SQL Server Query Notes  
 Mr. Sachin Sirohi  
 M.Tech.(CSE)

create database student

use student

select \* from employee

----------------------------------show duplicate record---------------------------

select distinct \* from employee;

Select empname FROM employee GROUP BY empname Having COUNT(\*) > 1;

------------------------------------dense raking---------------------------------

SELECT empname,empaddress,empid,

DENSE\_RANK () OVER (ORDER BY empid DESC)

price\_rank FROM employee;

insert into employee values(107,'Ravi Singh','Lko')

insert into employee(empname,empid,empaddress) values('Anita',5001,'Kanpur')

select \* from employee

select empid,empname from employee

select \* from employee where empid=101

select \* from employee where empaddress in('meerut','kanpur');

select \* from employee where empid not between 102 and 105

select \* from employee where empid>=102 and empid<=105

select \* from employee where empname like '\_R%'

select \* from employee where empname like '%h';

select \* from employee where CONTAINS(empaddress,'meerut')

use AdventureWorks2012

SELECT SUM(Freight) as TotalFreight,TerritoryID FROM [Sales].[SalesOrderHeader]

GROUP BY TerritoryID

SELECT SUM(Freight) as TotalFreight,TerritoryID FROM [Sales].[SalesOrderHeader]

GROUP BY TerritoryID HAVING SUM(Freight) > 700000

create table aptech.dbo.student

(

rollno int not null

);

create table product

(

hsncode int identity(1000,1)not null,

prodname char(50)not null,

price money not null default(100),

);

drop table product;

select \* from product2;

insert into product(prodname) values('Laptop');

create table product2

(

hsncode int identity(1000,1)not null,

prodname char(50)not null,

price money not null default(100),

highschrollno int unique,

myid uniqueidentifier default newid()

);

CREATE TABLE BikeParts (

BikeParts\_GUID AS 'ABCD-' + RIGHT(REPLICATE('0', 8) + CONVERT(VARCHAR, BikePart\_ID), 10),

BikePart\_ID INT IDENTITY(1, 1),

BikePart\_Name VARCHAR(100)

)

INSERT INTO BikeParts VALUES ('Break Cable')

INSERT INTO BikeParts VALUES ('Seat Cover')

INSERT INTO BikeParts VALUES ('Head Light')

INSERT INTO BikeParts VALUES ('Tail Lamp')

SELECT \* FROM BikeParts;

select \* from HumanResources.Department;

select Distinct Name from HumanResources.Department;

select top(5) \* from HumanResources.Department;

create table human

(

id bigint,

hname char(100),

gname varchar(200),

mdate datetime

);

select \* from human;

insert into human select \* from HumanResources.Department where DepartmentID between 1 and 2;

select \* from Sales.CurrencyRate;

select ToCurrencyCode,sum(AverageRate) from Sales.CurrencyRate group by ToCurrencyCode having ToCurrencyCode='AUD';

--Comman Table Expression-----------------

drop table human2;

use AdventureWorks2012

create table human2

(

id2 bigint,

hname2 char(100),

);

go

insert into human2 values(101,'Kevin');

with human (id,hname)

as

(

select \* from human2

)

select \* from human

--@declare use--

create table dcr

(

did int,

salary float,

bonus float

);

declare @salary float

set @salary=40000

declare @bns float

set @bns=5000

declare @myid int

set @myid=(select id2 from human2 where id2=101);

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

--create-insert-update-display------

create table emptable

(

ids int ,

employee varchar(20)

)

go

insert into emptable values(1,'Mat'),(2,'Joseph');

go

declare @updatetable table

(

--id int,

oldemp varchar(20),newmp varchar(20)

);

update emptable

set employee=Upper(employee)

output

--inserted.ids,

deleted.employee,

inserted.employee

into @updatetable

select \* from @updatetable

select \* from emptable;

drop table emptable

select \* from HumanResources.Department order by DepartmentID desc;

select \* from HumanResources.Department order by DepartmentID asc;

select \* from HumanResources.Department order by DepartmentID

select name from AdventureWorks2012.HumanResources.Department;

select \* from Production.WorkOrderRouting order by workorderid asc

select workorderid,sum(ActualResourceHrs) as 'TotalHours' from Production.WorkOrderRouting where WorkOrderID>=50 group by WorkOrderID

select workorderid,sum(ActualResourceHrs) as 'TotalHours' from Production.WorkOrderRouting where WorkOrderID>=50 group by WorkOrderID

