Objectives

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

• Describe the types of problems that subqueries can solve
• Define subqueries
• List the types of subqueries
• Write single-row and multiple-row subqueries
"Who has a salary greater than Jones'?"

"Which employees have a salary greater than Jones' salary?"

"What is Jones' salary?"
Subqueries

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

- The subquery (inner query) executes once before the main query.
- The result of the subquery is used by the main query (outerquery).

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

- WHERE clause
- HAVING clause
- FROM clause

In the syntax:

`operator` includes a comparison operator such as >, =, or IN

**Note:** Comparison operators fall into two classes: single-row operators

```
( >, =, >=, <, <>, <= )
```

and multiple-row operators ( IN, ANY, ALL ).
Using a Subquery

```sql
SELECT ename
FROM EMP
WHERE sal >
    ( SELECT sal
        FROM emp
        WHERE empno=7566);
```

<table>
<thead>
<tr>
<th>ENAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORD</td>
</tr>
<tr>
<td>SCOTT</td>
</tr>
<tr>
<td>KING</td>
</tr>
<tr>
<td>FORD</td>
</tr>
</tbody>
</table>
Using a Subquery

```
SELECT ename, sal, deptno, job
FROM EMP
WHERE job =
    ( SELECT job
        FROM emp
        WHERE empno=7369);
```

<table>
<thead>
<tr>
<th>ENAME</th>
<th>SAL</th>
<th>DEPTNO</th>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAMS</td>
<td>1100</td>
<td>20</td>
<td>CLERK</td>
</tr>
<tr>
<td>JAMES</td>
<td>950</td>
<td>30</td>
<td>CLERK</td>
</tr>
<tr>
<td>MILLER</td>
<td>1300</td>
<td>10</td>
<td>CLERK</td>
</tr>
<tr>
<td>SMITH</td>
<td>800</td>
<td>20</td>
<td>CLERK</td>
</tr>
<tr>
<td>ADAMS</td>
<td>1100</td>
<td>20</td>
<td>CLERK</td>
</tr>
<tr>
<td>JAMES</td>
<td>950</td>
<td>30</td>
<td>CLERK</td>
</tr>
<tr>
<td>MILLER</td>
<td>1300</td>
<td>10</td>
<td>CLERK</td>
</tr>
</tbody>
</table>

7 rows selected.
SELECT ename, sal, deptno
FROM EMP
WHERE sal IN
    ( SELECT MIN(sal)
        FROM emp
    )
GROUP BY deptno ;

<table>
<thead>
<tr>
<th>ENAME</th>
<th>SAL</th>
<th>DEPTNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAMES</td>
<td>950</td>
<td>30</td>
</tr>
<tr>
<td>SMITH</td>
<td>800</td>
<td>20</td>
</tr>
<tr>
<td>MILLER</td>
<td>1300</td>
<td>10</td>
</tr>
</tbody>
</table>
SELECT empno, ename, job
FROM emp
WHERE sal < ANY
    ( SELECT sal
        FROM emp
        WHERE job = 'CLERK' );
SELECT empno, ename, job
FROM emp
WHERE sal < ANY
    ( SELECT sal
        FROM emp
        WHERE job = 'CLERK'
    )
AND job <> 'CLERK';

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>7521</td>
<td>WARD</td>
<td>SALESMAN</td>
</tr>
<tr>
<td>7654</td>
<td>MARTIN</td>
<td>SALESMAN</td>
</tr>
</tbody>
</table>
SELECT empno, ename, job
FROM emp
WHERE sal > ALL
  ( SELECT sal
    FROM emp
    WHERE job = 'CLERK' )
AND job <> 'CLERK';

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>7499</td>
<td>ALLEN</td>
<td>SALESMAN</td>
</tr>
<tr>
<td>7566</td>
<td>JONES</td>
<td>MANAGER</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE</td>
<td>MANAGER</td>
</tr>
<tr>
<td>7782</td>
<td>CLARK</td>
<td>MANAGER</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>ANALYST</td>
</tr>
<tr>
<td>7839</td>
<td>KING</td>
<td>PRESIDENT</td>
</tr>
<tr>
<td>7844</td>
<td>TURNER</td>
<td>SALESMAN</td>
</tr>
<tr>
<td>7902</td>
<td>FORD</td>
<td>ANALYST</td>
</tr>
</tbody>
</table>

8 rows selected.
```
SELECT empno, ename, job
FROM emp
WHERE sal > ALL
    ( SELECT AVG(sal)
        FROM emp
        GROUP BY deptno ) ;
```

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>7566</td>
<td>JONES</td>
<td>MANAGER</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>ANALYST</td>
</tr>
<tr>
<td>7839</td>
<td>KING</td>
<td>PRESIDENT</td>
</tr>
<tr>
<td>7902</td>
<td>FORD</td>
<td>ANALYST</td>
</tr>
</tbody>
</table>
Guidelines for Using Subqueries

• Enclose subqueries in parentheses.
• Place subqueries on the right side of the comparison operator.
• Do not add an ORDER BY clause to a subquery.
• Use single-row operators with single-row subqueries.
• Use multiple-row operators with multiple-row subqueries.
Types of Subqueries

- **Single-row subquery**
- **Multiple-row subquery**
- **Multiple-column subquery**

**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 rows from the inner SELECT statement

Multiple-column subqueries: Queries that return more than one column from the inner SELECT statement.
Single-Row Subqueries

