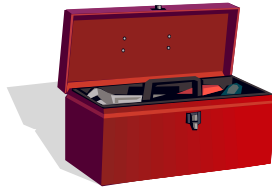


A Peek into the 24x7 VLDB DBA's Toolbox



David Simpson
Themis Inc.
dsimpson@themisinc.com

Platform: DB2 for z/OS



David Simpson is currently a Senior Technical Advisor at Themis Inc. He teaches courses on SQL, Application Programming, DB2 Administration as well as performance and tuning. He has supported transactional systems that use DB2 for z/OS databases in excess of 10 terabytes. David has worked with DB2 for 14 years as an application programmer, DBA and technical instructor. David is a certified DB2 DBA on both z/OS and LUW. David was voted Best User Speaker and Best Overall Speaker at IDUG North America 2006. He was also voted Best User Speaker at IDUG Europe 2006.

Disclaimer

The content of this presentation reflects my personal experience and is a product of the systems and applications I have worked with. Your results may vary. My results may or may not be typical.

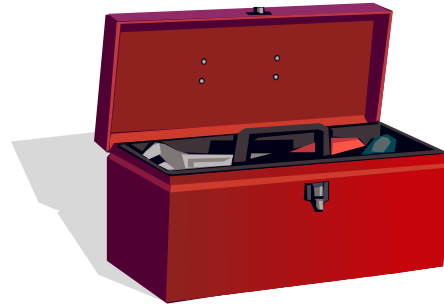
In other words....**IT DEPENDS!**



“Themis makes no representation, warranties or guarantees whatsoever in relationship to the information contained in this presentation. This presentation is provided solely to share information with the audience relative to the subject matter contained in the presentation and is not intended by the presenter or Themis to be relied upon by the audience of this presentation.”

Topics

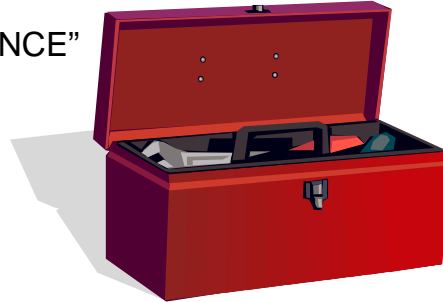
- Running utilities 24x7
- Performing table maintenance 24x7
- Access Path Analysis
- Automate DBA tasks with REXX
- Real-world examples



The topics in this presentation are all born from real live DBA experience. Running utilities and performing table maintenance are a challenge in any 24x7 environment. We have some developed some techniques that minimize the outage for these types of activities. We will also discuss tuning large-scale SQL and the automation of some common DBA tasks.

Utilities

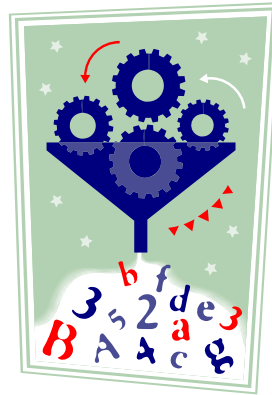
- Reorgs
 - Plan “A” – large partitioned tables with NPIs
 - Plan “B” – large partitioned tables w/o NPIs
 - Plan “C” – really, really large partitioned tables with NPIs
- Copy “SHRLEVEL REFERENCE”
- Rebuild LARGE indexes



Running utilities in a large 24x7 environment can be a challenge. Accomplishing reorgs, taking a “stable” copy of data, and rebuilding indexes will be addressed here. A small maintenance window is assumed for accomplishing these tasks.

Reorg Strategies

- Classify tables into 3 types
 - Plan “A” – large partitioned tables with NPIs
 - Plan “B” – large partitioned tables w/o NPIs
 - Plan “C” – really, really large partitioned tables with NPIs



It helps to divide mission critical 24x7 tables into three categories for purposes of reorganization.

Plan “A” addresses large partitioned tables that have non-partitioning indexes (NPIs). These tablespaces cannot be easily reorganized by partition due to the NPIs.

Plan “B” addresses large partitioned tables that do not have any NPIs. These tablespaces may be reorganized by partition.

Plan “C” addresses tables with NPIs that are too big to be reorganized in a single job.

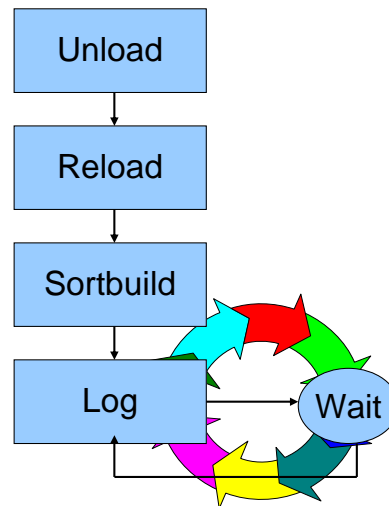
Reorg Plan "A"

```
REORG TABLESPACE CPPERD01.CINDVSE0
LOG(NO) SORTDATA SORTKEYS NOSYSREC
KEEPDICTIONARY
SORTDEVT SYSDA SORTNUM 32
COPYDDN(PLOCAL30)
SHRLEVEL CHANGE DEADLINE NONE MAXRO DEFER
MAPPINGTABLE PROD.MINDVSE0
STATISTICS TABLE(ALL) INDEX(ALL)
```



When a "Plan A" table needs to be reorganized, a job may be run to reorg the entire table and all indexes at once. The SHRLEVEL CHANGE option is used to allow application access to the table during the utility. Since these jobs run for a very long time (10-12 hours), the MAXRO DEFER option is used to control when the switch phase is attempted.

Reorg Plan “A” – Phases of Reorg



The log apply phase runs at the end of an online reorg to “catch up” the shadow copy of the data and indexes that was created during the reorg. Any insert, update or delete SQL activity that occurred during the reorg is applied to the shadow copy during this phase.

When MAXRO DEFER is specified, the reorg will go into a continuous loop during the log apply phase. The tablespace continues to be available for read/write access and any changes are continually applied to the shadow copy. The utility will not come out of the log phase until signaled to do so.

