# New Subqueries in Oracle8i

Pedro Bizarro

In this article, Pedro Bizarro describes the new subqueries that are allowed in Oracle8i. These new subqueries can be used in places that haven't been allowed before, such as in a SELECT clause, in an expression, or as a function/procedure parameter.

THERE'S been much discussion recently about the many new features of Oracle8i. However, I haven't read anything about the new subqueries. In fact, I didn't even know there were any new subqueries until a friend, who was asking for some SQL advice, wrote a SQL statement. With my pre-Oracle8i perspective, I saw a mistake in the code regarding subqueries. I was very surprised to see that the statement worked fine! And my friend was surprised by my surprise.

After that, I decided to do some research on this new kind of subquery. In short, what I found out is that you can use a subquery anywhere just as long as it returns only one value. In addition to this new kind of subquery, Oracle8i still supports all other previous ones.

## Subquery within a SELECT clause

In versions prior to Oracle8i, you had to use views or subqueries within FROM clauses if you wanted something with employee information and group information like this:

```
ENAME      SAL          AVG_SAL_DEPT
---------- ------------ ------------
ADAMS              1100         2175
ALLEN              1600         1567
BLAKE              2850         1567
CLARK              2450         2917
...                ...          ...
WARD               1250         1567
14 rows selected.
```

Here's the SELECT statement to produce such a result:

```
/* Solution in a database prior to Oracle8i:    */
/*   - Subquery within FROM.                     */
/*   - Join made in outer WHERE clause.          */
/*   - Subquery must be named with an alias.     */
/*   - Subquery may return more than one row.    */
SELECT e.ename,
       e.sal,
       avg_sal_dept
  FROM emp e,
       (SELECT deptno, round(avg(sal)) avg_sal_dept
          FROM emp
         GROUP BY deptno) dept_avg
 WHERE e.deptno = dept_avg.deptno
 ORDER BY ename;
```

Note that the subquery within FROM has to be given a name—dept_avg, in this example—in order to be referenced in the WHERE clause defining the join. Although it's not necessary, I also renamed round(avg(sal)) to avg_sal_dept.

With Oracle8i, there are two extra solutions: analytic functions (which I'm not going to talk about because everybody else is) and subqueries within a SELECT clause:

```
/* Solution in an Oracle8i database:            */
/*   - Subquery within SELECT.                  */
/*   - Join made with correlation.              */
/*   - Subquery doesn't need a name.            */
/*   - Subquery must return exactly one value.  */
SELECT e.ename,
       e.sal,
       (SELECT round(avg(sal))
          FROM emp
         WHERE deptno = e.deptno) avg_sal_dept
  FROM emp e
 ORDER BY ename;
```

Note that since I'm using a group function and correlation, I'm certain that the subquery will return exactly one value. I'm wittingly saying "one value" and not "one row" because this kind of subquery has to return just one column and one row—thus, one value.

### *Error messages*

If the subquery returns more than one column, Oracle replies with "ORA-00913: too many values." If the subquery returns more than one row, Oracle complains with "ORA-01427: single-row subquery returns more than one row."

If the Oracle8i solution is used in a previous version, Oracle responds with "ORA-00936: missing expression" at the beginning of the subquery.

## Subquery within an expression

Since you can use subqueries anywhere, you can use them on expressions as well. Before Oracle8i, you couldn't write the following UPDATE statement:

```
/* UPDATE in an Oracle8i database:              */
UPDATE emp e
   SET sal = 1.1 * (SELECT avg(sal)
                      FROM emp
                     WHERE deptno = e.deptno)
 WHERE some_condition;
```

*Continues on page 18*

```
  CURSOR cur_next_id IS
    SELECT my_table_sequence.NEXTVAL FROM Dual;
BEGIN
  OPEN cur_next_id;
  FETCH cur_next_id INTO :my_table.record_id;
  CLOSE cur_next_id;
END;
```

Remember, PRE-INSERT is a form trigger, not a database trigger. If you add records using something other than forms, you'll have to have similar code. ▲

Steve Miller has written for Pinnacle Publishing for many years, and has 12 years of software development experience. He recently became a member of Wycliffe Bible Translators, and anticipates starting his assignment soon. sdmiller@tir.com.

## Additional Information

For more information, see the following article from *Metalink*: GSX Problem—Solution No: 1015345.6.

# New Subqueries...

*Continued from page 12*

Instead, you had to place the multiplication inside the subquery, like this:

```
/* UPDATE in a database prior to Oracle8i:      */
UPDATE emp e
   SET sal = (SELECT 1.1 * avg(sal)
                FROM emp
               WHERE deptno = e.deptno)
 WHERE some_condition;
```

Although the new code is a bit cleaner, this isn't a fantastic achievement. Let's take a look at something different. For instance, I can multiply, divide, add, or subtract the returns of two SELECTs! Consider this statement (which is very unlikely to be needed, but it's possible):

```
/* For every employee, set his/her salary to    */
/* be 110% of the average between the average   */
/* of all salaries and the average of employees */
/* from his/her department.                     */
/* Executes only in Oracle8i.                   */
UPDATE emp e
   SET sal = 1.1 * ((SELECT avg(sal)   -- average of
                       FROM emp)       -- all emps
                    +
                    (SELECT avg(sal)   -- average of
                       FROM emp        -- dept
                      WHERE deptno = e.deptno))/2;
```

For each record, the first subquery returns the average salary of all employees (group by is the whole set), and the second returns the average salary of all employees from that employee's department (implicit group by deptno because of correlation). I then add the results and then divide by two to get the average of them both. Then I multiply by 1.1 to get a 10 percent raise on that average. Try to do that on a database prior to Oracle8i, and you'll end up with some very confusing code.

### *Error messages*

Strangely enough, if I remove the "1.1 *" factor from the statement, Oracle can't parse it anymore. It seems that the expression can't start with a subquery if I have more than one. Removing the "1.1 *" factor yields a parsing error message like "ORA-00907: missing right parenthesis" or "ORA-00933: SQL command not properly ended." So, what if I don't want to calculate the 110 percent of the average? I can do a workaround using "1.0 *," which doesn't change the result.

### Subquery as a parameter

Another way to use subqueries is as a function or procedure parameter. In Oracle8i, it's possible to write a SQL statement like this:

```
SELECT to_char((SELECT max(hiredate)
                  FROM emp),
               'yyyy-mm-dd') FROM dual;
```

A user-defined function or procedure can also have a subquery as any of its parameters. The only detail to keep in mind is that you must add surrounding parentheses to the subquery.

### Conclusion

This new kind of subquery can be used anywhere, as long as the following conditions are met:

- It returns only one value (one column and one row).
- It's surrounded by parentheses.
- In expressions with more than one subquery, the expression doesn't start with a subquery. ▲

Pedro Bizarro is a graduate student at University Nova de Lisboa and a teaching assistant in all database courses at University of Coimbra, Portugal. He'll be completing work for his master's degree soon. Pedro is already a Fulbright grantee for his Ph.D. program, starting September 2001. bizarro@dei.uc.pt.