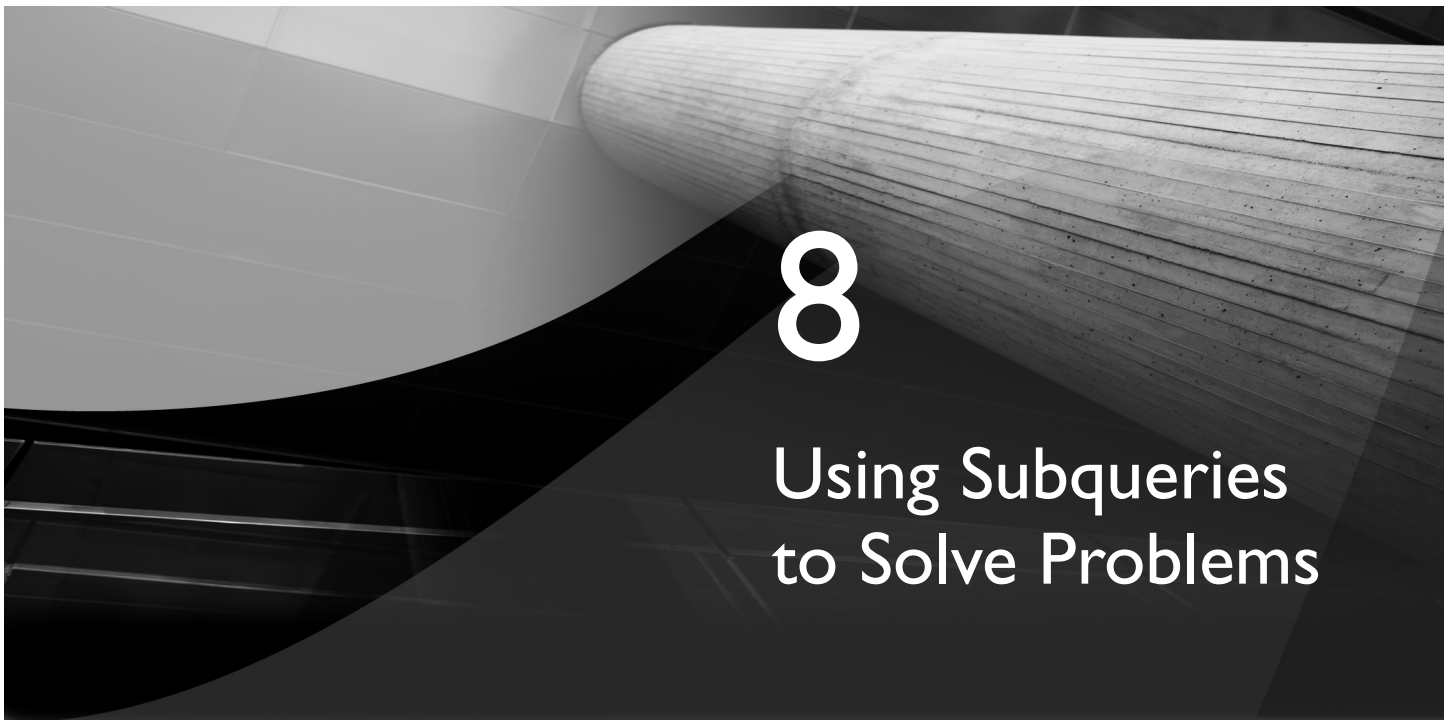


8 Using Subqueries to Solve Problems

CERTIFICATION OBJECTIVES

The previous six chapters have dealt with the SELECT statement in considerable detail, but in every case the SELECT statement has been a single, self-contained command. This chapter is the first of two that show how two or more SELECT commands can be combined into one statement. The first technique (covered in this chapter) is the use of *subqueries*. A subquery is a SELECT statement whose output is used as input to another SELECT statement (or indeed to a DML statement, as done in Chapter 10). The second technique is the use of set operators, where the results of several SELECT commands are combined into a single result set.

CERTIFICATION OBJECTIVE 8.01

Define Subqueries

A subquery is a query that is nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery. A subquery can return a set of rows or just one row to its parent query. A *scalar* subquery is a query that returns exactly one value: a single row, with a single column. Scalar subqueries can be used in most places in a SQL statement where you could use an expression or a literal value.

The places in a query where a subquery may be used are as follows:

- In the SELECT list used for column projection
- In the FROM clause
- In the WHERE clause
- In the HAVING clause

exam Watch

Subqueries can be nested to an unlimited depth in a FROM clause but to "only" 255 levels in a WHERE clause. They can be used in the SELECT list and in the FROM, WHERE, and HAVING clauses of a query.

