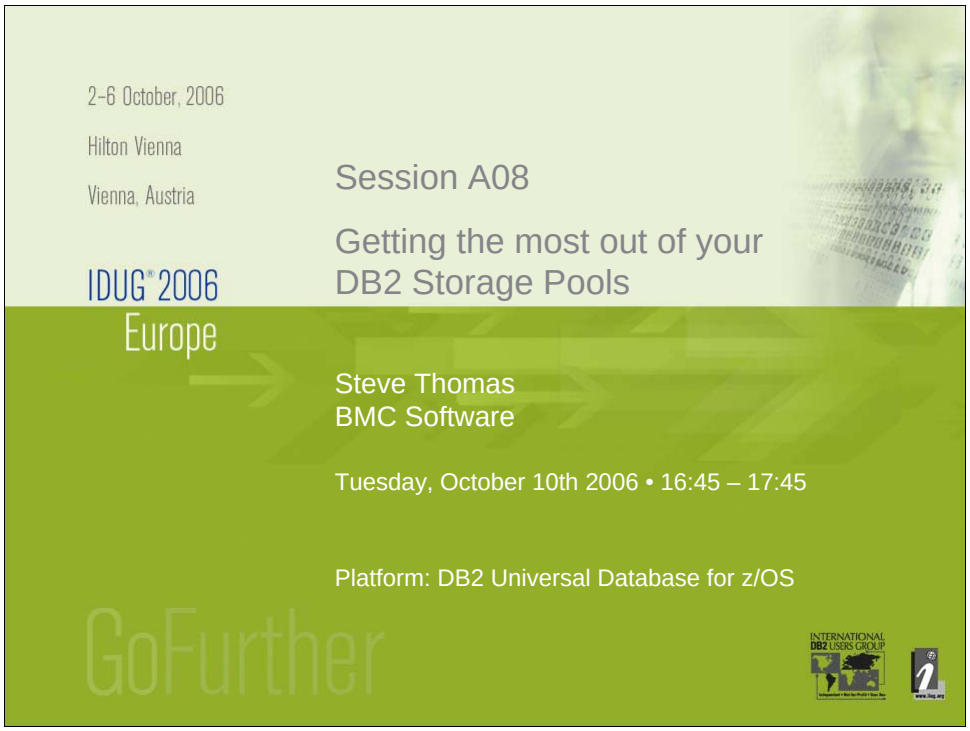
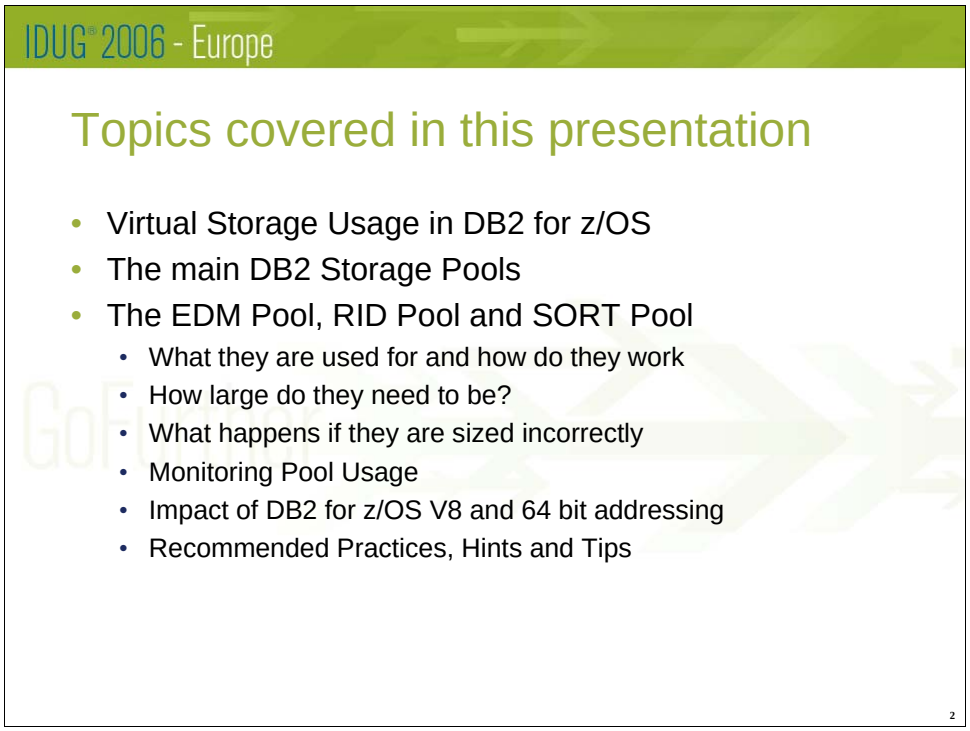


While there has always been a justified focus on Tuning DB2 Bufferpools, there are a number of other important Storage Pools used by DB2 for z/OS including the RID Pool, the EDM Pool and the SORT Pool. This presentation describes what information DB2 saves in each of these primary storage areas and explains how this is used. We will discuss the various methods which can be used to monitor Pool usage and to determine whether they are sized correctly together with the potential impact on Application and overall System Performance if they are sized incorrectly. The presentation also includes an explanation of the impact made by the introduction of 64 bit storage and the changes introduced in DB2 Version 8 for each of the Pools.

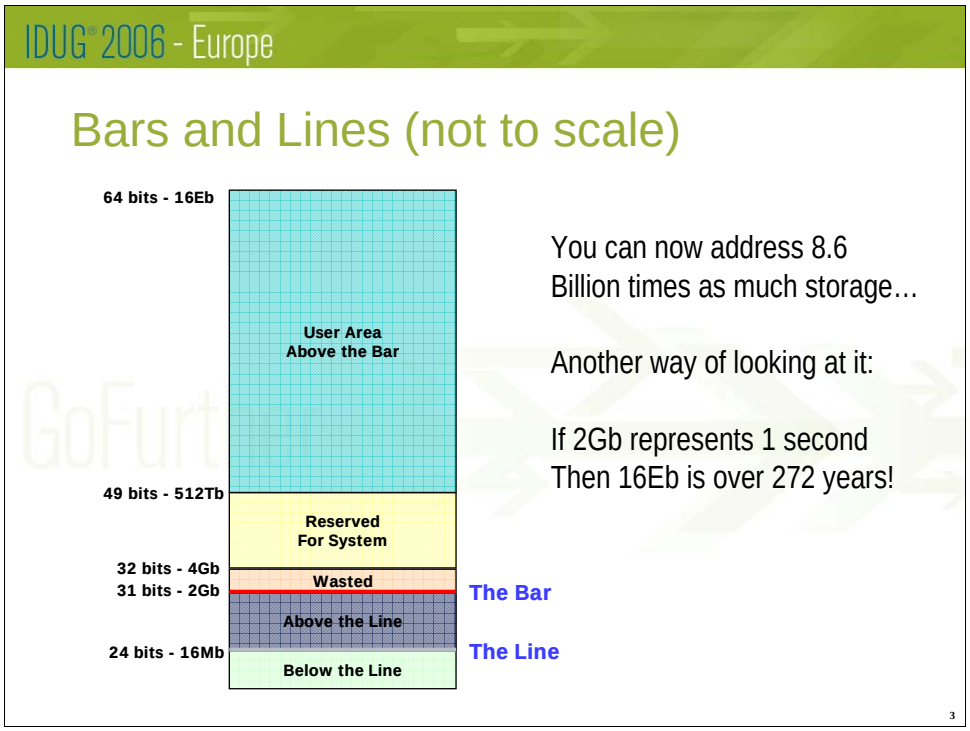


Presentations covering how DB2 Bufferpools are provided at most major DB2 Conferences, including IDUG here in Vienna. Such a focus is easily justified since the DB2 Bufferpools are usually the largest consumers of DB2 Storage and provide good opportunities for tuning performance. In contrast, the other major Pools, such as the EDM Pool, RID Pool and SORT Pools, are often comparatively neglected, which this session aims to correct by discussing these 'forgotten' Storage Pools in more detail.

The presentation begins with an overview of how Virtual Storage is used by DB2 and where it is allocated, including the new 'Above the Bar' storage provided by the introduction of 64 bit addressing in DB2 V8. I will then explain how DB2 splits the memory it uses into different Storage Pools, and what each of the largest areas is used for.

Following this introduction, I will be spending some time discussing the EDM, RID and SORT Pools in turn. I will explain how each Pool is used by DB2 to improve performance in more detail, how to calculate the correct size for each Pool in your own environment, what the consequences can be if it's the wrong size and how to monitor each Pool to ensure it is being used effectively. Finally we will discuss the changes introduced in DB2 V8, with a focus on the impact of 64 bit addressing.

Throughout the presentation I hope to be able to provide you with some recommendations, hints and tips which you can apply to your own environments. The presentation is targeted at Beginners to DB2 for z/OS and those with an Intermediate level of experience.

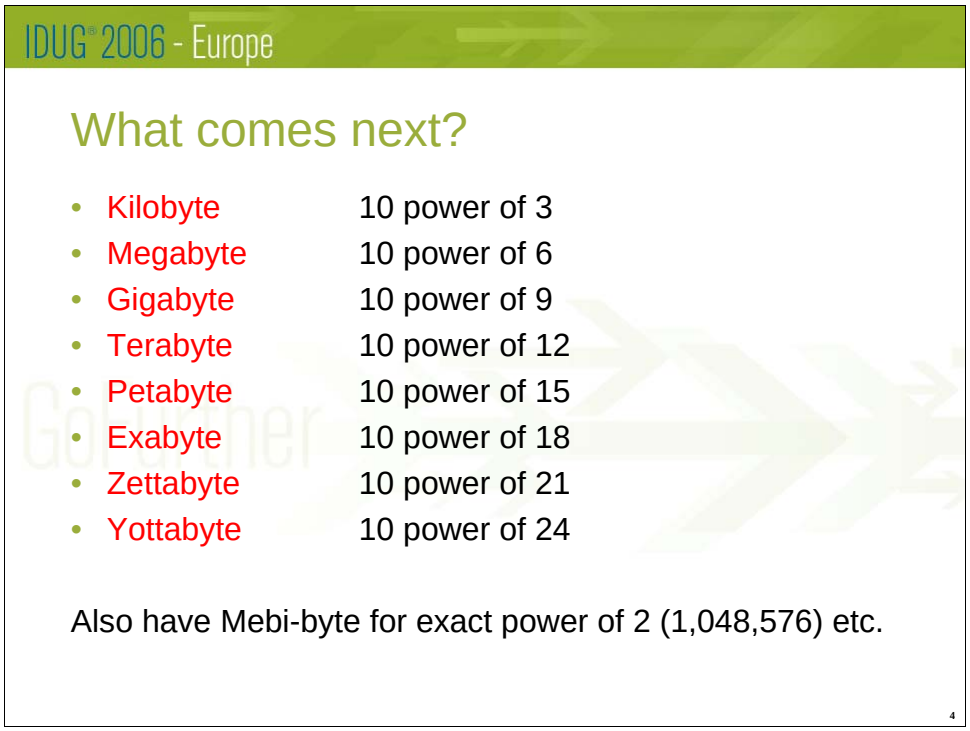


The first Mainframe operating systems used 24 bit memory addresses. This imposed the first address space limit of 16Mb, which is now known as 'Below the Line' Storage. Most storage is allocated in other areas today and so this does not present much of a problem at most sites, although you still need to be aware of it as certain control blocks still have to reside here.

When MVS/XA was released in 1981 we were able to address 31 bit addresses in Virtual Storage. This provided 2Gb of memory, and those of us who were around at the time will remember people saying you could never use this much which demonstrates how times change! No machine at the time had that much Real Storage available and so paging was used to keep within physical memory limits. The area between the 16Mb line and this 2Gb limit is generally called 'Above the Line' Storage. Until recently this is all that's been available, and it's this area that has been causing problems over the past few years as it started to run out, particularly in DB2's DBM1 address space.

In 2001, the introduction of z/OS 1.2 provided full 64 bit addressing (although there was some support in OS/390 2.10). At the time, very few applications or System features could use this – for example it wasn't until V8 that DB2 was able to exploit it properly. 64 bit support provides 16 Exabytes of addressable memory which seems like a vast amount, but remember the comments we all made about 2Gb not so long ago.

The 2Gb area between the 31 and 32 bit address limits (from 2Gb to 4Gb) is wasted, while the next region, up to 512Tb, has been reserved for system use. This sounds a lot, but it still leaves us with around 8.6 billion times more adressable storage to use than before which is an almost inconceivable amount, as can be seen by the comparisons on the slide. However, trust me that we will use all of this in time and probably in a lot less than the 20 years it took to use the old 2Gb limit.



Here's a quiz for the audience (although if you're reading the Notes you can already see the answer). What comes next? The SI unit prefixes for 10 are fairly well known up to Tera- (Power of 12), and we are starting to become more familiar with Peta- (15) and now Exa- (18), but what comes next?

