GIFT School of Engineering and Applied Sciences

Fall 2022

CS-244: Database Systems-Lab

Lab-12 Manual

Reporting Aggregated Data using Group Functions

Introduction to Lab

This lab further addresses functions. It focuses on obtaining summary information, such as averages, for group of rows. It discusses how to group rows in a table into smaller sets and how to specify the search criteria for group of rows.

The main topics of this lab include:

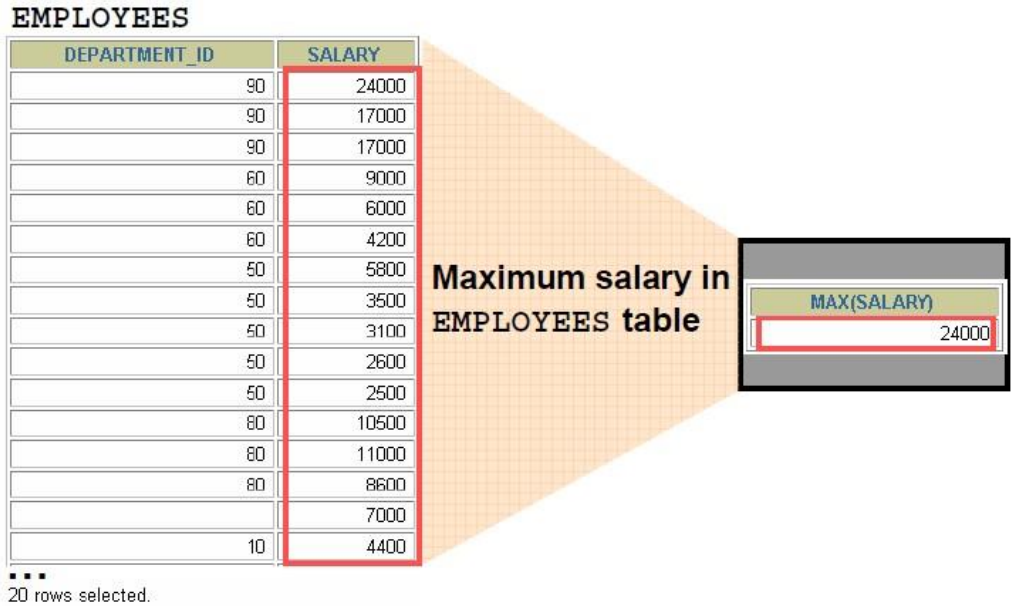
1. What are group functions?
2. Types of group functions
3. Group functions syntax
4. Using group functions
5. Using the COUNT function
6. Group functions and NULL values
7. Creating groups of data (the GROUP BY clause)
8. GROUP BY clause syntax
9. Using the GROUP BY clause
10. Grouping by more than one column
11. Illegal queries using group functions
12. Excluding group results (the HAVING clause)
13. Nesting group functions
14. Practice SQL Statements
15. SOLUTIONS: Practice SQL Statement

Objectives of this Lab

At the end of this lab, students should be able to:

a) Identify the available group functions
b) Describe the use of group functions.
c) Group data by using the GROUP BY clause
d) Include or exclude grouped rows by using the HAVING clause

1. What are Group Functions?



MySQL

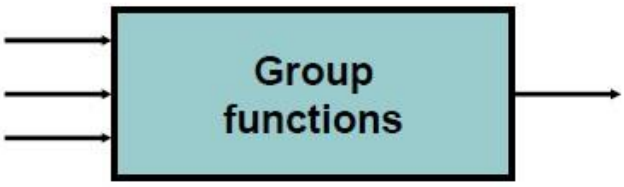
Unlike single-row functions, group functions operate on sets of rows to give one result per group. These sets may be the whole table or the table split into groups.

For example, what is the maximum salary in the employees table?