A subquery is often referred to as an *inner* query, and the statement within which it occurs is then called the *outer* query. There is nothing wrong with this terminology, except that it may imply that you can only have two levels, inner and outer. In fact, the Oracle implementation of subqueries does not impose any practical limits on the level of nesting: the depth of nesting permitted in the FROM clause of a statement is unlimited, and that in the WHERE clause is up to 255.

A subquery can have any of the usual clauses for selection and projection. The following are required clauses:

- A SELECT list
- A FROM clause

The following are optional clauses:

- WHERE
- GROUP BY
- HAVING

The subquery (or subqueries) within a statement must be executed before the parent query that calls it, in order that the results of the subquery can be passed to the parent.

EXERCISE 8-1

Types of Subqueries

In this exercise, you will write code that demonstrates the places where subqueries can be used. Use either SQL*Plus or SQL Developer. All the queries should be run when connected to the HR schema.

1. Log on to your database as user HR.
2. Write a query that uses subqueries in the column projection list. The query will report on the current numbers of departments and staff:

```
select sysdate Today,
(select count(*) from departments) Dept_count,
(select count(*) from employees) Emp_count
from dual;
```

3. Write a query to identify all the employees who are managers. This will require using a subquery in the WHERE clause to select all the employees whose EMPLOYEE_ID appears as a MANAGER_ID:

```
select last_name from employees where
(employee_id in (select manager_id from employees));
```

4. Write a query to identify the highest salary paid in each country. This will require using a subquery in the FROM clause:

```
select max(salary),country_id from
      (select salary,department_id,location_id,country_id from
      employees natural join departments natural join locations)
group by country_id;
```

CERTIFICATION OBJECTIVE 8.02

Describe the Types of Problems That the Subqueries Can Solve

There are many situations where you will need the result of one query as the input for another.

Use of a Subquery Result Set for Comparison Purposes

Which employees have a salary that is less than the average salary? This could be answered by two statements, or by a single statement with a subquery. The following example uses two statements:

```
select avg(salary) from employees;
select last_name from employees where salary < result_of_previous_query ;
```

Alternatively, this example uses one statement with a subquery:

```
select last_name from employees
where salary < (select avg(salary)from employees);
```

In this example, the subquery is used to substitute a value into the WHERE clause of the parent query: it is returning a single value, used for comparison with the rows retrieved by the parent query.

The subquery could return a set of rows. For example, you could use the following to find all departments that do actually have one or more employees assigned to them:

```
select department_name from departments where department_id in
(select distinct(department_id) from employees);
```

In the preceding example, the subquery is used as an alternative to a join. The same result could have been achieved with the following:

```
select department_name from departments inner join employees
on employees.department_id = departments.department_id
group by department_name;
```

If the subquery is going to return more than one row, then the comparison operator must be able to accept multiple values. These operators are IN, NOT IN, ANY, and ALL. If the comparison operator is EQUAL, GREATER THAN, or LESS THAN (which each can only accept one value), the parent query will fail.

on the job

Using NOT IN is fraught with problems because of the way SQL handles NULLs. As a general rule, do not use NOT IN unless you are certain that the result set will not include a NULL.

Star Transformation

An extension of the use of subqueries as an alternative to a join is to enable the *star transformation* often needed in data warehouse applications. Consider a large table recording sales. Each sale is marked as being of a particular product to a particular buyer through a particular channel. These attributes are identified by codes, used as foreign keys to dimension tables with rows that describe each product, buyer, and channel. To identify all sales of books to buyers in Germany through Internet orders, one could run a query like this:

```
select … from sales s, products p, buyers b, channels c
where s.prod_code=p.prod_code
and s.buy_code=b.buy_code
and s.chan_code=c.chan_code
and p.product='Books'
and b.country='Germany'
and c.channel='Internet';
```

This query uses the WHERE clause to join the tables and then to filter the results. The following is an alternative query that will yield the same result:

```
select … from sales
where prod_code in (select prod_code from products where product='Books')
and buy_code in (select buy_code from buyers where country='Germany')
and chan_code in (select chan_code from channels where channel='Internet);
```

The rewrite of the first statement to the second is the star transformation. Apart from being an inherently more elegant structure (most SQL developers with any

sense of aesthetics will agree with that), there are technical reasons why the database may be able to execute it more efficiently than the original query. Also, star queries are easier to maintain; it is very simple to add more dimensions to the query or to replace the single literals ('Books,' 'Germany,' and 'Internet') with lists of values.

