```
mysql> SHOW DATABASES;
+--------------+
| Database     |
+--------------+
| mysql        |
| test         |
+--------------+
```

2 rows in set (0.01 sec)

Let us discuss the important DDL (Data Definition Language) commands.

7.14.1 DDL Commands

The data stored is organized in the form of well-defined schemas or relations or what are most commonly referred to as tables.

SQL allows us to write statements for defining, modifying and deleting relation schemas. These are part of Data Definition Language (DDL).

1. Creating Databases

The **CREATE DATABASE** command is used to create a database in RDBMS.

Syntax for creating a database:

CREATE DATABASE <database_name>;

For example,

```
mysql> CREATE DATABASE School;
```
← Creates database with the name School.

When the above-mentioned command gets executed, a database with the name School will be created on the system.

2. Opening Databases

Once a database has been created, we need to open it to work on it. For this, **USE** command is required.

Syntax for opening a database:

USE <database_name>;

For example,

```
mysql> USE School;
Database changed
```

3. Removing Databases

To physically remove/delete a database along with all its tables, **DROP DATABASE** command is used.

Syntax for removing a database:

DROP DATABASE <database_name>;

For example,

```
mysql> DROP DATABASE School;
Database deleted
```

4. Creating a Table

The **CREATE TABLE** statement is used to create a table in a database. Tables are organized into rows and columns, and each table must have a name. It is the most extensively used DDL command. A table must have at least one column.

Syntax for creating a table:

CREATE TABLE *<table_name>*
(
<column_name1><data_type> [(size)],
<column_name2><data_type> [(size)],
<column_name3><data_type> [(size)],
....
);

For example,

```
mysql> CREATE TABLE Student
(   Rollno      integer         NOT NULL PRIMARY KEY,
    Name        varchar(20)     NOT NULL,
    Gender      char(1),
    Marks       integer(11)
    DOB         date );
Query OK, 0 rows affected (0.04 sec)
```

For each column, a name and a data type must be specified and the column name must be unique within the table definition. Column definitions are separated by comma. Upper case and lower case letters make no difference in column names.

5. Viewing a Table

To verify that the table has been created in the database, **SHOW TABLES** command is used.

```
mysql> SHOW TABLES;
+----------------------+
| Tables_in_school     |
+----------------------+
| Student              |
+----------------------+
2 rows in set (0.01 sec)
```

6. Viewing a Table Structure

To view a table structure, **DESCRIBE** or **DESC** command is used. It shows the structure of the table along with the name of the columns, data type of the columns and constraints applied on the columns.

Syntax: DESCRIBE <tablename>; or DESC <tablename>;

For example,

```
mysql> DESCRIBE Student;
```

Field	Type	Null	Key	Default	Extra
Rollno	integer(11)	NO	PRI	NULL	
Name	varchar(20)	NO		NULL	
Gender	char(1)	YES		NULL	
Marks	number(11)	YES		NULL	
DOB	date	YES		NULL	

5 rows in set (0.02 sec)

7. ALTER TABLE Command

The ALTER TABLE command is used to modify the definition (structure) of a table by modifying the definition of its columns. The ALTER TABLE command is used to perform the following operations:

- To add a column to an existing table.
- To rename any existing column.
- To change the data type of any column or to modify its size.
- To remove or physically delete a column.

(a) **Adding a column to an existing table:**

Once a table has been created, new columns can be added later on, if required. The new column is added with NULL values for all the records/rows in the table. It is possible to add, delete and modify columns with ALTER TABLE statement.

Syntax for adding a new column:

ALTER TABLE <table_name> ADD(<column_name><data type> [size]);

For example, to add a new column, Mobile_no, of type integer in the table student:

```
mysql> ALTER TABLE Student ADD(Mobile_no integer);
```

Thus, the above statement shall add a new column Mobile_no into the table student with NULL value in it.

POINT TO REMEMBER

We have just added a column and there will be no data (NULL) under this attribute. UPDATE command can be used to supply values/data to this column.

(b) **Adding a column with default value**

ALTER TABLE command can be used to add a new column to an existing table with default values.

Syntax for adding a column with a default value:

ALTER TABLE <table_name>

ADD ([column_name1]<data type1>default data);

For example,

```
mysql> ALTER TABLE Student ADD(City char(6) default "DELHI");
```

The above command will add a new column City with default value as "DELHI" to the student table.

Resultant Table: Student

Rollno	Name	Gender	Marks	DOB	Mobile_no	City
1	Raj Kumar	M	93	2000-11-17	NULL	DELHI
2	Deep Singh	M	98	1996-08-22	NULL	DELHI
3	Ankit Sharma	M	76	2000-02-02	NULL	DELHI
4	Radhika Gupta	F	78	1999-12-03	NULL	DELHI
5	Payal Goel	F	82	1998-04-21	NULL	DELHI
6	Diksha Sharma	F	80	1999-12-17	NULL	DELHI
7	Gurpreet Kaur	F	65	2000-01-04	NULL	DELHI
8	Akshay Dureja	M	90	1997-05-05	NULL	DELHI
9	Shreya Anand	F	70	1999-10-08	NULL	DELHI
10	Prateek Mittal	M	75	2000-12-25	NULL	DELHI

(c) **Modifying an existing column definition**

The MODIFY clause can be used with ALTER TABLE command to change the data type, size, constraint related to any column of the table.

Syntax for modifying existing column data type:

ALTER TABLE <table_name>

MODIFY([column_name1] <data type1>);

For example,

```
mysql> ALTER TABLE Student MODIFY Name varchar(25);
```

The above command will modify the data type size for the Name field from 20 to 25 characters.

(d) **Renaming a column**

The existing column in a relation can be renamed using ALTER TABLE command.

Syntax for renaming an existing column:

ALTER TABLE <table_name>

CHANGE[COLUMN]<old-column-name><new-column-name>column_definition;

For example,

```
mysql> ALTER TABLE Student CHANGE City State varchar(10);
```

The above command shall rename the City column to State.

Resultant Table: Student

Rollno	Name	Gender	Marks	DOB	Mobile_no	State
1	Raj Kumar	M	93	2000-11-17	NULL	DELHI
2	Deep Singh	M	98	1996-08-22	NULL	DELHI
3	Ankit Sharma	M	76	2000-02-02	NULL	DELHI
4	Radhika Gupta	F	78	1999-12-03	NULL	DELHI
5	Payal Goel	F	82	1998-04-21	NULL	DELHI
6	Diksha Sharma	F	80	1999-12-17	NULL	DELHI
7	Gurpreet Kaur	F	65	2000-01-04	NULL	DELHI
8	Akshay Dureja	M	90	1997-05-05	NULL	DELHI
9	Shreya Anand	F	70	1999-10-08	NULL	DELHI
10	Prateek Mittal	M	75	2000-12-25	NULL	DELHI

(e) Removing a column

To remove or drop a column in a table, ALTER TABLE command is used.

Syntax for removing a column:

ALTER TABLE <table-name> DROP <column-name>;

For example,

```
mysql> ALTER TABLE Student DROP (State);
```

The above command will drop State column from the student table.

