**Programming 9**

**Database Programming with SQL  
9-1: Using GROUP BY and HAVING Clauses  
Practice Activities**

Vocabulary  
Identify the vocabulary word for each definition below.

|  |  |
| --- | --- |
| **HAVING** | Used to specify which groups are to be displayed; restricts groups that do not meet group criteria |
| **GROUP BY** | Divides the rows in a table into groups |

1. In the SQL query shown below, which of the following is true about this query?  
   \_\_\_\_\_\_\_ a. Kimberly Grant would not appear in the results set.  
   \_\_\_\_\_\_\_ b. The GROUP BY clause has an error because the manager\_id is not listed in theSELECT clause.  
   \_\_\_\_\_\_\_ c. Only salaries greater than 16001 will be in the result set.  
   \_\_\_\_\_\_\_ d. Names beginning with Ki will appear after names beginning with Ko.  
   \_\_\_\_\_\_\_ e. Last names such as King and Kochhar will be returned even if they don’t havesalaries > 16000.  
   SELECT last\_name, MAX(salary)  
   FROM employees  
   WHERE last\_name LIKE 'K%'  
   GROUP BY manager\_id, last\_name  
   HAVING MAX(salary) >16000  
   ORDER BY last\_name DESC ;

The correct option is **(d)**: "Names beginning with Ki will appear after names beginning with Ko."

1. Each of the following SQL queries has an error. Find the error and correct it. Use Oracle  
   Application Express to verify that your corrections produce the desired results.
2. SELECT manager\_id  
   FROM employees  
   WHERE AVG(salary) <16000  
   GROUP BY manager\_id;

**Corrected:**

**SELECT manager\_id**

**FROM employees**

**GROUP BY manager\_id**

**HAVING AVG(salary) < 16000;**

1. SELECT cd\_number, COUNT(title)  
   FROM d\_cds  
   WHERE cd\_number < 93;

**Corrected:**

**SELECT cd\_number, COUNT(title)**

**FROM d\_cds**

**WHERE cd\_number < 93**

**GROUP BY cd\_number;**

1. SELECT ID, MAX(ID), artist AS Artist  
   FROM d\_songs  
   WHERE duration IN('3 min', '6 min', '10 min')  
   HAVING ID < 50  
   GROUP by ID;

**Corrected:**

**SELECT ID, artist AS Artist**

**FROM d\_songs**

**WHERE duration IN ('3 min', '6 min', '10 min')**

**AND ID < 50**

**GROUP BY ID, artist;**

1. SELECT loc\_type, rental\_fee AS Fee  
   FROM d\_venues  
   WHERE id <100  
   GROUP BY "Fee"  
   ORDER BY 2;

**Corrected:**

**SELECT loc\_type, rental\_fee AS Fee**

**FROM d\_venues**

**WHERE id < 100**

**GROUP BY loc\_type, rental\_fee**

**ORDER BY 2;**

1. Rewrite the following query to accomplish the same result:  
   SELECT DISTINCT MAX(song\_id)  
   FROM d\_track\_listings  
   WHERE track IN ( 1, 2, 3);

SELECT MAX(song\_id)

FROM d\_track\_listings

WHERE track IN (1, 2, 3)

GROUP BY NULL;

1. Indicate True or False  
   \_\_T\_\_\_ a. If you include a group function and any other individual columns in a SELECT clause,  
   then each individual column must also appear in the GROUP BY clause.  
   \_\_\_F\_\_ b. You can use a column alias in the GROUP BY clause.  
   \_\_\_F\_\_ c. The GROUP BY clause always includes a group function.
2. Write a query that will return both the maximum and minimum average salary grouped by department from the employees table.

