state of database and executes the query plan when the procedure is called

The compilation of the procedure and preparing a query plan is performed not only at the time of Procedure creation but each and everytime the server is restarted (Implicitly occurs)

With If the procedure is created by using the with Recompile procedure Attribute it is forced to be compiled each time it is executed and whenever it is compiled it re-prepares the query plan.

Forcing a procedure for recompilation and preparing a query plan is required when the database undergoes significant changes to its data or structure.

Another reason to force a procedure to recompile is if at all the table is added with new indexes from which the procedure might be benefited forcing for recompilation is very important Because we cannot wait until the server is restarted for preparing a new query plan.

Note:- Even if the with Recompile option is available it is not suggested to be used if at all there are no significant changes in the structure of the database.

Note:-
A query plan prepared for the execution of a Procedure is stored in the "cache memory". So, it is not physical that is the reason why everytime we restart the server it will Recompile and prepares the query plan.

Write a procedure for inserting a record into the Employee table when the name, Job, salary and Deptno are given by generating a unique Employeeno and returning it back.

```
Create procedure Emp_Insert(@Ename varchar(50), @Job
    varchar(50), @Sal Money, @Deptno Int, @Empno int output)
AS
  Begin
   Set No count on
   Begin Transaction
    Select @Empno = Max(Empno) + 1 From Emp
    Insert into Emp (Empno, Ename, Job, Sal, Deptno)
    values (@Empno, @Ename, @Job, @Sal, @Deptno)
    Commit Transaction
    End
```

Calling the above procedure
(or)
Executing

```
Declare @Empno Int
 Exec Emp_Insert 'xyz', 'Manager', 4500, 40, @Empno out
 Print @Empno
```

19/07/2013 Friday

Rewrite the above code.

```
Select @Empno = ISNULL(Max(Empno), 1000) + 1
From Emp
```

Write a procedure which takes Employee number as a Parameter and returns profident fund at 12% and professional tax at 3% which should be detected from the salary.

```
Create Procedure deductions (@Empno int, @PF Money out,
    @Pt Money out)
As
 Begin
  Select
  Declare @sal Money.
  Select @sal = Select From Emp Where Empno = @Empno
   Set @PF = @sal * 0.12
   Set @pt = @sal * 0.03
  End
```

Exec

```
Declare @vpf Money, @vpt Money
Exec Deductions 1001, @vpf out, @vpt out
Print @vpf
Print @vpt
```

Write a procedure which takes the Empno and prints the next salary of employee.

```
Create procedure Emp_next sal (@ Empno int) @sal
 Money)
As
 Begin
  Declare @sal Money, @Nsal Money, @PF money, @pt money
  Select @sal = sal From Emp Where Empno = @empno
   Set
```

```
Exec Deductions @Empno, @PF out, @pt out
    Set @Nsal = (@sal - (@PF + @pt)
    Print 'Net salary of the Employee is:' + cast (@Nsal as
    varchar).
    End
```

Executing

```
Exec Emp_Netsal 1001 .
```

A procedure for dividing two numbers

```
Create procedure Divide (@x int, @y int)
AS
 Begin
 Declare @z int
  Set @z = 0
  Set @z = @x / @y
 Print 'The Result is:' + cast (@z as varchar)
  End.
```

Execution

```
    Exec Divide 100, 5
    Exec Divide 100, 0
```

In the above case, when we execute the procedure we will get the Error at the second Execution. Because the divisor value is zero where a number cannot be divided by zero as per the rule of mathematics.

In SQL Server whenever an error occurs at a line of code it will first print the error message and then continues with the execution. So, in the above case, when the error occured in the procedure the result will be as following.

Msg 8134, Level 16, State 1, Procedure Divide, Line 6
Divide by zero Error encountered.

The Result is: 0

The problem in the above execution is Even if the Error occured in the program, it is still showing the Result, so there are chances of users being confused

Note:- Whenever any error occured while executing a program developed by using any programming Language, the program terminates abnormally on the line where the error got occured, but under SQL server it will still continue the Programs Execution.

In the above case, both the behaviours are wrong because when errors occur in a programming language it will skip the execution of all the statements after the error. Even if, those statements are not related with the error. Where as. In SQL Server because the execution will not stop when the error occured statements related with the error also will be executed, but it should not happen.

For Example in our above procedure, when the error got occured it is still displaying the "Result is: 0" which should not be displayed.

HANDLING ERRORS: We handle errors of a program Both in a programming language as well as Databases also. Where as handling an error in a programming Language needs stopping the abnormal termination and allowing the statements which are not related with the error to execute, where as handling an error in SQLServer means stopping the execution of statements which are related with the Error.

HANDLING ERRORS IN SQLSERVER: From SQLServer 2005 we are provided with a structure Error handling mechanism with the help of try and Catch blocks which should be used as following.

Begin Try

- Stmts which will cause the error.
- Stmts which are related with the error that shall not execute when error occured.

End Try

Begin catch

- Stmts which should be executed whenever the error occurs.

End catch.

To overcome the problem we faced in the Divide Procedure Rewrite the procedure as following.

Sol: TB

```
Alter procedure Divide(@x int, @y int)
As
 Begin
  Declare @z int
   Set @z=0
   Set @z=
   Begin Try
    Set @z = @x/@y
Print 'The Result:' + cast(@z as varchar)
 End Try
 Begin catch
  Print 'Divisor should not be zero'
 End catch
  End
```

→ Skip the execution

Re-execute the program again as Following.

```
Exec Divide 100, 50
Exec Divide 100, 0
```

When we execute with correct values any error will not occur in the program. So after executing all the statements in the try block control directly jumps to the statement present after catch block without executing catch block.

If any error occurs in the execution process in such case from the line where the error got occured in Try control directly jumps to catch block. So rest of the statement in Try will not execute. Where as, Catch block will execute.

Note:- In our above program, when the error got occured we are displaying an Error message "Divisor cannot be Shard zero", In place of that Error message we can also display the original error message associated with that data. By calling a function "Error_Message". To test this alter the above procedure by rewritting the code inside the catch block as following.

```
Print Error_Message()
```

20/07/2013 - Saturday

PREDEFINED ERRORS: Whenever a Error occurs under a Program Like dividing a number by zero violation of Primary key, Violation of check constraint etc. The system displays an Error message telling us the problem encountered in the code.

Every Error that occurs in the program is associated with four attributes.

1. Error Number
2. Error Message
3. Severity Level
4. State

Msg 8134 (Error Number), Level 16 (Severity Level), State 1 (State), Divide by zero error encountered (Error Message)

1. ERROR NUMBER:
Error Number is a unique identification given for each and Every Error that occurs in the program

This value will be below 50,000 for pre-defined Errors and must be above 50,000 for the Error is user defined.

2. ERROR MESSAGE:

It is a brief information describing about the error occured which should be Max from 2047 charanters.

3. SEVERITY LEVEL:

This tells about the importanke of the Error which can be ranging between 0 to 25. In which,

0 to 9 are not severe which can be considered as Information (or) status Messages

11 to 16 Indicates these Errors can be corrected by the user.

17 to 19 Indicates these are Software Errors cannot be corrected by the user. must be reported to System Administrator

20 to 24 Indicates Fatal Errors and if these Errors occur they can damage the system (or) database. So here the connection immediately terminates with the database.

4. STATE:

It is an arbitrary value which is not that important can be ranging between 0 to 127. We use this whenever the same Error has to occur in multiple places.

Note:- We can find the information of all predefined errors under the table "Sys Messages"

```
Select * From SysMessage Where
  MsgLangID = 1033
```

* 1033 represents English Language.

→ Write a procedure for inserting a record into dept table when we pass the deptno, dname and Location as Parameters but inside the procedure along with inserting the record into dept table a matching record should also be inserted into deptdetails table. By generating a unique Did and also comment.

Note:- Make sure in the procedure that all the two statements will be successfully executed (or) none of the statements execute.

```
Create procedure Dept_Insert(@Deptno int,@Dname varchar(50),
  @Loc varchar(50))
AS
  Begin Set Nocount ON
  Declare @Did Int
  Declare @Comments varchar(1000)
  Select @Did = IsNull(Max(Did),0)+1 From
  Dept Details
  Set @comments = 'This department is Located in'
    +@Loc+' and mainly involved in'+@Dname
  Begin Try
  Begin Transaction
    Insert Into Dept values (@Deptno, @Dname, @Loc)
    Insert Into Dept-Details values (@Did, @Deptno, @comments)
  Commit Transaction.
```

```
End Try
Begin Catch
Rollback Transaction
Print Error_Message()
End catch
End.
```

Calling procedure

```
Exec Dept 50, 'operation', 'pune'
```

22/07/13 Monday

Raising Errors Explicitly in a program:

Generally Errors will raise in a program on predefined reasons like dividing a number by zero, violation of Primary key, violation of check, violation of referential integrity etc.