select max(ActualResourceHrs) from Production.WorkOrderRouting

select workorderid,sum(ActualResourceHrs) as 'TotalHours' from Production.WorkOrderRouting where WorkOrderID>=50 group by all WorkOrderID

select workorderid,operationsequence,sum(ActualResourceHrs) as 'TotalHours' from Production.WorkOrderRouting group by WorkOrderID,OperationSequence having WorkOrderID>=50 and operationsequence>5

select \* from sales.SalesTerritory

select name,CountryRegionCode,sum(salesytd) from sales.SalesTerritory where name <> 'Australia' and name <> 'canada' group by name,CountryRegionCode with cube

select name,CountryRegionCode,sum(salesytd) from sales.SalesTerritory where name <> 'Australia' and name <> 'canada' group by name,CountryRegionCode with rollup

select Product.ProductID from Production.Product inner join Sales.SalesOrderDetail on Product.ProductID=SalesOrderDetail.ProductID;

select \* from Production.Product;

select Product.ProductID from Production.Product union select ProductId from Sales.SalesOrderDetail;

select Product.ProductID from Production.Product union all select ProductId from Sales.SalesOrderDetail;

select Product.ProductID from Production.Product intersect select ProductId from Sales.SalesOrderDetail;

select Product.ProductID from Production.Product except select ProductId from Sales.SalesOrderDetail;

select SalesOrderDetail.ProductID from Sales.SalesOrderDetail except select Product.ProductID from Production.Product;

select top 5 sum(salesYTD) as TotalSalesYTD,name from Sales.SalesTerritory group by name;

--arrange table with one row and six column

select \* from Sales.SalesTerritory

select top 5 'TotalSalesYTD' as GrandTotal,[NorthWest],[Northeast],[Central],[Southwest],[Southeast]

from

(select top 5 Name,SalesYTD from Sales.SalesTerritory)

as SourceTable

PIVOT

(

sum(SalesYTD) for Name IN([NorthWest],[Northeast],[Central],[Southwest],[Southeast] )

)as PivotTable;

--unpivot

select Names,salesYTD from (select GrandTotal,NorthWest,Northeast,Central,Southwest,Southeast from TotalTable)P

UNPIVOT

(SalesYTD FOR Names in (GradndTotal,NorthWest,Northeast,Central,Southwest,Southeast)) AS unpvt;

select \* from dbo.human;

select id,hname,gname from dbo.human group by grouping sets

(

(id,hname)

,(id,gname)

);

--pivot

SELECT 'AverageCost' AS Cost\_Sorted\_By\_Production\_Days,

[0], [1], [2], [3], [4]

FROM

(

SELECT DaysToManufacture, StandardCost

FROM Production.Product

) AS SourceTable

PIVOT

(

AVG(StandardCost)

FOR DaysToManufacture IN ([0], [1], [2], [3], [4])

) AS PivotTable;

--unpivot

CREATE TABLE VEmployee

(VendorID int,

Emp1Orders int,

Emp2Orders int,

Emp3Orders int,

Emp4Orders int,

Emp5Orders int)

GO

INSERT INTO VEmployee VALUES(1, 4, 3, 5, 4, 4)

select \* from VEmployee;

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

SELECT VendorID, Employee, Orders AS NumberOfOrders

FROM

(SELECT VendorID, Emp1Orders, Emp2Orders, Emp3Orders, Emp4Orders, Emp5Orders

FROM VEmployee

) AS p

UNPIVOT

(

Orders FOR Employee IN

(Emp1Orders, Emp2Orders, Emp3Orders, Emp4Orders, Emp5Orders)

) AS unpvt

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

select \* from aptech.dbo.product;

---------view is a virtual table------------------;

CREATE VIEW myview2 AS

select \* from Sales.CreditCard,aptech.dbo.product;

select \* from myview2;

--------view with create table---

create table employee\_Personal\_details

(

empid int not null,

firstname varchar(30),

lastname varchar(30),

address varchar(30)

);

create table employee\_salary\_details

(

empid int not null,

designation varchar(30),

salary int not null

);

create view vmemployee2 as

select e1.empid,firstname,lastname,designation,salary from employee\_Personal\_details e1 join employee\_salary\_details e2 on

e1.empid=e2.empid;

--drop view dbo.vmemployee;

go

select \* from vmemployee2;

insert into vmemployee2 values(2,'jack','wilson','software developer',16000);

