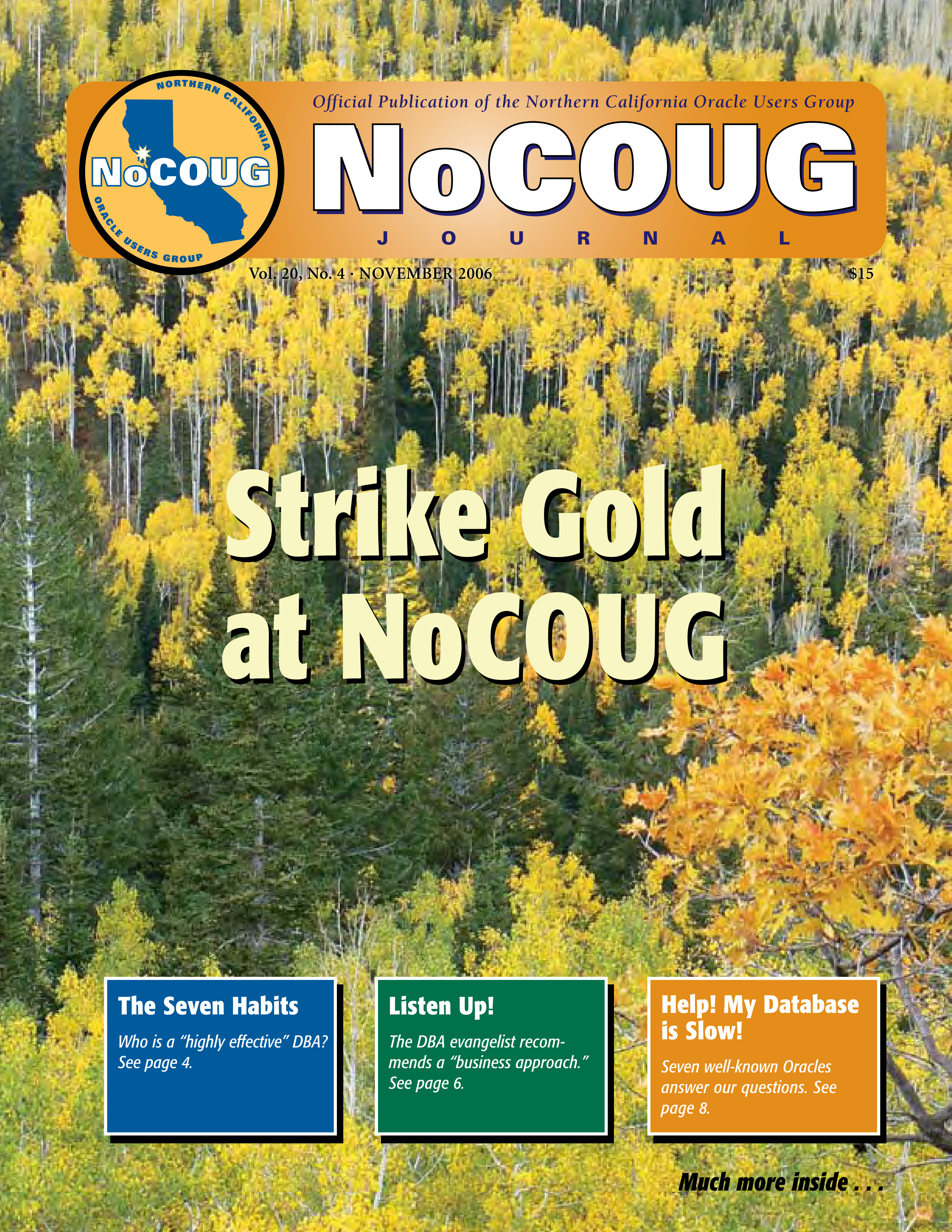
Official Publication of the Northern California Oracle Users Group

NoCOUG JOURNAL

Vol. 20, No. 4 · NOVEMBER 2006 $15



Strike Gold at NoCOUG

The Seven Habits

Who is a "highly effective" DBA? See page 4.