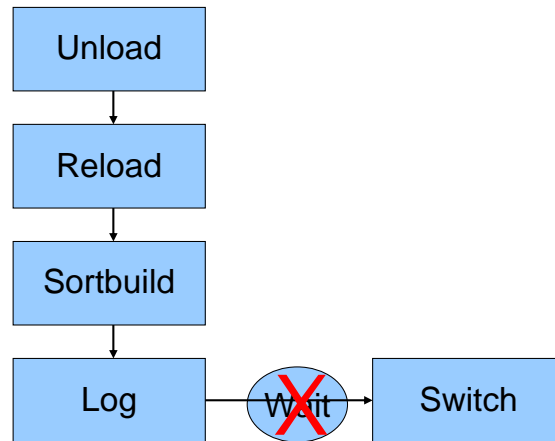
Reorg Plan “A” – Forcing a Switch

```
-ALTER UTILITY(A*) REORG MAXRO(300)
```



This is the DB2 command that may be issued to allow the reorg to complete. A specific utility id may be altered, or in this case, every utility id beginning with “A” will be altered. The value of MAXRO is changed from DEFER to a value of 300, meaning that the next cycle of the log phase estimated to complete in under 300 seconds will be the final pass of the log. The tablespace will be placed in read only status for this final pass to insure data integrity. Once the final pass of the log is complete, the utility will proceed to the switch phase.

Reorg Plan “A” – Phases or Reorg



Once the log phase is allowed to complete, the switch phase will begin. The switch phase requires that all activity be drained on the affected tablespace so that the underlying datasets may be swapped out. Since a plan “A” reorg is reorganizing all partitions, indexes and non-partitioning indexes (NPIs) of a tablespace, the switch phase can take a long time (10 to 15 minutes). By using MAXRO DEFER, the DBA can control when the switch occurs and force it into an application outage.

Reorg Plan “B” – Reorg Parts

- Pick a low volume time for your application.
- Reorg the next “n” partitions of these tables with SHRLEVEL CHANGE option.
- Attached REXX can help.



For large partitioned tablespaces that have no NPIs, a partition level reorg strategy may be used. These reorgs may generally be run with no outage provided you pick a low volume time for your application. The phases of the reorg are the same as those discussed earlier, but the read only status and the drain for switching datasets only needs to be accomplished for a single partition. The time to accomplish a switch for a single partition is negligible (a few seconds), so an application may not even notice when it happens.

Plan “B” tablespaces are reorganized over time. If you want to reorg the entire table every quarter and there are 254 partitions, have a job that reorganizes 20 partitions each week.

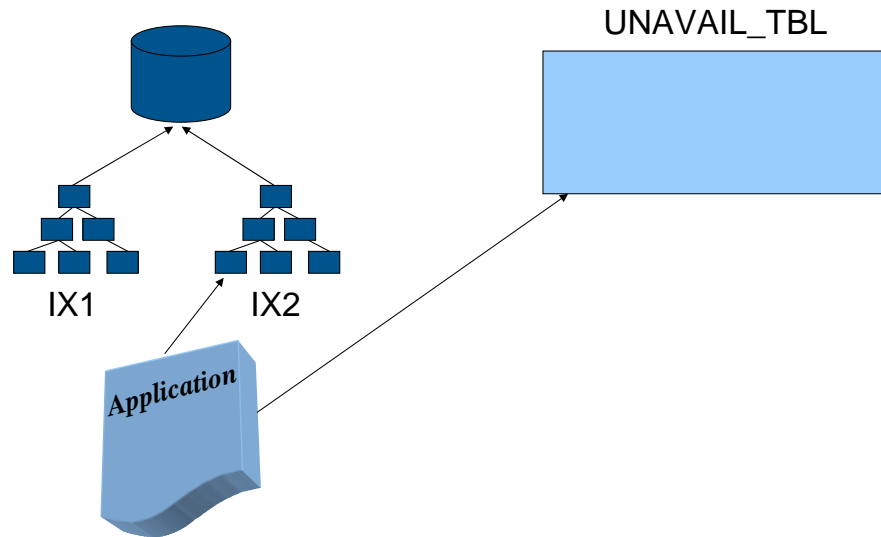
Reorg Plan “C” - Improvisation

- Implementing an “unavailable” solution
- Rotate between multiple copies of the table
- Could require use of log tools



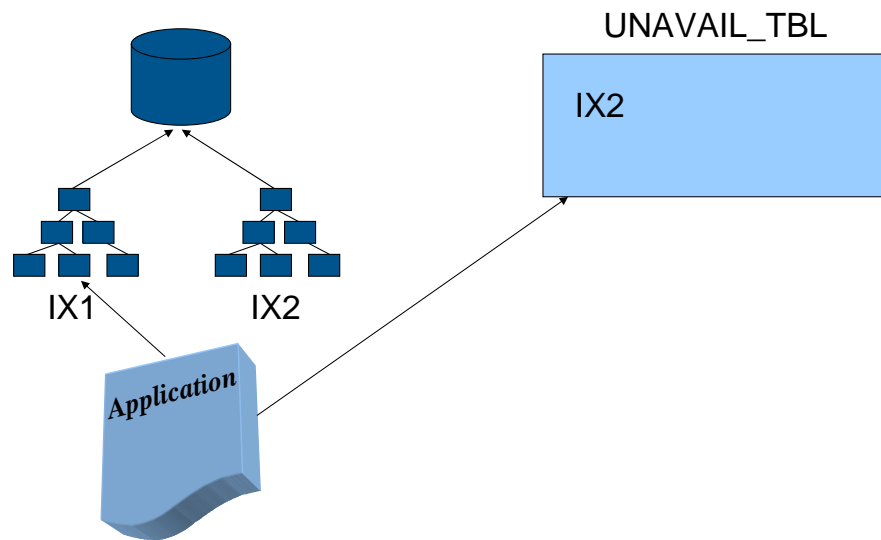
Plan “C” tablespaces are extra-large partitioned tablespaces that have NPIs. These tablespaces are too big to reorganize in a single step, so improvisation is needed to deal with them.

Reorg Plan "C" - Unavailable



One strategy is to create a table that may be used to check for the availability of a particular index. The application may be coded to periodically check to see if its primary access path is available. In this example, it is most efficient for the application to access the data via index IX2. As long as the unavailable table does not have a row indicating that IX2 is unavailable the application continues to use that access path.

Reorg Plan "C" - Unavailable



In this scenario, the application will shift to a less efficient access path by calling a different routine with a different set of packages based on the presence of a row in the unavailable table. IX2 may now be dropped for partition level reorgs on the base table. IX2 would be recreated at the end of the cycle and the row removed from the unavailable table. This scenario can only work if the application can also be put in a read only mode.

Reorg Plan “C” - Rotation

- Create a shadow table.
- Unload from primary, load to shadow
- Use a log tool to catch up
- Rename the tables
- Rebind all necessary packages
- DB2 9 has some built in functionality that will help this



Another way to deal with extremely large tables is to actually create a second copy of the table. Data is unloaded from the base table and loaded into the secondary table. If the application has been writing data to the primary table during the “reorg” log tools can be employed to gather the changes and apply them to the reorganized copy. During a short application outage, the tables are renamed and all appropriate packages must have a rebind to direct them to the newly reorganized table.

