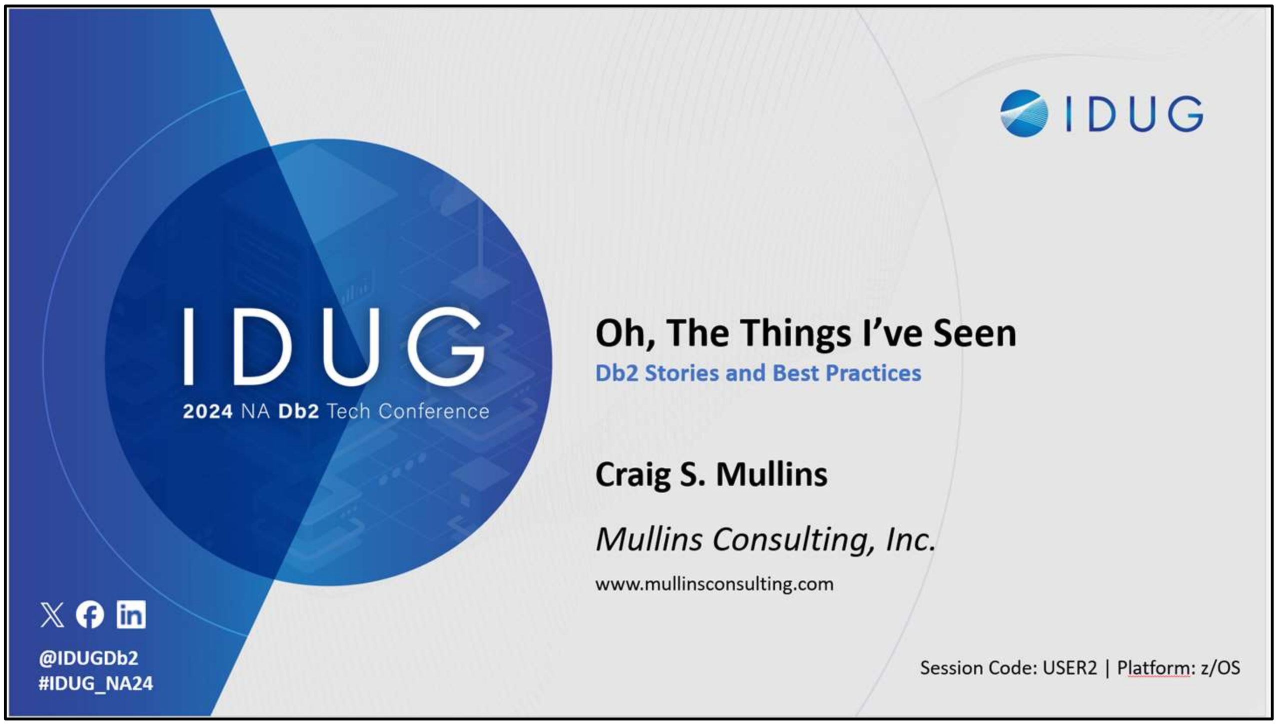


IDUG

2024 NA Db2 Tech Conference

Oh, The Things I've Seen

Db2 Stories and Best Practices

Craig S. Mullins

Mullins Consulting, Inc.

www.mullinsconsulting.com

@IDUGDb2
#IDUG_NA24

Session Code: USER2 | Platform: z/OS

Oh, The Things I've Seen
Db2 Stories and Best Practices

Craig S. Mullins
Mullins Consulting, Inc.
www.mullinsconsulting.com

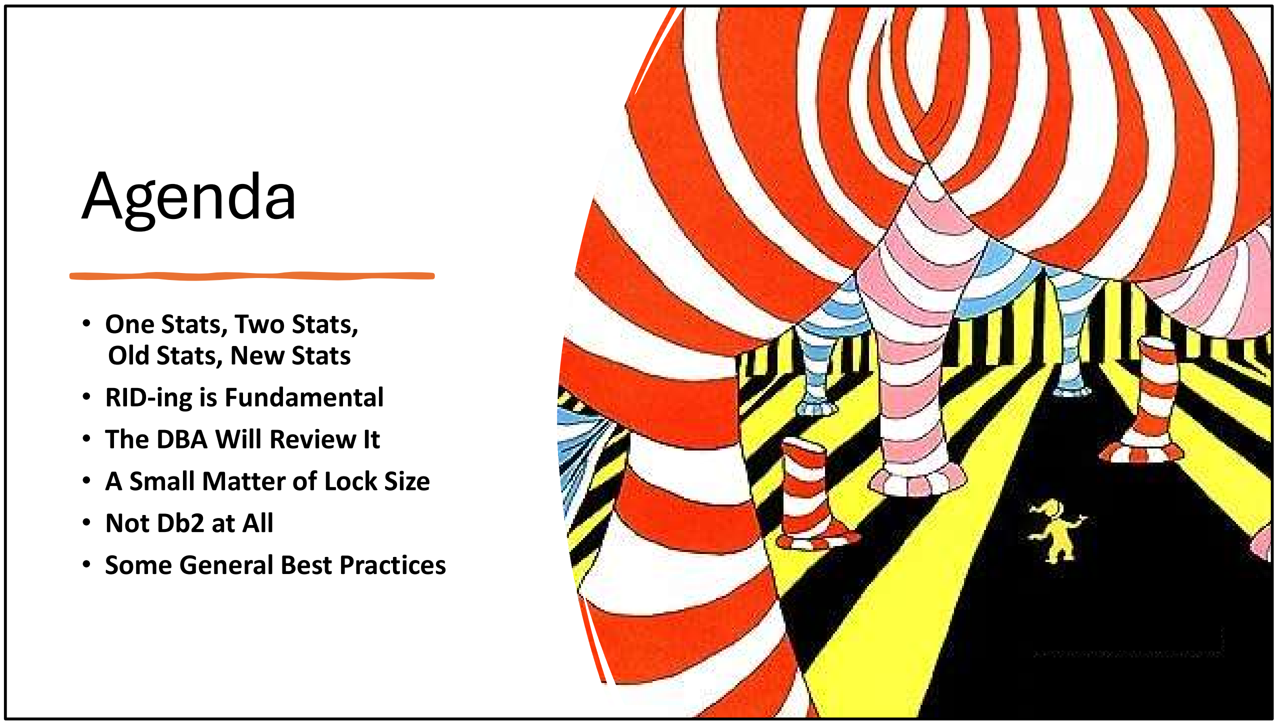
IBM Champion for Data and AI
IBM Gold Consultant

Oh, The Things I've Seen

Db2 Stories and Best Practices

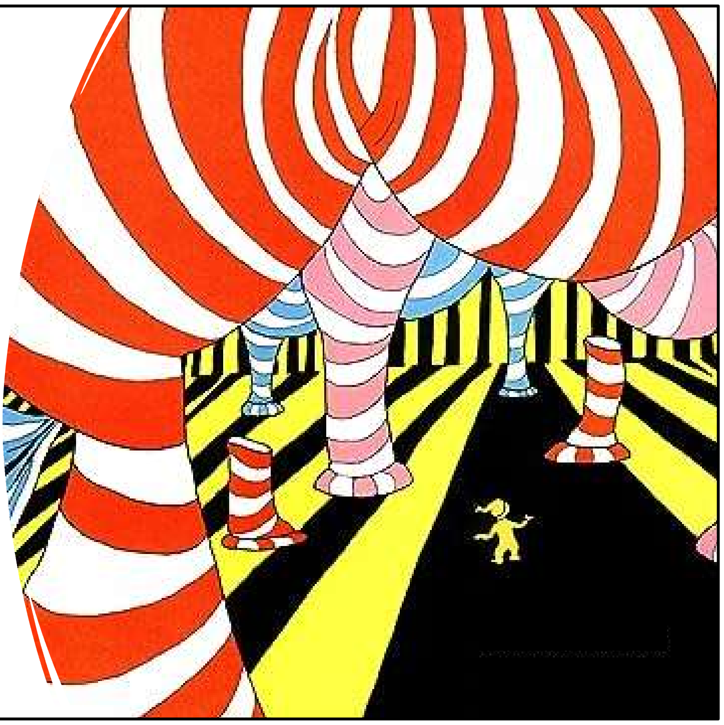
By Craig S. Mullins
(with a nod to Dr. Seuss)

Join me on a Suess-ian journey through some of the things I've seen during my days as a Db2 consultant



Agenda

- One Stats, Two Stats, Old Stats, New Stats
- RID-ing is Fundamental
- The DBA Will Review It
- A Small Matter of Lock Size
- Not Db2 at All
- Some General Best Practices



Along the way we will discuss several interesting things I've seen. We'll try to have some fun, while at the same time making sure we talk about actual Db2 activities and issues... and bring it all back around to the best practices that can eliminate those "things I've seen."

Reading the agenda here, you might be able to guess some of the things I've seen... but maybe not all of them. So join me if you will...

One Stats, Two Stats, Old Stats, New Stats

Dealing With Outdated RUNSTATS

And we will start our travels by discussing Db2 RUNSTATS

Things I've Seen…

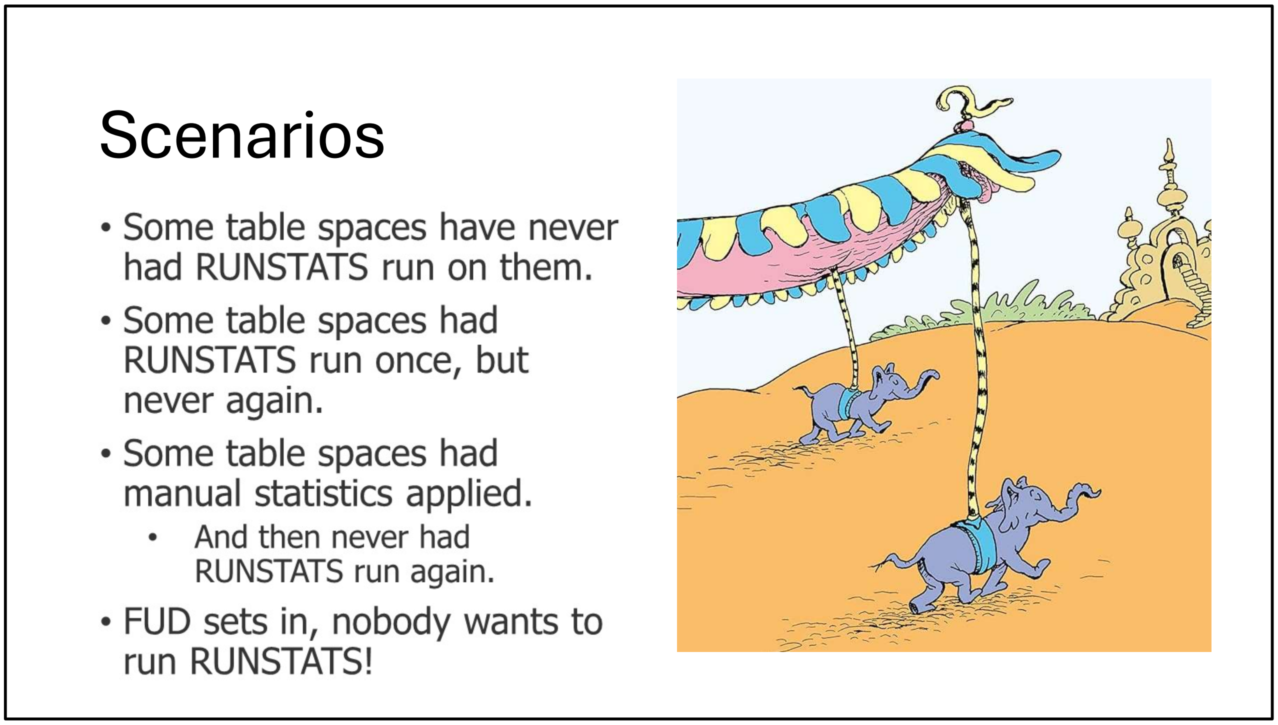
RUNSTATS aren't run,

Now RUNSTATS are old,

But RUNSTATS aren't needed,

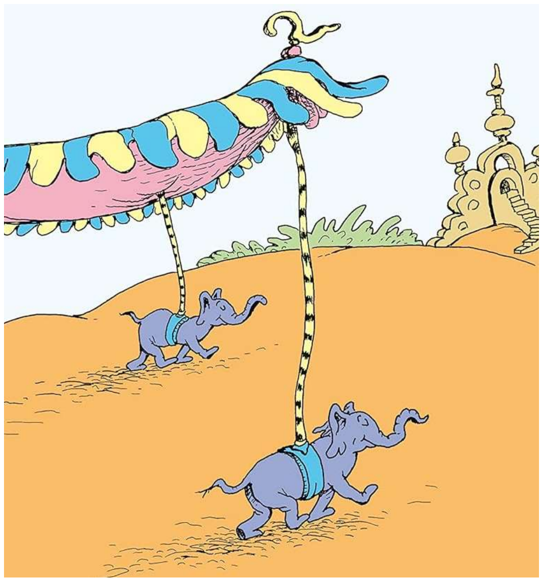
Or so we've been told!

So, what have I seen with RUNSTATS. A lot of times, I see RUNSTATS not being run. But why?



Scenarios

- Some table spaces have never had RUNSTATS run on them.
- Some table spaces had RUNSTATS run once, but never again.
- Some table spaces had manual statistics applied.
  - And then never had RUNSTATS run again.
- FUD sets in, nobody wants to run RUNSTATS!

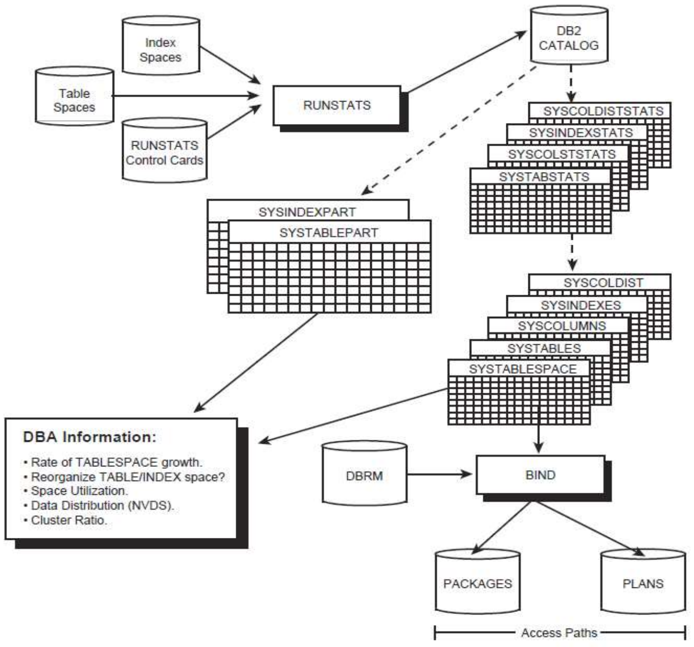


Sometimes I see table spaces with the dreaded -1 statistics, and that means that RUNSTATS were never run. And I see this not just on newer objects, but recently on table spaces that were over 10 years old!

Sometimes RUNSTATS are not run because the organization has been burned in the past (REORG/RUNSTATS/REBIND) with bad performance. So they get to a point where things are running OK and just stop running RUNSTATS.

Another frequent issue I see is when RUNSTATS are run, then some manuals statistics are applied to get the access paths needed. Yes, this may have happened 5, 10, 15, or more years ago. A good implementation of this approach will still schedule RUNSTATS but then add a step to make the manual changes. A bad implementation will just stop running RUNSTATS. And then lose the list of table spaces with manual stats, and then just spread FUD around so that RUNSTATS do not get run again or we'll go back to that problem we had in 1998.

