SQL Server

Triggers- trigger is a stored procedure that executes automatically in response to certain events (such as updates, inserts, and deletions). It is unable to accept input as a parameter. Transaction statements aren't allowed inside a trigger. Triggers do not have the ability to return values.
Advantages- **1-** Maintaining the integrity constraints in the database tables, especially when the primary key and foreign key constraints are not defined. **2-** Helps in maintaining the track of all the changes (update, deletion and insertion) occurring in the tables through inserting the changes values in the audits tables.

```
CREATE TRIGGER [schema_name.]trigger_name
ON table_name
AFTER  {[INSERT],[UPDATE],[DELETE]}
[NOT FOR REPLICATION]
AS  {sql_statements}
```

Stored procedures- set pre-compiled of code that performs a specified operation. It can be called by the user explicitly.It has the ability to accept input as parameter.Within a stored procedure, we can use transaction statements like begin transaction, commit transaction, and rollback. Values can be returned by stored procedures. we can't use a stored procedure inside UDF.
Advantages- Reduce compilation and execution time, Promotes reusability of code

SP performance tuning tips-

1. Use the "SET NOCOUNT ON" directive- you wouldn't see those messages(2 row(s) affected" or "5 row(s) updated), reducing the network traffic and boosting performance.
2. **NOLOCK** allows SQL to read data from tables by ignoring any locks and therefore not get blocked by other processes.This can improve query performance by removing the blocks, but introduces the possibility of dirty reads.
3. Pagination in SQL Server is the process of dividing a large set of query results into smaller subsets, or "pages". This improves performance and user experience. Pagination is often used in applications where displaying an entire dataset at once is impractical. Using CTE, TempTable

4. CREATE PROCEDURE mySchema.spEmployee- Using schema name before the table name, particularly when you have more than one database schema helps narrow the search and fetches results faster

5. Fetch only the required columns
6. Avoid unnecessary statements in between transaction-SQL derived tables, views and JOINs
7. Use EXISTS instead of COUNT in sub queries

```
if EXISTS (SELECT * FROM Sales.SalesOrderDetail WHERE ProductID = 324)
```

8. Do not use sp_<procname>- sp_<procedurename> is a reserved keyword that Microsoft uses to define the in-built stored procedures. Underscore will indicate system stored procedure and as a result to search for your stored procedure, the program has to go through all the system stored procedures (which are a whole lot!) again wasting precious time.

```
ALTER PROCEDURE uspFindProducts(@min_list_price AS DECIMAL)
AS BEGIN
    SELECT    product_name, list_price FROM  production.products
    WHERE  list_price >= @min_list_price ORDER BY list_price;
END;
EXEC uspFindProducts 100;
```

SQL server functions are pre-built actions that perform calculations, manipulate data, and return results. At the most fundamental level, these functions simplify complex queries and automate repetitive tasks.

Difference b/w SP, Function, Trigger

- **Executable**

Store procedure: We can execute the stored procedures when required.
Function: We can call a function whenever required. Function can't be executed because a function is not in pre-compiled form.
Trigger: Trigger can be executed automatically on specified action on a table like, updates, inserts, and deletions.

- **Calling**-

Stored procedure: Stored Procedures can't be called from a function because functions can be called from a select statement and Stored Procedures can't be called from. But you can call Stored Procedure from Trigger.
Function: Function can be called from Stored Procedure or Trigger.
Trigger: Trigger can’t be called from Stored Procedure or Function.

- **Parameter**

SP: Stored Procedures can accept any type of parameter. Stored Procedures also accept out parameters.
Function: Function can accept any type of parameter. But the function can’t accept out parameters.
Trigger: We can’t pass a parameter to trigger.

- **Return**

SP: Stored Procedures may or may not return any values (Single or table) on execution.
Function: Function must return any value.

Trigger: Trigger never return value on execution

Function- DML operations can be done in functions in SQL

SQL Query Optimization Tips

1- Using SELECT in place of SELECT*
2- Proper Indexing:
3- Try to avoid repeating queries
4- Avoid using HAVING, use WHERE instead

5- Avoid too many JOINS:

6- Avoiding correlated subqueries (correlated subquery runs row by row)

Cursor is a database object that retrieves data from a result set one row at a time. Cursors are used when you need to perform operations on each row of a result set rather than on the set as a whole.

```
DECLARE     @product_name VARCHAR(MAX),  @list_price   DECIMAL;
DECLARE cursor_product CURSOR
FOR SELECT Product_name, list_price FROM production.products;
OPEN cursor_product; //open the cursor
FETCH NEXT FROM cursor_product INTO     //fetch each row from the cursor and print out the product name and list price:
 @product_name, @list_price;
WHILE @@FETCH_STATUS = 0
    BEGIN       PRINT @product_name + CAST(@list_price AS varchar);
      FETCH NEXT FROM cursor_product INTO @product_name,@list_price;
    END;
```

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific job and ends when all the tasks in the group are successfully completed. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: **success** or **failure**.

ACID stands for Atomicity, Consistency, Isolation, Durability. They are database transaction properties which are used for guaranteeing data validity in case of errors and failures.

