

Unit-1

Introduction to SQL

By Aditi Barot

Assistant professor

President Institute of Computer Application

Topics

- Introduction to SQL
- Data Definition Commands
- Data Manipulation Commands
- Select Query
- Advanced Data Definition Commands

Introduction to SQL

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.

SQL Functions

- Data Definition
- Data Retrieval
- Data Manipulation
- Access Control
- Data sharing
- Data integrity

Function of DBMS

- 1) **Data Dictionary Management:** A Data Dictionary is central store house which store definitions of the data Elements and their relationship (Meta data). The DBMS uses data dictionary to look up the required data component structure and any changes made in the database structure are automatically recorded in the data dictionary.
- 2) **Transaction Management:** A transaction is a sequence of database operations that represents a logical unit of work and that accesses a database and transformation it from one state to another state.
- 3) **Concurrency Control:** These in a database management activity of coordinating the actions of database manipulation processes that operate concurrently that access shared data and potentially interfere with one another.
- 4) **Recovery Management:** This in a database ensures the aborted or failed transaction create no effect on the database or other transactions.
- 5) **Security Management:** Refers to the protection of data against un-authorized **access**.
- 6) **Storage Management:** The DBMS provides a mechanism for management of permanent storage of the data.
- 7) **Language Interface:** The DBMS provide support language for definition and manipulation of data in the database.

SQL Features and Benefits

- Vendor independence
- Portability across computer systems
- SQL standards
- IBM Endorsement
- Microsoft commitment
- Java Integration
- Client/server architecture
- Complete database language

- Interactive
- Multiple views of data
- High-level English like structure
- Dynamic data definition

Role of SQL:

- SQL is an interactive query language. Users type SQL commands into an interactive SQL program to retrieve data and display on the screen, providing a convenient, easy-to-use database queries.
- SQL is a database programming language. Programmers embed SQL commands into their application programs to access the data in database. Both user-written programs and database utility program (such as report writers and data entry tools) use this technique database access.
- SQL is a database administration language. The database administrator responsible for managing a minicomputer or mainframe database uses SQL to define the database structure control access to the stored data.
- SQL is a client/server language. Personal computer programs use SQL to communicate over a network with database servers that store shared data. This client/server architecture has become very popular for enterprise-class applications.
- SQL is an Internet data access language. Internet web servers that interact with corporate data and Internet applications servers all use SQL as a standard language for accessing corporate databases.

Data types

1. Binary Data types :

- There are four subtypes of this datatype which are given below :

Data Type	Description
binary	Maximum length of 8000 bytes (Fixed-Length binary data)
varbinary	Maximum length of 8000 bytes (Variable Length binary data)
varbinary(max)	Maximum length of 231 bytes (SQL Server 2005 only). (Variable Length binary data)
image	Maximum length of 2,147,483,647 bytes (Variable Length binary data)

2. Exact Numeric Data type :

- There are nine subtypes which are given below in the table. The table contains the range of data in a particular type.

Data Type	From	To
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	$-10^{38} + 1$	$10^{38} - 1$
numeric	$-10^{38} + 1$	$10^{38} - 1$
money	-922,337,203,685,477.5808	922,337,203,685,477.5808
smallmoney	-214,748.3648	214,748.3648

3. Approximate Numeric Data type :

- The subtypes of this data type are given in the table with the range.

-

Data Type	From	To
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

4. Character String Data type :

Data Type	Description
char	Maximum length of 8000 characters. (Fixed-Length non-Unicode Characters)
varchar	Maximum length of 8000 characters. (Variable-Length non-Unicode Characters)
varchar(max)	Maximum length of 231 characters (SQL Server 2005 only). (Variable Length non-Unicode data)
text	Maximum length of 2,147,483,647 characters (Variable Length non-Unicode data)

5. Unicode Character String Datatype :

Data Type	Description
nchar	Maximum length of 4000 characters. (Fixed-Length Unicode Characters)
Nvarchar	Maximum length of 4000 characters. (Variable-Length Unicode Characters)
nvarchar(max)	Maximum length of 231 characters (SQL Server 2005 only). (Variable Length Unicode data)

6. Date and Time Datatype :

Data Type	From	To
datetime	Jan 1, 1753	Dec 31, 9999
smalldatetime	Jan 1, 1900	Jun 6, 2079
date	Stores a date like June 30, 1991	
time	Stores a time of day like 12:30 P.M.	

