

## Unit: 1

### Introduction to SQL

#### 1) Introduction to SQL

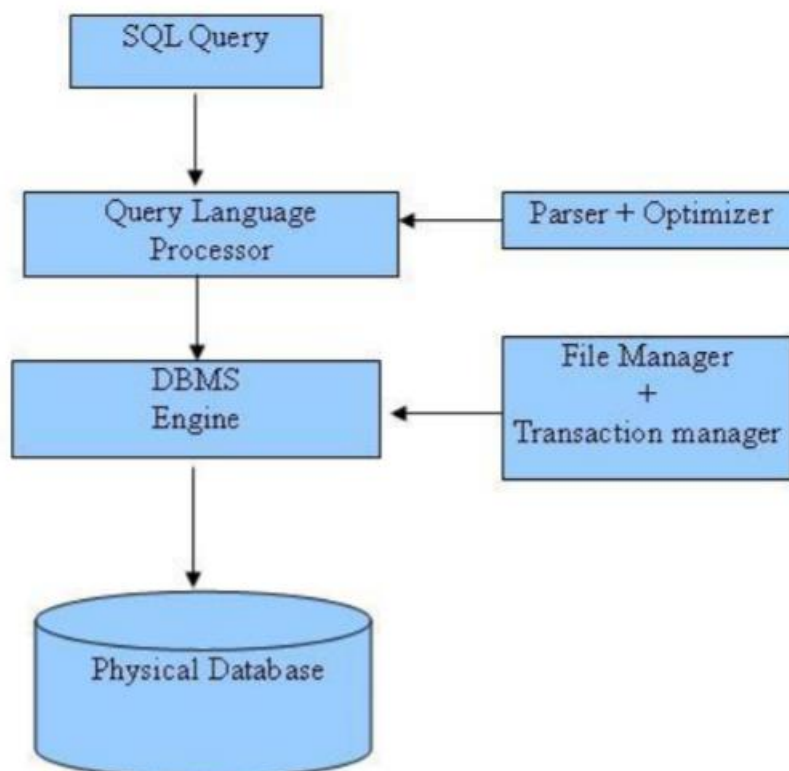
- SQL is a standard language for accessing and manipulating databases.
- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

#### **SQL Process:**

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task. There are various components included in the process. These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc. Classic query engine handles all non-SQL queries, but SQL query engine won't handle logical files.

Following is a simple diagram showing SQL Architecture



**SQL Commands:** The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their nature:

1. **DDL(CREATE, ALTER, DROP)**
2. **DML(INSERT, UPDATE, DELET)**

## 3. DCL(GRANT,REVOKE)

## 4. DQL(SELECT)

❖ Data Type

The data type of a column defines what value the column can hold: integer, character, money, date and time, binary, and so on.

SQL Data Type is an attribute that specifies the type of data of any object. Each column, variable and expression has a related data type in SQL. You can use these data types while creating your tables. You can choose a data type for a table column based on your requirement.

SQL Server offers six categories of data types for your use which are listed below –

**In MySQL there are three main data types: text, number, and date.**

**(1)Text data types:**

Data type	Description
<b>CHAR(size)</b>	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
<b>VARCHAR(size)</b>	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. <b>Note:</b> If you put a greater value than 255 it will be converted to a TEXT type
<b>Number(size)</b>	The number data type is used to store number (fixed or floating point).
<b>LONG</b>	This data type is used to store variable length character string containing upto 2 GB
<b>RAW/LONG RAW</b>	This data type is used to store binary data such as picture and image. RAW data type can have maximum length of 255 bytes. LONG RAW data type can contain up to 2 GB
<b>BLOB</b>	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data(4 giga bytes)

**(2)Number data types:**

Data type	Description
<b>TINYINT(size)</b>	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
<b>SMALLINT(size)</b>	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis
<b>MEDIUMINT(size)</b>	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
<b>INT(size)</b>	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
<b>BIGINT(size)</b>	-9223372036854775808 to 9223372036854775807 normal. 0 to

	18446744073709551615 UNSIGNED*. The maximum number of digits may be specified in parenthesis
<b>FLOAT(size,d)</b>	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
<b>DOUBLE(size,d)</b>	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
<b>DECIMAL(size,d)</b>	A DOUBLE stored as a string, allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

### (3) Date data types:

Data type	Description
<b>DATE</b>	A date. Format: YYYY-MM-DD <b>Note:</b> The supported range is from '1000-01-01' to '9999-12-31'
<b>TIMESTAMP</b>	*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MI:SS <b>Note:</b> The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC

## ❖ Creating Table Structure

The CREATE TABLE statement is used to create a new table in a database.

