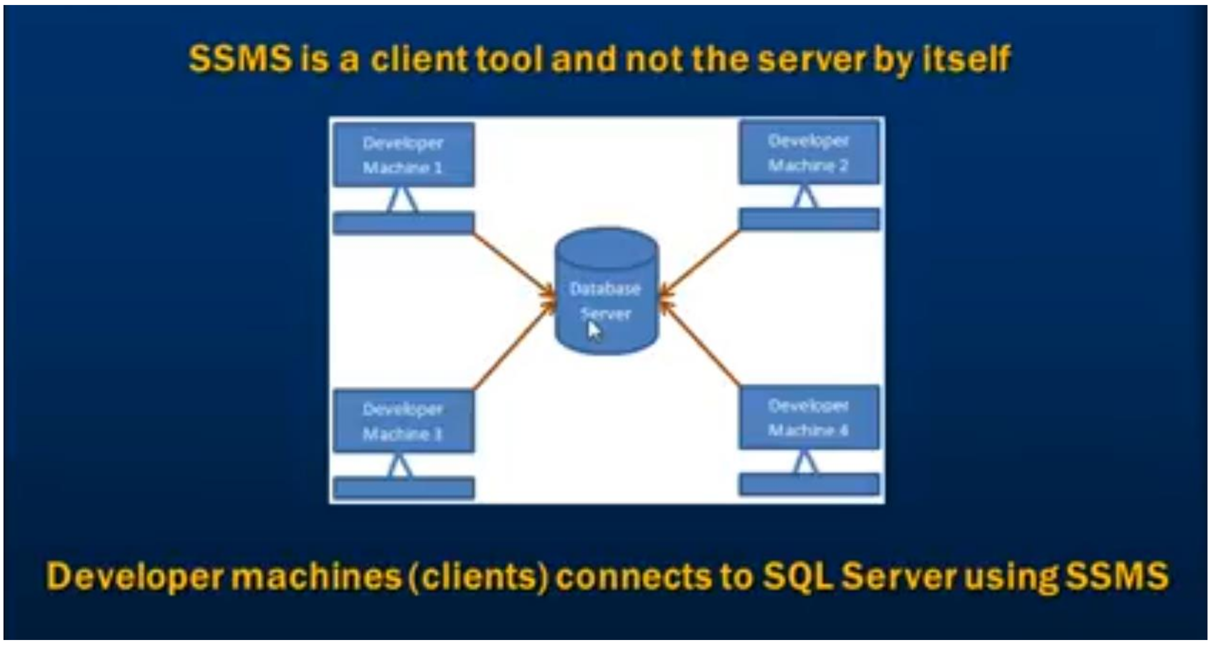
SSMS


SSMS is a client tool and not the server by itself

Developer machines (clients) connects to SQL Server using SSMS

Creating and Altering a Database

A SQL Server database can be created, altered and dropped

1. Graphically using SQL Server Management Studio (SSMS) or
2. Using a Query

To Create the database using a query

```
Create database Sample1
```

While, a database is created graphically using the designer or, using a query, the following 2 files gets generated

- .MDF file-Data File (Contains actual data)
- .LDF file-Transaction Log file (Used to recover the database)

To alter a database, once it's created

```
Alter database Sample2 Modify Name=Sample1
```

Alternatively, you can also use system stored procedure

```
Execute sp_renameDB 'Sample1', 'Sample2'
```

Deleting or Dropping a database

To Delete or Drop a database

```
Drop Database Sample1
```

Dropping a database, deletes the LDF and MDF files.

If a database is currently in use, it cannot be deleted or altered. There will be an error stating- **Cannot drop database "<DatabaseName>" because it is currently in use**.

So, if other users are connected, need to put the database in single user mode and then drop the database.

```
Alter Database Sample1 Set SINGLE_USER with Rollback Immediate
```

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

To revoke `ALTER DATABASE Sample1 SET MULTI_USER WITH ROLLBACK IMMEDIATE;`

Note: System databases cannot be dropped.

Creating Tables

```
Use sample1
// Where Sample1 is the DBName, this is a good practice to use the DB name,
else it may be created in another database.

create table tblGender
(
Id int not null primary key,
Gender varchar(6) not null
)

Use sample1
create table tblPerson
(
Id int not null primary key,
Name varchar(50),
Email varchar(100),
GenderId int not null
)
```

Constraints

Foreign Key

```
Alter table <TableName> add constraint <FK_Name>
Foreign Key <ColumnName> references <ReferencingTable(<RefTableColumnName>)>

Alter table tblPerson add constraint FK_tblPerson_genderId
Foreign Key (GenderId) references tblGender(Id)
```

Foreign keys are used to enforce database integrity.

In layman's terms, A foreign key in one table points to a primary key in another table. The foreign key constraint prevents invalid data form being inserted into the foreign key column. The values that entered into the foreign key column, has to be one of the values contained in the reverencing table it points to.

Primary Key

A table should have only one Primary Key

Default Constraint

A column default can be specified using Default constraint. The DEFAULT constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified, including NULL.

