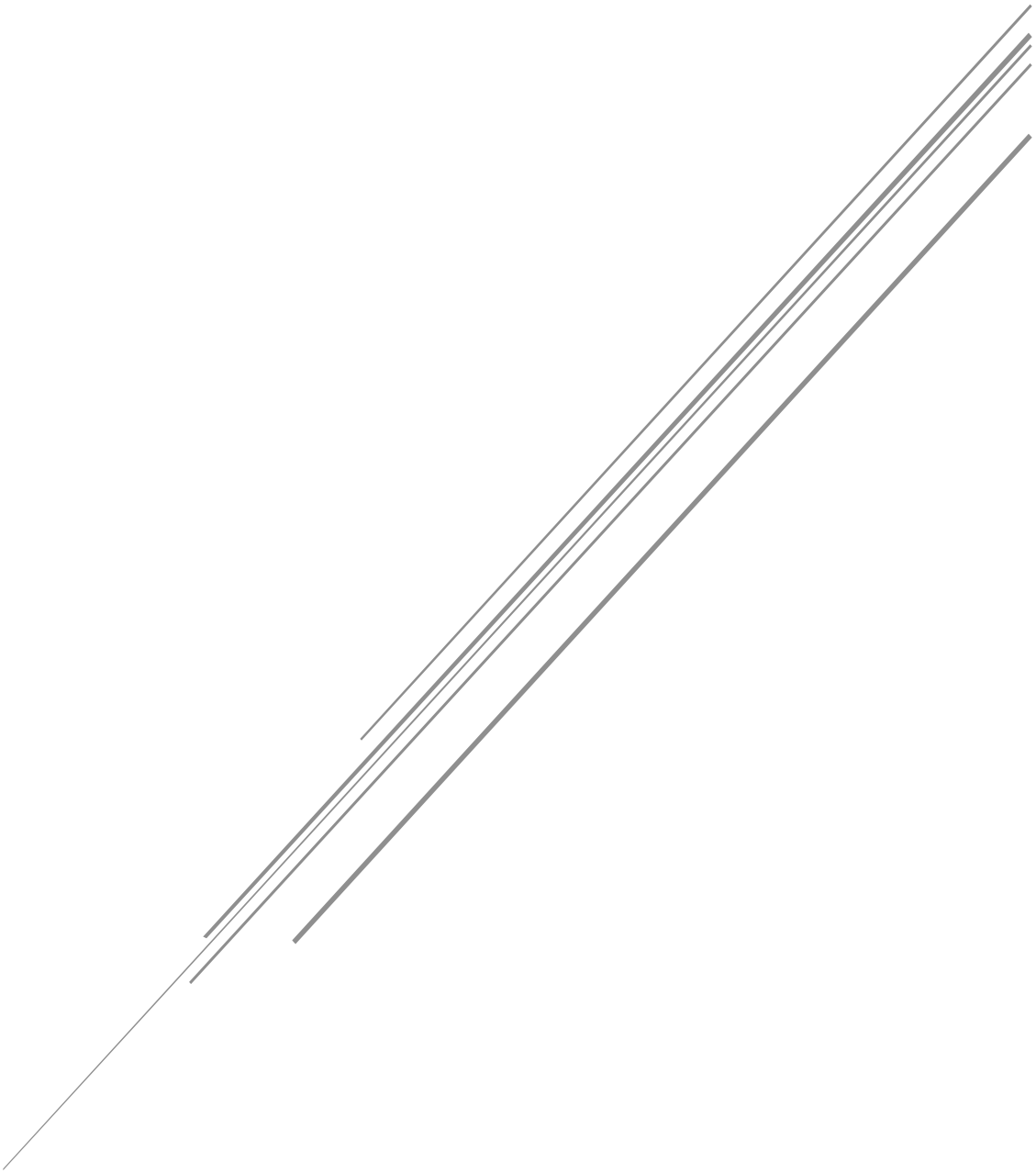
[SQL]

[Beginner's Guide]



Author: Non-Tech to TECH

OVERVIEW

SQL is a standard language for storing, manipulating and retrieving data in databases.

