Group Functions (Aggregate Functions)

Overview

Many requests require statistical analysis on the results returned. The SQL standard has included aggregate functions that combine all results to perform a statistical operation that returns a single value. The aggregate functions we will discuss are COUNT, SUM, AVG, MAX, and MIN. These functions are valuable tools for those who understand how to use them.

The powerful statistical tool of aggregate functions is greatly enhanced with the capability to group subcategories of data. Without that capability, subcategories of data would have to be gathered using a separate SQL statement for each subcategory. Fortunately, the SQL standard has provided a way to group data into subcategories. The GROUP BY clause enables users to group data for statistical analysis using aggregate functions. This timesaving tool is vital for database users to understand.

Objectives

- Use the SUM and AVG functions for numeric calculations.
- Use the COUNT function to return the number of records containing non-NULL values.
- Use the MIN and MAX functions with nonnumeric fields.
- Determine when to use the GROUP BY clause to group data.
- Explain when the HAVING clause should be used.
- Nest a group function inside a single-row function.

1. Types of Functions

There are two main types of MySQL database functions:

1. **Aggregate functions** operate on multiple rows at the same time and return one row of output. An example aggregate function is AVG(x), which returns the average value of x.
   Aggregate functions, also called *column functions*, perform a calculation on the values in a set of selected rows.
2. **Single-row functions** operate on one row at a time and return one row of output for each input row. An example single-row function is CONCAT(x, y) which appends y to x and returns the resulting string.

2. Aggregate Functions

Aggregate functions are functions that take a collection (a set or multi-set) of values as input and return a single value. Aggregate functions return results based on groups of rows. By default, the entire result is treated as one group. SQL offers five built-in aggregate functions.

FUNCTION	OUTPUT
COUNT	The number of rows containing non-null values
MIN	The minimum attribute value encountered in a given column
MAX	The maximum attribute value encountered in a given column
SUM	The sum of all values for a given column
AVG	The arithmetic mean (average) for a specified column

Table 1 Basic SQL Aggregate Functions

Syntax of aggregate functions:

```
AVG([ALL|DISTINCT] expression)
SUM([ALL|DISTINCT] expression)
MIN([ALL|DISTINCT] expression)
MAX([ALL|DISTINCT] expression)
COUNT([ALL|DISTINCT] expression)
COUNT(*)
```

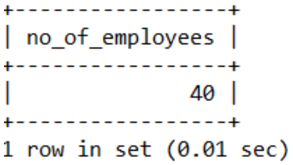
- The SUM and AVG functions apply to numeric data only, whereas the COUNT, MAX, and MIN functions can be used for any data types.
- An asterisk is used as the argument for the COUNT function to include rows containing null values. null values are ignored by the other functions.

2.1. COUNT

The COUNT function is used to tally the number of non-null values of an attribute. COUNT can be used in conjunction with the DISTINCT clause.

Example 1: Display the total number of employees in the company.

```
mysql> SELECT count(*) AS no_of_employees
       FROM employee;
```



```
+-----------------+
| no_of_employees |
+-----------------+
|              40 |
+-----------------+
1 row in set (0.01 sec)
```

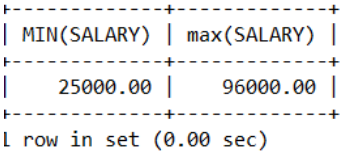
COUNT always returns the number of non-null values in the given column. Another use for the COUNT function is to display the number of rows returned by a query, including the rows that contain rows using the syntax COUNT(*). All of the aggregate functions except for COUNT(*) ignore null values.

2.2. MAX and MIN

The MAX and MIN functions are used to find answers to problems such as what is the highest and lowest salary in all departments.

Example 2: Count the number of distinct salary values in the database.

```
mysql> SELECT MIN(SALARY),max(SALARY)
       FROM employee;
```



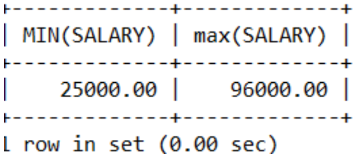
```
+-------------+-------------+
| MIN(SALARY) | max(SALARY) |
+-------------+-------------+
|    25000.00 |    96000.00 |
+-------------+-------------+
1 row in set (0.00 sec)
```

2.3. SUM and AVG

The SUM function computes the total sum for any specified attribute, using whatever condition(s) you have imposed. The AVG function calculates the arithmetic mean (average) for a specified attribute.

Example 3: The following query displays the average salary.

