SQL SERVER LECTURE QUERIES COMPLETED

```
use Northwind

sp_renamedb 'NorthwindNew','Northwind'


create table Students_Tbl(

S_ID int not null primary key identity(1,1),

S_Name varchar(50),

S_Course varchar(50),

S_Teacher varchar(50)

)

insert into Students_Tbl values ('Ahmed','ERP System Admin','Aon Ali')

insert into Students_Tbl values ('Ali','ERP SAP ABAP','Faraz')

insert into Students_Tbl values ('Kashif','ERP SAP FICO','Faisal Masood')

insert into Students_Tbl values ('Hamza','ERP SAP Sales Distribution','Faizan Asher')

insert into Students_Tbl values ('Taha','ERP SAP Inventory Management','Muhammad Ahtesham')

insert into Students_Tbl values ('Sajid','ERP Implementaion Methodoligies','Humayun Qureshi')

insert into Students_Tbl values ('Akram','ERP SAP Human Resource Management','Faisal Masood')

insert into Students_Tbl values ('Shahid','ERP SAP Production Planning','Muhammad Sajid')

select * from Students_Tbl


SELECT * from Teachers_Tbl


drop table Teachers_Tbl


/* create table Teachers_Tbl

(

foreign key T_ID int not null identity(1,1),
```

```
T_Name varchar(50),
T_Course varchar(50),
T_Salary int

foreign key(T_ID) REFERENCES
Students_Tbl(S_ID)
)
*/

--Cascading Referential Integrity Constraints In SQL Server - No Action - Cascade
Create Table Customers_Tbl
(
CustomerID int primary key not null identity(1,1),
CustomerName varchar(50),
CustomerContact varchar(50),
CustomerAddress varchar(50)
)
drop table Orders__Tbl
CREATE TABLE Orders__Tbl (
  OrderID INT PRIMARY KEY NOT NULL IDENTITY(1,1),
  CustomerID INT,
  OrderDate DATE,
  CONSTRAINT FK_CustomerID FOREIGN KEY (CustomerID)
    REFERENCES Customers_Tbl(CustomerID) ON DELETE CASCADE
);
select * from Customers_Tbl
select * from Orders__Tbl

insert into Orders__Tbl values(1,GETDATE())
```

```
insert into Orders__Tbl values(2,GETDATE())
insert into Orders__Tbl values(3,GETDATE()
insert into Orders__Tbl values(4,GETDATE())
insert into Orders__Tbl values(5,GETDATE())
insert into Orders__Tbl values(6,GETDATE())
insert into Orders__Tbl values(7,GETDATE())
insert into Orders__Tbl values(8,GETDATE())
insert into Orders__Tbl values(9,GETDATE())
insert into Orders__Tbl values(10,GETDATE())
insert into Orders__Tbl values(11,GETDATE())

insert into Customers_Tbl values('Ahmed','237484939','Karachi,Pakistan')
insert into Customers_Tbl values('Ali','237483439','Islamabad,Pakistan')
insert into Customers_Tbl values('Akram','237484849','Lahore,Pakistan')
insert into Customers_Tbl values('Rashid','930484939','Peshawar,Pakistan')
insert into Customers_Tbl values('Younas','237485678','Quetta,Pakistan')
insert into Customers_Tbl values('Nabeel','098484939','Multan,Pakistan')
insert into Customers_Tbl values('Azan','657484939','Sialkot,Pakistan')
insert into Customers_Tbl values('Asif','237454939','Rawalpindi,Pakistan')
insert into Customers_Tbl values('Afaq','237484769','Gilgit,Pakistan')
insert into Customers_Tbl values('Faiq','237484009','Kashmir,Pakistan')
insert into Customers_Tbl values('Iqra','290984939','SWAT,Pakistan')

delete from Customers_Tbl where CustomerID = 11

select  * from Customers where CustomerID < 1
select * From Orders where OrderID < 1
select * from OrderDetails where OrderDetailID < 1
```

```
select A.CustomerID,B.EmployeeID,C.OrderID,C.ProductID,
A.CustomerName,A.City,A.Country,
B.OrderDate,C.Quantity From Customers A
inner join Orders B
on A.CustomerID = B.CustomerID inner join OrderDetails C
on c.ProductID = B.CustomerID

select OrderID From Orders
union
select OrderID from OrderDetails

select OrderID From Orders
union all
select OrderID from OrderDetails

select OrderID From Orders  -- **INTERSECT**: Returns rows that appear in both result sets.
intersect
select OrderID from OrderDetails

select OrderID From Orders  --- **EXCEPT**: Returns rows that appear in the first result set but not in the second
except
select OrderID from OrderDetails

--AGGREGATE FUNCTIONS:

select max(Price) as [Max Price] from Products
select min(Price) as [Min Price] from Products
select avg(Price) as [Avg Price] from Products
```

```
select max(Price) as [Second Highset Price] from Products
where Price < (select MAX(Price) -1 from Products )
select Price from Products

select top 3 Price from Products order by Price desc
select Price from Products

--GROUP BY and HAVING:
 SELECT ProductID, SUM(Price) AS [Total Price]
 FROM Products
 GROUP BY ProductID
 HAVING SUM(Price) > 123

 select * from Products order by Price desc

--Views in SQL Server:

CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';

create view Vw_Top10ProductsWithHighPrice
as
select top 10 Price from Products order by Price desc

create view Vw_Top10ProductsWithHighPriceNoDuplication
as
```

```
select distinct top 10 Price from Products order by Price desc

select * from Vw_Top10ProductsWithHighPrice
select * from Vw_Top10ProductsWithHighPriceNoDuplication

--LIKE Operator:
select * from Employees where FirstName like 'A%'
select * from Employees where FirstName like '%A%'

--Subquery in SQL Server:
select max(Price) as [Second Highset Price] from Products
where Price < (select MAX(Price) -1 from Products )
select Price from Products

 SELECT CustomerID, CustomerName  --Co-Related Sub Query
  FROM Customers
  WHERE CustomerID IN (
    SELECT TOP 5 CustomerID
    FROM Orders
    GROUP BY CustomerID
    ORDER BY COUNT(*) DESC
  );

--InnerJoin
SELECT *
FROM Orders A
INNER JOIN OrderDetails B ON A.OrderID = B.OrderID;

--Left Join
```

```
SELECT *
FROM Orders A
LEFT JOIN OrderDetails B ON A.OrderID = B.OrderID;

--RIGHT join
SELECT *
FROM Orders A
RIGHT JOIN OrderDetails B ON A.OrderID = B.OrderID;

--FULL JOIN
SELECT *
FROM Orders A
FULL JOIN OrderDetails B ON A.OrderID = B.OrderID;

--CROSS JOIN
SELECT *
FROM Orders A
CROSS JOIN OrderDetails B ;

--PERCENT
SELECT TOP(11) PERCENT * FROM Employees order by EmployeeID asc

--select into
select * into BackupOrderDetails_Tbl from Orders
select * from BackupOrderDetails_Tbl

select City into BackupCityEmployees_Tbl from Customers
select * from BackupCityEmployees_Tbl
```

```
select * into LondonCustomers_Tbl from Customers
where City = 'London'
select * from LondonCustomers_Tbl

select * from OrderDetails
where ProductID in (select ProductID from Products where Quantity between 10 and 40)

--insert into with select statement
insert into  LondonCustomers_Tbl select * from Customers where City = 'Brazil'

--CHANGING OR RENAMING DATABASE NAME AND TABLE NAME IN SQL
exec sp_rename 'LondonCustomers_Tbl','Tbl_LondonCustomers'
exec sp_renamedb'Northwind','New_Northwind'
exec sp_renamedb'New_Northwind','Northwind'

---create stored procedure in sql server
create proc GetEmployeeSalary
@EmployeeID int,
