SQL

Structured Query Language

Introduction:

SQL is a language which is used to interact with RDBMS is basically just a software application which we can use to create and manage different databases.

- Database is collection of data
- Structured Query Language (SQL) lets you perform CRUD operations Create/Read/Update or Delete operations on a database
- Perform administration tasks (security, import/export etc)
- There are many types of database - Relational, Hierarchical, Network, NoSQL etc.
- Examples of RDBMS
    - MySql – OpenSource
    - SQL Server – Microsoft
    - Oracle – IBM
    - PostgreSQL – OpenSource

Types of SQL:

1. **Data Query Language (DQL):**

   Ex: SELECT

   Purpose: Used to query the database for info
2. **Data Definition Language (DDL):**

   Ex: Create, Alter and drop

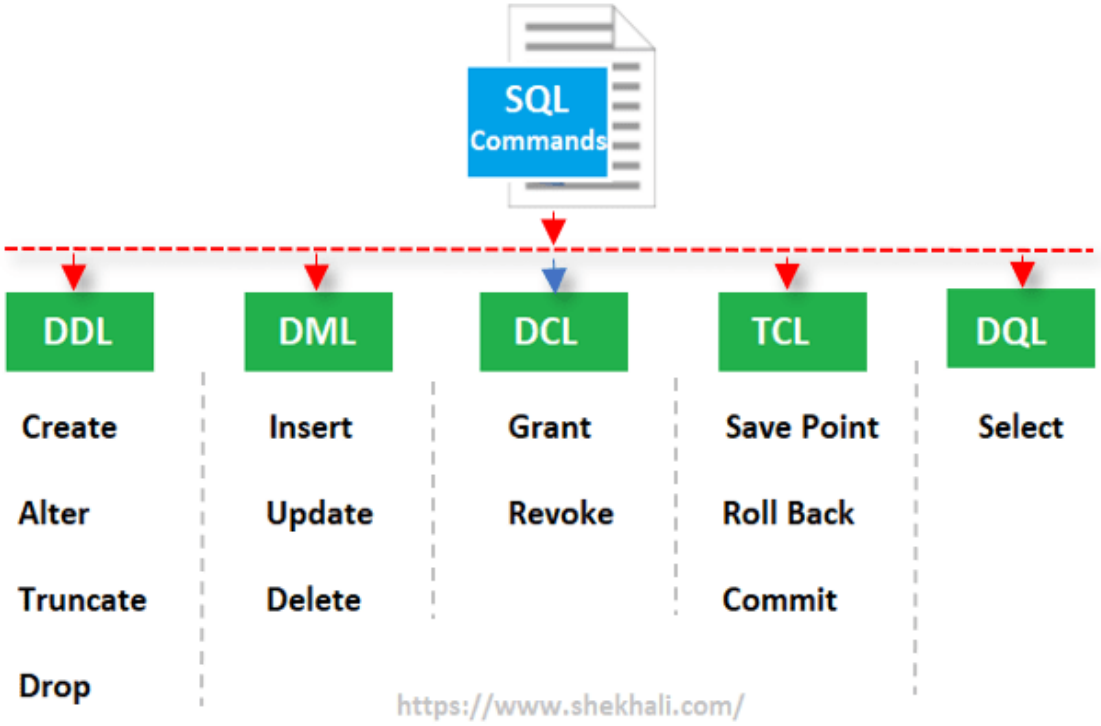
   Purpose: Used for defining database schemas
3. **Data Manipulation Language (DML):**

