**#1**

**Server Authentication**

* Windows Authentication
* SQL Authentication

**#2**

**System Database**

1. **Master**
2. **Model**
3. **Msdb**
4. **Tempdb**

**#3**

**Whenever we create a database on SQL Server, it will generate two database files are**

1. .mdf (master data file)
2. .ldf(Log data file)

**#4**

**Types In Database**

1. **Interger** Data Type [TinyInt(1), SmallInt(2), Int(4), BigInt(8)]
2. **Decimal** Data Type[Decimal(P, S), Numeric(P, S)]
3. **Money** (SmallMoney(4), Money(8))
4. **DateTime** (Date(yy/mm/dd), Time(hh/mm/ss.ms), DateTime)
5. **Character**

* **Char**(n) (1 byte) (1-8000) (non-unicode)
* **VarChar**(n/max) (1 byte) (1-8000) (non-unicode)
* **Text** (It is same as Varchar(max) data type)
* **NChar**(n) (2 byte) (1-4000) (unicode)
* **Nvarchar**(n/max) (2 byte) (1-4000) (unicode)
* **Ntext**(It is same as nvarchar(max))

1. **Binary**

* **Binary**(n) (1-8000)
* **VarBinary**(n/max) (1-8000)
* **Image**(it is same as varbinary(max))

**#5**

Structured Query Language is divided into 4 Sub Language

1. **DDL** (Data Definition Language)[Drop, SP\_Rename, Create, Alter, Truncate]
2. **DML** (Data Manipulation Language) [Select, Update, Delete, Insert]
3. **TCL** (Transaction Control Language) [Commit, RollBack, Save Transaction]
4. **DCL** (Data Control Language)[Grant, Revoke]

**#6**

**DDL (Data Definition Language)**

This Contains 5 commands

1. **Create**
2. **Alter**
3. **SP**\_**Rename**
4. **Truncate**
5. **Drop**

**#7**

**Rules For Creating Table in DB**

* Table name must be unique
* Column names should not repeat
* Table name never starts with number or special char
* No space in column or table name
* Every Object can contain only (1-128) characters
* Max no of columns can be 1024

**#8**

**DML** (Data Manipulation Language)

This contains 4 commands

1. **Insert**
2. **Update**
3. **Delete**
4. **Select**

**#9**

**Difference between Truncate and Delete**

|  |  |
| --- | --- |
| **Truncate** | **Delete** |
| DDL | DML |
| Permanent | Temporary |
| Specific is not possible | Specific possible |
| No "Where" | "Where" |
| Can't Rollback | Can Rollback |
| Reset the identity value | Don't Reset the identity value |

**#10**

Using select command we can retrieve the data from the table in 3 ways

1. Projection
2. Selection
3. Joins

**#11**

**Where, Alias, Identity, Distinct**

**#12**

**Built In Functions in SQL (System Functions)**

* **Aggregate Function**
* Configuration Function
* Curser Function
* **Date and Time Function**
* **Mathematical Function**
* Metadata Function
* Other Function
* Hierarchy Id Function
* Rowset Function
* Security Function
* String Function
* System statistical function
* Text and Image Function

**#13**

**Operators In SQL**

1. Assignment (=)
2. Arithmetic (+, -, /, %)
3. Comparison (>, <, >=, <=, !=(or <>), !<, !>)
4. Logical (And, OR, Not, Between, Not Between, Like, Not Like, IN, Not IN, Exists, Not Exists, Any, All, Some)
5. Set (Union(non duplicate), Union All(duplicate), Intersect, Except)

**#14**

**Clauses In SQL**

1. Where (Used to Filter)
2. Order By (Used to sort or arrange data)
3. TOP N Clause (fetch top N)
4. Group By (to arrange similar data into groups)
5. Having (Filtering and restring the data)

**#15**

**Difference between where and having**

|  |  |
| --- | --- |
| **Where** | **Having** |
| Used to restrict the records before grouping | Used to restrict the records after grouping |
| Can apply without group by clause | Cant apply without group by clause |

**#16**

**Syntax to create table from existing table**

* Select \* into <NewTable> from <OldTable>
* Select EmpID, Name into <NewTable> from <OldTable>
* Select \* into Dummy\_Emp from employee where 1=2 (Dummy Table without data in it)