The answers are Zetta- (Power of 21) and Yotta- (Power of 24), so don't be surprised if you start seeing these appearing more often soon.

Actually, because computers work with powers of 2 but people tend to work in powers of 10, there are some inconsistencies in the way most people speak. When you say a Megabtye you tend to mean 1,000,000 bytes whereas a Computer Megabyte is more accurately 1,048,576 bytes. As the powers get bigger, these differences become more noticeable. As a result there are another set of prefixes which were agreed by the International Elecrotechnical Commission in 1998. These use prefixes such as kibi-, mebi- and gibi- to provide exact powers of 2 (the idea is that mebi- stands for megabinary, or 2^10 squared). Thus, we could use a Megabyte to mean exactly 1,000,000 bytes and a Mebibyte to mean 1,048,576. As far as I know these haven't yet caught on in commercial computing, so we're stuck with the slight inconsistencies but perhaps that's easier to deal with.

IDUG® 2006 - Europe

Where DB2 Storage is allocated

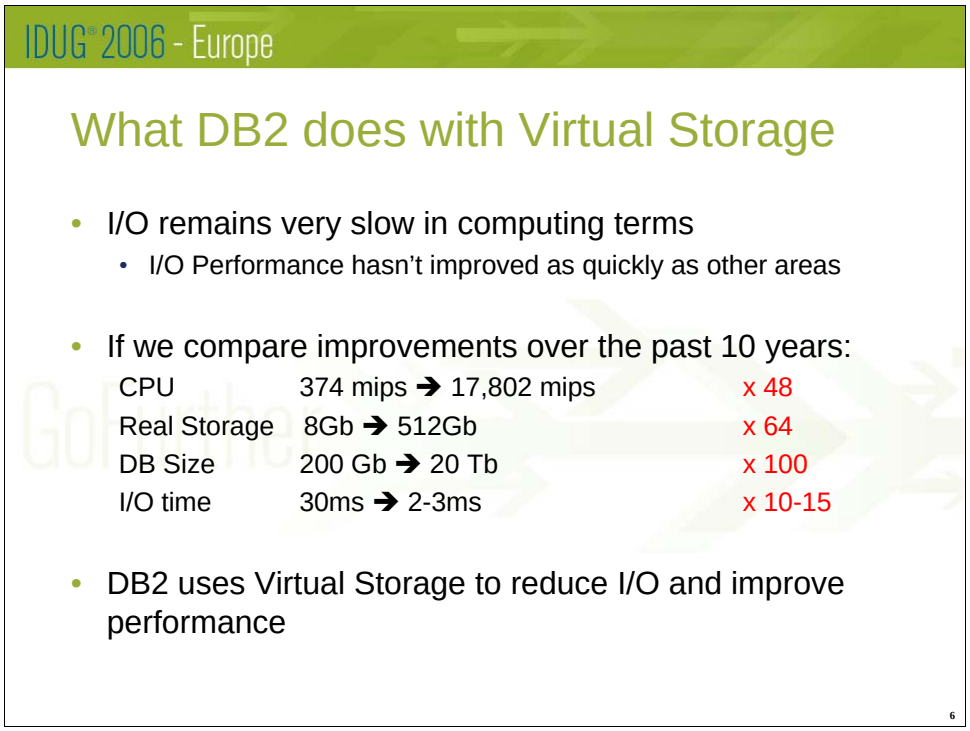
- Locks Stored in IRLM
- Most of the rest comes out of DBM1
  - Limited to 2Gb below line before V8
  - Some relief available by using Data Spaces
  - Much more help in V8 using 64 bit addressing
- Many conflicting demands for limited resource
  - Results in a juggling act at many sites
  - One of the main reasons for early V8 adopters
  - Others are better qualified to discuss this in detail
  - For example John Campbell's Sessions A06/A07 'Virtual Storage Management Revisited"

5

A DB2 subsystem comprises of many address spaces, with each being responsible for certain types of work. However, apart from Locking information which is stored within the IRLM address space (and is always above the Bar when using DB2 V8), most of the storage intensive processing is associated with the DBM1 address space.

Before V8, DB2 always used 31 bit addressing and therefore each address space could only use 2Gb of Virtual Storage. It's a very serious problem if you do reach or even get close to this limit. At best you'll reduce system throughput and at worst the entire DB2 subsystem can fail. A number of methods of providing reducing Virtual Storage requirements within the DBM1 address space, are available such as using Data Sharing to spread the load amongst several subsystems and placing Virtual Bufferpools in Data Spaces rather than in the Address Space itself. However, even when using these some sites have been in danger of hitting this 2Gb constraint within their DBM1 Address Spaces unless they took further action to reduce Virtual Storage usage. For these customers, it usually ends up being something of a juggling act to get the best workload throughput while remaining safe as far as Storage is concerned, and this has been one of the main reasons for moving to DB2 V8 among the early adopters.

There are a number of experts in this field who are much better qualified than me to discuss this subject in more detail. I recommend in particular that if you're interested in this area you attend John Campbell's session 'Virtual Storage Management Revisited' in Sessions A06 & A07 of this Conference, or obtain one of his excellent papers on the subject, for example from the IBM ftp site at ftp://ftp.software.ibm.com/software/data/db2zos/DB2VSTORprez.pdf

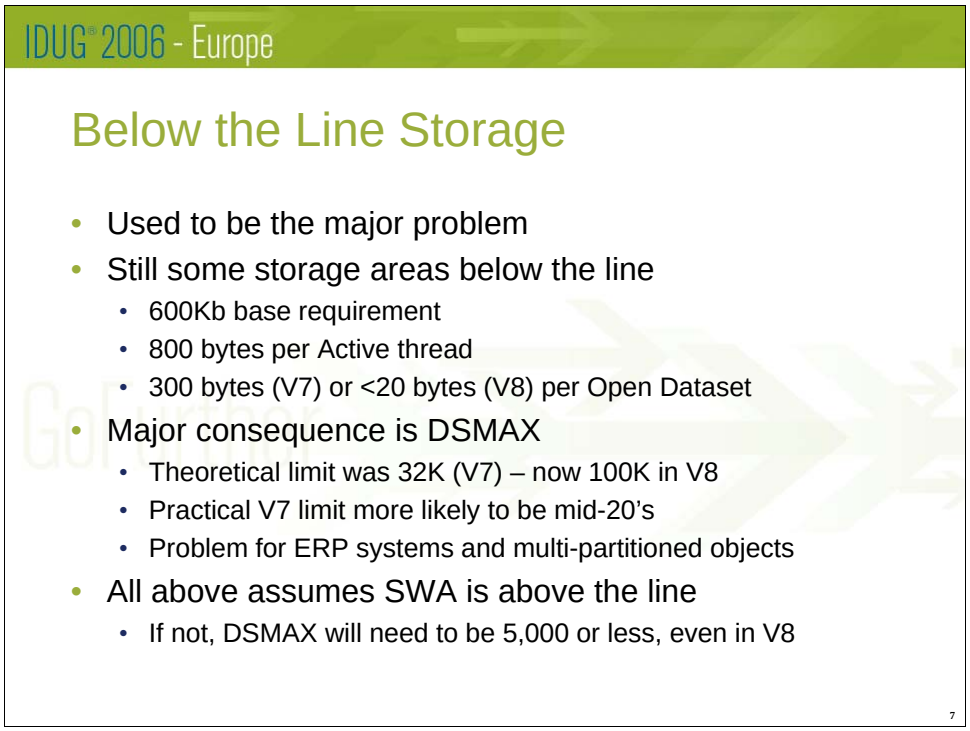


I/O has always been slow in computing terms, and improvements in back-end I/O performance have not been as marked as in many other areas. There have been major improvements in Disk Technology, particularly in terms of capacity. However, many of the performance benefits have been provided by the use of large Caches and intelligent Disk Controllers rather than straight I/O performance.

To demonstrate this, compare a range of factors from 10 years ago we were using IBM 9672 G5 processors to today's systems using z9's . The largest machine available in 1996 was the RY4 model, rated at 374 mips which could be configured with 8Gb of real storage. In contrast, the largest today is a z9 Model 754, rated at around 17,802 mips configurable with up to 512Gb of real storage. These represent 48-fold and 64-fold improvements respectively. Disk technology is harder to measure in many ways, but I estimate that a large database in 1996 would have been around 200Gb, while databases of up to 20Tb are not uncommon today, which is a 100-fold increase.

At the same time the I/O response time for a synchronous read has improved, but not as dramatically. Back in 1996 you might have expected such an I/O to take around 30ms, whereas now it is likely to be more like 2-3ms, which is 'only' a 10 to 15-fold improvement. Even if you were optimistic and estimated that an I/O took 1ms, this still represents only a 30-fold improvement, which is much less than the other components.

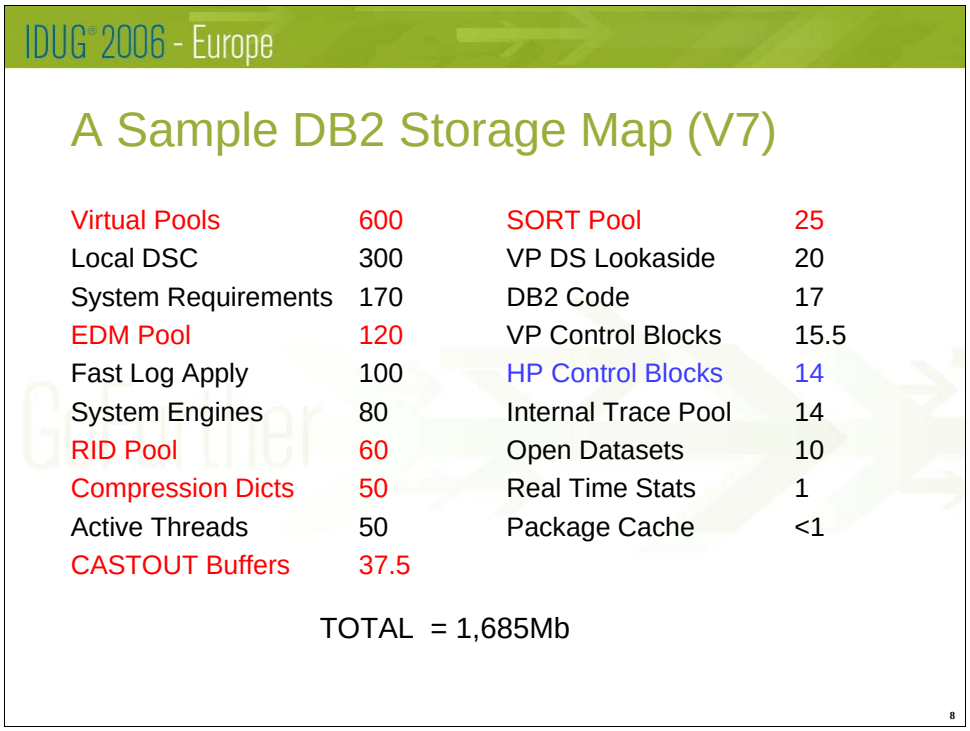
DB2 uses Virtual Storage to provide a Cache for data that is frequently accessed in an attempt to reduce I/O as much as possible. Bufferpools are used to Cache Application Data, while the other Storage Pools do the same for system data such as access path information, and to reduce I/O required during the SQL evaluation process.



Out of the 16Mb of storage below the line, DB2 requires around 600Kb as a base requirement in both V7 and V8. Each active thread requires around 800 bytes, again consistent between V7 and V8. One big difference between the releases is in the storage required for each Open Dataset. In Version 7 this was around 300 bytes, but it's gone down to less than 20 bytes in V8 (0.016Kb according to the Installation Guide).

Given that many customers have a relatively fixed requirement for how many users they have, the major consequence of Below the Line Storage is the number of Open Datasets. For a long time, the upper limit (in DSNZPARM DSMAX) was 10,000 directly as a result of this requirement for below the line Storage. As the amount of space required per dataset has reduced, the maximum number of Open datasets could be increased. In DB2 V7 this was theoretically 32K but in reality few sites could manage anything above the mid-20's. In DB2 V8 this DSMAX limit is now 100,000, largely due to the massive reduction in Storage required below the line per dataset. The Storage is still required somewhere (it's now mostly above the line), but it's no longer in this very constrained area.

