SQL “Trick”-Control The Optimization Strategy

by Jeff Ash

Optimization of SQL queries is key to maximizing the performance of an application or report. The BASIS SQL engine optimizes queries using a strategy whereby it examines the available indexes on the tables involved in the query, checks some statistical data gathered during table analysis, and makes a “best” decision for an index to use that will return the results as quickly as possible. However, there are some edge cases where the SQL engine cannot know for certain which index would be best, so it simply picks one. Sometimes it doesn’t pick the best option.

Let’s take a look at a simple, real-world example that BASIS encountered recently. We noticed one of our databases was responding significantly slower than expected to certain queries, to the point that it was detrimental to the operation of a key business process. Using the “Complete Optimization” SQL Engine Log Level setting in the Enterprise Manager (EM), we were able to see the index being chosen for optimization. It was not the one we wanted. After examining the nature of the WHERE clause, it was clear that the issue was simply a matter of the SQL engine seeing two equally useful indexes and picking the “wrong” one.

So, how did we change our query to force the SQL engine to use a different index when the BBj SQL engine does not have a hinting feature? We used a little trick to remove certain columns from consideration for optimization, thus limiting the SQL engine’s options and forcing it to choose the index we wanted. We leveraged a fundamental concept related to optimization, which is that optimization can only occur on a standalone column reference where no expressions, formulas, aggregations, or scalar functions are in play on that column reference.

For example, suppose we have a table FOO with an index on the COL_1 column and a second index on COL_2:

The SQL engine can use either column for optimization in this case:

```
SELECT * FROM foo WHERE col_1 = 10 AND col_2 > 5
```

We can force the SQL engine to only use COL_2 for optimization in this case:

```
SELECT * FROM foo WHERE col_1 + 0 = 10 AND col_2 > 5
```

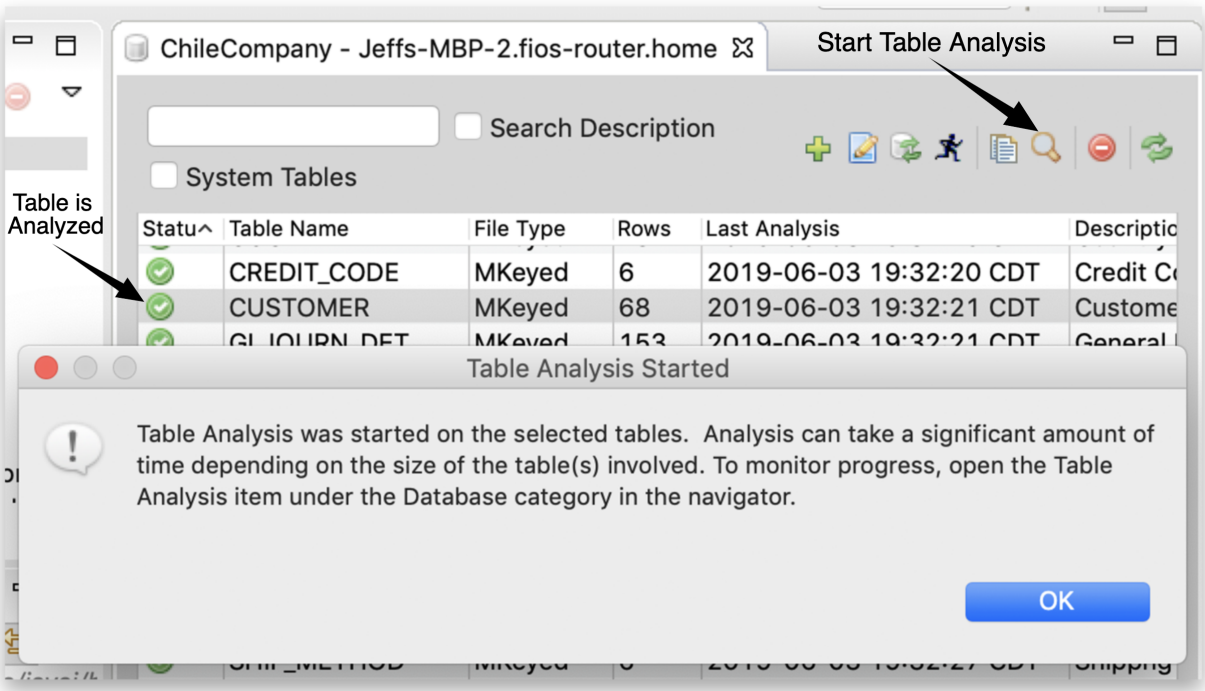
Notice the difference. By changing the COL_1 reference to an expression, the SQL engine cannot use it for optimization any longer. Now, the only option is COL_2. We haven't changed the results, but we changed what the SQL engine can choose in order to optimize the retrieval of those results.