**#17**

**Copying data from one existing table to another table**

* Insert into <DummyTable> select \* from <TableName>

**#18**

**Constraint in SQL**

**Why Constraint in SQL :**

* Constraint is using to restrict the insertion of unwanted data in any columns .
* We can create single or multiple columns of any table

Types of Constraint

1. Unique Key
2. Not Null
3. Check
4. Primary Key
5. Foreign Key

**#19**

1. **On Update cascade**,
2. **On Delete Cascade**

**#20**

**JOINS IN SQL**

Joins can be classified into the following types

1. **EQUI JOIN**
2. **INNER JOIN** (Inner join return only those records that match in both table)
3. **OUTER JOIN** (It will retrieve or get matching data from both table as well as UN matching data from left hand side table)
4. **LEFT OUTER JOIN** (It will retrieve or get matching data from both table as well as UN matching data from right hand side table)
5. **RIGHT OUTER JOIN** (It will retrieve or get matching data from both table as well as UN matching data from left hand side table plus right hand side table also.)
6. **FULL OUTER JOIN** (If we join tables with any condition other than equality condition then we call as a non Equi join.)
7. **NON EQUI JOIN** (Joining a table by itself is known as self join. When we have some relation between the columns within the same table then we use self join.)
8. **SELF JOIN**
9. **CROSS JOIN** (Cross join is used to join more than two tables without any condition we call as a cross join. In cross join each row of the first table join with each row of the second table.)
10. **NATURAL JOIN** (It is used to avoid duplicate column from the tables)



**#21**

**TCL(TRANSACTION CONTROLL LANGUAGE)**

**TRANSACTION**: A transaction is a unit of work that is performed against a database or set of statement (Insert, Update and Delete) which should be executed as one unit.

To manage transaction we have provide with transaction control language that provides a commands like

1. **BEGIN TRANSACTION** (Begin Transaction command is used to start the transaction)
2. **COMMIT** (Commit command is used to end the transaction and save the data permanent part of the database)
3. **ROLLBACK** (Rollback command is used to undo the transactions and gets back to the initial state where transaction started.)
4. **SAVE** **POINT** (Save point is used for dividing (or) breaking a transaction into multiple units)

**#22**

**Sub Query** : A select query contains another select query is called sub Query

**Syntax**: select \* from <Table Name> where (condition) (select \* from…….. (Select \* from….. (select \* from……..)));

Disadvantage of SubQuery is its Execution is slow

**#23**

**INDEXES IN SQL:**

* **Why** - One of the most important routes to high performance in a SQL Server database is the index. Indexes speed up the querying process by providing quickly access to rows in the data tables
* Index is a database object which is used for the quick retrieving of the data from the table
* An index contains keys built from one or more columns in the table and map to the storage location of the specified data.
* By using indexes we can save time and can improve the performance of database queries and applications
* When we create an indexes on any column, SQL server internally maintain a separate table called index table. So that when ever user trying to retrieve the data from existing table depends on index table SQL server directly go to the table and retrieve required data very quickly.
* In a table we can use max 250 indexes
* Two Types of Indexes

1. Clustered
2. Non-Clustered

* **Clustered** :
* A table can have only 1 clustered index on it
* The only time the data rows in a table are stored in sorted (ascending order only) order structure is when the table contains a clustered index.
* **Non-Clustered**
* In a table we can create 249 non-clustered index
* If we dont mention clustered indexes in a table then default is stored as non-clustered indexes.
* **Syntax** : Create Index <Index Name> on <Table Name>(Column Name);

**#24**

**Views In SQL :**

* View is database object which is like table but logical.
* We can call it as a logical or virtual table because it does not has a physical existence.
* It is a logical table use to get the required information from the table. View will be created by using select statement and table used for the creation of the view is called as base table.
* View will not store records in it and will not occupy memory space with help of structure existing in it and records will be displayed from table.
* **Views are used for security purpose in databases**
* views restricts the user from viewing certain column and rows means by using view we can apply the restriction
* **Why** ? : - To protect the data
* View is classified into two types