DDL Data Definition Language (DDL)

- statements are used to define the database structure or schema.

Some examples:

- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- COMMENT - add comments to the data dictionary RENAME - rename an object

Create Database

- The CREATE DATABASE statement is used to create a new SQL database.
- Always the database name should be unique within the RDBMS and name cannot contain space.
- Syntax:

`CREATE DATABASE databasename;`

Ex: `CREATE DATABASE students;`

Drop database

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

- Syntax/command :

DROP DATABASE *databasename*;

Ex: DROP DATABASE *students*;

Create Table

- The CREATE TABLE statement is used to create a new table in a database.
- CREATE TABLE *table_name* (
 column1 datatype,
 column2 datatype,
 column3 datatype,

);

Drop table

- The DROP TABLE statement is used to drop an existing table in a database.
- DROP TABLE *table_name*;

Alter table

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

- ALTER TABLE - ADD Column:

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

- Ex:
- The following SQL adds an "Email" column to the "Customers" table:
ALTER TABLE Customers
 ADD Email varchar(255);

Alter table

- ALTER TABLE - DROP COLUMN

```
ALTER TABLE table_name  
DROP COLUMN column_name;
```

- Ex: ALTER TABLE Customers
DROP COLUMN Email;

Alter table

- ALTER TABLE - ALTER/MODIFY COLUMN
- To change the data type of a column in a table, use the following syntax:

ALTER TABLE *table_name*

ALTER COLUMN *column_name* *datatype*;

DML

Data Manipulation Language (DML) statements are used for managing data within schema objects.

- SELECT - retrieve data from the a database
- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - deletes all records from a table, the space for the records remain
- MERGE - UPSERT operation (insert or update) CALL - call a PL/SQL or Java subprogram EXPLAIN PLAN - explain access path to data
- LOCK TABLE - control concurrency

SQL Constraints

- Constraints are used to limit the type of data that can go into a table.
- Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).
- We will focus on the following constraints:
 1. NOT NULL
 2. UNIQUE
 3. PRIMARY KEY
 4. FOREIGN KEY
 5. CHECK
 6. DEFAULT

[Details for the same is attached here.](#)

SQL NOT NULL Constraint

- The NOT NULL constraint enforces a column to NOT accept NULL values.
- The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.
- The following SQL enforces the "P_Id" column and the "LastName" column to not accept NULL values:
- ```
CREATE TABLE Persons
(
 P_Id int NOT NULL,
 LastName varchar(255) NOT NULL,
 FirstName varchar(255),
 Address varchar(255),
 City varchar(255)
)
```

# SQL UNIQUE Constraint

- The UNIQUE constraint uniquely identifies each record in a database table.
- The UNIQUE and PRIMARY KEY constraints both provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.
- Note that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.
- The following SQL creates a UNIQUE constraint on the "P\_Id" column when the "Persons" table is created:
- ```
CREATE TABLE Person  
(  
  P_Id int NOT NULL UNIQUE,  
  LastName varchar(255) NOT NULL,  
  FirstName varchar(255),  
  Address varchar(255),  
  City varchar(255)  
)
```

- **SQL UNIQUE Constraint on ALTER TABLE**
- To create a UNIQUE constraint on the "P_Id" column when the table is already created, use the following SQL:
- **MySQL / SQL Server / Oracle / MS Access:**
- ALTER TABLE Persons
ADD UNIQUE (P_Id)

- **To DROP a UNIQUE Constraint**
- To drop a UNIQUE constraint, use the following SQL:
- **MySQL:**
- ALTER TABLE Persons
DROP INDEX uc_PersonID

SQL PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a database table.
- Primary keys must contain unique values.
- A primary key column cannot contain NULL values.
- Each table should have a primary key, and each table can have only one primary key.