- **Atomicity**: This property ensures that the transaction is completed in all-or-nothing way.
- **Consistency**: This ensures that updates made to the database are valid and follows rules and restrictions.
- **Isolation**: This property ensures integrity of transactions that are visible to all other transactions.
- **Durability**: This property ensures that the committed transactions are stored permanently in the database

COALESCE function returns the first non-null value from a list. If all the values evaluate to null then it will also return null.
SELECT COALESCE(NULL, 2, 1, 3) /* OUTPUT: 2 */

Exceptions in SQL Server are handled using the try and catch block.

```
BEGIN TRY
    --code which might raise exception
END TRY
BEGIN CATCH
    --code to run if error occurs in try block
END CATCH
```

There are some special functions which are only usable inside the catch block. Outside the catch block they return NULL. These functions are:

- ERROR_NUMBER(): As the name says, it returns the error number.
- ERROR_STATE(): It returns the state number of the error.
- ERROR_SEVERITY(): This function returns the severity value of the error.
- ERROR_PROCEDURE(): It returns the name of the stored procedure or function in which the error has occurred.
- ERROR_LINE(): Returns the line number at which the error has occurred.
- ERROR_MESSAGE(): Returns the message about the error.

Temporary Variable-

Temporary Table creation-
Temporary tables in SQL Server also known as temp tables are used by database developers and DBAs to store data temporarily and work on this data for faster performance. Local temporary tables(prefix #,) are stored in TempDB.

Global temporary tables (prefix ##) are also stored in tempdb. Global temporary tables are temporary tables that are available to all sessions and all users.
A temporary table is used to store data within a session from multiple tables using complex queries and where you need data manipulation on the server side before the data can be returned.

```
CREATE TABLE #TEMP_TABLE (ID INT)

INSERT INTO #TEMP_TABLE VALUES(1)
INSERT INTO #TEMP_TABLE VALUES(2)
INSERT INTO #TEMP_TABLE VALUES(3)
```

Table Variable is a variable that can store the complete table of the data inside it. It is similar to a Table Variable but as I said a Table Variable is variable. So how do we declare a variable in SQL? Using the @ symbol.

```
Declare @TempTable TABLE(
id int, Name varchar(20)
)
```
Table Variable is not available after the execution of the complete query so you cannot run a single query but a temporary table is available after executing the query.
Transaction (Commit and Rollback) operation is not possible in a Table Variable but in a temporary table, we can perform transactions (Commit and Rollback).

RANK- function is used to calculate each row's ranking depending on defined attributes and clauses. It skips rank which has the same record value.

DENSE_RANK- function assigns rank number to each row. Not skips any rank which has the same record value.

CTE (Common table expression)- temporary result set which can be used within execution of single insert, update, delete.

Index- It is a database object that is used to improve the performance of search operations. When we create an index on any column of a table, SQL Server internally maintains a separate table called index table. and when we are trying to retrieve the data from the existing table, depending on the index table, SQL Server directly goes to the table and retrieves the data very quickly. In a table, we can use a maximum of 250 indexes.
SQL Server uses indexes of a table provided that the select or update or delete statement contained the "WHERE" clause and moreover the where condition column must be an indexed column. If the select statement contains an "ORDER BY" clause then also the indexes can be used.
```
CREATE [UNIQUE] [CLUSTERED/ NON-CLUSTERED] INDEX <INDEX NAME> ON
<TABLE NAME> (<COLUMN LIST>)
```
To see the index: `sp_helpindex Employee`
`Execute sp_helpindex Employee`
To drop an index: `Drop index Employee.IX_Employee_Id`

SQL Server Indexes are divided into two types. They are as follows:

1. Clustered index
2. Non- clustered index

Clustered Index defines the order in which the data is physically stored in a table.
Table data can be sorted in only one way, therefore, there can be only one clustered index per table. In SQL Server, the primary key constraint automatically creates a clustered index on that particular column. Clustered indexes are faster than non-clustered indexes since they don't involve any extra lookup step.
It is not possible to create more than one clustered index in a table.

```
CREATE CLUSTERED INDEX IX_Employee_Salary ON Employee(Salary)
DROP INDEX Employee.PK__Employee__3214EC07AED992AA
```

create the clustered index with multiple columns and if we do so, then it is called a composite clustered index.

```
CREATE CLUSTERED INDEX IX_Employee_Gender_Salary ON Employee(Gender
DESC, Salary ASC)
```

Identity is a property that can be applied to a column of a table whose value is automatically created by the server. So, whenever you mark a column as identity, then that column will be filled in an auto-increment way by SQL Server. That means as a user we cannot insert a value manually into an identity column.

Syntax: `IDENTITY [(seed,increment)]`

```
PersonId int identity(1, 1),
```
or
```
Alter Table Person Add PersonId INT IDENTITY(1,1)
```

Explicitly supply Values for Identity Column -

1. First, turn on identity insert – `SET Identity_Insert Person ON`
2. Secondly, you need to specify the identity column name in the insert query as shown below

```
Insert into Person(PersonId, Name) values(3, 'Sara')
SET Identity_Insert Person OFF
```

SCOPE_IDENTITY() function returns the last identity values generated in any table in the current session.. It only applies to cases where an identity column is used during data insertion.

Non-Clustered Index A non-clustered index doesn't sort the physical data inside the table. In fact, a non-clustered index is stored at one place and table data is stored in another place. This is similar to a textbook where the book content is located in one place and the index is located in another. This allows for more than one non-clustered index per table.

It is important to mention here that inside the table the data will be sorted by a clustered index. However, inside the non-clustered index data is stored in the specified order. The index contains column values on which the index is created and the address of the record that the column value belongs to.

Non-clustered indexes are created separately from the actual data, so a table can have more than one non-clustered index in SQL Server.

```
CREATE NONCLUSTERED INDEX IX_tblOrder_ProductId
ON dbo.tblOrder (ProductId)
INCLUDE ([Id],[CustomerId],[ProductName]) GO
```

Difference between clustered and non-clustered indexes in SQL Server?

