Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 20, No. 2 · MAY 2006 $15

Add Color to Your Career with NoCOUG

Down Memory Lane

An interview with Bill Schwimmer. *See page 4.*

Is 24x7 a Myth?

Six well-known Oracles answer our questions. *See page 7.*

Hello, World!

A new column on programming languages. Guest columnist Krishna Kakatur introduces us to SQLJ. *See page 19.*

Big discount on Steven Feuerstein seminar! ***See page 3.*** ***Much more inside . . .***

SQL Sucks! – Part I

by Iggy Fernandez

Iggy Fernandez

The other terror that scares us from self-trust is our consistency; a reverence for our past act or word, because the eyes of others have no other data for computing our orbit than our past acts, and we are loath to disappoint them. . . . Bring the past for judgment into the thousand-eyed present, and live ever in a new day. . . . A foolish consistency is the hobgoblin of little minds, adored by little statesmen and philosophers and divines. . . . Speak what you think now in hard words, and tomorrow speak what tomorrow thinks in hard words again, though it contradict every thing you said today. **—Ralph Waldo Emerson**

Summary

In the first part of this multi-part article, we argue that query optimizers (including the Oracle query optimizer) are severely handicapped in their ability to generate good query execution plans and that, therefore, the promise of relational technology has gone largely unfulfilled. In the second part of this article, we will discuss the workarounds.

A Personal Note

In a previous issue of the *Journal*, I wrote a couple of articles extolling the power of SQL and received the following comment from Ravi Kulkarni of Phoenix, AZ.

"I think it is a stretch to say that the Oracle optimizer can optimize every possible SQL statement written using convoluted but severely limited syntax, the same way as an expert DBA with a procedural language."

Subsequent to the publication of those articles, I had occasion to participate in a major SQL tuning effort for a Fortune 500 company and encountered many cases in which the Oracle query optimizer was generating poor query execution plans. It appeared that Ravi was right!

Technical Background

SQL is what is called a "nonprocedural" language; i.e., an SQL query simply identifies a subset of data in the database without specifying how to go about extracting that data. The *promise* of relational technology was that the SQL programmer would be relieved from *most* of the responsibility for making the best use of computer resources and would only need to consider the "logical data model," not the "physical data model." The intention was that the major responsibility would be divided as follows:

1. First, the responsibility would devolve to the database engine, which would provide a selection of data processing algorithms (e.g., merge join, hash join, parallel processing, etc.), data storage methods (e.g., index-organized tables, table clusters, data partitioning), and indexing methods (e.g., b-tree indexes, bitmap indexes, function-based indexes).
2. Second, the responsibility would devolve to the database administrator who, in partnership with the application architect, would convert the "logical" database design into a "physical" database design and choose storage methods and indexing methods geared to the requirements of the application. The database administrator would also work with the users of the application to ensure that there was sufficient computer capacity (CPU, memory, I/O bandwidth, and network bandwidth) to meet the current and future needs.
3. Finally, the responsibility would devolve to a special component of the database engine called the "Query Optimizer," which would rely on statistical information such as row counts and histograms to make decisions about table processing order and index usage and to choose from among available data processing algorithms. The goal being to identify the "query execution plan" that minimizes the consumption of computer resources and the time required.

In this article, we will present certain defects of the query optimizer and argue that these defects are incurable. Note that we are not singling out Oracle in particular. The same defects exist in the technology of every other vendor.

Incurable Defects!

We really need to prepare the ground adequately with an overview of the inner workings of the query optimizer, but we only have space for the barest minimum of examples. Suppose that A is a "parent table" and B is a "child table" and that the optimizer is presented with a query of the following sort.

```
select * from A, B
 where <filtering criteria to be applied to A>
   and <filtering criteria to be applied to B>
   and <criteria to match records of A and B>
```

The optimizer can use either table A or table B as the "driving table" for the query. It will instruct the database engine first to filter the chosen table using the applicable filtering criteria and then to match the resulting record set with matching records from the other table using the

applicable matching criteria. It is obvious that the key to the choice of the driving table is the size of the record sets that result from applying the applicable filters to each table. For example, if the optimizer knew in advance that the filters on table B were very restrictive, then it would be wise to choose table B to be the driving table.

Now consider an example involving three tables.

```
select * from A, B, C
 where <filtering criteria to be applied to A>
   and <filtering criteria to be applied to B>
   and <filtering criteria to be applied to C>
   and <criteria to match records of A and B>
   and <criteria to match records of B and C>
   and <criteria to match records of A and C>
```

Since only two tables can be processed at a time, the optimizer must decide which two tables to process first, and the resulting record set must then be joined with the remaining table. For example, if the optimizer had some way of knowing that the matching criteria to be used to associate records of table B with those of table C were very restrictive, then it would be wise to process those tables first, so as to reduce the work required in finding matching records in table A. *Once again, we see that estimation of the number of qualifying records at each step of query processing is the key to the problem.*

The optimizer attempts to solve the problem by using statistical information such as row counts and histograms. A histogram is a summary of the data values in any one column of a table; e.g., we might have a histogram that tells us that California is the most populous state in the Union. However, histogram summaries for single columns are of no use when we are faced with more than one filter. Similarly, histograms offer no help in assessing how many rows in one table will match rows of another table.