SQL is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS). It is particularly useful in handling structured data, i.e. data incorporating relations among entities and variables.

PROCEDURAL EXTENSIONS

SQL is designed for a specific purpose: to query data contained in a relational database. SQL is a set-based, declarative programming language, not an imperative programming language like C or BASIC. However, extensions to Standard SQL add procedural programming language functionality, such as control-of-flow constructs. These include:

Source	Abbreviation	Full name
ANSI/ISO Standard	SQL/PSM	SQL/Persistent Stored Modules
Interbase / Firebird	PSQL	Procedural SQL
IBM Db2	SQL PL	SQL Procedural Language (implements SQL/PSM)
IBM Informix	SPL	Stored Procedural Language
IBM Netezza	NZPLSQL[18]	(based on Postgres PL/pgSQL)
Invantive	PSQL[19]	Invantive Procedural SQL (implements SQL/PSM and PL/SQL)
MariaDB	SQL/PSM, PL/SQL	SQL/Persistent Stored Module (implements SQL/PSM), Procedural Language/SQL (based on Ada)
Microsoft / Sybase	T-SQL	Transact-SQL
Mimer SQL	SQL/PSM	SQL/Persistent Stored Module (implements SQL/PSM)
MySQL	SQL/PSM	SQL/Persistent Stored Module (implements SQL/PSM)
MonetDB	SQL/PSM	SQL/Persistent Stored Module (implements SQL/PSM)
NuoDB	SSP	Starkey Stored Procedures
Oracle	PL/SQL	Procedural Language/SQL (based on Ada)
PostgreSQL	PL/pgSQL	Procedural Language/PostgreSQL Structured Query Language (based on reduced PL/SQL)
SAP R/3	ABAP	Advanced Business Application Programming
SAP HANA	SQLScript	SQLScript
Sybase	Watcom-SQL	SQL Anywhere Watcom-SQL Dialect
Teradata	SPL	Stored Procedural Language

INCOMPATIBILITY

SQL implementations are incompatible between vendors and do not necessarily completely follow standards. In particular, date and time syntax, string concatenation, NULLs, and comparison case sensitivity vary from vendor to vendor. Particular exceptions are PostgreSQL and Mimer SQL which strive for standards compliance, though PostgreSQL does not adhere to the standard in all cases. For example, the folding of unquoted names to lower case in PostgreSQL is incompatible with the SQL standard,[24] which says that unquoted names should be folded to upper case.[25] Thus, Foo should be equivalent to FOO not foo according to the standard.

Popular implementations of SQL commonly omit support for basic features of Standard SQL, such as the DATE or TIME data types. The most obvious such examples, and incidentally the most popular commercial and proprietary SQL DBMSs, are Oracle (whose DATE behaves as DATETIME,[26][27] and lacks a TIME type)[28] and MS SQL Server (before the 2008 version). As a result, SQL code can rarely be ported between database systems without modifications.

Several reasons for this lack of portability between database systems include:

- The complexity and size of the SQL standard means that most implementers do not support the entire standard.
- The standard does not specify database behavior in several important areas (e.g. indices, file storage...), leaving implementations to decide how to behave.
- The SQL standard precisely specifies the syntax that a conforming database system must implement. However, the standard's specification of the semantics of language constructs is less well-defined, leading to ambiguity.
- Many database vendors have large existing customer bases; where the newer version of the SQL standard conflicts with the prior behavior of the vendor's database, the vendor may be unwilling to break backward compatibility.
- Little commercial incentive exists for vendors to make changing database suppliers easier (see vendor lock-in).
- Users evaluating database software tend to place other factors such as performance higher in their priorities than standards conformance.

