COURSE NAME: DBMS
COURSE CODE:23AD2102A

TOPIC:

AGGREGATION FUNCTIONS AND VIEWS

Session - 11

AIM OF THE SESSION

To familiarize students with the advance and complex Subqueries in PostgreSQL.

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

1. Discuss the subqueries.
2. Various guidelines and types of subqueries.

LEARNING OUTCOMES

At the end of this session, you should be able to understand the basic concepts of Subqueries and learn how to write complex subqueries with PostgreSQL commands.

AGGREGATE FUNCTIONS

- An aggregate function in SQL performs a calculation on multiple values and returns a single value. SQL provides many aggregate functions that include avg, count, sum, min, max, etc. An aggregate function ignores NULL values when it performs the calculation, except for the count function

- Used to accumulate information from multiple tuples, forming a single- tuple summary
- Built-in aggregate functions
  - `COUNT`, `SUM`, `MAX`, `MIN`, and `AVG`
- Used in the `SELECT` clause

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
-------------------------
```

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
```

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
```

CREATING AND DROPPING VIEWS

- A view can contain all rows of a table or selected rows from one or more tables. A view can be created from one or many tables, which depends on the written PostgreSQL query to create a view.

- Views, which are kind of virtual tables, allow users to do the following:

  a. Structure data in a way that users or classes of users find natural or intuitive.

  b. Restrict access to the data such that a user can only see limited data instead of complete table.

  c. Summarize data from various tables, which can be used to generate reports.

Creating Views: The PostgreSQL views are created using the CREATE VIEW statement. The PostgreSQL views can be created from a single table, multiple tables, or another view.

Syntax:

```
CREATE [TEMP | TEMPORARY] VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

CREATING AND DROPPING VIEWS

Example: Consider, the following COMPANY table is having the following records:

```
id | name  | age | address    | salary
---+-------+-----+------------+--------
 1 | Paul  |  32 | California |  20000
 2 | Allen |  25 | Texas      |  15000
 3 | Teddy |  23 | Norway     |  20000
 4 | Mark  |  25 | Rich-Mond  |  65000
 5 | David |  27 | Texas      |  85000
 6 | Kim   |  22 | South-Hall |  45000
 7 | James |  24 | Houston    |  10000
```

To create a view from COMPANY table. This view would be used to have only few columns from COMPANY table:

```
testdb=# CREATE VIEW COMPANY_VIEW AS
SELECT ID, NAME, AGE
FROM  COMPANY;
```

We can query COMPANY_VIEW in a similar way as we query an actual table as the following:

```
testdb=# SELECT * FROM COMPANY_VIEW;
```

Output:

```
id | name  | age
---+-------+-----
 1 | Paul  |  32
 2 | Allen |  25
 3 | Teddy |  23
 4 | Mark  |  25
 5 | David |  27
 6 | Kim   |  22
 7 | James |  24
(7 rows)
```

Dropping Views: To drop a view, simply use the DROP VIEW statement with the view_name. The basic DROP VIEW syntax is as follows:

```
testdb=# DROP VIEW COMPANY_VIEW;
```

COMPOUND STATEMENTS

A Compound Statement is in principal the essential block of the SQL/PSM language (PSM --> "Persistent, Stored Modules" allows us to store procedures as database schema elements. PSM = a mixture of conventional statements (if, while, etc.) and SQL). It enables us to enter the sequence of statements, declare local variables, conditions and subroutines, errors (exceptions) and warnings handling. The declaration of variables can not be intersected by the declaration of cursors, errors handling, etc. The variables can be set to the default value. The range of a compound statement is determined by the pair BEGIN END.

Example:

```
CREATE OR REPLACE FUNCTION report()
RETURNS void AS
$$
  bl:BEGIN
      DECLARE done boolean DEFAULT false;
      DECLARE a, b integer;
      DECLARE cx CURSOR FOR SELECT f.a, f.b FROM Foo f;
      DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = true;

      OPEN cx;
      FETCH cx INTO bl.a, bl.b;

      WHILE NOT done
      DO
        PRINT bl.a, bl.b;
        FETCH cx INTO bl.a, bl.b;
      END WHILE;

      CLOSE cx;
    END bl;
$$ LANGUAGE plpgpsm;
```

USER DEFINED FUNCTIONS

PostgreSQL uses the CREATE FUNCTION statement to develop user-defined functions.