@Salary decimal(10,2)output
as
begin
declare @EmployeeName nvarchar(100),
select @EmployeeName = EmpName
from Employees_Tbl
where EmpID = @EmployeeID

set @Salary = 125000 + LEN(@EmployeeName) * 100
RETURN
end
```

```
DECLARE @EmpSalary DECIMAL(10, 2);
EXEC GetEmployeeSalary @EmployeeID = 101, @Salary = @EmpSalary OUTPUT;

/* Scalar Functions:
Scenario: Calculating the total price of items in an order including tax. */
create function CalculateTotalPrice(@OrderID int)
returns decimal(10,2)
as
begin

declare @TotalPrice decimal(10,2)
select @TotalPrice = sum(A.Price * B.Quantity) * 1.1 --Tax
from Products A inner join OrderDetails B
on A.ProductID = B.ProductID
group by A.Price

return @TotalPrice
end;

SELECT dbo.CalculateTotalPrice(1001) AS TotalPriceWithTax;

/*Aggregate Functions:
Scenario: Finding the total number of products sold in each category. */

select Cat.CategoryName, count(prd.ProductID) as [Total Products Sold]
from Products prd
INNER JOIN OrderDetails OD on prd.ProductID = OD.ProductID
inner join Categories Cat on prd.CategoryID = Cat.CategoryID
group by CategoryName
```

```
/* Table-Valued Functions:
Scenario: Retrieving all orders placed by a customer. */

alter function GetOrdersByCustomers(@CustomerID int)
returns Table
as
return
(
select * From Orders where CustomerID = @CustomerID
)
SELECT * FROM dbo.GetOrdersByCustomers(90);

/* System Functions:
Scenario: Getting the current date and time of a server. */

SELECT GETDATE() AS CurrentDateTime;

/*  Ranking Functions:
Scenario: Ranking employees based on their sales performance.*/
select OrderID, ProductID, Quantity,
ROW_NUMBER() over (order by Quantity desc) as [Row Number]
from OrderDetails

/* String Functions:
**Scenario:** Extracting the domain name from email addresses.
 */
create table Student_Id_Name_Email(
StudentID int primary key not null identity(1,1),
```

```
StudentEmail varchar(50),
StudentCourse varchar(50)
)
insert into Student_Id_Name_Email values ('hamzamughal@gmail.com','SAP ABAP')
insert into Student_Id_Name_Email values ('hamzajabbar403@gmail.com','Machine Learning')
insert into Student_Id_Name_Email values ('jr.developer.98@gmail.com','Data Science')
insert into Student_Id_Name_Email values  ('hamzamughal@gmail.com','Deep Learning')
insert into Student_Id_Name_Email values  ('hamza.razvi.786@gmail.com','Generative AI')
UPDATE Student_Id_Name_Email SET StudentEmail = 'hamzamughal8096@gmail.com' where StudentID = 4
SELECT * FROM Student_Id_Name_Email

select StudentEmail, SUBSTRING(StudentEmail,CHARINDEX('@',StudentEmail) + 1,
LEN(StudentEmail) - CHARINDEX('@',StudentEmail)) AS DOMAIN
FROM  Student_Id_Name_Email

/* Date and Time Functions:
**Scenario:** Calculating the age of customers. */
drop table Customers_ID_Name_BirthDate_Gender
create table  Customers_ID_Name_BirthDate_Gender(
CustomerID int primary key not null identity(1,1),
CustomerName varchar(50),
CustomerBirthDate datetime,
CustomerGender varchar(50)
)
truncate table Customers_ID_Name_BirthDate_Gender
insert into Customers_ID_Name_BirthDate_Gender values ('Ahemd',convert(varchar(50),'12-06-2001',107),'Male')
insert into Customers_ID_Name_BirthDate_Gender values ('Akram',convert(varchar(50),'17-07-2002',107),'Male')
```

```
insert into Customers_ID_Name_BirthDate_Gender values ('Sadia',convert(varchar(50),'02-02-2007',107),'Female')

insert into Customers_ID_Name_BirthDate_Gender values ('Abida',convert(varchar(50),'28-08-2011',107),'Female')


select * from Customers_ID_Name_BirthDate_Gender

select CustomerID,CustomerName,CustomerGender,DATEDIFF(YEAR,CustomerBirthDate,GETDATE()) as CustomerAge

from Customers_ID_Name_BirthDate_Gender


/* . Logical Functions:

**Scenario:** Categorizing sales into high or low based on sales amount.

*/

select * from Products

select A.ProductID, A.ProductName, B.Quantity,A.Price,

iif(A.Price > 100, 'High','Low')as ProductCategory

from Products A

inner join OrderDetails B

on A.ProductID = B.ProductID


select A.ProductID, A.ProductName,A.Price, iif(A.Price > 60, 'High','Low')as ProductCategory

from Products A


/* Conversion Functions:

**Scenario:** Converting a string to an integer.

 */

select '234' as StringValue, cast('234' as int) as IntergerValue


/* Mathematical Functions:
```

```
 */
select ABS(-10.8) as AbsoluteValue
select ABS(-10.8-12) as AbsoluteValue
select ABS(-10.8*12) as AbsoluteValue
select ABS(-10.8-12-3) as AbsoluteValue

/* Scenario:** Calculating the distance between two geographical points.*/
DECLARE @Latitude1 DECIMAL(9,6) = 40.730610;
DECLARE @Longitude1 DECIMAL(9,6) = -73.935242;
DECLARE @Latitude2 DECIMAL(9,6) = 34.052235;
DECLARE @Longitude2 DECIMAL(9,6) = -118.243683;

SELECT SQRT(POWER(@Latitude2 - @Latitude1, 2) + POWER(@Longitude2 - @Longitude1, 2))
AS Distance;

/* Inline Table-Valued Function */
create function GetCustomersByCity(@CityName varchar(50))
returns table
as
return
(
select * from Customers where City = @CityName
)
select * from dbo.GetCustomersByCity('London')

/* System Functions:
System functions are built-in functions provided by SQL Server for various purposes.*/
SELECT GETDATE() AS CurrentDateTime;
```

```
--stored procedure with output parameters in SQL Server:
create procedure GetEmployeeSalary
@EmployeeID int,
@Salary decimal(10,2) output
as
begin
declare @EmployeeName nvarchar(50)
select @EmployeeName = EmpName from Employees_Tbl where EmpID = @EmployeeID
set @Salary = 50000 + LEN(@EmployeeName) * 100
return
end
DECLARE @EmpSalary DECIMAL(10, 2);
EXEC GetEmployeeSalary @EmployeeID = 101, @Salary = @EmpSalary OUTPUT;

/* Spatial Functions:**
 - `STDistance()`: Returns the shortest distance between two spatial objects
 */
CREATE TABLE SpatialTableA (
  ID INT PRIMARY KEY not null identity(1,1),
  GeometryColumnA GEOMETRY
);

CREATE TABLE SpatialTableB (
  ID INT PRIMARY KEY not null identity(1,1),
  GeometryColumnB GEOMETRY
);

INSERT INTO SpatialTableA
VALUES
```

```
  (geometry::Point(10, 20, 0)); -- Inserting a point at coordinates (10, 20)

INSERT INTO SpatialTableB
VALUES
  (geometry::Point(15, 25, 0)); -- Inserting a point at coordinates (15, 25)
SELECT
  A.ID AS A_ID,
  B.ID AS B_ID,
  A.GeometryColumnA.STDistance(B.GeometryColumnB) AS Distance
FROM
  SpatialTableA A
CROSS JOIN
  SpatialTableB B;

select * from SpatialTableA,SpatialTableB

/*XML Functions:**
 - `XMLAGG()`: Aggregates a sequence of XML values into a single XML value.
*/
CREATE TABLE XmlTable (
  ID INT PRIMARY KEY,
  XmlData XML
);
INSERT INTO XmlTable (ID, XmlData)
VALUES
  (1, '<Root><Item>Value1</Item><Item>Value2</Item></Root>'),
  (2, '<Root><Item>Value3</Item><Item>Value4</Item></Root>');
SELECT XMLAGG(XmlData) AS AggregatedXml
FROM XmlTable;
```

