Working with Oracle SQL

Chapter 5:
Aggregating Information

Chapter Objectives

In this chapter, we will discuss:

- Reporting about groups of data
  - Aggregate functions
- Defining the groups
  - The `GROUP BY` and `HAVING` clauses
- `JOIN`ing issues
- Working with subqueries

Chapter Concepts

Aggregate Functions

The `GROUP BY` Clause

The `HAVING` Clause

`JOIN`ing Issues

Subqueries

Chapter Summary

Aggregate Functions

- Aggregate functions return one result per group of rows, rather than one result per row
  - For simplicity of discussion, assume one group is retrieved
    - A `SELECT` statement can retrieve multiple groups by utilizing the `GROUP BY` clause, which will be discussed later
- The common aggregate functions are:
  - `COUNT`
  - `SUM`
  - `AVG`
  - `MAX`
  - `MIN`
- **All aggregate functions ignore `NULL` values**
  - Except `COUNT(*)`

Aggregate Function: `COUNT`

- `COUNT` always returns a number
  - Even if no rows are counted, `COUNT` will return 0
- `COUNT` can be used several ways
  - `COUNT(*)` returns the number of rows in the group
  - `COUNT(some_column)` returns the number of non-NULL values in the group
  - `COUNT(DISTINCT some_column)` returns the number of distinct (unique) values

COUNT: Example

- Given the following set
  - How many rows are there?
  - How many rows have a minimum salary?
  - How many distinct minimum salary values are there?

```
SELECT job_title, min_salary
FROM jobs
WHERE min_salary < 5000 ORDER BY min_salary;

JOB_TITLE                          MIN_SALARY
---------------------------------- ----------
Stock Clerk                              2000
Purchasing Clerk                         2500
Shipping Clerk
Administration Assistant                 3000
Programmer                               4000
Human Resources Representative           4000
Marketing Representative                 4000
Accountant                               4200
Public Accountant                        4200
Public Relations Representative          4500
```

COUNT: Example (continued)

- Answer:

```
SELECT COUNT(*), COUNT(min_salary), COUNT(DISTINCT min_salary)
FROM jobs
WHERE min_salary < 5000;

  COUNT(*) COUNT(MIN_SALARY) COUNT(DISTINCTMIN_SALARY)
---------- ----------------- --------------------------
        10                 9                          6
```

- Question: How many employees have commission, how many do not, and what is the total?

```
SELECT COUNT(commission_pct) "Have Commission",
       COUNT(*) - COUNT(commission_pct) "Do Not Have Commission",
       COUNT(*) "Total Emps"
FROM employees;

Have Commission Do Not Have Commission Total Emps
--------------- ----------------------- ----------
             35                      72        107
```

Aggregate Functions: SUM and AVG

- Remember, the aggregate functions ignore `NULL` values
  - `SUM(some_column)` totals up the non `NULL` values
  - `AVG(some_column)` averages the non `NULL` values
  - `DISTINCT` can also be used
    - `SUM(DISTINCT some_column)`
    - `AVG(DISTINCT some_column)`
- Be sure that this value is what you intend to return!
  - You may prefer to treat `NULL` as 0 by using `COALESCE`

SUM and AVG Examples

- What are the sum and average of commission percent values?

```
SELECT SUM(commission_pct)                                    AS sum,
       COUNT(commission_pct)                                  AS "COUNT NOT NULL",
       ROUND(AVG(commission_pct), 6)                          AS avg,
       ROUND(SUM(commission_pct) / COUNT(commission_pct), 6)  AS "AVG BY CALC",
       COUNT(*)                                               AS count,
       ROUND(AVG(COALESCE(commission_pct, 0)), 6)             AS "AVG (NULL=0)",
       ROUND(SUM(commission_pct) / COUNT(*), 6)               AS "AVG BY CALC (NULL=0)"
FROM employees;

    SUM COUNT NOT NULL        AVG AVG BY CALC   COUNT AVG (NULL=0) AVG BY CALC (NULL=0)
------- -------------- --------- ----------- ------- ------------ --------------------
    7.8             35   .222857     .222857     107     .072897              .072897
```

Aggregate Functions: MAX and MIN

- These functions return the largest and smallest values of a column in a set of data
- What are the largest and smallest commission percent values?
  - Notice that NULL values do not show as either MAX or MIN

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

MAX(COMMISSION_PCT) MIN(COMMISSION_PCT)
------------------- -------------------
                 .4                  .1
```

- Non-numeric data
  - What employee's last name appears at the end of a sorted listing?

