

A Methodology for Managing Rdb Performance

Jeffrey S. Jalbert, JCC Consulting, Inc.

Thomas H. Musson, JCC Consulting, Inc.

7/18/99

Copyright 1998, 1999, JCC Consulting, Inc., All rights reserved.

Confidential and proprietary to JCC Consulting, Inc.

Introduction

The performance of an Rdb database is dependent on many things:

- Query execution times
- Update speed
 - Concurrency issues
 - Numbers of Indexes
 - Physical Health of storage areas
- Overall quality of database storage areas
- I/O performance of system
 - Which is strongly affected by the memory model for the database

DBA Workload

- In an active database, even if it was performing correctly, these issues require attention over time:
 - Developers change the application
 - Users change the data they store
 - Data volumes increase
 - Changed equipment suggests alternative models
- Automated support for detecting and managing database issues is needed

Requirements For Automated Tools

- Single point of management for all database run-time parameters
- Detect all queries in application and status of “tuning” of those queries
 - Detect new queries not seen before
 - Maintain a list of all queries and their optimizations
 - Detect any queries whose optimizations change
 - Detect differences between production and development
 - Support analysis for indexes that are unused in practice

Requirements for Automated Tools

- Detect locking by table or index
 - Deadlocks
 - Stalls
- Detect any changes to the physical database
 - Storage areas extend
 - Empty space in storage area drops below thresholds
 - Some change to the metadata
 - Data Fragmentation
 - TSN availability [not really a problem any more]
 - Index depth

Requirements For Automated Tools

- Requires a centralized data store about the application
 - Query-able
 - Update-able
- Reasonable ad-hoc tool to manage interface to data store
 - Screens
 - Reports

Requirements For Automated Tools

- All background work must be performed with minimal additional tools
 - DCL Procedures
 - VMS Mail
 - Rdb database
 - SQL procedures
 - Batch or Scheduler job control
- Special program only for unusual work
 - Parse BLR back into SQL
 - Written in “C” for widest operating system support

Single Point Management

- A single indexed RMS file is used to store the values for all Rdb logical names for all procedures
 - Provides central control of all Rdb run-time optimizations
- All application procedures provide an interface to this indexed file
 - ACMS programs supply an initialization procedure written in code
 - DCL procedures call DBA-supplied DCL routine to access parameters file.
- Each procedure supplies a single unique identifying argument when calling parameter setup functions

Single Point Management

- One record in parameter file per application procedure
- File is indexed for fast lookup of values
- Default record supplies values for application procedures that are not pre-defined

Single Point Management

- Standard DCL interface to edit indexed file
- Usual functions for Add, Modify, Query, Delete and Report on contents
 - Managed as a DCL terminal interface
 - Menu driven
 - Automatic entry of changed file into CMS
 - In sequential format for better change control

Management Main Menu

2-NOV-1998 09:02

Maintain Rdb Logical Names

- 1) Edit a current entry
- 2) Add a new entry
- 3) Delete a current entry
- 4) View a current entry
- 5) List all entries
- 6) Exit

Option [Exit]:5

Displaying Entries

2-NOV-1998 09:05

Maintain Rdb Logical Names

List:

DBA_TEMPLATE

INTERACTIVE

NETWORK

OTHER

UTILITY

Press <return> to continue...

Viewing A Single Entry

2-NOV-1998 09:08

View:

Maintain Rdb Logical Names

Entry Name: DBA_TEMPLATE

```
RDM$BIND_BUFFERS = "200"  
RDM$BIND_CKPT_TRANS_INTERVAL = "500"  
RDM$BIND_RUJ_EXTEND_BLKCNT = "9999"  
RDM$BIND_VM_SEGMENT = "1"  
RDMS$AUTO_READY = "1"  
RDMS$BIND_OUTLINE_FLAGS = "USE"  
RDMS$BIND_OUTLINE_MODE = "0"  
RDM$BIND_SORT_WORKFILES = "2"  
SORTWORK0 = "RDB_WORK0:[SORTWORK]"  
SORTWORK1 = "RDB_WORK1:[SORTWORK]"  
RDMS$BIND_WORK_FILE = "RDB_WORK2:[WORK]"  
RDMS$BIND_WORK_VM = "65535"  
RDMS$DEBUG_FLAGS = "PBSOTE30Ss"  
RDMS$RUJ = "RDB_WORK:[RUJ]"
```

Press <return> to continue...

Editing Entries

- Allows modification of existing entries
 - No error checking, can define anything
- Can delete a logical name
- Can create a logical name
- Always have the two `DEBUG_FLAGS` logical names defined
 - Can augment definitions in scripts
 - Provides the key to determining what queries are run
 - Little run-time overhead
 - Resulting files are manageable in size

Capturing Rdb Run-time Information

- Always define two standard Rdb Logical Names
- RDMS\$DEBUG_FLAGS defined to PSsOBTE
 - 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
 - T: Records transaction statistics. This flag is optional, especially for high-transaction rate environments.
 - E: Provides an execution trace for later analysis of the dynamic query optimization

Capturing Rdb Run-time Information

- RDMS\$DEBUG_FLAGS_OUTPUT defined to be metrics_store:<Application_parameter>.statistics
 - Application parameter is what is passed in to the logical names assignment procedure
- Directory is processed nightly to determine which application routines have run

Capturing Rdb Run-time Information

- 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.

Database Bind Information

```
ATTACH #1, Database $1$DIA2:[NEW_DB]DB_ROOT.RDB;1
~P Database Parameter Buffer (version=2, len=86)
0000 (00000) RDB$K_DPB_VERSION2
0001 (00001) RDB$K_FACILITY_ALL
0002 (00002) RDB$K_DPB2_IMAGE_NAME
    "BGBCKS::$1$DIA2:[RMS_APPLICATION.REL_02_04.][EXE]CR0170.EXE;1"
0047 (00071) RDB$K_FACILITY_ALL
0048 (00072) RDB$K_DPB2_DBKEY_SCOPE (Attach)
004C (00076) RDB$K_FACILITY_ALL
004D (00077) RDB$K_DPB2_REQUEST_SCOPE (Attach)
0051 (00081) RDB$K_FACILITY_RDB_VMS
0052 (00082) RDB$K_DPB2_CDD_MAINTAINED (No)
RDMS$BIND_WORK_FILE =
    "USER_ROOT:[WORK]RDMSTTBL$ZLCLET45NIH.TMP;" (Visible = 0)
```

Capturing Rdb 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

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 Outline

- Query outlines are the key to controlling the behavior of the optimizer
- 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

Query Outline

```
-- Rdb Generated Outline : 31-OCT-1998 16:00
create outline QO_19BBC6395969C335_00000000
id '19BBC6395969C33575007DAF90162C76'
mode 0
as (
  query (
-- For loop
    subquery (
      ACCOUNT 0          access path index          ACCOUNT_S_03
    )
  )
)
compliance optional    ;
```