/* System Functions:**

- `OBJECT_NAME()`: Returns the name of the schema-scoped object from the object identification number.

***/**

```
SELECT OBJECT_NAME(OBJECT_ID) AS ObjectName FROM sys.tables;
```

/*Logical Functions:**

- `CASE`: Evaluates a list of conditions and returns one of multiple possible result expressions.

***/**

```
  SELECT
    CASE
      WHEN Price > 50 THEN 'High'
      ELSE 'Low'
    END AS PriceStatus
  FROM Products;
```

/*Mathematical Functions:**

- `CEILING()`: Returns the smallest integer greater than or equal to a numeric expression.

***/**

```
SELECT CEILING(5.7) AS RoundedUpValue;
SELECT CEILING(9.3) AS RoundedUpValue;
```

/*File System Functions:**

- `FILE_EXISTS()`: Checks whether a file exists on the file system.

***/**

```
SELECT FILE_EXISTS('C:\Users\Administrator\Desktop\Coursera Financial Aid.txt') AS FileExists;
```

```
/*Compression Functions:**
  - `COMPRESS()` : Compresses a string using the GZIP algorithm.
 */
SELECT COMPRESS('Hello World') AS CompressedString;

/*Collation Functions:**
  - `COLLATIONPROPERTY()` : Returns property information about a collation.
 */
 SELECT COLLATIONPROPERTY('Latin1_General_CI_AS', 'CodePage') AS CodePage;

/*ENCRYPTBYKEY()**:
  - This function encrypts data using a symmetric key.
 */
 --ENCRYPTBYKEY(key_id, { 'cleartext' | @cleartext }, [add_authenticator, [authenticator]])
 DECLARE @encrypted VARBINARY(8000);
   OPEN SYMMETRIC KEY SymmetricKey1 DECRYPTION BY CERTIFICATE MyCertificate;
   SET @encrypted = ENCRYPTBYKEY(KEY_GUID('SymmetricKey1'), 'SensitiveData');
   CLOSE SYMMETRIC KEY SymmetricKey1;
--Inline Table-Valued Function (Inline TVF):
create function dbo.MyInlineTVF_1(
@num1 int
)
returns table
as
return
(
select CustomerID,City,Country from Customers where CustomerID = @num1
)
select * from dbo.MyInlineTVF_1(76)
```

```
/*Multi-Statement Table-Valued Function (Multi-Statement TVF):
 */
ALTER FUNCTION dbo.MyMultiStatementTVF
(
   @Param1 INT
)
RETURNS @ResultTable TABLE
(
   City VARCHAR(50),
   Country VARCHAR(50)
)
AS
BEGIN
   INSERT INTO @ResultTable (City, Country)
   SELECT City,Country
   FROM Customers
   WHERE CustomerID = @Param1;
   RETURN;
END;
select * from dbo.MyMultiStatementTVF(7)

use Northwind_BackUp_DataBase
 alter PROCEDURE GetEmployeeDetails
    @EmployeeID INT
  AS
  BEGIN
    SELECT * FROM Employees_BackUp WHERE EmployeeID = @EmployeeID;
  END;
```

```
  Exec GetEmployeeDetails @EmployeeID = 10;

  select * from Employees_BackUp

  select * from Customers_BackUp


ALTER PROCEDURE GetProductDetails

   @SupplierID varchar(50),

   @CategoryID varchar(50)

AS

BEGIN

   SELECT * FROM Products_BackUp WHERE SupplierID = @SupplierID  AND CategoryID =
@CategoryID;

END;

EXEC GetProductDetails @SupplierID= 3, @CategoryID = 7


---DML Triggers In SQL Server---

/*Types of DML Triggers**:

  - **INSERT Trigger**: Executes after an `INSERT` operation is performed on the table.

  - **UPDATE Trigger**: Executes after an `UPDATE` operation is performed on the table.

  - **DELETE Trigger**: Executes after a `DELETE` operation is performed on the table.*/

  create trigger Tr_Insert_Products_BackUp

  ON Products_BackUp

  after insert

  AS

  BEGIN

   select * from Products_BackUp

  END

  insert into Products_BackUp values ('French Outback Larger',12,7,'24 - 355 ml bottles',200)


   drop trigger Tr_Update_Products_BackUp
```

```
 create trigger Tr_Delete_Products_BackUp
 on Products_BackUp
 after delete
 as
 begin
  select * from Products_BackUp
 end
 delete from Products_BackUp where ProductID = 7


create trigger Tr_Update_Products_BackUp
 on Products_BackUp
 after update
 as
 begin
 select * from inserted
 select * from deleted
 select * from Products_BackUp
 end
 update Products_BackUp set SupplierID = 23 where ProductID = 80

 select * from Employees_BackUp

 CREATE TRIGGER AuditEmployeesChanges
ON Employees
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
  IF EXISTS(SELECT * FROM inserted) -- Check if rows were inserted or updated
```

```
    BEGIN
        INSERT INTO EmployeeAudit (EmployeeID, Action, DateModified)
        SELECT EmployeeID, 'INSERT/UPDATE', GETDATE() FROM inserted;
    END
    ELSE IF EXISTS(SELECT * FROM deleted) -- Check if rows were deleted
    BEGIN
        INSERT INTO EmployeeAudit (EmployeeID, Action, DateModified)
        SELECT EmployeeID, 'DELETE', GETDATE() FROM deleted;
    END
END
select * from Employees
select * from EmployeeAudit

insert into Employees values ('Lionel','Messi',GETDATE(),'EmpID11.pic','He is football champion')

disable trigger AuditEmployeesChanges on Employees

--INSTEAD OF` triggers in SQL Server
create view MyView
as
select
A.CustomerID,A.CustomerName,A.City,A.Country,
B.EmployeeID,B.FirstName,B.LastName,B.Notes,
C.*,D.OrderDetailID,D.OrderID from Customers A
inner join Employees B
on A.CustomerID = B.EmployeeID inner join
Products C on B.EmployeeID = C.ProductID
inner join OrderDetails D on D.OrderDetailID = C.ProductID
```

```
select * from MyView

create trigger insteadofinsert_Categories
on Categories
instead of insert
as
begin
print 'You are not allowed to insert any record in this table!'
select * from inserted
end
insert into Categories values ('Beverages','Soda Mint')

create trigger InsteadOfInsert_Products
on Products
instead of insert
as
begin
print'You Are Not Allowed To Insert Any Record In This Table!!!'
select * from inserted
end
select * from Products
insert into Products values ('Cup',3,9,'10 boxes x 20 bags',234)

alter trigger InsteadOfDelete_Products
on Products
instead of delete
as
begin
```

```
print 'You Are Not Allowed To Delete Any Record In This Table!!!'
select * from deleted
end
delete from Products where ProductID = 77
select * from Products


create trigger InsteadOfUpdate_Products
on Products
instead of update
as
begin
print 'You Are Not Allowed To Update Any Record In This Table!!!'
select * from inserted
select * from deleted
end

update Products set Price = 16 where ProductID = 3

--DDL Triggers In SQL Server - CREATE ALTER DROP - Triggers In SQL - SQL Triggers - SQL
CREATE TRIGGER Tr_Table
ON ALL SERVER
FOR CREATE_TABLE
AS
BEGIN
   PRINT 'You are not allowed to create table'
   ROLLBACK; -- Optionally rollback the transaction
END;
drop table TR_TABLE
```

```
CREATE TABLE TR_TABLE(

Tbl_ID int not null primary key identity,
Tbl_Name varchar(50))
disable trigger Tr_Table on all server

CREATE TRIGGER Tr_Index
ON ALL SERVER
FOR CREATE_INDEX
AS
BEGIN
   PRINT 'You are not allowed to create Index'
   ROLLBACK; -- Optionally rollback the transaction
END;
DISable Trigger Tr_Index on all server

CREATE INDEX Ix_Product
ON Products (ProductID ASC);
disable trigger Tr_Index on all server

CREATE TRIGGER Tr_View
ON ALL SERVER
FOR CREATE_VIEW
AS
BEGIN
   PRINT 'You are not allowed to create View'
   ROLLBACK; -- Optionally rollback the transaction
END;
disable trigger Tr_View on all server
```

```
create view Vw_Prdoucts
as
select * from Products
disable trigger Tr_View on all server
disable trigger Tr_Index on all server

CREATE TRIGGER DDLTriggerExample
ON DATABASE
FOR CREATE_TABLE, ALTER_TABLE, DROP_TABLE
AS
BEGIN
 DECLARE @EventType NVARCHAR(100);
 SET @EventType = EVENTDATA().value('(/EVENT_INSTANCE/EventType)[1]', 'NVARCHAR(100)');
 DECLARE @ObjectName NVARCHAR(255);
 SET @ObjectName = EVENTDATA().value('(/EVENT_INSTANCE/ObjectName)[1]',
'NVARCHAR(255)');
 DECLARE @EventTime DATETIME;
 SET @EventTime = GETDATE();
 INSERT INTO DDLLog (EventType, ObjectName, EventTime)
 VALUES (@EventType, @ObjectName, @EventTime);
END;
disable trigger  DDLTriggerExample on database

create trigger Trigger1
on Products
after insert
as
begin
select * from Products
```

```
end
GO

create trigger Trigger2
on Products
after insert
as
begin
select * from Employees
end
GO

create trigger Trigger3
on Products
after insert
as
begin
select * from Customers
end
GO

insert into Products values ('Cake',9,2,'8 Pond 10 pieces','12500')

--GUID In SQL - Globally Unique Identifier - SQL GUID - GUID In SQL Server - SQL
create table [GUID](
 ID UNIQUEIDENTIFIER PRIMARY KEY DEFAULT NEWID(),
 [Name] varchar(50),
 [Contact] varchar(50)
)
```

```
insert into [GUID] values (default,'Muhammad Hamza','03086998765')
insert into [GUID] values (default,'Muhammad Hammad','03086548765')
select * from [GUID]

create table CompositePrimaryKey(
ProductID INT ,
OrderID INT ,
 Quantity INT,
 OrderDate DATE
 primary key(OrderID, ProductID)
)
insert into CompositePrimaryKey values (1,3,4,Getdate())