Table Maintenance – 24x7

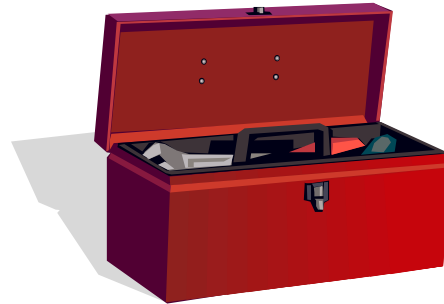
- Similar to the table rotation for reorg plan “C”.
- Unload from primary, load to shadow
- Use a log tool to catch up
- Rename the tables
- Rebind all necessary packages
- The DB2 9 functions do not address this if DDL changes are being implemented.



Similar issues can result when table maintenance is needed in a 24x7 environment and tables must be dropped and recreated. An often used techniques involves the use of a second table and renames (like my reorg plan “C”). Applying log updates may be more difficult if the tables are not identical, so a plan must be implemented to deal with this type of change.

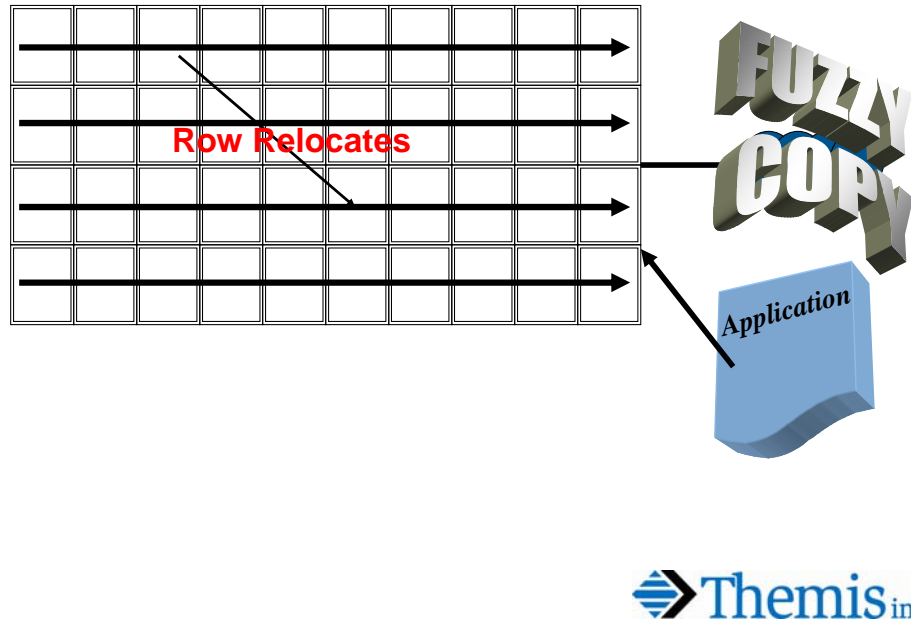
Obtaining Stable Image Copies

- Problems with SHRLEVEL CHANGE copies.
- Incremental / MERGECOPY Approach



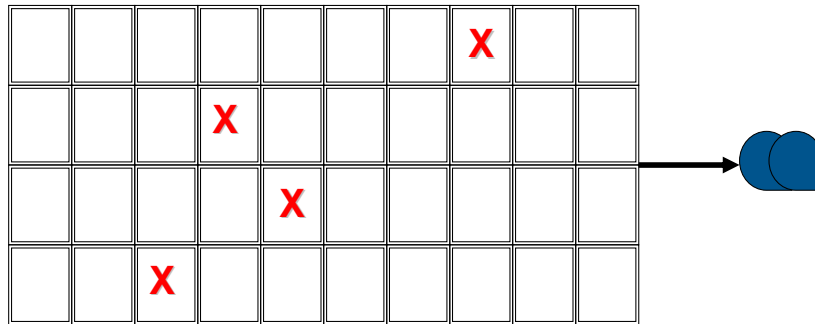
In a 24x7 environment, image copies must be taken SHRLEVEL CHANGE. It is often desirable, however, to have a SHRLEVEL REFERENCE copy available.

SHRLEVEL CHANGE Copy



SHRLEVEL CHANGE copies do not represent a picture of the data at a stable point in time. When used in a recovery situation, the DB2 log must also be available to bring the recovered data to a consistent point. In addition to having rows from different points in time, it is also possible for duplicate rows to be present in the image copy due to row relocation.

Incremental Copy



If a stable copy is needed, incremental copies may be helpful. Be sure the tablespace has the `TRACKMOD YES` property set and run an incremental copy. This copies only pages that have changed since the last copy. If few enough pages have changed, then this copy may be run as `SHRLEVEL REFERENCE`, thus representing a stable copy and minimizing the application outage.

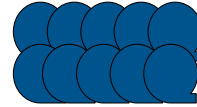
MergeCopy



Full Copy
SHRLEVEL CHANGE



Incremental Copy
SHRLEVEL REFERENCE



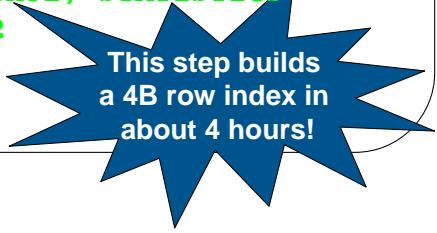
Full Copy
SHRLEVEL
REFERENCE



When the full SHRLEVEL CHANGE copy is merged with the incremental SHRLEVEL REFERENCE copy via the MERGECOPY utility, the result is a full image copy that is SHRLEVEL REFERENCE.

Rebuild LARGE Index

```
//RECOVIX EXEC PGM=DSNUTILB,REGION=800M,  
//          PARM=(DBA1,CDREJAA2),TIME=2500  
//STEPLIB DD DISP=SHR,DSN=DBA1.DB2.SDSNEXIT  
//          DD DISP=SHR,DSN=DBA1.DB2.SDSNLOAD  
//SYSPRINT DD SYSOUT=*  
//UTPRINT DD SYSOUT=*  
//SYSUDUMP DD SYSOUT=*  
//SYSIN DD *  
          REBUILD INDEX (PROD.CACCTX31) STATISTICS  
          SORTDEVT SYSDA SORTNUM 12  
//DFSPARM DD *  
          FILSZ=E600000000,HIPRMAX=0
```



This step builds
a 4B row index in
about 4 hours!



