Rdb Performance Tuning

A Methodology

Abstract

The process of tuning Oracle Rdb databases has numerous elements.

This presentation will cover the highlights of Rdb tuning and present a map of the tuning process used by JCC consultants. This tuning process utilizes three major threads:

- ❑ The physical database and transaction design.
- ❑ Memory Management.
- ❑ Tuning of individual queries.

The presentation will outline, in a general way, the rules of thumb used to accomplish physical tuning. It will also address the use of the JCC BLR parser and metrics methodology. (A license for which is also included in the "Database Performance and Tuning: Query Tuning and Locking" seminar.)

What Constitutes Transaction Design

- Transaction design describes the context of transactions.
- Dimensions:
  - Transaction isolation levels.
    - Reduced isolation levels (repeatable read, read committed) cause the Rdb engine to release locks during the course of read write transactions.
  - Snapshot control.
    - Disabled.
    - Enabled deferred.
    - Enabled (the default.)
- Transaction Processing Monitor or direct access.

Isolation Levels and Transactions

- Reduced isolation levels can allow much greater update activity to a database.
  - Provided the flow of transactions is *controlled* – defer updates as late as possible in the transaction.
    - Shorter lock durations imply less conflict.
  - Provide the potential for side effects.
    - Most tools today are exposed to side effects anyway, they read data first in one transaction and then write in another.
  - Most other databases usually operate at reduced isolation levels.

Eliminate Read Only Transactions

- For update intensive applications.
- To support read-only transactions the Rdb engine must maintain transaction start and end context in the root file.
  - Current releases relax this somewhat if the database is accessed by only <u>one node</u> in a cluster.
- This support requires *at least* one I/O per transaction to the root file and can be extremely limiting in throughput as the root file becomes the busiest file in the computer.

With No Read-Only Transactions

- Can mark the database as snapshots disabled or enabled deferred.
  - Eliminate writes to the snapshot files.
  - Backup runs as a read-only transaction and so snapshots deferred is probably preferable.

Other Performance Enhancers

- These are appropriate *only* for update intensive applications.
- Fast Commit.
  - Required for row cache, anyway.
  - Expect somewhat greater DBA overhead to ensure that processes checkpoint frequently.
    - Use periodic forced checkpoints.
- Commit to Journal.
  - Misnamed feature.
  - Allocates TSNs in bunches. Further reduces I/O to root.
  - Must *not* be performing snapshot transactions.

Fastest Update Transaction Model

- Database open on one node.
- Read committed transactions.
- Snapshots enabled deferred or disabled.
- Commit to journal.
- Fast commit.

- Not suitable for all applications.

Physical Database Design

- What to do depends on the application.
  - Lots of random access and update of small numbers of rows – Use hash indexing and placement.
    - Generates real DBA management problems because all databases seem to grow over time.
  - Range queries of any kind will imply the use of sorted indexes.
    - Most reports require range queries.
    - Partial matches to keys imply range queries.
    - Spending a lot of time on node sizing tends to be a waste. We use 1000 byte nodes almost all the time. (See later as to why this number.)
    - Probably will defeat the utility of hashed indexes.
  - Synthetic chronological key values are almost certain to dictate hashed techniques.
    - Applications can often do their own hashing with synthetic keys and therefore dodge that requirement.
    - JCC has packaged that into a kit. Rdb 7.1 has packaged that into sequences.
  - The presence of sorted indexes can lead to significant lock contention, even with read committed transactions.

Partitioned Designs

- Whenever concurrent updates are being done, you should consider how to partition the *data and servers managing that data*.
  - Each update process gets its own partition.
  - No update process will "see" another and no lock conflicts will be experienced.
- The presence of more than one sorted index will probably render partitioning less satisfactory.
  - It is not likely that all indexes can be partitioned similarly to the tables.

Design of Storage Areas

- We pick big page sizes for tables that will have high cardinality or be used for hash placement.
  - Fewer SPAM pages in uniform areas, faster inserts.
    - More SPAM contention, though.
  - The Rdb hashing algorithms do better at distributing data *evenly* when the number of objects on a page are large.
    - Areas containing only hashed indexes inherently have lots of objects on a page.

Design of Storage Areas

- We generally place each index in its own area.
- We generally place each table in its own area.
- Only place hash index in the same area as another hash index if it is on exactly the same column(s) for different tables.
- This results in lots of storage areas.
  - Some implications for backup performance.
  - Allows assessment of where costs are in application.