- Scans data
- Gathers details on data
- Records statistics in the Db2 Catalog
- Used to generate access paths
- Can be used by DBAs, too
  - RTS usually more accurate



So what is RUNSTATS? RUNSTATS is a utility that scans your data and updates statistical metadata in the Db2 Catalog about your objects. This includes things like number of number of records, number of pages, average record length, number of key values, column cardinality, and so on. The Db2 optimizer uses these statistics to determine access paths to the data.

Real Time Stats are different and can be used by DBAs and tools to automate maintenance tasks. Like when to run RUNSTATS!

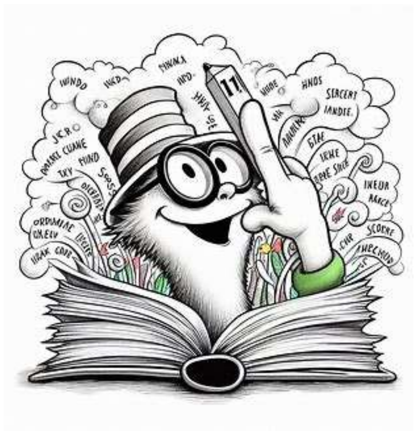
Table Statistics

- CARDF
  - **Number of rows in the table (or partition)**
- NPAGESF
  - **Number of pages containing data**
- NACTIVEF
  - **Number of active pages in the table space**
- PCTROWCOMP
  - **Percentage of rows that are compressed**

Some statistics provide data on tables

Index Statistics

- **NLEAF – number of active leaf pages**
- **NLEVELS – number of levels in the index**
- **CLUSTERRATIOF – the percentage of rows that are in clustered order**
- **CLUSTERING – whether the index is the clustering index**
- **FIRSTKEYCARDF – number of distinct values for the first column of the index**
- **FULLKEYCARDF – number of distinct values for the complete index key (all columns in the key)**



Some statistics provide data on indexes

Column Statistics

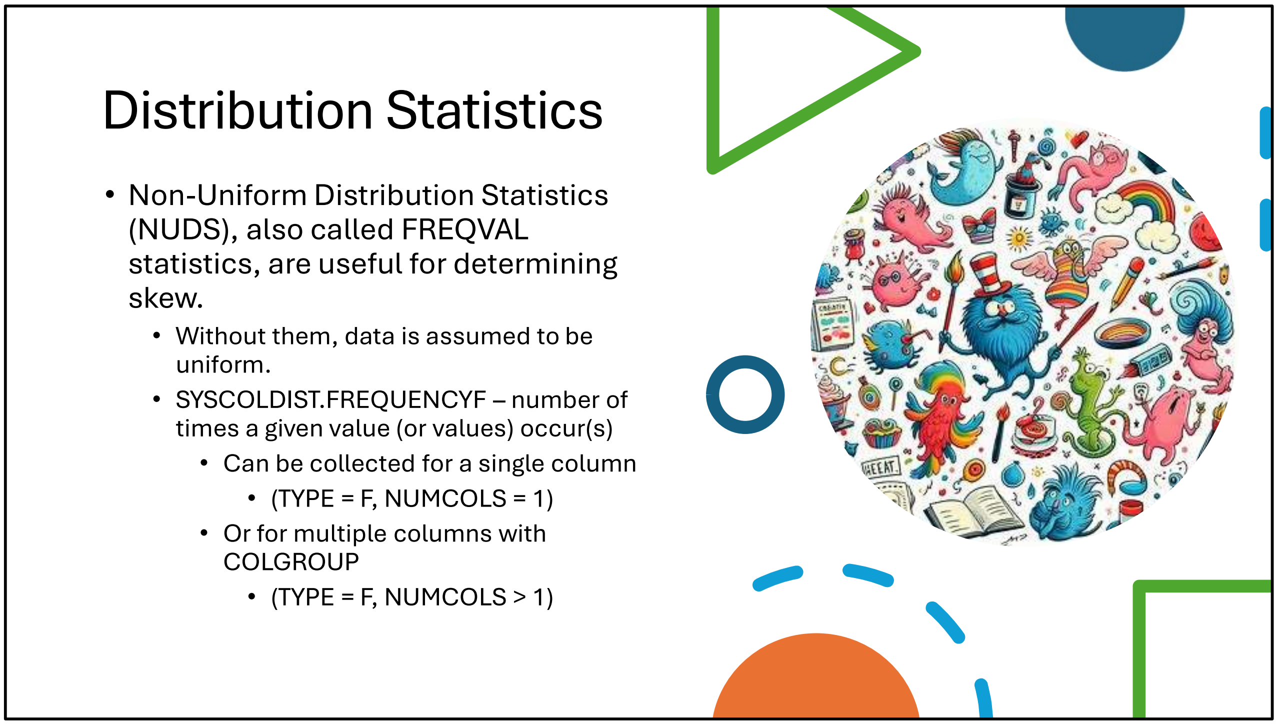
Single Column

- COLCARDF – Number of distinct values for a column
    - Assumes data is uniformly distributed
- HIGH2KEY – Second highest key value
- LOW2KEY – Second lowest key value

Multiple Columns

- SYSCOLDIST.CARDF – Number of distinct values for a group of column
    - Useful for correlated data
    - (TYPE=C, NUMCOLS>1)

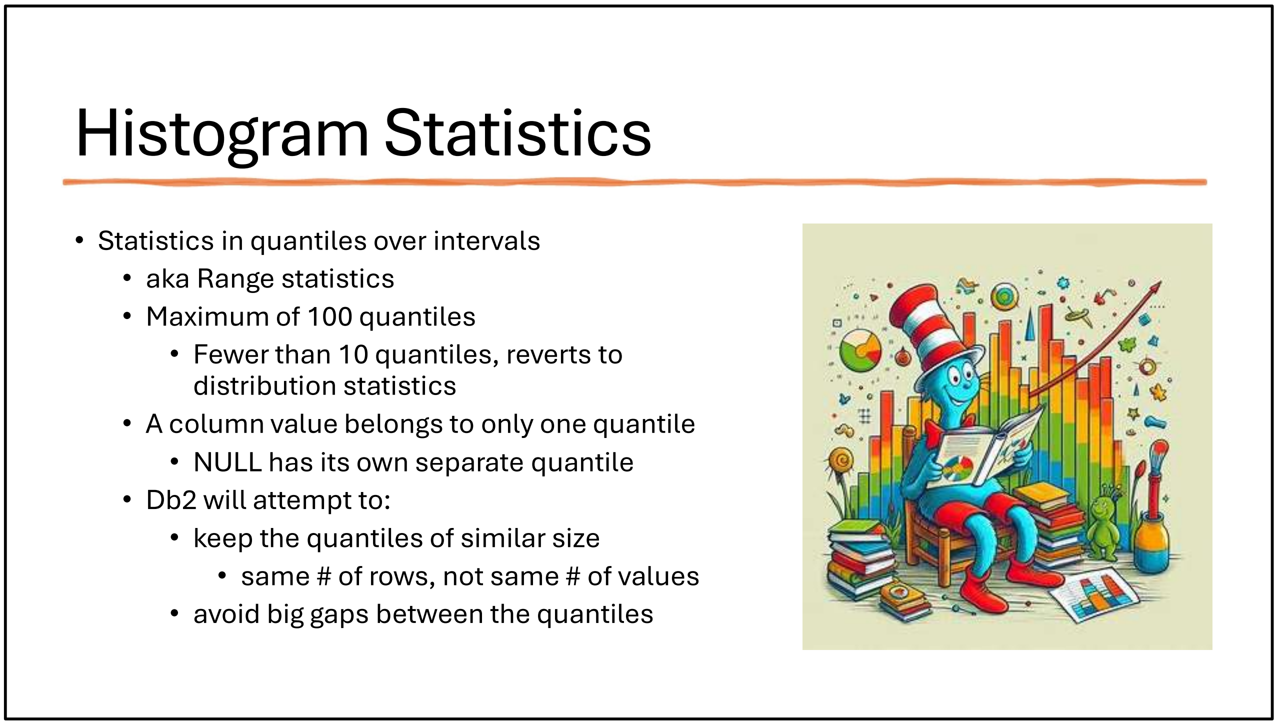
And still other statistics provide data on columns



Distribution Statistics

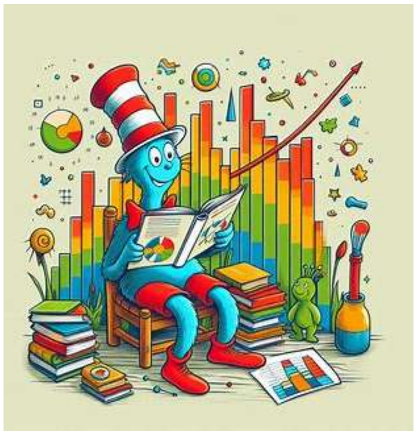
- Non-Uniform Distribution Statistics (NUDS), also called FREQVAL statistics, are useful for determining skew.
  - Without them, data is assumed to be uniform.
  - SYSCOLDIST.FREQUENCYF – number of times a given value (or values) occur(s)
    - Can be collected for a single column
      - (TYPE = F, NUMCOLS = 1)
    - Or for multiple columns with COLGROUP
      - (TYPE = F, NUMCOLS > 1)

Then there are statistics for special situations



Histogram Statistics

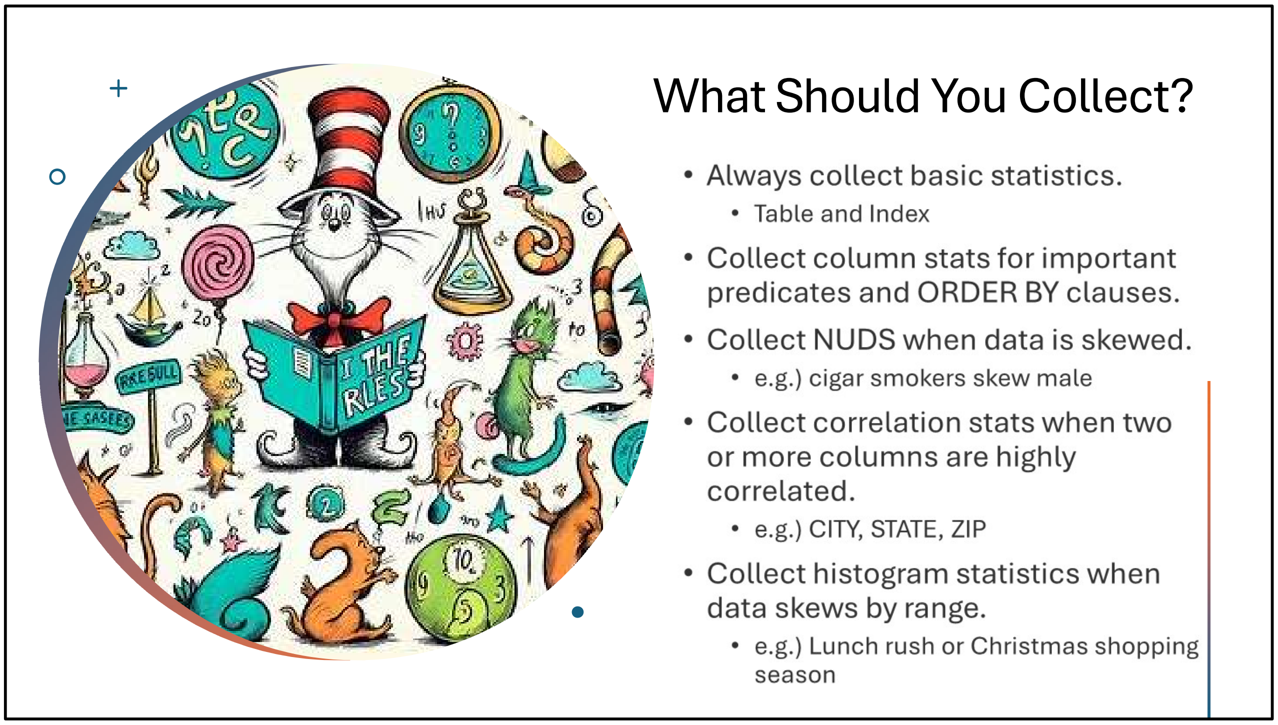
- Statistics in quantiles over intervals
    - aka Range statistics
    - Maximum of 100 quantiles
        - Fewer than 10 quantiles, reverts to distribution statistics
    - A column value belongs to only one quantile
        - NULL has its own separate quantile
    - Db2 will attempt to:
        - keep the quantiles of similar size
            - same # of rows, not same # of values
        - avoid big gaps between the quantiles



Another special situation is handled by histogram statistics. A histogram is a way of summarizing data that's measured on an interval scale. A histogram is particularly helpful when you want to highlight how data is distributed, to determine if data is symmetrical or skewed, and to indicate whether or not outliers exists.

The histogram is appropriate only for variables whose values are numerical and measured on an interval scale. To be complete, let's define interval: a set of real numbers between two numbers either including or excluding one or both of them. Histograms are generally used when dealing with large data sets. And you will be interested in histogram statistics because they can be quite useful to the Db2 optimizer for certain types of data and queries. Instead of the frequency statistics, which are collected for only a subset of the data, sometimes Db2 can improve access path selection by estimating predicate selectivity from histogram statistics, which are collected over all values in a table space.

Consider collecting histogram statistics to improve access paths for troublesome queries with RANGE, LIKE, and BETWEEN predicates. They also can help in some cases for equality (=), IS NULL, IN LIST, and COL op COL predicates.



What Should You Collect?

- Always collect basic statistics.
  - Table and Index
- Collect column stats for important predicates and ORDER BY clauses.
- Collect NUDS when data is skewed.
  - e.g.) cigar smokers skew male
- Collect correlation stats when two or more columns are highly correlated.
  - e.g.) CITY, STATE, ZIP
- Collect histogram statistics when data skews by range.
  - e.g.) Lunch rush or Christmas shopping season

So, what is a good rule of thumb for what to collect???

First of all, you should be running RUNSTATS regularly if your data is changing. And most data changes, right?

Then, always collect basic statistics, for tables and indexes. Next, collect column stats for important predicates and ORDER BY clauses. Collect non-uniform distribution statistics when data is skewed, for example, cigar smokers skew male. And collect correlation stats when two or more columns are highly correlated, such as CITY, STATE, and ZIP code. Finally, consider collecting histogram statistics when data skews by range, such as to capture events like a Lunch rush or Christmas shopping season

RUNSTATS Guidelines (1)

- Collect RUNSTATS for all indexes

```
RUNSTATS TABLESPACE DSN8D81A.DSN8S81D
    INDEX (ALL)
```

- Collect RUNSTATS for columns in sensitive WHERE clauses

```
RUNSTATS TABLESPACE DSN8D81A.DSN8S81E
  TABLE (DSN8810.EMP)
  COLUMN (FIRSTNME, LASTNAME, SALARY)
```

Here we see some examples of collecting what I just recommended.

The first example shows collecting RUNSTATS on all indexes of tables in the identified table space.

