

LAB 9: Subqueries.

Objectives

After completing this lesson, you should be able to do the following:

- ✓ Describe the types of problem that subqueries can solve
- ✓ Define subqueries
- ✓ List the types of subqueries
- ✓ Write single-row and multiple-row subqueries

Lesson Aim

In this lesson, you will learn about more advanced features of the SELECT statement. You can write subqueries in the WHERE clause of another SQL statement to obtain values based on an unknown conditional value. This lesson covers single-row subqueries and multiple-row subqueries.

Using a Subquery to Solve a Problem Who has a salary greater than Abel's?

Using a Subquery to Solve a Problem

Suppose you want to write a query to find out who earns a salary greater than Abel's salary.

To solve this problem, you need two queries: one to find what Abel earns, and a second query to find who earns more than that amount.

You can solve this problem by combining the two queries, placing one query *inside* the other query.

The inner query, also called the *subquery*, returns a value that is used by the outer query or the main query. Using a subquery is equivalent to performing two sequential queries and using the result of the first query as the search value in the second query.

Subquery Syntax

```
SELECT select_list
FROM table
WHERE expr operator
(SELECT select_list
FROM table);
```

Subqueries


A subquery is a SELECT statement that is embedded in a clause of another SELECT statement. You can build powerful statements out of simple ones by using subqueries. They can be very useful when you need to select rows from a table with a condition that depends on the data in the table itself.

You can place the subquery in a number of SQL clauses, including:

- The WHERE clause
- The HAVING clause
- The FROM clause

In the syntax: *operator* includes a comparison condition such as >, =, or IN

Note: Comparison conditions fall into two classes: single-row operators (>, =, >=, <, <=) and multiple row operators (IN, ANY, ALL).



Using a Subquery

```
SELECT last_name
FROM employees
WHERE salary >
(SELECT salary
FROM employees
WHERE last_name = 'Abel');
```

Using a Subquery

In the slide, the inner query determines the salary of employee Abel. The outer query takes the result of the inner query and uses this result to display all the employees who earn more than this amount.

Guidelines for Using Subqueries

- A subquery must be enclosed in parentheses.
- Place the subquery on the right side of the comparison condition for readability.
- Prior to release Oracle8i, subqueries could not contain an `ORDER BY` clause. Only one `ORDER BY` clause can be used for a `SELECT` statement, and if specified it must be the last clause in the main `SELECT` statement. Starting with release Oracle8i, an `ORDER BY` clause can be used and is required in the subquery to perform top-n analysis. Top-n analysis is covered in a later lesson.
- Two classes of comparison conditions are used in subqueries: single-row operators and multiple-row operators.

Types of Subqueries

- Single-row subqueries: Queries that return only one row from the inner `SELECT` statement
- Multiple-row subqueries: Queries that return more than one row from the inner `SELECT` statement

Note: There are also multiple-column subqueries: queries that return more than one column from the inner `SELECT` statement. This is discussed in a subsequent lesson.

Single-Row Subqueries

A single-row subquery is one that returns one row from the inner `SELECT` statement. This type of subquery uses a single-row operator. The slide gives a list of single-row operators.

Example

Display the employees whose job ID is the same as that of employee 141.

```
SELECT last_name, job_id
FROM employees
WHERE job_id =
  (SELECT job_id
   FROM employees
   WHERE employee_id = 141);
```

Executing Single-Row Subqueries

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id =
  (SELECT job_id
   FROM employees
   WHERE employee_id = 141)
AND salary >
  (SELECT salary
   FROM employees
   WHERE employee_id = 143);
```

Executing Single-Row Subqueries

A `SELECT` statement can be considered as a query block. The example on the slide displays employees whose job ID is the same as that of employee 141 and whose salary is greater than that of employee 143. The example consists of three query blocks: the outer query and two inner queries. The inner query blocks are executed first, producing the query results `ST_CLERK` and 2600, respectively. The outer query block is then processed and uses the values returned by the inner queries to complete its search conditions. Both inner queries return single values (`ST_CLERK` and 2600, respectively), so this SQL statement is called a single-row subquery.

Note: The outer and inner queries can get data from different tables.

Using Group Functions in a Subquery

```
SELECT last_name, job_id, salary
FROM employees
WHERE salary =
(SELECT MIN(salary)
FROM employees);
```

Using Group Functions in a Subquery

You can display data from a main query by using a group function in a subquery to return a single row.

The subquery is in parentheses and is placed after the comparison condition.

The example on the slide displays the employee last name, job ID, and salary of all employees whose salary is equal to the minimum salary. The `MIN` group function returns a single value (2500) to the outer query.

The HAVING Clause with Subqueries

Find the job with the lowest average salary.

