



Counting Down The DB2 Performance Top Forty

*Forty Guidelines for Improving
DB2 for z/OS Performance*

Sponsored by:



Craig S. Mullins
Mullins Consulting, Inc.
<http://www.craigsmullins.com>



Author

This presentation was prepared by:

Craig S. Mullins
President & Principal Consultant

Mullins Consulting, Inc.
15 Coventry Ct
Sugar Land, TX 77479
Tel: 281-494-6153
Fax: 281.491.0637
E-mail: craig@craigsmullins.com



This document is protected under the copyright laws of the United States and other countries as an unpublished work. Any use or disclosure in whole or in part of this information without the express written permission of Mullins Consulting, Inc. is prohibited.

© 2010 Craig S. Mullins, Mullins Consulting, Inc. All rights reserved.

About CDB Software

- Founded in 1985
- Focused on DB2 for z/OS
- Corporate Philosophy
 - CDB enables businesses to meet growing DB2 demands by improving data management processes with intelligent solutions
- High Speed Automated Utilities that keep your database in top condition while you focus on your applications



CDB / Auto-Online Reorg

- High Speed Fully Online Reorg
 - Applications never leave R/W
 - Patented No-Sort Process
 - Handles any size objects
 - Full SQL Subselect capability to Purge Data
 - Utilizes Real Time Statistics without the need to ever generate JCL
 - Only objects that have to be reorged are processed
 - Never waste CPU reorganizing objects that don't need it
 - » Indexes or Tablespaces
 - Database is fully maintained using the least amount of resources (both CPU and Human) possible

Overview

Assuring optimal performance for database applications is the most time-consuming task for most DBAs. Planning and implementing databases and systems with performance in mind is the best approach to help minimize the overarching burden of constant performance tuning and tweaking. This presentation will count down 40 performance-focused DB2 tips and techniques so you will be better armed to battle the DB2 performance beast.



The Performance Categories

- General Performance Advice**
- Application and SQL Guidelines and Tips**
- Database Design Advice**
 - > Logical Database Design
 - > Physical Database Design
- System and Subsystem Performance**
 - > DB2 Things
 - > Non-DB2 Things
- Performance Monitoring Guidelines**
- Additional Thoughts and Advice**



General Performance Advice

1 Don't Panic!

- > Orderly and Iterative
 - Tune one thing at a time: how else do you know whether the action helped or not?

2 All tuning optimizes at least one of three things:

- > CPU ↓
- > I/O ↓
- > Concurrency ↑



And these three things can impact elapsed time, too.

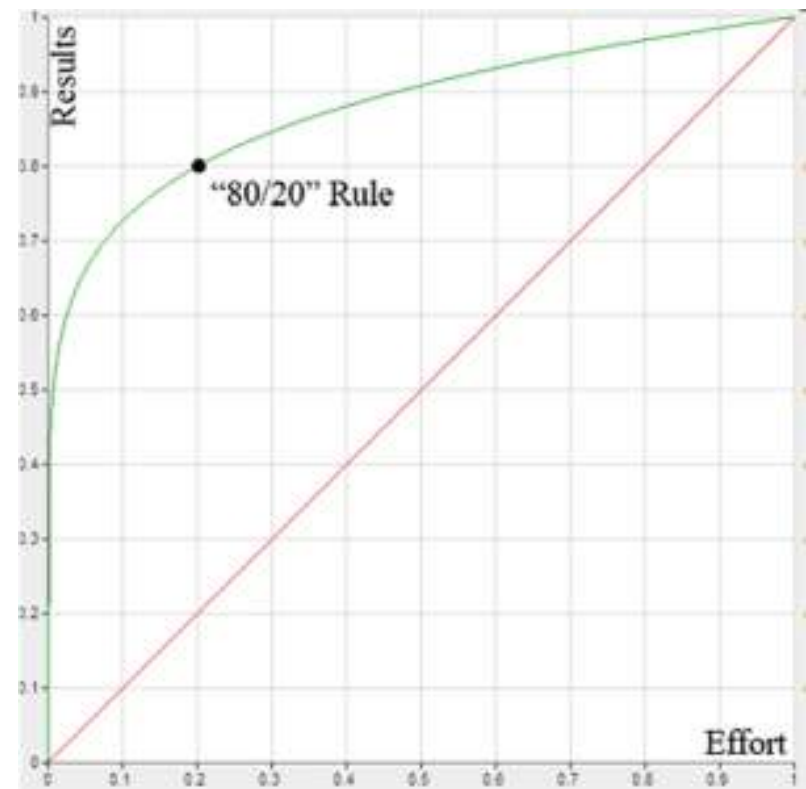


The Pareto Principle

Remember the 80/20 Rule

- 80% of the results of tuning come from 20% of the tuning effort
- 20% of your applications will likely cause 80% of your problems

Don't try to
boil the ocean



Application and SQL

4 Simpler is easier to understand, but complex SQL can be very efficient

- > In general, let SQL do the work, not the program

5 Don't ask for more than you need

- > Retrieve the absolute minimum # of rows required
- > Retrieve only those columns required - never more
- > Always provide join predicates
- > Avoid tablespace scans for large tables
- > Avoid placing columns from equal predicates in your SELECT-list

