MULTIPLE-COLUMN SUBQUERIES
-- CHAPTER 7 --

- Multiple –Column Subqueries: So we have written single-row subqueries where only one column was compared in the WHERE clause or HAVING clauses of the SELECT statement. If you want to compare two or more columns, you must write a compound WHERE clause using logical operators.
- Let’s take a look at the syntax:

```
SELECT column, column, ...
FROM table
WHERE (column, column, ...) IN
    (SELECT column, column, ...
     FROM table
     WHERE someCondition);
```

- Check out the following multiple-column query with subquery…

```
SQL> r
1  select empno, ename, job, sal, deptno
2  from emp
3  where (sal, deptno) in
4    (select sal, deptno
5     from emp
6     where sal between 1000 and 2000 and
7        deptno between 20 and 40)
8* order by empno
```

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
<th>SAL</th>
<th>DEPTNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>7369</td>
<td>SMITH</td>
<td>ANALYST</td>
<td>1500</td>
<td>20</td>
</tr>
<tr>
<td>7499</td>
<td>ALLEN</td>
<td>SALESMAN</td>
<td>1000</td>
<td>30</td>
</tr>
<tr>
<td>7521</td>
<td>WARD</td>
<td>SALESMAN</td>
<td>1000</td>
<td>30</td>
</tr>
<tr>
<td>7566</td>
<td>JONES</td>
<td>MANAGER</td>
<td>2000</td>
<td>20</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE</td>
<td>MANAGER</td>
<td>2000</td>
<td>30</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>ANALYST</td>
<td>2000</td>
<td>20</td>
</tr>
<tr>
<td>7844</td>
<td>TURNER</td>
<td>SALESMAN</td>
<td>1000</td>
<td>30</td>
</tr>
<tr>
<td>7876</td>
<td>ADAMS</td>
<td>ANALYST</td>
<td>1500</td>
<td>20</td>
</tr>
</tbody>
</table>

- In this case, we selected empno, ename, job, sal, and deptno from the ‘emp’ table such that all those employees whose salary was between 1000 and 2000 and their deptno was between 20 and 40. A glance at the data allows us to verify that all records returned by the query satisfy those conditions. Note also that the records are ordered by the empno as dictated by the last line of the subquery.
- In the following example, we add a couple of more lines to the previous query to further restrict the search, and thereby eliminate the record s for SMITH and JONES…

```
SQL> r
1  select empno, ename, job, sal, deptno
2  from emp
3  where (sal, deptno) in
4    (select sal, deptno
5     from emp
6     where sal between 1000 and 2000 and
7        deptno between 20 and 40)
8  where empno not in
9    (select empno
10   from emp
11   where ename = 'SMITH' or ename = 'JONES')
12* order by empno
```

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>JOB</th>
<th>SAL</th>
<th>DEPTNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>7844</td>
<td>TURNER</td>
<td>SALESMAN</td>
<td>1000</td>
<td>30</td>
</tr>
</tbody>
</table>
Column Comparisons: Pairwise Versus Nonpairwise Comparisons

The above examples are called ‘Pairwise Comparisons’ since in order to meet the criteria, the records being compared had to meet two conditions simultaneously...i.e., their both their sal ‘AND’ their deptno had to be in a certain range. If you want a nonpairwise comparison (a cross product), you must use a WHERE clause with multiple conditions. A candidate row must match the multiple conditions in the WHERE clause but the values are compared individually.

```sql
SQL> r
1  select empno, ename, job, sal, deptno
2  from emp
3  where sal in
4  (select sal
5  from emp
6  where sal between 1000 and 2000)
7  and empno in
8  (select empno
9  from emp
10  where deptno between 20 and 40)
11* order by empno
```
8 rows selected.

- Unfortunately, this is not the best example since it yielded the same results as the pairwise comparison, but theoretically, this is just a coincidence...other times the results may differ between these two types of comparisons.