What Kinds of Indexes

- If the hash key is numeric, use ordered hash.
  - If the key values are not biased number of pages should be determined by storage needs.
  - Otherwise prime number for initial size of hash area.
- If have a sorted index, tend to use sorted ranked indexes.
  - If a unique index on one column may use regular sorted indexes.

How Many Indexes?

- Only relevant to the indexes on a single table.
  - Larger numbers of indexes tend to increase the complexity of transactions.
  - Concurrency problems increase as the 4th power of complexity!
- Negotiate with the application developers.
- Don’t forget that the query optimizer can do “and” optimization using multiple indexes.
  - Thereby reduce rows checked for a query.

Shorter Transactions

- Shorter transactions lead to less lock conflict.
- What to do about large transactions, e.g. billing:
  - Commit this work more frequently.
  - Use Holdable cursors.
    - Got to be careful of side effects of these cursors.
  - Use high water tables.
    - Write current position at each commit.
  - Make commit intervals a run-time parameter.

Memory Management

- If multiple processes access the same database objects you should *not* be using private buffers for the database.
- Two alternatives which are compatible.
  - Global Buffers.
  - Row Caches.

Global Buffers

- Provide a large pool of memory commonly accessible in computer.
- Large global buffer pools (e.g. 500 Megabytes, 1+ Gb) are not manageable from a VMS context.
  - *Unless* you use system space global buffers.
  - Requires a compromise between Rdb tuning and VMS tuning.
    - NPAGVIR should be drastically reduced from the gross values generated by SYSGEN: 4*NPAGDYN.

Global Buffers

- Very few databases do not benefit from the use of global buffers.
- It is, by default, a part of our design.

Row Caches

- Row caches are a separate mechanism for storing data in memory.
  - Checked before pages are accessed, leading to reduced database locking activity.
  - Allow deployment of > 2^{31} bytes of memory, gigabytes and gigabytes.
  - Reduces database reads for objects already read.
- Very useful for sorted indexes and rows.
- Marginally useful for large hash indexes and placed rows.
  - By assumption these are random anyway, so repeated access is not occurring.

Row Caches

- Require two things:
- Fast commit with its implications for DBA overhead to ensure good checkpoint behavior.
- The database open on one node in a cluster or on a Galaxy (Rdb 7.1).
- Quite frequently we have found that Rdb remote access to a node running row cache will result in (much) greater transaction throughput than opening on multiple nodes.

Default Memory Deployment

- By default we assume large memory systems.
- Row Caches.
  - For a limited number of tables.
  - All sorted indexes go into 1 cache.
    - 1000 byte line size for good mapping of memory.
    - Larger node size implies more data to AIJ
- Global buffers (about 1^+ Gb if possible.)
  - For 'stuff' that row caches cannot handle such as SPAM or AIP or snapshot pages.

Index Tuning

- Hashed indexes are useful only for exact match queries.
  - Predicate must specify data values for *all* columns in the index key.
  - If duplicates are permitted and the cardinality of average duplicate values is high (> 50 or 100) then this may lead to a potentially poor result.
    - Don't worry too much about 1 or a few key values with extreme cardinality. But removing rows with those key values will tend to be slower.
  - Two hashed indexes with the same key prefixes are undesirable.
    - Too much overhead in maintenance.
    - (column 1, column 2, column 3) (eliminate this one)
    - (column 1, column 2)

Sorted Index Definition

- Sorted indexes is where the real tuning can come in.
  - Especially for joins and range queries.
- Design sorted indexes to leverage on the strategies in the Rdb optimizer:
  - Partial key matches.
  - Provide data in sorted orders.
  - Zigzag query solutions.
  - Index-only strategies.
- Add columns to allow restrictions in index without reading the data rows.

Sorted Index Design

- Tuning sorted indexes well requires knowledge of the queries being run against the database.
  - Often one will trade off the performance of one query for another to avoid introducing another index.
  - Some of these are more important than others.
- The queries applied to a database can change over time.
  - Indexes can become obsolete

Query Capture Methodology

- Always define two standard Rdb Logical Names
- RDMS$DEBUG_FLAGS defined to PBSsO
  - P: Records database bind information
  - Ss: Records strategy used for query together with template query outline
  - O: Records optimizer estimated costs
  - B: Records BLR describing query
- Alternatively use the set_flags logical name

Query Capture

- P: Records database bind information
- Determines which database is used
- Which machine the application is run on
- Which program [.EXE] is accessing the database, including full path name and version
- What the database bind sequence number is
  - Can detect programs that bind to the database more than once.

