6

Reporting Aggregated Data Using the Group Functions

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

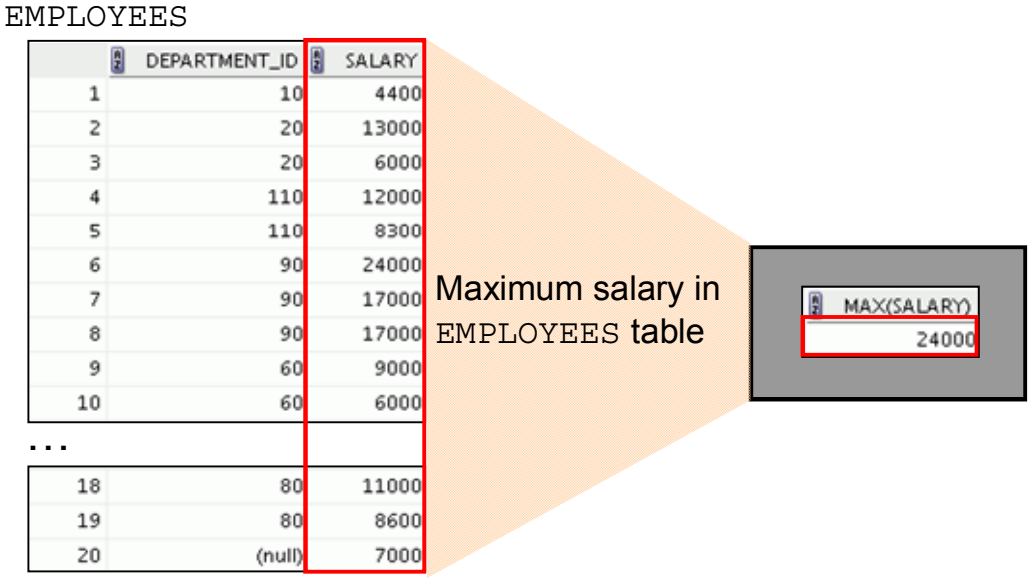
This lesson further addresses functions. It focuses on obtaining summary information (such as averages) for groups of rows. It discusses how to group rows in a table into smaller sets and how to specify search criteria for groups of rows.

Lesson Agenda

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

What Are Group Functions?

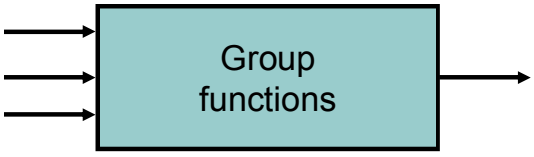
Group functions operate on sets of rows to give one result per group.



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

Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- SUM
- LISTAGG
- STDDEV
- 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	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]*n*)	Standard deviation of *n*, ignoring null values
SUM([DISTINCT|ALL]*n*)	Sum values of *n*, ignoring null values
LISTAGG	Orders data within each group specified in the ORDER BY clause and then concatenates the values of the measure column
VARIANCE([DISTINCT|ALL]*n*)	Variance of *n*, ignoring null values

Group Functions: Syntax

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


The group function is placed after the `SELECT` keyword. You may have multiple group functions separated by commas.

Syntax:

group_function`(`[`DISTINCT`|`ALL`] *expr*`)`

Guidelines for using the 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. To substitute a value for null values, use the `NVL`, `NVL2`, `COALESCE`, `CASE`, or `DECODE` functions.

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%';
```

	AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
1	8150	11000	6000	32600

You can use the AVG, SUM, MIN, and MAX functions against the columns that can store numeric data. The example in the slide displays the average, highest, lowest, and sum of monthly salaries for all sales representatives.

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;
```

	MIN(HIRE_DATE)	MAX(HIRE_DATE)
1	13-JAN-01	29-JAN-08

You can use the MAX and MIN functions for numeric, character, and date data types. The example in the slide 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 alphabetic list of all employees:

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

	MIN(LAST_NAME)	MAX(LAST_NAME)
1	Abel	Zlotkey

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:

1
```
SELECT COUNT(*)
FROM   employees
WHERE  department_id = 50;
```

	COUNT(*)
1	5

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

2
```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 50;
```

	COUNT(COMMISSION_PCT)
1	0

ORACLE

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 that are in the column identified by *expr*.

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

Examples

1. The example in the slide displays the number of employees in department 50.
2. The example in the slide displays the number of employees in department 50 who can earn a commission.