LANGUAGE ELEMENTS

The SQL language is subdivided into several language elements, including:

- Keywords are words that are defined in the SQL language. They are either reserved (e.g. SELECT, COUNT and YEAR), or non-reserved (e.g. ASC, DOMAIN and KEY). List of SQL reserved words.
- Clauses, which are constituent components of statements and queries. (In some cases, these are optional.
- Expressions, which can produce either scalar values, or tables consisting of columns and rows of data
- Predicates, which specify conditions that can be evaluated to SQL three-valued logic (3VL) (true/false/unknown) or Boolean truth values and are used to limit the effects of statements and queries, or to change program flow.
- Queries, which retrieve the data based on specific criteria. This is an important element of SQL.
- SQL statements also include the semicolon (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.
- Insignificant whitespace is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.

Operator	Description	Example
=	Equal to	Author = 'Alcott'
<>	Not equal to (many DBMSs accept != in addition to <>)	Dept <> 'Sales'
>	Greater than	Hire_Date > '2012-01-31'
<	Less than	Bonus < 50000.00
>=	Greater than or equal	Dependents >= 2
<=	Less than or equal	Rate <= 0.05
[NOT] BETWEEN [SYMMETRIC]	Between an inclusive range. SYMMETRIC inverts the range bounds if the first is higher than the second.	Cost **BETWEEN** 100.00 **AND** 500.00
[NOT] LIKE [ESCAPE]	Begins with a character pattern	Full_Name **LIKE** 'Will%'
	Contains a character pattern	Full_Name **LIKE** '%Will%'
[NOT] IN	Equal to one of multiple possible values	DeptCode **IN** (101, 103, 209)
IS [NOT] NULL	Compare to null (missing data)	Address **IS NOT NULL**
IS [NOT] TRUE *or* IS [NOT] FALSE	Boolean truth value test	PaidVacation **IS TRUE**
IS NOT DISTINCT FROM	Is equal to value or both are nulls (missing data)	Debt **IS NOT DISTINCT FROM** - Receivables
AS	Used to change a column name when viewing results	**SELECT** employee **AS** department1

COMMENTS

Standard SQL allows two formats for comments: -- comment, which is ended by the first newline, and /* comment */, which can span multiple lines.

SQL SYNTAX

1. The SQL SELECT Statement:

The SELECT statement is used to select data from a database. The data returned is stored in a result table, called the result-set.

- `SELECT * FROM table_name;` → *If you want to select all the fields available in the table*
- `SELECT column1, column2, ...`
  `FROM table_name;` → *Column1, column2, ... are the field names of the table you want to select data from*

2. SQL SELECT DISTINCT Statement:

The SELECT DISTINCT statement is used to return only distinct (different) values.
Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

- `SELECT DISTINCT column1, column2, ...`
  `FROM table_name;`
- `SELECT DISTINCT Country FROM Customers;` → *Distinct values ignoring duplicate entries*

3. SQL WHERE Clause:

The WHERE clause is used to filter records. It is used to extract only those records that fulfill a specified condition.

- `SELECT column1, column2, ...`
  `FROM table_name`
  `WHERE condition;`
- `SELECT * FROM Customers`
  `WHERE Country='India';` → *Select fields where country mentioned as India*

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as!=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

4. SQL AND, OR and NOT Operators:

The WHERE clause can be combined with AND, OR, and NOT operators.
The AND / OR operators are used to filter records based on more than one condition:
The AND operator displays a record if all the conditions separated by AND are TRUE.
The OR operator displays a record if any of the conditions separated by OR is TRUE.
The NOT operator displays a record if the condition(s) is NOT TRUE.

- ```
  SELECT column1, column2, ...
  FROM table_name
  WHERE condition1 AND condition2 AND condition3 ...;
  ```
  → *AND conditions*
- ```
  SELECT column1, column2, ...
  FROM table_name
  WHERE condition1 OR condition2 OR condition3 ...;
  ```
  → *OR conditions*
- ```
  SELECT column1, column2, ...
  FROM table_name
  WHERE NOT condition;
  ```
  → *NOT conditions*

5. SQL ORDER BY:

The ORDER BY keyword is used to sort the result-set in ascending or descending order. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

- ```
  SELECT column1, column2, ...
  FROM table_name
  ORDER BY column1, column2, ... ASC|DESC;
  ```