---create view in single table

create view singleview as

select empid,firstname from employee\_Personal\_details;

insert into singleview values(101,'Ram');

select \* from singleview;

select \* from employee\_Personal\_details;

update singleview set firstname='Ram' where empid=101;

--update singleview set firstname .write('Sa',1,2) where empid=101;

create table employee\_Personal

(

empid int not null,

firstname nvarchar(300) not null

);

insert into employee\_Personal values(101,'Internal Hard disk');

insert into employee\_Personal values(102,'Internal Hard disk');

select \* from employee\_Personal;

create view emppersonal as

select empid,firstname from employee\_Personal;

select \* from emppersonal;

update emppersonal SET firstname .WRITE(N'EX',0,2) where firstname='Internal Hard disk';

delete from employee\_Personal where empid=101;

drop view emppersonal

exec sp\_helptext emppersonal;

----show all table of database------------

create table product

(

hsncode int,

prodname char(50),

price float

);

insert into product values(101,'Computer',900),(102,'Mouse',800);

select \* from product;

use aptech

select \* from sys.tables;

select \* from product;

create view productview as SELECT hsncode,prodname,price from dbo.product

where hsncode>1000;

update productview set price=500 where hsncode>1000

select \* from product

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

CREATE VIEW productviewbind WITH SCHEMABINDING

AS

SELECT hsncode,prodname,price from dbo.product

drop view productviewbind

-----after creating schemabinding you can not delete table--

select \* from productviewbind

ALTER TABLE dbo.Product ALTER COLUMN price bigint;

--------after change view you can delete---

alter view productviewbind with SCHEMABINDING

as

select hsncode,prodname from dbo.product

go

----Now you can delete-------------

drop table product

go

------------sp\_refreshview---------

create table customers

(

custid int,

custname varchar(50),

addresss varchar(60)

)

drop table customers;

insert into customers values(101,'James','Meerut'),(102,'Hari','Lucknow'),(102,'Amit','Kanpur')

--drop table customers;

----

--drop view vmcustomers

create view vmcustomers

as

select \* from customers

----

select \* from vmcustomers

----

alter table customers add age int

----not showing age in customers -----

select \* from vmcustomers

------exec sp\_refreshview -----------

EXEC sp\_refreshview 'vmcustomers'

-----------check I think age is added------

select \* from vmcustomers

--------stored procedure------------

select \* from customers

create procedure myprocedure

@cid int

as

begin

select \* from customers where custid=@cid

end

execute myprocedure 102

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

select \* from customers

create procedure myprocedure2

@cid int,

@result int output

as

begin

select \* from customers where custid=@cid

set @result=500

end

declare @addition int

execute myprocedure2 102,@addition output

select @addition

exec sp\_helptext myprocedure2

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

create procedure myprocedure3

@cid int,

@result int output

with encryption

as

begin

select \* from customers where custid=@cid

set @result=500

end

declare @addition int

execute myprocedure3 102,@addition output

select @addition

exec sp\_helptext myprocedure3;

sp\_tables;

sp\_changedbowner;

---cursor is an temporary areawork into database it is two type 1)implicit (DML) 2)explicit

select \* from customers;

DECLARE s1 CURSOR FOR SELECT \* FROM customers;

OPEN s1

close s1

DEALLOCATE s1

FETCH NEXT FROM s1;

---update cursor---

select \* from customers

DECLARE s2 CURSOR FOR update customers set custname='Hari' where custid=101;

OPEN s2

FETCH NEXT FROM s2;

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

FIRST is used to fetch only the first row from cursor table.

LAST is used to fetch only last row from cursor table.

NEXT is used to fetch data in forward direction from cursor table.

PRIOR is used to fetch data in backward direction from cursor table.

ABSOLUTE n is used to fetch the exact nth row from cursor table.

RELATIVE n is used to fetch the data in incremental way as well as decremental way.