• Return only one row
• Use single-row comparison operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&lt;&gt; , !=</td>
<td>Not equal to</td>
</tr>
</tbody>
</table>

```
SELECT ename, job
FROM EMP
WHERE job =
    ( SELECT job
      FROM emp
      WHERE empno=7369 ) ;
```

<table>
<thead>
<tr>
<th>ENAME</th>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAMS</td>
<td>CLERK</td>
</tr>
<tr>
<td>JAMES</td>
<td>CLERK</td>
</tr>
<tr>
<td>MILLER</td>
<td>CLERK</td>
</tr>
<tr>
<td>SMITH</td>
<td>CLERK</td>
</tr>
<tr>
<td>ADAMS</td>
<td>CLERK</td>
</tr>
<tr>
<td>JAMES</td>
<td>CLERK</td>
</tr>
<tr>
<td>MILLER</td>
<td>CLERK</td>
</tr>
</tbody>
</table>

7 rows selected.
Executing Single-Row Subqueries
Using Group Functions in a Subquery

```
SELECT ename, sal, deptno
FROM EMP
WHERE sal IN
    ( SELECT MIN(sal)
      FROM emp
      GROUP BY deptno ) ;
```

<table>
<thead>
<tr>
<th>ENAME</th>
<th>SAL</th>
<th>DEPTNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>800</td>
<td>20</td>
</tr>
<tr>
<td>JAMES</td>
<td>950</td>
<td>30</td>
</tr>
<tr>
<td>MILLER</td>
<td>1300</td>
<td>10</td>
</tr>
</tbody>
</table>
HAVING Clause with Subqueries

• The Oracle Server executes subqueries first.
• The Oracle Server returns results into the HAVING clause of the main query.

```sql
SELECT job, AVG(sal)
FROM emp
GROUP BY job
HAVING AVG(sal) =
    ( SELECT MIN(AVG(sal))
        FROM emp
        GROUP BY job );
```

<table>
<thead>
<tr>
<th>JOB</th>
<th>AVG(SAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLERK</td>
<td>1037.5</td>
</tr>
</tbody>
</table>
What Is Wrong With This Statement?

SELECT empno, ename
FROM emp
WHERE sal =
  (SELECT MIN(sal)
   FROM emp
   GROUP BY deptno);

ERROR at line 4:
ORA-01427: single-row subquery returns more than one row
Will This Statement Work?

```
SELECT ename, job 
FROM emp 
WHERE job = 
  ( SELECT job 
    FROM emp 
    WHERE ename = 'SMYTHE' ) ;
```

no rows selected
Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>Equal to any member in the list</td>
</tr>
<tr>
<td>ANY</td>
<td>Compare value to each value returned by the subquery</td>
</tr>
<tr>
<td>ALL</td>
<td>Compare value to every value returned by the subquery</td>
</tr>
</tbody>
</table>

```sql
SELECT ename, sal, deptno
FROM emp
WHERE sal IN (SELECT MIN(sal)
FROM emp
GROUP BY deptno);
```

<table>
<thead>
<tr>
<th>ENAME</th>
<th>SAL</th>
<th>DEPTNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>800</td>
<td>20</td>
</tr>
<tr>
<td>JAMES</td>
<td>950</td>
<td>30</td>
</tr>
<tr>
<td>MILLER</td>
<td>1300</td>
<td>10</td>
</tr>
</tbody>
</table>
Using ANY Operator in Multiple-Row Subqueries

```
SELECT ename, sal, job
FROM emp
WHERE sal < ANY
    ( SELECT sal
      FROM emp
      WHERE job = 'CLERK'
    )
AND
job <> 'CLERK';
```

<table>
<thead>
<tr>
<th>ENAME</th>
<th>SAL</th>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>WARD</td>
<td>1250</td>
<td>SALESMAN</td>
</tr>
<tr>
<td>MARTIN</td>
<td>1250</td>
<td>SALESMAN</td>
</tr>
</tbody>
</table>
Using ALL Operator in Multiple-Row Subqueries

SELECT  ename, sal, job
FROM emp
WHERE sal >  ALL
( SELECT  AVG(sal)
FROM emp
GROUP BY deptno );

<table>
<thead>
<tr>
<th>ENAME</th>
<th>SAL</th>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>JONES</td>
<td>2975</td>
<td>MANAGER</td>
</tr>
<tr>
<td>SCOTT</td>
<td>3000</td>
<td>ANALYST</td>
</tr>
<tr>
<td>KING</td>
<td>5000</td>
<td>PRESIDENT</td>
</tr>
<tr>
<td>FORD</td>
<td>3000</td>
<td>ANALYST</td>
</tr>
</tbody>
</table>
Summary

Subqueries are useful when a query is based on unknown values.
SELECT select_list
FROM table
WHERE expr

(SELECT select_list
FROM table
WHERE expr);

SELECT select_list
FROM table
WHERE expr

(SELECT select_list
FROM table
WHERE expr);

SELECT select_list
FROM table
WHERE expr
Practice Overview

Creating subqueries to query values based on unknown criteria
Using subqueries to find out what values exist in one set of data and not in another

1. Write a query to display the employee name and hiredate for all employees in the same department as Blake. Exclude Blake.

