14
More on SQL

In This Chapter

14.1 INTRODUCTION

You have worked with basic SQL commands in your previous class and used database MySQL This chapter will talk about some SQL commands in details. After this chapter, you shall be abl to display groups of records, summary of groups of records and display selected groups o records.

Let us begin with our discussion on how you can order records of a table using ORDER BY clause.

14.2 ORDERING RECORDS IN RESULT – ORDER BY CLAUSE

Although you have read about ORDER BY clause of SQL SELECT statement in your previous class, here we are covering it again in a detailed manner.

14.2.1 Recalling SQL SELECT ORDER BY Clause

The result set generated by the SQL SELECT statement is not ordered in any form by default. However, if you want to sort or order the result set, you can use the ORDER BY clause of SQL SELECT statement as per following format :

```
SELECT <comma separated select list> FROM <table>
[WHERE <condition>]
ORDER BY <fieldname> [ASC|DESC] [,<fieldname> [ASC|DESC], … ] ;
```

Keywords ASC and DESC denote the order – **ASC** stands for ***ascending*** and the DESC stands for ***descending***. If you do not specify any order keyword ASC or DESC, then by default, the ORDER BY clause sorts the result set in ascending order.

For example, consider the table **Data** having records as shown below :

```
+---------+----------+-----------+-------+-----------+
| rollno  | name     | marks     | grade | section   |
+---------+----------+-----------+-------+-----------+
| 101     | Ruhani   | 76.80     | A     | A         |
| 102     | George   | 71.20     | B     | A         |
| 103     | Simran   | 81.20     | A     | B         |
| 104     | Ali      | 61.20     | B     | C         |
| 105     | Kushal   | 51.60     | C     | C         |
| 106     | Arsiya   | 91.60     | A+    | B         |
| 107     | Raunaq   | 32.50     | F     | B         |
| 108     | Meera    | 97.20     | A+    | B         |
| 109     | Amaal    | 57.20     | C     | B         |
| 111     | Simran   | 66.00     | B     | A         |
| 112     | Adam     | 74.20     | B     | C         |
| 113     | Gurnoor  | 93.50     | A+    | B         |
| 115     | Rabiya   | 72.50     | B     | B         |
| 117     | Rahil    | 32.00     | F     | C         |
| 118     | Neha     | 59.50     | C     | A         |
+---------+----------+-----------+-------+-----------+
```

Now the statement

```
mysql> SELECT * FROM data
    -> ORDER BY marks ;
```

will produce result set with marks arranged in ascending order as we did not specify ASC keyword explicitly, *i.e.*,

```
+---------+----------+-----------+-------+-----------+
| rollno  | name     | marks     | grade | section   |
+---------+----------+-----------+-------+-----------+
| 117     | Rahil    | 32.00     | F     | C         |
| 107     | Raunaq   | 32.50     | F     | B         |
| 105     | Kushal   | 51.60     | C     | C         |
| 109     | Amaal    | 57.20     | C     | B         |
| 118     | Neha     | 59.50     | C     | A         |
| 104     | Ali      | 61.20     | B     | C         |
| 111     | Simran   | 66.00     | B     | A         |
| 102     | George   | 71.20     | B     | A         |
| 115     | Rabiya   | 72.50     | B     | B         |
| 112     | Adam     | 74.20     | B     | C         |
| 101     | Ruhani   | 76.80     | A     | A         |
| 103     | Simran   | 81.20     | A     | B         |
| 106     | Arsiya   | 91.60     | A+    | B         |
| 113     | Gurnoor  | 93.50     | A+    | B         |
| 108     | Meera    | 97.20     | A+    | B         |
+---------+----------+-----------+-------+-----------+
15 ROWS IN SET (0.01 SEC)
```

The above SQL statement is equivalent to statement shown below where ASC word is explicitly specified.

```
mysql> SELECT * FROM data
    -> ORDER BY marks ASC ;
```

This statement will also produce the same result as above because ASC order is taken by default.

14.2.2 Ordering Data on Multiple Columns

To order the result set on multiple columns, you can specify the multiple column names in ORDER by clause along with the desired sort order, *i.e.*, as :

```
SELECT
:
ORDER BY <fieldname1> [ASC|DESC] [,<fieldname1> [ASC|DESC], ... ] ;
```

For example, the following statement will sort the records firstly on the column name **Section** and then on the basis of descending order of column **marks.**

```
mysql> SELECT * FROM data
    -> ORDER BY section ASC , marks DESC ;
```

*First sort field is **section** in ascending order (as **section ASC**) and for all the records of same section, the sort field is **marks** with descending order (**marks DESC**)*

```
+---------+----------+----------+-------+-----------+
| rollno  | name     | marks    | grade | section   |
+---------+----------+----------+-------+-----------+
| 101     | Ruhani   | 76.80    | A     | A         |
| 102     | George   | 71.20    | B     | A         |
| 111     | Simran   | 66.00    | B     | A         |
| 118     | Neha     | 59.50    | C     | A         |
| 108     | Meera    | 97.20    | A+    | B         |
| 113     | Gurnoor  | 93.50    | A+    | B         |
| 106     | Arsiya   | 91.60    | A+    | B         |
| 103     | Simran   | 81.20    | A     | B         |
| 115     | Rabiya   | 72.50    | B     | B         |
| 109     | Amaal    | 57.20    | C     | B         |
| 107     | Raunaq   | 32.50    | F     | B         |
| 112     | Adam     | 74.20    | B     | C         |
| 104     | Ali      | 61.20    | B     | C         |
| 105     | Kushal   | 51.60    | C     | C         |
| 117     | Rahil    | 32.00    | F     | C         |
+---------+----------+----------+-------+-----------+
15 rows in set (0.00 sec)
```

See all records of same section are listed together

*All records with same section are arranged in descending order of **marks** column*

14.2.3 Ordering Data on the basis of an Expression

Sometimes, you need to display the result of a calculation or a mathematical expression in the result set. In such cases, you may want or need to arrange your result set in the order of the calculated expression. The ORDER BY clause allows you to include the mathematical expression to order the result set by it. However, to arrange a result set on the basis of a mathematical expression, you should preferably (though not a necessity but preferably) include the mathematical expression in the select list so that it becomes easy to comprehend the result.