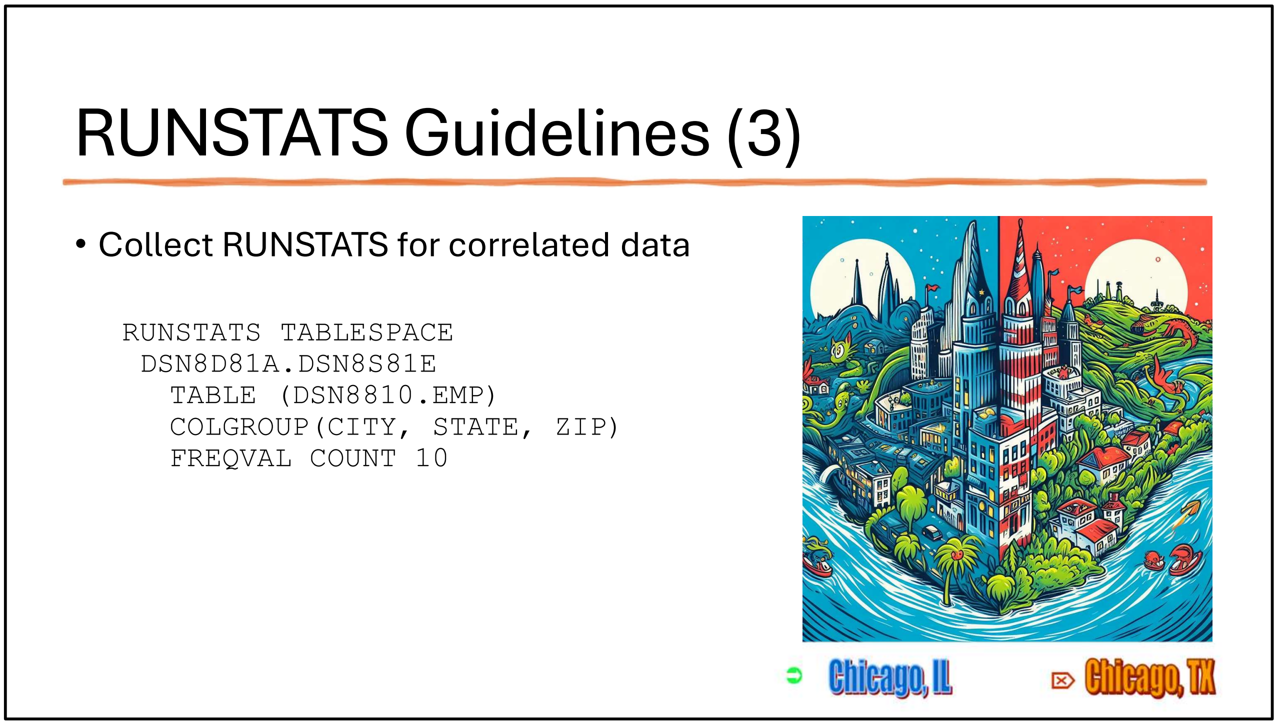
Then we show an example for collecting stats for named columns in sensitive WHERE clauses.

RUNSTATS Guidelines (2)

- Collect RUNSTATS for skewed data

```
RUNSTATS TABLESPACE
DSN8D81A.DSN8S81E
  TABLE (DSN8810.EMP)
  COLGROUP(JOB) FREQVAL COUNT 10
```

Here we see an example where the JOB column is skewed. That is, there will be more worker bees than there will be managers, or CxOs. So we capture that using FREQVAL statistics.



RUNSTATS Guidelines (3)

- Collect RUNSTATS for correlated data

```
RUNSTATS TABLESPACE
 DSN8D81A.DSN8S81E
   TABLE (DSN8810.EMP)
   COLGROUP(CITY, STATE, ZIP)
   FREQVAL COUNT 10
```

Chicago, IL

Chicago, TX

Here we capture information on correlated data – like CITY, STATE, and ZIP – using a COLGROUP with FREQVAL.

RUNSTATS Guidelines (4)

- Collect Histogram RUNSTATS for range skews
  - Stores QUANTILENO, LOWVALUE, and HIGHVALUE for up to 100 quantiles

```
RUNSTATS INDEX (OWNER.XTRG)
   HISTOGRAM NUMCOLS 2
```

e.g.) DATE, TIMESTAMP

Breakfast	6:00-6:45	Slow
	6:46-8:10	Busy
	8:11-11:25	Slow
Lunch	11:25-1:15	Busy
	1:16-1:45	Moderate
	1:46-4:55	Slow
Dinner	4:56-5:45	Moderate
	5:46-7:15	Busy
	7:16-11:00	Slow

And here is an example collecting histogram statistics.

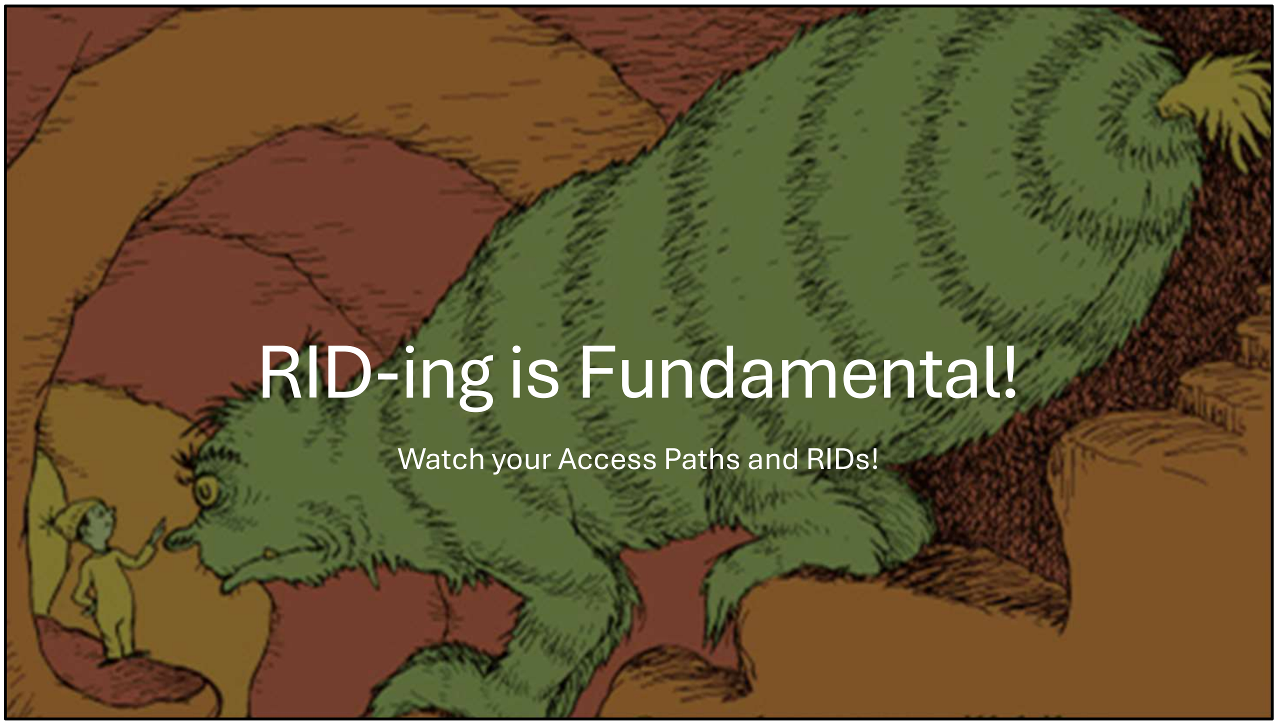
You can tell RUNSTATS to collect histogram statistics by coding the HISTOGRAM keyword in conjunction with the COLGROUP option. In this way, you can collect histogram statistics for a group of columns. You also must tell DB2 the number of quantiles to collect by specifying the NUMQUANTILES parameter. NUMQUANTILES also can be specified with the INDEX parameter, in which case, it indicates that histogram statistics are to be collected for the columns of the index.

In this case, we want to capture the time periods for a lunch rush and we are collecting histogram statistics for the columns of the index XTRG.

Things I've Seen

Outdated statistics,
a silent foe,
Caused the RID pool
to overflow.
Queries stumbled,
indexes failed,
As SQL's prowess
became impaled.

So, be sure to have a plan for keeping you RUNSTATS up-to-date!

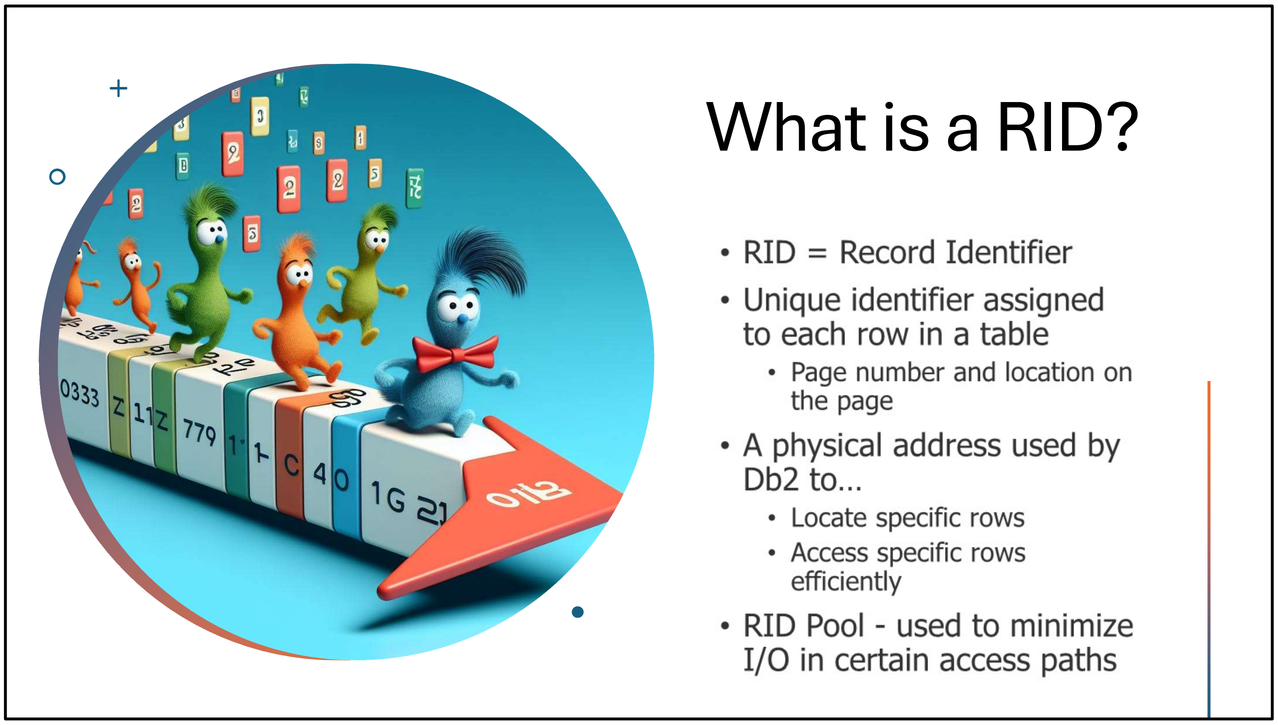


RID-ing is Fundamental!

Watch your Access Paths and RIDs!

RIDs are another area that I've seen cause problems, especially when coupled with a lack of RUNSTATS.

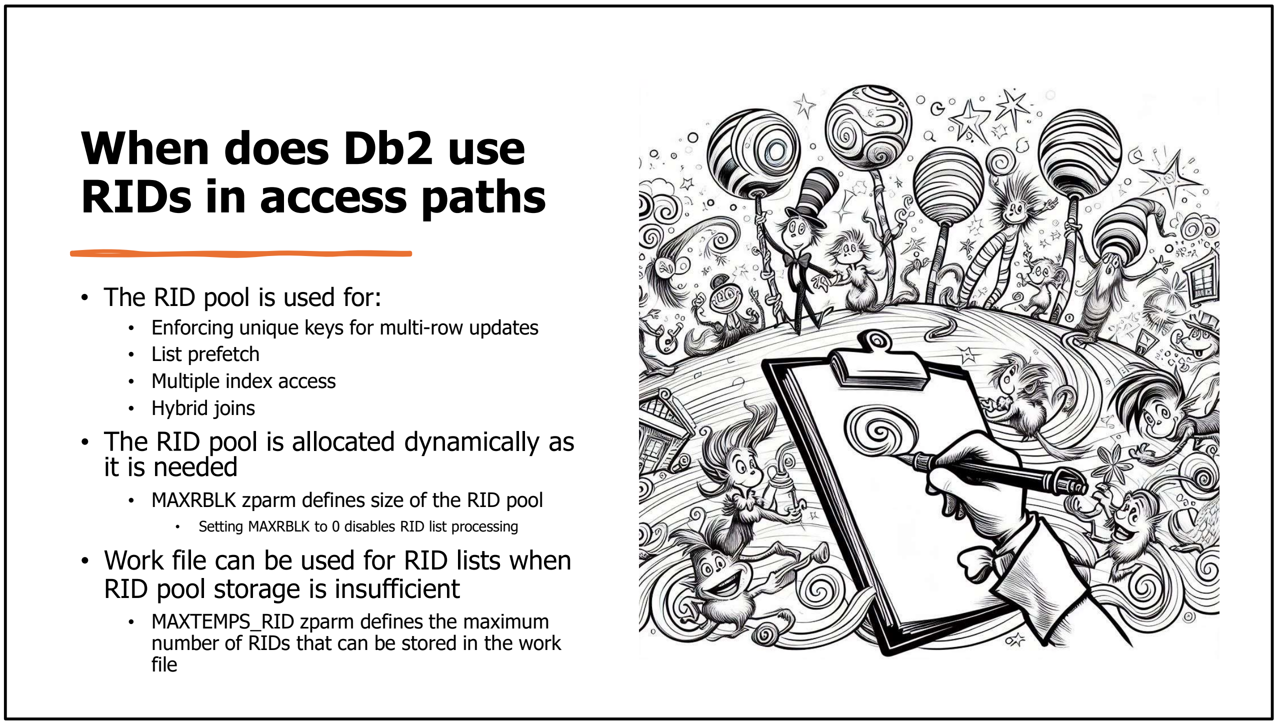
Also, as of Db2 12 for z/OS, RID usage in access paths has increased. Note this blog post that explains why: https://tinyurl.com/listpreDb2-12



What is a RID?

- RID = Record Identifier
- Unique identifier assigned to each row in a table
  - Page number and location on the page
- A physical address used by Db2 to…
  - Locate specific rows
  - Access specific rows efficiently
- RID Pool - used to minimize I/O in certain access paths

In Db2 for z/OS, a RID (**R**ecord **ID**entifier) is a unique identifier assigned to each row within a table. It consists of a page number and a slot number within that page. The RID serves as a physical address that allows Db2 to locate and access specific rows efficiently. RIDs are commonly used in Db2's internal processing for navigating and retrieving data from tables.