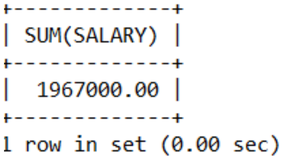
```
mysql> SELECT AVG(SALARY)
       FROM employee;
```



```
+-------------+-------------+
| MIN(SALARY) | max(SALARY) |
+-------------+-------------+
|    25000.00 |    96000.00 |
+-------------+-------------+
1 row in set (0.00 sec)
```

Example 4: Display the sum of the salaries of all employees.

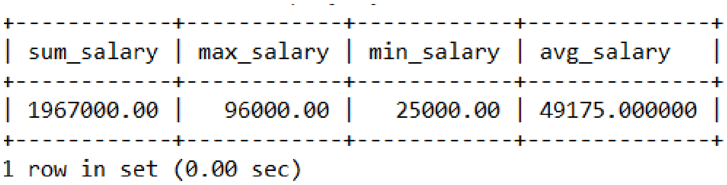
```
mysql> SELECT SUM(SALARY)
       FROM employee;
```



```
+-------------+
| SUM(SALARY) |
+-------------+
|  1967000.00 |
+-------------+
1 row in set (0.00 sec)
```

Example 5: Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.

```
mysql> SELECT SUM(salary) AS sum_salary
       , MAX(salary) AS max_salary
       ,MIN(salary) AS min_salary
       , AVG(salary) AS avg_salary
       FROM employee;
```



```
+------------+------------+------------+--------------+
| sum_salary | max_salary | min_salary | avg_salary   |
+------------+------------+------------+--------------+
| 1967000.00 |   96000.00 |   25000.00 | 49175.000000 |
+------------+------------+------------+--------------+
1 row in set (0.00 sec)
```

3. Aggregation with Grouping Rows

There are circumstances where we would like to apply the aggregate function not only to a single set of tuples, but also to a group of sets of tuples. We specify this wish in SQL using the group by clause. The attribute or attributes given in the group by clause are used to form groups. Tuples with the same value on all attributes in the group by clause are placed in one group.

3.1. The GROUP BY Clause

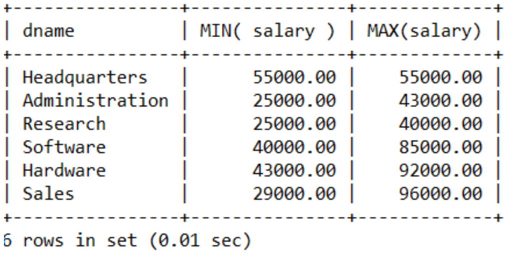
The GROUP BY clause is generally used when you have attribute columns combined with aggregate functions in the SELECT statement. It is valid only when used in conjunction with one of the SQL aggregate functions, such as COUNT, MIN, MAX, AVG and SUM. The GROUP BY clause appears after the WHERE statement.

Syntax of a SELECT statement with GROUP BY:

```
SELECT *|columnlist
FROM tablelist
[WHERE search_condition]
[GROUP BY columnname, columnname...]
[HAVING group condition]
[ORDER BY order_by_list]
```

Example 6: The following query displays the minimum and maximum salary of all departments. Notice that the query groups only by the department number, as no aggregate function is applied to this attribute in the SELECT statement.

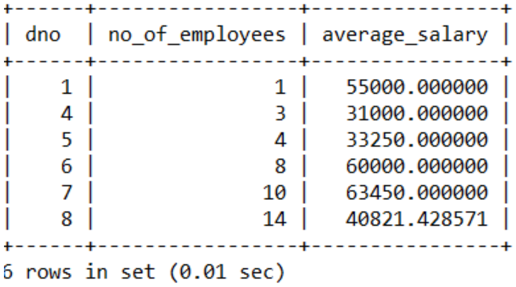
```
mysql> SELECT d.dname, MIN( salary ) , MAX(salary)
       FROM employee e JOIN department d
       ON e.dno = d.dnumber
       GROUP BY e.dno;
```



```
+----------------+--------------+-------------+
| dname          | MIN( salary ) | MAX(salary) |
+----------------+--------------+-------------+
| Headquarters   |     55000.00 |    55000.00 |
| Administration |     25000.00 |    43000.00 |
| Research       |     25000.00 |    40000.00 |
| Software       |     40000.00 |    85000.00 |
| Hardware       |     43000.00 |    92000.00 |
| Sales          |     29000.00 |    96000.00 |
+----------------+--------------+-------------+
6 rows in set (0.01 sec)
```