Using the DISTINCT Keyword

- 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:

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

	COUNT(DISTINCTDEPARTMENT_ID)
1	7

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

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

Group Functions and Null Values

Group functions ignore null values in the column:

1
```
SELECT AVG(commission_pct)
FROM   employees;
```

	AVG(COMMISSION_PCT)
1	0.2125

The NVL function forces group functions to include null values:

2
```
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

	AVG(NVL(COMMISSION_PCT,0))
1	0.0425

ORACLE

All group functions ignore null values in the column.

However, the NVL function forces group functions to include null values.

Examples

1. 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 calculated as the total commission that is paid to all employees divided by the number of employees receiving a commission (four).
2. 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 calculated as the total commission that is paid to all employees divided by the total number of employees in the company (20).

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	2500	
5	50	2600	
6	50	3100	3500
7	50	3500	
8	50	5800	
9	60	9000	6400
10	60	6000	
11	60	4200	
12	80	11000	10033
13	80	8600	
...			
18	110	8300	
19	110	12000	
20	(null)	7000	

Average salary in the EMPLOYEES table for each department

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

Until this point in the discussion, all group functions have treated the table as one large group of information. At times, however, you need to divide the table of information into smaller groups. This can be done by using the `GROUP BY` clause.

Creating Groups of Data: GROUP BY Clause Syntax

You can divide rows in a table into smaller groups by using the GROUP BY clause.

```
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 the columns whose values determine the basis for grouping rows

Guidelines

- If you include a group function in a SELECT clause, you cannot select individual column 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 can substitute *column* by an Expression in the SELECT statement.
- 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 the 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 ;
```

	DEPARTMENT_ID	AVG(SALARY)
1	(null)	7000
2	90	19333.333333333333333333333333333333333
3	20	9500
4	110	10154
5	50	3500
6	80	10033.333333333333333333333333333333333
7	60	6400
8	10	4400

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. The example in the slide displays the department number and the average salary for each department. Here is how this SELECT statement, containing a GROUP BY clause, is evaluated:

- The SELECT clause specifies the columns to be retrieved, as follows:
  - Department number column in the EMPLOYEES table
  - The average 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 WHERE clause specifies the rows to be retrieved. Because there is no WHERE clause, all rows are retrieved by default.
- The GROUP BY clause specifies how the rows should be grouped. The rows are grouped by department number, so the AVG function that is applied to the salary column calculates the average salary for each department.

Note: To order the query results in ascending or descending order, include the ORDER BY clause in the query.

Using the GROUP BY Clause

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

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

	AVG(SALARY)
1	7000
2	19333.333333333333333333333333333333333
3	9500
4	10154
5	3500
6	10033.333333333333333333333333333333333
7	6400
8	4400

ORACLE

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.

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

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

	DEPARTMENT_ID	AVG(SALARY)
1	50	3500
2	10	4400
3	60	6400
4	(null)	7000
5	20	9500
6	80	10033.333333333333333333333333333333333
7	110	10154
8	90	19333.333333333333333333333333333333333

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_CLERK	2500
5	50	ST_CLERK	2600
6	50	ST_CLERK	3100
7	50	ST_CLERK	3500
8	50	ST_MAN	5800
9	60	IT_PROG	9000
10	60	IT_PROG	6000
11	60	IT_PROG	4200
12	80	SA_REP	11000
13	80	SA_REP	8600
14	80	SA_MAN	10500
...			
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	110	AC_ACCOUNT	8300
2	110	AC_MGR	12008
3	10	AD_ASST	4400
4	90	AD_PRES	24000
5	90	AD_VP	34000
6	60	IT_PROG	19200
7	20	MK_MAN	13000
8	20	MK_REP	6000
9	80	SA_MAN	10500
10	80	SA_REP	19600
11	(null)	SA_REP	7000
12	50	ST_CLERK	11700
13	50	ST_MAN	5800

ORACLE

Sometimes, you need to see results for groups within groups. The slide shows a report that displays the total salary that is paid to each job title in each department.

The EMPLOYEES table is grouped first by the department number, and then by the job title within that grouping. For example, the four stock clerks in department 50 are grouped together, and a single result (total salary) is produced for all stock clerks in the group.

The following SELECT statement returns the result shown in the slide:

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

Using the GROUP BY Clause on Multiple Columns

```
SELECT   department_id, job_id, SUM(salary)
FROM     employees
WHERE     department_id > 40
GROUP BY department_id, job_id
ORDER BY department_id;
```

	DEPARTMENT_ID	JOB_ID	SUM(SALARY)
1	50	ST_CLERK	11700
2	50	ST_MAN	5800
3	60	IT_PROG	19200
4	80	SA_MAN	10500
5	80	SA_REP	19600
6	90	AD_PRES	24000
7	90	AD_VP	34000
8	110	AC_ACCOUNT	8300
9	110	AC_MGR	12008

You can return summary results for groups and subgroups by listing multiple GROUP BY columns. The GROUP BY clause groups rows but does not guarantee the order of the result set. To order the groupings, use the ORDER BY clause.

In the example in the slide, the SELECT statement that contains a GROUP BY clause is evaluated as follows:

- The SELECT clause specifies the column to be retrieved:
  - DEPARTMENT ID 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 WHERE clause reduces the result set to those rows where department ID is greater than 40.
- The GROUP BY clause specifies how you must group the resulting rows:
  - First, the rows are grouped by the DEPARTMENT ID.
  - Second, the rows are grouped by JOB ID in the DEPARTMENTID groups.
- The ORDER BY clause sorts the results by department ID.

Note: The SUM function is applied to the salary column for all job IDs in the result set in each DEPARTMENT ID group. Also, note that the SA_REP row is not returned. The DEPARTMENT ID for this row is NULL and, therefore, does not meet the WHERE condition.

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"

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"

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

Whenever you use a mixture of individual items (DEPARTMENT_ID) and group functions (COUNT) in the same SELECT statement, you must include a GROUP BY clause that specifies the individual items (in this case, DEPARTMENT_ID). If the GROUP BY clause is missing, the error message "not a single-group group function" appears and an asterisk (*) points to the offending column. You can correct the error in the first example in the slide by adding the GROUP BY clause:

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

Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause. In the second example in the slide, job_id is neither in the GROUP BY clause nor is it being used by a group function, so there is a "not a GROUP BY expression" error. You can correct the error in the second slide example by adding job_id in the GROUP BY clause.

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

Illegal Queries Using Group Functions

- 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;
```