Resultant Table: Student

Rollno	Name	Gender	Marks	DOB	Mobile_no
1	Raj Kumar	M	93	2000-11-17	NULL
2	Deep Singh	M	98	1996-08-22	NULL
3	Ankit Sharma	M	76	2000-02-02	NULL
4	Radhika Gupta	F	78	1999-12-03	NULL
5	Payal Goel	F	82	1998-04-21	NULL
6	Diksha Sharma	F	80	1999-12-17	NULL
7	Gurpreet Kaur	F	65	2000-01-04	NULL
8	Akshay Dureja	M	90	1997-05-05	NULL
9	Shreya Anand	F	70	1999-10-08	NULL
10	Prateek Mittal	M	75	2000-12-25	NULL

(f) Adding and Deleting Primary Key constraints

We can add or delete a Primary Key constraint even if the table has already been created using ALTER TABLE command.

For example, add a Primary Key constraint on the column Emp_ID in the table Employee.

```
mysql> ALTER TABLE EMPLOYEE ADD PRIMARY KEY(Emp_ID);
```

Note: Make sure while using ALTER TABLE command to add a primary key that the primary key column(s) must be declared with NOT NULL constraint (when the table was first created).

- To delete a Primary Key constraint from the table Employee:

```
mysql> ALTER TABLE EMPLOYEE DROP PRIMARY KEY;
```

- NULL values are stored and displayed as NULL only without any quotes.
- If the data is not available for all the columns, then the column-list must be included following the table name.

CTM: In SQL, we can repeat or re-execute the last command typed at SQL prompt by pressing up and down arrow keys followed by Enter key.

(b) **Inserting data by specifying all the column names and associated values into a table:** The second form specifies both the column names and the values to be inserted.

Syntax: **INSERT INTO <table_name> (column1,column2,columnN,...)**
VALUES (value1,value2,valueN,...);

Here, column1, column2, ...columnN—the names of the columns in the table for which you want to insert data.

For example, mysql> `INSERT INTO Student(RollNo, Name, Gender, Marks, DOB)`
`VALUES(2,'Deep Singh', 'M', 98, '1996-08-22');`

CTM: When adding a row, only the characters or date values should be enclosed within single quotes.

(c) **Inserting data into specific columns of a table:**

Syntax for SQL INSERT is:

INSERT INTO <table_name>[(column1, column2, ... columnN)]

VALUES [(value1, value2,.... valueN)];

For example, to insert a record into the student table for the columns Rollno, Name and Marks only, the SQL insert query is:

```
mysql> INSERT INTO Student (Rollno, Name, Marks)
VALUES (4,"Radhika Gupta",78);
```

The above statement shall insert the values for specific columns—Rollno, Name and Marks respectively. Also, the columns Gender and DOB shall hold the values as NULL.

(d) **Inserting NULL values into a table:** If a column in a row has no value or missing value, then the column is said to be null or holding NULL value. Null value can be given to any column other than being assigned as primary key or Not Null constraint. It is advisable to use Null when the actual value is not defined or unavailable. NULL values are treated differently from other values as they represent missing unknown data. By default, a column in a table can hold NULL values.

If a column in a table is optional, we can insert a new record or can modify an existing tuple without adding values to this column. In other words, the values in every record for this column/field shall be stored as NULL. We can insert NULL value into any column in a table. It can be done by typing NULL without quotes.

Null is not equivalent to 0, *i.e.*, NULL ≠ 0. It acts as a placeholder for unknown or inapplicable values.

For example, mysql> `INSERT INTO Student(Rollno, Name, Gender, Marks, DOB)`
`VALUES(12, 'Swati Mehra', 'F', NULL, NULL);`

After the execution of the above command, NULL values shall be inserted for the fields Marks and DOB respectively.

CTM: Null means unavailable or undefined value. Any arithmetic expression containing a NULL always evaluates to null.

2. Modifying Data in a Table

To modify data in a table or to make changes for some or all of the values in the existing records in a table, we use the UPDATE statement. The UPDATE command specifies the rows to be modified using the WHERE clause and the new data is written into the respective record using the SET keyword.

Syntax for UPDATE:

UPDATE <table_name>
SET <column1> = <value1>, <column2> = <value2>,.....
WHERE <condition>;

For example,
```
mysql> UPDATE Student
       SET Marks = 90
       WHERE Rollno = 8;
```

The above statement shall change the value of Marks field to 90 for the student whose roll number is 8.

(a) **Updating multiple columns**

Modifying the values in more than one column can be done by separating the columns along with the new values using SET clause, separated by commas.

For example,
```
mysql> UPDATE Student
       SET Marks = 70, DOB='1998-08-11'
       WHERE Name="Payal";
```

The above statement shall change the values for both the fields Marks and DOB to 70 and '1998-08-11' respectively for the student whose name is Payal.

(b) **Updating to NULL values**

The values for the attributes in a relation can also be entered as NULL using UPDATE command.

For example,
```
mysql> UPDATE Student
       SET Marks = NULL
       WHERE Rollno = 9;
```

The above statement shall change the value of the field Marks to Null for the student whose roll number is 9.

(c) **Updating using an expression or formula**

For example,
```
mysql> UPDATE Student
       SET Marks = Marks + 10
       WHERE (Rollno = 5 or Rollno =10);
```

The above statement shall increment the value of Marks by 10 for all the records with Roll number 5 or 10.

(d) **Updating all rows using an expression or formula**

For example, mysql> `UPDATE Student`

`SET Marks = Marks + 10;`

The above statement shall increase the value of Marks of all the students by 10.

3. Removing Data from a Table

The **DELETE** statement is used to delete rows from a table.

Syntax for DELETE Statement:

DELETE FROM <table_name> WHERE <condition>;

here <table_name> is the table whose records are to be deleted.

POINT TO REMEMBER

The WHERE clause in the SQL delete command is optional and it identifies the rows in the column that get deleted. If you do not include the WHERE clause, all the rows in the table are deleted.

For example, mysql> `DELETE FROM Student WHERE Rollno = 10;`

The above statement shall delete the record only for roll number 10.

- Deleting all rows from the table:

To delete all the rows from the student table, the DELETE statement will be:

mysql> `DELETE FROM Student;`

- SQL TRUNCATE Statement

The **SQL TRUNCATE** command is used to delete all the rows from the table and free the space containing the table.

Syntax to TRUNCATE a table:

TRUNCATE TABLE <table_name>;

For example,

To delete all the rows from student table, the statement will be:

mysql> `TRUNCATE TABLE Student;`

Difference between DELETE and TRUNCATE Statements

DELETE Statement: This command deletes only the rows from the table based on the condition given in the where clause or deletes all the rows from the table if no condition is specified. But it does not free the space containing the table.

TRUNCATE Statement: This command is used to delete all the rows from the table and free the space containing the table.

