--creating database

create database dbEmployee

--creating table/relation

create table tblEmployee (iEmpId int,

cName char(10),

vAddress varchar(50))

--adding records to employee table

insert into tblEmployee values(100,'mohan','blore')

insert into tblEmployee values(101,'raj','chennai')

--adding values for specific columns- INTO IS OPTIONAL

insert into tblEmployee (iEmpId,cName) values (101,'Naveen')

insert into tblEmployee (iEmpId,cName) values (103,'rahul')

--display table

select \* from tblEmployee

-- updating records

update tblEmployee

set vAddress='Chennai'

where iEmpId=102

--updating multiple columns

update tblEmployee

set vAddress='Mumbai', cName='ramya'

where iEmpId=102

update tblEmployee

set vAddress='Hyd'

where iEmpId=103

--Restriction ,selection;

select \* from tblEmployee where vAddress='blore'

--to display specific columns

select cName from tblEmployee

--for deleting the records

delete from tblEmployee where iEmpId=103

--Implementing Entity integrity constraint

create table tblNewEmployee

(iEmpId int primary key not null,

cName char(20),

vAddress varchar(50),

iSalary int)

--insert

insert into tblNewEmployee values(500,'Rao','Delhi',30000)

insert into tblNewEmployee values(501,'Rajat','punjab',35000)

insert into tblNewEmployee values(502,'sahana','lucknow',45000)

--violations on primary key(Cannot insert duplicate key in object 'dbo.tblNewEmployee')

insert into tblNewEmployee values(500,'Rathi','Delhi',50000)

--violations on primary key(Cannot insert the value NULL into column 'iEmpId')

insert into tblNewEmployee values(null,'Rathi','Delhi',50000)

--department table

create table tblDepartment (iDeptId int primary key not null,

cDeptName char(20), vDeptLocation varchar(20),

vDeptHead varchar(20))

--insert in department table

insert into tblDepartment values(1,'Development','Blore','Mohan')

insert into tblDepartment values(2,'Finance','Blore','Ashok')

insert into tblDepartment values(3,'Accounts','Blore','Bhaskar')

--implementing referential integrity using foreign key

create table tblNewEmployee1

(iEmpId int primary key not null,

cName char(20),

vAddress varchar(50),

iSalary int,

iDeptId int references tblDepartment(iDeptId))

--The INSERT statement conflicted with the FOREIGN KEY constraint

insert into tblNewEmployee1 values(500,'Rao','Delhi',30000,12)

insert into tblNewEmployee1 values(500,'Rao','Delhi',30000,1)

insert into tblNewEmployee1 values(501,'Rahul','Delhi',35000,2)

insert into tblNewEmployee1 values(502,'Rajat','blore',30000,3)

select \* from tblNewEmployee

--assignment1

create table tblPosition

( cPositionCode char(4)primary key not null,

cDesignation char(10),

iCurrentStrength int,

iVacancy int

)

create table tblNewEmployee2

(iEmpId int primary key not null,

cName char(20),

vAddress varchar(50),

iSalary int,

cPositionCode char(4) references tblPosition(cPositionCode))

insert into tblPosition values('P001','Accountant',30, 70)

insert into tblPosition values('P002','Developer',20, 50)

insert into tblPosition values('P003','Manager',30, 60)

insert into tblNewEmployee2 values (600,'naj','bangalore',40000,'P003')

insert into tblNewEmployee2 values (602,'ankit','bihar',40000,'P002')

insert into tblNewEmployee2 values (603,'adarsh','kerala',40000,'P001')

--A) alter(add new column) and update

alter table tblNewEmployee2 add cCity char(10)

update tblNewEmployee2 set iSalary =60000 where cName='ankit'

--B) QUERY: DISPLAY EMPLOYEE DETAILS SALARY (20000 TO 50000)

select \* from tblNewEmployee2 where iSalary > 20000 and iSalary<50000

--rough

select \* from tblNewEmployee1

select \* from tblEmployee

select \* from tblDepartment

update tblNewEmployee1 set iEmpId =100 where cName='Rao'

update tblNewEmployee1 set iEmpId =103 where cName='Rajat'

