Using a SELECTIVITY clause to influence the optimizer

Paul Yip December 04, 2003

This article explains how you can influence the DB2 SQL optimizer by providing the expected selectivity using a selectivity clause.

Acknowledgements:

- John Hornibrook
- Samir Kapoor

Introduction

This article is written for DB2® Universal DatabaseTM version 8.1 for Linux®, UNIX®, and Windows®.

Since I participate actively in the art of Oracle conversions to DB2, I am frequently asked if DB2 supports SQL hints like Oracle. The short answer is no.

Here is a longer answer:

At IBM®, we have a different philosophy regarding SQL optimization. If DB2 UDB does not choose the optimal access plan and the poor access plan is not due to a limitation inherent in the query, we consider it a defect in the product and prefer to fix the problem at the source so that all DB2 users may benefit as well. Hence, you should find that there is less need for hints in DB2 to begin with. For other cases, where limitations inherent in the query make proper access plan selection difficult, you can influence the DB2 SQL optimizer by providing additional selectivity information.

For example, consider the following SQL statement:

```
SELECT * FROM T1 where col1 >= ?
```

If the values of col1 in table T1 ranged from 0 to 100, the value provided through the parameter marker can greatly vary the actual selectivity of the predicate. That is, if the values stored in col1 were evenly distributed between 0 and 100, the number of rows which meet the requirements of the WHERE clause would vary greatly if the parameter marker value were 10 instead of 90.

However, you can influence the optimizer by providing the expected selectivity using a selectivity clause. It can be useful in situations where you want to encourage (or discourage) DB2 to

use certain indexes. Using DFT_QUERYOPT=5 (the default setting) seems to work most predictably with this feature. To enable selectivity clauses, you must first set the registry variable DB2_SELECTIVITY=YES and restart the instance.

```
db2set DB2_SELECTIVITY=YES
     db2 force application all   (to kick off all connected users)
     db2stop
     db2start
```

The SELECTIVITY clause can only be used with basic predicates (as defined in the SQL reference), not predicates such as LIKE or BETWEEN. A lower selectivity value (very small number) will tell DB2 that the predicate will qualify fewer rows (and encourage use of indexes defined on that column). A higher selectivity value (close to 1) will mean the opposite. Example:

```
SELECT c1, c2, c3, FROM T1, T2, T3
     WHERE T1.x = T2.x AND
     T2.y=T3.y AND
     T1.x >= ? selectivity 0.00001 AND
     T2.y gt; ? selectivity 0.5 AND
     T3.z = ? selectivity 0.2 AND
     T3.w = ?
```

Example of SELECTIVITY in action

Notes:

- The EMPLOYEE and DEPARTMENT tables are provided in the SAMPLE database provided with DB2.
- The index DEPTNOIDX on column DEPTNO of table department was created to illustrate this example.

Tip:

Use of parameter markers is allowed in DB2's Visual Explain utility.

Here is the original query:

```
SELECT * FROM EMPLOYEE e, DEPARTMENT d
     WHERE e.workdept = d.deptno
     AND d.deptno = ?
```

Here is the same query modified with different selectivity clauses:

```
SELECT * FROM EMPLOYEE e, DEPARTMENT d
     WHERE e.workdept = d.deptno
     AND d.deptno = ? selectivity 0.9



     SELECT * FROM EMPLOYEE e, DEPARTMENT d
     WHERE e.workdept = d.deptno
     AND d.deptno = ? selectivity 0.25
```

Now, compare the output of visual explain. Notice how we were able to influenced use of index.

Figure 1. With selectivity of 0.9 (lower selectivity)

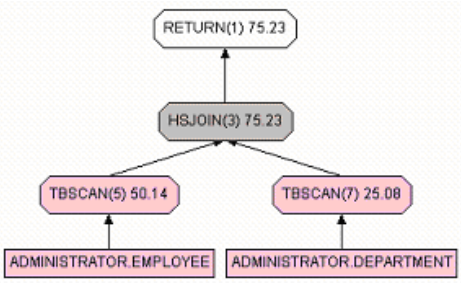
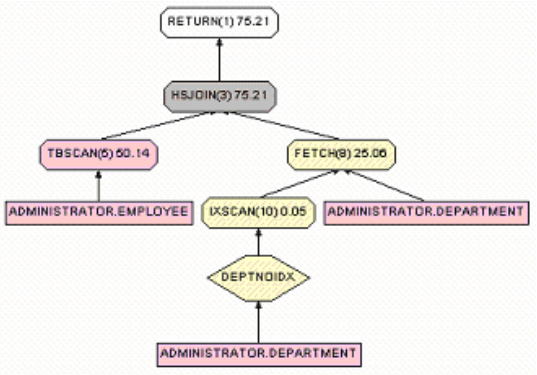


Figure 2. With selectivity of 0.25 (higher selectivity)



Use of the selectivity clause should be considered a last resort. Before using it:

- Experiment with different SQL optimization classes. The default optimization class is controlled by the DFT_QUERYOPT parameter in the database configuration file.
- Attempt to resolve any performance problems by ensuring that proper database statistics have been collected. The more detailed the statistics, the better the optimizer can perform. (See RUNSTATS in the DB2 Command Reference).
- If the poor access plan is the result of rapidly changing characteristics of the table (i.e. grows very quickly such that statistics get out of date quickly), try marking the table as VOLATILE using the ALTER TABLE command.
- Try explaining the query using literal values instead of parameter markers in your predicates. If you are getting different access plans when using parameter markers, it will help you understand the nature of the performance problem better. You may find that using literals in your application will yield a better plan (and therefore better performance) at the cost of SQL compilation overhead.
- Try using DB2's index advisor (db2advis) to see if there are any useful indexes which you may have overlooked.

Related topic

- Build your next development project with IBM trial software, available for download directly from developerWorks.