```
SELECT MAX(last_name) FROM employees;
MAX(LAST_NAME)
-------------------------
Zlotkey
```

– What is the oldest hire date for employees on file?

```
SELECT MIN(hire_date) FROM employees;
MIN(HIRE_DA
-----------
17-JUN-1987
```

Mixing Single Row (Scalar) and Aggregate Values

- When reporting an aggregate value about a group, we cannot also return a scalar value
- Given this set, we cannot return an aggregate value AND a single row value

```
JOB_TITLE                           MIN_SALARY
----------------------------------- ----------
Stock Clerk                               2000
Purchasing Clerk                          2500
Shipping Clerk
Administration Assistant                  3000
Programmer                                4000
Human Resources Representative            4000
Marketing Representative                  4000
Accountant                                4200
Public Accountant                         4200
Public Relations Representative           4500
```

```
SELECT job_title, MAX(min_salary) FROM jobs
WHERE min_salary < 5000;
SELECT job_title, MAX(min_salary) FROM jobs
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

Exercise 5.1: Using the Aggregate Functions

20 min

- Please complete this exercise in your Exercise Manual

Chapter Concepts

- Aggregate Functions
- **The `GROUP BY` Clause**
- The `HAVING` Clause
- `JOIN`ing Issues
- Subqueries
- Chapter Summary

Specifying the Groups

- The sets of data we have been aggregating have been tables
  - Or a result set of tables filtered by the `WHERE` clause
- Suppose we wanted to describe aggregates for different groups
  - The average minimum and maximum salaries for different jobs
    - Defined by the first two characters of the `job_id` column
- We could run a series of aggregate queries against the jobs table
  - Using the `WHERE` clause to filter out one group at a time

Groups the Hard Way

- This approach works, but is making several assumptions
  - That we know all of the `job_id` strings (we could query them first)
  - That we have the patience to build and execute *many* statements!

```
SELECT AVG(min_salary), AVG(max_salary), COUNT(*)
FROM jobs
WHERE SUBSTR(job_id, 1, 2) = 'AD';

AVG(MIN_SALARY) AVG(MAX_SALARY)   COUNT(*)
--------------- --------------- ----------
     12666.6667      25333.3333          3
```

```
SELECT AVG(min_salary), AVG(max_salary), COUNT(*)
FROM jobs
WHERE SUBSTR(job_id, 1, 2) = 'FI';

AVG(MIN_SALARY) AVG(MAX_SALARY)   COUNT(*)
--------------- --------------- ----------
           6200           12500          2
```

Groups the Easy Way

- Fortunately, SQL allows us to specify `GROUP`s in the `SELECT` statement
  - The `GROUP BY` clause
- And, we can filter out groups we decide not to include in final result set
  - The `HAVING` clause
    - Discussed in a few minutes
- Remember the overall structure of the `SELECT` statement

```
SELECT   column or expression, column or expression …
FROM        table
WHERE    condition 1  AND/OR condition 2 …
GROUP BY column or expression, column or expression …
HAVING   condition 1  AND/OR condition 2 …
ORDER BY column or expression or column alias or position, …
```

GROUP BY Clause

- **All columns in the SELECT clause must be in the GROUP BY clause or must be part of an aggregate function**
- The aggregate will produce 1 row per group
  - NULL values in the GROUP BY column will be grouped into a single group
- The condition(s) in the HAVING clause are then applied to the grouped sets
  - And are accepted or filtered out
- Remember, if the result set sequence is important, specify an ORDER BY clause

