SQL Server

Create a Database:

We can create database in two ways, using Graphical and query. Whenever we create database, the system will create two files on the server. One is **.MDF** data file (contains actual data) and another one is **.LDF** Transaction file (Used to recover the database).

```
CREATE DATABASE LEARNSQL
```

Alter a Database :

We can alter database name in two ways, One is usin **aleter** key word and another way is using system stored procedure.

```
1. ALTER DATABASE LEARNSQL MODIFY NAME = LEARNSQL1

2. EXECUTE SP_RENAMEDB 'LEARNSQL1','LEARNSQL'
```

Drop a Database:

```
DROP DATABASE LEARNSQL
```

When we delete database, the system will delete the associated **.MDF** & **.LDF** files from the server. The database cannot be deleted when other users are using it. Which means we can delete unused database.

So if other users are connected, if it is currently in use. We get an error "Cannot drop database because it is currently in use". If we really want to delete database then change mode "MULTI_USER" to "SINGLE_USER".

```
ALTER DATABASE LEARNSQL SET SINGLE_USER WITH ROLLBACK IMMEDIATE
```

With "Rollback Immediate" option, will rollback all incomplete transaction and closes the connection to the database.

Note: System database cannot be dropped.

Create a Table:

Here also we can create table in two ways, one is graphical and another way is using query.

```
CREATE TABLE tblGender
(
      ID int NOT NULL Primary Key,
      Gender nvarchar(50) Not NULL
)
```

Primary Key Constraint: It is an identity column, basically used for identify a record in a table. A table can have only one primary key. If we want to enforce uniqueness on 2 or more columns, then have UNIQUE KEY constraint.

Foreign Key Constraint: It will establish a relationship between two tables. If we set a column as foreign key it would not allow to enter invalid value in that column but allows NULL, which means allows only values that are present in main table (tblGender – ID). Database integrity is possible with foreign key.

```
ALTER table tblPerson add constraint tblPerson_GenderId_FK
Foreign KEY(GenderId) references tblGender(ID)
```

Default Constraint:

Default constraint will set value if we miss the value in the query otherwise it will take the passing value.

```
ALTER TABLE TBLPERSON
ADD CONSTRAINT DF_TBLPERSON_GENDERID
DEFAULT 3 FOR GENDERID
```

Ex: `INSERT INTO tblPerson (ID, Name, Email, GenderId)`
`VALUES(5,'STEVE','M@M.COM',NULL)` ➔GenderID = NULL

Ex: `INSERT INTO tblPerson (ID, Name, Email)`
`VALUES(5,'STEVE','M@M.COM')` ➔GenderID = 3

Ex: `INSERT INTO tblPerson (ID, Name, Email, GenderId)`
`VALUES(5,'STEVE','M@M.COM',1)` ➔GenderID = 1

Adding a new column, with default value, to existing table:

```
ALTER TABLE TBLPERSON
ADD AGE INT NULL
CONSTRAINT DF_TBLPERSON_AGE DEFAULT 0
```

Drop constraint from a table column:

```
ALTER TABLE TBLPERSON
DROP CONSTRAINT DF_TBLPERSON_AGE
```

`Drop a column:`

```
ALTER TABLE TBLPERSON
DROP COLUMN AGE
```

In this case if we want to drop a column, so first we have to drop default constraint and then drop a column. Otherwise it will throw exception.

Cascading referential integrity constraint: It allows to define actions MS Sql Server should take when a user to delete or update key to which an existing foreign keys points (tblGender table).

```
DELETE from tblGender where ID = 2
```

1. No Action: if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, an error is raised and the DELETE or UPDATE is rolled back.

2. Set Default: specifies that id an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are set to default values.

   Ex: if we delete row 2 in tblGender table, all the rows will get update with default value GenderId = 3 in tblPerson table.

3. Set Null: The dependency rows GenderId will be updated with NULL.

4. Cascade: if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, all rows containing those foreign keys are also deleted or updated.

   Ex: if we delete a row from tblGender then the referential rows from the other table (tblPerson) also deleted.

Check Constraint: It is used to limit the range of the values that can be entered for a column. It prevent to being entered invalid value in that column. We can add graphically and also using query.

```
ALTER TABLE {Table_Name}
ADD CONSTRAINT {Constraint_Name} CHECK (Boolean_Expression)
```

Ex:

```
ALTER TABLE TBLPERSON
ADD CONSTRAINT CH_TBLPERSON_AGE CHECK (AGE > 0 AND AGE < 200)
```

If the Boolean expression returns true, then the CHECK constraint allows the value. Otherwise it doesn't. If the column is nullable, when we pass null for the column the Boolean expression returns UNKNOWN and allows null value.

