

EXERCISE-8

Aggregating Data Using Group Functions

Objectives

After the completion of this exercise, the students be will be able to do the following:

- Identify the available group functions
- Describe the use of group functions
- Group data by using the GROUP BY clause
- Include or exclude grouped rows by using the HAVING clause

What Are Group Functions?

Group functions operate on sets of rows to give one result per group

Types of Group Functions

- 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 <i>expr</i> evaluates to something other than null (count all selected rows using *, including duplicates and rows with nulls)
MAX ([DISTINCT ALL] expr)	Maximum value of <i>expr</i> , ignoring null values
MIN ([DISTINCT ALL] expr)	Minimum value of <i>expr</i> , 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

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 nonduplicate 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.

Using the AVG and SUM Functions

You can use AVG and SUM for numeric data.

```
SELECT AVG(salary), MAX(salary),
MIN(salary), SUM(salary)
FROM employees
WHERE job_id LIKE '%REP%';
```

Using the MIN and MAX Functions

You can use MIN and MAX for numeric, character, and date data types.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM employees;
```

You can use the MAX and MIN functions for numeric, character, and date data types. example displays the most junior and most senior employees.

The following example displays the employee last name that is first and the employee last name that is last in an alphabetized list of all employees:

```
SELECT MIN(last_name), MAX(last_name)
FROM employees;
```

Note: The AVG, SUM, VARIANCE, and STDDEV functions can be used only with numeric data types. MAX and MIN cannot be used with LOB or LONG data types.

Using the COUNT Function

COUNT(*) returns the number of rows in a table:

```
SELECT COUNT(*)
FROM employees
WHERE department_id = 50;
COUNT(expr) returns the number of rows with nonnull
values for the expr:
```

```
SELECT COUNT(commission_pct)
FROM employees
WHERE department_id = 80;
```

Using the DISTINCT Keyword

- COUNT(DISTINCT *expr*) returns the number of distinct non-null values of the *expr*.

- To display the number of distinct department values in the EMPLOYEES table:

```
SELECT COUNT(DISTINCT department_id) FROM employees;
```

Use the **DISTINCT** keyword to suppress the counting of any duplicate values in a column.

Group Functions and Null Values

Group functions ignore null values in the column:

```
SELECT AVG(commission_pct)
FROM employees;
```

The **NVL** function forces group functions to include null values:

```
SELECT AVG(NVL(commission_pct, 0))
FROM employees;
```

Creating Groups of Data

To divide the table of information into smaller groups. This can be done by using the **GROUP BY** clause.

GROUP BY Clause Syntax

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

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 as well, *unless* the individual column appears in the **GROUP BY** clause. You receive an error message if you fail to include the column list 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.

Using the GROUP BY Clause

All columns in the **SELECT** list that are not in group functions must be in the **GROUP BY** clause.

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id ;
```

The **GROUP BY** column does not have to be in the **SELECT** list.

```
SELECT AVG(salary) FROM employees GROUP BY department_id ;
```

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

```
SELECT department_id, AVG(salary) FROM employees GROUP BY department_id ORDER BY  
AVG(salary);
```

Grouping by More Than One Column

```
SELECT department_id dept_id, job_id, SUM(salary) FROM employees  
GROUP BY department_id, job_id ;
```

Illegal Queries Using Group Functions

Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP

BY clause:

```
SELECT department_id, COUNT(last_name) FROM employees;
```

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

```
SELECT department_id, count(last_name) FROM employees GROUP BY department_id;
```

You cannot use the WHERE clause to restrict groups.

- You use the HAVING clause to restrict groups.
- You cannot use group functions in the WHERE clause.

```
SELECT department_id, AVG(salary) FROM employees WHERE AVG(salary) > 8000  
GROUP BY department_id;
```

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

```
SELECT department_id, AVG(salary) FROM employees  
HAVING AVG(salary) > 8000 GROUP BY department_id;
```

Restricting Group Results

With the HAVING Clause .When you use the HAVING clause, the Oracle server restricts groups as follows:

1. Rows are grouped.
2. The group function is applied.
3. Groups matching the HAVING clause are displayed.

Using the HAVING Clause

```
SELECT department_id, MAX(salary) FROM employees  
GROUP BY department_id HAVING MAX(salary)>10000 ;
```

The following example displays the department numbers and average salaries for the departments with a maximum salary that is greater than \$10,000:



```
SELECT department_id, AVG(salary) FROM employees GROUP BY department_id  
HAVING max(salary)>10000;
```

Example displays the job ID and total monthly salary for each job that has a total payroll exceeding \$13,000. The example excludes sales representatives and sorts the list by the total monthly salary.

```
SELECT job_id, SUM(salary) PAYROLL FROM employees WHERE job_id NOT LIKE  
'%REP%'  
GROUP BY job_id HAVING SUM(salary) > 13000 ORDER BY SUM(salary);
```

Nesting Group Functions

Display the maximum average salary:

Group functions can be nested to a depth of two. The slide example displays the maximum average salary.

```
SELECT MAX(AVG(salary)) FROM employees GROUP BY department_id;
```

Summary

In this exercise, students should have learned how to:

- Use the group functions COUNT, MAX, MIN, and AVG
- Write queries that use the GROUP BY clause
- Write queries that use the HAVING clause

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

Find the Solution for the following:

Determine the validity of the following three statements. Circle either True or False.

