IDUG
Leading the DB2 User Community since 1988

IDUG Db2 Tech Conference NA
Philadelphia, PA | April 29 - May 3, 2018

#IDUGDb2

Table Space Odyssey

Pavel Sustr
IBM Toronto Lab

Session code: D10

May 2nd, 2018 9:20-10:20

Db2 for Linux, UNIX, and Windows

1

Agenda

- Basics: Naming and Layout Conventions
- Common Internals: Pages, Extents
- DMS Internals: Ranges, Stripes, SMPs, EMPs, and Object Table
- REBALANCE vs REDUCE
- Storage Groups
- Caching, Parallelism
- Migration
- Best Practices

Legacy: Stuff That Everyone Knows (Sorry, Eh! ☺)

- DMS
  - **CREATE TABLESPACE … MANAGED BY DATABASE USING …**
  - Data resides in pre-allocated space in file-based (or raw) containers
  - Reclaimable storage (*"new-style"* DMS) or not (*"old-style"* DMS)
- SMS
  - **CREATE TABLESPACE … MANAGED BY SYSTEM USING …**
  - Data resides in a filesystem directory
  - Each directory contains multiple files representing objects (table, indexes, etc…)
  - Allocation happens on demand

3

Use of any of these architectures is not recommended. Automatic is the future.

Automatic Storage

- **Common misconception:** Automatic storage is not "yet another table space type. Instead, it is the attribute of a table space, whose underlying properties are *loosely* based on one of the architectures previously known as DMS or SMS.
- *"Automatic"* means that Db2 will take care of the placement, naming, allocation, and future growth of the table space automatically
- Automatic SMS is only supported with **temporary** table spaces
- Automatic (DMS for data, SMS for temps) is the only table space type supported in IBM Db2 pureScale ® and Db2 with BLU Acceleration

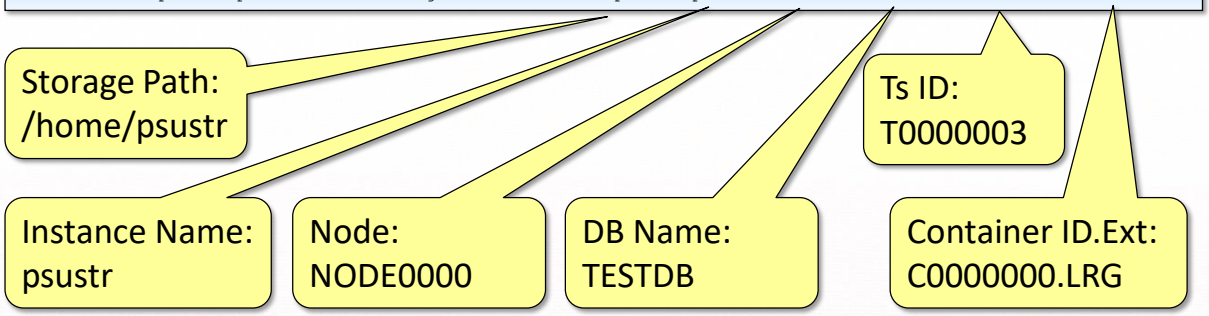
4

Although not identical to the previous architectures, we will be using the terms "automatic DMS" and "automatic SMS" for simplicity.

Automatic DMS Layout Explained

```
$ db2 "create tablespace ts_01a"
DB20000I  The SQL command completed successfully.

$ ls -l /home/psustr/psustr/NODE0000/TESTDB/T0000003/*
-rw------- 1 psustr pdxdb2 5242880 Aug 16 16:17 /home/psustr/psustr/NODE0000/TESTDB/T0000003/C0000000.LRG
```



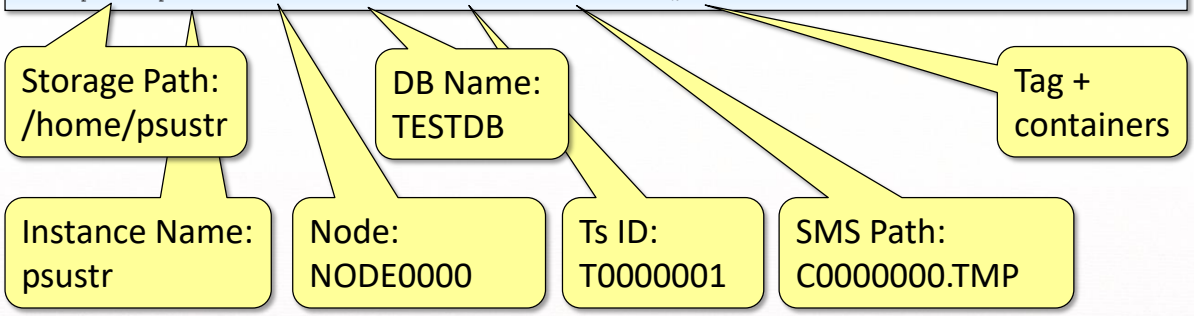
Automatic DMS Naming Convention

- Catalogs:
  <storage path>/<instance>/NODE####/<dbname>/T#######/C#######.**CAT**
- System Temporary Tablespaces:
  <storage path>/<instance>/NODE####/<dbname>/T#######/C#######.**TMP**
- User Temporary Tablespaces:
  <storage path>/<instance>/NODE####/<dbname>/T#######/C#######.**UTM**
- Regular Tablespaces:
  <storage path>/<instance>/NODE####/<dbname>/T#######/C#######.**USR**
- Large Tablespaces:
  <storage path>/<instance>/NODE####/<dbname>/T#######/C#######.**LRG**

Automatic SMS Layout Explained

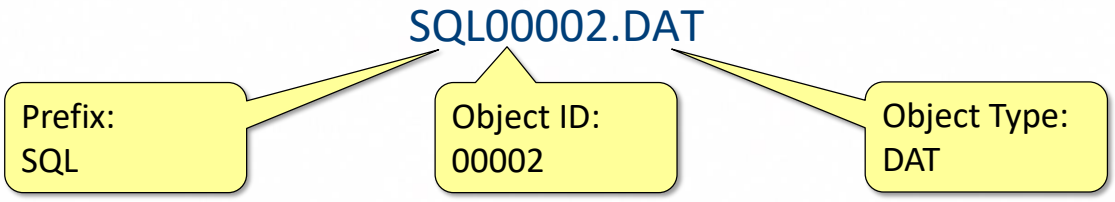
```
$ db2 "create database testdb"
DB20000I  The SQL command completed successfully.

> ls /home/psustr/psustr/NODE0000/TESTDB/T0000001/C0000000.TMP/*
/home/psustr/psustr/NODE0000/TESTDB/T0000001/C0000000.TMP/SQLTAG.NAM
```



Probably the simplest example of how to get an automatic SMS table space. Another example would be creating a user temporary table space. By default, the table space will be automatic SMS.