2. Types of Group Functions

The group functions available are:

- **AVG**
- **COUNT**
- **MAX MIN**
- **STDDEV**
- **SUM**
- **VARIANCE**



Each of the functions accepts an argument. The following table identifies the options that you can use in the syntax:

Function	Description
AVG([DISTINCT|ALL]*n*)	Average value of *n*, ignoring null values
COUNT({*|[DISTINCT|ALL]*expr* })	Number of rows, where *expr* evaluates to something other than null (count all selected rows using *, including duplicates and rows with nulls)

MAX([DISTINCT|ALL]*expr*)	Maximum value of *expr*, ignoring null values
MIN([DISTINCT|ALL]*expr*)	Minimum value of *expr*, ignoring null values
STDDEV([DISTINCT|ALL]*x*)	Standard deviation of *n*, ignoring null values
SUM([DISTINCT|ALL]*n*)	Sum values of *n*, ignoring null values
VARIANCE([DISTINCT|ALL]*x*)	Variance of *n*, ignoring null values

3. Group Functions Syntax

```
SELECT      [column,] group_function(column), ...
FROM        table
[WHERE      condition]
[GROUP BY   column]
[ORDER BY   column];
```

Guidelines for Using Group Functions

- **DISTINCT** makes the function consider only non-duplicate values; **ALL** makes it consider every value, including duplicates. The default is ALL and therefore does not need to be specified.
- The data types for the functions with an expr argument may be **CHAR, VARCHAR2, NUMBER, or DATE.**
- All group functions ignore null values. To substitute a value for null values, use the **NVL, NVL2, or COALESCE** functions.

4. Using Group Functions

You can use AVG, MIN, MAX, and SUM functions against columns that store numeric data. MIN and MAX functions can be used for any data type.

Note: The DISTINCT clause is used to suppress counting of duplicate values in the functions.

Examples of the usage of group functions:

```
1. SELECT AVG (sal), MAX (sal), MIN (sal), SUM (sal) FROM emp;
2. SELECT AVG (sal), MAX (sal), MIN (sal), SUM (sal) FROM emp WHERE
   job LIKE '%MAN';
3. SELECT AVG(ALL sal) FROM emp;
4. SELECT AVG(DISTINCT sal) FROM emp;
5. SELECT MIN(hiredate), MAX(hiredate) FROM emp;
```

The above query displays the most junior and most senior employees.

```
6. SELECT MIN (ALL hiredate), MAX (DISTINCT hiredate) FROM emp;
7. SELECT MIN (DISTINCT hiredate), MAX (DISTINCT hiredate) FROM
   emp;
8. SELECT MIN (ename), MAX (ename) FROM emp;
```

The above query displays the employee name that is first and the employee name that is last in an alphabetized list of all employees.

Note: The AVG, SUM, VARIANCE, and STDDEV functions can be used only with numeric data types.

5. Using the COUNT Function

The COUNT function has three formats:

```
COUNT (*)

COUNT (expr)

COUNT (DISTINCT expr)
```

COUNT (*) returns the number of rows in a table that satisfy the criteria of the SELECT statement, including duplicate rows and rows containing null values in any of the columns. If a WHERE clause is included in the SELECT statement, COUNT (*) returns the number of rows that satisfy the condition in the WHERE clause.

In contrast, COUNT (*expr*)returns the number of non-null values in the column identified by *expr*.

COUNT (DISTINCT *expr*) returns the number of unique, non-null values in the column identified by *expr*.

Examples:

```
1. SELECT COUNT (*) FROM emp;
2. SELECT COUNT (*) FROM emp WHERE deptno = 10;
3. SELECT COUNT (comm) FROM emp;
4. SELECT COUNT (comm) FROM emp WHERE deptno = 30;
5. SELECT COUNT (deptno) FROM emp;
6. SELECT COUNT (DISTINCT deptno) FROM emp;
```

7. Group Functions and Null Values

All group functions ignore null values in the column. For example,

```
SELECT AVG(comm)FROM emp;
```