Consider the following example statement that arranges the result set on the basis of a calculated result :

```
mysql> SELECT rollno, name, grade, section, marks*.35 FROM data
    -> WHERE marks > 70
    -> ORDER BY section ASC, marks*0.35 DESC ;
```

```
+---------+----------+----------+----------+-----------+
| rollno  | name     | grade    | section  | marks*.35 |
+---------+----------+----------+----------+-----------+
| 101     | Ruhani   | A        | A        | 26.8800   |
| 102     | George   | B        | A        | 24.9200   |
| 108     | Meera    | A+       | B        | 34.0200   |
| 113     | Gurnoor  | A+       | B        | 32.7250   |
| 106     | Arsiya   | A+       | B        | 32.0600   |
| 103     | Simran   | A        | B        | 28.4200   |
| 115     | Rabiya   | B        | B        | 25.3750   |
| 112     | Adam     | B        | C        | 25.9700   |
+---------+----------+----------+----------+-----------+
8 rows in set (0.02 sec)
```

If you want, you can provide a column alias name to the mathematical expression in the select list, *e.g.*, following statement will also produce the same result as above but it will name the column **marks*.35** as **Term1**:

```
mysql> SELECT rollno, name, grade, section, marks*.35 as Term1 FROM data
    -> WHERE marks > 70
    -> ORDER BY section ASC, Term1 DESC ;
```

```
+---------+----------+----------+----------+-----------+
| rollno  | name     | grade    | section  | Term1     |
+---------+----------+----------+----------+-----------+
| 101     | Ruhani   | A        | A        | 26.8800   |
| 102     | George   | B        | A        | 24.9200   |
| 108     | Meera    | A+       | B        | 34.0200   |
| 113     | Gurnoor  | A+       | B        | 32.7250   |
| 106     | Arsiya   | A+       | B        | 32.0600   |
| 103     | Simran   | A        | B        | 28.4200   |
| 115     | Rabiya   | B        | B        | 25.3750   |
| 112     | Adam     | B        | C        | 25.9700   |
+---------+----------+----------+----------+-----------+
8 rows in set (0.00 sec)
```

14.2.4 Specifying Custom Sort Order

Sometimes, you have a column where you want to arrange data as per your own specified order. For example, if there is column called *Project* that stores the status of project made by the students. It can have possible values as ***Evaluated, Submitted, Pending, Assigned***. If you want to arrange the result set on the basis of this *Project* column as per this order : ***Evaluated, Pending, Submitted, Assigned***. For this, you need to use the FIELD function in ORDER BY clause as per this format :

```
SELECT
:
ORDER BY FIELD(<column name>, <values specifying order>) ;
```

The FIELD function internally maps the values-specifying-order to a list of numeric values and then uses those numbers for sorting. You need not do anything, FIELD( ) does all this on its own.

To understand, let us consider the same data table with added project column with these values :

```
+--------+---------+-------+-------+---------+-----------+
| rollno | name    | marks | grade | section | project   |
+--------+---------+-------+-------+---------+-----------+
| 101    | Ruhani  | 76.80 | A     | A       | Pending   |
| 102    | George  | 71.20 | B     | A       | Submitted |
| 103    | Simran  | 81.20 | A     | B       | Evaluated |
| 104    | Ali     | 61.20 | B     | C       | Assigned  |
| 105    | Kushal  | 51.60 | C     | C       | Evaluated |
| 106    | Arsiya  | 91.60 | A+    | B       | Submitted |
| 107    | Raunaq  | 32.50 | F     | B       | Submitted |
| 108    | Meera   | 97.20 | A+    | B       | Evaluated |
| 109    | Amaal   | 57.20 | C     | B       | Pending   |
| 111    | Simran  | 66.00 | B     | A       | Pending   |
| 112    | Adam    | 74.20 | B     | C       | Pending   |
| 113    | Gurnoor | 93.50 | A+    | B       | Assigned  |
| 115    | Rabiya  | 72.50 | B     | B       | Assigned  |
| 117    | Rahil   | 32.00 | F     | C       | Submitted |
| 118    | Neha    | 59.50 | C     | A       | Evaluated |
+--------+---------+-------+-------+---------+-----------+
15 rows in set (0.00 sec)
```

Now to order the above table as per the mentioned order above *i.e.*, as per **Project** field having values in this order : ***'Evaluated'***, ***'Pending'***, ***'Submitted'***, ***'Assigned'***, you can write the SELECT statement's ORDER BY clause with FIELD( ) as shown below :

The sort field

```
mysql> select * from data
    -> ORDER BY FIELD(Project, 'Evaluated', 'Pending', 'Submitted', 'Assigned' ) ;
```

The desired sort order; Notice, string values are given in quotes

```
+--------+---------+-------+-------+---------+-----------+
| rollno | name    | marks | grade | section | project   |
+--------+---------+-------+-------+---------+-----------+
| 118    | Neha    | 59.50 | C     | A       | Evaluated |
| 108    | Meera   | 97.20 | A+    | B       | Evaluated |
| 105    | Kushal  | 51.60 | C     | C       | Evaluated |
| 103    | Simran  | 81.20 | A     | B       | Evaluated |
| 101    | Ruhani  | 76.80 | A     | A       | Pending   |
| 112    | Adam    | 74.20 | B     | C       | Pending   |
| 111    | Simran  | 66.00 | B     | A       | Pending   |
| 109    | Amaal   | 57.20 | C     | B       | Pending   |
| 102    | George  | 71.20 | B     | A       | Submitted |
| 107    | Raunaq  | 32.50 | F     | B       | Submitted |
| 106    | Arsiya  | 91.60 | A+    | B       | Submitted |
| 117    | Rahil   | 32.00 | F     | C       | Submitted |
| 113    | Gurnoor | 93.50 | A+    | B       | Assigned  |
| 115    | Rabiya  | 72.50 | B     | B       | Assigned  |
| 104    | Ali     | 61.20 | B     | C       | Assigned  |
+--------+---------+-------+-------+---------+-----------+
15 rows in set (0.04 sec)
```

Result set order as per the custom order as specified by you.