--Assignment 2 A)LIST ALL EMPLOYEES BELONGING TO DEPARTMENT ACCOUNTS OR FINANCE (subquery returns multiple value use IN operator)

select \* from tblNewEmployee1

inner join tblDepartment

on tblNewEmployee1.iDeptId=tblDepartment.iDeptId

where cDeptName='Accounts'or cDeptName='Finance'

--or

select \* from tblNewEmployee1

where iDeptId in

( select iDeptId from tblDepartment

where cDeptName='Accounts' or cDeptName='Finance')

--LIST ALL EMPLOYEES BELONGING TO DEPARTMENT ACCOUNTS

select \* from tblNewEmployee1

where iDeptId =

( select iDeptId from tblDepartment

where cDeptName='Accounts')

--B) EMPLOYEES OF CITY BANGALORE AND SALARY > 50000

select \* from tblNewEmployee1

where vAddress='blore' and iSalary>50000

--C) EMPLOYEES OF CITY BANGALORE AND SALARY > 50000

select \* from tblNewEmployee1

where iSalary!>50000

--D)Give alternative name for column (cname) as emp name

select cName AS [Emp Name] from tblNewEmployee1

select cName AS "Emp Name" from tblNewEmployee1

--rename column

sp\_rename 'tblNewEmployee1.vAddress', 'vCity' , 'column'

--E)report of employee name, dept name and designation from 3 tables

select e.cName,d.cDeptName,p.cDesignation

from tblNewEmployee1 e inner join tblDepartment d

on e.iDeptId=d.iDeptId inner join tblPosition p

on d.iDeptId=p.iDeptId

--left outer join

Select cName, cDeptName from tblNewEmployee1 left outer join tblDepartment

on tblNewEmployee1.iDeptId=tblDepartment.iDeptId

--equijoin

select cName, cDeptName from tblNewEmployee1 join tblDepartment

on tblNewEmployee1.iDeptId=tblDepartment.iDeptId

--right outer join

select cName, cDeptName from tblNewEmployee1 right outer join tblDepartment

on tblNewEmployee1.iDeptId=tblDepartment.iDeptId

--full outer join

select cName, cDeptName from tblNewEmployee1 full outer join tblDepartment

on tblNewEmployee1.iDeptId=tblDepartment.iDeptId

--remove column

alter table tblDepartment drop column cDeptName

--EMPLOYEES NOT BELONGING TO DEVELOPMENT OR ACCOUNTS

select \* from tblNewEmployee1

where iDeptId not in

( select iDeptId from tblDepartment

where cDeptName='Accounts' or cDeptName='Finance')

--OR

select \* from tblNewEmployee1

where iDeptId in

( select iDeptId from tblDepartment

where cDeptName<>'Accounts' AND cDeptName<>'Finance')

--USING COMPUTED COLUMNS( salary with bonus)

select cName "Employee Name", iSalary, iSalary+iSalary\*10/100 "total salary" from tblNewEmployee1

--DISPLAY DATA IN ASCENDING AND DESCENDING ORDER (ASC KEYWORD OPTIONAL AND ITS DEFAULT, DESC DESCENDING)

select \* from tblNewEmployee1 order by iSalary desc

--multilevel ordering (2 columns with different orderings)

select \* from tblNewEmployee1 order by iSalary desc, cName desc

--Assignment3) A) display number of employees in each department (groupby)

select d.cDeptName as Department, COUNT(\*)AS NUMBER from tblNewEmployee1 e, tblDepartment d

where e.iDeptId=d.iDeptId group by d.cDeptName

--B)display number of employees in each city (groupby)

select vCity, COUNT(\*)as "number of emp" from tblNewEmployee1 group by vCity

--C)display number of employees working in bangalore (having)

select e.vCity as City, COUNT(\*)as "number of employees" from tblNewEmployee1 e

group by e.vCity having e.vCity='Delhi'

--D)usage of count and distinct

select distinct COUNT(cDeptName) as "total number of departments" from tblDepartment

--E)usage of distinct

select distinct vCity from tblNewEmployee1

--string function

SELECT CONCAT( 'Happy ', 'Birthday ', 11, '/', '25' ) AS Result;

PRINT DIFFERENCE('Green','Greene')

PRINT LEN('abcdefg')

PRINT LOWER('ABCD')

PRINT UPPER('abcd')

PRINT 'ss'+LTRIM(' ABCD')