Where as if required we can also raise an error in our programs in two different ways.

1. Using Raiserror Statement
2. Using throw statement (new Feature of Sql Server 2012)

Syntax:

```
Raiserror (errorid |errormsg, severity, state) [with log]

throw errorid, errormsg, state
```

Difference between Raiserror function and Throw Statement

Ans: If we use any of these two statements in a program for raising an error without try and catch blocks, raiserror statement after raising the error will still continue

the execution of the program where as throw statement will terminate the program abnormally on that line. But if they are used under try block both will behave in the same way that is will jump directly to catch block from where the error got raised.

Raiserror Statement will give an option of specifying Severity of the error message where as we dont have these option in case of throw statement where all error meseages will have a default severity of 16

In case of Raiserror there is chance of recording the Error messages into the sever log file by using the With log option Where as we cannot do in case of throw.

In case of throw you need to specify both errorid and errormessage to raise the error where as in case of raiserror we can specify either id (or) meg. If id is not specified default errorid is 50,000 but if we want to specify only Errorid first we need to add the error message in the sys messages table by specifying a unique id to the message.

Write a procedure for dividing two numbers and raise an error in the program if the divisor is 1. by using raiserror statement.

```
Create procedure DivByone1 (@x int, @y int)
AS
 Begin
  Declare @z int
  Set @z = 0
  Begin Try
  if(@y=1
  Raiserror('Divisor can't be one', 16, 1)
  Set @z = @x/@y
  Print 'The result is :' +cast( @z as varchar)
  End Try
  Begin catch
   Print Error_number()
   Print Error_message()
   Print Error_Severity()
   Print Error_State()
  End catch
 End
```

(Executing)

```
Execute DivByone1 100, 1
```

The above procedure can also be defined with the help of a throw statement with in place of raiserror as following.

```
Create procedure DivByone2 (@x int, @y int)
 AS
 Begin
```

```
Declare @z int
Set @z = 0
Begin Try
if @y = 1
throw 60001, 'Divisor can't be one', 1
Set @z = @x/@y
Print 'The result is :' + Cast(@z as varchar)
End Try
Begin Catch
Print Error_number()
Print Error_message()
Print Error_Severity()
Print Error_state()
End catch
End.
```

(Execution)

```
Exec DivByone2 100, 1
```

Additional options With Raiserror statement.

1. With Log : By using this option in the raiserror statement we can record the error message in the Sql server log file so that if the errors are fatal database Administrator can take care of fixing those Errors.

Note:- If the severity of the error is greater than 20 specifying the with log option is Mandatory.

To test this Alter the procedure DivByone1 By changing the raiserror statement as following

Raiserror ('Divisor can't be one', 16, 1) with log

Now execute the procedure and whenever the given error raises we can watch the error messages recorded under sqlserver Logfile.

To view the Logfile open

Object Explorer → Go to Management node → sqlserverLogs node → open the current Logfile by double clicking on it.

Using substitutional parameters in the errormessage of RaisError.

Just like c Language we can substitute values into the error message to make the errormessages as dynamic as following.

Raiserror ('The number %d can't be divided by %d', 16, 1, @x, @y) With Log.

Raising errors by storing the errormessage in Sysmessages table

We can Raise an error without giving the error message in Raiserror statement but in place of Error message we need to specify the errorid and to specify the errorid first we need to record that errorid with Errormessage in sysmessages

table by using "sp_addmessage" system stored procedure

sp_addmessage <errorid>, <severity>, <errormsg>

To test this first add a record a sysmessages table as following.

sp_addmessage 51000, 16, 'Divide by one Error encountered'

Now Alter the Procedure DivByone1 by changing the raiserror statement as following.

Raiserror (51000, 16, 1) With log

→ Deleting our Errormessages From sysmessages

~~sp_Help~~

sp_DropMessage <errorid>

sp_Dropmessage 51000

→ Write a procedure For transfering the funds from one account to the another account and make sure of the transaction management in the procedure as well as raising an Error if at all the amount being transfered Plus 1000 (Minimum Balance) is greater than the Balance of the customer.

Ans:- Create Procedure customer_Funds (@srcId int, @destid int, @Amount Decimal(9,2))

As

Begin

Declare @count1 int @count2 int

```
Declare @Balance Decimal(7,2)
Select @Balance = Balance from customer where
custid = @srcId and status=1
Begin Try
IF ((@tAmount +1000) > @Balance
Begin
--throw 50001, 'In sufficient Funds', 1
Rais error('In sufficient Funds', 16, 1)
End
Begin Transaction
update customer set Balance -= @tAmount where
custid = @srcId and status=1
Set @count1 = @@Rowcount
Update customer set Balance += @tAmount where
custid = @DestId and status=1
Set @count2 = @@Rowcount
IF @count1 = @count2
Commit Transaction
Else
Rollback Transaction
End Try
Begin catch
Print Error_message()
End catch
End
```

Structure of table

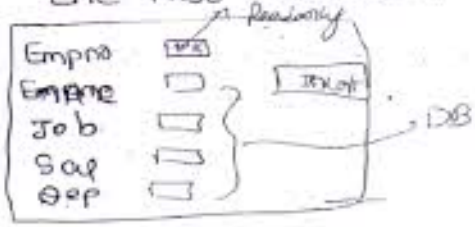
Primary key

custid	cname	Balance	Status
101	Zane	15000.00	1

Bit

23/07/2013 - Tuesday

@@Rowcount: It is an implicit variable which returns the number of rows affected by the last executed DML statement.



FUNCTIONS:

It is also a sub-program like a stored procedure which is defined for performing an action such as a compler calculation and returns of result of the action as a value.

Differences between function and a procedure

* A function must return a value where as procedure never returns a value
* A procedure can have parameters of both input and output where as a function can have only input parameters.
* In a procedure we can perform Select, insert, update and Delete operations. Where as function can be used only to perform select. cannot be used to perform insert, update and delete operations that can change the state of the database

* Procedures provides the option for to perform transaction management, error handling etc where as these operations are not permitted in a function

* We call a procedure using Execute command so we cannot embedd (or) use them in a select statement where as Functions are called by using select command only.

* From a procedure we can call another procedure (or) a Function also where as from a function we can call another function but not a procedure.

Functions are of three types in sql server

1. Scalar function
2. Inline Table valued function
3. Multi-statement Table valued function

SCALAR FUNCTIONS: These functions returns a single value as a output

Syntax:- Create/Alter function <Name>(

@<param> <dtype>[size][=default], ---- n)

Returns <scalar type>

[with <function Attributes>]

AS

Begin

<function body>

Return <scalar expression>

End

Q. Write a function that takes an employee number and returns the net salary of that employee.

Ans:
```
Create function Emp_GetNetSal (@Empno int)
Returns Money
As
Begin
Declare @Sal Money, @Comm Money
Declare @PF money, @Pt Money, @NSal Money
Select @Sal = Sal, @Comm = comm From Emp Where
 Empno = @Empno
Set @PF = @Sal * 0.12
 Set @Pt = @Sal * 0.03
 Set @NSal = @Sal - (@PF + @Pt) + IsNull (@comm, 0)
 Return @NSal
 End
```

Calling a Scalar Function:

Syntax:- Select <owner>.<Function> (<list of values>)

Ex:- Select Dbo.Emp_GetNetSal (1005)

```
Select Empno, Sal, Comm, Dbo.Emp_GetNetSal (Empno)
As NetSal From Emp.
```

TABLE VALUED FUNCTIONS: In this case we can return a table as an output from the function. These are again of two types.

1. Inline table value
2. Multi-statement table value:

INLINE TABLE VALUED FUNCTIONS:

In this case the body of the function will have only a single select statement prefixed with "Return" other than that it cannot contain any content

Syntax: Create | Alter function <Name>(
@<param> <dtype> [size] = [default], ----n)
Returns Table
[With <Function attributes>]
AS
Return (<select stmt>)

Q.
⇒ Write a Function that gets the data of the Employees working in a department when we specify the department name where the data should come from Emp, Dept and salgrade tables

```
Create Function Get_EmpDetails (@Dname Varchar
(50)) Returns Table
AS
Return ( select E.Empno, E.Ename, E.Job, E.sal,
S.Grade, D.Deptno, D.Dname, D.Loc From
Emp E Inner Join Dept D on E.Deptno = D.Deptno
Inner Join salgrade S on E.Sal Between S.Losal
And S.Hisal where D.Dname = @Dname)
```

Calling a Table valued Functions

Select <colist> From <owner>.<Function>(list of values)

```
Select * From dbo.Get_EmpDetails ('sales')
(or)
```

Select Empno, Ename, Sal, Grade, Deptno, Dname
From Dbo.Get_EmpDetails ('Marketing')

MULTI-STATEMENT TABLE VALUED FUNCTION: This is same as the above which can return a table as an output but here the body can contain more than one statement and also the structure of the table being ~~Return~~ Returned ~~can be defined~~ can be defined by us.

