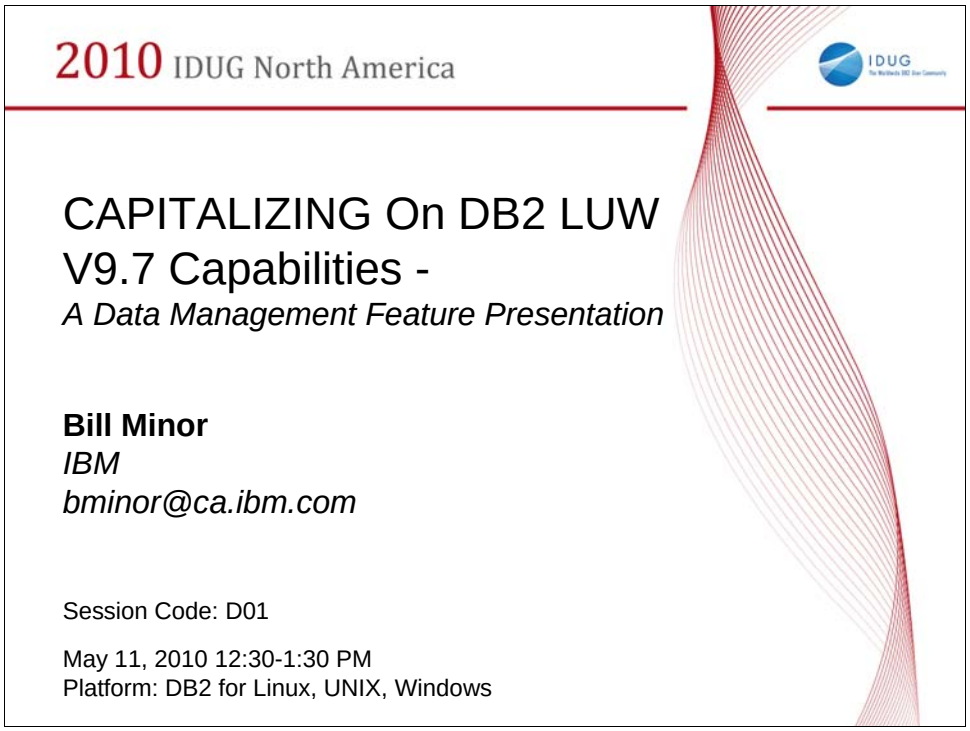


Speaker Biography

Bill is a Senior Technical Development Manager with the IBM DB2 LUW Development Team. He works in the DB2 LUW engine or kernel and specializes in the area of data object storage, access, and maintenance. Bill has presented at a number of DB2 Conferences and User Groups. Bill has been with the DB2 Team since 1996. In his spare time, he tries to learn how to strum a guitar.

Abstract and Objectives

Controlling data growth and optimizing storage use are critical for keeping costs in check and ensuring peak performance. Consolidation or mergers, regulatory compliance and increased demand for business analytics and data mining are causing data volumes to proliferate. Consequently, it is important to be able to understand, choose and maintain the right storage strategy in order to meet your business objectives and successfully work within restricted operating budgets.

This session will focus on new features and Best Practices for successfully deploying and utilizing the storage management capabilities of DB2 V9.7. Topics of discussion will include compression, utility maintenance, data evolution, data life cycle management, space reclaim, and monitoring.

Objective 1: Illustrate the new storage capabilities of DB2 LUW V9.7
Objective 2: Demonstrate approaches for ease of feature implementation
Objective 3: Demonstrate improvements in availability, monitoring, and performance of DBA tasks
Objective 4: Gain an understanding of how DB2's storage capability can reduce your Total Cost of Ownership
Objective 5: Relate Best Practices for maximizing your Return on Investment

This is the Abstract and session Objectives as recorded in the conference schedule.

Data Management Features:

- This is a storage based presentation. While comprehensive, it is not an exhaustive overview of Database storage in DB2 LUW
- Features to be discussed include:
  - Table Compression
  - Index Compression
  - LOB Inlining
  - Range Partitioned Tables
  - Space Reclaim: Tables and Tablespaces
  - Truncate Table
  - Last Used

What is this presentation all about …

This presentation is specific to DB2 Features that allow one to management data growth.

Won't talk about things like Automatic Storage (ASM) or Hierarchical Storage Management (HSM) – the tiering of data of different types of storage media

Not going to talk about Optim Data Growth Solution

Lots and Lots of Data …

- Search engine G processes about 20PB of data/day
- Avatar: 1PB of local storage for rendering 3D effects
- Experiments at the Large Hadron Collider will produce over 15PB of data/year
- The Internet Archive: 2PB of data with current growth rate of about 20TB per month

Before we jump into DB2, we know data is collected every where and that data volumes are proliferating. Look at these interesting examples.

