Manually Tuning SQL

It has been said many times in books, articles, and other publications that over 90% of all performance problems on a database are due to poorly written SQL. Often, database administrators are given the task of “fixing the database” when queries are not performing adequately. The database administrator is often guilty before proven innocent—and often has the task of proving that a performance problem is not the database itself, but rather, simply, SQL statements that are not written efficiently. The goal, of course, is to have SQL statements written efficiently the first time. This chapter’s focus is to help monitor and analyze existing queries to help show why they may be underperforming, as well as show some steps to improve queries.

If you have SQL code that you are maintaining or that needs help to improve performance, some of the questions that need to be asked first include the following:

- Has the query run before successfully?
- Was the query performance acceptable in the past?
- Are there any metrics on how long the query has run when successful?
- How much data is typically returned from the query?
- When was the last time statistics were gathered on the objects referenced in the query?

Once these questions are answered, it helps to direct the focus to where the problem may lie. You then may want to run an explain plan for the query to see if the execution plan is reasonable at first glance. The skill of reading an explain plan takes time and improves with experience. Sometimes, especially if there are views on top of the objects being queried, an explain plan can be lengthy and intimidating. Therefore, it’s important to simply know what to look for first, and then dig as you go.

At times, poorly running SQL can expose database configuration issues, but normally, poorly performing SQL queries occur due to poorly written SQL statements. Again, as a database administrator or database developer, the best approach is to take time up front whenever possible to tune the SQL statements prior to ever running in a production environment. Often, a query’s elapsed time is a benchmark for efficiency, which is an easy trap in which to fall. Over time, database characteristics change, more historical data may be stored for an application, and a query that performed well on initial install simply doesn’t scale as an application matures. Therefore, it’s important to take the time to do it right the first time, which is easy to say, but tough to accomplish when balancing client requirements, budgets, and project timelines.
9-1. Displaying an Execution Plan for a Query

Problem
You want to quickly retrieve an execution plan from within SQL Plus for a query.

Solution
From within SQL Plus, you can use the AUTOTRACE feature to quickly retrieve the execution plan for a query. This SQL Plus utility is very handy at getting the execution plan, along with getting statistics for the query's execution plan. In the most basic form, to enable AUTOTRACE within your session, execute the following command within SQL Plus:

```sql
SQL> set autotrace on
```

Then, you can run a query using AUTOTRACE, which will show the execution plan and query execution statistics for your query:

```sql
SELECT last_name, first_name
FROM employees NATURAL JOIN departments
WHERE employee_id = 101;
```

<table>
<thead>
<tr>
<th>LAST_NAME</th>
<th>FIRST_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kochhar</td>
<td>Neena</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td>1</td>
<td>33</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>1</td>
<td>NESTED LOOPS</td>
<td></td>
<td>1</td>
<td>33</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>2</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>EMPLOYEES</td>
<td>1</td>
<td>26</td>
<td>1 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td></td>
<td>INDEX UNIQUE SCAN</td>
<td>EMP_EMP_ID_PK</td>
<td>1</td>
<td></td>
<td>0 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 3</td>
<td>TABLE ACCESS BY INDEX ROWID</td>
<td>DEPARTMENTS</td>
<td>11</td>
<td>77</td>
<td>1 (0)</td>
<td>00:00:01</td>
</tr>
<tr>
<td>* 4</td>
<td>INDEX UNIQUE SCAN</td>
<td>DEPT_ID_PK</td>
<td>1</td>
<td></td>
<td>0 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>
```

Statistics

- 0 recursive calls
- 0 db block gets
- 4 consistent gets
- 0 physical reads
- 0 redo size
- 490 bytes sent via SQL*Net to client
- 416 bytes received via SQL*Net from client
- 2 SQL*Net roundtrips to/from client
- 0 sorts (memory)
- 0 sorts (disk)
- 1 rows processed