Syntax : FETCH NEXT/FIRST/LAST/PRIOR/ABSOLUTE n/RELATIVE n FROM cursor\_name

FETCH FIRST FROM s1

FETCH LAST FROM s1

FETCH NEXT FROM s1

FETCH PRIOR FROM s1

FETCH ABSOLUTE 7 FROM s1

FETCH RELATIVE -2 FROM s1

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

sp\_cursor\_list;

sp\_indexes

-------stored procedure--------------

create procedure uspGetCustTerritory

as

begin

select \* from customers

end

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

create procedure uspGetSales

as

begin

select \* from product2

end

---------------show all tables

select \* from sys.tables;

--------nested procedure------

create procedure nestedprocedure

as

begin

exec uspGetCustTerritory

exec uspGetSales

end

-----------

exec nestedprocedure

execute sp\_executesql N'select @@nestlevel';

select @@NESTLEVEL;

exec('select @@nestlevel');

select name,object\_id,type,type\_desc from sys.tables;

select TABLE\_CATALOG,TABLE\_SCHEMA,TABLE\_NAME,TABLE\_TYPE from Information\_schema.tables;

select session\_id,login\_time,PROGRAM\_NAME from sys.dm\_exec\_sessions where login\_name='sa' and is\_user\_process=1;

---cursor is an temporary areawork into database it is two type 1)implicit (DML) 2)explicit

--use aptech database

select \* from customers;

DECLARE s1 CURSOR FOR SELECT \* FROM customers;

OPEN s1

FETCH NEXT FROM s1;

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

DECLARE @custname AS varchar(20);

SELECT @custname = 'Anita';

DECLARE employee\_cursor CURSOR FOR SELECT custname FROM customers where custid=101;

OPEN employee\_cursor;

FETCH NEXT FROM employee\_cursor

WHILE @@FETCH\_STATUS = 0

BEGIN

UPDATE customers

SET custname = @custname

WHERE CURRENT OF employee\_cursor;

FETCH NEXT FROM employee\_cursor

END;

deallocate employee\_cursor;

--A trigger is a special type of stored procedure that automatically runs when an event occurs in the

--database server. DML triggers run when a user tries to modify data through a data manipulation language

--(DML) event. DML events are INSERT, UPDATE, or DELETE statements on a table or view

use aptech;

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

-- Create Employee table

CREATE TABLE Employee

(

Id int Primary Key,

Name nvarchar(30),

Salary int,

Gender nvarchar(10),

DepartmentId int

)

GO

-- Insert data into Employee table

INSERT INTO Employee VALUES (1,'Pranaya', 5000, 'Male', 3)

INSERT INTO Employee VALUES (2,'Priyanka', 5400, 'Female', 2)

INSERT INTO Employee VALUES (3,'Anurag', 6500, 'male', 1)

INSERT INTO Employee VALUES (4,'sambit', 4700, 'Male', 2)

INSERT INTO Employee VALUES (5,'Hina', 6600, 'Female', 3)

select \* from employee;

--drop table employee--------------------------------------------

CREATE TRIGGER trInsertEmployee

ON Employee

FOR INSERT

AS

if (select salary from inserted) < 9000

BEGIN

PRINT 'YOU CANNOT PERFORM INSERT OPERATION'

ROLLBACK TRANSACTION

END

--------------Alter Trigger--------------------------------------

ALTER TRIGGER [dbo].[trInsertEmployee]

ON [dbo].[Employee]

FOR INSERT

AS

BEGIN

PRINT 'YOU CANNOT PERFORM INSERT OPERATION'

ROLLBACK TRANSACTION

END

--drop trigger trInsertEmployee--------------------------------------

INSERT INTO Employee VALUES (9, 'Saroj', 40000, 'Male', 2);

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

-------------------------create after----------------------------------------------------------------------------------

CREATE TABLE Employee2

(

Id int Primary Key,

Name nvarchar(30),

Salary int,

Gender nvarchar(10),

DepartmentId int

)

select \* from Employee2;

--drop table Employee2;

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

CREATE TRIGGER trInsertEmployee2

ON Employee

after INSERT

as

BEGIN

INSERT INTO Employee2 VALUES (4, 'Saroj', 4000, 'Male', 2);

print 'Value inserted successfully'

END

--drop trigger trInsertEmployee2

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

INSERT INTO Employee VALUES (10, 'James', 14000, 'Male', 2);

--------------------------------------similar inserted---------

CREATE TRIGGER trInsertEmployee3

ON Employee

after INSERT

as

SET NOCOUNT ON

declare @id int

declare @name varchar(50)

declare @salary float

declare @gender varchar(50)

declare @dept varchar(100)

select @id=i.Id,@name=i.Name,@salary=i.Salary,

@gender=i.Gender,@dept=i.DepartmentId from inserted i;

BEGIN

INSERT INTO Employee2 VALUES (@id,@name,@salary,@gender,@dept);

print 'Value inserted successfully'

END

--drop trigger trInsertEmployee3

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

INSERT INTO Employee VALUES (11, 'Pratik', 44000, 'Male', 1);

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

select \* from Employee2;

-----------------------------insted of trigger--------------------------------------------------------------------------------------------

--Instead Of triggers are executed instead of any of the Insert, Update or Delete operations.