PRINT RTRIM('ABCD ')+'ss'

PRINT NCHAR(116)

PRINT PATINDEX('%ter%', 'interesting data')

PRINT QUOTENAME('abc def')

SELECT REPLACE('abcdefghicde','cde','xxx');

PRINT REPLICATE('0', 4)

PRINT REVERSE('ABCD')

PRINT RIGHT('abcdefg',2)

SELECT SOUNDEX ('Smith'), SOUNDEX ('Smythe');

PRINT 'ADC'+ SPACE(2)+ 'DEF'

SELECT STR(123.45, 6, 1);

SELECT STUFF('abcdef', 2, 3, 'ijklmn')

PRINT SUBSTRING('ABCJHFJH',1,4)

SELECT UNICODE('A HSDFJH 33')

DECLARE @document varchar(64);

SELECT @document = 'Reflectors are vital safety' +

' components of your bicycle.';

SELECT CHARINDEX('vital', @document, 5);

SELECT PATINDEX('%Delhi%',vCity)

FROM tblNewEmployee1

select QUOTENAME('SQL Server Rider','<')

select reverse(cDeptName)from tblDepartment where iDeptId=1

--date functions

--A) Value that contains the date and time of the

--computer on which the instance of SQL Server is running

PRINT SYSDATETIME()

--B) value that contains the date and time of the computer on

--which the instance of SQL Server is running. The time zone offset is included.

PRINT SYSDATETIMEOFFSET ( )

--C) The date and time is returned as UTC time (Coordinated Universal Time).

PRINT SYSUTCDATETIME ( )

--D) date and time of the computer on which the instance of SQL Server is running.

PRINT CURRENT\_TIMESTAMP

--E) date and time of the computer on which the instance of SQL Server is running

PRINT GETDATE ( )

--F) The date and time is returned as UTC time (Coordinated Universal Time).

PRINT GETUTCDATE()

--G) Returns a character string that represents the specified datepart of the specified date.

PRINT DATENAME(year, '12:10:30.123')

PRINT DATENAME(year, GETDATE())

--H) Returns an integer that represents the specified datepart of the specified date.

PRINT DATEPART(MONTH,GETDATE())

--I) Returns an integer that represents the day day part of the specified date.

PRINT DAY(GETDATE())

--J) Returns an integer that represents the month part of a specified date

PRINT MONTH(GETDATE())

--K) Returns an integer that represents the year part of a specified date.

PRINT YEAR(GETDATE())

--L) Returns a date value for the specified year, month, and day.

PRINT DATEFROMPARTS(2016,7,18)

--M) Returns a datetime2 value for the specified date and time and with the specified precision

PRINT DATETIME2FROMPARTS(2010,12,31,23,59,59,5,3)

--N) Returns a datetime value for the specified date and time.

PRINT DATETIMEFROMPARTS(2010,12,31,23,59,59,0)

--O) Returns a datetimeoffset value for the specified date and time and

--with the specified offsets and precision.

PRINT DATETIMEOFFSETFROMPARTS(2010,12,31,14,23,23,0,12,0,3)

--P) Returns a smalldatetime value for the specified date and time.

PRINT SMALLDATETIMEFROMPARTS(2010,12,31,23,59)

--Q) Returns a smalldatetime value for the specified date and time.

PRINT TIMEFROMPARTS(23,59,59,0,4)

--R) Returns a new datetime value by adding an

--interval to the specified datepart of the specified date.

PRINT DATEADD(YEAR,21,'2006-07-31')

--S) Returns the last day of the month that contains the specified date, with an optional offset.

PRINT EOMONTH(GETDATE())

--T) SWITCHOFFSET changes the time zone offset of a DATETIMEOFFSET

--value and preserves the UTC value.

PRINT SWITCHOFFSET('2017-01-18 7:45:50.71345 -5:00','-08:00')

--U)TODATETIMEOFFSET transforms a datetime2 value into a datetimeoffset value.

--The datetime2 value is interpreted in local time for the specified time\_zone.

PRINT TODATETIMEOFFSET(GETDATE(),'-07:00')

--V) Returns the current value, for the session, of SET DATEFIRST.

--Sets the first day of the week to a number from 1 through 7.

