Statistics: A key to improving Oracle Performance

By Dennis Adams, Quest Software [1]

With the launch of Oracle 7, Oracle introduced the Cost-Based Optimizer, as an alternative to the previous Rule-Based Optimizer.

This was a major advance for Oracle, and can result in potentially large performance improvements. Unfortunately, this also results in an additional workload for the DBA.

A key factor in getting the best out of the Oracle 7 Cost-Based Optimizer is the proper maintenance of table statistics.

What the Optimizer does

The Query Optimizer is part of the Oracle "kernel". It is responsible for taking a simple statement, such as…

```
SELECT E.EMPNO, E.ENAME, D.DNAME
FROM EMP E, DEPT D
WHERE E.DEPTNO=D.DEPTNO;
```

... and generating a plan (a sequence of physical steps) for returning the actual data which the user wants.

In order to do this, the optimizer will need to..

- Verify that the SQL syntax is correct,
- Check that the tables and columns specified in the statement actually exist,
- Identify which indexes may be available,
- Estimate the cost of each possible execution plan for accessing the data,
- Finally, the optimizer will need to determine what it considers the most efficient plan.

For example, given the above query, one valid plan (a "Nested Loop") may be…

1. Read every row from EMP (a full table scan),
2. For each row in EMP – use the DEPTNO column to look up the index for DEPT, and use this to extract the DNAME.

An alternative plan (a "Hash Join", which may be more efficient) could be…

1. Read every row from EMP (a full table scan),
2. Read every row from DEPT (a second full table scan),
3. Merge the two sets of data together.

The Query Optimizer's role is to determine which of these options would be more appropriate.

---

[1] Dennis wrote this article when he was UK Technical Support Manager at Quest Software. It appeared as a Technical Feature in theWinter 1997 Issue 29 of "Relate", The UK Oracle User Group Journal.

How Queries are planned

In Oracle 7.0 and above, there are three modes of operation which are available for the optimizer. These can be set by…

- Using an ALTER SESSION command, or
- Using "hints", or
- By using the init.ora parameter OPTIMIZER_MODE

The following modes are available…

- RULE tells the optimizer to use the older rule-based approach
- COST tells it to use the cost-based approach
- CHOOSE enables the optimizer to chose which approach.

The Cost-Based optimizer uses database information (table size, number of rows, key spread etc.), rather than rigid rules, to determine what "plan" to follow for accessing data.

Here are a number of issues which may guide the Cost-Based Optimizer's choice…

- If there is a unique index on a table column, it maybe worthwhile using it.
- If an index is very "selective" (i.e. a small proportion of the data would be returned), it would be advantageous to use it.
- If there are two tables, one large and one small, it may be more efficient to read one of them first (the "driving table") and use it as the basis for reading the second one.
- If we expect to get more than a certain percentage of data returned from one table, it may be more efficient to read it directly, rather than using the index.
- At times, Oracle may decide that it is more efficient to read an entire table ("full table scan"), instead of using the index. This can be surprising at first, until you realise that using an index requires two I/O read requests – one to the index to extract a "rowid", and the second read to get the actual table data.

However, this planning is only as good as the information provided to it. Such information is stored in the system tables DBA_TABLES, DBA_TAB_COLUMNS and DBA_INDEXES. These are loaded using the ANALYZE command.

Note that if no statistics are available for a particular table, the optimizer is forced to use the older Rule-Based logic when planning. This could result in inefficient plans being generated.

Loading Statistics

As mentioned above, the key to getting the best out of the Cost-Based Optimizer is to ensure that it has up-to-date information on the data distribution.

This data is loaded using the ANALYZE command.

An example of the syntax is…

```
ANALYZE TABLE SCOTT.EMP
COMPUTE STATISTICS;
```

This would read all rows of Scott's EMP table and populate the relevant columns of the DBA_TABLES catalog, so providing the relevant information for the Cost-Based Optimizer.