1. We can create only one clustered index per table whereas we can create more than one non-clustered index per table in SQL Server.
2. In the clustered index, the leaf node actually holds the data and in the case of a non-clustered index, the leaf node actually points to the leaf node of a clustered index or points to the heap table if the table does not have any clustered index.

3. The SQL Server Clustered Index determines the order in which the data is physically stored in a table and hence does not require additional disk space whereas a Non-Clustered Index in SQL Server is stored separately from the actual table, so additional disk space is required.

Which Index is faster, Clustered or Non-Clustered Index? The Clustered Index is slightly faster than the Non-Clustered Index. This is because, in the case of the clustered index, the leaf node actually holds the actual data, and hence when we search any data, it directly gets the data from the leaf node. On the other hand, in the case of a Non-Clustered Index, the leaf node actually points to the leaf node of clustered index or to the heap table and hence there is an extra look-up from the Non-Clustered Index to the actual table (leaf node of a clustered index or heap table) to fetch the actual data.

Unique Index. When we create an Index by using the Unique option then it is called Unique Index. Then the column(s) on which the unique index is created will not allow duplicate values i.e. it works as a unique constraint. The Unique Index in SQL Server gives guarantees that the column on which the index is created will not accept any duplicate values.

Advantages of using Indexes - Searching For Records, Sorting Records, Grouping Records, Maintaining a Unique Column

Disadvantages of Indexes- Additional Disk space, Insert, Update and delete statement become Slow

View is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields/columns in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table. view always shows up-to-date data! The database engine recreates the view, every time a user queries it.

```
CREATE VIEW view_name AS
SELECT column1, column2, … FROM table_name WHERE condition;
```

Update- `CREATE OR REPLACE VIEW`

In SQL Server there are two types of authentication i.e. Windows Authentication and SQL Server Authentication.

Different Types of Database in SQL Server.

1. **System databases-**

Master database: This database is used to store all system-level information such as system id, culture, server id no, server version, server culture, etc

Model database: The model database will act as a template for creating new databases under a server environment.

Msdb (Microsoft database): Microsoft database will store jobs and alerts information i.e. backup file information.

Tempdb database: It is a temporary database location that is allocated by the server when the user connects to the SQL Server for storing temporary table information.

2. **User databases** CREATE DATABASE database_name;

Whether we create a database graphically using the designer window or using a query, the following 2 files get generated.

.MDF file: Master Data File (Contains actual data). This file will store all Tables data and will be saved with an extension of .mdf (master data file)

.LDF file:(log data file) Transaction Log file (Used to recover the database). This file will store transaction Query information (insert, update, delete. Create, etc) and saved with an extension of .ldf (log data file)

- Once you create a database, then you can modify the name of the database using the Alter command as shown below. Alter database _DbName Modify Name = _NewDbName
- Alternatively, you can also use the following system-defined stored procedure to change the name. Execute sp_renameDB '_OldDbName', '_NewDbName'
- In order to delete or drop a database in SQL Server, you need to use the following DROP command. Drop Database _DbName

Whenever you drop a database in SQL Server, internally it deletes the LDF and MDF files. You cannot drop a database if it is currently in use and at that time you will get an error stating – Cannot drop database "DatabaseName" because it is currently in use. So, if other users are connected to your database, then first you need to put the database in single-user mode and then drop the database. In order to put the database in single-user mode, you need to use the following command.

Alter Database _DatabaseName Set SINGLE_USER With Rollback Immediate

Note: System databases cannot be dropped.

SQL - It is a non-procedural language that is used to communicate with any database such as Oracle, SQL Server, etc. SQL is also called Sequel. It stands for Structured English Query Language. SQL is not a case-sensitive language; it means that all the commands of SQL are not case-sensitive. Every command of SQL should end with a semicolon (;) (It is optional for SQL Server)

SQL Sub Languages: (subsets)

1. **DDL** (5 commands- drop, create, alter, truncate, sp_rename) {DCAT}. define the structures like schema, database, tables, constraints etc.
2. **DML** (3 commands- Insert, Update, Delete). {DUI}.deals with the manipulation of data present in the database.
3. **DQL/ DRL** (1 command- select).
4. **TCL** (3 commands- commit, rollback, savepoint)
5. **DCL** (2 commands- Grant, Revoke). mainly deal with the rights, permissions, and other controls

Data Definition Language (DDL):

1. Data Definition Language (DDL) is used to define database objects such as tables, synonyms, views, procedures, functions, triggers, etc. that means DDL statements are used to alter/modify a database or table structure and schema
2. DDL commands are working on the structure of a table, not on the data of a table.
3. This language contains five commands. Those are (CREATE, ALTER, SP_RENAME, TRUNCATE, DROP)

CREATE command is used to create a new database object in a database such as tables, views, functions etc. In SQL Server, all database objects (tables, views, etc) are saved with an extension of "dbo.<object name>"

```
CREATE TABLE <TABLE NAME>
(
    <COLUMN NAME1> <DATA TYPE> [SIZE],
    <COLUMN NAME2> <DATA TYPE> [SIZE]
    <COLUMN NAME N><DATA TYPE> [SIZE]
)
```

While creating a database, you need to follow the below rules.

1. The table name should be unique under a database.
2. The column name should be unique within the table definition.
3. A Table name should not start with numeric and special characters except the (_) underscore symbol.
4. Don't provide space in the table name. If you want to provide space in a table name then you can use the underscore symbol.
5. A table name should contain a minimum of 1 character and a maximum of 128 characters.
6. A table should contain a minimum of 1 column and a maximum of 1024 columns.

ALTER command is used to change or modify the structure of a table.