When does Db2 use RIDs in access paths

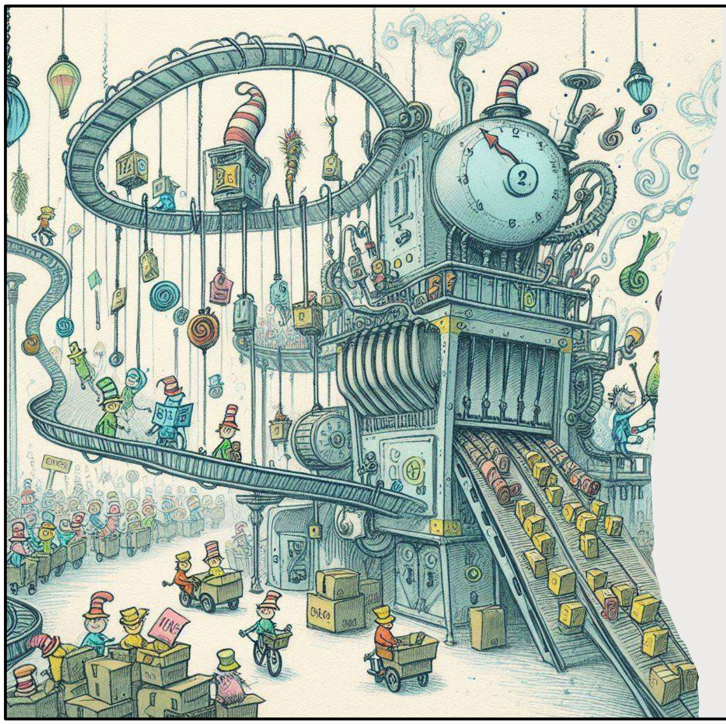
- The RID pool is used for:
  - Enforcing unique keys for multi-row updates
  - List prefetch
  - Multiple index access
  - Hybrid joins
- The RID pool is allocated dynamically as it is needed
  - MAXRBLK zparm defines size of the RID pool
    - Setting MAXRBLK to 0 disables RID list processing
- Work file can be used for RID lists when RID pool storage is insufficient
  - MAXTEMPS_RID zparm defines the maximum number of RIDs that can be stored in the work file

The RID pool is used for all record identifier (RID) processing, including enforcing unique keys for multi-row updates, list prefetch, multiple index access paths, and hybrid joins.

Db2 collects RIDs that match the selection criteria and places them in a list in the RID pool. The list is sorted by page number, which is contained in the RID. DB2 then uses the sorted list to access the table by reading up to 32 pages per I/O and attempting to read ahead one block of 32 pages before use. The RID pool is allocated dynamically as it is needed. The maximum size of the pool is determined by the ZPARM MAXRBLK.

The work file database is used to store a RID list when the RID pool storage cannot contain all the RIDs of the list. When RID pool storage overflow occurs for a RID list, DB2 attempts to store the RID list in work file storage instead of falling back to a relational scan.

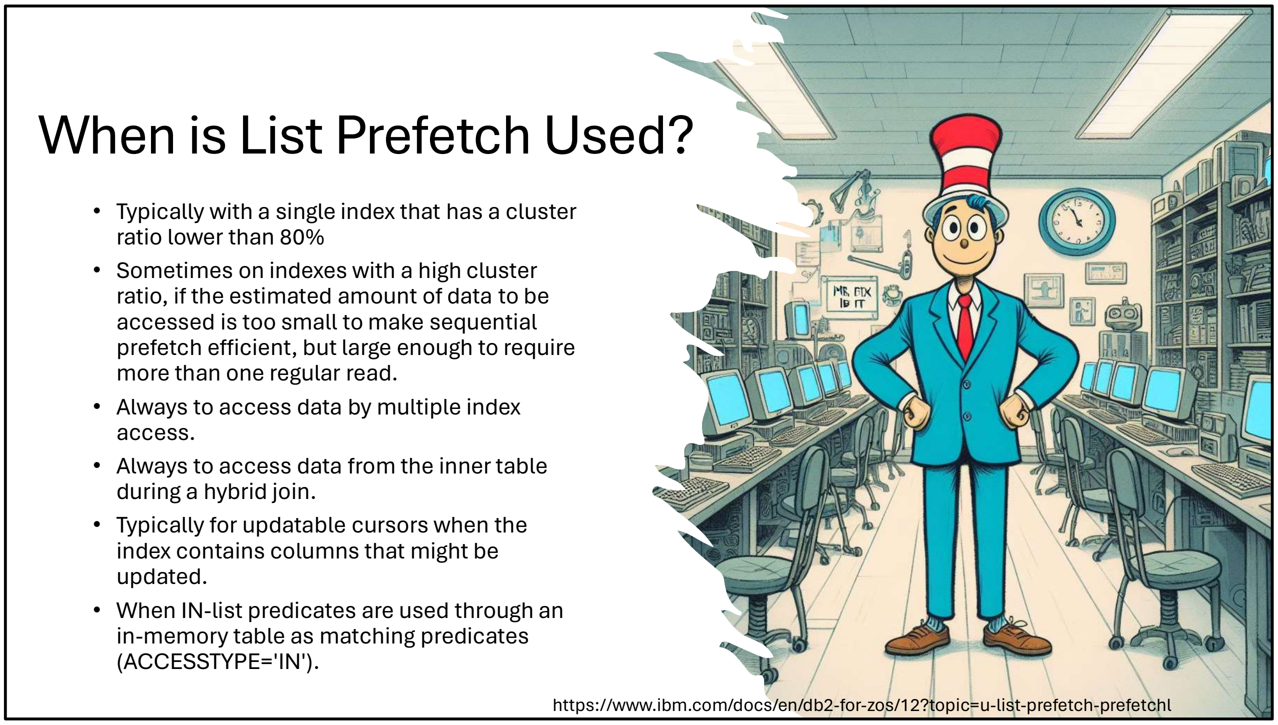
The maximum number of RIDs (measured in RID blocks) that DB2 is allowed to store in the work file database is determined by ZPARM MAXTEMPS_RID. Setting it to NO LIMIT can help to prevent reverting to table space scan when an arbitrary limit for work file usage is reached.



What is List Prefetch?

- When list prefetch is involved in an access path:
  - Db2 retrieves a list of RIDs through a matching index scan on one or more indexes.
  - The list of RIDs is sorted in ascending order by page number.
  - Pages are prefetched in order, using the sorted list of RIDs.

List prefetch **reads a set of data pages determined by a list of record identifiers (RIDs) taken from an index**. List prefetch access paths are ideally suited for queries where the qualified rows, as determined in index key sequence, are not sequential, are skip-sequential but sparse, or when the value of the DATAREPEATFACTORF statistic is large.

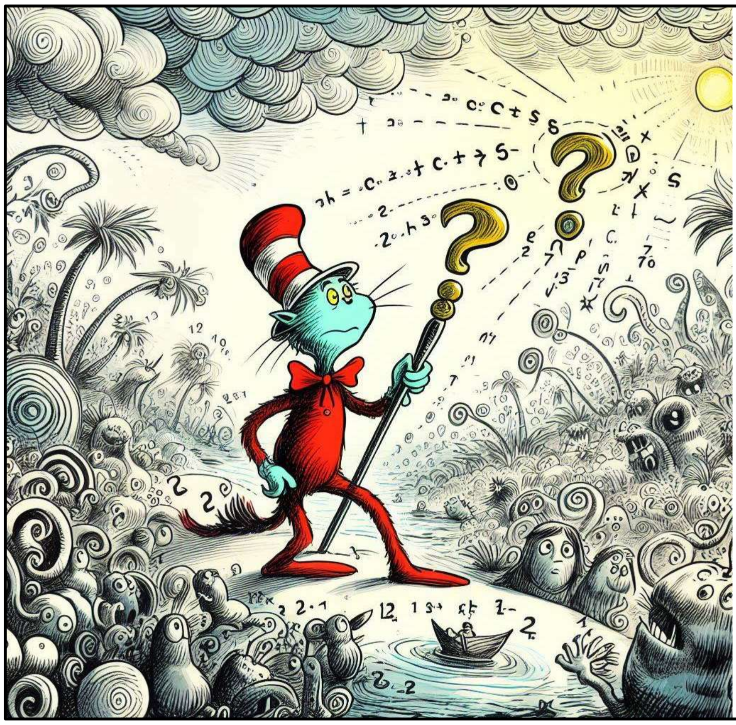


When is List Prefetch Used?

- Typically with a single index that has a cluster ratio lower than 80%
- Sometimes on indexes with a high cluster ratio, if the estimated amount of data to be accessed is too small to make sequential prefetch efficient, but large enough to require more than one regular read.
- Always to access data by multiple index access.
- Always to access data from the inner table during a hybrid join.
- Typically for updatable cursors when the index contains columns that might be updated.
- When IN-list predicates are used through an in-memory table as matching predicates (ACCESSTYPE='IN').

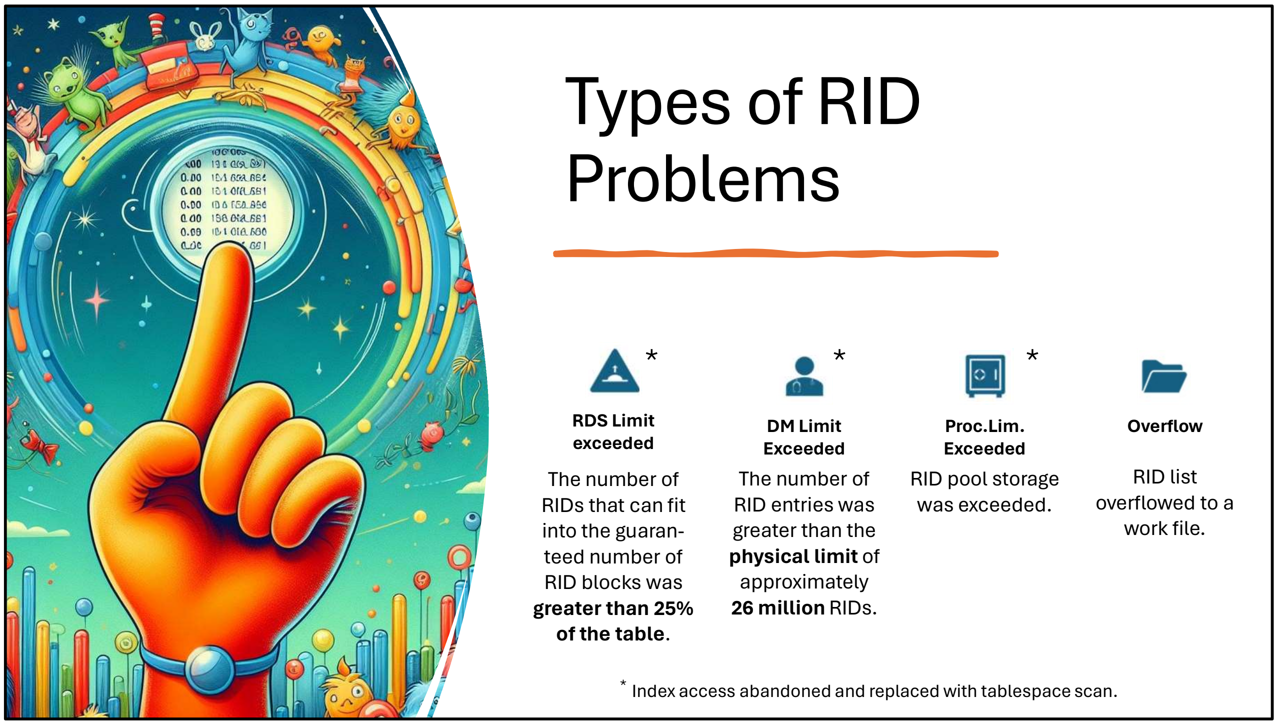
https://www.ibm.com/docs/en/db2-for-zos/12?topic=u-list-prefetch-prefetchl

List prefetch may be used in multiple circumstances, as outlined here. This list comes directly from the IBM Db2 documentation as listed at the URL on the bottom of the slide.



But List Prefetch Can Cause Problems

Although it can be helpful at times, sometimes list prefetch can cause problems, too!



Types of RID Problems

RDS Limit exceeded *

The number of RIDs that can fit into the guaran-teed number of RID blocks was **greater than 25% of the table**.

DM Limit Exceeded *

The number of RID entries was greater than the **physical limit** of approximately **26 million** RIDs.

Proc.Lim. Exceeded *

RID pool storage was exceeded.

Overflow

RID list overflowed to a work file.

* Index access abandoned and replaced with tablespace scan.

RDS Limit exceeded

The number of RIDs that can fit into the guaranteed number of RID blocks was greater than 25% of the table.
Index access abandoned and replaced with tablespace scan.

DM Limit Exceeded

The number of RID entries was greater than the physical limit of approximately 26 million RIDs.
Index access abandoned and replaced with tablespace scan.

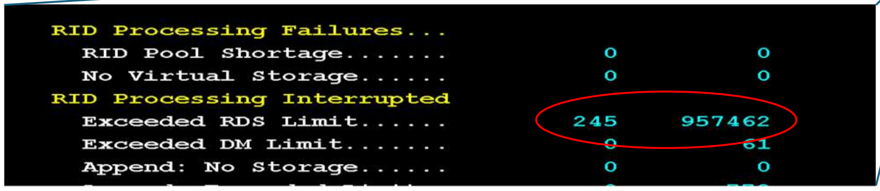
Proc.Lim. Exceeded

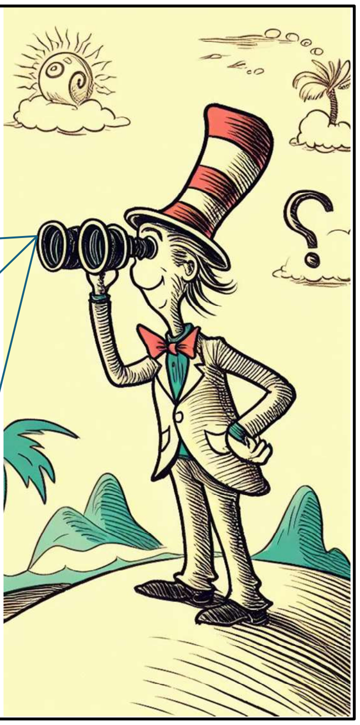
RID pool storage was exceeded.

Overflow

RID list overflowed to a work file.

RDS Problems Identified

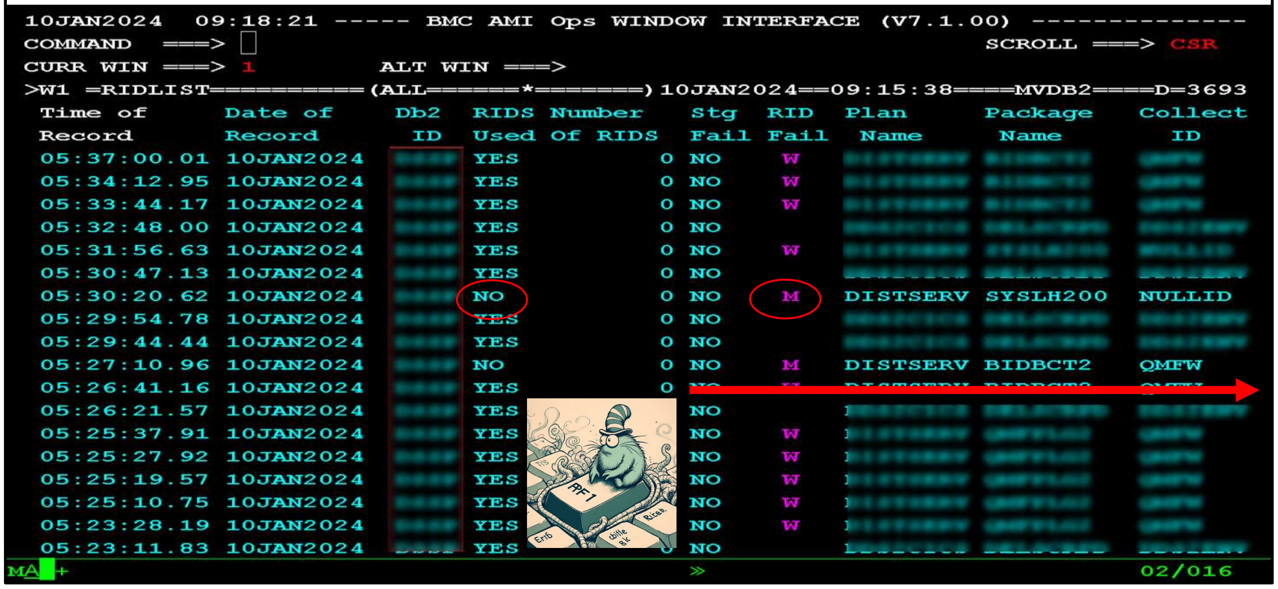

```
RID Processing Failures...
   RID Pool Shortage.......        0          0
   No Virtual Storage......        0          0
RID Processing Interrupted
   Exceeded RDS Limit......      245     957462
   Exceeded DM Limit.......        0         61
   Append: No Storage......        0          0
```



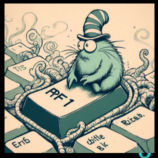
Used a monitor (in this case Mainview) to identify RID issues.