insert into CompositePrimaryKey values (2,5,3,Getdate())
select * from CompositePrimaryKey

--String Functions In SQL Server - SQL String Functions - SQL Tutorial - SQL Server
SELECT CONCAT('Hello', ' ', 'World') AS Result;
SELECT len('Hello') AS [Length];
SELECT SUBSTRING('Hello World', 7, 5) AS [Substring];
SELECT SUBSTRING('World SQL Server', 7, 5) AS [Substring];
select upper('hello world!') as [Upper];
select lower('Hello World!') as [Lower];
SELECT REPLACE('Hello World', 'World', 'Universe') AS ReplacedString;
SELECT REPLACE('Hello SQL', 'SQL', 'SQL Server') AS ReplacedString;
SELECT CHARINDEX('World', 'Hello World') AS Position;


-- Creating Clustered & Non Clustered Indexes in SQL Server:
create clustered index Ix_Shippers_ShipperID
```

```
on Shippers (ShipperID asc)

create nonclustered index Ix_Shippers_ShipperID
on Shippers (ShipperID asc)

select * from Shippers where ShipperID  = 33
insert into Shippers values ('TradeEx Shippers','(098)-987-032')
select * from Shippers

-- Creating  Uniques & Non Unique Indexes in SQL Server:
create unique index Ix_Catogries_CategoryID --Uniques Index
on Categories (CategoryID asc)

create  index Ix_Catogries_CategoryName  --Non-Unique Index
on Categories (CategoryName asc)

select * from Categories

create table Unique_And_NoN_Unique_Index
(
PrdID int,
SupplierID INT
)
create unique index Ix_PrdID
on Unique_And_NoN_Unique_Index (PrdID asc)

create index Ix_SupplierID
on Unique_And_NoN_Unique_Index (SupplierID asc)
```

```
insert into Unique_And_NoN_Unique_Index values (1,3)
insert into Unique_And_NoN_Unique_Index values (2,4)
insert into Unique_And_NoN_Unique_Index values (3,3)
insert into Unique_And_NoN_Unique_Index values (3,8)
insert into Unique_And_NoN_Unique_Index values (4,8)

select * from Unique_And_NoN_Unique_Index

--COMPUTED COLUMNS OR CALCULATED COLUMNS IN SQL
drop table Computed_Columns_Table
create table Computed_Columns_Table(
StudentID int primARY key not null identity(1,1),
Student_Name varchar(50),
Degree_Program varchar(50),
Total_Marks int,
Obtained_Marks float,
[percentage] AS (Obtained_Marks) /Total_Marks * 100 persisted)

truncate table Computed_Columns_Table

insert into Computed_Columns_Table values ('Ahmed','BS-CS',800,600)
insert into Computed_Columns_Table values ('Ahmed','BS-CS',800,687.5)
insert into Computed_Columns_Table values ('Iqra','BS-CS',800,608.75)
insert into Computed_Columns_Table values ('Akram','BS-CS',800,475.5)
insert into Computed_Columns_Table values ('Javed','BS-CS',800,511)
insert into Computed_Columns_Table values ('Fariha','BS-CS',800,407.75)
insert into Computed_Columns_Table values ('Mahnoor','BS-CS',800,687)
insert into Computed_Columns_Table values ('Fabiha','BS-CS',800,751)
insert into Computed_Columns_Table values ('Taha','BS-CS',800,705)
```

```
insert into Computed_Columns_Table values ('Arsalan','BS-CS',800,755)
insert into Computed_Columns_Table values ('Hamza','BS-CS',800,784)
insert into Computed_Columns_Table values ('Talha','BS-CS',800,780)
insert into Computed_Columns_Table values ('Fahad','BS-CS',800,749)
insert into Computed_Columns_Table values ('Aneesa','BS-CS',800,451.25)
insert into Computed_Columns_Table values ('Maria','BS-CS',800,608.75)

select * from Computed_Columns_Table
select *, DENSE_RANK() over(order by [Percentage] desc) as [Rank] from
Computed_Columns_Table

create index Ix_Perecentage_Computed_Columns_Table
on Computed_Columns_Table ([percentage] )

--Cube And Rollup Command In SQL Server - Cube - Rollup - SQL Server - SQL Tutorial
select Student_Name,[percentage], avg([percentage]) as Avg_Percentage
from Computed_Columns_Table
group by cube(Student_Name,[percentage])

select Student_Name,[percentage], avg([percentage]) as Avg_Percentage
from Computed_Columns_Table
group by rollup(Student_Name,[percentage])

--Grouping Sets In SQL Server - SQL Grouping Sets - SQL Server - SQL Tutorial - SQL
SELECT C.City, C.Country,
    SUM(P.Price) AS Total_Price,
    AVG(P.Price) AS Avg_Price,
    MIN(P.Price) AS Min_Price,
    MAX(P.Price) AS Max_Price
```

```
FROM Customers C
INNER JOIN Products P ON P.ProductID = C.CustomerID
GROUP BY GROUPING SETS (
  (C.Country, C.City),
  (C.Country),
  (C.City),
  ()
);

--MERGE STATEMENT IN SQL SERVER
--MERGE INTO, USING, ON
-- WHEN MATCHED, WHEN NOT MATCHED BY SOURCE, WHEN NOT MATCHED BY TARGET
select * from Target_Table
SELECT *
FROM INFORMATION_SCHEMA.TABLES

SELECT TABLE_SCHEMA, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_CATALOG = 'Northwind';

--MERGE STATEMENT IN SQL SERVER
drop table  Target_Table
drop table  Source_Table
create table Target_Table(
Id uniqueidentifier primary key  default newid(),
FirstName varchar(50),
LastName varchar(50)
)
create table Source_Table(
```

```
Id uniqueidentifier primary key  default newid(),
FirstName varchar(50),
LastName varchar(50)
)
insert into Target_Table values (default,'Ahemd','Khan')
insert into Target_Table values (default,'Emaan','Khan')
insert into Target_Table values (default,'Ali','Ahmed')

insert into Source_Table values (default,'Ahemd','Khan')
insert into Source_Table values (default,'Emaan','Khan')
insert into Source_Table values (default,'Ali','Ahmed')
insert into Source_Table values (default,'Farooq','Ahmed')

        merge into Target_Table AS [Target]
        using Source_Table AS [Source]
        on [Target].Id = [Source].Id
        when matched then
        UPDATE SET [Target].FirstName = [Source].FirstName, [Target].LastName =
[Source].LastName
        when not matched by target then
        insert (Id,FirstName,LastName) values ([Source].Id,[Source].FirstName,
[Source].LastName)
        when not matched by source then
        delete;

--Transactions In SQL | ACID Properties In SQL - SQL Transactions - SQL Server - SQL
begin tran
select * from Products where Price > 60
commit tran
```

```
begin tran
select * from Products where Price > 60
insert into Products values ('Printing Cups',8,7,'45-67 Pieces', 568)
rollback
commit tran

select * from Target_Table
select * from Source_Table
select * from Employees_Tbl
select * FROM Categories where CategoryID < 1
select * FROM Employees where EmployeeID < 1
select * from Products where ProductID < 1
select * from OrderDetails where OrderDetailID < 1
select * from Orders where OrderID < 1
select * from Customers where CustomerID < 1



create trigger Insert_trigger_Customers
on Customers
for insert
as
begin
   declare @CustomerID int
  select @CustomerID from inserted
  insert into Orders(CustomerID,OrderID)
  values (@CustomerID, ORIGINAL_LOGIN())
```

```
  print'Insert Trigger Executed'
end

-- creating stored procedure
create  procedure MyProcedure
@id int
as
begin
 if @id > 0
 begin
 select * from Customers where @id = CustomerID
 end
 ELSE
 BEGIN
 select * from Customers
 end
end

create procedure SelectAllCustomers
as
select * from Customers

create procedure SelectAllCustomers1
as
select * from Customers
select ProductName, ProductID from Products
order by (
case
   when Price > 39 then 'Price is less under 39'
```

```
 else ProductName
end
)
create procedure SelectCustomers @City nvarchar(30)
as
 select * from Customers where City = @City

create procedure SelectCustomers_new @City nvarchar(30),  @Postalcode nvarchar(30)
as
 select * from Customers where City = @City and PostalCode = @Postalcode

create procedure SelectCustomersnew @City nvarchar(30), @Country nvarchar(30)
as
 select * from Customers where City = @City and Country = @Country

-- creating stored procedure (Query)
create procedure sp_CustomerSegmentation (@CountryFilter varchar(20) null)
as
begin
 select
   CustomerID,
    CustomerName,
    ContactName,
    Address,
    City,
    PostalCode,
    Country,
  Case
   when City = 'MainOfficeCity' then 'local'
```

```
  when Country = @CountryFilter then 'Domestic'
  else 'International'
 end as CustomerSegment
 from Customers
 where (@CountryFilter IS NULL OR Country = @CountryFilter)
end

create procedure spGetCustomersDataByCountry
@City varchar(50),
@CustomersCount int output
as
begin
select  @CustomersCount = Count(CustomerID) from Customers
where City = @City
end

create trigger tr_Vw_CustomerDetails_delete
on Vw_CustomerDetails
instead of delete
as
begin
   delete from Orders where CustomerID in (select CustomerID from deleted)
   delete from Customers where CustomerID in (select CustomerID from deleted)
end

create view Vw_CustomerDetails
as
select Orders.CustomerID, Orders.OrderDate, Orders.ShipperID,