1. Group functions work across many rows to produce one result per group.

True/ False True

2. Group functions include nulls in calculations.

True/False False

3. The WHERE clause restricts rows prior to inclusion in a group calculation.

True/ False True

The HR department needs the following reports:

4. Find 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

```
SELECT ROUND(MAX(salary)) AS "Maximum",  
  
ROUND(MIN(salary)) AS "Minimum",
```

ROUND(SUM(salary)) AS "Sum",

ROUND(AVG(salary)) AS "Average"

FROM employees;

5. Modify the above query to display the minimum, maximum, sum, and average salary for each job type.

```
SELECT job_id,  
       ROUND(MIN(salary)) AS "Minimum",  
       ROUND(MAX(salary)) AS "Maximum",  
       ROUND(SUM(salary)) AS "Sum",  
       ROUND(AVG(salary)) AS "Average"  
FROM employees  
GROUP BY job_id;
```

6. Write a query to display the number of people with the same job. Generalize the query so that the user in the HR department is prompted for a job title.

```
SELECT COUNT(*) AS "Number of People"  
FROM employees  
WHERE job_id = '&job_title';
```

7. Determine the number of managers without listing them. Label the column Number of Managers. Hint: Use the MANAGER ID column to determine the number of managers.

```
SELECT COUNT(DISTINCT manager_id) AS "Number of Managers"  
FROM employees  
WHERE manager_id IS NOT NULL;
```

8. Find the difference between the highest and lowest salaries. Label the column DIFFERENCE.

```
SELECT MAX(salary) - MIN(salary) AS "DIFFERENCE"  
FROM employees;
```

9. Create a report to display the manager number 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 \$6,000 or less. Sort the output in descending order of salary.

```
SELECT manager_id, MIN(salary) AS "Lowest Salary"  
FROM employees  
WHERE manager_id IS NOT NULL  
ORDER BY MIN(salary) DESC;
```



```
GROUP BY manager_id  
HAVING MIN(salary) > 6000  
ORDER BY "Lowest Salary" DESC;
```

10. Create a query to display the total number of employees and, of that total, the number of employees hired in 1995, 1996, 1997, and 1998. Create appropriate column headings.

```
SELECT job_id,  
       SUM(CASE WHEN department_id = 20 THEN salary ELSE 0 END) AS "Dept 20 Salary",  
       SUM(CASE WHEN department_id = 50 THEN salary ELSE 0 END) AS "Dept 50 Salary",  
       SUM(CASE WHEN department_id = 80 THEN salary ELSE 0 END) AS "Dept 80 Salary",  
       SUM(CASE WHEN department_id = 90 THEN salary ELSE 0 END) AS "Dept 90 Salary",  
       SUM(salary) AS "Total Salary"  
FROM employees  
WHERE department_id IN (20, 50, 80, 90)  
GROUP BY job_id;
```


11. Create a matrix query to display the job, the salary for that job based on department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

```
SELECT job_id,
       SUM(CASE WHEN department_id = 20 THEN salary ELSE 0 END) AS "Dept 20 Salary",
       SUM(CASE WHEN department_id = 50 THEN salary ELSE 0 END) AS "Dept 50 Salary",
       SUM(CASE WHEN department_id = 80 THEN salary ELSE 0 END) AS "Dept 80 Salary",
       SUM(CASE WHEN department_id = 90 THEN salary ELSE 0 END) AS "Dept 90 Salary",
       SUM(salary) AS "Total Salary"
FROM employees
WHERE department_id IN (20, 50, 80, 90)
GROUP BY job_id;
```

12. Write a query to display each department's name, location, number of employees, and the average salary for all the employees in that department. Label the column name-Location, Number of people, and salary respectively. Round the average salary to two decimal places.

```
SELECT d.department_name || '-' || l.city AS "Name-Location",
       COUNT(e.employee_id) AS "Number of People",
       ROUND(AVG(e.salary), 2) AS "Salary"
FROM employees e
JOIN departments d ON e.department_id = d.department_id
JOIN locations l ON d.location_id = l.location_id
GROUP BY d.department_name, l.city;
```

Evaluation Procedure	Marks awarded
Query(5)	
Execution (5)	

Viva(5)	
Total (15)	
Faculty Signature	