SMS Naming Convention



- In IBM Db2 pureScale®, *".MEMBERxxxx"* is appended to SMS temps
- Each database object (table, index, …) resides in a separate file
- The file grows and shrinks on demand
- Mapping ***Object ID*** to the object name:

```
SELECT NAME FROM SYSIBM.SYSTABLES WHERE FID = :objectID and TID = :tablespaceID
```

SMS Object Type Extensions

EXT	MEANING
.DAT	Data
.TDA	Temporary Data
.DTR	Reorg Temporary Data
.INX	Index
.TIX	Temporary Index
.IN1	Shadow Index

EXT	MEANING
.LF	Long Field (LF)
.TLF	Temporary LF
.LFR	Reorg Temporary LF
.LB	Large Object (LOB)
.TLB	Temporary LOB
.RLB	Reorg Temporary LOB
.LBA	Lob Allocation (LBA)
.TBA	Temporary LBA
.RBA	Reorg Temporary LBA

EXT	MEANING
.BKM	Block Map (BMP)
.TBM	Temporary BMP
.BMR	Reorg Temporary BMP
.XDA	XML Storage (XDA)
.TXD	Temporary XDA
.RXD	Reorg Temporary XDA
.CDE	Columnar Data Engine
.TCE	Temporary CDE

Table Space Support in Current Releases

	SMS	DMS	AS SMS	AS DMS
Db2 10.5	☑ (Deprecated)	☑ (Deprecated)	☑	☑
Db2 11.1	☑ (Deprecated)	☑ (Deprecated)	☑	☑
IBM Db2 pureScale ®	☒ SQL1419N	☒ SQL1419N	☑	☑

- Deprecated for user permanent, but acceptable for catalog and temporary
- Automatic is the way to go

Unformatted Page (db2dart /DP)

```
$ db2dart test /dp /tsi 3 /ps 128 /np 1 /v n /rptn ts3.page128.rpt
0000  *3000D00F 00000000 0005002A 03000000*     *0..............*
0010  *80000000 04000000 00000000 FFFFFFFF*     *................*
0020  *71BC0500 00000000 881B4A98 53E5E40A*     *q.........J.S...*
0030  *11009500 00000000 D100B600 00000000*     *................*
0040  *00000000 9C0FB00B 640B540B 830AB209*     *........d.T.....*
0050  *E1081008 3F076E06 9D05CC04 FB032A03*     *......n.........*
0060  *59028801 B7000000 00000000 00000000*     *Y...............*
0070  *00000000 00000000 00000000 00000000*     *................*
<…skipping…>
00F0  *00000000 00000000 00000000 00D10001*     *................*
0100  *00C90061 61616161 61616161 61616161*     *...aaaaaaaaaaaa*
0110  *61616161 61612020 20202020 20202020*     *aaaaaa..........*
<…skipping…>
01D0  *0100C900 63636363 63636363 63636363*     *....cccccccccccc*
01E0  *63636363 63632020 20202020 20202020*     *cccccc..........*
```

Header: object/disk page #, object type/ID, ts ID, LSN, checksum…

Payload: the contents depend on the page, in this case we can see unencrypted user data such as ‘aaaaaaaaaa’, ‘ccccccccc’, …

11

In this example we are dumping page 128 from table space 3. The page exists, and it belongs to table T1 containing strings such as ‘aaaaaaaaaaaa’, ‘ccccccccc’, etc… The database is not encrypted.

Formatted Page (db2dart /DD) 1/2

```
$ db2dart test /dd /tsi 3 /tn 'T1' /ps 0 /np 1 /v y /rptn ts3.t1.rpt
            Page Data Offset = 48
            Page Data Length = 4048
                    Page LSN = 000000000005BC71
          Object Page Number = 0
            Pool Page Number = 128
                   Object ID = 4
                 Object Type = Data Object

     Data Page Header: Slot Count = 17 Total Free Space = 149

   Slot 0:      Record Type = Data Object Header Control Record
   Slot 1:      Record Type = Free Space Control Record
   Slot 2:      Record Type = Table Directory Record
   Slot 3:      Record Type = Table Description Record
   Slot 4:      Record Type = Table Data Record (FIXEDVAR) …etc
```

Header: see the previous slide

Payload: in this example the individual page *"slots"* contain metadata pertinent to a regular data page

The same page as on the previous slide, except that it is formatted. The output has been truncated for easier reading. Note that /v (verbose) switch. Without this, user data will NOT be formatted.

Formatted Page (db2dart /DD) 2/2

```
Slot 4:
   Offset Location = 2691   (xA83)
   Record Length = 209   (xD1)
   Record Type = Table Data Record (FIXEDVAR) (PUNC)
   Record Flags = 0
   Fixed part length value = 201

      Column 1:
Fixed offset: 0
      Type is Fixed Length Character String
          61616161 61616161 61616161 20202020    aaaaaaaaaaaa

Slot 5:
   <...skipping...>
          63636363 63636363 63636363 20202020    cccccccccccc
```

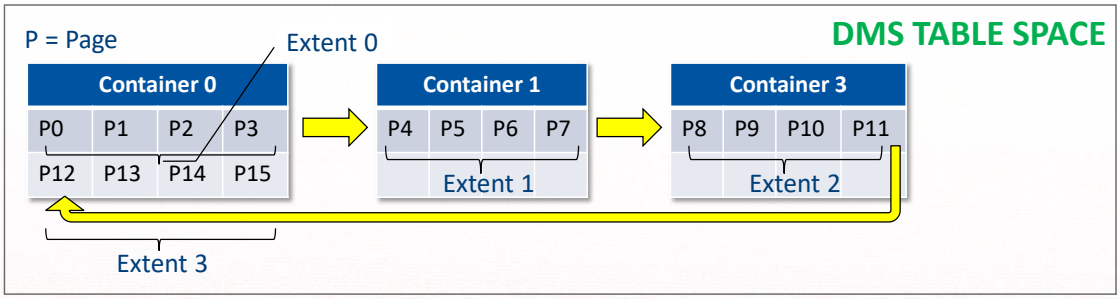
This is what the formatted user data looks like. Notice the 'aaaaa…' and 'cccc…' strings.

Page Consistency

- CBITs
  - Basic method to ***detect partial writes***, used in Db2 releases prior to 9.8
  - A bit from each 512-byte page sector is set to the same value before writing the page; the original value of the bit saved to reserved space in the page header
  - All such bits verified upon reading the page
  - Very fast, but not designed to detect problems *"in the middle"* of the page
- Checksums
  - The default for all new pages starting with 9.8
  - A full page checksum calculated using the Fletcher checksum
  - While still fast, capable of detecting problems ***anywhere in the page***

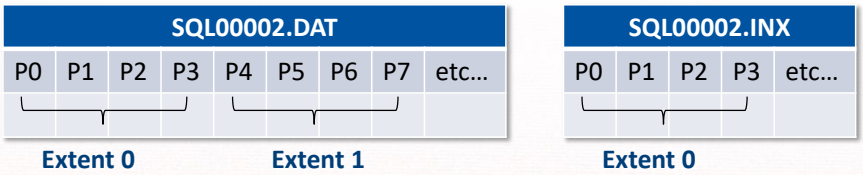
Extent

- Block of 2 – 256 pages (configurable)
- The smallest unit of allocation in a DMS table space
- In multi-container tablespaces, one extent is the number of pages written to the container before moving on to the next container