6. SQL INSERT INTO:

The INSERT INTO statement is used to insert new records in a table.

- ```
  INSERT INTO table_name (column1, column2, column3, ...)
  VALUES (value1, value2, value3, ...);
  ```
  → *Specific Table & columns*
- ```
  INSERT INTO table_name
  VALUES (value1, value2, value3, ...);
  ```
  → *Insert to all columns in specific table*

7. SQL NULL:

A field with a NULL value is a field with no value. If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

You have to use the IS NULL and IS NOT NULL operators instead.

- ```
  SELECT column_names
  FROM table_name
  WHERE column_name IS NULL;
  ```
  → *Null finding*
- ```
  SELECT column_names
  FROM table_name
  WHERE column_name IS NOT NULL;
  ```
  → *Not Null*

8. SQL UPDATE:

The UPDATE statement is used to modify the existing records in a table.

Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

- ```
  UPDATE table_name
  SET column1 = value1, column2 = value2, ...
  WHERE condition;
  ```
  → *Set – Values to be update, Where – where to update*

9. SQL DELETE:

The DELETE statement is used to delete existing records in a table.

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

- ```
  DELETE FROM table_name WHERE condition;
  ```

10. SQL TOP, LIMIT, FETCH FIRST or ROWNUM Clause:

The SELECT TOP clause is used to specify the number of records to return. The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

Note: Not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses FETCH FIRST n ROWS ONLY and ROWNUM.

- ```
  SELECT TOP number|percent column_name(s)
  FROM table_name
  WHERE condition;
  ```
  → *SQL Server / MS Access Syntax*
- ```
  SELECT column_name(s)
  FROM table_name
  WHERE condition
  LIMIT number;
  ```
  → *MySQL Syntax*

11. SQL MIN() and MAX():

The MIN() function returns the smallest value of the selected column. The MAX() function returns the largest value of the selected column.

- ```
  SELECT MIN(column_name)
  FROM table_name
  WHERE condition;
  ```
  → *Replace Min with Max to get MAX values*

12. SQL COUNT(), AVG() and SUM():

The COUNT() function returns the number of rows that matches a specified criterion. The AVG() function returns the average value of a numeric column. The SUM() function returns the total sum of a numeric column.

- ```
  SELECT COUNT(column_name)
  FROM table_name
  WHERE condition;
  ```
  → *Replace Count with AVG & SUM*

13. SQL LIKE Operator:

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. There are two wildcards often used in conjunction with the LIKE operator:

The percent sign (%) represents zero, one, or multiple characters.

The underscore sign (_) represents one, single character.

Note: MS Access uses an asterisk (*) instead of the percent sign (%), and a question mark (?) instead of the underscore (_).The percent sign and the underscore can also be used in combinations!

- ```
  SELECT column1, column2, ...
  FROM table_name
  WHERE columnN LIKE pattern;
  ```

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

14. SQL Wildcard Characters:

A wildcard character is used to substitute one or more characters in a string. Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
  ```

Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and hat
-	Represents any single character within the specified range	c[a-b]t finds cat and cbt

15. SQL IN:

The IN operator allows you to specify multiple values in a WHERE clause. The IN operator is a shorthand for multiple OR conditions.

- ```
  SELECT column_name(s)
  FROM table_name
  WHERE column_name IN (value1, value2, ...);
  ```
- ```
  SELECT column_name(s)
  FROM table_name
  WHERE column_name IN (SELECT STATEMENT);
  ```

16. SQL BETWEEN:

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
The BETWEEN operator is inclusive: begin and end values are included.

