Inner, Outer, Full? Oracle9i Join syntax

John King, King Training Resources

Introduction

Joining data together is one of the most significant strengths of a relational database. Oracle was the first commercially available database to support the outer-join construct. The more-recent ISO/ANSI standard includes join semantics that are more comprehensive than those supported by Oracle8i or earlier releases. Oracle9i supports the "new" join including Inner Join as well as Left, Right, and Full Outer Join capability. Until now, the only way to accomplish a Full join (values missing on both sides of a query) was to use UNION. Furthermore, the "new" Join syntax allows the separation of Join criteria from WHERE selection criteria sometimes simplifying code. The use of the ISO/ANSI join syntax means that Oracle SQL will be more easily ported to/from other environments and will fit in more easily with SQL statement generators. In this paper, the "new" syntax will be explained and demonstrated so that developers may improve SQL statement effectiveness.

Understanding Joins

Joins allow database users to combine data from one table with data from one or more other tables (or views, or synonyms). Tables are "joined" two at a time making a new table containing all possible combinations of rows from the original two tables (sometimes called a "full join" or "cartesian product").

A "join condition" is usually used to limit the combinations of table data to just those rows containing columns that match columns in the other table. Most joins are "equi-joins" where the data from a column in one table exactly matches data in the column of another table. It is also possible (though usually less efficient) to join using ranges of values or other comparisons between the tables involved. A table may be "joined" to another table, tables, or itself!

It is important to understand that whenever two or more tables/views/synonyms are listed in a FROM clause a join results. Join conditions serve the purpose of limiting the number of rows returned by the join.

The absence of a join condition results in all possible combinations of rows from the involved tables which is usually not useful information.

Original Join Syntax

The original join semantics in SQL were to include two (or more) table/view/synonym names in a FROM clause and to use the WHERE clause to describe the join condition. This results in combining row selection criteria with join conditions in the WHERE clause sometimes causing confusion for those reading/maintaining/tuning the code.

```
select distinct nvl(dname,'No Dept'),count(empno) nbr_emps
   from many_emps emp,many_depts dept
   where emp.deptno = dept.deptno
     and emp.sal between 2000 and 3000
             and emp.job = 'SALESMAN'
   group by dname;
```

New Join Syntax

The ISO/ANSI Join syntax has been used for several years in some non-Oracle SQL environments. Since Oracle had invented the original Outer-join syntax they were slow to accept the new style. The significant advantage of the ISO/ANSI Join syntax is that it is supported by third party SQL tools. Using the ISO/ANSI syntax also makes it easier to move code to a non-Oracle platform if desired.

Perhaps the single most advantageous feature of the new semantics is the separation of join criteria from other row selection criteria. When using the new syntax, join criteria is added to the FROM clause using an IN or USING clause that is clearly meant for the previous pair of tables/views/synonyms.

Cross Join/Natural Join

Cross Join indicates a join of the same type as when comma-delimited, requiring specification of join conditions in the WHERE clause:

```
select ename,dname
  from emp cross join dept
  where emp.deptno = dept.deptno
```

Natural joins indicate an equi-join where the database automatically chooses join criteria using any column names that happen to match between the two tables. Natural joins may also specify ISO/ANSI join types (INNER, LEFT, RIGHT, FULL; discussed later…). Additional criteria may be specified using the WHERE clause.

```
select ename,dname
  from emp natural join dept
```

LOB columns may not be selected when using NATURAL JOIN.

Inner/Outer Join

Traditional Inner Joins look for rows that match rows in the other table(s). Depending upon the data involved, it is also possible that a user might be interested in rows that DO NOT match rows in the other table(s). Finding rows without matches is often referred to as Outer Join (sometimes Anti-Join). The next section deals specifically with Outer Join criteria.

