**SUBQUERY**

A subquery is a SQL query nested inside a larger query.

* A subquery may occur in:
* In PostgreSQL subquery can be nested inside a SELECT, INSERT, UPDATE, DELETE, SET, or DO statement or inside another subquery.
* A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
* You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, SOME, or ALL.
* A subquery can be treated as an inner query, which is a SQL query placed as a part of another query called as outer query.
* The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.

**Emp ={**employee\_id , first\_namem, last\_name, email, phone\_number, hire\_date, job\_id, salary, commission\_pct, manager\_id, department\_id}

1. list the name of the employees, paid more than 'Alexander' from employees.

SELECT first\_name,last\_name, salary FROM employees

WHERE salary >

(SELECT max(salary) FROM employees

WHERE first\_name='Alexander');

**Types of Subqueries**

* The Subquery as Scalar Operand
* Comparisons using Subqueries
* Subqueries with ALL, ANY, IN, or SOME
* Row Subqueries
* Subqueries with EXISTS or NOT EXISTS
* Correlated Subqueries
* Subqueries in the FROM Clause

**Subquery :**

A scalar subquery is a subquery that returns exactly one column value from one row.

* The SELECT query is executed and the single returned value is used in the surrounding value expression.
* It is an error to use a query that returns more than one row or more than one column as a scalar subquery.
* During a particular execution, if the subquery returns no rows, that is not an error; the scalar result is taken to be null.

SELECT employee\_id, last\_name,

(CASE WHEN department\_id=(

SELECT department\_id from departments WHERE location\_id=2500)

THEN 'Canada' ELSE 'USA' END)

FROM employees;

**Subqueries: Using Comparisons**

A subquery can be used before or after any of the comparison operators. The subquery can return at most one value. The value can be the result of an arithmetic expression or a column function. SQL then compares the value that results from the subquery with the value on the other side of the comparison operator.

suppose you want to find the employee id, first\_name, last\_name, and salaries for employees whose average salary is higher than the average salary throughout the company.

SELECT employee\_id,first\_name,last\_name,salary

FROM employees

WHERE salary >

(SELECT AVG(SALARY) FROM employees);

**Subqueries with ALL operator**

The ALL operator compares value to every value returned by the subquery. The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result using the given operator, which must yield a Boolean result.

* The result of ALL is true if all rows yield true (including the case where the subquery returns no rows).
* The result is false if any false result is found.
* The result is NULL if the comparison does not return false for any row, and it returns NULL for at least one row.

The following query selects the department with the highest average salary. The subquery finds the average salary for each department, and then the main query selects the department with the highest average salary

SELECT department\_id, AVG(SALARY)

FROM employees GROUP BY department\_id

HAVING AVG(SALARY)>=ALL

(SELECT AVG(SALARY) FROM employees

GROUP BY department\_id);

**Subqueries with ANY/SOME operator:**

The ANY operator compares the value to each value returned by the subquery. Therefore ANY keyword (which must follow a comparison operator) returns TRUE if the comparison is TRUE for ANY of the values in the column that the subquery returns.

SOME is a synonym for ANY. IN is equivalent to = ANY.

The following query selects any employee who works in the location 1700. The subquery finds the department id in the 1700 location, and then the main query selects the employees who work in any of these departments.

SELECT first\_name, last\_name,department\_id

FROM employees

WHERE department\_id= ANY

(SELECT DEPARTMENT\_ID

FROM departments WHERE location\_id=1700);

**Subquery, IN operator:**

SELECT first\_name, last\_name,department\_id

FROM employees

WHERE department\_id IN

(SELECT DEPARTMENT\_ID FROM departments

WHERE location\_id=1800);

**Subqueries with EXISTS operator:**

The argument of EXISTS is an arbitrary SELECT statement, or subquery. The subquery is evaluated to determine whether it returns any rows. If it returns at least one row, the result of EXISTS is true; if the subquery returns no rows, the result of EXISTS is false.