Consider the following two examples.

```
select * from CarSales
 where Manufacturer = 'Toyota'
   and Model = 'Celica';

select * from CarSales
 where Manufacturer = 'Toyota'
   and ModelYear < 1975;
```

Now, the Oracle optimizer can use histograms to estimate the percentage of sales satisfying any single one of the criteria listed in the above SQL queries but has no way of accurately estimating what percentage of sales satisfy two or more criteria. To us it is obvious that all Celicas are Toyotas and that, therefore, the percentage of Celicas manufactured by Toyota equals the percentage of cars that are Celicas. It is also obvious to us that Toyota was not making cars prior to 1975 and that, therefore, the percentage of Toyotas sold before 1975 is, in fact, zero. Oracle, on the other hand, assumes that there is never any correlation between data items and, therefore, uses the following formulae.

```
Probability(Manufacturer = 'Toyota' and Model =
'Celica') =
    Probability (Manufacturer = 'Toyota')
  x Probability(Model = 'Celica')

Probability(Manufacturer = 'Toyota' and
ModelYear < 1975) =
    Probability(Manufacturer = 'Toyota')
 x  Probability(ModelYear < 1975)
```

In the first case, Oracle has underestimated the size of the result, and, in the second case, it has overestimated.

The error worsens as the number of filters increases, as in the following query, in which Oracle will further underestimate the number of rows in the result set.

```
select * from CarSales
 where Manufacturer = 'Toyota'
   and Model = 'Celica'
   and ModelYear >= 1975
```

Finally, consider what might happen if we encounter an "OR" conjunction such as the one in the following example.

```
select * from CarSales
 where Manufacturer = 'Toyota'
   or Model = 'Celica'
```

Let X, Y, and Z be any three filters. For readers familiar with the language of probability and statistics, we state the following results.

```
Probability(X and Y) =
    Probability(X) x Probability(Y given X)[1]

Probability(X and Y and Z) =
    Probability(X)
 x  Probability(Y given X)
 x  Probability(Z given X and Y)[2]

Probability(X or Y) =
    Probability(X) + Probability(Y)
 -  Probability(X and Y)

Probability(X or Y or Z) =
    Probability(X)
 +  Probability(Y)
 +  Probability(Z)
 -  Probability(X and Y)
 -  Probability(X and Z)
 -  Probability(Y and Z)
 +  Probability(X and Y and Z)
```

Since Oracle has no knowledge of "conditional probabilities," it uses the following alternatives (which are accurate if and only if X and Y are so-called "independent events") and consequently exposes itself to the dangers of underestimation and overestimation.

```
Probability(X and Y) =
    Probability(X) x Probability(Y)

Probability(X and Y and Z) =
    Probability(X) x Probability(Y) x
Probability(Z)

Probability(X or Y) =
    Probability(X) + Probability(Y)
  - Probability(X) x Probability(Y)

Probability(X or Y or Z) =
    Probability(X)
  + Probability(Y)
  + Probability(Z)
  - Probability(X) x Probability(Y)
  - Probability(X) x Probability(Z)
  - Probability(Y) x Probability(Z)
  + Probability(X) x Probability(Y) x
Probability(Z)
```

[1] The probability that the second filter is also satisfied when the first filter is satisfied.

[2] The probability that the third filter is also satisfied when the first and second filters are both satisfied.

We now turn our attention to the problem of joining tables using record matching criteria to associate records from one table with those of another. Let A and B be two tables and let *NR_A* and *NR_B* represent the number of rows in A and B respectively. Further, let *NDV_A* represent the number of distinct values in the joining column of A and let *NDV_B* represent the number of distinct values in the joining column of B. In the absence of better information, we could assume that each distinct value of the joining column in B is equally represented; i.e., it occurs exactly (*NR_B / NDV_B*) times. If we could further assume that every value in the joining column of A is represented in the joining column of B, then the number of records resulting from the joining operation would be *NR_A x* (*NR_B / NDV_B*). A symmetric argument could be used to estimate the answer to be *NR_B x* (*NR_A / NDV_A*). Oracle chooses the minimum of these two answers, which can be expressed as (*NR_A x NR_B*) / *maximum*(*NDV_A, NDV_B*).

Once again, we see that Oracle is using assumptions that may be far from the truth. As the number of filter criteria and the number of tables grow, the errors also grow and the query optimizer's strategy is undermined.

A participant on the "Ask Tom" website expressed the problem perfectly with the following comment.

"I became interested in the CBO's selectivity calculations trying to understand why it comes up with some of the ridiculously low cardinality estimates (like 1 when in reality there are 80,000+) which then lead to disastrous access plans that take hours, provided they finish at all, instead of minutes or seconds."[3]

Concluding Remarks

We should warn the reader that the opinions we expressed in this article don't have mainstream support. The mainstream opinion is that most, if not all, failures of Oracle's query optimizer can be remedied by improving the quality of the statistical information on which the optimizer relies. Here, for example, is a quote from an article published this year in the journal of the Independent Oracle Users Group (IOUG).

"One of the greatest problems with the Oracle Cost-based optimizer was the failure of the Oracle DBA to gather accurate schema statistics. . . . The issue of stale statistics and the requirement for manual analysis resulted in a "bum rap" for Oracle's cost-based optimizer, and beginner DBAs often falsely accused the CBO of failing to generate optimal execution plans when the real cause of the sub-optimal execution plan was the DBA's failure to collect complete schema statistics."[4]

Further Reading

Document 68992.1 in the Oracle knowledge base (Metalink) is a good source of information on the formulae and assumptions used by the Oracle query optimizer. ▲

Iggy Fernandez is a DBA for the Verizon phone company. You can reach him at **iggy_fernandez@hotmail.com**.

[3] **asktom.oracle.com/pls/ask/f?p=4950:8:::::F4950_P8_DISPLAYID:4344365159075**.

[4] **www.ingentaconnect.com/content/ioug/sj/2006/00000013/00000001/art00003**.