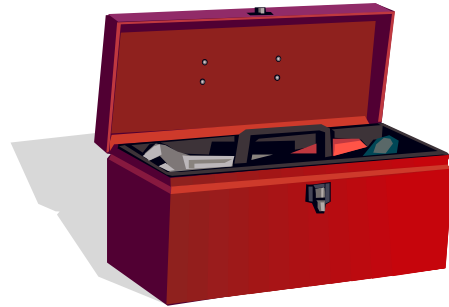
This step is used to build a 4 billion row index using DB2 Version 8 utilities with DFSORT. The SORTNUM parm indicates that DB2 will allocate 12 sort work datasets per parallel task. The DFSPARM override specifies that each subtask will sort approximately 600 million records and use of hiperspaces is disabled for this sort. Sort work files are dynamically allocated for this job.

When sorting in parallel, always request lots of memory!

Note: If you are on DB2 Version 8, you are using DFSORT as the external sort tool. If you are not yet on Version 8, you are using your shop's sort tool of choice which may or may not be DFSORT. The above example assumes DFSORT as the tool. If you are using a different one some of the parms will differ.

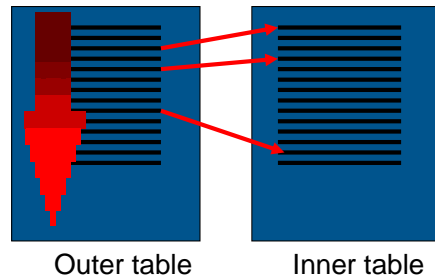
Access Path Analysis

- Join type
- Join order
- Frequency Statistics

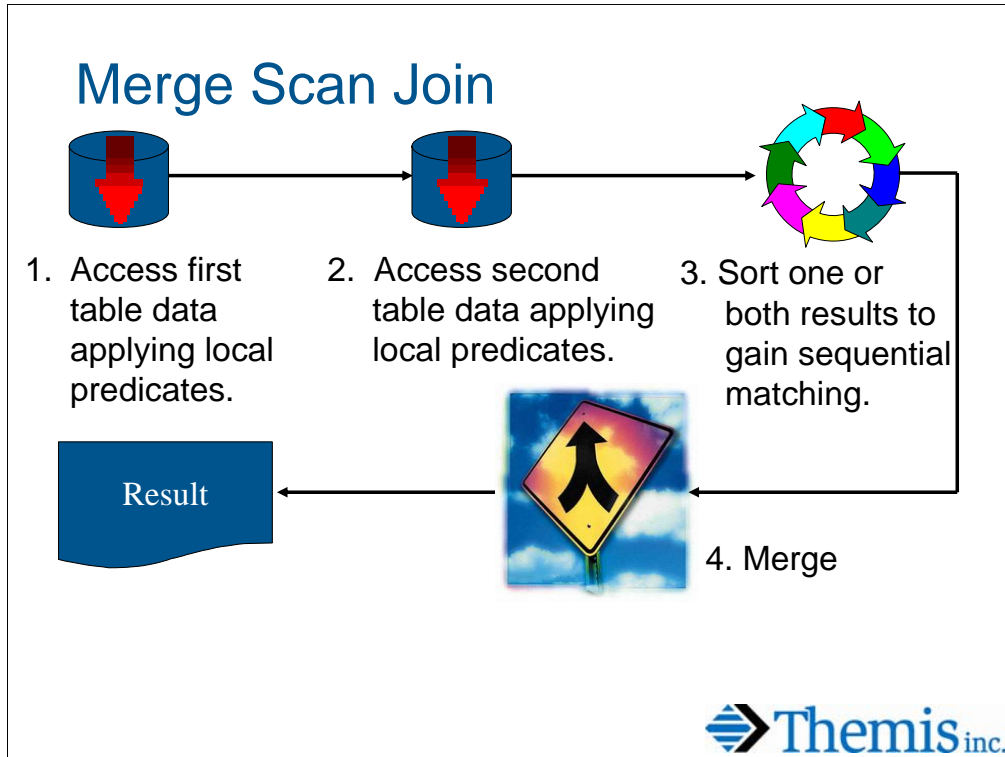


Tuning large OLTP queries presents unique challenges in a very large database. Basic access paths will be discussed as well as desirable access paths for an OLTP environment. The impact of frequency statistics and the increasing importance of gathering them is also discussed.

Nested Loop Join



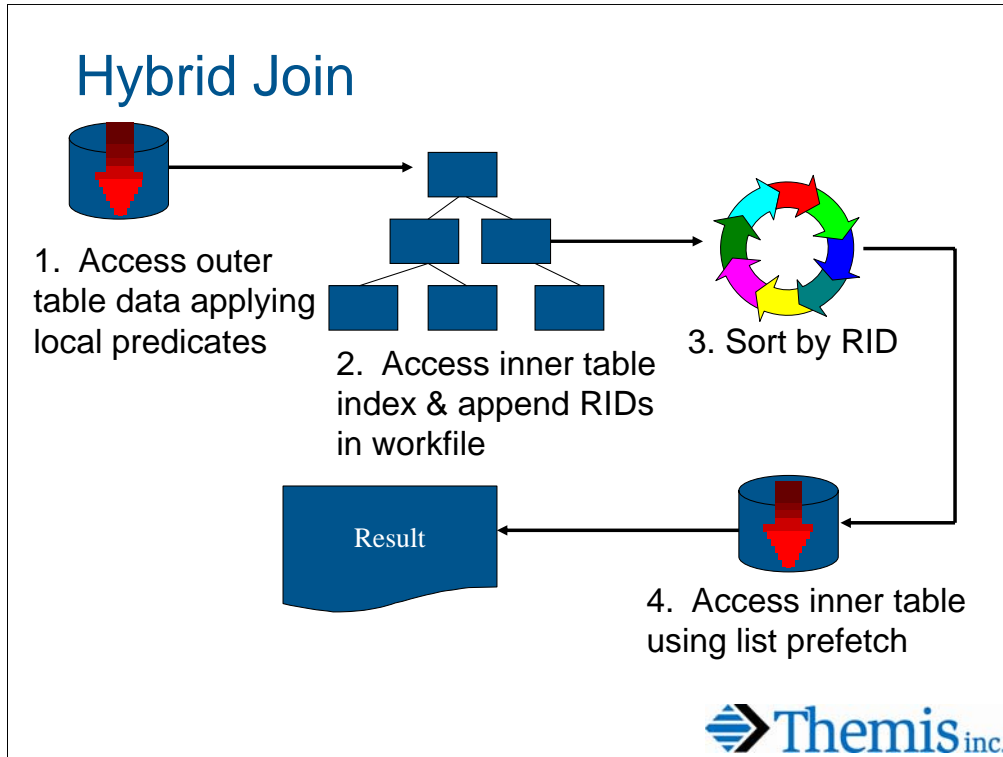
Nested loop joins are usually the optimal access path for fast running queries in an OLTP environment. An outer table is chosen and local predicates are applied. For each row selected in the outer table, the inner table is accessed (hopefully via an index) and rows are matched up. Access continues to bounce between the two tables until no more qualifying rows are found on the outer table.