```
SELECT job_id, AVG(salary)
FROM employees
GROUP BY job_id
HAVING AVG(salary) = (SELECT MIN(AVG(salary))
FROM employees
GROUP BY job_id);
```

What Is Wrong with This Statement?

```
SELECT employee_id, last_name
FROM employees
WHERE salary =
(SELECT MIN(salary)
FROM employees
GROUP BY department_id);
```

Errors with Subqueries

One common error with subqueries is more than one row returned for a single-row subquery.

In the SQL statement in the slide, the subquery contains a `GROUP BY` clause, which implies that the subquery will return multiple rows, one for each group it finds. In this case, the result of the subquery will be 4400, 6000, 2500, 4200, 7000, 17000, and 8300.

The outer query takes the results of the subquery (4400, 6000, 2500, 4200, 7000, 17000, 8300) and uses these results in its `WHERE` clause. The `WHERE` clause contains an equal (`=`) operator, a single-row comparison operator expecting only one value. The `=` operator cannot accept more than one value from the subquery and hence generates the error.

To correct this error, change the `=` operator to `IN`.

Multiple-Row Subqueries

Subqueries that return more than one row are called multiple-row subqueries. You use a multiple-row operator, instead of a single-row operator, with a multiple-row subquery. The multiple-row operator expects one or more values.

```
SELECT last_name, salary, department_id
FROM employees
WHERE salary IN (SELECT MIN(salary)
FROM employees
GROUP BY department_id);
```

Using the ANY Operator in Multiple-Row Subqueries

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ANY
(SELECT salary
FROM employees)
```

```
WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG'
```

Multiple-Row Subqueries (continued)

The ANY operator (and its synonym the SOME operator) compares a value to *each* value returned by a subquery. The slide example displays employees who are not IT programmers and whose salary is less than that of any IT programmer. The maximum salary that a programmer earns is \$9,000.

<ANY means less than the maximum. >ANY means more than the minimum. =ANY is equivalent to IN.
<ALL means less than the maximum. >ALL means more than the minimum.

Using the ALL Operator in Multiple-Row Subqueries

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ALL
(SELECT salary
FROM employees
WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG';
```

Multiple-Row Subqueries (continued)

The ALL operator compares a value to *every* value returned by a subquery. The example in the slide displays employees whose salary is less than the salary of all employees with a job ID of IT_PROG and whose job is not IT_PROG.

>ALL means more than the maximum, and <ALL means less than the minimum.

The NOT operator can be used with IN, ANY, and ALL operators

Practice

- Write a query to display the last name and hire date of any employee in the same department as Zlotkey. Exclude Zlotkey.

LAST_NAME	HIRE_DATE
Abel	11-MAY-96
Taylor	24-MAR-98

- Create a query to display the employee numbers and last names of all employees who earn more than the average salary. Sort the results in ascending order of salary.

EMPLOYEE_ID	LAST_NAME	SALARY
103	Hunold	9000
149	Zlotkey	10500
174	Abel	11000
205	Higgins	12000
201	Hartstein	13000
101	Kochhar	17000
102	De Haan	17000
100	King	24000

8 rows selected.

- Write a query that displays the employee numbers and last names of all employees who work in a department with any employee whose last name contains a *u*. Place your SQL statement in a text file named lab6_3.sql. Run your query.

EMPLOYEE_ID	LAST_NAME
124	Mourgos
141	Rajs
142	Davies
143	Matos
144	Vargas
103	Hunold
104	Ernst
107	Lorentz

8 rows selected.

4. Display the last name, department number, and job ID of all employees whose department location ID is 1700.

LAST_NAME	DEPARTMENT_ID	JOB_ID
Whalen	10	AD_ASST
King	90	AD PRES
Kochhar	90	AD_VP
De Haan	90	AD_VP
Higgins	110	AC_MGR
Gietz	110	AC_ACCOUNT

6 rows selected.

5. Display the last name and salary of every employee who reports to King.

LAST_NAME	SALARY
Kochhar	17000
De Haan	17000
Mourgos	5800
Zlotkey	10500
Hartstein	13000

6. Display the department number, last name, and job ID for every employee in the Executive department.

DEPARTMENT_ID	LAST_NAME	JOB_ID
90	King	AD PRES
90	Kochhar	AD_VP
90	De Haan	AD_VP