While PB databases are certainly not the norm yet, databases are rapidly increasing in size and controlling that growth and managing the data content is becoming increasingly more important. I am sure you all have experiences that you could relate in this regard.

DMS Tablespace Capacity

Page Size	4 Byte RID	6 Byte RID ('Large RIDs') V9.1	**6 Byte RID ('Large RIDs') V9.7 (Default)**
4KB	64GB	2TB	**8TB**
8KB	128GB	4TB	**16TB**
16KB	256GB	8TB	**32TB**
32KB	512GB	16TB	**64TB**

6 Byte RID:
- For tables in LARGE table spaces (DMS only)
- Also all SYSTEM and USER temporary table spaces

Let's begin by first looking at the capacity of a DB2 database.

This chart shows what the storage limit is for a DMS tablespace as a function of page size.

DB2 LUW Database Capacity

- Maximum size of a DMS Tablespace as of V9.7: 64TB
- Maximum number of Tablespaces in a Database: 32768
- Maximum number of Database Partitions (DPF): 1000

So the biggest database you could have would be
64TB x 1000 x 32768
or
64PB x 32768
or approximately
2.1ZB (zettabytes) !!!!!!!

So, 'Yes' we have the capacity to build huge gigantic ridiculously large databases …. Could you imagine managing something this big or maybe better yet, imagine being the Sales Rep who sold that much storage?!!! ☺

What is the point here? We all know databases are getting bigger with big ones being in the TB upwards to PB level. The amount of hardware required to support such systems is big. But … wait that is not all … the 'cost' associated with maintaining and administering such large systems can outstrip the hardware investment.

The goal of this presentation is to introduce DB2 LUW features that you can utilize to help optimize your investment: better use of storage, higher availability and better performance.

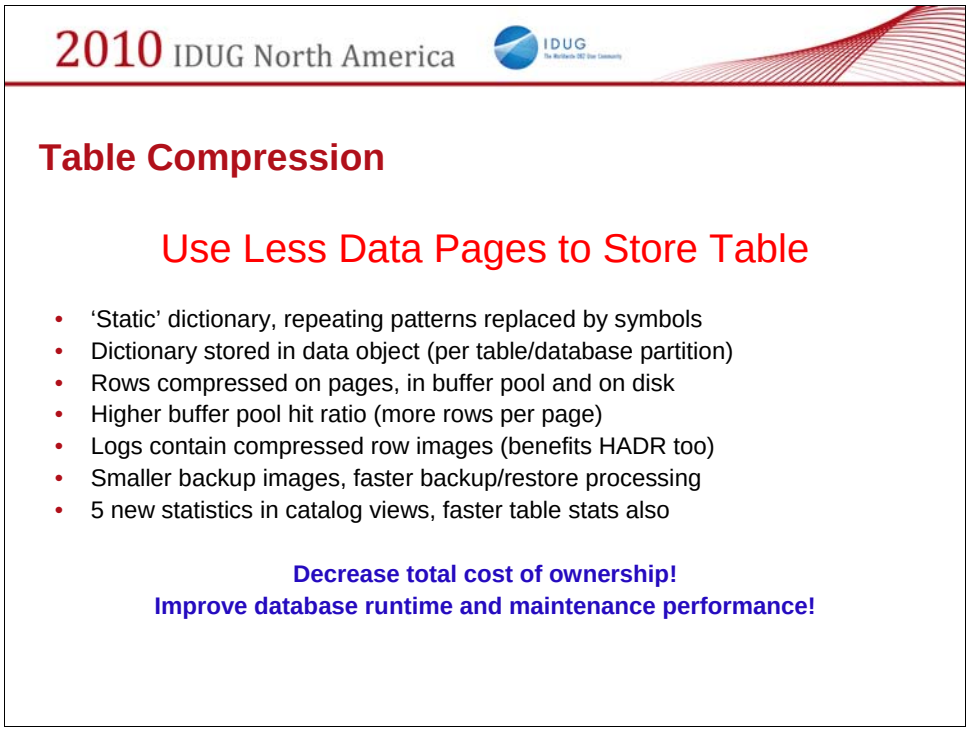
Table Compression

- Table Compression is designed to help with ever increasing data volumes
- Table Compression Best Practices Paper:
  http://www.ibm.com/developerworks/db2/bestpractices
  - Determining what tables to compress
  - Estimating compression ratios
  - Understanding and choosing a specific approach to compressing tables
  - Monitor and gauge effectiveness of compression using statistics
  - ….
- Table or Row Compression is a fundamental piece of the overall DB2 Storage Optimization strategy
  - We will briefly review that technology next and then move into a discussion of new compression and storage management capabilities in V9.7

Table or Row Compression, as you are probably aware, was first introduced into DB2 V9.1. The main value proposition with this feature is to use less data pages to store your table data. While reducing your overall database footprint this feature has also been shown to improve performance in systems where I/O has been the traditional bottleneck. Other benefits are smaller database backup images with faster Backup and Restore processing, reduced logging, and more efficient memory utilization.