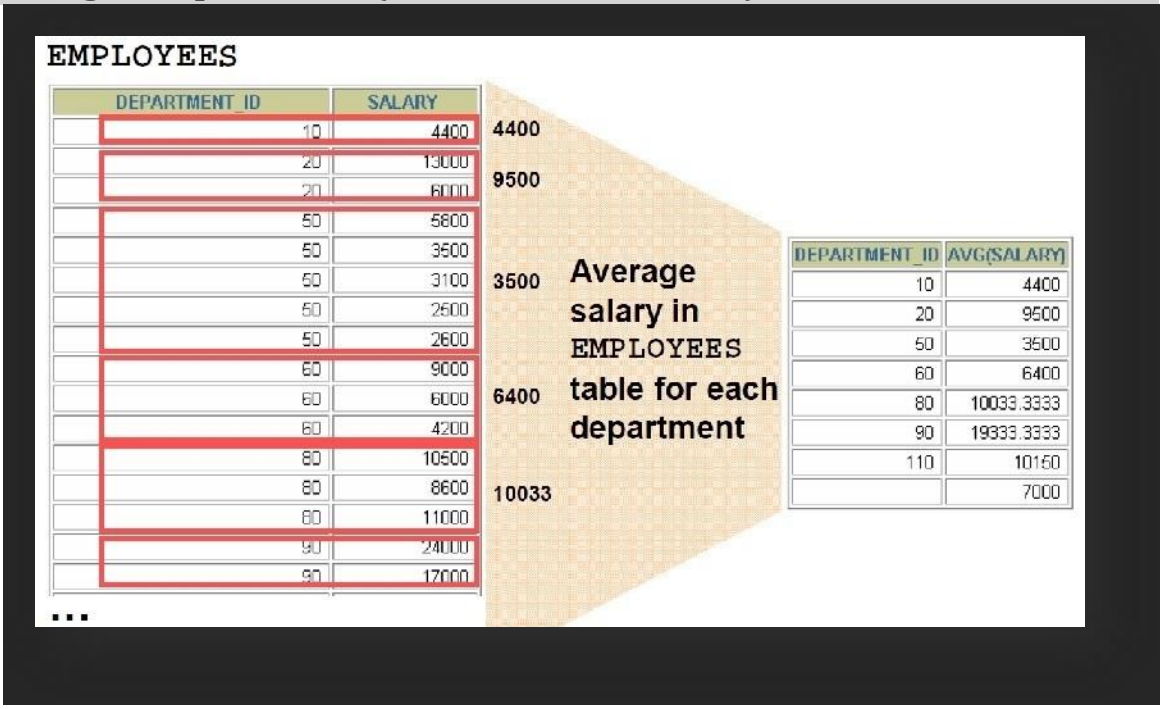
In the above query, the average is calculated based *only* on the rows in the table where a valid value is stored in the COMM column. The average is calculated as the total commission paid to all employees divided by the number of employees receiving commission.

The IFNULL function forces group functions to include null values.

```
SELECT AVG(IFNULL(comm, 0)) FROM emp;
```

The average is calculated based on *all* rows in the table, regardless of whether null values are stored in the COMM column. The average is calculated as the total commission that is paid to all employees divided by the total number of employees in the company.

8. Creating Groups of Data (the GROUP BY clause)



Until now, all group functions have treated the table as one large group of information. At times, you need to divide the table of information into smaller groups. For example, display the average salary in EMP table for each department.

This can be done using the GROUP BY clause.

9. GROUP BY Clause Syntax

```
SELECT        column, group_function(column)
FROM          table
[WHERE        condition]
[GROUP BY     group_by_expression]
[ORDER BY     column];
```

You can use the GROUP BY clause to divide the rows in a table into groups. You can then use the group functions to return summary information for each group.

In the syntax, ***group_by_expression*** specifies columns whose values determine the basis for grouping rows.

Guidelines:

- If you include a group function in a SELECT clause, you cannot select individual results, *unless* the individual column appears in the GROUP BY clause. You receive an error message if you fail to include the column in the GROUP BY clause.
- Using a WHERE clause, you can exclude rows before dividing them into groups.
- You must include the columns in the GROUP BY clause.
- You cannot use a column alias in the GROUP BY clause.
- By default, rows are sorted by ascending order of the columns included in the GROUP BY list. You can override this by using the ORDER BY clause.