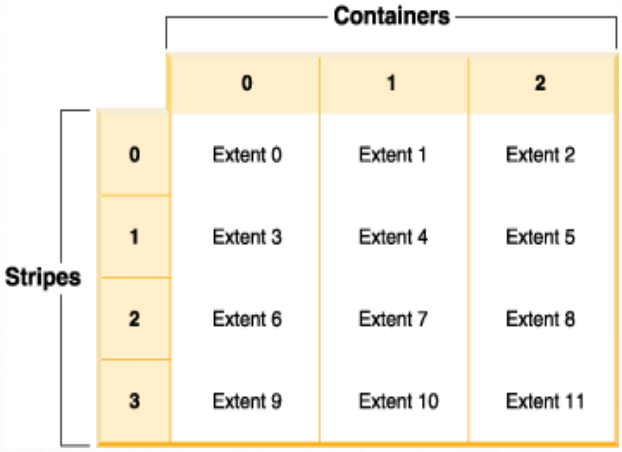
The Simple Stuff – SMS Container

- Each database object type resides in its own file, e.g. SQL00002.DAT
- An SMS container only contains real data, i.e. no metadata pertinent to the table space management *(such as in the DMS case, see the next slides)*
- Allocation and deallocation happens on demand, the container grows and shrinks as soon as the owning object does
- MULTIPAGE_ALLOC (DB CFG, enabled by default) causes Db2 to pre-allocate one extent at a time, otherwise it is one page



DMS Internals: Stripe

- Contiguous block of extents spanning distinct containers
- Used to spread I/O across multiple containers
- Not to be confused with disk stripes
- Example: 3 containers, 12 extents, 4 stripes

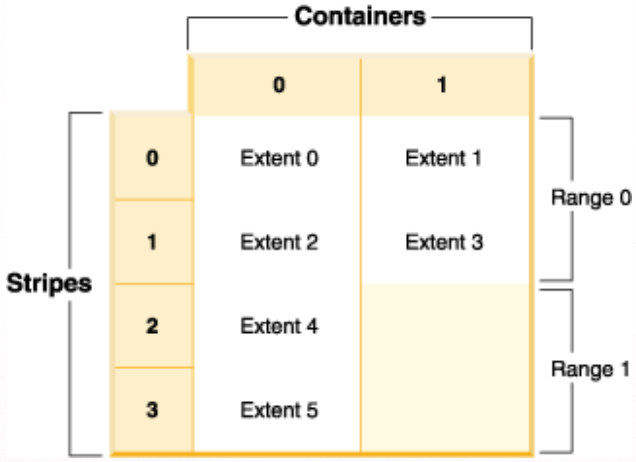


17

Before moving on to the DMS table space structure, some internals must be explained first.

DMS Internals: Range

- Contiguous range of stripes, all of which contain the same set of containers
- Range latch is used to ensure consistency during online backup and rebalancing:
  - Backup/Rebalancer acquires the range latch in exclusive (X) mode
  - All other readers or writers get the latch in shared (S) mode

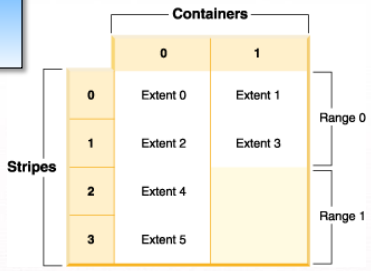


DMS Internals: Map

- Text representation of ranges and extents
- User-viewable by GET SNAPSHOT FOR TABLESPACES ON <db>
- Also printed to Db2 diagnostic log during selected conditions (usually errors)

Range Number	Stripe Set	Stripe Offset	Max Extent	Max Page	Start Stripe	End Stripe	Adj.	Containers
[0]	[0]	0	3	99	0	1	0	2 (0, 1)
[1]	[0]	0	5	149	2	3	0	1 (0)

- This map represents the picture from the previous slide =>
- Extent size: 25
- Adj: offset used by rebalancing, 0 if rebalancing is not in progress



19

There are four extents in the first range, and therefore the maximum extent number addressed in this range (Max Extent) is 3. Each extent has 25 pages and therefore there are 100 pages in the first range. Since page numbering also starts at 0, the maximum page number addressed in this range (Max Page) is 99. The first stripe (Start Stripe) in this range is 0 and the last stripe (End Stripe) in the range is stripe 1. There are two containers in this range and those are 0 and 1. The stripe offset is the first stripe in the stripe set, which in this case is 0 because there is only one stripe set. The range adjustment (Adj.) is an offset used when data is being rebalanced in a table space. (A rebalance might occur when space is added or dropped from a table space.) When a rebalance is not taking place, this is always 0.

There are two extents in the second range and because the maximum extent number addressed in the previous range is 3, the maximum extent number addressed in this range is 5. There are 50 pages (2 extents * 25 pages) in the second range and because the maximum page number addressed in the previous range is 99, the maximum page number addressed in this range is 149. This range starts at stripe 2 and ends at stripe 3.

DMS Internals: SMP – Space Map Page

- Metadata page with object ID 65534 (0xFFFE) indicating which extents in the associated table space are used/free/pending delete by using a simple bitmap for the state of each mapped extent:
  - 0 = Free, 1 = Used, 2 = Pending Delete, 3 = Used + Pending Delete
- No association with the actual object, i.e. an SMP does not “know” which object owns the allocation (see EMP later in this presentation)
- A table space may contain multiple SMP extents
  - The larger the table space, the more SMP extents exist
- SMP extents are at fixed locations in the table space => fast lookup
  - The placement is a function of page size and extent size

DMS Internals: SMP Extent Locations – Example

4 K Page, Extent Size 2

SMP SMP#	Pages in Extent	SMP Extent Location	Pages Mapped	Extents Mapped	Size
0	0 - 1	2	0 - 31999	0 - 15999	131072000
1	2 - 3	32000	32000 - 63999	16000 - 31999	262144000
2	4 - 5	64000	64000 - 95999	32000 - 47999	393216000
3	6 - 7	96000	96000 - 127999	48000 - 63999	524288000
4	8 - 9	128000	128000 - 159999	64000 - 79999	655360000
5	10 - 11	160000	160000 - 191999	80000 - 95999	786432000
6	12 - 13	192000	192000 - 223999	96000 - 111999	917504000
7	14 - 15	224000	224000 - 255999	112000 - 127999	1048576000

<…etc…>

DMS Internals: Pending Delete Extents

- ***"Pending Delete"*** extents are created by removing extents from an owning object (object truncation, reload, object drop, …)
- Such extents cannot be freed until the outcome of the extent removal transaction is known (commit vs. rollback?), thus "pending delete"
- Once it is determined there are no in-flight operations against the given SMP, existing or future transactions are allowed to free pending delete extents, for example when searching for free space during new extent allocation requests

