4

Reporting Aggregated Data Using the Group Functions

ORACLE

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

ORACLE

Objectives

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.

What Are Group Functions?

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

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	110	12000
5	110	8300
6	90	24000
7	90	17000
8	90	17000
9	60	9000
10	60	6000
11	60	4200
12	50	5800
13	50	3500
14	50	3100
15	50	2600

...

Maximum salary in EMPLOYEES table

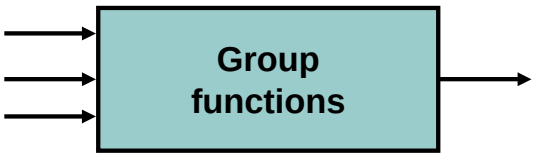
	MAX(SALARY)
1	24000

Group Functions

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
- STDDEV
- SUM
- VARIANCE



Types of Group Functions

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

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. To substitute a value for null values, use the `NVL`, `NVL2`, or `COALESCE` 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

ORACLE

Using the Group Functions

You can use AVG, SUM, MIN, and MAX functions against 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	17-JUN-87	29-JAN-00

ORACLE

Using the Group Functions (continued)

You can use the MAX and MIN functions for numeric, character, and date data types. The slide 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;
```

	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 = 80;
```

	COUNT(COMMISSION_PCT)
1	3

ORACLE

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 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 slide example displays the number of employees in department 50.
2. The slide example displays the number of employees in department 80 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

DISTINCT Keyword

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

Group Functions and Null Values

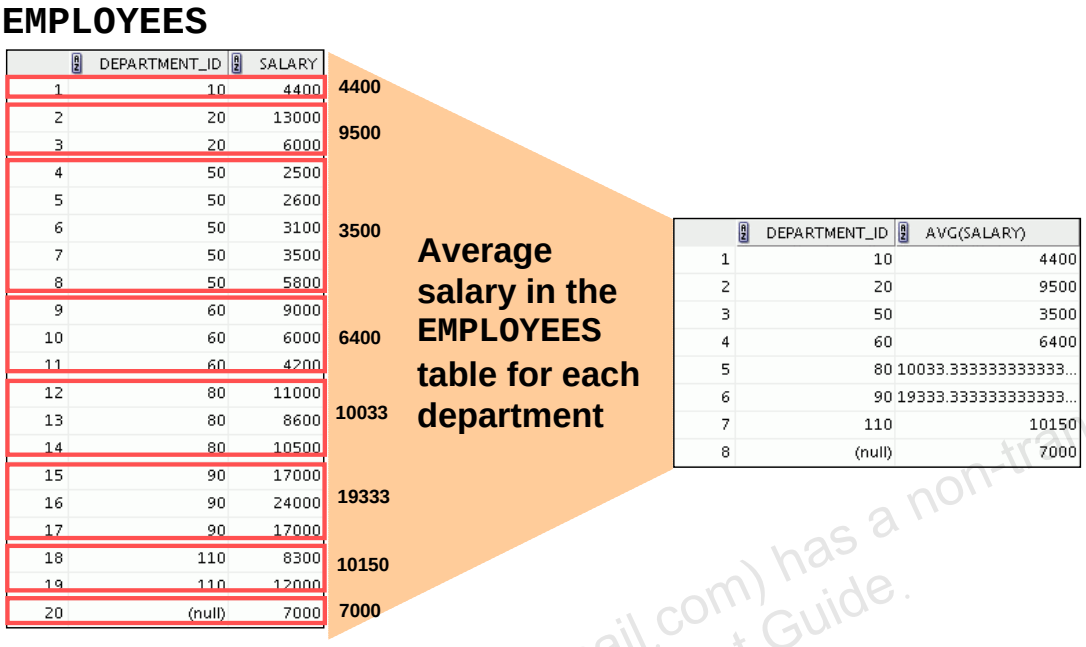
All group functions ignore null values in the column.

The NVL function forces group functions to include null values.

Examples

1. The average is calculated based on *only* those rows in the table where 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).

Creating Groups of Data



Creating Groups of Data

Until this point in our 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. You can do this by using the GROUP BY clause.