Capturing Rdb Run-time Information

- O: Records optimizer estimated costs
- Not generally useful as an exact predictor of actual run-time costs
- Useful when comparing costs of alternative solutions
 - Available to database tuner when working on individual queries
- Will indicate whether physical database statistics have been collected [RMU/Analyze workload]
 - Table and index cardinalities
 - Null counts

Estimated Costs

```
Solutions tried 4
Solutions blocks created 2
Created solutions pruned 1
Cost of the chosen solution 1.5871724E+03
Cardinality of chosen solution 1.00000000E+00
~0: Physical statistics used
```


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 Query

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

Sample Query Analysis

- The following slides describe the BLR for one sample query
 - The optimization was presented earlier
- The code was looking for accounts in the database.
 - Sites are identified by (cycle_number, map_number, block_number and installation_number)
 - Successive accounts at the same site have serially increasing occupant_numbers, want to scroll through the data in order by sequence above

BLR for Selected Query

```
00A4 (00164) | | | | BLR$K_RSE 1
00A6 (00166) | | | | BLR$K_RELATION ACCOUNT 1
00B0 (00176) | | | | BLR$K_FIRST
00B1 (00177) | | | | | BLR$K_LITERAL
00B2 (00178) | | | | | DSC$K_DTYPE_W 0 "1"
```

- Table aliases are reduced to numeric values
- This query has a limit to 1 row

BLR for Selected Query

```
00B6 (00182) | | | | BLR$K_BOOLEAN
00B7 (00183) | | | | | BLR$K_OR
00B8 (00184) | | | | | BLR$K_OR
00B9 (00185) | | | | | | BLR$K_OR
00BA (00186) | | | | | | BLR$K_OR
```

- There are 5 disjunctive clauses in this query
 - The 4 “OR” disjuncts connect 5 clauses
- We will display only the first one in the following slides

First Disjunctive Clause

00BB (00187)						BLR\$K_AND
00BC (00188)						BLR\$K_AND
00BD (00189)						BLR\$K_AND
00BE (00190)						BLR\$K_AND
00BF (00191)						BLR\$K_AND
00C0 (00192)						BLR\$K_EQL
00C1 (00193)						BLR\$K_FIELD 1 CYCLE_NUMBER
00D0 (00208)						BLR\$K_PARAMETER 3 0
00D4 (00212)						BLR\$K_EQL
00D5 (00213)						BLR\$K_FIELD 1 MAP_NUMBER
00E2 (00226)						BLR\$K_PARAMETER 3 1
00E6 (00230)						BLR\$K_EQL
00E7 (00231)						BLR\$K_FIELD 1 BLOCK_NUMBER
00F6 (00246)						BLR\$K_PARAMETER 3 2
00FA (00250)						BLR\$K_EQL
00FB (00251)						BLR\$K_FIELD 1 INSTALLATION_NUMBER
0111 (00273)						BLR\$K_PARAMETER 3 3
0115 (00277)						BLR\$K_GTR
0116 (00278)						BLR\$K_FIELD 1 OCCUPANT_NUMBER
0128 (00296)						BLR\$K_PARAMETER 3 4
012C (00300)						BLR\$K_EQL
012D (00301)						BLR\$K_FIELD 1 CYCLE_FILE_FLAG
013F (00319)						BLR\$K_LITERAL
0140 (00320)						DSC\$K_DTYPE_CHAR 1 (sub-type: 0) "Y"

Equivalent SQL

This is equivalent to the following SQL:

where

```
(
    CYCLE_NUMBER          = <parameter>
  and MAP_NUMBER          = <parameter>
  and BLOCK_NUMBER        = <parameter>
  and INSTALLATION_NUMBER = <parameter>
  and OCCUPANT_NUMBER     > <parameter>
  and CYCLE_FILE_FLAG     = 'Y'
)
```

Order by Clause

0281	(00641)					BLR\$K_SORT	6
0283	(00643)					BLR\$K_ASCENDING	
0284	(00644)					BLR\$K_FIELD	1 CYCLE_NUMBER
0293	(00659)					BLR\$K_ASCENDING	
0294	(00660)					BLR\$K_FIELD	1 MAP_NUMBER
02A1	(00673)					BLR\$K_ASCENDING	
02A2	(00674)					BLR\$K_FIELD	1 BLOCK_NUMBER
02B1	(00689)					BLR\$K_ASCENDING	
02B2	(00690)					BLR\$K_FIELD	1 INSTALLATION_NUMBER
02C8	(00712)					BLR\$K_ASCENDING	
02C9	(00713)					BLR\$K_FIELD	1 OCCUPANT_NUMBER
02DB	(00731)					BLR\$K_ASCENDING	
02DC	(00732)					BLR\$K_FIELD	1 CHECK_DIGIT

Order By Clause

This is equivalent to the following SQL

```
order by  cycle_number  
         ,map_number  
         ,block_number  
         ,installation_number  
         ,occupant_number  
         ,check_digit
```

Comparison with the Strategy

- This correlates with the last entry in the reported strategy
- Other clauses included “greater than” restrictions on the remaining “Number” fields

The query strategy again:

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

Index on the Table

```
SQL> show index account_s_03;
```

```
Indexes on table ACCOUNT:
```

```
ACCOUNT_S_03
```

```
with column CYCLE_NUMBER  
and column MAP_NUMBER  
and column BLOCK_NUMBER  
and column INSTALLATION_NUMBER  
and column OCCUPANT_NUMBER  
and column CHECK_DIGIT  
and column CYCLE_FILE_FLAG  
and column CONSUMER_ID  
and column SITE_ID  
and column ACCOUNT_ID  
and column ACCOUNT_STATUS_CODE
```

```
No Duplicates allowed
```

```
Type is Sorted
```

```
Compression is DISABLED
```

```
Node size 960
```

```
Store clause:
```

```
STORE in ACCOUNT_S_03_U
```

Analysis of Query and Solution

- Interpreting long runs of BLR is quite tedious
 - Can often be confusing especially if there are correlated subqueries , 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

Capturing Rdb Run-time Information

- T: Records transaction statistics.
- All the modes in the set transaction statement
 - Read-only & Read-write
 - Reserving clauses
 - Isolation levels
- Can be used to discover stray transactions or renegade read-write transactions or renegade read-only transactions

Capturing Rdb Run-time Information

- T: Records transaction statistics.
- This flag is optional, especially for high-transaction rate environments.
- Describes start and end of transactions
 - Set transaction
 - Commit
 - Rollback
 - Can be used to find transactions which waste work
- Number of lines in log file is unlimited
 - All other reports selected have limited amount of data included in the log files

Transaction Statistics

```
Commit_transaction on db: X000000001
Prepare_transaction on db: X000000001
~T Transaction Parameter Block: (len=2)
0000 (00000) TPB$K_VERSION = 1
0001 (00001) TPB$K_READ (read only)
Start_transaction on db: X000000001, db count=1
```

Capturing Rdb Run-time Information

- E: Provides an execution trace for later analysis of the dynamic query optimization
- Useful to query tuner to understand the way dynamic queries are executing
- Not used very often, only when trying to figure out how well dynamic optimizations are working