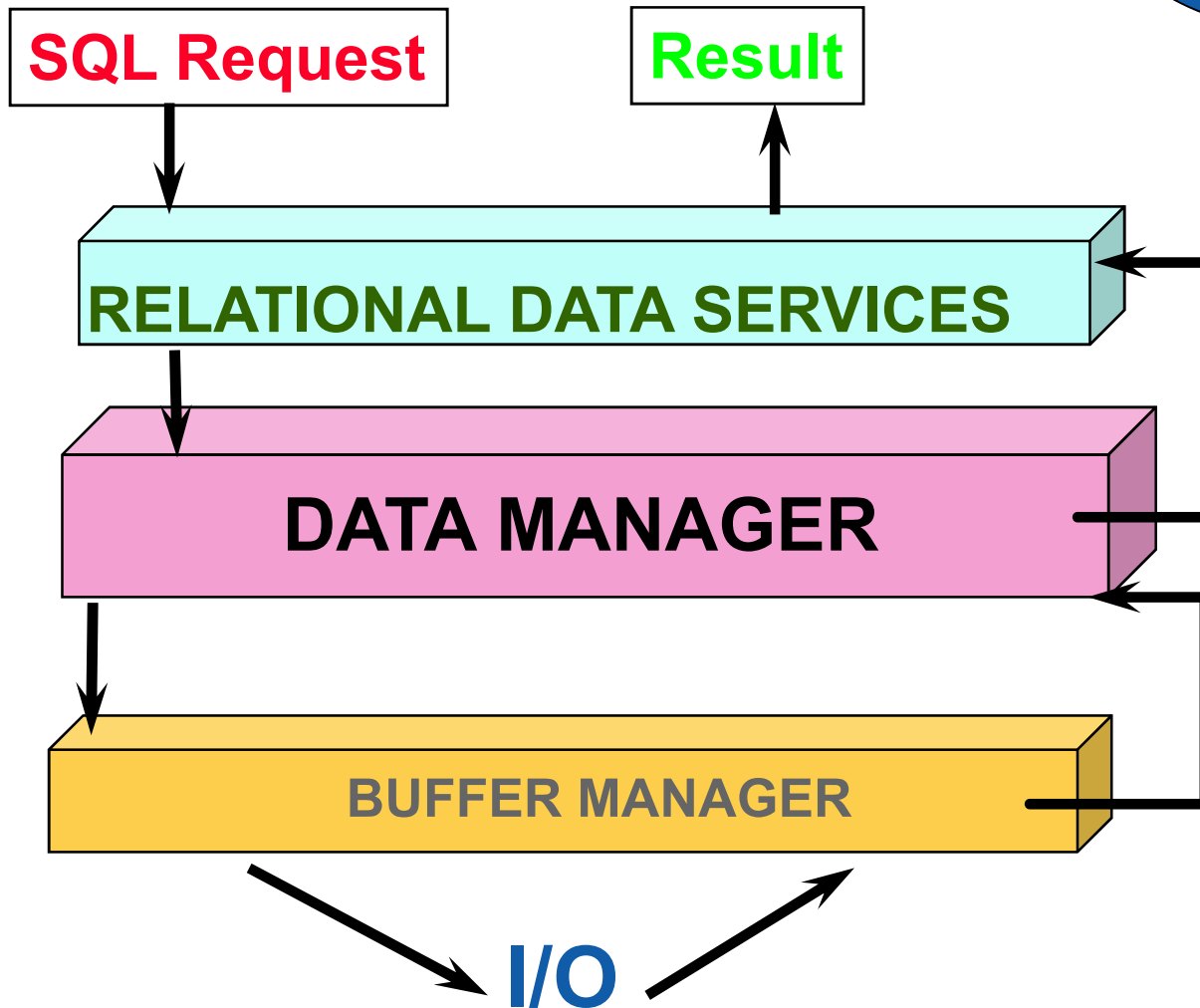
IF-THEN-ELSE

```
SELECT LAST_NAME, FIRST_NAME,  
       JOB_CODE, DEPT, PHONENO  
FROM   DSN8810.EMP  
WHERE  JOB_CODE = 'A' ←  
AND    DEPT = 'MIS'; ←
```



Favor Stage 1 and Indexable Predicates

Performance Monitoring and Tuning Guide
Chapter 14, Page 253



STAGE 2 - Evaluated after data retrieval (non-sargable) via the RDS (Relational Data Services) which is more expensive than the Data Manager.

STAGE 1 - Evaluated at the time the data rows are retrieved (sargable). There is a performance advantage to using Stage 1 predicates because fewer rows are passed to Stage 2 via the Data Manager



Unlearn the “Flat File” Mentality

The number one tendency of new DB2 programmers is to write programs the same way they always have

That is a sure recipe for poor performance

Do NOT simulate master file processing with cursors

- That is: Open cursor, read a row, use value(s) to prime another cursor, read rows, go back to original cursor, etc.
- Instead, code a join!