Example 7: For each department, retrieve the department name, the number of employees in the department, and their average salary. Write this query as follows:

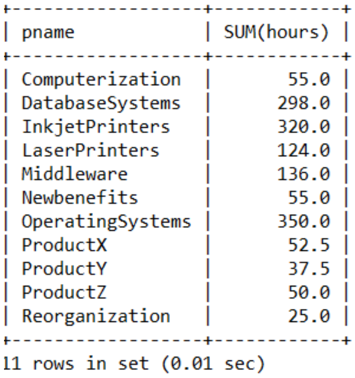
```
mysql> SELECT e.dno,count(*) AS no_of_employees
       ,avg(salary)AS average_salary
       FROM employee e JOIN department d
       ON (e.dno = d.dnumber)
       GROUP BY e.dno;
```



```
+------+-----------------+----------------+
| dno  | no_of_employees | average_salary |
+------+-----------------+----------------+
|    1 |               1 |   55000.000000 |
|    4 |               3 |   31000.000000 |
|    5 |               4 |   33250.000000 |
|    6 |               8 |   60000.000000 |
|    7 |              10 |   63450.000000 |
|    8 |              14 |   40821.428571 |
+------+-----------------+----------------+
6 rows in set (0.01 sec)
```

Example 8: For each project, list the project name and the total hours per week (by all employees) spent on that project.

```
mysql> SELECT p.pname, SUM(hours)
       FROM project p JOIN works_on a
       ON (p.pnumber = a.pno)
       GROUP BY p.pname;
```



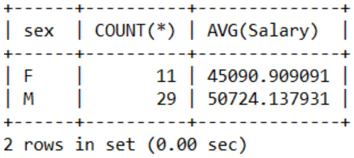
```
+-----------------+------------+
| pname           | SUM(hours) |
+-----------------+------------+
| Computerization |       55.0 |
| DatabaseSystems |      298.0 |
| InkjetPrinters  |      320.0 |
| LaserPrinters   |      124.0 |
| Middleware      |      136.0 |
| Newbenefits     |       55.0 |
| OperatingSystems|      350.0 |
| ProductX        |       52.5 |
| ProductY        |       37.5 |
| ProductZ        |       50.0 |
| Reorganization  |       25.0 |
+-----------------+------------+
11 rows in set (0.01 sec)
```

When using the GROUP BY clause, remember the following:

- Columns used to group data in the GROUP BY clause do not have to be listed in the SELECT clause. They are included in the SELECT clause only to have the groups identified in the output.
- Column aliases cannot be used in the GROUP BY clause.
- Results returned from a SELECT statement that includes a GROUP BY clause will present the results in ascending order of the column(s) listed in the GROUP BY clause. To present the results in a different sort sequence, use the ORDER BY clause.
- When a SELECT statement includes a GROUP BY clause, the SELECT clause can include the columns used for grouping, aggregate functions, and expression that result in a constant value.

Example 9: For each department, retrieve the gender, the number of employees in the company, and employee average salary.

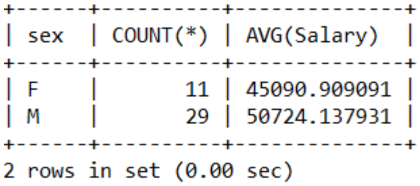
```
mysql> SELECT sex, COUNT(*), AVG(Salary)
       FROM employee
       GROUP BY sex;
```



```
+------+----------+--------------+
| sex  | COUNT(*) | AVG(Salary)  |
+------+----------+--------------+
| F    |       11 | 45090.909091 |
| M    |       29 | 50724.137931 |
+------+----------+--------------+
2 rows in set (0.00 sec)
```

Example 10: For each project, retrieve the project number, the project name, and the number of employees from department 5 who work on the project.

```
mysql> SELECT p.pnumber, p.pname, COUNT(*)
       FROM   project p
       JOIN works_on a ON  p.pnumber = a.pno
       JOIN employee e ON (e.ssn = a.essn)
       WHERE p.dnum = 5
       GROUP BY  p.pnumber, p.pname;
```

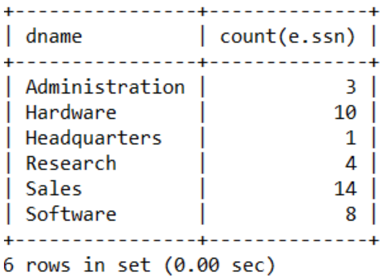


```
+------+----------+--------------+
| sex  | COUNT(*) | AVG(Salary)  |
+------+----------+--------------+
| F    |       11 | 45090.909091 |
| M    |       29 | 50724.137931 |
+------+----------+--------------+
2 rows in set (0.00 sec)
```

If you include two or more columns or expression in the GROUP BY clause, they form a hierarchy where each column or expression is subordinate to the previous one.