Example: PostgreSQL Subqueries with EXISTS

The following query finds employees (employee\_id, first\_name, last\_name, job\_id, department\_id) from employees table who have at least one person reporting to them.

SELECT employee\_id, first\_name, last\_name, job\_id, department\_id

FROM employees E

WHERE EXISTS

(SELECT \* FROM employees

WHERE manager\_id = E.employee\_id);

SELECT employee\_id, first\_name, last\_name, job\_id, department\_id

FROM employees E

WHERE NOT EXISTS

(SELECT \* FROM employees

WHERE manager\_id = E.employee\_id);

**Correlated Subqueries:**

A correlated subquery is a subquery that contains a reference to a table (in the parent query) that also appears in the outer query. PostgreSQL evaluates from inside to outside.

Find all employees who earn more than the average salary in their department.

SELECT last\_name, salary, department\_id

FROM employees outerr

WHERE salary>

(SELECT AVG(salary)

FROM employees

WHERE department\_id = outerr.department\_id);

**job\_history table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMPLOYEE\_ID** | **START\_DATE** | **END\_DATE** | **JOB\_ID** | **DEPARTMENT\_ID** |
| 102 | 13-Jan-93 | 24-Jul-98 | IT\_PROG | 60 |
| 101 | 21-Sep-89 | 27-Oct-93 | AC\_ACCOUNT | 110 |
| 101 | 28-Oct-93 | 15-Mar-97 | AC\_MGR | 110 |
| 201 | 17-Feb-96 | 19-Dec-99 | MK\_REP | 20 |
| 114 | 24-Mar-98 | 31-Dec-99 | ST\_CLERK | 50 |
| 122 | 1-Jan-99 | 31-Dec-99 | ST\_CLERK | 50 |
| 200 | 17-Sep-87 | 17-Jun-93 | AD\_ASST | 90 |
| 176 | 24-Mar-98 | 31-Dec-98 | SA\_REP | 80 |
| 176 | 1-Jan-99 | 31-Dec-99 | SA\_MAN | 80 |
| 200 | 1-Jul-94 | 31-Dec-98 | AC\_ACCOUNT | 90 |

From the employees and job\_history tables display details of those employees who have changed jobs at least once.

SELECT first\_name, last\_name, employee\_id, job\_id

FROM employees E

WHERE 1 <=

(SELECT COUNT(\*) FROM Job\_history

WHERE employee\_id = E.employee\_id);

**Subqueries in the FROM Clause:**

Subqueries works in a SELECT statement's FROM clause. The syntax is:

SELECT ... FROM (subquery) [AS] name ...

Every table in a FROM clause must have a name, therefore the [AS] name clause is mandatory. Any columns in the subquery select list must have unique names.

table tb1:

CREATE TABLE tb1 (c1 INT, c2 CHAR(5), c3 FLOAT);

INSERT INTO tb1 VALUES (1, '1', 1.0), (2, '2', 2.0), (3, '3', 3.0);

SELECT \* FROM tb1;

Sample Output:

c1 | c2 | c3

----+-------+----

1 | 1 | 1

2 | 2 | 2

3 | 3 | 3

(3 rows)

**Example:** subquery in the FROM clause, using the example table (tb1)

SELECT sc1, sc2, sc3

FROM (SELECT c1 AS sc1, c2 AS sc2, c3\*3 AS sc3 FROM tb1) AS sb

WHERE sc1>1;

Sample Output:

sc1 | sc2 | sc3

-----+-------+-----

2 | 2 | 6

3 | 3 | 9

(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.

### Example

Consider the [COMPANY](https://www.tutorialspoint.com/postgresql/company.sql) table 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

INSERT INTO COMPANY\_BKP

SELECT \* FROM COMPANY

WHERE ID IN (SELECT ID

FROM COMPANY) ;

## Subqueries with the UPDATE Statement

UPDATE COMPANY

SET SALARY = SALARY \* 0.50

WHERE AGE IN (SELECT AGE FROM COMPANY\_BKP

WHERE AGE >= 27 );