Syntax:-

```
Create / Alter Function <Name>(
@<Param> <dtype> [Size] [= default], ... n)
Returns @<table var> Table (<column def's>)
[With <Function Attributes>]
AS
 Begin
 <Function body>
  Return
  End
```

Note:- In case of multi-statement table valued function we need to define our own structure to the table being return.

24/07/13 - Wed

Write a function which returns a table with the salary details of all the employees like empno, sal, Comm, PF, pt and Netsal.

Sol:- Create Function Emp_GetSalDetails()
Returns @MyTable Table(Empno int, salMoney,
comm Money, PF money, pt money, Netsal Money)

```
As
Begin
Declare @Empno int
Declare @sal Money, @comm money
Declare @PF money, @pt money, @Nsal Money
Declare Ecur Cursor For Select Empno, sal, Comm
From Emp
open Emp Cur
Fetch Next From Ecur into @Empno, @sal, @comm
While @@Fetch_status = 0
Begin
  Set @PF = @sal * 0.12
  Set @pt = @sal * 0.03
  Set @Nsal = @sal - (@PF + @pt) + IS NULL(@comm, 0)
  Insert into @MyTable values (@Empno, @sal, @comm
  @PF, @pt, @Nsal)
  Fetch Next From Ecur Into @Empno, @sal, @comm
  End
  Close Ecur
  Deallocate Ecur
End Return
End.
```

FUNCTION ATTRIBUTES:

While creating a Function we can define Attributes to the function like.

1. With encryption
2. With SchemaBinding.

TRIGGERS

A Trigger is a special type of stored procedure that Automatically executes when an event occurs in the database server without being explicitly called.

Triggers are of two types

1. DML Triggers
2. DDL Triggers (Introduced in sql server 2005)

DML Triggers:

DML Trigger Execute when the user tries to modify data through data manipulation Language Event. Those are Insert, update and delete statements on a table or view These triggers fire when any valid event is Fired regardless of whether or not any table rows are affected in the table.

Note:- Insert, update and delete statement are also known as Triggering sql statements because those three are responsible for the trigger to Fire.

DML Triggers can be used to enforce business rules and data integrity.

DML Triggers are similar to constraints in the way they can enforce integrity.

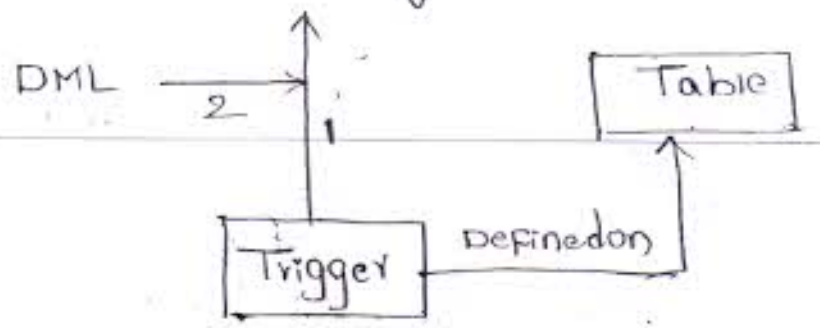
With the help of a DML Trigger we can enforce integrity which cannot be done with Constraints that is comparing values with values of another table etc..

Syntax:- Create | Alter Trigger <Name>
on <TableName> | <viewName>
[With <Trigger attributes>]
For | After | Instead of
[Insert, update, Delete]
AS
Begin
- Trigger Body
End

As per the specifications of Sql we have three different types of triggers

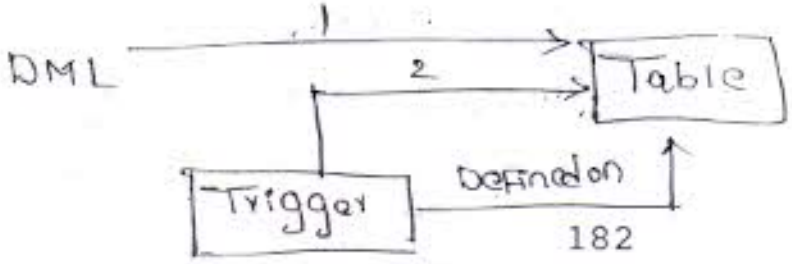
1. Before trigger
2. After trigger
3. Instead of trigger

1. Before trigger: This trigger fires before the Execution of Triggering Sql statement



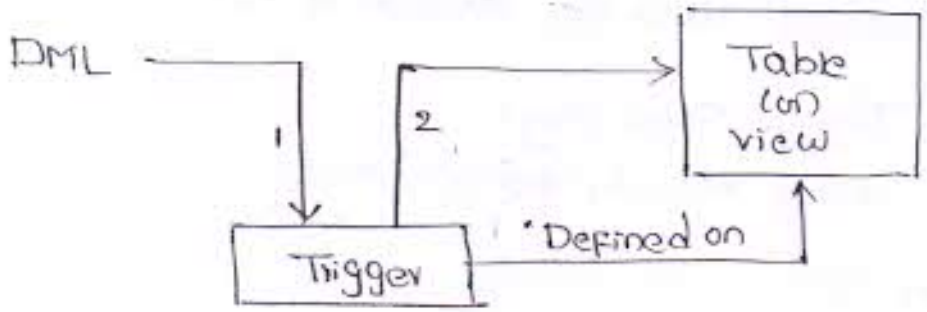
[verification is performed based on movies tickets in cinema hall)

2. After trigger: These trigger fires after the execution of Triggering sql statement.



[verification is performed in the train ticket=8 (T.T.R)]

Instead of Trigger: In this case the triggering Sql statement is only responsible in calling the trigger but will not reach the table directly. Now on behalf of the triggering sql statement trigger performs operation on the table (or) view.



In Sql server database we have only After and Instead of triggers but not before triggers.

In other databases like oracle we have all the three triggers but instead of triggers can be defined on "Views" only. Where as in sql server instead of trigger can be defined on "tables" also which will give the same behaviour like a before trigger.

While defining a trigger on Sql server on tablename or view name in the syntax refers to the table or view on which we are defining the trigger.

For/After Specifies that the trigger fires only after trigger sql statements is executed.

Note:- After triggers cannot be defined on views.

Instead of is to specify the trigger is executed on behalf of the triggering sql statement.

Insert, update, Delete is to specify which statement will activate the trigger and we need to use atleast one option or combination of option also can be used.

Define a trigger on Emp table which will restrict DML operations on the table apart from business hours where business hours are between 9 to 5.

Ans:-
```
Create Trigger Emp_Trg1
on Emp After Insert, update, Delete
AS
 Begin
 Declare @Hours int
 Set @Hours = Datepart (HH, GetDate())
 IF @Hours Not Between 9 to 16
  Begin
  Rollback Transaction
  Raiserror ('Transaction can't be performed
            now', 16, 1)
  End
 End
```

Try to perform any DML operations on the table Apart from the business hours immediately the operation is rolled back by displaying an error message.

Note:- Whenever we perform a DML operation on any table which is defined with any trigger an implicit transaction gets started before the execution of DML statements which can be roned back inside the trigger if required. if not rolled back in the

end of Triggers execution Transaction is committed.

25/07/2013 Thursday

Write a Trigger on the Dept table so that it will convert the department name and Locations values into uppercase in whatever case they are entered while inserting.