1. Simple view(Updatable view) (one table)
2. Complex view(Non-Updatable view) (more than one table)

* **Syntax** : create view <view name> as select \* from <table name>

**#25**

**SYNONYM** :

* synonym is database object which can be created as an “alias” for any object like table, view, procedure etc.
* If we apply any DML operations on synonym the same operations automatically effected to corresponding base table and vice versa
* If we create a synonym, the synonym will be created on entire table. It is not possible to create the synonym on partial table.
* **Syntax**: Create synonym <synonym name> for <object name>
* **Syntax to drop a synonym**: Drop synonym <synonym name>

**#Query**

WAQ for finding employee with Nth highest salary

select Top(1) \* from (select top(2) \* from Employee order by Employee.SALARY desc) as Emp order by Emp.SALARY

#26

**T/SQL Programming (Transact Structure Query Language)**

* SQL language will not provide reusability facilities where as T/SQL language will provide reusability facilities by defining objects such as Procedures and Functions
* T/SQL Program blocks can be divided into two types. Those are

1. Anonymous Blocks
2. Sub-Program Blocks

* **Anonymous** :- Unnamed block
* **Declare variable**
* **Syntax** : Declare @<var> [as] <datatype> [size]......
* **Assigning value to variable**
* **Syntax** : Set @<var>=<value>
* **Print value**
* **Syntax** : print @<var>
* Conditional statements :

1. IF-ELSE
2. CASE
3. While

* **Sub**-**Program**: Named block of code

1. Stored Procedure
2. Functions

* **Stored** **procedure** :
* A stored procedure is a database object which contains precompiled queries.
* Stored Procedures are a block of code designed to perform a task whenever we called.
* Advantages of SP ?
* As this is precompiled burden on DB will reduce
* Application performance will improve
* code reusability
* **Create** :

**Syntax** :

Create Procedure <SP\_Name>

(Passing parameters)

As

Begin

<stmts>

End

* **Call :**

Syntax : Exec <SP\_Name>

* **Function** :
* Function is a block of code similar to a stored procedure which is also used to perform an action and returns result as a value.
* Function can be divided into two types

1. **Scalar-Valued Function**

* In this case we can return a attribute datatype as an output from the function.
* **Create Syntax** :

Create Function <Function Name> (@parameter <Data Type> [size])

Returns <return attribute data type>

As

Begin

<Function Body>

Return <return attribute name>

End

* **Call** **Syntax**:

Select dbo.<Function Name> (value)

1. **Table**-**Valued** **Function**

* In this case we can return a table as an output from the function.
* **Create Syntax** :

Create Function <Function Name> (@parameter <Data Type> [size])

Returns <Table>

As

Return <return select statement>

* **Call** **Syntax**:

select \* from functionname(value)

* **Drop** **Syntax** : Drop Function <Function \_Name>
* **Difference between Stored Procedure and Function**

|  |  |
| --- | --- |
| **Function** | **Stored Procedure** |
| Returns value | Don't return value |
| Can't have output parameter | Can have output parameter |
| Use to perform select only | Use for all DML operations |
| Call with **Select** | Call with **Execute** |

**#27**

**Triggers**

* A trigger is a special type of procedure that will used to provide restrict on the tables when a language events executed.
* Two Types of Triggers

1. **DML**
2. DML triggers execute when the user tries to modify or change data through data manipulation language events. Those are Inserting, Update and Delete statements on the table.
3. DML triggers can be used to enforce business rules and data integrity. With the help of a DML trigger we can enforce integrity which cannot be done with constraints.
4. **Create Syntax**:

Create Trigger <Trigger Name> on <table Name>

For [Insert, Update,Delete]

AS

Begin

<Statements>

End

1. **Drop** **Syntax** :

Drop <Trigger> <Trigger Name>

1. **DDL**
2. DDL triggers fire in response to a data definition language event like create, Alter, drop etc.A DDL triggers is a special type of procedure that executes in response to a server scoped or database scoped events.
3. Create Syntax:

Create Trigger <Trigger Name> on database after <Event type>

As

Begin

<Statements>

End

1. Drop Syntax:

Drop <Trigger> <Trigger Name> on Database

**#28**

**Magic** **Tables**:

* SQL Server allows you to define a Magic Table. Magic Tables are invisible tables or virtual tables. You can see them only with the help Triggers in SQL Server.
* Magic table are those tables which allows you to hold inserted, deleted, updated values during DML operations on a Table in SQL server
* Basically there are two types of magic table in SQL Server