### Syntax

CREATE TABLE table\_name

```
(
  column1 datatype,
  column2 datatype,
  column3 datatype,
  ....
);
```

- First, specify the table name to which the new table belongs on the CREATE TABLE clause.
- Second, list all columns of the table within the parentheses. In case a table has multiple columns, you need to separate them by commas (,).
- The column parameters specify the names of the columns of the table
- The data type parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

### SQL CREATE TABLE Example:

The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

#### Example

```
CREATE TABLE Persons
```

```
(  
    PersonID int,  
    LastName varchar(25),  
    FirstName varchar(25),  
    Address varchar(25),  
    City varchar(25)  
);
```

### Creating a new table from exiting table

#### Syntax

```
CREATE TABLE NEW_TABLE_NAME  
AS  
SELECT [column1, column2, ..., columnN ]  
FROM EXISTING_TABLE_NAME  
[WHERE];
```

Here, column1, column2... are the fields of the existing table and the same would be used to create fields of the new table.

#### Example

Following is an example, which would create a table SALARY using the CUSTOMERS table and having the fields customer ID and customer SALARY –

```
CREATE TABLE SALARY  
AS  
  
SELECT ID, SALARY  
  
FROM CUSTOMERS;
```

## ❖ SQL Constraints

Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

### A CONSTRAINT can be one of the following:

Constraints could be either on a column level or a table level. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.

- **Column-level constraint**

Column-level constraints refer to a single column in the table and do not specify a column name (except check constraints). They refer to the column that they follow.

- **Table-level constraint**

Table-level constraints refer to one or more columns in the table. Table-level constraints specify the names of the columns to which they apply. Table-level CHECK constraints can refer to 0 or more columns in the table.

Following are some of the most commonly used constraints available in SQL.

1. **NOT NULL** Constraint – Ensures that a column cannot have NULL value.
2. **DEFAULT** Constraint – Provides a default value for a column when none is specified.
3. **UNIQUE** Constraint – Ensures that all values in a column are different.
4. **PRIMARY Key** – uniquely identifies each row/record in a database table.
5. **FOREIGN Key** – uniquely identifies a row/record in any of the given database table.
6. **CHECK** Constraint – The CHECK constraint ensures that all the values in a column satisfies certain conditions.

### 1) Not Null Constraints

By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such a constraint on this column specifying that NULL is now not allowed for that column.

**Syntax:**

```
CREATE TABLE table_name
(
    ...
    column_name data_type(size) NOT NULL,
    ...
);
```

### Example

For example, the following SQL query creates a new table called CUSTOMERS and adds five columns, three of which, are ID NAME and AGE, In this we specify not to accept NULLs –

```
CREATE TABLE CUSTOMERS
(
    ID          NUMBAR (3) NOT NULL,
    NAME        VARCHAR (20) NOT NULL,
    AGE         NUMBAR (3) ,
    ADDRESS     VARCHAR(25) ,
    SALARY      NUMBER(10)
);
```

If CUSTOMERS table has already been created, then to add a NOT NULL constraint to the SALARY column in Oracle and MySQL, you would write a query like the one that is shown in the following code block.

### ALTER TABLE Using NOT NULL Constrain

```
ALTER TABLE CUSTOMERS  
MODIFY SALARY NUMBER(10) NOT NULL;
```

### Drop NOT NULL Constraint

```
ALTER TABLE TABLENAME  
MODIFY COLUMNNAME DATATYPE NULL;
```

## 2) DEFAULT

The **DEFAULT** constraint is used to set a default value for a column. the DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

### Syntax:

```
CREATE TABLE table_name  
(  
    ...  
    column_name data_type(size) DEFAULT <value>,  
    ...  
);
```

### Example

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, the SALARY column is set to 5000.00 by default, so in case the INSERT INTO statement does not provide a value for this column, then by default this column would be set to 5000.00.

```
CREATE TABLE CUSTOMERS  
(  
    ID          NUMBAR (3) NOT NULL,  
    NAME        VARCHAR (20) ,  
    AGE         NUMBAR (3) ,  
    ADDRESS     VARCHAR(25) ,  
    SALARY      NUMBER(10) DEFAULT 2000  
);
```

If the CUSTOMERS table has already been created, then to add a DEFAULT constraint to the SALARY column, you would write a query like the one which is shown in the code block below.