7.15 SQL QUERY PROCESSING

After creating the database, the table is created and the data is stored in it. Now, it is time to perform query processing on the already-created tables to retrieve and view the data on the screen. Retrieving information from the tables is done mainly using the SELECT command. The SQL SELECT statement is used to fetch data from one or more database tables. It is used to select rows and columns from a database/relation.

7.15.1 SQL SELECT Statement

This command can perform selection as well as projection. It is the most extensively used SQL command. The SELECT statement can be used to retrieve a subset of rows or columns from one or more tables present in a database.

1. Projection

This capability of SQL returns all rows from a relation with selective attributes.

Syntax:

SELECT <column-name1> [, <column-name2>...]
FROM <table-name>;

For example,

```
mysql> SELECT Name, Gender FROM Student;
```

Resultant table: Student

Name	Gender
Raj Kumar	M
Deep Singh	M
Ankit Sharma	M
Radhika Gupta	F
Payal Goel	F
Diksha Sharma	F
Gurpreet Kaur	F
Akshay Dureja	M
Shreya Anand	F
Prateek Mittal	M

10 rows in a set (0.01 sec)

The above command displays only Name and Gender attributes from the student table.

CTM: The asterisk (*) means "All". SELECT * means displaying all the columns from a relation.

To display all columns along with the respective rows, we use the command:

```
mysql> SELECT * FROM Student;
```

Resultant table: Student

Rollno	Name	Gender	Marks	DOB	Mobile_no
1	Raj Kumar	M	93	2000-11-17	NULL
2	Deep Singh	M	98	1996-08-22	NULL
3	Ankit Sharma	M	76	2000-02-02	NULL
4	Radhika Gupta	F	78	1999-12-03	NULL
5	Payal Goel	F	82	1998-04-21	NULL
6	Diksha Sharma	F	80	1999-12-17	NULL
7	Gurpreet Kaur	F	65	2000-01-04	NULL
8	Akshay Dureja	M	90	1997-05-05	NULL
9	Shreya Anand	F	70	1999-10-08	NULL
10	Prateek Mittal	M	75	2000-12-25	NULL

10 rows in a set (0.02 sec)

The above command displays all the rows of all the columns according to the column-list defined in the table structure. The salient features of SQL SELECT statement are as follows:

- SELECT command displays the columns of the table in the same order in which they are selected from the table.
- To retrieve all the columns in the column-list from a table using SELECT command, asterisk (*) is used and the columns are displayed in the same order in which they are stored in the table.
- All the statements (inclusive of SELECT statement) in SQL are terminated with a semicolon (;). Use of semicolon is dependent on the version in use.

2. Selection: Selecting Specific Rows—WHERE Clause

This capability of SQL returns some of the rows and all the columns in the relation. Use of WHERE clause is required when specific tuples are to be fetched or manipulated. To select all the columns from a table, the asterisk (*) can be used.

SELECT <what_to_select>
FROM <which_table>
WHERE <conditions_to_satisfy>;

For example,

```
mysql> SELECT * FROM Student WHERE Gender = 'M';
```

Resultant table: Student

Rollno	Name	Gender	Marks	DOB	Mobile_no
1	Raj Kumar	M	93	2000-11-17	NULL
2	Deep Singh	M	98	1996-08-22	NULL
3	Ankit Sharma	M	76	2000-02-02	NULL
8	Akshay Dureja	M	90	1997-05-05	NULL
10	Prateek Mittal	M	75	2000-12-25	NULL

5 rows in a set (0.01 sec)

The above command will display all attributes but specific tuples (rows) which satisfy the condition in the Student table.

Using WHERE clause

```
mysql> SELECT * FROM Student WHERE Rollno<=8;
```

The above command shall display only those records whose Rollno is less than or equal to 8.

Resultant table: Student

Rollno	Name	Gender	Marks	DOB	Mobile_no
1	Raj Kumar	M	93	2000-11-17	NULL
2	Deep Singh	M	98	1996-08-22	NULL
3	Ankit Sharma	M	76	2000-02-02	NULL
4	Radhika Gupta	F	78	1999-12-03	NULL
5	Payal Goel	F	82	1998-04-21	NULL
6	Diksha Sharma	F	80	1999-12-17	NULL
7	Gurpreet Kaur	F	65	2000-01-04	NULL
8	Akshay Dureja	M	90	1997-05-05	NULL

8 rows in a set (0.02 sec)

When a WHERE clause is used with a SELECT statement, the SQL query processor goes through the entire table one row/record at a time and checks each row to determine whether the condition specified is true with respect to that row or not. If it evaluates to True, the corresponding row is selected, retrieved and displayed, else it returns an empty set (*i.e.*, no data found).

CTM: SQL is case-insensitive, which means keywords like SELECT and select have same meaning in SQL statements.

3. Re-ordering Columns while Displaying Query Results

While displaying the result for a query, the order of the columns to be displayed can be changed according to the user's requirement. But this is done only for the display purpose and no actual (physical) rearrangement of the columns is done.

For example,

```
mysql> SELECT Name, Rollno, DOB, Marks FROM Student;
```

After executing the above statement, the column shall be displayed in the changed order as Name shall be displayed as the first column, Rollno as the second column, DOB as the third column and Marks as the fourth column respectively.

Resultant table: Student

Name	Rollno	DOB	Marks
Raj Kumar	1	2000-11-17	93
Deep Singh	2	1996-08-22	98
Ankit Sharma	3	2000-02-02	76
Radhika Gupta	4	1999-12-03	78
Payal Goel	5	1998-04-21	82
Diksha Sharma	6	1999-12-17	80
Gurpreet Kaur	7	2000-01-04	65
Akshay Dureja	8	1997-05-05	90
Shreya Anand	9	1999-10-08	70
Prateek Mittal	10	2000-12-25	75

10 rows in a set (0.02 sec)

CTM: The order in which the columns are displayed using the SELECT command is in accordance with the order in which they are actually stored in the table.

4. Eliminating Duplicate/Redundant Data—DISTINCT clause

DISTINCT clause is used to remove duplicate rows from the results of a SELECT statement. It is used to retrieve only unique values for a column in the table. The DISTINCT keyword can be used only once with a given SELECT statement.

Syntax: SELECT DISTINCT <column-name> from <table-name>;

For example,

Suppose we have added a new column Stream to the table student.

Resultant table: Student

Rollno	Name	Gender	Marks	DOB	Mobile_no	Stream
1	Raj Kumar	M	93	2000-11-17	9586774748	Science
2	Deep Singh	M	98	1996-08-22	8988886577	Commerce
3	Ankit Sharma	M	76	2000-02-02	NULL	Science
4	Radhika Gupta	F	78	1999-12-03	9818675444	Humanities
5	Payal Goel	F	82	1998-04-21	9845639990	Vocational
6	Diksha Sharma	F	80	1999-12-17	9897666650	Humanities
7	Gurpreet Kaur	F	65	2000-01-04	7560567890	Science
8	Akshay Dureja	M	90	1997-05-05	9560567890	Commerce
9	Shreya Anand	F	70	1999-10-08	NULL	Vocational
10	Prateek Mittal	M	75	2000-12-25	9999967543	Science

10 rows in a set (0.02 sec)

With reference to the above table, if we write the SELECT statement as:

```
mysql> SELECT Stream FROM Student;
```

this statement shall return all the tuples for field Stream from table student. It will return duplicate values also. Thus, in order to remove these duplicate values, DISTINCT clause is used.

