DB2 9 Utilities are focusing on improved availability. Come and learn more about new online utilities, improvements in the backup/recovery area and other useful enhancements for this tool set that support this new DB2 release. You will also learn about features that were added to DB2 V8 utilities.

Session Objectives:

•Learn more how you can run your utilities without having to specify the SORTNUM parameter for large sorts.

•See how performance was improved in DB2 9 for z/OS utilities.

•Learn about new features added in the system backup & recovery area.

•Learn more about the increased number of online utilities.

•Learn how we provide better availability with the changes in the COPY utility.

Disclaimer

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS AND/OR SOFTWARE.

Agenda

- DB2 Allocated Sort Work Data Sets
- Utility Functions added since GA
- Reviewing DB2 9 Utility Enhancements
- Utility Best Practices (inline)

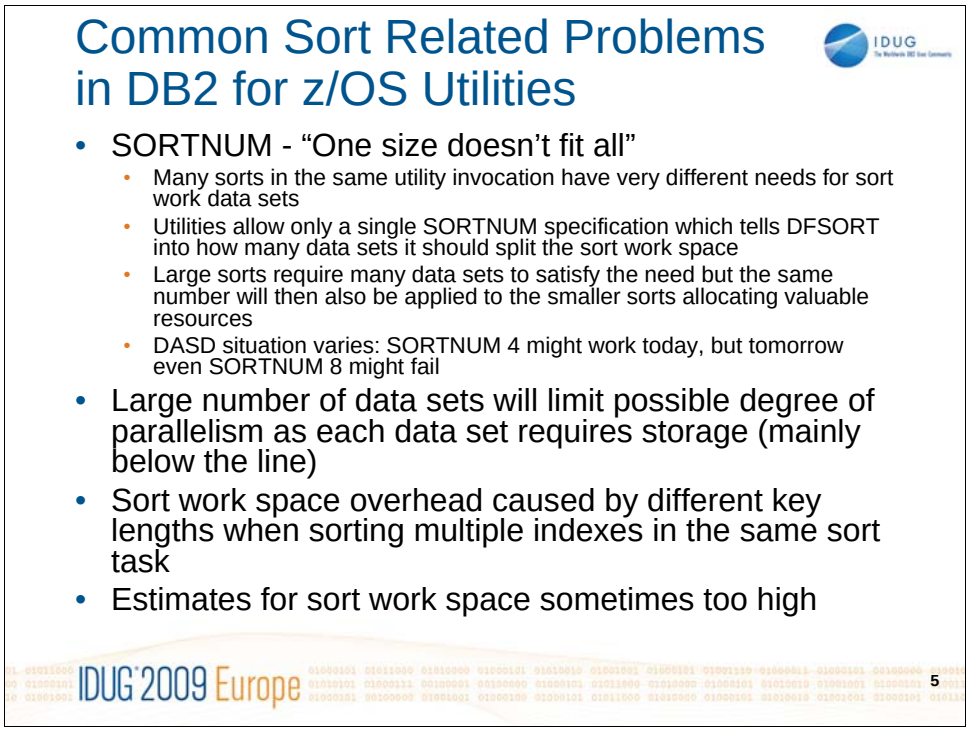
Most of this session will cover recent enhancements to DB2 utilities like DB2 allocated sort work data sets (aka SORTNUM elimination), and maintenance stream updates that provide new function.

Since many customers are now in the process of migrating to DB2 9 we will also have a short recap of some DB2 9 Utility enhancements that can be used in the new version. We will only cover the major enhancements in that part, there are many more features available.

Finally while talking about different topics, we will also have some Best Practices mentioned for the different utilities where appropriate.

DB2 Allocated Sort Work Data Sets

IDUG
The Worldwide DB2 User Community



When using dynamic allocation of sort work data sets so far the DBAs had to provide the SORTNUM parameter that specifies into how many individual data sets DFSORT should break the required sort space. This can become really difficult if different DASD situations and different object sizes have to be covered from a JCL template for example. For that reason many DBAs have chosen to set SORTNUM to a relative larger number in order to be able to allocate the work space even for large sorts in many small data sets. This practice however has its limitations – a large number of sort work data sets allocated will limit the possible degree of parallelism that can be exploited by the utility, which has a direct effect on the elapsed time.

Another problem in the parallel sorts invoked by CHECK INDEX and REBUILD INDEX is that DB2 has to combine multiple indexes into the same sort if parallelism is constrained. These utilities did not do the best job, when multiple indexes were sorted in the same task and it could happen that keys with very different key length were assigned to the same sort causing unnecessary overhead.

Finally, all sorts need some estimates on the number of records being sorted as well as the length of these records. DB2 utilities use different algorithms to estimate the number of rows in a table space, but the result is limited by the quality of statistics collected by the RUNSTATS utility. If that information is not very current, then the estimates may be incorrect – either overestimates which would cause overallocation, but sometimes they were also too low possibly causing utility failure.

"SORTNUM Elimination" Concept

- DB2 determines sort work data set size for each DFSORT invocation using information from Real-Time Statistics
- DB2 allocates sort work data sets before invoking DFSORT:
  - Ensure that disk space can be allocated
  - Know exactly how many data sets were allocated
  - Determine degree of parallelism according to storage consumption
- Invoke DFSORT with pre-allocated sort work data sets
- DB2 de-allocates data sets after sort completion

- *Developed in close cooperation with several customers from Germany, USA, Netherlands*

In 2008 IBM provided an enhancement to the DB2 utilities, which we mostly refer to as "SORTNUM elimination". This name explains what it mostly does: the DBA no longer has to specify a SORTNUM value for dynamic sort work data set allocation. Instead, DB2 will do the dynamic allocation of sort work data sets. The total size of those data sets will be determined from Real-Time Statistics (RTS) if available, which contain up to date information on the number of rows in a table space or number of keys in an index. Data sets will be allocated as large as possible in order to reduce to overall count of data sets being used which can have a positive effect on the degree of parallelism and elapsed time of the utility. Once all sort work data sets are allocated, DFSORT can be invoked to do the actual sorting. For DFSORT it looks like the sort work data sets were allocated in JCL, so it will just use them if necessary. Once the sort task has finished, DB2 will then de-allocate the data sets again.

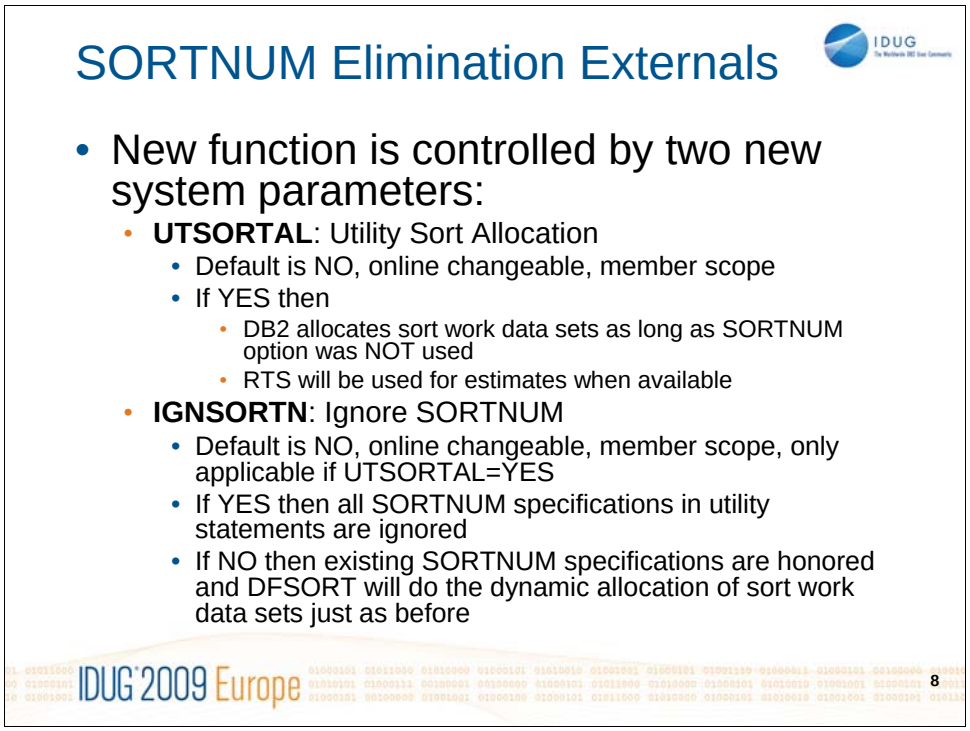
This enhancement has been developed in close cooperation with several customers from different countries and we have incorporated their ideas and suggestions before making that enhancement available.

SORTNUM Elimination Details

- Required sort work space is divided by two for first allocation
  - Maximum data set size used is Mod-54 volume size
- Large data sets are supported, consider using large sort volumes
- If allocation fails, size is gradually reduced until allocation is successful
  - You may see some DFSMS failure messages, e.g. IDG17272
- Allocation of further data sets continues with previous size until total required space has been satisfied
  - Number of data sets varies even for same job depending on current DASD situation
- Utilities with parallelism
  - Data sets with largest space requirements are allocated first
  - Use actual number of allocated data sets for calculation of possible degree of parallelism

When sort work data sets are allocated, DB2 will start allocating for two data sets for each sort, as this is the number of data sets with which DFSORT is running most efficiently. If the requested data set size can not be allocated, an iterative process will be used where the data set size is reduced in different steps until successful. So you may see DFSMS failure messages like "IGD17272I VOLUME SELECTION HAS FAILED FOR INSUFFICIENT SPACE FOR DATA SET...". This is standard behavior and no reason for concerns. We could have suppressed these error messages, but decided to better let you know what we're doing. DB2 will continue allocating data sets with the previously successful size or further reduced size until the total requirement has been satisfied. DFSORT and DB2 do support LARGE data sets, so this is a chance to further reduce the total number of data sets even for very large sorts, if large volumes are available in your sort pool.

Utilities with parallel sorts have also been enhanced to allocate the sort work data sets for the largest sorts first – as long as the sort pool has the highest chance to satisfy those requests with large available chunks of disk space. The determination of the degree of parallelism also is an iterative process now where the number of data sets allocated so far is taken into account. Since the number of data sets allocated for a specific utility run may vary for every invocation, you may now also see different degrees of parallelism for the same job – but we will always attempt to use the available resources in the most efficient way.