   ```sql
   SELECT ename, hiredate
   FROM emp
   WHERE deptno =
       ( SELECT deptno
         FROM emp
         WHERE ename = 'BLAKE')
   AND ename <> 'BLAKE';
   ```

2. Create a query to display the employee number and name for all employees who earn more than the average salary. Sort the results in descending order of salary.

   ```sql
   SELECT empno, ename
   FROM emp
   WHERE sal >
       ( SELECT AVG(sal)
         FROM emp );
   ```

3. Write a query to display the employee number and name for all employees who work in a department with any employee whose name contains a T. Save your SQL statement in a file called p6q3.sql.

   ```sql
   SELECT empno, ename
   FROM emp
   WHERE deptno IN
       ( SELECT deptno
         FROM emp
         WHERE ename LIKE '%T%' );
   ```
4. Display the employee name, department number, and job title for all employees whose department location is Dallas.

Solution with subquery:

```
SELECT ename, empno, job
FROM emp
WHERE deptno = (SELECT deptno
                FROM dept
                WHERE loc = 'DALLAS') ;
```

Solution with equijoin:

```
SELECT ename, empno, job
FROM emp e, dept d
WHERE e.deptno = d.deptno
AND d.loc = 'DALLAS';
```

<table>
<thead>
<tr>
<th>ENAME</th>
<th>EMPNO</th>
<th>JOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>7369</td>
<td>CLERK</td>
</tr>
<tr>
<td>JONES</td>
<td>7566</td>
<td>MANAGER</td>
</tr>
<tr>
<td>SCOTT</td>
<td>7788</td>
<td>ANALYST</td>
</tr>
<tr>
<td>ADAMS</td>
<td>7876</td>
<td>CLERK</td>
</tr>
<tr>
<td>FORD</td>
<td>7902</td>
<td>ANALYST</td>
</tr>
</tbody>
</table>

5. Display the employee name and salary of all employees who report to King.

Solution with self join:

```
SELECT e.ename, e.sal
FROM emp e, emp d
WHERE e.mgr = d.empno
AND d.ename = 'KING';
```

Solution with subquery:
SELECT ename, sal
FROM emp
WHERE mgr = (SELECT empno
FROM emp
WHERE ename = 'KING');

6. Display the department number, name, and job for all employees in the Sales department.

SELECT e.deptno, e.ename, e.job, d.dname
FROM emp e, dept d
WHERE e.deptno = d.deptno
AND
d.dname = 'SALES'

If you have time, complete the following exercises:

7. Modify p6q3.sql to display the employee number, name, and salary for all employees who earn more than the average salary and who work in a department with any employee with a T in their name. Rerun your query.

SELECT empno, ename, sal
FROM emp
WHERE sal > (SELECT AVG(sal)
FROM emp)
AND
depthno IN (SELECT deptno
FROM emp
WHERE ename LIKE '%T%');

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>7902</td>
<td>FORD</td>
<td>3000</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>3000</td>
</tr>
<tr>
<td>7566</td>
<td>JONES</td>
<td>2975</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE</td>
<td>2850</td>
</tr>
</tbody>
</table>
Exists

In the previous section, we used IN to link the inner query and the outer query in a subquery statement. IN is not the only way to do so -- one can use many operators such as >, <, or =. EXISTS is a special operator that we will discuss in this section.

EXISTS simply tests whether the inner query returns any row. If it does, then the outer query proceeds. If not, the outer query does not execute, and the entire SQL statement returns nothing.

The syntax for EXISTS is:

```
SELECT "column_name1"
FROM "table_name1"
WHERE EXISTS
  ( SELECT *
      FROM "table_name2"
      WHERE [Condition] )
```
SELECT ename, deptno
FROM emp
WHERE EXISTS
    (SELECT *
     FROM emp
     WHERE sal >3500 )

<table>
<thead>
<tr>
<th>ENAME</th>
<th>DEPTNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>20</td>
</tr>
<tr>
<td>ALLEN</td>
<td>30</td>
</tr>
<tr>
<td>WARD</td>
<td>30</td>
</tr>
<tr>
<td>JONES</td>
<td>20</td>
</tr>
<tr>
<td>MARTIN</td>
<td>30</td>
</tr>
<tr>
<td>BLAKE</td>
<td>30</td>
</tr>
<tr>
<td>CLARK</td>
<td>10</td>
</tr>
<tr>
<td>SCOTT</td>
<td>20</td>
</tr>
<tr>
<td>KING</td>
<td>10</td>
</tr>
<tr>
<td>TURNER</td>
<td>30</td>
</tr>
<tr>
<td>ADAMS</td>
<td>20</td>
</tr>
<tr>
<td>JAMES</td>
<td>30</td>
</tr>
<tr>
<td>FORD</td>
<td>20</td>
</tr>
<tr>
<td>MILLER</td>
<td>10</td>
</tr>
</tbody>
</table>

14 rows returned in 0.01 seconds

SELECT ename, deptno
FROM emp
WHERE EXISTS
    (SELECT *
     FROM emp
     WHERE sal >3500 )
	no data found
WHERE clause

Most often, the subquery will be found in the WHERE clause. These subqueries are also called nested subqueries.

```sql
select *
from all_tables tabs
where tabs.table_name in
     (select cols.table_name
      from all_tab_columns cols
      where cols.column_name = 'ENAME');
```

Limitations:

Oracle allows up to 255 levels of subqueries in the WHERE clause.
FROM clause

A subquery can also be found in the FROM clause. These are called inline views.