Capturing Run-time Information

- S: Records strategy used for query
- Necessary information for database tuner
- Report requires interpretation
  - Key to determining what the optimizer is doing
- The form “Ss” causes the optimizer to also emit a prototype query outline

Query Strategy

- This is a sample of a reported strategy.

```
Firstn  Conjunct        Index only retrieval of relation ACCOUNT
  Index name  ACCOUNT_S_03 [1:0,2:1,3:2,4:3,5:4]
```

Query Outlines

- Query outlines are the key to controlling the behavior of the optimizer.
  - If the optimizer won't do it your way.
- In most circumstances, few outlines will be required.
- Sometimes they are necessary.
- Require that either queries be named or that you obtain the hashed key of the outline.
  - Supplied by Rdb when the query is optimized.
- If an outline is used by the optimizer this fact is recorded in the strategy reported.

Capturing Rdb Run-time Information

- B: Records BLR describing query.
- BLR is a structured form of the query.
- The actual form of BLR will vary depending on how the initial compiler did its work.
  - SQL may differ from some application-generated BLR.
- Can be very complicated due to the inherently complex form that SQL can be written in.
  - Correlated sub-queries.
  - Derived tables.
  - Joins.
  - Unions.
  - Disjunctive forms.

BLR for Queries

Simple BLR has several separate sections

- A prefix describing the data to be interchanged between the Rdb engine and the application.
  - Not very interesting for this work.
  - Can be mostly ignored.
- A statement of the tables that are queried.
- The predicate.
- Sort order.
- Functions performed on columns.
- Subqueries.

Analysis of Query Solution

- Interpreting long runs of BLR is quite tedious.
  - Can often be confusing especially if there are correlated sub queries , joins, unions, complex predicates etc.
  - Cannot afford the time per query to reverse translate to SQL.
- Having the BLR and strategy together allows one to validate the query solution used by the optimizer.
- Need to have some automatic ability to translate BLR back to SQL.
  - Can be satisfied with 98% translation effectiveness.
  - Exclude complex SQL programs masked as SQL procedures.
  - Should be "C" code to allow porting to platforms other than VMS.

Storing the Run-time Information

- We have been calling the output files from the optimizer, Rdb "Statistics" files.
- Statistics files all are placed in a known directory.
  - Sometimes several from the same procedure each day.
  - Can become numerous.
  - If required, they could be distributed to several directories.
- Statistics files need to be processed in temporal order by program or procedure.
- Results of processing are placed in an Rdb database
  - Convenient because we know Rdb well.
  - Allows us to use SQL procedures to maintain and process the database.

Metrics Database

- The database containing the parsed queries is called the "Metrics" database.
- Also is used to store other production database data.
  - Logical and physical areas.
  - Area sizing and utilization.
  - Locking and deadlocking information.
  - Table cardinality history.
  - Whatever the DBA finds convenient.
- Can be accessed from remote computers via:
  - Rdb Remote.
  - ODBC.

Harvesting the Crop

- All statistics files are stored in one directory.
- Moved nightly to development environment.
- Processed via DCL running parser nightly.
- New and *changed* queries inserted into metrics database.
- Processed files are renamed into a [.processed] subdirectory where they are retained for 5 versions
  - Allows the DBA some retrospective ability.
  - Retention not based on date because some subsystems may not be in production yet and programs haven't run recently.

Processing Daily Files

- After the new queries are added there is an automatic scan:
  - Runs in SQL.
  - Any queries which duplicate [same BLR & optimization] existing approved queries are approved. This is due to subprograms being used by several programs.
  - Any queries which duplicate existing unapproved queries are marked duplicate.
  - Result is that only a few unapproved queries must be inspected.
- Counts of unapproved queries are produced by SQL and mailed nightly to the DBA.
  - Always know where you stand at the start of the day.