Minimize Sorting Overhead

Avoid sorting when possible:

- › Consider indexes for ORDER BY and GROUP BY
- › Judicious use of DISTINCT
- › Use care with set operations:
 - UNION ALL versus UNION
 - INTERSECT ALL versus INTERSECT
 - EXCEPT ALL versus EXCEPT

More on sort issues later...





Avoid Black Boxes



Use ROWSET Cursors (for Multi-Row FETCH)



Multi-row FETCH and INSERT statements requests DB2 to send multiple rows of data, at one time, to and from the database.

DRDA
DSNTEP4
DSNTIAUL
QMF *with APAR*

Significant performance gains (up to 50%)

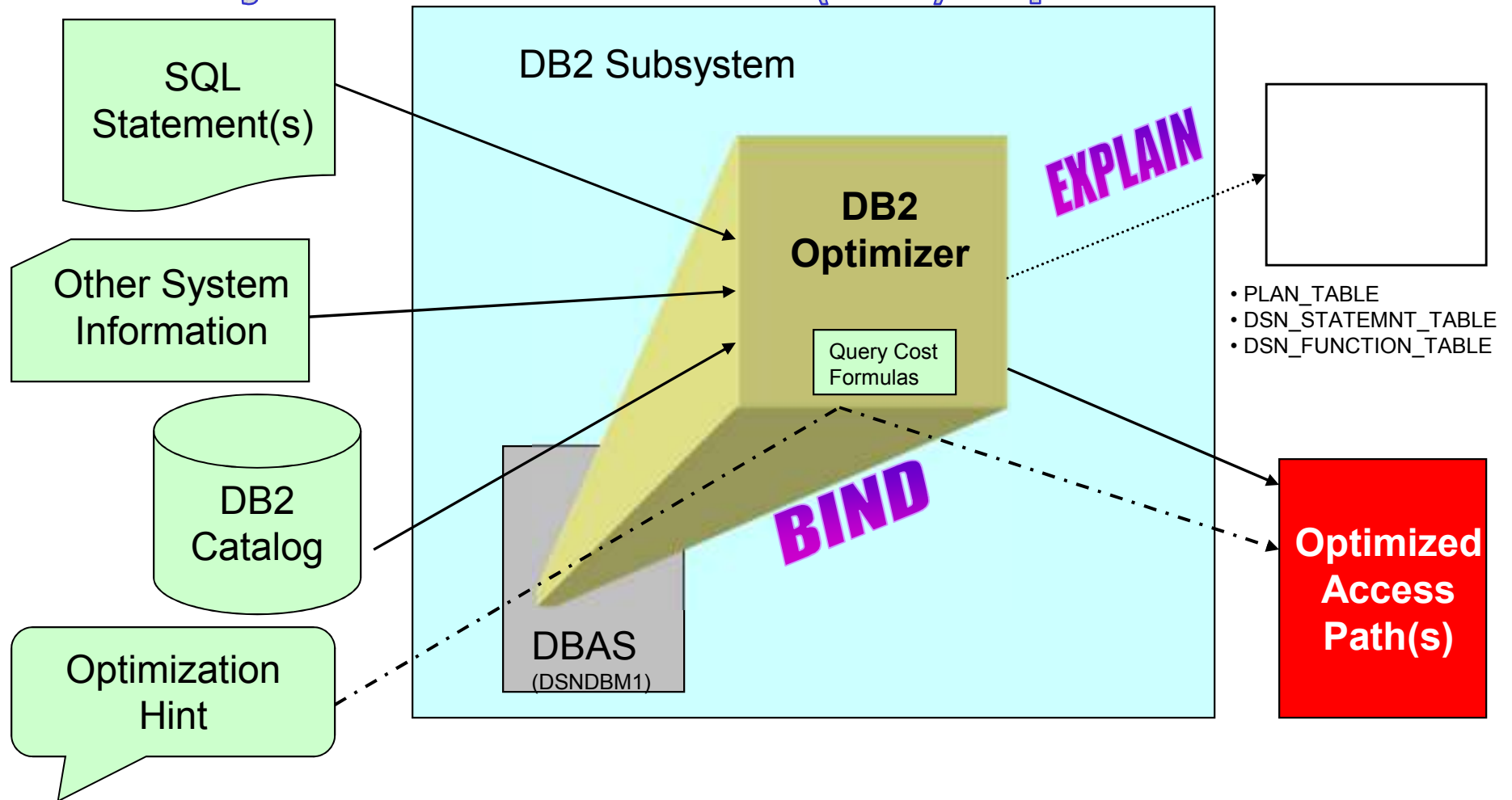
> Avoids API overhead

- Local applications: fewer database accesses.
- Distributed applications: fewer network operations and a significant improvement in performance.