Example 11: Display the list of all departments, with the total number of employees in each department.

```
mysql> SELECT d.dname, count(e.ssn)
       FROM department d
       JOIN employee e ON (d.dnumber = e.dno)
       GROUP by d.dname;
```



```
+----------------+--------------+
| dname          | count(e.ssn) |
+----------------+--------------+
| Administration |            3 |
| Hardware       |           10 |
| Headquarters   |            1 |
| Research       |            4 |
| Sales          |           14 |
| Software       |            8 |
+----------------+--------------+
6 rows in set (0.00 sec)
```

3.2. The HAVING Clause

The HAVING clause is an extension to the GROUP BY clause and is applied to the output of a GROUP BY operation. The HAVING clause:

- enables summarizations across the groups of related data within tables.
- determines which groups will be displayed in the result of a query and, consequently, which groups will not be displayed in the result of the query.

A query that contains a HAVING clause ***must*** also contain a GROUP BY clause.

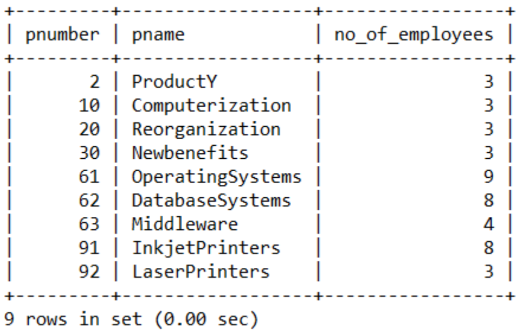
A WHERE clause cannot contain aggregate functions. A HAVING clause can contain aggregate functions.

When a SELECT statement contains WHERE, GROUP BY, and HAVING clauses in the same statement, **they are executed in this order:**

1. WHERE (to restrict rows retrieved from the table) clause,
2. GROUP BY (to group data) clause, and
3. HAVING (to restrict group data displayed in the output) clause.

Example 12: For each project on which more than two employees work, retrieve the project number, project name, and the number of employees who work on the project.

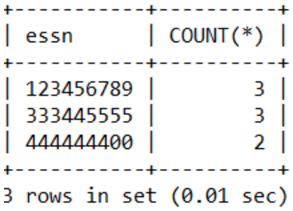
```
mysql> SELECT p.pnumber, p.pname, count(*)AS no_of_employees
       FROM project p
       JOIN works_on a ON (p.pnumber = a.pno)
       GROUP BY p.pnumber, p.pname
       HAVING count(*)>2;
```



```
+---------+------------------+-----------------+
| pnumber | pname            | no_of_employees |
+---------+------------------+-----------------+
|       2 | ProductY         |               3 |
|      10 | Computerization  |               3 |
|      20 | Reorganization   |               3 |
|      30 | Newbenefits      |               3 |
|      61 | OperatingSystems |               9 |
|      62 | DatabaseSystems  |               8 |
|      63 | Middleware       |               4 |
|      91 | InkjetPrinters   |               8 |
|      92 | LaserPrinters    |               3 |
+---------+------------------+-----------------+
9 rows in set (0.00 sec)
```

Example 13: Display the total number of dependents for each employee with at least two dependents.

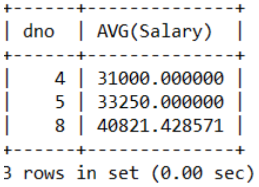
```
mysql> SELECT essn, COUNT(*)
       FROM dependent
       GROUP BY essn
       HAVING COUNT(*) >= 2;
```



```
+-----------+----------+
| essn      | COUNT(*) |
+-----------+----------+
| 123456789 |        3 |
| 333445555 |        3 |
| 444444400 |        2 |
+-----------+----------+
3 rows in set (0.01 sec)
```

Example 14: Display average employee salary by department. Do not include departments with an average salary less than or equal to $50,000.

```
mysql> SELECT dno, AVG(Salary)
       FROM employee
       GROUP BY dno
       HAVING AVG(Salary) <= 50000;
```



```
+------+--------------+
| dno  | AVG(Salary)  |
+------+--------------+
|    4 | 31000.000000 |
|    5 | 33250.000000 |
|    8 | 40821.428571 |
+------+--------------+
3 rows in set (0.00 sec)
```

