# SQL Notes

## 1. What is SSMS?

**Answer:**
SSMS (SQL Server Management Studio) is a Microsoft-developed software application used to manage SQL Server databases. It provides tools to configure, manage, and administer SQL Server instances, write and execute queries, and access database objects.

---

| Aspect | GUI (Graphical User Interface) | CUI (Character User Interface) |
|---|---|---|
| **Full Form** | Graphical User Interface | Character User Interface |
| **Interaction Method** | Users interact using **visual elements** like buttons, icons, windows, and menus. | Users interact by **typing text commands** using a keyboard. |
| **Ease of Use** | More **user-friendly** and intuitive, even for non-technical users. | Requires knowledge of specific commands, hence **less user-friendly**. |
| **Learning Curve** | Easy to learn and use. | Steeper learning curve, especially for beginners. |
| **Speed of Operations** | Generally **slower** due to graphical rendering and mouse use. | Often **faster** for expert users who can type commands quickly. |
| **Resource Usage** | Requires more **memory and processing power** due to graphics. | Uses **less system resources**. |
| **Examples** | Windows OS, macOS, Android, web browsers, MS Word, Photoshop. | MS-DOS, Linux Terminal, Command Prompt (cmd), Bash Shell. |
| **Device Support** | Needs **keyboard, mouse, or touchscreen** to operate. | Only requires a **keyboard** for input. |
| **Multitasking** | Supports **multitasking easily** through multiple windows and tabs. | Limited multitasking; usually one task at a time. |

---

## 2. What services does SSMS provide?

**Answer:**

1. **Database Engine** – Allows executing SQL queries to interact with the database.
2. **SSIS (SQL Server Integration Services)** – Used for data integration between databases.
3. **SSRS (SQL Server Reporting Services)** – Converts data into reports (like exporting Word to PDF).
4. **SSAS (SQL Server Analysis Services)** – Transforms data into analytical visuals like graphs and charts.
5. **Azure Integration** – Enables cloud-based connection and services (added after 2020).

---

## 3. What is platform dependent and platform independent?

**Answer:**

- **Platform Dependent**: Software that runs only on specific OS.
    - *Example:* SQL, C (can't run directly on macOS/Linux).
- **Platform Independent**: Software that runs on multiple OS without changes.
    - *Example:* Java, Python.

| Aspect | Platform Dependent | Platform Independent |
|---|---|---|
| **Definition** | Software that **runs only on the specific operating system or environment** it was developed for. | Software that **can run on multiple operating systems or platforms** without modification. |
| **Code Compatibility** | Code may need to be rewritten or modified to run on a different platform. | Code can be executed across different platforms **without modification**. |

| | | |
|---|---|---|
| **Example Languages** | C, C++, Assembly — because the compiled files (like `.exe`) are OS-specific. | Java — because `.java` code is compiled to `.class` (bytecode) and runs on the JVM across platforms. |
| **Execution Dependency** | Directly depends on system architecture (like Windows/Linux/Mac). | Requires an **interpreter or runtime environment** like JVM or Python interpreter. |
| **Ease of Portability** | Low portability — difficult to run the same code on different platforms. | High portability — easy to run the same code across systems. |
| **Compilation Target** | Compiles to **machine code specific** to the host operating system. | Compiles to **intermediate code** (e.g., bytecode) that can run anywhere with the required runtime. |
| **Example** | A C program compiled on Windows produces `.exe` that **won't run on Linux**. | A Java program compiled to bytecode runs on **any OS** with a compatible JVM. |

---

## 4. What are DDL functions with syntax?

**Answer:**
DDL (Data Definition Language) modifies the structure of database objects.

1. **CREATE**: Creates table
   `CREATE TABLE Employees (ID INT, Name VARCHAR(50));`

2. **ALTER**: Modifies table

    - Add column: `ALTER TABLE Employees ADD Age INT;`

- Change datatype: `ALTER TABLE Employees ALTER COLUMN Age SMALLINT;`
- Drop column: `ALTER TABLE Employees DROP COLUMN Age;`