14.3 AGGREGATE FUNCTIONS

Till now you have learnt to work with functions that operate on individual rows in a table *e.g.*, if you use **Round( )** function then it will round off values from each row of the table.

MySQL also supports and provides **group functions** or **aggregate functions**. As you can make out that the group functions or aggregate functions work upon groups of rows, rather than on single rows. That is why, these functions are sometimes also called **multiple row functions.**

Many group functions accept the following options :

DISTINCT This option causes a group function to consider only distinct values of the argument expression.

ALL This option causes a group function to consider all values including all duplicates.

The usage of these options will become clear with the coverage of examples in this section. All the examples that we'll be using here, shall be based upon following table **empl**.

Table 14.1 *Database table empl*

```
+-----+----------+----------+-----+----------+--------+--------+------+
|EMPNO|  ENAME   |   JOB    | MGR | HIREDATE |  SAL   |  COMM  |DEPTNO|
+-----+----------+----------+-----+----------+--------+--------+------+
| 8369 | SMITH    | CLERK    | 8902 | 1990-12-18 | 800.00  | NULL    |  20  |
| 8499 | ANYA     | SALESMAN | 8698 | 1991-02-20 | 1600.00 | 300.00  |  30  |
| 8521 | SETH     | SALESMAN | 8698 | 1991-02-22 | 1250.00 | 500.00  |  30  |
| 8566 | MAHADEVAN | MANAGER  | 8839 | 1991-04-02 | 2985.00 | NULL    |  20  |
| 8654 | MOMIN    | SALESMAN | 8698 | 1991-09-28 | 1250.00 | 1400.00 |  30  |
| 8698 | BINA     | MANAGER  | 8839 | 1991-05-01 | 2850.00 | NULL    |  30  |
| 8839 | AMIR     | PRESIDENT | NULL | 1991-11-18 | 5000.00 | NULL    |  10  |
| 8844 | KULDEEP  | SALESMAN | 8698 | 1991-09-08 | 1500.00 | 0.00    |  30  |
| 8882 | SHIAVNSH | MANAGER  | 8839 | 1991-06-09 | 2450.00 | NULL    |  10  |
| 8886 | ANOOP    | CLERK    | 8888 | 1993-01-12 | 1100.00 | NULL    |  20  |
| 8888 | SCOTT    | ANALYST  | 8566 | 1992-12-09 | 3000.00 | NULL    |  20  |
| 8900 | JATIN    | CLERK    | 8698 | 1991-12-03 | 950.00  | NULL    |  30  |
| 8902 | FAKIR    | ANALYST  | 8566 | 1991-12-03 | 3000.00 | NULL    |  20  |
| 8934 | MITA     | CLERK    | 8882 | 1992-01-23 | 1300.00 | NULL    |  10  |
+-----+----------+----------+-----+----------+--------+--------+------+
```

1. AVG

This function computes the average of given data.

Syntax

```
AVG([DISTINCT | ALL] n)
```

▲ Returns average value of parameter(s) *n*.

Argument type : Numeric *Return value* : Numeric

EXAMPLE 14.1. *Calculate average salary of all employees listed in table empl.*

Solution.

```
mysql> SELECT AVG(sal) "Average"
       FROM empl ;
```

```
+-------------+
| Average     |
+-------------+
| 2073.928571 |
+-------------+
1 row in set (0.01 sec)
```

This function counts the number of rows in a given column or expression.

Syntax

```
COUNT({ * [DISTINCT | ALL] expr})
```

- Returns the number of rows in the query.
- If you specify argument *expr*, this function returns rows where *expr* is not null. You can count either all rows, or only distinct values of *expr*.
- If you specify the asterisk (*), this function returns all rows, including duplicates and nulls.

Argument type : Numeric   *Return value* : Numeric

EXAMPLE 14.2. *Count number of records in table empl.*

Solution.

```
mysql> SELECT COUNT(*) "Total"
       FROM empl ;
```

```
+-----------+
| Total     |
+-----------+
|    14     |
+-----------+
1 row in set (0.00 sec)
```

EXAMPLE 14.3. *Count number of jobs in table empl.*

Solution.

```
mysql> SELECT COUNT(job) "Job Count"
       FROM empl ;
```

```
+-------------+
| Job Count   |
+-------------+
|    14       |
+-------------+
1 row in set (0.01 sec)
```

EXAMPLE 14.4. *How many distinct jobs are listed in table empl ?*

Solution.

```
mysql> SELECT COUNT(DISTINCT job) "Distinct Jobs"
       FROM empl ;
```

```
+----------------+
| Distinct Jobs  |
+----------------+
|      15        |
+----------------+
1 row in set (0.04 sec)
```

3. MAX

This function returns the maximum value from a given column or expression.

Syntax

```
MAX([DISTINCT|ALL] expr)
```

- Returns maximum value of argument *expr*.

Argument type : Numeric   *Return value* : Numeric

EXAMPLE 14.5. *Display maximum salary from table empl.*

Solution.

```
mysql> SELECT MAX(sal) "Maximum Salary"
       FROM empl ;
```

```
+----------------+
| Maximum Salary |
+----------------+
|    5000.00     |
+----------------+
1 row in set (0.01 sec)
```

4. MIN

This function returns the minimum value from a given column or expression.

Syntax

```
MIN([DISTINCT | ALL] expr)
```

- Returns minimum value of *expr*.

Argument type : Numeric *Return value* : Numeric

EXAMPLE 14.6. *Display the Joining date of seniormost employee.*

Solution.

```
mysql> SELECT MIN(hiredate) "Minimum Hire Date"
       FROM empl ;
```

```
+-----------------------
| Minimum Hire Date    |
+-----------------------
|      1990-12-18
+-----------------------
1 row in set (0.06
```

5. SUM

This function returns the sum of values in given column or expression.

