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 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;
```

- 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;
```

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

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                (NUL

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.