Can also be freed manually by LIST TABLESPACE SHOW DETAIL

DMS Internals: SMP Contents – Time Lapse

```
$ db2dart test /dp /tsi 3 /ps 32 /np 1 /v n
 Page 32 of 32.
     0000  *3000D00F 00000000 00050E2A 03000000*     *0..............*
     0010  *20000000 FEFF0000 00000000 FFFFFFFF*     *................*
     0020  *00870700 00000000 3A5DE246 132C2409*     *..........F....*
     0030  *02000000 00000000 00000000 03000000*     *................*
     0040  *E6850700 00000000 00000000 00000000*     *................*
     0050  *00000000 00000000 00001011 00000000*     *................*
     0060  *00000000 00000000 00000000 00000000*     *................*
     0070  *00000000 00000000 00000000 00000000*     *................*
```

- Table Space Empty
- Table created
- Data inserted
- Data deleted
- Table dropped
- Commit
- Extents freed

- **Deletion of data does not result in freeing extents**
- **E.g. object drop will mark the extents *"pending delete"***

23

Animated slide: 1 means Used, 3 is Pending Delete. This example is taken on a little-endian architecture, hence the bits are being set from the least significant bit (right) to the most significant bit (left).

DMS Internals: EMP – Extent Map Page

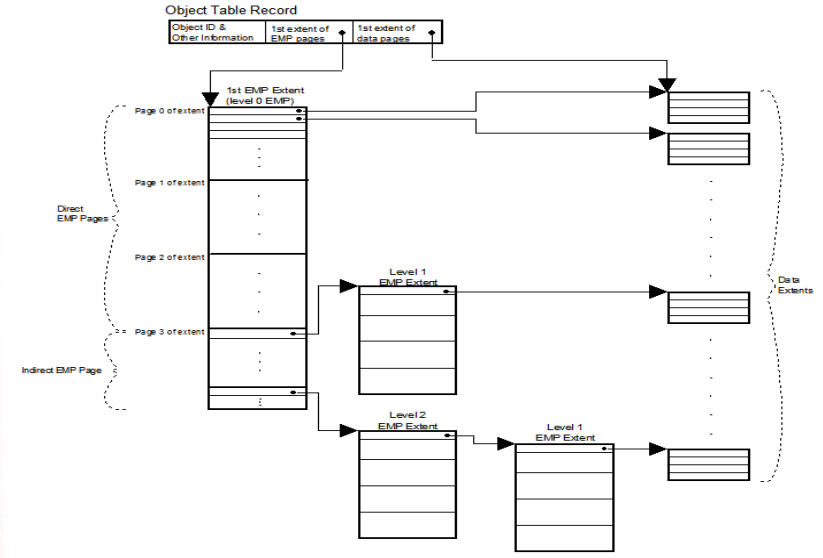
- Unlike SMPs (per-table space), EMPs is a per-object map of extents
- An array of numbers which maps object relative pages to table space relative pages
- No fixed location, can be present almost anywhere in the table space
- The first EMP, a.k.a. ***"Extent Anchor"***, is stored in the ***"Object Table"*** (see next slides)
- The map represented by EMPs can be best visualized by a tree (see next slides)

Discrepancy between SMP and EMP mappings can be serious:

- ***"Orphaned" extent***
  - extent marked as used in SMP, but not mapped by any object's EMP
  - CONCERNING, but can be fixed by db2dart
- ***Extent in use but marked free in SMP***
  - extents mapped by an object's EMP, but marked as free in the SMP
  - **SERIOUS**, must be resolved immediately (IBM Db2 Support, recovery, …)

Any "discrepancies" should be tested for while the database is offline. An attempt to run db2dart on a live database will likely result in false positives.

DMS Internals: EMP "Tree"



Direct EMPs:

- directly pointing to data pages

Single indirect EMPs

- pointing to additional L1 EMPs
- L1 EMPs pointing to data pages

Double indirect EMPs

- pointing to additional L2 EMPs
- L2 EMPs pointing to L1 EMPs
- L1 EMPs pointing to data pages

25

There are 3 types of EMP extents: Level 0, 1, and 2.

All of the EMP pages in a level 0 type EMP extent, except for the last page, are **direct** EMP pages. This means that the entries on it point to actual data extents for the table object. The last page is an **indirect** EMP page and the entries on this page point to other EMP extents. These entries are broken down into two types: **single indirect** entries and **double indirect** entries. All of the entries in the last page, except for the last 16, are single indirect and the last 16 are double indirect. For example, if page size was 4 KB (which means 1000 entries on each page) then there would be 984 single indirect and 16 double indirect entries.

Single indirect entries point to level 1 type EMP extents. All of the EMP pages in a level 1 type EMP extent have direct entries pointing to data extents.

Double indirect entries point to level 2 type EMP extents. All of the EMP pages in a level 2 EMP extent have indirect entries that point to level 1 type EMP extents.

DMS Internals: EMP Contents – Time Lapse

```
$ db2dart test /demp /tn T1 /tsi 3
  DAT extent anchor: 96
   Traversing extent map for object type: 0
      Tablespace ID: 3, Tablespace Seed: 3, Object: 4 EMP page class: 64,
   <...skipping...>
   INX extent anchor: 160
   Traversing extent map for object type: 1
      Tablespace ID: 3, Tablespace Seed: 3, Object: 4 EMP page class: 65,
      EMP pool page: 160, # entries: 1
      Page LSN = 0000000000099B21
      Pool relative page #'s :
        192
```

- Table T1 created
- Data inserted
- Data deleted
- Table truncated
- Index created

- **Deletion of data does not result in freeing extents**
- **Each object type (data, index, …) has its own anchor**

Animated slide.

The first object relative page for table T1 is table space relative page 128, the second one at 160, the third one at 192 etc…

This sequence also reveals the extent size: 32

DMS Internals: Object Table

- A regular relational table, but invisible to the user
- Contains mapping between a given object and its ***"Extent Anchors"***
  - Extent Anchor – root EMP in the aforementioned tree, allowing to find the "beginning" of extents of the given type (data, index, LOBs, … – see next slide)
- Object ID of the Object Table is 65535 (0xFFFF)
- The extent map for the object table consists of only one page, extent 0, table space page 1
- The first extent containing data for the object table is always extent 2