3. **SP_RENAME**: Renames column
   `SP_RENAME 'Employees.OldName', 'NewName';`

4. **DROP**: Deletes entire table
   `DROP TABLE Employees;`

5. **TRUNCATE**: Deletes all rows (cannot be rolled back)
   `TRUNCATE TABLE Employees;`

```
create----->which is used to create a table inside the database
syntax=> create table <table name> (<column name><datatype>,...)

alter------>3 DIFF TECHNIQUES

            1.using alter we can change the data type of column
syntax=> alter table <table name> alter column <column name> <new
data type>

          2.USING ALTER WE CAN ADD NEW COLUMN
syntax=> alter table <table name> add <column name> <datatype>

            3.Using alter we can remove a column also
syntax=> alter table <table name> drop column <column name>

sp_rename-->USEDt change column name
syntax=>sp_rename '<tn>.<old cn>','<new cn>'

drop------->WILL DELETE TABLE FROM DATABASE
SYNTAX = DROP TABLE <TN>

truncate--->IS USED TO REMOVE ALL THE ROWS FROM THE TABLE

            SINGLE ROW IS NOT POSSIBLE TO REMOVE FROM THE TABLE BY
TRUNCATE

            SYNTAX: TRUNCATE TABLE <TN>
```

---

## 5. What are DML functions with syntax?

**Answer:**
DML (Data Manipulation Language) is used for managing data in tables.

1. **INSERT**: Adds data
   ```
   INSERT INTO Employees VALUES (1, 'John');
   ```

2. **UPDATE**: Modifies data
   ```
   UPDATE Employees SET Name = 'Alice' WHERE ID = 1;
   ```

3. **DELETE**: Removes data
   ```
   DELETE FROM Employees WHERE ID = 1;
   ```

---

## 6. What is Auto Seed Increment Concept?

**Answer:**
It allows automatic generation of unique numbers in a column (usually primary key).

**The purpose of auto seed increment is to automatically generate unique values for a specific column (usually a Primary Key) in a sequential manner.**
**It helps in automating the generation of IDs when new rows are inserted, avoiding manual input and ensuring uniqueness.**

---

## ✅ What is the default value of the seed?

**The default seed value is 1.**
**This means that the first inserted row will start at 1, and each subsequent row will increment by 1 (unless specified otherwise).**

---

## ✅ Can we customize the seed value?

**Yes, the seed value can be customized.**

**You can specify both the starting seed and the increment value when defining the column.**

**—----------**

**Auto Increment** is a property in SQL that **automatically generates a unique sequential number** every time a new row is inserted into a table. It is typically used to **automatically assign values to a column**, especially **primary key** columns.

**AUTO(SEED/INCREMENT) CONCEPT**

---

RULES:

---

1. AUTO (SEED / INCREMENT) CONCEPT WILL BE USED ONLY ON NUMERICAL COLUMNS

2. THE DEFAULT SEED VALUE WILL START FROM 1.

3. IF WE WANT TO SELECT A STARTING_VALUE IT IS POSSIBLE

4. WHILE CREATING THE TABLE ONLY WE NEED TO SELECT THE RANGE

## 🔹 Key Features of Auto Increment:

1. **Automatically generates unique numbers** for each new record.
2. It is **commonly used with primary key columns** to ensure each row has a unique identifier.
3. It **starts from a defined initial value**, with the default being 1.
4. It **automatically increments by 1** (or another defined step value) for every new row inserted into the table.

## Example in MySQL:

```
CREATE TABLE Employees (
    EmpID INT AUTO_INCREMENT,
    Name VARCHAR(50),
    PRIMARY KEY (EmpID)
);
```

```
CREATE TABLE Customer (
  CustID INT IDENTITY,
  CustName CHAR(10),
  ProdName CHAR(10),
  Price FLOAT
);
```

On inserting:

```
INSERT INTO Customer VALUES ('Ajay', 'Sugar', 1000);
```

It auto-generates `CustID` as 1, 2, 3, ...

---

## 7. What are Set Operators in SQL?

**Answer:**
Set operators combine results from two SELECT queries:

- **UNION**: Returns unique records from both queries.
- **UNION ALL**: Returns all records including duplicates.
- **INTERSECT**: Returns common records.
- **EXCEPT**: Returns records from first query not in second.

