

The White Papers

Performance Tuning for Mixed-Case Queries

By Kevin Loney

Contents

<i>Introduction</i>	3
<i>Configuring the Test Environment</i>	3
<i>The Problem</i>	4
<i>Rewriting the Query</i>	6
<i>Simplifying the Comparison Process</i>	9
<i>About the Author</i>	10

Performance Tuning for Mixed-Case Queries

By Kevin Loney

Introduction

One of TOAD's most useful capabilities is that it supports quick evaluation of multiple execution paths for a single query. In this article, you will see how TOAD can be used to quickly judge the value of different solutions to a common problem — how to perform index-based queries of mixed-case data.

A column containing people's last names may contain values such as McMahon, von Hagel, and Dell'Amico. If you store the data in a mixed-case format and your users can use the name column as a limiting condition in their queries, then you will need to support index-based lookups of that field. If you are using Oracle8i, you can use function-based indexes to support this type of query, but there are alternative — and possibly better — methods available to users of all versions of Oracle. In this article, the focus is on a query tuning solution that does not involve using function-based indexes.

Configuring the Test Environment

For this set of tests, the queried column will be mixed-case values in the Ename column of the standard EMP table:

```
create table EMP
(Empno    NUMBER(4) NOT NULL,
Ename     VARCHAR2(10),
Job       VARCHAR2(9),
Mgr       NUMBER(4),
Hiredate  DATE,
Sal       NUMBER(7,2),
Comm      NUMBER(7,2),
Deptno    NUMBER(2));
```

In its standard demo configuration, EMP contains 14 records, all with uppercase names. In order to create a suitable set of test data, those records were repeatedly re-inserted into the table with different values for the Ename and Empno columns. Ultimately, the EMP table contained 86,016 records, divided evenly among all uppercase, all lowercase, and mixed case names. After the table was populated, an index was created on the Ename column:

```
create index ENAME_NDX
on EMP(Ename)
storage (initial 1M next 1M pctincrease 0)
tablespace users;
```

The table and index were then analyzed using the **compute statistics** option.

To analyze the query performance, TOAD and the optional Xpert Tuning module were installed on the client machine.

The Problem

Like many architecture decisions, the indexing strategy for the EMP table is driven by conflicting business requirements. Users need to query the data based on the Ename column, but the values must be displayed to the user properly in mixed case. Furthermore, the table is subject to a large number of performance-sensitive **inserts** and **updates**, so very few indexes are permitted on the table. Any query tuning solution that impacts the performance of **inserts** is unacceptable. For that reason, you cannot maintain a second column in the table (such as Ename_Upper) populated in a case-consistent manner via triggers.

This limitation on the tuning options may seem unfair, but it is increasingly common in packaged or performance-sensitive OLTP systems. Also, creating an additional index on the EMP table affects the availability of the database, since a DDL lock is required during index creation. If the system requires high availability, then adding additional indexes in an ad hoc fashion may not be an option.

Given those restrictions, here is the query to tune — all names that begin with ‘Jo’, regardless of whether they are entered as ‘JO’, ‘jo’, or ‘Jo’:

```
select * from EMP
  where UPPER(Ename) like 'JO%';
```

In the sample data set, this query will return 237 of the 86,016 rows. That is a small set of records, so ideally this query will use an index. Since there is already an index on the Ename column, the goal of this tuning effort is to force that index to be used.

Since we’ll be evaluating multiple query execution paths, start with TOAD’s SQL Editor (from the Menu, select Database > SQL Editor). Enter the query in the SQL window, as shown in Figure 1. To see the explain plan, click on the ambulance icon shown on the top toolbar. If you have installed the TOAD explain plan table, you will see the explain plan as shown in Figure 1.