Identity column:

What is Identity column: if a column is marked as an identity column, then the values are automatically generated, when we insert a new row into the table. There are two properties to define identity.

```
CREATE TABLE TBLGROUP
(
      GROUPID INT IDENTITY(1,1) not null primary key,
      Name nvarchar(50)
)
```

1. Seed – Sets starting value.

2. Increment – Sets amount of increment value.

Difference between Primary Key (without identity) & Identity Column:

1. Both the column won't allow duplicate values.

2. We have to pass value for PK column but not for identity.

3. By default we cannot reuse the old *ID* value with identity but we can use with primary key.

How to reuse old ID with identity: An explicit value for the identity column in table, can only be specified when a column list is used and IDENTITY_INSERT is ON.

To turn on IDENTIY_INSERT on: `SET IDENTITY_INSERT TBLGROUP ON`

How to reset identity value: we can reset by using DBCC CHECKIDENT function. But if we set seed to 0 and already if any record exists with some id then it will throw exception when we insert same value. Better we should delete all the records before set seed to 0.

```
DBCC CHECKIDENT(TBLGROUP, RESEED,0)
```

Retrieve Last Generated Identity column value: We can retrieve last identity column value in three ways.

1. SCOPE_IDENTITY() – Get the last identity value for the same session & same scope. It's most common way to get identity value.

2. @@IDENTITY – Get the last identity of same session & across the scope.

3. IDENT_CURRENT('TableName') – Specific table across any session and any scope.

Unique Key Constraint: We use this constraint to enforce uniqueness of a column. Create a unique key constraint.

```
ALTER TABLE TABLE_NAME
ADD CONSTRAINT CONSTRAINT_NAME UNIQUE (COLUMN_NAME)
```

EX:
```
ALTER TABLE TBLPERSON
ADD CONSTRAINT UQ_TBLPERSON_EMAIL UNIQUE (EMAIL)
```
Primary key doesn't allow null value whereas unique key will allow only one null value.

Difference B/W WHERE and HAVING clause:

1. Where clause can be used with – Select, Insert, Update and Delete statements, where as having clause can only be used with the Select statement.

2. Where filters rows before aggregation (Grouping), whereas, Having filters groups, after the aggregations are performed.

3. Aggregate functions cannot be used with WHERE clause, unless it is in a sub query contained in a HAVING clause, whereas, aggregate functions can be used in having clause.

Replace a NULL value:

1. Using CASE

2. Using ISNULL() function

3. Using COALESCE() function

COALESCE(): Returns the first not null value.

```
declare @fName varchar(10)= NULL, @mName varchar(10)= NULL,
@lName varchar(10)= NULL
select COALESCE(@fName, @mName, @lName, 'No Name') ➔ 'No Name'

declare @fName varchar(10)= NULL, @mName varchar(10)= NULL,
@lName varchar(10)= 'last Name'
select COALESCE(@fName, @mName, @lName, 'No Name') ➔ 'last Name'
```

UNION & UNION ALL & JOIN:

Difference b/w UNION & UNION ALL:

1. UNION removes duplicate rows, whereas UNION ALL does not.

2. UNION has to perform distinct sort to remove duplicates, which makes it less faster than UNION ALL

Note: Sorting result of a UINON or UNION ALL: Order by clause should be used only on the last SELECT statement in the UNION query.

Difference B/W UINON and JOIN: UNION combines the result set of 2 or more select queries into single result-set which includes all the rows from all the queries, whereas JOINS, retrieve data from 2 or more tables based on logical relationship b/w the tables.

In short, UNION combines rows from 2 or more tables, whereas JOINS combines columns from 2 or more tables.

Stored Procedure

A stored procedure is a group of T-SQL (Transact SQL) statements, if we have a situation, where we write the same query over and over again, we can save that specific query as a stored procedure and call it just by its name.

Advantages:

1. Execute plan retention and reusability
    a. Execution plan – Better way to retrieve data. Stored procedure will reuse execution plan. Whereas inline query will also reuse but if there any slight change in the query then it will recreate the execution plan.
2. Reduces network traffic
3. Code reusability and better maintainability
4. Better security
5. Avoid SQL injections