- ```
  SELECT column_name(s)
  FROM table_name
  WHERE column_name BETWEEN value1 AND value2;
  ```
  → *Put the between values you need for. Like between 10 & 20*

17. SQL Aliases:

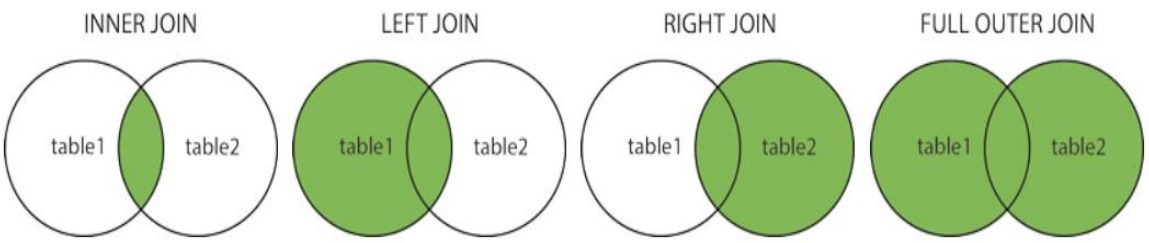
SQL aliases are used to give a table, or a column in a table, a temporary name.
Aliases are often used to make column names more readable.
An alias only exists for the duration of that query.
An alias is created with the AS keyword.

- ```
  SELECT column_name AS alias_name
  FROM table_name;
  ```
  → *Column Syntax*
- ```
  SELECT column_name(s)
  FROM table_name AS alias_name;
  ```
  → *Table Syntax*

18. SQL Joins:

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

- ```
  SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
  FROM Orders
  INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
  ```
  → *Example Shows Orders customerID inner joined with Customers CustomerID which is common between the tables.*



Different Types of SQL JOINs

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table

19. SQL INNER JOIN:

The INNER JOIN keyword selects records that have matching values in both tables.

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

20. SQL LEFT JOIN:

The 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;
```

21. SQL RIGHT JOIN:

The 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.

Note: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

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

22. SQL FULL OUTER JOIN:

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

Note: FULL OUTER JOIN and FULL JOIN are the same.

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```
→ *Returns very large dataset*

23. SQL Self Join:

A self-join is a regular join, but the table is joined with itself.

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

```
SELECT A.CustomerName AS CustomerName1,
B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

24. SQL UNION:

The UNION operator is used to combine the result-set of two or more SELECT statements.

Every SELECT statement within UNION must have the same number of columns
The columns must also have similar data types
The columns in every SELECT statement must also be in the same order

- ```
  SELECT column_name(s) FROM table1
  UNION
  SELECT column_name(s) FROM table2;
  ```
- ```
  SELECT column_name(s) FROM table1
  UNION ALL
  SELECT column_name(s) FROM table2;
  ```
  → *The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL*

25. SQL GROUP BY:

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country.
The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

- ```
  SELECT column_name(s)
  FROM table_name
  WHERE condition
  GROUP BY column_name(s)
  ORDER BY column_name(s);
  ```

26. SQL HAVING:

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

- ```
  SELECT column_name(s)
  FROM table_name
  WHERE condition
  GROUP BY column_name(s)
  HAVING condition
  ORDER BY column_name(s);
  ```
- ```
  SELECT COUNT(CustomerID), Country
  FROM Customers
  GROUP BY Country
  HAVING COUNT(CustomerID) > 5;
  ```
  → *Include Customer More than 5*

27. SQL EXISTS:

The EXISTS operator is used to test for the existence of any record in a subquery.
The EXISTS operator returns TRUE if the subquery returns one or more records.

- ```
  SELECT column_name(s)
  FROM table_name
  WHERE EXISTS
  (SELECT column_name FROM table_name WHERE condition);
  ```

28. SQL ANY and ALL:

The **ANY** and **ALL** operators allow you to perform a comparison between a single column value and a range of other values.
The **ANY** operator:

- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition
- **ANY** means that the condition will be true if the operation is true for any of the values in the range.