### ALTER TABLE Using Default Constraint

```
ALTER TABLE CUSTOMERS  
MODIFY SALARY NUMBER(10) DEFAULT 5000.00;
```

### Drop Default Constraint

To drop a DEFAULT constraint, use the following SQL query.

```
ALTER TABLE TABLENAME  
MODIFY COLUMNNAME DEFAULT;
```

### 3) UNIQUE

- 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.
- The **UNIQUE** constraint designates a column or combination of columns as a unique key. To satisfy a **UNIQUE** constraint, no two rows in the table can have the same value for the unique key. However, the unique key made up of a single column can contain nulls.
- A unique key column cannot be of datatype LONG or LONG RAW. You cannot designate the same column or combination of columns as both a unique key and a primary key or as both a unique key and a cluster key. However, you can designate the same column or combination of columns as both a unique key and a foreign key.

#### Syntax:

```
CREATE TABLE table_name  
(  
    ...  
    column_name data_type(size) UNIQUE  
    ...  
);
```

#### Example: At column level

```
CREATE TABLE CUSTOMERS  
(  
    ID          NUMBAR (3) UNIQUE,  
    NAME        VARCHAR (20) NOT NULL,  
    AGE         NUMBAR (3) NOT NULL,  
    ADDRESS     VARCHAR(25) ,  
    SALARY      NUMBER(10)  
);
```

#### Example: At Table level

```
CREATE TABLE CUSTOMERS  
(  
    ID          NUMBAR (3) ,  
    NAME        VARCHAR (20) NOT NULL,  
    AGE         NUMBAR (3) NOT NULL,  
    ADDRESS     VARCHAR(25) ,
```

```
SALARY    NUMBER(10),
```

```
UNIQUE(ID)
```

```
);
```

### ALTER TABLE Using Unique key Constraint

```
ALTER TABLE Persons  
ADD UNIQUE (ID);
```

### Drop Default Constraint

```
ALTER TABLE Persons  
DROP UNIQUE(ID);
```

## 4) Primary Key

- 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).

### Syntax:

```
CREATE TABLE table_name  
(  
    ...  
    column_name data_type(size) primary key,  
    ...  
);
```

### Example: At column level

```
CREATE TABLE Persons  
(  
    ID            number(3) PRIMARY KEY,  
    LastName      varchar(20) NOT NULL,  
    FirstName     varchar(25),  
    Age           number(2)  
);
```



### Example: At Table level

CREATE TABLE Persons

```
(
  ID          number(3) ,
  LastName   varchar(20) NOT NULL,
  FirstName  varchar(25),
  Age        number(2)

  CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

### ALTER TABLE Using Primary key

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID) ;
```

### Drop Primary key Constraint

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

## 5) FOREIGN Key

- 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.

**Look at the following two tables:**

**Persons Table**

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

**Orders Table**

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

- Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.
- The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.
- The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.
- The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

### Syntax:

```
CREATE TABLE table_name
(
    ...
    column_name data_type(size) foreign key references Persons(PersonID),
    ...
);
```

### Example: At column level

```
CREATE TABLE Orders
(
    OrderID    number(3) PRIMARY KEY,
    OrderNumber number(3) NOT NULL,
    PersonID   number(3) REFERENCES Persons(PersonID)
);
```

### Example: At Table level

```
CREATE TABLE Orders
(
    OrderID      number(3) NOT NULL,
    OrderNumber  number(3) NOT NULL,
    PersonID     number(3),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
);
```

### ALTER TABLE Using foreign key Constraint

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

### Drop foreign key Constraint

```
ALTER TABLE Orders
DROP FOREIGN KEY FK_PersonOrder;
```

### 6) CHECK

- 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.

#### Syntax:

```
CREATE TABLE table_name
(
    ...
    column_name data_type(size) check <condition>,
    ...
);
```

#### Example: At column level

```
CREATE TABLE Persons
(
    ID          number(3) NOT NULL,
    LastName    varchar(20) ,
    FirstName   varchar(25),
    Age         number(2) CHECK (Age>=18)
);
```

#### Example: At Table level

```
CREATE TABLE Persons
(
    ID          number(3) NOT NULL,
    LastName    varchar(20) ,
    FirstName   varchar(25),
    Age         number(2) ,
    CHECK (Age>=18)
);
```