There is an instance initialization parameter, STAR_TRANSFORMATION_ENABLED, which (if set to true) will permit the Oracle query optimizer to re-write code into star queries.

Generate a Table from Which to SELECT

Subqueries can also be used in the FROM clause, where they are sometimes referred to as *inline views*. Consider another problem based on the HR schema: employees are assigned to a department, and departments have a location. Each location is in a country. How can you find the average salary of staff in a country, even though they work for different departments? Like this:

```
select avg(salary),country_id from
      (select salary,department_id,location_id,country_id from
      employees natural join departments natural join locations)
group by country_id;
```

The subquery constructs a table with every employee's salary and the country in which his department is based. The parent query then addresses this table, averaging the SALARY and grouping by COUNTRY_ID.

Generate Values for Projection

The third place a subquery can go is in the SELECT list of a query. How can you identify the highest salary and the highest commission rate and thus what the maximum commission paid would be if the highest salaried employee also had the highest commission rate? Like this, with two subqueries:

```
select
(select max(salary) from employees) *
(select max(commission_pct) from employees)
/ 100
from dual;
```

In this usage, the SELECT list used to project columns is being populated with the results of the subqueries. A subquery used in this manner must be scalar, or the parent query will fail with an error.

Generate Rows to be Passed to a DML Statement

DML statements are covered in detail in Chapter 10. For now, consider these examples:

```
insert into sales_hist select * from sales where date > sysdate-1;
update employees set salary = (select avg(salary) from employees);
delete from departments
where department_id not in (select department_id from employees);
```

exam Watch

A subquery can be used to select rows for insertion but not in a VALUES clause of an INSERT statement.

The first example uses a subquery to identify a set of rows in one table that will be inserted into another. The second example uses a subquery to calculate the average salary of all employees and passes this value (a scalar quantity) to an update statement. The third example uses a subquery to retrieve all DEPARTMENT_IDs that are in use and passes the list to a DELETE command, which will remove all departments that are not in use.

Note that it is not legal to use a subquery in the VALUES clause of an insert statement; this is fine:

```
insert into dates select sysdate from dual;
```

But this is not:

```
insert into dates (date_col) values (select sysdate fom dual);
```

EXERCISE 8-2

More Complex Subqueries

In this exercise, you will write some more complicated subqueries. Use either SQL*Plus or SQL Developer. All the queries should be run when connected to the HR schema.

1. Log on to your database as user HR.
2. Write a query that will identify all employees who work in departments located in the United Kingdom. This will require three levels of nested subqueries:

```
select last_name from employees where department_id in
      (select department_id from departments
      where location_id in
            (select location_id from locations
```

```
                        where country_id =
                              (select country_id from countries
                              where country_name='United Kingdom')
                        )
                  );
```

3. Check that the result from step 2 is correct by running the subqueries independently. First, find the COUNTRY_ID for the United Kingdom:

```
select country_id from countries where country_name='United Kingdom';
```

The result will be UK. Then find the corresponding locations:

```
select location_id from locations where country_id = 'UK';
```

The LOCATION_IDs returned will be 2400, 2500, and 2600. Then find the DEPARTMENT_IDs of department in these locations:

```
select department_id from departments where location_id in (2400,2500,2600);
```

The result will be two departments, 40 and 80. Finally, find the relevant employees:

```
select last_name from employees where department_id in (40,80);
```

4. Write a query to identify all the employees who earn more than the average and who work in any of the IT departments. This will require two subqueries, not nested:

```
select last_name from employees
where department_id in
(select department_id from departments where department_name like 'IT%')
and salary > (select avg(salary) from employees);
```

CERTIFICATION OBJECTIVE 8.03

List the Types of Subqueries

There are three broad divisions of subquery:

- Single-row subqueries
- Multiple-row subqueries
- Correlated subqueries

Single- and Multiple-Row Subqueries

The *single-row* subquery returns one row. A special case is the scalar subquery, which returns a single row with one column. Scalar subqueries are acceptable (and often very useful) in virtually any situation where you could use a literal value, a constant, or an expression. *Multiple-row* subqueries return sets of rows. These queries are commonly used to generate result sets that will be passed to a DML or SELECT statement for further processing. Both single-row and multiple-row subqueries will be evaluated once, before the parent query is run.