Since this new functionality will show a significant change in behavior of the DB2 utilities, it will be controlled by two new system parameters, that are initially turned off. Both ZPARMs are online changeable and have member scope only, so they can easily be tested out on a single member.

The first parameter UTSORTAL basically enables all the new functionality:

•DB2 controlled sort work data set allocation

•Use of RTS for sort estimates

•Improved assignment of indexes to sort tasks when parallelism is constrained in CHECK INDEX and REBUILD INDEX

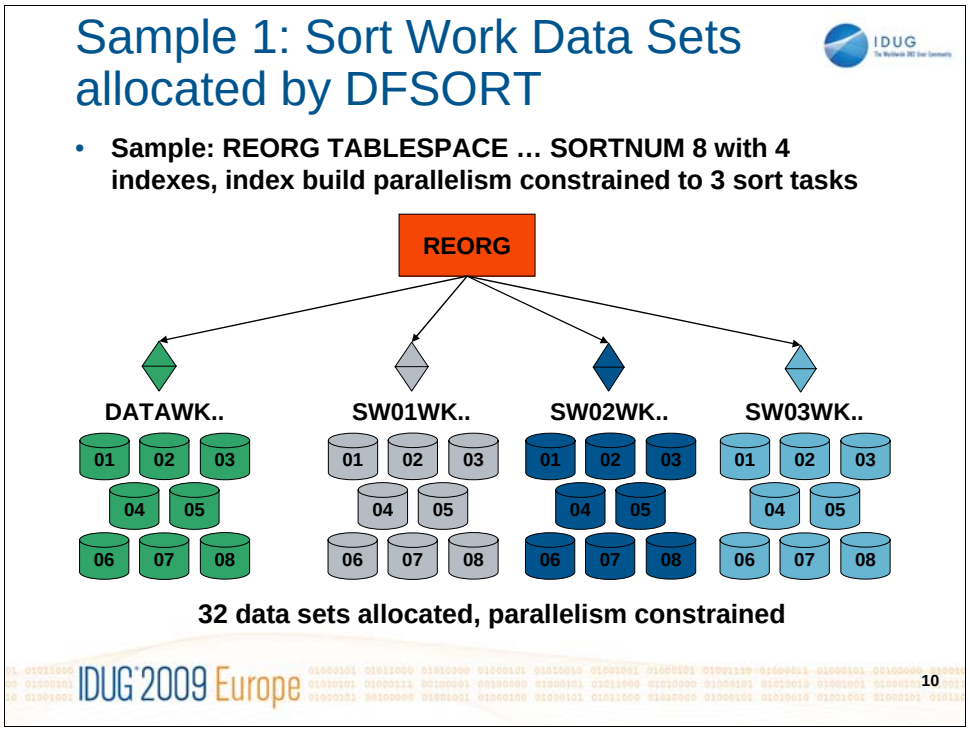
The second parameter is used to override SORTNUM parameter specifications on the utility statement. With only UTSORTAL set to YES, DB2 would honor the SORTNUM parameter on a utility statement, assuming that the DBA has spent some thoughts on what would be the right value for it. However if you want to use the new DB2 allocated sort work data sets you probably don't want to change all the hundreds or more JCLs or job templates and remove the SORTNUM parameter. That's what IGNSORTN is used for: once set to YES, any specified SORTNUM parameter will be treated as if it wasn't there.

SORTNUM Elimination Externals

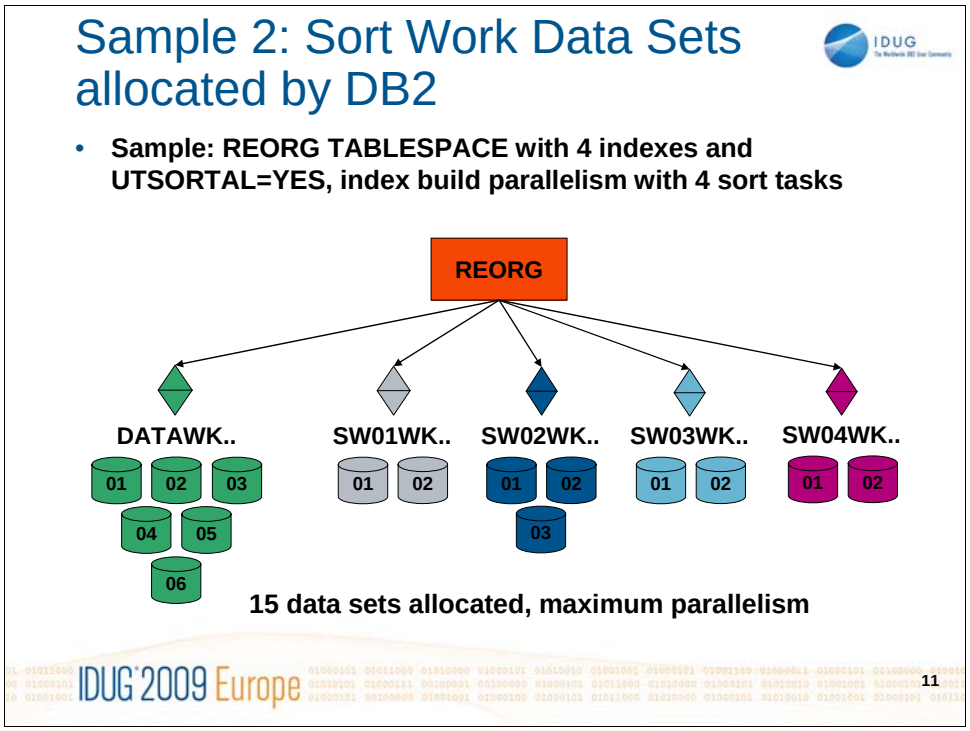
- Hard coded sort work DD cards always disable DB2 allocation of sort work data sets
- Message “DSNU3340I - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE” indicates use
- Used for all sorts in:
  - LOAD, REORG, CHECK INDEX, REBUILD INDEX, CHECK DATA, RUNSTATS

Even though UTSORTAL was set to yes, DB2 will not allocate the sort work data sets if it finds a sort work DD card already allocated for that particular sort tasks (with the different naming schemes for sort work DD cards in the different utilities, like SORTWK01, SW01WK01, DATAWK01, DA01WK01, etc.). In that case DB2 will let DFSORT use those hard coded sort work data sets instead.

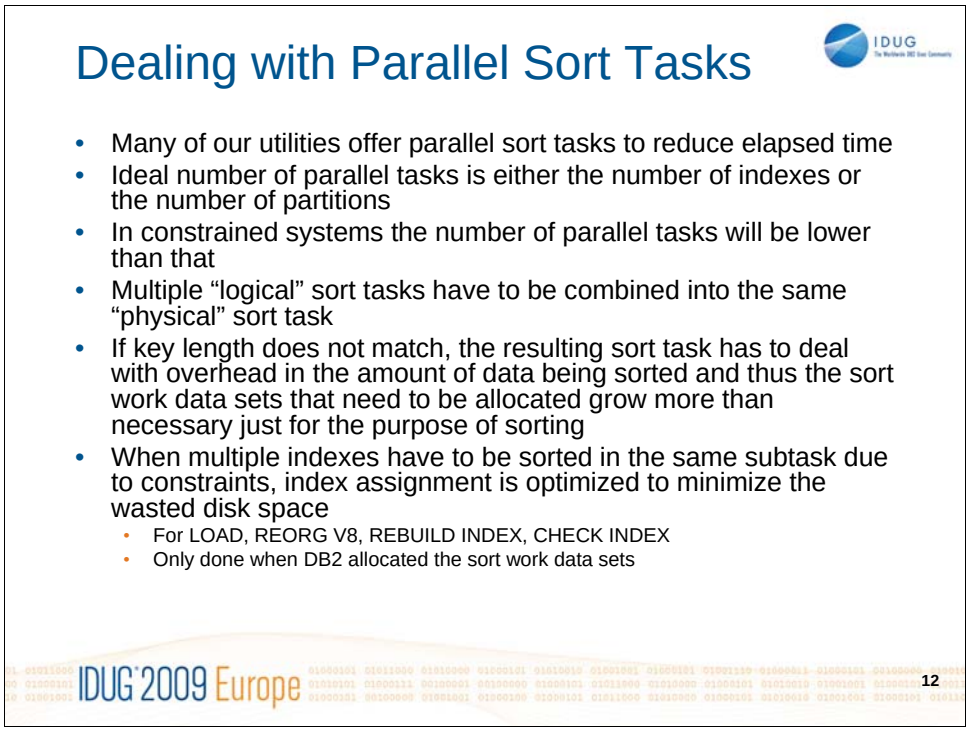
If all the requirements for DB2 controlled sort work data set allocation are fulfilled, you’ll see that by the new message “DSNU3340I - UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE” that will be issued. You will see that message from all utilities that invoke DFSORT.



So how does this compare to the existing SORTNUM parameter? Let's assume a simple case with REORG on a table space with 4 indexes defined. Due to the size of the data being sorted, a SORTNUM value of 8 was found out to be most effective for that object. However this number of data sets (8 for the data sort, plus 8 for each of the up to 4 index sorts) can not be contained in the current storage situation, so parallelism needs to be constrained to 3 parallel index sort tasks instead of 4. Of course, this is only a hypothetical example – you shouldn't be constrained by 32 sort work data sets, this is only used to illustrate the new behaviour ;-). So in this example DFSORT would allocate 32 data sets, even if the (usually smaller) index sorts don't really need that many data sets. So much for the "old" behaviour.



Now we take a look at exactly the same table space with SORTNUM elimination turned on: Let's be pessimistic and assume that DB2 also finds out that 6 data sets need to be allocated to accommodate the data sort. But we would probably still see that most of the index sorts can be allocated with our initial 2 data set approach, except for 1. That would still total to 15 data sets allocated in total with full degree of parallelism and less elapsed time due to the optimal distribution of index keys to individual sort tasks. In reality your savings should be even better than that, as experience shows that even the REORG data sorts can usually be accommodated with less than 6 data sets, even for very large objects – given that you are using large volumes in your sort pool.