An awful lot of RDS failures were occurring.

RDS Limit Failure Details



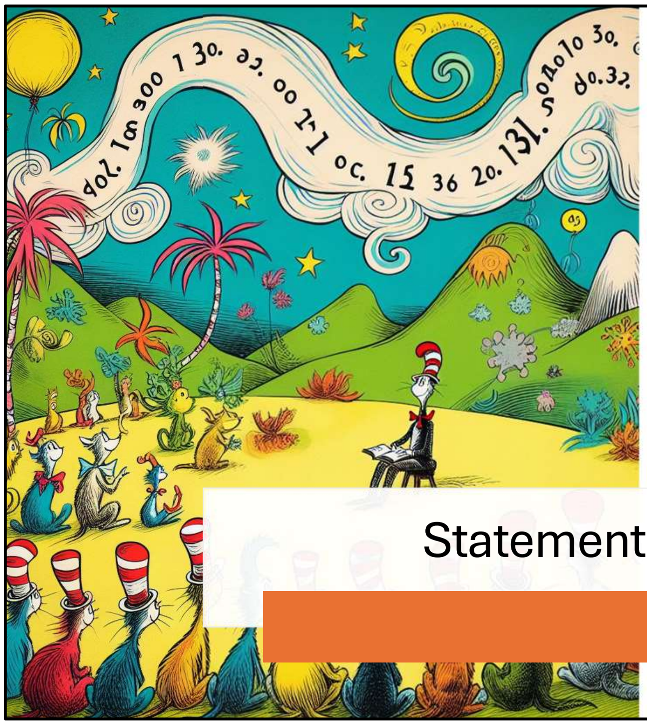
```
10JAN2024   09:18:21 ----- BMC AMI Ops WINDOW INTERFACE (V7.1.00) --------------
COMMAND    ===>                                                 SCROLL ===> CSR
CURR WIN   ===> 1             ALT WIN ===>
>W1 =RIDLIST===========(ALL=======*========)10JAN2024==09:15:38====MVDB2====D=3693
 Time of       Date of     Db2   RIDS Number       Stg  RID  Plan      Package    Collect
 Record        Record       ID   Used Of RIDS      Fail Fail  Name      Name        ID
 05:37:00.01  10JAN2024 [redacted] YES          0  NO   W   [redacted] [redacted] [redacted]
 05:34:12.95  10JAN2024 [redacted] YES          0  NO   W   [redacted] [redacted] [redacted]
 05:33:44.17  10JAN2024 [redacted] YES          0  NO   W   [redacted] [redacted] [redacted]
 05:32:48.00  10JAN2024 [redacted] YES          0  NO       [redacted] [redacted] [redacted]
 05:31:56.63  10JAN2024 [redacted] YES          0  NO   W   [redacted] [redacted] [redacted]
 05:30:47.13  10JAN2024 [redacted] YES          0  NO       [redacted] [redacted] [redacted]
 05:30:20.62  10JAN2024 [redacted] NO           0  NO   M   DISTSERV  SYSLH200   NULLID
 05:29:54.78  10JAN2024 [redacted] YES          0  NO       [redacted] [redacted] [redacted]
 05:29:44.44  10JAN2024 [redacted] YES          0  NO       [redacted] [redacted] [redacted]
 05:27:10.96  10JAN2024 [redacted] NO           0  NO   M   DISTSERV  BIDBCT2    QMFW
 05:26:41.16  10JAN2024 [redacted] YES          0  [illegible]
 05:26:21.57  10JAN2024 [redacted] YES             NO       [redacted] [redacted] [redacted]
 05:25:37.91  10JAN2024 [redacted] YES             NO   W   [redacted] [redacted] [redacted]
 05:25:27.92  10JAN2024 [redacted] YES             NO   W   [redacted] [redacted] [redacted]
 05:25:19.57  10JAN2024 [redacted] YES             NO   W   [redacted] [redacted] [redacted]
 05:25:10.75  10JAN2024 [redacted] YES             NO   W   [redacted] [redacted] [redacted]
 05:23:28.19  10JAN2024 [redacted] YES             NO   W   [redacted] [redacted] [redacted]
 05:23:11.83  10JAN2024 [redacted] YES             NO       [redacted] [redacted] [redacted]
MA +                                               >>                          02/016
```

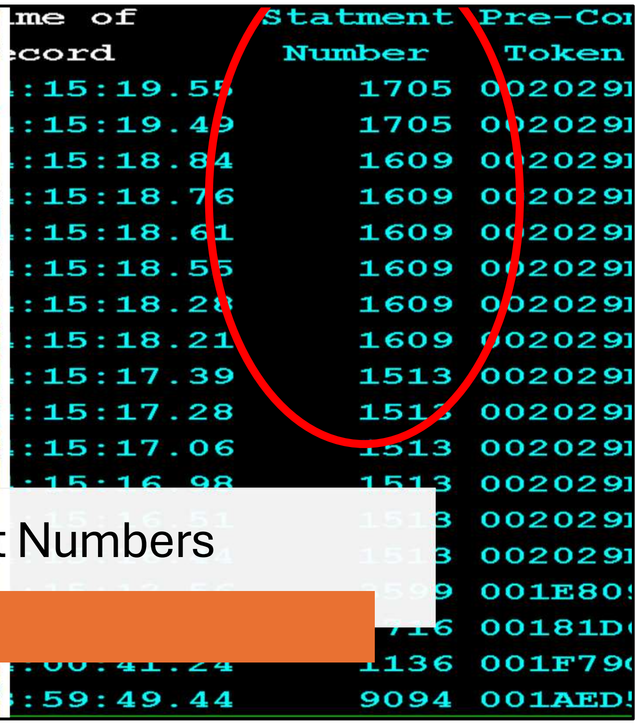


Drilled down in the monitor to identify the packages and statements that were causing the RDS failures.

Here we can identify RDS Limit failures with RID Fail equal to M and RID used equal to NO.
This particular shop had a day's worth of data available to the monitor, so I had to scroll through the RID list data looking for RDS failures each day. There were pages and pages of data.

After identifying a collection and package to review you have to PF11 over to the right to get the statement number.



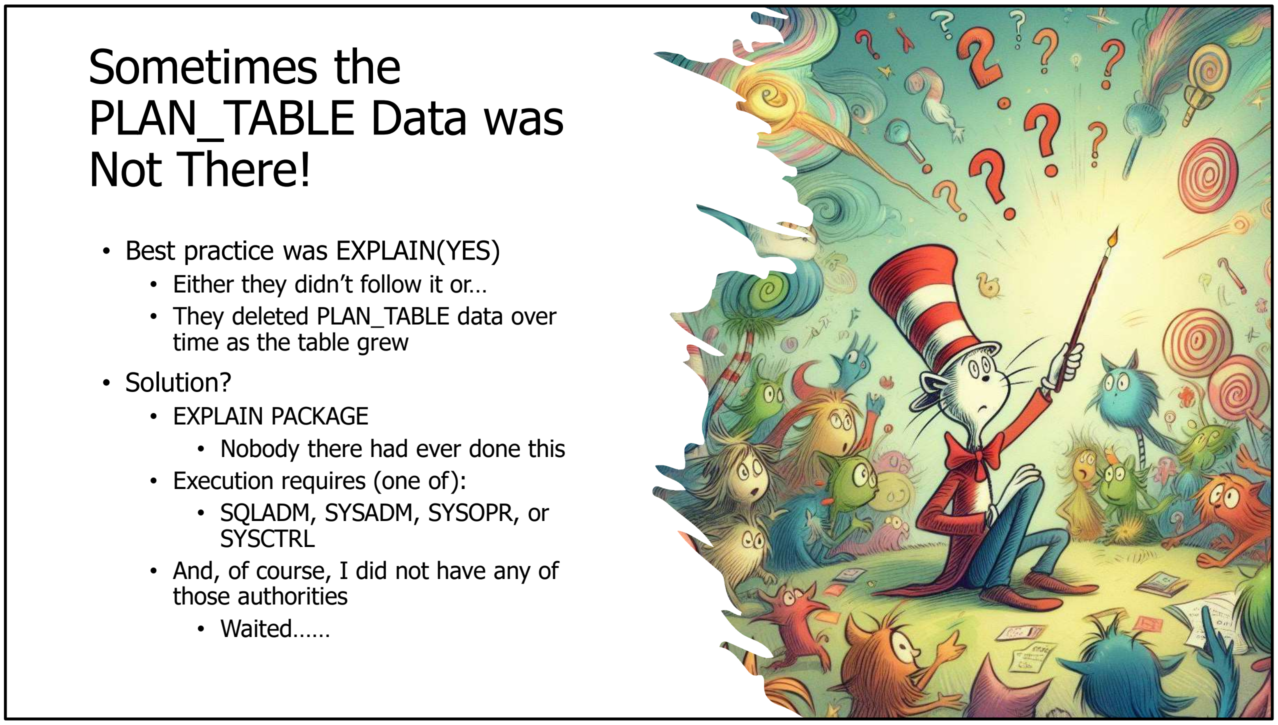


```
me  of              Statment  Pre-Co
cord                 Number    Token
:15:19.55              1705  0020291
:15:19.49              1705  0020291
:15:18.84              1609  0020291
:15:18.76              1609  0020291
:15:18.61              1609  0020291
:15:18.55              1609  0020291
:15:18.28              1609  0020291
:15:18.21              1609  0020291
:15:17.39              1513  0020291
:15:17.28              1513  0020291
:15:17.06              1513  0020291
:15:16.98              1513  0020291
:15:16.51              1513  0020291
                       1513  0020291
                        599  001E80
                        716  00181D
:00:41.24              1136  001F79
:59:49.44              9094  001AED
```

Statement Numbers

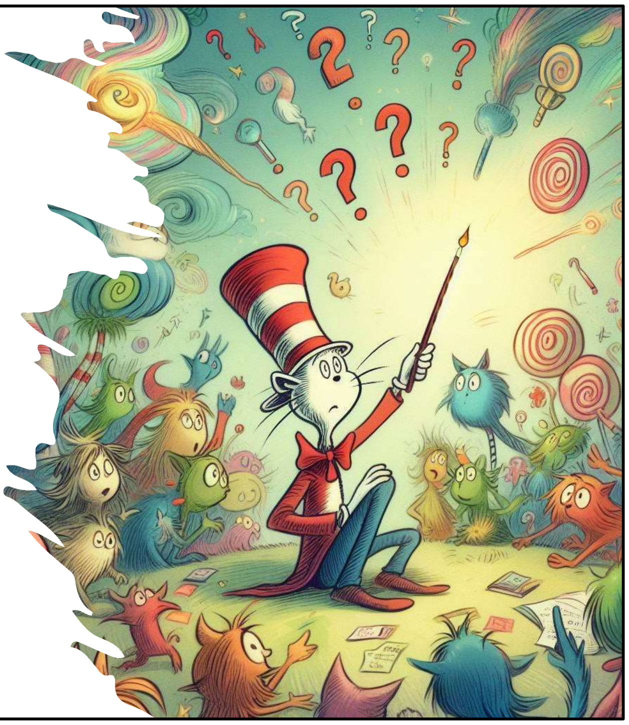
After scrolling to the right with PF11 you can see the statement numbers associated with each entry. Armed with collection, package, and statement number you can go to the Db2 Catalog and find the offending statement.

The next step was to check the PLAN_TABLE to see the access path and verify that the statement was using List Prefetch in the access path for the identified statement number.



Sometimes the PLAN_TABLE Data was Not There!

- Best practice was EXPLAIN(YES)
  - Either they didn't follow it or…
  - They deleted PLAN_TABLE data over time as the table grew
- Solution?
  - EXPLAIN PACKAGE
    - Nobody there had ever done this
  - Execution requires (one of):
    - SQLADM, SYSADM, SYSOPR, or SYSCTRL
  - And, of course, I did not have any of those authorities
    - Waited……



Even though this site had a best practice in place to BIND everything that went to production with EXPLAIN(YES), there were still times that the access paths were missing from the PLAN_TABLE!

Most Common Reason for RID Failures

- RUNSTATS!
  - Never Run
  - Inaccurate
  - Outdated

One of the most common reasons for RID pool process failures is not running RUNSTATS.

If you look back at the first story about RUNSTATS not being run, yes, this particular client had that problem. Manual stats were added (decades ago), RUNSTATS were not run to keep the manual stats in place, but the list of tables this was done for (although documented I'm told) could not be found. So, some RUNSTATS were being done but nobody wanted to run anything additional for fear of messing up 30-year-old tuning efforts.

Dealing with RID Pool Failures

- Run RUNSTATS
- Add (or change) an index
- OPTIMIZE FOR 1 ROW
- REOPT(ALWAYS)
- IDAA?

The client in this case was the same one that was afraid of running RUNSTATS, so that was off the table. At least until a lot of reviewing was done.

But we wanted to eliminate the RDS failures because there were a LOT of them and they were chewing up CPU.

What I Did

- Monitored daily for RDS failures
- Captured package and StmtNo
- Reviewed access paths
  - Not always there
- Recommended fixes
  - OPTIMIZE FOR 1 ROW
    - Not always possible
  - New index
  - REBIND
  - IDAA QUERYACCELERATION(ELIGIBLE)

So what did we do?

Reviewed access paths (which were not always there)

Added OPTIMIZE FOR 1 ROW to the statement to remove List Prefetch

Worked well, except for “environments” (some were failing, others not)

In some cases added a covering index to resolve and remove List Prefetch

Also, cannot add OPTIMIZE FOR 1 ROW to an UPDATE statement!

Some issues arose

Could not add OPTIMIZE FOR 1 ROW to a SELECT COUNT(*)

Singleton SELECT

Changed to a cursor and added OPTIMIZE FOR 1 ROW

Could not add OPTIMIZE FOR 1 ROW to an UPDATE statement

Added an index or REBIND in some cases

Sometimes a simple REBIND helped…

when it had been a looong time since the last REBIND and data volumes changed

A few packages were accessing data already on IDAA, but a table or 2 was missing

Reviewed and loaded the table to IDAA

Rebound with QUERYACCELERATION (ELIGIBLE)

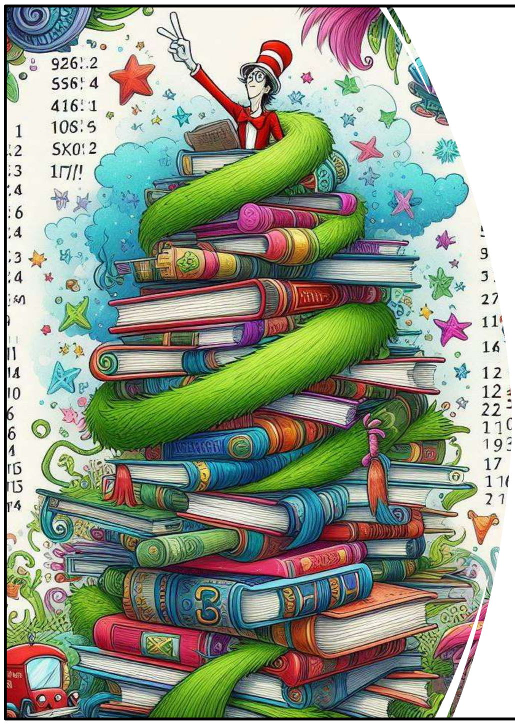
Things I’ve Seen

So let us heed
this cautionary tale,
And keep our statistics
fresh without fail.
For in the world of data,
chaos can reign,
And success relies
on keeping stats sane.