Single- and multiple-row subqueries can be used in the WHERE and HAVING clauses of the parent query, but there are restrictions on the legal comparison operators. If the comparison operator is any of the ones in the following table, the subquery must be a single-row subquery:

Symbol	Meaning
=	equal
>	greater than
>=	greater than or equal
<	less than
<=	less than or equal
<>	not equal
!=	not equal

If any of the operators in the preceding table are used with a subquery that returns more than one row, the query will fail. The operators in the following table can use multiple-row subqueries:

Symbol	Meaning
IN	equal to any member in a list
NOT IN	not equal to any member in a list
ANY	returns rows that match any value on a list
ALL	returns rows that match all the values in a list

exam Watch

The comparison operators valid for single-row subqueries are =, >, >=, <, <=, and <>. The comparison operators valid for multiple-row subqueries are IN, NOT IN, ANY, and ALL.

Correlated Subqueries

A *correlated subquery* has a more complex method of execution than single- and multiple-row subqueries and is potentially much more powerful. If a subquery references columns in the parent query, then its result will be dependent on the parent query. This makes it impossible to evaluate the subquery before evaluating the parent query. Consider this statement, which lists all employees who earn less than the average salary:

```
select last_name from employees
where salary < (select avg(salary) from employees);
```

The single-row subquery need only be executed once, and its result substituted into the parent query. But now consider a query that will list all employees whose salary is less than the average salary of their department. In this case, the subquery must be run for each employee to determine the average salary for her department; it is necessary to pass the employee's department code to the subquery. This can be done as follows:

```
select p.last_name, p.department_id from employees p
where p.salary < (select avg(s.salary) from employees s
where s.department_id=p.department_id);
```

In this example, the subquery references a column, `p.department_id`, from the select list of the parent query. This is the signal that, rather than evaluating the subquery once, it must be evaluated for every row in the parent query. To execute the query, Oracle will look at every row in EMPLOYEES and, as it does so, run the subquery using the DEPARTMENT_ID of the current employee row.

The flow of execution is as follows:

1. Start at the first row of the EMPLOYEES table.
2. Read the DEPARTMENT_ID and SALARY of the current row.
3. Run the subquery using the DEPARTMENT_ID from step 2.

4. Compare the result of step 3 with the SALARY from step 2, and return the row if the SALARY is less than the result.
5. Advance to the next row in the EMPLOYEES table.
6. Repeat from step 2.

A single-row or multiple-row subquery is evaluated once, before evaluating the outer query; a correlated subquery must be evaluated once for every row in the outer query. A correlated subquery can be single- or multiple-row, if the comparison operator is appropriate.

on the job

Correlated subqueries can be a very inefficient construct, due to the need for repeated execution of the subquery. Always try to find an alternative approach.

EXERCISE 8-3

Investigate the Different Types of Subqueries

In this exercise, you will demonstrate problems that can occur with different types of subqueries. Use either SQL*Plus or SQL Developer. All the queries should be run when connected to the HR schema: it is assumed that the EMPLOYEES table has the standard sets of rows.

1. Log on to your database as user HR.
2. Write a query to determine who earns more than Mr. Tobias:

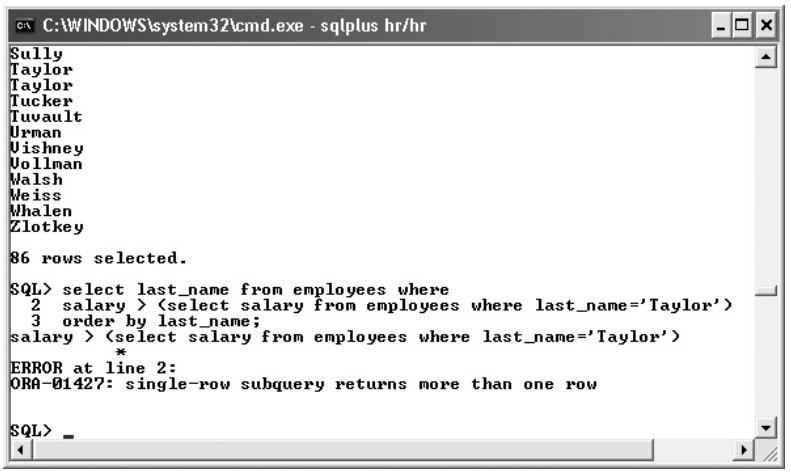
```
select last_name from employees where
salary > (select salary from employees where last_name='Tobias')
order by last_name;
```

This will return 86 names, in alphabetical order.

3. Write a query to determine who earns more than Mr. Taylor:

```
select last_name from employees where
salary > (select salary from employees where last_name='Taylor')
order by last_name;
```

This will fail with the error "ORA-01427: single-row subquery returns more than one row." The following illustration shows the last few lines of the output from step 2 followed by step 3 and the error, executed with SQL*Plus.