1. Increase/decrease the width of a column.
2. Change the data type of a column.
3. Change the NOT NULL to NULL or NULL to NOT NULL.
4. Used to add a new column to an existing table.
5. Used to drop an existing column.
6. We can add a new constraint.
7. It can drop an existing constraint on a table.
8. Disable or re-enable check constraint of a table.
9. Changing a column name in the table.

This command is used to change a data type from an old data type to a new data type and also to change the size of a data type of a column.

Syntax: `ALTER TABLE <TblNAME> ALTER COLUMN <COLNAME> <NEW DATA TYPE>[NEW SIZE]`

Truncate Command- Whenever you want to delete all the records or rows from a table without any condition, then you need to use the Truncate command. So, using this command you cannot delete specific records from the table because the truncate command does not support the “where” clause. Note: The truncate command will delete rows but not the structure of the table. `TRUNCATE TABLE tableName`

Drop Command If you want to delete the table from the database, then you need to use the DROP command in SQL Server. `DROP TABLE <OBJECT NAME>`

Delete	Truncate
It is a DML command.	It is a DDL command
By using the delete command we can delete a specific record from the table.	But it is not possible with the truncate command.
Delete supports the WHERE clause.	Truncate does not support the WHERE clause
It is a temporary deletion	It is a permanent deletion
Delete supports rollback transactions for restoring the deleted data.	Truncate doesn’t support rollback transaction so that we cannot restore the deleted information
Delete command will not reset identity property.	But it will reset the identity property

Data Types are the attribute that specifies what types of data entered by the user such as integer, character, decimal, date time, etc.

1- Integer data types 2- Decimal data types 3- Money / currency data types
4- Date and Time data types 5- Character data types 6-Binary data types
7- Special data types

Integer Data Types

Data Type	Range	Stored Memory
TinyInt	0-255	1byte
SmallInt	-32768 to 32767	2bytes
Int	$-2 * 10^{-8}$ to $2 * 10^{8}$	4 bytes
BigInt	$-9 * 10^{-8}$ to $9 * 10^{8}$	8 bytes

Date and Time data types

1. Date: This data type will accept date format information only. The default format of the date data type is ‘YYYY/MM/DD’
2. Time: It allows time format information only. The default format of the time data type is ‘hh:mm:ss.ms’

3. DateTime: It allows date and time format information. The default format of DateTime data type is 'YYYY/MM/DD hh:mm: ss.ms'.

Character Data Type

1. **Non Unicode data types: char (Size), varchar (size/max), Text**

Char (size): It is a fixed-length data type (static data type). It will store the data type in the Non-Unicode mechanism that means it will occupy 1 byte for 1 character. The maximum length of the char data type is from 1 to 8000 bytes. Disadvantages: memory wastage because size cannot be changed at runtime.

Varchar (size/max): It is a variable-length data type (dynamic data type) and will store the character in a non-Unicode manner that means it will take 1 byte for 1 character. The maximum length of the varchar data type is from 1 to 8000 bytes.

Text: The text data type is the old version data type of SQL Server and similar to the varchar(max) data type

2. **Unicode data types: nchar(size), nvarchar(size), ntext**

nchar(Size) data type:It is a fixed-length data type and will store the characters in the Unicode manner that means it will take 2 bytes memory per single character. The maximum length of nchar data type is from up to 4000 bytes.

nvarchar(size/max) data type: It is a variable-length data type and will store the data type in the Unicode manner that means it will occupy 2 bytes of memory per single character. The maximum length of nvarchar data type is from up to 4000 bytes.
ntext data type: It is an old version data type of SQL Server and similar to nvarchar(max/size) data type. Here 'n' represents the national.

Binary data type: These data types are used to store image files, audio files, and video files into a database location.Binary data types again classified into three types, such as Binary(size):

1. It is a fixed-length data type and will store binary format information (0,1).
2. The maximum length of the binary data type is from up to 8000 bytes.

Boolean Type:

1. To hold the Boolean values it provides a bit data type that can take a value of 1, 0, or NULL.

Note: The string values TRUE and FALSE can be converted to bit values. TRUE is converted to 1 and FALSE is converted to 0.

Normalization is the process of minimizing redundancy from a relation or set of relations. Redundancy in relation may cause insertion, deletion, and update anomalies. So, it helps to minimize the redundancy in relations.
Normal forms are used to eliminate or reduce redundancy in database tables.

First Normal Form (1NF): This is the most basic level of normalization. In 1NF, each table cell should contain only a single value, and each column should have a unique name. The first normal form helps to eliminate duplicate data and simplify queries.

Second Normal Form (2NF): 2NF eliminates redundant data by requiring that each non-key attribute be dependent on the primary key. This means that each column should be directly related to the primary key, and not to other columns.

Third Normal Form (3NF): 3NF builds on 2NF by requiring that all non-key attributes are independent of each other. This means that each column should be directly related to the primary key, and not to any other columns in the same table.

Boyce-Codd Normal Form (BCNF): BCNF is a stricter form of 3NF that ensures that each determinant in a table is a candidate key. In other words, BCNF ensures that each non-key attribute is dependent only on the candidate key.

Fourth Normal Form (4NF): 4NF is a further refinement of BCNF that ensures that a table does not contain any multi-valued dependencies.

Fifth Normal Form (5NF): 5NF is the highest level of normalization and involves decomposing a table into smaller tables to remove data redundancy and improve data integrity.

Data integrity means the data contained in the database is accurate, consistent, and reliable. To provide data integrity, RDBMS provides us a set of integrity constraints that ensures that the data entered into the database is going to be accurate, consistent, and reliable.