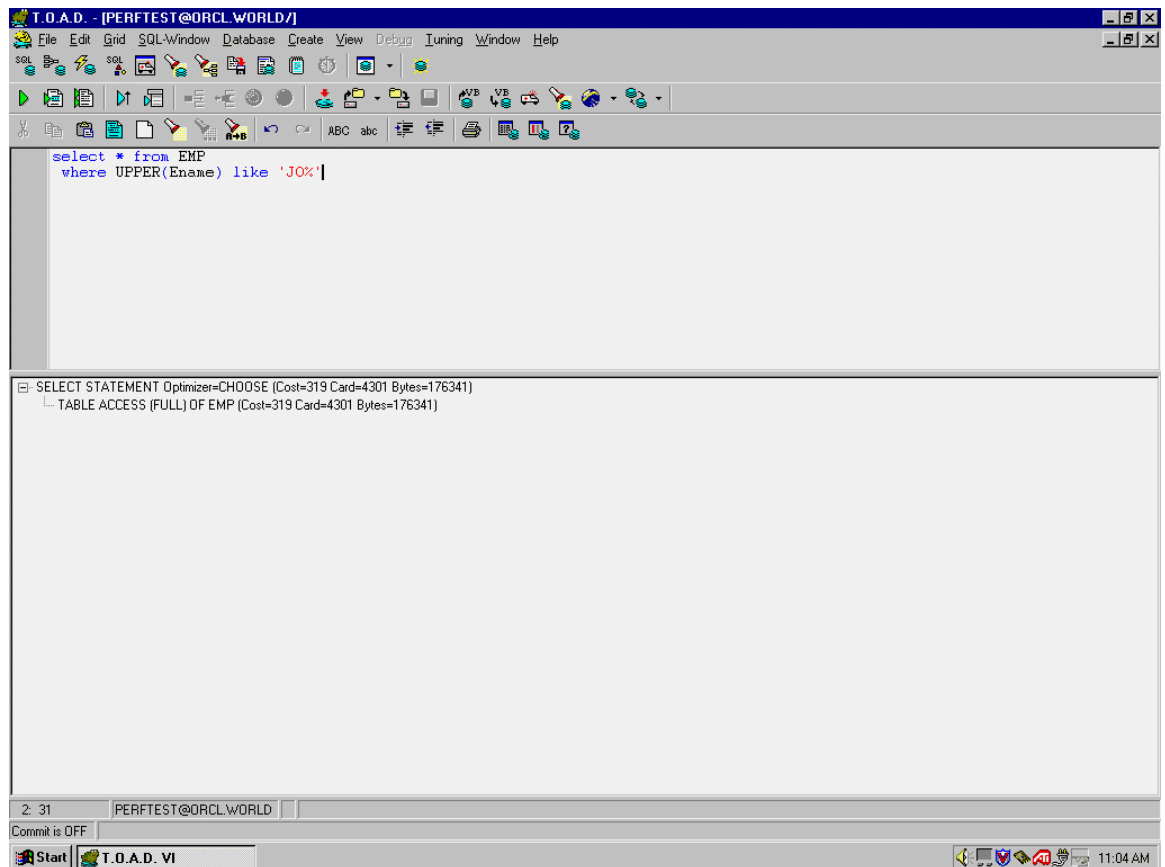


Figure 1. Initial Query and Explain Plan

The explain plan shows that the query will perform a full table scan, with a cost of 319. As expected, the index on Ename was not used because the UPPER function was performed on that column in the **where** clause.

What happens if you add an INDEX hint?

```
select /*+INDEX(emp)*/
      * from EMP
      where UPPER(Ename) like 'JO%';
```

Oracle ignores the hint, as shown in Figure 2. In this case, using the index would require Oracle to perform a full index scan followed by Rowid-based accesses of multiple rows in the table.