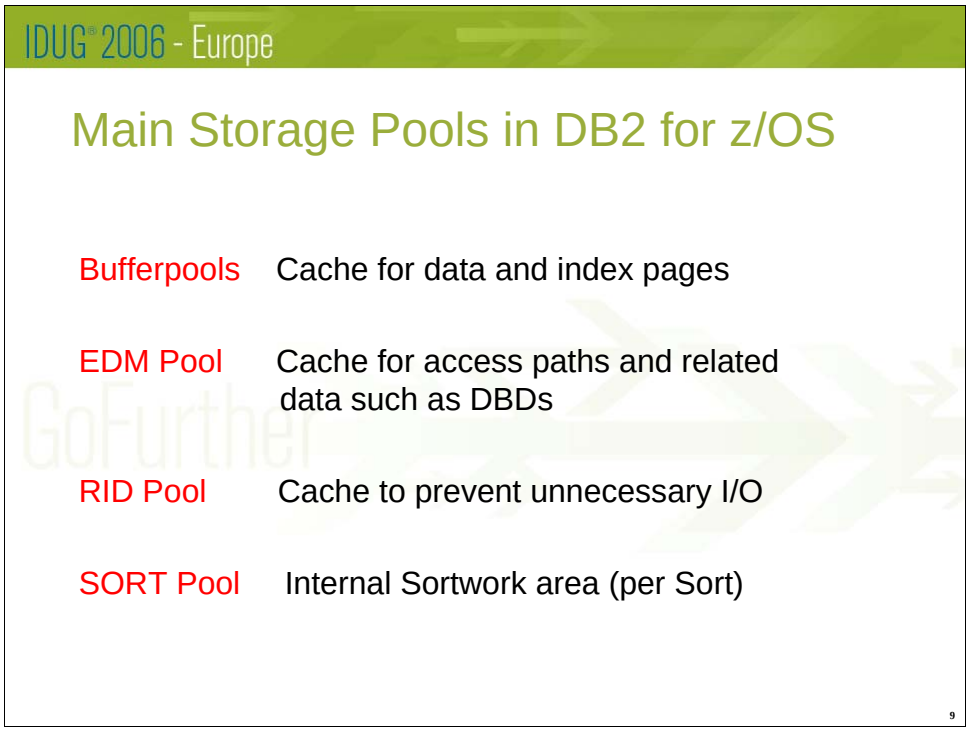
All the above assumes that the SWA, or Scheduler Work Area, is defined to go above the line. The SWA is an area of storage that contains Control Blocks which MVS uses to store information from JCL, such as data about DD cards. It also contains information about any dynamically created file, so the number of open DB2 datasets is a big factor on how large it is. If the SWA is stored below the line, then around 1.2 Kb per dataset is required below the line, and so DSMAX naturally needs to be smaller. According to the DB2 Installation Guide (DB2 V8), if the SWA remains below the line a DB2 subsystem should not have DSMAX set higher than 5,000.



The remaining Storage used by the DBM1 address space is stored Above the Line in Virtual Storage, in the 2,032Mb between the Line and the Bar in my previous diagram. It's generally this area where DB2 sites have had problems in recent years.

The Storage Map on this chart can be produced by examining IFCID 225 (Storage Summary data, available through Statistics Trace Class 6) and IFCID 217 (Storage Detail, available through a Global Trace Class 10). Most of the performance monitors on the market today can also provide the figures. It shows how the Above the Line storage is broken down by DB2, and is discussed in much more detail by the papers and presentations referred to earlier.

I have highlighted in Red those Storage areas which have been moved Above the Bar, into the region provided by 64 bit addressing, by DB2 V8 so you can immediately see that many of the big hitters have now been removed which should help relieve Storage Constraint problems. Having said that, other areas may now be much larger, for example if you have 1200 bytes per open dataset and 100,000 datasets this area could now be 120Mb. The Blue highlighted area, Hiperpool Control Blocks, are no longer required as Hiperpools are not supported by DB2 V8 since Bufferpools can be so large in Above the Bar Storage.



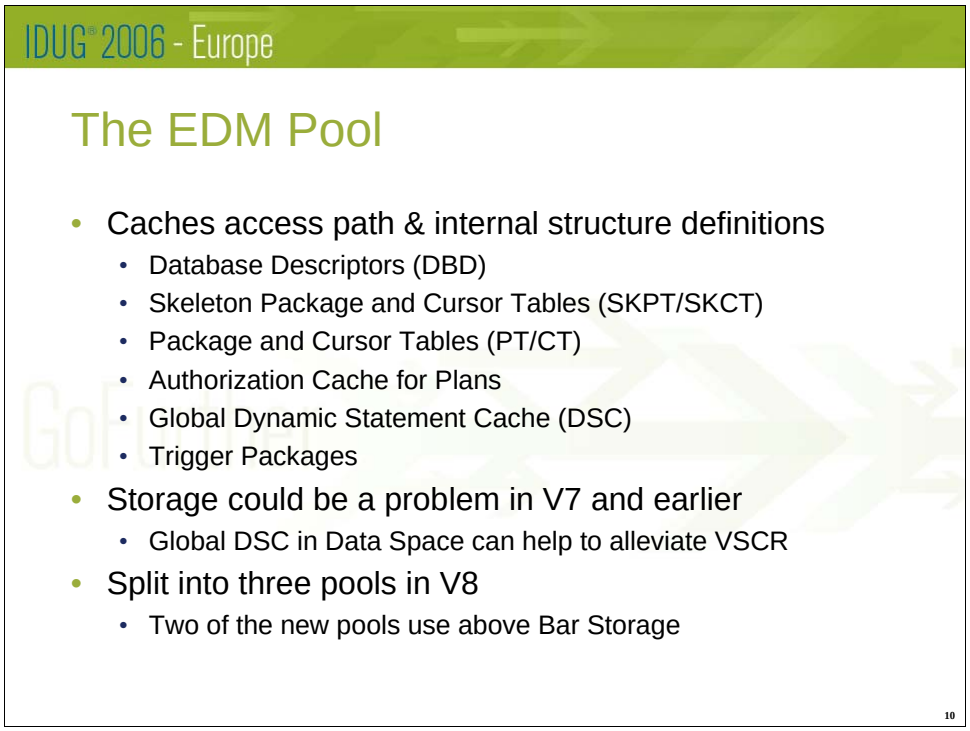
These are the 4 major Storage Pools used in DB2 for z/OS. Virtual Bufferpools (to give them their more correct name) are used as a Cache for frequently used Data and Index pages for your application data and are generally the largest consumer of Storage in most DB2 subsystems. The remaining Pools will be the subject of more detailed discussion during the rest of this presentation.

The EDM Pool is used mainly as a Cache for internal DB2 data as opposed to Application data. This includes information from the Directory (DBDs which are DB2's internal representation of your table structures) and Access Path data (stored in Cursor and Package Tables). If there is a lack of space here you will start seeing additional I/O and you may get fewer concurrent threads being processed.

Next comes the RID Pool, which is a used to manipulate Row Identifiers (RIDS) during SQL execution, for example during List Prefetch and Multi-Index Access operations. If this isn't used properly then you will get a lot more I/O as more optimal access paths revert to Tablespace Scans.

Finally we have the SORT Pool, which provides a Sortwork area for DB2's internal Sorting process used to satisfy ORDER BY requests and join operations. If not enough space is available here then Sorts will overflow into the Workfile database, again generating much more I/O.

You can see a common thread developing here – these Pools all reduce I/O and improve performance, although in very different ways.



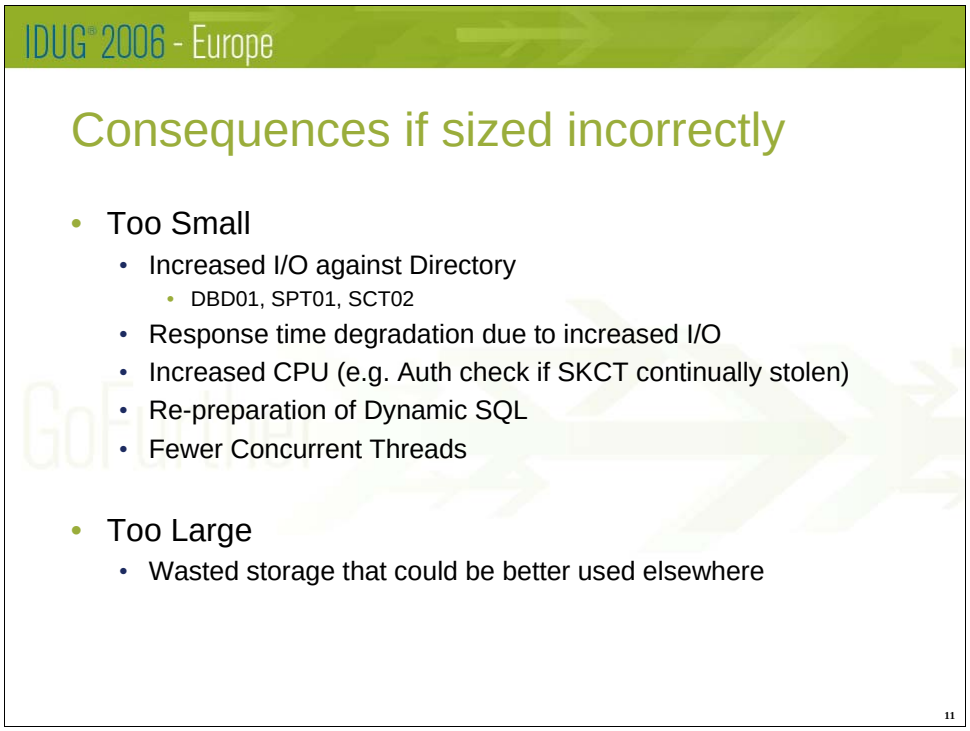
The first of the Pools we will discuss is the EDM Pool which is effectively a Cache for the internal data DB2 needs while executing SQL.

Database Descriptors are loaded from the Directory (DSNDB01.DBD01) and contain an internal representation of the structure of your database. Every Database that contains an object being processed by an SQL statement must have it's DBD loaded and available in the EDM Pool.

The remaining data stored all relates to Access Paths in one way or another, with the possible exception of the Plan Authorization Cache, which is used to prevent unnecessary access to the Authorization Tables in the Catalog when executing plans. Trigger packages are internal DB2 Packages which are dynamically bound by DB2 when you execute a Create Trigger statement. They are executed whenever a trigger is activated.

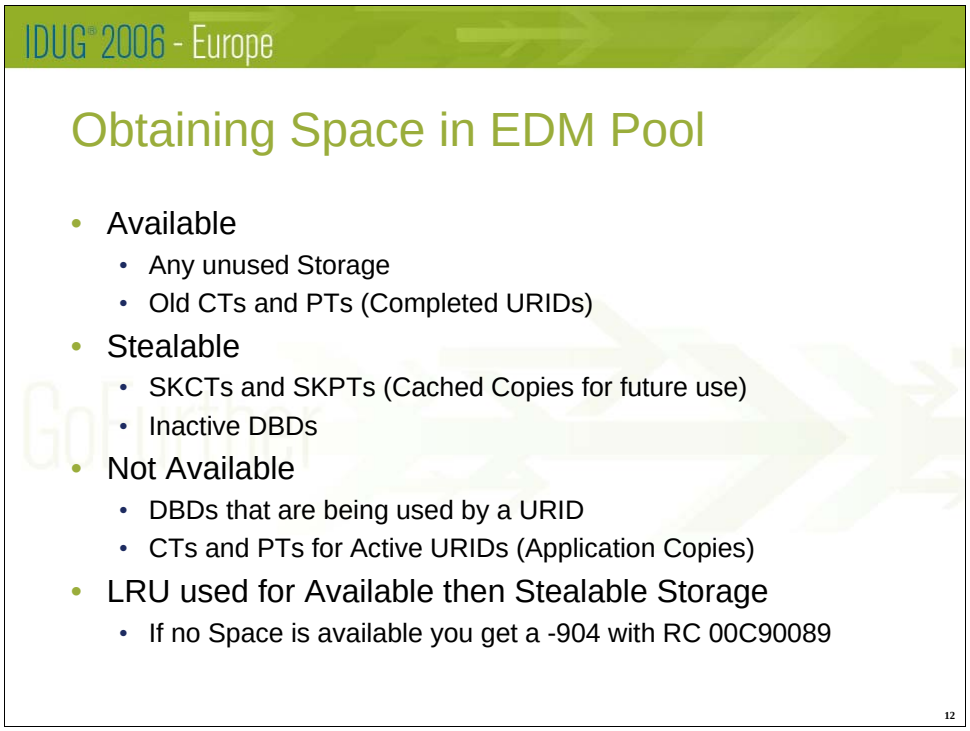
One element of the EDM Pool which may need to be quite large if you're using a lot of Dynamic SQL is the Global Dynamic Statement Cache. If this isn't big enough, you may continually need to re-optimize the same SQL statement which is obviously a performance issue. Using DB2 V7, relief was available by defining that the Global DSC should be in a Data Space, using the EDMDSPAC DSNZPARM parameter.

In DB2 V8, the EDM Pool has been split into 3 distinct Pools, 2 of which use Above the Bar Storage, and so the DSC Data Space is no longer required or indeed supported. A later slide provides more details on the changes introduced in DB2 V8.



There are a number of potentially quite serious consequences If the EDM Pool is not large enough. You will see increased I/O against objects in the DB2 Directory, particularly DBD01, SPT01 and SCT02, as DBD and Access Path information have to be reloaded into the Pool which will in turn lead to slower response times. There may also be increased CPU due to the need to re-prepare Dynamic SQL statements (which will also lead to further response time degradation) as well as any additional authorization checks which will be necessary if the Skeleton Cursor Tables have been stolen from the EDM Pool. If the Pool is more seriously undersized, you may also get fewer concurrent threads if there is not enough space to retain all the various DBDs, Cursor and Package Tables which must be available during thread execution.