Customers.CustomerName,Customers.ContactName,
```

```
Customers.City,Customers.Country

from Customers

inner join Orders

on

Orders.CustomerID = Customers.CustomerID


SELECT SYSTEM_USER [Administrator], GETDATE() as [Date Time];


create function fn_GetEmployeesByNames(@FirstName varchar(100))

returns @mytable table (employee_id int, employee_firstname varchar(100),
employee_lastname varchar(100))

as

begin

   insert into @mytable

  select EmployeeID,FirstName,LastName from Employees where FirstName = @FirstName

  return

end



create trigger Insert_trigger

on Employees

for insert

as

begin

   declare @EmployeeID int

  select @EmployeeID from inserted

  insert into Orders(CustomerID,EmployeeID)

  values (@EmployeeID, ORIGINAL_LOGIN())
```

```
  print'Insert Trigger Executed'
end

create trigger Insert_trigger_Customers
on Customers
for insert
as
begin
   declare @CustomerID int
  select @CustomerID from inserted
  insert into Orders(CustomerID,OrderID)
  values (@CustomerID, ORIGINAL_LOGIN())

  print'Insert Trigger Executed'
end

create trigger tr_instead_of_Products_Orders_Record
on vW_Products_Orders_Record
instead of delete
as
begin
   delete from Orders where OrderDate in (select OrderDate from deleted)
end

create trigger tr_Vw_CustomerDetails_delete
on Vw_CustomerDetails
instead of delete
as
begin
```

```
    delete from Orders where CustomerID in (select CustomerID from deleted)
    delete from Customers where CustomerID in (select CustomerID from deleted)
end

create view Vw_CustomerDetails
as
select Orders.CustomerID, Orders.OrderDate, Orders.ShipperID,
Customers.CustomerName,Customers.ContactName,
Customers.City,Customers.Country
from Customers
inner join Orders
on
Orders.CustomerID = Customers.CustomerID

create view Vw_Employee_Order_Deatails
as
select A.EmployeeID,B.ProductName, a.OrderDate,B.Price from Products as B
right join Orders as A
on B.ProductID = A.EmployeeID

create view vW_Products_Orders_Record
as
select A.OrderID, convert(varchar(50),A.OrderDate,107) as [Order
Date],A.EmployeeID,A.ShipperID,
B.ProductID,B.ProductName,B.CategoryID,B.Unit,B.Price
from Orders as A
inner join Products as B
on A.EmployeeID = B.ProductID
```

```
create view Vw_Suppliers_Orders
as
select A.*, B.OrderDate,B.OrderID from Suppliers as A
left join Orders as B
on A.SupplierID = B.ShipperID
where City in ('Londona','New Orleans','Tokyo','Sydney')

(-- [Age]>=(18))
--([Age]>=(18))

create view [Products Higher Than Avg Price]
as
select  ProductName, Price, ProductID from Products
where Price > (select avg(Price) from Products)

CREATE VIEW PRODUCTS_DATA
AS
SELECT ProductID from products

create procedure SelectAllCustomers1
as
select * from Customers
select ProductName, ProductID from Products
order by (
case
   when Price > 39 then 'Price is less under 39'
 else ProductName
end
)
```

```
create view Vw_Employee_Order_Deatails
as
select A.EmployeeID,B.ProductName, a.OrderDate,B.Price from Products as B
right join Orders as A
on B.ProductID = A.EmployeeID

create view vW_Products_Orders_Record
as
select A.OrderID, convert(varchar(50),A.OrderDate,107) as [Order
Date],A.EmployeeID,A.ShipperID,
B.ProductID,B.ProductName,B.CategoryID,B.Unit,B.Price
from Orders as A
inner join Products as B
on A.EmployeeID = B.ProductID

create  trigger tr_Shippers_Audit_for_delete
on Shippers
after delete
as
begin
   declare @id int
  select @id = ShipperID from deleted
  insert into tbl_Shippers_Audit values ('Existing Shipper id ' + cast(@id as varchar(50)) +
  ' is deleted at ' + cast(getdate() as varchar(50)))
end

CREATE trigger tr_Shippers_audit_for_insert
on Shippers
after insert
```

```
as
begin
    declare @id  int
  select @id = ShipperID from inserted
  insert into tbl_Shippers_Audit values ('Shipper with id ' + cast(@id as varchar(50)) +
  ' is inserted at ' + cast(getdate()as varchar(50)))
end

create trigger tr_Shippers_for_delete
on Shippers
after delete
as
begin
select * from deleted
end

CREATE trigger tr_Shippers_for_insert
on Shippers
after insert
as
begin
   print 'Record is Successfully Inserted In Shippers Table! '
 select * from inserted
end

create view Vw_Suppliers_Orders
as
select A.*, B.OrderDate,B.OrderID from Suppliers as A
left join Orders as B
```

```
on A.SupplierID = B.ShipperID
where City in ('Londona','New Orleans','Tokyo','Sydney')

create trigger tr_Shippers_Audit_for_delete
on Shippers
after delete
as
begin
   declare @id int
  select @id = ShipperID from deleted
  insert into tbl_Shippers_Audit values ('Existing Shipper id ' + cast(@id as varchar(50)) +
  ' is deleted at ' + cast(getdate() as varchar(50)))
end

CREATE trigger tr_Shippers_audit_for_insert
on Shippers
after insert
as
begin
   declare @id int
  select @id = ShipperID from inserted
  insert into tbl_Shippers_Audit values ('Shipper with id ' + cast(@id as varchar(50)) +
  ' is inserted at ' + cast(getdate()as varchar(50)))
end

CREATE trigger tr_tbl_Shippers_Audit_for_delete
on tbl_Shippers_Audit
after delete
```

```
as
begin
print 'Record is deleted succesfully'
select * from deleted
end


sp_depends [Products]
sp_helptext [MatrixDB]

--TRY CATCH OR ERROR HANDLING IN SQL SERVER
begin try
select 10/0
end try
begin catch
print 'Division with zero is not possible'
end catch

begin try
update Books set BookName = 123 where BookID = 2
end try

begin catch
select
ERROR_NUMBER() [Error Number],
ERROR_SEVERITY() [Error Severity],
ERROR_LINE() [Error LINE],
ERROR_STATE() [Error State],
ERROR_PROCEDURE() [Error Procedure]
```

```
end catch

--TRANSACTIONS WITH TRY CATCH IN SQL SERVER
select * from Books

begin try
begin tran
insert into Books values (6,'Data Science','Artificial Intelligence',5600,'NVIDIA')
insert into Books values (7,'Machine Learning','Artificial Intelligence',6000,'NVIDIA')
insert into Books values (6,'Deep Learning','Artificial Intelligence',7000,'NVIDIA')
commit tran
print 'Trasaction Has Been Done Successfully!'
end try

begin catch
rollback tran
print 'Trsaction Failed!'
end catch

begin tran
update Books set BookID = 8 where BookName = 'Deep Learning'
commit tran --permanent data save

select * from Books1

begin try
begin tran
insert into Books1 values ('Data Science', 'Artificial Intelligence', 7800,'AI Group')
insert into Books1 values ('Machine Learning', 'Artificial Intelligence', 9800,'AI Group')
```

```
insert into Books1 values ('Deep Learning', 'Artificial Intelligence', 8000,'AI Group')
insert into Books1 values ('Big Data', 'Artificial Intelligence', 8900,'AI Group')
commit tran
print 'Transaction Has Been Done Successfully!'
end try

begin catch
rollback tran
select ERROR_MESSAGE()
print 'Transaction Failed!'
end catch

ALTER TABLE Books1
ADD BookID INT IDENTITY(1,1);

ALTER TABLE Books1
ADD CONSTRAINT PK_Books1 PRIMARY KEY (BookID);

ALTER TABLE Books1
DROP COLUMN BookID        ; -- This assumes you want to drop the duplicate column

IF NOT EXISTS (SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS WHERE
TABLE_NAME = 'Books1' AND CONSTRAINT_TYPE = 'PRIMARY KEY')
BEGIN
  ALTER TABLE Books1
  ADD CONSTRAINT PK_Books1 PRIMARY KEY (BookID);
END

truncate table Books1
```

```
--TEMPORARY TABLES / LOCAL TEMPORARY TABLE IN SQL SERVER
create table #EmpData
(
EmpId int primary key not null identity(1,1),
EmpName varchar(50),
EmpDepart varchar(50)
)
insert into #EmpData values ('Ahmed','IT')
insert into #EmpData values ('Ali','Finance')
insert into #EmpData values ('Ashir','Marketing')

select * from #EmpData

select name from tempdb..sysobjects
where name like '%EmpData%'

create proc Pr_LOCALEmpData
as
begin
create table #EmpData
(
EmpId int primary key not null identity(1,1),
EmpName varchar(50),
EmpDepart varchar(50)
)
insert into #EmpData values ('Ahmed','IT')
insert into #EmpData values ('Ali','Finance')
insert into #EmpData values ('Ashir','Marketing')
```

```
select * from #EmpData
end

exec Pr_LOCALEmpData

--GLOBAL TEMPORARY TABLES IN SQL SERVER
create table ##EmpData
(
EmpId int primary key not null identity(1,1),
EmpName varchar(50),
EmpDepart varchar(50)
)
insert into ##EmpData values ('Ahmed','IT')
insert into ##EmpData values ('Ali','Finance')
insert into ##EmpData values ('Ashir','Marketing')

select * from ##EmpData

--COALESCE FUNCTION IN SQL SERVER
--return only first not null value

select coalesce(null,null,'Ali', null,'Ahmed')
select coalesce('Ali', null,'Ahmed')

create table FullNmaeTbl(
Id int primary key identity(1,1),
First_Name varchar(50),
Last_Name varchar(50),
```