Constraint as a property that can be assigned to a column or columns of a table. The constraints are mainly used to maintain data integrity. Constraints are used to restrict the insertion of unwanted data in any columns. The short-cut key to put NULL is (CTRL + 0).

1. Default Constraint
2. UNIQUE KEY constraint
3. NOT NULL constraint
4. CHECK KEY constraint
5. PRIMARY KEY constraint
6. FOREIGN KEY constraint.

Default constraint is used to fill the column with a default value that is defined during the creation of a table if the user does not supply any value while inserting the data.
Note: IDENTITY and timestamp columns can’t be associated with the default constraint.

NOT NULL Constraint When you want a column not to accept NULL then you need to apply the NOT NULL constraint to that column. That means this constraint is used to avoid NULL values but it accepts duplicate values into a column.

UNIQUE Constraint is used to avoid duplicate values but it accepts a single NULL value in that column. `CREATE TABLE Customer ( Id INT UNIQUE, NAME VARCHAR(30) UNIQUE, Emailid VARCHAR(100) UNIQUE )`

You can find the constraint in the keys folder

Note- The drawback with a **NOT NULL** constraint is it will allow duplicate values whereas in the case of a **UNIQUE** constraint it allows null values.

Creating Constraint with own name:

```
CREATE TABLE customer ( Id INT CONSTRAINT cid_unique UNIQUE, NAME
VARCHAR(30), Emailid VARCHAR(100) CONSTRAINT email_unique UNIQUE )
```

Imposing Constraint in SQL Server: no difference in behavior whether the constraint is imposed at the table level or at the column level

1. Imposing constraints on Column level

```
CREATE TABLE customer (
  Id      INT CONSTRAINT cid_unique UNIQUE,
  NAME    VARCHAR(30),
  Emailid VARCHAR(100) CONSTRAINT email_unique UNIQUE )
```

2. Imposing constraints on Table level

```
CREATE TABLE customer (
   Id      INT,
   NAME    VARCHAR(30),
   Emailid VARCHAR(100),
   CONSTRAINT cid_unique UNIQUE(Id),
   CONSTRAINT email_unique UNIQUE(Emailid)  )
```

Composite Constraints Whenever a constraint is created based on more than one column then it is called Composite Constraints.

```
CREATE TABLE BranchDetails(City VARCHAR(50),
    BranchCode VARCHAR(10),BranchLocation VARCHAR (30),
    CONSTRAINT city_bc_unique UNIQUE(City, BranchCode)  )
```

Check Constraint - Check Constraint is used to enforce domain integrity. Domain integrity ensures that the values going to store in a column must follow some defined rules such as range, type, and format.

```
CREATE TABLE Employee (
    Emp_id       INT NOT NULL CHECK(Emp_id BETWEEN 0 AND 1000),
Emp_name     VARCHAR(30) NOT NULL,
    Entered_date DATETIME NOT NULL CHECK(Entered_date <=
CURRENT_TIMESTAMP),
    Dept_no      INT CHECK(Dept_no > 0 AND Dept_no < 100)  )
```

Primary Key is the combination of Unique and Not Null Constraint. That means it will not allow either NULL or Duplicate values into a column or columns on which the primary key constraint is applied. Using the primary key value we should uniquely identify a record.

The Primary Key constraint can be applied to any data type like integer, character, decimal, money, etc.
Note: The primary key is also called a candidate key.

1. NULLs should not be allowed.
2. It should be unique
3. It can not be modified.

```
CREATE TABLE Branches
(  Bcode INT PRIMARY KEY,  Bname VARCHAR(40), Block  CHAR(40)  )
```

Composite Primary key -A table should contain only 1 Primary Key which can be either on a single or multiple columns i.e. the composite primary key. It is only possible to impose the Composite Primary Key at the table level, it is not possible at the column level.

Note: In a composite primary key, each column can accept duplicate values but the duplicate combination should not be duplicated.

```
CREATE TABLE BranchDetails (
   City  VARCHAR(40),  Bcode INT,  Block  VARCHAR(30),  PRIMARY
KEY(City, Bcode) )
```

Differences between the primary key and unique key

The primary key creates a unique clustered index whereas the unique key creates a unique non-clustered index on the column. Another difference between them is that the primary key column doesn't allow NULL values whereas the unique key column allows only one NULL value.

Foreign Key Constraint: foreign key in one TABLE points to a primary key or unique key in another table. The foreign key constraints are used to enforce referential integrity. Foreign Key constraint is used for binding two tables with each other.

We require two tables for binding with each other and those two tables must have a common column for linking the tables.
The common column that is present in both the tables need not have the same name but their data type must be the same.

The common column that is present under the parent table or master table is known as the reference key column and moreover, the reference key column should not contain any duplicate values. So we need to impose either UNIQUE or PRIMARY key constraints on that column.

```
CREATE TABLE Dept (   Dno    INT PRIMARY KEY,  Dname  VARCHAR(30),
Dloc  CHAR(40) )
GO
CREATE TABLE Employee  ( Eid     INT PRIMARY KEY,  Ename
VARCHAR(30),  Salary  MONEY, Dno   INT FOREIGN KEY REFERENCES
Dept(Dno) )
```

Create Foreign key constraints in SQL Server at the column level?

```
CREATE TABLE Employee (
 Empid  INT,  Ename  VARCHAR(40),  Job VARCHAR(30), Salary MONEY,
    Deptno INT CONSTRAINT deptn0_fk REFERENCES Dept(Dno) )
```

create Foreign key Constraint in SQL Server at table level?

```
CREATE TABLE Employee(
 Empid  INT,  Ename  VARCHAR(40), Job VARCHAR(30),Salary MONEY,
 Deptno INT,
 CONSTRAINT deptno_fk FOREIGN KEY (Deptno) REFERENCES Dept(Dno) )
```

Primary Key:

1. The Primary Key Constraint in SQL Server uniquely identifies a record in the table.
2. Primary Key constraint neither accepts null values nor duplicate values on the column on which it is applied.
3. By default Primary Key Constraint in SQL Server creates a unique clustered index that will physically organize the data in the table.
4. We can create only one Primary Key on a table in SQL Server. The most important point is that you can create the primary key either on a single column or multiple columns.