   Ex: Insert, Update and delete

   Purpose: Used for CRUD
4. **Data Control Language (DCL):**

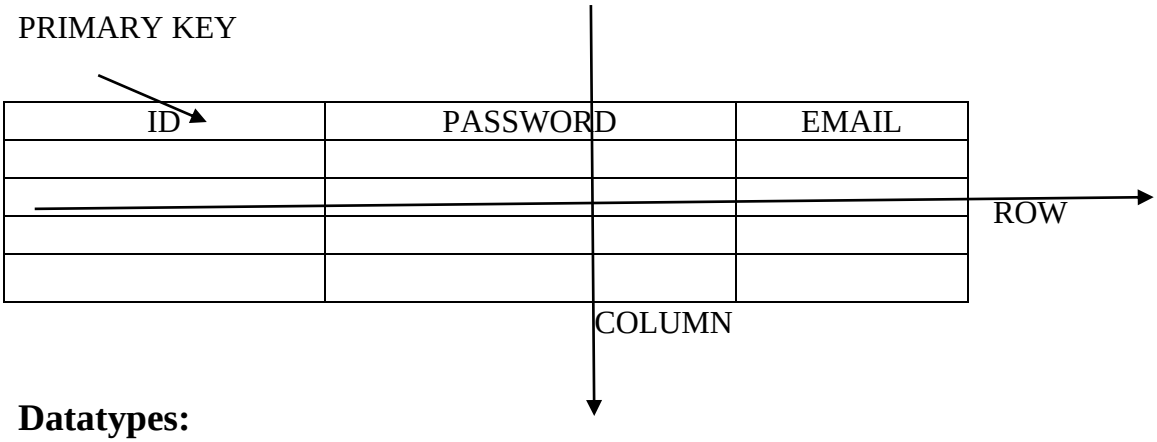
   Ex: Grant, Revoke

   Purpose: User and permission management



Relational Databaes(sql):

PRIMARY KEY



Datatypes:

Character Data

char - eg:char(5) stores fixed length string of length 5. Max 255 bytes.

varchar - eg:varchar(5) stores variable length string of length 5. Max 65535 bytes.

Example Query:

SHOW CHARACTER SET; -- shows various character sets that are supported.

Text Data

All images in the section are from Learning SQL by Alan Beaulieu

Text type	Maximum number of bytes
Tinytext	255
Text	65,535
Mediumtext	16,777,215
Longtext	4,294,967,295

Temporal Data :

Type	Default format	Allowable values
Date	YYYY-MM-DD	1000-01-01 to 9999-12-31
Datetime	YYYY-MM-DD HH:MI:SS	1000-01-01 00:00:00 to 9999-12-31 23:59:59
Timestamp	YYYY-MM-DD HH:MI:SS	1970-01-01 00:00:00 to 2037-12-31 23:59:59
Year	YYYY	1901 to 2155
Time	HHH:MI:SS	-838:59:59 to 838:59:59

Data Definition Language (DDL): Create, Alter, Drop and Truncate

Create:

- Used to create a database/schema
- Used to create a new table

EX:

```
CREATE DATABASE SAMPLE;   -- creates a new database
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Department VARCHAR(50),
    Salary DECIMAL(10, 2)
);
```

ALTER

- It is used to modify/alter the value in a table

Alter:

-- (Adding a new column in table)

```
ALTER table employee
ADD email varchar(20);
--( Datatype change )
Alter table employee
Alter column email varchar(100);
--(renaming a name column )
EXEC sp_rename 'table_name.old_column_name', 'new_column_name', 'COLUMN';
```

DROP

```
-- TO DELETE A DATABASE
```

- It is used to drop/delete a table,view and index

DROP DATABASE SAMPLE;

DROP SCHEMA SAMPLE; -- same as above. u can use DATABASE Or SCHEMA

DROP SCHEMA IF EXISTS SAMPLE; -- prevents error if db not found

TRUNCATE

- It is used to delete the rows of a table and remains the table structure

TRUNCATE TABLE example_table;

DATABASE COMMANDS:

SHOW

SHOW DATABASES; -- shows all the databases

SHOW SCHEMAS; -- same as above. shows schemas/db

USE SYS; -- uses this database for all further commands

SHOW TABLES;-- shows all tables in the database being used

Data Manipulation Language (DML): Insert,update and delete

Inserting Data:

- Insert keyword is used to data/value in a table

Method 1

insert into customer values (1,"John Doe","392 Sunset Blvd","New york","NT","10059");

insert into customer values (2,"Mary Smith","6900 Main St.","San Franciso","CA","94032");

insert into customer values (3,"Richard Newman","2040 Riverside Rd.","san diego","ca","92010");

insert into customer values (4,"Cathy cook","4010 speedway","tucson","Az","85719");

Method 2:

INSERT INTO employees (employee_id, first_name, last_name, salary)

VALUES (1, 'John', 'Doe', 50000);

(2, 'Jane', 'Smith', 60000),

(3, 'Bob', 'Johnson', 55000),

(4, 'Alice', 'Williams', 70000);

SELECT

- Select keyword is used to retrieve a data

SELECT * from Student

UPDATE

- Update keyword is used to change a value and set a new value

UPDATE employees

SET first_name = 'Jonathan'

WHERE employee_id = 123;

DELETE

- Delete keyword is used to delete a row values

DELETE FROM employees

WHERE employee_id = 3;

AGGREGRATE FUNCTIONS:

- Aggregrate functions is used to calculate a set of values and return a single value.