4. Determine why the query in step 2 succeeded but failed in step 3. The answer lies in the state of the data:

```
select count(last_name) from employees where last_name='Tobias';
select count(last_name) from employees where last_name='Taylor';
```

The use of the "greater than" operator in the queries for steps 2 and 3 requires a single-row subquery, but the subquery used may return any number of rows, depending on the search predicate used.

5. Fix the code in steps 2 and 3 so that the statements will succeed no matter what LAST_NAME is used. There are two possible solutions: one uses a different comparison operator that can handle a multiple-row subquery; the other uses a subquery that will always be single-row.

The first solution:

```
select last_name from employees where
salary > all (select salary from employees where last_name='Taylor')
order by last_name;
```

The second solution:

```
select last_name from employees where
salary > (select max(salary) from employees where last_name='Taylor')
order by last_name;
```

SCENARIO & SOLUTION	
How can you best design subqueries such that they will not fail with "ORA-01427: single-row subquery returns more than one row" errors?	There are two common techniques: use an aggregation so that if you do get multiple rows they will be reduced to one, or use one of the IN, ANY, or ALL operators so that it won't matter if multiple rows are returned. But these are both hacker's solutions; the real answer is always to use the primary key when identifying the row to be returned, not a nonunique key.
Sometimes there is a choice between using a subquery or using some other technique: the star transformation is a case in point. Which is better?	It depends on the circumstances. It is not uncommon for the different techniques to cause a different execution method within the database. Depending on how the instance, the database, and the data structures within it are configured, one may be much more efficient than another. Whenever such a choice arises, the statements should be subjected to a tuning analysis. Your DBA will be able to advise on this.

CERTIFICATION OBJECTIVE 8.04

Write Single-Row and Multiple-Row Subqueries

Following are examples of single- and multiple-row subqueries. They are based on the HR demonstration schema.

How would you figure out which employees have a manager who works for a department based in the United Kingdom? This is a possible solution, using multiple-row subqueries:

```
select last_name from employees
where manager_id in
(select employee_id from employees where department_id in
(select department_id from departments where location_id in
(select location_id from locations where country_id='UK')));
```

In the preceding example, subqueries are nested three levels deep. Note that the subqueries use the IN operator because it is possible that the queries could return several rows.

You have been asked to find the job with the highest average salary. This can be done with a single-row subquery:

```
select job_title from jobs natural join employees group by job_title
having avg(salary) =
(select max(avg(salary)) from employees group by job_id);
```

The subquery returns a single value: the average salary of the department with the highest average salary. It is safe to use the equality operator for this subquery because the MAX function guarantees that only one row will be returned.

The ANY and ALL operators are supported syntax, but their function can be duplicated with other more commonly used operators combined with aggregations. For example, these two statements, which retrieve all employees whose salary is above that of anyone in department 80, will return identical result sets:

```
select last_name from employees where salary > all
(select salary from employees where department_id=80);
select last_name from employees where salary >
(select max(salary) from employees where department_id=80);
```

The following table summarizes the equivalents for ANY and ALL:

Operator	Meaning
< ANY	less than the highest
> ANY	more than the lowest
= ANY	equivalent to IN
> ALL	more than the highest
< ALL	less than the lowest

EXERCISE 8-4

Write a Query That Is Reliable and User Friendly

In this exercise, develop a multi-row subquery that will prompt for user input. Use either SQL*Plus or SQL Developer. All the queries should be run when connected to the HR schema; it is assumed that the tables have the standard sets of rows.

1. Log on to your database as user HR.
2. Design a query that will prompt for a department name and list the last name of every employee in that department:

```
select last_name from employees where department_id =
(select department_id from departments
where department_name = '&Department_name');
```

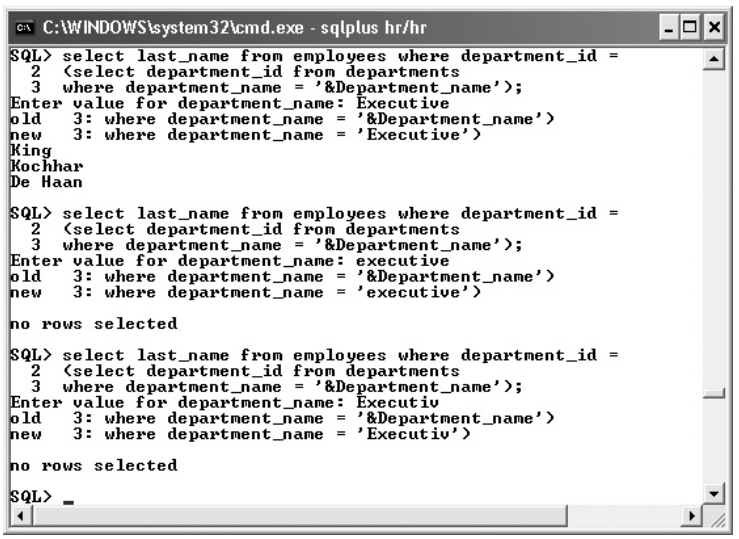
3. Run the query in step 2 three times, when prompted supplying these values:

   first time, Executive

   second time, executive

   third time, Executiv

The following illustration shows the result, using SQL*Plus:



4. Note the results from step 3. The first run succeeded because the value entered was an exact match, but the other failed. Adjust the query to make it more user friendly, so that it can handle minor variations in case or spelling:

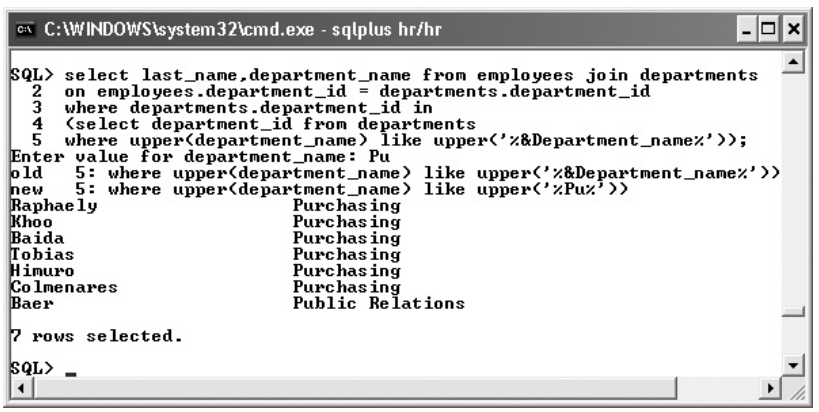
```
select last_name from employees where department_id =
(select department_id from departments
where upper(department_name) like upper('%&Department_name%'));
```

5. Run the query in step 4 three times, using the same values as used in step 3. This time, the query will execute successfully.
6. Run the query in step 4 again, and this time enter the value Pu. The query will fail, with an "ORA-01427: single-row subquery returns more than one row" error, because the attempt to make it more user-friendly means that the subquery is no longer guaranteed to be a single-row subquery. The string Pu matches two departments.

7. Adjust the query to make it resilient against the ORA-01427 error, and adjust the output to prevent any possible confusion:

```
select last_name,department_name from employees join departments
on employees.department_id = departments.department_id
where departments.department_id in
(select department_id from departments
where upper(department_name) like upper('%&Department_name%'));
```

The following illustration shows this final step: code that is approaching the ideal of being both bullet proof and user friendly:



INSIDE THE EXAM

Use of Subqueries

Subqueries come in three general forms: single-row, multiple-row, and correlated. A special case of the single-row subquery is the scalar subquery, a subquery that returns exactly one value. This is a single-row single-column subquery. For the first SQL OCP exam, detailed knowledge is expected only of scalar subqueries and single-column multiple-row subqueries. Correlated subqueries and multiple column subqueries are unlikely to be examined at this level, but a general knowledge of them may be tested.

When using subqueries in a WHERE clause, you must be aware of which operators will succeed with single-row subqueries and which will succeed with multiple-row subqueries.

CERTIFICATION SUMMARY

A subquery is a query embedded within another SQL statement. This statement can be another query or a DML statement. Subqueries can be nested within each other with no practical limits.

Subqueries can be used to generate values for the select list of a query to generate an inline view to be used in the FROM clause, in the WHERE clause, and in the HAVING clause. When used in the WHERE or HAVING clauses, single-row subqueries can be used with these comparison operators: =, >, >=, <, <=, <>; multiple-row subqueries can be used with these comparison operators: IN, NOT IN, ANY, ALL.

TWO-MINUTE DRILL

Define Subqueries

- ❑ A subquery is a select statement embedded within another SQL statement.
- ❑ Subqueries can be nested within each other.
- ❑ With the exception of the correlated subquery, subqueries are executed before the outer query within which they are embedded.

Describe the Types of Problems That the Subqueries Can Solve

- ❑ Selecting rows from a table with a condition that depends on the data within the table can be implemented with a subquery.
- ❑ Complex joins can sometimes be replaced with subqueries.
- ❑ Subqueries can add values to the outer query's output that are not available in the tables the outer query addresses.