Now, we write a query for displaying the distinct contents on the basis of the field Stream from student table:

For example,

```
mysql> SELECT DISTINCT Stream FROM Student;
```

Resultant table: Student

Stream
Science
Commerce
Science
Humanities
Vocational
Humanities
Science
Commerce
Vocational
Science

Science displayed 4 times

(Displays all rows with duplicate values as well)

10 rows in a set (0.02 sec)

Resultant table: Student

Stream
Science
Commerce
Humanities
Vocational

Science displayed once only

(Duplicate values are removed)

4 rows in a set (0.02 sec)

7.15.2 SQL Operators

While working with SELECT statement using WHERE clause, condition-based query can be carried out using four types of SQL operators:

(a) Arithmetic Operators
(b) Relational Operators
(c) Logical Operators
(d) Special Operators

```
mysql> SELECT 5 * 4 FROM DUAL;
+---------------+
| 5 * 4         |
+---------------+
| 20            |
+---------------+
mysql> SELECT 47 % 5 FROM DUAL;
+---------------+
| 47 % 5        |
+---------------+
| 2             |
+---------------+
```

The modulus (%) operator returns the remainder as the answer after performing the division operation. Hence, the above statement shall return the value 2 as the output.

POINT TO REMEMBER

DUAL is the default table automatically created in MySQL. It has one row with a value X and one column 'dummy' defined as varchar2(1).

Evaluating Scalar expression with SELECT statement

MySQL permits calculations on the contents of the columns and then displays the calculated result using SELECT statement. We can write scalar expression and constant values for the selected columns. If we are taking NULL value in the expression, it shall result in NULL only. Along with NULL, arithmetic operators can be used while evaluating scalar expressions.

For example, `mysql> SELECT Rollno, Name, Marks + 10 FROM Student;`

The above command, on execution, shall increment the value for all the rows of the field Marks by 10 and shall display the Rollno, Name and Marks for all the students, increased by 10.

Resultant table: Student

Rollno	Name	Marks + 10
1	Raj Kumar	103
2	Deep Singh	108
3	Ankit Sharma	86
4	Radhika Gupta	88
5	Payal Goel	92
6	Diksha Sharma	90
7	Gurpreet Kaur	75
8	Akshay Dureja	100
9	Shreya Anand	80
10	Prateek Mittal	85

10 rows in a set (0.02 sec)

(b) **Relational Operators**

A relational (comparison) operator is a mathematical symbol which is used to compare tw values. It is used to compare two values of the same or compatible data types. Compariso operators are used for conditions where two expressions are required to be compared wit each other, which results in either true or false. They are used with WHERE clause.

The following table describes different types of comparison operators in SQL:

OPERATOR	DESCRIPTION
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>, !=	Not equal to

For comparing character data type values, < means earlier in the alphabetical sequence and > means later in the alphabetical sequence.

For example, mysql> `SELECT Rollno, Name, Marks FROM Student where Marks>=90;`

The above command shall display the Rollno, Name and Marks of all the students with marks either equal to or greater than 90.

Resultant table: Student

Rollno	Name	Marks
1	Raj Kumar	93
2	Deep Singh	98
3	Akshay Dureja	90

3 rows in a set (0.02 sec)

For example, mysql> `SELECT * FROM Student`
`WHERE Stream <> 'Commerce';`

The above command shall display the records of all the students who are not from Commerce stream.

Resultant table: Student

Rollno	Name	Gender	Marks	DOB	Mobile_no	Stream
1	Raj Kumar	M	93	2000-11-17	9586774748	Science
3	Ankit Sharma	M	76	2000-02-02	8567490078	Science
4	Radhika Gupta	F	78	1999-12-03	9818675444	Humanities
5	Payal Goel	F	82	1998-04-21	9845639990	Vocational
6	Diksha Sharma	F	80	1999-12-17	9897666650	Humanities
7	Gurpreet Kaur	F	65	2000-01-04	7560567890	Science
9	Shreya Anand	F	70	1999-10-08	8876543988	Vocational
10	Prateek Mittal	M	75	2000-12-25	9999967543	Science

8 rows in a set (0.02 sec)

Thus, while using relational operators in a WHERE clause with a SELECT statement, the database program goes through the entire table checking each record one by one and compares with the condition specified. If it is true, the corresponding row is selected for display, otherwise it is ignored.

CTM: While comparing character, date and time data using relational operators, it should be enclosed in single quotation marks.

(c) Logical Operators

The SQL logical operators are the operators used to combine multiple conditions to narrow the data selected and displayed on the basis of the condition specified in an SQL statement. Logical operators are also known as Boolean operators. The three logical operators in SQL are—AND, OR and NOT operator. Out of these, AND and OR operators are termed as Conjunctive operators since these two operators combine two or more conditions. The AND and OR operators are used to filter records based on more than one condition.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

CTM: The order of precedence for logical operators (AND, OR, NOT operator) is NOT(!), AND(&&) and OR(||).

1. AND operator

The AND operator displays a record and returns a true value if all the conditions (usually two conditions) specified in the WHERE clause are true.

Condition 1	Condition 2	Result (AND operation)
True	True	True
True	False	False
False	True	False
False	False	False

As shown in the table, when both condition 1 and condition 2 are true, then only is the result true. If either of them is false, the result becomes false.

For example, to list the details of all the students who have secured more than 80 marks and are male.

```
mysql> SELECT * FROM Student
       WHERE Marks > 80 AND Gender= 'M';
```

Resultant table: student

RollNo	Name	Gender	Marks	DOB	Mobile no	Stream
1	Raj Kumar	M	93	2000-11-17	9586774748	Science
2	Deep Singh	M	98	2000-08-22	8988886577	Commerce
8	Akshay Dureja	M	90	1997-05-05	9560567890	Commerce

3 rows in a set (0.02 sec)

2. OR operator

The OR operator displays a record and returns a true value if either of the conditions (usually two conditions) specified in the WHERE clause is true.

Condition 1	Condition 2	Result (OR operation)
True	True	True
True	False	True
False	True	True
False	False	False

As shown in the table, when either condition 1 or condition 2 is true, the result is tru
If both of them are false, then only the result becomes false.

For example, to display the roll number, name and stream of all the students who ar
in either Science or Commerce stream.

```
mysql> SELECT Rollno, Name, Stream FROM Student WHERE
       Stream= 'Science' OR Stream= 'Commerce';
```

Resultant table: Student

Rollno	Name	Stream
1	Raj Kumar	Science
2	Deep Singh	Commerce
3	Ankit Sharma	Science
7	Gurpreet Kaur	Science
8	Akshay Dureja	Commerce
10	Prateek Mittal	Science

6 rows in a set (0.02 sec)

3. **NOT operator**

NOT operator is also termed as a negation operator. Unlike the other two operators
this operator takes only one condition and gives the reverse of it as the result. I
returns a false value if the condition holds true and vice versa.