SET DATEFIRST 5;

SELECT @@DATEFIRST, DATEPART(dw,SYSDATETIME())

--W) Sets the order of the dateparts (month/day/year)

--for entering datetime or smalldatetime data.

SET DATEFORMAT dmy

DECLARE @datevar datetime2=GETDATE()

SELECT @datevar

--X)Determines whether a datetime or

--smalldatetime input expression is a valid date or time value.

IF ISDATE('2009-05-12 10:19:41.177')=1

PRINT 'VALID'

ELSE

PRINT 'INVALID'

--Y) Returns the number of date or time datepart boundaries

--that are crossed between two specified dates.

PRINT DATEDIFF(YEAR,'2005-12-31 23:59:59.9999999','2006-01-01 00:00:00.0000000')

--Default

create table tblPrac( iId int primary key not null, cName char(10),

vDetails varchar(15) default 'unknown');

insert into tblPrac(iId,cName) values (1,'mohan')

select \* from tblPrac

--rule

create rule range\_rule AS @range>= 1000 AND @range <750000;

create table tblRule1(iNum int primary key, iSalary int)

EXEC sp\_bindrule 'range\_rule','tblRule1.iSalary';

insert into tblRule1 values(1,10000)

select \* from tblRule1

insert into tblRule1 values(5,760000)

EXEC sp\_unbindrule 'tblRule1.iSalary';

--CREATE VIEW

create view vwEmpSalary AS

select cName as "Employee Name", iSalary as "Salary"

from tblNewEmployee1

--display view

select \* from vwEmpSalary

--nested view

create view vwEmpName AS select "Employee Name" from vwEmpSalary

select \* from vwEmpName

--updating view(base table value is also updated"

update vwEmpSalary

set Salary=50000 where [Employee Name]='Rajat'

--drop view

drop view vwEmpName

--INDEX

--create table and insert values

CREATE TABLE Sales(

ID INT IDENTITY(1,1)

,PCode VARCHAR(20)

,Price FLOAT(53))

CREATE PROCEDURE InsertIntoSales

AS

SET NOCOUNT ON

BEGIN

DECLARE @PC VARCHAR(20)='A12CB'

DECLARE @Price INT = 50

DECLARE @COUNT INT = 0

WHILE @COUNT<500

BEGIN

SET @PC=@PC+CAST(@COUNT AS VARCHAR(20))

SET @Price=@Price+@COUNT

INSERT INTO Sales(PCode,Price) VALUES (@PC,@Price)

SET @PC='A12CB'

SET @Price=50

SET @COUNT+=1

END

END

EXEC InsertIntoSales

SELECT \* FROM Sales

--1:logical reads =88

set statistics io on

select \* from Sales where ID=88

--2: build a clustered index, logical reads=3

CREATE CLUSTERED INDEX clInd ON SALES(ID)

--3: logical reads=2

set statistics io on

select \* from Sales where ID=88

--4: 11 records, logical reads=5

SET STATISTICS IO ON

SELECT \* FROM Sales WHERE PCode like 'A12CB35%' order by Price

--5: NCI would be built on the CI, logical reads=5

CREATE NONCLUSTERED INDEX ncInd ON SALES(PCode)

--6: logical reads=5

SET STATISTICS IO ON

SELECT \* FROM Sales WHERE PCode like 'A12CB35%' order by Price

--7:

--logical reads=3

DROP INDEX Sales.clInd

--logical reads=3

SET STATISTICS IO ON

SELECT \* FROM Sales WHERE PCode like 'A12CB35%' order by Price

--8

DROP INDEX Sales.ncInd

drop proc InsertIntoSales

drop table Sales

* A clustered index determines the order in which the rows of the table will be stored on disk – and it actually stores row level data in the leaf nodes of the index itself. A non-clustered index has no effect on which the order of the rows will be stored.
* Using a clustered index is an advantage when groups of data that can be clustered are frequently accessed by some queries. This speeds up retrieval because the data lives close to each other on disk. Also, if data is accessed in the same order as the clustered index, the retrieval will be much faster because the physical data stored on disk is sorted in the same order as the index.
* A clustered index can be a disadvantage because any time a change is made to a value of an indexed column, the subsequent possibility of re-sorting rows to maintain order is a definite performance hit.
* A table can have multiple non-clustered indexes. But, a table can have only one clustered index.
* Non clustered indexes store both a value and a pointer to the actual row that holds that value. Clustered indexes don’t need to store a pointer to the actual row because of the fact that the rows in the table are stored on disk in the same exact order as the clustered index – and the clustered index actually stores the row-level data in it’s leaf nodes.