- ```
  SELECT column_name(s)
  FROM table_name
  WHERE column_name operator ANY
    (SELECT column_name
    FROM table_name
    WHERE condition);
  ```

  Note: The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=)

  The SQL **ALL** Operator

  - Returns a boolean value as a result
  - Returns TRUE if ALL of the subquery values meet the condition
  - Is used with SELECT, WHERE and HAVING statements
- ```
  SELECT ALL column_name(s)
  FROM table_name
  WHERE condition;
  ```
  → *All syntax with All*

- ```
  SELECT column_name(s)
  FROM table_name
  WHERE column_name operator ALL
    (SELECT column_name
    FROM table_name
    WHERE condition);
  ```
  → *All Syntax With WHERE or HAVING*

  Note operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=)

29. SQL SELECT INTO:

The SELECT INTO statement copies data from one table into a new table.

- ```
  SELECT *
  INTO newtable [IN externaldb]
  FROM oldtable
  WHERE condition;
  ```
  → *Copy all columns into a new table*
- ```
  SELECT column1, column2, column3, ...
  INTO newtable [IN externaldb]
  FROM oldtable
  WHERE condition;
  ```
  → *Copy only some columns into a new table*

The new table will be created with the column-names and types as defined in the old table. You can create new column names using the AS clause.

30. SQL INSERT INTO SELECT Statement:

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.
The INSERT INTO SELECT statement requires that the data types in source and target tables match.
Note: The existing records in the target table are unaffected

- ```
  INSERT INTO table2
  SELECT * FROM table1
  WHERE condition;
  ```
  → *Copy all columns from one table to another table*
- ```
  INSERT INTO table2 (column1, column2, column3, ...)
  SELECT column1, column2, column3, ...
  FROM table1
  WHERE condition;
  ```
  → *Copy only some columns from one table into another table*
  ```

31. SQL CASE:

The CASE statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

- ```
  CASE
      WHEN condition1 THEN result1
      WHEN condition2 THEN result2
      WHEN conditionN THEN resultN
      ELSE result
  END;
  ```
- ```
  SELECT OrderID, Quantity,
  CASE
      WHEN Quantity > 30 THEN 'The quantity is greater than 30'
      WHEN Quantity = 30 THEN 'The quantity is 30'
      ELSE 'The quantity is under 30'
  END AS QuantityText
  FROM OrderDetails;
  ```
  → *Similar to IF statements*

32. SQL NULL:

SQL IFNULL(), ISNULL(), COALESCE(), and NVL() Functions

33. SQL Stored Procedures for SQL Server:

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again. So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it. You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed

- ```
  CREATE PROCEDURE procedure_name
  AS
  sql_statement
  GO;
  ```
- ```
  EXEC procedure_name;
  ```

SQL CREATE DATABASE STATEMENT

1. CREATE DATABASE:

The CREATE DATABASE statement is used to create a new SQL database.

Tip: Make sure you have admin privilege before creating any database. Once a database is created, you can check it in the list of databases with the following SQL command: SHOW DATABASES;

- ```
  CREATE DATABASE databasename;
  ```

2. DROP DATABASE:

The DROP DATABASE statement is used to drop an existing SQL database.

Note: Be careful before dropping a database. Deleting a database will result in loss of complete information stored in the database!

- ```
  DROP DATABASE databasename;
  ```

3. SQL BACKUP DATABASE:

The BACKUP DATABASE statement is used in SQL Server to create a full back up of an existing SQL database.

- ```
  BACKUP DATABASE databasename
  TO DISK = 'filepath';
  ```
- SQL BACKUP WITH DIFFERENTIAL Statement
- ```
  BACKUP DATABASE databasename
  TO DISK = 'filepath'
  WITH DIFFERENTIAL;
  ```

4. SQL CREATE TABLE:

The CREATE TABLE statement is used to create a new table in a database.
The column parameters specify the names of the columns of the table.
The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).
Tip: For an overview of the available data types, go to our complete Data Types Reference.

- ```
  CREATE TABLE table_name (
      column1 datatype,
      column2 datatype,
      column3 datatype,
     ....
  );
  ```

The column parameters specify the names of the columns of the table.
The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).
Tip: For an overview of the available data types, go to our complete Data Types Reference.