Sol:- Create Trigger Dept_Trg1
on Dept After Insert
AS
Begin
Declare @Deptno int, @Dname varchar(50), @Loc varchar(50)
Select @Deptno = Deptno, @Dname = Dname, @Loc = Loc
From Inserted
update Dept set Dname = upper(@Dname), Loc = upper(@Loc) where Deptno = @Deptno
End

After creating the trigger test the trigger by inserting Record as following

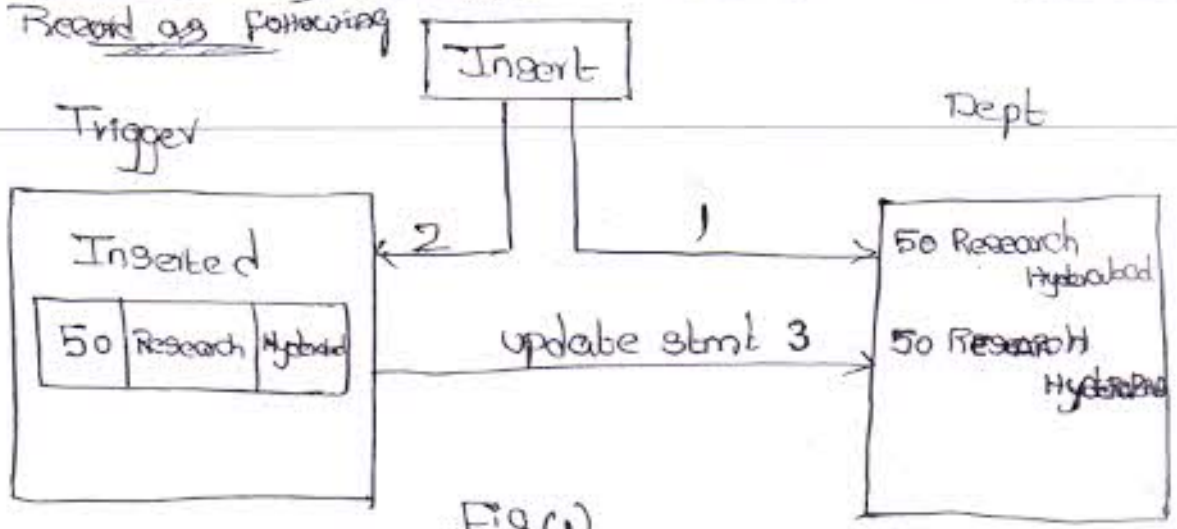


Fig (a)

After creating the trigger test the trigger by inserting Record as following

→ Insert Into Dept values (50, 'Research', 'Hyderabad')
Select * From Dept

Whenever a trigger fires because of an insert statements, the values that are being inserted in the table will be captured in the trigger under a table known as inserted using which we can find out what values are being provided by the user for inserting into the table.

In the above case our trigger is a after trigger so first the insert statement executes and values will be inserted into table then the trigger executes so that first the values are captured into the inserted table and then from the trigger we are updating the inserted values of the table. (Show the Fig)

Q.
Write a trigger on the Dept table which generates a unique deptno and insert into the table when the users insert a record without specifying the Deptno.

Ans:- Create Trigger Dept_Trg2
on Dept Instead of Insert
as
Begin
Declare @Deptno int, @Dname varchar(50) @Loc varchar(50)
Select @Deptno = Deptno, @Dname = Dname, @Loc = Loc From
Inserted.
If @Deptno IS NULL
Begin
Select @Deptno = IS NULL (MAX (Deptno), 0) + 10 From Dept
End.

```
Insert into Dept values(@Deptno, @Dname, @Loc)
End
```

Execute the at

Test the above Trigger by Inserting the following records

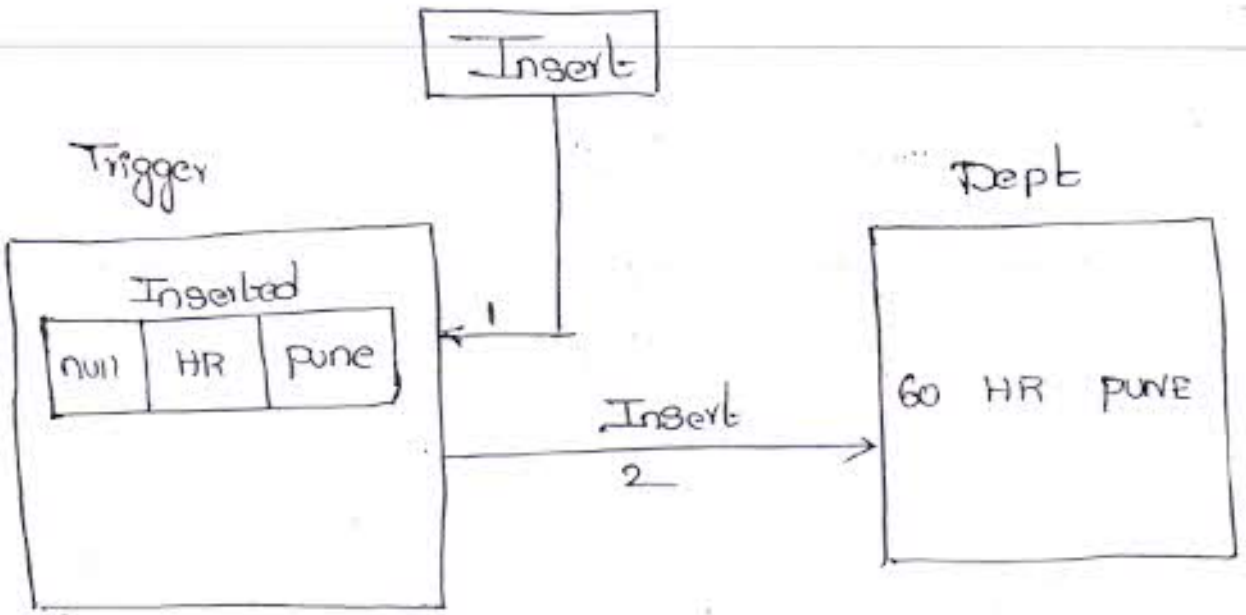
Insert into Dept (Dname, Loc) values ('HR', 'pune')

- In this case a new Deptno is generated

Insert into Dept values (65, 'Aaa', 'Bbb')

- In this case new Deptno is not generated b'coz the user has given it.

In the above case, we have defined a instead of trigger so here first the insert statement will call the trigger and submits the value being inserted then the trigger is generating a deptno if not existing and finally insert the values into the table.



Note:- In the above case after the insert statement is inserting & Executed from the trigger immediately our previous trigger Dept_trg1 will also fire and Converts the data into uppercase.

Write a trigger on the Emp table which fires when we Perform a Delete operation on the table and restrict those operations if the job of the Employee is President

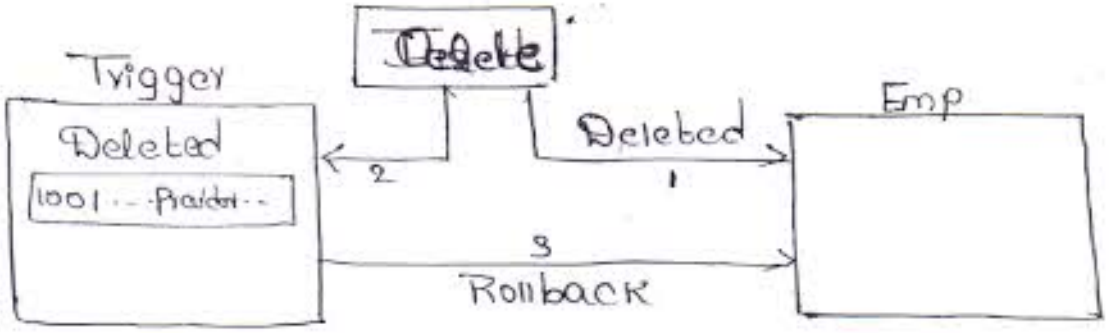
Sol:- Create Trigger Emp_Delete_Trg
on Emp After Delete
As
Begin
Declare @Job varchar(50)
Select @Job=Job From Deleted
If @Job='president'
Begin
Rollback Transaction.
Raiserror ('can't delete president's Info.', 16,1)
End
End

By depault our Employee table as a self referential Integrity constraint so even the trigger is defined on a table constraint comes into picture restricting the delete operation because it has child records so to test the trigger drop the constraint.

Alter Table Emp Drop Constraint Mgr_Ref

Now test the trigger by executing the following statement

Delete From Emp Where Empno=1001



Whenever we perform a Delete operation on a table the record being deleted from the table is now captured under the trigger within a Magic table deleted.