Dynamic Optimizer Execution Trace

Firstn

Leaf#01 FFirst BUS_PRCSS_EVENT Card=305

BgrNdx1 BUS_PRCSS_EVENT_S_05 [5:5] Fan=13

BgrNdx2 BUS_PRCSS_EVENT_S_04 [7:7] Fan=10

~E#0002.01(1) Estim Ndx:Lev/Seps/DBKeys 2:1/0/0 1:1/1/1

~E#0002.01(1) BgrNdx2 Termin* DBKeys=1 Fetches=0+0 RecsOut=1`CUT

~E#0002.01(1) FgrNdx FFirst DBKeys=1 Fetches=0+0 RecsOut=1`CUT`ABA

~E#0002.01(2) Estim Ndx:Lev/Seps/DBKeys 2:1/0/0 1:1/0/0

~E#0002.01(2) BgrNdx2 EofData DBKeys=0 Fetches=0+0 RecsOut=0 #Bufs=0

- The dynamic optimizer selected index “_04” as the most selective after re-evaluating the strategy
- One row was returned and the scan using the “05” index was abandoned

Another Instance of Same Query

```
~E#0002.01(3) Estim    Ndx:Lev/Seps/DBKeys 2:1/0/0 1:1/1/1
~E#0002.01(3) BgrNdx2 Termin*  DBKeys=1  Fetches=0+0  RecsOut=1`CUT
~E#0002.01(3) FgrNdx  FFirst    DBKeys=1  Fetches=0+0  RecsOut=1`CUT`ABA
```

- Multiple instances of the execution trace are reported in each statistics file
- Limited by the nn in “Enn”
- Again, the dynamic optimizer corrects the actual strategy

Dynamic Optimizer Execution Trace

- The actual query specified 7 restrictions
- The static optimizer became confused when it saw a potential of two indexes that could be used to solve the query
- It selected both plus dynamic optimization
- The dynamic optimizer correctly analyzed the fact that it should use the “_04” index [all 7 restrictions]
- If there is much dynamic optimization, use psuedo-ranked indexes
 - Easier for dynamic optimizer to make a better estimate
 - Remove some problems for duplicate indexes
 - More compressed in memory

Dynamic Execution Trace

- We have not yet incorporated a scan of the output files for analysis of the execution trace
- It is currently completely manual
- A limited number of execution traces is requested to limit the overall statistics file sizes
- Can be very helpful when trying to understand the actual behavior of the dynamic optimizer
- For the new world of GUI queries, with partially specified windows and a myriad of alternatives, the dynamic optimizer is the only way to get decent performance from these screens.

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

SQL Query Table Definition

Columns for table SQL_QUERY:

Column Name	Data Type	Domain
-----	-----	-----
SQL_QUERY_ID	INTEGER	ID_DOMAIN
Primary Key constraint SQL_QUERY_PRIMARY_SQL_QUERY_ID		
EXECUTABLE_NAME	CHAR(39)	VMS_FILENAME_DOMAIN
DIRECTORY_NAME	CHAR(256)	NAME_DOMAIN
NODE_NAME	CHAR(6)	VMS_NODENAME_DOMAIN
DATABASE_NAME	CHAR(256)	NAME_DOMAIN
BLR_TEXT	CHAR(40960)	BIG_TEXT_DOMAIN
BLR_LENGTH	INTEGER	NVALUE_DOMAIN
OPTIMIZATION_TEXT	CHAR(2048)	TEXT_DOMAIN
OPTIMIZATION_LENGTH	INTEGER	NVALUE_DOMAIN
SQL_TEXT	CHAR(16384)	MEDIUM_TEXT_DOMAIN
SQL_LENGTH	INTEGER	NVALUE_DOMAIN
NEXT_SQL_QUERY_ID	INTEGER	ID_DOMAIN
DBA_ACCEPTED	CHAR(1)	Y_N_DOMAIN
DATE_INSERTED	DATE VMS	CURRENT_DATE_TIME

SQL Query Table

- Synthetic Ids are used to maintain unambiguous track of each query
 - Simple serial number
- Store the BLR in clear text format
 - Embedded control characters for easy formatting
 - Strip out prefix information because it adds nothing to the process but consumes substantial space
- Stores the reverse parsed SQL in clear text
 - Embedded control characters for easy formatting
 - Table aliases denoted by their context numbers in BLR
- Column lengths limited by overall table row length

SQL Query Table

- Length information for long text columns to allow easier matching to find identical queries
- History of solutions maintained by pointer to replacement `next_sql_query_id`
 - Most recent solution for a particular query has value of zero
 - Non-zero denotes a query whose optimization has changed
- Status of query is indicated in `DBA_ACCEPTED` column
 - Yes
 - No
 - Duplicate of some other query
 - Obsolete

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 Statistics 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

Additional Metrics DB Tables

- Additional metrics tables are populated by BLR parser
- One for Rdb indexes used by the query
- One for the date when query last encountered & for storage of the query outline
- Data model is due to historic reasons
 - But must limit total size of Rdb rows
 - Long query row already a problem

Rdb Indexes Used by Queries

Columns for table SQL_QUERY_INDEX:

Column Name	Data Type	Domain
-----	-----	-----
SQL_QUERY_ID	INTEGER	ID_DOMAIN
SEQUENCE_NUMBER	INTEGER	ID_DOMAIN
INDEX_NAME	CHAR (31)	RDB_NAME_DOMAIN
STRATEGY_TEXT	CHAR (128)	SMALL_TEXT_DOMAIN

- This table provides the basis for determining which queries use which index

Date Information for Queries

Columns for table SQL_QUERY_MORE:

Column Name	Data Type	Domain
-----	-----	-----
SQL_QUERY_ID	INTEGER	ID_DOMAIN
Primary Key constraint	SQL_QUERY_MORE_PRIMARY1	
OUTLINE_TEXT	CHAR(2048)	TEXT_DOMAIN
DATE_LAST_ENCOUNTERED	DATE VMS	CURRENT_DATE_TIME

- This table allows us to determine when a query was last encountered
- And what the Rdb-generated outline is
- Location in separate table due to row-length limitations

