Reporting Aggregated Data Using the Group Functions

Lesson Objectives

After completing this lesson, you should 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

Lesson Agenda

- Group functions:
  - Types and syntax
  - Use `AVG`, `SUM`, `MIN`, `MAX`, `COUNT`
  - Use `DISTINCT` keyword within group functions
  - `NULL` values in a group function
- Grouping rows:
  - `GROUP BY` clause
  - `HAVING` clause
- Nesting group functions

Lesson Agenda

- **Group functions:**
  - **Types and syntax**
  - **Use `AVG`, `SUM`, `MIN`, `MAX`, `COUNT`**
  - **Use `DISTINCT` keyword within group functions**
  - **`NULL` values in a group function**
- Grouping rows:
  - `GROUP BY` clause
  - `HAVING` clause
- Nesting group functions

What are Group Function ?

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

EMPLOYEES

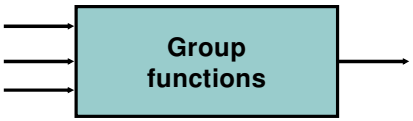
	DEPARTMENT_ID	SALARY
1	90	24000
2	90	17000
3	90	17000
4	60	9000
5	60	6000
6	60	4200
7	50	5800
8	50	3500
9	50	3100
10	50	2600
...		
18	20	6000
19	110	12000
20	110	8300

Maximum salary in EMPLOYEES table

MAX(SALARY)
24000

Types of Group Functions

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 row 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
SUM([DISTINCT|ALL] n)	Sum value of *n*, ignoring null values



Group Function

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

- The group function is placed after the SELECT keyword. You may have multiple group functions separated by commas.
- DISTINCT makes the function consider only nonduplicate values; ALL makes it consider every value, including duplicates.
- The data types for the functions with an column argument may be CHAR, VARCHAR2, NUMBER, or DATE

Using the AVG and SUM Functions

- You can use AVG and SUM for numeric data.
- The example in the slide displays the average, highest, lowest, and sum of monthly salaries for all sales representatives.

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

	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8272.727272727272727272727272727272727273	11500	6000	273000

Using the MAX and MIN Functions

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

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

	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	17-JUN-87	21-APR-00

Using the MAX and MIN Functions

PRACTICE:

Display the employee last name that is first and the employee last name that is last in an alphabetic list of all employees:

	FIRST_LASTNAME	LAST_LASTNAME
1	Abel	Zlotkey

Using COUNT Function

- The COUNT function has three formats:
- First: COUNT(*) returns the number of rows in a table:

  The example in the slide displays the number of employees in department 50.

```
SELECT  COUNT(*)
FROM    employees
WHERE   department_id =  50;
```

	COUNT(*)
1	45

Using COUNT Function

- Second: COUNT(expr) returns the number of rows with non-null values for expr:

  The example in the slide displays the number of employees in department 80 who can earn a commission.

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

	COUNT(COMMISSION_PCT)
1	34

Using COUNT Function

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

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

  The example in the slide displays the number of distinct department values that are in the EMPLOYEES table.

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

	COUNT(DISTINCTDEPARTMENT_ID)
1	11

Group Functions and Null Values

- Group functions ignore null values in the column:
- The average is calculated based on only those rows in the table in which a valid value is stored in the COMMISSION_PCT column. The average is calculate as the total commission that is paid to all employees divided by the number of employees receiving a commission.

```
SELECT  _____________________________
FROM    employees;
```

	AVG(COMMISSION_PCT)
1	0.22285714285714285714285714285714285714

Group Functions and Null Values

- The NVL function forces group functions to include null values:
- The average is calculated based on *all* rows in the table, regardless of whether null values are stored in the COMMISSION_PCT column. The average is calculate as the total commission that is paid to all employees divided by the total number of employees in the company.

```
SELECT  _____________________________
FROM    employees;
```

	AVG(NVL(COMMISSION_PCT,0))
1	0.072897196261682242990654205607476635514

Lesson Agenda

- Group functions:
  - Types and syntax
  - Use AVG, SUM, MIN, MAX, COUNT
  - Use DISTINCT keyword within group functions
  - NULL values in a group function
- **Grouping rows:**
  - **GROUP BY clause**
  - **HAVING clause**
- Nesting group functions