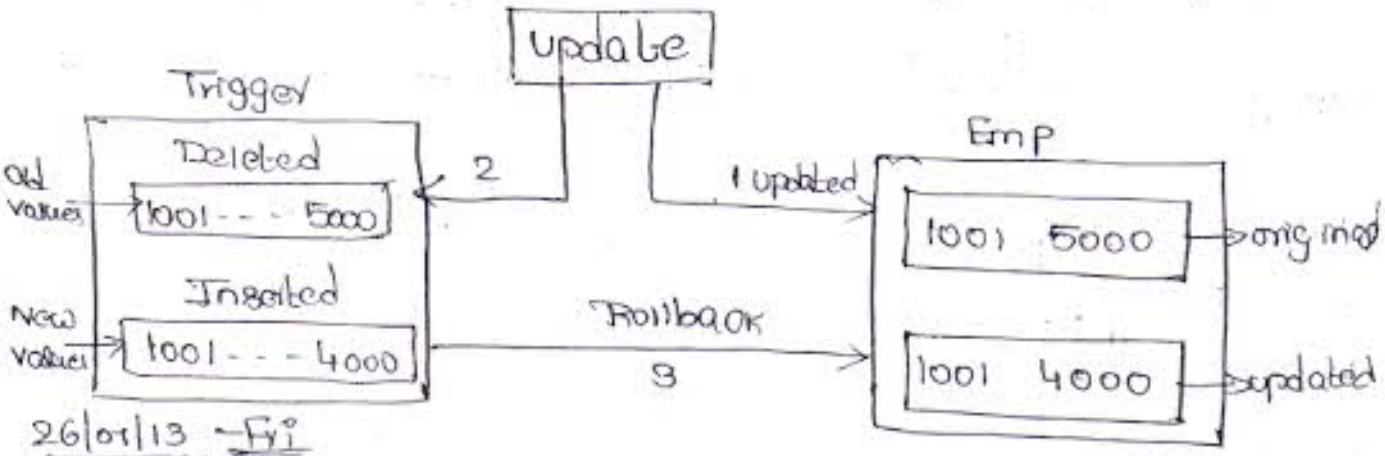
→ Write a trigger on the Emp table that fires when the salary is updated and restricts the operation if at all the new salary is less. than the old salary.

Sol:-
```
Create Trigger Emp_Update_Trg
On Emp After update
AS
 Begin
 Declare @OSalMoney, @NSal Money
 Select @Osal = Sal From Deleted
 Select @ Nsal = Sal From Inserted
 IF @Nsal < Osal
Begin
Ralback Transaction
Raiserror ('newsalary must be greater than old salary',
                                    16,1)
End
End
```

Test the above trigger by executing the following update statement.

Update Emp set Sal = 4000 where Empno = 1001

Whenever we update a record the trigger treats the update operation as a Delete and Insert. So inside the trigger we will be having both the two magic tables deleted and inserted where Deleted table contains the old values of table and the inserted tables contains the new values of updation.



26/01/13 - Fri

Magic tables:

These are specially created inside of a trigger when we perform insert, update and Delete operations. We have two Magic tables inserted and Deleted under a trigger.

Inserted

This table is created when we perform an insert operation that provides access to the values being inserted into the table.

Deleted:- This table is created when we are performing a delete operation providing access to the record being deleted.

When we perform an update operation we will be having both inserted and deleted table also where inserted will provide access to the new values being inserted and deleted/updated table provides access to the old values of the table.

→ Define a trigger on the dept table so that whenever a record is inserted into the table it will verify whether the corresponding Deptno information is present in DeptDetails table or not and if not present will insert a new record into the table by generating a unique departmentid and given deptno with null in the comments.

```
Create Trigger Dept_Nested
On Dept After Insert
As
Begin
Declare @Deptno int, @Did int
Select @Deptno = Deptno From Inserted
If Not Exists ( Select * From DeptDetails Where
                                    Deptno = @Deptno)
Begin
 Select @Did = ISNULL (MAX(Did), 0) + 1 From DeptDetails
 Insert Into DeptDetails Values(@Did, @Deptno, null)
End
End
```

Test the above trigger by inserting a record in the table as following

```
Insert Into Dept (Dname, Loc) Values ('Research', 'Hyderabad')

Select * From Dept
Select * From DeptDetails
```

→ Write a trigger on the emptable so that whenever a record is being inserted into the table first it verifies the given deptno is existing in the dept table or not if not existing will first insert the deptno into the deptable with name and Loc as Null and then inserts the actual record into Emp table.

Ans:- Create Trigger Emp_Nested
on Emp Instead of Insert
As
Begin
Declare @deptno int
select @ deptno = Deptno From Inserted
If NotExists (Select * From Dept where
Deptno = @Deptno)
Begin
Insert into Dept values (@Deptno, Null, Null)
End
Insert into Emp(Empno, Ename, Job, sal, Deptno)
Select Empno, Ename, Job, sal, Deptno from
Inserted.
End

To test the above trigger write an insert statement as Following.

Insert into Emp (Empno, Ename, Job, sal, Deptno)
Values (1017, 'Abc', 'clerk', 2000, 60)

In the above case when the record is inserted into the table internally the trigger fires first and captures all the values in it.

Because the trigger is a instead of trigger, then it verifies the given deptno is existing in the dept table or not and if not existing will insert a new record into dept table.

once the record is inserted into dept table then the trigger we defined earlier on the dept table will also fire and corresponding record is inserted into Dept-details table. These process of a trigger invoking another trigger for execution is known as Nested triggers.

Insert → Emp
→ Emp-Nested [Trigger]
→ Dept
→ Dept-details Nested [Trigger]
→ Dept details

Note:- In SQLServer, the support for nested triggers is given upto 32 levels.

Instead of Triggers: Instead of Triggers are designed for defining on views actually that is as per the specifications of SQL they can be used only on views. To make the non-updatable Complex views as updatable

By default all Complex views are that are defined on multiple tables were not-updatable.

If we want to make those complex views as updatable we can do it by defining a instead of trigger on that table. So that we can perform DML operations

on Complex Views also.

To test this process first create a complexview as following.

```
Create View Emp_Dept
 As
 Select E.Empno, E.Ename, E.Job, E.Sal, D.Deptno,
    D.Dname, D.Loc
  From Emp E Inner Join Dept D
  On E.Deptno = D.Deptno
```

Now try to insert a record into this View as following

```
Insertinto Emp_Deptvalues(100, 'williams', 'manager',
        4000, 100, 'HR', 'pune')
```

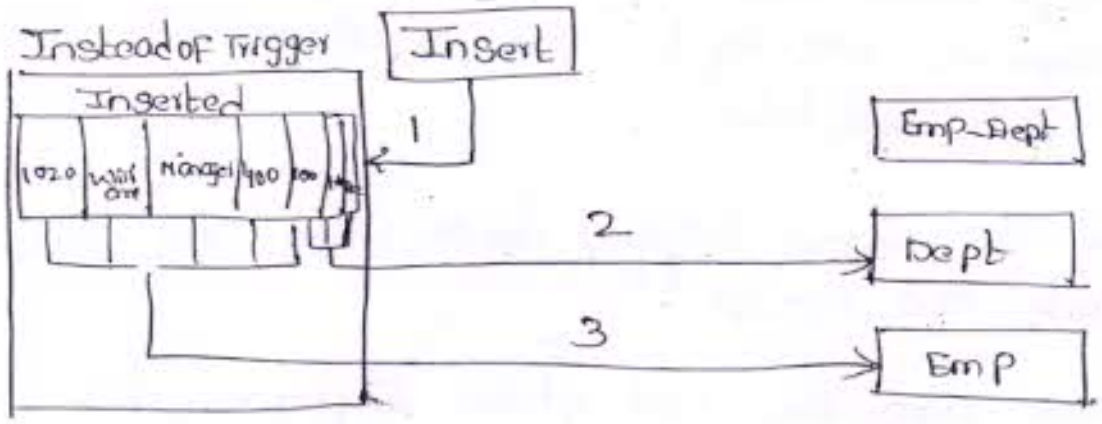
When we execute the above Statement the operation fails reporting an error the view Emp_Dept is not updatable.

Now to make it updatable we need to define an instead of trigger on the view.

```
Create Trigger View_Trg
On Emp_dept Instead of Insert
 As
 Begin
 Insert into Dept (Deptno, Dname, Loc)
 Select Deptno, Dname, Loc From Inserted

Insert Into Emp (Empno, Ename, Job Salary, Deptno)
     Select Empno, Ename, Job, Salary, Deptno
      From Inserted
End
```

Now Execute our previous insert statement and Insert a record into the view which internally calls the trigger and inserts the values into the two tables.