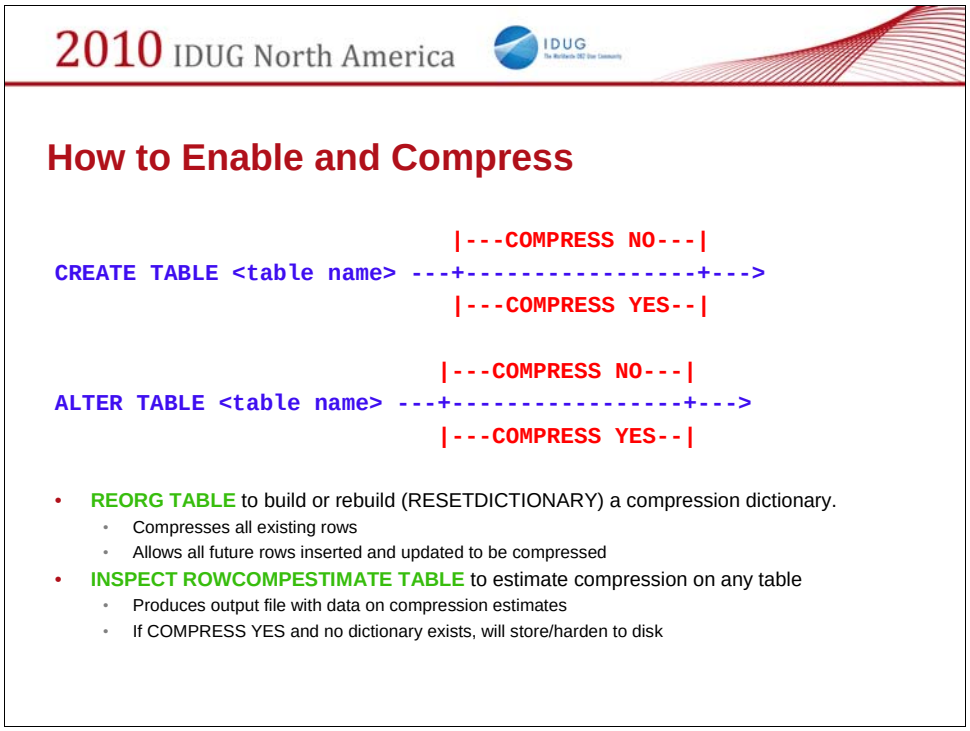
The Compression Best Practices paper at the developerWorks site provides an excellent overview of what Table Compression is, how to implement it and how to best manage it.

In the next few slides I will provide a brief overview of the Table Compression capability as it is a fundamental piece of the overall DB2 Storage Optimization solution. We will then move into new features in DB2 V9.7 which include extended compression capabilities but also features specific to storage management and optimization as a whole.



DB2 9 delivered the initial functionality and capability of compression. The design and implementation were predicated on well known approaches, most notably those well established via DB2 zOS compression.

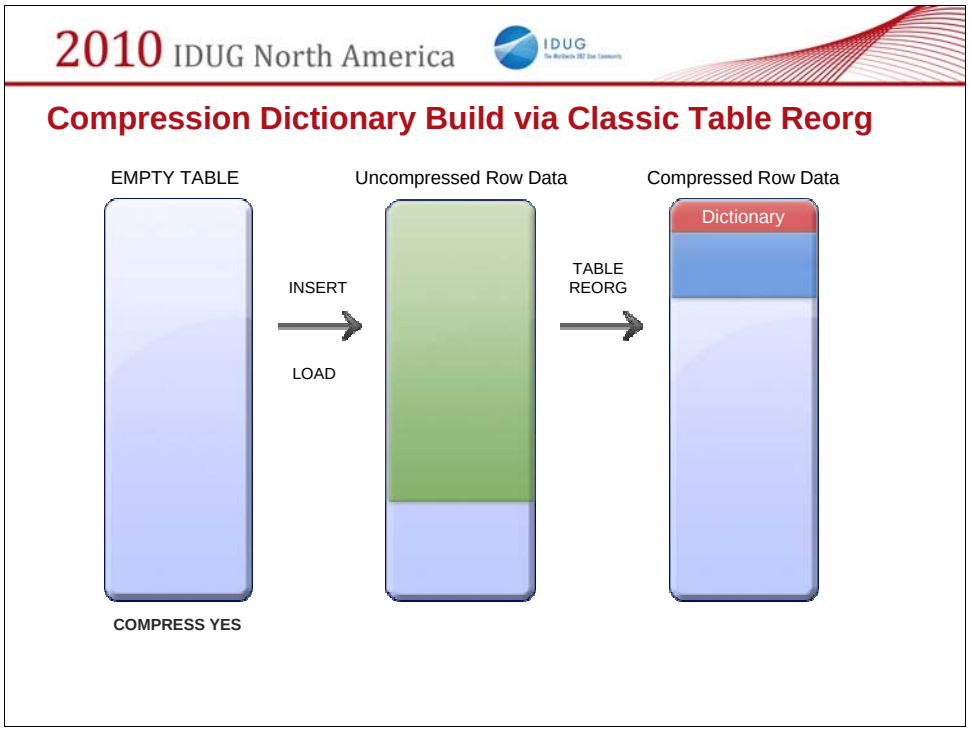
This list identifies many of the benefits of compression