The DBA Will Review It

No Need to Worry!

The DBAs are the center of the Universe, right? They will review EVERYTHING and make sure it is OK!

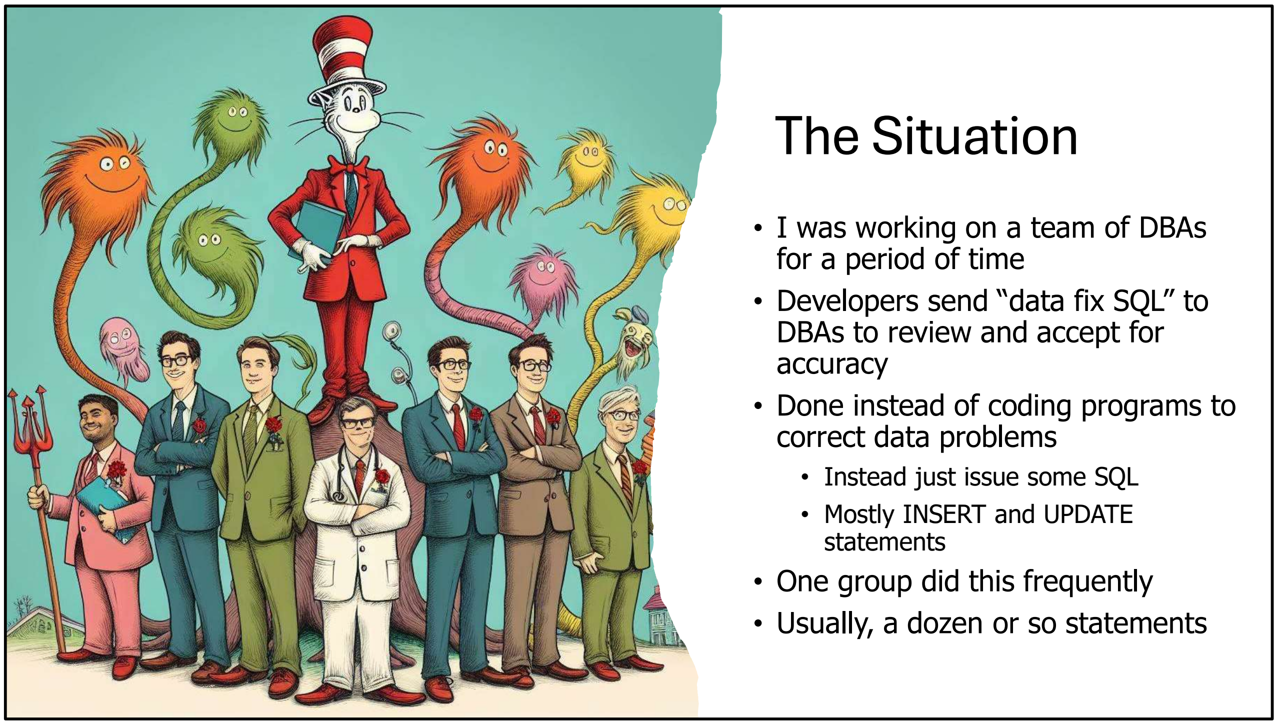


Things I've Seen

With scrutinizing eyes
and a meticulous hand,
The DBA reviews all,
a task ever grand.
All queries and updates,
each column and row,
No badness escapes,
this we all know!

But can the DBA review EVERYTHING all the time?

Is that the right thing to make the DBA team do?



The Situation

- I was working on a team of DBAs for a period of time
- Developers send "data fix SQL" to DBAs to review and accept for accuracy
- Done instead of coding programs to correct data problems
  - Instead just issue some SQL
  - Mostly INSERT and UPDATE statements
- One group did this frequently
- Usually, a dozen or so statements

Developers created and then sent "data fix SQL" to the DBA team to review and accept for accuracy.

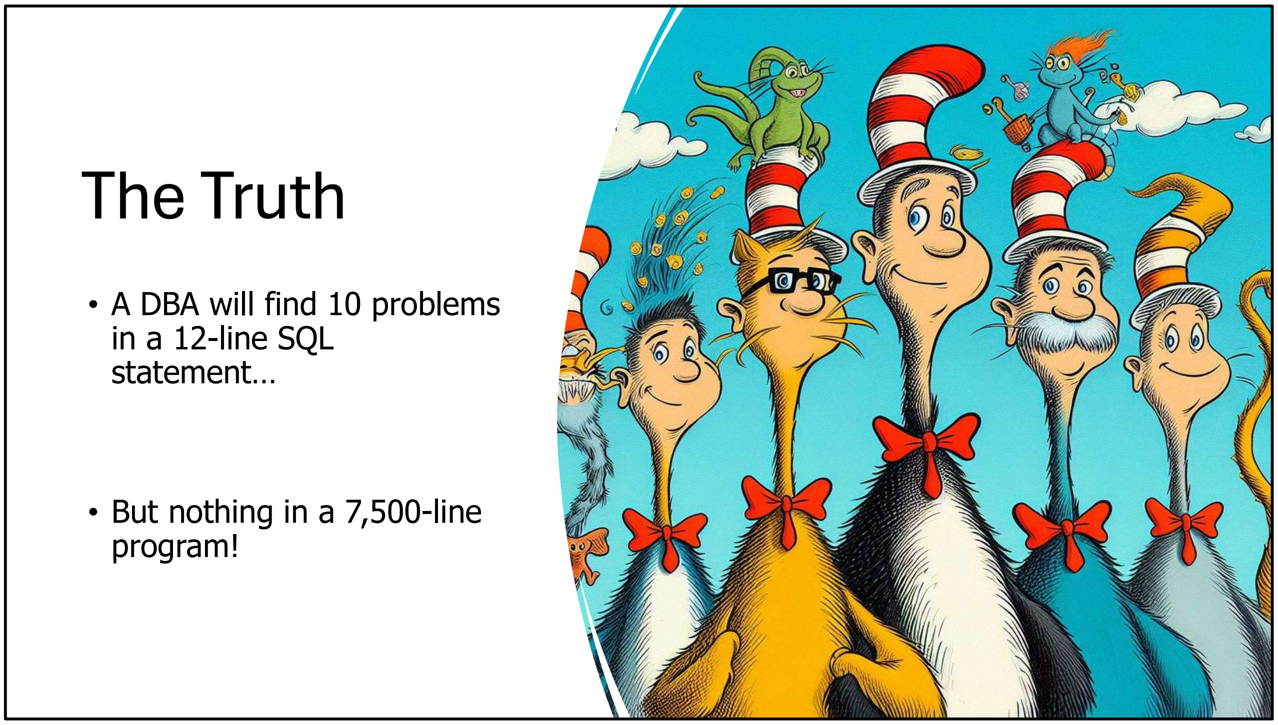
This was being done instead of coding programs to correct data problems.

Works fine for a few statements but not for pages and pages of SQL!

Then…

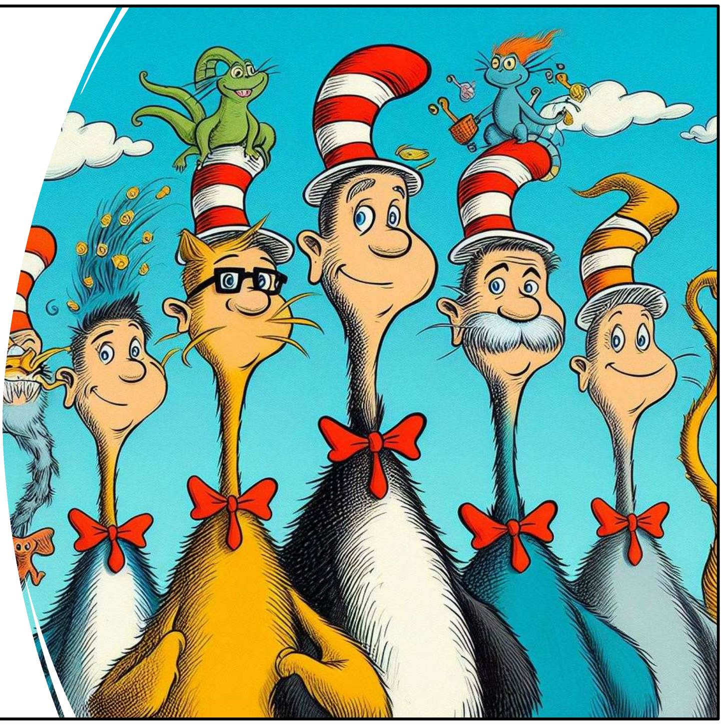
- …I got a request with hundreds of statements!

I am not a human compiler, nor is any DBA!



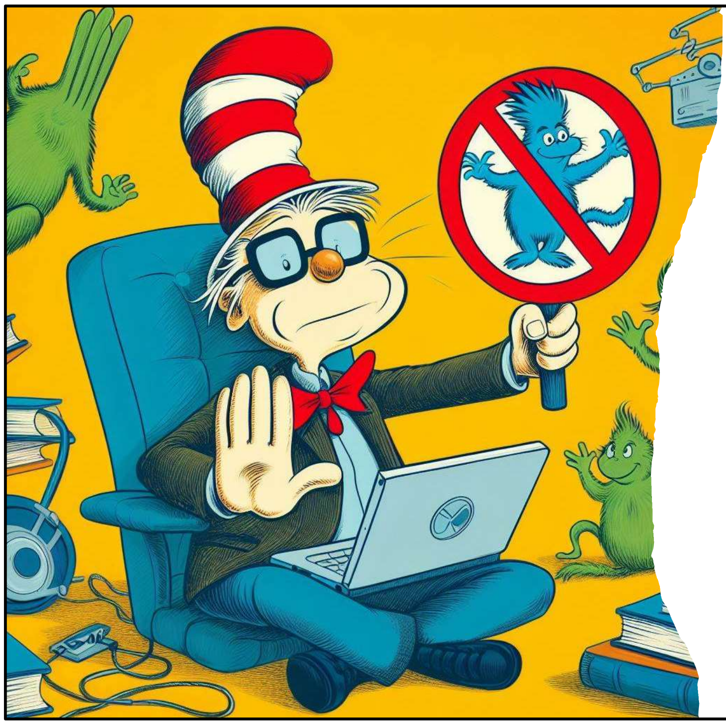
The Truth

- A DBA will find 10 problems in a 12-line SQL statement…
- But nothing in a 7,500-line program!



A DBA (or any reviewer) will find 10 problems in a 12-line program…

But nothing in a 7,500-line program!



What I Did

- I refused to review it.
  - It is an application issue.
- Instead, I took an image COPY of the tables that were being modified before leaving for the day.
  - I knew I'd have something to fall back to if the modifications were somehow wrong.

But I did NOT want to take responsibility for reviewing hundreds of SQL statements for accuracy and if any failed then it would be the DBA's fault!

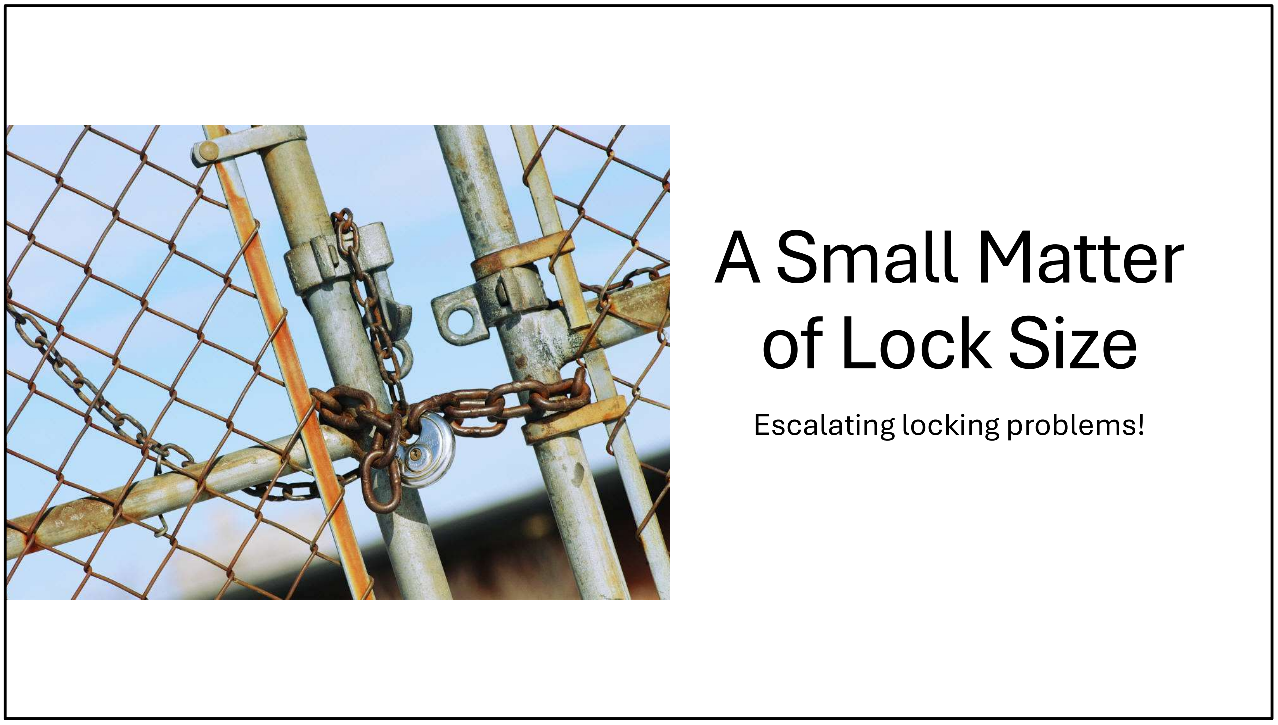
Also

- I recommended to the DBA team lead that reviewing SQL this way is not a good plan moving forward
- Why?
  - If the DBA reviews it and misses something **it becomes the DBA's fault**!
  - It is better to write "fix" programs to implement changes because it can be **easier to track and back out** problems
  - If data is constantly needing to be modified like this there is some sort of **root cause** that should be analyzed and remediated instead of just constantly fixing data

In the realm of data,
    where things are not right,
Lies a daunting task,
    a DBA's great fight.
Hundreds of statements,
    awaiting review,
But why the DBA? What did he do?

Line by line, the DBA delves,
Through the pages of statements,
    where the data is shelved.
But as the pile grows taller,
    the hours they wane,
And efficiency falters,
    under the strain.

Instead let us honor the
    programmer's art,
In crafting a program
    to play its true part.
To change so much data
    correctly it seems,
A program solves
    all of the DBA's dreams!