DMS Internals: Object Table – Example

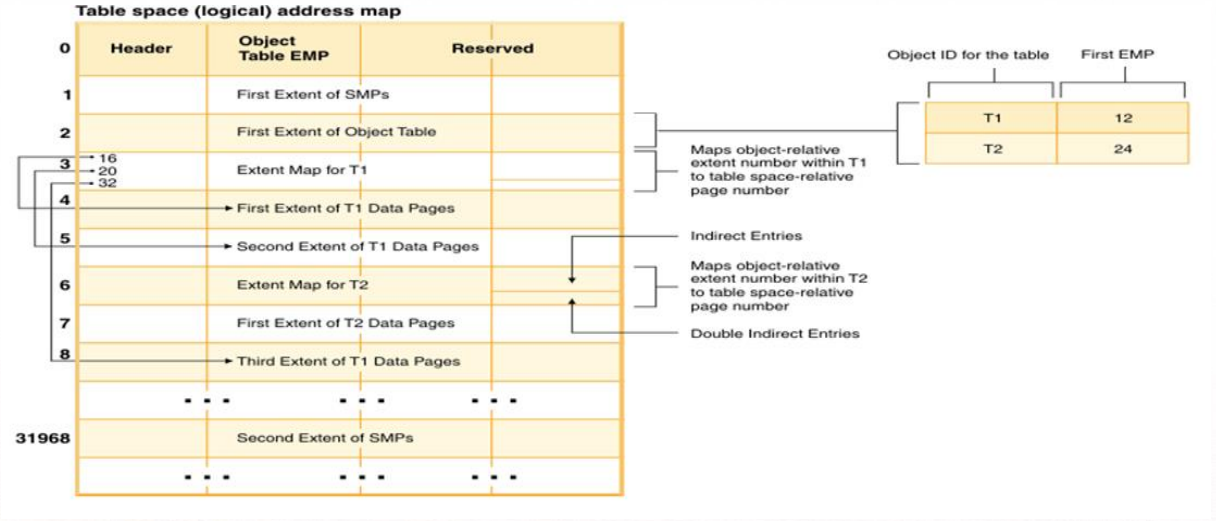
```
$ db2dart test /dd /tsi 3 /oi 65535 /ps 0 /np 0 /v y

 Object Table record:
               Type: BASE              Version: x00
                  Pool  ObjID  Extent Anchor      Page 0    Obj State  ObjType    Life LSN

      DATA        3      4           96             0        EXIST   VALID  0000000000099AF8
      BKM         3      4            0             0    NOT EXIST   VALID  0000000000099AF8
      INDEX       3      4          160             0        EXIST   VALID  0000000000099B32
      LONG        3      0            0             0    NOT EXIST   VALID  0000000000000000
      XDA         3      0            0             0    NOT EXIST   VALID  0000000000000000
      LOB         3      0            0             0    NOT EXIST   VALID  0000000000000000
      LOBA        3      0            0             0    NOT EXIST   VALID  0000000000000000
```

- **Data Extent Anchor at page 96, Index Extent Anchor at page 160 of the table space (see the previous slide)**

Finally, the Big Picture! DMS Layout



29

This is a logical map of a DMS table space. Page 0 (marked here as "Header") does not actually start at physical offset 0. The first extent (residing at physical offset 0) is used by the table space tag, see the Backup Slides section.

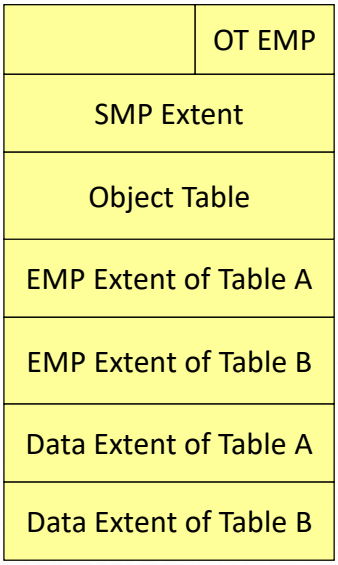
Within the table space address map, there are two types of map pages: extent map pages (EMP) and space map pages.

The object table is an internal relational table that maps an object identifier to the location of the first EMP extent in the table. This EMP extent, directly or indirectly, maps out all extents in the object. Each EMP contains an array of entries. Each entry maps an object-relative extent number to a table space-relative page number where the object extent is located. Direct EMP entries directly map object-relative addresses to table space-relative addresses. The last EMP page in the first EMP extent contains indirect entries. Indirect EMP entries map to EMP pages which then map to object pages. The last 16 entries in the last EMP page in the first EMP extent contain double-indirect entries.

The extents from the logical table space address map are striped in round-robin order across the containers associated with the table space.

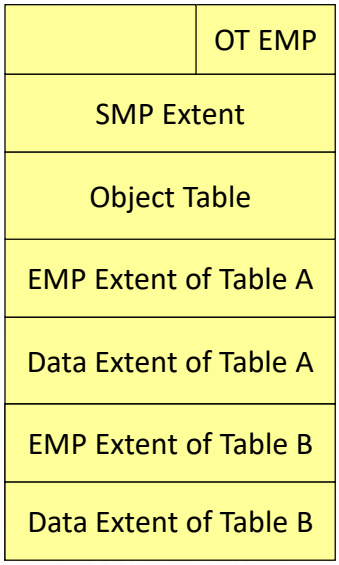
What Happens during Disk Page Read?

1. A read page request arrives (table space ID, object ID, object type, object-relative page number)
2. For the table space identified by the pool ID, find the Object Table EMP to locate the Object Table
3. Search the Object Table to locate the first EMP extent for the given object ID (say, Table A)
4. Traverse the EMP extents of Table A to locate the given object-relative page number and its translation to the pool-relative page number
5. Use the pool-relative page number to locate the container and offset to read the extent into the buffer pool *(unless not inlined LOB/LONG)*



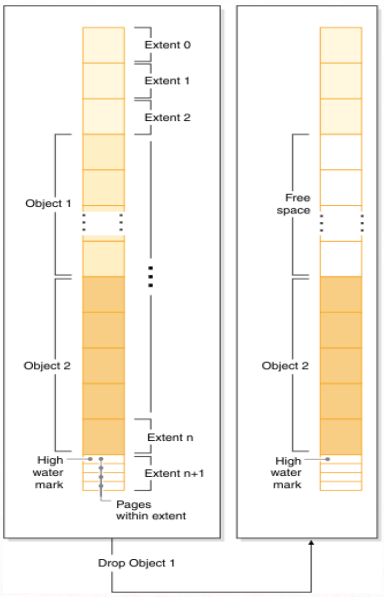
What Happens during CREATE TABLE?

1. Insert a new row into the Object Table and obtain TID (object ID) and FID (table space ID)
2. Allocate an EMP extent by looking at the SMP for free space and marking the extent used
3. Allocate a data extent and update the EMP extent with the pool page number
4. If all is well, update the object table with the extent anchor and page 0 for the newly created table



High Water Mark

- The last allocated extent in a DMS table space
- Unlike with water, there may be “holes” under the HWM, i.e. the allocations may not (and usually are not) contiguous
- The table space cannot be reduced in size unless the object holding the HWM is moved (or dropped)
- The documentation claims that *“Practically speaking, it's virtually impossible to determine the high water mark yourself”*… we beg to differ! (next slide ☺)



Displaying the High Water Mark

```
$ db2dart test /dhwm
High water mark:  538 pages, 269 extents (extents #0 - 268)

 [0000] 65534 0x0e   [0001] 65534 0x0e   [0002] 65535 0x00   [0003] == EMPTY ==
 [0004] == EMPTY ==  [0005] == EMPTY ==  [0006] == EMPTY ==  [0007] == EMPTY ==
 <...skipping...>
 [0132] == EMPTY ==  [0133] == EMPTY ==  [0134] == EMPTY ==  [0135] == EMPTY ==
 [0136]      5 0x40* [0137]      5 0x00* [0138]      5 0x43* [0139]      5 0x03*
 [0140]      5 0x44* [0141]      5 0x04* [0142]      5 0x00  [0143]      5 0x00
 [0144]      5 0x00  [0145]      5 0x00  [0146]      5 0x00  [0147]      5 0x00
 <...skipping...>
 [0268]      5 0x00

Object holding high water mark:
 Object ID: 5
 Type:      Table Data Extent
```