1. **SUM()**

Adding the number by using the sum()

EX:

SELECT sum(student_name) from student

2. **MAX()**

- Find the maximum value in column

EX:

Select MAX(student_mark) from student

3. **MIN()**

- Find the minimum value in column

EX:

Select MIN(student_mark) from student

4. **AVG()**

- Calculate the average value

EX:

Select avg(student_mark) from student

5. COUNT()

- Count the number of rows in a table

  EX:

  Select count(student_mark) from student
  where department =’CEC’

ORDER BY

- Two types of ORDER BY exists : Ascending order and Descending order
- ASCENDING ORDER SORTS FROM LOWEST TO HIGHEST VALUE
- DESENDING ORDER SORTS FROM HIGHEST TO LOWEST VALUE
- When you use the ORDER BY clause in SQL without specifying ASC (ascending) or DESC (descending), the default sorting order is ascending.

Print the names of students and their corresponding marks in ascending order

SELECT student_name,mark from student **order** by mark **asc**

Print the names of students and their corresponding marks in descending order

SELECT student_name,mark from student **order** by mark **desc**

GROUP BY

- GROUP BY helps organize and summarize data, often used with aggregate functions.
- It allows you to group rows based on common values in specific columns.

SELECT AVG(mark),department from student **group by** department

Using agg,group by and order by

Select count(student_name),department from student **group by** department **order by** count(student_name) **desc**

HAVING CLAUSE

- HAVING clause is used in combination with the GROUP BY clause to filter the results of a grouped query based on specified conditions.
- Having clause doesn't filter the values without using group by

EX:

SELECT AVG (salary), department from employee **group by** department **having** (salary) < 60000

GROUP BY AND HAVING
WHERE Filters Rows
Having Filters Groups

LIKE AND WILDCARDS

- LIKE is used in combination with wildcard characters to perform pattern matching in strings.

Basic Idea:

LIKE is a keyword used to search for a specified pattern in a column.

Wildcard Characters:

- % (Percent Sign): Represents zero or more characters.
- _ (**Underscore**): Represents a single character.

Examples:

LIKE 'a%': Finds any values that start with "a".

LIKE '%a': Finds any values that end with "a".

LIKE '%or%': Finds values that have "or" in any position.

LIKE '_r%': Finds any values that have "r" in the second position.

LIKE 'a_%_%': Finds any values that start with "a" and are at least 3 characters in length.

Use Case:

Commonly used in SELECT statements with WHERE clauses to filter rows based on specific patterns.

EX:

Select * from employee where ename like ‘A%’; o/p: Ashok

Select * from employee where ename like ‘A%A’; o/p: Abinaya

Select * from employee where ename like ‘%Z%’; o/p: EZE

Select * from employee where ename like ‘---i%’;

(Starting towards 3rd letter will be ‘I’) o/p: Abinaya

If we want to check whether the percentage sign (%) is located in the middle of the name...?

Select * from employee where ename like ‘—i\%’;

DISTINCT:

- Eliminate duplicate values and obtain the unique or distinct key, resulting in a single value.