Listen Up!

The DBA evangelist recommends a "business approach." See page 6.

Help! My Database is Slow!

Seven well-known Oracles answer our questions. See page 8.

Much more inside . . .

SQL Sucks! – Part III

by Iggy Fernandez

Then felt I like some watcher of the skies;
When a new planet swims into his ken;
Or like stout Cortez when with eagle eyes
He star'd at the Pacific–and all his men
Look'd at each other with a wild surmise–
Silent, upon a peak in Darien.—**John Keats**

Iggy Fernandez

Recap

In the first installment of this multi-part essay, we explained that SQL is what is called a "non-procedural" language; i.e., an SQL query simply identifies a subset of data in the database without specifying *how* to go about extracting that data. This has led to the pervasive belief that application programmers have no responsibility for SQL performance. In the second installment of this essay, we explored the theoretical underpinnings of the SQL language–an understanding of which is crucial to taking responsibility for SQL performance. In this installment–the last–we consider our options.[1]

The Horseless Carriage

In Part II of this essay, we listed a number of "dangerous" beliefs centered around the theme that application programmers have no responsibility for the efficiency of the SQL statements they write. An example of a belief that we labeled dangerous is the pervasive belief that application programmers may deploy SQL statements without carefully tailoring them to make use of existing indexes and that database administrators should set traps for SQL statements that have been deployed and that are performing badly and determine what new indexes should be created to make them perform better. Of course, I have to admit that I have been as guilty as anyone else in this regard. I am trying hard to break the habit but very old habits die very hard. I have many long years of experience in writing SQL statements that look like "pretzels created by somebody who is experimenting with hallucinogens"[2] but very little in writing SQL that is guaranteed to perform well.

I now see that writing SQL statements without regard to indexes and other access paths is like unyoking the trusty steed from the carriage and trusting that Oracle will find us a speedy stallion when we need one–not the old gray mare who "ain't what she used to be many long years ago." In the pre-relational days of yore, programmers were keenly aware of the available indexes. In fact the indexes were built into the data files in a manner similar to Oracle's Index Organized Tables (IOTs), making it extremely difficult to add new index paths. Here is an example of a COBOL data file that is indexed by EmployeeID and EmployeeLastName.

```
SELECT EmployeeFile
 ASSIGN TO "EMPLOYEE.DAT"
 ORGANIZATION IS INDEXED
 ACCESS MODE IS DYNAMIC
 RECORD KEY IS EmployeeId
 ALTERNATE RECORD KEY IS EmployeeLastName
WITH DUPLICATES.
```

There are only three things you can do with the file described above. You can perform a full scan of the file, you can retrieve the unique record whose EmployeeID value matches the value you specify, and you can retrieve records whose EmployeeLastName value is greater than or equal to a value you specify–we won't bore you with the details.

Fast-forward to the wonderful world of relational databases where programmers have been raised to ignore such niceties as performance. Edgar Codd, the inventor of relational theory, insisted on "Physical Data Independence"–but what exactly did he say? Referring to his magnum opus *The Relational Model of Database Management–Version 2,* we find the following rule in Chapter 20–*Protection of Investment.*

RP-1 Physical Data Independence–The DBMS permits a suitably authorized user to make changes in storage representation, in access method, or in both–for example, for performance reasons. Application programs and terminal activities remain **logically** *[emphasis ours] unimpaired whenever any such changes are made. (This feature is Rule 8 in the 1985 set.)*

In Chapter 21-*Principles of Database Design*, we find the following rule.

RD-10 Changing Storage Representation and Access Options–Commands must be available to the DBA for dynamically changing the storage representation and access method in use for any base relation without causing **logical** [extra emphasis ours] impairment of any transaction in **source code form** [extra emphasis ours] *(whether already compiled or not), or any noticeable*

[1] You can download Part I and Part II of this essay from **www.nocoug.org/presentations.html**.