Syntax

```
SUM([DISTINCT | ALL] n)
```

- Returns sum of values of *n*.

Argument type : Numeric *Return value* : Numeric

EXAMPLE 14.7. *Display total salary of all employees listed in table empl.*

Solution.

```
mysql>  SELECT SUM(sal) "Total Salary"
        FROM empl ;
```

```
+-----------------+
|  Total Salary   |
+-----------------+
|    29035.00     |
+-----------------+
1 row in set (0.01 sec)
```

Some more examples of group functions are being given below :

NOTE

These functions are called ***aggregate functions*** because they operate on aggregates of tuples. The result of an aggregate function is a single value.

Examples :

1. To calculate the total gross for employees of grade 'E2', the command is :

```
SELECT SUM(gross) FROM employee
WHERE grade = 'E2' ;
```

2. To display the average gross of employees with grades 'E1' or 'E2', the command used is:

```
SELECT AVG(gross) FROM employee
WHERE (grade = 'E1' OR grade = 'E2') ;
```

3. To count the number of employees in *employee* table, the SQL command is :

```
SELECT COUNT(*)
FROM employee ;
```

4. To count the number of cities, the different members belong to, you use the following command :

```
SELECT COUNT(DISTINCT city)   FROM members ;
```

Here the DISTINCT keyword ensures that multiple entries of the same *city* are ignored. The * is the only argument that includes NULLs when it is used only with COUNT, functions other than COUNT disregard NULLs in any case.

If you want to count the entries including repeats, the keyword ALL is used. The following command will COUNT the number of non NULL *city* fields in the *members* table :

```
SELECT COUNT(ALL city)
FROM members ;
```

In general, GROUP functions

- Return a single value for a set of rows.
- Can be applied to any numeric values, and some Text types and DATE values.

14.4 TYPES OF SQL FUNCTIONS

Now that you have learnt different types of functions, let us talk about their broad categories. SQL supports many and many functions. All these functions can be generally categorized into following *two* types :

- Single Row (or Scalar) functions.
- Multiple Row (or Group or Aggregate) functions.

(i) ***Single Row functions*** work with a single row at a time. A single row function returns a result for every row of a queried table.

(ii) ***Multiple Row or Group functions*** work with data of multiple rows at a time and return aggregated value.

Examples of multiple row functions are the group functions that you have learnt in previous section *i.e.*, sum( ), count( ), max( ), min( ), Avg( ) etc.

The difference between these two types of functions is in the number of rows they act upon. A **single row function** works with the data of a single row at a time and returns a single result for each row queried upon ; a **multiple row function** works with a group of rows and returns a single result for that group.

14.5 GROUPING RESULT – GROUP BY

The GROUP BY clause combines all those records that have identical values in a particular field or a group of fields. This grouping results into one summary record per group if group-functions are used with it. In other words, the GROUP BY clause is used in SELECT statements to divide the table into groups. Grouping can be done by a column name, or with aggregate functions in which case the aggregate produces a value for each group.

For example, to calculate the *number of employees in each grade*, you use the command

```
SELECT job, COUNT(*)
FROM empl
GROUP BY job ;
```

```
+------------+----------+
|  job       | COUNT(*) |
+------------+----------+
| ANALYST    |        2 |
| CLERK      |        4 |
| MANAGER    |        3 |
| PRESIDENT  |        1 |
| SALESMAN   |        4 |
+------------+----------+
```

GROUP BY applies the aggregate functions independently to a series of groups that are defined by having a field value in common.

Now consider the following query, which is also grouping records based on *deptno*.

```
mysql>  SELECT deptno, COUNT(*), SUM(sal)
        FROM empl
        GROUP BY deptno ;
```

```
+------------+----------+-----------+
|  deptno    | COUNT(*) |  SUM(sal)  |
+------------+----------+-----------+
|     10     |     3    |  8750.00   |
|     20     |     5    |  10885.00  |
|     30     |     6    |  9400.00   |
+------------+----------+-----------+
```

As you can make out that the above query is displaying count of records and sum of salaries in each group and the groups are formed on the basis of *deptno*. Thus, from the above output, you can make out that in department number 10, there are 3 employees (records) and total of all salaries is 8750.00 ; in department number 20, there are 5 employees and total of salaries is 10885.00 ; and so on.

14.5.1 Nested Groups – Grouping on Multiple Columns

With GROUP BY clause, you can create groups within groups. Such type of grouping is called ***Nested grouping***. This can be done by specifying in GROUP BY expression, where the *first field* determines the *highest group level*, the *second field* determines the *second group level*, and so on. The *last field* determines the *lowest level of grouping*.

In order to fully understand this concept, consider Table **14.1 empl**.

See there are multiple records having same value for field **Deptno**, we can group records on the basis of field *Deptno*. For instance, if you want to count the number of employees in each group, you need to issue a query statement as given below :

```
SELECT COUNT(empno) FROM empl
GROUP BY Deptno ;
```

And the result produced by this query is

```
+-------------+
|count(empno) |
+-------------+
|      3      |
|      5      |
|      6      |
+-------------+
```

But can you make out, these are employee-counts for which departments ? To get this information, you may modify the SELECT list as :

```
SELECT Deptno, COUNT(empno)
FROM empl
GROUP BY Deptno ;
```

Now the result will be :

```
+----------+-------------
| deptno   | count(empno)
+----------+-------------
|  10      |      3
|  20      |      5
|  30      |      6
+----------+-------------
```

See, now it is more clear. But one thing that you should keep in mind is that while grouping, you should include only those values in the *select list* that either have the same value for a group or contain a group (aggregate) function *i.e.*, a **group-expression**. Like in the above query, the first expression **Deptno** field has one (same) value for a group and the other expression **COUNT(empno)** contains a group function.

MySQL as such would not create any error even if you include a **non-group expression** in the select-list. A **non-group field** (or expression) is the field that has different values in the rows belonging to the group.