The NOT operator displays a record and returns a true value if either of the conditions
(usually two conditions) specified in the WHERE clause is true.

Condition 1	Result (NOT operation)
True	False
False	True

As shown in the table, when the condition is true, the result is false. If the condition is
false, then the result becomes true.

For example, to display the name and marks of all the students who are not in the
vocational stream.

```
mysql> SELECT Name, Marks FROM Student
       WHERE NOT (Stream = 'Vocational');
```

Resultant table: Student

Name	Marks
Raj Kumar	93
Deep Singh	98
Ankit Sharma	76
Radhika Gupta	78
Diksha Sharma	80
Gurpreet Kaur	65
Akshay Dureja	90
Prateek Mittal	75

8 rows in a set (0.02 sec)

(d) **SQL Special Operators**

Apart from several standard library functions discussed earlier, there are some special operators in SQL that perform some specific functions.

1. **Conditions Based on a Range—BETWEEN...AND**

SQL provides a BETWEEN operator that defines a range of values that the column value must fall within for the condition to become true. The range includes both the lower and upper value. The values can be numbers, text or dates.

Syntax for BETWEEN:

```
mysql> SELECT <column_name(s)>
       FROM <table_name>
       WHERE <column_name> BETWEEN <value1> AND <value2>;
```

For example,

```
mysql> SELECT Rollno, Name, Marks FROM Student WHERE Marks BETWEEN
       80 AND 100;
```

The above command displays Rollno, Name along with Marks of those students whose Marks lie in the range of 80 to 100 (both 80 and 100 are included in the range).

Resultant table: Student

Rollno	Name	Marks
1	Raj Kumar	93
2	Deep Singh	98
5	Payal Goel	82
6	Diksha Sharma	80
8	Akshay Dureja	90

5 rows in a set (0.02 sec) NOT BETWEEN

The NOT BETWEEN operator works opposite to the BETWEEN operator. It retrieves the rows which do not satisfy the BETWEEN condition.

For example,

```
mysql> SELECT Rollno, Name, Marks FROM Student WHERE Marks NOT
       BETWEEN 80 AND 100;
```

Resultant table: Student

Rollno	Name	Marks
3	Ankit Sharma	76
4	Radhika Gupta	78
7	Gurpreet Kaur	65
9	Shreya Anand	70
10	Prateek Mittal	75

5 rows in a set (0.02 sec)

2. Conditions Based on a List—IN

To specify a list of values, IN operator is used. This operator selects values that match any value in the given list. The SQL IN condition is used to help reduce the need for multiple **OR conditions** in a SELECT statement.

Syntax for IN:

SELECT <column_name(s)>

FROM <table_name>

WHERE <column_name> IN (value1,value2,...);

For example,

```
mysql> SELECT * FROM Student WHERE Stream IN ('Science', 'Commerce',
       'Humanities');
```

Resultant table: Student

Rollno	Name	Gender	Marks	DOB	Mobile_no	Stream
1	Raj Kumar	M	93	2000-11-17	9586774748	Science
2	Deep Singh	M	98	1996-08-22	8988886577	Commerce
3	Ankit Sharma	M	76	2000-02-02	NULL	Science
4	Radhika Gupta	F	78	1996-12-03	9818675444	Humanities
6	Diksha Sharma	F	80	1999-12-17	9897666650	Humanities
7	Gurpreet Kaur	F	65	2000-01-04	7560875609	Science
8	Akshay Dureja	M	90	1997-05-05	9560567890	Commerce
10	Prateek Mittal	M	75	2000-12-25	9999967543	Science

8 rows in a set (0.02 sec)

The above command displays all those records whose Stream is either Science or Commerce or Humanities.

NOT IN

The NOT IN operator works opposite to IN operator. It returns the rows that do not match the list.

For example,

```
mysql> SELECT * FROM Student WHERE Stream NOT IN ('Science',
       'Commerce', 'Humanities');
```

Resultant table: student

Rollno	Name	Gender	Marks	DOB	Mobile_no	Stream
5	Payal Goel	F	82	1998-04-21	9845639990	Vocational
9	Shreya Anand	F	70	1999-10-08	NULL	Vocational

2 rows in a set (0.02 sec)

3. Conditions Based on Pattern—LIKE

The LIKE operator is used to search for a specified pattern in a column. This operator is used with the columns of type CHAR and VARCHAR. The LIKE operator searches the column to find if a part of this column matches the string specified in the parentheses after the LIKE operator in the command.

Conditions Based on Pattern—WILD CARD CHARACTERS

The SQL LIKE condition allows you to use wild cards to perform pattern matching. SQL provides two wild card characters that are used while comparing the strings with LIKE operator:

a. **Percent(%)** Matches any string

b. **Underscore(_)** Matches any one character

Syntax for LIKE:

SELECT <column_name(s)>
FROM <table_name>
WHERE <column_name> LIKE <pattern>;

For example,

```
mysql> SELECT * FROM Student WHERE Name LIKE "D%";
```

Resultant table: Student

Rollno	Name	Gender	Marks	DOB	Mobile_no	Stream
2	Deep Singh	M	98	1996-08-22	8988886577	Commerce
6	Diksha Sharma	F	80	1999-12-17	9897666650	Humanities

2 rows in a set (0.02 sec)

The above command shall display those records where the name begins with character 'D'.

```
mysql> SELECT * FROM Student WHERE Name LIKE "%a";
```

Resultant table: Student

Rollno	Name	Gender	Marks	Mobile_no	Stream
3	Ankit Sharma	M	76	NULL	Science
4	Radhika Gupta	F	78	9818675444	Humanities
6	Diksha Sharma	F	80	9897666650	Humanities
8	Akshay Dureja	M	90	9560567890	Commerce

4 rows in a set (0.02 sec)

The above command shall display the records for those students whose name ends with the letter 'a'.

```
mysql> SELECT * FROM Student WHERE Name LIKE "%e%";
```

Resultant table: Student

Rollno	Name	Gender	Marks	DOB	Mobile_no	Stream
2	Deep Singh	M	98	1996-08-22	8988886577	Commerce
5	Payal Goel	F	82	1998-04-21	9845639990	Vocational
7	Gurpreet Kaur	F	65	2000-01-04	7560875609	Science
8	Akshay Dureja	M	90	1997-05-05	9560567890	Commerce
9	Shreya Anand	F	70	1999-10-08	NULL	Vocational
10	Prateek Mittal	M	75	2000-12-25	9999967543	Science

6 rows in a set (0.02 sec)

As the resultant table shows, the above command displays the records of all the students whose Name contains the character 'e' anywhere in it.

```
mysql> SELECT * From Student WHERE Name LIKE "_e%";
```

Resultant table: Student

Rollno	Name	DOB
2	Deep Singh	1996-08-22

1 row in a set (0.02 sec)

This command shall display the Rollno, Name and DOB of all the students whose Name contains the letter 'e' at the second place.