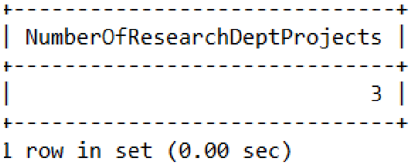
4. Null Values and Aggregates

Aggregate functions treat nulls according to the following rules:

- All aggregate functions except **count (*)** ignore null values in their input collection. As a result of null values being ignored, the collection of values may be empty.
- The count of an empty collection is defined to be 0, and all other aggregate operations return a null value when applied on an empty collection.

Example 15: How many projects are being run by the Research Department? Be sure to assign an appropriate column name to the computed results.

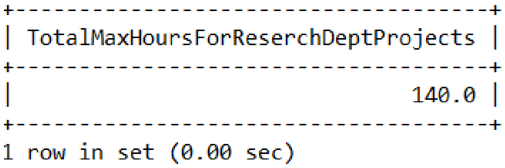
```
mysql> SELECT COUNT(*) AS NumberOfResearchDeptProjects
       FROM project JOIN department ON (dnumber = dnum)
       WHERE dname ='Research';
```



```
+------------------------------+
| NumberOfResearchDeptProjects |
+------------------------------+
|                            3 |
+------------------------------+
1 row in set (0.00 sec)
```

Example 16: What is the total MaxHours of projects being run by the Research Department? Be sure to assign an appropriate column name to the computed results.

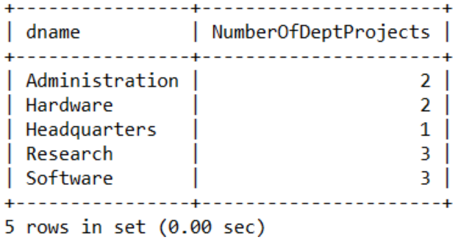
```
mysql> SELECT SUM(hours) AS TotalMaxHoursForReserchDeptProjects
       FROM department
       JOIN project ON (dnumber = dnum)
       JOIN works_on ON (pnumber = pno)
       WHERE dname ='Research';
```



```
+-------------------------------------+
| TotalMaxHoursForReserchDeptProjects |
+-------------------------------------+
|                               140.0 |
+-------------------------------------+
1 row in set (0.00 sec)
```

Example 17: How many projects are being run by ech department? Be sure to display each department name and to assign an appropriate column name to the computed results.

```
mysql> SELECT dname, COUNT(*) AS NumberOfDeptProjects
       FROM PROJECT JOIN department ON(dnumber = dnum)
       GROUP BY dname;
```

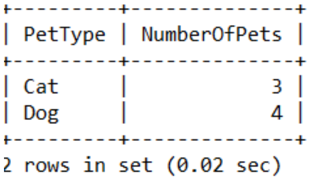


```
+----------------+----------------------+
| dname          | NumberOfDeptProjects |
+----------------+----------------------+
| Administration |                    2 |
| Hardware       |                    2 |
| Headquarters   |                    1 |
| Research       |                    3 |
| Software       |                    3 |
+----------------+----------------------+
5 rows in set (0.00 sec)
```

We will use the Pet database for the following examples:

Example 18: Write a SQL statement that shows each type (that is, dog or cat) and the number of each type (that is, how many dogs and how many cats) in the database.

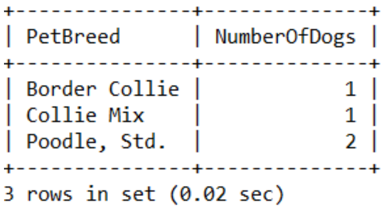
```
mysql> SELECT PetType, COUNT(PetType) AS NumberOfPets
       FROM PET
       GROUP BY PetType;
```



```
+---------+--------------+
| PetType | NumberOfPets |
+---------+--------------+
| Cat     |            3 |
| Dog     |            4 |
+---------+--------------+
2 rows in set (0.02 sec)
```

Example 19: Write a SQL statement to display each Breed of dog and the number of each Breed in the database.