Select **distinct** job_desc from employee;

String related functions:

UPPERCASE – UCASE

LOWERCASE - lcase

CHAR_LENGTH – char_length

CONCAT - CONCAT function is used to concatenate (combine) two or more strings into a single string**.**

EX:

Select ucase**(**name),salary from employee

Select lcase(name),salary from employee

Select ename,chart_length(ename) from employee

Select ename,concat('rs',salary) from employee

CONSTRAINTS:

- **Primary key**
- **Foreign key**
- **Unique**
- **Check**
- **Default**

- **Not NULL**

Primary key	**Unique Key**
Doesn’t accept null values	Accepts null values
A table can have only one primary key	A Table can have multiple unique constraints

EX_QUERY

```
CREATE TABLE Employee (

   Emp_ID INT PRIMARY KEY IDENTITY(1,1),

   Ename VARCHAR(100) NOT NULL,

   Job_Desc VARCHAR(100) DEFAULT 'unassignment',

   Salary INT,

   Pan VARCHAR(10) UNIQUE,

   CHECK (Salary > 100000)

);
```

NOT NULL

Column to not accept null values

Column should always have a value

--Add not null constraint

ALTER TABLE Employee

ALTER COLUMN Ename VARCHAR(100) **NOT NULL**;

--drop not null constraint

ALTER TABLE employee

ALTER COLUMN email **DROP NOT NULL**;

--Add Default Constraint

ALTER TABLE YourTable

ADD CONSTRAINT DF_ConstraintName DEFAULT default_value FOR YourColumn;

--**Drop default constraint**

ALTER TABLE YourTable

DROP CONSTRAINT ConstraintName;

--add check constraint

ALTER TABLE YourTable

ADD CONSTRAINT ConstraintName CHECK (condition);

--drop check constraint

ALTER TABLE YourTable

DROP CONSTRAINT ConstraintName;

UNIQUE CONSTRAINT

All values in column should be different

Guarantee the uniqueness of data

Single column/combinational unique column

```
Create table unique_const (

Id int not null,

Name varchar(10),

Constraint cons_name unique (id,name)

);
```

PRIMARY KEY

- Which Uniquely identifies the each record in a table
- Primay key will Not allows NULL
- Pirmary key will not allow duplicate values
- Primary key can have one or multiple column

EX:

```
Student_id int primary key
```

PRIMARY KEY VS UNIQUE KEY

PRIMARY KEY	UNIQUE KEY
It will not allow duplicate values	It will also not allow duplicate values
Primary key will not allow null values	It will allow one null value
You can define only one primary key in a table	You can define more than 1 unique key in table
Primary key is used in foreign key constraint	Unique key cannot be used in foreign key constraint

FOREIGN KEY

- Relationship between two tables based on the values of one or more columns.
- Two tables will be connected in sql with the use of foreign key.
- Foreign key table is called as child and primary key table is called as parent
- All values in a foreign key column will refer to primary key in another table

Consider two tables: Orders and Customers.

- Orders Table:

OrderID (Primary Key)

Product

CustomerID (Foreign Key)

- Customers Table:

  CustomerID (Primary Key)

  CustomerName

  ContactNumber

-- Customers table

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(255),

ContactNumber VARCHAR(15)

);

-- Orders table with a foreign key constraint

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

Product VARCHAR(255),

CustomerID INT,

constraint fk_customerid foreign key (CustomerID) REFERENCE Customers(customer ID);

Drop table foreign key

ALTER TABLE YourTable

DROP CONSTRAINT YourForeignKeyConstraint;

TEMPORARY TABLE

- Temporary tables store temporary data during a database session
- Visible and accessible only within the session that created them.

```
Create table #temp_student(

Student_id int primary key,

Student_name varchar(20),

Student_dept varchar(20),

);

Select * from #temp_student
```

COMMON TABLE EXPRESSION

- **CTE** is used to breaking down the query into smaller
- It is not store in database and only valid in the duration

EX:

```
WITH Employee_CTE AS (
   SELECT AVG(salary) AS avg_salary, job_desc
   FROM employee
   GROUP BY job_desc
)
SELECT *
FROM Employee_CTE
```

ORDER BY avg_salary DESC;

SUBQUERY

- Subquery is also known as nested query
- Subquery is a query nested into another query

Ex:

Select name from students

Where age > (**select** avg (age) from students) ;

SELECT TOP

- It is used to print the no. of limit in rows

Select * from students

Where age=30

Limit 3

ADDING VALUE WHERE AGE IS NULL

- If you want to update a column in a table where the age is currently NULL, you can use the UPDATE statement with a SET clause.

Select

Case

When age is null then -1

Else age

End as Modified_age

From students;

Select * from students

INCREMENTING AGE

- If you want to increment the values in the "age" column for all rows in a table, you can use the **UPDATE** statement with the **SET** clause along with the **+** operator for incrementing.