The direct consequences of the EDM Pool being too large are not as serious. The main drawback, particularly for DB2 V7 customers, is that some of the limited storage available above the line is being wasted while it could be better used elsewhere to improve performance.

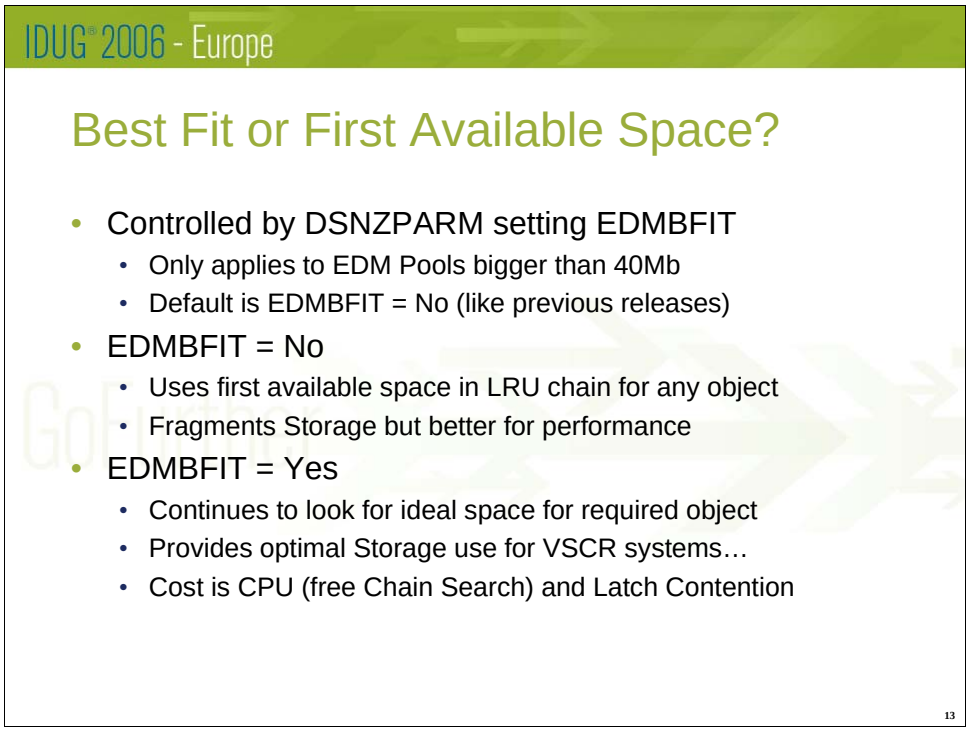


DB2 manages space in the EDM Pool by splitting the available space into three categories. The first is Available Storage, which includes any space that was allocated to the Pool when DB2 was started but which has not yet been used, as well as Cursor Tables and Package Tables from URIDs that have completed since these will never need to be accessed again. Available Storage will always be used first if any space is required in the Pool as the space is not being used and so there is no impact on any other DB2 service from using it.

The second category is Stealable Storage, which consists of data which could be used again but which can be loaded again from the Directory if necessary. This type of data includes Skeleton Cursor and Skeleton Package Tables (SKPT and SKCT), as these are effectively templates which are copied to create the application specific CTs and PTs which an application requires during execution time. Another example of Steablable Storage are any DBDs which are not being used by any SQL statement executing at the time.

Finally there is Unavailable storage, which is being used by active SQL statements and as a result cannot be released. Typical examples include the DBDs for databases whose objects are being processed by executing SQL and the individual application copies of the Cursor and Package Tables.

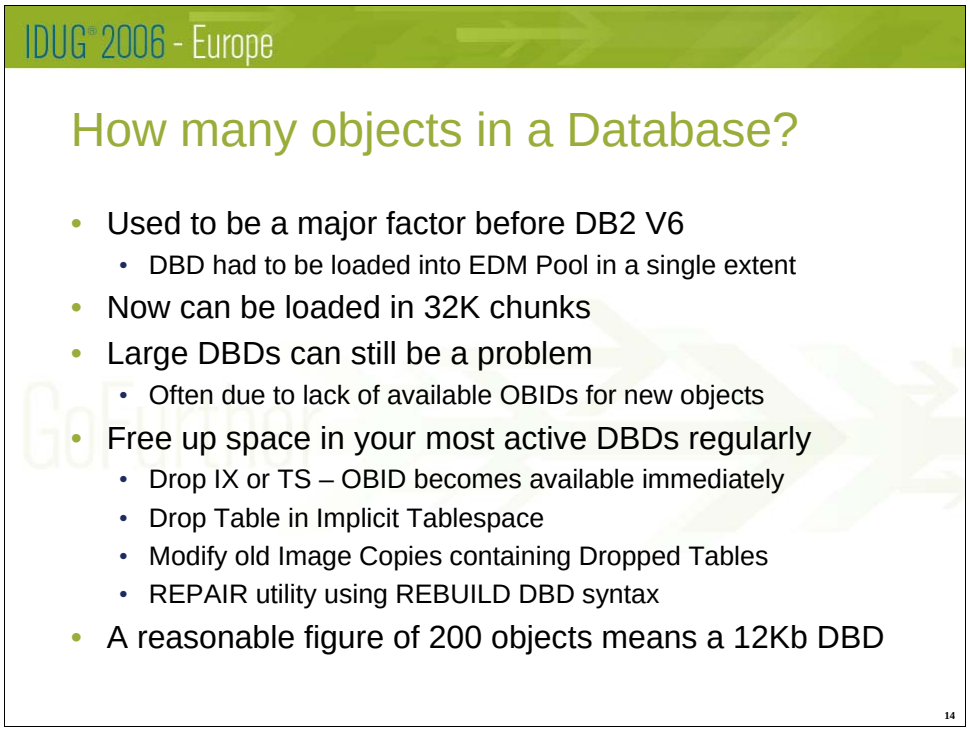
DB2 maintains an LRU (Least Recently Used) queue which is searched when an object if an object cannot be allocated in Available Storage. This ensures that any objects which are stolen will be those which haven't been used for the longest time. If no Available or Stealable Storage exists in the EDM Pool when you application needs it the you will see an SQLCODE -904 Unavailable Resource with a reason code of 00C90089.



A relatively new DSNZPARM parameter (introduced by DB2 V7) called EDMBFIT is used by DB2 to control how it looks for available space when searching the LRU queue described in the last foil. It only applies to EDM Pools which are bigger than 40Mb but in my experience that covers most of them.

The default setting is EDMBFIT = No to make it compatible with older DB2 releases. In this case, DB2 uses the first available space it finds which is large enough to hold the object it's trying to store. This is the best setting for performance, but at the possible cost of fragmenting the EDM Pool.

The new setting, EDMBFIT = Yes makes DB2 continue to search the LR chain and to use the ideal (or best fit) space for the object. This prevents the EDM Pool getting fragmented, and so you can probably make it smaller saving some Storage for highly Storage Constrained systems. The downside of using it is worse performance (it takes more CPU to look though the complete LRU chain) and increased Latch Contention.

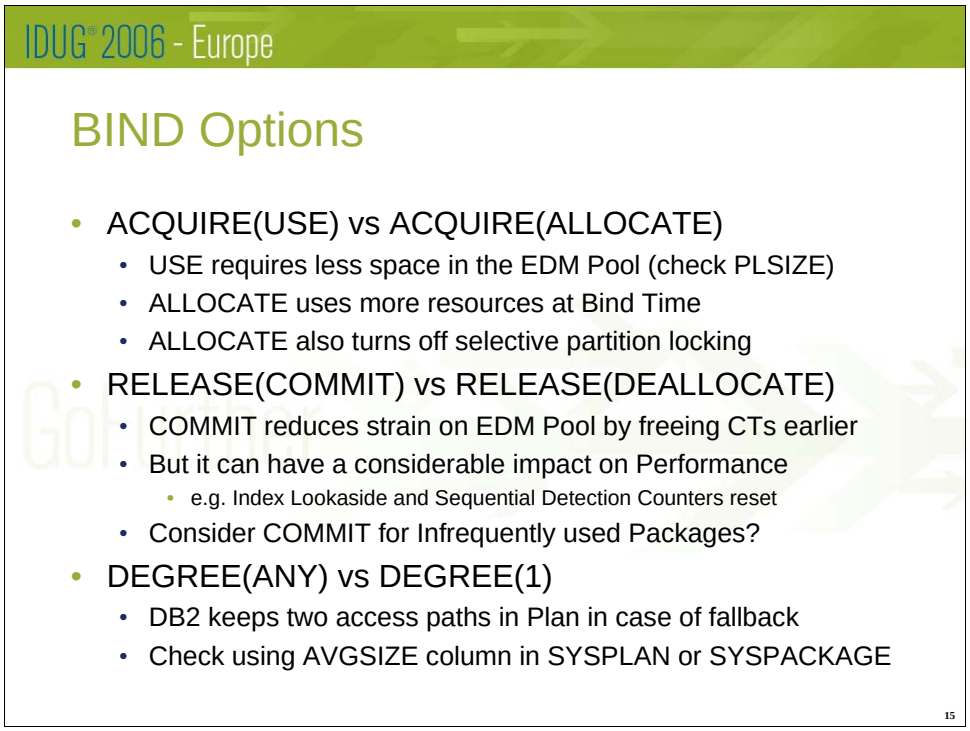


One factor which used to cause a lot of problems was DBD size, which largely depends on how many objects you have in the database. Before DB2 V6 DBDs had to be stored in the EDM Pool in a single extent, and it was quite common that EDM Pool fragmentation made it impossible to find an extent large enough to load large DBDs, which could mean closing down the subsystem. This was quite a common problem for databases such as the QMF Save database, where users can use the 'Save Data' command after running a query which creates a DB2 table in a named database. Even if the table is subsequently dropped the Table OBID may not be reusable, and as a result the DBD continued growing until it no longer loaded into the EDM Pool.

This is not such a problem today, since any DBD created from V6 onwards can be split into 32K chunks when being loaded into the EDM Pool. However, it's worth remembering that a DBD created before V6 but which has not been changed will still need to be loaded in a single extent, although hopefully this won't be very common (V5 went out of support at the end of 2002).

Databases with a large number of objects can still cause issues due to a lack of available OBIDs. The methods available to free up OBIDs in a database and to allow them to be reused are listed in the foil – please refer to the Administration Guide or the Utility Guide and Reference for more details. Another potential issue is that DB2 commands operate at the Database Level and so can be less granular if too many objects are in one database.

To give you an idea of the space required, a typical database containing say 200 Objects (Tablespaces, Tables and Indexes) will have a DBD of around 12Kb, although I've seen numerous instances where DBD sizes have gone up to something closer to a Megabyte in the past.

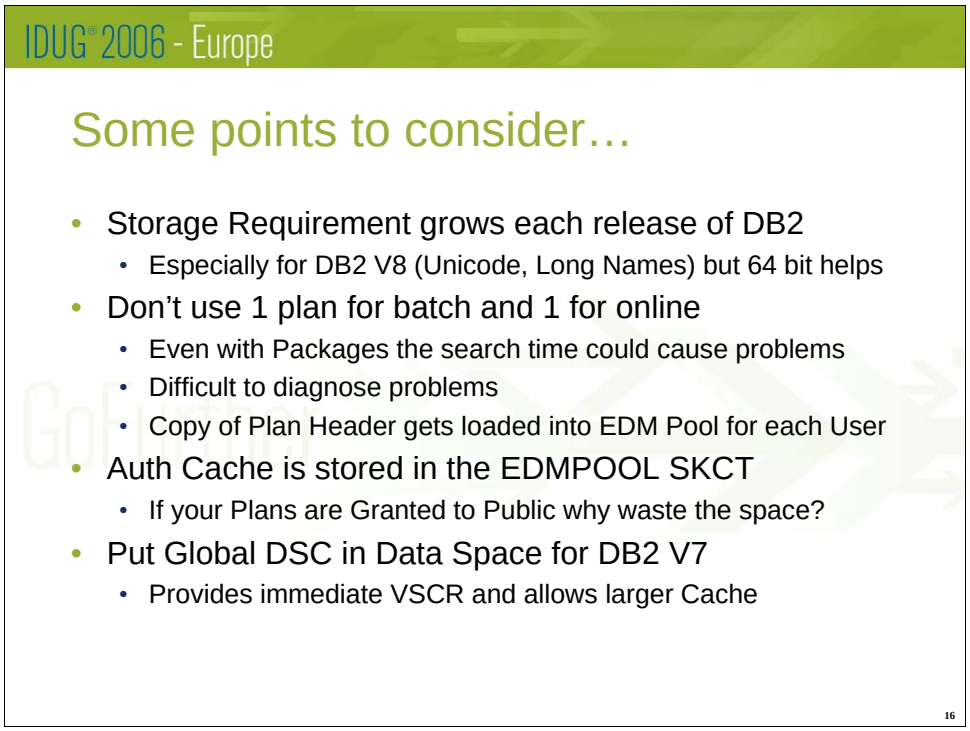


Deciding what BIND options to use requires something of a balancing act between providing best performance, improving concurrency and minimizing EDM Pool usage. Often the options to provide the best in one area may cause problems in another, so it probably comes down to a value judgment at your own site as to what is the most important. The benefits and drawbacks are explained very well in the DB2 Application Programming and SQL Reference Manual.