SELECT department\_id, MAX(AVG(salary)) AS max\_avg\_salary, MIN(AVG(salary)) AS min\_avg\_salary

FROM employees

GROUP BY department\_id;

1. Write a query that will return the average of the maximum salaries in each department for the employees table.

SELECT AVG(MAX(salary)) AS avg\_max\_salary

FROM employees

GROUP BY department\_id;

**Database Programming with SQL  
9-2: Using ROLLUP and CUBE Operations and GROUPING SETS  
Practice Activities**

Vocabulary  
Identify the vocabulary word for each definition below.

|  |  |
| --- | --- |
| **ROLLUP** | Used to create subtotals that roll up from the most detailed level to a grand total, following a grouping list specified in the clause |
| **CUBE** | An extension to the GROUP BY clause like ROLLUP that produces cross-tabulation reports |
| **GROUPING SETS** | Used to specify multiple groupings of data |

Try It / Solve It  
1. Within the Employees table, each manager\_id is the manager of one or more employees who  
each have a job\_id and earn a salary. For each manager, what is the total salary earned by all of  
the employees within each job\_id? Write a query to display the Manager\_id, job\_id, and total  
salary. Include in the result the subtotal salary for each manager and a grand total of all salaries.

SELECT

manager\_id,

job\_id,

SUM(salary) AS total\_salary

FROM

employees

GROUP BY

ROLLUP(manager\_id, job\_id);

2. Amend the previous query to also include a subtotal salary for each job\_id regardless of the  
manager\_id.

SELECT

manager\_id,

job\_id,

SUM(salary) AS total\_salary

FROM

employees

GROUP BY

CUBE(manager\_id, job\_id);

3. Using GROUPING SETS, write a query to show the following groupings:  
• department\_id, manager\_id, job\_id  
• manager\_id, job\_id  
• department\_id, manager\_id

SELECT

department\_id,

manager\_id,

job\_id,

SUM(salary) AS total\_salary

FROM

employees

GROUP BY

GROUPING SETS (

(department\_id, manager\_id, job\_id),

(manager\_id, job\_id),

(department\_id, manager\_id) );

**Database Programming with SQL  
9-3: Set Operators  
Practice Activities**

Vocabulary  
Identify the vocabulary word for each definition below.

|  |  |
| --- | --- |
| **UNION** | operator that returns all rows from both tables and eliminates duplicates |
| **Derived Columns** | columns that were made up to match queries in another table that are not in both tables |
| **UNION ALL** | operator that returns all rows from both tables, including duplicates |
| **Set Operations** | used to combine results into one single result from multiple SELECT statements |
| **MINUS** | operator that returns rows that are unique to each table |
| **INTERSECT** | operator that returns rows common to both tables |

Try It / Solve It

1. Name the different Set operators?

* UNION
* UNION ALL
* INTERSECT
* MINUS

1. Write one query to return the employee\_id, job\_id, hire\_date, and department\_id of all employees and a second query listing employee\_id, job\_id, start\_date, and department\_id from the job\_history table and combine the results as one single output. Make sure you suppress duplicates in the output.

SELECT employee\_id, job\_id, hire\_date AS start\_date, department\_id

FROM employees

UNION

SELECT employee\_id, job\_id, start\_date, department\_id

FROM job\_history;

1. Amend the previous statement to not suppress duplicates and examine the output. How many extra rows did you get returned and which were they? Sort the output by employee\_id to make it easier to spot.

SELECT employee\_id, job\_id, hire\_date AS start\_date, department\_id

FROM employees

UNION ALL

SELECT employee\_id, job\_id, start\_date, department\_id

FROM job\_history

ORDER BY employee\_id;

1. List all employees who have not changed jobs even once. (Such employees are not found in the job\_history table)

SELECT employee\_id, job\_id, hire\_date, department\_id

FROM employees

WHERE employee\_id NOT IN (SELECT employee\_id FROM job\_history);

1. List the employees that HAVE changed their jobs at least once.

SELECT employee\_id, job\_id, hire\_date, department\_id

FROM employees

WHERE employee\_id IN (SELECT employee\_id FROM job\_history);

1. Using the UNION operator, write a query that displays the employee\_id, job\_id, and salary of ALL present and past employees. If a salary is not found, then just display a 0 (zero) in its place.

SELECT employee\_id, job\_id, NVL(salary, 0) AS salary

FROM employees

UNION

SELECT employee\_id, job\_id, 0 AS salary

FROM job\_history;