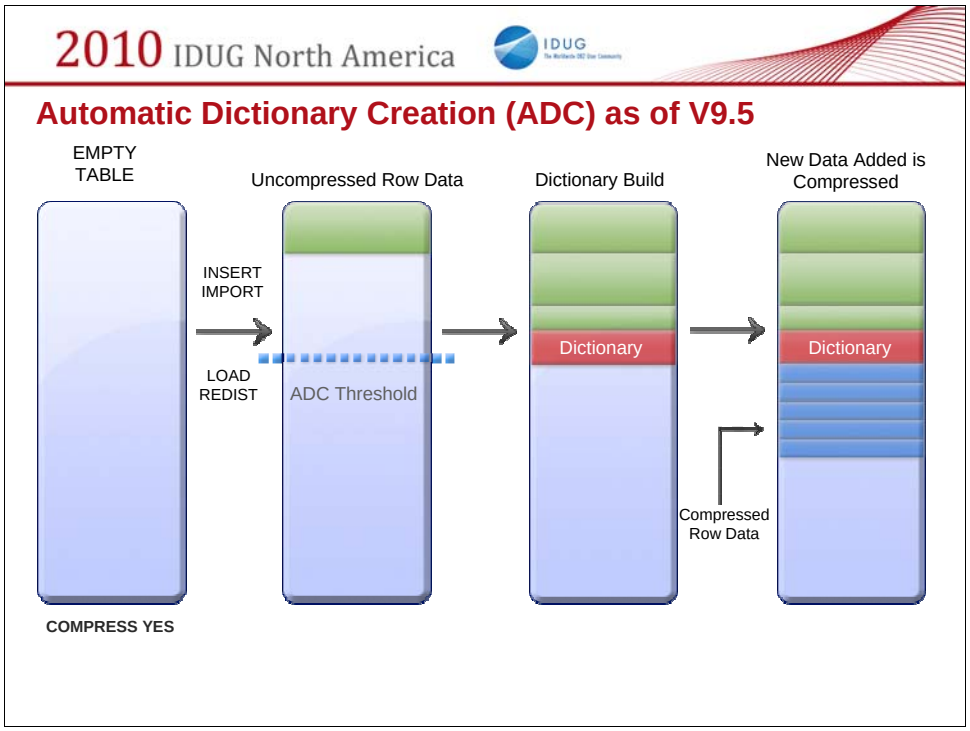
DDL is extended for both the CREATE TABLE and ALTER TABLE statements to enable or disable compression.

Before rows can be compressed, a dictionary must be created and stored in the table's data object.

To build a dictionary (the only ways in V9.1), REORG the table or use the INSPECT ROWCOMPESTIMATE utility. REORG will build a dictionary and compress all existing rows, while the INSPECT ROWCOMPESTIMATE utility will build a dictionary and save it (so long as the table has COMPRESS YES and no dictionary currently exists). This does not compress existing rows, but allows future INSERT and UPDATE activity to benefit from compression.