The older syntax names all tables in comma-delimited form and uses the WHERE clause to name Join criteria, note that in the example below Join criteria is mixed with row selection criteria.

```
select distinct nvl(dname,'No Dept'),count(empno) nbr_emps
   from many_emps emp,many_depts dept
   where emp.deptno = dept.deptno
     and emp.job in ('MANAGER','SALESMAN','ANALYST')
   group by dname;
```

To use ISO/ANSI Inner Join semantics, use the term INNER JOIN (or simply JOIN) between the table(s) involved and specify one-or-more Join criteria with the ON/USING clause. Correlation (alias) table names may be specified in the same manner as with comma-delimited joins. With the new syntax the WHERE clause names only non-Join criteria.

```
select distinct nvl(dname,'No Dept'),count(empno) nbr_emps
  from many_emps emp join many_depts dept
    on emp.deptno = dept.deptno
  where emp.job in ('MANAGER','SALESMAN','ANALYST')
  group by dname;
```

The new Join semantics allow the use of parentheses to control the order in which joins occur.

Left/Right Outer Join

A problem with inner join (old or new syntax) is that only rows that match between tables are returned. Oracle invented the first syntax for solving the outer Join issue years ago. This is the “(+)” notation used on side of the Join criteria WHERE clause where null rows are to be created to match the other table. The two queries below illustrate the problem:

```
SQL> select distinct deptno from many_emps;

    DEPTNO
----------
        10
        20
        30
        40
        60
        70            Departments 60, 70, and 80 are not matched by ANY_DEPTS table rows
        80

SQL> select distinct deptno from many_depts;

    DEPTNO
----------
        10
        20
        30
        40            Departments 50 and 51 are not matched by MANY_EMPS table rows
        50
        51
```

Note that the two queries above show that rows exist in each table that are not matched by the other table. (Clearly, the database above has some problems and needs referential integrity added soon!) A request to list each department name and the number of employees assigned to the department would yield numbers for departments 10, 20, 30, and 40 only. With Oracle’s Outer Join operator the query can be caused to create null MANY_EMPS rows to match rows in the MANY_DEPTS table. The Oracle Outer Join operator “(+)” tells SQL which table to generate null rows for.

```
select distinct nvl(dname,'No Dept'),count(empno) nbr_emps
   from many_emps emp,many_depts dept
   where emp.deptno(+) = dept.deptno
   group by dname;
```

The new ISO/ANSI Join syntax provides three separate capabilities: LEFT, RIGHT, and FULL OUTER JOIN (the word OUTER is redundant and usually omitted). With the new syntax, LEFT and RIGHT indicate which side of the join represents the complete set, the opposite side is where null rows will be created. The example below solves the same problem as the Oracle Outer Join operator example above:

```
select distinct nvl(dname,'No Dept'),count(empno) nbr_emps
  from many_emps emp right join many_depts dept
    on emp.deptno = dept.deptno
   group by dname;
```

Again, RIGHT JOIN in the example above indicates that MANY_DEPTS is the complete set and that null rows are to be temporarily created for MANY_EMPS to match. To create null MANY_DEPT rows to match departments in the MANY_EMP table LEFT JOIN would be used.
Unfortunately, neither the Oracle Outer Join operator nor the new LEFT/RIGHT Outer Join semantics solve the problem highlighted earlier of rows in both tables that do match the opposite table.

Full Outer Join

To cause SQL to generate rows on both sides of the join required a UNION using the old Oracle Outer Join operator syntax as illustrated below:

```
select nvl(dname,'No Dept') deptname,count(empno) nbr_emps
   from many_emps emp,many_depts dept
   where emp.deptno(+) = dept.deptno   ←
   group by dname
union
select nvl(dname,'No Dept') deptname,count(empno) nbr_emps
   from many_emps emp,many_depts dept
   where emp.deptno = dept.deptno(+)
   group by dname;
```

The solution using UNION works fine, unfortunately, it does not conform to the ISO/ANSI standard. Using UNION also means the SQL developers must be sure that the queries involved match properly.
The new ISO/ANSI Outer Join mechanism is simpler to code. To cause rows to be created on either side of a Join as required to align the two tables use the FULL OUTER JOIN (FULL JOIN) syntax.

```
select distinct nvl(dname,'No Dept') deptname,count(empno) nbr_emps
  from many_emps emp full join many_depts dept   ←
    on emp.deptno = dept.deptno
   group by dname;
```