One of the problems with the above command is that it can be relatively time-consuming. It would require a full-scan of the table EMP, and therefore could take a similar amount of time as an index build. (It also requires a large amount of temporary segments – at least as big as the table itself.)

As an alternative, therefore, it is possible to estimate the statistics, using a sample. For example…

```
ANALYZE TABLE SCOTT.EMP
ESTIMATE STATISTICS
SAMPLE 10 PERCENT;
```

... would take a 10% sample of all rows on the table and estimate the statistics. This would obviously be faster, but may be less accurate, depending upon the sampling accuracy etc.

For an entire schema, it is possible to run a DBMS_UTILITY package. For example…

```
EXECUTE DBMS_UTILITY.ANALYZE_SCHEMA
('SCOTT', 'ESTIMATE');
```

... would automatically estimate statistics for all tables in Scott's schema.

One of the things the analyze table command calculates (or estimates) are figures on "index selectivity", i.e. the percentage of rows returned by any one value of an index.

Histograms – further complications

The statistics held in DBA_TABLES etc. are useful for the Optimizer, but they may be misleading. For example, in training courses, I sometimes use the following table…

```
CREATE TABLE PEOPLE(
PTYPE CHAR(1) CHECK PTYPE BETWEEN 'A' AND 'Z',
NAME CHAR(40),
SOC_SEC_NUM CHAR(9) . . .

);
```

If the above table was loaded with 26,000 rows, you might be forgiven for thinking that the SQL…

```
SELECT *
FROM PEOPLE
WHERE PTYPE = 'M';
```

…would return approx. 1,000 rows. (i.e. 1/26th of the total). In that case, the "index selectivity" would be 0.038 (1/26).

In practice, the likelihood of any row being 'M' would be far more of less than 1/26th. For example, if the column ptype was being used to indicate the sex of the person, the number of rows returned could be 13,000 or more !

Imagine the situation if the Oracle query processor, expecting to receive 1,000 rows of data, suddenly finds itself flooded with thirteen times that number !

In the above example, if Oracle had been aware that 50% of the data would be returned, it may decide that it would be more efficient to by-pass the indexes entirely and do a full table scan.

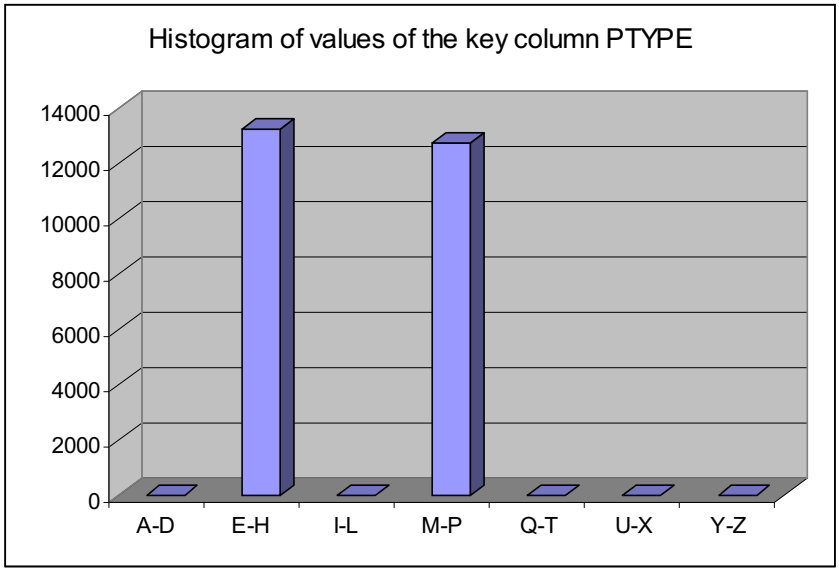
For this reason, and others, Oracle version 7.3 introduced a further enhancement to the ANALYZE command, by populating an additional system table called DBA_HISTOGRAMS. The purpose of the histogram is to ensure that Oracle is "fore-warned" of the expected amount of data it could receive for any given query.

