Aggregate Row Functions

DB2 Midterm

AGGREGATE ROW FUCTIONS

- Aggregate Row functions give the user the ability to answer business questions such as:
  - What is the average salary of an employee in the company?
  - What were the total salaries for a particular year?
  - What are the maximum and minimum salaries in the Computer's Department?

AGGREGATE ROW FUCTIONS

- Aggregate functions perform a variety of actions such as counting all the rows in a table, averaging a column's data, and summing numeric data.
- Aggregates can also search a table to find the highest "MAX" or lowest "MIN" values in a column.

AGGREGATE ROW FUCTIONS

- List of aggregate functions including their syntax and use.

Function Syntax	Function Use
SUM([ALL | DISTINCT] expression)	The total of the (distinct) values in a numeric column/expression.
AVG([ALL | DISTINCT] expression)	The average of the (distinct) values in a numeric column/expression.
COUNT([ALL | DISTINCT] expression)	The number of (distinct) non-NULL values in a column/expression.
COUNT(*)	The number of selected rows.
MAX(expression)	The highest value in a column/expression.
MIN(expression)	The lowest value in a column/expression.

AGGREGATE ROW FUCTIONS

- There are two rules that you must understand and follow when using aggregates:
- Aggregate functions can be used in both the SELECT and HAVING clauses (the HAVING clause is covered later in this chapter).
- Aggregate functions cannot be used in a WHERE clause.

EXAMPLE

- The following query is wrong and will produce the Oracle ORA-00934 *group function is not allowed here* error message.

```
SELECT *
FROM employee
WHERE emp_salary > AVG(emp_salary);
```

```
ERROR at line 3: ORA-00934: group function is not
  allowed here.
```

COUNT( )

- If a manager needs know how many employees work in the organization, COUNT(*) can be used to produce this information.
- The COUNT(*) function counts all rows in a table.
- The wild card asterisk (*) would be used as the parameter in the function.

```
SELECT COUNT(*)
FROM employee;

  COUNT(*)
  --------
         8
```

COUNT( )

- The result table for the COUNT(*) function is a single *scalar* value.
- Notice that the result table has a column heading that corresponds to the name of the aggregate function specified in the SELECT clause.
- The output column can be assigned a more meaningful column name as is shown in the revised query .

COUNT( )

- This is accomplished by simply listing the desired column name inside double-quotes after the aggregate function specification.

```
SELECT COUNT(*) "Number of Employees"
FROM employee;

Number of Employees
-------------------------
                        8
```

COUNT( )

- COUNT(*) is used to count all the rows in a table.
- COUNT(column name) does almost the same thing. The difference is that you may define a specific column to be counted.
- When column name is specified in the COUNT function, rows containing a NULL value in the specified column are omitted.
- A NULL value stands for “unknown” or “unknowable” and must not be confused with a blank or zero.

COUNT ( )

```
SELECT COUNT(emp_superssn) "Number Supervised Employees"
FROM employee;

Number Supervised Employees
---------------------------
                          7
```

- In contrast the count(*) will count each row regardless of NULL values.

```
SELECT COUNT(*) "Number of Employees“
FROM employee;
Number of Employees
-------------------
                  8
```

Using the AVG Function

- AVG function is used to compute the average value for the *emp_salary* column in the *employee* table.
- For example, the following query returns the average of the employee salaries.

```
SELECT AVG(emp_salary) "Average Employee
  Salary"
FROM employee;
```

```
Average Employee Salary
-----------------------
                 $35,500
```

More Examples

- What is the average salary offered to employees?
- This question asks you to incorporate the concept of computing the average of the distinct salaries paid by the organization.
- The same query with the DISTINCT keyword in the aggregate function returns a different average.

```
SELECT AVG(DISTINCT emp_salary) "Average Employee Salary"
FROM employee;

Average Employee Salary
-----------------------
                $38,200
```