--For example consider an Instead of Trigger for Delete operation, whenever a Delete is performed the Trigger will be executed first and if the Trigger deletes record then only the record will be deleted.

CREATE TRIGGER trInsertEmployeedel2

ON Employee

instead of delete

as

BEGIN

delete from employee2 where id in(select id employee from deleted)

print 'Value inserted successfully'

END

delete from Employee where Id=11;

select \* from Employee2;

--delete only employee2 not delete in employee

----------view trigger-----------------------------------------------------

create view empview

as

select id,name,salary from Employee

--drop view empview;

select \* from empview;

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

create trigger del\_empview

on empview

instead of delete

as

begin

delete from Employee2 where id in (select id from deleted)

end

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

delete from empview where id=4;

CREATE TRIGGER trUpdateEmployee

ON Employee

FOR UPDATE

AS

BEGIn

PRINT 'YOU CANNOT PERFORM UPDATE OPERATION'

ROLLBACK TRANSACTION

END

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

UPDATE Employee SET Salary = 90000 WHERE Id = 1

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

CREATE TRIGGER trDeleteEmployee2

ON Employee

FOR DELETE

AS

BEGIN

PRINT 'YOU CANNOT PERFORM DELETE OPERATION'

ROLLBACK TRANSACTION

END

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

DELETE FROM Employee WHERE Id = 1

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

DROP TRIGGER trDeleteEmployee

DROP TRIGGER trInsertEmployee

DROP TRIGGER trUpdateEmployee

---------triger for alter table

CREATE TRIGGER trAllDMLOperationsOnEmployee

ON Employee

FOR INSERT

AS

BEGIN

PRINT 'YOU CANNOT PERFORM INSERT OPERATION'

ROLLBACK TRANSACTION

END

--------------alter-----

--Create a Trigger that will restrict all the DML operations on the Employee table on MONDAY only.

--SUN DAY = 1

--MON DAY = 2

--TUE DAY = 3

--WED DAY = 4

--THU DAY = 5

--FRI DAY = 6

--SAT DAY = 7

ALTER TRIGGER trAllDMLOperationsOnEmployee

ON Employee

FOR INSERT, UPDATE, DELETE

AS

BEGIN

IF DATEPART(DW,GETDATE())= 5

BEGIN

PRINT 'DML OPERATIONS ARE RESTRICTED ON MONDAY'

ROLLBACK TRANSACTION

END

END

INSERT INTO Employee VALUES (14, 'Ritik', 44000, 'Male', 1);

--drop trigger trAllDMLOperationsOnEmployee ;

-----------Create a Trigger that will restrict all the DML operations on the Employee table before 1 pm.

ALTER TRIGGER trAllDMLOperationsOnEmployee

ON Employee

FOR INSERT, UPDATE, DELETE

AS

BEGIN

IF DATEPART(HH,GETDATE()) < 13

BEGIN

PRINT 'INVALID TIME'

ROLLBACK TRANSACTIONs

END

END

INSERT INTO Employee VALUES (14, 'Ritik', 44000, 'Male', 1);

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

USE AdventureWorks2012;

GO

declare @find varchar(30)='Man%';

--declare @find varchar(30);

--set @find=Man%';

SELECT p.lastname,p.Firstname,ph.phonenumber from Person.Person as p join

Person.Personphone as ph on p.BusinessEntityid=ph.businessEntityID

where lastname like @find;

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

USE AdventureWorks2012;

GO

Declare @var1 nvarchar(30);

declare @var2 varchar(40)='Unnamed Company';

SELECT @var1=name from sales.store where BusinessEntityID=292;

select @var1 as 'Company Name',@var2;

--SELECT \* from sales.store

Use AdventureWorks2012

GO

BEGIN TRANSACTION;

IF @@TRANCOUNT=0

BEGIN

SELECT FIRSTNAME,MIDDLENAME FROM PERSON.Person WHERE FirstName = 'syed';

ROLLBACK TRANSACTION;

PRINT N'ROLLING BACK THE TRANSACTION TWO TIMES WOULD CAUSE AN ERROR.';

END;

ROLLBACK TRANSACTION;

PRINT N'ROLLED BACK THE TRANSACTION.';