Extent Number

Object ID

Object Type

REBALANCE vs. REDUCE – Do Not Confuse! 1/2

- Both REBALANCE and REDUCE are about moving extents from one place to another, but they are fundamentally different
- REBALANCE happens when a table space container is added or removed, and the user issues **ALTER TABLESPACE … REBALANCE** in an effort to spread the data evenly across the available containers
  - The only thing that changes is the table space map and data physical location
  - Page metadata (e.g. disk page number) remains unchanged
  - Forward rebalancing = containers added, reverse = containers removed
  - The operation is fully online, although impact on running workload may be easily noticeable (could be a "heavy" impact operation)

34

Rebalance is not logged. Once started, the table space state will not go back to normal unless the operation has completed. However, rebalance can be suspended (ALTER TABLESPACE … REBALANCE SUSPEND) and resumed (ALTER TABLESPACE … REBALANCE RESUME). Rebalance is not available in IBM Db2 pureScale ® as of now.

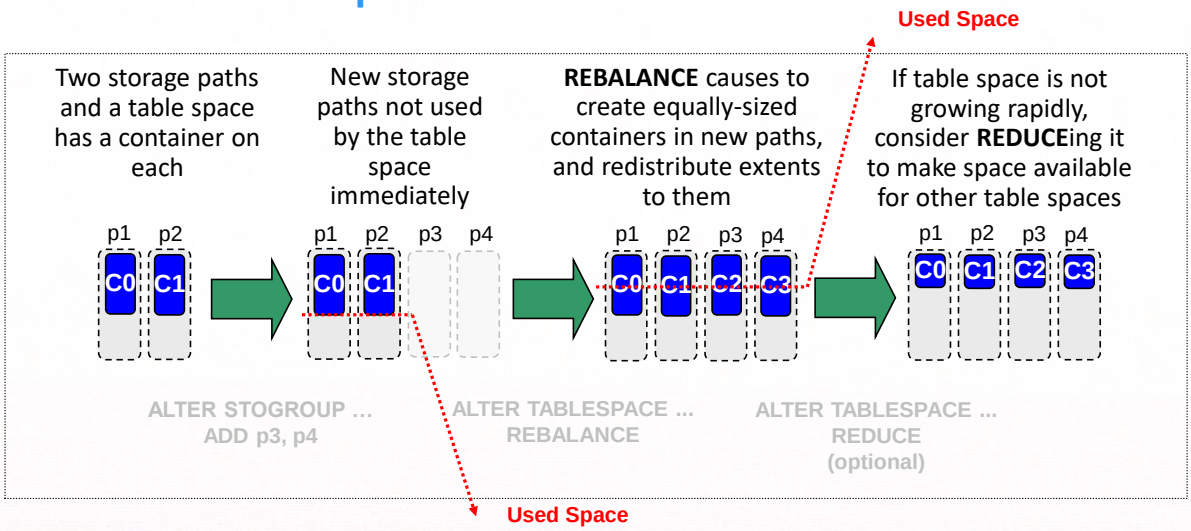
REBALANCE vs. REDUCE – Do Not Confuse! 2/2

- REDUCE (a.k.a. *"Extent Reclaim"*) is triggered by the user issuing **ALTER TABLESPACE … REDUCE** in an effort to reduce the high water mark for the table space
  - Supported with reclaimable storage DMS table spaces (created in 9.7 or newer)
  - One extent reclaim per table space, one mover thread EDU per extent reclaim
  - Performance not critical, usually negligible impact
  - Incompatible with operations such as backup, load, and others; extent reclaim will yield to other operations and pause automatically
  - Page metadata changes – the page is physically moved to a different location in the table space
  - The table space map remains unchanged

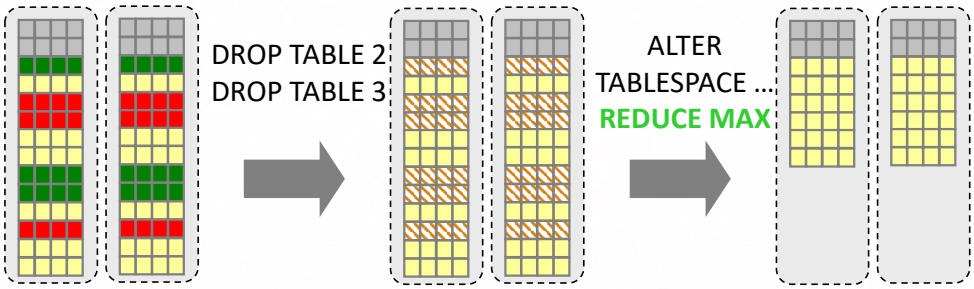
35

Unlike rebalance, reduce is logged. The operation can be stopped and terminated at anytime. Reduce is not available in IBM Db2 pureScale ® as of now.

REBALANCE Example



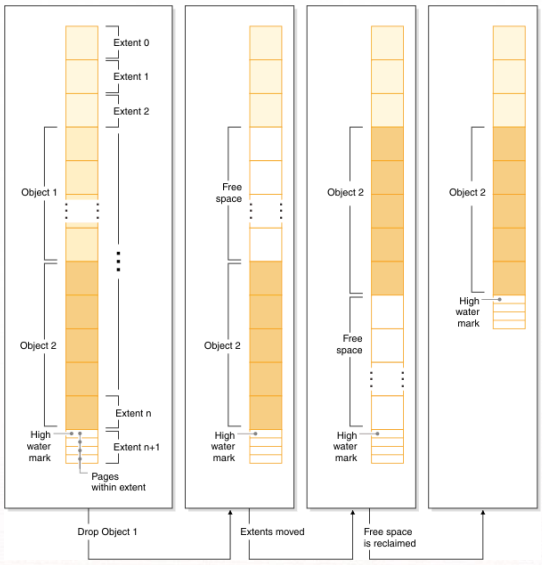
REDUCE Example 1/2



- Internal table space metadata extents
- Table 1
- Table 2
- Table 3
- Extent that is allocated to a table space, but not to a table