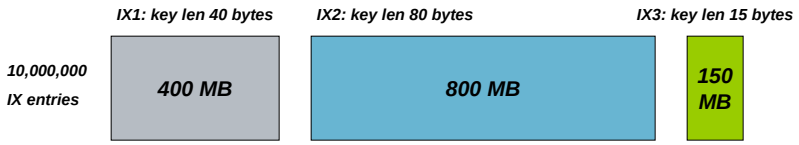


Why are we putting so much focus on parallel sort tasks you probably ask? Because this is the feature that we use to reduce elapsed time for the utility execution. The larger the data base objects grow, the more difficult it gets to get object maintenance done in an acceptable amount of time. The more we can split the work into smaller parts that can be executed in parallel, the better will your elapsed time be. The optimal degree of parallelism is to have one subtask per index or one per partition. But each subtask also has resource requirements that need to be balanced by the utility to find the best possible degree of parallelism. If the actual number of tasks needs to be lower than the optimal number, then multiple "logical" sorts need to be combined into the same "physical" sort. When combining multiple indexes into the same sort then it does hurt if indexes with a big difference in key length are assigned to the same task. In that case DFSORT needs to sort all the index keys at the maximum key length of all the indexes assigned to the same task. Since these are fixed length record sorts, all the shorter keys need to be padded to the maximum length. This has a direct hit on the required sort work space and it's obvious that it should be attempted to combine indexes with similar key lengths for best results and to prevent unnecessary overhead.

When the new code path for DB2 controlled allocation of sort work data sets is executed, the LOAD, REORG (V8), REBUILD INDEX and CHECK INDEX utilities will now benefit from the optimized assignment of indexes to sort tasks. The primary objective will be to prevent or at least reduce the necessary overhead by combining indexes of different key length. DB2 will still prevent assigning 95% of the index keys to the same task just because the key length fits, so some overhead in disk space allocation is still accepted by that algorithm. When DB2 does not allocate sort work data sets, the improved assignment is not done but the old algorithm will be used.

Sample: Multiple Sort Tasks

- Optimal distribution: 3 tasks for 3 indexes

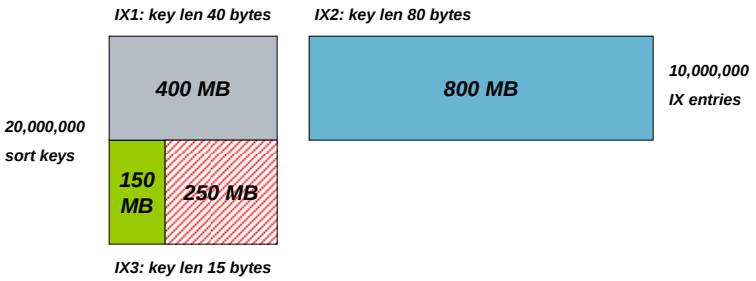


Required space: 1,350 MB

This is an example to illustrate the necessary overhead when combining multiple indexes into less sort tasks. We have chosen three indexes with significantly different key lengths of 15, 40, and 80 bytes. If all those indexes can be sorted in their own sort task, the total required sort space is 1,350 MB assuming 10 million keys per index.

Sample: Combining Multiple Sort Tasks (1)

- Constrained system: 2 tasks for 3 indexes

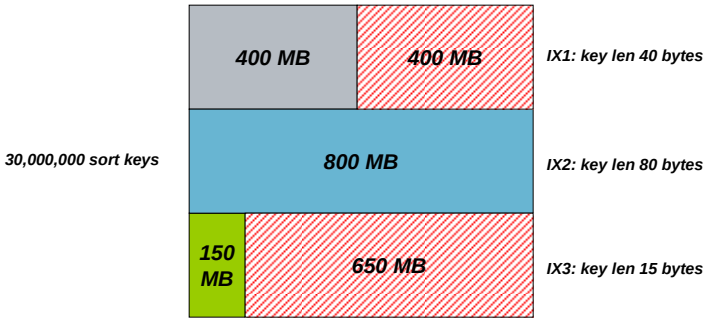


Required space: 1,600 MB (250 MB overhead)

When the degree of parallelism has to be reduced by one, then the best fit is to combine the indexes with 15 and 40 byte key length into the same sort and leave the 80 byte index in its own sort. However that means that now 10 million 15 byte keys need to be padded with 25 bytes to match the 40 byte keys from the first index. That already results in an overhead of 250 MB compared to the original 1,350 MB sort work space. If the old algorithm is still active, it might even be that DB2 combined the 15 byte and 80 byte index resulting in an overhead of 650 MB.

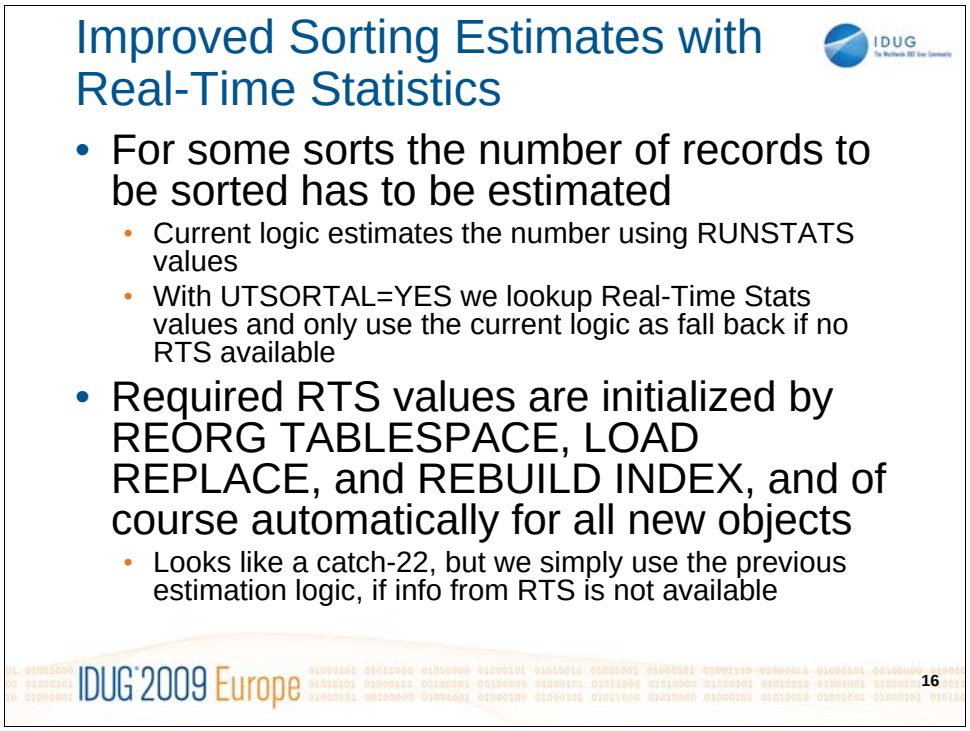
Sample: Combining Multiple Sort Tasks (2)

- Fully constrained system: 1 task for 3 indexes



Required space: 2,400 MB (1,050 MB overhead)

The worst case in this scenario happens when all indexes have to be sorted by the same sort task which means that all index keys have to be padded to a length of 80 bytes. The total sort work space requirement would then be 2,400 MB, which is an overhead of 1,050 MB over the original 1,350 MB. This is a significant overhead, which has an impact on your DASD situation but as well on your overall resource consumption, as the additional bytes need to be moved around. So it's pretty obvious that it's beneficial to keep the number of parallel tasks as high as possible which is easier achieved, if the number of sort work data sets is kept low.



During the beta test of this enhancement we soon found out that a good estimate on the number of records being sorted can be the key to successful utility execution. In some utilities the exact count is already known, e.g. from a previous unload phase, but most of the utilities need to initialize the sort processes long before the exact count is known. So these utilities rely on estimates that are based on table space statistics collected by RUNSTATS as well as allocated pages for the data sets. If RUNSTATS information has not been collected for a long time or never at all, then these estimates may not be very accurate. If the estimate is too high, then the sort work data sets will be largely over allocated but the utility would run successfully. However if the estimate is too low, then chances are that the utility will terminate with “ICE046A SORT CAPACITY EXCEEDED”.

One of the goals of this enhancement also was to reduce over allocations of sort work data sets, so we needed to get rid of overestimated record counts. Real-Time Statistics (RTS) offer a good way to get around these estimation problems. RTS has up to date information on the number of rows in a table space or table space partition or the number of keys in an index. That information can easily be retrieved and used for the sort work data set allocation. As soon as the new ZPARM UTSORTAL has been set to YES, DB2 utilities will look at RTS tables first when they have to estimate the number of records in a table space or the number of keys in an index and only fall back to the previous estimation logic, if no RTS values are available at all.

Currently only columns TOTALROWS in RTS table SYSIBM.(SYS)TABLESPACESTATS and TOTALENTRIES in SYSIBM.(SYS)INDEXSPACESTATS are used for that purpose. These columns are either initialized and maintained when new objects are created while RTS was already enabled or they will be initialized to the current values, when one of the following utilities is being run: LOAD REPLACE, REORG TABLESPACE, or REBUILD INDEX.

Since these utilities are also exploiting RTS, this first looks like a catch-22, but we do use fall-back logic, which is explained on the next slide.