Understand Optimization

Always BIND with EXPLAIN(YES) in production





...But Realize That The Optimizer Might Get It Wrong

According to Terry Purcell (IBM), the top reasons why the optimizer might not formulate the best access path are:

- Unknown data skew
- Lack of correlation statistics
- Optimistic range predicate estimates (with host variables or parameter markers)
- Lack of knowledge of number of required rows



Source: IDUG Europe 2008 (A13) - Stabilizing Access Paths Across Rebind, Terry Purcell, IBM SVL



Assisting the Optimizer

RUNSTATS: still the #1 approved method for generating efficient access paths

VOLATILE

OPTHINT

REOPT

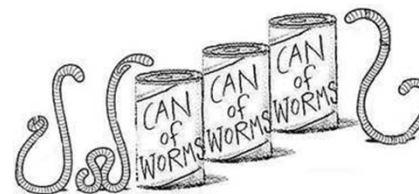
OPTIMIZE FOR n ROWS

Modify the SQL (rearrange tables, changes predicates, etc.)

SQL Tweaks

- > (OR 0=1)
- > +0 or *1

Change statistics in the DB2 Catalog



And learn to use Visual EXPLAIN, Optimization Service Center (OSC), Data Studio, and/or your favorite SQL tuning tool



Follow the Five R's

What about REBINDing?

The best approach is to perform regular REBINDs over time as your data changes.

The shorthand for this approach is the Five Rs, consisting of the following steps:

1. Real-time statistics (RTS) inspection
2. REORG
3. RUNSTATS
4. REBIND
5. Review the results

Code with Locking in Mind

15 Minimize deadlocks by coding modification statements in the same sequence for all programs

16 Minimize lock duration by issuing modification statements as close to the end of the UOW as possible

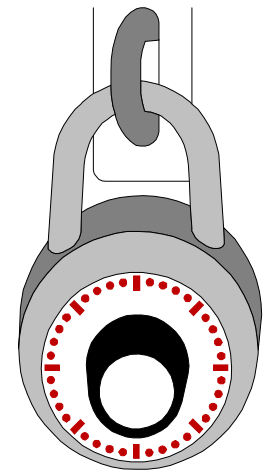
- the later in the UOW the update occurs, the shorter the duration of the lock

17 Encourage Lock Avoidance

- ISOLATION(CS) / CURRENTDATA(NO)
- Can be used only by read only cursors

18 Some other locking ideas...

- > Use LOCK TABLE if it makes sense
- > Consider ISOLATION(UR) to avoid locking



Plan and implement a COMMIT strategy
or experience TIMEOUTs and DEADLOCKS



Fear of COMMITing

- 911 timeout/deadlock w/ rollback
- 913 timeout/deadlock w/o rollback



Deciding Between BETWEEN and <=,>=

Either is acceptable but BETWEEN is easier to code and understand.

```
WHERE COL1 BETWEEN :HOST_VAR1 AND :HOST_VAR2
```

```
WHERE COL1 >= :HOST_VAR1 AND COL1 <= :HOST_VAR2
```

```
WHERE :HOST_VAR BETWEEN COL1 AND COL2
```

Do not use BETWEEN here as that makes the predicate STAGE 2.

```
WHERE :HOST_VAR >= COL1 AND :HOST_VAR <= COL2
```

But this equivalent formulation is STAGE 1.



Be Careful With LIKE

If you specify a wild card at the beginning of the comparison string, LIKE cannot use direct index lookup.

LIKE '%string' or LIKE '_string'

Sometimes you can convert LIKE into an IN-list, or a BETWEEN predicate, if you know your data.

```
LIKE 'BIN_  
IN ('BIN1', 'BIN2', 'BIN3')
```

If you only have three bins.

Retrieve all employees whose last name starts with K?

```
LIKE 'K%'  
BETWEEN 'KAAAAAAAAAAAA' AND 'KZZZZZZZZZZZZZZ'
```



Predicate Order Can Matter!

First, DB2 applies the predicates that match the indexes selected in the access path. The sequence in which these predicates are applied is based on the order of the column in the index.

After applying matching index predicates, DB2 then applies

1. Stage 1 predicates that were not chosen as matching predicates but still refer to index columns, followed by
2. Stage 1 predicates in columns that were not in the indexes being used,
3. and then, any Stage 2 predicates.

Within each of these three groups, predicates are evaluated based on predicate type and the sequence in which the predicate appears in the SQL statement.

Predicate types are applied in the following sequence:

1. All equality predicates (including column IN-list, where list has only one element)
2. All range predicates and predicates specifying `column IS NOT NULL`
3. All other predicate types, but non-correlated subqueries are processed before correlated subqueries

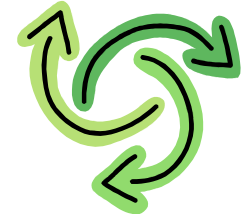


Code Most-Filtering Predicates Before Others Within Predicate Type

Place the predicate that will eliminate the greatest number of rows first (within predicate type).

For example, consider the following statement:

```
SELECT  EMPNO, FIRSTNME, LASTNAME
FROM    DSN8810.EMP
WHERE   WORKDEPT = 'D21'
AND     SEX = 'F';
```



Suppose that **WORKDEPT** has 10 distinct values and **SEX** has only 2 distinct values.