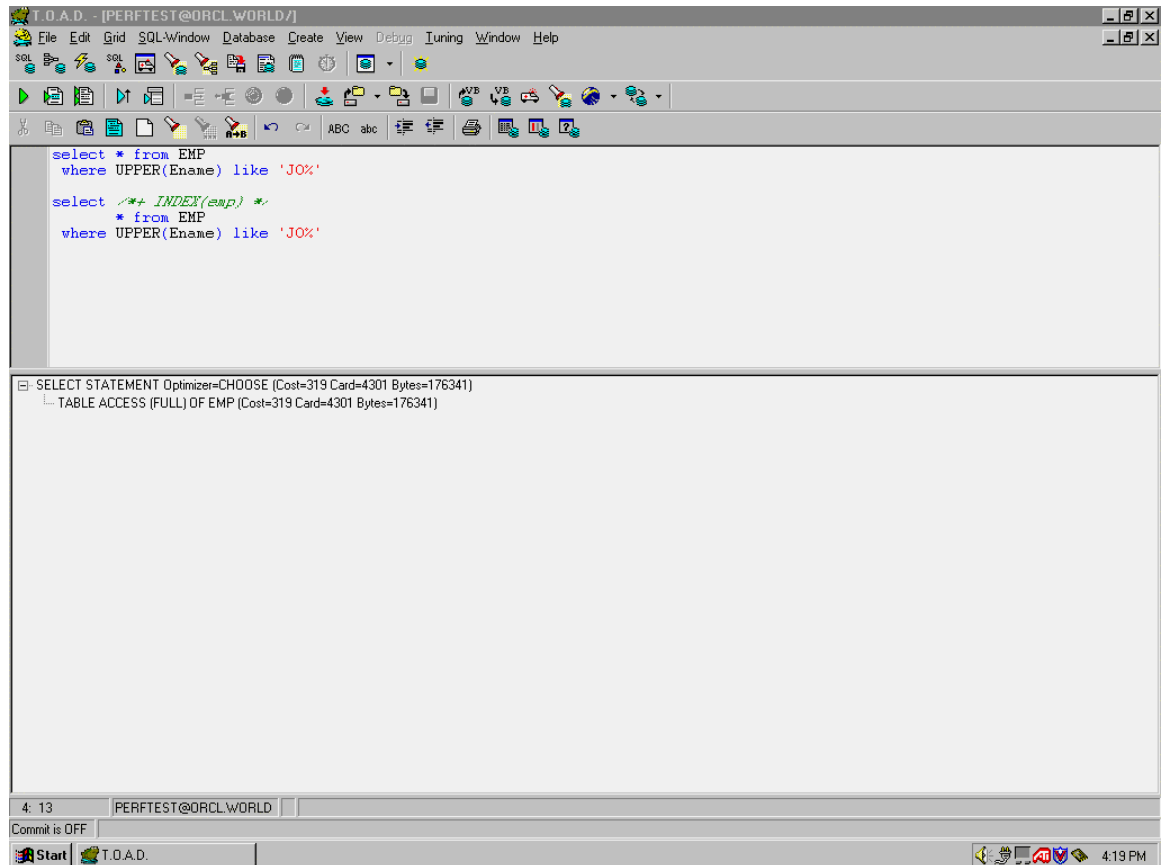


Figure 2. Query with INDEX hint ignored

Rewriting the Query

If we cannot add new indexes without impacting the performance and availability of the data entry system, then the first tuning approach should be to force the query to efficiently use the available index. The performance of the index-based version of the query can then be compared to the table scan version. To get Oracle to consider using the query, we need to rewrite the query.

First, apply some logic to the **where** clause. As written originally, it is:

```
where UPPER(Ename) like 'JO%';
```

That is, it is querying for cases where the name begins with 'Jo', 'JO', 'jo', and 'jO'. What is the performance of a query of just one of those cases? Enter the modified query in TOAD's SQL Editor and generate the plan. Note that you do not have to delete the old query – just be sure your cursor is on the new query when you generate the plan. The new query is:

```
select * from EMP
  where Ename like 'JO%';
```

As shown in Figure 3, the explain plan for this query reports a cost of 81.