Merge scan joins access each table independently while applying local predicates. One or both of the results may be sorted to obtain sequential matching. The two results are then merged into a result set.

Merge scan joins work well with large results, particularly when the data is not clustered or accessed in the same sequence. Indexes may be used on either table to more efficiently apply the local predicates.

For OLTP systems, this is usually not a desirable access path.



The hybrid join was developed to take advantage of list prefetch when joining via a non-clustered index. In theory, it has always been a good thing when used appropriately. In practice it has always generated a knee-jerk reaction to eliminate it from an access path.

For OLTP systems, this is usually not a desirable access path.

In version 8, we started seeing many more hybrid joins in access paths for larger batch queries. We started testing alternatives that forced DB2 to use either a nested loop join or a merge scan join. We found that in most cases the hybrid join was the optimal access path.

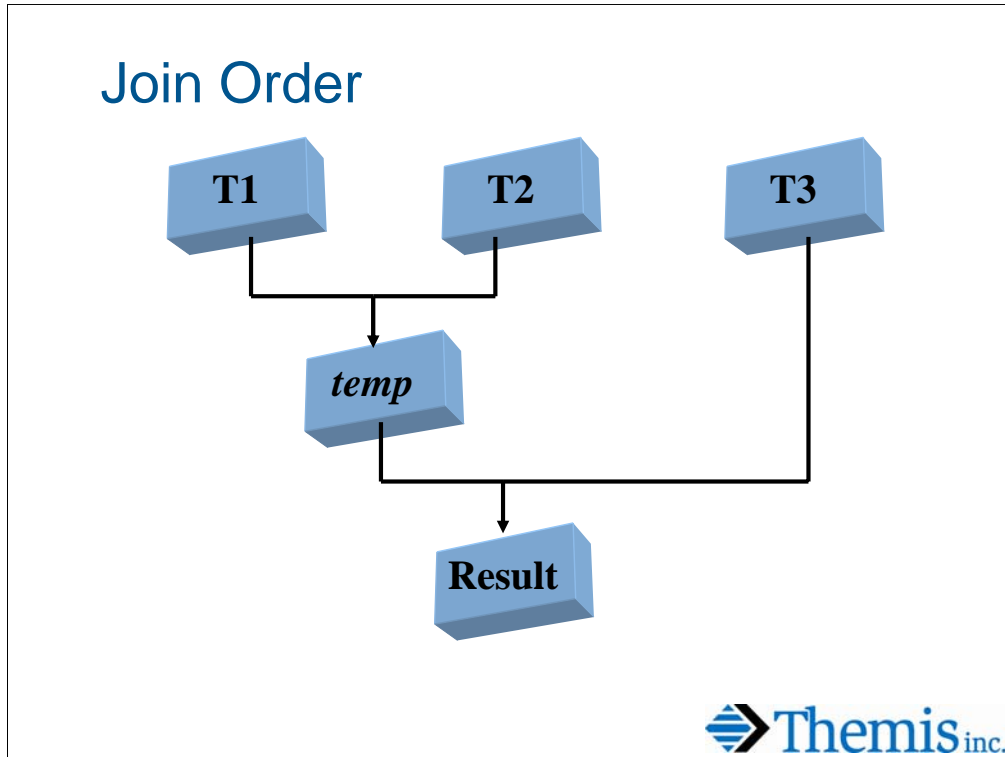
So... it still depends!

Join Order

- In general, the table with the local predicates that identifies the least amount of data is desirable as the outer table
- Critical for Nested Loop join
- Important for Hybrid join
- Not as important for Merge Scan Join
- Runstats guide the optimizer in making these determinations (more on that later).



Join order is one of the most critical factors in the performance of multi-table joins. This is particularly true if the join itself is providing lots of filtering. Queries that eliminate rows from consideration early in the join will perform better than those where rows are thrown away late in the join. As always, statistics provide the optimizer with the information necessary to determine these costs. Giving the optimizer good information is critical for insuring good decisions.



Tables are joined 2 at a time. In a three or more table join, intermediate results are formed at each step and then joined to the next table. Join order can be critical to query performance. Better performance results when the fewest number of records is passed to the next level. It is desirable to have the best filters on the first tables joined.

Of course DB2 determines the actual order of tables joined (based on available statistics), not the author of the SQL.

Frequency Statistics – V8

```
RUNSTATS TABLESPACE CQCNSD01.CINDVS10
SORTDEVT SYSDA SORTNUM 16
TABLE(QUAL01.PERSON_MRKT) COLUMN(ALL)
  COLGROUP(FRST_NME) FREQVAL COUNT 10 BOTH
  COLGROUP(LAST_NME) FREQVAL COUNT 10 BOTH
  COLGROUP(NAME_SFX) FREQVAL COUNT 10 BOTH
INDEX(ALL)
```

Works in V8
Compatibility
Mode!



Version 8 provides the ability to gather distribution statistics for non-indexed columns. Distribution statistics have existed in prior releases of DB2, but the RUNSTATS utility was not able to collect them for non-indexed columns. The DBA could always provide them via inserts to SYSCOLDIST. DSTATS is a popular sample program used to gather and insert these statistics for desired columns prior to version 8.

By default, RUNSTATS will continue to gather distribution statistics on indexed columns only. The DBA must request the collection of these statistics on other columns, as shown above.

The syntax above is requesting distribution statistics on 3 non-indexed columns for the 10 most frequent and 10 least frequent values for the column.

Frequency Statistics – Finding Them

```
SELECT TOWNER,  
       TBNAME,  
       NAME,  
       DECIMAL(FREQUENCYF*100,5,2) FREQ,  
       COLVALUE  
FROM SYSIBM.SYSCOLDIST  
WHERE TOWNER = 'QUAL01'  
      AND TBNAME = 'PERSON_MRKT'  
ORDER BY NAME, FREQ DESC  
WITH UR;
```



Here is a query to display the distribution statistics for a table.