In this case, it will return **the value from first record** of the group for that non-group field *e.g.*, if you issue command like :

```
SELECT deptno, count(empno), mgr
FROM empl
GROUP BY deptno ;
```

This is non-group field as it has multiple values for a group.

The output returned will be :

```
+---------+--------------+------+
| deptno  | count(empno) | mgr  |
+---------+--------------+------+
|   10    |       3      | NULL |
|   20    |       5      | 8902 |
|   30    |       6      | 8698 |
+---------+--------------+------+
```

See, first record of group with deptno 10 has value NULL in mgr field ; first record of group with deptno 20 has 8902 in mgr field ; and first record of group with deptno 30 has value 8698 in mgr field. Hence this output.

To create a group within a group *i.e.*, nested group, you need to specify multiple fields in the GROUP BY expression. If you have a look at the records of *Empl* table, you can make out that there exists a *group of jobs* within the *department group* as there are same values for **Job** field, in one department group's records.

NOTE

In the select list of a group, only those fields or expressions state be included that either return single value for a group or are constants. Otherwise you may not get authentic results.

To group records *job wise* within *Deptno* wise, you need to issue a query statement like :

```
SELECT Deptno, Job, COUNT(empno)
FROM empl
GROUP BY Deptno, Job ;
```

And the result produced is :

```
+--------+-----------+--------------+
| deptno |    job    | count(empno) |
+--------+-----------+--------------+
|   10   | CLERK     |      1       |
|   10   | MANAGER   |      1       |
|   10   | PRESIDENT |      1       |
|   20   | ANALYST   |      2       |
|   20   | CLERK     |      2       |
|   20   | MANAGER   |      1       |
|   30   | CLERK     |      1       |
|   30   | MANAGER   |      1       |
|   30   | SALESMAN  |      4       |
+--------+-----------+--------------+
```

14.5.2 Placing Conditions on Groups – HAVING Clause

The HAVING clause places conditions on groups in contrast to WHERE clause that places conditions on individual rows. While WHERE conditions cannot include aggregate functions, HAVING conditions can do so.

Check Point 14.1

1. Can you arrange the result set of an SQL query on multiple columns ?
2. What is the significance of "ORDER BY" in the given query?

```
SELECT emp_id, fname, lname
FROM person
ORDER BY emp_id;
```

(*a*) Data of table **person** on the basis of column **emp**_id will be sorted in descending order

(*b*) Data of table **person** on the basis of column **emp**_id will be sorted in ascending order

(*c*) Only data of column emp_id will be sorted in descending order

(*d*) Only data of column emp_id will be sorted in ascending order

3. What will be the order of sorting in the given query?

```
SELECT emp_id, emp_name
FROM person
ORDER BY emp_id, emp_name;
```

(*a*) Firstly on ***emp_id*** and then on ***emp_name***

(*b*) Firstly on ***emp_name*** and then on ***emp_id***

(*c*) Firstly on ***emp_id*** but not on ***emp_name***

(*d*) None of the mentioned

4. If column ***emp_id*** contains the following set {9, 7, 6, 4, 3, 1, 2}, what will be the output on execution of the given query?

```
SELECT emp_id FROM person
ORDER BY emp_id;
```

(*a*) {9, 7, 6, 4, 3, 1, 2}

(*b*) {1, 2, , 3, 4, 6, 7, 9}

(*c*) {2, 1, 4, 3, 7, 9, 6}

(*d*) None of these

5. Which function can you use with ORDER BY clause to specify custom sort order ?

(*a*) SORT( )

(*b*) CUSTOM( )

(*c*) FIELD( )

(*d*) All of these

For example, to calculate the average gross and total gross for employees belonging to 'E4' grade, the command would be :

```
SELECT AVG(gross), SUM(gross)
FROM employee
GROUP BY grade
HAVING grade = 'E4' ;
```

This condition would be applicable on group and not on individual rows

To display the jobs where the number of employees is less than 3, you use the command :

```
SELECT JOB, COUNT(*)
FROM empl
GROUP BY job
HAVING count(*) < 3 ;
```

This will produce the following output :

```
+-----------+----------+
| JOB       | COUNT(*) |
+-----------+----------+
| ANALYST   |        2 |
| PRESIDENT |        1 |
+-----------+----------+
2 rows in set (0.11 sec)
```

The HAVING clause can contain either a simple boolean expression (*i.e.*, an expression or condition that results into *true* or *false*) or use aggregate function in the having condition.

You can include more than one condition in HAVING clause, of course, by using logical operators. Consider this :

```
SELECT Deptno, AVG(Comm), AVG(Sal)
FROM empl
GROUP BY Deptno
HAVING AVG(Comm) > 750 AND
AVG(Sal) > 2000 ;
```

Also, you can use an aggregate function in the HAVING clause even if it is not in the SELECT list. Consider the following query to understand this :

```
SELECT Deptno, AVG(Sal)
FROM empl
GROUP BY Deptno
HAVING COUNT(*) <= 3 ;
```

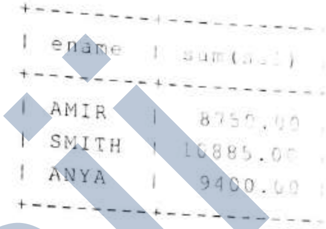
You can also use IN or BETWEEN operators with HAVING clause. Following two queries illustrate this :

```
SELECT Deptno, Job, AVG(Sal)
FROM empl
GROUP BY Deptno, Job
HAVING JOB IN ('CLERK', 'SALESMAN') ;
SELECT Deptno, Job, SUM(sal)
FROM empl
GROUP BY Deptno, Job
HAVING SUM(sal) BETWEEN 3000 AND 7000 ;
```

14.5.3 Non-Group Expressions with GROUP BY

As mentioned before, if you include a non-group expression in the select-list of a query with GROUP BY, MySQL will not produce any error. Rather it will pick value of the specified non-group field from the first row of the group. But we do not recommend this practice because it will produce ambiguous results. For instance, consider the following query and its output.

```
mysql>  SELECT ename, sum(sal)
        FROM empl
        GROUP BY deptno ;
```



```
+--------+-----------+
| ename  | sum(sal)  |
+--------+-----------+
| AMIR   |   8750.00 |
| SMITH  |  10885.00 |
| ANYA   |   9400.00 |
+--------+-----------+
```

See, isn't it conveying that AMIR's salary-sum is 8750.00, SMITH'S 10885.00 and ANYA'S 9400.00 ?