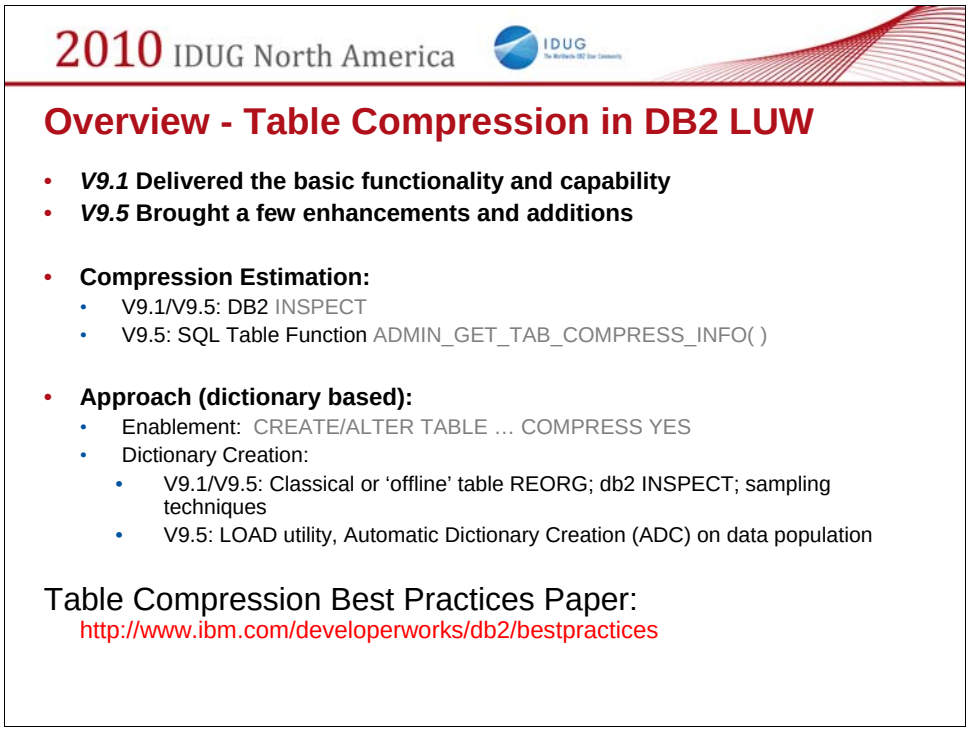


This slide is a pictorial representation of compression dictionary building and table compression. The figure on the far left represents an empty table which has been enabled for compression. It is subsequently populated with data which is uncompressed – the green shading in the middle figure rectangle depicts uncompressed data residing in the table. A subsequent table reorg creates a compression dictionary and compresses all the records that exist within the table. All new data which may be moved into the table is now also subject to being compressed.



This pictorial is meant to illustrate how Automatic Dictionary Creation takes place.

First, we have an empty table which has the table COMPRESS attribute set to YES. Data then begins to be moved into the table. Table growth can proceed by SQL INSERT processing or utility processing such as IMPORT, LOAD, or REDISTRIBUTE. As the data begins to populate the table it resides in the table as uncompressed – symbolized by the green colored rectangles. Note the dotted blue line in the figure – this represents the threshold for triggering ADC. It is an internally set threshold. Once the table reaches a certain size (on the order of 1 to 2MB of pages) and contains a sufficient amount of data, dictionary creation is automatically triggered. As the threshold is breached automatic dictionary creation occurs and the compression dictionary is stored in the table. Data is able to be populate into the table as this occurs. Once the dictionary build has completed and the dictionary has been inserted into the table (the red rectangle) any subsequent data to enter the table 'respects' the dictionary and is compressed (as denoted by the light blue rectangles).



This slides summarizes the Table Compression features in DB2 LUW.

Once again, please refer the Compression BP to help with your further understanding of Table Compression and it's deployment.

V9.7 Index Compression

- Indexes are composed of many entries that have the form: (*keypart, RID*)
  - Keypart => the table column data that is indexed
  - RID => Record Identifier, the location within the table where the row with the keypart data resides
- The basic premise behind compression is to eliminate redundancy or duplicates
- Index compression works by compressing the index keypart data (Prefix Compression) and by compressing the RID data (RID List Compression)
- Benefits:
  - Smaller indexes via fewer index levels and less index leaf pages
  - Less logical and physical I/Os; Improved bufferpool hit ratio

Index Compression is new V9.7. After table data, index data is likely to be a major consumer of storage in your database.

Let's talk a bit about what this technology is then move on to how to deploy and use it.

Index Compression

- Compression of index objects via
  - COMPRESS clause on CREATE INDEX statement
  - COMPRESS clause on new ALTER INDEX statement
    - Index reorganization required to rebuild the index in compressed (or uncompressed) format
    - 'Offline'/Classic Table REORG and LOAD rebuild indexes so they can be rebuilt in compressed format
- By default, new indexes on a row compressed table will be compressed (except for MDC block indexes and XML meta and paths indexes)
- Table (row) compression is dictionary based
  - Data patterns change, dictionary rebuild through classic reorg may provide better compression
- Index compression is dynamic
  - No need for reorg to achieve better compression

Table compression was introduced in DB2 9.1 (Viper), and enhanced in DB2 9.5 (Viper 2).

Index compression is being introduced in DB2 9.7, extending the value proposition for customers with the Storage Optimization Feature!

Indexes compression defaults to the table compression option, with some exceptions. New COMPRESS option on CREATE INDEX to have control of user indexes.