GO

-----------while loop----------

declare @flag int

set @flag=10

while (@flag<=95)

begin

if @flag%2=0

print @flag

set @flag=@flag+1

continue;

end

go

------synonyms------------------

use AdventureWorks2012;

go

create synonym sny2

for aptech.customers

go

select \* from sys.synonyms;

select \* from dbo.sny;

select \* from sny2;

-----------

select power(5,2);

select round(256.3146,1);

select @@TOTAL\_WRITE;

select @@TOTAL\_READ;

select @@TOTAL\_ERRORS;

select GETDATE();

select '2'\*'2';

select 2\*2;

SELECT CONVERT(int, 25.65);

SELECT CAST(25.65 AS int)+60;

SELECT CAST('2017-08-25' AS datetime);

SELECT ISDATE('2017-08-25');

select CHECKSUM(null);

Select checksum('SQL', 'Server', 'Rider');

select checksum('James');

--create function---------

use AdventureWorks2012;

go

if OBJECT\_ID (N'Sales.Cust',N'IF') is not null

drop function Sales.Cust;

go

create function sales.customers()

returns table

as

return

(

select \* from Sales.Customer

);

--create function 2-----------

use AdventureWorks2012;

go

CREATE FUNCTION fudf\_GetEmployee()

RETURNS TABLE

AS

RETURN (SELECT \* FROM Sales.SalesOrderDetail)

--run function----

select \* from fudf\_GetEmployee();

---------alter function--------------

use AdventureWorks2012

go

ALTER FUNCTION product\_calculation(

@quantity INT,

@price DEC(10,2),

@discount DEC(10,2)

)

RETURNS DEC(10,2)

AS

BEGIN

RETURN (@quantity \* @price ) - @discount;

END;

SELECT dbo.product\_calculation(2, 500, 50) AS net\_sales;

---------Window over-----------

use AdventureWorks2012

go

select salesorderid,ProductID,OrderQty,sum(orderqty) over

(partition by salesorderid) as Total,max(orderqty) over(partition by salesorderid)as Maximum

from Sales.SalesOrderDetail where ProductID in (776,773);

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

use AdventureWorks2012

go

select customerid,StoreID,rank() over(order by storeid desc)

as rankall from sales.Customer;

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

select productid,Shelf,Quantity,sum(quantity) over(partition by productid

order by locationid rows between unbounded preceding and current row)

as Quantity from production.ProductInventory;

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

select \* from production.ProductInventory;

--the NTILE() function results the groups of two sizes with the difference by one

CREATE TABLE geeks\_demo (

ID INT NOT NULL );

INSERT INTO geeks\_demo(ID)

VALUES(1), (2), (3), (4), (5), (6), (7), (8), (9), (10);

SELECT \* FROM geeks\_demo;

SELECT ID,NTILE (5) OVER (ORDER BY ID) Group\_number

FROM geeks\_demo;

----row number()-----

use AdventureWorks2012

go

select SalesQuota,Bonus,ROW\_NUMBER() over

(order by businessEntityid) as RowNumber from Sales.SalesPerson;

select \* from Sales.SalesPerson;

----row number and rank and dense\_rank------

SELECT BusinessEntityID,TerritoryID, ROW\_NUMBER() OVER(ORDER BY territoryid) RowNumber,

RANK() OVER(ORDER BY territoryid) Ranks,DENSE\_RANK() OVER(ORDER BY territoryid) D\_Ranks

FROM Sales.SalesPerson;

---offset time function-------------

create table test

(

col\_date\_offset datetimeoffset

);

go

insert into test values('1998-09-20 7:45:50.71345 -5:00');

go

select switchoffset (col\_date\_offset,'-08:00') from test;

go

--drop table test;

---fetch values---------

select col\_date\_offset from test;

-------datetimeoffsetfromparts---------

select DATETIMEOFFSETFROMPARTS (2022,12,31,14,23,23,0,12,0,7) as Result;

------date function-----

select sysdatetime(),SYSDATETIMEOFFSET(),SYSUTCDATETIME();

--------LEAD() function will allows to access data of the following row, or the row after the subsequent row, and continue on.