Thus, we recommend not to use non-group expressions in GROUP BY query unless otherwise necessary.

LET US REVISE

- *The ORDER BY clause lets you arrange the result set in the order of single column, multiple columns, on the basis of an expression and as per custom sort order too.*
- *The GROUP BY clause combines all those records that have identical value in a particular field or a group of fields.*
- *GROUP BY clause is used to divide the result in groups.*
- *A group within another group is called* **Nested Group**.
- *Nested grouping can be done by providing multiple fields in the GROUP BY expression.*
- *All fields containing a NULL value are considered to have a value and are grouped to have a value and are grouped with the fields containing non-NULL values.*
- *The SELECT list of a group can include expressions returning single value per group or constants.*
- *The HAVING clause is used to specify filtering condition for groups.*
- *The difference between WHERE and HAVING clause is that WHERE conditions are applicable on individual rows whereas HAVING conditions are applicable on groups as formed by GROUP BY clause.*

Solved Problems

1. *The SQL SELECT provides clauses for sorting data and for summarizing results. Write the names of clauses for these.*

   Solution. The ORDER BY clause of SQL SELECT statement allows to sort the data of result set.

   The GROUP BY clause of SQL SELECT statement allows to create summarized results of grouped data from table.

2. *You want to group the result set based on some column's value. Also, you want that the grouped result should appear in a sorted order. In which order will you write the two clauses (for sorting and for grouping). Give example to support your answer.*

   Solution. When we use GROUP BY clause (for grouping of data) and ORDER BY clause (for sorting data) together, the ORDER BY clause always follows other clauses. That is, the GROUP BY clause will come before ORDER BY clause.

For example,

```
SELECT userid, SUM(score) AS total_score
FROM user_score
GROUP BY userid
ORDER BY userid ASC;
```

3. *In a table **Apply**, there is a column namely **Experience** that can store only one of these values : 'Fresher', 'Private-sector-experience', 'Public-sector-experience', 'Govt.-sector experience'.*

   *You want to sort the data of table based on column **experience** as per this order : 'Govt-sector-experience', 'Public-sector-experience', 'Private-sector-experience', 'Fresher'.*

   Write an SQL query to achieve this.

   Solution.

```
SELECT * FROM Apply
ORDER BY FIELD (Experience, 'Govt.-sector-experience', 'Public-sector-experience',
                       'Private-sector-experience', 'Fresher') ;
```

4. *What are different types of SQL functions ?*

   Solution. (*i*) Single Row (or Scalar) functions.

   (*ii*) Multiple Row (or Group or Aggregate) functions.

   (*i*) ***Single Row Functions*** work with a single row at a time. A single row function returns a result for every row of a queried table.

   (*ii*) ***Multiple Row or Group Functions*** work with data of multiple rows at a time and return aggregated value.

5. *What is the significance of GROUP BY clause in a SQL query ?*

   Solution. The GROUP BY clause combines all those records that have identical values in a particular field or a group of fields. This grouping results into one summary record per group if group-functions are used with it.

6. *What is the difference between a WHERE clause and a HAVING clause of SQL SELECT statement ?*

   Solution. The difference between WHERE and HAVING clause is that WHERE conditions are applicable on individual rows whereas HAVING conditions are applicable on groups as formed by GROUP BY clause.

7. *Write a query to display the Sum, Average, Highest and Lowest salary of the employees.*

   Solution.

```
mysql> SELECT SUM (sal), AVG (sal), MAX (sal), MIN (sal)
       FROM empl ;
```

8. *Write a query to display the Sum, Average, Highest and Lowest salary of the employees grouped by department number.*

   Solution.

```
mysql> SELECT SUM (sal), AVG (sal), MAX (sal), MIN (sal)
          FROM empl GROUP BY deptno ;
```

9. *Write a query to display the Sum, Average, Highest and Lowest salary of the employees grouped by department number and sub-grouped by job.*

   Solution.

```
mysql> SELECT SUM (sal), AVG (sal), MAX (sal), MIN (sal)
       FROM empl
       GROUP BY deptno, job ;
```

10. *Write a query to display the number of employees with same job.*

Solution.

```
mysql> SELECT COUNT(*) "No_of_Emps", job
       FROM emp
       GROUP BY job ;
```

11. *Write a query to display the difference of highest and lowest salary of each department having maximum salary > 4000.*

Solution.

```
mysql> SELECT MAX (sal) - MIN (sal) "Difference" FROM empl
       GROUP BY deptno
       HAVING MAX (sal) > 4000 ;
```

12. *Gopi Krishna is using a table Employee. It has the following columns :* [CBSE D 2014]

```
Code, Name, Salary, Deptcode
```

He wants to display maximum salary departmentwise. He wrote the following command :

```
SELECT Deptcode, Max(Salary) FROM Employee ;
```

But he did not get the desired result.

Rewrite the above query with necessary changes to help him get the desired output.

Solution.

```
SELECT  Deptcode, Max(Salary)
FROM Employee
GROUP BY Deptcode ;
```

13. *Shanya Khanna is using a table Employee. It has the following columns :* [CBSE OD 2014]

```
Admno, Name, Agg, Stream
```

[column Agg contains Aggregate marks]

She wants to display highest Agg obtained in each Stream.

She wrote the following statement :

```
SELECT Stream, MAX(Agg) FROM Employee ;
```

But she did not get the desired result. Rewrite the above query with necessary changes to help her get the desired output.