A new ALTER INDEX statement is introduced to be able to change compression attribute. Reorg of the index (without CLEANUP ONLY) required to change state.

Index Create and Rebuild Considerations

- NOTE:
  - Classic or 'Offline' Table Reorg rebuilds all the indexes that are defined on the table being reorganized
  - Index reorganization is a shadow copy approach
    - Dual storage is required throughout the processing
    - There is no ability to built the index shadow copy anywhere but within the tablespace that the index being reorganized resides in (use LARGE tbspcs)
  - As of V9.5 index create/rebuild parallelized without have to set DBM CFG parameter INTRA_PARALLEL to YES
  - Index reorgs now have detailed monitoring via db2pd (see upcoming slides)

After upgrading to V9.7, in order to get existing indexes compressed, index reorg or index rebuild/recreate is required once (note an index reorg is essentially an index rebuild).

Here are some interesting considerations to note with regards to index rebuild (reorg and recreate).

Index Reorg Monitoring

- Index reorg had very little monitoring beyond db2diag.log and admin log messages
- New option: *"db2pd –reorg index"*

```
Index Reorg Stats:
Retrieval Time: 11/25/2009 18:10:08
TbspaceID: 2       TableID: 4
TableName: T1
Access: Allow write
Status: Completed
Start Time: 11/25/2009 18:09:35   End Time: 11:25/2009 18:10:00
Total Duration: 00:00:25
Prev Index Duration:  -
Cur Index Start: 11/25/2009 18:10:00
Cur Index: 1          Max Index: 1          Index ID: 1
Cur Phase: 3        (Catchup)   Max Phase: 3
Cur Count: 0                  Max Count: 0
Total Row Count:   3214
```

•Best Practices

Save new db2pd reorg index output

This will show the number of indexes and number of rows in the table at time of last index reorg

The timestamps can be used to correlate the data with the history file

Interpreting "db2pd –reorg index" Output

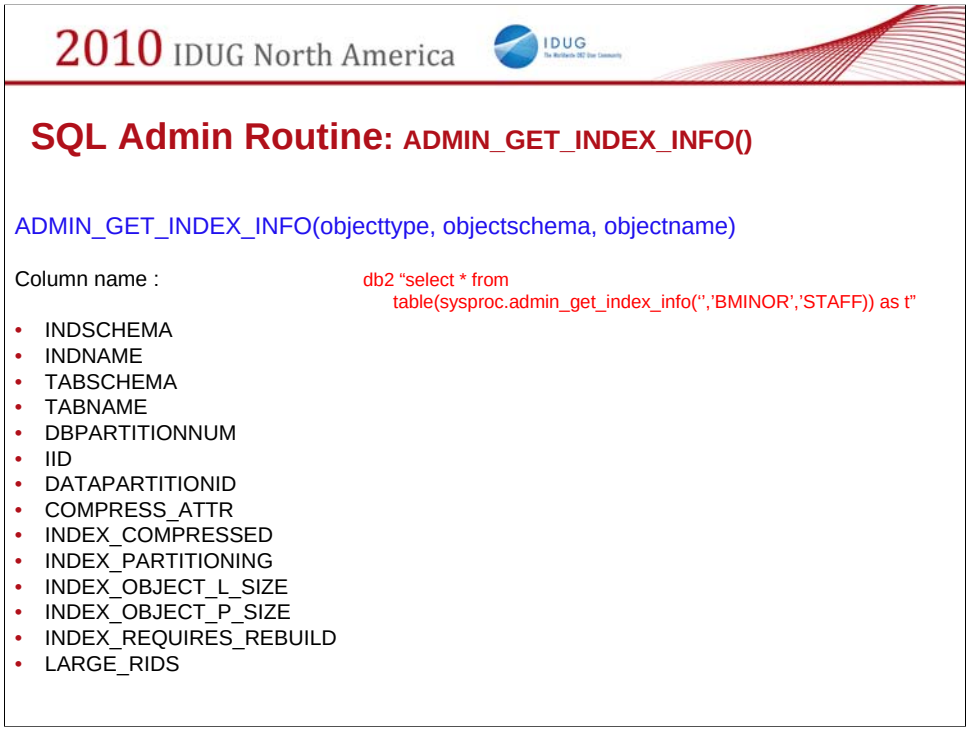
- Compare Cur Phase vs Max Phase
  - This will give you an indication of how far along the reorg of the current index is
- Compare Cur Index Start vs Retrieval Time
  - To get the current elapsed time for this index
  - Compare this result to Prev Index Duration if it is non-zero. This will give you an indication of how much longer it will take to complete the processing for the current index
- Compare Cur Index vs Max Index
  - This will tell you how many indexes are left to reorganize.
  - As a rule of thumb, each index will take approximately Prev Index Duration length of time
- Compare Cur Count vs Max Count
  - This will give you an indication of how far along the processing of the current phase is

Let's walk through an explanation of the various metrics and look at how one uses this information.