For example:

```sql
select dept.deptno, subquery1.total_amt
from dept,
     ( select emp.deptno, Sum(emp.sal) total_amt
           from emp
           group by deptno) subquery1
WHERE subquery1.deptno = dept.deptno
```

<table>
<thead>
<tr>
<th>DEPTNO</th>
<th>TOTAL_AMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>9400</td>
</tr>
<tr>
<td>10</td>
<td>8750</td>
</tr>
<tr>
<td>20</td>
<td>10875</td>
</tr>
</tbody>
</table>

3 rows returned in 0,02 seconds

In this example, we've created a subquery in the FROM clause as follows:

```sql
( select emp.deptno, Sum(emp.sal) total_amt
     from emp
     group by deptno) subquery1
```

This subquery has been aliased with the name `subquery1`. This will be the name used to reference this subquery or any of its fields.

**Limitations:**

Oracle allows an unlimited number of subqueries in the FROM clause.
SELECT clause

A subquery can also be found in the SELECT clause.

For example:

```sql
select tbls.owner, tbls.table_name,
(select count(column_name) as total_columns
 from all_tab_columns cols
 where cols.owner = tbls.owner
 and cols.table_name = tbls.table_name) subquery2
from all_tables tbls;
```

<table>
<thead>
<tr>
<th>OWNER</th>
<th>TABLE_NAME</th>
<th>SUBQUERY2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS</td>
<td>DUAL</td>
<td>1</td>
</tr>
<tr>
<td>SYS</td>
<td>SYSTEM_PRIVILEGE_MAP</td>
<td>3</td>
</tr>
<tr>
<td>SYS</td>
<td>TABLE_PRIVILEGE_MAP</td>
<td>2</td>
</tr>
<tr>
<td>SYS</td>
<td>_STMT_AUDIT_OPTION_MAP</td>
<td>3</td>
</tr>
<tr>
<td>SYS</td>
<td>AUDIT_ACTIONS</td>
<td>2</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>DEF$_TEMP$LOB</td>
<td>3</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>HELP</td>
<td>3</td>
</tr>
<tr>
<td>CTXSYS</td>
<td>DR$OBJECT_ATTRIBUTE</td>
<td>14</td>
</tr>
<tr>
<td>CTXSYS</td>
<td>DR$POLICY_TAB</td>
<td>2</td>
</tr>
<tr>
<td>CTXSYS</td>
<td>DR$NUMBER_SEQUENCE</td>
<td>1</td>
</tr>
<tr>
<td>MDSYS</td>
<td>OGIS_SPATIAL_REFERENCE_SYSTEMS</td>
<td>5</td>
</tr>
<tr>
<td>MDSYS</td>
<td>OGIS_GEOMETRY_COLUMNS</td>
<td>10</td>
</tr>
<tr>
<td>MDSYS</td>
<td>SDO_UNITS_OF_MEASURE</td>
<td>12</td>
</tr>
<tr>
<td>MDSYS</td>
<td>SDO_Prime_Meridians</td>
<td>6</td>
</tr>
<tr>
<td>MDSYS</td>
<td>SDO_ELLIPSOIDS</td>
<td>10</td>
</tr>
</tbody>
</table>

More than 15 rows available. Increase rows selector to view more rows.

15 rows returned in 0,15 seconds   CSV Export

In this example, we've created a subquery in the SELECT clause as follows:

```sql
(select count(column_name) as total_columns
 from all_tab_columns cols
 where cols.owner = tbls.owner
 and cols.table_name = tbls.table_name) subquery2
```
The subquery has been aliased with the name subquery2. This will be the name used to reference this subquery or any of its fields.

The trick to placing a subquery in the select clause is that the subquery must return a single value. This is why an aggregate function such as **SUM**, **COUNT**, **MIN**, or **MAX** is commonly used in the subquery.
Stepping through Sub-Queries in Oracle
(Page 1 of 5)

This is the first article in a series concentrating on working with sub-queries in Oracle. Sub-queries really have tremendous depth. In this series I will show you several scenarios where they can be efficiently used to retrieve information from Oracle databases.

As this is the first in series, I will be concentrating on the following topics:

- How to work with a “dual” table.
- How to analyze and identify the steps needed to deal with a sub-query.
- How to frame queries for each of the identified steps.
- How to combine all the framed queries and design a single command to retrieve the final output.

A primer on the “dual” table in Oracle

This section mainly explains the “dual” table in Oracle. I shall use this table in a creative manner wherever required in this article as well as upcoming articles. If you are already familiar with the "dual” table, feel free to skip to the next section.

What is a “dual” table? It is a simple table which is created/installed automatically during the installation of the Oracle database. To understand it, let us consider the following SELECT statement:

```
SELECT 123 FROM dual;
```

The above statement simply returns 123. Let us work with another statement:

```
SQL> SELECT 10,20,'Veri Tabanı Yönetim Sistemleri',3400 FROM dual;
```

<table>
<thead>
<tr>
<th>10</th>
<th>20</th>
<th>'VERITABANI YÖNETIMSISTEMLERI'</th>
<th>3400</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20</td>
<td>Veri Tabanı Yönetim Sistemleri</td>
<td>3400</td>
</tr>
</tbody>
</table>

1 rows returned in 0,00 seconds

This returns any constant values you provide as columns. So “dual” is just a convenience table. It is simply a one column and one row table that exists as part of SYS user. You can use the DESC command to display the structure of a “dual” table as follows:
DESC dual;

The above statement returns the following output:

<table>
<thead>
<tr>
<th>Table</th>
<th>Column</th>
<th>Data Type</th>
<th>Length</th>
<th>Precision</th>
<th>Scale</th>
<th>Primary Key</th>
<th>Nullable</th>
<th>Default</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUAL</td>
<td>DUMMY</td>
<td>Varchar2</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

You can observe that there exists only one column named “dummy” in the "dual" table. Similarly, you can even display all the rows in the "dual" table as follows:

SELECT * FROM dual;

DUMMY

-----

X

1 rows selected

From the above, you can observe that there exists only one row with a dummy value of "x."