- **SQL PRIMARY KEY Constraint on CREATE TABLE**
- The following SQL creates a PRIMARY KEY on the "P_Id" column when the "Persons" table is created:
- **MySQL:**
- ```
CREATE TABLE Persons
(
 P_Id int NOT NULL,
 LastName varchar(255) NOT NULL,
 FirstName varchar(255),
 Address varchar(255),
 City varchar(255),
 PRIMARY KEY (P_Id)
)
```

- To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:
- **MySQL / SQL Server / Oracle / MS Access:**
- CREATE TABLE Persons  
(  
P\_Id int NOT NULL,  
LastName varchar(255) NOT NULL,  
FirstName varchar(255),  
Address varchar(255),  
City varchar(255),  
CONSTRAINT pk\_PersonID PRIMARY KEY (P\_Id,LastName)  
)



- **SQL PRIMARY KEY Constraint on ALTER TABLE**
- To create a PRIMARY KEY constraint on the "P\_Id" column when the table is already created, use the following SQL:
- **MySQL / SQL Server / Oracle / MS Access:**
- ALTER TABLE Persons  
ADD PRIMARY KEY (P\_Id)

- **To DROP a PRIMARY KEY Constraint**
- To drop a PRIMARY KEY constraint, use the following SQL:
- **MySQL:**
- ALTER TABLE Persons  
DROP PRIMARY KEY

# Adding table rows

- Insertion of data into tables : When inserting a single row of data into the table , the insert operation.
- Creates a new row in the database table
- Loads the values passed into all the columns specified.
- Syntax:
- INSERT INTO <tablename> (column name, column name)Values(expression,expression);
- INSERT INTO student (s\_name, s\_city, s\_age , s\_result)   Values("mitesh","hmt",24,"pass");

Table Name : Student.

| s_name | s_city | S_age | s_result |
|--------|--------|-------|----------|
| Mukesh | Hmt    | 24    | Pass     |
| Dhruvi | Idar   | 26    | Pass     |
| Magan  | Khed   | 25    | Fail     |

+ Table Name : Student.

| s_name | s_city | S_age | s_result |
|--------|--------|-------|----------|
| Mukesh | Hmt    | 24    | Pass     |
| Dhruvi | Idar   | 26    | Pass     |
| Magan  | Khed   | 25    | Fail     |
| Mitesh | Hmt    | 24    | pass     |

# Saving table changes

- COMMIT :

COMMIT in SQL is a transaction control language which is used to permanently save the changes done in the transaction in tables/databases. The database cannot regain its previous state after the execution of it.

# Listing table rows

- This command is used to viewing data into the tables , once data has been inserted into a table we can achieve (access) data by using the SELECT COMMAND.
- Syntax:
- SELECT (columnname1,.....columnname n)FROM <tablename>;
- SELECT s\_name,s\_city,s\_age,s\_result from student;  
Or
- SELECT \* from student;

Output:

| <u>s_name</u> | <u>s_city</u> | <u>S_age</u> | <u>s_result</u> |
|---------------|---------------|--------------|-----------------|
| Mukesh        | Hmt           | 24           | Pass            |
| Dhruvi        | Idar          | 26           | Pass            |
| Magan         | Khed          | 25           | Fail            |

# Filtering Table Data:

- SELECT column name, column name, FROM <tablename> WHERE search condition;
- SELECT s\_name,s\_city FROM student  
WHERE age>25;
- 

⊕ Output:

| s_name | s_city |
|--------|--------|
| Dhruvi | Idar   |
| Magan  | Khed   |

# Updating table rows

- This UPDATE command is used to change or modify data values in a table.
- Syntax:
- UPDATE <tablename> SET columnname = expression, columnname = expression..... WHERE <search condition>;
- UPDATE student SET s\_city= "Ahmedabad", s\_result = "Fail" WHERE s\_age=26;

Output:

| <u>s name</u> | <u>s_city</u> | <u>S age</u> | <u>s_result</u> |
|---------------|---------------|--------------|-----------------|
| Mukesh        | Hmt           | 24           | Pass            |
| Dhruvi        | Ahmedabad     | 26           | Fail            |
| Magan         | Khed          | 25           | Fail            |



