**Database:** create database database\_name

**Use Database:** use database\_name

**Drop Database:** drop database database\_name

**Constraints:** not null, unique, default (default value), primary key, foreign key (foreign key references table\_name (col\_name)), check (check (condition)), identity (identity (1,1) aka identity (start, step)

**Data Types:**

*Numeric:*

bit – 0, 1 or null (1 byte)

tinyint – 0 to 255 (1 byte)

smallint – -32768 to +32767 (2 bytes)

int – -2,147,483,648 to +2,147,483,647 (4 bytes)

bigint – -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 (8 bytes)

decimal (7,3) – 7 digit whole numbers and 3 digits after decimal (XXXX.XXX) (5 to 17 bytes)

*Text:*

char (255) – Alphanumeric

nchar (255) – Unicode

varchar (8000) – Alphanumeric

nvarchar (4000) – Unicode

text - 62000+ Chars (Product Feedback, Comment)

ntext - 62000+ Special Characters (Product Feedback, Comment)

varchar(max) – Alphanumeric

nvarchar(max) – Unicode

*Datetime (YYYY-MM-DD):*

date (YYYY-MM-DD) (3 bytes)

smalldatetime (YYYY-MM-DD HH:MM:SS) – precision is up to seconds (4 bytes)

datetime2 (YYYY-MM-DD HH:MM:SS:XXXXXXX) – precision is up to milliseconds (6 to 8 bytes)

datetime (YYYY-MM-DD HH:MM:SS) – precision is up to seconds (8 bytes)

**System Defined Functions:**

*String Functions:*

select upper('rajat') – converts the text to upper case

select lower('RAJAT') – returns the text in lower case

select len(' rajat123 ') as lengthofstring – returns the length of the text

select 'Rajat' + ' ' + 'Dutta' – concatenates the strings

select left('rajat',2) – returns the 2 characters for the left of the string

select right('rajat',4) – returns the 4 characters from the right of the string

select trim(' Rajat Dutta ') – o/p 'Rajat Dutta', removes leading and trailing spaces from the string

select replace('Rajat', 'aja', 'j') as op – replaces a part of the string with another value

select reverse('Rajat') – reverses a string

select substring('Rajat',3,10) – returns the part of text from starting position to the number of characters specified

select charindex('j','Rajat') – tries to search for the expression and returns its first location

select concat('Rajat',' ', 'Dutta') – concatenates multiple strings

*DateTime Functions:*

select getdate() – This function returns the current server date time

select getutcdate() - This gets us the UTC date and time (Universal Time Coordinated/Greenwich Mean Time)

select getdate() + 2 – This will add 2 days to my current date

select getdate() - 2 – This will subtract 2 days from my current date

select dateadd(hour, 2, getdate()) – This will add 2 hours to now

select dateadd(hour, -2, getdate()) – This will deduct 2 hours from the current date time

select dateadd(year, 2, getdate()) – This adds 2 years to the current date time

select dateadd(year, -2, getdate()) – This will deduct 2 years from the current date time

select dateadd(month, 2, getdate()) – This adds 2 months to the current date time

select dateadd(month, 2, dateadd(hour, 3, getdate())) – This adds 3 hours and 2 months to the current date

select year(getdate()) – Returns the year of the specified date

select month(getdate()) – Returns the month of the specified date

select datepart(day, getdate()) – Returns the date part of the current datetime

select datepart(hour, getdate()) – Returns the hour part of current datetime

select datepart(minute, getdate()) – Returns the minute part of current datetime

select datepart(year, getdate()) – Returns the year part of current datetime

select datepart(month, getdate()) – Returns the month part of current datetime

select datename(weekday, getdate()) – Get name of weekday

select datename(month, getdate()) – Get name of month

select datediff(day, '09/03/2021', '09/07/2021') - returns the difference in days between the start and end date

select datediff(hour, '09/03/2021', '09/07/2021')

select datediff(minute, '09/07/2021 20:00', '09/07/2021 23:00')’

**Table:** create table table\_name (col\_name1 datatype [constraint1|constraint2 …], col\_name2 datatype [constraint1 | constraint2 …] …)

**Insertion:** insert into table\_name [(col\_name1, col\_name2 …] values (value1, value2 …) **OR**

insert into table\_name [(col\_name1, col\_name2 …)] select value1, value2 … **OR**

insert into table\_name [(col\_name1, col\_name2 …)] output inserted.col\_name1, inserted.col\_name2 …) values (value1, value2 …)

**Explicit Insertion into Identity Column:** set identity\_insert table\_name on (after insertion) set identity\_insert table\_name off

**Backup Table:** select \* into backup\_table\_name from table\_name **–** Entire Table **OR**

select \* into backup\_table\_name from table\_name [where condition] **–** Selected Rows **OR**

select col\_name1, col\_name2 into backup\_table\_name from table\_name [where condition] **–** Selected Columns [and Rows]