Improved Sorting Estimates with Real-Time Statistics

- If RTS not available for a specific object type, we use fall back logic:
  - Use index (partition) RTS value instead of table space (partition) RTS, or vice versa
  - Use sum of one index RTS counts for each table in table space for table space stats
  - Not for XML objects as there is no 1:1 relationship
- Don't replace DB2 objects outside DB2's control to maintain RTS accuracy (e.g. DSN1COPY or DFSMS copy) or let DB2 "know":
  - Set TOTALROWS in SYSIBM.(SYS)TABLESPACESTATS or TOTALENTRIES in SYSIBM.(SYS)INDEXSPACESTATS to NULL to invalidate existing statistics or to the known number of rows/keys if replacing with significantly different data

If the utility finds out that RTS values are not available for a specific object, we'll use alternative ways to estimate the number of rows or keys:

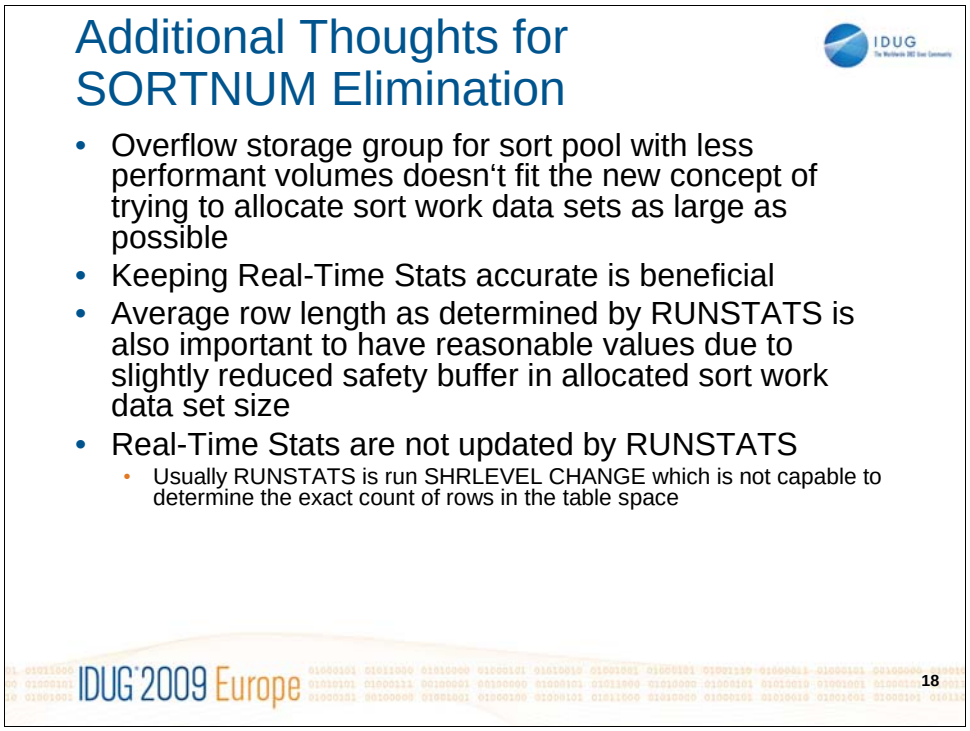
•The number of index keys in an index partition does match the number of rows in a table space partition and can be used

•Single table table spaces have the same number of rows as any index defined on that single table

•If all tables in a multi table table space have at least one index with valid RTS, we can determine the total number of rows in the table space from the sum of each of these table indexes

That assumption however is not true for XML table spaces introduced in DB2 9, as there can be more than 0 to n keys for each row in the XML table space.

Relying on RTS requires some hygiene procedures though – you should not interfere with DB2's real-time monitoring of table spaces and indexes by replacing them without letting DB2 "know" about it. Since DB2 only monitors changes to those objects (inserts and deletes) and then accumulates them to the already known values, it would result in incorrect values, if you'd replace a table space using DSN1COPY or DFSMS copy with a different table space, e.g. from production or from a prior point in time and DB2 would then continue to accumulate inserts and updates to that new object based on the previous copy's values. In that particular case, we suggest that you set the matching TOTALROWS or TOTALENTRIES value for that object to NULL to indicate that the count for this object is not known and has to be re-established on the next REORG or REBUILD INDEX. If the exact count of rows is known, you can also set that value into TOTALROWS when replacing the object. Both RTS tables are user updateable, even if they are in the catalog. This is especially important, if the table space has been replaced with a copy of significantly different size, if the number of rows is within 5% of the previous version, then you might be fine to just wait until the next REORG and just live with a small inaccuracy.



Some more notes on observations made since we have shipped this enhancement:

•Allocating sort work data sets will try to allocate data sets as large as possible and then reduce the allocation size of not successful. So if you have set up your sort pool in a way that uses some low performance large volumes in the overflow storage group, and only smaller volumes in your regular sort storage group, then these overflow volumes will be selected for most of the DB2 allocated sort work data sets, as they provide room for larger data sets than the regular volumes.

•As mentioned before, keeping RTS accurate is beneficial, i.e. activities that interfere with DB2 keeping the stats up to date should be avoided, like stopping the RTS table space or replacing objects outside of DB2. Also RTS values may not have been externalized to the tables yet in case of a DB2 crash, so that information may be lost. Since RTS are externalized with the STATSINT interval, consider setting it to a smaller value from the default of 30 minutes to reduce the possible loss in a DB2 crash.

•While we do use RTS for the number of records, REORG will still use the average row length from RUNSTATS to estimate the sort work data set size for the data sort. So RUNSTATS should still be run if the characteristics of the data have changed significantly.

•Talking about RUNSTATS: there’s still the conception that RUNSTATS would also update the global counts in RTS tables, however this is not correct. The majority of our customers will run RUNSTATS with SHRLEVEL CHANGE and this execution mode is not capable to determine an exact count of rows in the table space as it will read through the table space sequentially and miss parallel updates to already processed pages.

Deliverables

- Function was first made available in 02/2008:
  - DB2 for z/OS V8: PK45916 / UK33692 (PUT0802, RSU0806)
  - DB2 9 for z/OS: PK41899 / UK33636 (PUT0802, RSU0806)
  - Informational APAR: II14296
- Some recommended APARs:
  - PK64624 (UK36436/UK36437 - PUT0805, RSU0806): Different abends in LOAD with multiple INTO TABLE
  - PK64915 (UK36331/UK36332 - PUT0805, RSU0809): Improve estimates for REBUILD INDEX/CHECK INDEX with segmented table spaces and missing RTS
  - PK66597 (UK37865/UK37866 - PUT0807, RSU0808): LOAD ABEND0C4 RC00000011 when SYSTEMPL DD specified but not used
  - PK71733 (UK41940/UK41941 - PUT0812, RSU0903): DB2 over allocates sort work data sets when data sets are written in many fragments
  - PK80947 (UK45425/UK45426 - PUT0904, RSUxx): Utility failure when RTS value contains negative values or values exceeding physical database limits
  - PK90293 (UK48962 - PUTxx, RSUxx): REORG TABLESPACE ABEND04E RC00E40045 during partition unload/reload parallelism
  - DFSORT APAR PK63409 (PTF UK35433): ICE046A SORT CAPACITY EXCEEDED when estimate is slightly below actual value
    - Fix for DFSORT in z/OS 1.10 is provided in PK66899

This list of APARs is not exhaustive, but mentions the most important fixes related to the SORTNUM elimination enhancement and to the sorting area in general.

Sort Work Data Set Allocation Best Practices

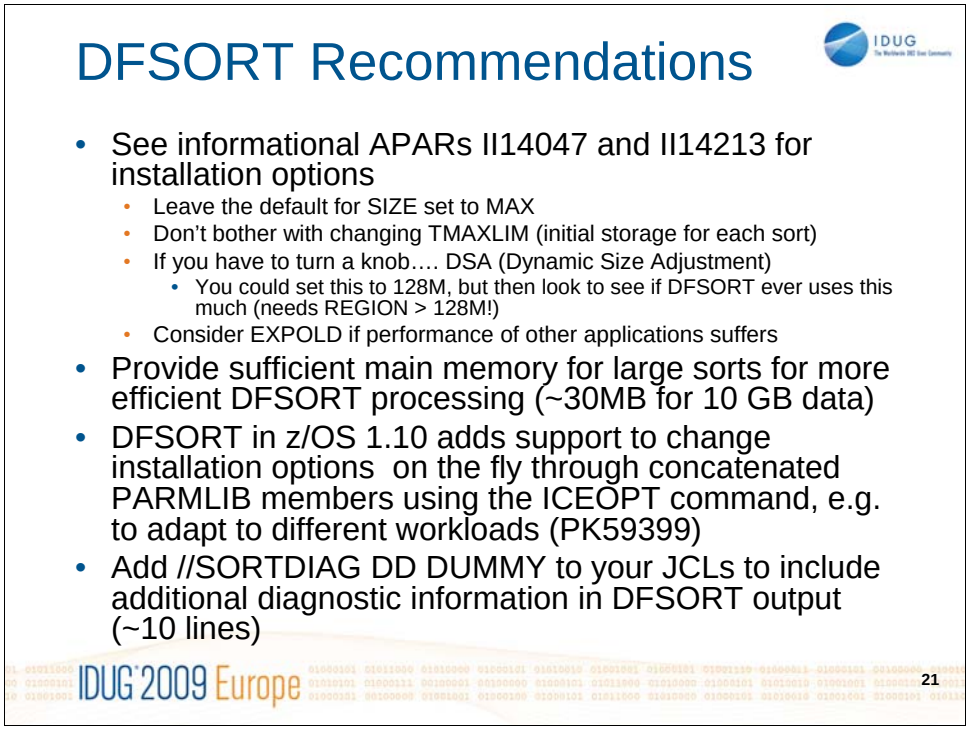
- Use large sort volumes, >64K tracks supported since z/OS V1R7
- To direct data sets to storage group, use ACS (see DFSMSrmm SMS ACS Support reference on References slide)
- If assigning data class through ACS:
  - make sure that it's defined as single volume only
  - Turn off space constraint relief in data class
- Sort work DD refresher for ACS routines: SORTWKnn, SWxxWKnn, DATAWKnn, DAxxWKnn, STATWKnn, STxxWKnn

Some best practices regarding sort processing in DB2 utilities:

•DFSORT is able to support large data sets for a long time now, and it also runs most efficiently with a small number of data sets. A small number of data sets also benefits the maximum degree of parallelism for utilities, so you can likely improve your elapsed time, if you have large sort volumes to reduce the number of data sets for very large sorts.

•Many of you use ACS routines to route sort work data sets to specific storage groups or data classes. A list of possible sort work data set DD names is listed at the end of this slide.

•When assigning a data class it's most important that you turn off the ability to allocate sort work data sets on multiple volumes (either directly defining them as multi-volume or letting space constraint relief expand a data set to multiple volumes). DFSORT can only use the part of the sort work data set that resides on the first volume, all other parts on additional volume will go unused by DFSORT – wasting disk space and likely causing SORT CAPACITY EXCEEDED failures.



Many customers want to tune the behaviour of DFSORT by providing additional resources (memory) but don't know, which knobs provide the best results. Our experience and also the recommendations from the DFSORT folks are to let DFSORT allocate main storage dynamically. This can be achieved by setting SIZE=MAX (we also pass MAINSIZE=MAX in DB2 9). With that configuration DFSORT will be able to increase their memory consumption up to the value defined in DSA (Dynamic Storage Adjustment). Of course you need to provide enough REGION to your job to let DFSORT's memory consumption grow to that value, especially in utilities with multiple parallel sort invocations. TMAXLIM is not a good value to modify in that configuration as DFSORT will start allocating the amount of memory defined in TMAXLIM when SIZE=MAX. So if TMAXLIM would be very large, all the dynamic storage adjustment would no longer be used and all DFSORT tasks simply allocate that much memory.