### ALTER TABLE Using foreign key Constraint

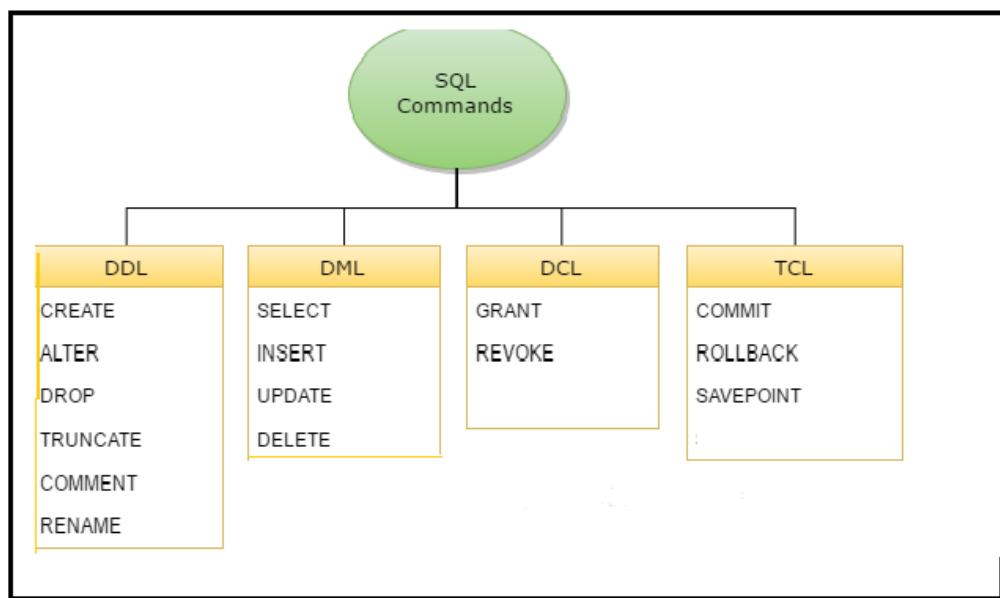
```
ALTER TABLE Persons
ADD constraint c1 CHECK (Age >=25);
```

### Drop foreign key Constraint

```
ALTER TABLE Persons
```

```
DROP CONSTRAINT c1;
```

## ❖ SQL Command



### 1. Data Manipulation Command(DML)

#### 1) Insert Command

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

**It is possible to write the INSERT INTO statement in two ways:**

#### 1) Specify both the column names and the values to be inserted:

**Syntax:**

```
INSERT INTO table_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);
```

**Example:**

```
INSERT INTO Person (id,name,age)
VALUES (1,'dixita',30);
```

- 2) If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

**Syntax:**

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

**Example:**

```
INSERT INTO Person
VALUES (1,'dixita',30);
```

### 2) Update Command

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

**Syntax:**

```
UPDATE table_name
SET column1 = value1, column2 = value2
WHERE condition;
```

**Example:**

```
UPDATE Customers
SET ContactName = 'Alfred'
WHERE CustomerID = 1;
```

### 3) Delete Command

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

**Syntax:**

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

**Example:**

```
DELETE FROM Customers WHERE CustomerID = 1;
```

### 4) Select Command

- The SELECT statement is used to select data from a database.

### Syntax:

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

Here, column1, column2, are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

### Example:

```
SELECT * FROM Person;
```

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

## 1. SELECT with DISTINCT Clause

- 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.

### Syntax:

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

### Example:

```
SELECT Country FROM Customers;
```

## 2. SELECT with WHERE Clause

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

### Syntax:

```
SELECT column1, column2,
FROM table_name
WHERE condition;
```

**Example:**

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

### 3. SELECT with ORDER BY Keyword

- 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.

**Syntax:**

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

**Example:**

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

### 4. SELECT with Group by Clause

- 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.

**Syntax:**

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

CustomerID	CustomerName	ContactName	State	City	PostalCode	Country
1	Urvi	Nivaan	Sydney	Perth	82108	Australia
2	Khusbu	Mishaka	Toronto	Otawa	05021	Canada
3	Anjali	Rudra	Rajasthan	Chittod	05023	India
4	Heena	Shiv	Boston	London	82481	UK
5	Dixita	Mit	Gujarat	Rajkot	958 22	India