Because both are equality predicates, the predicate for the **WORKDEPT** column should be coded first (as shown) because it eliminates more rows than the predicate for the **SEX** column.



Functions in Your Predicates?

Using a function in your WHERE clause can cause DB2 to avoid using an index.

```
WHERE DAYOFMONTH(HIREDATE) = 31
```

As of DB2 9, consider building an index on the expression:

```
CREATE INDEX userid.XDOM  
ON userid.TAB (DAYOFMONTH(HIREDATE))
```

...

Logical to Physical Database Design Decisions are Important!

Start from a well-structured logical database design

- 24
- > 3rd normal form
 - > PK for every entity
 - > Well-formed names



Transform logical to physical and review

- 25
- > Entity to table; attribute to column; etc.
 - ...but differences **CAN** and **WILL** occur.
 - > Don't let data modelers dictate physical design... they're not the ones called when performance suffers.
 - > But don't deviate from 3NF unless performance suffers
 - > Always synchronize the logical model with the physical

Order Columns for Logging

Sequence columns based on logging

- › Infrequently updated non-variable columns first
- › Static (infrequently updated) variable columns
- › Frequently updated columns last
- › Frequently modified together, next to each other

New DB2 9
Format

Reordered
Row Format

Example



Static,
infrequently
updated

Frequently updated at
the same time (marriage)
...but not frequently updated.

Frequently
updated

One table per table space (usually)

- Partitioned or segmented over simple TS
- Do not create base table views

going... going...

Determine amount of free space

- PCTFREE & FREEPAGE
- Based on volatility: don't just default everything to 10
- Give enough space to grow between REORGs
- Static tables? No free space, please.

Compression versus VARCHAR?

- Compression = less overhead
- Compression requires no programmatic handling



Create Appropriate Indexes!

A proper indexing strategy can be the #1 factor to ensure optimal performance

First take care of unique/PK constraints

Then for foreign keys (usually)

Heavily used queries - predicates

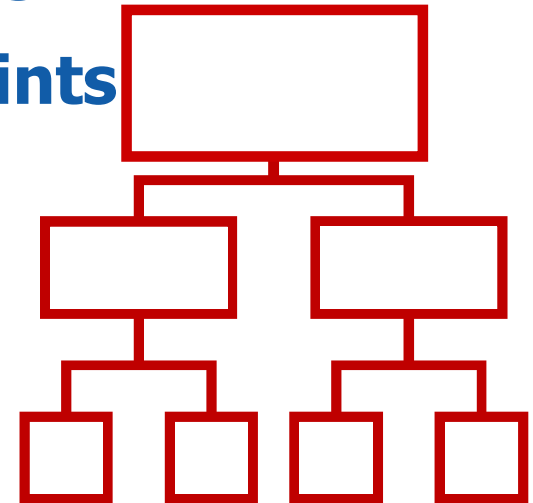
Overloading of columns for IXO

Index to avoid sorting

ORDER BY, GROUP BY

Consider modification implications (INS / UPD / DEL)

Any indexes that are not used?



Any Indexes Not Used?

```
SELECT DISTINCT TBNAME, NAME AS INDEXNAME
FROM   SYSIBM.SYSINDEXES
WHERE  DBNAME LIKE '<database-name>%'
AND    NAME NOT IN
      (SELECT BNAME
       FROM SYSIBM.SYSPACKDEP
       UNION
       SELECT BNAME
       FROM SYSIBM.SYSPLANDEP) ;
```

Catalog
Stats

```
SELECT CREATOR, NAME AS INDEXNAME,
       PARTITION, LASTUSED
FROM   SYSIBM.SYSINDEXSPACESTATS
WHERE  DBNAME LIKE '<database-name>%' ;
```

RTS
V9



Choose Proper Data Type

Store numeric data using a numeric data type

- > INTEGER, SMALLINT, DECIMAL, FLOAT, etc.
- > INTEGER versus DECIMAL(x,0)?
 - Control over domain vs. storage requirements

CHAR for data that does not need arithmetic?

- > Maybe, but filter factors can cause problems
- > Many more possible values for CHAR(5) than SMALLINT

For temporal data, use DATE, TIME, or TIMESTAMP

- > Use DATE instead of CHAR or numeric for dates
- > "DATE and TIME" versus TIMESTAMP
 - Ease of use/storage vs. precision/arithmetic

DB2 9 adds BIGINT, DECFLOAT, BINARY, VARBINARY, and XML data types

Reorganize Wisley

Be sure to run RUNSTATS

- › as data volume changes, new data structures added
- › followed by (RE)BIND with /EXPLAIN(YES)

Review statistics (RTS) to determine when to REORG

- › NEARINDREF and FARINDREF
- › LEAFDIST, PERCDROP
- › For clustering indexes
 - NEAROFFPOSF and FAROFFPOSF
 - CLUSTERRATIOF
- › Use the Real Time Statistics!
- › Analyze access patterns before reorganizing
 - Random vs. sequential
 - Consider automation