List the Types of Subqueries

- ❑ Multiple-row subqueries can return several rows, possibly with several columns.
- ❑ Single-row subqueries return one row, possibly with several columns.
- ❑ A scalar subquery returns a single value; it is a single-row, single-column subquery.
- ❑ A correlated subquery is executed once for every row in the outer query.

Write Single-Row and Multiple-Row Subqueries

- ❑ Single-row subqueries should be used with single-row comparison operators.
- ❑ Multiple-row subqueries should be used with multiple-row comparison operators.
- ❑ The ALL and ANY operators can be alternatives to use of aggregations.

SELF TEST

Define Subqueries

1. Consider this generic description of a SELECT statement:

   SELECT *select_list*
   FROM *table*
   WHERE *condition*
   GROUP BY *expression_1*
   HAVING *expression_2*
   ORDER BY *expression_3* ;

   Where could subqueries be used? (Choose all correct answers.)

   A. *select_list*
   B. *table*
   C. *condition*
   D. *expression_1*
   E. *expression_2*
   F. *expression_3*

2. A query can have a subquery embedded within it. Under what circumstances could there be more than one subquery? (Choose the best answer.)

   A. The outer query can include an inner query. It is not possible to have another query within the inner query.
   B. It is possible to embed a single-row subquery inside a multiple-row subquery, but not the other way around.
   C. The outer query can have multiple inner queries, but they must not be embedded within each other.
   D. Subqueries can be embedded within each other with no practical limitations on depth.

3. Consider this statement:

```
select employee_id, last_name from employees where
salary > (select avg(salary) from employees);
```

   When will the subquery be executed? (Choose the best answer.)

   A. It will be executed before the outer query.
   B. It will be executed after the outer query.

C. It will be executed concurrently with the outer query.
D. It will be executed once for every row in the EMPLOYEES table.

4. Consider this statement:

```
select o.employee_id, o.last_name from employees o where
o.salary > (select avg(i.salary) from employees i
where i.department_id=o.department_id);
```

When will the subquery be executed? (Choose the best answer.)

A. It will be executed before the outer query.
B. It will be executed after the outer query.
C. It will be executed concurrently with the outer query.
D. It will be executed once for every row in the EMPLOYEES table.

Describe the Types of Problems That the Subqueries Can Solve

5. Consider the following statement:

```
select last_name from employees join departments
on employees.department_id = departments.department_id
where department_name='Executive';
```

and this statement:

```
select last_name from employees where department_id in
(select department_id from departments where department_name='Executive');
```

What can be said about the two statements? (Choose two correct answers.)

A. The two statements should generate the same result.
B. The two statements could generate different results.
C. The first statement will always run successfully; the second statement will error if there are two departments with DEPARTMENT_NAME 'Executive.'
D. Both statements will always run successfully, even if there are two departments with DEPARTMENT_NAME 'Executive.'

List the Types of Subqueries

6. What are the distinguishing characteristics of a scalar subquery? (Choose two correct answers.)

A. A scalar subquery returns one row.
B. A scalar subquery returns one column.

C. A scalar subquery cannot be used in the SELECT LIST of the parent query.
D. A scalar subquery cannot be used as a correlated subquery.

7. Which comparison operator cannot be used with multiple-row subqueries? (Choose the best answer.)
A. ALL
B. ANY
C. IN
D. NOT IN
E. All the above can be used.

Write Single-Row and Multiple-Row Subqueries

8. Consider this statement:

```
select last_name, (select count(*) from departments) from employees
where salary = (select salary from employees);
```

What is wrong with it? (Choose the best answer.)
A. Nothing is wrong—the statement should run without error.
B. The statement will fail because the subquery in the SELECT list references a table that is not listed in the FROM clause.
C. The statement will fail if the second query returns more than one row.
D. The statement will run but is extremely inefficient because of the need to run the second subquery once for every row in EMPLOYEES.

9. Which of the following statements are equivalent? (Choose two answers.)
A. select employee_id from employees where salary < all (select salary from employees where department_id=10);
B. select employee_id from employees where salary < (select min(salary) from employees where department_id=10);
C. select employee_id from employees where salary not >= any (select salary from employees where department_id=10);
D. select employee_id from employees e join departments d on e.department_id= d.department_id where e.salary < (select min(salary) from employees) and d.department_id=10;

10. Consider this statement, which is intended to prompt for an employee's name and then find all employees who have the same job as the first employee:

```
select last_name,employee_id from employees where job_id =
(select job_id from employees where last_name = '&Name');
```

What would happen if a value were given for &Name that did not match with any row in EMPLOYEES? (Choose the best answer.)