)

insert into FullNmaeTbl values (null,'Ahmed')

insert into FullNmaeTbl values ('Ahmed',null)

insert into FullNmaeTbl values (null,null)

insert into FullNmaeTbl values ('Ali','Ahmed')

select * from FullNmaeTbl

select Id, coalesce(First_Name,Last_Name) from FullNmaeTbl

--Difference Between Coalesce And IsNull Function in Sql Server

--ISNULL FUNCTION WILL TAKE ONLY TWO ARGS BUT IN COALESCE FUNCTION WILL TAKE MULTIPLE ARGS

select Id, coalesce(First_Name,Last_Name) AS[COALESCE] from FullNmaeTbl

select Id, ISNULL(First_Name,Last_Name) AS [IS NULL] from FullNmaeTbl

--Cast Function In SQL Server - SQL Cast Function - SQL Tutorial - SQL Server

select cast(23.56 as int) as Value;

declare @Num1 decimal = 45.89

select CAST(@Num1 as int) Value

select @Num1

select cast('2024-03-13' as datetime) [Date Time]

select * from Employees

select FirstName + ' - ' + cast(BirthDate as varchar) as [Name + BirthDate]from Employees

--Convert Function - Difference Between Cast & Convert Function In SQL Server - SQL

select convert(int,8998.88)

declare @num2 decimal = 989.899

select convert(int,@num2) [Value]

select @num2

select GETDATE()

select convert(nvarchar(50), GETDATE(), 106)

select convert(nvarchar(50), GETDATE(), 104)

select convert(nvarchar(50), GETDATE(), 0)Fe

--1- Cursor In SQL Server - SQL Cursor - What Is Cursor In SQL - SQL Tutorial - SQL

-- Implicit --Created automatically by SQL Server

-- Explicit -- Created Manually By User

-- Methods Of Cursor

--Next --> Used For select Next Value

--Prior --> Used For select Previous Value

--First --> Used For select First Value

--Last --> Used For select Last Value

--Obsolute n --> Used For select specific Value

--Relative n --> it will take both negative or Positive values like 300-2 = 298 , 300+2 = 302, it will return or select 298 or 302 record

---- Types Of Cursor

--Declaring Cursor --> A cursor is declared by defining the sql statement

--Opening Cursor --> Cursor is opened for storing data retreived form the result set

--Fetching Cursor --> When a cursor is opened, rows can be fetched from the cursor one by one or in a block to the data manipulation

--closing cursor --> The Cursor should be closed explicitly after data manipulation

--De-allocating Cursor --> Cursors should be de-allocated to delete cursor definition and release all the system resources associated with the cursor

-- we can use Cursors in 2 ways:

-- 1) With Cursor Variables

-- 2) Without Cursor Variables

--WITHOUT CURSOR VARIABLES

```
declare mycursor cursor scroll for select * from Employees
open mycursor
fetch first from mycursor
fetch next from mycursor
fetch prior from mycursor
fetch prior from mycursor
fetch prior from mycursor
fetch absolute 8 from mycursor
fetch relative 2 from mycursor
fetch relative -2 from mycursor
close mycursor
deallocate mycursor
```

```
declare Two_Table_Cursor cursor scroll for select
A.CustomerID, A.OrderID,A.ShipperID,
B.LastName,B.FirstName,B.Notes from Orders A
inner join Employees B
on A.EmployeeID = B.EmployeeID
open Two_Table_Cursor
fetch first from Two_Table_Cursor
fetch next from Two_Table_Cursor
fetch next from Two_Table_Cursor
fetch prior from Two_Table_Cursor
fetch absolute 9 from Two_Table_Cursor
fetch relative 2 from Two_Table_Cursor
```

```
fetch relative -2 from Two_Table_Cursor

close Two_Table_Cursor

deallocate Two_Table_Cursor



--WITH CURSOR VARIABLES

declare Variable_Cursor cursor scroll for

select A.EmployeeID,B.FirstName,B.Notes from Orders A

inner join Employees B

on A.EmployeeID = B.EmployeeID

order by A.EmployeeID asc

declare @EmployeeID int, @FirstName varchar(50), @Notes varchar(50)

open Variable_Cursor

fetch first from Variable_Cursor into @EmployeeID,@FirstName,@Notes

print 'EmployeeID is: ' + cast(@EmployeeID as varchar(50)) + ' And EmployeeName is : ' +
cast(@FirstName as varchar(50)) + ' And The Note is : ' + cast(@Notes as varchar(50))

fetch next from Variable_Cursor into @EmployeeID,@FirstName,@Notes

print 'EmployeeID is: ' + cast(@EmployeeID as varchar(50)) + ' And EmployeeName is : ' +
cast(@FirstName as varchar(50)) + ' And The Note is : ' + cast(@Notes as varchar(50))

fetch last from Variable_Cursor into @EmployeeID,@FirstName,@Notes

print 'EmployeeID is: ' + cast(@EmployeeID as varchar(50)) + ' And EmployeeName is : ' +
cast(@FirstName as varchar(50)) + ' And The Note is : ' + cast(@Notes as varchar(50))

fetch prior from Variable_Cursor into @EmployeeID,@FirstName,@Notes

print 'EmployeeID is: ' + cast(@EmployeeID as varchar(50)) + ' And EmployeeName is : ' +
cast(@FirstName as varchar(50)) + ' And The Note is : ' + cast(@Notes as varchar(50))

fetch absolute 7 from Variable_Cursor into @EmployeeID,@FirstName,@Notes

print 'EmployeeID is: ' + cast(@EmployeeID as varchar(50)) + ' And EmployeeName is : ' +
cast(@FirstName as varchar(50)) + ' And The Note is : ' + cast(@Notes as varchar(50))

fetch relative -1 from Variable_Cursor into @EmployeeID,@FirstName,@Notes

print 'EmployeeID is: ' + cast(@EmployeeID as varchar(50)) + ' And EmployeeName is : ' +
cast(@FirstName as varchar(50)) + ' And The Note is : ' + cast(@Notes as varchar(50))
```

```
fetch relative 2 from Variable_Cursor into @EmployeeID,@FirstName,@Notes

print 'EmployeeID is: ' + cast(@EmployeeID as varchar(50)) + ' And EmployeeName is : ' +
cast(@FirstName as varchar(50)) + ' And The Note is : ' + cast(@Notes as varchar(50))

close Variable_Cursor

deallocate Variable_Cursor


--OVER CLAUSE WITH PARTITION BY IN SQL SERVER

select * from Employees_Tbl


select A.EmpName,A.Gender,A.Salary,Genders.Gender_Total,

Genders.Max_Salary,Genders.Min_Salary,Genders.Avg_Salary

from Employees_Tbl_A

inner join

(select Gender, Count(*) as Gender_Total,

Max(Salary) as Max_Salary,

Min(Salary) as Min_Salary,

Avg(Salary) as Avg_Salary

from Employees_Tbl

group by Gender) as Genders

on A.Gender = Genders.Gender


select EmpName, Gender, Salary,count(Gender)  over (partition by Gender) as [Gender_Total]
from Employees_Tbl


select EmpName, Gender, Salary,

count(Gender) over (partition by Gender) as [Gender Total],

max(Salary) over (partition by Salary) as [Max Salary],

min(Salary) over (partition by Salary) as [Min Salary],

avg(Salary) over (partition by Salary) as [Avg Salary]
```

```
from Employees_Tbl



select Gender,
Count(Gender) as Gender_Total,
Max(Salary) as Max_Salary,
Min(Salary) as Min_Salary,
Avg(Salary) as Avg_Salary
from Employees_Tbl
group by Gender

-- windowing --- it opens the same aggegating function or partition reesult of every record

--Retrieving Last Generated Identity Column Value in SQL Server - Scope_Identity VS @@identity
create table customer_tbl (
CustomerID int primary key identity(1,1),
CustomerName varchar(50)
)
insert into customer_tbl values ('Ahmed')
insert into customer_tbl values ('Ali')
insert into customer_tbl values ('Farhan')
insert into customer_tbl values ('Hamza')

select SCOPE_IDENTITY(); --Function returns the last idenetity created in the same session and the same scope or connection.
select @@IDENTITY;     -- it returns the last identity created in the same session and in any scope.
```