Solution.

```
SELECT Stream, MAX(Agg)
FROM Employee
GROUP BY Stream ;
```

14. *Write a query that counts the number of salespeople registering orders for each day. (If a salesperson has more than one order on a given day, he or she should be counted only once.).*

Solution.

```
SELECT ord_date, count (DISTINCT salesman_code) FROM orders
GROUP BY ord_date ;
```

15. *Write a query on the customers table that will find the highest rating in each city. Put the output in this form : For the city (city), the highest rating is : (rating).*

Solution.

```
SELECT 'For the city', city, 'the highest rating is :', MAX (rating)
FROM customers
GROUP BY city ;
```

GLOSSARY

Single Row function	A function that works on a value in single row.
Multiple Row function	A function that works on values of a group of rows.
Group function	Function that works on a group of rows. A multiple row function.
Aggregate function	Function that works on aggregate of rows. A multiple row function.

Assignment

Type A : Short Answer Questions/Conceptual Questions

1. What is the use of ORDER BY clause?
2. What is the default sort order of ORDER BY clause?
3. Which function do you use in ORDER BY clauses to specify custom sort order?
4. Write an example query that sorts on three columns.
5. Write a query that sorts the data of table **student** on the basis of Project-Group (in ascending order), section (in descending order), Marks (in descending order).
6. What is the difference between HAVING and WHERE clause?
7. What is the use of GROUP BY clause?
8. What are aggregate functions? What is their use? Give some examples.
9. What type of functions can you use with GROUP BY and HAVING clauses?

Type B : Application Based Questions

Following questions are based on these tables :

Table ***BOOK_INFORMATION***

Column Name
BOOK_ID
BOOK_TITLE
PRICE

Table ***SALES***

Column Name
STORE_ID
SALES_DATE
SALES_AMOUNT

Table ***EXAM_RESULTS***

STU_ID	FNAME	LNAME	EXAM_ID	EXAM_SCORE
10	LAURA	LYNCH	1	90
10	LAURA	LYNCH	2	85
11	GRACE	BROWN	1	78
11	GRACE	BROWN	2	72
12	JAY	JACKSON	1	95
12	JAY	JACKSON	2	92
13	WILLIAM	BISHOP	1	70
13	WILLIAM	BISHOP	2	100
14	CHARLES	PRADA	2	85

1. Which SQL statement allows you to find the highest price from the table BOOK_INFORMATION?
   (a) SELECT BOOK_ID, BOOK_TITLE, MAX(PRICE) FROM BOOK_INFORMATION;
   (b) SELECT MAX(PRICE) FROM BOOK_INFORMATION;
   (c) SELECT MAXIMUM(PRICE) FROM BOOK_INFORMATION;
   (d) SELECT PRICE FROM BOOK_INFORMATION ORDER BY PRICE DESC;

2. Which SQL statement lets you find the sales amount for each store?
   (a) SELECT STORE_ID, SUM(SALES_AMOUNT) FROM SALES;
   (b) SELECT STORE_ID, SUM(SALES_AMOUNT) FROM SALES ORDER BY STORE_ID;
   (c) SELECT STORE_ID, SUM(SALES_AMOUNT) FROM SALES GROUP BY STORE_ID;
   (d) SELECT STORE_ID, SUM(SALES_AMOUNT) FROM SALES HAVING UNIQUE STORE_ID;

3. Which SQL statement lets you list all stores whose total sales amount is over 5000 ?
   (a) SELECT STORE_ID, SUM(SALES_AMOUNT) FROM SALES GROUP BY STORE_ID HAVING
   SUM(SALES_AMOUNT) > 5000;
   (b) SELECT STORE_ID, SUM(SALES_AMOUNT) FROM SALES
   GROUP BY STORE_ID HAVING SALES_AMOUNT > 5000;
   (c) SELECT STORE_ID, SUM(SALES_AMOUNT) FROM SALES
   WHERE SUM(SALES_AMOUNT) > 5000 GROUP BY STORE_ID;
   (d) SELECT STORE_ID, SUM(SALES_AMOUNT) FROM SALES
   WHERE SALES_AMOUNT > 5000 GROUP BY STORE_ID;

4. Which SQL statement lets you find the total number of stores in the SALES table?
   (a) SELECT COUNT(STORE_ID) FROM SALES;
   (b) SELECT COUNT(DISTINCT STORE_ID) FROM SALES;
   (c) SELECT DISTINCT STORE_ID FROM SALES;
   (d) SELECT COUNT(STORE_ID) FROM SALES GROUP BY STORE_ID;

5. Which SQL statement allows you to find the total sales amount for Store ID 25 and the total sales amount for Store ID 45?
   (a) SELECT STORE_ID, SUM(SALES_AMOUNT) FROM SALES
   WHERE STORE_ID IN (25,45) GROUP BY STORE_ID;
   (b) SELECT STORE_ID, SUM(SALES_AMOUNT) FROM SALES
   GROUP BY STORE_ID HAVING STORE_ID IN (25,45);
   (c) SELECT STORE_ID, SUM(SALES_AMOUNT) FROM SALES WHERE STORE_ID IN (25,45);
   (d) SELECT STORE_ID, SUM(SALES_AMOUNT) FROM SALES
   WHERE STORE_ID = 25 AND STORE_ID = 45 GROUP BY STORE_ID;

6. What SQL statement do we use to find the average exam score for EXAM_ID = 1?
   (a) SELECT AVG(EXAM_SCORE) FROM EXAM_RESULTS;
   (b) SELECT AVG(EXAM_SCORE) FROM EXAM_RESULTS GROUP BY EXAM_ID WHERE EXAM_ID = 1;
   (c) SELECT AVG(EXAM_SCORE) FROM EXAM_RESULTS GROUP BY EXAM_ID HAVING EXAM_ID = 1;
   (d) SELECT COUNT(EXAM_SCORE) FROM EXAM_RESULTS WHERE EXAM_ID = 1;

7. Which SQL statement do we use to find out how many students took each exam?

   (*a*) SELECT COUNT(DISTINCT STU_ID) FROM EXAM_RESULTS GROUP BY EXAM_ID;