If we want the engine to ignore a CHAR or VARCHAR column in an index, we can do something similar:

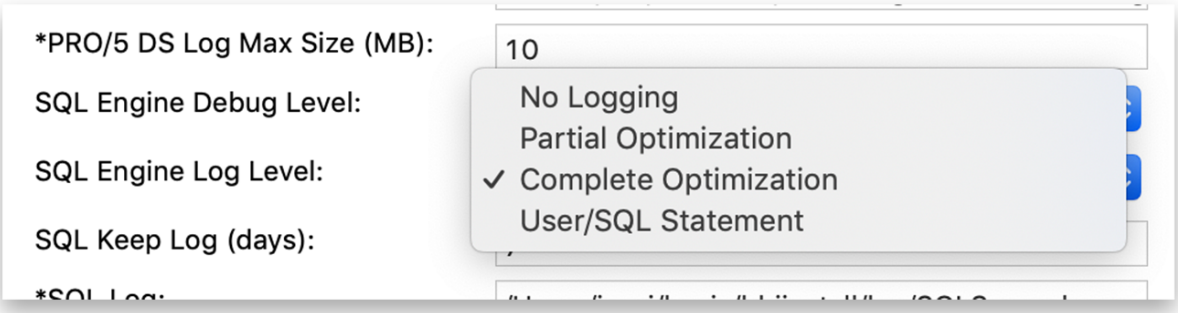
```
SELECT * FROM foo WHERE col_1 + '' = 'Bar' AND col_2 > 5
```

What was the result of our optimization? The query improved its performance by ***300-fold*** in the most extreme situation! Be sure to consider employing this trick yourself whenever you find a query running slower than expected. Here are the steps to take:

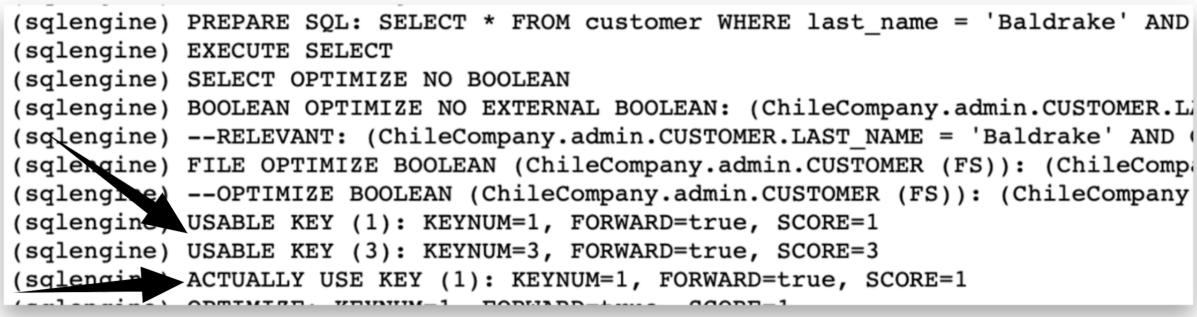
1. Ensure that table analysis has been run on the tables used in the query. Generally, this action is only required one time unless new indexes have been added to the tables or if the "shape" of the table has changed through the addition of a substantial number of new records with different characteristics. See Quickly fix slow SQL for more detail.



2. Set the SQL engine log level to "Complete Optimization" in EM.



3. Re-run the query and examine the log to determine which index was selected by the SQL engine. Take note of the “USABLE KEY” items (these are the options available) and the “ACTUALLY USE KEY” which indicates the key the SQL engine chose as “best.”



4. Force the SQL engine to use the correct index by applying an expression, formula, aggregation, or scalar function to the column reference:

```
SELECT * FROM foo WHERE col_1 + 0 = 10 AND col_2 > 5
```

5. Re-run the query and check the log. If all is well, reset the logging level and report success!

A little creativity, in conjunction with a solid understanding of how the SQL engine optimizes queries and utilizes indexes, can go a long way in coming up with solid solutions that meet the end user’s needs.