A. The statement would fail with an error.

B. The statement would return every row in the table.

C. The statement would return no rows.

D. The statement would return all rows where JOB_ID is NULL.

LAB QUESTION

Exercise 8-3 included this query that attempted to find all employees whose salary is higher than that of a nominated employee:

```
select last_name from employees where
salary > (select salary from employees where last_name='Taylor')
order by last_name;
```

The query runs successfully if `last_name` is unique. Two variations were given that will run without error no matter what value is provided.

The first solution was as follows:

```
select last_name from employees where
salary > all (select salary from employees where last_name='Taylor')
order by last_name;
```

The second solution was as follows:

```
select last_name from employees where
salary > (select max(salary) from employees where last_name='Taylor')
order by last_name;
```

There are other queries that will run successfully; construct two other solutions, one using the ANY comparison operator, the other using the MIN aggregation function. Now that you have four solutions, do they all give the same result?

All these "solutions" are in fact just ways of avoiding error. They do not necessarily give the result the user wants, and they may not be consistent. What change needs to be made to give a consistent, unambiguous, result?

SELF TEST ANSWERS

Define Subqueries

1. ☑ **A, B, C, D, E.** Subqueries can be used at all these points.
   ☒ **F.** A subquery cannot be used in the ORDER BY clause of a query.
2. ☑ **D.** Subquery nesting can be done to many levels.
   ☒ **A, B,** and **C. A** and **C** are wrong because subqueries can be nested. **B** is wrong because the number of rows returned is not relevant to nesting subqueries, only to the operators being used.
3. ☑ **A.** The result set of the inner query is needed before the outer query can run.
   ☒ **B, C,** and **D. B** and **C** are not possible because the result of the subquery is needed before the parent query can start. **D** is wrong because the subquery is only run once.
4. ☑ **D.** This is a correlated subquery, which must be run for every row in the table.
   ☒ **A, B,** and **C.** The result of the inner query is dependent on a value from the outer query; it must therefore be run once for every row.

Describe the Types of Problems That the Subqueries Can Solve

5. ☑ **A, D.** The two statements will deliver the same result, and neither will fail if the name is duplicated.
   ☒ **B, C. B** is wrong because the statements are functionally identical, though syntactically different. **C** is wrong because the comparison operator used, IN, can handle a multiple-row subquery.

List the Types of Subqueries

6. ☑ **A, B.** A scalar subquery can be defined as a query that returns a single value.
   ☒ **C, D. C** is wrong because a scalar subquery is the only subquery that can be used in the SELECT LIST. **D** is wrong because scalar subqueries can be correlated.
7. ☑ **E.** ALL, ANY, IN, and NOT IN are the multiple-row comparison operators.
   ☒ **A, B, C, D.** All of these can be used.

Write Single-Row and Multiple-Row Subqueries

8. ☑ **C.** The equality operator requires a single-row subquery, and the second subquery could return several rows.
   ☒ **A, B, D. A** is wrong because the statement will fail in all circumstances except the unlikely case where there is zero or one employees. **B** is wrong because this is not a problem; there need be no relationship between the source of data for the inner and outer queries. **D** is wrong because the subquery will only run once; it is not a correlated subquery.

9. ☑ **A** and **B** are identical.
☒ **C** is logically the same as **A** and **B** but syntactically is not possible; it will give an error. **D** will always return no rows, because it asks for all employees who have a salary lower than all employees. This is not an error but can never return any rows. The filter on DEPARTMENTS is not relevant.

10. ☑ **C.** If a subquery returns NULL, then the comparison will also return NULL, meaning that no rows will be retrieved.
☒ **A, B, D. A** is wrong because this would not cause an error. **B** is wrong because a comparison with NULL will return nothing, not everything. **D** is wrong because a comparison with NULL can never return anything, not even other NULLs.

LAB ANSWER

The following are two possible solutions using ANY and MIN:

```
select last_name from employees where
salary > any (select salary from employees where last_name='Taylor')
order by last_name;

select last_name from employees where
salary not < (select min(salary) from employees where last_name='Taylor')
order by last_name;
```

These are just as valid as the solutions presented earlier that used ALL and MAX, but they do not give the same result. There is no way to say that these are better or worse than the earlier solutions. The problem is that the subquery is based on a column that is not the primary key. It would not be unreasonable to say that all these solutions are wrong, and the original query is the best; it gives a result that is unambiguously correct if the LAST_NAME is unique, and if LAST_NAME is not unique, it throws an error rather than giving a questionable answer. The real answer is that the query should be based on EMPLOYEE_ID, not LAST_NAME.