Altering an existing column to add a default constraint:

```
ALTER TABLE <TableName>
ADD CONSTRAINT <ConstraintName>
DEFAULT <Value> FOR <ColumnName>;
```

```
ALTER TABLE tblPerson
ADD CONSTRAINT Default_Constraint
DEFAULT 3 FOR GenderId;
```

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

```
ALTER TABLE <TableName>
ADD <New ColumnName> <Datatype>
CONSTRAINT <ConstraintName> DEFAULT <Value>
```

```
ALTER TABLE tblPerson
ADD Phone_Number int not null
CONSTRAINT DeFault_phone DEFAULT 0
```

Here `DEFAULT` is optional.

Dropping a constraint:

```
ALTER TABLE <TableName>
drop CONSTRAINT <ConstraintName>
```

```
ALTER TABLE tblPerson
drop CONSTRAINT DeFault_phone
```

Cascading Referential Integrity Constraints

Cascading referential integrity constraint allows to define the actions Microsoft SQL Server should take when a user attempts to delete or update a key to which an existing foreign keys point.

For example, if delete row with ID-1 from tblGender table, then row with ID-3 from tblPerson table becomes an orphan record. So, cascading referential integrity constraint can be used to define actions Microsoft SQL Server should take when this happens. By default, there will be an error and the DELETE or UPDATE statement is rolled back.

```
delete from tblGender  where Id=1
```

```
Msg 547, Level 16, State 0, Line 1
The DELETE statement conflicted with the REFERENCE constraint "FK_tblPerson_genderId".
The conflict occurred in database "Sample1", table "dbo.tblPerson", column 'GenderId'.
The statement has been terminated.
```

Options when setting up Cascading referential integrity constraint:

1. **No Action**: This is the default behaviour. No Action specifies that 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. **Cascade**: Specifies that 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.**

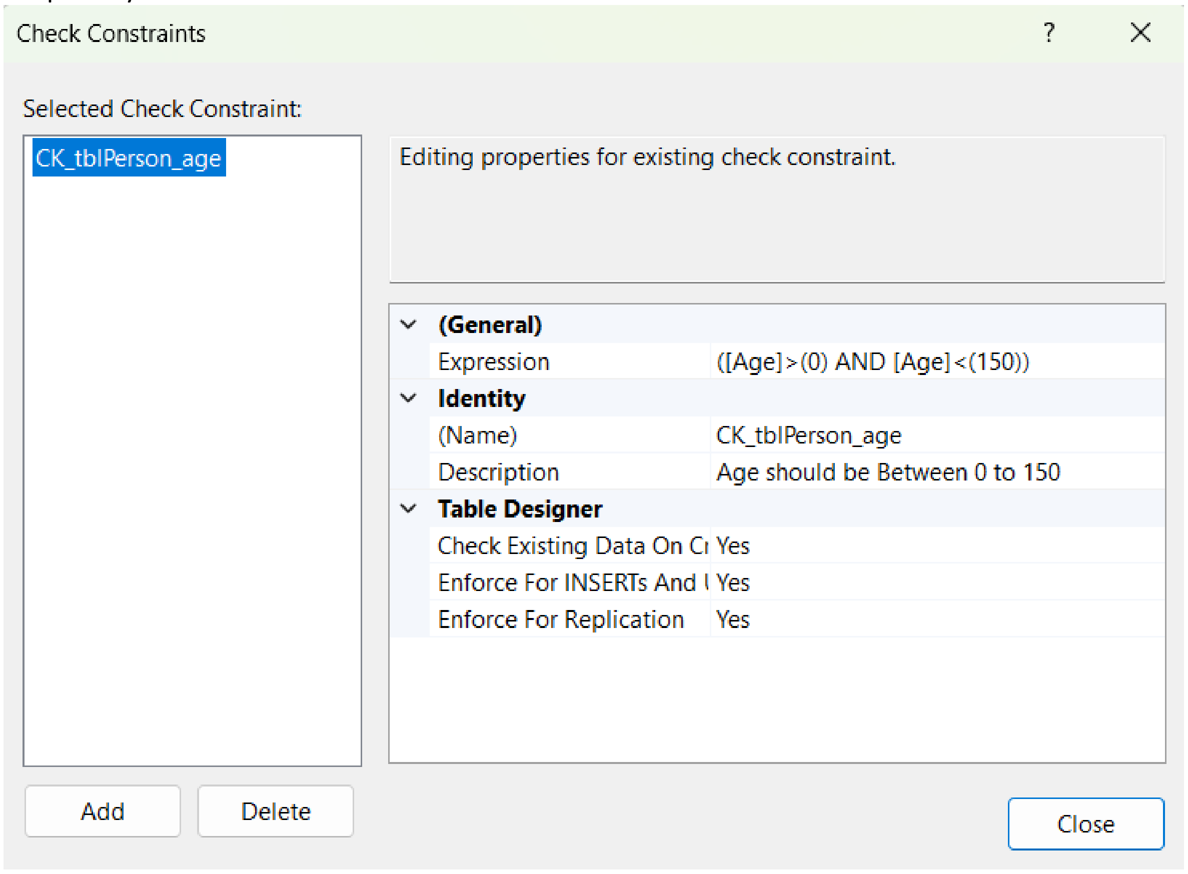
3. **Set NULL**: Specifies that if an attempt is made to delete or update a row with a key referenced by foreign keys in existing rows in other tables, **all rows containing those foreign keys are set to NULL**

4. **Set Default**: Specifies that 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 set to default values.**

Check Constraint

This is used to limit the range of values , that can be generated for a column.