Index Compression – Monitoring and Stats

- Two new columns added to SYSIBM.SYSINDEXES and SYSIBM.SYSINDEXPARTITIONS
  - COMPRESSION
  - PCTPAGESSAVED
- Two new table functions (see upcoming slides)
  - ADMIN_GET_INDEX_INFO
  - ADMIN_GET_INDEX_COMPRESS_INFO
    - Index compression estimation only through this admin function, no INSPECT support like there is for tables (example next slide)

Returning to Index Compression … let's look at the monitoring capabilities and statistics that are available to help you understand the effectiveness of Index Compression.



New administrative table function for indexes completely analogous to the one that exists for tables. Here we can an example of how one invokes this table function along with all the relevant information that can be obtained.

Index Compression Estimation

- ADMIN_GET_INDEX_COMPRESS_INFO()
  - Compression attribute/status
    - COMPRESS_ATTR, as defined by DDL
    - INDEX_COMPRESSED, actual physical status
  - Compression statistics
    - COMPRESS_ATTR = 'N', will estimate percent of pages saved if index were to be compressed
    - COMPRESS_ATTR = 'Y', will report actual statistics from the SYSCAT views

Table compression is assisted by the ADMIN_GET_TAB_COMPRESS_INFO table function, which will estimate compression.

Index compression will also come with integrated tooling, specifically the new ADMIN_GET_INDEX_COMPRESS_INFO table function.

Parameter lists are different – for various reasons, indexes have their own schema and name, while there is also no "report/estimate" option – it's implicit.

Just as with table compression, the definition of an index may be compress YES or NO, but the actual physical characteristic of the index may be the same or different.

The ADMIN_GET_INDEX_COMPRESS_INFO function will report the COMPRESS status – for DDL and actual physical state of the index.

If the index is not yet in a compressed format, DB2 will go through the index to calculate and estimate what the compression savings would be.

If the index is in a compressed format, DB2 will report the compression statistics found in the catalog view.

Index Compression Savings:
ADMIN_GET_INDEX_COMPRESS_INFO

SQL Table Function: ADMIN_GET_INDEX_COMPRESS_INFO(objecttype, objectschema, objectname, dbpartitionnum, datapartitionid)

- **Compression Estimation: Example**

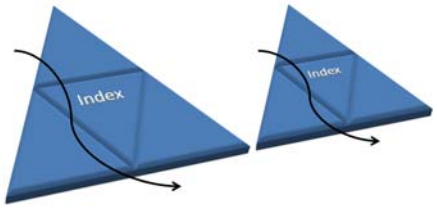
```
SELECT compress_attr, iid, dbpartitionnum, index_compressed,
       pct_pages_saved, num_leaf_pages_saved
FROM TABLE(sysproc.admin_get_index_compress_info('', 'S', 'T1', 2, 3)) AS t

COMPRESS_ATTR IID DBPARTITIONNUM INDEX_COMPRESSED PCT_PAGES_SAVED NUM_LEAF_PAGES_SAVED
------------- --- ------------- ---------------- --------------- --------------------
            N   1             2                N              50                  200
            N   2             2                N              45                  150
```

This is an example of how one goes about using the admin_get_index_compress_info SQL admin function to estimate index compression savings. In this example the values in the “INDEX_COMPRESSED” column are “N”, which indicates that the indexes are not yet compressed such that that the values returned for “PCT_PAGES_SAVED” represent an estimation of compress savings one would get for these indexes if you actually went off and compressed them. Once compressed, “INDEX_COMPRESSED” would show “Y” and “PCT_PAGES_SAVED” would be the actual compression savings that was achieved (it should be close to what was originally estimated!!!).

Index Compression: Performance Attributes

- Fewer index levels
  - Fewer logical and physical I/Os for key search (insert, delete, select)
  - Better buffer pool hit ratio
- Fewer index leaf pages
  - Fewer logical and physical I/Os for index scans
  - Fewer splits
  - Better buffer pool hit ratio
- Tradeoff
  - Some additional CPU cycles needed for compress and decompress operations
    - 0-10% in early measurements
  - Typically outweighed by reduction in I/O resulting in higher overall throughput