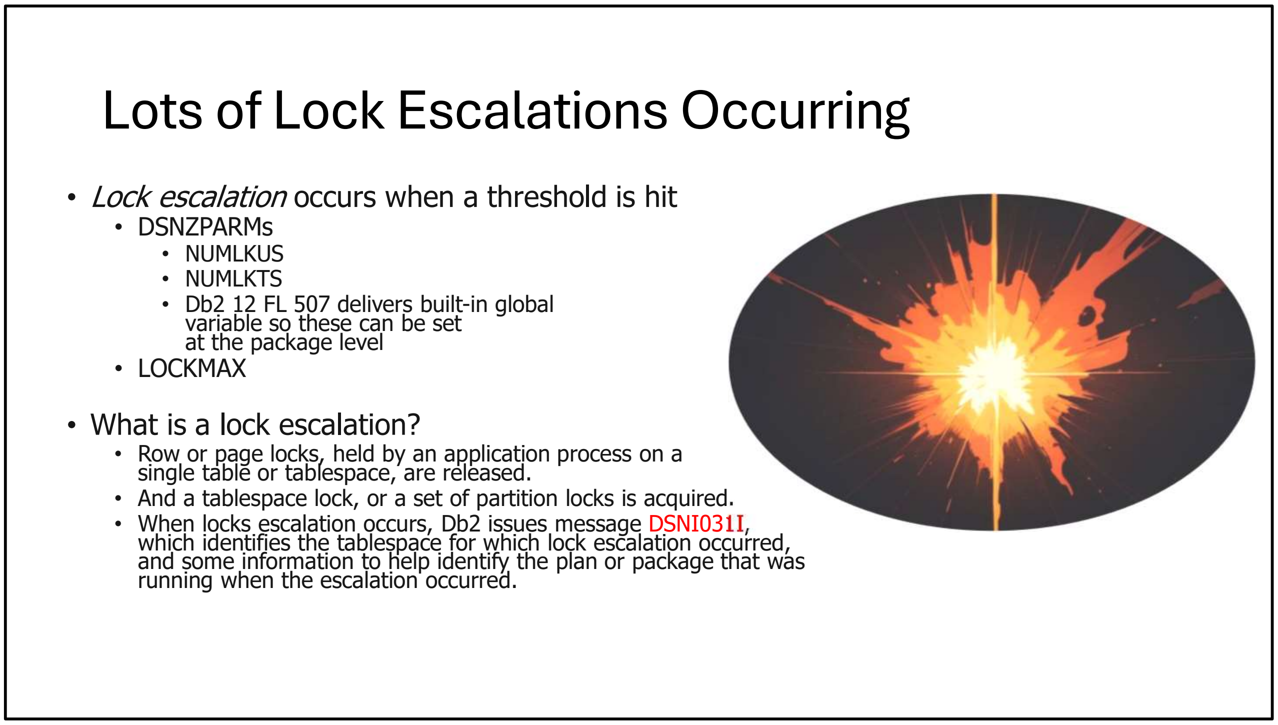
A Small Matter of Lock Size

Escalating locking problems!

Another situation I have seen at some sites is a ton of lock escalations.

Lock Escalation!

In Db2 land,
so big and so wide,
Lived apps that locked,
oh, how they'd collide!
When queries ran,
all eager and quick,
Locks would escalate,
creating quite the shtick.



Lots of Lock Escalations Occurring

- *Lock escalation* occurs when a threshold is hit
  - DSNZPARMs
    - NUMLKUS
    - NUMLKTS
    - Db2 12 FL 507 delivers built-in global variable so these can be set at the package level
  - LOCKMAX
- What is a lock escalation?
  - Row or page locks, held by an application process on a single table or tablespace, are released.
  - And a tablespace lock, or a set of partition locks is acquired.
  - When locks escalation occurs, Db2 issues message DSNI031I, which identifies the tablespace for which lock escalation occurred, and some information to help identify the plan or package that was running when the escalation occurred.

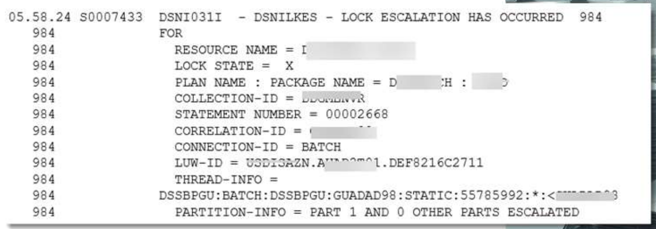
What is a lock escalation?

When a threshold is reached (either a ZPARM or a tablespace parameter), all row or page locks held by an application process on a
single table or tablespace, are released. And a tablespace lock, or a set of partition locks is acquired.

When locks escalation occurs, Db2 issues message **DSNI031I**, which identifies the tablespace for which lock escalation occurred, and some information to help identify the plan or package that was running when the escalation occurred.

Finding Them

- Viewed MSTR log
- Looking for DSNI031I
  - Table space (Resource)
  - Package



```
05.58.24 S0007433  DSNI031I  - DSNILKES - LOCK ESCALATION HAS OCCURRED  984
   984             FOR
   984               RESOURCE NAME = I[redacted]
   984               LOCK STATE =  X
   984               PLAN NAME : PACKAGE NAME = D[redacted]H : [redacted]
   984               COLLECTION-ID = [redacted]R
   984               STATEMENT NUMBER = 00002668
   984               CORRELATION-ID = [redacted]
   984               CONNECTION-ID = BATCH
   984               LUW-ID = USDISAZN.A[redacted]1.DEF8216C2711
   984               THREAD-INFO =
   984             DSSBPGU:BATCH:DSSBPGU:GUADAD98:STATIC:55785992:*:<[redacted]
   984               PARTITION-INFO = PART 1 AND 0 OTHER PARTS ESCALATED
```

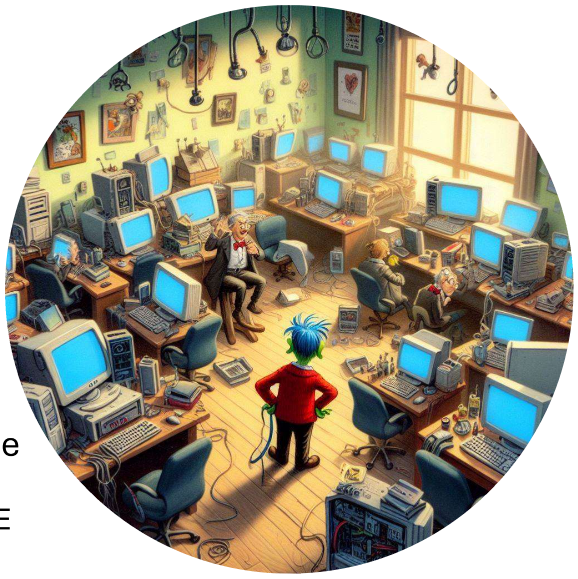
Finding lock escalations can be a tedious process. There are tools that can help but not every organization has such a tool.

You can always look at the DSNMSTR log and search for the DSNI031I messages. Or, you can always just search for “escalation” which is easier to spell!

Then you have the table space being impacted and the package causing the escalation.

Recurring Theme

- Escalating on tablespaces with LOCKSIZE ROW
- Never re-evaluated LOCKMAX
  - All were set to SYSTEM
- Moving from PAGE to ROW locking means acquiring more locks because there are multiple rows per page
  - But when they changed from ANY/PAGE to ROW no add'l analysis was done!



A recurring theme at this shop was that lock escalations were occurring on table spaces with LOCKSIZE ROW. I did a bit of digging and it seems that "long ago there were problems" so some of the table spaces were modified from PAGE locks to ROW locks.

OK, but at the same time they SHOULD have re-evaluated the number of locks for the table space by setting the LOCKMAX parameter from something other than SYSTEM. Using SYSTEM, the LOCKMAX is the same as the NUMLKTS DSNZPARM value. But there are more row locks taken than page locks, at least most of the time, because there are multiple rows per page. But no analysis was done and all the table spaces used LOCKMAX SYSTEM.

Check COMMIT Frequency

- Some programs had no COMMIT logic
  - This should NEVER be the case for any program that changes data!
- Adjustments to programs with COMMIT logic
  - COMMIT more frequently to reduce the instances of lock escalation

So the first thing we did was throw things over to the application teams and make sure that every program had COMMIT logic in place. And, of course, some did not. Those programs had to be changed to add COMMIT logic.

The programs that had COMMITs in them had to be changed to COMMIT more frequently.

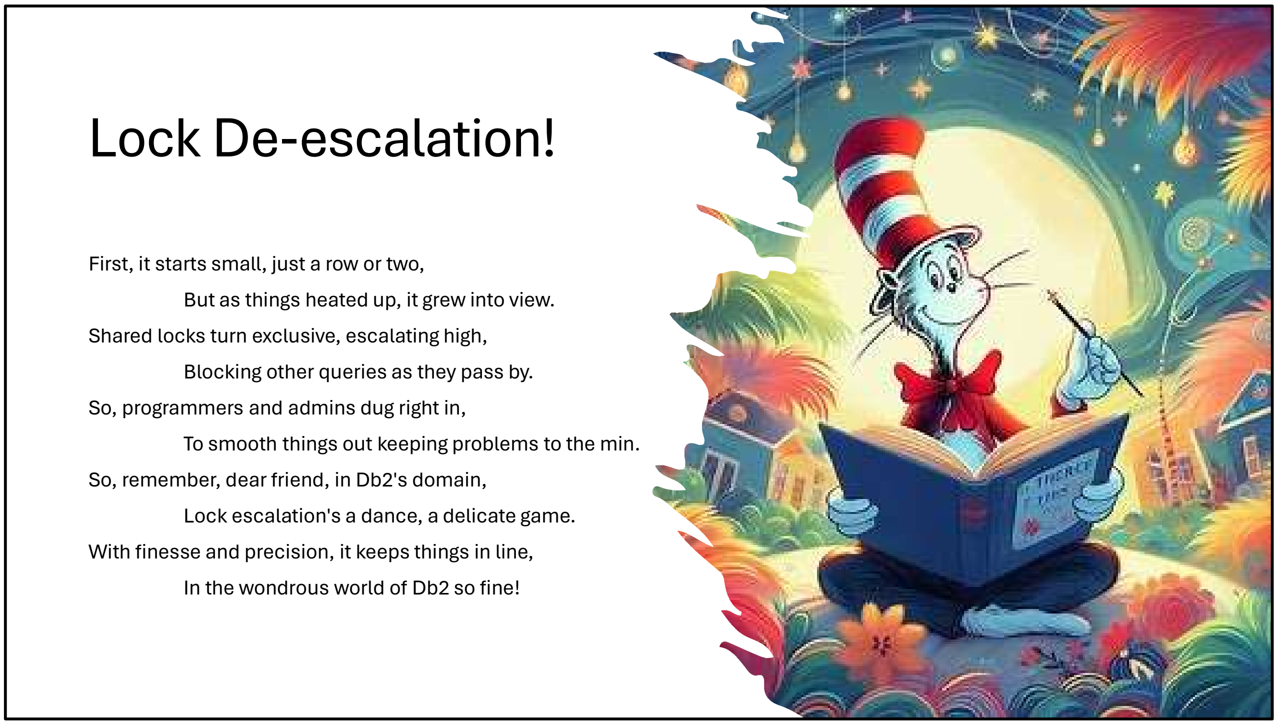
Committing, or committing more frequently, helped to alleviate many of the issues.

Consider Setting LOCKMAX

- Number of rows per page multiplied by NUMLKTS
    - Also take into account compression
- Still may have issues with NUMLKUS
    - Consider the host variables added with Db2 FL507

Perhaps a better approach would have been to modify the LOCKMAX setting. If NUMLKTS is 10,000, then setting LOCKMAX to 10,000 x (rows per page) would be a good approach to consider. Be sure to also take compression into account, which can increase the number of rows/page.

Of course, there still may be NUMLKUS issues (because a user is now taking a lot more locks). The Db2 12 FL507 host variables can be used for those packages that still present lock escalation issues.



Lock De-escalation!

First, it starts small, just a row or two,
    But as things heated up, it grew into view.
Shared locks turn exclusive, escalating high,
    Blocking other queries as they pass by.
So, programmers and admins dug right in,
    To smooth things out keeping problems to the min.
So, remember, dear friend, in Db2's domain,
    Lock escalation's a dance, a delicate game.
With finesse and precision, it keeps things in line,
    In the wondrous world of Db2 so fine!

It can take time, but with a team effort lock escalations can be tackled!

Not Db2 at all

Big problems, it must be Db2!

Things I’ve Seen

In the realm of the mainframe,
a mystery did brew,

A problem perplexed,
with no clue what to do.

Db2, they suspected,
the culprit for sure,

But beneath the surface,
dwelled a different cure!

The Situation

- Hired to help with a "significant performance problem"
- System was an entangled mass of parts
  - Old application that ran on batch, CICS, VSAM and flat files
  - Many program from old app converted to Db2 and CICS, but not all
  - Other portions rewritten in .NET using Oracle

It can take time and effort to dig through a system best described as "A big old mess of stuff"

A Few More Details

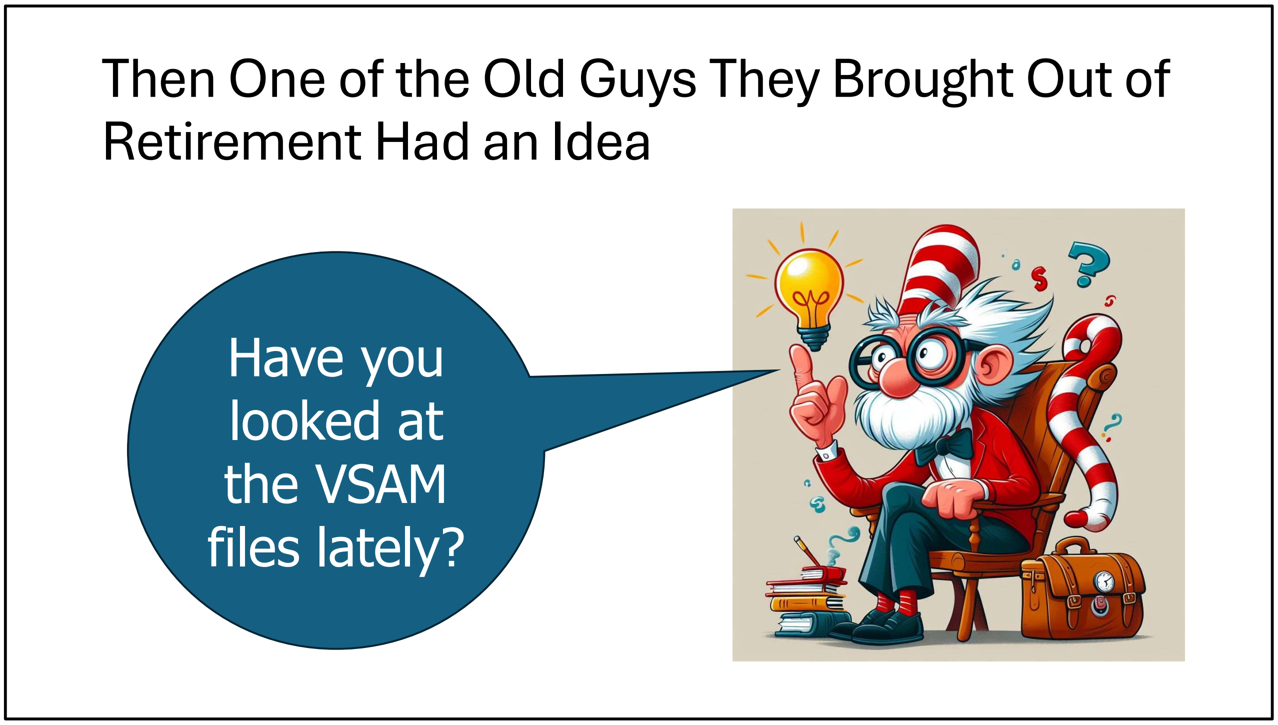
- They had hired other experts for the other parts of the application
- Wanted me to focus on their Db2 SQL
- I reviewed access paths and tuned many statements
  - Had to be implemented by programmers
- Regular tests of the entire application conducted
  - Small performance gains

This was a long-running project where the client had hired many experts for the other parts of the application. I was brought in to focus only on the Db2 SQL, looking for potential performance issues.

