POSTGRESQL - SUB QUERIES

A subquery or Inner query or Nested query is a query within another PostgreSQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE and DELETE statements along with the operators like =, <, >, >=, <=, IN, etc.

There are a few rules that subqueries must follow:

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN, EXISTS, NOT IN, ANY/SOME, ALL operator.
- The BETWEEN operator cannot be used with a subquery; however, the BETWEEN can be used within the subquery.

Subqueries with the SELECT Statement:

Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows:

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
      (SELECT column_name [, column_name ]
      FROM table1 [, table2 ]
      [WHERE])
```

Example:

Consider 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
(7 rows)
```

Now, let us check following sub-query with SELECT statement:

```
testdb=# SELECT *
     FROM COMPANY
     WHERE ID IN (SELECT ID
                  FROM COMPANY
                  WHERE SALARY > 45000) ;
```

This would produce the following result:

```
 id | name  | age |  address    | salary
----+-------+-----+-------------+--------
  4 | Mark  |  25 | Rich-Mond   |  65000
  5 | David |  27 | Texas       |  85000
(2 rows)
```

Subqueries with the INSERT Statement:

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date, or number functions.

The basic syntax is as follows:

```
INSERT INTO table_name [ (column1 [, column2 ]) ]
           SELECT [ *|column1 [, column2 ]
           FROM table1 [, table2 ]
           [ WHERE VALUE OPERATOR ]
```

Example:

Consider a table COMPANY_BKP with similar structure as COMPANY table and can be created using same CREATE TABLE using COMPANY_BKP as table name. Now to copy complete COMPANY table into COMPANY_BKP, following is the syntax:

```
testdb=# INSERT INTO COMPANY_BKP
     SELECT * FROM COMPANY
     WHERE ID IN (SELECT ID
                  FROM COMPANY) ;
```

Subqueries with the UPDATE Statement:

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

The basic syntax is as follows:

```
UPDATE table
SET column_name = new_value
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
   FROM TABLE_NAME)
   [ WHERE) ]
```

Example:

Assuming, we have COMPANY_BKP table available which is backup of COMPANY table.

Following example updates SALARY by 0.50 times in COMPANY table for all the customers, whose AGE is greater than or equal to 27:

```
testdb=# UPDATE COMPANY
     SET SALARY = SALARY * 0.50
     WHERE AGE IN (SELECT AGE FROM COMPANY_BKP
                   WHERE AGE >= 27 );
```

This would impact two rows and finally COMPANY table would have the following records:

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

```
  1 | Paul  |  32 | California  |  10000
  5 | David |  27 | Texas       |  42500
(7 rows)
```

Subqueries with the DELETE Statement:

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

The basic syntax is as follows:

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
   (SELECT COLUMN_NAME
   FROM TABLE_NAME)
   [ WHERE) ]
```

Example:

Assuming, we have COMPANY_BKP table available which is backup of COMPANY table.

Following example deletes records from COMPANY table for all the customers, whose AGE is greater than or equal to 27:

```
testdb=# DELETE FROM COMPANY
     WHERE AGE IN (SELECT AGE FROM COMPANY_BKP
                   WHERE AGE > 27 );
```

This would impact two rows and finally COMPANY table would have the following records:

```
 id | name  | age | address     | salary
----+-------+-----+-------------+--------
  2 | Allen |  25 | Texas       |  15000
  3 | Teddy |  23 | Norway      |  20000
  4 | Mark  |  25 | Rich-Mond   |  65000
  6 | Kim   |  22 | South-Hall  |  45000
  7 | James |  24 | Houston     |  10000
  5 | David |  27 | Texas       |  42500
(6 rows)
```