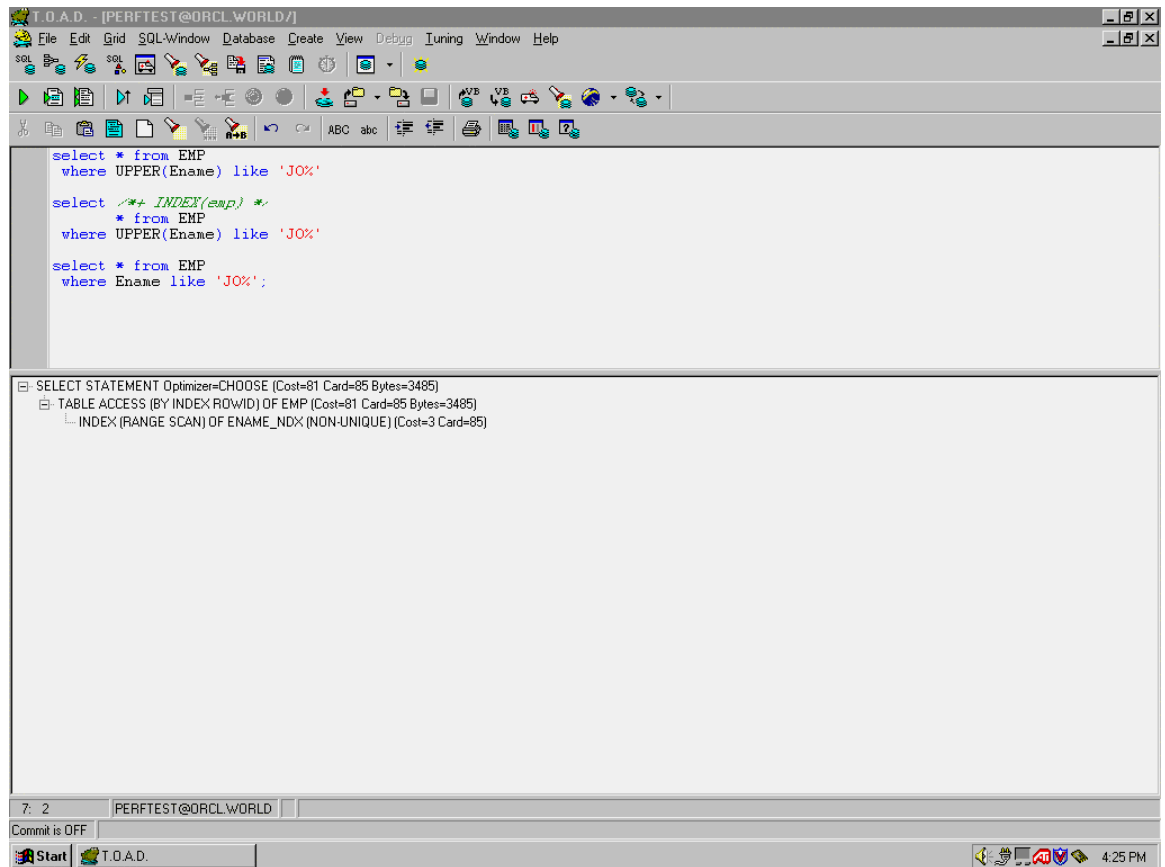


Figure 3. Explain Plan for modified query

What is the cost for a query of the 'Jo%' records?

```

select * from EMP
where ENAME like 'Jo%';

```

This query also has a cost of 81. Both of these queries show the use of the ENAME_NDX index. If we write a new query that uses the **union all** operator to combine the results of these two queries, the cost will be 162. If we add in a third query to cover the case where the name is like 'jo%', the total cost will be 243, as shown in Figure 4.