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

Cannot use the `WHERE` clause to restrict groups

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.

However, you can correct the error in the example by using the `HAVING` clause to restrict groups:

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

	DEPARTMENT_ID	AVG(SALARY)
1	90	19333.333333333333333333333333333333333
2	20	9500
3	110	10154
4	80	10033.333333333333333333333333333333333

Restricting Group Results

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	50	2500
5	50	2600
6	50	3100
7	50	3500
8	50	5800
9	60	9000
10	60	6000
11	60	4200
12	80	11000
13	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	90	24000
3	110	12000
4	80	11000

You use the `HAVING` clause to restrict groups in the same way that you use the `WHERE` clause to restrict the rows that you select. To find the maximum salary in each of the departments that have a maximum salary greater than $10,000, you need to do the following:

1. Find the average salary for each department by grouping by department number.
2. Restrict the groups to those departments with a maximum salary greater than $10,000.

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.

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

ORACLE

You use the HAVING clause to specify the groups that are to be displayed, thus further restricting the groups on the basis of aggregate information.

In the syntax, *group_condition* restricts the groups of rows returned to those groups for which the specified condition is true.

The Oracle server performs the following steps when you use the HAVING clause:

1. Rows are grouped.
2. The group function is applied to the group.
3. The groups that match the criteria in the HAVING clause are displayed.

The HAVING clause can precede the GROUP BY clause, but it is recommended that you place the GROUP BY clause first because it is more logical. Groups are formed and group functions are calculated before the HAVING clause is applied to the groups in the SELECT list.

Note: The WHERE clause restricts rows, whereas the HAVING clause restricts groups.

Using the HAVING Clause

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

	DEPARTMENT_ID	MAX(SALARY)
1	90	24000
2	20	13000
3	110	12008
4	80	11000

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

You can use the GROUP BY clause without using a group function in the SELECT list. If you restrict rows based on the result of a group function, you must have a GROUP BY clause as well as the HAVING clause.

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

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

	DEPARTMENT_ID	AVG(SALARY)
1	90	19333.333333333333333333333333333333333
2	20	9500
3	110	10154
4	80	10033.333333333333333333333333333333333

Using the HAVING Clause

```
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);
```

	JOB_ID	PAYROLL
1	IT_PROG	19200
2	AD_PRES	24000
3	AD_VP	34000

ORACLE

The example in the slide 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.

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

Display the maximum average salary:

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

	MAX(AVG(SALARY))
1	19333.333333333333333333333333333333333333

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.

Note that `GROUP BY` clause is mandatory when nesting group functions.

Quiz

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

a. You cannot use a column alias in the `GROUP BY` clause.
b. The `GROUP BY` column must be in the `SELECT` clause.
c. By using a `WHERE` clause, you can exclude rows before dividing them into groups.
d. The `GROUP BY` clause groups rows and ensures order of the result set.
e. If you include a group function in a `SELECT` clause, you must include a `GROUP BY` clause.

Answer: a, c

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];
```


There are several group functions available in SQL, such as AVG, COUNT, MAX, MIN, SUM, STDDEV, and VARIANCE.

You can create subgroups by using the GROUP BY clause. Further, groups can be restricted using the HAVING clause.

Place the HAVING and GROUP BY clauses after the WHERE clause in a statement. The order of the GROUP BY and HAVING clauses following the WHERE clause is not important. You can have either the GROUP BY clause or the HAVING clause first as long as they follow the WHERE clause. Place the ORDER BY clause at the end.

The Oracle server evaluates the clauses in the following order:

1. If the statement contains a WHERE clause, the server establishes the candidate rows.
2. The server identifies the groups that are specified in the GROUP BY clause.
3. The HAVING clause further restricts result groups that do not meet the group criteria in the HAVING clause.

Note: For a complete list of the group functions, see *Oracle Database SQL Language Reference* for 12*c* database.

Practice 6: Overview

This practice covers the following topics:

- Writing queries that use the group functions
- Grouping by rows to achieve more than one result
- Restricting groups by using the `HAVING` clause

In this practice, you learn to use group functions and select groups of data.