Frequency Statistics

PERSON_MRKT	NAME_SFX	95.49	
PERSON_MRKT	NAME_SFX	2.62	JR
PERSON_MRKT	NAME_SFX	1.09	SR
PERSON_MRKT	NAME_SFX	1.09	SR
PERSON_MRKT	NAME_SFX	.39	III
PERSON_MRKT	NAME_SFX	.39	III
PERSON_MRKT	NAME_SFX	.26	II
PERSON_MRKT	NAME_SFX	.26	II
PERSON_MRKT	NAME_SFX	.06	IV
PERSON_MRKT	NAME_SFX	.06	IV



The results of the query show the values as percentages of the total number of rows on the table. Some values occur twice in this example because there were fewer than 20 possible values. This means that some values were both in the top 10 most frequent and infrequent values on the table.

This data indicates that 95% of the time, the value for NAME_SFX is blank.

Frequency Stats – Join Order

```
SELECT ...  
FROM PERSON_MRKT PM  
JOIN T2 ON PM.PARTY_ID = T2.PARTY_ID  
JOIN T3 ON PM.PARTY_ID = T3.PARTY_ID  
WHERE PM.NAME_SFX = ''  
AND T2.COL1 BETWEEN 10 AND 20;
```

Assumptions: Each table has 1M rows.
 PARTY_ID is indexed on all 3 tables
 No other indexes exist
 The T2 predicate identifies 100,000 rows



Consider the impact of distribution statistics on the above query. Since there are no indexes on columns other than the key, the optimizer will be forced to scan one of the tables and then join based on the key to the other two. It will be important for performance to identify the right table to scan first. If distribution statistics are provided on NAME_SFX, the optimizer will know that this predicate qualifies 95% of the rows on that table. It will then likely choose T2 as the outer table, since only 100,000 rows would need to be joined to the other two.

Frequency Stats – Join Order

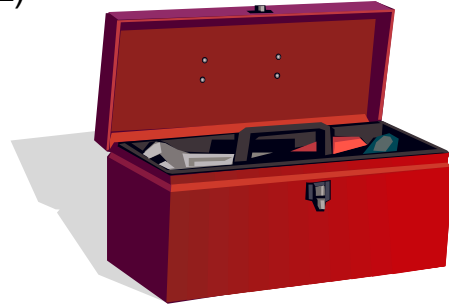
```
SELECT ...  
  FROM PERSON_MRKT PM  
  JOIN T2 ON PM.PARTY_ID = T2.PARTY_ID  
  JOIN T3 ON PM.PARTY_ID = T3.PARTY_ID  
 WHERE PM.NAME_SFX = 'IV'  
       AND T2.COL1 BETWEEN 10 AND 20;
```



In this example, less than 1% of the data on PERSON_MRKT will qualify, so the optimizer would be more likely to scan this table first.

Automate DBA Tasks

- Display Utility (DISPU)
- Display Database (DISPDB)
- Mass Update (SQLGEN)
- Thread Killer (THRDKILL)

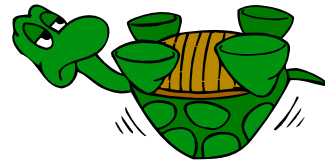


All DBAs have re-occurring tasks that can be time consuming. Third party products help fill some gaps, but there is often the need to perform a task that the products do not address (or we don't want to pay for). Here are three such tasks where we have written our own custom solution. REXX execs are used that are relatively easy to write.

Display Utility

Challenge: Interpret output from IBM's -DISPLAY UTILITY

- Critical and non-critical information mixed in same display.
- No commas in large numeric strings
- Data not aligned for easy viewing
- No commas in large numeric strings
- Data is spread over many pages
- No commas in large numeric strings
- No summary display
- **No commas in large numeric strings!!**



Interpreting output from the standard -DISPLAY UTILITY command can be a challenge. The data that comes out of this display is comprehensive, but there is not a summary screen. A major complaint among DBAs is the lack of commas in the output counts. Also, critical and non-critical information are mixed in same display.

```

DSNU100I  -DB2A DSNUGDIS - USERID = ROGUE
           MEMBER =
           UTILID = ROGUE.ROUGE1
           PROCESSING UTILITY STATEMENT 1
           UTILITY = LOAD
           PHASE = RELOAD   COUNT = 61197984
           NUMBER OF OBJECTS IN LIST = 1
           LAST OBJECT STARTED = 1
           STATUS = STOPPED
DSNU100I  -DB2A DSNUGDIS - USERID = JOEPGMR
           MEMBER =
           UTILID = JOEPGMR.JOEPGMR2
           PROCESSING UTILITY STATEMENT 1
           UTILITY = LOAD
           PHASE = RELOAD   COUNT = 0
           NUMBER OF OBJECTS IN LIST = 1
           LAST OBJECT STARTED = 1
           STATUS = STOPPED
DSNU105I  -DB2A DSNUGDIS - USERID = COOLDBA
           MEMBER =
           UTILID = COOLDBA.REORG1
           PROCESSING UTILITY STATEMENT 1
           UTILITY = REORG

```



Here is the output from the standard –DISPLAY UTILITY(*) command in an environment with 4 utilities running. 2 are active and 2 are stopped. Note that the output takes three screens to cover. Also note the lack of commas.

```
PHASE = BUILD      COUNT = 3260046018
                   NUMBER OF OBJECTS IN LIST = 1
                   LAST OBJECT STARTED = 1
DSNU111I  -DB2A DSNUGDIS - SUBPHASE = COPY COUNT = 9044775
DSNU111I  -DB2A DSNUGDIS - SUBPHASE = SORTOUT COUNT = 2056453640
DSNU111I  -DB2A DSNUGDIS - SUBPHASE = RUNSTATS COUNT = 1720416
DSNU105I  -DB2A DSNUGDIS - USERID = RFAZIO
                   MEMBER =
                   UTILID = REORG001
                   PROCESSING UTILITY STATEMENT 1
                   UTILITY = REORG
                   PHASE = LOG      COUNT = 0
                   NUMBER OF OBJECTS IN LIST = 1
                   LAST OBJECT STARTED = 1
                   STATUS = ACTIVE
DSNU347I  -DB2A DSNUGDIS -
                   DEADLINE = NONE
DSNU384I  -DB2A DSNUGDIS -
                   MAXRO = DEFER
                   LONGLOG = CONTINUE
                   DELAY = 1200 SECONDS
DSNU383I  -DB2A DSNUGDIS - CURRENT ITERATION NUMBER = 207
WRITE ACCESS ALLOWED IN THIS ITERATION = YES
```



```
ITERATION BEFORE PREVIOUS ITERATION:
  ELAPSED TIME = 01:41:12
  NUMBER OF LOG RECORDS PROCESSED = 3746459374
PREVIOUS ITERATION:
  ELAPSED TIME = 00:23:10
  NUMBER OF LOG RECORDS PROCESSED = 84763764
CURRENT ITERATION:
  ESTIMATED ELAPSED TIME = 04:10:11
  ACTUAL ELAPSED TIME SO FAR = 03:01:03
  ACTUAL NUMBER OF LOG RECORDS BEING PROCESSED = 5857634451
CURRENT ESTIMATE FOR NEXT ITERATION:
  ELAPSED TIME = 01:34:11
  NUMBER OF LOG RECORDS TO BE PROCESSED = 36465284634
DSNU111I  -DB2A DSNUGDIS - SUBPHASE = COPY COUNT = 5613883
DSN9022I  -DB2A DSNUGCCC '-DISPLAY UTILITY' NORMAL COMPLETION
```



