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

Objectives

- Construct and execute a SQL query using GROUP BY
- Construct and execute a SQL query using GROUP BY ... HAVING
- Construct and execute a GROUP BY on more than one column
- Nest group functions

Vocabulary

Identify the vocabulary word for each definition below.

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

Try It / Solve It

1. In the SQL query shown below, which of the following is true about this query?

___T____ a.Kimberly Grant would not appear in the results set.

____F___ b.The GROUP BY clause has an error because the manager_id is not listed in the SELECT clause.

___F____ c. Only salaries greater than 16001 will be in the result set.

____F___ d. Names beginning with Ki will appear after names beginning with Ko.

____F___ e. Last names such as King and Kochhar will be returned even if they don't have salaries > 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 ;
```

2. 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.

a.
```
SELECT manager_id
FROM employees
WHERE AVG(salary) <16000
```

```
GROUP BY manager_id;
SELECT manager_id
FROM employees
GROUP BY manager_id
HAVING AVG(salary) < 16000;
```
b.
```
SELECT cd_number, COUNT(title)
FROM d_cds
WHERE cd_number < 93;
SELECT cd_number, COUNT(title)
FROM d_cds
WHERE cd_number < 93
GROUP BY cd_number;
```
c.
```
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;
SELECT ID, artist AS Artist
FROM d_songs
WHERE duration IN ('3 min', '6 min', '10 min')
AND ID < 50;
```
d.
```
SELECT loc_type, rental_fee AS Fee
FROM d_venues
WHERE id <100
GROUP BY "Fee"
ORDER BY 2;
SELECT loc_type, rental_fee AS Fee
FROM d_venues
WHERE id < 100
ORDER BY 2;
```
3. 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 track;
```
4. 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.

5. Write a query that will return both the maximum and minimum average salary grouped by department from the employees table.

```
SELECT MAX(avg_salary) AS max_avg_salary, MIN(avg_salary) AS min_avg_salary
FROM (
   SELECT department_id, AVG(salary) AS avg_salary
   FROM employees
   GROUP BY department_id
) avg_salaries;
```

6. 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 (
   SELECT department_id, MAX(salary) AS max_salary
   FROM employees
   GROUP BY department_id
) dept_max_salaries;
```

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

Objectives

- Use ROLLUP to produce subtotal values
- Use CUBE to produce cross-tabulation values
- Use GROUPING SETS to produce a single result set
- Use the GROUPING function to identify the extra row values created by either a ROLLUP or CUBE operation

Vocabulary

Identify the vocabulary word for each definition below.

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

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)
ORDER BY 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 ROLLUP(job_id, manager_id)
ORDER BY job_id, manager_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), manager_id, job_id
(department_id, manager_id) )
ORDER BY  department_id, manager_id, job_id;
```

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

Objectives

- Define and explain the purpose of SET operators
- Use a set operator to combine multiple queries into a single query
- Control the order of rows returned using set operators

Vocabulary

Identify the vocabulary word for each definition below.

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

- operator that returns rows that are unique to each table MINUS
- operator that returns rows common to both tables INTERSECT

Try It / Solve It

1. Name the different Set operators?

UNION:
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;

UNION ALL:
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;

INTERSECT:
SELECT column_name(s) FROM table1
INTERSECT
SELECT column_name(s) FROM table2;

EXCEPT:
SELECT column_name(s) FROM table1
EXCEPT (or MINUS)
SELECT column_name(s) FROM table2;

2. 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;
```

3. 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;
```

4. 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);
```

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

```
SELECT employee_id, job_id, hire_date, department_id
FROM employees
INNER JOIN job_history
ON employee_id = employee_id;
```

6. 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, salary
FROM employees
UNION ALL
SELECT employee_id, job_id, COALESCE(salary, 0) AS salary
FROM job_history;
```