24/07/2013 - Monday

Defining the instead of trigger for performing delete Operation on Complex view

```
Sol:- Create Trigger view_Delete
      on Emp_Dept Instead of Delete
      As
      Begin
      Declare @Empno int, @Deptno int, @count int
      Select @Empno = Empno, @Deptno = Deptno from
                                       Deleted
      Select @count = count (*) From Emp where
                  Deptno = @Deptno
      Delete From Emp Where Empno = @Empno
      if @count = 1
      Begin
       Delete from Dept where Deptno = @Deptno
       End
      End.
```

Note:- In the above case, if their is only one Employee working for the department we want to delete after deleting the record from emp table it will delete from dept table also where as if there are more than one Employee working in the department, it will not delete from dept table

To test the above trigger, write the following statement and execute.

→ Delete From Emp_Dept Where Empno = 1020.

Disabling the Nested Trigger

As we discussed Earlier a trigger can fire another Trigger For execution and we can Nest them upto 32 levels.

It is possible to disable these Nested triggers if not required by changing the Server Setting of Nested triggers as OFF (0). Default is ON (1).

sp_configure 'Nested Triggers', 0

Reconfigure

Note:- When we change the Server Settings the setting gets affected only after restarting the server but without restarting the server we can change the Server Setting by using the statement "reconfigure"

Trigger → Trigger → Trigger
After → After → After
Instead of → After → Trigger

DDL TRIGGERS AND LOGON TRIGGERS:

DDL Triggers Fire in response to a variety of data definition Language Events like Create, Alter, drop, Grant, deney & Revoke

We use DDL triggers when we want to do any of the following Certain

1. Prevent changes to ~~alter~~ our database.
2. For something to occur in the database in response to a change in your database.
3. Record changes are Events in the database.

A DDL trigger is a special type of Stored procedure that executes in response to a Server scoped or database Scoped Events

DDL triggers Fire only after the DDL Statement executes so we cant use "Instead of triggers" here and more over DDL triggers will not Fire in response to events that effect local ~~temp~~ temporary tables.

DDL Triggers doesn't create the special Magic table inserted and deleted.

DDL Trigger scope:

DDL Trigger can Fire in response to an SQL event processed in the current database ~~or~~ on the Current server. The Scope of the trigger depends on the Event specified to define the trigger

Syntax:- Create (or Alter) Trigger <Name>
on All Server | Database
[With <trigger attributes>]
For | After <event_type>
AS
Begin
- Trigger Body
End

<Event type> refers to the event that will fire the trigger which can be anything like Create_table, Drop-table, Alter-table ... etc which must be following a convention of event_type that is which event and which type.

Events { Create_Table, Drop_view, Alter_procedure } types.

Write a trigger which restricts dropping of a table from the database at any time.

```
Create Trigger Restrict_DropTable
on Database After Drop_Table
AS
Begin
Rollback
Raiserror ('can't drop table under this database.', 16, 1)
End
```

After creating the table, try to drop a table which gets restricted.

Note:- Drop DDL triggers and Logon triggers immediately after testing.

Dropping a DML Trigger

Drop Trigger <Trigger Name>.

Dropping a DDL Trigger

Drop Trigger <Trg Name> on All Server | Database

Dropping a Logon Trigger

Drop Trigger <Trg Name> on All Server

Dropping the above trigger.

Drop Trigger Restrict_DropTable on Database

→ While a trigger which restricts creating and Altering of a table apart from business hours.

```
Create Trigger Restrict_DB create or Alter Table
on Database After create_Table, Alter_Table
AS
 Begin
 Declare @Hours Int
  set @Hours = Datepart (hh, Getdate())
  If @Hours Not Between 9 and 16
  Begin
  Rollback
  Raiserror ('can't create or alter table now.', 16, 1)
  End
  End
```

Dropping the above trigger.

Drop Trigger Restrict_DB on All Server

Define a trigger on a server so that it will restrict Create, Alter and drop operation on a database.

```
Create Trigger Restrict_DB
On All Server After
Create_Database, Alter_Database, Drop_Database
AS
 Begin
  Rollback
  Raiserror ('you can't perform any operations on a
  database, most be a sys admin user to do them.',16,1)
  End.
```

30/07/2013 – Tuesday

LOGIN Triggers:

These are same as DDL Triggers, but Fires only when we login into the server.

Syntax:-
```
Create or Alter Trigger <Name>
 on All Server
 [With <trigger attributes>]
  For |After Logon
  AS
   Begin
    -Trigger body
   End
```

Logon triggers executes in response to a Logon Event. This Event is raised when a user session is established with the Server.

LoGon Trigger fires after the authenication face of logining in Finshes but before the user Session is actually established.

Note:- Logon Triggers can be defined only Server level but not database level.

Ex:- Create Trigger Connect_Success
On All Server After Logon
AS
Begin
Print 'Login operation was Successful'
End.

→ Write a bigger that fires when a user login into a Server to allow only three Connections i.e, Fourth Connection should not be established.

Sol:- Create Trigger Connection_Limit
On All Sewer After Logon
AS
Begin
IF original_Login() = 'sa' And (Select Count(*)
From sys.DM_Exec_Sessions where Is_user_
Process=1 And original_Login_Name = 'sa') >3
Rollback
End.

[original_Login() → Predefined Function
original_Login_Name → column Name]

Temporary tables

These are tables that are created on the database specifically for using at a point of time only that is they are not Permanently stored on the database.

Generally while defining procedures, Functions, if we want to store a set of records to be used in that sub-program we use this temporary tables.

If required a temporary table can be created directly under the database just like other tables.

Temporary tables are of two types

1. Local temporary table.
2. Global temporary table.

We create a temporary table same as we create a Normal table but to specify it is a temporary table we prefix the table name with a single "#" if it is local temporary and "##" if it is global temporary.

Ex:- create Table #LocalTest(Id int, Name varchar(50))

create Table ##GlobalTest (Id int, Name varchar(50))

Note:- In whatever database we create a temporary table it gets stored under a tempdb database only. you can check it from the object Explorer below tempdb database.

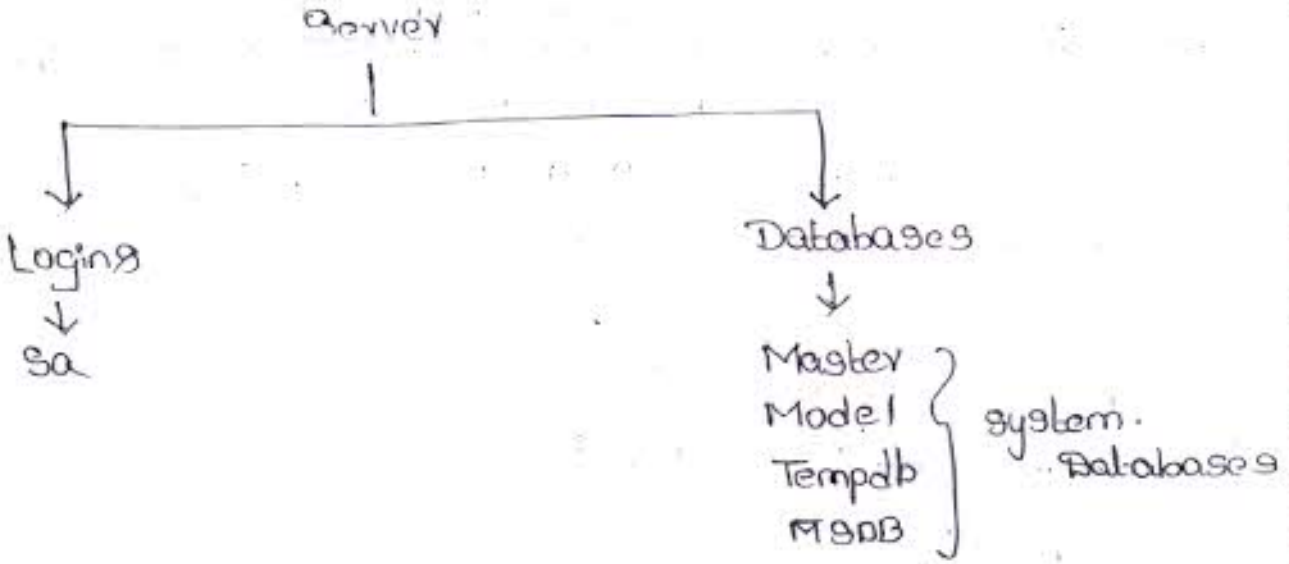
A Local temporary table can be accessed only within the session were the table was created where as a

Global temporary table can be accessed from other sessions also.

The temporary tables whatever we are created under a connection will be dropped or deleted once the connection where the table is created is closed.

Logins under database Server:-

By depault when we install sql server on a machine it comes with logins and databases as following.



Sa(sys.admin) is the depault login account under Sql server and considered as the owner of all the server so Logining an as "sa" we can perform each and every operation on the server without any restrictions.