5. SQL DROP TABLE:

The DROP TABLE statement is used to drop an existing table in a database.
Note: Be careful before dropping a table. Deleting a table will result in loss of complete information stored in the table!

- ```
  DROP TABLE table_name;
  ```

6. SQL ALTER TABLE:

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

- ```
  ALTER TABLE table_name
  ADD column_name datatype;
  ```

7. SQL Constraints:

SQL constraints are used to specify rules for data in a table.
Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

- ```
  CREATE TABLE table_name (
      column1 datatype constraint,
      column2 datatype constraint,
      column3 datatype constraint,
      ....
  );
  ```

8. SQL NOT NULL Constraint:

By default, a column can hold NULL values. The NOT NULL constraint enforces a column to NOT accept NULL values. This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

- SQL NOT NULL on CREATE TABLE

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

9. SQL UNIQUE Constraint:

The UNIQUE constraint ensures that all values in a column are different.
Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.
A PRIMARY KEY constraint automatically has a UNIQUE constraint.
However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

```
CREATE TABLE Persons (
    ID int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

10. SQL PRIMARY KEY Constraint:

The PRIMARY KEY constraint uniquely identifies each record in a table.
Primary keys must contain UNIQUE values, and cannot contain NULL values.
A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

11. SQL FOREIGN KEY Constraint:

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.
The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

12. SQL CHECK Constraint:

The CHECK constraint is used to limit the value range that can be placed in a column.
If you define a CHECK constraint on a column it will allow only certain values for this column.
If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

→ *My SQL Example*

13. SQL DEFAULT Constraint:

The DEFAULT constraint is used to set a default value for a column.
The default value will be added to all new records, if no other value is specified.

- ```
  CREATE TABLE Persons (
      ID int NOT NULL,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age int,
      City varchar(255) DEFAULT 'Sandnes'
  );
  ```
  → *My SQL / SQL Server / Oracle / MS Access*

14. SQL CREATE INDEX Statement:

The CREATE INDEX statement is used to create indexes in tables.
Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.
Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against

- ```
  CREATE INDEX index_name
  ON table_name (column1, column2, ...);
  ```
  → *CREATE INDEX Syntax*
- ```
  CREATE UNIQUE INDEX index_name
  ON table_name (column1, column2, ...);
  ```
  → *CREATE UNIQUE INDEX Syntax*

15. SQL AUTO INCREMENT Field:

AUTO INCREMENT Field
Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.
Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

- ```
  CREATE TABLE Persons (
      Personid int NOT NULL AUTO_INCREMENT,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age int,
      PRIMARY KEY (Personid)
  );
  ```
  → *Syntax for MySQL*

MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.
By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.
To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:

16. SQL Working With Dates:

The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.
As long as your data contains only the date portion, your queries will work as expected. However, if a time portion is involved, it gets more complicated.
SQL Date Data Types
MySQL comes with the following data types for storing a date or a date/time value in the database:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY
- SQL Server comes with the following data types for storing a date or a date/time value in the database:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- SMALLDATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: a unique number

17. SQL Views:

In SQL, a 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 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.
A view is created with the CREATE VIEW statement.

- ```
  CREATE VIEW view_name AS
  SELECT column1, column2, ...
  FROM table_name
  WHERE condition;
  ```
  → *Create view syntax*

18. SQL Injection:

SQL injection is a code injection technique that might destroy your database.
SQL injection is one of the most common web hacking techniques.
SQL injection is the placement of malicious code in SQL statements, via web page input.
SQL in Web Pages
SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.

Look at the following example which creates a SELECT statement by adding a variable (txtUserId) to a select string. The variable is fetched from user input (getRequestString):

- ```
  txtUserId = getRequestString("UserId");
  txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
  ```

JOIN TO EXPLORE MORE

Non-Tech to Tech basically designed for aspirants, employees, Students who don't belong / have any technical degree / work experience. Let's join our hands & guide each other in best possible way to achieve our profession goal / growth. Share your experience.

Do join me for more such informative data stories and resources.

LINK TO JOIN/FOLLOW BY CLICKING THE NAME / SNAP

[Bimal Subhasis]

Thanks & Warm Regards,

BIMAL SUBHASIS