REDUCE Example 2/2

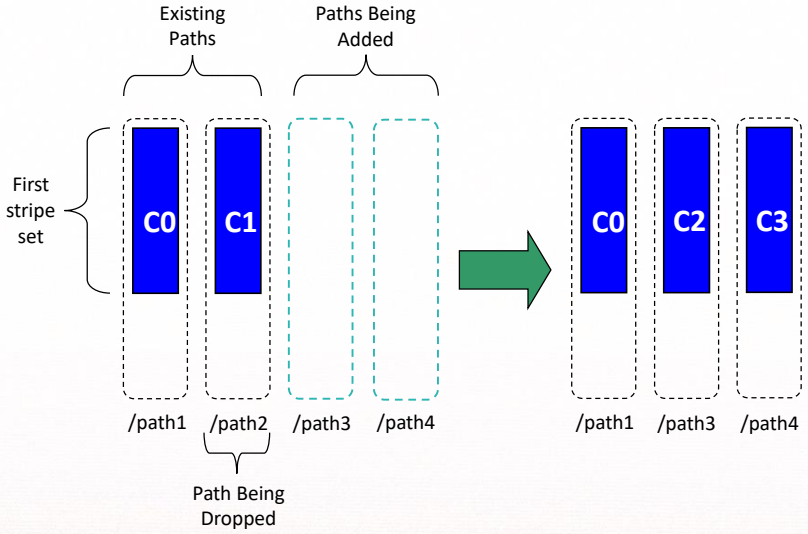
- Live extents are moved to occupy unused extents lower in the table space
- Once there is free space at the end of the table space, the table space can be reduced in size, and the free space can be released back to the file system
- The principle is similar to that of a file system defragmentation



Storage Groups/Paths

- Storage Group
  - A named group of storage paths
  - **CREATE STOGROUP sg ON '/path1', '/path2'**
  - Often used to manage multi-temperature storage
  - Each group typically represents a different class of disk storage
  - At least one default group, IBMSTOGROUP, exists in an AS-enabled database
- Storage Path
  - A filesystem path, a member of a storage group
  - **ALTER STOGROUP IBMSTOGROUP ADD '/path3'**
  - The ALTER DATABASE … ADD STORAGE statement has been **deprecated**

Storage Path ADD, DROP, and REBALANCE Example

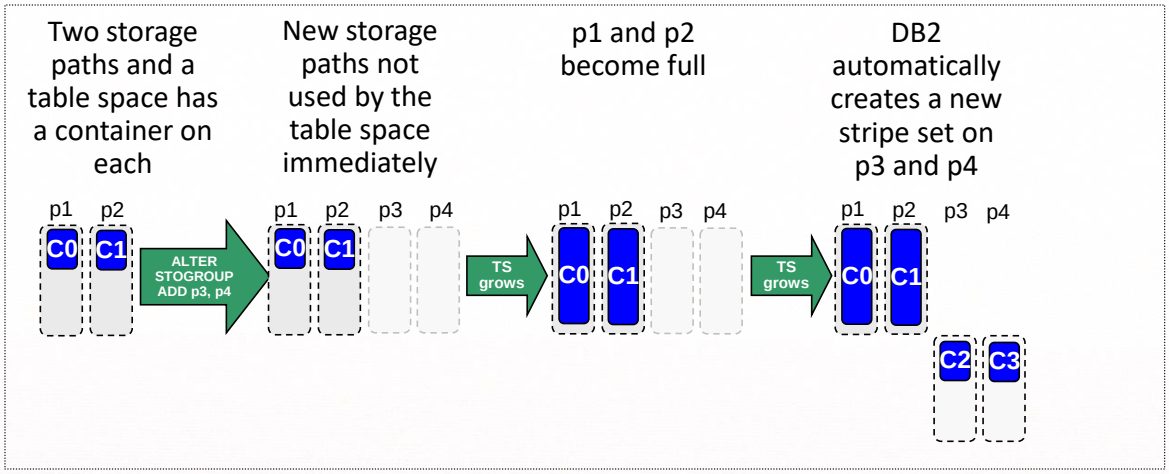


A two-pass rebalance will be done in this case:

1) Rebalance moving space across the new paths (forward)
2) Rebalance moving space off of the path being dropped (reverse)

- Newly added paths will not be "In Use" unless REBALANCE has been performed
- Likewise, the existing path to be dropped will be in "Drop Pending" unless REBALANCE has been performed

Automatic Table Space Growth



Filesystem Caching

1. Direct I/O (DIO): data is transferred directly from/to the disk without going through the filesystem cache
2. Concurrent I/O (CIO): similar to DIO, but avoids inode locking
   - The inode lock is used to control write serialization. Bypassing the inode lock means that the application must be designed to handle serialization in its own way. Db2 supports both DIO and CIO.
   - Usually provides raw-like performance
3. Write-thru: regardless of caching configuration, ensures the data has hit the disk; critical for recovery from outages

Caching in Db2

- Controlled by **ALTER TABLESPACE … [NO] FILE SYSTEM CACHING**
- Since 9.5, the default has been NO FILE SYSTEM CACHING except for
  - AIX JFS
  - Linux for System z®
  - All SMS temporary table space files
  - LOB/LONG data files in SMS permanent table space files
- Caching usually results in improved **read** performance, especially in data warehouses systems with an under-configured extent size
- On the other hand, caching may slow down **write** performance

43

See Best Practices at the end of this presentation for some additional thoughts

Parallelism and Prefetch Size Calculations

a) For DMS, the initial parallelism is the largest number of containers in a stripe set. For SMS, it is the number of containers for that table space.

b) Parse DB2_PARALLEL_IO. The default is "6" in pureScale®, or undefined in all other cases. Use this value as “num_disks_per_container” below.

c) If the prefetch size is automatic, calculate the prefetch size:
prefetch_size = num_containers * extent_size [* num_disks_per_container] <= if defined

d) If DB2_PARALLEL_IO is set, prevent the parallelism from underflowing to zero (may happen if the prefetch size is not automatic) :
parallelism = (prefetch_size + extent_size - 1) / extent_size
if parallelism == 0 then parallelism = 1

e) If DB2_PARALLEL_IO is unset, use the parallelism calculated in step a)

44

Table space parallelism is utilized by many areas, for example prefetching, index scans, LOB I/O, backup, and others. The algorithm may change without notice.

Migration to Automatic Storage 1/3

- Method 1: Table Space Alter
  - **ALTER TABLESPACE … MANAGED BY AUTOMATIC STORAGE** followed by **ALTER TABLESPACE … REBALANCE**
  - Works with DMS table spaces only
  - Fully online
- Method 2: Redirected Restore
  - **SET TABLESPACE CONTAINERS … USING AUTOMATIC STORAGE** followed by **ALTER TABLESPACE … MANAGED BY AUTOMATIC STORAGE** after a connect
  - Works with DMS table spaces only
  - Can also be used to convert from raw to filesystem-based containers

Method 2: The ALTER TABLESPACE command is required in order to update the system catalog with the new table space type.

