**Relational Database Management System(RDMS)**

**Normalization:** Normalization is the process of organizing the data to avoid duplicates and redundancy.

**Data Types:** Character, Numeric, Date and Time, and Binary(varbinary(n).

**SQL Statements**

**Data Definition Language:** Create, Alter, Drop.

**Data Manipulation Language**: Select, Insert, Update, Delete.

**Data Control Language:** Grant, Revoke.

**Transaction Control Language:** Commit, Rollback, Savepoint.

**Create a Database**

USE master;

GO

CREATE DATABASE Sales

ON

( NAME = Sales\_dat,

FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\DATA\saledat.mdf',

SIZE = 10,

MAXSIZE = 50,

FILEGROWTH = 5 )

LOG ON

( NAME = Sales\_log,

FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL13.MSSQLSERVER\MSSQL\DATA\salelog.ldf',

SIZE = 5MB,

MAXSIZE = 25MB,

FILEGROWTH = 5MB );

GO

**Create a Table**

Use Database name

CREATE TABLE Test1

( empid int primary key,

L\_name varchar(50) NOT NULL,

F\_name varchar(50) NOT NULL

)

**Alter Table**

ALTER TABLE (Name) ADD (Column Name) dtype

ALTER TABLE (Name) DROP Column (Column Name)

Also you alter dtype of the columns.

**Drop Table**

DROP TABLE <Table Name>

**Various Constraints**

NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK.

CREATE TABLE emp

(

emp\_id INT PRIMARY KEY,

L\_NAME VARCHAR(50), NOT NULL OR

CONSTRAINT emp\_pk PRIMARY KEY (emp\_id) (to create multiple primary key use this)

);

emp-pk is an constant and it should be unique in the database.

**Unique column can have NULL values, but primary key can’t.**

**How to create two related table like parent and child table.**

CREATE TABLE product

( prod\_id int primary key,

Prod\_name varchar(50) not null,

Category varchar(50)

);

CREATE TABLE orders

( order\_id int primary key,

Prod\_id int not null,

Quantity int,

CONSTRAINT fk\_product\_id

FOREIGN KEY (prod\_id)

REFERENCES product (prod\_id)

ON DELETE CASCADE(when you delete column in parent child also deletes use this key word)

);

**Creating a VIEW**

A view is named, derived, virtual table. A view takes the output of a query and treats it as a table.

CREATE VIEW <NAME> AS SELECT \* from <table name>.

**TABLE COMMANDS**

Insert statements :

USE <db\_name)

Insert into <table\_name) (column1, column2..) values (val1, val2…);

**You can also insert data from another table into another table.**

Update Statements

UPDATE <table\_name> SET <field name> = Value where <Condition>;

Delete Statements

DELETE from <table name> where <condition>

Select Statements

SELECT <fields…> from <table name> where <conditions>;

Order By : Used along with where clause to display the specified column in ascending order or descending order.

Filtering: Logic operators[AND, OR, NOT]

Filtering: Comparison Operators[>, <, !=, <>, >=, Between, LIKE, IN, NOT IN]

**[= operator don’t work with NULL, You have to use IS NULL to drifter column have NULL values]**

CASE Expression

SELECT CASE( column\_name)

When condition1 then result1

When condition 2 then result2

Else result

END

**SQL JOINS**

SELF JOIN : A table can be joined to itself in a self join.

INNER JOIN : Inner join fetches records that have mating values in both the tables.

SELECT < column\_name>

From <table name> as t1

Inner join <table name 2> as t2

ON t1.column name = t2.column name;

LEFT OUTER JOIN : Return the rows to the left, even if there are now rows on the right of joins. If there are no matching rows on right table, it returns NULL values for that particular row.

SELECT < column\_name>

From <table name> as t1

Left outer join <table name 2> as t2

ON t1.column name = t2.column name;

RIGHT OUTER JOIN :Opposite to above.

