



CCAPRINT

A Newsletter Excerpt for Model 204 Users

June 2006

USE OF AND ACCESS TO PRODUCTS AND FEATURES ARE IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF THE USER'S SOFTWARE LICENSE. THE PRESENTATION OF MATERIAL HEREIN DOES NOT, IN ANY MANNER, MODIFY SUCH TERMS AND CONDITIONS.

SQL Performance Tuning

By Mark LaRocca

Model 204 provides many tuning parameters for SQL performance. Performance should be monitored and these parameters adjusted as required. So, assume we intend to use SQL fairly heavily and want to assure maximum performance. In the Figure A example, I am running an SQL request that retrieves 1,820 records from a seven million record file.

SQL Parameters on an IODEV 19 Thread Definition

Here are the initial SQL tuning parameters:

- **LHEAP=200000**
This is dynamic storage used by SQL routines.
- **LPDLST=32760**
This is the maximum for this parameter (32K).
- **SERVSZ=700000**
This is the size of the Model 204 server area to allocate. Make sure that several servers are available for SQL users. Otherwise, server swapping occurs that impacts overall performance.
- **SQLBUFSZ=100000**
This SQL buffer must be big enough to contain the largest SQL statement. Remember, some SQL statements are generated, especially those based on views. This means that, although an individual statement appears simple, its underlying generated statement may be much larger.
- **SQLIQBSZ=32752**
This is an internal SQL buffer set to the maximum (32K).

Test Query Against a Seven Million Record File

The query I am running tests performance for an actual Model 204 customer. Of course, I changed the file and field definitions, but the statistics are real.

Figure A. Querying a 7-million record file

```
SELECT *
FROM TEST.INVENTORY
WHERE RECRDTYPE = 'AAA' AND
CREATDDATE BETWEEN '20051101' AND '20051102' AND
CHARGTO < '7000'
```

Coding an Efficient Query

Before we attempt to run this query, we should look at two things in advance to enhance performance.

- Are all columns (Model 204 fields) in the WHERE clause ORDERED?
Columns in a WHERE clause should always be indexed to assure maximum performance.
- Can we query specific columns instead of the generic SELECT * ... ?
Specifying columns reduces catalog lookup time, which can be significant for queries that are run repetitively.

Based on the previous two points, I have determined that the columns in the WHERE clause are ordered, and I have found that I can change our query to retrieve specific columns as shown in Figure B.

Figure B. Querying specific columns

```
SELECT Col1, Col2, Col3, Col4, Col5
FROM TEST.INVENTORY
WHERE RECRDTYPE = 'AAA' AND
CREATDDATE BETWEEN '20051101' AND '20051102' AND
CHARGTO < '7000'
```

Now, let's run our query.

Identifying Performance Problems

To our surprise, despite our tuning efforts, the query takes about 30 seconds to retrieve 1820 rows of data. This is far below expected Model 204 performance. Why? Let's look at a short checklist of possible performance issues.

- Server swapping
- Disk I/O
File access or when was the last time the file was reorganized?
- Client/server application
Do we connect and disconnect (Logon and Logoff) for every request? Is our fetch size set too high?

We know that each of the possible issues can affect performance. But, how do we determine which, if any, of the possible issues are the problem. The answer is LAUDIT.

Set LAUDIT=255 on your IODEV 19 threads. LAUDIT processing displays on the audit trail all information concerning the query you are executing. Also, to audit RK lines, set the X'20' bit on for SYSOPT.

Now that we have these set in our Online, try the query again and take a look at some of the performance statistics in CCAUDIT.

Remember, an SQL query is translated to IFAM code and, as such, is really a User Language request. In CCAUDIT you will see the SELECT statement and the User Language code into which it was translated, followed by the FIND statistics and finally a long list of S2 lines.

These lines might say S3 or S4, but usually, on a first request, they are S2. These are the actual fetches for each row; in our example there are 1,820 of these S2 lines.

Now, how do we spot the problem? First, look for server swapping. Does the request begin and end running in the same server? Look for the IODEV 19 login to spot the beginning. Figure C is the request output--edited for space--showing the points of interest.

Figure C. Request output

```

06097110549 2 5 17 AD /// M204.0352: IODEV=19, OK MARK MARK LOG
06097110549 3 5 17 MS *** M204.0353: MARK MARK LOGIN 06 A
06097110549 4 5 17 MS *** M204.2566: LINK=LINKTCP, LOCALID=192.207.28.
...

06097110549 0 5 17 QT PREPARE statement SQL_STMT_1_ (SELECT Col1, Col2, Col3,
06097110549 1 5 17 XX Col4, Col5 FROM TEST.INVENTORY WHERE ...

...

06097110550 6 5 17 LI IN INVPOCS FD 'CREATED.DATE' IS ALPHA IN RANGE FROM '
06097110550 7 5 17 XX I' AND 'CHARGE.TO' IS ALPHA < '7000'

...

06097110550 29 5 17 LI S0
06097110616 0 5 17 RK ... M204.0880: IFFIND COMPLETE
06097110616 1 5 17 RK ... M204.0881: IFCOUNT = 1820
06097110616 2 5 17 LI S1
06097110616 3 5 17 ST $$$ USERID='CCA06 ' ACCOUNT='CCA06 ' LAST='EX
06097110616 4 5 17 XX PROC='SQL_STMT_1_' QTBL=158 STBL=60 TTBL=6 VTBL=111 P
06097110616 5 5 17 XX 6 DKRD=4845 IN=2 FINDS=1 PCPU=18 RQTM=26466 BXNEXT=11
06097110616 6 5 17 LI S2

...

06097110616 7 5 17 LI S2
06097110616 8 5 17 LI S2
06097110616 9 5 17 LI S2
06097110616 10 5 17 LI S2
06097110616 11 5 17 LI S2

...

06097110620106 5 17 LI S2
06097110620107 5 17 LI S2
06097110620108 5 17 LI S2
06097110620109 5 17 LI S2
06097110620110 5 17 LI S1
06097110620111 5 17 LI IN S0
06097110620112 5 17 QT CLOSE cursor SQL_CUR_1_ on statement SQL_STMT_1_
06097110620113 5 17 QT COMMIT -- all cursors closed
06097110620114 5 17 QT DROP statement SQL_STMT_1_
06097110620115 5 17 RK ... M204.0889: IFFLUSH
06097110620116 5 17 LI EU2,S0,S1,S2