1. Inserted Magic Table
2. Deleted Magic Table

* **Inserted Magic Table**: Whenever you insert a record on that table, that record will be shown in the Inserted Magic table.
* **Deleted Magic Table** : : Whenever you delete a record on that table, that record will be shown in the Inserted Magic table.
* Update the record in Table : For update record we need to use both Magic tables one shows the inserted and one shows the deleted table

**#29**

**Exception** **Handling**:

* We handle errors of a program both in a programming language as well as databases also. whereas handling an error in a programming language needs stopping the abnormal termination and allowing the statements which are not related with the error to execute where as handling as error in sql server means stopping the execution of statements which are related with the error
* Handling errors in SQL Server : we can handle errors with TRY\_CATCH

Begin Try

<stmts>

End Try

Begin Catch

<stmts>

End Catch

* Error Message(): It is used to display the information about the error occurred.

**#30**

**Cursor :**

* Cursor is a memory location for storing database tables.
* cursor is a temporary work area allotted to the client at server when a SQL statement is executed.
* A cursor contains information on a select statement and the rows of data accessed by it.
* This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the Result set.
* There are Two Types of Curser

1. Implicit Cursors
2. Explicit Cursors

* **Implicit** **Cursor** : These cursors will be created by SQL server by default when select statements will show records in the table as set or result set.
* **Explicit Cursor** : When user can create a memory location to store the tables then it is called as Explicit Cursors. These cursors will access the records in the table record by record or one by one only. Whenever we want to go for record by record manipulation then explicit cursors will be used.
* **Steps to Create Cursor**

1. **Declare** : Declare <CursorName> cursor for <select statement>
2. **Opening** : Open <CursorName>
3. **Fetch Data From Cursor** : Fetch first/last/next/prior/absolute n/relative n from <CursorName> into variable
4. **Closing** : Close <Cursor>
5. **De**-**allocate** : Deallocate <CursorName>

* **Forward only & static Cursors**: If a cursor is declare as forward only it allows you to navigate only to the next records in sequential order and more over it supports only a singleton fashion method that is fetch next(one-by-one) where as a scroll cursor allows you to navigate/fetch Bidirectional that is top- bottom or bottom-top also. And it supports six different fetch methods are Fetch Next, Fetch First, Fetch Last, Fetch Prior, Fetch Absolute ,Fetch Relative.
* **Static & Dynamic Cursors**: If a cursor is declare as static after opening the cursor any modifications that are performed to the data in the table will not be reflected into cursor so the cursor contains old values only in it.

**#31**

**Data Control Language :**

* **Authentication**: Authentication is a process of verifying the credentials of a user to login into the system.
* **Authorization**: Authorization is process of verifying whether the user as permissions to perform any operation on the database.
* **Data Control Language**: DCL commands are used to enforce database security in multiple users’ database environment. These are two types.

1. GRANT
2. REVOKE

* ***GRANT***: Grant command is used for giving a privilege or permission for a user to perform operations on the database.
* **Syntax**:

GRANT <Privilege Name> on <object name>

To {User} [With GRANT OPTION]

* **Privilege Name**: Used to granted permission to the users for some rights are ALL, EXECUTE and SELECT.
* **Object** **Name**: It is the name of database objects like Table, Views and Stored Procedure etc.
* **User**: Used for to whom an access rights is being granted.
* **With** **Grant** **Option**: Allows a user to grant access rights to other users.
* ***REVOKE***: Revoke command removes user access rights / privileges to the database OR taking back the permission that is given to a user.
* **Syntax**: Revoke <privilege name> on <object name > from {user}

**#32**

**Normalization :**

* Normalization is the process of efficiently organizing data in a database. There are two goals of the normalization process are,

1. Eliminating redundant data (for example, storing the same data in more than one table) and
2. Ensuring data dependencies make sense (only storing related data in a table).

* Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored.
* There are several benefits for using Normalization in Database.

1. Eliminate data redundancy
2. Improve performance
3. Query optimization
4. Faster update due to less number of columns in one table
5. Index improvement

* There are different types of Normalizations form available in the Database.

1. 1NF (First Normal Form)
2. 2NF
3. 3NF
4. 4NF

-- Data : facts and statistics collected together for reference or analysis.

-- Information : Processed Data is known as Information.

-- Database : Collection of information belongs to a particular topic written in a predetermined manner stored at a particular place is called as database.