- **NULL values in a subquery.** When SQL encounters a NULL value in a comparison of some sort, SQL will simply not return any rows as output. All conditions that compare a null value result in a null. So, whenever null values are likely to be part of the resultant set of a subquery, do not use the **NOT IN** operator. As an example, compare the following two queries...

```sql
SQL> select employee.ename
2  from emp employee
3  where employee.empno NOT IN
4  (select manager.mgr
5  from emp manager);

no rows selected
```

Compared to this one...

```sql
SQL> select employee.ename
2  from emp employee
3  where employee.empno IN
4  (select manager.mgr
5  from emp manager);

ENAME
------
JONES
BLAKE
CLARK
SCOTT
KING
REDCORNER

6 rows selected.
```

- Notice that the null value as part of the resultant set of a subquery will not be a problem if you are using the **IN** operator.
- Finally, let’s look at how we can use a subquery in the **FROM** clause...

```sql
SQL> r
```
1) select a.ename, a.sal, a.deptno, b.salavg
2) from emp a, (select deptno, avg(sal) salavg
3) from emp
4) group by deptno) b
5) where a.deptno = b.deptno
6) and a.sal > b.salavg

<table>
<thead>
<tr>
<th>ENAME</th>
<th>SAL</th>
<th>DEPTNO</th>
<th>SALAVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>KING</td>
<td>3000</td>
<td>10</td>
<td>2000</td>
</tr>
<tr>
<td>SCOTT</td>
<td>2000</td>
<td>20</td>
<td>1750</td>
</tr>
<tr>
<td>JONES</td>
<td>2000</td>
<td>20</td>
<td>1750</td>
</tr>
<tr>
<td>BLAKE</td>
<td>2000</td>
<td>30</td>
<td>1250</td>
</tr>
<tr>
<td>SHANK</td>
<td>5500</td>
<td>40</td>
<td>5250</td>
</tr>
<tr>
<td>BOBBY</td>
<td>3000</td>
<td>50</td>
<td>1978.8947</td>
</tr>
<tr>
<td>&quot;HELLO&quot;</td>
<td>2000</td>
<td>50</td>
<td>1978.8947</td>
</tr>
<tr>
<td>BUSH</td>
<td>2000</td>
<td>50</td>
<td>1978.8947</td>
</tr>
<tr>
<td>HOMER</td>
<td>2000</td>
<td>50</td>
<td>1978.8947</td>
</tr>
<tr>
<td>RED</td>
<td>2000</td>
<td>50</td>
<td>1978.8947</td>
</tr>
<tr>
<td>REDCORNER</td>
<td>2000</td>
<td>50</td>
<td>1978.8947</td>
</tr>
<tr>
<td>ALBERT</td>
<td>4000</td>
<td>50</td>
<td>1978.8947</td>
</tr>
<tr>
<td>TESTTIME</td>
<td>2000</td>
<td>50</td>
<td>1978.8947</td>
</tr>
<tr>
<td>NKLGNFGLKB</td>
<td>2000</td>
<td>50</td>
<td>1978.8947</td>
</tr>
<tr>
<td>HOLLYO</td>
<td>2000</td>
<td>50</td>
<td>1978.8947</td>
</tr>
<tr>
<td>GREEN</td>
<td>2000</td>
<td>50</td>
<td>1978.8947</td>
</tr>
<tr>
<td>KWIATKI</td>
<td>2000</td>
<td>50</td>
<td>1978.8947</td>
</tr>
<tr>
<td>FREY</td>
<td>2000</td>
<td>50</td>
<td>1978.8947</td>
</tr>
<tr>
<td>GORDON</td>
<td>2000</td>
<td>50</td>
<td>1978.8947</td>
</tr>
<tr>
<td>DUTT</td>
<td>2000</td>
<td>50</td>
<td>1978.8947</td>
</tr>
</tbody>
</table>

21 rows selected.

- You can use a subquery in the FROM clause of a SELECT statement, which is very similar to how views are used. A subquery in the FROM clause of a SELECT statement defines a data source for that particular SELECT statement, and only that SELECT statement.