Additional Reports on Queries

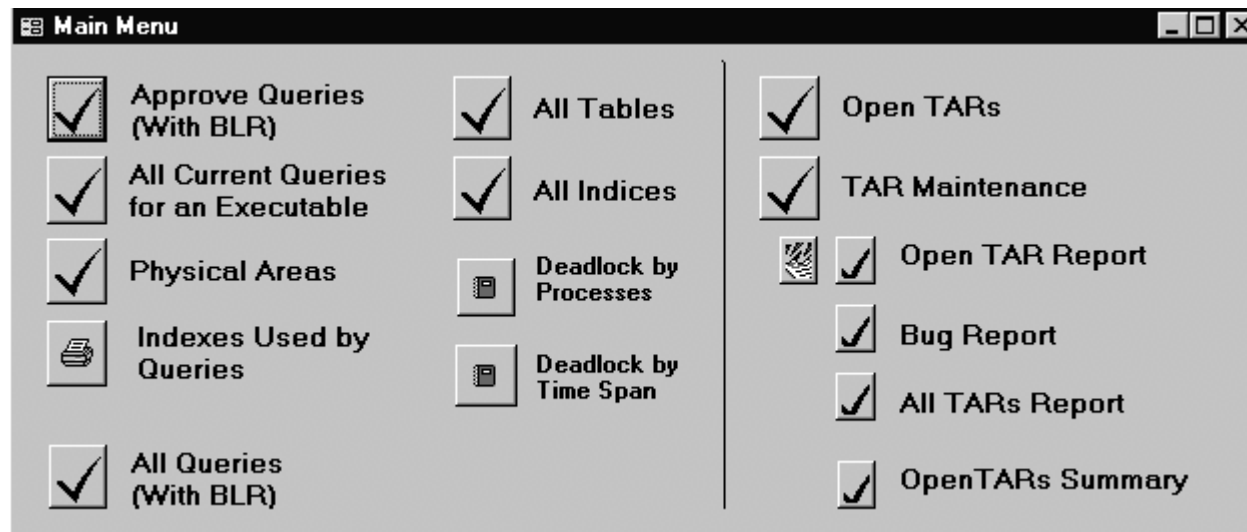
- By joining data from various query tables we can report which indexes have been used recently
- Which indexes have not been used recently
- Which programs use which index
- Which index is used by which program

- You name it

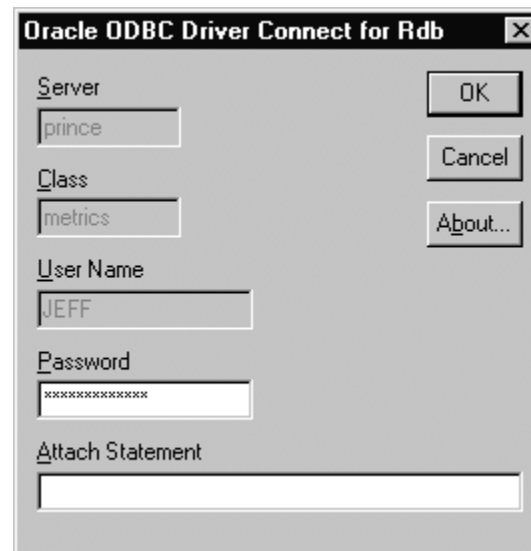
GUI Interface to Metrics Database

- 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

Access Main Menu



Logging In



The image shows a dialog box titled "Oracle ODBC Driver Connect for Rdb". It contains several input fields and buttons. The "Server" field contains the text "prince". The "Class" field contains the text "metrics". The "User Name" field contains the text "JEFF". The "Password" field contains a series of asterisks "*****". The "Attach Statement" field is empty. On the right side of the dialog, there are three buttons: "OK", "Cancel", and "About...".



Actions to do

Table Cardinality

Query is approved

EXE: CR0170
 Node: BGBKX5
 Inserted: 4/28/98 7:53:11 PM
 Accepted?:

Accept
 Reject
 Obsolete

SQL:

Database: \$1\$DIA2:[NEW_DB]SCP_DB_ROOT.RDB;
 Directory: \$1\$DIA2:[RMS_APPLICATION.RELEASE_02_03.][EXE]
 ID: 577
 Next ID: 0

Optimization:

Other queries that use this index

```

-- 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.CYCLE_FILE_FLAG = "Y"
--     )
--   )
-- Order by 1.CYCLE_NUMBER ASC
--         1.MAP_NUMBER ASC
--         1.BLOCK_NUMBER ASC
--         1.INSTALLATION_NUMBER ASC
--         1.OCCUPANT_NUMBER ASC
  
```

Change Font Size
Disjunctive SQL form

Comment characters already present

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

Scrolling box

```

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$K_RSE 1
00A6 (00166) | | | | BLR$K_RELATION ACCOUNT 1
  
```

Scrolling box

Scrolling box

Record: 32 of 33 (Filtered)

Work to be done

Approve Queries Window

- Automatically restricts queries to those with approval flag set to “N” and next_id = 0
- May be tailored to restrict the result table on any other criteria
 - A by-product of using Microsoft Access
 - Provides a quick way to generate reports on selected criteria such as table names or index names
 - SQL partial field matching provides powerful tool to generate subsets of information

File Edit View Insert Format Records Tools Window Help

Filter By Form
 Filter By Selection
 Filter Excluding Selection
 Advanced Filter/Sort...

Sort
 Apply Filter/Sort
 Remove Filter/Sort

Save Record Shift+Enter
 Refresh

Data Entry

Table Cardinality

Directory: \$TSDIAZ:[CASH.RELEASE_01_00].[EXE]
 ID: 138
 Next ID: 0

Optimization: Firstn Conjunct Get
 Retrieval sequentially of relation BANK_CODE_TABLE

Other queries that use this index

Not Yet Accepted

EXE: CASH_TRANSACTION
 Node: BGBCKS
 Inserted: 4/28/98 6:55:26 P
 Accepted?: N
 SQL: ↑ ↓ Query

```
-- From CASH_TRANSACTION_L...
--
-- Select 1.BANK_CODE
-- From 1.BANK_CODE_TABLE
-- Where 1.OLD_BANK = <param>
-- Limit to 2 rows
```

```
0000 (00000) BLR$K_VERSION4
0001 (00001) | BLR$K_BEGIN
0002 (00002) | BLR$K_MESSAGE 1 4
... 4 message vector lines ignored...
0011 (00017) | BLR$K_MESSAGE 2 0
... 0 message vector lines ignored...
0015 (00021) | BLR$K_MESSAGE 3 1
... 1 message vector lines ignored...
001C (00028) | BLR$K_RECEIVE 3
001E (00030) | | BLR$K_BEGIN
001F (00031) | | BLR$K_SEND 1
0021 (00033) | | BLR$K_BEGIN
0022 (00034) | | BLR$K_ASSIGNMENT
0023 (00035) | | | BLR$K_LITERAL
0024 (00036) | | | DSC$K_DTYPE_L 0 "100"
002A (00042) | | | BLR$K_PARAMETER 1 0
002E (00046) | | | BLR$K_FOR
002F (00047) | | | BLR$K_RSE 1
0031 (00049) | | | BLR$K_RELATION BANK_CODE_TABLE 1
```

Record: 1 of 23 (Filtered)

Form View FLTR NUM

We Have Parsed Very Complex Queries

- Much too large for a Power Point slide
- A 4 foot printout
- Can show via Adobe Acrobat
 - 24 table join
 - 4 subqueries

- 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

Query Strategy Report

Production optimizations changed! 1998-10-21 10:13:26.80

Node Executable	Old New	Inserted	Encountered	ID
NODEE		1998-10-01	1998-10-01	83840
CTML4200		1998-10-03		84035