   (*b*) SELECT EXAM_ID, MAX(STU_ID) FROM EXAM_RESULTS GROUP BY EXAM_ID;

   (*c*) SELECT EXAM_ID, COUNT(DISTINCT STU_ID) FROM EXAM_RESULTS GROUP BY EXAM_ID;

   (*d*) SELECT EXAM_ID, MIN(STU_ID) FROM EXAM_RESULTS GROUP BY EXAM_ID;

8. What SQL statement do we use to print out the record of all students whose last name starts with 'L'?

   (*a*) SELECT * FROM EXAM_RESULTS WHERE LNAME LIKE 'L%';

   (*b*) SELECT * FROM EXAM_RESULTS WHERE LNAME LIKE 'L';

   (*c*) SELECT * FROM EXAM_RESULTS WHERE LNAME = 'L';

   (*d*) SELECT * FROM EXAM_RESULTS WHERE LNAME <> 'L';

9. What is the result of the following SQL statement?

   SELECT MAX(EXAM_SCORE) FROM EXAM_RESULTS GROUP BY EXAM_ID HAVING EXAM_ID = 1;

   (*a*) 90 (*b*) 85 (*c*) 100 (*d*) 95

10. Given the following table :

Table : CLUB

COACH-ID	COACHNAME	AGE	SPORTS	DATOFAPP	PAY	SEX
1.	KUKREJA	35	KARATE	27/03/1996	1000	M
2.	RAVINA	34	KARATE	20/01/1998	1200	F
3.	KARAN	34	SQUASH	19/02/1998	2000	M
4.	TARUN	33	BASKETBALL	01/01/1998	1500	M
5.	ZUBIN	36	SWIMMING	12/01/1998	750	M
6.	KETAKI	36	SWIMMING	24/02/1998	800	F
7.	ANKITA	39	SQUASH	20/02/1998	2200	F
8.	ZAREEN	37	KARATE	22/02/1998	1100	F
9.	KUSH	41	SWIMMING	13/01/1998	900	M
10.	SHAILYA	37	BASKETBALL	19/02/1998	1700	M

Give the output of following SQL statements :

(*i*) SELECT COUNT (DISTINCT SPORTS) FROM Club ;

(*ii*) SELECT MIN(Age) FROM CLUB WHERE Sex = 'F' ;

(*iii*) SELECT AVG(Pay) FROM CLUB WHERE Sports = 'KARATE' ;

(*iv*) SELECT SUM(Pay) FROM CLUB WHERE Datofapp > '31/01/98' ;

11. Given the following table :

Table : STUDENT

No.	Name	Stipend	Stream	AvgMark	Grade	Class
1	Karan	400.00	Medical	78.5	B	12B
2	Divakar	450.00	Commerce	89.2	A	11C
3	Divya	300.00	Commerce	68.6	C	12C
4	Arun	350.00	Humanities	73.1	B	12C
5	Sabina	500.00	Nonmedical	90.6	A	11A
6	John	400.00	Medical	75.4	B	12B
7	Robert	250.00	Humanities	64.4	C	11A
8	Rubina	450.00	Nonmedical	88.5	A	12A
9	Vikas	500.00	Nonmedical	92.0	A	12A
10	Mohan	300.00	Commerce	67.5	C	12C

Give the output of following SQL statements :

(*i*) SELECT MIN(AvgMark) FROM STUDENT WHERE AvgMark < 75 ;

(*ii*) SELECT SUM(Stipend) FROM Student WHERE Grade = 'B' ;

(*iii*) SELECT AVG(Stipend) FROM Student WHERE Class = '12A' ;

(*iv*) SELECT COUNT(DISTINCT) FROM Student ;

12. *In a Database, there are two tables given below :*

Table : EMPLOYEE

EMPLOYEEID	NAME	SALES	JOBID
E1	SUMIT SINHA	1100000	102
E2	VIJAY SINGH TOMAR	1300000	101
E3	AJAY RAJPAL	1400000	103
E4	MOHIT RAMNANI	1250000	102
E5	SHAILJA SINGH	1450000	103

Table : JOB

JOBID	JOBTITLE	SALARY
101	President	200000
102	Vice President	125000
103	Administration Assistant	80000
104	Accounting Manager	70000
105	Accountant	65000
106	Sales Manager	80000

Write SQL Queries for the following :

(*i*) *To display employee ids, names of employees, job ids with corresponding job titles.*

(*ii*) *To display names of employees, sales and corresponding job titles who have achieved sales more than 1300000.*

(*iii*) *To display names and corresponding job titles of those employees who have 'SINGH' (anywhere) in their names.*

(*iv*) *Identify foreign key in the table EMPLOYEE.*

(*v*) Write SQL command to change the JOBID to 104 of the EMPLOYEE with ID as E4 in the table 'EMPLOYEE'.

[CBSE D 15]

13. Show the average salary for all departments with more than 3 people for a job.

14. Display only the jobs with maximum salary greater than or equal to 3000.

15. Find out number of employees having "Manager" as Job.

16. List the count of employees grouped by deptno. (table EMPL)

17. List the sum of employees' salaries grouped by department. (table EMPL)

18. List the maximum salary of employee grouped by their department number.

Consider the tables Customers, Parts and Orders given in solved problems.
Answer Questions 19 to 24 based on these :

19. List the total of customers' orders grouped by customer (id).
20. List all customers (name) who have orders (use EXISTS).
21. List the sum of the totals of orders grouped by customer and state.
22. List the sum of the totals of orders where this sum is greater than $1000 grouped by customer (id) an state and ordered by state.
23. List the customers (name) and their orders' details.
24. List the customers (name) and the total amount of all their orders.

Consider tables EMPL, Dept, SalaryGrade. The Empl table has already been listed in earlier chapters. Schemas of tables SalaryGrade and Dept are being shown below :

```
SALARYGRADE (Lowsal, Highsal, Grade)
Dept (Deptno, DeptName, Location)
```

Answer Questions 25 and 26 on the basis of these tables :

25. List the department names and the number of their employees.
26. List the employee names and the name of their departments.