Creating Groups of Data: 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 Clause

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 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 ;
```

	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

ORACLE

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. 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 the 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.*

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	9500
3	19333.333333333333333333333333333333333
4	10150
5	3500
6	10033.333333333333333333333333333333333
7	4400
8	6400

ORACLE

Using the GROUP BY Clause (continued)

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 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	10033.3333333333...
7	110	10150
8	90	19333.3333333333...

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
15	90	AD_VP	17000
16	90	AD_PRES	24000
17	90	AD_VP	17000
18	110	AC_ACCOUNT	8300
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

Groups Within Groups

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 department number and then by 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.

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 ;
```

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

ORACLE

Groups Within Groups (continued)

You can return summary results for groups and subgroups by listing more than one GROUP BY column. You can determine the default sort order of the results by the order of the columns in the GROUP BY clause. In the slide example, the SELECT statement containing a GROUP BY clause is evaluated as follows:

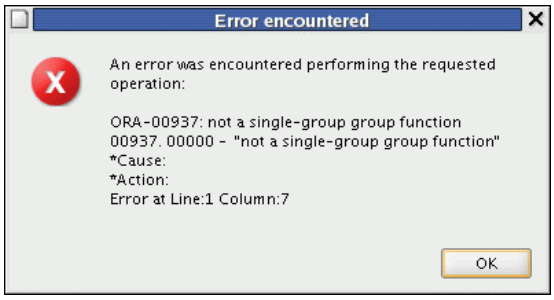
- The SELECT clause specifies the column to be retrieved:
  - Department number in the EMPLOYEES table
  - Job ID in the EMPLOYEES table
  - The sum of all the salaries in the group that you specified in the GROUP BY clause
- The FROM clause specifies the tables that the database must access: the EMPLOYEES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

So the SUM function is applied to the salary column for all job IDs in each department number group.

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;
```



Column missing in the GROUP BY clause

ORACLE

Illegal Queries Using Group Functions

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, then the error message "not a single-group group function" appears. You can correct the error in the slide by adding the GROUP BY clause:

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

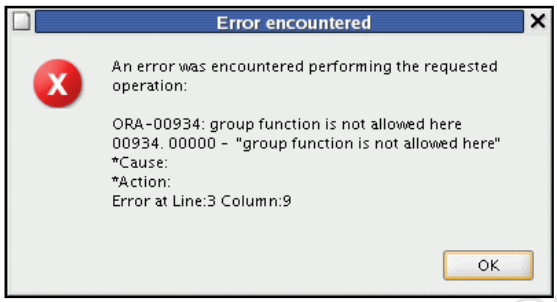
	DEPARTMENT_ID	COUNT(LAST_NAME)
1	(null)	1
2	20	2
3	90	3
4	110	2
5	50	5
6	80	3
7	10	1
8	60	3

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

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;
```



Cannot use the WHERE clause to restrict groups

ORACLE

Illegal Queries Using Group Functions (continued)

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

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;
```

	DEPARTMENT_ID	AVG(SALARY)
1	20	9500
2	90	19333.333333333333333...
3	110	10150
4	80	10033.333333333333333...

Restricting Group Results

EMPLOYEES

	DEPARTMENT_ID	SALARY
1	10	4400
2	20	13000
3	20	6000
4	110	12000
5	110	8300
6	90	24000
7	90	17000
8	90	17000
9	60	9000
10	60	6000
11	60	4200
12	50	5800
13	50	3500
14	50	3100
15	50	2600

...

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

ORACLE

Restricting Group Results

In the same way that you use the `WHERE` clause to restrict the rows that you select, you use the `HAVING` clause to restrict groups. 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:

- 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];
```

ORACLE

Restricting Group Results with the HAVING Clause

You use the HAVING clause to specify which groups 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 that is more logical. Groups are formed and group functions are calculated before the HAVING clause is applied to the groups in the SELECT list.

Using the HAVING Clause

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

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

Using the HAVING Clause

The slide example displays department numbers and maximum salaries for those departments with a maximum salary that is 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 that is greater than $10,000:

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

	DEPARTMENT_ID	AVG(SALARY)
1	20	9500
2	90	19333.333333333333333333...
3	110	10150
4	80	10033.333333333333333333...