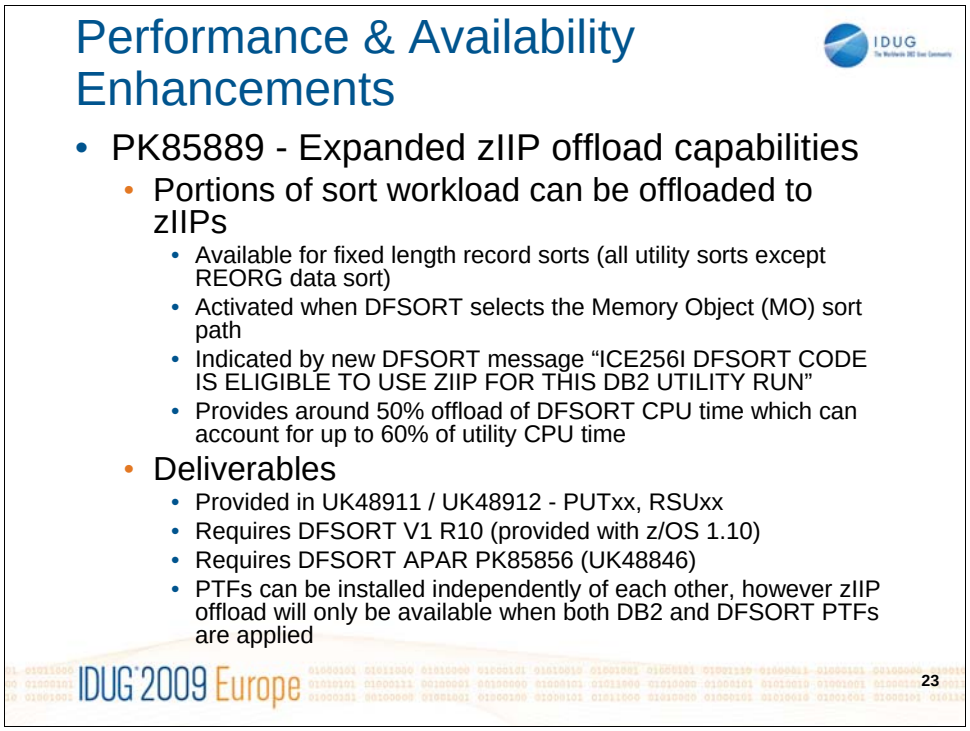
The larger the amount of data gets, that has to be sorted, the more main memory should be provided to DFSORT (through REGION and DSA settings). To give a rough idea in how much memory DFSORT feels most comfortable, here's a short list of data sizes and estimated memory consumption (data size=number of records * size of records): 1 GB data can well be sorted in 10 MB of storage, 10 GB data should have around 30 MB storage and 100 GB data should have at least 70 MB available for DFSORT. When less memory is available to DFSORT then it will still be able to complete the job, but will have to use less efficient algorithms, e.g. using intermediate merges which will increase the over all requirement for work space (e.g. sort work data sets, so that the estimated sort work data set size may not be enough).

DFSORT in z/OS 1.10 has provided a nice enhancement to modify installation defaults more easily by different members in PARMLIB that can be activated on the fly. Since DB2 also looks up current installation defaults for its own processing and estimation of parallel sort invocations, you should make sure to have PK59399 installed in DB2 if you're making use of the new PARMLIB options.

Finally an idea to improve your turn-around time when contacting DFSORT support about any problems: DFSORT will write additional diagnostic information to the JOBLOG if the SORTDIAG DD card exists. That information will help support folks dramatically to analyze the problem and in many cases they will ask you to try to recreate the problem after adding SORTDIAG DD to your job. So you can save on at least one turn between you and IBM if that information is available right from the beginning. The amount of data added is usually less than 10 lines so it should not hurt.

Maintenance Stream Updates

IDUG
The Worldwide DB2 User Community



DB2 utilities have been using zIIPs for index maintenance operations since they were introduced. Now we have expanded our zIIP offload capabilities further into sort processing. This additional offload is available for all utility sorts that use fixed length records (all sorts except REORG data sort) and it will be activated when DFSORT selects the Memory Object sort path. This sort path is usually chosen when enough main memory is available to DFSORT. When zIIP offload is activated, you will see the new message "ICE256I DFSORT CODE IS ELIGIBLE TO USE ZIIP FOR THIS DB2 UTILITY RUN" and your savings should be around 50% of the CPU time spent in DFSORT, as we have found during performance tests at the lab. Your results in savings may vary with availability of zIIP engines and also with the size of your sorts.

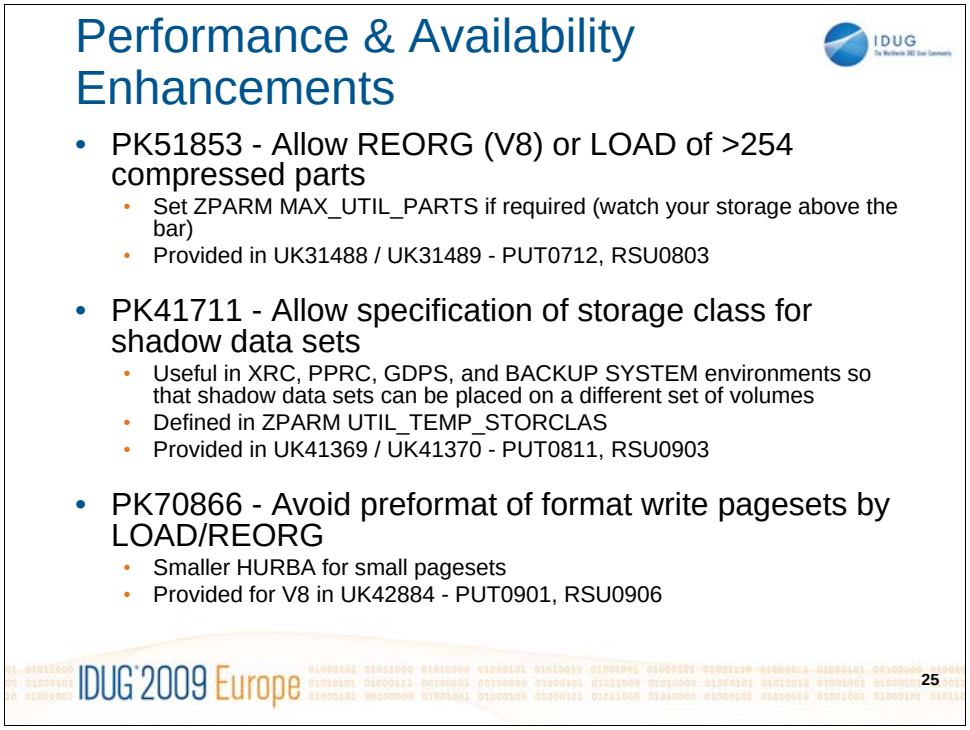
To benefit from this enhancement PTFs need to be installed for both DB2 and DFSORT. DB2 utilities support the expanded zIIP offload for both DB2 V8 and DB2 9. DFSORT has provided the necessary changes with DFSORT V1 R10 which comes with z/OS 1.10. PTFs for DB2 and DFSORT can be installed independently of each other, but the offload will only be available if both have been applied to the system.

Performance & Availability Enhancements

- PK60956 - SORTBLD performance improvement for indexes with small SECQTY
  - SORTBLD elapsed up to 20x improvement!!!
  - Recommendation: Set appropriate PRIQTY/SECQTY to avoid extend processing
  - Provided in UK34807 / UK34808 - PUT0804, RSU0809
- PK61759 - LOAD & REORG performance improvement
  - 10% CPU & ET improvement in reload phase
  - 10% CPU reduction in sort phase
  - Provided in UK36305 / UK36306 - PUT0805, RSU0809
- PK60612 - Allow unload from a non-segmented image copy when the table is segmented
  - Useful for retention/archiving since unable to create simple table spaces in V9
  - Provided in UK35132 - PUT0804, RSU0809

PK60956: While this PTF addresses a performance issue when creating additional extents for data sets, we still recommend to avoid extent processing as much as possible by either setting PRIQTY and SECQTY appropriately or let DB2 calculate the extent sizes

PK60612 is useful for retention purposes. DB2 9 no longer allows creation of simple table spaces, so you can no longer restore an old image copy of a simple table space if the table space had been dropped and has to be recreated. However with this support you can still run UNLOAD FROMCOPY to recover historical data even if the table space was dropped and recreated in the meantime.



PK51853: So far LOAD and REORG were limited to 254 partitions when handling compressed table spaces (except for REORG in DB2 9 which no longer has this restriction). It is caused by the intermediate storage requirements when building the compression dictionary. If you have slightly more than 254 partitions, you can no longer use LOAD or REORG for the whole table space, so this ZPARM can be used to moderately lift this threshold. You need to watch your storage consumption above the bar though that you're not over allocating memory if running too many utilities in parallel that are building dictionaries for many partitions.

PK41711: This is very useful in XRC, PPRC, or GDPS environments so that you no longer have to mirror the intermediate shadow data sets created by a couple of online utilities. With the new ZPARM you can specify a dedicated storage class to be used for those intermediate data sets, so that they end up on non-mirrored volumes.

Performance & Availability Enhancements

- PK70269 - TEMPLATE Utility enhancement to support LOAD/UNLOAD from/to a z/OS UNIX file (HFS) or pipe
  - Requires z/OS 1.8 and APARs OA25204, OA25206, and OA25280
  - Utility can't be restarted and cannot use DISCARDDDN (cannot distinguish between file and pipe)
  - Provided in UK43947 / UK43948 - PUT0903, RSU0906

```
TEMPLATE DATAFIL RECFM=V LRECL=80
         PATH='/u/cmichel/LoadInput.data'
         PATHOPTS=ORDONLY FILEDATA=RECORD
         PATHDISP=(KEEP,KEEP)
 LOAD DATA INDDN DATAFIL LOG NO RESUME YES
      INTO TABLE "abc"."xyz"
```


PK79269: LOAD had supported Batchpipes for some time now, and we have now added support of USS files and pipes to both LOAD and UNLOAD utilities. These USS pipes are quite handy to load a lot of data from remote sources into DB2 without first having to materialize that data as data sets in z/OS. An exploiter of this functionality for example is SAP's FastLoad procedure which benefits significantly from the removal of intermediate storage of the data on the host.