The results of the analysis are placed in the new SYS table DBA_HISTOGRAMS.

When creating a histogram of data, you need to determine the number of "buckets" which hold the data. See the enclosed figure for an example histogram.

In the example of the PEOPLE table, you could realistically describe the total values of the key in two "buckets".

Possible Column Values	A-D	E-H	I-L	M-P	Q-T	U-X	Y-Z	TOTAL
No of occurrences	0	13251	0	12749	0	0	0	26000



Oracle 7.3 also includes an enhanced syntax for the analyze command, which enables statistics to be computed (or estimated) for…

1. TABLE – i.e. just the columns themselves, not any indexes,
2. COLUMNS – i.e. specifying individual columns,
3. ALL INDEXED COLUMNS – i.e. just columns which are indexed.

There is also the additional clause "FOR ALL INDEXES" which enables all the indexed columns to be analyzed.

The following is an example of the Oracle 7.3 syntax….

```
ANALYZE TABLE PEOPLE
ESTIMATE STATISTICS
FOR ALL INDEXED COLUMNS
SIZE 20
SAMPLE 4000 ROWS;
```

Creating a strategy for maintaining statistics

There are a number of issues which you need to address when using statistics…

1. The frequency of analyzing the tables
2. The number of buckets used
3. Compute or Estimate sample size ?

DBAs need to have some idea of the way in which data is being accessed and updated.

For example, if a table is relatively static (a reference table, for example), the statistics could be loaded once and then left untouched.

A table which is subject to lots of changes (e.g. a GL_INTERFACE table),, would need the statistics updating more frequently. This is particularly true of tables which are used for data feeds to management information systems etc.

Many customers have order processing or finance systems with linearly increasing keys (such as order_number, customer_number etc.). As a result, the average "value" of key column increases over time, and it's selectivity and histogram show a "skew". In these circumstances, not only do indexes have to be periodically rebuilt, but so do the statistics.

Wherever possible, use the COMPUTE STATISTICS clause to collect real data, rather than estimating it. However, experience suggests that using ESTIMATE not only reduces the amount of time required, but often gives very accurate results, providing that the sample size is appropriate.

By default, the Estimate statistics clause causes Oracle to read only the first 1064 rows of a table during its operation. This is generally considered to be an accurate sample, but if your data is "Skewed", you may need to change the sampling rate.

It is possible to create an SQL script to automatically generate COMPUTE or ESTIMATE commands, based on some pre-determined criteria (e.g. table size, or table name etc.)

Don't forget to enhance your table ANALYZE commands as tables are modified and/or added.

Note that the Oracle SYS tables and data dictionary should never be analyzed, since Oracle 7 system table accesses have been tuned to take advantage of the Rule-Based Optimizer !!!

Conclusions

Oracle are increasingly recommending that customers use the Cost-based optimizer for their applications wherever possible.

If you intend to follow their advice, make sure that you have implemented a sensible strategy for statistics management.

Sources

1. Oracle Performance Tuning 2nd Edition by Mark Gurry & Peter Corrigan. O'Reilly & Associates, ISBN 1-56592-237-9
2. Advanced Oracle Tuning and Administration by Eyal Aronoff, Kevin Loney & Noorali Sonawalla. Oracle Press / Osborne Books. ISBN 0-07-882241-6
3. Oracle DBA Handbook 7.3 Edition by Kevin Loney. Oracle Press / Osborne Books. ISBN 0-07-882289-0
4. Performance Considerations Converting From Rule to Cost. Mark Gurry. (Mark Gurry and Associates – White Paper)

Dennis Adams is UK Technical Support Manager for Quest Software, which supplies Database Management and Monitoring Software for the Oracle DBMS. He can be contacted via email on dennis@oz.quests.com.