-- DBMS : It is software which is present inside the database, which maintains Database

-- Types of Database

-- 1. FMS (File Managment System)

-- 2. HMS (Hierarchy Managment System)

-- 3. NDBMS (Network Database Management System)

-- 4. RDMS (Relational Database Management System)

-- CODD RULES

-- SQL SERVER: SQL Server is an RDBMS product which was designed and developed by Microsoft Company.

-- Server Authentication :

-- 1. Windows Authentication

-- 2. SQL Server Authentication

-- System Database :

-- Master

-- Model

-- Msdb

-- Tempdb

-- Data Types in SQL Server

-- 1. Integer Data Types

-- 2. Decimal Data Types

-- 3. Money (or) Currency Data Types

-- 4. Date & Time Data Types

-- 5. Character Data Types

-- 6. Binary Data Types

-- 1. Integer Data Types

-- TinyInt (0-255) 1 byte

-- SmallInt (-32768 to 32767) 2 byte

-- Int (-2,147,483,648 to 2,147,483,647) 4 byte

-- Bigint (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807) 8 byte

-- 2. Decimal Data Types

-- Decimal (P,S) -----> P=Precision & S=Scale

-- Numeric (P,S)

-- 3. Money Data Type

-- Small Money (4 byte)

-- Money (8 byte)

-- 4. Date and Time Data Type

-- Date & Time ---> "yy/mm/dd" "hh/mm/ss.ms"

-- 5. Character Data Types

-- Char(n)

-- varchar(n/max)

-- Text [varchar(max)]

-- nchar(n)

-- nvarchar(n/max)

-- ntext [nvarchar(max)]

-- 6. Binary Data Type

-- Binary(n)

-- varbinary(n/max)

-- Image [varbinary(max)]

-- Structured query language

-- |\_\_\_\_\_\_\_ 1. DDL (Data Defination Language) [Drop, SP\_RENAME, Create, Alter, Truncate]

-- |\_\_\_\_\_\_\_ 2. DML (Data Manipulaion Language) [Select, Insert, Update, Delete]

-- |\_\_\_\_\_\_\_ 3. TCL (Transaction Control Language) [Commit, Rollback, Save Transaction]

-- |\_\_\_\_\_\_\_ 4. DCL (Data Control Language) [Grant, Invoke]

-- Every object name should contain minimum one character and maximum 128 characters

-- The maximum no. of columns a table can have 1024 columns.

sp\_databases -- list databases

SP\_Tables -- list tables

Drop database NathanArk --Remove database

Create Database NathanArk -- Create database

use NathanArk --set default database

--Create Table

Create table Department

(

DeptID int primary key Identity(1, 1), --set identity and primary key at creation

DepartmentName nvarchar(150)

)

-- Check how to add primary key Forign key, Identity to created tables?

Create Table Employee

(

EmpID int primary key Identity(1, 1),

FirstName nvarchar(150),

LastName nvarchar(150),

Age tinyint,

Location nvarchar(150),

DeptID int foreign Key references Department(DeptID) -- Set forign key

)

insert into Department (DepartmentName) values ('IT'), ('Sales'), ('Marketing'), ('Clark'), ('Manager')

select \* from Department

Insert into Employee

(FirstName, LastName, Age, Location, DeptID)

values

('Navjyot', 'Gurhale', 24, 'Hyderabad', 1),

('Netaji', 'Jadhav', 29, 'Chenai', 2),

('Shekhar', 'Kashid', 27, 'Bangalore', 3),

('Nitin', 'Kulkarni', 26, 'Mumbai', 2),

('Ankit', 'Rai', 30, 'Pune', 1)

Select \* from Employee

--Alter

alter table Department alter column DepartmentName nvarchar(200) --update column

alter table Department add NewColumn nvarchar(100) --Add Column

alter table Department drop column NewColumn -- drop column

-- SP\_Rename

alter table Department add NewColumn nvarchar(100) --Add Column

SP\_Rename 'Department.NewColumn', 'LocationColumn', 'Column'

alter table Department drop column LocationColumn

SP\_Rename 'Department', 'Dept' -- Rename table Name

SP\_Rename 'Dept', 'Department' -- Rename table Name

-- Truncate

Truncate table Department

--Drop

Drop table Department

--SP\_help

sp\_help Department

-- Difrence between Delete and truncate

-- Projection (Retrieving the data from specific columns is known as Projection)

-- Selection (Retrieving the data based on some condition is called selection)

-- ALIAS ALIAS is a duplicate name (or) alternate name for the original column name (or) Table name (or) an expression name.