Old Opt:
| ~S: Outline "CTML4200_01" used

| Get Temporary relation Retrieval by index of relation
TRADER_SHORT_CODE
| Index name TRADER_SHORT_CODE_S_01 [4:4] Direct lookup

New Opt:
| Conjunct Get Temporary relation

| Retrieval by index of relation TRADER_SHORT_CODE

| Index name TRADER_SHORT_CODE_S_03 [3:3]

Query Strategy Report

- It is clear something has gone wrong with this database or query
- Old strategy was
 - Direct lookup
 - Used a query outline
- New strategy uses less optimal [3:3] index scan
 - Outline is gone
- Diagnosis: Somehow the query outline was blown away.
- DBA Response: Investigate and replace outline

Another Query Strategy Report

Production optimizations changed! 1998-10-21 10:13:26.80

Node Executable	Old New	Inserted	Encountered	ID
NODEE		1998-10-01	1998-10-01	83854
CTML7600		1998-10-03		84036

Old Opt:
| Conjunct Get Retrieval by index of relation EXTERNAL_TRANS

| Index name EXTERNAL_TRANS_S_01 [2:3]

New Opt:
| Sort

| Leaf#01 BgrOnly EXTERNAL_TRANS Card=<num>

| BgrNdx1 EXTERNAL_TRANS_S_01 [3:2] Fan=<num>

Analysis of the New Report

- The External Trans table is one of the hot spots of the database
 - Anything going wrong there can substantially impede the application
- The new query is scanning the index in the opposite order
- And then doing a sort
 - Inspection showed that sort work files were being created
 - At a very high rate
 - Which caused high operating system overhead
- Obviously, this query is broken and needs to be adjusted.

Another New Report

Production optimizations changed! 1998-11-03 00:39:35.32

Node	Old	Inserted	Encountered	ID
Executable	New			

BGBCKS		1998-05-18	1998-09-26	1895
CR0170		1998-10-10		4840

Old Opt:

- | Firstn Conjunct Get Retrieval by index of relation
- | METER_EVENT
- | Index name METER_EVENT_S_03 [2:2]

New Opt:

- | Firstn
- | Leaf#01 Sorted METER_EVENT Card=<num>
- | FgrNdx METER_EVENT_S_02 [1:1] Fan=<num>
- | BgrNdx1 METER_EVENT_S_04 [2:2] Fan=<num>

Number of changed optimizations - 1

- This is just a switch to the dynamic optimizer, probably no problem to worry about

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
- 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 Usage

Indices Being Used by Queries

SHORT_INDEX_NAME	SHORT_EXECUTABLE	DATE_LAST_ENCOUNTED	NODE_NAME	DATE_INSERTED	SQL_QUERY_ID
ACCOUNT_BATCH_PROCESS_S_01	ADD015	7/28/98 5:40:25 P M	6G BC HS	26/98 4:37:57 P M	2353
	ADD018	9/5/98 7:38:20 P M	6G BC HS	4/98 10:43:19 P M	2748
	BARDLY	9/26/98 7:37:42 P M	6G BC HS	28/98 6:50:42 P M	69
	BELCHG	9/5/98 7:39:40 P M	6G BC HS	20/98 3:19:37 P M	2939
	CCS010	9/5/98 8:20:19 P M	6G BC HS	4/98 11:12:19 P M	2777
	CMF010	9/5/98 8:21:41 P M	6G BC HS	26/98 5:25:57 P M	2387
	CMF030	8/29/98 8:41:23 P M	6G BC HS	27/98 7:55:00 P M	2679
	CRD017	9/26/98 8:27:11 P M	6G BC HS	28/98 7:38:29 P M	225
	CRD023	9/26/98 8:29:20 P M	6G BC HS	28/98 7:38:51 P M	255
	CRD023	9/26/98 8:29:27 P M	6G BC HS	28/98 7:39:04 P M	279
	CRD029	9/5/98 8:24:23 P M	6G BC HS	4/98 11:14:13 P M	2812
	CRD032	9/5/98 8:24:43 P M	6G BC HS	4/98 11:14:35 P M	2840
	CRD035	9/5/98 8:25:07 P M	6G BC HS	4/98 11:14:55 P M	2868
	CRD060	9/26/98 8:31:17 P M	6G BC HS	28/98 7:40:37 P M	292
	CRD071	9/5/98 8:25:27 P M	6G BC HS	4/98 11:15:34 P M	2897
	CRD140	9/26/98 8:34:54 P M	6G BC HS	28/98 7:42:45 P M	443
	CRD140	9/26/98 8:35:08 P M	6G BC HS	28/98 7:42:12 P M	400
	CRD145	9/5/98 8:38:07 P M	6G BC HS	9/5/98 8:38:07 P M	4203
	CRD160	9/5/98 8:40:40 P M	6G BC HS	18/98 6:57:01 P M	1818
	CRD160	9/5/98 8:41:05 P M	6G BC HS	18/98 6:57:30 P M	1859
	CRD162	9/26/98 8:36:27 P M	6G BC HS	28/98 7:52:19 P M	511
	CRD164	9/26/98 8:36:49 P M	6G BC HS	28/98 7:52:44 P M	549
	CRD170	10/10/98 7:00:45 P M	6G BC HS	10/98 7:00:44 P M	4838
	CRD173	8/29/98 9:08:23 P M	6G BC HS	29/98 9:08:23 P M	4084

Determining 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

Report for Old Indexes

Old indices - > 60 days old 1998-09-14 01:29:28.31

Index	Last Used	# of approved queries
EVENT_STATS_S_03	1998-06-17	11
NODEB Y 1997-10-23 CPMBU130		
POSITIONS_DATABASE_ROOT:[DATABASE.RDB_01]MY_DATABASE.RDB;1		
NODEB Y 1997-10-25 CPMBU140		
POSITIONS_DATABASE_ROOT:[DATABASE.RDB_01]MY_DATABASE.RDB;1		
NODEB Y 1997-10-25 CTMA8550		
INTEGRATED_DATABASE_ROOT:[DATABASE]MY_DATABASE.RDB;5		

etc.