GROUP BY Example

- What is the highest and lowest employee id in each department?

```
SELECT department_id, MIN(employee_id), MAX(employee_id)
FROM   employees
WHERE  department_id > 10
GROUP BY department_id
ORDER BY department_id;

DEPARTMENT_ID MIN(EMPLOYEE_ID) MAX(EMPLOYEE_ID)
------------- ---------------- ----------------
           20              201              202
           30              114              119
           40              203              203
           50              120              199
           60              103              107
           70              204              204
           80              145              179
           90              100              102
          100              108              113
          110              205              206
```

GROUP BY Example (continued)

- Add the count of how many employees are in each department and list the largest departments first and by department number descending within the employee count
  - Place employee 178 at the bottom

```
SELECT department_id, COUNT(*), MIN(employee_id), MAX(employee_id)
FROM employees
GROUP BY department_id
ORDER BY COUNT(*) DESC, department_id DESC NULLS LAST;

DEPARTMENT_ID   COUNT(*) MIN(EMPLOYEE_ID) MAX(EMPLOYEE_ID)
------------- ---------- ---------------- ----------------
           50         45              120              199
           80         34              145              179
          100          6              108              113
           30          6              114              119
           60          5              103              107
           90          3              100              102
          110          2              205              206
           20          2              201              202
           70          1              204              204
           40          1              203              203
           10          1              200              200
                       1              178              178
```

Chapter Concepts

- Aggregate Functions
- The `GROUP BY` Clause
- **The `HAVING` Clause**
- `JOIN`ing Issues
- Subqueries
- Chapter Summary

The `HAVING` Clause

- The `WHERE` clause filters the rows selected from the table(s)
- The `HAVING` clause filters groups once the `GROUP BY` has completed
- The conditions are constructed using the same comparison operators as the `WHERE` clause
- The `HAVING` clause should only reference aggregates
  - Data that is NOT known until this time

HAVING Clause Example

- Restrict the reporting about departments to those which have at least five employees

```
SELECT department_id, COUNT(*), MIN(employee_id), MAX(employee_id)
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5
ORDER BY COUNT(*) DESC, department_id DESC NULLS LAST;

DEPARTMENT_ID   COUNT(*) MIN(EMPLOYEE_ID) MAX(EMPLOYEE_ID)
------------- ---------- ---------------- ----------------
           50         45              120              199
           80         34              145              179
          100          6              108              113
           30          6              114              119
```

Invalid `WHERE` Clause

- Can we filter out those departments with few employees in the `WHERE` clause?
  - Why or why not?

```
SELECT department_id, COUNT(*), MIN(employee_id), MAX(employee_id)
FROM employees
WHERE COUNT(*) > 5
GROUP BY department_id
ORDER BY COUNT(*) DESC, department_id DESC NULLS LAST;

WHERE COUNT(*) > 5
      *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

- We cannot because the data value is not yet available
  - Aggregate functions are not allowed in the `WHERE` clause

A Legal, But Inefficient HAVING Clause

- Suppose we wanted to restrict the list to department numbers less than 100
- The following query returns the correct result

```
SELECT department_id, COUNT(*), MIN(employee_id), MAX(employee_id)
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5
AND department_id <> 100
ORDER BY COUNT(*) DESC , department_id DESC NULLS LAST;

DEPARTMENT_ID   COUNT(*) MIN(EMPLOYEE_ID) MAX(EMPLOYEE_ID)
------------- ---------- ---------------- ----------------
           50         45              120              199
           80         34              145              179
           30          6              114              119
```

Same Answer, Only Quicker

- The `department_id` is known at the scalar level
  - And can be filtered in the `WHERE` clause

```
SELECT department_id, COUNT(*), MIN(employee_id), MAX(employee_id)
FROM employees
WHERE department_id <> 100
GROUP BY department_id
HAVING COUNT(*) > 5
ORDER BY COUNT(*) DESC , department_id DESC NULLS LAST;