You don’t really need to use the “dual” table at all. It is only used when you want to add/include any constant values in your queries. You can even do calculations as follows:

SQL> SELECT 12 * 13 + 14 FROM dual;

12*13+14

----------------------

170

1 rows selected
Before explaining sub-queries, let us consider the following scenario:

SQL> SELECT empno, ename, sal, deptno FROM emp
WHERE sal = 5000;

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>SAL</th>
<th>DEPTNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>7839</td>
<td>KING</td>
<td>5000</td>
<td>10</td>
</tr>
</tbody>
</table>

1 rows selected

Let us modify the above statement as follows:

SQL> SELECT empno, ename, sal, deptno FROM emp
WHERE sal = (SELECT 5000 FROM dual);

I already explained the “dual” table in the previous section. In the above statement, I have two SELECT statements as part of a single command. The following is the order in which the above statement gets executed:

- The innermost query gets executed first.
- In this case, the query “select 5000 from dual” gets executed first.
- Once the innermost query gets executed, it returns a value to the immediate outer query. In this case, it is 5000.
- The entire innermost query gets replaced with the new value returned by it. In this case, the outer query virtually becomes “select empno, ename, sal, deptno from emp where sal = 5000.”
- And finally, the outer query gets executed, which retrieves KING’s details.

In the above case, I used a SELECT query as part of another SELECT query; thus it is called a “sub-query.” You can even modify the above statement to include an expression as follows:

SQL> SELECT empno, ename, sal, deptno FROM emp
WHERE sal = (SELECT 2000+3000 FROM dual);

The above statement first evaluates “2000+3000” (which results in 5000) as part of executing the inner query. Based on the returned value (which is 5000), the outer query gets executed.

The next section will show you a few simple and practically used sub-queries.
Stepping through Sub-Queries in Oracle - A sub-query with aggregate functions (or group functions) in Oracle

I already introduced sub-queries in the previous section. In this section, I shall start giving you some practical examples.

Let us consider that I would like to retrieve the details of the highest paid employee. Let us write the question in a meaningful manner and identify the steps as follows:

1. Find the highest salary from the table.
2. Based on the value you get, retrieve the employee details like empno, ename, etc. belonging to that salary.

The following is the statement which retrieves the highest salary:

```
SELECT MAX(sal) FROM emp
```

To retrieve an employee's details based on a given salary (say 5000), the query would be as follows:

```
SELECT empno,ename,sal,deptno FROM emp
WHERE sal = 5000
```

Just replace the value 5000 with the query that gives you the highest salary. The complete statement would be as follows:

```
SQL> SELECT empno,ename,sal,deptno FROM emp
WHERE sal = (SELECT MAX(sal) FROM emp);
```

Now, let us walk through its execution:

- The innermost query gets executed first.
- In this case, the query “select max(sal) from emp” gets executed first. It retrieves the highest value in the column “sal” from the table “emp.”
- Once the innermost query gets executed, it returns a value to the immediate outer query. In this case, it is 5000.
- The entire innermost query gets replaced with the new value returned by it. In this case, the outer query virtually becomes “select empno, ename, sal, deptno from emp where sal = 5000.”
• And finally, the outer query gets executed, which retrieves KING’s details.

Let us end this section with a final touch. Why can’t I write the above query as follows?

```sql
SQL> SELECT empno,ename,sal,deptno FROM emp
WHERE sal = MAX(sal)
```

Or even the following:

```sql
SQL> SELECT empno,ename,sal,deptno FROM emp
WHERE MAX(sal) = sal
```

None of the above two queries get executed successfully. The reason is that a condition in a `WHERE` clause cannot contain any aggregate function (or group function) without a sub-query!
Stepping through Sub-Queries in Oracle - Designing sub-queries to deal with more than one table (or different tables)

Let us consider that I would like to retrieve KING's department name. All department names are available in the table “dept,” which is quite different from “emp” table. Let us write the question in a meaningful manner and identify the steps as follows:

From the above figure, you have two steps to go through with the query. The following is the order you must follow (based on the above figure):

- Find KING's department number (1).
- Based on the value you get, retrieve the department details like dname, loc. etc belonging to that department number (2)

The following is the statement which retrieves KING's department number:

```
SELECT deptno FROM emp WHERE ename = 'KING'
```

To retrieve department details based on a given department number (say 30), the query would be as follows:

```
SELECT dname FROM dept
WHERE deptno = 30
```

Just replace the value 30 with the query that gives you KING's department number. The complete statement would be as follows:

```
SQL> SELECT dname FROM dept
WHERE deptno = (SELECT deptno FROM emp WHERE ename='KING');
```

Now, let us walk through its execution:

- The innermost query gets executed first.
- In this case, the query “select deptno from emp where ename='king’” gets executed first. It retrieves KING's department number.
- Once the innermost query gets executed, it returns a value to the immediate outer query. In this case, it is 10.
- The entire innermost query gets replaced with the new value returned by it. In this case, the outer-query virtually becomes “select dname from dept where deptno = 10.”
- And finally, the outer query gets executed, which retrieves KING’s department details
Stepping through Sub-Queries in Oracle - An example of a nested sub-query (or multi-level sub-query)

Let us consider that I would like to retrieve the department name of the highest paid employee. Let us write the question in a meaningful manner and identify the steps as follows:

1. Find the highest salary.
2. Based on the value you get, retrieve the department number belonging to that salary.
3. Based on the value you get, find the department name belonging to that department number.

The following is the statement which finds the highest salary:

```
SELECT MAX(sal) FROM emp
```

To retrieve a department number based on a given salary (say 2000), the query would be as follows:

```
SELECT deptno FROM emp
WHERE sal = 2000
```

To retrieve a department name based on a given department number (say 20), the query would be as follows:

```
SELECT dname FROM dept
WHERE deptno = 20
```

Combining all of the above queries according to the order given above, the new query would be as follows:

```
SQL> SELECT dname FROM dept
WHERE deptno = (SELECT deptno FROM emp
               WHERE sal = (SELECT MAX(sal) FROM EMP));
```

You can observe the following figure to understand how the execution takes place. You can also observe the underlined columns on how they relate logically:
Now, let us walk through its execution:

- The innermost query gets executed first.
- In this case, the query “select max(sal) from emp” gets executed first. It retrieves the highest salary. In this case it would be 5000.
- The entire innermost query gets replaced with the new value returned by it. In this case, the immediate outer query virtually becomes “select deptno from emp where sal = 5000.” Once this query gets executed, it returns a department number, which is 10 in this case.
- And finally, the outermost query virtually becomes “select dname from dept where deptno = 10,” which retrieves KING’s department details.

Any bugs, doubts, suggestions, feedback etc. are highly appreciated at http://jagchat.spaces.live.com
Sub-Queries with multiple columns in Oracle

Let us try to work with the following query:

```sql
SELECT *
FROM emp
WHERE (sal,mgr) = (3000,7566)
```

ORA-00920: invalid relational operator

The above would never work and results in giving you the “invalid relational operator” error. This is because you can compare only one value at a time and not more than one. The following is a small trick to overcome the problem:

```sql
SELECT *
FROM emp
WHERE (sal,mgr) = (SELECT 3000,7566 FROM dual)
```

2 rows returned in 0,00 seconds

I just made both of those two values part of the sub-query and it works fine! Both of those values are not from any table, so I used “dual.” Now you can observe the power of the “dual” table. If you want to learn more about “dual,” please refer to my first article in this series.

Let us work with a more practical sub-query rather than with a “dual” table as follows:

```sql
SELECT *
FROM emp
WHERE (sal,mgr) =
(SELECT sal,mgr FROM emp
WHERE sal = (SELECT MIN(sal) FROM EMP
WHERE sal > (SELECT MIN(sal) FROM emp)))
```

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
<th>MGR</th>
<th>HIREDATE</th>
<th>SAL</th>
<th>COMM</th>
<th>DEPTNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>ANALYST</td>
<td>7566</td>
<td>09/12/1982</td>
<td>3000</td>
<td>-</td>
<td>20</td>
</tr>
<tr>
<td>7902</td>
<td>FORD</td>
<td>ANALYST</td>
<td>7566</td>
<td>03/12/1981</td>
<td>3000</td>
<td>-</td>
<td>20</td>
</tr>
</tbody>
</table>

2 rows returned in 0,00 seconds  CSV Export
<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
<th>MGR</th>
<th>HIREDATE</th>
<th>SAL</th>
<th>COMM</th>
<th>DEPTNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>7900</td>
<td>JAMES</td>
<td>CLERK</td>
<td>7698</td>
<td>03/12/1981</td>
<td>950</td>
<td>-</td>
<td>30</td>
</tr>
</tbody>
</table>

1 rows returned in 0,00 seconds  

CSV Export

The above returns all the employees who are earning the same salary and working under the same manager of the employee earning the second least salary!
Inserting Sub-Queries in SELECT Statements in Oracle - Sub-Queries returning single and multiple values in Oracle

Let us consider the following statement:

```sql
SELECT *
FROM emp
WHERE sal = 800
```

When you execute the above statement, the condition in the WHERE clause works with only a single value. Let us rewrite the above statement with a plain sub-query as follows:

```sql
SELECT *
FROM emp
WHERE sal = (SELECT MIN(sal) FROM emp)
```

From the above statement you can understand that the sub-query returns one and only one value (or single value) and the query works fine.