select IDENT_CURRENT('customer_tbl'); --it will returns the last identity created for a specific table or view in any session.

```
create table cust_details(
CustomerID int primary key identity(1,1),
Date_Time datetime
)
insert into cust_details values(GETDATE())
select * from cust_details
truncate table cust_details
truncate table customer_tbl

create trigger Tr_InsertForCustDetails
on customer_tbl
after insert
as
begin
insert into cust_details values (GETDATE())
end

select * from customer_tbl
select * from cust_details

--Row_Number Function In SQL Server - Row_Number With Partition By Clause
--syntax  ROW_NUMBER() OVER (ORDER BY COLUMN_NAME)
select * from FullTimeEmployees order by Id asc
select *, ROW_NUMBER() over (partition by Gender order by Salary asc )as Numebering
from FullTimeEmployees
```

```
select *, ROW_NUMBER() over (partition by Designation order by Salary asc )as Numebering
from FullTimeEmployees

--Rank And Dense_Rank Function In SQL Server - Rank VS Dense_Rank In SQL Server

create table Student_Marks (
StudentID int primary key not null identity(1,1),
StudentName varchar(50),
StudentGender varchar(50),
StudentAge int,
StudentMarks nvarchar(50)
)
insert into Student_Marks values('Ahmed','Male',18,'87.5')
insert into Student_Marks values('Alisha','Female',17,'73')
insert into Student_Marks values('Afsar','Male',18,'65.5')
insert into Student_Marks values('Farhan','Male',19,'87.5')
insert into Student_Marks values('Laiba','Female',18,'82.65')
insert into Student_Marks values('Naima','Female',17,'67')
insert into Student_Marks values('Ali','Male',18,'70')
insert into Student_Marks values('Fazal','Male',18,'80')
insert into Student_Marks values('Alina','Female',19,'70')
insert into Student_Marks values('Anas','Male',19,'60')
insert into Student_Marks values('M Talha','Male',19,'90')

select * from Student_Marks

--Rank Function ---> You don't pass any Args in Rank() Function -- It skips the Rank Sequence when similar numbers having in the table
select *, RANK() over (order by StudentMarks desc) as [StudentRank] from Student_Marks
```

```
select *, Rank() over (partition by StudentGender order by StudentMarks) as [StudentRank]
from Student_Marks


--Dense Rank --> It does not skips the Number Sequence when similar numbers are present in
the Table

select *, DENSE_RANK() over (order by StudentMarks desc) as [StudentRank] from
Student_Marks

select *, DENSE_RANK() over (partition by StudentGender order by StudentMarks desc) as
[StudentRank] from Student_Marks

select *, DENSE_RANK() over (order by StudentMarks) as [DenseRank], RANK() over (order by
StudentMarks) as [Rank]

from Student_Marks

select *, DENSE_RANK() over (partition by StudentGender order by StudentMarks) as
[DenseRank],

RANK() over (partition by StudentGender order by StudentMarks) as [Rank]

from Student_Marks


--Cross Apply & Outer Apply In SQL Server - Apply Operator in SQL Server


-- SQL query to create the TEACHER TABLE

CREATE TABLE TEACHER (

  teacher_id INT PRIMARY KEY identity(1,1) not null,

  teacher_name VARCHAR(255),

  teacher_gender VARCHAR(50),

  teacher_qual VARCHAR(255),

  teacher_sal INT

);


-- SQL query to create the STUDENT TABLE

CREATE TABLE STUDENT (

  std_id INT PRIMARY KEY identity(1,1) not null,
```

```
  std_name VARCHAR(255),
  std_gender VARCHAR(50),
  std_age INT,
  std_class INT,
  t_id INT,
  FOREIGN KEY (t_id) REFERENCES TEACHER(teacher_id)
);

-- SQL query to insert values into the TEACHER TABLE
INSERT INTO TEACHER  VALUES
('Asad', 'Male', 'BSIT', 28000),
('Khalid', 'Male', 'MBA', 29000),
('Amjad', 'Male', 'MPHIL', 30000),
('Adeel', 'Male', 'BSIT', 27000),
('Sahar', 'Female', 'BSC', 25000),
('Farukh', 'Male', 'BBA', 26000);

-- SQL query to insert values into the STUDENT TABLE
INSERT INTO STUDENT  VALUES
( 'Anas', 'Male', 22, 12, 2),
( 'Anum', 'Female', 21, 11, 3),
( 'Zain', 'Male', 23, 12, 1),
( 'Furqan', 'Male', 21, 10, 2),
( 'Saba', 'Female', 21, 11, 4),
( 'Amna', 'Female', 23, 12, 3);

select * from TEACHER
select * from STUDENT
```

```
-- Cross Apply & Outer Apply Functions only work wit Table Valued Functions

select * from TEACHER AS T

inner join STUDENT AS S

on T.teacher_id = s.t_id


select T.teacher_name, T.teacher_qual, S.std_name ,S.std_age

from TEACHER AS T

INNER JOIN STUDENT S

ON T.teacher_id = S.t_id



select T.teacher_name, T.teacher_qual, S.std_name ,S.std_age

from TEACHER AS T

LEFT JOIN STUDENT S

ON T.teacher_id = S.t_id


select T.teacher_name, T.teacher_qual, S.std_name ,S.std_age

from TEACHER AS T

LEFT JOIN STUDENT S

ON T.teacher_id = S.t_id


-- it will give error because function can't be used with joins thats why we use cross apply or outer apply function

select T.teacher_name, T.teacher_qual, S.std_name ,S.std_age

from TEACHER AS T

INNER JOIN fn_GetStudentsByTeacherID(T.teacher_id)

ON T.teacher_id = S.t_id
```

```
select ROW_NUMBER() over (partition by teacher_name ORDER BY teacher_id) from TEACHER

--Table Valued Function
alter function fn_GetStudentsByTeacherID(@TeacherID int)
returns table
as
return
(
select * from TEACHER where teacher_id = @TeacherID
)

select * from fn_GetStudentsByTeacherID(4)

select t.teacher_name, t.teacher_qual, s.std_name, s.std_age
from TEACHER as t
cross apply fn_GetStudentsByTeacherID(t.teacher_id) as s

select t.teacher_name, t.teacher_qual, s.std_name, s.std_age
from TEACHER as t
outer apply fn_GetStudentsByTeacherID(t.teacher_id) as s

ALTER FUNCTION fn_GetStudentsByTeacherID(@TeacherID INT)
RETURNS TABLE
AS
RETURN
(
  SELECT s.std_name, s.std_age
  FROM TEACHER t
```

```
    JOIN STUDENT s ON t.teacher_id = s.t_id
    WHERE t.teacher_id = @TeacherID
);


SELECT t.teacher_name, t.teacher_qual, s.std_name, s.std_age
FROM TEACHER t