**Alter Table:** alter table add col\_name datatype **OR**

alter table add col\_name datatype [constraints] **OR**

alter table add constraint [constraint1 | constraint2 …] (col\_name1, col\_name2 …) **OR**

alter table table\_name alter column col\_name datatype **OR**

alter table table\_name drop column col\_name1, col\_name2 … **OR**

alter table drop column [if exists] col\_name

**Update Table:** update table\_name set col\_name = value [where condition]

**Delete Records from Table:** delete table\_name where col\_name = value

**Truncate Table:** truncate table table\_name

**Drop Table:** drop table [if exists] table\_name

**Listing all tables in a DB:** select \* from sys.tables **OR** select \* from INFORMATION\_SCHEMA.TABLES

**Query Process Steps**

1. Getting Data (from, join)

2. Row Filter (where)

3. Grouping (group by)

4. Group Filter (having)

5. Return Expressions (select)

6. Order & Paging (order by & limit/offset)

**Variables:**

*Declaring a variable:*

declare @variable\_name1 [as] datatype, @variable\_name2 [as] datatype … **OR**

declare @variable\_name1 [as] datatype [=''], @variable\_name2 [as] datatype [=''] … **OR**

declare @variable\_name1 [as] datatype [=value1], @variable\_name2 [as] datatype [=value2] …

*Setting a variable:*

set @variable\_name1 = value1

set @variable\_name2 = value2

:

**OR**

select @variable\_name1 = value1

:

**Table Variable:** declare @table\_name table (col\_name1 datatype [primary key | unique | not null | check (condition)], col\_name2 datatype [unique | not null | check (condition)] …)

*Using a Table Variable:* select \* from @table\_name [as alias]

**Common Table Expression (CTE):** ;with cte\_name1 [col\_names] [, cte\_name2 [col\_names]] as (SQL query)

select \* from cte\_name1 [union select \* from cte\_name2] *(CTE Usage is Mandatory right after creation in the same batch and it can be used just once)*

**Index:** create [or alter] [clustered | nonclustered | unique] index index\_name on table\_name (col\_name1 [, col\_name2, col\_name3… ])

**Ideal column to create Clustered Index:** Ideally the column should Unique, Static, Narrow, Non-nullable, Fixed Width, Ever-increasing

**Covering Index using INCLUDE:** create [or alter] [nonclustered | unique] index index\_name on table\_name (col\_name1) include (col\_name2, col\_name3…)

**Filtered Index:** create [or alter] [nonclustered | unique] index index\_name on table\_name (col\_name1, col\_name2…) where col\_name1 = value and col\_name2 = value2

**View:** create [or replace] view view\_name [with schemabinding] as select column1, column2 ... from table\_name [where condition]

*Calling a view:* select \* from view\_name [as alias] [where condition]

**Used Defined Functions:**

**Scaler-valued Functions:**

create [or alter] function function\_name (@param\_name1 datatype, param\_name2 datatype …) returns datatype as

begin

SQL statements

return value

end

*Calling a function:* select dbo.function\_name (value1, value2 …) [as alias] from table\_name [where condition]

**Table-valued Functions:** create [or alter] function function\_name (@param\_name1 datatype, param\_name2 datatype …) returns table as return SQL statements

**Stored Procedure:**

create [or alter] procedure (OR proc) procedure\_name [@param\_name1 datatype [='' OR =value1], param\_name2 datatype [=value2 OR =''] …] as

[begin]

SQL statements (select, insert, delete, update statements)

[end]

*Executing a Procedure:* execute (OR exec) procedure\_name [param\_value1, param\_value2 …]

**Checking for existing SPs:** select \* from sys.procedures **OR** select \* from sys.objects where type='P' [and name like '%sp\_%']

**Dynamic SQL:**  
create [or alter] proc (OR procedure) procedure\_name @param\_name varchar(max) as

declare @SQL varchar(max)='select \* from ' + @param\_name

exec (@SQL)

*Executing a Dynamic SQL:* execute (OR exec) procedure\_name [param\_value1, param\_value2 …]

**DML Triggers:**

create [or alter] trigger trigger\_name on table\_name {for, after, instead of} {insert, delete, update} [not for replication] as SQL statements (select, insert, update or delete statements)

**Error Handling:**

begin try

--SQL statements

end try

begin catch

-- SQL statements if an error occurs

end catch

ERROR\_NUMBER – Returns the internal number of the error

ERROR\_STATE – Returns the information about the source

ERROR\_SEVERITY – Returns the information about anything from informational errors to errors user of DBA can fix, etc.

ERROR\_LINE – Returns the line number at which an error happened on

ERROR\_PROCEDURE – Returns the name of the stored procedure or function

ERROR\_MESSAGE – Returns the most essential information and that is the message text of the error

**Data Storage Structure/Allocation Units:**

IN\_ROW\_DATA

Every row in a table has an IN\_ROW\_DATA portion

All fixed width columns must be stored in the IN\_ROW\_DATA portion

IN\_ROW\_DATA can’t span pages

ROW\_OVERFLOW\_DATA

Tables with potential wider rows might have one or more columns that can overflow to ROW\_OVERFLOW\_DATA structure

Data types in this category: varchar(n), nvarchar(n), varbinary(n), sqlvariant

Vertical partitioning of tables basis most common use patterns can be a solution to avoid critical columns to overflow into this unit

LOB\_DATA

Tables with potential for extremely wide rows might have one or more columns that have been stored in LOB\_DATA structure

Data types in this category: text, ntext & image (legacy LOB datatypes till SQL Server 2005); varchar(max), nvarchar(max), varbinary(max), XML & CLR user defined types (new LOB datatypes)