**DON'T JUST REORG WEEKLY
OF MONTHLY**

Reorganizing indexes is important, too



Use Multiple Buffer Pools

Don't throw everything in BP0

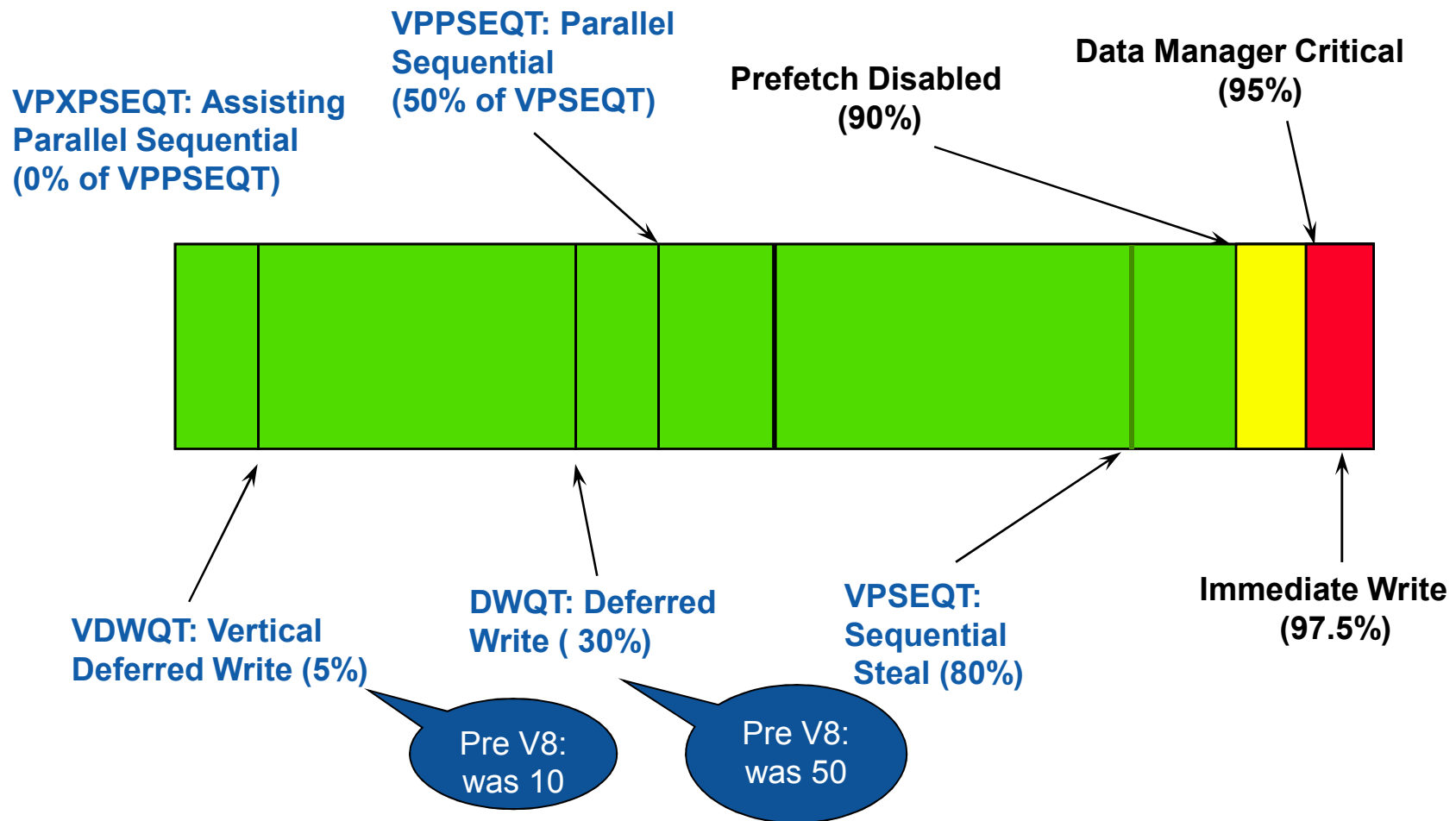
- > isolate the catalog in BP0
- > separate indexes from table spaces
- > optimize BP strategy for your processing: sequential vs. random
- > isolate heavily hit data
 - consider pinning frequently accessed object(s) in memory
- > isolate sort work area
- > consider reserving a buffer pool for tuning

DB2 provides up to 80 buffer pools – *use them*

- 4K: BP0 thru BP49
- 8K: BP8K0 thru BP8K9
- 16K: BP16K0 thru BP16K9
- 32K: BP32K thru BP32K9



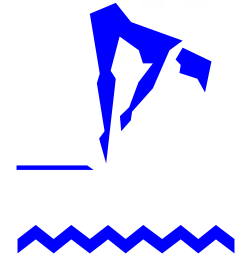
Understand and Adjust Your Buffer Pool Thresholds