NOT LIKE

The NOT LIKE operator works opposite to LIKE operator. It returns the rows that do not match the specified pattern.

For example,

```
mysql> SELECT Rollno, Name, Marks, DOB FROM Student WHERE Name
       NOT LIKE "%r_ _";
```

Resultant table: Student

Rollno	Name	Gender	Marks	DOB
1	Raj Kumar	M	93	2000-11-17
2	Deep Singh	M	98	1996-08-22
4	Radhika Gupta	F	78	1999-12-03
5	Payal Goel	F	82	1998-04-21
7	Gurpreet Kaur	F	65	2000-01-04
8	Akshay Dureja	M	90	1997-05-05
9	Shreya Anand	F	70	1999-10-08
10	Prateek Mittal	M	75	2000-12-25

8 rows in a set (0.02 sec)

This command shall display the Rollno, Name, Marks and DOB of all the students whose Name does not contain the letter 'r' from the third last position.

7.15.3 Comments in SQL

A comment is a text which is ignored by the SQL compiler and is not executed at all. It is given for documentation purpose only. A comment usually describes the purpose of the statement given within an application.

SQL Comments are used to understand the functionality of the program without looking into it. The comments give us an idea about what is written in the given SQL statement and how it works. The comments can make an application or code easier to read as well as to maintain. A comment can be placed between any keywords, parameters or punctuation marks in a statement. Comments can be either single-line comments or multiple-line comments.

SQL or MySQL supports three comment styles:

- **Comments beginning with -- (followed by a space):** The two dash lines indicate a single-line comment in SQL statements. These single-line comments are basically used to show the comments at the start and end of a program. A user can easily use this comment type to explain the flow of program. This text cannot extend to a new line and ends with a line break.
- **Comments beginning with #:** The comments begin'with '#' symbol followed by the text to be displayed for the user's information. This text cannot extend to a new line and ends with a line break.
- **Comments beginning with /*:** Multi-line comments begin with a slash and an asterisk (/*) followed by the text of the comment. This text can span multiple lines. The comment ends with an asterisk and a slash (*/). The opening and terminating characters need not be separated from the text by a space or a line break.

For example,

```
mysql> SELECT Rollno, Name, Stream
    -> /* This statement shall display the records of all those students
       who are in Science stream and have secured marks more than 75. */
    -> FROM Student # student table in use
    -> WHERE Stream= 'Science' and Marks > 75;--condition for selection
```

7.16 SQL ALIASES

SQL aliases are used to give an alternate name, *i.e.*, a temporary name, to a database table or a column in a table. We can rename a table or a column temporarily by giving another name called alias name which leads to a temporary change (renaming) and does not change the actual name in the database.

Aliases can be used when

- more than one table is involved in a query.
- functions are used in the query.
- column names are big or not very readable.
- two or more columns are combined together.

Using column alias name, we can give different name(s) to column(s) for display (output) purpose only. They are created to make column names more readable. SQL aliases can be used both for tables as well as columns.

- **COLUMN ALIASES** are used to make column headings in the query result set easier to read.
- **TABLE ALIASES** are used to shorten a table name by giving an easy alternate name, making it easier to read or when performing a self-join (*i.e.*, listing the same table more than once in the FROM clause).

Syntax for table alias:

```
SELECT <columnname1>, <columnname2>.....
FROM <table_name> AS <alias_name>;
WHERE [<condition>];
```

Syntax for column alias:

```
SELECT <column-name> AS <"alias_name">
FROM <table_name>
WHERE [<condition>];
```

For example,

Table: Student

Student_name	Date_of_birth
Raj Kumar	2000-11-17
Deep Singh	1996-08-22
Ankit Sharma	2000-02-02
Radhika Gupta	1999-12-03
Payal Goel	1998-04-21
Diksha Sharma	1999-12-17
Gurpreet Kaur	2000-01-04
Akshay Dureja	1997-05-05
Shreya Anand	1999-10-08
Prateek Mittal	2000-12-25

```
mysql> SELECT Name AS "Student_name", DOB AS "Date_of_birth"
    -> FROM Student;
```

Alias name for fields, Name and DOB

10 rows in a set (0.02 sec)

- **Alias name can be given to a mathematical expression also:**

For example,

```
mysql> SELECT 22/7 AS PI;
```

Output:

```
+------------+
|     PI     |
+------------+
|  3.1429    |
+------------+
```

POINTS TO REMEMBER

- If the **alias_name** contains spaces, you must enclose it in quotes.
- It is acceptable to use spaces when you are aliasing a column name. However, it is not generally a good practice to use spaces when you are aliasing a table name.
- The **alias_name** is only valid within the scope of the SQL statement.

7.17 PUTTING TEXT IN THE QUERY OUTPUT

In order to get an organized output from a SELECT query, we can include some user-defined columns at runtime. These columns are displayed with a valid text, symbols and comments in the output only. These included columns will appear as column heads along with the contents for that column.

This makes the query output more presentable by giving a formatted output.

For example,

```
mysql> SELECT Rollno, Name, 'was born on', DOB
    -> FROM Student;
```

Text to be displayed

The above command, on execution, shall display the text "was born on" with every record (tuple) of the table.

Resultant table: student

Rollno	Name	was born on	DOB
1	Raj Kumar	was born on	2000-11-17
2	Deep Singh	was born on	1996-08-22
3	Ankit Sharma	was born on	2000-02-02
4	Radhika Gupta	was born on	1999-12-03
5	Payal Goel	was born on	1998-04-21
6	Diksha Sharma	was born on	1999-12-17
7	Gurpreet Kaur	was born on	2000-01-04
8	Akshay Dureja	was born on	1997-05-05
9	Shreya Anand	was born on	1999-10-08
10	Prateek Mittal	was born on	2000-12-25

10 rows in a set (0.02 sec)

7.18 SORTING IN SQL—ORDER BY

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order based on one or more columns. The ORDER BY keyword is used to sort the result-set by one or more fields in a table. This clause sorts the records in the ascending order (ASC) by default. Therefore, in order to sort the records in descending order, DESC keyword is to be used. Sorting using ORDER BY clause can be done on multiple columns, separated by comma.

Syntax for ORDER BY clause:

SELECT <column-list> FROM <table_name> [WHERE <condition>] ORDER BY <column_name> [ASC|DESC];

Here, WHERE clause is optional.

For example,

- **To display the roll number, name and marks of students on the basis of their marks in the ascending order.**

```
mysql> SELECT Rollno, Name, Marks FROM
Student ORDER BY Name;
```

Rollno	Name	Marks
8	Akshay Dureja	90
3	Ankit Sharma	76
2	Deep Singh	98
6	Diksha Sharma	80
7	Gurpreet Kaur	65
5	Payal Goel	82
10	Prateek Mittal	75
4	Radhika Gupta	78
1	Raj Kumar	93
9	Shreya Anand	70

- **To display the roll number, name and marks of all the students in the descending order of their marks and ascending order of their names.**

```
mysql> SELECT Rollno, Name, Marks FROM Student
ORDER BY Marks DESC, Name;
```

Sorting on Column Alias

If a column alias is defined for a column, it can be used for displaying rows in ascending or descending order using ORDER BY clause.