# Restoring table contents

- ROLLBACK in SQL is a transactional control language which is used to undo the transactions that have not been saved in database. The command is only be used to undo changes since the last COMMIT.
- `sql> ROLLBACK;`

### Difference between COMMIT and ROLLBACK :

| COMMIT                                                            | ROLLBACK                                               |
|-------------------------------------------------------------------|--------------------------------------------------------|
| COMMIT permanently saves the changes made by current transaction. | ROLLBACK undo the changes made by current transaction. |
| Transaction can not undo changes after COMMIT execution.          | Transaction reaches its previous state after ROLLBACK. |
| When transaction is successful, COMMIT is applied.                | When transaction is aborted, ROLLBACK occurs.          |

# Deleting table rows

- The verb DELETE in database is used to remove rows from table.
- Syntax:
- DELETE FROM <tablename>;  
DELETE FROM student; //delete all rows from table.
- Removal of a specified Row/s:
- Syntax:
- DELETE FROM <tablename> WHERE search condition;
- DELETE FROM student WHERE age=25;

# SELECT QUERY

- Conditional restriction
- Arithmetic operator
- Logical operator
- Special operator

# Select query

- The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

- Syntax:

```
SELECT column1, column2, columnN FROM
table_name;
```

- If you want to fetch all the fields available in the field, then you can use the following syntax:

```
SELECT * FROM table_name;
```

# Example:


- `SELECT * from student;`

Output:

| <u>s_name</u> | <u>s_city</u> | <u>S_age</u> | <u>s_result</u> |
|---------------|---------------|--------------|-----------------|
| Mukesh        | Hmt           | 24           | Pass            |
| Dhruvi        | Idar          | 26           | Pass            |
| Magan         | Khed          | 25           | Fail            |

# With conditional restriction

- Where clause :
- Filtering Table Data:
- SELECT column name, column name, FROM <tablename> WHERE search condition;
- SELECT s\_name,s\_city FROM student WHERE age>25;

 Output:

| s_name | s_city |
|--------|--------|
| Dhruvi | Idar   |
| Magan  | Khed   |

# Arithmetic operators

| Operator           | Description                                                            | Example               |
|--------------------|------------------------------------------------------------------------|-----------------------|
| + (Addition)       | Adds values on either side of the operator.                            | $a + b$ will give 30  |
| - (Subtraction)    | Subtracts right hand operand from left hand operand.                   | $a - b$ will give -10 |
| * (Multiplication) | Multiplies values on either side of the operator.                      | $a * b$ will give 200 |
| / (Division)       | Divides left hand operand by right hand operand.                       | $b / a$ will give 2   |
| % (Modulus)        | Divides left hand operand by right hand operand and returns remainder. | $b \% a$ will give 0  |



# SQL Comparison Operators

| Operator | Description                                                                                                         | Example              |
|----------|---------------------------------------------------------------------------------------------------------------------|----------------------|
| =        | Checks if the values of two operands are equal or not, if yes then condition becomes true.                          | (a = b) is not true. |
| !=       | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.         | (a != b) is true.    |
| <>       | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.         | (a <> b) is true.    |
| >        | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (a > b) is not true. |

# SQL Comparison Operators

|    |                                                                                                                                 |                       |
|----|---------------------------------------------------------------------------------------------------------------------------------|-----------------------|
| >  | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.             | (a > b) is not true.  |
| <  | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.                | (a < b) is true.      |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (a >= b) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.    | (a <= b) is true.     |
| !< | Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.            | (a !< b) is false.    |
| !> | Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.         | (a !> b) is true.     |

# SQL Logical Operators

## AND

The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

## OR

The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause

## NOT

The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator

# AND

- The WHERE condition in MySQL when used together with the AND logical operator, is only executed if ALL filter criteria specified are met.
- Let's now look at a practical example – Suppose we want to get a list of all the movies in category 2 that were released in 2008, we would use the script shown below to achieve that.
- `SELECT * FROM `movies` WHERE `category_id` = 2 AND `year_released` = 2008;`
- o/p:

| movie_id | title                    | director         | year_released | category_id |
|----------|--------------------------|------------------|---------------|-------------|
| 2        | Forgetting Sarah Marshal | Nicholas Stoller | 2008          | 2           |