**Example:**

```
SELECT COUNT (CustomerID), Country  
FROM Customers  
GROUP BY Country;
```

**Output:**

CustomerID	Country
1	Australia
1	Canada
2	India
1	UK

**5. SELECT with Having Clause**

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









**Syntax:**

```
SELECT expression1, expression2, ... expression_n,  
        aggregate_function (aggregate_expression)  
FROM tables  
WHERE conditions  
GROUP BY expression1, expression2, ... expression_n  
HAVING having_condition;  
ORDER BY column_name(s);
```

- **expression1, expression2, expression\_n:** It specifies the expressions that are not encapsulated within aggregate function. These expressions must be included in GROUP BY clause.
- **aggregate\_function:** It specifies the aggregate functions i.e. SUM, COUNT, MIN, MAX or AVG functions.
- **aggregate\_expression:** It specifies the column or expression on that the aggregate function is based on.
- **tables:** It specifies the table from where you want to retrieve records.
- **conditions:** It specifies the conditions that must be fulfilled for the record to be selected.
- **having\_conditions:** It specifies the conditions that are applied only to the aggregated results to restrict the groups of returned rows.

**Table Name: sales\_department**



EDIT	ITEM	SALE	BILLING_ADDRESS
	Shoes	120	Agra
	Belts	105	Kolkata
	Shoes	45	Allahabad
	Sari	210	Varanasi
	Sari	5000	Chennai
	Medicines	250	Salem
	Computer	210	Delhi
	Shoes	1000	Kanpur
row(s) 1 - 8 of 8			

**Example:**

```
SELECT item, SUM(sale)
FROM salesdepartment
GROUP BY item
HAVING SUM(sale) < 1000;
```

**Output:**

ITEM	Total Sales
Belts	105
Medicines	250
Computer	210

## 2. Arithmetic Operators

- Arithmetic operators can perform arithmetical operations on numeric operands involved. Arithmetic operators are addition (+), subtraction (-), multiplication (\*) and division (/). The + and - operators can also be used in date arithmetic.
- For example we have salary table which containing following field
- (id , name, b\_sal, hr , da , t\_sal ,tax )

Operator	Meaning	Operates on	Example
+ (Add)	Addition	Numeric value	SELECT id, name,( b_sal + hr +da ) FROM salary
- (Subtract)	Subtraction	Numeric value	SELECT id, name, (t_sal –tax) FROM salary

<b>*</b> <b>(Multiply)</b>	Multiplication	Numeric value	SELECT id, name, (b_sal * 2) FROM salary
<b>/ (Divide)</b>	Division	Numeric value	SELECT id, name, (tax/100) FROM salary
<b>%</b> <b>(Modulo)</b>	Returns the integer remainder of a division.	Numeric value	SELECT id, name, (tax%10) FROM salary

### 3. Comparison Operators

Operator	Description	Syntax
=	Equal	SELECT * from EMP WHERE city = 'surat' ;
<>	Not equal. <b>Note:</b> In some versions of SQL this operator may be written as !=	SELECT * from EMP WHERE city <> 'Ahmadabad' ; OR SELECT * from EMP WHERE city != 'Ahmadabad' ;
>	Greater than	SELECT id , fnm, lnm , city from EMP WHERE b_sal > 5000 ;
<	Less than	SELECT id , fnm, lnm , city from EMP WHERE b_sal > 10000 ;
>=	Greater than or equal	SELECT id , fnm, lnm , city from EMP WHERE b_sal >= 5000 ;
<=	Less than or equal	SELECT id , fnm, lnm , city from EMP WHERE b_sal <= 10000 ;
<b>BETWEEN</b>	Between a certain range	SELECT id , fnm, lnm , city from EMP WHERE b_sal BETWEEN 5000 AND

		<p>15000 ;</p> <p>(BETWEEN is inclusive of both values defining the range; the result set includes courses that b_sal 5000 and 15000 and everything in between. The lower end of the range must be listed first.)</p>
<b>LIKE</b>	Search for a pattern	<p>Like performs pattern-matching using the percent (%) or underscore (_) characters as wildcards.</p> <p>The percent wildcard is used to denote multiple characters, while the underscore wildcard is used to denote a single character.</p> <p>The next query retrieves rows where the last name begins with the uppercase letter S and ends in anything else:</p> <p><b>SELECT id, fnm, lnm, b_sal FROM EMP WHERE lnm LIKE 'S%' ;</b></p> <p>The % character may be placed at the beginning, end, or anywhere within the literal text, but always within the single quotes.</p> <p>This is also true of the underscore wildcard character, as in this statement:</p> <p><b>SELECT id, fnm, lnm, b_sal FROM EMP WHERE lnm LIKE '_a%' ;</b></p> <p>The WHERE clause returns only rows where the last name begins with any character, but the second letter must be a lowercase a.</p>
<b>IN</b>	To specify multiple possible values for a column	<p><b>SELECT * FROM EMP WHERE city IN (Rajkot, Baroda) ;</b></p> <p>(The IN operator works with a list of values, separated by commas, contained within a set of parentheses. The following query looks for courses where the city is either Rajkot or Baroda.)</p>