I reviewed access paths and tuned many statements. In some cases adding indexes and in others making code changes that had to be implemented by programmers.

Regular tests of the entire application were conducted

And each time there were small, but measurable performance gains



Then One of the Old Guys They Brought Out of Retirement Had an Idea

Have you looked at the VSAM files lately?

Then…

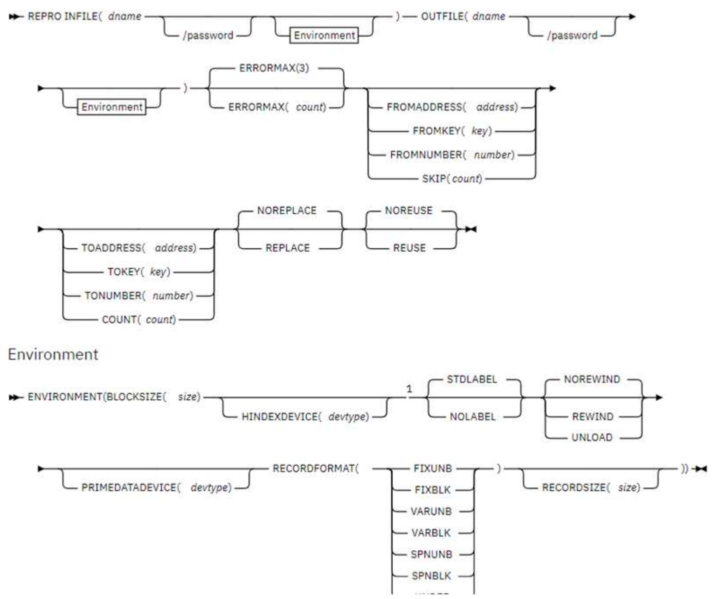
And…

- They had not been attended to in years…
  - Actually decades!
- Very disorganized.
- Somebody suggested running AMS REPRO

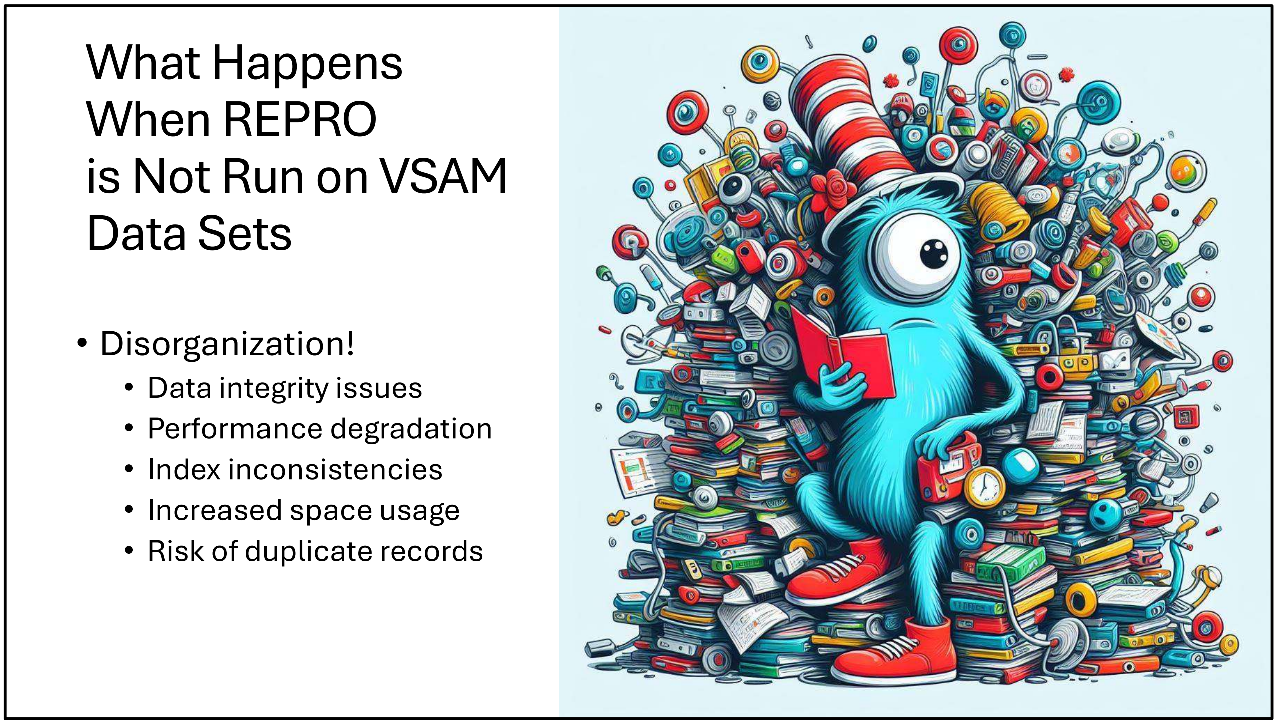
The VSAM files were still a part of the overall application but they had not been looked at in literally “years”

Turns out, the VSAM files were a mess.

The suggestion was to use REPRO to reorganize them.

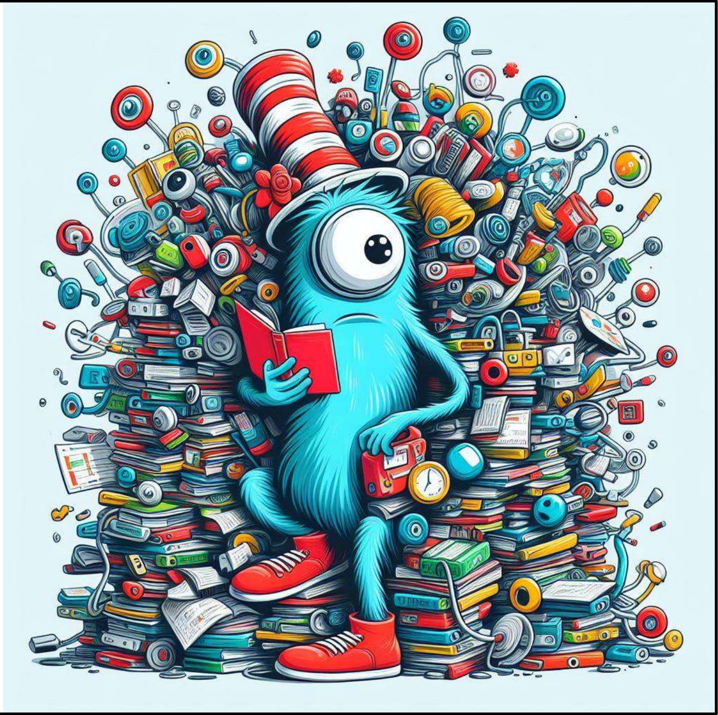


REPRO is a VSAM utility program that can perform a lot of functions on your VSAM files.



What Happens When REPRO is Not Run on VSAM Data Sets

- Disorganization!
  - Data integrity issues
  - Performance degradation
  - Index inconsistencies
  - Increased space usage
  - Risk of duplicate records



When **REPRO** is **not** run on a VSAM data set and it becomes disorganized, several consequences can occur:

1. **Data Integrity Issues**: If the **REPRO** command is not used to copy or reorganize the VSAM data set, the data within it may become fragmented or disordered. This can lead to incorrect or incomplete records being retrieved during subsequent read operations.
2. **Performance Degradation**: Disorganization can impact the performance of VSAM data sets. Retrieving records from a disorganized VSAM data set may require additional I/O operations, resulting in slower access times.
3. **Index Inconsistencies**: In VSAM, index entries are crucial for efficient data retrieval. When a VSAM data set becomes disorganized, index entries may no longer accurately reflect the actual data location. This can lead to incorrect record retrieval or even data loss.
4. **Increased Space Usage**: Disorganization can cause unused space within the data set, leading to inefficient storage utilization. As data control intervals and control areas fill up, free space may not be optimally managed.
5. **Risk of Duplicate Records**: Without proper organization, duplicate records may inadvertently be introduced into the data set. If the **REPRO** command is not used to handle duplicates, this can lead to data inconsistencies.

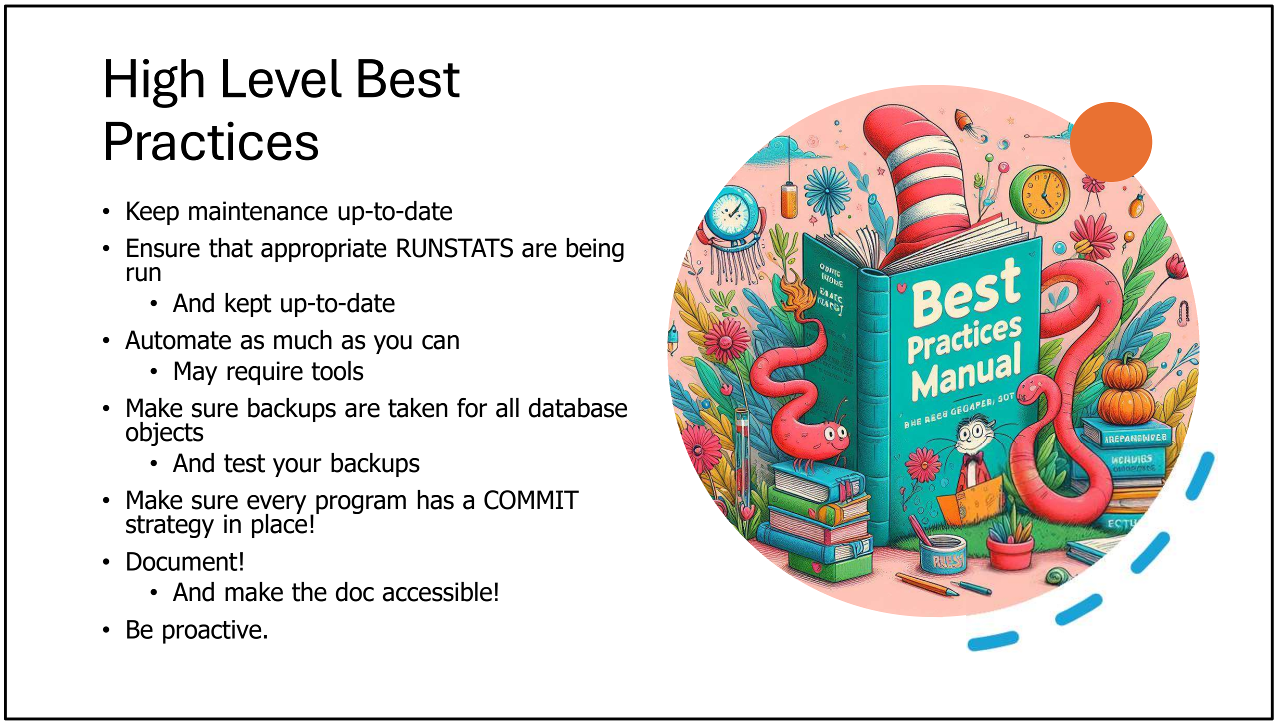
Running **REPRO** is essential for maintaining the integrity, performance, and organization of VSAM data sets. Neglecting to do so can result in data-related issues and inefficiencies.

The Lesson

Oh, the lesson learned,
in this database tale,
To look beyond the obvious,
without fail.
For sometimes the problem,
though it may seem,
Is lurking elsewhere,
in a different scheme.

Best Practices

Some Good High-Level "Things" to Do



High Level Best Practices

- Keep maintenance up-to-date
- Ensure that appropriate RUNSTATS are being run
  - And kept up-to-date
- Automate as much as you can
  - May require tools
- Make sure backups are taken for all database objects
  - And test your backups
- Make sure every program has a COMMIT strategy in place!
- Document!
  - And make the doc accessible!
- Be proactive.

Keep mtce up to date

- Talk about planned migration to another system
- Kept system folks but not DBAs
- Ran into the RBA expansion issue

Still waiting for some doc at one client that was requested last November!

- *My guess is that it no longer exists.*

Always Be Learning

The MORE that you READ,
The MORE things you will KNOW.
The MORE that you LEARN,
The more PLACES you'll GO.
–Dr. Seuss

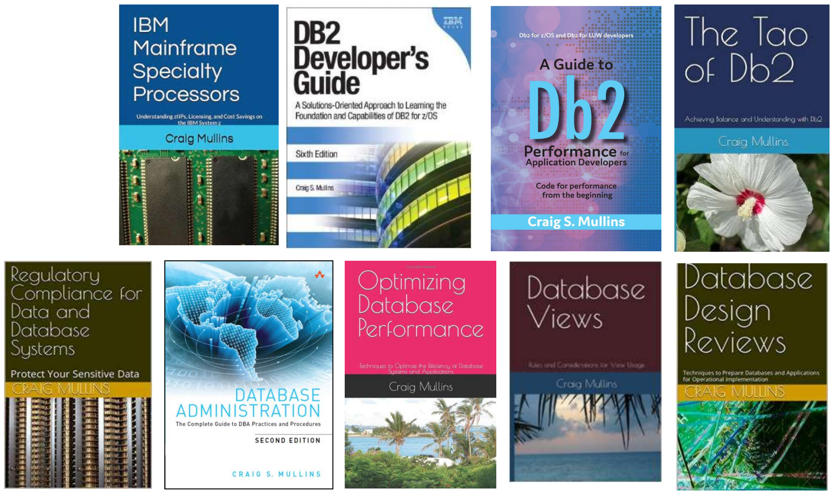
Oh, The Things I've Seen!

Craig S. Mullins
President & Principal Consultant
IBM Champion for Data & AI
IBM Gold Consultant
mullc@craigsmullins.com

www.mullinsconsulting.com

IBMCHAMPION

IBM GOLD Consultant


IBM Mainframe Specialty Processors
Craig Mullins
DB2 Developer's Guide
A Solutions-Oriented Approach to Learning the Foundation and Capabilities of DB2 for z/OS
Sixth Edition
Craig S. Mullins
A Guide to Db2 Performance for Application Developers
Code for performance from the beginning
Craig S. Mullins
The Tao of Db2
Achieving Balance and Understanding with Db2
Craig Mullins
Regulatory Compliance for Data and Database Systems
Protect Your Sensitive Data
CRAIG MULLINS
DATABASE ADMINISTRATION
The Complete Guide to DBA Practices and Procedures
SECOND EDITION
CRAIG S. MULLINS
Optimizing Database Performance
Craig Mullins
Database Views
Craig Mullins
Database Design Reviews
Techniques to Prepare Databases and Applications for Operational Implementation
CRAIG MULLINS

https://www.mullinsconsulting.com/books.htm

62

Thank you for your time and all the best as you work to avoid these (ugly) things I've seen.