--fillfactor

ALTER INDEX clInd ON Sales

REBUILD WITH (FILLFACTOR = 80)

--creating stored procedures

--1)create procedure to display records of employee table

create procedure prcDisplayRecords

AS

select \* from tblNewEmployee1

--execute procedure

EXEC prcDisplayRecords

--2)create procedure to display employee and department name

create procedure prcDisplayEmpDept

as

select e.cName,d.cDeptName from tblNewEmployee1 e join tblDepartment d

on e.iDeptId=d.iDeptId

exec prcDisplayEmpDept

--PARAMETERISED PROCEDURE

--Display emp records based on location

create proc prcDisplaySpecificEmp

@address varchar(20)

AS

select \* from tblNewEmployee1 where vCity=@address

--toexecute parameterised proc (delhi is input parameter)

exec prcDisplaySpecificEmp 'Delhi'

--stored procedure using input and output paramenters and displaying result

create proc prcSpecificEmp2

@id int,

@location char(10)OUTPUT

AS

SELECT @location = vCity from tblNewEmployee1 where iEmpId=@id

--executing procedure using output parameter

DECLARE @location char(10)

EXEC prcSpecificEmp2 @id = 100, @location = @location OUTPUT

SELECT @location as city

--NESTED PROCEDURE

--DISPLAY EMPLOYEE AND DEPARTMENT RECORDS

create proc prcEmp

as

select \* from tblNewEmployee1

create proc prcDept

as

select \* from tblDepartment

create proc prcDiplayEmDe

AS

EXEC prcEmp

EXEC prcDept

EXEC prcDiplayEmDe

--ALTER PROCEDURE( FROM ABOVE EG ITSELF)

alter proc prcEmp

as

select \* from tblNewEmployee1 where iEmpId=100

EXEC prcEmp

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

--USING RETURN CODES

---A)

CREATE PROCEDURE prcRetCod

@val1 char(10) = NULL, -- NULL default value

@res char(10) =null OUTPUT

AS

declare @ret\_code int

-- Validate the @val1 parameter.

IF @val1 IS NULL

BEGIN

PRINT 'ERROR: You must specify a name for the employee.'

RETURN(1) --(1)Required parameter value is not specified

END

ELSE

BEGIN

-- Make sure the value is valid.

IF (SELECT COUNT(\*) FROM tblNewEmployee1

WHERE cName = @val1) = 0

RETURN(2) --(2) specified parameter value not valid

END

-- Get the department for the specified name and assign it to the output parameter.

SELECT @res = cDeptName

FROM tblNewEmployee1 e

JOIN tblDepartment d ON e.iDeptId = d.iDeptId

WHERE cName = @val1

-- Check for SQL Server errors.

IF @@ERROR <> 0 --(<>)not equal

BEGIN

RETURN(3)--(3) error occured in getting value

END

ELSE

BEGIN

-- Check to see if the output value is NULL.

IF @res IS NULL

RETURN(4) --(4) null value for result

ELSE

RETURN(0)-- SUCCESS!!

END

--B)-----------------------------------------------------------

-- Run the stored procedure without specifying an input value.

EXEC prcRetCod

--C)-----------------------------------------------------------

-- Execute the procedure specifying a last name for the input parameter

-- and saving the output value in the variable @SalesYTD

-- Run the stored procedure with an input value.

DECLARE @resForval1 char(10), @ret\_code int;

EXECUTE @ret\_code= prcRetCod

N'', @res = @resForval1 OUTPUT

PRINT N'dept name is ' +

CONVERT(varchar(10), @resForval1);

IF @ret\_code = 0

BEGIN

PRINT 'Procedure executed successfully'

-- Display the value returned by the procedure.

PRINT 'dept name is ' + @resForval1

END

ELSE IF @ret\_code = 1

PRINT 'ERROR: You must specify a last name for the sales person.'

ELSE IF @ret\_code = 2

PRINT 'EERROR: You must enter a valid last name for the sales person.'

ELSE IF @ret\_code = 3

PRINT 'ERROR: An error occurred getting sales value.'