### Wild card character used in LIKE

**% Any string of zero or more characters**

**\_ Any single character**

[ ] Any single character within the specified range (e.g., [a-f]) or set (e.g., [abcdef])

[^] Any single character not within the specified range (e.g., [^a – f]) or set (e.g., [^abcdef])

#### 4. Logical Operators

- The Logical operators are those that are true or false. They return true or false values to combine one or more true or false values.

Operator	Description
<b>AND</b>	Logical AND compares between two Booleans as expression and returns true when both expressions are true...
<b>OR</b>	Logical OR compares between two Booleans as expression and returns true when one of the expression is true...
<b>NOT</b>	Not takes a single Boolean as an argument and changes its value from false to true or from true to false....

#### 5. Special operators

Operator	Description
<b>IN</b>	The IN operator checks a value within a set of values separated by commas and retrieves the rows from the table which are matching....
<b>BETWEEN</b>	The SQL BETWEEN operator tests an expression against a range. The range consists of a beginning, followed by an AND keyword and an end expression....
<b>ANY</b>	ANY compares a value to each value in a list or results from a query and evaluates to true if the result of an inner query contains at least one row....
<b>ALL</b>	ALL is used to select all records of a SELECT STATEMENT. It compares a value to every value in a list or results from a query. The ALL must be preceded by the comparison operators and evaluates to TRUE if the query returns no rows....
<b>SOME</b>	SOME compare a value to each value in a list or results from a query and evaluate to true if the result of an inner query contains at least one row...
<b>EXISTS</b>	The EXISTS checks the existence of a result of a subquery. The EXISTS subquery tests whether a subquery fetches at least one row. When no data is returned then this operator returns 'FALSE'...

#### 1. In Operator

- In clause is used with SELECT, INSERT, UPDATE, or DELETE statement to decrease the use of multiple OR conditions.

**Syntax:**

expressions **IN** (value1, value2,... value n)

**Table Name:Stud**

ID	NAME	AGE
1	shristee	23
2	vishal	20
3	shashank	26
4	dolly	18
5	charu	28
6	shweta	26
7	sid	22
8	avinash	27
9	heena	29

**Query:** select \*from Stud where name in ('shristee', 'dolly', 'sid')

**Output:**

ID	NAME	AGE
1	shristee	23
4	dolly	18
7	sid	22

## 2. Between Operator

- BETWEEN is used to get the values from given range in select, insert, delete or update statement

**Syntax:**

expression **BETWEEN** value1 **AND** value2;

**Example:**

select id, name, age from Stud where age between 18 AND 28;

**Output:**

ID	NAME	AGE
3	shashank	26
4	dolly	18
5	charu	28

**a. Adding Table row**

To add a column in a table, use the following syntax:

**Syntax:**

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

**Example**

```
ALTER TABLE Customers
ADD Email varchar(25);
```

**Example**

```
ALTER TABLE Persons
ADD DateOfBirth date;
```

**b. Saving table row**

COMMIT

**c. Listing table row**

```
SELECT * FROM Table Name
```

**d. Updating Table row**

```
UPDATE Table Name
SET column name = value
WHERE condition;
```

**e. Deleting Table Row**

```
DELETE FROM Table Name
WHERE Condition;
```

**2) Advance Data definition Command**

**a. Changing a Column's Data Type**

```
ALTER TABLE Table_Name
MODIFY Column_Name DataType ;
```

**b. Changing a Column's Data Characteristic**

```
ALTER TABLE Table_Name
MODIFY Column_Name DataType <constrain>;
```

**c. Adding a column**

```
ALTER TABLE Table_Name
```

ADD New\_Column DataType

**d. Dropping a column**

ALTER TABLE Table\_Name

DROP COLUMN Column\_Name;

**e. Advanced Data Update**

UPDATE Table\_Name

SET column\_name= value

WHERE <Condition>;

**f. Copying Parts of Table**

CREATE TABLE suppliers

AS

(SELECT id, address, city, state, zip

FROM companies

WHERE id > 1000);