10. Using the GROUP BY Clause

When using the GROUP BY clause, make sure that all columns in the SELECT list that are not group functions are included in the GROUP BY clause. For Example,

```
1. SELECT deptno, AVG(sal)FROM emp GROUP BY deptno;
2. SELECT AVG(sal)FROM emp GROUP BY deptno;
```

The GROUP BY column does not have to be in the SELECT list as in query-2 above.

You can use the group function in the ORDER BY clause:

```
SELECT deptno, AVG(sal)
```

```
FROM emp
GROUP BY deptno
ORDER BY AVG(sal);
```

11. Grouping by More Than One Column (Groups within Groups)

Sometimes you need to see results for groups within groups. For example, display the results by adding up salaries in the EMP table for each job, grouped by department.

```
1. SELECT deptno "Dept_ID", job "Job", SUM(sal)FROM emp GROUP BY
   deptno, job;
2. SELECT deptno "Dept_ID", job "Job", SUM(sal)FROM emp GROUP BY
   deptno, job ORDER BY deptno;
```

12. Illegal Queries Using Group Functions

Any column or expression in the SELECT list that is not an aggregate function must be in a GROUP BY clause. Failure to do so will result in an incorrect result. For example,

```
SELECT deptno, COUNT(ename) FROM emp;
```

You can correct the result by adding the GROUP BY clause:

```
SELECT deptno, COUNT(ename)
FROM emp
GROUP BY deptno;
```

Also, the WHERE clause cannot be used to restrict groups. For example, the following query will result in an error:

```
SELECT deptno, AVG(sal)
FROM emp
WHERE AVG(sal) > 1500
GROUP BY deptno;
```

You can correct this error by using the HAVING clause to restrict groups:

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno;
HAVING AVG(sal) > 1500
```

13. Excluding Group Results (the HAVING clause)

In the same way you use the WHERE clause to restrict rows that you select; you use the HAVING to restrict groups. For example, find the maximum salary per department when it is greater than $2000.

```
SELECT        column, group_function
FROM          table
[WHERE        condition]
[GROUP BY     group_by_expression]
[HAVING       group_condition]
[ORDER BY     column];
```

By using the HAVING clause, we can restrict the groups on the basis of aggregate function. In the syntax, the *group_condition* is used to restrict the group of rows returned to those groups for which the specified condition is true.

Returning to the above given example,

```
SELECT deptno, MAX(sal)
FROM emp
GROUP BY deptno
HAVING MAX(sal) > 2000;
```

Note: You can use a GROUP BY clause without using a group function in the SELECT list.

For example,

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno
HAVING MAX(sal) > 2000;
```

And also,

```
SELECT job, SUM(sal) PAYROLL
FROM emp
WHERE job NOT LIKE '%MAN%'
GROUP BY job
HAVING SUM(sal) > 2000
```

```
ORDER BY SUM(sal);
```

14. Practice SQL Statements

1. Display the highest, lowest, sum, and average salary of all employees. Label the columns *Maximum*, *Minimum*, *Sum*, and *Average* respectively. Round your results to the nearest whole number.
2. Modify the above query to display the *minimum*, *maximum*, *sum*, and *average* salary for each job type.
3. Write a query to display the number of people with the same job.
4. Determine the number of managers without listing them. Label the column Number of Managers. **Hint**: Use the MGR column to determine the number of managers.
5. Write a query that displays the difference between the highest and lowest salaries. Label the column **DIFFERENCE.**
6. Display the manager id and the salary of the lowest paid employee for that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is less than $1500. Sort the output in descending order of salary.
7. Write a query to display each department's name, location, number of employees, and the average salary of all employees of that department. Label the columns **Name, Location, Number of People,** and **Salary** respectively.
8. Create a query that will display the total number of employees, and of that total, the number of employees hired in 1980, 1981, 1982, and 1983. Create appropriate column headings.

15. SOLUTIONS: Practice SQL Statements

The End