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

Using the HAVING Clause (continued)

The slide 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.

Nesting Group Functions

Display the maximum average salary:

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

	MAX(AVG(SALARY))
1	19333.333333333333333333333333333333333

Nesting Group Functions

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

Summary

In this lesson, you 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];
```


Summary

Several group functions are available in SQL, such as the following:
AVG, COUNT, MAX, MIN, SUM, STDDEV, and VARIANCE

You can create subgroups by using the GROUP BY clause. 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 HAVING and GROUP clauses following the WHERE clause is not important. Place the ORDER BY clause last.

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 SQL Reference*.

Practice 4: Overview

This practice covers the following topics:

- Writing queries that use the group functions
- Grouping by rows to achieve multiple results
- Restricting groups by using the `HAVING` clause

Practice 4: Overview

After completing this practice, you should be familiar with using group functions and selecting groups of data.

Practice 4

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

2. Group functions include nulls in calculations.
   True/False

3. The `WHERE` clause restricts rows before inclusion in a group calculation.
   True/False

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. Place your SQL statement in a text file named `lab_04_04.sql`.

	Maximum	Minimum	Sum	Average
1	24000	2500	175500	8775

5. Modify the query in `lab_04_04.sql` to display the minimum, maximum, sum, and average salary for each job type. Resave `lab_04_04.sql` as `lab_04_05.sql`. Run the statement in `lab_04_05.sql`.

	JOB_ID	Maximum	Minimum	Sum	Average
1	AC_MGR	12000	12000	12000	12000
2	AC_ACCOUNT	8300	8300	8300	8300
3	IT_PROG	9000	4200	19200	6400
4	ST_MAN	5800	5800	5800	5800
5	AD_ASST	4400	4400	4400	4400
6	AD_VP	17000	17000	34000	17000
7	MK_MAN	13000	13000	13000	13000
8	SA_MAN	10500	10500	10500	10500
9	MK_REP	6000	6000	6000	6000
10	AD_PRES	24000	24000	24000	24000
11	SA_REP	11000	7000	26600	8867
12	ST_CLERK	3500	2500	11700	2925

Practice 4 (continued)

6. Write a query to display the number of people with the same job.

	JOB_ID	COUNT(*)
1	AC_ACCOUNT	1
2	AC_MGR	1
3	AD_ASST	1
4	AD_PRES	1
5	AD_VP	2
6	IT_PROG	3
7	MK_MAN	1
8	MK_REP	1
9	SA_MAN	1
10	SA_REP	3
11	ST_CLERK	4
12	ST_MAN	1

Generalize the query so that the user in the HR department is prompted for a job title. Save the script to a file named `lab_04_06.sql`.

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.*

	Number of Managers
1	8

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

	DIFFERENCE
1	21500

If you have time, complete the following exercises:

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.

	MANAGER_ID	MIN(SALARY)
1	102	9000
2	205	8300
3	149	7000

Practice 4 (continued)

If you want an extra challenge, complete the following exercises:

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.

	TOTAL	1995	1996	1997	1998
1	20	1	2	2	3

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

	Job	Dept 20	Dept 50	Dept 80	Dept 90	Total
1	AC_MGR	(null)	(null)	(null)	(null)	12000
2	AC_ACCOUNT	(null)	(null)	(null)	(null)	8300
3	IT_PROG	(null)	(null)	(null)	(null)	19200
4	ST_MAN	(null)	5800	(null)	(null)	5800
5	AD_ASST	(null)	(null)	(null)	(null)	4400
6	AD_VP	(null)	(null)	(null)	34000	34000
7	MK_MAN	13000	(null)	(null)	(null)	13000
8	SA_MAN	(null)	(null)	10500	(null)	10500
9	MK_REP	6000	(null)	(null)	(null)	6000
10	AD_PRES	(null)	(null)	(null)	24000	24000
11	SA_REP	(null)	(null)	19600	(null)	26600
12	ST_CLERK	(null)	11700	(null)	(null)	11700