ACQUIRE(USE) creates a smaller plan and uses less system resources at Bind time. Another downside of ACQUIRE(ALLOCATE) is that it turns off selective partition locking – all partitions are locked when plan execution starts. However, as locks for all objects which may be accessed are obtained earlier it using it can reduce deadlocks, although this can also be minimized by good programming techniques.

A combination of USE/DEALLOCATE is the best for overall system resource usage and is often used in Data Sharing to reduce Global Lock Contention. The alternative is USE/COMMIT which is the default and provides the greatest concurrency, but at a CPU cost. In the context of this presentation this combination releases EDM Pool space used by Cursor Tables earlier, but using it resets counters such as those for Index Lookaside and Sequential Detection.

If you use a DEGREE of ANY you should be aware that DB2 creates 2 access paths in the Plan, the Parallel one and another using DEGREE(1) to which it can fallback if required. You can check the impact of this by looking at the AVGSIZE column is SYSPLAN or SYSPACKAGE.

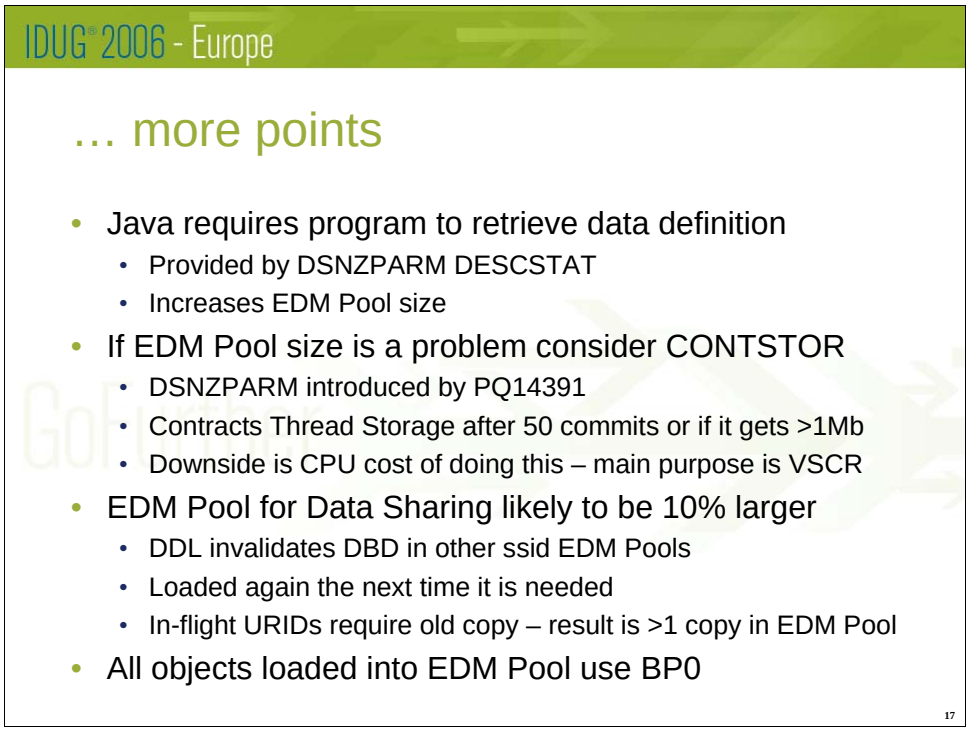


Every new version of DB2 requires more space in the EDM Pool. This is particularly true for DB2 V8 when the Catalog and Directory objects are likely to be bigger due to Unicode and Long Names, although the impact has been mitigated to a large extent by the introduction of 64 bit addressing so we're no longer restricted to the 2Gb line.

I still occasionally see customers who use only 2 plans for their whole system – 1 for Online and the other for Batch. This is usually a bad idea. Package search times can get longer and it's harder to diagnose problems as the Plan name is the only thing reported in many instances. Another problem is that the full Plan header gets loaded into the EDM Pool for each user of the Plan since it's a part of the Cursor Table, so if the Plan Header is large then the amount of space required can increase dramatically.

Something else which frequently wastes EDM Pool space is using a Plan Authorization Cache if execution privileges are granted to Public. The Cache is copied into the EDM Pool as part of the SKCT so you're just wasting space. It is defined using the BIND parameter CACHESIZE. In recent releases of DB2 this defaults to the value of the AUTHACH DSNZPARM. There's some confusion in the manuals here – the V8 Command Reference says this defaults to 0 but the Installation Guide quotes 3Kb so it's worth checking your setting. In older DB2 versions there was a direct default of 1Kb so if a plan has only been rebound since then you may still be using this.

If you're using DB2 V7 and dynamic SQL then it's almost imperative that you put the Global Dynamic Statement Cache in a Data Space to relieve the pressure on the EDM Pool space below the Line. You do this by defining the EDMDSPAC DSNZPARM (assuming of course that you're keeping Dynamic Statements past Commit and that MAXKEEPD is non-zero).

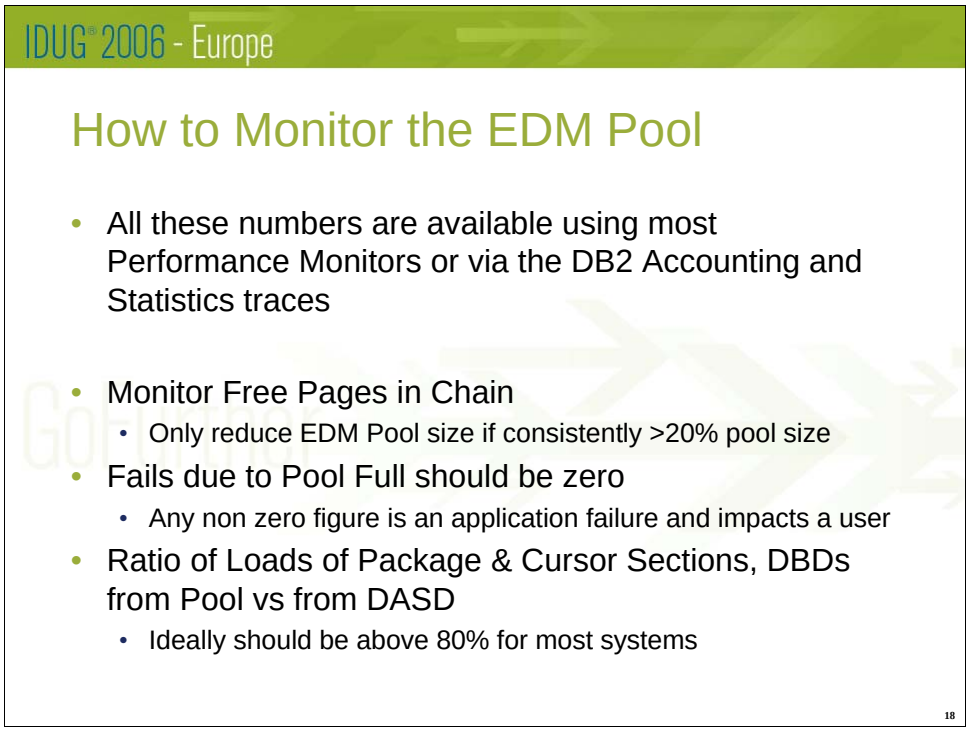


More applications are now being written in Java, which requires a program to retrieve the definition of the table being queries from the DBMS. In DB2, this is achieved for Static SQL (such as SQLJ) by setting the DSNZPARM DESCSTAT to Yes which is the default in V8, whereas it was NO in V7. What this option does is to build a DESCRIBE SQLDA for Static SQL statements during the Bind process. This naturally makes the Plan larger, and increases the load of the EDM Pool.

Another V7 DSNZPARM to consider, particularly for sites suffering from Virtual Storage problems, is CONSTOR which was introduced by PQ14391. When this is set, DB2 contracts the Storage used by a thread in the EDM Pool after every 50 Commit statements or if the total thread storage gets over 1Mb. This has the obvious benefit of releasing EDM Pool space for use by other applications earlier, but there is a CPU cost associated with this, so generally this parameter is only considered for storage constrained subsystems.

When you are using Data Sharing, executing DDL against a database causes any copies of the DBD to be invalidated in the EDM Pools of other members of the Group. The next time an application uses an object in the database, the new and updated DBD is reloaded from the Directory. However, any in-flight units of recovery will continue to use the old copy of the DBD until they Commit, and so there will be a short period when multiple copies of a DBD could be loaded in an EDM Pool at the same time. This results in the EDM Pool for a Data Sharing system being larger, perhaps up to 10%, although most sites will probably find this number rather smaller unless they execute a lot of DDL statements.

Finally, remember that all Objects loaded into the DBD will use BP0, so tuning this will improve EDM Pool performance.

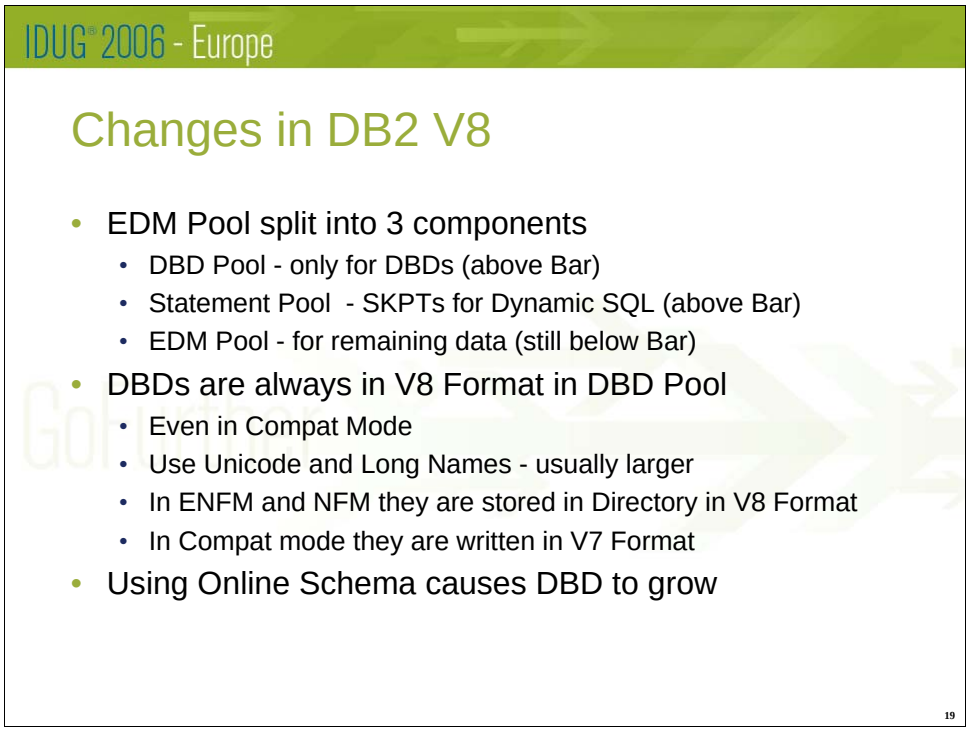


Monitoring the EDM Pool is fairly straightforward as the key statistics are provided by the Performance and Accounting traces which almost all sites run continuously. Most Commercially available Performance monitors also provide these numbers both in Batch and Online reports.

The first statistic to watch is the number of free pages in the Chain, which is the LRU chain referred to earlier. You should compare this figure to the number of Pages in the entire EDM Pool (a Page is 4K). The number of Free pages can get quite large at times, but you should only consider reducing the size of your EDM Pool is the number of free pages is consistently above 20% of the Pool Size.

The next figure to watch is the number of failures due to the EDM Pool being Full. If this is non zero it requires investigation since this is a situation which will have directly impacted an application and caused a -904 resource unavailable failure.

Finally, check the ratio of the number of times a Cursor Section, Package Section or DBD was loaded from the EDM Pool rather than from DASD. A hit ratio of over 80% indicates that the EDM Pool is working well. If the number drops below 80% then unused DBDs, SKCTs and SKPTs may be being stolen more frequently than is ideal which means the Pool may be too small.