-- IDENTITY

-- It is use to generate unique values in sequential order without user interaction.

-- The default value of identity is Identity (1, 1).

-- Built In Functions

-- ABS ()

select ABS(-15)

-- CEILING ()

select CEILING(15.003)

select CEILING(-3.14)

-- FLOOR ():

select FLOOR(15.003)

select FLOOR(-3.14)

-- SQUARE ()

select SQUARE(2)

--SQRT()

select SQRT(16)

select SQRT(17)

--Power()

select Power(2, 3)

-- OPERATORS IN SQL

-- 1. Assignment operator

-- 2. Arithmetic operator

-- 3. Comparison operator

-- 4. Logical operator

-- 5. Set operator

-- Set Operators

-- UNION / UNION ALL / INTERSECT /EXCEPT

Create Table Employee\_Chennai

(

EmpID int,

Name nvarchar(100)

)

Create Table Employee\_Pune

(

EmpID int,

Name nvarchar(100),

Age tinyint

)

Insert into Employee\_Chennai (Name) values ('Navjyot Gurhale'), ('Arun Agrwal'), ('Will Smith')

Insert into Employee\_Pune (Name, Age) values ('Navjyot Gurhale', 25), ('Netaji Jadhav', 29), ('Girish Kangne', 45)

select EmpID, Name from Employee\_Chennai

union

Select EmpID, Name from Employee\_Pune

select EmpID, Name from Employee\_Chennai

union all

Select EmpID, Name from Employee\_Pune

select EmpID, Name from Employee\_Chennai

intersect

Select EmpID, Name from Employee\_Pune

select Name from Employee\_Chennai

Except

Select Name from Employee\_Pune

except

select Name from Employee\_Chennai

-- CLAUSES IN SQL

-- Where

Select \* from Employee where DeptID=1

-- Order By

Select \* from Employee order by EmpID

Select \* from Employee order by EmpID desc

-- Top N clause

Select top(2) \* from Employee

-- Group By

SELECT DeptID, COUNT=COUNT (\*) FROM Employee GROUP BY DeptID

-- Having

SELECT DeptID, COUNT=COUNT (\*) FROM Employee GROUP BY DeptID having count (\*)>=2

-- Differences Between WHERE and HAVING Clause

-- Types of constraints in SQL Server:-

-- 1. Unique Key constraint.

Create Table Person

(

UserID nvarchar(150) unique

)

-- 2. Not Null constraint.

Create Table Student

(

StudID int not null

)

-- 3. Check constraint

create table emp4

(

eno int,ename varchar(50),

age int check (age between 18 and 40)

)

-- 4. Primary key constraint.

create table emp

(

EID int primary key,

ENAME varchar(50),SALARY money

)

-- For exissting table

alter table emp add constraint pk\_Eid primary key (EID)

alter table emp add primary key (EID)

-- Composite Key

create table Emp01

(

Eid int,

Ename nvarchar(100),

primary key(Eid, Ename)

)

alter table Emp01 add constraint Composite\_Key primary key (Eid, ename)

alter table Emp01 drop constraint Composite\_Key

alter table Emp01 add primary key (Eid, Ename)

-- 5. Foreign Key constraint.

create table Employee

(

EID int,ENAME varchar(50),

SALARY money,

Deptno int foreign key references Department(Deptno)

)

-- Existing Table

alter table Employee add foreign key (DeptID) references Department(DeptId)

alter table Employee drop foreign key FK\_DeptID --or drop constraint FK\_DeptId

-- JOINS IN SQL

-- EQUI JOIN

Select \* from Employee, Department where (Employee.DeptID=Department.DeptID)

-- INNER JOIN

Select \* from Employee E inner join Department D on E.DeptID=D.DeptID

Select Name=E.FirstName + ' ' + E.LastName, D.DepartmentName from Employee E inner join Department D on E.DeptID=D.DeptID

-- OUTER JOIN

-- LEFT OUTER JOIN

select \* from Employee Left outer join Department on Employee.DeptID=Department.DeptID