If we would like to work with more than one value in a condition (of equality), you may need to work something like the following:

```sql
SELECT *
FROM emp
WHERE empno IN (7902,7788)
```

Let us frame the above with a sub-query as follows:

```sql
SELECT *
FROM emp
WHERE empno IN (SELECT empno FROM emp
                  WHERE sal = (SELECT MAX(sal) FROM EMP)
```
WHERE sal < (SELECT MAX(sal) FROM emp))

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
<th>MGR</th>
<th>HIREDATE</th>
<th>SAL</th>
<th>COMM</th>
<th>DEPTNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>ANALYST</td>
<td>7566</td>
<td>09/12/1982</td>
<td>3000</td>
<td>-</td>
<td>20</td>
</tr>
<tr>
<td>7902</td>
<td>FORD</td>
<td>ANALYST</td>
<td>7566</td>
<td>03/12/1981</td>
<td>3000</td>
<td>-</td>
<td>20</td>
</tr>
</tbody>
</table>

2 rows returned in 0.03 seconds

The above query gives us all the employees earning the second highest salary! In the above case, the second level sub-query returns more than one value (or two employee numbers). Therefore, I provided the IN operator as part of the condition. If you replace the IN with “=” in the above query, it will return with an error stating “single row sub-query returns more than one row.” When you receive such an error, try to replace “=” with “IN” wherever appropriate.
BETWEEN is a unique operator in Oracle. Let me show you a small example of this:

```sql
SELECT * 
FROM emp 
WHERE sal BETWEEN 1000 and 3000
```

It is very easy to understand the above query as it returns all the employees who earn salaries between 1000 and 3000 (both inclusive). Let us rewrite the above query as follows:

```sql
SELECT * 
FROM emp 
WHERE sal BETWEEN (SELECT 1000 FROM dual) AND 3000
```

I just replaced 1000 with a simple sub-query using “dual.” If you are new to the “dual” table, please refer to my first article in this series. Let us work a bit more practically as follows:

```sql
SELECT * 
FROM emp 
WHERE sal BETWEEN
    (SELECT MIN(sal) FROM emp) and 2000
```

We can even work a bit differently. Let us go through the following query:

```sql
SELECT Grade 
FROM salgrade 
WHERE
    (SELECT MAX(sal) FROM emp) BETWEEN losal AND hisal
```

The above gives you the salary grade of the highest salary!
Inserting Sub-Queries in SELECT Statements in Oracle - Derived tables (or inline views) with Sub-Queries in Oracle

Can we write sub-queries in (or as part of) the FROM clause? The answer is YES and you can use it very efficiently in some scenarios.

Let us consider the following statement:

```
SELECT empno, ename, sal, sal*12 AS AnnSal
FROM emp
WHERE AnnSal > 30000
```

If I execute the above statement, it will result in an error saying that “AnnSal” is an “Invalid Identifier.” The reason is that “AnnSal” is not a column existing in the table “emp.” It is simply an alias (or some reasonable name). We are not allowed to work with a column alias in any of the conditions present in WHERE clause.

Let us modify the above statement to make it work. Try the following now:

```
SELECT empno, ename, sal, AnnSal
FROM (  
    SELECT empno, ename, sal, sal*12 AS AnnSal
    FROM emp
    )
WHERE AnnSal > 30000
```

The above statement is totally different from the previous one. Within the above statement, the outer query doesn’t rely on any specific physical table. The output (or result set) of the inner query is considered as a table for the outer query! The inner query is very similar to that of a view which doesn’t have any physical view name, and it gets created and destroyed on the fly.

So, according to the inner query, it retrieves four columns (empno, ename, sal, AnnSal). The outer query can work with all four columns as if they are directly from a physical table.

As you are trying to define/derive your own table of information from an existing physical table, you call it as a derived table (or inline view). Even the derived tables can be nested to any number of levels with further sub-derived tables as part of FROM clauses.
Now let us go through an interesting topic on CASE structure. Let us see how a CASE structure works. Consider the following statement:

```
SELECT empno, ename, 
    CASE job
        WHEN 'SALESMAN' THEN 'SALES'
        WHEN 'MANAGER' THEN 'MGMT'
        ELSE job
    END AS jobfunction,
    sal
FROM emp
```

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOBFUNCTION</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>7369</td>
<td>SMITH</td>
<td>CLERK</td>
<td>800</td>
</tr>
<tr>
<td>7499</td>
<td>ALLEN</td>
<td>SALES</td>
<td>1600</td>
</tr>
<tr>
<td>7521</td>
<td>WARD</td>
<td>SALES</td>
<td>1250</td>
</tr>
<tr>
<td>7566</td>
<td>JONES</td>
<td>MGMT</td>
<td>2975</td>
</tr>
<tr>
<td>7654</td>
<td>MARTIN</td>
<td>SALES</td>
<td>1250</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE</td>
<td>MGMT</td>
<td>2850</td>
</tr>
<tr>
<td>7782</td>
<td>CLARK</td>
<td>MGMT</td>
<td>2450</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>ANALYST</td>
<td>3000</td>
</tr>
<tr>
<td>7839</td>
<td>KING</td>
<td>PRESIDENT</td>
<td>5000</td>
</tr>
<tr>
<td>7844</td>
<td>TURNER</td>
<td>SALES</td>
<td>1500</td>
</tr>
<tr>
<td>7876</td>
<td>ADAMS</td>
<td>CLERK</td>
<td>1100</td>
</tr>
<tr>
<td>7900</td>
<td>JAMES</td>
<td>CLERK</td>
<td>950</td>
</tr>
<tr>
<td>7902</td>
<td>FORD</td>
<td>ANALYST</td>
<td>3000</td>
</tr>
<tr>
<td>7934</td>
<td>MILLER</td>
<td>CLERK</td>
<td>1300</td>
</tr>
</tbody>
</table>

When the above query is executed, it returns four columns (empno, ename, jobfunction, sal). The only eye-catching issue from the above is the following structure:

```
CASE job
    WHEN 'SALESMAN' THEN 'SALES'
    WHEN 'MANAGER' THEN 'MGMT'
    ELSE job
END AS jobfunction
```
The above dictates something very similar to the following:

- If the value of “job” is “salesman” return “sales.”
- If the above condition fails and if the value of “job” is “manager” return “mgmt.”
- If both of the above conditions fail then return the same value of “job.”
- All the values must be returned in a new column with the heading “jobfunction.”