Select

Case

When age is not null and age <=22 then age +2

Else age

End as Modified_age

From students

Select * from students

JOINS

- Joins has been connected one or more table
- Joins is to be interact one or more table to extract the values,

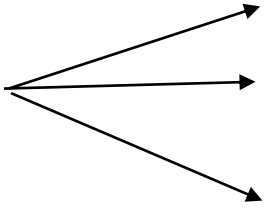
Joins are 4 types:

INNER JOIN

OUTER JOIN

SELF JOIN

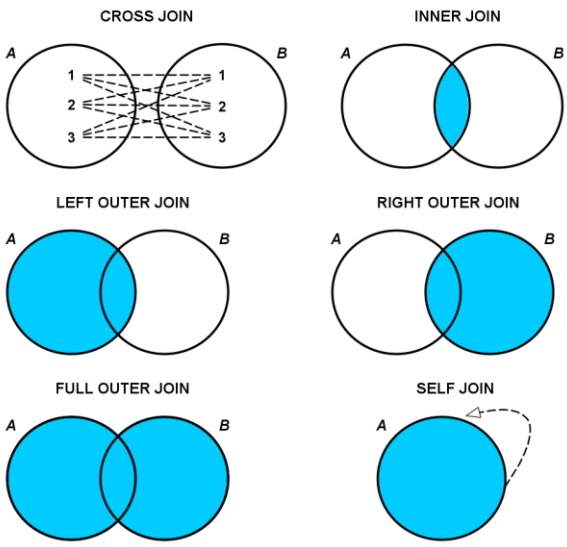
CROSS JOIN



LEFT OUTER JOIN

RIGTH OUTER JOIN

FULL OUTER JOIN



EX:

Create table branch (

branch_id int primary key identity (1,1),

branch_name varchar(100) not null,

address varchar(100)

);

Create table Employee(

emp_id int primary key identity (1,1),

ename varchar(100) not null,0

job_desc varchar(50),

salary int,

branch_id int,

constraint fk_branchid foreign key(branch_id) references branch(branch_id)

);

select * from Employee

select * from branch

Inner join

- Inner join is used to combine a table with one or more and will be retrieve the values with matching values in specified column and it removes the non- matching values.
- In SQL Server, you can use either the INNER JOIN or JOIN keyword to perform Inner Join. If you use only the JOIN keyword, then it is also going to perform Inner JOIN in SQL Server.

Ex:

```
select
employee.emp_id,employee.ename,employee.job_desc,branch.branch_name

from employee

join branch

on employee.branch_id=branch.branch_id

order by emp_id
```

Left join

- It is used to retrieve all the matching rows from both the tables involved in the join as well as non-matching rows from the left side table.

```
select
employee.emp_id,employee.ename,employee.job_desc,branch.branch_name

from employee
```

left join branch

on employee.branch_id=branch.branch_id

order by emp_id

Right join

- A RIGHT JOIN returns all the rows from the right table and the matching rows from the left table.

select
employee.emp_id,employee.ename,employee.job_desc,branch.branch_name

from employee

right join branch

on employee.branch_id=branch.branch_id

order by emp_id

Using table name alias

- Alias is use to be temporary named column or table
- Alias is mainly useful for joins

SELF JOIN

SELECT e.emp_id,e.ename,e.job_desc,b.branch_name

FROM employee AS e

INNER JOIN branch AS b

ON e.branch_id=b.branch_id

ORDER BY e.emp_id;

VIEW

- View as virtual table
- It doesn’t store data in database

CREATE VIEW HIGHEST_SALARY AS

SELECT ENAME,SALARY

FROM EMPLOYEE

WHERE SALARY >= 800000;

SELECT * FROM HIGHEST_SALARY

ROW NUMBER

- Row number is a function that generates the row numbers.

select row_number() over(order by salary desc) as ROW_NUMBER,salary

from Employee

RANK

- Function is used to assign a unique rank to each row within the function

Select row_number() over(order by salary desc) as ROW_NUMBER,

rank() over(order by salary desc) as RANK , salary

from employee

DENSERANK

- Similar to RANK(), the DENSE_RANK() function is used to assign ranks to rows based on the values of one or more columns.

```
select row_number() over(order by salary desc) as ROW_NUMBER,

rank() over(order by salary desc) as RANK,

DENSE_RANK() over(order by salary desc) as DENSE_RANK

from employee
```

NOTES:

Rank - If two values match, the subsequent value will skipped.