REXX DISPU Summary Screen

```
DB2 Utility Report for DB2 Subsystem ID DBT4 Display Mode: TERSE MOD5
UTILID Mask(*)

REF
NBR  USERID  -----UTILID-----  ---TYPE ---PHASE  -----COUNT-----  STM--LIST
                                NBR  CUR  TOT  STATUS  MEMBER
-----
1    ROGUE    ROGUE.ROUGE1    LOAD  RELOAD    61,197,984    1    1    1  STOPPED
-----
2    JOEPGMR  JOEPGMR.JOEPGMR2  LOAD  RELOAD          0    1    1    1  STOPPED
-----
3*   COOLDBA  COOLDBA.REORG1   REORG  BUILD    3,260,046,018  1    1    1  ACTIVE
-----
4*   RFAZIO   REORG001         REORG  LOG          0    1    1    1  ACTIVE
-----

* - Additional utility info available for display
<Enter> to loop, <Quit> to exit, <Help> for more options
```



Here is the summary screen for our REXX called DISPU. On a single screen you can view all 4 utilities and their status. Check out those commas!

Rich Fazio is the author of this REXX. It is part of a bundle of scripts that he will provide when you email him at rfazio@transunion.com.

REXX DISPU Detail Screen

DB2 Utility Report for DB2 Subsystem ID DBT4 Display Mode: TERSE MOD5
UTILID Mask(*)

REF	USERID	UTILID	TYPE	PHASE	COUNT	STM--LIST			STATUS	MEMBER
NBR						NBR	CUR	TOT		
3	COOLDBA	COOLDBA.REORG1	REORG	BUILD	3,260,046,018	1	1	1	ACTIVE	
3	COOLDBA	COOLDBA.REORG1	SUBPHASE	COPY	9,044,775					
3	COOLDBA	COOLDBA.REORG1	SUBPHASE	SORTOUT	2,056,453,640					
3	COOLDBA	COOLDBA.REORG1	SUBPHASE	RUNSTATS	1,720,416					



A utility may be selected for greater detail. On the detail screen a line for each subtask is displayed with its counts. From either screen utilities may be terminated. A mask may be supplied on either screen to determine which utilities are shown.

Display Database

Challenge: Interpret output from IBM's
-DISPLAY DATABASE

- Lengthy multi-page display of information
- Truncation of messages beyond 65k
- No “summary” display
- Miss entries in massive lists
- Hit <Enter> one time too many times
- Does not show corresponding table/ index information



-DISPLAY DATABASE has similar issues to -DISPLAY UTILITY....however, it's almost worse.

Messages span many pages, also, it is easy for a restricted state to sneak by.

Lastly, the display buffer is only so big (64k)...once it's full, the display ends (whether you have a LIMIT(*) or not).

In V8, this gets better and worse... worse because there can be up to 4000 partitions, but there have been some display efficiencies built into the IBM display that cut down on the amount of screen space used.

DISPDB

TSO DISPDB DBP5 CPCNSD%

DB Name	Pageset	RW	UT	RO	STOP	UTRO	OTHER
CPCNSD01	CCPS5SK0	-	-	-	-	-	RW,UTRW...254
CPCNSD03	CACCTS20	212	30	-	-	-	STOP,RECP....12
CPCNSD03	CCPS5SC0	184	70	-	-	-	
CPCNSD03	CCPS5SH0	-	50	-	-	-	RO,COPY...204
CPCNSD03	CCPS5SK0	184	70	-	-	-	
CPCNSD03	CCPS6SB0	181	73	-	-	-	
CPCNSD03	CCPS6SC0	204	50	-	-	-	
CPCNSD03	CINDVSA0	196	58	-	-	-	
CPCNSD03	CINDVS50	196	58	-	-	-	
CPCNSD03	CINQYS10	204	50	-	-	-	
CPCNSD03	CINQYS20	204	50	-	-	-	
CPCNSD03	CPUBRS10	235	19	-	-	-	

Total	3048	2000	578	-	-	-	470
Percent	100	66	19	-	-	-	15

Exclusion list provided: UTRW
Tablespaces Selected: 78 Reported: 12



DISPDB Rexx focuses upon consolidating the status messages into a single line. Additional information is also gathered and displayed on a second line.

This REXX is also part of Rich Fazio's suite.

Doing SQL by partition

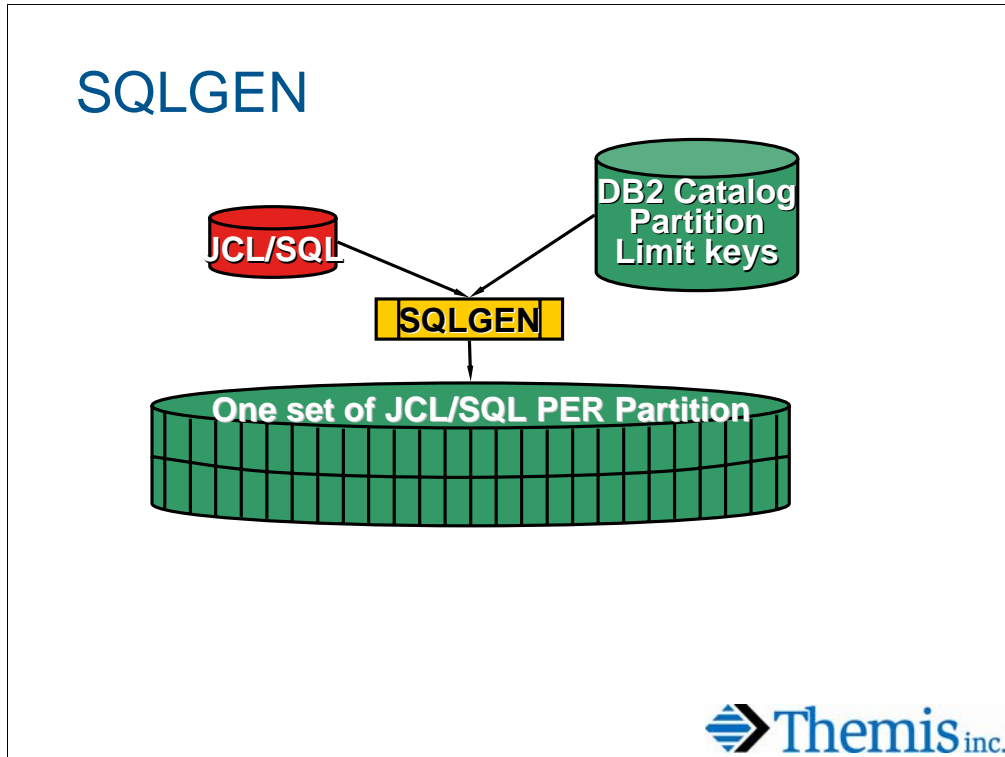
Example: Need to update lots of rows on a very large table

```
LOCK TABLE primary.table IN EXCLUSIVE MODE;  
  
UPDATE primary.table PT  
  SET whatever_flg = 'Y'  
  WHERE whatever_flg = 'N'  
    AND NOT EXISTS  
    (SELECT 1  
     FROM dependant.table DT  
     WHERE PT.PRIM_KEY = DT.PRIM_KEY)
```