Foreign Key:

1. The Foreign Key is a field in a table that is a unique key in another table.
2. A Foreign Key can accept both null values and duplicate values.
3. By default, the foreign key does not create any index. If you need then you can create an index on the foreign key column manually.
4. We can create more than one Foreign key on a table in SQL Server.

Note- We can create a table in SQL Server with multiple unique and foreign keys. But it is not possible to create a table can with multiple primary keys in SQL Server.

Is it possible that a foreign key references a non-primary key.- Yes, it is possible. The point that you need to keep in mind is that a foreign key actually references a key that should contain unique values. So it may be a primary key or unique key as both keys maintain the uniqueness of the column of a table.

foreign key in SQL Server can accept NULL values. This is because a Foreign key can reference unique or non-primary keys which may hold NULL values.

Keys in multiple tables

```
CREATE TABLE Orders
(   Odid     INT PRIMARY KEY,   Ordate   DATE,  Quantity INT,
    Cid      INT FOREIGN KEY REFERENCES Customer(Cid),
    Pcode    INT FOREIGN KEY REFERENCES Products(Pcode) )
```

Add constraint to existing tables

Before adding the primary key constraint we need to make it NOT NULL like below

```
ALTER TABLE EMP ALTER COLUMN EMPID INT NOT NULL
```

Now we can add the primary key to the column like below

```
ALTER TABLE EMP ADD CONSTRAINT X PRIMARY KEY (EMPID)
```

Now the EMPID column contains the primary key.

Case2: Adding a unique constraint to an existing column in the table.

```
ALTER TABLE EMP ADD CONSTRAINT Y UNIQUE (ENAME)
```

Case3: Adding CHECK constraint to an existing column.

```
ALTER TABLE EMP ADD CONSTRAINT z CHECK (SALARY > 8000)
```

Case4: Adding a FOREIGN KEY constraint to an existing column.

Let create another table with the name DEP as Below

```
CREATE TABLE DEP(DNO INT, DNAME VARCHAR(30), EID INT)
```

Now we can make the EID column of the DEP table as FOREIGN KEY because the EID column is the primary key in the EMP column.

```
ALTER TABLE DEP ADD CONSTRAINT Q FOREIGN KEY (EID) REFERENCES
EMP(EMPID)
```

How to remove constraints from an existing table?

```
ALTER TABLE<TABLE_NAME> DROP CONSTRAINT<KEY VARIABLE NAME>
```

Example: `ALTER TABLE EMP DROP CONSTRAINT Y`

Note: While dropping the primary key constraint we first need to drop the foreign key and then only we can delete the primary key.

Sequence is an object in SQL Server that is used to generate a number sequence. This can be useful when we need to create a unique number to act as a primary key.

Select Statement is used to return records in the form of a result set from one or more tables or views. Generally, the Select Statement retrieves the data in the form of rows from one or more database tables or views.

Clauses- are used to filtering the records, sorting the records, fetching the records, and grouping the records.
SQL Server supports the following clauses

1. Where (Filtering the records in a table)
2. Order by clause (sorting the records in ascending or descending order)
3. Top n clause (Fetching top n records)
4. Group by clause (Grouping a set of rows)
5. Having Clause (Filtering the data like where clause)

Where clause- this clause is used to extract only those results from a SQL statement (such as SELECT, INSERT, UPDATE, or DELETE statement) that fulfill a specified condition.

- SELECT * FROM Employee WHERE (CITY = 'MUMBAI' AND GenderID = 1) OR (DepartmentID = 3);

- UPDATE Employee SET Salary = 37000 WHERE DepartmentID = 3
- DELETE FROM Employee WHERE CITY = 'MUMBAI'

Order By Clause is used for sorting the data either in ascending or descending order of a query based on a specified column or list of columns. By default, the Order By Clause will sort the data in ascending order.

```
SELECT * FROM Employee ORDER BY Name ASC;
ORDER BY Gender DESC, Name ASC;
```

OFFSET Option- skips record from top
skip the first 5 records in the result set

```
SELECT * FROM Employee ORDER by Name ASC OFFSET 5 ROWS
```

OFFSET as 0 means it will skip 0 records

FETCH Option- after skipping record, it will fetch records
skip 3 records from the top and then fetch 4 records only as part of the result set.

```
SELECT * FROM Employee ORDER by ID ASC OFFSET 3 ROWS FETCH NEXT 4 ROWS ONLY
```

Fetch 0 means it will return 0 records

Top n Clause in SQL Server is used to specify the number of data rows to return. In large tables with thousands or millions of data rows, it takes more time to return all the records, which cause database performance issues. To fix this problem, we can return the specified number of data rows from a table using Top n Clause in SQL Server.

```
SELECT TOP(3)
SELECT TOP (70) PERCENT
```

```
UPDATE TOP (3) Person SET Salary = 72000
DELETE TOP (2) FROM Person
```

Group by Clause is used to divide similar types of records or data as a group and then return. If we use group by clause in the query then we should use grouping/aggregate function such as count(), sum(), max(), min(), and avg() functions.

find total employees in the organization.

```
SELECT COUNT(*) AS TotalEmployee FROM Employee
```

find the number of employees working in each department in the company

```
SELECT Department, COUNT(*) AS TotalEmployee
FROM Employee GROUP BY Department
```

Query to get the highest salary for the organization.

```
SELECT MAX(Salary) as MaxSalary FROM Employee
```

Having Clause is used for filtering the data just like the where clause. Having clause is typically used with a GROUP BY clause. HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

Aggregation is the task of collecting a set of values to return a single value. It is done with the help of aggregate functions, such as SUM, COUNT, and AVG.