Graphically



CK_tblPerson_age

Editing properties for existing check constraint.

(General)	
Expression	([Age]>(0) AND [Age]<(150))
Identity	
(Name)	CK_tblPerson_age
Description	Age should be Between 0 to 150
Table Designer	
Check Existing Data On Cı	Yes
Enforce For INSERTs And l	Yes
Enforce For Replication	Yes

The general formula for adding check constraint in SQL Server:

```
ALTER TABLE <TableName>
ADD CONSTRAINT <ConstraintName> CHECK ([<ColumnToCheck]>(Value) AND [ColumnToCheck]<(
Value))
```

```
ALTER TABLE tblPerson
ADD CONSTRAINT CK_tblPerson_age CHECK ([Age]>(0) AND [Age]<(150))
```

If the BOOLEAN_EXPRESSION returns true, then the CHECK constraint allows the value, otherwise it doesn't. Since, AGE is a nullable column, it's possible to pass null for this column, when inserting a row. When a NULL is passed for the AGE column, the Boolean expression evaluates to UNKNOWN, and allows the value.

To drop the CHECK constraint:

```
ALTER TABLE tblPerson
DROP CONSTRAINT CK_tblPerson_Age
```

Identity Column

If a column is marked as an identity column, then the values for this column are automatically generated, when you insert a new row into the table.

```
CREATE TABLE tblCity(
[PersonId] [int] IDENTITY(1,1) PRIMARY KEY NOT NULL,
[City] [varchar](50) NULL)
```

Note: Seed and increment values are optional. If you don't specify the identity and seed, they both default to 1

For inserting Column names and the value for Identity column are optional

```
insert into tblCity1 values ('Kolkata')
```

If a row is deleted, explicitly could not reuse that identity value.

To explicitly supply a value for identity column

1. First turn on identity insert-SET Identity_Insert tblPerson ON
2. In the insert query specify the column list

```
set Identity_insert tblCity1 on
insert into tblCity1 (PersonId,City) values (1,'Howrah')
```

If the Identity_insert is on, Column names are mandatory in Insert script. Or make

```
set Identity_insert tblCity1 off
```

If you have deleted all the rows in a table, and you want to reset the identity column value, use DBCC CHECKIDENT command

```
DBCC CHECKIDENT('tblCity1',RESEED ,0)
```

Retrieving Identity Column values

There are several ways in sql server, to retrieve the last identity value that is generated. The most common way is to use SCOPE_IDENTITY() built in function.
Note: You can also use @@IDENTITY and IDENT_CURRENT('TableName')

Difference:

`SCOPE IDENTITY()-` Same session and the same scope.
```
select SCOPE_IDENTITY()
```
Fetched the last generated Identity value of the current scope

`@@IDENTITY` - Same session and across any scope.
```
Select @@IDENTITY
```
IDENT_CURRENT('TableName')-Specific table across any session and any scope.
```
select IDENT_CURRENT('test2')
```

This will fetch the last Identity number of the table, across any session or any scope.

Full Code:
```
insert into test1 values ('ABCD')

select SCOPE_IDENTITY()
Select @@IDENTITY

select * from test1
select * from test2

create  trigger trg_test1 on test1 for insert
as
begin
insert into test2 values ('XYZW')
end
```

In the above code the current session and scope is `test1` table. So, `SCOPE_IDENTITY()` will fetch the last identity value of the table. But after insertion in `test1` a trigger will be fired and a new row will be inserted in `test2`. So, in this session the scope is shifted to `test2`. Hence, `@@IDENTITY` will return the last identity number of `test2`

Unique Key

We use UNIQUE constraint to enforce uniqueness of a column i.e. the column shouldn't allow any duplicate values. We can add a Unique constraint thru the designer or using a query.

To create the unique key using a query:

```
alter table tblPerson
add constraint uq_tblPerson_AdhaarNum unique (AdhaarNum)
```

But primary key and unique key are used to enforce, the uniqueness of a column. So, when do you choose one over the other?
A table can have, only one primary key. If you want to enforce uniqueness on 2 or more columns, then we use unique key constraint.
What is the difference between Primary key constraint and Unique key constraint?

1. A table can have only one primary key, but more than one unique key
2. Primary key does not allow nulls, whereas unique key allows one null

```
alter table tblPerson
drop constraint uq_tblPerson_AdhaarNum
```

Adding new Column

```
Alter table tblPerson
Add  Age int
```

Altering Column DataType

```
ALTER TABLE tblGender
Alter Column Gender VARCHAR(10)
```