# OR

- The WHERE clause when used together with the OR operator, is only executed if any or the entire specified filter criteria is met.
- The following script gets all the movies in either category 1 or category 2
- `SELECT * FROM `movies` WHERE `category_id` = 1 OR `category_id` = 2;`

| movie_id | title                      | director         | year_released | category_id |
|----------|----------------------------|------------------|---------------|-------------|
| 1        | Pirates of the Caribbean 4 | Rob Marshall     | 2011          | 1           |
| 2        | Forgetting Sarah Marshal   | Nicholas Stoller | 2008          | 2           |

# NOT

- The WHERE clause when used together with the NOT IN keyword DOES NOT affects the rows whose values matches the list of values provided in the NOT IN keyword.
- The following query gives rows where membership\_number is NOT 1 , 2 or 3.
- `SELECT * FROM `members` WHERE `membership_number` NOT IN (1,2,3);`

| membership_number | full_names      | gender | date_of_birth | physical_address | postal_address |
|-------------------|-----------------|--------|---------------|------------------|----------------|
| 4                 | Gloria Williams | Female | 14-02-1984    | 2nd Street 23    | NULL           |

# Special operator

- The different special operators in SQL are as follows –
- IN operator
- NOT IN Operator
- BETWEEN Operator
- NOT BETWEEN Operator
- LIKE Operator

SQL Operator

Now let us create a table to understand the examples of special operators –

<Employee>

| Emp_ID | Emp_Name | Emp_Salary | Emp_DeptID | Emp_DeptName |
|--------|----------|------------|------------|--------------|
| 1      | Aaron    | 10000      | 10         | Technical    |
| 2      | Harry    | 12000      | 20         | Operations   |
| 3      | Mary     | 5000       | 30         | Finance      |
| 4      | Angel    | 55000      | 10         | Technical    |
| 5      | Will     | 20000      | 30         | Finance      |

<Dependents>

| Dep_ID | Emp_ID | Dep_Name | Dep_Age |
|--------|--------|----------|---------|
| 1001   | 2      | Keith    | 88      |
| 1002   | 3      | Kim      | 5       |
| 1003   | 5      | Lucy     | 90      |



# IN operator

- The IN operator is true if the query results in values that are contained in the list of constant values for the IN operator. For example –
- `Select * from Employee Where Emp_ID IN (1,2,5);`
- This query gives details about the employees whose Employee number is 1,2 or 5 i.e Aaron, Harry and Will.

| Emp_ID | Emp_Name | Emp_Salary | Emp_DeptID | Emp_DeptName |
|--------|----------|------------|------------|--------------|
| 1      | Aaron    | 10000      | 10         | Technical    |
| 2      | Harry    | 12000      | 20         | Operations   |
| 5      | Will     | 20000      | 30         | Finance      |

# BETWEEN operator

- The BETWEEN operator returns the information within a given range of values, where the minimum and maximum of the range is specified. For example –
- `Select * from Employee Where Emp_Salary BETWEEN 20000 AND 60000;`
- This query returns information about all the employees whose salary is between the range of 20000 and 60000 i.e. Angel and Will in this case.

| Emp_ID | Emp_Name | Emp_Salary | Emp_DeptID | Emp_DeptName |
|--------|----------|------------|------------|--------------|
| 4      | Angel    | 55000      | 10         | Technical    |
| 5      | Will     | 20000      | 30         | Finance      |

# LIKE operator

- The LIKE operator is used to select the values that match the patterns specified in the query. Two wildcard operators are used for this. For example –
- `Select * from Employee Where Emp_Name LIKE "A%"`
- This query returns the data of all the employees whose name starts with A i.e. Aaron and Angel in our example.

| Emp_ID | Emp_Name | Emp_Salary | Emp_DeptID | Emp_DeptName |
|--------|----------|------------|------------|--------------|
| 1      | Aaron    | 10000      | 10         | Technical    |
| 4      | Angel    | 55000      | 10         | Technical    |

| 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"                       |

# Copying parts of table

- MySQL copy table to a new table:

```
CREATE TABLE new table SELECT * FROM
exisitng_table;
```