Where Clause in SQL Server is used to filter the rows before aggregation, whereas the Having clause in SQL Server is used to filter the groups that mean after aggregations.

following 2 queries produce the same result

Filtering rows using WHERE clause

```
SELECT City, SUM(Salary) as TotalSalary FROM Employee WHERE City = 'MUMBAI' GROUP BY City
```

Filtering groups using the HAVING clause, after all, aggregations take place.

```
SELECT City, SUM(Salary) as TotalSalary FROM Employee GROUP BY City
HAVING City = 'MUMBAI'
```

Difference Between Where Clause and Having Clause in SQL Server

1. WHERE clause cannot be used with aggregate functions whereas HAVING clause can be used with aggregate functions. This means the WHERE clause is used for filtering individual rows on a table whereas the HAVING clause is used to filter groups.
2. WHERE comes before GROUP BY. This means WHERE clause filters rows before aggregate calculations are performed. HAVING comes after GROUP BY. This means HAVING clause filters groups after aggregate calculations are performed. So, from a performance standpoint, HAVING is slower than WHERE and should be avoided when possible.
3. WHERE and HAVING clauses can be used together in a SELECT query. In this case WHERE clause is applied first to filter individual rows. The rows are then grouped and aggregate calculations are performed, and then the HAVING clause filters the groups.

4. WHERE clause can be used with – Select, Insert, and Update statements whereas HAVING clause can only be used with the Select statement.

Operator is a symbol that performs some specific operation on operands or expressions. These operators are classified as follows in SQL Server.
1-**Assignment** operator - used to assign the values to a variable.
DECLARE @MyCounter INT; SET @MyCounter = 1; Compound Assignment Operators - (*=)
2- **Arithmetic** operator- performing mathematical calculations such as Addition, Subtraction, Multiplication, and Division on the given operand values.
3- Comparison operator- used to compare two values
Equal (=) Operator, Not Equal (!= or <>) Operator
Greater Than (>) Operator, Less Than (<) Operator
Greater Than or Equal To (>=), Less Than or Equal To (<=) Operator
Not Greater Than (!<) Operator, Not Less Than (!>) Operator
4- Logical operator- used to test for the truth of some condition
AND – TRUE if both Boolean expressions are TRUE.
OR – TRUE if either Boolean expression is TRUE.
NOT – Reverses the value of any other Boolean operator.
SELECT * FROM Employee WHERE Department = 'IT' OR Department = 'HR'
SELECT * FROM Employee WHERE NOT CITY = 'Mumbai'
Nested logical operator-
SELECT * FROM Employee WHERE Salary >= 27000 AND Salary <= 30000 OR NOT CITY = 'Mumbai'
5- Set operator-

BETWEEN operator in SQL Server is used to get the values within a range. Generally, we use the BETWEEN operator in the WHERE clause to get values within a specified range.
SELECT * FROM Employee WHERE ID BETWEEN 3 AND 7

1. Between Operator returns true if the operand is within a range.
2. Between Operators will return records including the starting and ending values.
3. This operator supports only the AND operator.
4. The BETWEEN Operator takes the values from small to big range in the query.

NOT BETWEEN- return data where the column values not in between the range values.
SELECT * FROM Employee WHERE ID NOT BETWEEN 3 AND 7

IN Operator in SQL Server is used to search for specified values that match any value in the set of multiple values it accepts.
SELECT * FROM Employee WHERE Department IN ('IT', 'HR')

LIKE operator in SQL Server is used to search for character string with the specified pattern.
'%' is a wildcard character
SELECT * FROM Employee WHERE Name LIKE 'P%'

1. % symbol represents any no of characters in the expression.
2. _ will represent a single character in the expression.
3. The [] symbol indicates a set of characters in the expression.
4. [^] will represent any single character, not within the specified range

ANY Operator is used to compare a value to each value in a list of results from a query and evaluate to true if the result of an inner query contains at least one row. ANY must match at least one row in the subquery and must be preceded by comparison operators.

get all the records from the PermanentEmployee table where the Age is at-least greater than one value from the Age column of the ContractEmployee table
SELECT * FROM PermanentEmployee WHERE Age > ANY (SELECT MIN(Age) FROM ContractEmployee)
Greater than ANY means greater than at least one value that is greater than the minimum.
SELECT * FROM PermanentEmployee WHERE Age > ANY (SELECT Age FROM ContractEmployee)

SOME Operator in SQL Server is used to compare a value to each value in a list of results from a query and evaluate to true if the result of an inner query contains at least one row. SOME must match at least one row in the subquery and must be preceded by comparison operators.
greater than (>) with SOME means greater than at least one value
SELECT * FROM PermanentEmployee WHERE Age > SOME (SELECT Age FROM ContractEmployee)

EXISTS operator is used to check the existence of a result of a subquery.
EXISTS operator is used in combination with a subquery and is considered to be met if the subquery returns at least one row. It can be used in a SELECT, INSERT, UPDATE, or DELETE statement.

```
SELECT *
FROM EmployeeDetails
WHERE EXISTS (SELECT *
        FROM EmployeeContactDetails
        WHERE EmployeeDetails.ID = EmployeeContactDetails.EmployeeID);
```

5. **SET Operators** are mainly used to combine the result of more than 1 select statement and return a single result set to the user. There are the following 4 set operators available. They are as follows:

1. **UNION**: Combine two or more result sets into a single set, without duplicates.
2. **UNION ALL**: Combine two or more result sets into a single set, including all duplicates.
3. **INTERSECT**: Takes the data from both result sets which are in common.