For example,

```
SELECT Rollno, Name, Marks AS Marks_obtained
    FROM Student
    ORDER BY Marks_obtained;
```

Alias name

7.19 AGGREGATE FUNCTIONS

Till now, we have studied about single-row functions which work on a single value. SQL also provides multiple-row functions which work on multiple values. So, we can apply SELECT query on a group of records rather than the entire table. Therefore, these functions are called Aggregate functions or Group functions.

Generally, the following aggregate functions are applied on groups as described below:

Table 7.6: Aggregate Functions in SQL

S.No.	Function	Description/Purpose
1	MAX()	Returns the maximum/highest value among the values in the given column/expression.
2	MIN()	Returns the minimum/lowest value among the values in the given column/expression.
3	SUM()	Returns the sum of the values under the specified column/expression.
4	AVG()	Returns the average of the values under the specified column/expression.
5	COUNT()	Returns the total number of values/records under the specified column/expression.

Consider a table Employee (employee code, employee name, salary, job and city) with the following structure:

Ecode	Name	Salary	Job	City
E1	Ritu Jain	5000	Manager	Delhi
E2	Vikas Verma	4500	Executive	Jaipur
E3	Rajat Chaudhary	6000	Clerk	Kanpur
E4	Leena Arora	7200	Manager	Bangalore
E5	Shikha Sharma	8000	Accountant	Kanpur

- **MAX()**

  MAX() function is used to find the highest value among the given set of values of any column or expression based on the column. MAX() takes one argument which can be either a column name or any valid expression involving a particular column from the table.

  For example,

```
mysql> SELECT MAX(Salary) FROM EMPLOYEE;
```

  Output:

```
+-------------+
| MAX(Salary)|
+-------------+
| 8000        |
+-------------+
```

  This command, on execution, shall return the maximum value from the specified column (Salary) of the table Employee, which is 8000.

- **MIN()**

  MIN() function is used to find the lowest value among the given set of values of any column or expression based on the column. MIN() takes one argument which can be either a column name or any valid expression involving a particular column from the table.

For example,

```
mysql> SELECT MIN(Salary) FROM EMPLOYEE;
```

Output:

```
+---------------+
| MIN(Salary) |
+---------------+
| 4500          |
+---------------+
```

This command, on execution, shall return the minimum value from the specified column (Salary) of the table Employee, which is 4500.

- **SUM()**

SUM() function is used to find the total value of any column or expression based on a column. It accepts the entire range of values as an argument, which is to be summed up on the basis of a particular column, or an expression containing that column name. The SUM() function always takes argument of integer type only. Sums of String and Date type data are not defined.

For example,

```
mysql> SELECT SUM(Salary) FROM EMPLOYEE;
```

Output:

```
+---------------+
| SUM(Salary) |
+---------------+
| 30700         |
+---------------+
```

This command, on execution, shall return the total of the salaries of all the employees from the specified column (Salary) of the table Employee, which is 30700.

- **AVG()**

AVG() function is used to find the average value of any column or expression based on a column. Like sum(), it also accepts the entire range of values of a particular column to be taken average of, or even a valid expression based on this column name. Like SUM() function, the AVG() function always takes argument of integer type only. Average of String and Date type data is not defined.

For example,

```
mysql> SELECT AVG(Salary) FROM EMPLOYEE;
```

Output:

```
+---------------+
| AVG(Salary) |
+---------------+
| 6166.66       |
+---------------+
```

Ecode	Name	Salary	Job	City
E1	Ritu Jain	NULL	Manager	Delhi
E2	Vikas Verma	4500	Executive	Jaipur
E3	Rajat Chaudhary	6000	Clerk	Kanpur
E4	Leena Arora	NULL	Manager	Bangalore
E5	Shikha Sharma	8000	Accountant	Kanpur

This command, on execution, shall return the average of the salaries of all the employees from the specified column (Salary) of the table Employee, which is 6166.66.

- **COUNT()**

COUNT() function is used to count the number of values in a column. COUNT() takes one argument, which can be any column name, or an expression based on a column, or an asterisk (*). When the argument is a column name or an expression based on the column, COUNT() returns the number of non-NULL values in that column. If the argument is asterisk (*), then COUNT() counts the total number of records/rows satisfying the condition along with NULL values, if any, in the table.

For example,

```
mysql> SELECT COUNT(*) FROM EMPLOYEE;
```

Output:

```
+----------------+
| COUNT(*)   |
+----------------+
| 5          |
+----------------+
```

This command, on execution, shall return the total number of records in the table Employee, which is 5.

```
mysql> SELECT COUNT(DISTINCT City) FROM EMPLOYEE;
```

Output:

```
+-----------------------------+
| COUNT(DISTINCT City) |
+-----------------------------+
| 4                           |
+-----------------------------+
```

This command, on execution, shall return the total number of records on the basis of city with no duplicate values, *i.e.*, Kanpur is counted only once; the second occurrence is ignored by MySQL because of the DISTINCT clause. Thus, the output will be 4 instead of 5.

- **Aggregate Functions & NULL Values**

Consider the table Employee given in the previous section with NULL values against the Salary field for some employees.

None of the aggregate functions takes NULL into consideration. NULL values are simply ignored by all the aggregate functions as clearly shown in the examples given below:

```
mysql> SELECT SUM(Salary) FROM Employee;
```
Output: 18500

```
mysql> SELECT MIN(Salary) FROM Employee;
```
Output: 4500 (NULL values are not considered.)

```
mysql> SELECT MAX(Salary) FROM Employee;
```
Output: 8000 (NULL values are ignored.)

```
mysql> SELECT COUNT(Salary) FROM Employee;
```
Output: 3 (NULL values are ignored.)

```
mysql> SELECT AVG(Salary) FROM Employee;
```
Output: 6166.66 (It will be calculated as 18500/3, *i.e.*, sum/total no. of records, which are 3 after ignoring NULL values.)

```
mysql> SELECT COUNT(*) FROM Employee;
```

Output: 5

```
mysql> SELECT COUNT(Ecode) FROM Employee;
```

Output: 5 (No NULL value exists in the column Ecode.)

```
mysql> SELECT COUNT(Salary) FROM Employee;
```

Output: 3 (NULL values are ignored while counting the total number of records on the basis of Salary.)

7.20 GROUP BY

The GROUP BY clause can be used in a SELECT statement to collect data across multiple records and group the results by one or more columns. It groups the rows on the basis of the values present in one of the columns and then the aggregate functions are applied on any column of these groups to obtain the result of the query.

This clause can be explained with reference to the table student; the rows can be divided into four groups on the basis of the column Stream. One group of rows belongs to "Science" stream, another belongs to "Commerce" stream, the third group belongs to "Humanities" stream and the fourth belongs to "Vocational" stream. Thus, by using GROUP BY clause, the rows can be divided on the basis of the stream column.