Report for Old Indexes

- In total there had been 11 queries that used the index
- Need to investigate whether the code changed and this is now obsolete
- May be that index was created for code that has not yet been released
- Requires DBA investigation

Database Changes

- If there are several DBAs sometimes communications is not perfect
- Also, may need to create an audit trail of what has happened to the database
- Database metadata extracted daily via RMU/EXTRACT
- File sent to development system

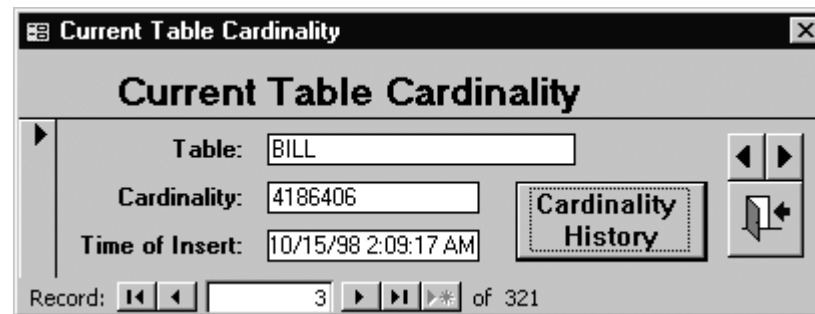
Database Changes

- Compared to previous version
- If differences are found
 - Send mail message to DBA
 - Update version in CMS
- Complete history of production metadata thereby recorded in CMS
 - Used to trace anniversary of changes
 - Can detect unauthorized changes
 - Must be policed by management and DBAs

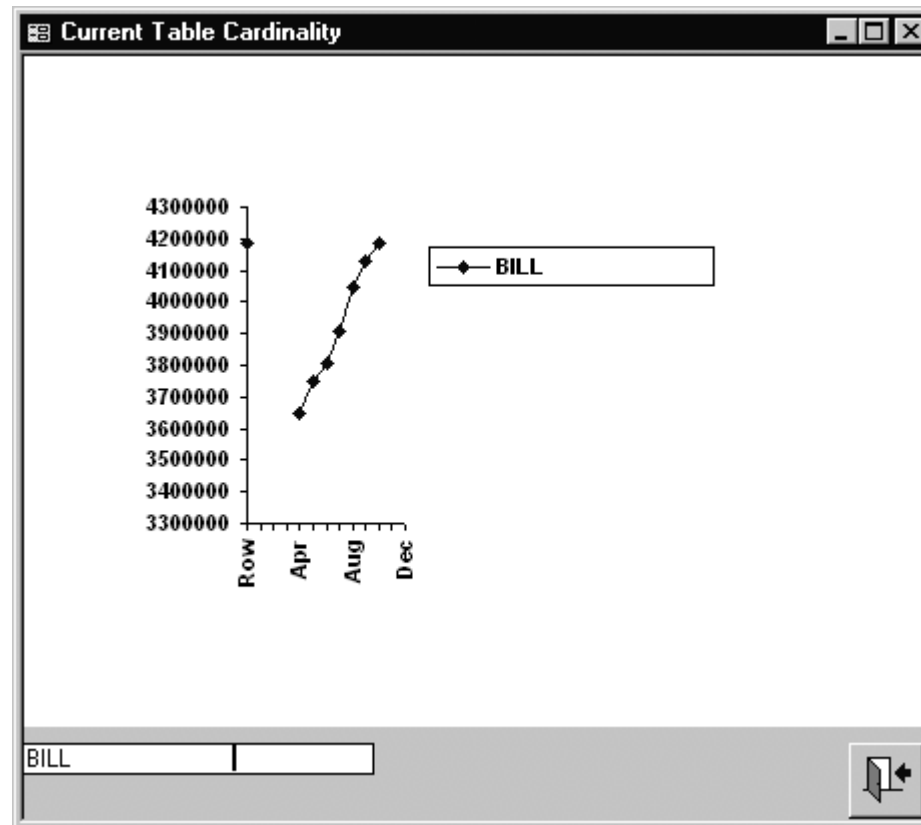
Table Cardinality

- One potential for database management problems is the growth of cardinality for tables
- Results in storage areas that extend
- Fragmented [VMS] files
 - Overhead for random access
 - Lower performance
- Cardinality can be obtained periodically, for instance daily or weekly or monthly
- Stored in a Metrics database table
- Queried via Access application
 - Graph results

Table Cardinality Query



Cardinality Growth Over Time



The graphics is derived from Access capabilities

Physical And Logical Areas

- It is important to have a good readable source of Rdb logical and physical area information
 - To understand the association of logical and physical areas for I/O analysis
 - To understand the implications of locking analyses obtained from RMU
- This is obtained directly from Rdb
 - Physical areas obtained from RMU/DUMP/HEADER
 - Logical areas obtained from RMU/DUMP/LAREA=rdb\$aip
 - Must parse the resulting files
 - Create a series of insert statements for loading into Rdb
 - Could create delimited text file also

Parsing The Logical and Physical Areas

```
@JCC_DBA_DISK:[DBA_TOOLS.DEVELOPED_TOOLS]METRICS_ADD_PHYSICAL_LOGICAL_AREAS.COM -
  testing_db
Writing header dump
Writing AIP dump
Parsing the header dump
End of file detected on HEADER DUMP file
Parsing the AIP dump
End of file detected on AIP file
  Attaching to the database
  Deleting previous data
0 rows deleted
0 rows deleted

  Detaching from database
  Attaching to database
  Inserting logical areas
etc.
```

Physical And Logical Areas Mapping

PHYSICAL AREA

Area ID	16	Area Name:	CODE_TABLES_TBL_U	Access Mode	R
Page Size	12	Snapshot Area	240	Page Format	U
Initial Allocation	200	Row Level Locking	D	SPAM Interval	2719
Current Allocation	201	SPAM Count	1	Extends	0
FILE_NAME:	JWOCKY\$DKA500:(TESTING.DATABASE)DB_CODE_TABLES.RDA;1				

Larea	169
ABM Page	2
Physical Area	16
Area Name	ACCOUNT_STATUS_CODE_T/
Snaps Enabled TSN	24
Record Length	33
Flags	0
Total Bytes	0

Record: 1 of 83

Record: 16 of 428

Physical and Logical Areas Mapping

- The form can be used to query for particular logical area numbers
 - Allows interpretation of DB-key stalls in RMU/SHOW statistics
- Can query for physical area numbers
 - Similar interpretation of page stalls and page reads/writes
- Can also write whatever reports are desirable
- Queries can locate areas that have extended
- Logical area information such as record length useful for sizing and planning
 - For instance row caches require line sizing information