-- RIGHT OUTER JOIN

select \* from Employee right outer join Department on Employee.DeptID=Department.DeptID

-- FULL OUTER JOIN

select \* from Employee full outer join Department on Employee.DeptID=Department.DeptID

-- NON EQUI JOIN

--SELECT \* FROM EMP, SALGRADE WHERE (SALARY > LOWSAL) AND (SALARY < HIGHSAL)

-- SELF JOIN

SELECT E.EmpID, E.FirstName, E.LastName, M.EmpID, M.FirstName, M.LastName FROM Employee E, Employee M WHERE E.EmpID=M.EmpID

-- CROSS JOIN

SELECT \* FROM Employee CROSS JOIN Department

-- NATURAL JOIN

SELECT EmpID, FirstName, LastName, DepartmentName FROM Employee E, Department D WHERE E.DeptID=D.DeptID

-- Create table from existing Table

--Select \* into <New Table Name> from <Old Table Name>

-- TRANSACTION CONTROLL LANGUAGE

-- TRANSACTION

-- BEGIN TRANSACTION

-- COMMIT

-- ROLLBACK

-- SAVE POINT

BEGIN TRANSACTION

INSERT INTO EMPLOYEE VALUES('Kamal', 'Hasan', 26 ,'MUMBAI', 4)

INSERT INTO EMPLOYEE VALUES('Sujata', 'Singh', 31, 'DELHI', 5)

COMMIT

BEGIN TRANSACTION

DELETE FROM EMPLOYEE WHERE EmpID=1

DELETE FROM EMPLOYEE WHERE EmpID=2

BEGIN TRANSACTION

ROLLBACK

BEGIN TRANSACTION

UPDATE EMPLOYEE SET SALARY=99000 WHERE EID=101

UPDATE EMPLOYEE SET SALARY=88000 WHERE EID=102

SAVE TRANSACTION S1

UPDATE EMPLOYEE SET SALARY=77000 WHERE EID=103

UPDATE EMPLOYEE SET SALARY=66000 WHERE EID=104

SAVE TRANSACTION S2

UPDATE EMPLOYEE SET SALARY=55000 WHERE EID=105

UPDATE EMPLOYEE SET SALARY=44000 WHERE EID=106

BEGIN TRANSACTION

ROLLBACK TRANSACTION S1

BEGIN TRANSACTION

ROLLBACK TRANSACTION S2

select \* from Employee

-- Sub Query

-- Syntax: select \* from <Table Name> where (condition) (select \* from…….. (Select \* from….. (select \* from……..)));

--INDEXES IN SQL

-- Index is a database object which is used for the quick retrieving of the data from the table.

-- types of indexes

-- 1. Clustered (1)

-- 2. Non-Clustered (249)

-- Syntax: Create Index <Index Name> on <Table Name>(Column Name);

create Index NoTest1 on Employee(EmpID, FirstName)

-- VIEWS IN SQL: View is database object which is like table but logical.

-- Why We Need Views: To protect the data

-- View is classified into two types.

-- Simple view(Updatable view) [we create a view based on one table is called simple view or Updatable view]

-- Complex view(Non-Updatable view) [we create a view based on more than one table is called complex view or Non-Updatable view]

-- Syntax: create view <view name> as select \* from <table name>

Create view AllEmpDetails

as

select Emp.EmpID, Name=Emp.FirstName +' '+ Emp.LastName, Department=Dept.DepartmentName from Employee as Emp, Department as Dept where Emp.DeptID=Dept.DeptID

select \* from AllEmpDetails

--• synonym is database object which can be created as an “alias” for any object like table, view, procedure etc.

create synonym MySynonym for AllEmpDetails

select \* from MySynonym

-- \*\*\*\*\*\*\*\*T/SQL Programming\*\*\*\*\*\*\*\*\*\* (Transact Structure Query Language)

declare @EmpId int; declare @Name nvarchar(150);

set @EmpId=101; Set @Name='Navjyot Gurhale'

Print @EmpId; Print @Name;

--IF\_Else

Declare @num1 int;

Declare @num2 int;

set @num1=12;

set @num2=22;

IF(@num1>@num2)

Print '1st num is big'

else if(@num1<@num2)

Print '2nd no is big'

else

Print 'it seems both are equal'

-- Switch

DECLARE @WEEK INT

SET @WEEK=DATEPART(DW, GETDATE())

