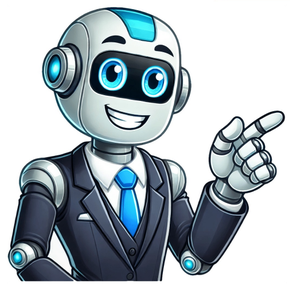
Click Here



A subquery in SQL is a smaller query that is nested inside a larger query. It can be used in various parts of a SQL statement, such as the SELECT, FROM, WHERE, or HAVING clause. The subquery is executed first, and its results are then passed to the outer query. This allows you to perform complex queries that involve comparing values, determining if an expression is included in the results, or checking if there are any rows selected. Subqueries can be used in various SQL statements, including SELECT, INSERT, UPDATE, and DELETE. They can be used to compare expressions, determine if an expression is included in the results, or check if there are any rows selected. The syntax for a subquery is simple: the inner query (subquery) is executed first, and then its results are passed to the outer query. Here's an example of how you might use a subquery in a SELECT statement: ```sql SELECT first_name, (SELECT department_name FROM departments WHERE departments.department_id = employees.department_id) AS department_name FROM employees; ``` This query selects the first name and department name for each employee. The subquery retrieves the department name from the departments table based on the department ID of each employee. The main locations where you can use a subquery are: * In the SELECT clause, to return a single value or a set of values * In the FROM clause, to join tables based on common columns * In the WHERE clause, to filter rows based on conditions specified by the subquery * In the HAVING clause, to group and aggregate data based on conditions specified by the subquery SELECT * FROM (SELECT first_name, salary FROM employees WHERE salary > 5000) AS "high_salaried"; SELECT first_name FROM employees WHERE department_id IN (SELECT department_id FROM departments WHERE location_id>1500); SELECT department_id, AVG(salary) FROM employees GROUP BY department_id HAVING AVG(salary) > (SELECT AVG(salary) FROM employees); Given article text here This SQL query links two tables based on student ID, filters results where total marks are greater than 80 for each student, and retrieves data about students with higher scores than another student. The subquery helps in dynamic filtering by comparing the total marks of other students with a specific value from the 'marks' table. Given article text here SELECT * FROM employees WHERE department_id IN (SELECT department_id FROM departments); To extract specific orders from a database table, SQL queries can be used to filter and copy data into a new table. For instance, in the given example, a query is provided that inserts data from the 'orders' table into the 'neworder' table based on the advance amount. The SQL code snippet for inserting data into the 'neworder' table selects all columns where the advance_amount is either 2000 or 5000. This means only rows with these specific advance amounts will be copied into the new table. Additionally, an example of using subqueries with the UPDATE statement is provided to update a column in the 'neworder' table based on conditions specified by another query. In this case, it updates the ord_date to '15-JAN-10' for rows where the difference between ord_amount and advance_amount is less than the minimum value of ord_amount in the 'orders' table. Given article text here DELETE FROM neworder WHERE advance_amount < (SELECT MAX(advance_amount) FROM orders); The provided text explains the concept of subqueries in SQL, their types, and how they can be used in various SQL statements. It highlights the difference between inner and outer queries, the general rules for using subqueries, and their common applications. A subquery is a query nested inside another query, which executes first and provides its result to the outer query. Subqueries are used to compare expressions, determine if an expression is included in the results of another query, or check whether the query selects any rows. Subqueries can be used with various SQL statements such as SELECT, INSERT, UPDATE, and DELETE statements. They must be enclosed in parentheses and placed on the right side of the comparison operator. Subqueries cannot include an ORDER BY clause, but it can be used in the main (outer) query. The text also explains the different types of subqueries, including scalar, column, multiple column, single row, multiple row, table, correlated, and nested subqueries. Additionally, it mentions that subqueries can be used with INSERT, UPDATE, and DELETE statements to insert or update data based on the results of another query. Note: Outputs shown here are taken using Oracle Database 10g Express Edition. To further develop your skills in using SQL subqueries to craft flexible queries for data retrieval from databases. This tutorial will cover how to utilize SQL subqueries within other queries to retrieve data efficiently. A subquery is an SQL query nested inside another query, with the outer query being referred to as a main query. To write a subquery, it's essential to have a thorough understanding of the SELECT statement syntax: `SELECT select_list FROM table1 INNER JOIN table2 ON join_condition WHERE filter_condition;`. The JOIN clause can be either an INNER JOIN, LEFT JOIN, RIGHT JOIN, or FULL JOIN. In this syntax: * The SELECT clause can accept a single value, which can be a column or expression. * The FROM and INNER JOIN clauses can accept a result set like a table. * The WHERE clause can accept a single value, which can be a column or expression. Based on the shape of the data each clause accepts, you can embed an appropriate subquery. A subquery in the SELECT clause returns a single value, while a subquery in the FROM or INNER JOIN clauses returns a result set. A subquery in the WHERE clause also returns a single value. Job IDs related to sales roles are extracted: job_id 15, 16. Next, employees with these job IDs are selected. A subquery retrieves first names, salaries, and average salaries for all employees: SELECT first_name, salary, (SELECT ROUND(AVG(salary), 2) average_salary FROM employees) FROM employees ORDER BY salary; The result is presented in a table showing employee information along with the overall average salary. To illustrate using a subquery in the FROM clause, consider calculating the average departmental salary by combining employee salaries within each department: SELECT ROUND(AVG(department_salary), 0) average_department_salary FROM (SELECT department_id, SUM(salary) department_salary FROM employees GROUP BY department_id); This query generates a result set showing department IDs and total salaries. The outer query then calculates the overall average departmental salary, rounded to zero decimal places. A subquery can also be used in the INNER JOIN clause to identify employees earning above the company's average salary: SELECT first_name, last_name, salary, s.avg_salary FROM employees e INNER JOIN (SELECT ROUND(AVG(salary), 0) AS avg_salary FROM employees) s ON e.salary > s.avg_salary ORDER BY salary; This query retrieves employee information and compares it to the average salary.

Why are joins better than subqueries. Why subqueries are bad. Why subqueries. What are subqueries and why are they used.