If required just like we can create New databases, we can also create new logins under the server by using "Create Login" Statement.

Syntax:- Create Login <LoginName> With
Password='<pwd>'
[Must_change]

Check_Expiration = on/OFF,
Default_Database = [<DB Name>]

Note:- To create a New login we must be a system Administrator.

Create Login Raju with password = '321'

Create Login Raju with password = '321'
Must_change, check_Expiration = ON

In this case first time the user logs in the Password gets expired so must be change immediately

Now, we can login with the new login account that can access only master, msdb and tempdb databases but not model database (or) any other user database also.

For the New login it is not possible to give access to modeldb, bubt we can give access to user databases if required

We can give access to user databases with the help of a built in stored procedure

Sp_GrantDB Access

only the owner of the database can give access on a database to other users.

sa (sql11)
sp_Grant-DB Access 'Raju'

Authentication and Authorization:

In the above case, after giving access to the database to the new login, the new login can access the database that is he is authenticated for accessing the database but after accessing the database he cannot perform any operation on the database because he is not authorized to perform any action.

Authentication is a process of verifying the credentials of a user to login into the system. Where as authorization is a process of verifying whether the user as permissions to perform any operations on the database.

DATA CONTROL LANGUAGE:

This is basically used for authorizing a user to Perform any actions on a database which comes with the three commands

1. Grant
2. Revoke and
3. Deny

1. **Grant**: This command is used for giving a previlege (or) permission for a user to perform an action on the database. previlages are of two types
   1. System previlage
   2. object previlage

31/7/13 Wed

System previlege: It is a permission which is given on the complete database to perform any operation on the database.

Syntax:- Grant <permission> to <user | Role>

Ex:- Grant create Table to Raju
Grant Alter Table to Raju.

Note: If a new user creates a table on other users database he doesn't requeries any permission for manipulating his table. because he is the ower of that table.

Object previlege: If a Permission is given on a specific object it is an object previlege.

Syntax:- Grant <permission> [columns)] on <object>
to <user | Role> [with Grant option]

Ex:- Grant Select on Emp to Raju
(or)
Grant select (Empno, Ename, Job, sal) on Emp to Raju

Inside same user

Grant Insert, Delete on Emp to Raju

Grant update(Empno, Ename Job, sal) on Emp to Raju

Giving Execute permission on sp:
Grant Execute on DivByone1 to Raju

Giving select Permission on Function
Grant select on Emp-GetEmpDetails to Raju.

With Grant option: If a Permission is given to a user/ Role by using with Grant-option that user can give that Permission to another user.

ROLE: A role is a group of permissions that is a set of previleges combined together as a group and can be assigned at a time to the user.

Syntax:- Create Role <Role name>

Ex:- Create Role: MyRole

After creating a role grants permissions to the role we have created just like we granted to a user.

Ex: Grant Select, Insert, update, Delete on Dept to myRole.

Grant select, update on Dept-details to MyRole
Grant select, Delete on Salgrade to MyRole
Grant select, Insert on customer to MyRole.

once. after adding Permissions to the role we can add Members under the role so that the user will get all the Permissions that are added under the role.

Adding the member under a role

sp_AddRoleMember <rolename>, <loginname>

sp_AddRoleMember 'MyRole', 'Raju'.

Removing / Deleting a Member from a role

Sp_DropRolemember <rolename>, <loginName>

Sp_DropRoleMember 'MyRole', Raju.

REVOKE COMMAND: This is for taking back the Permission that are given to a user (or) a Role.

- Syntax:- Revoking a system privelege

Syntax: Revoke <permission> From <user | role>

Ex:- Revoke Create Table From Raju

Revoke Creat Alter Table From Raju

Revoking a object privilege

Syntax:- Revoke <permission> [(column(s))] on object From <user | role> [cascade]

Ex:- Revoke Delete on Emp From Raju

Revoke insert, update on Emp From Raju.

on role ← Revoke select, Delete on Salgrade From myRole

Cascade: We use Cascade on Revoke when a Grant Permission is given by using with Grant option so that cascade will Revoke permission from all the users to whom the permission as been given by the grantee [Here raja is a grantor] and [Ba is a grantee]

Denay: This is used for deniing a permission from a user so that it prevents the user from inheriting the Permission from a role also.

Syntax:- Deny (denying a system Previlege) <permission> To <user | role>

Deny create Table to Raju

Deny Alter Table to Raju

- Denying a object previlege:

Deny <permission> [[column(s)]] on object
To <user | role> [cascade]

ex:- Deny select on Dept To Raju

In the above case, we have given a select permission on Dept table to Raju through a role so that if we want to take back that permission Revoke cannot be used because the permission is not given directly in such cases we use deny which denies the permission from that user. But still, that permission can be used by other users of the Role.

1/08/13 - Thursday

carrying or copying a database from one machine to the other.

If we want to carry the database from one machine to the other we have various options.

1. Backup and Restore: A Backup is a file with .bak extension. To take a backup of a database in the object explorer, right click on the database select tasks → Backup, will open a window click on the ok button which will create a backup copy of a database in the following location.

<drive>:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\Backup.

We can carry the above backup file onto the destination system where sqlserver was present and restore it there. To do this open the object Explorer right click on databases select restore database. which opens a window

in that window, select the option device and click on the button beside it which opens a window for selecting the backup device click add button which opens a dialogue box using it select the backup file from its physical file and click ok → ok which restores the database under the server.

Q. Copying mdf and ldf files associated with the database:

As we are aware that every database is associated with a .mdf and .ldf files to carry the database from one place to another we can also copy these two files.

By default we will find those files in the following location.

<drive>:\programfiles\Microsoft SQL server\MSSQL11.MSSQLSERVER\MSSQL\Data

Note: We cannot copy these two files when the database is connected under the server. So to copy them first we need to de-attach the database from the server. To de-tach the database

From the server :-

open Object Explorer right click on the desired database → select task → detach which opens a window Just click on the ok button. Now we can copy the mdf and ldf files

Now copy the mdf files and ldf files onto the target system where SQL server was available and copy those files on to the target machine into any folder (but best location is data Folder)

Now those files can be attached under the server. which will add the database under server. To do this right click on databases node in object Explorer and select attach which opens a window -> click on Add button that open a dialogue box using it select the .mdf file from its physical location where we copied it, click ok, ok. which will attach the database under server.

3. Generating a script file to the complete database and its object

We can generate a script file (.sql) and run the script file on the destination so that the database and the objects gets created there. To do this in object Explorer → right click on the database → select task → select Generate scripts which opens a wizard guiding you in the process of script generation.

In the wizard open select Next button, choose the option of entire database (or) selected objects. and click Next which opens a window in that under a filename specify the location where we wanted to save the script file and click Next. which displays a review of selections click Next which displays the progress and success of the script click finish and close the window. Now we can find the script file in the specified location. To run the script file on a machine copy the file on to that machine and double click on it which directly opens a Management studio. execute.

(use sqll)

SQL SERVER REPORTING SERVICES:

It is a tool from sqlServer used for designing of reports. To design a report by using this it must be first installed on the machine.

Note:- In many Editions of Sql server reporting services comes as a part of it.

Using Reporting services we can design reports with a wide variety of datasources like Sql server (or) oracle, xml, Excel etc.

It provides a full range of ready to use tools and services to help you create, deploy and manage reports for organisational needs.

Reporting Services is a ~~server~~ Server based reporting platform that provides comprehensive reporting functionality for a variety of datasources. Reporting Services tool work within the Microsoft visual studio environment and are fully integrated with Sql table tools and components.

With reporting services we can create interactive tabular Graphical (or) Free form reports where the reports can include rich data visualization including charts and maps.

Creating a simple report using reporting services.

Go to → startmenu → All programs → Microsoft visual Studio 2010 and click on it open which opens you "VisualStudio 2010 shell".

In the window open in the lefthand side we find an option new project Click on it which opens

the new project window, select business Intelligence and under it select reporting services. Now in the right select report server project. Now in the bottom specify a name to the project Ex:- Sqll reports pr. and below it specify the location to save. and click ok which opens a wizard for creating a report. Click Next button Specify a Name to the datasource or leave the same. Choose the type as Microsoft Sqlserver. click on the edit button and specify the connection details. and click ok. which displays the connection string in the text box click on the next button which will prompt for userid and Password confirmation click ok that opens a window (Design the query). In it under the query string textbox enter select * From Emp. and click Next. Now choose the Report type as tabular. click Next. Now add all the columns into the details list box. and click Next button. choose a table style in the list and click Next and click Finish. by specifying the Report name as Emp which gets saved as Emp.rdl (report definition Language).