SELECT CASE @WEEK

WHEN 1 THEN 'SUNDAY'

WHEN 2 THEN 'MONDAY'

WHEN 3 THEN 'TUESDAY'

WHEN 4 THEN 'WEDNESDAY'

WHEN 5 THEN 'THURSDAY'

WHEN 6 THEN 'FRIDAY'

WHEN 7 THEN 'SATURDAY'

END

-- While

DECLARE @X INT

SET @X=0

WHILE @X<10

BEGIN

SET @X=@X+1

PRINT @X

END

-- 1. Stored Procedures/Procedure

-- 2. Stored Functions/Functions

-- 1. Stored Procedures/Procedure

-- Syntax 1:

-- Create Procedures <Procedures Name>

-- As

-- Begin

-- <Statements>

-- End

-- Syntax 2:

-- Create Procedures <Procedures Name>

-- (Passing parameters)

-- As

-- Begin

-- <Statements>

-- End

-- Syntax: Exec <Procedure name>

-- Drop Procedure <Procedure Name>

-- 2. Stored Functions/Functions

-- 1. Scalar valued Functions

-- Syntax:

-- Create Function <Function Name> (@parameter <Data Type> [size])

-- Returns <return attribute data type>

-- As

-- Begin

-- <Function Body>

-- Return <return attribute name>

-- End

-- Select dbo.<Function Name> (value)

-- 2. Table-Valued Fuction

-- Syntax:

-- Create Function <Function Name> (@parameter <Data Type> [size])

-- Returns <Table>

-- As

-- Return <return select statement>

-- select \* from ft1('hyd')

-- Drop Function <Function Name>

-- Difference between Function And Procedure

-- A function must return a value where as procedure never returns a value. A procedure can have parameters of both input (with parameters) and output (without parameters) where as a function can have only input (with parameters) parameters only.

-- In procedure we can perform select, insert, update and delete operation where as function can used only to perform select. Cannot be used to perform insert, update and delete operations.

-- A procedure provides the option for to perform transaction management where as these operations are not permitted in a function.

-- We call a procedure using execute command where as function are called by using select command only.

--Triggers

--WAQ of trigger to Capitalise FirstName

create trigger CapitaliseWords on Employee

for Insert

as

Begin

declare @EmpID int, @FirstName nvarchar(300)

select @EmpID=EmpID, @FirstName=FirstName from inserted

update Employee set FirstName=upper(@FirstName) where EmpID=@EmpID

end

INSERT INTO EMPLOYEE VALUES('Shruti', 'Hasan', 26 ,'Hyderabad', 4)

select \* from EMPLOYEE

-- WAQ of trigger to resctrict DML on Table

create trigger NoDMLOnEmployee on Employee

for insert, Delete, Update

as

begin

print 'No DML Allowded by Trigger NoDMLOnEmployee'

rollback transaction

end

--A program which will restrict the Delete operation if the Job Location is Pune

create trigger NoDeleteForPune on Employee

for Delete

As

Begin

Declare @Location nvarchar(200)

select @Location=Location from Deleted

IF @Location='Pune'

begin

rollback

end

End

Delete from Employee where EmpID=5

--Magic Table

-- Insert magic table

create trigger InsertMagicTable on Employee

for insert

as

begin

select \* from inserted

end

INSERT INTO EMPLOYEE VALUES('Shruti', 'Hasan', 26 ,'Hyderabad', 4)

select \* from EMPLOYEE

--Deleted magic table

create trigger DeletedMagicTable on Employee

for delete

as

begin

select \* from deleted

end

delete from Employee where EmpID=8

-- Update

create trigger updateMagicTable on Employee

for update

as

begin

select \* from deleted

select \* from inserted

end

Update Employee set LastName='Sharma' where EmpID=6

select \* from Employee

--CURSOR

--Create an explicit cursor to display all the records from the table.

declare @EmpID int, @FirstName nvarchar(200)

declare c1 cursor for select EmpID, FirstName from Employee

open c1

fetch next from c1 into @EmpID, @FirstName

while @@FETCH\_STATUS=0

begin

print @EmpID

print @FirstName

fetch next from c1 into @EmpID, @FirstName

end

close c1

deallocate c1

--DATA CONTROL LANGUAGE

--Normalization