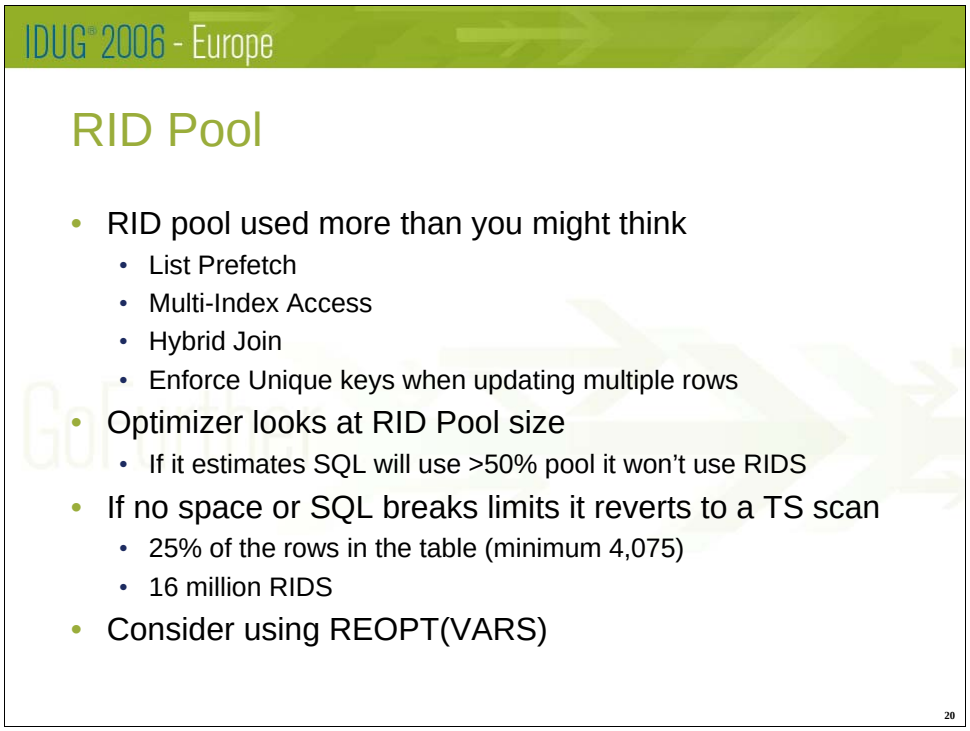
The changes introduced with DB2 Version 8 split the data held in the EDM Pool into three components. The first is called the DBD Pool and contains purely DBDs, which are stored above the Bar, which should reduce how often unused DBDs get stolen from the EDM Pool.

The second new Pool is the Statement Pool, which is also stored above the Bar. It contains Skeleton Package Tables (SKPTs) for dynamic SQL, and performs the same function as the the old Global Dynamic Statement Cache which was stored in a Data Space in most V7 subsystems where dynamic access paths were being saved. Since this is now above the Bar, the old Data Space is no longer required.

The remaining data is still stored in the EDM Pool, which remains below the Bar as before. This data stored here includes thread specific data (Cursor Tables and Package Tables), all SKCTs and any SKPTs for Static SQL.

When using V8, DBDs are always stored in the DBD Pool in V8 format, which uses Unicode and Long Names. As a consequence, DBDs in V8 will probably be larger than in V7 and the EDM DBD Pool will therefore need to be large than it's V7 equivalent. When you're using either ENFM or NFM mode, the DBDs are also stored in the Directory in V8 format, but in Compatibility mode they are stored in the old V7 format, with the consequence that a conversion is required every time a DBD is loaded into the DBD Pool when using V8 Compatibility Mode.

You also need to be aware that using Online Schema changes in V8 causes the DBD to grow, and all versions of an object must be stored in the DBD in case DB2 comes across one of the older version rows when processing an object.

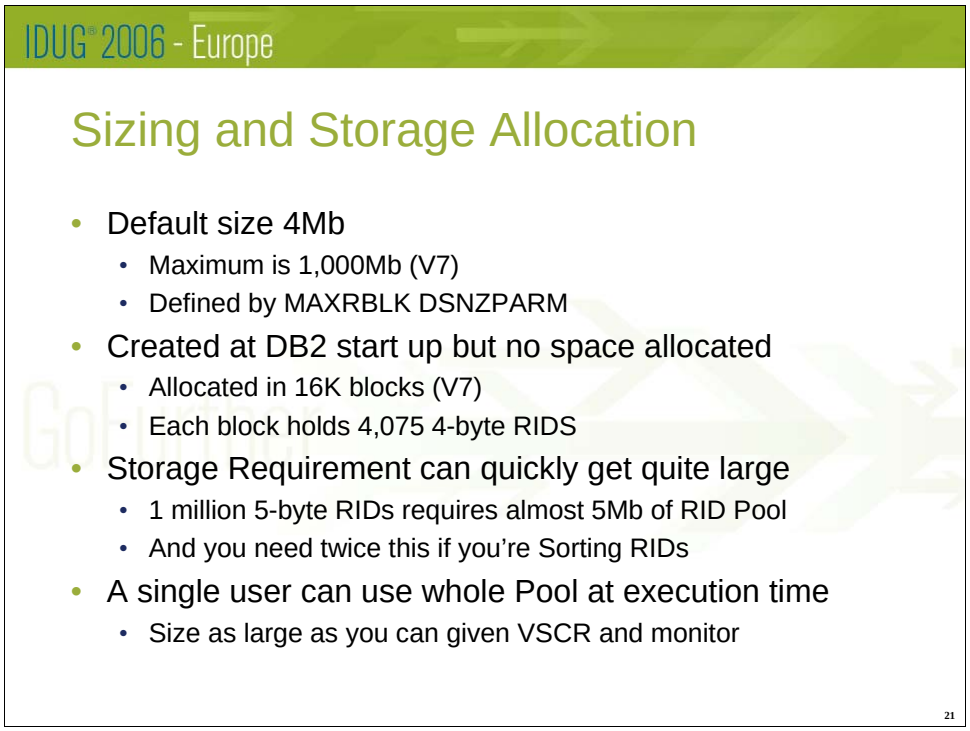


The RID Pool is used by DB2 during SQL execution to improve performance by manipulating Row Identifiers (RIDs), which are usually obtained from indexes. For example, you might be able to avoid accessing a row completely by eliminating it using a second index (multi-index access) or prevent re-reading pages by ensuring data is accessed sequentially (List Prefetch). There are a number of other instances where DB2 uses the RID Pool at execution time, and it's used more frequently than many people realise.

When an SQL statement is Prepared (either at Bind time or dynamically), DB2 looks at the RID size of the subsystem where the Bind occurs. If it estimates that the statement would use more than half the RID Pool it won't consider using a RID based access path.

At execution time, the access path being used dynamically reverts to a tablespace scan if the RID Pool is full or if certain limits are exceeded, in which case any work done up to that point may be wasted. For DB2 V7, a RID list cannot exceed 16 million entries, and the RID list cannot represent more than 25% of the table, although a minimum of 4,075 RIDS are always allowed regardless of table size. It's also worth noting that the number of rows this last rule is based on is what the Optimizer believes are in the table based on the table statistics at Bind time and not either the actual or statistics figure at execution time.

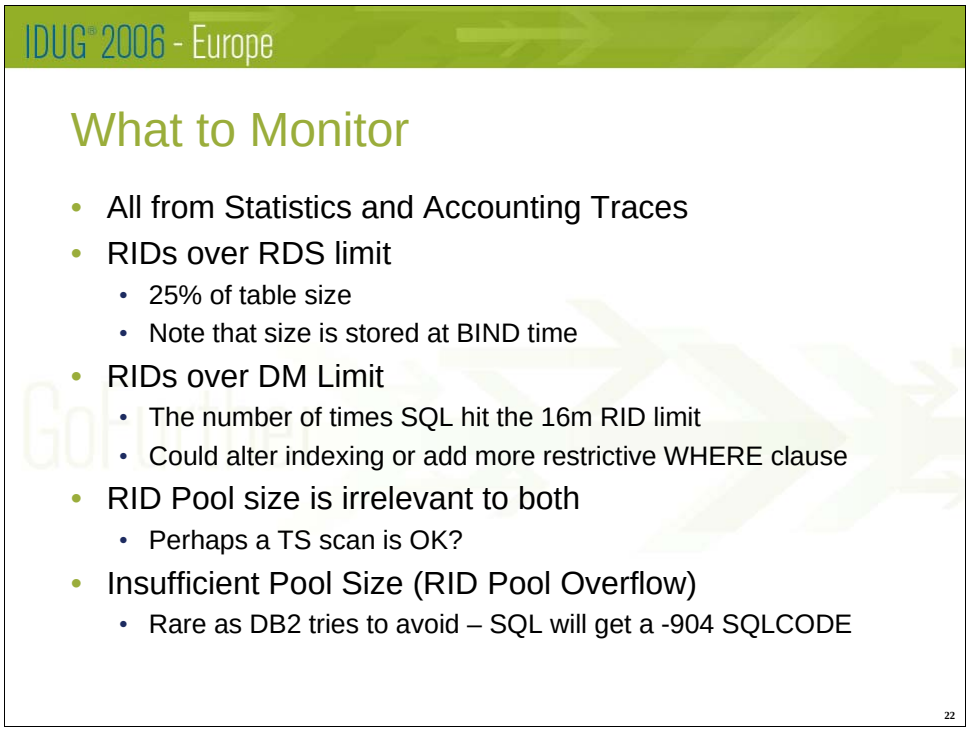
If an access path may benefit from RID access but only depending on the host variables being used, you can consider using REOPT(VARS) so that the access path is optimized based on the access path statistics at execution time. You could even change the program to call a separate module to execute that piece of SQL and make it a separate Package so that only specific SQL statements will use this feature.



The RID Pool has a default size of 4Mb, with a maximum size in DB2 V7 of 1,000Mb although in reality nobody could get close to this figure due to Storage Constraints. The Pool is created when DB2 starts up but no space is allocated at that time. Instead space is allocated in 16K blocks as an when it's required. A 16K block can hold 4,075 4-byte RIDs, which is where the minimum number allowed for a table referred to in the previous foil comes from. Space is allocated until the RID Pool reaches the maximum size allowed, which is set in the MAXRBLK DSNZPARM parameter.

The amount of RID storage used by a single SQL statement can easily get very large. A RID list of 1 million entries could be 5 Mb if the rows are defined in a Large tablespace, and a 16 million entry RID list could consume 80 Megabytes in the RID Pool if it was allowed to do so. If you are Sorting RIDs you will need twice this amount of space to handle the sort results.

Although DB2 restricts projected use of the RID Pool at Bind time based on the percentage it thinks you might use, at execution time there are no restrictions assuming the 16 million RIDS or Table size limits are not reached. A single user could use the whole Pool. As a result, it's best to make the RID Pool as large as you can given Virtual Storage Constraints and to monitor failures as explained in the next page.



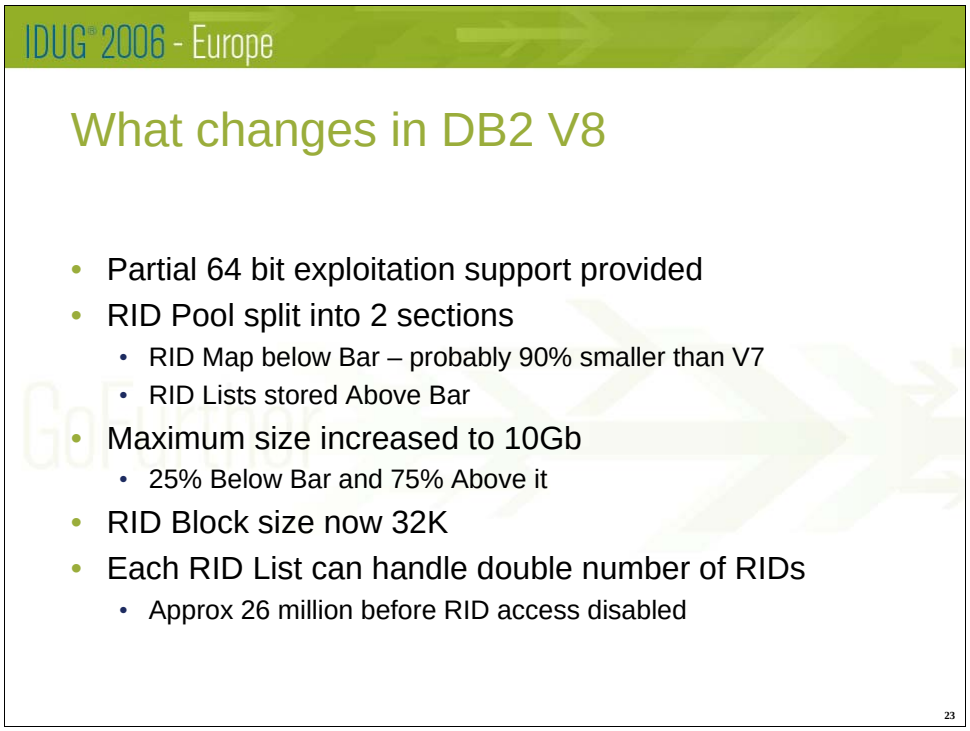
Monitoring the RID Pool is fairly straightforward as the figures are provided by the Standard DB2 Statistics and Accounting Trace records which most sites already collect. The first two mentioned will probably be non-zero, but you should make sure they don't get out of hand and also that it's not just a few statements causing most of the failures. In both cases, increasing the RID Pool will make no difference to the number of failures as they occur due to execution time problems with individual SQL statements.