Creating Groups of Data

EMPLOYEES

	DEPARTMENT_ID	SALARY	
1	10	4400	4400
2	20	13000	9500
3	20	6000	
4	50	5800	3500
5	50	2500	
6	50	2600	
7	50	3100	
8	50	3500	
9	60	4200	6400
10	60	6000	
11	60	9000	
12	80	11000	10033
13	80	10500	
14	80	8600	
...			
19	110	12000	
20	(null)	7000	

Average salary in EMPLOYEES table for each department

	DEPARTMENT_ID	AVG(SALARY)
1	10	4400
2	20	9500
3	50	3500
4	60	6400
5	80	10033.333333333333...
6	90	19333.333333333333...
7	110	10150
8	(null)	7000

GROUP BY Clause Syntax

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

- You can divide rows in a table into smaller groups by using the GROUP BY clause.
- group_by_expression:
- specifies columns whose values determine the basis for grouping rows

Using the GROUP BY Clause

- All columns in the SELECT list that are not in group functions must be in the GROUP BY clause.
- The example in the slide displays the department number and the average salary for each department.

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

	DEPARTMENT_ID	AVG(SALARY)
1	100	8600
2	30	4150
3	(null)	7000
4	90	19333.333333333333333333333333333333333
5	20	9500
6	70	10000
7	110	10150
8	50	3475.555555555555555555555555555555555556
9	80	8955.882352941176470588235294117647058824
10	40	6500
11	60	5760
12	10	4400

Using the GROUP BY Clause

- The GROUP BY column does not have to be in the SELECT list.
- The GROUP BY column does not have to be in the SELECT clause. For example, the SELECT statement in the slide displays the average salaries for each department without displaying the respective department numbers. Without the department numbers, however, the results do not look meaningful.

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

	AVG(SALARY)
1	8600
2	4150
3	7000
4	19333.333333333333333333333333333333333
5	9500
6	10000
7	10150
8	3475.555555555555555555555555555555555556
9	8955.882352941176470588235294117647058824
10	6500
11	5760
12	4400

Using the GROUP BY Clause

PRACTICE:

Modify the previous slide use the group function in the ORDER BY clause and format this follow:

	Averag Salary
1	3,475.56
2	4,150.00
3	4,400.00
4	5,760.00
5	6,500.00

Display 12 rows.

Grouping by More than One Column

EMPLOYEES

	DEPARTMENT_ID	JOB_ID	SALARY
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_MAN	5800
5	50	ST_CLERK	2500
6	50	ST_CLERK	2600
7	50	ST_CLERK	3100
8	50	ST_CLERK	3500
9	60	IT_PROG	4200
10	60	IT_PROG	6000
11	60	IT_PROG	9000
12	80	SA_REP	11000
13	80	SA_MAN	10500
14	80	SA_REP	8600
...			
19	110	AC_MGR	12000
20	(null)	SA_REP	7000

Add the salaries in the EMPLOYEES table for each job, grouped by department.

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	50	ST_CLERK	11700
5	50	ST_MAN	5800
6	60	IT_PROG	19200
7	80	SA_MAN	10500
8	80	SA_REP	19600
9	90	AD_PRES	24000
10	90	AD_VP	34000
11	110	AC_ACCOUNT	8300
12	110	AC_MGR	12000
13	(null)	SA_REP	7000

Using the GROUP BY Clause on Multiple Columns

- In the example in the slide, the SELECT statement containing a GROUP BY clause is evaluated as follows:
- The SELECT clause specifies the column to be retrieved:
  - Department number in the EMPLOYEES table
  - Job ID in the EMPLOYEES table
  - The sum of all salaries in the group that you specified in the GROUP BY clause
- The FROM clause specifies the tables that the database must access: the EMPLOYEES table
- The GROUP BY clause specifies how you must group the rows:
  - First, the rows are grouped by the department number.
  - Second, the rows are grouped by job ID in the department number groups.