Advice on Those Other Pools

EDM Pool – actually multiple pools as of V8

- > **EDMPOOL**: EDM Pool stores only CTs, PTs; (SKCTs and SKPTs in v8)
- > **EDMDBDC**: DBDs
- > **EDMSTMTC**: Cached Dynamic Statements
- > **EDM_SKELETON_POOL**: SKCT, SKPT (skeletons)
- **General ROT - 80% efficiency**



RID Pool – used for enforcing unique keys while updating multiple rows and sorting RIDs during list prefetch, multiple index access, hybrid joins

- > **RID Pool Overflow**: requests exceed ZPARM or DBM1 address space size
 - More than 16 million RID entries used or a single SQL statement consumes more than 50% of the RID Pool
 - The SQL request causing the condition gets a -904.
 - Should not occur frequently; perhaps more or better indexes required.
- > **RDS Limit**: RID list greater than 25% of the rows in the table; TS scan
 - Determined at BIND time
 - You can disable access paths requiring RID pool by setting RID pool size to 0.
 - If you do, don't forget to **REBIND** to change access paths that require RID pool

IFCID 0125



Sort Performance

The better sorted the data is originally, the more efficient the sort will be.

If the data fits into the sort pool, workfiles are not required.

The sort pool is the maximum size of the sort work area allocated for each concurrent sort user.

- > Default is 2MB.

Allocate additional physical work files in excess of the defaults, and put those work files in their own buffer pool (e.g. BP7).

- > At least 5, sized the same, with no secondary.

If the buffer pool deferred write threshold (DWQT) or data set deferred write threshold (VDWQT) are reached, writes are scheduled (for sorts requiring the work file database).

- > For a large sort using many logical work files, this can be difficult to avoid, even if a very large buffer pool is specified.



Minimize Amount of Data You Need to Sort

DB2 uses a tournament sort (*more efficient*) unless...

If Sorted Record > 4075, DB2 will use a tag sort (*less efficient*) because the data no longer fits on a 4K page:

- > Data to sort put directly into 32K workfile
 - For this reason be sure to always allocate at least one 32K workfile in DSNDB07
- > Keys + RID are sorted
- > Data retrieved from the sort using the RID

So, limit the columns on your ORDER BY to only those needed

- > Similar advice for DISTINCT, GROUP BY, UNION, and anything that kicks off a sort

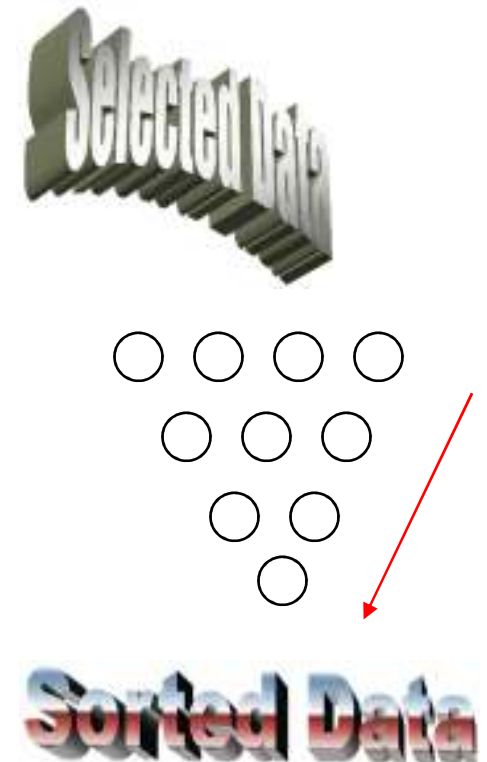
Tournament Sort

DB2 generally uses a Tournament Sort

- › Built into the hardware microcode and very efficient

How Does it Work?

- › Input data to be sorted passes through a tree structure
- › At each level in the tree the data is compared to data already there
- › The 'winner' (lowest value for an ASC) moves up the tree
- › At the top of the tree, the sorted entries are placed into runs
- › Winning entries are removed from the tree and the next value inserted
- › If there is more than one run, the runs must be merged



backup material



Set System Checkpoint

Periodically DB2 takes a checkpoint, containing:

- Currently open unit of recoveries (UR) within DB2, all open page sets, a list of page sets with exception states, and a list of page sets updated by any currently open UR
- Dirty pages are written out at checkpoint – so make sure DWQT is sized correctly!

Specified in the **CHKFREQ*** parameter in DSNZPARM

- Number of log records written
- Or, as of V7, number of minutes

Can be changed dynamically using:

- SET LOG or *(temporary)*
- SET SYSPARM (V7) *(permanent)*

*5 minute intervals
for checkpoints during
peak processing times.*

Other Interesting Settings

Locking Stuff

DSNZPARAMs

NUMLKTS – max locks user per TS

NUMLKUS – max locks per user

DDL

MAXLOCKS – max locks per TS

Thread/User Stuff

CTHREAD – max users

CONDBAT, MAXDBAT – max remote connected, active

IDFORE, IDBACK – max TSO, batch

Logging Stuff

LOGAPSTG – storage for fast log apply

TWOACTV

TWOARCH

TWOBSDS

Other Stuff

DSMAX – maximum number of open data sets

IFCID 0106

Other Interesting Settings

PADIX

LOGEXTS1

There are lotsa ZPARMS!