[2] A colorful description coined by Steven Feuerstein in an interview published in May 2006 in the *NoCOUG Journal.*

delays in the execution of the transactions in progress or of transactions waiting to be processed.

My interpretation of these rules is that Codd would not object if the relational language allows the programmer to explicitly specify an access path (such as an index) as long as the relational language program remains *logically* unimpaired if the access path were to suddenly become unavailable[3]–in fact Codd seems to anticipate that relational languages might allow the programmer to do so. As a historical note, we might add that Codd was not involved with the development of SQL and has many critical things to say about SQL in his book.

All Oracle Hints meet Codd's requirement that an SQL query not be invalidated if an explicitly specified access path suddenly becomes unavailable. The purist in us might object to referring by name to an index that might be dropped at a future point in time and this is no longer necessary in Oracle 10*g* which allows us to specify an indexed access path by listing the *columns* which the index contains instead of the *name* of the index.

A Delicate Question

Does the optimizer need a hint? Experience tells us it often does. In Part I of this essay, we discussed the problematic assumption of attribute independence and the unsatisfactory assumptions that the Oracle optimizer is forced to make to determine how many rows will result from the joining of two tables. Another hard problem for the optimizer to solve is that of determining the equivalence of SQL statements. If two SQL statements are equivalent then the query execution plan that is optimal in the case of the first is also optimal in the case of the second. However, if Oracle cannot recognize their equivalence it is less likely to identify the same plan in both cases. In a 1988 paper,[4] Fabian Pascal wrote a query in seven different ways and checked how long it took Oracle to retrieve the data in each case. These were his results.

ACCESS METHOD	COMPLETION TIME
JOIN	19 seconds
IN1	22 seconds
ANY1	20 seconds
IN2	525 seconds (25X)
ANY2	525 seconds (25X)
EXISTS	525 seconds (25X)
COUNT	1818 seconds (90X)

I repeated the same tests recently and obtained the following results. Completion times were predictably faster because of faster hardware but the variations remained, indicating that the problem remains as hard as ever.

[3] See, however, Rule RD-2 in Chapter 21—*Principles of DBMS Design* read together with Codd's remarks on optimizability in Chapter 26—*Advantages of the Relational Approach.*

[4] **www.dbdebunk.com/page/page/1317920.htm**.

Lies, Damn Lies, and SQL!

In the last issue of the *Journal* we offered a prize of a SanDisk Sansa M240 1 GB MP3 Player for the best solution to the following puzzle sent to us by Sumit Sengupta from Columbus, OH.

Describe a combination of circumstances in which the COUNT function could produce the anomalous results seen in the example below.

```
SQL> DESCRIBE employees;

Name                  Null?     Type
--------------        --------  ---------------
NAME                  NOT NULL  VARCHAR2(25)
SALARY                NOT NULL  NUMBER(8,2)
COMMISSION_PCT        NOT NULL  NUMBER(4,2)

SQL> SELECT COUNT(name) FROM employees;

COUNT(NAME)
-----------
          3

SQL> SELECT COUNT(salary) FROM employees;

COUNT(SALARY)
-------------
          2

SQL> SELECT COUNT(commission_pct) FROM
employees;

COUNT(COMMISSION_PCT)
---------------------
          1

SQL> SELECT COUNT(1) FROM employees;

COUNT(1)
----------
          0
```

Chris Lawson and Roger Schrag submitted correct solutions to the puzzle. Chris Lawson's solution took advantage of Oracle's read consistency scheme and posited that a second session deleted one record (and committed its work) after each of the above queries. Roger Schrag submitted a solution that uses materialized views and does not require coordination with a second session—though it does require that all the rows in the table be deleted. The original solution provided by Sumit Sengupta does not require the deletion of rows and is based on the column level access control provided by Oracle 10*g* R2.