DEPARTMENT_ID   COUNT(*) MIN(EMPLOYEE_ID) MAX(EMPLOYEE_ID)
------------- ---------- ---------------- ----------------
           50         45              120              199
           80         34              145              179
           30          6              114              119
```

Chapter Concepts

Aggregate Functions

The `GROUP BY` Clause

The `HAVING` Clause

`JOIN`ing Issues

Subqueries

Chapter Summary

Query Problems Can Be Masked by Aggregates

- The result set being passed into aggregate processing could be coming from a `JOIN`
  - The `JOIN` may be syntactically correct
    - But not what you intended
  - Since the aggregate is a number, it could seem reasonable
    - But might lead to the wrong business decision
- Things to bear in mind
  - 1:M joins may result in column values being duplicated in the pre-aggregation results set
  - Outer joins introduce `NULL`s, which may not be handled as you expect
- **ALWAYS** make certain that the `JOIN` is complete first, then add the `GROUP BY` functionality

Exercise 5.2: GROUP BY and HAVING

30 min

- Please complete this exercise in your Exercise Manual

Chapter Concepts

- Aggregate Functions
- The `GROUP BY` Clause
- The `HAVING` Clause
- `JOIN`ing Issues
- **Subqueries**
- Chapter Summary

Simple Subqueries

- A subquery is a `SELECT` statement that occurs inside a condition of another `SELECT` statement
  - Each row of the parent statement is compared with the result of the subquery
    - If the comparison fails, the row is rejected
- The subquery is the same as the regular `SELECT` statement without the `ORDER BY` clause
- A subquery can have its own subqueries
- Normally used in the `WHERE` clause of the parent `SELECT` statement
- Common comparison operators for a subquery
  - Relational operators (=, <>, >, >=, <, <=)
  - `IN` and `NOT IN`

Simple Subquery Using a Relational Operator

- Subquery must return a single value
- Example:
  - Find employees who have the maximum salary

```
SELECT e.last_name, e.first_name, e.salary
FROM employees e
WHERE e.salary = (
    SELECT MAX(e2.salary)
    FROM employees e2
);
```

Simple Subquery Using an IN Operator

- Subquery returns a list of valid values
- Example:
  - Select the department_id and department_name for all departments being managed by any manager who is also managing employees named Smith
  - Employee name is in the employees table
    - The employees subquery is expected to return multiple manager ids
    - Use the IN operator

```
SELECT department_id, department_name
FROM departments
WHERE manager_id IN (
     SELECT manager_id
     FROM employees
     WHERE last_name = 'Smith'
);
```

- NOT IN can replace the IN operator in this query to get departments for all other managers

Correlated Subquery

- A simple subquery is evaluated once, and its result is compared with each row of the parent query
- A correlated subquery is evaluated and compared once for each row of the parent table
  - Necessary because a correlated subquery uses data from the parent query which is expected to change with every row
  - Powerful because the decision is based on data in the parent query and subquery
- Syntax:
  - Same as a simple subquery, but references a column of the parent query
  - Reference is made by specifying a column from one of the tables of the parent query

`EXISTS` Operator

- Allows testing for existence rather than for specific values
  - Evaluates to `TRUE` if the subquery returns at least one row; `FALSE` otherwise
  - Efficient because the subquery stops after finding the first row
- Syntax:

```
WHERE [NOT] EXISTS subquery
```

- Does not compare the results of the subquery with anything
- `NOT EXISTS`
  - Evaluates to `TRUE` if the subquery returns no rows
  - Not efficient because the subquery must check all rows to complete

Correlated Subquery Example

- Find departments that do not have any employees
  - For each department
    - Find all employees for this department (correlate on `department_id`)

```
SELECT d.department_name
FROM departments d
WHERE NOT EXISTS (
    SELECT 1
    FROM employees e
    WHERE e.department_id = d.department_id
);
```

- Because `EXISTS` tests for existence, the data selected by the subquery are not used
  - "1" is used as a dummy value because the `SELECT` list cannot be empty

Exercise 5.3: Using Subqueries

20 min

- Please complete this exercise in your Exercise Manual

Chapter Concepts

- Aggregate Functions
- The `GROUP BY` Clause
- The `HAVING` Clause
- `JOIN`ing Issues
- Subqueries
- **Chapter Summary**

Chapter Summary

In this chapter, we have discussed:

- Aggregate functions
  - `MAX, MIN, COUNT, SUM, AVG`
- `GROUP BY` clause
- The `HAVING` clause
- Final display:
  - `ORDER BY`
  - `JOIN`ing issues
  - Constructing aggregate in steps
- Working with subqueries