--up one by one value in next column

use AdventureWorks2012

go

SELECT BusinessEntityID, TerritoryID,

LEAD (TerritoryID,1) OVER (ORDER BY territoryid) AS next\_marks

FROM Sales.SalesPerson;

select \* from Sales.SalesPerson;

----------first\_value-------------

use AdventureWorks2012

go

select name,listprice,FIRST\_VALUE(name) over(order by listprice asc)

as lessExpensive from Production.Product where ProductSubcategoryID=37;

select name,listprice from Production.Product;

select \* from Production.Product;

--------------------- last value----------------

SELECT DISTINCT LAST\_VALUE (businessEntityiD)

OVER (ORDER BY businessEntityiD ASC

RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)

AS "HIGHEST"

FROM Sales.SalesPerson;

select \* FROM Sales.SalesPerson;

--------transaction------------------

use AdventureWorks2012;

go

declare @tranName varchar(30);

select @tranName='FirstTransaction';

BEGIN TRANSACTION @tranName;

DELETE From HumanResources.JobCandidate where JobCandidateid=13;

--rollback transaction

select \* From HumanResources.JobCandidate

BEGIN TRANSACTION;

go

DELETE From HumanResources.JobCandidate where JobCandidateid=12;

go

commit transaction;

go

--commit can not be rollback

BEGIN TRANSACTION deletecandidate

with mark N'Deleting a Job Candidate'

go

DELETE From HumanResources.JobCandidate where JobCandidateid=9;

go

commit transaction deletecandidate;

go

--rollback transaction deletecandidate

--select \* From HumanResources.JobCandidate

--error in transaction-----------

CREATE TABLE Product (

Product\_id INT PRIMARY KEY,

Product\_name VARCHAR(40),

Price INT,

Quantity INT

)

--Next, execute the below scripts to insert data into this table:

INSERT INTO Product VALUES(111, 'Mobile', 10000, 10),

(112, 'Laptop', 20000, 15),

(113, 'Mouse', 300, 20),

(114, 'Hard Disk', 4000, 25),

(115, 'Speaker', 3000, 20);

BEGIN TRANSACTION

INSERT INTO Product VALUES(115,'Speaker', 3000, 25)

-- Check for error

IF(@@ERROR > 0)

BEGIN

ROLLBACK TRANSACTION

END

ELSE

BEGIN

COMMIT TRANSACTION

END

--exception

Begin tran

Begin try

INSERT INTO Product VALUES(115,'Speaker', 3000, 25)

commit

End try

begin catch

rollback

End catch

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

--Two global temp tables with sample data for demo purpose

CREATE TABLE ##TableA (

ID INT IDENTITY,

Val CHAR(1)

)

GO

INSERT INTO ##TableA (Val)

VALUES ('A'), ('B')

GO

CREATE TABLE ##TableB(

ID INT IDENTITY,

Val CHAR(1)

)

GO

INSERT INTO ##TableB (Val)

VALUES ('C'), ('D')

GO

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

-- run this in query window 1

BEGIN TRANSACTION

--1

UPDATE ##TableA

SET Val = 'E'

WHERE ID = 1

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

WAITFOR DELAY '00:00:07'

--3

UPDATE ##TableB

SET Val= N'G'

WHERE ID = 1

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

COMMIT

SELECT Val, GETDATE() AS CompletionTime FROM ##TableA WHERE ID=1

SELECT Val, GETDATE() AS CompletionTime FROM ##TableB WHERE ID=1

-----------------------\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*----------------------------------------------

-- run this in query window 2

BEGIN TRANSACTION

--2

UPDATE ##TableB

SET Val = N'F'

WHERE ID = 1

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

WAITFOR DELAY '00:00:07'

--4

UPDATE ##TableA

SET Val = N'H'

WHERE ID = 1

COMMIT

SELECT Val, GETDATE() AS CompletionTime FROM ##TableA WHERE ID=1

SELECT Val, GETDATE() AS CompletionTime FROM ##TableB WHERE ID=1

------------------------------------#######################################

-- run this in query window 1

BEGIN TRANSACTION

SELECT @@SPID AS FirstTransactionProcessID

SELECT ID

FROM ##TableB WITH (UPDLOCK)

WHERE ID=1

--1

UPDATE ##TableA

SET Val = 'E'

WHERE ID = 1

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

WAITFOR DELAY '00:00:07'

--3

UPDATE ##TableB

SET Val= N'G'

WHERE ID = 1

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

COMMIT

SELECT Val, GETDATE() AS CompletionTime FROM ##TableA WHERE ID=1

SELECT Val, GETDATE() AS CompletionTime FROM ##TableB WHERE ID=1

---------------------------------------###############################################################

BEGIN TRANSACTION

--2

SELECT @@SPID AS SecondTransactionProcessID

EXEC sp\_lock

UPDATE ##TableB

SET Val = N'F'