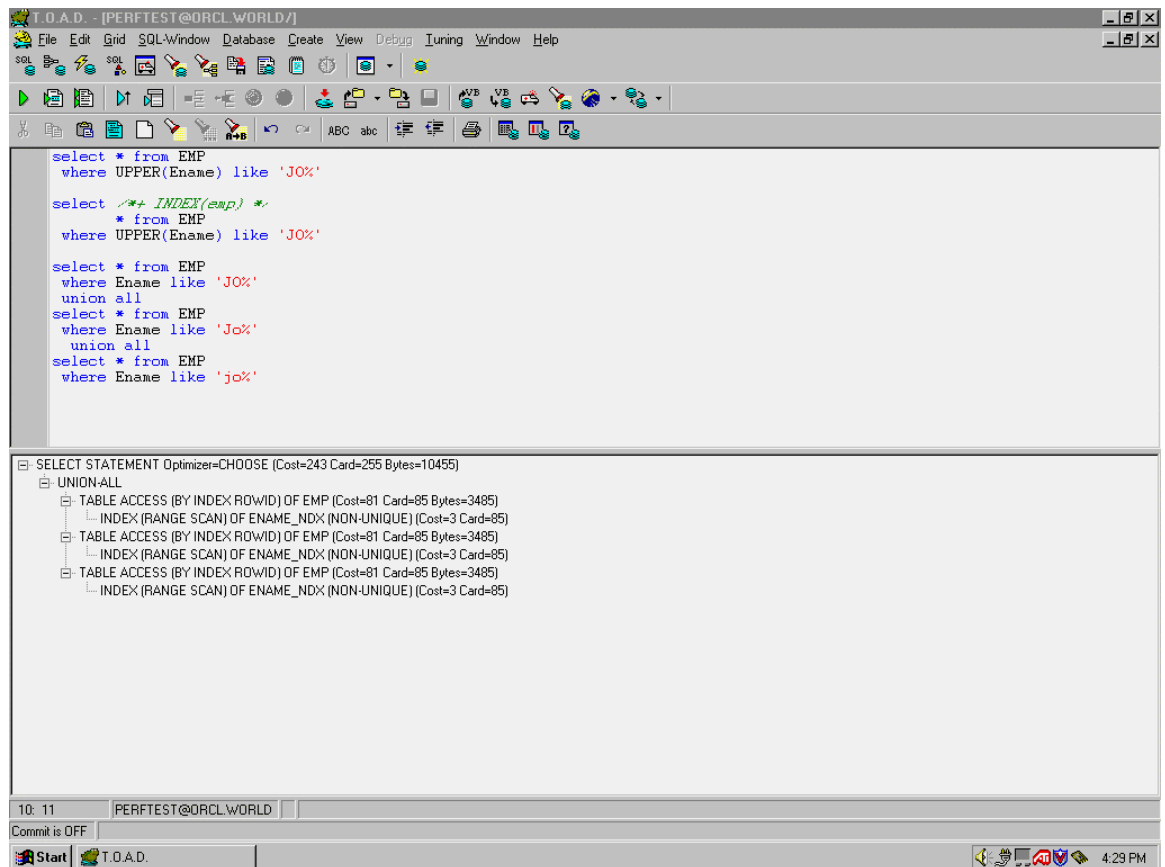


Figure 4. Explain query for UNION ALL query

What if we combine these criteria in the **where** clause instead of using separate **unioned** queries? The conditions can be combined via **or** clauses:

```

select * from EMP
where
(Ename like 'JO%' or Ename like 'Jo%' or Ename like 'jo%')
    
```

Using the SQL Editor, generate the explain plan, as shown in Figure 5. The cost of this query – which uses concatenation instead of a **union**, is 92. The cost of the original query has been cut from 319 to 92.