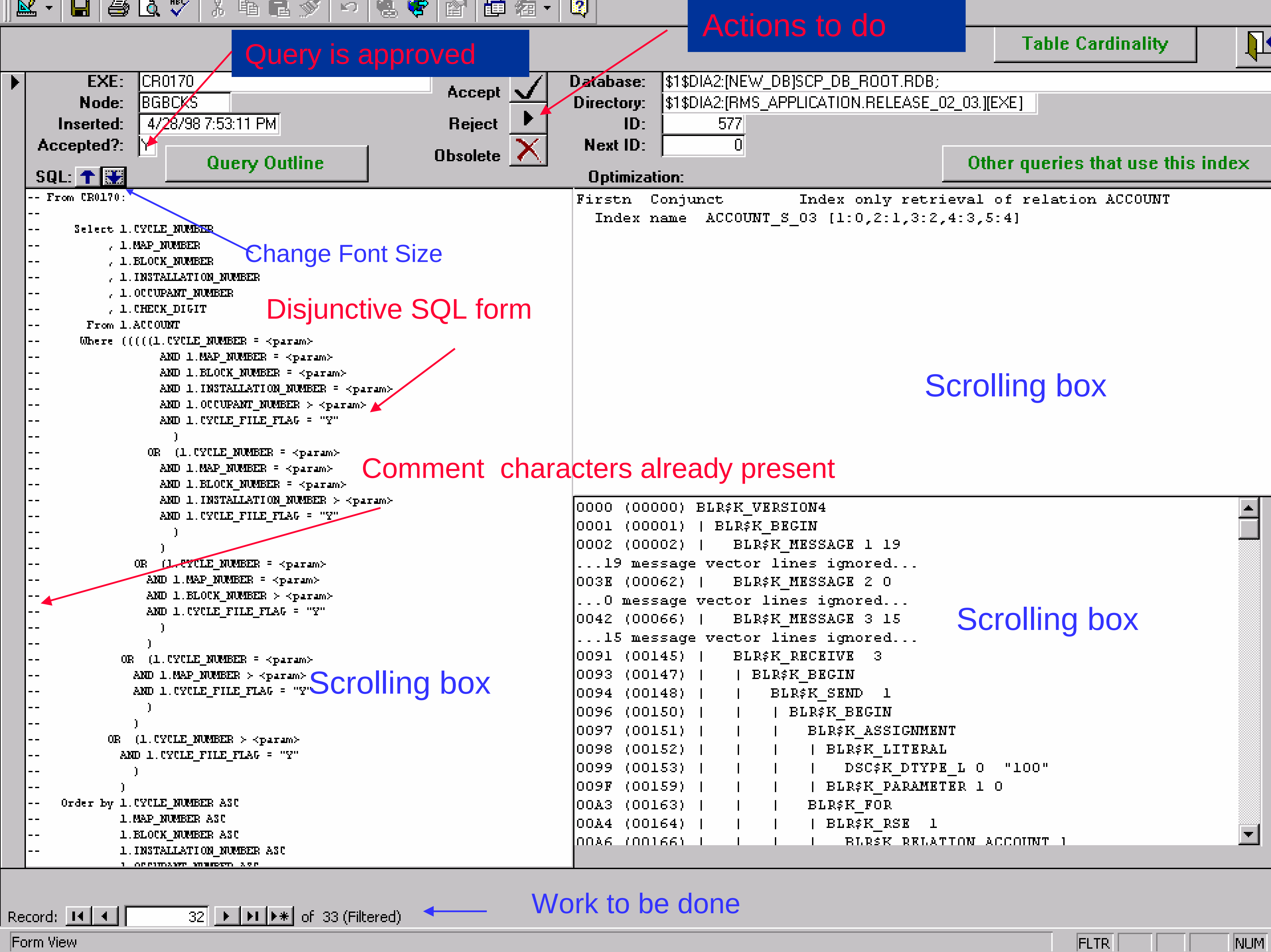
Reports on Queries

- By joining data from various query tables we can report which indexes have been used recently.
- Which indexes have not been used recently.
- Which programs use which index.
- Which index is used by which program.

- You name it.

GUI Interface to Metrics DB

- Given an ODBC connection to the database, a simple Microsoft Access application provides an easy GUI front-end to the metrics database.
- Menu driven.
- Allows pop-up windows.
- Allows scrolling boxes to support the generous text column sizes
- Passes through most of the work to Rdb and so is reasonably efficient.
- Produces acceptable reports.
- Could be constructed with any tool that supports ODBC.



EXE:	CR0170
Node:	BGBCKS
Inserted:	4/28/98 7:53:11 PM
Accepted?:	Y
Database:	$1$DIA2:[NEW_DB]SCP_DB_ROOT.RDB;
Directory:	$1$DIA2:[RMS_APPLICATION.RELEASE_02_03.][EXE]
ID:	577
Next ID:	0