To copy parts of data from existing table,we can use where caluse:

```
CREATE TABLE new table SELECT * FROM
exisitng_table WHERE conditions;
```

- It is very important to check whether the table you want to create that already exists before creating it. To do so, you use IF NOT EXIST clause in the CREATE TABLE statement.
- `CREATE TABLE IF NOT EXIST new_table SELECT * FROM exisitng_table;`

# Example:

- In cases we want to copy the offices in the US only, we can add the WHERE clause to the SELECT statement as follows:
- `CREATE TABLE IF NOT EXISTS offices_usa  
SELECT * FROM offices WHERE country = 'USA';`

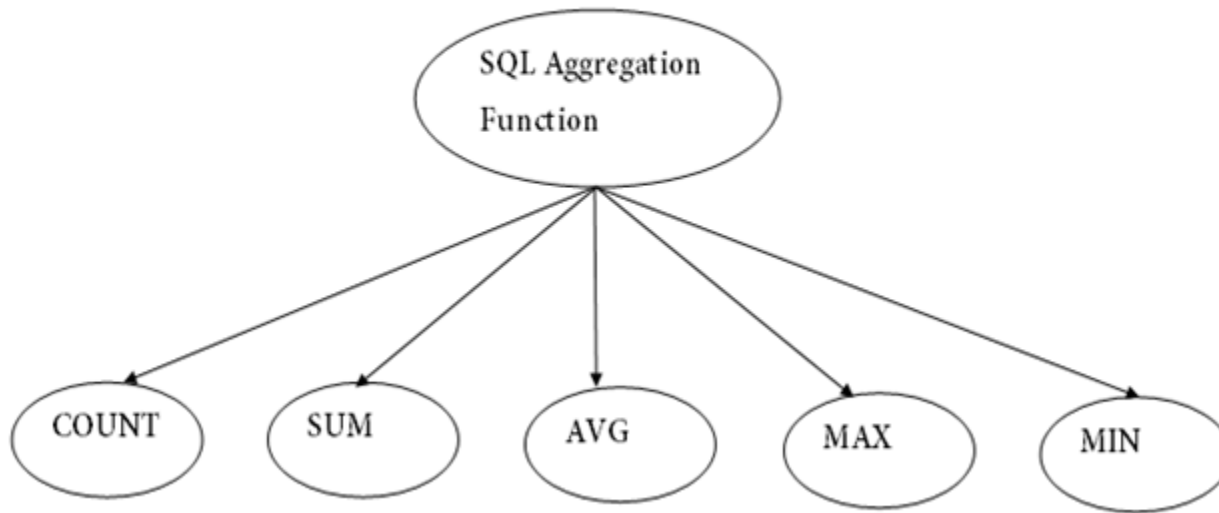
Use for more detail: [copy in my sql](#)

# Adding primary and foreign key designations

- When table is copied ,integrity rules do not copy which we need to add manually.
- Syntax:
- ALTER TABLE tablename ADD PRIMARY KEY (field name);
- EX: ALTER TABLE Employee  
ADD PRIMARY KEY (e\_id)  
ADD Foreign key (v\_code) References Vendor;



# Aggregate functions



# COUNT FUNCTION

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT(\*) that returns the count of all the rows in a specified table. COUNT(\*) considers duplicate and Null.
- EX: SELECT COUNT(\*) FROM table\_name;
- SELECT COUNT(\*) FROM PRODUCT\_MAST

# SUM Function

- Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.
- `SELECT SUM(COST) FROM PRODUCT_MAST;`

# MAX Function

- MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.
- `SELECT MAX(RATE) FROM PRODUCT_MAST;`

# MIN Function

- MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.
- `SELECT MIN(RATE) FROM PRODUCT_MAST;`
- [Detailed link](#)(Refer for having clause)
- <https://www.javatpoint.com/dbms-sql-aggregate-function>

# Views in SQL

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table.
- Link for practical  
: <https://www.javatpoint.com/dbms-sql-view>