The first statistic to watch is 'RIDs over RDS Limit', which means that the number of RIDs processed reached 25% of the size of the table. As was mentioned in an earlier slide, this size limit is determined at BIND time, so if a table has increased in size, this issue could potentially be resolved by Rebinding the Plan or Package after checking or correcting the Catalog Statistics.

The Second statistic to watch is 'RIDs over DM Limit' which means that DB2 had extracted over 16 million RIDs which is the Data Manager limit. The only way to change this is to reduce the number of RIDs being processed, either by adding a more restrictive where clause (providing the columns are indexed) or by changing the indexing on the table, bearing in mind of course that other SQL statements will be using these as well.

One thing to bear in mind is that a Tablespace scan may well be the best access path. However you don't want to start using one access method and then throw away all the work performed to date. It may be better to start with a Tablespace Scan in these circumstances.

The final statistic to monitor is failures due to Insufficient Pool Size, or RID Pool Overflow. This should happen very rarely as DB2 tries to avoid it. If it does occur, the SQL will fail with a -904 Resource Unavailable message.



DB2 V8 introduced partial 64 bit exploitation support for the RID Pool. The Pool has been split into two sections, with the larger capable of being stored above the Bar. The section below the Bar now stores a RID Map, which is almost an index for the main RID list. The RID Lists themselves are stored in a new section stored above the Bar. First indications are that these changes reduce the space requirements below the Bar by around 90%.

As a result of these changes, the maximum size for the complete RID Pool has been increased to 10Gb. It is still defined in the DSNZPARM using a single parameter, MAXRBLK as before. The DB2 Installation Guide states that 25% of the space will be defined below the Bar and 75% is above it, although it doesn't explain what happens if you attempt to define a huge RID Pool of over 8Gb which would be unable to use this breakdown.

The RID Block size is now 32K, so storage is allocated in 32K chunks. A RID Map can hold details of over 4,000 RID List entries, while each RID List entry can itself hold 6,400 RIDs. This means that a full RID List under V8 can now hold over 26 million entries before RID access is disabled (the Data Manager limit mentioned in the monitoring section).

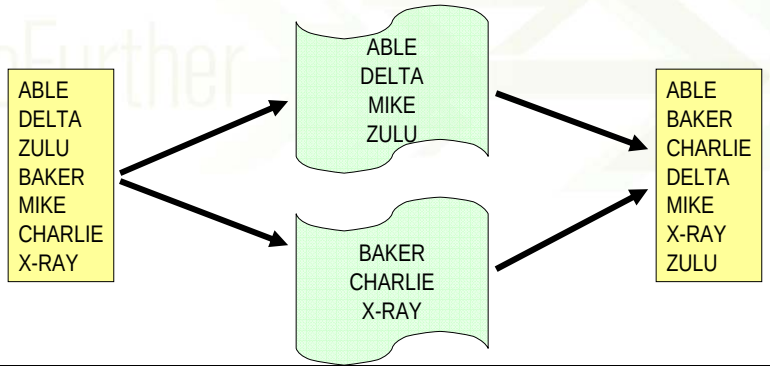
The SORT Pool provides working storage for the internal sort performed by DB2. In many ways you can think of it as equivalent to the Region size used by your own system Sort package such as DFSORT, SYNCSORT or CASORT. There are many other similarities between how the DB2 internal sort method works and external Sort packages. For example DB2 uses the Workfile database (probably DSNDB07 in non-data sharing systems) as an overflow area for sorts which don't fit into the SORT Pool which is somewhat similar to the way packages use SORTWORK files, and if you can perform a sort entirely in storage (i.e. in the SORT Pool) then performance will be much improved.

A big difference between the SORT Pool and the other major DB2 Storage Pools is that there isn't a single parameter to define how much storage should be used. Space is allocated for each thread performing a Sort and not in one large Pool. Each Thread manages it's own SORTPOOL, and just because one thread is suffering it doesn't necessarily follow that the entire system SORT capability has problems, although this could be the case if the rogue thread has filled up the Workfile database objects.

The size of the SORT Pool for each thread is defined by the SRTPOOL DSNZPARM setting. This defaults to 1Mb in V7, but in past releases it used to be 10% of the sum of BP0, BP1, BP2 and BP32K so it's worth checking this has been reset. If a thread performs a Sort, DB2 allocates 240Kb initially and the adds additional space in 16K chunks until SRTPOOL is reached.

How DB2 Sorts data

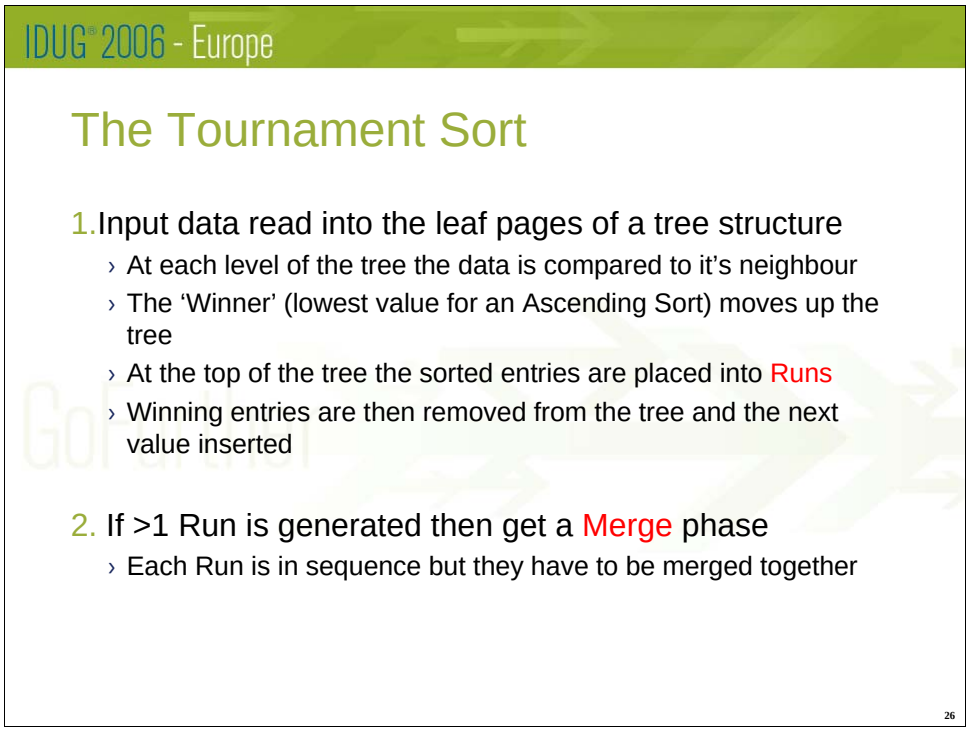
- DB2 uses a Sort Algorithm called a Tournament Sort
  - This has been built into the Hardware Microcode
  - Provides DB2 with highly efficient Sorts



25

DB2 usually uses a Sort Algorithm called a Tournament Sort which is highly efficient for the types of Sorting that DB2 expects to perform. This has been built into the Hardware Microcode of the Processors uses to run DB2 which further increases efficiency and reduces CPU costs.

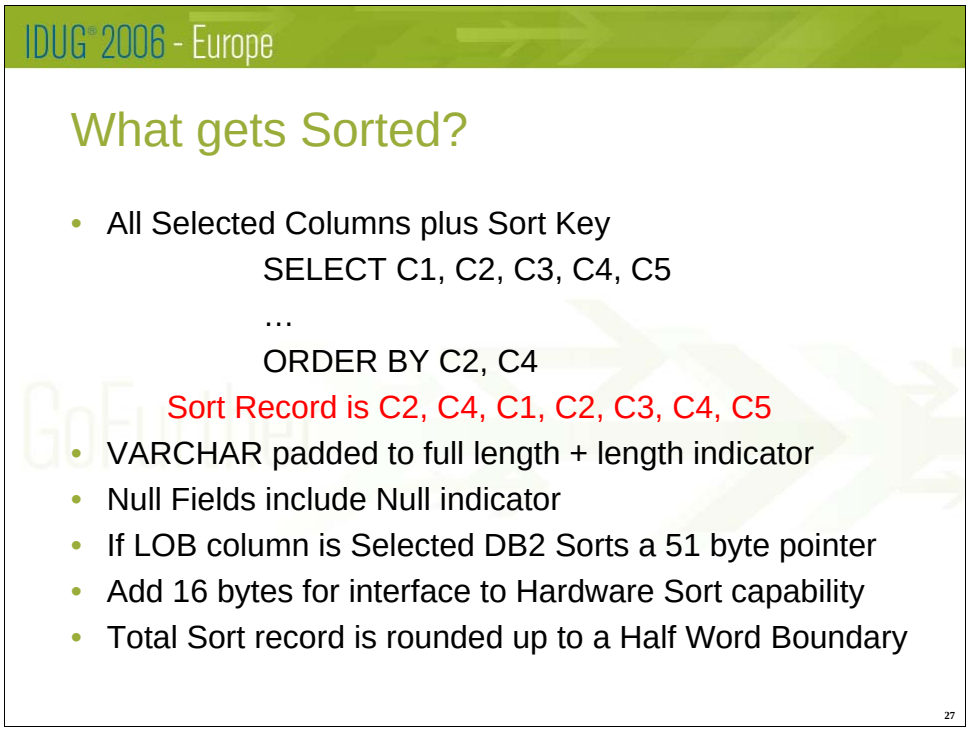
This diagram attempts to provide an overview of how a Tournament Sort works. It is explained in more detail on the next slide.



During the first pass of an ascending sort, the data to be sorted is read into the leaf pages of a tree structure. The first entry is then compared to it's immediate neighbour, and the 'winner' or lowest value entry is moved up to the next level (the 2nd) of the tree (actually it's only an index entry that's moved, but this is beyond the scope of this presentation). The next two leaf pages are then processed, and the same thing happens. We now have two adjacent entries in the 2nd level of the tree, so these are compared and the 'winner' is moved up again to the next level (this time the 3rd). We now know that this is the lowest key of the 4 keys processed thus far.

Continuing the process again, then once 8 keys in total have been read we will have a 'winner' in the 4th level of the tree, and so on in consecutive powers of 2. This process continues until all keys have been read or until the tree has reached 14 levels (32,767 entries if you work it out). The overall winner is then moved into another structure called a 'RUN'. The winning entries and then removed and the next value from the input file inserted and compared to it's neighbour and the process bubbles up again until another 'Winner' has been determined. If this is higher than the previous winner, the data is placed into the same run, otherwise another is started.

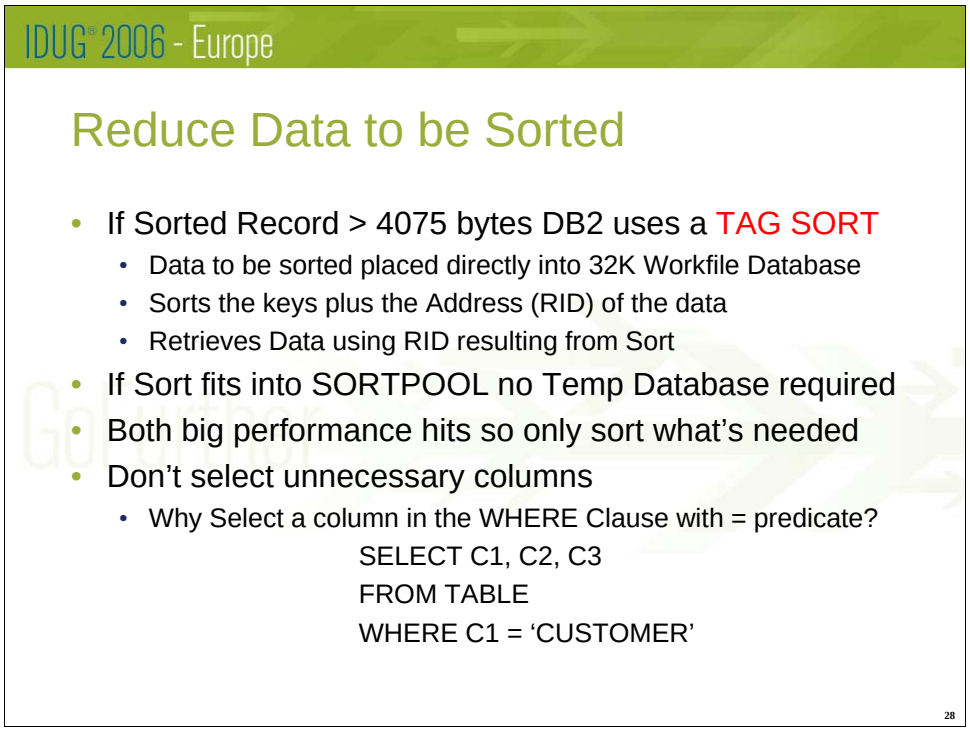
What you end up with as a result of this process is a number of Runs, all of which are sorted into the correct sequence. These runs are then merged in a second phase to produce the final sorted output. If you sort less than 32,767 entries (assuming the tree fits into the SORTPOOL) then you will only require 1 run and there will be no need for a Merge phase, otherwise it will almost always be required.