Result of Full Outer Join

```
DEPTNAME         NBR_EMPS
------------- ----------
ACCOUNTING          12503
Join Testing            0
Join Testing 2          0
No Dept                14
OPERATIONS          12499
RESEARCH            12505
SALES               12506
```

Issues and Recommendations

Oracle recommends that you use the new ISO/ANSI Outer Join semantics instead of the Oracle Outer Join operator. Queries using the Oracle Outer Join operator must conform to restrictions that do not apply to the new ISO/ANSI semantics including:
Queries using the Oracle Outer Join operator "(+)" are subject to the following rules and restrictions, which do not apply to the ANSI syntax:

1. Queries may not mix the Oracle Outer Join operator with the new ISO/ANSI semantics
2. If multiple join conditions are present the Oracle Outer Join operator must be specified for every condition
3. The Oracle Outer Join operator may be applied only to a column, not to an expression
4. The Oracle Outer Join operator may not be combined with another condition using the OR operator
5. The Oracle Outer Join operator may not be used wityh an IN condition
6. The Oracle Outer Join operator may only be used for one table in a query

Performance and Known Problem

In the both Oracle9i implementations (version 9.0 and 9.2) it appears that Oracle Inner Joins behave differently using the older comma-delimited syntax than when using the new ISO/ANSI syntax. In at least some cases, the new syntax creates a different plan and sometimes does not perform as well as the earlier methods. When large quantities of data are involved you should test both methods to make sure that one method does not have a distinct advantage/disadvantage over the other.

- In at least on other vendor's implementations (IBM's DB2/UDB), the ISO/ANSI join semantics have been improved over time such that they out-perform the older comma-delimited style.
- Right/Left Outer Joins (in both versions 9.0 and 9.2) on the other hand appear to use exactly the same execution plan for queries using either the older Oracle Outer Join operator or the new Outer Join semantics. Therefore, there should be no performance difference. (Though testing is ALWAYS a good idea…)
- Full Outer Join in version 9.0 appears to generate the same plan as when using the Outer Join Operator and a Union, however, version 9.2 Full Outer Join plans seem to show different and improved plans.
- In the first release of 9i (version 9.0) there seems to be a bug in the implementation of FULL JOIN. In many cases Oracle generates an error when a View is named rather than a Table. Views seem to work fine with LEFT and RIGHT OUTER JOIN semantics, just not with FULL JOIN. This problem was solved in Oracle 9.2.
- In the first release of 9i (version 9.0) there was a significant security bug when using the new Join syntax allowing access to any table in the database regardless of schema and user permissions! A patch is available for all but Windows environments. This bug has been fixed in Oracle 9.2.

Conclusion

In today's increasingly standardized programming world using ISO/ANSI-standard syntax rather than vendor-specific code makes a tremendous amount of good sense. In addition the number of programming, code-generation, development, documentation, and code review tools that recognize the new syntax is growing. For maintainability and portability purposes it is probably a good idea to adopt the new syntax as soon as possible.

In the case of Outer Joins the choice of the new syntax is made easier since performance does not suffer and with 9.2 seems to improve when using Full Outer Join. You should certainly test in your own environment to be sure you have not introduced a problem.

The simpler syntax of the new semantics especially with FULL JOIN is probably worth adopting the new style alone. Oracle's recommendation to use the new syntax hints that the older Outer Join operator may some day appear on the list of deprecated features in a future release.

In the case of Inner Joins, the performance issues cannot be ignored by many larger installations. YOUR MILEAGE MAY VARY. There is no substitute for testing specific queries with both methods to decide if one is clearly superior.

About the Author

John founded King Training Resources in 1988, he has been working with Oracle products since Oracle Version 4 and has been teaching Oracle-related courses since 1986. He has presented papers at various user group conferences including: IOUG-A Live!, EOUG, UKOUG, ODTUG, SEOUC, MAOP, and the RMOUG Training Days.

This paper, the presentation slides, and sample SQL may be downloaded from http://www.kingtraining.com.

References

1. Oracle SQL Reference