Denserank - If two values match, the subsequent value isn't skipped.

PARTITION

- It is used to divide the result set into partitions based on one or more columns.

```
select row_number() over(Partition by salary order by salary desc) as
ROW_NUMBER,

rank() over(order by salary desc) as RANK,

DENSE_RANK() over(order by salary desc) as DENSE_RANK

from employee

SELECT * FROM EMPLOYEE
```

TRANSACTION

- If there is an error in either update, the transaction can be rolled back to maintain data consistency.

EX:

Begin transaction

Update production set reorder_point=245 where product_ID = 1

Rollback transaction

SYSTEM DATABASES

- The System databases are used to store system wide data and metadata.
- System databases are mandatory databases. They are 4 types of sys databases

1. Master
2. Model.
3. MSDB.
4. Temp db.

- Each of system database is used by SQL Server for Separate purposes.
- From all the databases, master database is the most important database.

MSDB Database

- The msdb is used by the SQL Server Agent for scheduling jobs and alerts. Also, it stores the history

MODEL Database

- SQL Server uses the model database as the template for creating other databases.
- When you create a new database, SQL Server copies the contents of the model database including database options to the new database.

TEMPDB

- The tempdb database stores temporary user objects that you explicitly create , table variables.

MASTER

The master database stores all the system-level information of an SQL Server instance, which includes:

- Server configuration settings
- Logon accounts
- If the master database is unavailable, the SQL Server cannot start.

Server databases:

There are 3 types of server databases :

- **MDF**
- **LDF**
- **NDF**

MDF(Primary Data File):

- MDF stands for Master data file or primary data file.
- Its is mainly used to store the system tables, user tables in sql server database

LDF (Log data file):

- LDF Stands for log data file
- It is used to store the transaction file.
- While updating , inserting or deleting the transaction log will be stored

NDF (secondary data file):

- NDF stands for secondary data file
- It is useful for secondary data file set, it stores the small data files

Stored procedure

- A stored procedure is like a set of instructions that you can save and reuse in your database.
- It can also pass parameters to store the procedure
- Stored procedure are written and store directly in database

EX:

Basic syntax:

```
Create procedure sp_sales (procedure name)

As

Begin

Select * from sales.creditcard (table name)

End

EXEC sp_sales
```

Altering Store Procedure:

Stored procedure will pass input parameters

```
Alter procedure sp_sales

As
```

```
Begin

Select * from sales.creditcard

Where ExpMonth = @ExpMonth and CardType = @CardType and
ExpYear = @ExpYear

End

EXEC sp_sales @Expmonth = 3,

                @CardType = 'vista',

               @ExpYear = 2006
```

Creating Optional Parameter (Default Values of Passing Parameter) :

```
Alter procedure sp_sales ( @Expmonth as int =3,

                        @CardType as varchar(10) = 'vista' )

As

Begin

Select * from sales.creditcard

Where ExpMonth = @ExpMonth and CardType = @CardType and
ExpYear = @ExpYear

End

EXEC sp_sales
```

Getting OutPut Parameters

```
ALTER PROCEDURE SP_SALES(
                                        @CARDNUMBER NVARCHAR (50),
                                        @EXPYEAR INT ,
                                        @CNT INT OUTPUT)
AS
BEGIN
SELECT @CNT = count(1) FROM SALES.CreditCard
WHERE CARDNUMBER = @CARDNUMBER  AND
EXPYEAR=@EXPYEAR
END


DECLARE @NEW_CNT INT
EXEC SP_SALES
                        @cardnumber='11117174633569',
                        @expyear=2006,
                        @CNT = @NEW_CNT OUTPUT ;
SELECT @NEW_CNT
```

Index

- Index are used to retrieve the data from database more quickly compare to others.
- It Improves the data retrieve speed
- It Creates using index one or more column in a table
- It Creates an index files that contains only the logical order of rows along with their physical position
- Index are 2 types:

- Cluster index
- Non-Cluster index

- When you create a **primary key** constraint by default creates an **Cluster** index in sql server.
- When you define a **unique key** in sql server a **Non-Cluster** index is automatically created.