ELSE IF @ret\_code = 4

PRINT 'ERROR: No sales recorded for this employee.'

--D)--------------------------------------------------------

drop proc prcRetCod

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

--USING RAISE ERROR

--A) IN TRY AND CATCH BLOCK

BEGIN TRY

-- RAISERROR with severity 11-19 will cause execution to

-- jump to the CATCH block.

RAISERROR ('Error raised in TRY block.', -- Message text.

16, -- Severity.

1 -- State.

);

END TRY

BEGIN CATCH

DECLARE @ErrorMessage NVARCHAR(4000);

DECLARE @ErrorSeverity INT;

DECLARE @ErrorState INT;

SELECT

@ErrorMessage = ERROR\_MESSAGE(),

@ErrorSeverity = ERROR\_SEVERITY(),

@ErrorState = ERROR\_STATE();

-- Use RAISERROR inside the CATCH block to return error

-- information about the original error that caused

-- execution to jump to the CATCH block.

RAISERROR (@ErrorMessage, -- Message text.

@ErrorSeverity, -- Severity.

@ErrorState -- State.

);

END CATCH;

--CREATING AD HOC MESSAGE IN SYS.MESSAGES

sp\_addmessage @msgnum = 50005,

@severity = 10,

@msgtext = N'<<%7.3s>>';

RAISERROR (50005, -- Message id.

10, -- Severity,

1, -- State,

N'hgvhjgjgjabcdefghijkln'); -- First argument supplies the string.

-- The message text returned is: << abc>>.

sp\_dropmessage @msgnum = 50005;

--USING LOCAL VARIABLE TO SUPPLY MESSAGE TEXT

DECLARE @StringVariable NVARCHAR(50);

SET @StringVariable = N'<<%6.3s>>';

RAISERROR (@StringVariable, -- Message text.

10, -- Severity,

1, -- State,

N'abcde'); -- First argument supplies the string.

-- The message text returned is: << abc>>.

--EXECUTING SYSTEM PROCEDURE

EXEC sys.sp\_who

--execute system extended stored procedure

exec sys.xp\_subdirs 'C:\'

--TRIGGERS

--INSERT TRIGGER - create trigger to update position table, vacancy and current strenght

--info whenever a new emp record is added to emp table

create trigger trgUpdatePosOnAddEmp

on tblNewEmployee1 for insert

as

update tblPosition set iCurrentStrength=iCurrentStrength+1, iVacancy=iVacancy-1

where iDeptId= (select iDeptId from inserted)

insert into tblNewEmployee1 values(110,'chikoo','Rajasthan',50000,1)--curStr=30,vac=70

select \* from tblPosition --CurStr=31,Vac=69

--UPDATE TRIGGER- Create trigger employee when employee designation is updated, the position

--table reflects updates

create trigger trgUpdateDesOnUpEmp

on tblNewEmployee1 for update

as

begin

update tblPosition

set iCurrentStrength=iCurrentStrength-1, iVacancy=iVacancy+1

where iDeptId=(select iDeptId from deleted)

update tblPosition

set iCurrentStrength=iCurrentStrength+1, iVacancy=iVacancy-1

where iDeptId=(select iDeptId from inserted)

end

update tblNewEmployee1 set iDeptId=1 where iEmpId=112

select \* from tblNewEmployee1

select \* from tblPosition

--USER DEFINED FUNCTIONS

--SCALAR FUNCTION

CREATE FUNCTION dbo.demoFunc(@depId int)

RETURNS int

AS

BEGIN

DECLARE @ret int;

SELECT @ret = AVG(e.iSalary)

FROM tblNewEmployee1 e

WHERE e.iDeptId = @depId

IF (@ret IS NULL)

SET @ret = 0;

RETURN @ret;

END;

SELECT dbo.demoFunc(e.iDeptId) --EXECUTE SCALAR FUNCTION

FROM tblNewEmployee1 e where e.iEmpId=100

--INLINE TABLE VALUED FUNCTION

CREATE FUNCTION func1 (@id int)

RETURNS TABLE

AS

RETURN

(

SELECT d.iDeptId, d.cDeptName,d.vDeptLocation

FROM tblDepartment d

JOIN tblNewEmployee1 e ON d.iDeptId=e.iDeptId

JOIN tblPosition p ON e.iDeptId=p.iDeptId

WHERE d.iDeptId = @id

);