Managing Lock Stalls and Deadlock

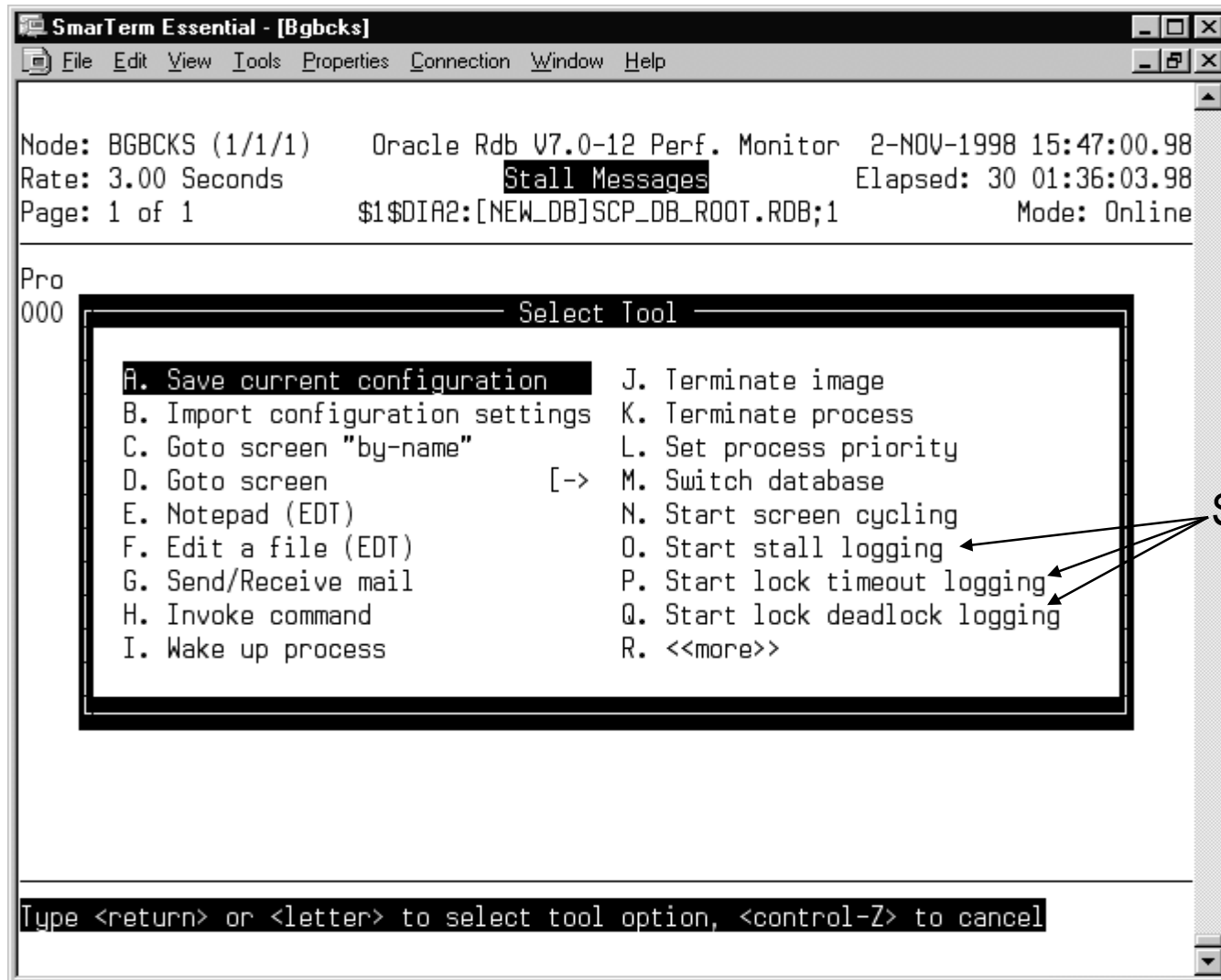
- Some of the most difficult database performance problems are associated with lock stalls and deadlocks
- Frequently they are ephemeral, here now, gone quickly
- The performance of a database can be severely disrupted by any kind of locking stalls
 - Stalls create a lot of friction in the application
 - Serious problems can cascade severely

Addressing Lock Stalls and Deadlocks

This is a four step process:

- Detecting the stalls & deadlocks
- Attributing them to processes & programs
- Understanding the causes
- Resolving the issues
 - Often is an application design problem
 - Some database design remedies
 - Horizontal table partitioning for multiple concurrent insert servers
 - Hashed indexes for chronological problems
 - Smaller B-tree index nodes

RMU/Show Statistics Screen



Stall Logs

Parsing the Stall Logs

- The stall logs are in ad-hoc format
- To obtain useful information they must be parsed
- Each file format is distinct
- Results of parsing should be entered into the database
- Reports can be run using SQL or the Access application

Deadlock Report

ANALYSIS OF DEADLOCKS

Logical AREA_NAME	Count	Physical Area
L: ASSET_TRANS_MSG_S_02	72	
L: SETL_INSTR_CODE_S_01	62	
L: SETL_RPTG_DB_S_05	32	
L: SETTLEMENT_CYCLE_S_01	13	
L: EXECUTION_TIME_RUL_S_01	10	
P: EVENT_TRIGGER	9	TMS_DATA_78
L: EXC_DEF_BY_CURR_S_05	8	
L: AST_TP_BY_CURN_S_01	6	
P: EVENT_TRIGGER	6	TMS_DATA_77
P: EXECUTION_TIME_RUL_S_05	5	SETTLEMENT_INDX_43
L: SETL_INSTR_S_04	4	
L: COMMON_BANKING_REP_S_01	2	
L: SETTLEMENT_REQMT_S_04	2	
L: ALLOCATION	1	
etc.		

Deadlock Report

- The data gathered for this report occupied a full 8-hour production interval
 - Quite a few deadlocks were reported
 - Most were line deadlocks indicating real application design problems or database design problems
- It is clear where the deadlocking problems in the application reside
 - The line locks are all on B-tree indexes [suffix S_nn]
 - Required because of particular queries in programs
 - Can use index usage report to trace down the reason for each index
 - May change code to eliminate

Lock Stalls Report

ANALYSIS OF LOCK STALLS

Logical Area Name	Count	Physical Area Name
L: ALLOCATION	1429	
L: EXC_DEF_BY_CURR_S_03	394	
L: SETL_RPTG_DB_S_04	260	
L: ASSET_TRANS_MSG_S_02	173	
L: SETL_INSTR_CODE_S_01	143	
L: TRADE	137	
P: EXECUTION_TIME_RUL_S_05	98	SETTLEMENT_INDX_43
L: EXC_DEF_BY_CURR_S_04	95	
L: EXTERNAL_TRANS	92	
L: SETL_RPTG_DB_S_05	74	
L: EXECUTION_TIME_RUL_S_01	55	
P: EVENT_TRIGGER	49	TMS_DATA_77
P: EVENT_TRIGGER	46	TMS_DATA_78
L: EXC_DEF_BY_CURR_S_05	43	
L: AST_TP_BY_CURN_S_01	38	

etc.

Lock Stalls Report

- Clearly this application suffers from stalls on the allocation table
- Traced to the flow-through nature of the application
- Requires some application design to determine how to respond
- Now know where to spend application engineering effort

Lock Stalls and Deadlocks

- The parsed lock and deadlock files contain process Ids
- These can be converted to process names if VMS process information is collected
- All of this allows attribution of stalls and deadlocks to individual processes
 - If process name is meaningful, then this can allow analysis of application
- Of course, this is a non-trivial amount of work, but the necessary diagnostic information is available

Lock Stalls and Deadlocks

- The reports displayed here were focused on page and line stalls
- Additional stalls possible, for instance reads and writes
- Additional tailored reports possible to reveal that information
- It is clear that analysis of stall information can be extremely useful in isolating problems in an application to:
 - A particular process
 - A particular storage area
 - The kind of stalls, read, write, line, page

Physical Database Analysis

- RMU can be run to produce binary files suitable for loading into a database
- Provides physical storage area analysis
- Index analysis
- Once in the database this data can be used to determine whether there are potential management issues with the structure of the database
- Data can be tracked historically to alert the DBA to future problems