EX:

Single index:

Create index index_name on table_name (column_name)

Creating cluster index:

Create clustered index index_name on table_name (column_name)

Creating Non-cluster index:

Create Nonclustered index index_name on table_name(column_name)

Creating index using of index:

Create unique index index_ame on table_name(column_name)

Drop index:

Drop index index_name on table_name

Triggers

- Triggers cannot be manually executed.
- If your main table changes, a trigger can automatically change record values.

- Triggers work in the background, waiting for the right moment to jump into action.

Three types of triggers , they are:

- DDL trigger
- DML trigger
- LOGON trigger

DDL Triggers:

- DDL command such as Create_table, Create_view, drop_table, Drop_view, and Alter_table triggers to be activated.
- It is used to respond in changes to structure of the database
- Triggers execute automatically when associated DDL events occurs

EX:

```
CREATE TRIGGER trigger_name

ON DATABASE

AFTER CREATE_TABLE, ALTER_TABLE, DROP_TABLE

AS

BEGIN

   -- SQL statements to execute

END;
```

DML Triggers

- DML command such as Insert, Update, and Delete set off the DML triggers.
- It is used to respond the changes in data

EX:

```
create trigger trg_emp_audit

on employee

after insert

as

begin

insert into employee_audit

select emp_id,'ins',getdate()

from inserted

end
```

Logon trigger

Triggers is a database code that gets executed in case of a precisely defined event. We can use logon triggers to control the SQL login security

Differences between functions and procedures

Functions	Procedures
A function has a return type and returns a value.	A procedure does not have a return type.
You cannot use a function with Data Manipulation queries. Only Select queries are allowed	You can use **DML queries** such as **insert**, **update**, **select** etc… with procedures.
A function does not allow output parameters	A procedure allows both input and output parameters.
You cannot call stored procedures from a function	You can call a function from a stored procedure.
You can call a function using a select statement.	You cannot call a procedure using select statements.

View Limitations:

- View don't store data themselves, they are virtual tables created from existing table.
- You cannot pass parameters to SQL Server views.
- Cannot use an Order By clause with views
- Views cannot be created on Temporary Tables.
- Some types of views, known as indexed views, have more limitations, especially regarding the types of operations

Cursor:

- Cursors are declared and associated with a SELECT query.
- Rows are fetched one at a time using FETCH.
- Forward-Only (one direction) and Scrollable (both directions).

- Cursors are closed with CLOSE.
- Resources are released with DEALLOCATE.

EX:

```
DECLARE @ID int;

DEClARE @value int;

DECLARE @RunningTotal int = 0;

DECLARE  RunningTotalcursor cursor for

select ID,value from salesData

order by id;

--

OPEN Runningtotalcursor

FETCH NEXT FROM RunningTotalcursor INTO @ID,@VALUE


WHILE @@FETCH_STATUS = 0

BEGIN

        SET @RUNNINGTOTAL = @RunningTotal + @VALUE

        UPDATE SALESDATA SET RUNNINGTOTAL =
@RUNNINGTOTAL WHERE ID = @ID

        FETCH NEXT FROM RUNNINGTOTALCURSOR INTO
@ID,@VALUE

END

CLOSE RUNNINGTOTALCURSOR

DEALLOCATE RUNNINGTOTALCURSOR
```

SELECT * FROM SALESDATA

HOW INDEX HELPS TO RETRIEVE DATA FASTER:

- Index makes a search faster by creating a B-TREE structure
- When the records are searched sequentially without B-TREE index and it is called as table scan.
- Index scan uses to quickly locate and retrieve specific rows based on query conditions.
- When the records are searched with the use of B-TREE structure and it called as index seek
- Scans the entire table, row by row, to find and retrieve rows that meet the query conditions.

EXECUTION PLAN:

- step-by-step guide created by the query optimizer to execute a SQL query.
- execution plans and chooses the most efficient one based on factors like indexes, statistics, and query structure
- Plans are reused if subsequent queries match the cached plan structure
- The plan reveals which indexes are utilized during query execution