Now open the Solution Explorer in the right hand side of the studio and under the reports folder we find Emp.rdl right click on it and select run which opens a Report viewer. Enter the username and password and click view report which displays the data.

2/08/2013 - Friday

Creating a Report by loading the data from multiple tables.

Open the project sqlll reports we have created earlier and in the right hand of the objee project we find the Solution Explorer open it for adding a new report in the project

SHARED DATASOURCES:

Every report requires the connection details to specify from where the data has to be loaded i.e, userid, Password, Database name, Servername etc So while configuring a report we need to provide all these details as we done it in our first report.

Suppose if we want to design multiple reports accessing data from same database without providing the connection details under the Each report we can take the help of shared datasources that is once we configure these shared datasource we can start consuming it under multiple reports directly.

To create a shared datasource, in the Solution Explorer rightclick on shared datasources under the project and select add new datasource which opens a window in it Enter a name to the datasource Ex:- MyDS. Choose the Server type as Microsoft SQL Server, click on the Edit button which opens a window. In that specify the connection details click -> ok -> ok which adds the datasource under the Shared datasources node.

Note:- The shared datasource we created right now can now be used under all the reports we want to design basing on a database we have configured with.

Creating a new report:

Now rightClick on the reports node in Solution Explorer Select add new report → Click on the next button now. under Shared datasource we will find our mydb that as been created above. select it and click Next. Now under the query string text box right the following query.

```
SELECT E.Empno, E.Ename, E.Job, E.Mgr, E.Hiredate, E.Sal,
E.Comm, D.Deptno, D.DName, D.Loc, DD.Did, DD.Comments,
S.Grade. FROM Salgrade S
INNER JOIN Emp E
ON E.Sal BETWEEN S.LoSal AND S.HiSal
INNER JOIN Dept D
ON E.Deptno = D.Deptno
INNER JOIN Deptdetails DD
ON D.Deptno = DD.Deptno.
```

Click on the Next Button choose tabular and click on the Next button

If we want to display the report page by page basing on any column we can do it by adding the Column into the page display Field list box.

In our current report. let us display the report basing on the Deptno So that Employees of Each department will be shown in Seperate page.

To do this Select deptno and click on the page button so that it's get added into list box beside. Now add the remaining column into Details box. Click on the next button Choose a style for the table and click Next now enter the name of the report as multi-table report and Click finish which adds the .rdl File under reports. Now right-click on it and select run. which displays the information page by page basing on the department number.

Note:- Currently when we run the report it will ask for the username and password. Every time we run the reports to avoid this open the solution Explorer and doubleclick on the shared datasource we created earlier in it select the option credentials and select the radio button, use this username and password. and enter the username & Password and click ok. Now if we run the report again it will not ask for username and password.

Note: for adding any titles to a Report we are given with an option of adding a header to the report, to do it. Go to reports and select add pageheader which will add a header section on top of the report now open the tool box and drag and drop required fields on to the section and do the neccessary alignments.

Same as above we can also add a page footer section to the bottom of the report.

Drilldown reports using GroupBy clause: Add a new report in the project → click Next → choose the shared datasource and click next → Now right the query under the query Builder String text box as following

Select Deptno, Job, Max(Sal) as MaxSal From Group By Deptno, Job.

Next click on the next button → choose tabular and click on next button now add the Deptno and Job columns into Group list box and maxsal into Details list box. Gia. Next select Stepped radio button and Enable drilldown checkboxes. click the next button choose a table style, Click Next, Specify the name of the report as drilldown report and click Finish. Now right click on the report Select run.

Matrix Reports: These reports are used for summarizing the information to design this report add a new report in the project click Next choose the datasource and click Next and write the following Query in the query string text box as following

Select Deptno, Job, sum(sal) AS SalSum From Emp Group By cube (Deptno, Job)

→click Next → choose the report style as matrix →

→click Next now select deptno and add it under rows list box Select Job and add it under columns list box Seled SalSum and add it under details list box click next choose a style for the matrix click next → name the report as matrix report and click Finish. Right click on

the report and select Run.

3/08/13 - Sat

Parametrized Reports:

These are used for making your reports more dynamic that is with the help of the parameters we can send values to the reports for execution so that those values can be sent to the report in run-time and basing on that report queries gets executed.

To design a parametrized report. Add a new report under the project click Next. Choose the data source and click next and write the following query in the query string textbox.

```
Select
  E.Empno, E.Ename, E.Job, E.Sal, S.Grade, E.Comm,
  D.Deptno, D.Dname, D.Loc From Emp E
  Inner Join Salgrade S
  On E.Sal Between S.Losal and S.Hisal
  Inner Join Dept D
    On E.Deptno = D.Deptno
     Where D.Deptno = @deptno
```

Click Next Choose Tabular and click Next. Now add the Columns into details list box click Next. Choose your table style, click Next Enter the report name as Report Parameters and Click finish. Now run the report. which will first ask for the parameter Enter the value to the parameter and click View report.

Note:- The parameter of the report we added that is Deptno can be found under a window report data in left hand side of the studio under parameters node. Right click on the parameter and select parameter Properties in which we find name, Prompting text etc. Under prompting text enter the text we want to prompt For entering of the value. click ok.

Desiging Reports Manually:

Till now we are desiging the reports by using a report wizard but if required we can manually also design the reports where as to design the reports manually First we need to configure a dataset. Where a dataset is collection of fields associated with single core (or) multiple tables. Dataset can also be created as shared Just like shared datasources (or) we can also create Embedded datasets also. where a Embedded dataset is Specific to that report only.

open the solution Explorer right click on the reports and Select add new item which opens a window select report in it naming it as manual report.rdl and click ok where we will get a blank report. Now open the Report data in the left hand side and there we find a node Datasets right click on it select add datasets which opens a window name the dataset as customer os choose the radio button. use the dataset Embedded in my report Next click on the new button beside the datasource and choose our shared datasource that is Myps Next in the query textbox below right the

Query as following.

```
Select Custid, Cname, Balance, Status
FROM Customer
Where (Status = @Status)
```

Next click on the ok button which adds the customers as dataset under datasets node.

Above datasets we find parameters Expand it to view our Status parameter. Right click on it and select Parameter properties. and the window opened enter the Prompt as "Enter status value either true or false". click ok. Now open the toolbox take a text box and place it on the top of the report and enter my company name into it and do any necessary alignments to it.

Place another text box for displaying the address and do the neccessary alignments. Now in the toolbox select line item and draw a line below company Name and address. From the toolbox place a table which shows with two rows, First row reserve for head and second row reserve for data Now put your mouse in the first cell of the second row which displays an icon click on it that displays the columns of your dataset. select Customer id, Select customer name, select balance. By default it comes with three columns only now right click on the top of the third column selects insert column right which adds a new column. to that column bind the status. Now run the report by entering the status number.

Northwind and pubs (Google)

↓

Employee

Displaying Images into Reports

Note:- Let us create the current report configuring with Employee table of Northwind database using a shared dataset

1. First create a shared datasource configuring with Northwind database
2. Now in the solution Explorer rightclick on shared dataset and select new dataset. give a name to the dataset as Employee DS, Next under the datasource choose the datasource configred with Northwind choose the query type as text and write the following query-

```
Select EmployeeID, LastName, FirstName, Country, Photo
  From Employee
```

Click the ok button so that the dataset is added under shared dataset node with a name as Employeeds.rsd Now add it new report in the project that is blank report naming it as image reports.rdl. Open report data window right click on datasets select add dataset. Select use a shared dataset and below select Employee DS dataset. and click ok which will display the columns of your table.

Now place a table on the report and add each column to each cell that is employee id LastName, FirstName, Country and Photo.

When we bind the column of the table to a cell by default it contains a textbox for displaying the data but rightnow under photocolumn we cannot display a image in the textbox. So select and delete the text box in the second row. and place a pimage box in that cell Selecting from a toolbox.

After the image is added into the cell right-click on it and select image properties. In the window open under select the image Source combo box choose database option. Select that under used this field Combo box choose the photocolumn. Under use this MIME type combo box select image/Jpeg and click ok Now run the report where we will be getting the image in the photo column.

Note:- By default if the image is compatable it will directly display the image if the image is not displayed again go to image properties click on the button beside the photocolumn and in that on the top text box we find a value as following.

= Fields!photo.value'

Delete it and write it as following.

= System.Convert.FromBase64String(Mid(system.Convert.ToBase64String(Fields!photo.value),105))

m.bangaraju@gmail.com.