Syntax for the GROUP BY clause is:

SELECT <column1, column2, ...column_n>, <aggregate_function (expression)>

FROM <tables>

WHERE <conditions>

GROUP BY <column1>, <column2>, ... <column_n>;

Here, *column_names* must include the columns on the basis of which grouping is to be done.

aggregate_function can be a function such as sum(), count(), max(), min(), avg(), etc.

For example, to display the name, stream, marks and count the total number of students who have secured more than 90 marks according to their stream.

```
mysql> SELECT Name, Stream, count(*) AS "Number of students"
       FROM Student
       WHERE Marks>90
       GROUP BY Stream;
```

Resultant table: Student

Name	Stream	Number of students	Marks
Raj Kumar	Science	1	93
Deep Singh	Commerce	2	98

2 rows in a set (0.02 sec)

7.21 HAVING CLAUSE

The HAVING clause is used in combination with the GROUP BY clause. It can be used in a SELECT statement to filter the records by specifying a condition which a GROUP BY returns.

The purpose of using HAVING clause with GROUP BY is to allow aggregate functions to be used along with the specified condition. This is because the aggregate functions are not allowed to be used with WHERE clause as it is evaluated on a single row whereas the aggregate functions are evaluated on a group of rows.

Thus, if any aggregate function is to be used after the FROM clause in a SELECT command, then instead of using the WHERE clause, HAVING clause should be used.

The Syntax for HAVING clause is:

SELECT <column1>, <column2>, ...<column_n>, <aggregate_function (expression)>

FROM <tables>

WHERE <condition/predicates>

GROUP BY [<column1, column2, ... column_n]

HAVING [<condition1 ... condition_n>];

For example,

```
mysql> SELECT Stream, SUM(Marks) AS "Total Marks"
       FROM Student
       GROUP BY Stream
       HAVING MAX(Marks) <85;
```

Stream	Total Marks
Science	151
Commerce	0
Humanities	158
Vocational	152

In the given output, the sum(Marks) for Commerce Stream is 0 since none of the values for field Marks in the Commerce Stream is less than 85.

CTM: SELECT statement can contain only those attributes which are already present in the GROUP BY clause.

7.22 AGGREGATE FUNCTIONS AND CONDITIONS ON GROUPS (HAVING CLAUSE)

You may use any condition on group, if required. HAVING <condition> clause is used to apply a condition on a group.

```
mysql> SELECT Job, SUM(Pay) FROM EMP GROUP BY Job HAVING SUM(Pay)>=8000;
mysql> SELECT Job, SUM(Pay) FROM EMP GROUP BY Job HAVING AVG(Pay)>=7000;
mysql> SELECT Job, SUM(Pay) FROM EMP GROUP BY Job HAVING COUNT(*)>=5;
mysql> SELECT Job, MIN(Pay), MAX(Pay), AVG(Pay) FROM EMP
       GROUP BY Job HAVING SUM(Pay)>=8000;
mysql> SELECT Job, SUM(Pay) FROM EMP WHERE City='Dehradun'
       GROUP BY Job HAVING COUNT(*)>=5;
```

Table 7.5: SQL Operators and their Functions

OPERATOR/FUNCTION	DESCRIPTION
ARITHMETIC OPERATORS	Used in arithmetic calculations and expressions. +, -, *, /, %, **
COMPARISON/ RELATIONAL OPERATORS	
=, >, <, >=, <=, <>	Used in Conditional expressions.
LOGICAL OPERATORS	
AND/OR/NOT	Used in Conditional expressions.
SPECIAL OPERATORS	
BETWEEN/NOT BETWEEN	Checks whether an attribute value is within a range or not.
IS NULL/IS NOT NULL	Checks whether an attribute value is NULL or not.
LIKE/NOT LIKE	Checks whether an attribute matches a given string pattern or not.
IN/NOT IN	Checks whether an attribute value matches any value with a given list or not.
DISTINCT	Permits only unique values. Eliminates duplicate ones.
AGGREGATE FUNCTIONS	Used with SELECT to return mathematical results/values on the basis of the operation performed on the columns.
COUNT()	Returns the total number of records with non-null values for a given column.
MIN()	Returns the minimum/lowest attribute value found in a given column.
MAX()	Returns the maximum/highest attribute value found in a given column.
SUM()	Returns the sum of all the values for a given column.
AVG()	Returns the average of all the values for a given column.

(b) Arithmetic Operators

Arithmetic operators are used to perform simple arithmetic operations like addition (+), subtraction (–), multiplication (*), division (/) and modulus (%). These operators are used with conditional expressions and for performing simple mathematical calculations. The arithmetic operators with SELECT command are used to retrieve rows computed with or without reference to any table.

```
mysql> SELECT 5 + 10 FROM DUAL;
```

The above statement returns the value 15 as the result.

```
+----------+
| 5 + 10   |
+----------+
| 15       |
+----------+
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-----------+---------+
| SIN(PI()/4) | (4+1)*5 |
+-----------+---------+
| 0.707107  |   25    |
+-----------+---------+
```

8. DROP TABLE Command

Sometimes, we may need to physically remove a table which is not in use. DROP TABLE command is used to remove/delete a table permanently. If you drop a table, all the rows in the table are deleted along with its structure. Once a table is dropped, we cannot get it back and all other references to the table become invalid. This command completely destroys the table structure.

Syntax for removing a table:

DROP TABLE <table-name>;

For example,

```
mysql> DROP TABLE Student;
```

This command will permanently remove the table student from the database school.

7.14.2 DML Commands

When we create a table, only its structure is created but the table has no data. To populate records in the table, INSERT statement is used. Also, table records can be deleted or updated using DELETE and UPDATE statements. These SQL statements are part of Data Manipulation Language (DML).

Data Manipulation using a database means either insertion of new data, removal of existing data or modification of existing data in the database.

1. Inserting Data into a Table

The **INSERT INTO** command is used to insert a new record/row/tuple in a table.

It is possible to write the INSERT INTO statement in the following different forms:

(a) **Inserting data (for all the columns) into a table:** In the first method, it does not specify the column names where the data will be inserted, only their values.

Syntax for SQL INSERT is:

INSERT INTO <table_name> VALUES (*value1, value2, value3...*);

For example,

```
mysql> INSERT INTO Student VALUES (1,"Raj Kumar", 'M', 93,
'2000-11-17');
```

While inserting a row, if we are adding value for all the columns of the table, we need not specify the column(s) name in the SQL query. But we need to make sure that the order of the values is in the same order as the columns represented in the structure of the table. The following points should be kept in mind while inserting data in a relation:

- When values are inputted using INSERT INTO command, it is termed as single row insert since it adds one tuple at a time into the table.
- The INTO clause specifies the target table and the VALUES clause specifies the data to be added to the new record of the table.
- The argument/values of character or text and date datatype are always enclosed in double or single quotation marks.
- Column values for the date data type of a column are provided within single quotes in 'yyyy-mm-dd' or "yyyy/mm/dd" format.