SELECT \* FROM func1 (1); --EXECUTE TABLE VALUED FUNCTION

drop function func1

--MULTISTATEMENT TABLE VALUED UDF

CREATE FUNCTION funcMultiStmt( @firstname char(20) )

RETURNS @details table (

cid Varchar(11),

cAddress varchar(10))

AS

BEGIN

INSERT INTO @details

SELECT iEmpId,vCity

FROM tblNewEmployee1

WHERE cName=@firstname

IF @@ROWCOUNT = 0

BEGIN

INSERT INTO @details VALUES ('','No Data Found')

END

RETURN

END

GO

SELECT \* FROM funcMultiStmt('Rahul') --EXECUTE FUNCTION

--DROP FUNCTION

drop function demoFunc

--BATCH

--EXPLICIT BATCH- two or more SQL statements separated by semicolons (;)

INSERT INTO tblPosition values('P004','Security',50,30,3);

INSERT INTO tblPosition values('P005','Head',40,40,1);

INSERT INTO tblPosition values('P006','Tim',60,20,3)

select \* from tblPosition

--PROCEDURES- If a procedure contains more than one SQL statement, it is considered

--to be a batch of SQL statements

CREATE PROCEDURE getDetails (@EID INT) AS

SELECT \* FROM tblNewEmployee1 WHERE iDeptId = @EID

SELECT cDeptName FROM tblDepartment

WHERE iDeptId = @EID

EXEC getDetails 1 --EXEC procedure\_name parameter\_value

--ARRAYS OF PARAMETERS

INSERT INTO tblEmployee (cName, cName, vAddress)

VALUES (?,?,?)

select \* from tblEmployee

--GLOBAL VARIABLES

--@@CONNECTIONS-number of logins or attempted logins since SQL Server was last started

SELECT GETDATE() AS 'Today''s Date and Time',

@@CONNECTIONS AS 'Login Attempts'

--maximum number of simultaneous connections that can be made with SQL Server in this computer

SELECT @@MAX\_CONNECTIONS AS 'Max Connections'

--amount of time, in ticks, that the CPU has spent doing SQL Server work since the last time

--SQL Server was started.

SELECT @@CPU\_BUSY \* CAST(@@TIMETICKS AS FLOAT) AS 'CPU microseconds',

GETDATE() AS 'As of' ;

--check the error status (succeeded or failed) of the most recently executed statement

IF @@ERROR <> 0

PRINT 'Your error message';

--The last value inserted into an IDENTITY column by an insert or select into statement

INSERT INTO [TempE].[dbo].[CaseExpression]

([Code]) VALUES (5)

GO

SELECT @@IDENTITY AS 'Identity';

--The amount of time, in ticks, that SQL Server has been idle since it was last started.

SELECT @@IDLE \* CAST(@@TIMETICKS AS float) AS 'Idle microseconds',

GETDATE() AS 'as of'

--The amount of time, in ticks, that SQL Server has spent doing input and output

--operations since it was last started.

SELECT @@IO\_BUSY\*@@TIMETICKS AS 'IO microseconds',

GETDATE() AS 'as of'

--The local language id of the language currently in use (specified in syslanguages.langid).

SET LANGUAGE 'Italian'

SELECT @@LANGID AS 'Language ID'

SET LANGUAGE 'us\_english'

SELECT @@LANGID AS 'Language ID'

--The name of the language currently in use (specified in syslanguages.name).

SELECT @@LANGUAGE AS 'Language Name';

--The maximum length, in bytes, of a character in SQL Server's default character set.

SELECT @@MAX\_PRECISION AS 'Max Precision'

--The number of input packets read by SQL Server since it was last started.

SELECT @@PACK\_RECEIVED AS 'Packets Received'

--The number of output packets written by SQL Server since it was last started.

SELECT @@PACK\_SENT AS 'Pack Sent'

--The number of errors that have occurred while SQL Server was sending and receiving packets.

SELECT @@PACKET\_ERRORS AS 'Packet Errors'

--The number of rows affected by the last command

IF @@ROWCOUNT = 0

PRINT 'Warning: No rows were updated';

--The name of the local SQL Server

SELECT @@SERVERNAME AS 'Server Name'

--The server process ID number of the current process.