Using the SUM Function

- The SUM function can compute the total of a specified table column.
- The SELECT statement shown here will return the total of the *emp_salary* column from the *employee* table.

```
SELECT SUM(emp_salary) "Total Salary"
FROM employee;

Total Salary
------------
    $284,000
```

More Examples

- If management is preparing a budget for various departments, you may be asked to write a query to compute the total salary for different departments.
- The query shown here will compute the total emp_salary for employees assigned to department #7.

```
SELECT SUM(emp_salary) "Total Salary Dept 7"
FROM employee
WHERE emp_dpt_number = 7;

Total Salary Dept 7
-------------------
            $136,000
```

MIN and MAX Functions

- The MIN function returns the lowest value stored in a data column.
- The MAX function returns the largest value stored in a data column.
- Unlike SUM and AVG, the MIN and MAX functions work with both numeric and character data columns.

Example

- A query that uses the MIN function to find the lowest value stored in the *emp_last_name* column of the *employee* table.
- This is analogous to determine which employee's last name comes first in the alphabet.
- Conversely, MAX() will return the employee row where last name comes last (highest) in the alphabet.

```
SELECT MIN(emp_last_name), MAX(emp_last_name)
FROM employee;

MIN(EMP_LAST_NAME)         MAX(EMP_LAST_NAME)
-------------------------- --------------------------
Amin                       Zhu
```

Using GROUP BY with Aggregate Functions

- The power of aggregate functions is greater when combined with the GROUP BY clause.
- In fact, the GROUP BY clause is rarely used without an aggregate function.
- It is possible to use the GROUP BY clause without aggregates, but such a construction has very limited functionality, and could lead to a result table that is confusing or misleading.

Example

- The following query displays how many employees work for each department?

```
SELECT emp_dpt_number "Department",
COUNT(*) "Department Count"
FROM employee
GROUP BY emp_dpt_number;

Department Department Count
---------- ----------------
         1                1
         3                3
         7                4
```

GROUP BY Clause

- Some RDBMs provides considerable flexibility in specifying the GROUP BY clause.
- The column name used in a GROUP BY does not have to be listed in the SELECT clause; however, it must be a column name from one of the tables listed in the FROM clause.

Example

- We could rewrite the last query without specifying the *emp_dpt_number* column as part of the result table, but as you can see below, the results are rather cryptic without the *emp_dpt_number* column to identify the meaning of the aggregate count.

```
SELECT COUNT(*) "Department Count"
FROM employee
GROUP BY emp_dpt_number;
Department Count
----------------
              1
              3
              4
```

Example

However, the reverse is NOT true!

```
SELECT emp_dpt_number, COUNT(*) "Department
    Count"
FROM employee;
```

```
SELECT emp_dpt_number, COUNT(*) "Department
    Count"
      *
ERROR at line 1:
ORA-00937: not a single-group group function
```

GROUP BY Clause

- ***To keep it simple, just remember the following:***

  1. If you have column name(s) AND Aggr. Function(s) in the SELECT clause, then you MUST also have a GROUP BY clause.

  2. The column name(s) in the SELECT clause MUST match column name(s) listed in the GROUP BY clause.

Example

```
SELECT  emp_dpt_number "Department",
            emp_gender "Gender",
            COUNT(*) "Department Count"
FROM employee
GROUP BY emp_dpt_number;
```

```
ERROR at line 2:
ORA-00979: not a GROUP BY expression
```

Example

```
SELECT  emp_dpt_number "Department",
            emp_gender "G",
            COUNT(*) “Employee Count"
FROM employee
GROUP BY emp_dpt_number, emp_gender;
```

```
Department G Employee Count
---------- - --------------
         1 M              1
         3 F              2
         3 M              1
         7 F              1
         7 M              3
```

Using GROUP BY With a WHERE Clause

- The WHERE clause works to eliminates data table rows from consideration before any grouping takes place.
- The query shown here produces an average hours worked result table for employees with a social security number that is larger than 999-66-0000.

```
SELECT work_emp_ssn SSN,
AVG(work_hours) "Average Hours Worked"
FROM assignment
WHERE work_emp_ssn > 999660000
GROUP BY  work_emp_ssn;

SSN       Average Hours Worked
--------- --------------------
999666666
999887777                 20.5
999888888                 21.5
```

Using GROUP BY With an ORDER BY Clause

- The ORDER BY clause allows you to specify how rows in a result table are sorted.
- The default ordering is from smallest to largest value.
- A GROUP BY clause in a SELECT statement will determine the sort order of rows in a result table.
- The sort order can be changed by specifying an ORDER BY clause after the GROUP BY clause.