Using the GROUP BY Clause on Multiple Columns

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

	DEPT_ID	JOB_ID	SUM(SALARY)
1	10	AD_ASST	4400
2	20	MK_MAN	13000
3	20	MK_REP	6000
4	30	PU_CLERK	13900
5	30	PU_MAN	11000
6	40	HR_REP	6500
7	50	SH_CLERK	64300
8	50	ST_CLERK	55700
9	50	ST_MAN	36400

Display 20 rows.

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;
```

ORA-00937: not a single-group group function
00937. 00000 - "not a single-group group function"
*Cause:
*Action:
Error at Line: 1 Column: 7

A GROUP BY clause must be added to count the last names for each department_id.

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

ORA-00979: not a GROUP BY expression
00979. 00000 - "not a GROUP BY expression"
*Cause:
*Action:
Error at Line: 1 Column: 24

Either add job_id in the GROUP BY or remove the job_id column from the SELECT list.

Illegal Queries Using Group Functions

- You cannot use the WHERE 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;
```

ORA-00934: group function is not allowed here
00934. 00000 - "group function is not allowed here"
*Cause:
*Action:
Error at Line: 3 Column: 9

The WHERE clause cannot be used to restrict groups. The SELECT statement in the example in the slide results in an error because it uses the WHERE clause to restrict the display of the average salaries of those departments that have an average salary greater than $8,000.

Restricting Group Results

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	5800
5	50	2500
6	50	2600
7	50	3100
8	50	3500
9	60	4200
10	60	6000
11	60	9000
12	80	11000
13	80	10500
14	80	8600
...		
18	110	8300
19	110	12000
20	(null)	7000

The maximum salary per department when it is greater than $10,000

	DEPARTMENT_ID	MAX(SALARY)
1	20	13000
2	80	11000
3	90	24000
4	110	12000

Restricting Group Results with the HAVING Clause

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

- Rows are grouped.
- The group function is applied.
- Groups matching the HAVING clause are displayed.

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

Using the HAVING Clause

The example in the slide displays the department numbers and maximum salaries for those departments with a maximum salary greater than $10,000.

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

	DEPARTMENT_ID	MAX(SALARY)
1	100	12000
2	30	11000
3	90	24000
4	20	13000
5	110	12000
6	80	14000

Using the HAVING Clause

PRACTICE:

Display the department numbers and average salaries for those departments with a maximum salary greater than $12,000.

	DEPARTMENT_ID	Average Salary
1	90	19333.33
2	20	9500
3	80	8955.88

Using the HAVING Clause

PRACTICE:

Displays the job ID and total monthly salary for each job that has a total monthly salary exceeding $13,000 and job ID begin with 'S' and sorts the list by the total monthly salary.

	JOB_ID	PAYROLL
1	SA_MAN	61,000
2	SA_REP	250,500
3	SH_CLERK	64,300
4	ST_CLERK	55,700
5	ST_MAN	36,400

Lesson Agenda

- Group functions:
  - Types and syntax
  - Use AVG, SUM, MIN, MAX, COUNT
  - Use DISTINCT keyword within group functions
  - NULL values in a group function
- Grouping rows:
  - GROUP BY clause
  - HAVING clause
- **Nesting group functions**

Nesting Group Functions

- Group functions can be nested to a depth of two functions.
- The example in the slide calculates the average salary for each `department_id` and then displays the maximum average salary.

```
SELECT    _____________________
FROM      employees
GROUP BY  department_id;
```

	MAX(AVG(SALARY))
1	19333.333333333333333333333333333333333

Quiz

Identify the guidelines for group functions and the `GROUP BY` clause.

- ☐ You cannot use a column alias in the `GROUP BY` clause.
- ☐ The `GROUP BY` column must be in the `SELECT` clause.
- ☐ By using a `WHERE` clause, you can exclude rows before dividing them into groups.
- ☐ The `GROUP BY` clause groups rows and ensures order of the result set.
- ☐ If you include a group function in a `SELECT` clause, you cannot select individual results as well.

Summary

In this lesson, you should have learned how to:

- Use the group functions `COUNT`, `MAX`, `MIN`, `SUM`, 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];
```


Practice

Create a query that will 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.

	TOTAL	1995	1996	1997	1998
1	107	4	10	28	23