SELECT @@SPID AS 'ID', SYSTEM\_USER AS 'Login Name', USER AS 'User Name'

--The current value of the set textsize option, which specifies the maximum length, in bytes,

--of text or image data to be returned with a select statement. Defaults to 32K.

SET TEXTSIZE 2048

SELECT @@TEXTSIZE AS 'Text Size'

--The number of microseconds per tick. The amount of time per tick is machine dependent.

SELECT @@TIMETICKS AS 'Time Ticks';

--The number of errors that have occurred while SQL Server was reading or writing.

SELECT @@TOTAL\_ERRORS AS 'Errors', GETDATE() AS 'As of'

--The number of disk reads by SQL Server since it was last started.

SELECT @@TOTAL\_READ AS 'Reads', @@TOTAL\_WRITE AS 'Writes', GETDATE() AS 'As of'

--The nesting level of transactions. Each begin transaction in a batch increments the transaction

--count. When you query @@trancount in chained transaction mode, its value is

--never zero since the query automatically initiates a transaction.

PRINT @@TRANCOUNT

-- The BEGIN TRAN statement will increment the

-- transaction count by 1.

BEGIN TRAN

PRINT @@TRANCOUNT

BEGIN TRAN

PRINT @@TRANCOUNT

-- The COMMIT statement will decrement the transaction count by 1.

COMMIT

PRINT @@TRANCOUNT

COMMIT

PRINT @@TRANCOUNT

--The date of the current version of SQL Server.

SELECT @@VERSION AS 'SQL Server Version'

--LOCAL VARIABLES

-- Printing the value of a variable initialized by using SET

DECLARE @myvar char(20);

SET @myvar = 'This is a test';

SELECT @myvar

--Using a local variable assigned a value by using SET in a SELECT statement

DECLARE @state char(25);

SET @state = N'Delhi';

SELECT cName AS Name, iSalary

FROM tblNewEmployee1

WHERE vCity = @state;

--Using a compound assignment for a local variable

/\* Example one \*/

DECLARE @NewBalance int ;

SET @NewBalance = 10;

SET @NewBalance = @NewBalance \* 10;

SELECT @NewBalance;

--Using SET with a global cursor

DECLARE my\_cursor CURSOR GLOBAL

FOR SELECT \* FROM tblDepartment

DECLARE @my\_variable CURSOR ;

SET @my\_variable = my\_cursor ;

--There is a GLOBAL cursor declared(my\_cursor) and a LOCAL variable

--(@my\_variable) set to the my\_cursor cursor.

DEALLOCATE my\_cursor;

--There is now only a LOCAL variable reference

--(@my\_variable) to the my\_cursor cursor.

--Assigning a value from a query

DECLARE @rows int;

SET @rows = (SELECT COUNT(\*) FROM tblDepartment);

SELECT @rows;

--ERROR FUNCTIONS

CREATE PROCEDURE usp\_GetErrorInfo

AS

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_SEVERITY() AS ErrorSeverity

,ERROR\_STATE() AS ErrorState

,ERROR\_PROCEDURE() AS ErrorProcedure

,ERROR\_LINE() AS ErrorLine

,ERROR\_MESSAGE() AS ErrorMessage;

GO

exec usp\_GetErrorInfo

--TRY CATCH- COMBINING WITH PREVIOUS ERROR PROCEDURE

BEGIN TRY

-- Generate divide-by-zero error.

SELECT 1/0;

END TRY

BEGIN CATCH

-- Execute error retrieval routine.

EXECUTE usp\_GetErrorInfo;

END CATCH;

DROP PROC usp\_GetErrorInfo

--ERRORS NOT CAUGHT BY TRY CATCH EXAMPLE

BEGIN TRY

-- Table does not exist; object name resolution

-- error not caught.

SELECT \* FROM NonexistentTable;

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

--CURSOR

DECLARE cu1 CURSOR

SCROLL DYNAMIC

FOR SELECT \* FROM tblNewEmployee1

OPEN cu1

FETCH ABSOLUTE 2 FROM cu1 --NEXT,FIRST,PRIOR,LAST,ABSOLUTE n,RELATIVE n

CLOSE cu1

DEALLOCATE cu1