SQL:

```
-- From CR0170:
--
--      Select 1.CYCLE_NUMBER
--           , 1.MAP_NUMBER
--           , 1.BLOCK_NUMBER
--           , 1.INSTALLATION_NUMBER
--           , 1.OCCUPANT_NUMBER
--           , 1.CHECK_DIGIT
--        From 1.ACCOUNT
--       Where (((((1.CYCLE_NUMBER = <param>
--                  AND 1.MAP_NUMBER = <param>
--                  AND 1.BLOCK_NUMBER = <param>
--                  AND 1.INSTALLATION_NUMBER = <param>
--                  AND 1.OCCUPANT_NUMBER > <param>
--                  AND 1.CYCLE_FILE_FLAG = "Y"
--                    )
--                OR  (1.CYCLE_NUMBER = <param>
--                  AND 1.MAP_NUMBER = <param>
--                  AND 1.BLOCK_NUMBER = <param>
--                  AND 1.INSTALLATION_NUMBER > <param>
--                  AND 1.CYCLE_FILE_FLAG = "Y"
--                    )
--                  )
--             OR  (1.CYCLE_NUMBER = <param>
--                AND 1.MAP_NUMBER = <param>
--                AND 1.BLOCK_NUMBER > <param>
--                AND 1.CYCLE_FILE_FLAG = "Y"
--                  )
--                )
--           OR  (1.CYCLE_NUMBER = <param>
--              AND 1.MAP_NUMBER > <param>
--              AND 1.CYCLE_FILE_FLAG = "Y"
--                )
--              )
--         OR  (1.CYCLE_NUMBER > <param>
--            AND 1.CYCLE_FILE_FLAG = "Y"
--              )
--            )
--    Order by 1.CYCLE_NUMBER ASC
--             1.MAP_NUMBER ASC
--             1.BLOCK_NUMBER ASC
--             1.INSTALLATION_NUMBER ASC
```

Optimization:

```
Firstn  Conjunct        Index only retrieval of relation ACCOUNT
  Index name  ACCOUNT_S_03 [1:0,2:1,3:2,4:3,5:4]
```

```
0000 (00000) BLR$K_VERSION4
0001 (00001) | BLR$K_BEGIN
0002 (00002) |   BLR$K_MESSAGE 1 19
...19 message vector lines ignored...
003E (00062) |   BLR$K_MESSAGE 2 0
...0 message vector lines ignored...
0042 (00066) |   BLR$K_MESSAGE 3 15
...15 message vector lines ignored...
0091 (00145) |   BLR$K_RECEIVE  3
0093 (00147) |   | BLR$K_BEGIN
0094 (00148) |   |   BLR$K_SEND  1
0096 (00150) |   |   | BLR$K_BEGIN
0097 (00151) |   |   |   BLR$K_ASSIGNMENT
0098 (00152) |   |   |   | BLR$K_LITERAL
0099 (00153) |   |   |   |   DSC$K_DTYPE_L 0  "100"
009F (00159) |   |   |   | BLR$K_PARAMETER 1 0
00A3 (00163) |   |   |   BLR$K_FOR
00A4 (00164) |   |   |   | BLR$K_RSE  1
```

Record: 32 of 33 (Filtered)

The Parser Handles Complex Queries

- Record held by complicated query:
  - 24 table join.
  - 4 subqueries.
- Much too large for a Power Point slide.
- A 4 foot printout.
- The optimizer got this one right.

Queries Changing Strategy

- If the strategy changes for a query, then that appears as a new version of an existing query.
  - Previous query_id column is filled in.
- DBA work profile starts the day with MAIL, then a quick click on the unapproved queries button.
  - Unapproved queries are detected immediately by this process.
  - DBA may elect to approve the query.
    or
    to modify the database.

Queries Changing Strategy

- After review of the query strategy the DBA may elect to introduce or change an index.
- The base scripts are to be edited.
- Document base scripts by copying the query being solved directly to the script from the Access query window pane.
  - Comment characters are embedded into the text to facilitate this.
  - Together with the program name.
  - Supplies proper documentation as to why this change was introduced.

Index Usage

- Simple lists of which programs use which indexes can help the query tuner do his or her work.
- Can be prepared via simple Access reports
  - Or via DCL & SQL.
- Even looks good on paper, unlike server reports.
- Can be changed easily.
- Can be augmented easily.
- Report restricts to queries since a certain date.
  - Skip obsolete code.
- And limits to a single VMS node.
  - Allows restriction to a single production system.

Index Obsolescence

- Monthly SQL procedures are run at the end of each month to determine which indexes have not been accessed in 30, 60 and 90 days
- Reports written to log files using the “Trace” command.
- Log files mailed to database administrator(s).
- These reports form a list of indexes that are candidates for deletion from the database.
  - Review via indexes report.
  - Review with developers.

Where to Get These Tools

- The BLR parser and the Access database require significant understanding of the Rdb optimizer.
- Available with JCC consulting.
- A license is provided to students who complete the JCC-offered "Database Performance and Tuning: Query Tuning and Locking" course.

Join the International Rdb Discussion Group

Send Email to: oraclerdb-request@jcc.com

In body of message type:

Subscribe OracleRdb (not case sensitive)

Questions

Contact:
jeff@jcc.com