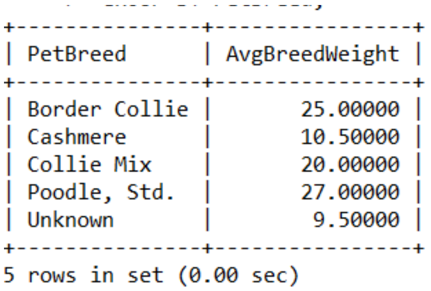
```
mysql> SELECT PetBreed, COUNT(PetBreed) AS NumberOfDogs
       FROM PET_3
       WHERE PetType = 'Dog'
      GROUP BY PetBreed;
```



```
+---------------+--------------+
| PetBreed      | NumberOfDogs |
+---------------+--------------+
| Border Collie |            1 |
| Collie Mix    |            1 |
| Poodle, Std.  |            2 |
+---------------+--------------+
3 rows in set (0.02 sec)
```

Example 20: Write a SQL statement to group the data by PetBreed and display the average Weight per breed.

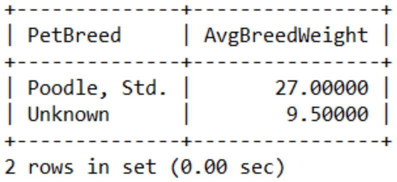
```
mysql> SELECT PetBreed, AVG(PetWeight) AS AvgBreedWeight
       FROM pet_3
       GROUP BY PetBreed;
```



```
+---------------+----------------+
| PetBreed      | AvgBreedWeight |
+---------------+----------------+
| Border Collie |       25.00000 |
| Cashmere      |       10.50000 |
| Collie Mix    |       20.00000 |
| Poodle, Std.  |       27.00000 |
| Unknown       |        9.50000 |
+---------------+----------------+
5 rows in set (0.00 sec)
```

Example 21: Write a SQL statement to group the data by PetBreed and display the average Weight per breed. Consider only breeds for which two or more pets are included in the database.

```
mysql> SELECT PetBreed, AVG(PetWeight) AS AvgBreedWeight
       FROM pet_3
       GROUP BY PetBreed
       HAVING count(*) > 1;
```



```
+--------------+----------------+
| PetBreed     | AvgBreedWeight |
+--------------+----------------+
| Poodle, Std. |       27.00000 |
| Unknown      |        9.50000 |
+--------------+----------------+
2 rows in set (0.00 sec)
```

5. Summary of SQL Queries

A retrieval query in SQL can consist of up to six clauses, but only the first two SELECT and FROM are mandatory. The query can span several lines and ends by a semicolon. Query terms are separated by spaces, and parentheses can be used to group relevant parts of a query in the standard way. The clauses are specified in the following order, with the clauses between square brackets [...] being optional:

```
SELECT <attribute and functionlist>
FROM <table list>
[ WHERE <condition> ]
[ GROUP BY <grouping attribute(s)> ]
[ HAVING <group condition> ]
[ ORDER BY <attribute list> ];
```

- The **SELECT** clause lists the attributes or functions to be retrieved. The **FROM** clause specifies all relations (tables) needed in the query, including joined relations, but not those in nested queries.
- The **WHERE** clause specifies the conditions for selecting the tuples from these relations, including join conditions if needed.
- **GROUP BY** specifies grouping attributes, whereas **HAVING** specifies a condition on the groups being selected rather than on the individual tuples. The built-in aggregate functions **COUNT**, **SUM**, **MIN**, **MAX**, and **AVG** are used in conjunction with grouping, but they can also be applied to all the selected tuples in a query without a GROUP BY clause.
- Finally, **ORDER BY** specifies an order for displaying the result of a query. To formulate queries correctly, it is useful to consider the steps that define the meaning or semantics of each query.
- A query is evaluated conceptually by first applying the FROM clause (to identify all tables involved in the query), followed by the WHERE clause to select and join tuples, and then by GROUP BY and HAVING.

Key Terms

- **AGGREGATE FUNCTION** – special SQL functions that apply to groups of rows and are used to calculate sums, averages, counts, maximum values, and minimum values.
- **AVG** – function that calculates the average value in a numeric range.
- **COUNT** – function that counts the number of rows in a table.
- **GROUP BY clause** – the clause that groups rows based on the specified column.
- **GROUPING** – creates groups of rows that share some common characteristic.
- **HAVING clause** – clause that limits a condition to the groups that are included.
- **IS NOT NULL** – operator used to specify no null values for a column in a query.
- **IS NULL** – operator that specifies null values for a column in a query.
- **MAX** – function that calculates the maximum value in a numeric range.
- **MIN** – function that calculates the minimum value in a numeric range.