CROSS APPLY fn_GetStudentsByTeacherID(t.teacher_id) s;

SELECT t.teacher_name, t.teacher_qual, s.std_name, s.std_age
FROM TEACHER t
OUTER APPLY fn_GetStudentsByTeacherID(t.teacher_id) s;

--cross apply like is just like a inner join just difference is that they join with table value function on right side as a result value
--outer apply just like lef join just difference is that they join with table value function on right side as a result value

--(Part-1) CTE in SQL - Common Table Expression In SQL - SQL CTE - CTE In SQL Server -
--it will give temporary result set as a output then we use this result set immediately with DML commands or with View s otherwise it will be destroyed.it will present till execution ended
--it improves readability while use with complex queries

-- SQL query to create the table with keys
CREATE TABLE Student_Tbl_For_CTE (
    Id INT PRIMARY KEY not null identity(1,1),
    [Name] VARCHAR(255),
    Gender VARCHAR(50),
    Age INT,
    [Standard] INT
```

```
);

insert into  Student_Tbl_For_CTE values ('Ahmed','Male',19,12)
insert into  Student_Tbl_For_CTE values ('Ali','Male',18,11)
insert into  Student_Tbl_For_CTE values ('Qasim','Male',17,10)
insert into  Student_Tbl_For_CTE values ('Alina','Female',18,11)
insert into  Student_Tbl_For_CTE values ('Aqsa','Female',15,9)
insert into  Student_Tbl_For_CTE values ('Javed','Male',18,12)
insert into  Student_Tbl_For_CTE values ('Tehreem','Female',19,12)
insert into  Student_Tbl_For_CTE values ('Fatima','Female',19,12)
insert into  Student_Tbl_For_CTE values ('Asad','Male',15,9)
insert into  Student_Tbl_For_CTE values ('Ayesha','Female',17,11)


select * from Student_Tbl_For_CTE

WITH NEW_CTE
AS
(
SELECT * FROM Student_Tbl_For_CTE where Gender = 'Male'
)
select count(*) from NEW_CTE

WITH NEW_CTE
AS
(
SELECT * FROM Student_Tbl_For_CTE where Gender = 'Male'
)
select * from NEW_CTE where Age > 15
```

```
select * from Student_Tbl_For_CTE

WITH NEW_CTE(std_id, std_name, std_class)
AS
(
 select Id, [Name],[Standard] from Student_Tbl_For_CTE
)
select std_id, std_name, std_class from NEW_CTE order by std_class asc

WITH NEW_CTE
AS
(
SELECT * FROM Student_Tbl_For_CTE
)
insert into Student_Tbl_For_CTE values ('Abbas','Male',22,14)


WITH NEW_CTE
AS
(
SELECT * FROM Student_Tbl_For_CTE
)
update Student_Tbl_For_CTE set [Name] = 'Asadullah' where Id = 9

WITH NEW_CTE
AS
(
SELECT * FROM Student_Tbl_For_CTE
```

```
)
delete from Student_Tbl_For_CTE where Id = 11

WITH NEW_CTE
AS
(
SELECT * FROM Student_Tbl_For_CTE
)
delete from NEW_CTE where Id in(15,16,17,18)
insert into NEW_CTE values ('Abbas','Male',22,14)
update NEW_CTE set [Name] = 'Muhammad Abbas Ali' where Id = 12

create view Vw_MyNewCTEView
as
with New_CTE
as
(
select * from Student_Tbl_For_CTE where [Standard] in (11,12,14)
)
select * from New_CTE
select * from Vw_MyNewCTEView

WITH NEW_CTE_1
AS
(
SELECT * FROM Student_Tbl_For_CTE WHERE [Standard] = 11
),
NEW_CTE_2
AS
```

```
(
SELECT * FROM Student_Tbl_For_CTE WHERE [Standard] = 12
)
SELECT * FROM NEW_CTE_1
UNION ALL
SELECT * FROM NEW_CTE_2

WITH NEW_CTE
AS
(
SELECT COUNT(*) AS [TOTAL NUMBER OF STUDENTS] from Student_Tbl_For_CTE_WHERE [Standard] = 11
)
SELECT * FROM NEW_CTE

--CTE IS CREATED BEFORE THE OUTER QUERY
-- SUB-QUERY IS CREATED AFTER THE OUTER QUERY

--Date & Time Functions In SQL - SQL Date & Time Functions - SQL Tutorial - SQL

--GetDate() -->  Returns the current DateTime
--sysDateTime() -->  Returns the current DateTime, returns 7 precision of seconds
--Current_TimeStamp() -->  Returns the current DateTime
--DateName() -->  Returns the name of the day, year, month etc from a given date
--DateDiff() -->  Returns the difference between two dates
--DateAdd() -->  Add or substracts  a specified time interval from a date
--DatePart()-->  Returns a single part of date/time
--Day()-->  Returns the day from given date
--Month()-->  Returns the Month from given date
```

```
--Year()-->     Returns the Year from given date

--IsDate()-->     CHECK if the expression is valid

select getdate()

select convert(varchar(50), GETDATE(),100)

select SYSDATETIME()

select CURRENT_TIMESTAMP

select DATENAME(MONTH, GETDATE())

select DATENAME(YEAR, GETDATE())

select DATENAME(DAY, GETDATE())

select DATENAME(HOUR, GETDATE())

select DATENAME(MINUTE, GETDATE())

select DATENAME(MONTH, GETDATE())

select DATENAME(SECOND, GETDATE())

select DATENAME(YEAR, 'january 05 2024')

select DATENAME(MONTH, 'january 05 2024')

select DATENAME(DAY, 'january 05 2024')

select DATENAME(YEAR, '1-05-2024')

select DATENAME(MONTH, '1-05-2024')

select DATENAME(DAY, '1-05-2024')

select DATEDIFF(YEAR, '04-01-2019',GETDATE())

select DATEDIFF(MONTH, '04-01-2019',GETDATE())

SELECT DATEDIFF(DAY,'04-01-2019',GETDATE())

SELECT DATEDIFF(DAY,'04-01-2019',CURRENT_TIMESTAMP)

SELECT DATEDIFF(HOUR,'04-01-2019',CURRENT_TIMESTAMP)

SELECT DATEDIFF(MINUTE,'04-01-2019',CURRENT_TIMESTAMP)

SELECT DATEDIFF(SECOND,'04-01-2019',CURRENT_TIMESTAMP)

select DATEADD(DAY, 2, GETDATE())

select DATEADD(DAY, -2, GETDATE())

select DATEADD(MONTH, 2, GETDATE())
```

select DATEADD(MONTH, -2, GETDATE())

select DATEADD(YEAR, 2, GETDATE())

select DATEADD(YEAR, -2, GETDATE())

select DATEADD(HOUR, 2, GETDATE())

select DATEADD(HOUR, -2, GETDATE())

select DATEADD(MINUTE, 2, GETDATE())

select DATEADD(MINUTE, -2, GETDATE())

select DATEADD(SECOND, 2, GETDATE())

select DATEADD(SECOND, -2, GETDATE())

select DATEPART(YEAR,GETDATE())

select DATEPART(MONTH,GETDATE())

select DATEPART(DAY,GETDATE())

select DATEPART(HOUR,GETDATE())

select DATEPART(MINUTE,GETDATE())

select DATEPART(SECOND,GETDATE())

SELECT DATENAME(MONTH,GETDATE())

SELECT DATENAME(YEAR,GETDATE())

SELECT DATENAME(MINUTE,GETDATE())

select DAY(GETDATE())

select MONTH(GETDATE())

select YEAR(GETDATE())