FULL OUTER JOIN: Return all the rows, If no matching values it returns NULL values.

CROSS JOIN /CARTESIAN PRODUCT : Displays all the rows and all the columns of the tables. This is like multiplying all the rows of table with all the rows of table b.

**GROUP BY :** groups the rows that have same values into a summary rows. Group by statement is used when we make use of aggregate functions. It is like if you are using group by it's mandatory to aggregate functions like AVG(), SUM(), MIN(). Also should inclue all the columns used in select statement in group by clause.

**HAVING** : Having clause was added to SQL because the where keyword can’t be used with aggregate functions. It is used algo with group by clause with aggregate function as filter.

**SQL BUILT-in FUNCTIONS :** To calculate and manipulate data.

String Functions, Conversion Functions, Logical Functions, Math Functions, Aggregate Functions, Date Functions

**Conversion Function:** Conversion function covert expression from one data type to another.

[CAST, CONVERT, PARSE, TRY\_CAST, TRY\_CONVERT, TRY\_PARSE]

Logical Function : Used to display one of the several values based on condition.

[CHOOSE, IFF]

CHOOSE (index, val 1, val 2, val 3)include

IFF(Boolean\_exp, ‘TRUE’, ‘FALSE’)

**MATH Functions :**

[ABS, RAND, EXP, ROUND, FLOOR, SQRT, CEILING, SQUARE, POWER]

**STRING Function**: Operation on string input value and return string or numeric value.

[ LTRIM, RTRIM, CHAR, STR, CONCAT, CHARINDEX,LEFT, RIGHT, UPPER, LEN, SUBSTRING, REPLACE]

**Aggregate Function:**

[MIN, MAX, AVG]

**DATE TIME FUNCTION:**

[SYSDATETIME, CURRENT\_TIMTSTAMP, DATETIME, DATEPART, DATEDIFF, DATEADD]

**STORED PROCEDURES**

A stored procedure is a set of SQL statements with a name, that has been created and stored in the database.

Store procedures can be defined as the set of logical group of sql statements which are grouped to perform a specific task. Like a function in other programming language.

CREATE PROCEDURE

Schema\_name.procedure\_name (

@parameter\_name parameter\_data\_type[OUT])

AS

BEGIN

<SQL statements >

END;

**To execute the stored procedure**

Exec stored \_rocedure\_name @parameter\_name=’Value’ → this will give result.

**User Defined Function**

1. Scalar Function - return only single value.

CREATE FUNCTION

Shema\_name.function\_name (@parameter\_name parameter\_data\_type)

RETURNS return\_data\_type

AS

BEGIN

RETURN <SQL statement>;

END

1. Inline Table valued function -Return a whole table.

CREATE FUNCTION

Shema\_name.function\_name (@parameter\_name parameter\_data\_type)

RETURNS TABLE

AS

RETURN

(

<SQL statement>

);

GO

**TRIGGERS**

A rigger is a kind of stored procedure that automatically executes when an event occurs in the database.

CRETE TRIGGER

Shema\_name.trigger\_name

ON TABLE | database

FOR | AFTER | INSTEAD OF

[INSET],[UPDATE],[DELETE]

AS

SQl\_statement.

There are DDL and DML triggers.

**SQL INDEX**

Indexes are used to retrieve data from the database more quickly than otherwise.

CREATE INDEX index\_name

On TABLE(column1, column2);

**WINDOW FUNCTIONS**

Window function perform a calculation across a set of table rows that are somehow related to current row.

Window function apply aggregate and ranking function over a particular window (set of rows)

SELECT column\_name

window\_function(column name) OVER (partition by column\_name ORDER\_BY column\_name)

AS column\_new\_name

From table.

RANK- 1,1,3

Dense Rank- 1,1,2

**SQL LIMIT :** Limit the record to display.

SELECT

select\_list

FROM

table\_name

LIMIT [offset,] row\_count; (for 2nd rank helpful)