When using that enhancement you cannot use the DISCARDDDN option or restart an interrupted utility as we can't distinguish between real files and transient pipes, so we cannot resume from the last committed point. An example specification for USS pipes to be used from LOAD is on the slide.

Performance & Availability Enhancements

- PK63324 & PK63325 - LOAD COPYDICTIONARY (V9)
  - Allow priming of a partition with a compression dictionary
  - Only available with INTO TABLE … PART REPLACE
  - Provided for V9 in UK37144 and UK37143 - PUT0806, RSU0809

```
LOAD RESUME NO COPYDICTIONARY 3 …
     INTO TABLE xyz PART 4 REPLACE …
     INTO TABLE xyz PART 5 REPLACE …
```


PK63324 and PK63325: Provides the ability to copy an existing compression dictionary from one partition to another partition where the LOAD utility normally would not have enough data to build a new compression dictionary. It can only be used with classic partitioned table spaces or UTS partition-by-range table spaces. The COPYDICTIONARY also can only be used with RESUME NO REPLACE YES options.

Performance & Availability Enhancements

- PK74993 - 20% elapsed time improvement for COPY of small datasets to tape
  - Provided in UK45790 / UK45791 - PUT0904, RSUxx
- PK76860 - Improved data conversion for Crossloader (performance & function)
  - Provided in UK46235 / UK46236 - PUT0905, RSU0906
- PK77061 - Permit use of ALIASes for LOAD, RUNSTATS & UNLOAD
  - Provided in UK44580 / UK44581 - PUT0903, RSU0906
- PK71816 - New EXTNDICT option for DSN1COMP to externalize dictionaries
  - For encryption tool to support encryption and compression combined
  - Provided in UK41354 / UK41355 - PUT0811, RSU0903

This is a list of small but useful performance enhancements for various utilities.

PK71816 is used by the Encryption tool to support both encryption and compression at the same time, which is not supported by DB2 natively.

Performance & Availability Enhancements

- PK75216 - LOB & XML FRV performance improvement for PDS datasets (V9)
  - 56% ET reduction on UNLOAD
  - 93% ET reduction on LOAD
  - Provided for V9 in UK43355 - PUT0902, RSU0903
- PK79136 - Reduced DFSORT memory allocation for parallel utility sort processing (V9)
  - DFSORT will allocate memory in a more dynamic way (from TMAXLIM up to DSA)
  - Provided in UK44703 - PUT0903, RSU0906
  - May need DFSORT APAR PK89889 to fix loop in ICEIPUT for MAINSIZE=MAX invocations

PK75216: Fixes bad performance for LOB and XML file reference variables when working on partitioned data set members. The bad performance was caused by repeatedly opening and closing the PDS and that has been fixed. Performance improvements caused by that fix are significant.

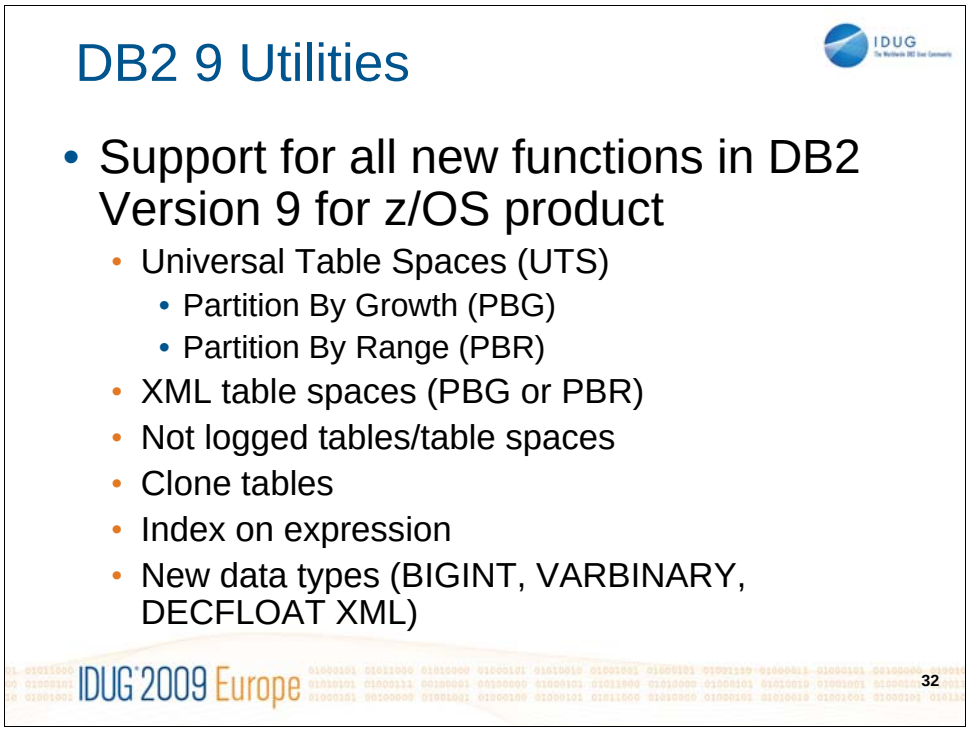
PK79136: This is especially useful for utilities with multiple parallel invocations of DFSORT. We now always pass MAINSIZE=MAX and tailor the maximum amount of memory for each DFSORT task via the DSA option (dynamic storage adjustment). That allows a more flexible memory allocation pattern for DFSORT. DFSORT will only allocate as much memory as needed for a particular invocation (depending on the number and size of the records that need to be sorted). The DFSORT parameter TMAXLIM will be used as the initial allocation size and the upper limit will be set by DB2 in the DSA parameter. This will result in significant less amount of memory allocated by DFSORT for most cases where the data being sorted is not too large. For very large sorts, DFSORT will still be able to grow its memory consumption to still be effective. Due to DFSORT's behaviour of starting the memory allocation at the value specified in TMAXLIM, it doesn't make much sense to change that installation option, especially not to set it to a very large value. Recent reports show a possible problem in DFSORT causing a loop after PK79136 was installed. PK89889 is currently open to address the loop - it's not confirmed yet that it's related to MAINSIZE=MAX invocations.

Performance & Availability Enhancements

- PK77313 - UNLOAD performance improvement in UTILINIT finding table names
  - Use DBD rather than catalog lookup – significant performance improvement if many tables in table space
  - Provided in UK43511 / UK43512 - PUT0902, RSU0906
- PK67154 - Avoid scanning NPI when running REORG TABLESPACE PART SHRLEVEL NONE on empty partition(s)
  - Provided in UK45137 / UK45138 - PUT0904, RSUxx
- PK78516 - DSN1COPY performance improvement for page sets with cylinder allocation
  - 20% ET improvement measured
  - Provided in UK43976 / UK43977 - PUT0902, RSU0906
- PK78865 - COPY performance with large LISTDEF lists
  - Reduce writes to SYSUTILX
  - Add diagnose option to avoid intermediate commits
  - Provided in UK45997 / UK45998 - PUT0905, RSUxx

Another list of smaller but useful performance enhancements listed on that slide.

The following section will give you a quick review of the most important DB2 9 utility enhancements. Since many customers are in the process to migrate to DB2 9 at this time (several have already completed their migration), this serves as a quick reminder what changes and enhancements you can expect from DB2 9.



DB2 9 provided a lot of new function that also needed modifications in DB2 utilities. So even without noticing on the externals, a lot of work has been done for those new functions, especially the new Universal Table Spaces, XML support, and clone tables. But still we have introduced a lot of new utility functions further detailed on the next pages.

Availability Enhancements

- REBUILD INDEX, CHECK DATA, CHECK LOB, REPAIR LOCATE now all support SHRLEVEL CHANGE
- LOAD REPLACE (SHRLEVEL CHANGE) functionality can be achieved through CLONE tables
- REORG LOB SHRLEVEL REFERENCE with space reclamation
- Partition level UNLOAD/RELOAD parallelism in REORG TABLESPACE
- No more BUILD2 phase for NPIs during REORG
- Limit of 254 parts per REORG on a compressed table space lifted; storage consumption reduced

A big goal for DB2 9 was to improve availability. For Utilities this usually means that we're providing utilities that can run with SHRLEVEL CHANGE, allowing full concurrent access by applications. Consequently REBUILD INDEX, CHECK DATA, CHECK LOB, and REPAIR LOCATE now all support SHRLEVEL CHANGE.

LOAD REPLACE can now also be used in a SHRLEVEL CHANGE behavior when using a clone table that is loaded and for the switch phase you simple exchange the base and the clone tables.

REORG LOB now supports SHRLEVEL REFERENCE and space reclamation which wasn't supported by SHRLEVEL NONE.

REORG TABLESPACE elapsed time can be reduced significantly by exploiting partition unload/reload parallelism. This will be selected automatically when appropriate and requirements are met (e.g. no hard coded DD cards that would limit parallelism). The BUILD2 phase was removed for NPIs during REORG, however that comes with a slight overhead, as the whole NPI will now be rewritten (not fully reorged) every time. Compressed table spaces can now also be REORGanized if they have more than 254 partitions, while the intermediate storage consumption to build the compression dictionaries was reduced significantly.

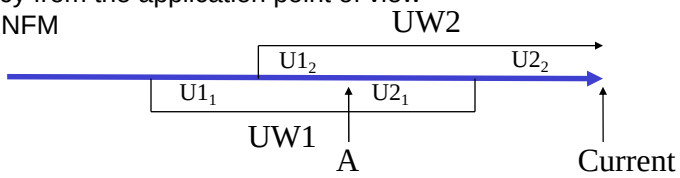
CPU Reductions

- Reductions mostly due to improved index processing (* with exceptions)
  - 5 to 20% in Recover Index, Rebuild Index, Reorg, and table space Image Copy* (even with forced CHECKPAGE YES)
    - Except REORG TABLESPACE SHR CHG PART with NPIs
  - 5 to 30% in Load
  - 20 to 60% in Check Index
  - 35% in Load Partition
  - 30 to 50% in Runstats Index
  - 40 to 50% in Reorg Index
  - Up to 70% in Load Replace Partition with dummy input

We improved all of our utilities in the index processing parts and were able to save quite a bit on CPU in these areas. Your mileage may vary depending on the number and key size of your indexes.