DBA Alerts

- Periodically the RMU data should be analyzed
 - Weekly is good enough for most situations
 - If proactive care is taken to respond to issues which are detected
- Known conditions are analyzed
- If found, the DBA should be alerted
- We use MAIL as the notification method

Some Issues to Watch Out For

- Mixed format areas have extended
- Assuming a good database design only hashed indexes or placement via hashed index tables are stored in mixed areas
- Indicates design error
or
Usage change for area
- Guarantees a performance problem for inserts
- Probably a performance problem for retrievals, but not as severe
- Considered a class 1 problem and must be addressed ASAP

Some Issues to Watch Out For

- Tables which keep on growing
- Indicated by significant percent growth in a day or week or month
- May indicate new table being loaded
 - In large projects, activities are not always coordinated
- May indicate storage area in need of physical file defragmenting if free space is not provided
- May indicate potential issues for disk space if storage area extends

Some Issues to Watch Out For

- Often the database disks are dedicated to the actual database files exclusively
- If so, then the storage area analysis may sum the total usage by disk drive
 - Assumes that the DBA does *not* use concealed logical names to hide devices
- If total space on disk falls below a threshold, then signal a problem
 - Assumes you know the size of each disk

Some Issues to Watch Out For

- Tables whose rows have become fragmented pose special performance problems
 - Rows retrieved require a potential of > 1 I/O per row
 - Updates can be costly also, especially when considering that two or more SPAM pages might also be involved
 - Can be due to metadata changes
 - Careful introduction of changes can sometimes avert actual fragmentation.
 - For instance creating a domain allowing nulls and then changing to a default value
- Reducing fragmentation can be done by
 - Change to storage map
 - Table unload/reload

Some Issues to Watch Out For

- If table fragmentation is bad, fragmented B-tree indexes are very bad
- Leads to multiple retrievals for each node accessed
- If not more I/O because fragments are in same buffer, is certainly more overhead
- Database design error
 - Page size and node size don't match
- Can fix by backing up and restoring area with a larger page size
 - Only fixes itself over time
- Probably should drop and recreate index
 - Use better match to page size and node size

Some Issues to Watch Out For

- If hash buckets overflow onto additional page then this indicates extra work
- Is likely to be a physical database design error
- Page size is too small or there are too few pages in storage area
- DBA response is to
 - Determine why this is happening, for instance, a purge never happened
 - Fix source of problem
 - Re-size area

Some Issues to Watch Out For

- Unique Hash indexes with more than one level indicate bucket overflow
- Duplicates hash indexes with more than two levels indicate higher duplicates cardinality than might be comfortable
- Can be caused by developers forgetting to initialize a column so it is always NULL for a large fraction of rows
- Can also indicate a physical design problem with an index not being appropriate for hashing because of high cardinality of duplicates
 - Replace with sorted index with additional columns from table key

Some Issues to Watch Out For

- Free space in storage areas represents the room for growth
 - Low amounts of free space serve as an alert
- If table cardinality grows unchecked, this will lead to reduced free space
- And ultimately to the storage area extending
- May indicate change in the use of the application
- DBA will probably want to intervene in the process and create a larger storage area
 - Aim for storage areas that are “almost” contiguous
 - Seven or fewer VMS file extents

Some Issues to Watch Out For

- If some developer loads a substantial amount of data into a table, the result might be that an area extends
- Extensions of uniform areas are not the problem that extensions of mixed areas are
- Requires DBA review of the VMS file characteristics
- If file is fragmented, DBA should intervene
 - Move storage area
 - Defragment the disk volume
 - While database is closed
 - Backup and restore the database [not likely with current database sizes]

Analysis For File Fragmentation

```
$ dump/header/blocks=count=0 <filename>
```

```
File Header
```

```
o  
o  
o
```

```
Map area
```

```
Retrieval pointers
```

Count:	735825	LBN:	1362890
Count:	711105	LBN:	3446325
Count:	195650	LBN:	2145580
Count:	740	LBN:	170990
Count:	10365	LBN:	180685
Count:	60	LBN:	251780
Count:	5	LBN:	252105
Count:	250	LBN:	320370
Count:	345	LBN:	343240
Count:	9015	LBN:	352845
Count:	5	LBN:	626740
Count:	150	LBN:	627925
Count:	18240	LBN:	629915
Count:	120035	LBN:	1465

Some Issues to Watch Out For

- TSNs used to be relatively limited in number
 - 32 bits, 4.2 thousand million
 - Is now 48 bits, 65,535 times more!
- Commit to Journal and high transaction rate databases could use up TSNs relatively quickly
 - In a matter of a couple of years
- Can be reset by either rebuilding the database or by using RMU
- Resetting is costly, requiring writes to each and every page of the database
- Availability of TSNs should be monitored so these rebuilds can be properly scheduled

Database Verification

- For critical databases, it is important that their internal data structures be completely verified periodically
- In addition to constraints, internal data structures need attention
 - Most certainly B-tree structures
 - Including matching columns between index entries and table rows
 - And performing this as quickly as possible
 - One sequential scan over the table
 - One scan over each index
- Rdb does not do this
 - Completely
 - Quickly

Database Verification

- We have constructed a “Fast Index Verify” utility
- The goal is to complement the Rdb verification
 - Finds problems that Rdb verification misses
 - Runs faster during leaf node verification
- Sequentially scans table extracting columns needed for index
- Sequentially scans level 1 of index
- Sorts table data into index order
- Matches ordered sets and compares DB-keys
- Complements the RMU index verification

Database Verification

- Perform nightly verification of restored copy of database
 - Index leaf and data verified
 - Use FIV and RMU together
 - Can miss some issues such as SPAM problems
 - Because RMU restore rebuilds these data structures
- Periodic verification of production database
 - Not always possible, there are some real 365x24 applications
- Always do page checksums when doing backup
- Parse all log files to determine exceptions
 - Mail only exceptions to DBAs

Conclusions

- The methodology provides a framework for organizing the DBA workload
 - Centered around mail
 - Can involve a team
 - Maintain mailing lists as system logical names
- Addresses all of the logistical parts of the job
 - Finding tuning problems
 - Finding out about events
- DBA can concentrate on the important parts of the job
 - Being proactive
 - Interacting with the rest of the application team

Conclusions

- Methodology collects the historical information necessary to forecast future database needs
- Is extensible since data store is, itself, a database

Some JCC Experience

- Have applied this methodology completely
 - High-volumes of current development
 - Lots of new queries
 - Both concentrate on GUI interfaces
- Managed the approval of tuning of thousands of queries. In need, can do over 100 in a single day
 - Requires really concentrated attention
- Automatic MAIL notification takes the worry out of missing something
 - Of course, you then have to respond (:-)

Questions?

- Session will be posted at: <http://www.jcc.com>



- Rdb list server, send mail to: **majordomo@jcc.com**
Body of message should say: **subscribe oraclerdb**
or **subscribe rdbd2k**