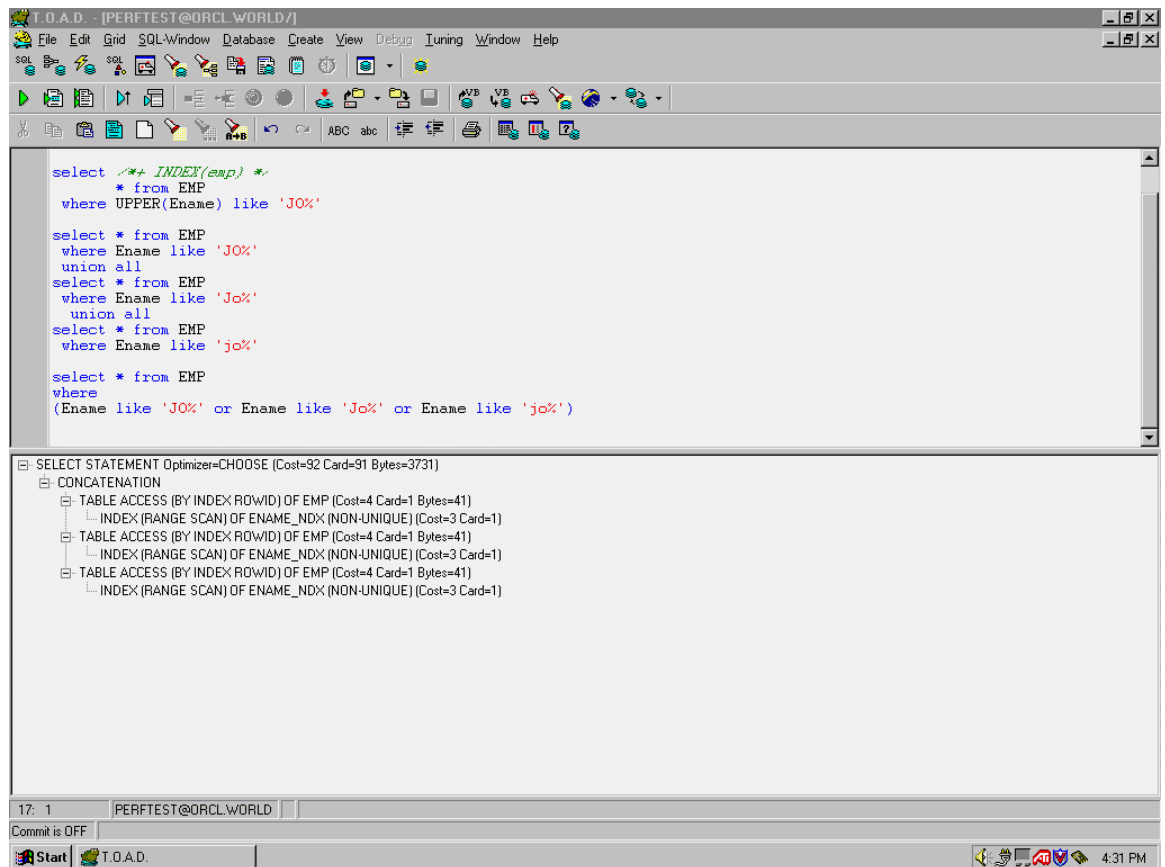


Figure 5. Explain Plan for the modified query

The new query is slightly different from the original query – it does not cover the situation in which the name is entered ‘jo%’. If you add that additional criteria, Oracle will by default perform a full table scan instead of an index-based query. To force it to use an index, you will need to give it an INDEX hint, which will lower the cost to 96. Clearly, you need to understand both the data distribution and the business impact of code changes made for performance reasons.

Simplifying the Comparison Process

To quickly compare multiple performance scenarios, use TOAD’s Xpert Tuning option. From the TOAD menu, select the Xpert Tuning option from the toolbar (note that if you have not used this option before, you will need to install the Xpert Tuning server-side objects first). When you select the Xpert Tuning option, you can automatically generate explain plans showing the impact of four separate hints on the SQL statement. If you need to investigate additional alternatives, you can create a new scenario to evaluate by clicking on the Create Scenario icon.

To compare the costs of multiple scenarios, generate explain plans for all of the scenarios (via the Generate Explain Plans icon) and then click on the Comparison View tab. Click the Plan tab, as shown in Figure 6, to see the query cost comparisons for the scenarios.