The next important factor is what gets sorted. The Sorted record comprises all the columns in selected by the SQL, prefixed by the ORER BY columns, so Sorted columns will appear twice in the Sorted record as in this example.

VARCHAR columns are padded to their full length, and the Length Indicator is also included so that the field can be reduced back to it's correct length in the sorted data with no need to refer back to the original result set. For the same reason Nullable columns will include the Null Indicator. If a LOB Column is selected, DB2 naturally cannot consider including the entire LOB, and so a 51 byte pointer is created which is included into the Sorted record. On top of all this data, DB2 adds a 16 byte field which is required to allow it to interface to the Hardware Sort capability, and then the final Sort record is rounded up to a halfword boundary.

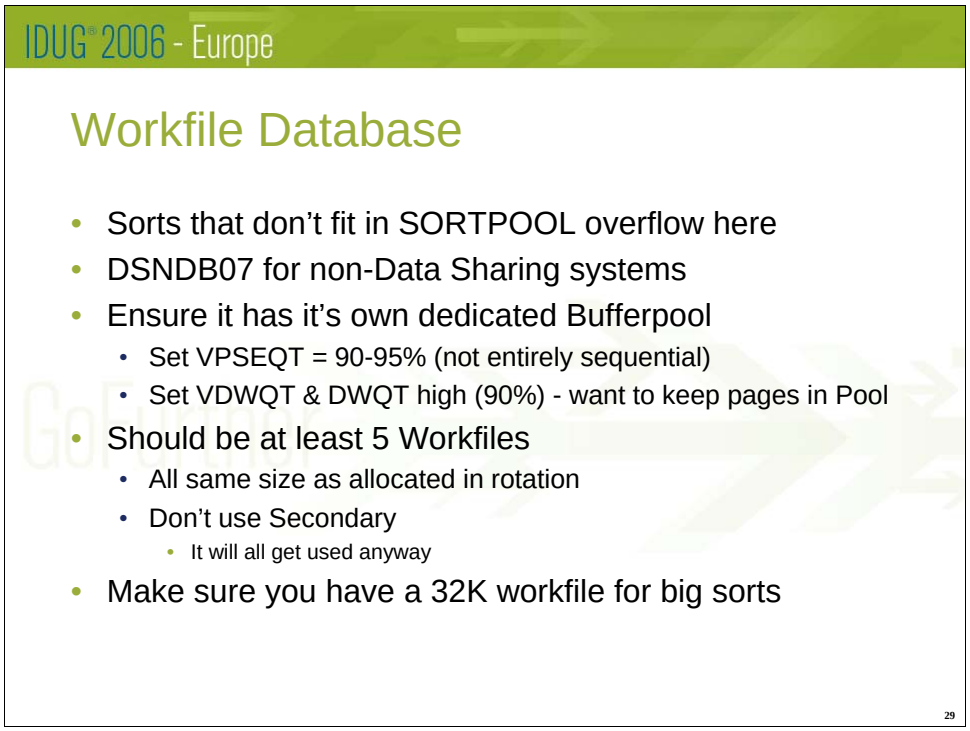
As you can easily see, the more data you select, particularly VARCHAR columns (and especially if they're long) then the longer the sort record needs to be, which leads directly to a larger Sort tree. This tree has to fit into the SORTPOOL – if it doesn't using the maximum of 14 levels then the number of levels is reduced until it does fit. This makes it more likely you will need a Merge phase and that the Sort will overflow into the Workfile database.



If the Sorted Record described on the previous foil is larger than 4,075 bytes then it will no longer for into a DB2 4K page, and so DB2 resorts to an alternative and much less efficient method of Sorting known as a Tag Sort, which is a three phase procedure. First the data to be sorted is placed directly into a 32K page in the Workfile database. DB2 then Sorts the original Sort keys together with the RID of the data that has just been saved. One this is completed, the original data can then be retrieved from the Workfile database in Sorted order using the saved RIDs. This is obviously a highly inefficient Sort compared to a Tournament Sort. It always uses the Workfile database and the 32K Bufferpool which is often undersized compared to others, and should be avoided wherever possible.

Even when using a Tournament Sort there are big benefits if the Sort fits directly into the SORTPOOL since there will be no merge phase and no workfile database access will be required.

Explaining a little of how this process works emphasises again that you should avoid selecting and sorting unnecessary columns wherever possible. Something which I see all the time when I look at program mode is the type of select in the foil, where a column is in a WHERE clause with a '=' predicate but is still listed in the SELECT statement even though you know what value it contains. A little care can make a huge difference in the performance of your application.

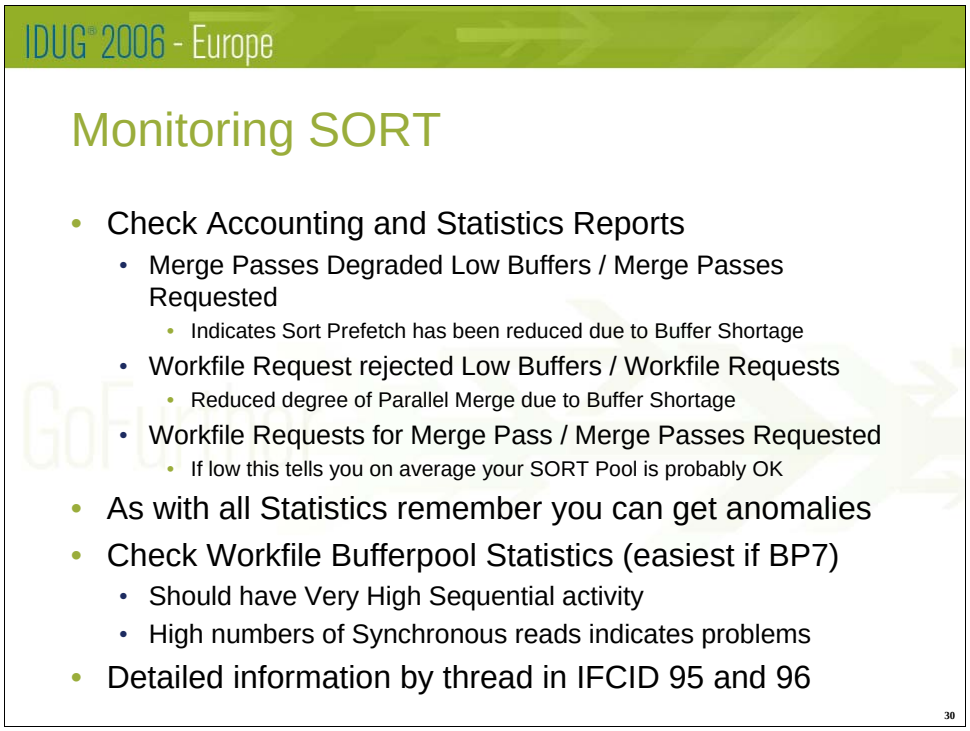


Sorts that don't fit into the SORT Pool overflow into the Workfile Database, which is likely to be DSNDB07 for Non-Data Sharing systems. This may be because the Sort Tree does not fit, or it could be because you are trying to sort more rows that will fit into the tree and the resulting Runs require more space than is available. This is more likely if you Sort more than 32,767 rows as that's the largest possible Sort tree and sorting anything larger than this makes it almost inevitable that you will need to Merge Runs together.

If you do end up needing to use the Workfile Database it then ensuring it's tuned correctly will avoid making a bad situation worse. The first thing to do is to place it into it's own dedicated Bufferpool, since the way it is accessed is likely to be very different from any of you application databases. Although it uses a lot of Sequential Access there is some random access as well, so it's best to set VPSEQT to 90-95%. The biggest difference may be the Write thresholds – you want to keep pages in the Pool so set both VDWQT and DWQT to a high value (say 90%).

You should allocate at least 5 Workfile Tablespaces, and probably more. These should all be the same size as they are used in rotation – there's no usage decision used. There's no point allocating any secondary quantity either as it will always get used. Just decide how large to make them and give this space all to the Primary.

Last, but by no means least, you need to ensure you allocate at least one 32K workfile in this database as some large sorts may be inevitable, especially if you're processing joins with large numbers of tables.



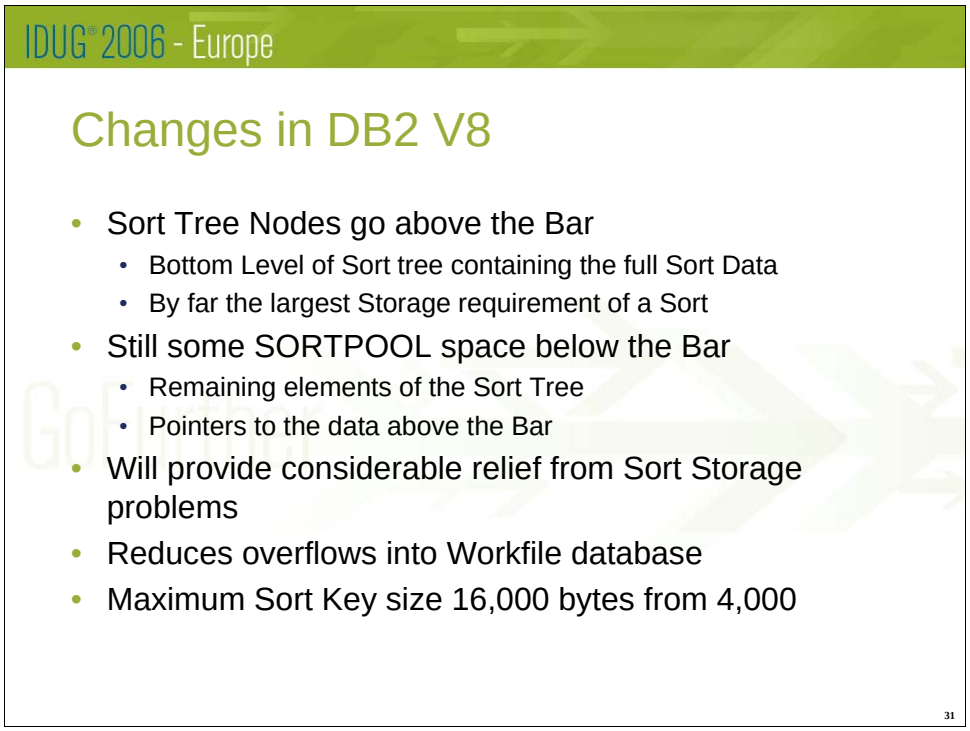
Monitoring Sorts at a high level can be accomplished using Accounting and Statistics data as with the other Pools discussed in this presentation, although you may want to look at ratios rather than at specific numbers as there may always be a few degraded and rejected sorts reported.

Both the number of Merge Passes Degraded due to Low Buffers (which implies you're not getting List Prefetch) and Workfile Requests rejected due to low buffers (meaning you're not performing Merges in Parallel) indicate a possible problem with the Bufferpool associated with the Workfile database, and increasing it may prove beneficial. You should also check the Bufferpool Thresholds.

It's also worth check how many Workfile Requests for Merge Pass you have when compared with Merge Passes Requested. If this number is low it indicates that your SORT Pool is almost certainly fine, as most of your Sorts are fitting into the Pool without needing to use Workfiles.

It's also worth checking your Bufferpool Statistics for the Workfile database, which is easiest of you use BP7 as it's easy to remember. The Pool should have very high Sequential Activity and only a very low number of Synchronous Reads. Also remember to check the 32K Bufferpools, as a lot of activity against these if you're not using them for Applications is likely to be an indicator that Tag sorts are being used.

If you're looking for more detailed data on DB2 SORT, you can get it from IFCID 95 and 96, which can be cut by running a Performance Trace with Class 3 or starting a specific trace requesting these IFCIDs are cut. These provide detailed data on each individual sort such as the number of records sorted, the sort key length, the number of Merge passes and workfiles used and can be useful diagnosing specific problems.



DB2 Version 8 has introduced one very significant change which will assist Sorting. The Sort Tree Nodes, which are the lowest level of the Sort tree discussed earlier and which contain the Full Sort Data Record, can now go above the Bar. This means that a lot more Sorts with larger keys and where many columns have been selected will be able to fit into the SORT Pool without the need to overflow into the Workfile database, which could provide a significant performance improvement. I don't believe the maximum size of the Sort tree (32,767 records) has been increased, but I will try and check this before the session itself.

The other elements of the Sort Tree remain in the existing SORT Pool below the bar, together with some pointers to the storage above the bar which contains the Sort record.

Other than this, the only two changes to SORT are that the maximum key size has been increased from 4,000 bytes to 16,000 (but using this will almost certainly result in a Tag Sort) and the Default Size of the SORT Pool is now 2Mb when it was 1Mb in V7.

Session A08

Getting the most out of your DB2 Storage Pools

Steve Thomas
BMC Software
Steve_Thomas@bmc.com

32