Locking Stuff

DSNZPARMS
NUMLKT - max locks per user
NUMLKI - max locks per user

DYNRULES

STATSINT

DDL

SYSADM, SYSADM2

MAYLOCK - max locks per

DSMAX

STAT

Thread Usage

ABIND

CTHREAD - max users
CONDBAT, MAYDBAT - max remote connected, active
NEWFUN - max TSO, batch

IXQTY

SECQTY

POOLINAC

OPTSUBQ1

Logging Stuff

LOGAPSTG - storage for fast
TWOACTV
TWOARCH

PARTKEYU

CACHEDYN

NPGETHRSEH

Other

D - maximum number
of

EXTSEC

LOGAPSTG

LOGID 0106

ENSCHHEME



Don't Forget the DB2 Catalog

RUNSTATS on the Catalog

REORG when necessary

Do not issue DDL during peak hours

Do not BIND/REBIND during peak hours

Eliminate redundant and outdated authorization



Utilize the information in the DB2 Catalog to help performance tuning efforts:

- Statistics used by the optimizer
- Indexes defined & type of index
- Average sizes (plan, package, DBD) for pool sizing



Basic DB2 Performance Monitoring Guidelines

Minimum Traces: Accounting Class 1 and 2; Statistics Class 1

- > Accounting Class 3 is good, too: tracks DB2 wait time and can prove helpful to track I/O issues & problems external to DB2
- > Accounting Class 7, 8: package level information (overhead!)

STATTIME 1

- > One minute intervals; trending (1440 intervals per day)

Consider adding a SQL performance monitor

- > To provide a comprehensive view into what is running

Consider adding an access path / BIND management solution

- > To get guidance on how to improve SQL

Don't forget PLAN_TABLE analysis

- > Dynamic statement cache:
DSN_STATEMENT_CACHE_TABLE



The DB2 Traces

| Trace | Started by | Description |
|-------------|-------------------------|--|
| Accounting | DSNZPARM or START TRACE | Records performance information about the execution of DB2 application programs |
| Audit | DSNZPARM or START TRACE | Provides information about DB2 DDL, security, utilities, and data modification |
| Global | DSNZPARM or START TRACE | Provides information for the servicing of DB2 |
| Monitor | DSNZPARM or START TRACE | Records data useful for online monitoring of the DB2 subsystem and DB2 application programs |
| Performance | START TRACE | Collects detailed data about DB2 events, enabling database and performance analysts to pinpoint the causes of performance problems |
| Statistics | DSNZPARM or START TRACE | Records information regarding the DB2 subsystem's use of resources |

backup material



General Monitoring Guidelines

Do not overdo monitoring and tracing.

- Excessive overhead for monitoring can cause problems in an otherwise good subsystem.

Plan & implement several types of monitoring strategies at your shop:

1. Ongoing, regular performance monitoring activities to uncover problems and exceptions, and;
2. Procedures for monitoring exceptions after they have been observed.

Before taking any corrective actions, make sure you have adequately defined the actual problem!

Know your performance history.



Some Additional Advice

Re-evaluate all of your shop standards.

There's *rarely* A PROBLEM WITH DB2!

Start with what changed.

- > Then check the application...
- > ...followed by the database...
- > ...then the subsystem...
- > ...and finally the OS and surrounding environment (network, disk, etc.).

Remove outdated & "bad" standards. *e.g.) no more than 3 indexes per table.*

Do one thing at a time and you can tune DB2 performance...

Let's Make Sure We All Leave With the Right Mindset

“Not everything that can be counted counts, and not everything that counts can be counted.” – *Albert Einstein*

“The problem is not that there are problems. The problem is expecting otherwise and thinking that having problems is a problem.” – *Theodore Rubin*

“The measure of success is not whether you have a tough problem to deal with, but whether or not it is the same problem you had last year.” - *John Foster Dulles*

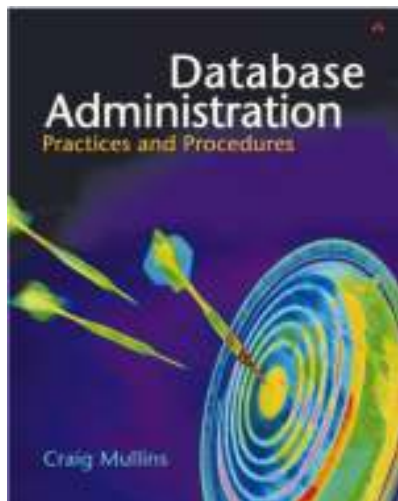
Here's hoping that the rest of your year is full of brand new problems!

Contact Information



Phone: 281-920-3305
info@cdbsoftware.com

<http://www.cdbsoftware.com>



Questions?
info@cdbsoftware.com

Craig S. Mullins
Mullins Consulting, Inc.
15 Coventry Court
Sugar Land, TX 77479
craig@craigsmullins.com
<http://www.craigsmullins.com>

<http://www.craigsmullins.com/cm-book.htm>