Roger Schrag wins the prize and we will discuss all three solutions in detail in the next installment of the SQL Corner. ▲

ACCESS METHOD	COMPLETION TIME
JOIN	0.0167 seconds
IN1	0.3959 seconds (25X)
ANY1	0.3907 seconds (25X)
IN2	0.4110 seconds (25X)
ANY2	0.4140 seconds (25X)
EXISTS	0.7291 seconds (40X)
COUNT	0.6991 seconds (40X)

The Robust Plan

But doesn't the best choice of query plan depend on the values of the bind variables? Yes, of course it does. However, remember that Oracle caches query plans and reuses them whenever possible, even when the values of the bind variables have changed–the aim being to avoid the overhead of repeated parsing of similar statements. Oracle constructs a query plan using the values submitted in the very first invocation of the SQL statement–this is called "bind variable peeking."[5] Obviously a query plan constructed using one set of bind variables is not necessarily ideal for all other values of the bind variables. In his book *SQL Tuning*, Dan Tow develops the theory of the "robust" execution plan. Such plans have only moderate sensitivity to the values of the bind variables (which implies that they continue to perform well as your tables grow) and their cost is proportional to the number of rows retrieved. They may not be the most efficient plans but they are not very likely to be far off the mark. Such plans are conducive to stability and predictability and are obviously very desirable. Refer to Tow's book for more details.

Hinting to the Max

Having perhaps persuaded the gentle reader that Hints are perhaps not such a bad idea after all, we present a technique that might be called "Hinting to the Max." We assume that the reader is conversant with the subject of Hints and their use and with access methods such as "nested loop join," "merge join," "hash join," "semi join," and "anti join." We use the relatively new technique of "subquery factoring" to achieve the intended results via a series of simple steps. At each step, we use Hints to define the access path we wish Oracle to take. We use the same Supplier-Parts example that we dissected in Part II of this essay. The list of suppliers who supply all parts can be retrieved using the following SQL statement which we provide without further explanation. We don't attempt to justify the access path chosen at each step because this example is only intended to illustrate how a query can be systematically broken up into smaller parts and systematically hinted.

```
with
 AllCombinations as (
   select /*+ NO_MERGE ORDERED FULL(Suppliers)
FULL(Parts) USE_NL(Parts) */
      SupplierName,
      PartName
   from
      Suppliers,
      Parts
 ),

 InvalidCombinations as (
   select /*+ NO_MERGE */
      SupplierName,
      PartName
   from
      AllCombinations
   where (SupplierName, PartName) not in (
      select /*+ INDEX(SuppliedParts
(SupplierName, PartName)) NL_AJ */
         SupplierName,
         PartName
      from
         SuppliedParts
   )
 ),

 UnwantedSuppliers as (
   select /*+ NO_MERGE */
      SupplierName
   from
      InvalidCombinations
 ),

 WantedSuppliers as (
   select /*+ NO_MERGE */
      SupplierName
   from
      Suppliers
   where SupplierName not in (
      select /*+ HASH_AJ */
         SupplierName
      from
         UnwantedSuppliers
   )
 )
select * from WantedSuppliers;
```

[5] Prior to 9*i*, Oracle *ignored* the values of the bind variables when choosing a query plan. In effect, this meant that Oracle ignored any histograms that the database administrator might have created for the tables in question.

Take Home Message

The *promise* of relational technology was that application programmers would be relieved of the responsibility for the efficiency of the SQL statements they write. We have argued that this promise is a *failed promise* and that there are tremendous barriers preventing it from ever being fulfilled. And so, therefore, our "take home message" is that *SQL performance requires conscious effort on the part of the developer*!

Concluding Remarks

I am aware that the positions adopted in this essay are not generally held in favor. I have only recently begun to favor these positions and, therefore, it is very possible that I have overlooked some good arguments both for and against them. I am also aware that the documentation of Oracle Hints is quite sparse, which makes it difficult to learn how to use them. Be as it may, I enjoyed writing this essay and welcome your comments.

Further Reading

Dan Tow's book provides guidance in manually designing a query execution plan. Another excellent book on SQL tuning is *Oracle SQL High Performance Tuning* by Guy Harrison. ▲

The author can be reached at **iggy_fernandez@hotmail.com**.