*Example:*

```
SELECT Name FROM TableA
UNION
SELECT Name FROM TableB;
```

RULES:
---
1. WE NEED 2 TABLES
2. IN BOTH THE TABLES WE NEED SAME NUMBER OF COLUMNS AND THERE DATA TYPES SHOULD MATCH
3. TO UNDERSTAND THE SET OPERATORS PLEASE MAINTAIN SOME COMMON ROWS

FUNCTIONS IN THE SET OPERATORS

WE HAVE 4 FUNCTIONS IN THE SET OPERATORS

1. UNION = WILL GIVE NUMBER OF EMPLOYESS PRESENT IN ALL THE TABLES WITHOUT DUPLICATES

SYNTAX: SELECT *FROM <TN1>
UNION
SELECT *FROM <TN2>

2. UNION_ALL = GIVE TOTAL DATA OF THE EMPLOYESS
3. INTERSECT = COMMON DATA BETWEEN THE TABLES
4. EXCEPT = IS USED TO GENERATE UNCOMMON DATA FROM LEFT SIDE TABLE

---

## 8. What is Primary Key and Foreign Key?

**Answer:**

| Primary Key | UNIQUE Key |
|---|---|
| Uniquely identifies each row in a table | Ensures all values in the column are unique |
| Does not allow NULL values | Allows one NULL value (in most databases) |
| Only one primary key is allowed per table | Multiple UNIQUE keys can be defined in a table |
| Automatically creates a unique clustered index | Creates a unique non-clustered index |
| Used to uniquely identify a record | Used to enforce uniqueness in a column without being a primary identifier |

---

- **Primary Key**: Uniquely identifies each record in a table.
- **Foreign Key**: Links one table to another.

**Rules**:

- Parent table has Primary Key.

- Child table references that column via Foreign Key.
- 1. To implement primary key and forgein key concept we need
- two tables
- 2. consider one table like parent and second table like child
- 3. In the parent table Maintain a column as a primay key
- 4. In the child table please maintain same column name and maintain as
- forgein key
- 5. Please Take the reference from forgein key to primary key
- 6. In the parent table which column is following primary key permission
- it should not be null

**Example**:

```
CREATE TABLE Department (
  DeptID INT PRIMARY KEY,
  DeptName VARCHAR(50)
);

CREATE TABLE Employee (
  EmpID INT,
  EmpName VARCHAR(50),
  DeptID INT FOREIGN KEY REFERENCES Department(DeptID)
);
```

---

## 9. What are Data Types in SQL?

**Answer:**

| Data Type | Purpose | Example |
|---|---|---|
| INT | Whole numbers | Age, ID |
| TINYINT | Small whole numbers | StdAge |
| CHAR(n) | Fixed-length strings | Name CHAR(10) |
| VARCHAR(n) | Variable-length strings | Email VARCHAR(100) |
| FLOAT | Decimal numbers | Price |
| MONEY | Currency values | Salary |
| DATE/TIME | Date/time data | DOB, DOJ |

---

## 10. What is TCL (Transaction Control Language)?

**Answer:**
Used to manage database transactions and control changes.

- **BEGIN TRANSACTION**: Starts a transaction block.
- **ROLLBACK**: Reverses changes since last `BEGIN`.

-IS A FUNCTION WHICH IS USED TO UNDO THE OPERATIONS

- WITHOUT TCL OPERATION IF WE WANT TO UNDO THE ACTION IT IS NOT POSSIBLE BECAUSE SQL IS GOING TO FIX THE ACTION IN THE DATABASE

- **COMMIT**: Saves all changes made during transaction.

**Example:**

```
BEGIN TRANSACTION;
DELETE FROM Employees WHERE ID = 5;
ROLLBACK; -- Restores deleted data
```

==========

| Command | Function | Can Rollback? | Affects Structure? | Speed |
|---|---|---|---|---|
| **DELETE** | The DELETE command removes specific rows from a table based on a condition using the `WHERE` clause. | Yes, DELETE can be rolled back if used with `COMMIT` or `ROLLBACK`. | No, it does not affect the structure of the table. | It is slower compared to other commands as it performs row-by-row deletion. |
| **TRUNCATE** | The TRUNCATE command removes all rows from a table without using a condition. | No, TRUNCATE cannot be rolled back. | No, it does not affect the table structure. | It is faster than DELETE. |
| **DROP** | The DROP command deletes the entire table | No, DROP cannot be rolled back. | Yes, it removes the entire table | It is the fastest among DELETE, TRUNCATE, and DROP. |

| | | |
|---|---|---|
| | including both data and structure. | structure from the database. |