Migration to Automatic Storage 2/3

- Method 3: Schema Transport
  - **RESTORE … TRANSPORT INTO**
  - Useful when converting SMS catalog table space to automatic DMS
  - http://www-01.ibm.com/support/docview.wss?uid=swg21984655
- Method 4: Data Unload
  - Unload data, create a new automatic table space, reload with data
  - Works with any table space type
  - db2move, ADMIN_MOVE_TABLE, db2look/EXPORT/LOAD, load from cursor
  - Requires manual work

Migration to Automatic Storage 3/3

- Method 5: Enterprise Migration Tool
  - IBM has a tool called XenoBridge
  - High performance unload/load designed for enterprise systems
  - In-built data integrity checks
  - Near-zero downtime using replication technology
  - Can also be used in endian and character set conversions

- Further Reading
  - Upgrading to the Db2 pureScale® Feature (see Appendix A-D)

Best Practices 1/3

- Use dedicated storage
  - Especially separate the transaction logs from table space data
- Do not bother with raw devices
- Create your filesystems uniformly
  - Same size, …
- Spread your table spaces evenly across all available file systems
- Use DB2_PARALLEL_IO to give a hint on the number of disks per path
- Set the extent size to a multiple of the RAID stripe size

48

- Strip – the amount of contiguous data on one RAID disk, for example 64 KB
- Stripe – the sum of all stripes, for example 64 KB * 4 RAID disks = 256 KB
- Assuming a 16 KB page size, a good extent size in our case is for example 32 (32 * 16 = 512 KB = 2 * 256 KB stripe)

Best Practices 2/3

- Use automatic storage for user data
  - Avoid old-style SMS for permanent table spaces
- Use automatic storage for temporary data
  - DMS is not prohibited (although not recommended), but may cause surprises:
    - For example, when using HADR with the Read on Standby feature. Queries at Standby might generate unique sorts, which may result in different DMS temporary table space sizes on Primary vs. Standby. Surprising?
- Migrate old-style SMS or DMS created before 9.7 (non-reclaimable) to reclaimable storage automatic DMS

Best Practices 3/3

- Plan non-critical operations such as REBALANCE or REDUCE for a low traffic or maintenance window
- LOB/LONG data is not cached in buffer pools (except when inlined):
  - Put LOB/LONG data to dedicated table spaces
  - Enable filesystem caching for table spaces with LOB/LONG data
- XML data is cached in buffer pools
  - Larger XML documents will be split into smaller page-sized pieces
  - Use a larger page size (32 KB) to create as few pieces as possible
  - Use a separate table space for XML data (CREATE … LONG IN … works with XML)
  - Use compression when possible

BACKUP SLIDES

Old DMS Layout Explained (Not Recommended, FYI Only)

```
$ db2 "create tablespace testts managed by database using (file 'file1' 1M, file 'file2' 1M)"
DB20000I  The SQL command completed successfully.

$ ls -l /home/psustr/psustr/NODE0000/SQL00001/file*
-rw------- 1 psustr pdxdb2 1048576 Aug 16 18:46 /home/psustr/psustr/NODE0000/SQL00001/file1
-rw------- 1 psustr pdxdb2 1048576 Aug 16 18:46 /home/psustr/psustr/NODE0000/SQL00001/file2
```

Storage Path: /home/psustr

Instance Name: psustr

Node: NODE0000

DB Directory: SQL000001

Container: file1, file2

Old SMS Layout Explained (Not Recommended, FYI Only)

```
$ db2 "create tablespace testsms managed by system using ('testsms')"
DB20000I  The SQL command completed successfully.

$ db2 "create table t1 (c1 integer) in testsms"
DB20000I  The SQL command completed successfully.

$ ls -l /home/psustr/psustr/NODE0000/SQL00001/testsms/*
-rw------- 1 psustr pdxdb2 4096 Aug 18 10:46 /home/psustr/psustr/NODE0000/SQL00001/testsms/SQL00002.DAT
-rw------- 1 psustr pdxdb2  512 Aug 18 10:43 /home/psustr/psustr/NODE0000/SQL00001/testsms/SQLTAG.NAM
```

Storage Path: /home/psustr

Instance Name: psustr

Node: NODE0000

DB Directory: SQL000001

SMS Path: testsms

Tag + containers

SQLTAG.NAM – Container Tag

- A 512-byte "stamp" allowing Db2 to recognize its own table space containers
- Prevents unintentional overwrites: an attempt to reuse a live container for a table space results in SQL0294N *"container in use"*
- For DMS, the tag is located in the first extent of each container
- For SMS, the tag is located in a regular file named "SQLTAG.NAM"
- Some of the contents: database identifier (a.k.a. *"seed"*), table space ID, container ID, *"timestamp"* of creation (*"life LSN"*), database name, instance name, database path, tag checksum, …
- **DO NOT DELETE ☺**

SQLSPCS.1/.2 – Table Space Metadata

- Bootstrap information for all table spaces
- Located in two copies in the database home path
- Each table space has a pre-defined constant location
  - This allows for concurrent writing => great concurrency
  - However, the file may be (unexpectedly) large if there are *"holes"* in table space IDs
  - 256 KB per table space entry
- Some of the contents: table space ID, name, flags, extent/prefetch size, backup and recovery info, size, storage group association, container info, etc…
- **ABSOLUTELY DO NOT DELETE ☺**

SQLSPCS.1/2
Table Space 0
Table Space 1
<Deleted/Unused>
Table Space 3
<Deleted/Unused>
<Deleted/Unused>
Table Space 6
<etc…>

SQLSGF.1/.2 – Storage Group File

- Bootstrap information for storage groups and paths
- Located in two copies in the database home path

1. Header
   - DBSTOGR eyecatcher, version, checksum, default storage group ID
2. Storage Group Information
   - SG ID, name, version, parallelism, checksum, path assignments
3. Storage Path Information
   - Path ID, path location, state (In Use, Drop Pending, Normal, …)

- **DO NOT DARE TO DELETE ☺**

56

Storage groups and paths are discussed in more detail later in this presentation.

Extracting Data/Metadata – Summary

Task	Command	Notes
Extract a page in hex (unformatted)	db2dart <db> /DP	
Extract and format a page	db2dart <db> /DD	Use /v (verbose) to also print page data
Extract an SMP	db2dart <db> /DP	You must know the SMP locations
Extract an EMP	db2dart <db> /DEMP	
Extract the Object Table	db2dart <db> /DD /OI 65535 /v	65535 is the object ID for the Object Table
Extract and format an index page	db2dart <db> /DI	
Extract and format an XML data page	db2dart <db> /DXA	
Extract an XML data page in hex	db2dart <db> /DXH	
Display High Water Mark	db2dart <db> /DHWM	

Additional Stuff That IBM May Ask For

Task	Command	Notes
Extract a page descriptor from a local buffer pool	db2pd -db <db> -dmpbufp opt=p mode=s	FYI only. A page descriptor, a.k.a. BPD, is internal metadata. Unformatted output.
Extract page contents from a local buffer pool	db2pd -db <db> -dmpages	FYI only. Internal. Unformatted output.
Extract a page from a global buffer pool (pureScale ®)	db2pd -db <db> -cfdump struct=gbp pgtype=page,dirent	FYI only. Internal. Unformatted output.

- For your reference only
- This data may be requested by IBM for problem determination

IDUG
Leading the DB2 User Community since 1988

IDUG Db2 Tech Conference NA
Philadelphia, PA | April 29 - May 3, 2018

#IDUGDb2

Pavel Sustr
IBM Toronto Lab
psustr@ca.ibm.com
@pavel_sustr

Session code: D10