```

Analyzing the Audit Trail

A quick look at the audit shows that the entire transaction took about 30 seconds, starting at 11:05:49 and ending (close cursor) around 11:06:20. We can see that the whole request ran in server 5. Examining the entire audit, we see that no other user took over server 5 in the middle of our request. So, there was no server swapping. Server swapping might mean that we have only one 700,000 size server and users are competing for it. We would then need to provide more servers of this size to reduce the possibility of server swapping.

Our next concern is the FIND processing. We can see from the previous code that the FIND started at 11:05:50 (see S0 line after Prepare lines) and ended (IFFIND COMPLETE) at 11:06:16, covering a total of 26 seconds. This is an indicator that, perhaps, our file needs to be reorganized. Take a look at the FIND statistics.

```
06097110616 5 5 17 XX 6 DKRD=4845 IN=2 FINDS=1 PCPU=18 RQTM=26466 BXNEXT=11
```

DKRD=4845 is the number of disk reads and RQTM=26466 is the total elapsed time in milliseconds for the request to this point. This means we are taking about five milliseconds per read. This is high because the first time we read, the records are not in disk cache and also not in Model 204 internal buffers.

Let's run the request again to test our disk cache theory. Running the same request again shows that the total transaction time was reduced from 30 seconds to 11 seconds. I have not provided the audit here, but you can run such I/O tests for any User Language request or SQL query and see the same result. Our speed increases dramatically. However, it is still too slow to meet our high Model 204 standards. The solution is: reorganize the file.

Here is the same DKRD and RQTM for an initial request not in disk cache once the file has been reorganized.

```
XX DKRD=1592 IN=2 FINDS=1 PCPU=50 RQTM=6137 BXNEXT=935 ...
```

As you can see, after the reorganization, DKRD dropped from 4845 to 1592. This is because the reorganization resulted in a vast improvement in Table B and Table D utilization, ordered index organization, reduced extension records, and so on. The data required for this request now resides on significantly fewer pages resulting in a 67% decrease in database I/O.

Secondly, notice that the RQTM dropped from 26 to six seconds. Certainly that file needed to be reorganized. (If fact, this file had not been reorganized for years.) Thus, CCA recommends that you reorganize frequently updated files once a week or minimally once a month.

Now, run the request again. At this point all rows are already in disk cache. We see the following.

```
XX DKRD=1592 IN=2 FINDS=1 PCPU=50 RQTM=3137 BXNEXT=935 ...
```

DKRDs remained the same, but actual request time decreased to 3137 milliseconds (or three seconds). However, should we expect even faster results on a FIND? Should the records already be in Model 204 storage? Yes, but they were not, because we had set MAXBUF=1500 for our Online and they were probably flushed out. So, set MAXBUF=30000 and rerun the same request. We see the following.

```
06097120827101 5 17 XX SLIC=1 IN=2 FINDS=1 PCPU=708 RQTM=264
```

Request time (RQTM=264ms) is virtually non-existent at 0.2 seconds. There is no DKRD statistic listed, because all records requested were already in Model 204 storage. The lesson of the example is:

- Reorganize files often.
- Set the MAXBUF parameter high for best performance.

But what about our final overall transaction time? It dropped from an initial 30 seconds to 11 seconds for a reorganized file to two to three seconds for rows in disk cache or Model 204 cache.

This brings us to our final action: tuning the client/server application.

Tuning the Client/Server Application

This particular test used a Java application using Connect★'s JDBC for Model 204. Model 204 fetch size defaults to whatever fits in a 4K buffer. In our test, this was 22 rows. To fetch 1820 rows, we must ask the server, Model 204, to send us a new buffer about 90 times. This conversation overhead is very high. I did not show it in the previous statistics, but increasing the fetch size speeds things up four to five times. To do this in Java requires a statement such as:

```
state.setFetchSize(25000);
```

This would fetch all rows of our request without requiring the JDBC driver to ask for more. In the previous test, all requests were run with this setting.

Also, if we run an initial request and stay connected for more requests, this eliminates LOGON/LOGOFF processing on Model 204--about one second--and also minimizes the chance that data is flushed from Model 204 buffers.

Smart client/server applications:

- Set a suitable fetch size for the application.
- Minimize connect and disconnect (Logon and Logoff) time.

In Summary

To optimize SQL performance, tuning is the key, so:

- Eliminate server swapping.
- Reorganize the file.
This provided our greatest performance gain, about 18 seconds.
- Set MAXBUF high
Most Onlines need a minimum of MAXBUF=20000.
- Use ordered fields in your WHERE clause.
- Tune the client/server application.
Set fetch size sufficiently high to minimize connect and disconnect.

© 2006 Computer Corporation of America
500 Old Connecticut Path, Framingham, MA 01701