---

| Clause | Purpose | Used With | Filter Type | Execution Order |
|---|---|---|---|---|
| **WHERE** | The `WHERE` clause filters individual rows from the result set before any grouping is performed. | It is used with `SELECT`, `UPDATE`, and `DELETE` statements. | It performs **row-level filtering**. | The `WHERE` clause is executed **before** the `GROUP BY` clause. |
| **HAVING** | The `HAVING` clause filters groups of rows after the `GROUP BY` clause has been applied. | It is used with `SELECT` statements that include `GROUP BY`. | It performs **group-level filtering**. | The `HAVING` clause is executed **after** the `GROUP BY` clause. |

```
-- Using WHERE to filter rows before grouping
SELECT department, COUNT(*)
FROM employees
WHERE salary > 50000
GROUP BY department;

-- Using HAVING to filter groups after grouping
SELECT department, COUNT(*)
FROM employees
GROUP BY department
HAVING COUNT(*) > 5;
```

---

| Criteria | CHAR | VARCHAR |
|---|---|---|
| **Full Form** | `CHAR` stands for **Character**. | `VARCHAR` stands for **Variable Character**. |
| **Storage** | It uses **fixed-length** storage, regardless of the actual data size. | It uses **variable-length** storage depending on the length of the actual data stored. |
| **Memory Usage** | `CHAR` always uses the defined length even if fewer characters are stored. | `VARCHAR` uses only the space required for the data plus **1 or 2 extra bytes** to store the length. |

| | | |
|---|---|---|
| **Performance** | `CHAR` is **faster** when dealing with fixed-size data due to consistent size. | `VARCHAR` is **slower** when data size varies, due to additional overhead in managing variable length. |
| **Padding** | It **pads extra spaces** to match the fixed length defined. | It **does not pad spaces**; only actual characters are stored. |
| **Use Case** | Best used when data length is **consistent**, like PIN codes or gender. | Best used when data length is **inconsistent or varies**, such as names or addresses. |

---

| UNION | UNION ALL |
|---|---|
| Removes duplicate rows | Includes duplicate rows |
| Slower due to duplicate removal | Faster as no duplicate removal |
| Automatically sorts the result set | Does not sort the result set |
| Syntax: UNION | Syntax: UNION ALL |

## Example (UNION):

```
SELECT Name FROM Employees
UNION
SELECT Name FROM Managers;
```

This query combines employee and manager names without duplicates.

## Example (UNION ALL):

```
SELECT Name FROM Employees
UNION ALL
SELECT Name FROM Managers;
```

This query combines employee and manager names, including duplicates.

---

| Criteria | UNION | UNION ALL |
|---|---|---|
| Duplicates | Removes duplicate rows | Includes all duplicate rows |
| Performance | Slower (due to duplicate removal) | Faster (no duplicate removal) |
| Sorting | Automatically sorts the result set | Does not sort the result set |
| Usage | Used when duplicate data is not required | Used when duplicate data needs to be preserved |
| Syntax | SELECT column FROM table1 UNION SELECT column FROM table2; | SELECT column FROM table1 UNION ALL SELECT column FROM table2; |

| GROUP BY | ORDER BY |
|---|---|
| Used to group rows based on the same values in one or more columns. | Used to sort the result set in ascending or descending order. |
| Always works with aggregate functions like COUNT(), SUM(), AVG(), etc. | Does not require aggregate functions. |
| Syntax: Comes before ORDER BY. | Syntax: Always comes after GROUP BY. |
| Groups the result into summary rows. | Sorts the entire result set. |
| Example: Group sales by product category. | Example: Sort sales by highest to lowest amount. |

---

| Clustered Index | Non-Clustered Index |
| --- | --- |
| Stores data physically sorted. | Stores pointers to data. |
| Only one per table. | Multiple indexes allowed. |
| Faster for data retrieval. | Slower than Clustered Index. |
| Automatically created on Primary Key. | Manually created on any column. |
| Affects physical order of table. | Does not affect physical order. |

| RANK() | DENSE_RANK() |
| --- | --- |
| Assigns a unique rank to each row, but skips the next rank if there are duplicate values. | Assigns a unique rank to each row without skipping ranks if there are duplicate values. |
| Gaps are created in ranking sequence. | No gaps in ranking sequence. |
| Slower in performance compared to DENSE_RANK(). | Faster than RANK() because it doesn't skip ranks. |