WHERE ID = 1

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

WAITFOR DELAY '00:00:07'

--4

UPDATE ##TableA

SET Val = N'H'

WHERE ID = 1

COMMIT

SELECT Val, GETDATE() AS CompletionTime FROM ##TableA WHERE ID=1

SELECT Val, GETDATE() AS CompletionTime FROM ##TableB WHERE ID=1

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

Begin try

DECLARE @num int;

SELECT @num=217/0;

END Try

BEGIN CATCH

PRINT 'Error occurred, undable to divide by 0'

END CATCH;

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

USE AdventureWorks2012;

GO

BEGIN TRY

SELECT 217/0;

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber ,

ERROR\_SEVERITY() AS ErrorSeverity,

ERROR\_LINE() AS ErrorLine,

ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

GO

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

select \* from Production.Product;

USE AdventureWorks2012;

GO

BEGIN TRANSACTION;

BEGIN TRY

DELETE FROM Production.Product Where ProductID=999;

END TRY

BEGIN CATCH

SELECT

ERROR\_SEVERITY() AS ErrorSeverity,

ERROR\_NUMBER() AS ErrorNumber,

ERROR\_STATE() AS ErrorState,

ERROR\_PROCEDURE() AS ErrorProcedure,

ERROR\_LINE() AS ErrorLine,

ERROR\_MESSAGE() AS ErrorMessage;

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

END CATCH

IF @@TRANCOUNT >0

COMMIT TRANSACTION;

GO

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

use AdventureWorks2012;

GO

BEGIN TRY

UPDATE HumanResources.EmployeePayHistory SET PayFrequency=4

WHERE BusinessEntityID=1;

END TRY

BEGIN CATCH

IF @@ERROR=547

PRINT N'Check Constraint Violation has occured';

END CATCH

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

Select \* from HumanResources.EmployeePayHistory;

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

RAISERROR (N'This is an error message %s - %d',5,1,N'Serial Number',23);

go

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

RAISERROR (N'This is an error message %s %d',10,1,N'Serial Number',23);

GO

RAISERROR (N'%7.3s',10,1,N'Hello World');

GO

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

BEGIN TRY

SELECT 217/0;

END TRY

BEGIN CATCH

SELECT ERROR\_STATE() AS ErrorState;

END CATCH

GO

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

BEGIN TRY

SELECT 217/0;

END TRY

BEGIN CATCH

SELECT ERROR\_SEVERITY() AS ErrorSeverity;

END CATCH

GO

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

USE AdventureWorks2012;

GO

IF OBJECT\_ID ('ups\_Example','p')IS NOT NULL

DROP PROCEDURE usp\_Example;

GO

CREATE PROCEDURE usp\_Example

AS SELECT 217/0;

GO

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

BEGIN TRY

EXECUTE usp\_Example;

END TRY

BEGIN CATCH

SELECT ERROR\_PROCEDURE()

END CATCH

GO

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

BEGIN TRY

EXECUTE usp\_Example

END TRY

BEGIN CATCH

SELECT ERROR\_PROCEDURE() AS

ErrorProcedure;

END CATCH;

GO

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

BEGIN TRY

SELECT 217/0;

END TRY

BEGIN CATCH

SELECT ERROR\_NUMBER() AS ErrorNumber;

END CATCH;

GO

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

USE AdventureWorks2012;

GO

BEGIN TRY

SELECT \* from product;

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber,

ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

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

IF OBJECT\_ID (N'sp\_Example',N'P') IS NOT NULL

DROP PROCEDURE sp\_Example;

GO

CREATE PROCEDURE sp\_Example

AS

SELECT \* from products;

GO

BEGIN TRY

EXECUTE sp\_Example

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber,

ERROR\_MESSAGE() AS ErrorMessage;

END CATCH;

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

use AdventureWorks2012;

GO

begin transaction

CREATE TABLE dbo.TestRethrow

(

ID INT PRIMARY KEY

);

BEGIN TRY

INSERT dbo.TestRethrow(ID) VALUES(1);

INSERT dbo.TestREthrow(ID) VALUES(1);

END TRY

BEGIN CATCH

if @@TRANCOUNT>0

PRINT 'Error Primary key rule conflict';

throw;

rollback;

END CATCH;

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

DROP TABLE dbo.TestRethrow;

select \* from dbo.TestRethrow;

---------json-----------only supported in sqlserver 2016

Select \* from HumanResources.EmployeePayHistory for json auto;