Backup & Recovery

- The ability to recover to any point in time with consistency
  - Uncommitted changes are backed out
  - Significantly reduces (eliminates?) the need to run QUIESCE which can be disruptive to applications
  - Does not apply to RECOVER TOCOPY, TOLASTCOPY and TOLASTFULLCOPY using SHRLEVEL CHANGE copy (consistency is not ensured – use RBA/LRSN after COPY point)
  - Include all relevant objects in same RECOVER ensure data consistency from the application point of view
  - Requires NFM



UWn - Unit of Work number n

Un_m – Update number n in Unit of Work m

In the Backup and Recovery area we now provide the ability to recover to any point in time with consistency. This feature is only enabled with DB2 NFM and will be used automatically for all point in time recoveries (not one of the TOCOPY invocations referring to a SHRLEVEL CHANGE copy). The important thing to note when using this functionality is to make sure that all related objects that were involved in open transactions are included in the RECOVER invocation. Only objects included in the RECOVER invocation can be processed in the roll-back of the in-flight transactions.

With this ability we think that the need to run the QUIESCE utility to establish a recovery point can be reduced greatly. It probably won't eliminate its use, but the need for it should be significantly less.

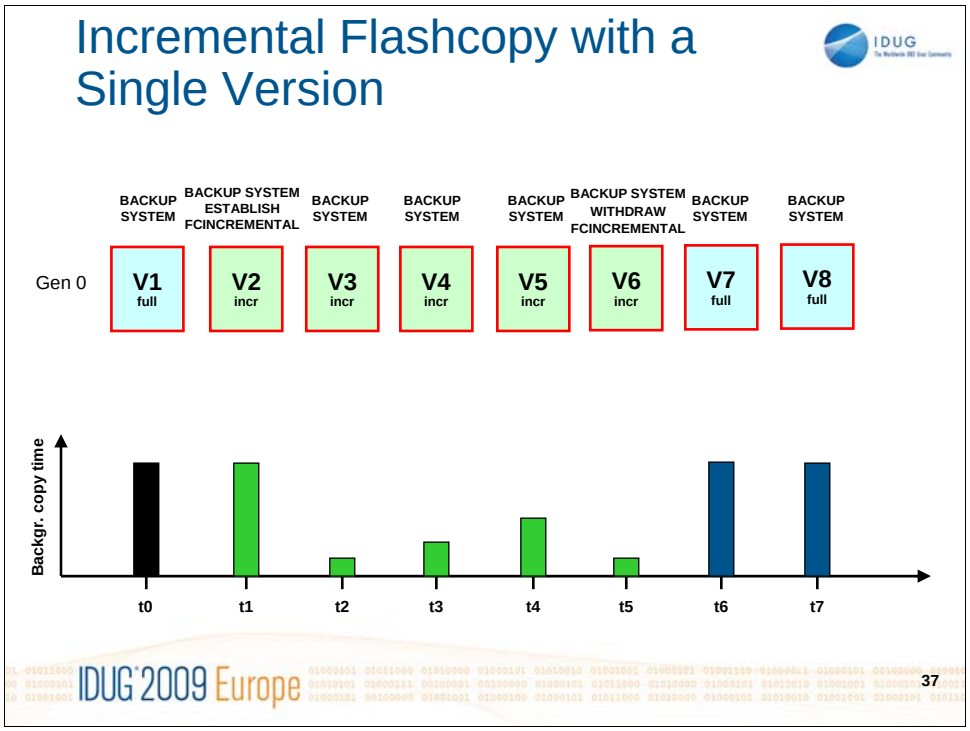
Backup & Recovery

- COPY now always performs CHECKPAGE (CPU reduced)
- RECOVER at the object level from system level backups
- BACKUP & RESTORE SYSTEM
  - Tape control added
  - Support for Incremental FlashCopy (APARs PK41001/PK42014)

The COPY utility now also does page checking all the time, it can no longer be selected as optional processing. For those customers who were afraid of spending the extra CPU for that function, we can assure you that even with CHECKPAGE being always performed, we have reduced the overall CPU consumption for the COPY utility compared to DB2 V8 without CHECKPAGE. When errors are found then the object will no longer be set into COPY PENDING, which made it inaccessible for applications, but we still issue the existing messages for page errors and then end with RC=8. So watch your return codes and messages from your automation processes to take the correct actions after page errors have been found.

SYSTEM BACKUP (volume based backups) have also been extended in several ways. One important change is that the RECOVER utility for single objects is now able to retrieve those objects from a system level backup source. This will save you from taking too many individual backups for objects that don't change too often without loosing recoverability.

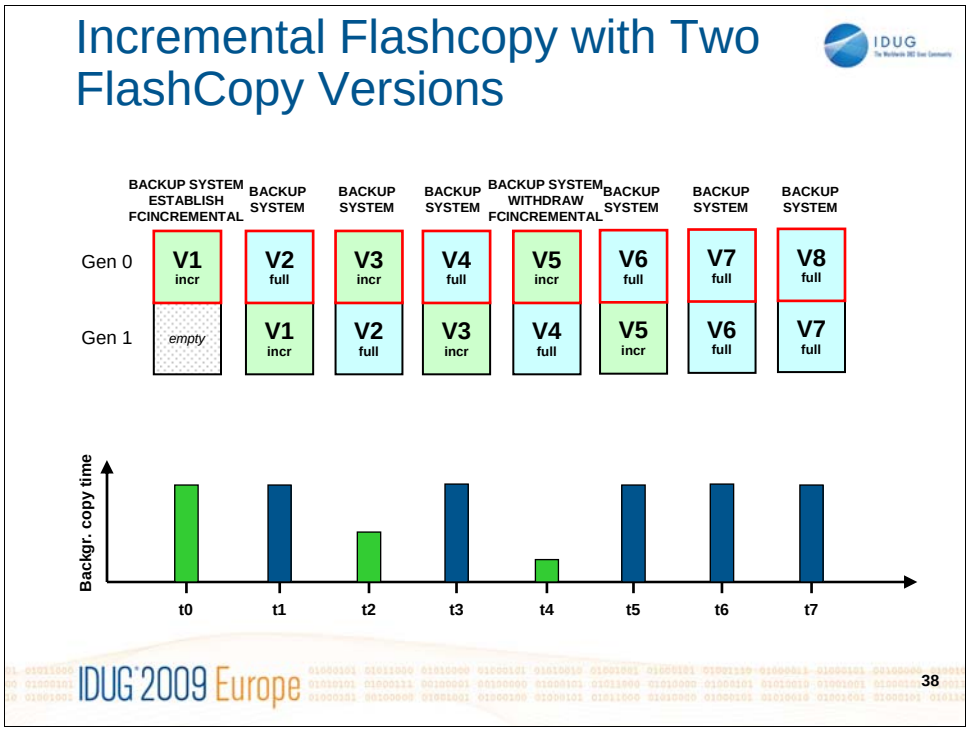
The BACKUP SYSTEM and RESTORE SYSTEM utilities have been enhanced to provide tape control right at the utility statement. You no longer need to migrate your flashcopy backups to tape, once the system backup is completed. After GA we also provided support for incremental flashcopy backups, which are further described on the following slides.



While the name sounds similar to incremental image copies, it's a totally different approach. The resulting backup copy is always a full copy of the source system, so you will always be able to fully restore all your data even if the last system backup taken was only an incremental flashcopy. The incremental flashcopy is qualified by the amount of data that will be copied from the source to the target. Once an incremental flashcopy relationship is established, DFSMS will monitor changes to the source volumes since the last backup was taken and will then only copy changed tracks to the target when a new backup is performed. This means that a persistent relationship between the source and target volumes is maintained by DFSMS and there can only be one incremental relationship active.

The example above illustrate the amount of data (and the required time to perform the background copy) for different backups. At t0 a regular BACKUP SYSTEM is performed which needs to copy all source volume tracks to the target. At t1 a new incremental relationship will be established. Since no information is available on which tracks were modified since t0, again all tracks need to be copied over to the target volumes. However DFSMS will now start monitoring the source volumes and keeps track of all the changes. So on subsequent BACKUP SYSTEM invocations at t2, t3 and so on, only the tracks need to be copied to the target which were changed since the last copy. The target will always contain a full copy of the volume as they were looking at those specific points in time.

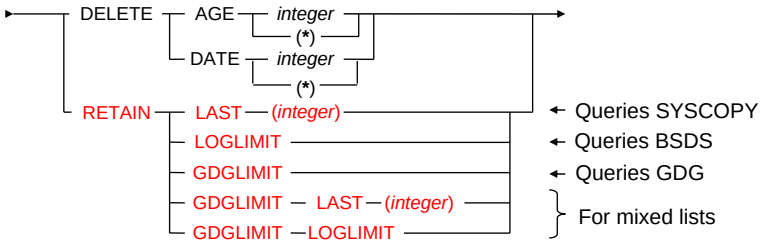
This example was using a single version in the copy pool, i.e. a 1:1 relationship between number of source volumes and target volumes.



If the copy pool supports two versions, then only one version can use the incremental relationship, while the other version always will perform full flashcopy. The two volume target groups will be alternating for each BACKUP SYSTEM invocation, so every second BACKUP SYSTEM will benefit from the incremental relationship. This could for example be the BACKUP SYSTEM invocation that is performed during high load times during the day, while the second BACKUP SYSTEM is invoked during the night when the load on your DASD is less and can also cope with full background copies.

MODIFY RECOVERY

- Modify Recovery simplification and safety



- Syslgrnx records deleted even if no syscopy records deleted (only available in NFM)
- Deletion works on dates not on timestamps, so more entries than requested might be kept

MODIFY RECOVERY has been extended with new keywords that help to determine the deletion date for SYSLGRNX and SYSCOPY records for a specific object. Instead of specifying an age or date which can match a different number of backups taken in that time frame, the RETAIN LAST and RETAIN GDGLIMIT keywords will attempt to retain the same number of backups for each object. RETAIN LAST simply specifies the number of full image copies that should be kept, while RETAIN GDGLIMIT will determine the number of full backups from the definition of the GDG where the image copies were stored.

When processing a large number of objects with LISTDEF, both options can be specified at the same time. If the first image copy found for an object was written to a GDG, then GDGLIMIT will be used, otherwise LAST would take effect.

RETAIN LOGLIMIT finally can be used to delete all entries that are no longer covered by archive and active log entries.