| DELETE | TRUNCATE |
| --- | --- |
| Removes specific rows based on a condition using the WHERE clause. | Removes all rows from the table without any condition. |
| Can be rolled back using ROLLBACK if inside a transaction. | Cannot be rolled back once executed. |
| Slower because it logs each row deletion. | Faster because it does not log individual row deletions. |
| Maintains table structure and identity column values. | Resets identity column values to the initial seed. |

| View | Table |
| --- | --- |
| Virtual table. | Physical table. |
| Does not store data physically. | Stores data physically. |
| Based on SQL queries. | Contains raw data. |
| Provides data security by restricting access to certain columns. | No data restriction unless applied. |
| Automatically updates when base table data changes. | Needs manual updates. |

| Aspect | ROW_NUMBER() | RANK() | DENSE_RANK() |
| --- | --- | --- | --- |

| | | | |
|---|---|---|---|
| **Basic Definition** | Assigns a **unique sequential number** to every row. | Assigns the **same rank to duplicate values**, and skips the next rank(s). | Assigns the **same rank to duplicate values**, but does **not skip** any ranks. |
| **Duplicate Handling** | Does **not** assign the same number to duplicate values. | Assigns the **same rank** to duplicate values. | Assigns the **same rank** to duplicate values. |
| **Number Skipping** | Does **not skip** any number; numbers are always in sequence. | **Skips numbers** after duplicate ranks. | Does **not skip** numbers after duplicates. |
| **Output Example** | `1, 2, 3, 4` (even if values are repeated). | `1, 2, 2, 4` (rank 3 is skipped due to duplicates). | `1, 2, 2, 3` (no skipped ranks). |
| **Use Case** | Best when each row must have a **unique identity** (e.g., pagination). | Useful when **ranking with positional gaps** is required (e.g., sports rankings). | Ideal for **continuous ranking** without gaps (e.g., grade ranks). |
| **Ranking Based On** | Purely **row position** in the order, not value duplicates. | Based on **value**, identical values get same rank; position affects gaps. | Based on **value**, identical values get same rank; no position-based gaps. |

---

**#DQL [DATA QUERY LANGUAGE] OPERATION**

---

1.**SELECTION METHOD ->>**Selection is the process of retrieving rows (records) from a table that meet specific conditions.
2.**PROJECTION METHODS->>**Projection is the process of retrieving specific columns from a table.
3.**JOINS**

JOINS = IS USED TO WORK WITH MORE THAT ONE TABLE
THERE ARE 2 TYPES OF JOINS

1. **ANSI JOINS** = ansi join are going to work with on condition
in ansi joins there are 3 types
A) INNER JOIN = (COMMON DATA WILL GIVE)
INNER JOIN FUNCTION IS USED TO GET COMMON ROWS FROM BOTH TABLES BASED ON THE GIVEN CONDITION
SYNTAX: SELECT *FROM <TN1> <FUNCTION_NAME> <TN2> ON <TN1>.<CN>=<TN2.CN>

B) OUTER JOIN

I) LEFT OUTER JOIN = THIS FUNCTION WILL GIVE ALL DATA FROM LEFT SIDE TABLE BUT ONLY MATCHING DATA FROM
RIGHT SIDE TABLE
SYNTAX: SELECT *FROM <TN1> <FUNCTION_NAME> <TN2> ON <TN1>.<CN>=<TN2.CN>

II) RIGHT OUTER JOIN = THIS FUNCTION WILL GIVE ALL DATA FROM RIGHT SIDE TABLE BUT ONLY MATCHING DATA FROM
LEFT SIDE TABLE
SYNTAX: SELECT *FROM <TN1> <FUNCTION_NAME> <TN2> ON <TN1>.<CN>=<TN2.CN>

III) FULL OUTER JOIN = THIS FUNCTION IS THE COMBINATION OF LEFT OUTER JOIN AND RIGHT OUTER JOIN
SYNTAX: SELECT *FROM <TN1> <FUNCTION_NAME> <TN2> ON <TN1>.<CN>=<TN2.CN>

C) CROSS JOIN = THIS FUNCTION IS USED 1 DATA POINTS WITH ALL DATA POINTS CONCEPT
SYNTAX: SELECT *FROM <TN1> CROSS JOIN <TN2>