Sometimes new columns get added to the database. I'll default the column to "N" and set all the "Y" values based upon the application criteria after the fact...this can take hours...but usually, it's ok.

Sometimes, the need to get these flags populated can be critical to the business. Speed is everything. A normal update (above) takes 2-3 hours. The only way to get this done quickly is with parallelism. Too bad DB2 does not have "UPDATE" parallelism.....well... not built in anyway.



SQLGEN reads in an SQL Stmt (or JCL w/SQL Stmt) and modifies a query to allow embedding of partition limit ranges.

The partition ranges for the system are retrieved from the DB2 catalog and applied to the SQL.

Multiple queries (jobs) are created for parallel execution.

SQLGEN

Input JCL/SQL	Output JCL/SQL
<pre> //U0664###JOB CARD.... /** //DSNTEP2 EXEC DSNTEP2,DB2ID='ssid' //SYSIN DD * LOCK TABLE primary.table PART @@@ IN EXCLUSIVE MODE; UPDATE primary.table PT SET whatever_flg = 'N' WHERE whatever_flg = 'Y' AND NOT EXISTS (SELECT 1 FROM dependant.table DT WHERE PT.PRIM_KEY = DT.PRIM_KEY :PRIM_KEY_PHRASE DT) ; COMMIT; </pre>	<pre> //U0664###JOB CARD.... /** //DSNTEP2 EXEC DSNTEP2,DB2ID='ssid' //SYSIN DD * LOCK TABLE primary.table PART @@@ IN EXCLUSIVE MODE; UPDATE primary.table PT SET whatever_flg = 'N' WHERE whatever_flg = 'Y' AND NOT EXISTS (SELECT 1 FROM dependant.table DT WHERE PT.PRIM_KEY = DT.PRIM_KEY AND DT.PRIM_KEY BETWEEN 1 AND 50) AND PT.PRIM_KEY BETWEEN 1 AND 50 ; COMMIT; </pre>
Input	Output

```

//U0664### .....JOB CARD....
//DSNTEP2 EXEC DSNTEP2,DB2ID='ssid'
//SYSIN DD *
LOCK TABLE primary.table PART @@@
      IN EXCLUSIVE MODE;
UPDATE primary.table PT
  SET whatever_flg = 'N'
  WHERE whatever_flg = 'Y'
  AND NOT EXISTS
  (SELECT 1
   FROM dependant.table DT
   WHERE PT.PRIM_KEY = DT.PRIM_KEY
        AND DT.PRIM_KEY BETWEEN 51 AND 100
  )
  AND PT.PRIM_KEY BETWEEN 51 AND 100
;
COMMIT;

```

...(251 more)



SQLGEN generates one job per partition with the appropriate limitkeys included in the SQL. The LOCK TABLE statement at the top of each job is desirable to avoid row or page locks from being acquired. In V8, this will automatically lock at the partition level. Prior to V8, this statement will lock the entire table by default. You may need to alter the tablespace to LOCKPART YES to avoid the jobs from colliding with each other.

Here's the inputs and outputs.

The LEFT JCL was read into the REXX. **Note the “:PRIM_KEY_PHRASE”**. This tells the processor where to insert the partition range and how. The processor defaults to generating an “AND” statement before the limit key range added.

The RIGHT JCL was generated by the REXX. Note: this is just one of 254 jobs created by this one execution. Each JOB has the partition range for a single DB2 Part.

THRDKILL – The Thread Killer

```
//STEP1 EXEC PGM=IKJEFT1B,  
// PARM='THRDKILL DBA1 SERVER DISTSERV G2S9APPL'  
//STEPLIB DD DISP=SHR,DSN=DBA1.DB2.SDSNEXIT  
// DD DISP=SHR,DSN=DBA1.DB2.SDSNLOAD  
//SYSEXEC DD DISP=SHR,DSN=CDB.BATCH.REXX  
//SYSTSPRT DD SYSOUT=*  
//SYSPRINT DD SYSOUT=*  
//SYSUDUMP DD SYSOUT=*  
//SYSTSIN DD DUMMY
```



THRDKILL looks for DB2 threads that are connected with a given plan and authid. The REXX issues –CANCEL THREAD commands for each thread located. It may optionally display the commands without actually executing them. This script comes in handy when an application that has hundreds of active threads needs to be canceled en masse.

This one is mine... email me for the source.

Summary

- **Use your tools... existing tools can be put to creative use.**
- **Build more tools.**
- **Understand your tables and queries.**
- **Explain, explain, explain!**



Credits

- Rich Fazio, TransUnion
- Dan Luksetich, YL&A
- Bob Little, Acxiom Corp.



Questions?



Reference

IBM Books

SC18-7426 DB2 UDB for OS/390 and z/OS SQL Reference V8

SC18-7413 DB2 UDB for OS/390 and z/OS Administration Guide V8

SC18-7427 DB2 UDB for OS/390 and z/OS Utility Guide and Reference V8

SG24-6079 DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More

Previous IDUG Presentations

IDUG North America 2004 – VLDB High Performance Techniques/ Experiences by Rich Fazio



A Peek into the 24x7 VLDB DBA's Toolbox

David Simpson

Themis Inc.

dsimpson@themisinc.com