Internally MODIFY RECOVERY still works on dates and not on timestamps. So even if you specified LAST 5 for example, it may be that the last 6 full image copies are retained if the oldest two image copies were taken on the same day. The example on the following pages will explain that.

MODIFY RECOVERY Best Practices

- Run MODIFY RECOVERY regularly to clean up old records in SYSCOPY and SYSLGRNX
- DB2 9 has RETAIN LAST n or GDGLIMIT
- Also resets “ALTER_ADD_COLUM” flag in OBD when deleting image copies with previous row versions
  - MODIFY RECOVERY DELETE AGE/DATE to delete everything before the REORG that follows the ALTER
  - Will make next REORG more efficient if no more old row versions exist
- Remember that MODIFY RECOVERY works on day boundaries

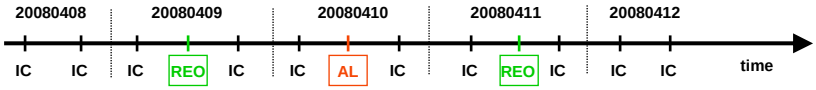
MODIFY RECOVERY is usually run to clean up old records in SYSLGRNX and SYSCOPY. But it does not only clean up those tables and prevents them from growing infinitely but it also resets a flag we keep in the OBD about alter added columns. This flag is needed for point in time recovery to work correctly. As long as there are image copies out there that can be recovered we need to make sure that we keep the record format information for the row versions that were used when those image copies were taken. Even if the data sets might have gone already DB2 still keeps track of those image copies in SYSCOPY. So make sure that MODIFY RECOVERY is run on a regular basis to get rid of all image copy entries after a new column was added to a table and before the next REORG that follows that ALTER which materialized the new column.

If the ALTER_ADD_COLUMN flag is not reset, REORG doesn’t know if the column has been materialized or not, so it must fully convert the rows to external format to populate the added column. After the flag is set, REORG doesn’t have to materialize any columns, so it can optimize the format of the data for unload and reloaded without fully converting it.

DB2 9 for z/OS has a new option RETAIN that can be used to retain a number of image copies or even retrieve the number of image copies to retain from the GDGLIMIT into which the image copies were written. This makes it even easier to run MODIFY RECOVERY

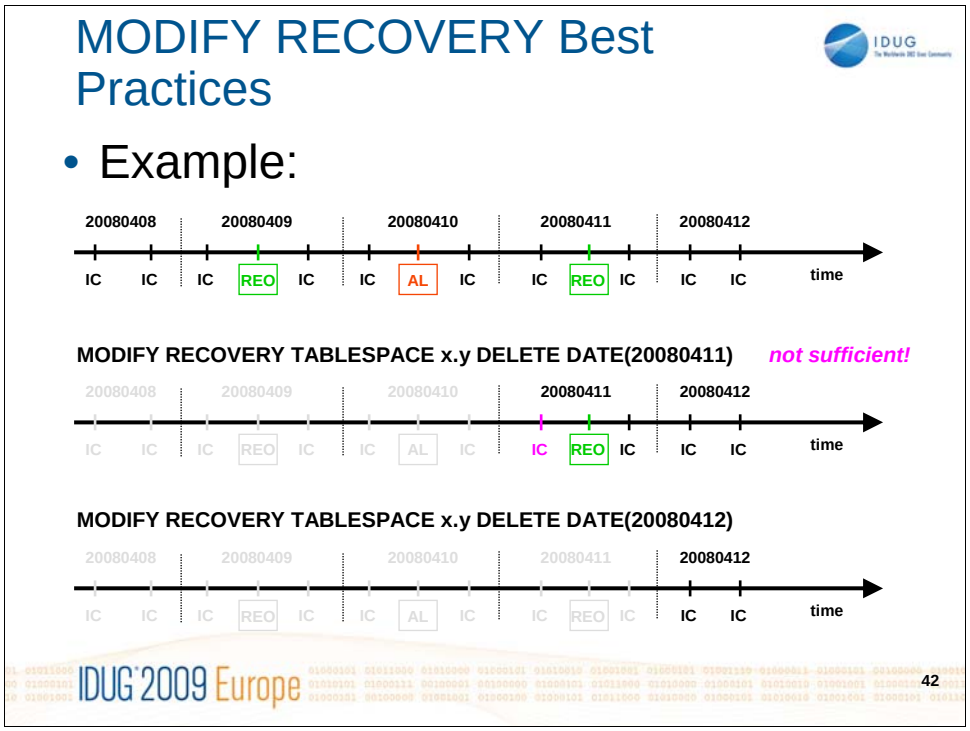
MODIFY RECOVERY Best Practices

- Example:



This is a short example to illustrate how MODIFY RECOVERY needs to be invoked in order to reset the ALTER ADDED COLUMN flag in the OBD. “IC” denotes image copies (and we just assume full image copies for all examples). “AL” is the ALTER where a new column is added to the table. “REO” finally illustrates REORG TABLESPACE.

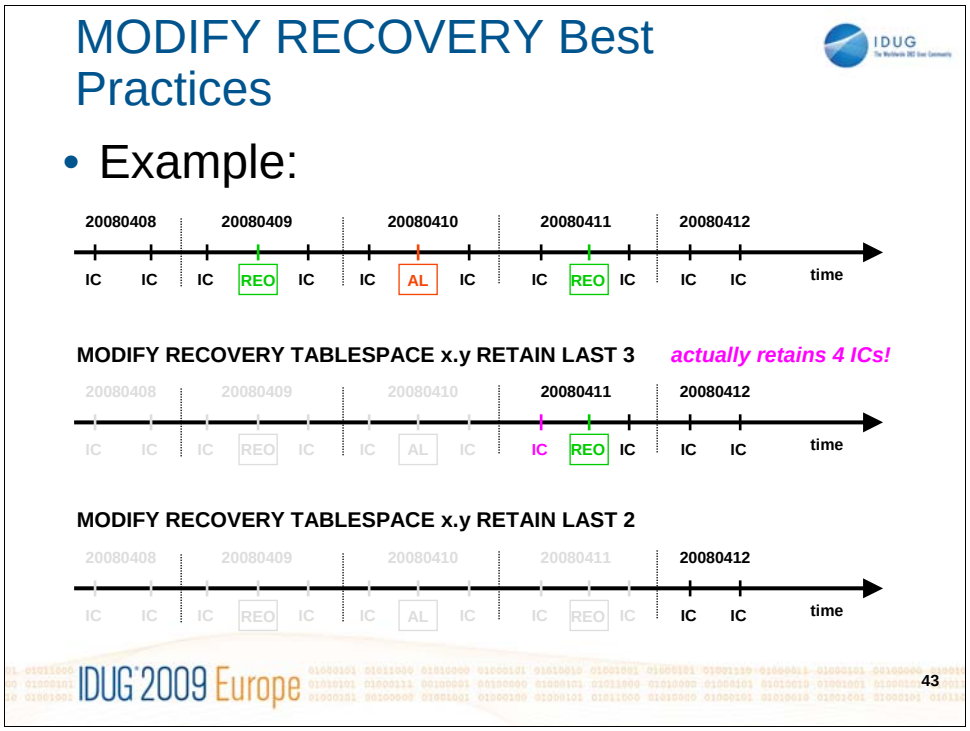
After the ALTER command the table space will still contain rows of the “old” format records. Those records will be part of image copies taken after the ALTER command. For that reason we need to keep track of the fact that there might be alter added records and we need to retain the table descriptor of the old format.



In order to get rid of the ALTER ADDED COLUMN flag and the old table descriptor it is necessary to run MODIFY RECOVERY to delete all image copies that could contain old format records. MODIFY RECOVERY needs to delete ALL image copies before the next REORG that followed the ALTER command (or the last ALTER command).

If MODIFY RECOVERY … DELETE DATE(20080411) is invoked then it will get rid of all image copies before the ALTER and even one after it but this is not sufficient as there is one more image copy that might contain old records. This means the ALTER ADDED COLUMN flag will not be reset with that invocation.

Only when MODIFY RECOVERY is run with DELETE DATE(20080412) will that flag be reset, as all “old format” image copies will then be deleted.



The same example can also be used to illustrate how RETAIN LAST works which was introduced in DB2 9 for z/OS. Internally it will always work on day boundaries. So if RETAIN LAST 3 is specified it will delete all records before the day when the last 3 full image copies were taken. As the third last image copy was taken on 2008-04-11 it will delete everything before that day as if invoked with DELETE DATE(20080411). This will actually retain 4 image copies, so you should not be surprised if that happens.

If invoked with RETAIN LAST 2 in that particular example this will result in exactly that number of full image copies being retained.

References

- DB2 for z/OS home page:
  http://www-306.ibm.com/software/data/db2/zos/
- Redbooks® DB2 9 for z/OS Technical Overview:
  http://www.redbooks.ibm.com/abstracts/sg247330.html?Open
- Redbooks® DB2 9 for z/OS Performance Topics:
  http://www.redbooks.ibm.com/abstracts/sg247473.html?Open
- Redbooks® DB2 for z/OS and OS/390 Version 7 Using the Utilities Suite:
  http://www.redbooks.ibm.com/abstracts/sg246289.html?Open
- Recommendations for Tuning Large DFSORT Tasks
  http://www.ibm.com/servers/storage/support/software/sort/mvs/tuning/index.html
- DFSMSrmm SMS ACS Support
  http://www.redbooks.ibm.com/abstracts/TIPS0530.html?Open
- IDUG Solutions Journal
  http://www.idug.org

DB2 for z/OS Information Resources

- Information Management Software for z/OS Solutions Information Center
  http://publib.boulder.ibm.com/infocenter/dzichelp/index.jsp
- DB2 for z/OS Information Roadmap
  http://ibm.com/software/db2zos/roadmap.html
- DB2 for z/OS library page
  http://ibm.com/software/db2zos/library.html
- DB2 for z/OS Exchange (examples trading post)
  http://ibm.com/software/db2zos/exHome.html
- DB2 for z/OS support
  http://ibm.com/software/db2zos/support.html
- Official Introduction to DB2 for z/OS
  http://ibm.com/software/data/education/bookstore

Summary

- SORTNUM elimination PTFs can help you to run all your sorting utilities more effectively
- List of enhancements in the maintenance stream
- DB2 9 enhancements revisited
- Any questions?

Thank you for your attention!

Christian Michel
IBM Germany, Boeblingen Lab
cmichel@de.ibm.com