Syntax:

```
CREATE FUNCTION function_name(p1 type, p2 type)
 RETURNS type AS
BEGIN
 -- logic
END;
LANGUAGE language_name;
```

Details of syntax:

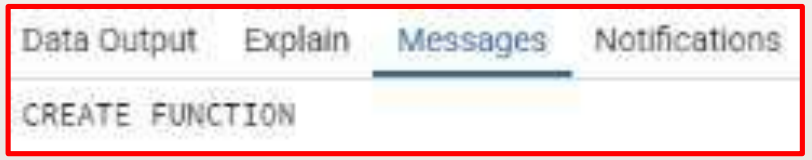
a. First, specify the name of the function after the CREATE FUNCTION keywords.
b. Then, put a comma-separated list of parameters inside the parentheses following the function name.
c. Next, specify the return type of the function after the RETURNS keyword.
d. After that, place the code inside the BEGIN and END block. The function always ends with a semicolon (;) followed by the END keyword.
e. Finally, indicate the procedural language of the function e.g., plpgsql in case PL/pgSQL is used.

USER DEFINED FUNCTIONS

Example: We will develop a very simple function named **inc** that increases an integer by 1 and returns the result.

```
CREATE FUNCTION inc(val integer) RETURNS integer AS $$
BEGIN
RETURN val + 1;
END; $$
LANGUAGE PLPGSQL;
```

If the function is valid, PostgreSQL will create the function and return the CREATE FUNCTION statement as the following.



We can call the **inc** function like any built-in functions as follows:

Output

```
SELECT inc(20);
```

If we call the **inc** function 2 times (nested), the result is as the following:

```
SELECT inc(inc(20));
```

SUMMARY

An aggregate function in SQL performs a calculation on multiple values and returns a single value. SQL provides many aggregate functions that include avg, count, sum, min, max, etc. An aggregate function ignores NULL values when it performs the calculation, except for the count function

SELF-ASSESSMENT QUESTIONS

1. Which of the following is true about sub-queries?

a) They execute after the main query executes.
b) They execute in parallel to the main query.
c) The user can execute the main query and then, if wanted, execute the sub-query.
d) They execute before the main query executes.

2. Which of the following clause is mandatorily used in a sub-query?

(a) SELECT
(b) WHERE
(c) ORDER BY
(d) GROUP BY

SELF-ASSESSMENT QUESTIONS

3. Which of the following multi-row operators can be used with a sub-query?

(a) IN
(b) ANY
(c) ALL
(d) ALL OF THE ABOVE

4. Which of the following is true about the result of a sub-query?

a) The result of a sub-query is generally ignored when executed.
b) The result of a sub-query doesn't give a result, it is just helpful in speeding up the main query execution.
c) The result of a sub-query is used by the main query.
d) The result of a sub-query is always NULL.

TERMINAL QUESTIONS

1. Describe various types of SQL complex subqueries.

2. List out the guidelines for creating the SQL subqueries.

3. Analyze the use of ALL,IN, or ANY operator while using subqueries in PostgreSQL.

REFERENCES FOR FURTHER LEARNING OF THE SESSION

Reference Books:

1. Database System Concepts, Sixth Edition, Abraham Silberschatz, Yale University Henry, F. Korth Lehigh University, S. Sudarshan Indian Institute of Technology, Bombay.

2. An Introduction to Database Systems by Bipin C. Desai

3. Fundamentals of Database Systems, 7th Edition, RamezElmasri, University of Texas at Arlington, Shamkant B. Navathe, University of Texasat Arlington.

Sites and Web links:

1. https://www.geeksforgeeks.org/postgresql-create-table/

2. https://www.tutorialsteacher.com/postgresql

THANK YOU

Team – DBMS