2. **NON-ANSI JOINS** = NON-ANSI JOINS ARE GOING TO WORK WITH WHERE CONDITION

IN NON-ANSI JOINS THERE ARE 2 TYPES

A) EQUI JOIN = THIS IS EXACTLY LIKE INNER JOIN ONLY WHICH MEANS IT IS GOING TO GET COMMON ROWS
BUT USING WHERE CONDITION | BUT WE SHOULD NOT USE OPERATORS [>,<,!=,==,]
USING WHERE FUNCTION AND = [COMMON DATA]
SELECT *FROM <TN1>,<TN2> WHERE <TN1>.<CN> = <TN2>.<CN>

B) NON-EQUI JOIN = USING WHERE FUNCTION AND OPERATORS LIKE >,<,>=,<=
SELECT *FROM <TN1>,<TN2> WHERE <TN1>.<CN> > <TN2>.<CN>

---

RULES TO IMPLEMENT JOINS CONCEPT:

---

1. WE NEED TO HAVE 2 TABLES AND SAME NUMBER OF COLUMNS AND COLUMN DATA TYPE SHOULD MATCH
2. MAINTAIN SOME COMMON INPUTS TO UNDERSTAND THE CONCEPT CLEARLLY

---

---

Raking Concept:-
---
RANKING CONCEPT IS USED TO GIVE THE RANKS FOR EVERY RECORDS
-IN REAL TIME
-ROW-> RECORD OR OBSERVATION
-COLUMN->FEATURES OR VARIABLES
RANKING FOR THE RECORD WILL DONE IN 2 DIFF WAYS

-IN SQL RANKING CAN BE GIVEN IN 3 WAYS MEANS USING 3 FUNCTIONS:-
1.ROW_NUMBER()
2.RANK()
3.DENSE_RANK()

1. ROW WISE RANKING ->>
1.CREATE TABLE
2.FILL THE TABLE WITH FEW ROWS
3.WE CAN ASSIGN THE RANKS BASED ON NUMERICAL COLUMN ONLY

2. GROUP WISE RANKING->>

---

| Aspect | `ON` Clause | `WHERE` Clause |
|---|---|---|
| **Purpose** | Used to specify the **join condition** between two tables. | Used to **filter rows** after the join or from a single table. |
| **Used In** | Mainly used with **JOIN** operations (INNER, LEFT, etc.). | Used in **SELECT, UPDATE, DELETE** statements. |
| **Executes When** | Executes **during the join** process. | Executes **after** the join is complete. |
| **Affects Join Result** | Yes, it decides **which rows to join**. | No, it **filters the result** of the join. |
| **Can Use Aliases?** | Yes | Yes |

---

| Aspect | `OVER` Keyword | `PARTITION BY` Keyword |
|---|---|---|
| **Definition** | Used to define a window or set of rows for a **window function** like `ROW_NUMBER`, `RANK`, `SUM`, etc. | Used with `OVER` to divide the result set into **partitions (groups)** for function calculation. |
| **Purpose** | Applies the function over the full dataset or within partitions. | Groups rows into partitions so the function (like SUM, AVG) resets per group. |
| **Used With** | Always used with window functions. | Used inside the `OVER` clause. |
| **Scope** | Can apply across the entire dataset or partitioned sets. | Limits the scope of window function to each partition/group. |
| **Can Be Used Alone?** | Yes, `OVER()` can be used without `PARTITION BY` (applies to entire result). | No, `PARTITION BY` must be used inside an `OVER()` clause. |

---

| PRIMARY KEY | FOREIGN KEY |
|---|---|
| A primary key is used to ensure data in the specific column is unique. | A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables. |
| It uniquely identifies a record in the relational database table. | It refers to the field in a table which is the primary key of another table. |
| Only one primary key is allowed in a table. | Whereas more than one foreign key is allowed in a table. |
| It is a combination of UNIQUE and Not Null constraints. | It can contain duplicate values and a table in a relational database. |
| It does not allow NULL values . | It can also contain NULL values. |
| Its value cannot be deleted from the parent table. | Its value can be deleted from the child table. |
| It constraint can be implicitly defined on the temporary tables. | It constraint cannot be defined on the local or global temporary tables. |

| Primary key | Foreign key |
| --- | --- |
| It must contain unique values. | It can contain duplicate values. |
| It cannot contain null values. | It can contain null values. |
| A database can have only one primary key. | A database can have more than one foreign key. |
| It is used to identify the records in a table uniquely. | It is used to make a relation between two tables. |

---