You need to observe that I specified the column (job) along with CASE. The conditions of WHEN work with the values available within that column. We can even work with different relational (and SQL) operators within the WHEN condition as shown in the following example:

```sql
SELECT
    empno,
    ename,
    CASE
        WHEN comm IS NULL OR comm=0 THEN '-NA-
        ELSE TO_CHAR(comm)
    END AS comm,
    sal
FROM emp
```

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>COMM</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>7369</td>
<td>SMITH</td>
<td>-NA-</td>
<td>800</td>
</tr>
<tr>
<td>7499</td>
<td>ALLEN</td>
<td>300</td>
<td>1600</td>
</tr>
<tr>
<td>7521</td>
<td>WARD</td>
<td>500</td>
<td>1250</td>
</tr>
<tr>
<td>7566</td>
<td>JONES</td>
<td>-NA-</td>
<td>2975</td>
</tr>
<tr>
<td>7654</td>
<td>MARTIN</td>
<td>1400</td>
<td>1250</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE</td>
<td>-NA-</td>
<td>2850</td>
</tr>
<tr>
<td>7782</td>
<td>CLARK</td>
<td>-NA-</td>
<td>2450</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>-NA-</td>
<td>3000</td>
</tr>
<tr>
<td>7839</td>
<td>KING</td>
<td>-NA-</td>
<td>5000</td>
</tr>
<tr>
<td>7844</td>
<td>TURNER</td>
<td>-NA-</td>
<td>1500</td>
</tr>
<tr>
<td>7876</td>
<td>ADAMS</td>
<td>-NA-</td>
<td>1100</td>
</tr>
<tr>
<td>7900</td>
<td>JAMES</td>
<td>-NA-</td>
<td>950</td>
</tr>
<tr>
<td>7902</td>
<td>FORD</td>
<td>-NA-</td>
<td>3000</td>
</tr>
<tr>
<td>7934</td>
<td>MILLER</td>
<td>-NA-</td>
<td>1300</td>
</tr>
</tbody>
</table>

14 rows returned in 0,01 seconds  

In the above case, the conditions are directly used within the WHEN statement and you need not specify any column with the CASE.

Finally, you can even work with sub-queries within the CASE structure as follows:
SELECT
    empno,
    ename,
    CASE
        WHEN sal >= (SELECT avg(sal) FROM emp) THEN 'HIGH'
        ELSE 'LOW'
    END AS pay,
    sal
FROM emp

The above returns a new column named “pay,” which contains either “HIGH” or “LOW” based on their salary compared to the average salary of all employees.
Inserting Sub-Queries in SELECT Statements in Oracle - Sub-Queries as (or part of) columns in Oracle SELECT statements

Before getting into sub-queries as part of columns, let us look at the following small query:

```sql
SELECT
    MAX(sal) AS highest,
    MIN(sal) AS least,
    COUNT(*) AS employees,
    SUM(sal) AS total
FROM emp
```

Everyone can easily understand that the above query returns only one row containing four values of aggregation. Let us rewrite the above query using sub-queries to get the same output.

```sql
SELECT
    (SELECT MAX(sal) FROM emp) AS highest,
    (SELECT MIN(sal) FROM emp) AS least,
    (SELECT COUNT(*) FROM emp) AS employees,
    (SELECT SUM(sal) FROM emp) AS total
FROM dual
```

You can observe that I replaced all aggregate functions with sub-queries! Another important issue to concentrate on in the above query is the “dual” table. As the sub-queries in the above statement are working individually by themselves, I need not work with any table and thus I used the “dual” table. If you want to learn more about the “dual” table, please go through my first article in this same series.

Now, let us look at an interesting query which deals with sub-queries at both the column level and the CASE level. The following is the query:
SELECT
    empno,
    ename,
    sal AS salary,
    ROUND((sal - (SELECT AVG(sal) FROM emp)), 2) AS avgcompare,
    CASE
        WHEN sal >= (SELECT AVG(sal) FROM emp) THEN 'HIGH'
        ELSE 'LOW'
    END AS paying
FROM emp

The following is the sample output of the above query:

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>SALARY</th>
<th>AVGCOMPARE</th>
<th>PAYING</th>
</tr>
</thead>
<tbody>
<tr>
<td>7839</td>
<td>KING</td>
<td>5000</td>
<td>2926.79</td>
<td>HIGH</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE</td>
<td>2850</td>
<td>776.79</td>
<td>HIGH</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7654</td>
<td>MARTIN</td>
<td>1250</td>
<td>-823.21</td>
<td>LOW</td>
</tr>
<tr>
<td>7499</td>
<td>ALLEN</td>
<td>1600</td>
<td>-473.21</td>
<td>LOW</td>
</tr>
</tbody>
</table>

Any bugs, doubts, suggestions, feedback etc. are highly appreciated at http://jagchat.spaces.live.com