4. **EXCEPT**: Takes the data from the first result set, but not in the second result set (i.e. no matching to each other)

Points to remember-
1.The result sets of all queries must have the same number of columns.
2. In every result set the data type of each column must be compatible (well-matched) to the data type of its corresponding column in other result sets.
3. In order to sort the result, an ORDER BY clause should be part of the last select statement. The column names or aliases must be found out by the first select statement

```
SELECT ID, Name, Gender, Department FROM EmployeeIndia
UNION ALL
Select ID, Name, Gender, Department FROM EmployeeUK.
```

UNION removes duplicate rows whereas UNION ALL does not remove the duplicate rows. When we use a UNION operator to remove the duplicate rows from the result set, the SQL server has to do a distinct operation which is time-consuming. For this reason, UNION ALL is much faster than UNION.

ORDER BY clause should be used on the last SELECT statement if we are using UNION or UNION ALL
JOINS and UNIONS are two different things. UNION combines the result-set of two or more select queries into a single result-set which includes all the rows from all the queries in the union, whereas **JOINS**, retrieves data from two or more tables based on logical relationships between the tables. In short, UNION combines rows from 2 or more tables, where JOINS combines columns from 2 or more tables.

EXCEPT operator in SQL Server is used to return unique rows from the left query which isn't present in the right query's results.
EXCEPT operator filters duplicate rows and return only DISTINCT rows from the left query that aren't in the right query's results, whereas NOT IN does not filter the duplicates rows.

```
SELECT ID, Name, Gender, Department FROM EmployeeIndia
EXCEPT
SELECT ID, Name, Gender, Department FROM EmployeeUK
```

You can also use the EXCEPT operator on a single table

```
Select Id, Name, Gender, Salary From Employees Where Salary >= 50000
Except
Select Id, Name, Gender, Salary From Employees Where Salary >= 60000
order By Name
```

INTERSECT operator in SQL Server is used to retrieve the common records of both the left and the right query of the Intersect operator.

```
SELECT ID, Name, Gender, Department FROM TableA
INTERSECT
SELECT ID, Name, Gender, Department FROM TableB
```

INTERSECT Operator filters duplicate rows and returns only the DISTINCT rows that are common between the LEFT and Right Query, whereas INNER JOIN does not filter the duplicates.

Joins are used to retrieve the data from two or more related tables. Tables involved in the joins must have a common field.
SQL Server Joins are classified into two types such as
1- ANSI format JOINS 2- NON-ANSI format JOINS
Again the ANSI format joins classified into three types such as
1- Inner join 2- Outer join 3- Cross join
NON-ANSI join in SQL Server are classified into four types such as
1- EQUI join 2- NON-EQUI join 3- SELF-join 4- Natural Join

INNER JOIN command returns rows that have matching values in both tables.

```
SELECT Orders.OrderID, Customers.CustomerName FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

OUTER JOIN returns matched data rows as well as unmatched data rows from both the tables involved in the join. Outer join is again classified into three types as follows. 1- Left outer join(left join) 2- Right outer join 3- Full outer join

- **LEFT JOIN** keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

```
SELECT column_name(s) FROM table1 LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

- **RIGHT JOIN** keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

```
SELECT column_name(s) FROM table1 RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

- **FULL OUTER JOIN** keyword returns all records when there is a match in left (table1) or right (table2) table records

```
SELECT column_name(s) FROM table1 FULL OUTER JOIN table2
ON table1.column_name = table2.column_name WHERE condition;
```

CROSS JOIN is created by using the CROSS JOIN keyword. CROSS JOIN does not contain an ON clause. In Cross Join, each record of a table is joined with each record of the other table. In SQL Server, the Cross Join should not have either an ON or where clause.

Self Join Joining a table by itself is called self-join in SQL Server. When we have some relation between the columns within the same table then we need to use the self-join mechanism.

CROSS APPLY and OUTER APPLY-

APPLY operators are used to join a table with the output of a table valued function.
CROSS APPLY operator is semantically similar to the INNER JOIN operator. It retrieves those records from the table valued function and the table being joined, where it finds matching rows between the two.
OUTER APPLY retrieves all the records from both the table valued function and the table, irrespective of the match.
Table Valued Function is a function that returns records in the form of a table.

```
CREATE FUNCTION fnGetBooksByAuthorId(@AuthorId int)
 RETURNS TABLE  AS RETURN ( SELECT * FROM Book WHERE author_id =
@AuthorId )
```

Execute SELECT * FROM fnGetBooksByAuthorId(3)

Aggregate function performs operations on a collection of values to return a single scalar value. Aggregate functions are often used with the GROUP BY and HAVING clauses of the SELECT statement. Following are the widely used SQL aggregate functions:

- AVG() - Calculates the mean of a collection of values.
- COUNT() - Counts the total number of records in a specific table or view.
- MIN() - Calculates the minimum of a collection of values.
- MAX() - Calculates the maximum of a collection of values.
- SUM() - Calculates the sum of a collection of values.
- FIRST() - Fetches the first element in a collection of values.
- LAST() - Fetches the last element in a collection of values.

Note: All aggregate functions described above ignore NULL values except for the COUNT function.