Using GROUP BY With an ORDER BY Clause

```
SELECT emp_dpt_number "Department",
  AVG(emp_salary) "Average Salary"
FROM employee
GROUP BY emp_dpt_number
ORDER BY AVG(emp_salary);

Department Average Salary
---------- --------------
         3        $31,000
         7        $34,000
         1        $55,000
```

GROUP BY With a HAVING Clause

- The HAVING clause is used for aggregate functions in the same way that a WHERE clause is used for column names and expressions.
- The HAVING and WHERE clauses do the same thing, that is filter rows from inclusion in a result table based on a condition.
- a WHERE clause is used to filter rows **BEFORE** the GROUPING action.
- a HAVING clause filters rows **AFTER** the GROUPING action.

GROUP BY With a HAVING Clause

```
SELECT emp_dpt_number "Department",
  AVG(emp_salary) "Average Salary"
FROM employee
GROUP BY emp_dpt_number
HAVING AVG(emp_salary) > 33000;

Department Average Salary
---------- --------------
         1        $55,000
         7        $34,000
```

Combining HAVING Clause with Where clause

```
SELECT emp_dpt_number "Department",
  AVG(emp_salary) "Average Salary"
FROM employee
WHERE emp_dpt_number <> 1
GROUP BY emp_dpt_number
HAVING AVG(emp_salary) > 33000;

Department Average Salary
---------- --------------
         7       $34,000
```

GROUP BY With a HAVING Clause

Conceptually, SQL performs the following steps in the query given above.

1. The WHERE clause filters rows that do not meet the condition
emp_dpt_number <> 1.
2. The GROUP BY clause collects the surviving rows into one or more groups for each unique *emp_dpt_number*.
3. The aggregate function calculates the average salary for each *emp_dpt_number* grouping.
4. The HAVING clause filters out the rows from the result table that do not meet the condition average salary greater than $33,000.

More Examples

```
SELECT emp_dpt_number "Department",
 COUNT(*) "Department Count",
 MAX(emp_salary) "Top Salary",
 MIN(emp_salary) "Low Salary"
FROM employee
GROUP BY emp_dpt_number
HAVING COUNT(*) >= 3;
```

Department	Department Count	Top Salary	Low Salary
3	3	$43,000	$25,000
7	4	$43,000	$25,000

GROUP BY With a HAVING Clause

- The HAVING clause is a conditional option that is directly related to the GROUP BY clause option because a HAVING clause eliminates rows from a result table based on the result of a GROUP BY clause.

- In Oracle, A HAVING clause will not work without a GROUP BY clause.

GROUP BY With a HAVING Clause

```
SELECT emp_dpt_number,
  AVG(emp_salary)
FROM employee
HAVING AVG(emp_salary) > 33000;

ERROR at line 1:
ORA-00937: not a single-group
  group function
```

Exercise

Function	Use Case	SQL Script	Screenshot
COUNT			
SUM			
AVERAGE			
MIN			
MAX			
GROUP BY			
Distinct			

- Github Repo: **DB2/<mmddyy>** e.g. DB2/013020
- Tasks (By Pair)
  - Complete the “aggregate functions” table shown and
  - Submit the complete ERD Northwind Schema with its relationship types (One-to-One, One-to-many, and Many-to-many.
- Deliverables
  - Lastname1_lastname2_aggregate.pdf (Kindly push also all scripts)
  - Northwind Schema ERD