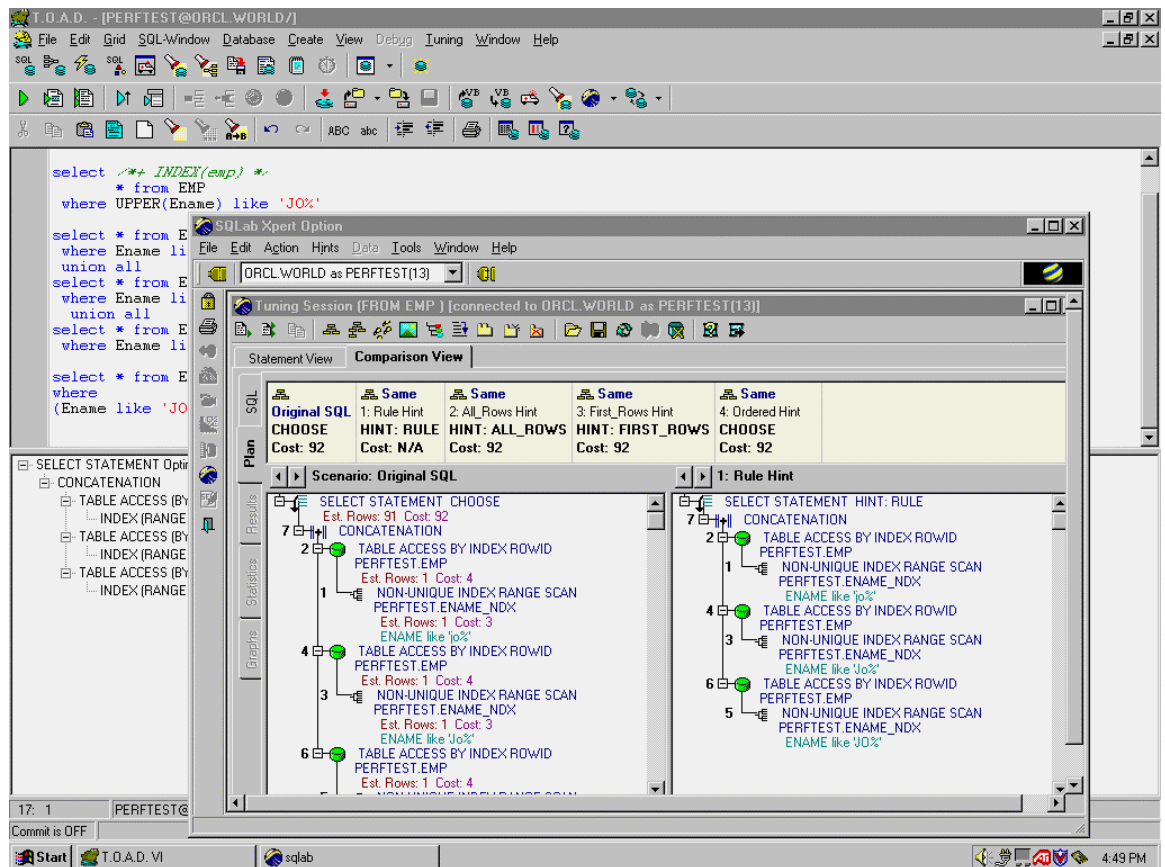


Figure 6. Query Cost Comparison for the Scenarios

As shown in Figure 6, the addition of the hints does not improve the query further. You can use the scenario comparisons to evaluate the impact of different objects, structures, and indexing options. Given constraints such as high availability and **insert** performance, favor rewriting SQL over creating new indexes.

About the Author

Kevin Loney is an independent consultant and author specializing in Oracle database administration and tuning. His online sites include <http://www.kevinloney.com> and <http://www.lonyx.com>, a search engine where database professionals can find scripts, tips, OCP help, and more.