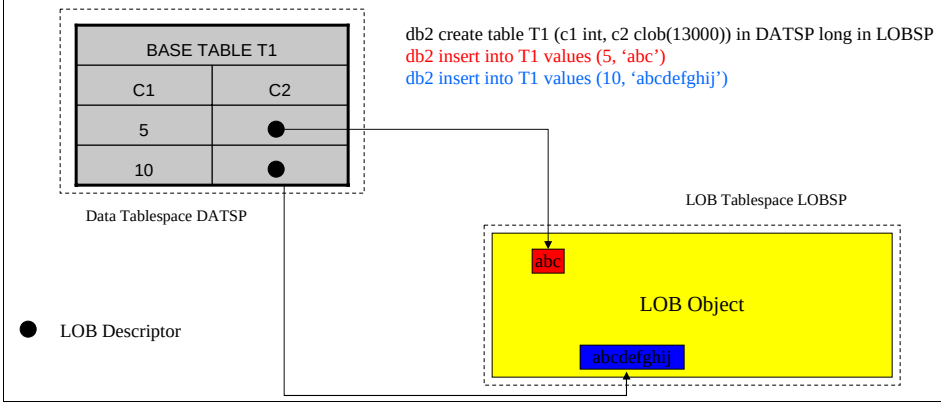


Having compression techniques enabled results in a few things. Imagine the triangles representing index structures – and as you can see you have shorter depth triangles (less intermediate index nodes – few index levels). This means fewer logical and physical I/Os for search and a better buffer pool hit ratio as well. You will potentially have fewer index leaf pages and even splits too. There is a CPU cost associate with splits, so savings there offset the CPU for the compression for example.

But there is a CPU tradeoff. 0-10% in early measurements, but this is typically outweighed in I/O.

LOB Storage

- Prior to V9.7 LOB is stored only in an auxiliary storage object i.e. not in the data record
- Only a descriptor is stored in the data record i.e. a locator to where LOB data actually resides
- LOB storage is allocated ('chunked out') in potential units of size 1KB,2KB,4KB,…,64MB



We have looked at tables and indexes and how to optimize their storage with comprssion. Let's now talk about another data object in the database which obviously consumes storage, LOBs or Large Objects (i.e. BLOBS, CLOBS, DBLOBS).

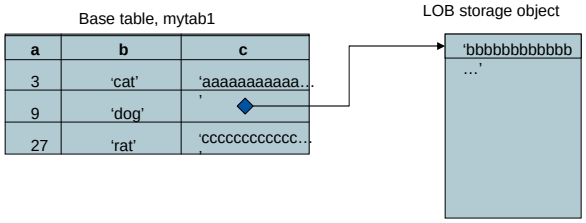
Let's look at how LOBs have been traditionally stored in the database. (Incidentally for a more comprehensive overview of LOBs please come to my session C14 "DB2 LUW LOBs – Past, Present, Future".

V9.7 LOB Inlining

- Instead of strictly storing LOB data in the LOB storage object, the LOBs, if sufficiently sized, can be stored in the formatted rows of the base table
- Dependant on page size, the maximum length a LOB can be to qualify for inlining is 32 669 bytes
- LOB inlining is analogous to XML inlining for XML data

Example:

```
create table mytab1 (a int, b char(5), c clob inline length 1000)
```



Let's now introduce the concept of inlining and how it applies to LOBs in V9.7 and what benefits are associated with this feature.

Rows 1 and 3 contain inlined LOB data

Inlining Benefits

- If a table possesses LOB data that can be inlined, there are considerable benefits with respect to performance and storage use
- *Performance*
  - Whenever LOBs are inserted/retrieved, a disk I/O cost is incurred each time since this data is not buffered (unlike with base table data)
  - When inlined, this I/O cost is reduced since this data get buffered along with the base table data they are inlined with
- *Storage*
  - Storage allocated to the storage object is reduced by inlining XML/LOB data in the base table (though base table storage increases)
  - Inlining small XML/LOBs can result in a noticeable decrease in net total storage since the decrease in storage size is greater than the increase in base table storage size
  - XML/LOBs inlined within the base table data can be compressed

Inline: to move column data that does not reside within the data row into the data row.

Considerations for LOB Inlining

- Good candidates:
  - Fit within a specified data page size (4KB, 8KB, 16KB, 32KB)
  - Frequently accessed
  - Not already compressed
- Guidance: Administrative Table Functions
  - *ADMIN_EST_INLINE_LENGTH(<column name>)*
    - Returns an estimate of the inline length required to inline the data, a negative value if data cannot be inlined, or if data already inlined, the actual inline length is returned
  - *ADMIN_IS_INLINED (<column name>)*
    - Will report whether the LOB/XML documents in a column are inlined or not

Example: select admin_is_inlined(xml_doc1) as IS_INLINED, admin_est_inline_length(xml_doc1) as EST_INLINE_LENGTH from tab1

```
IS_INLINED  EST_INLINE_LENGTH
----------  -----------------
1           292
0           450
0           454
```

There are two new administrative table functions that you can leverage to help you determine an appropriate inline length for your LOB data.

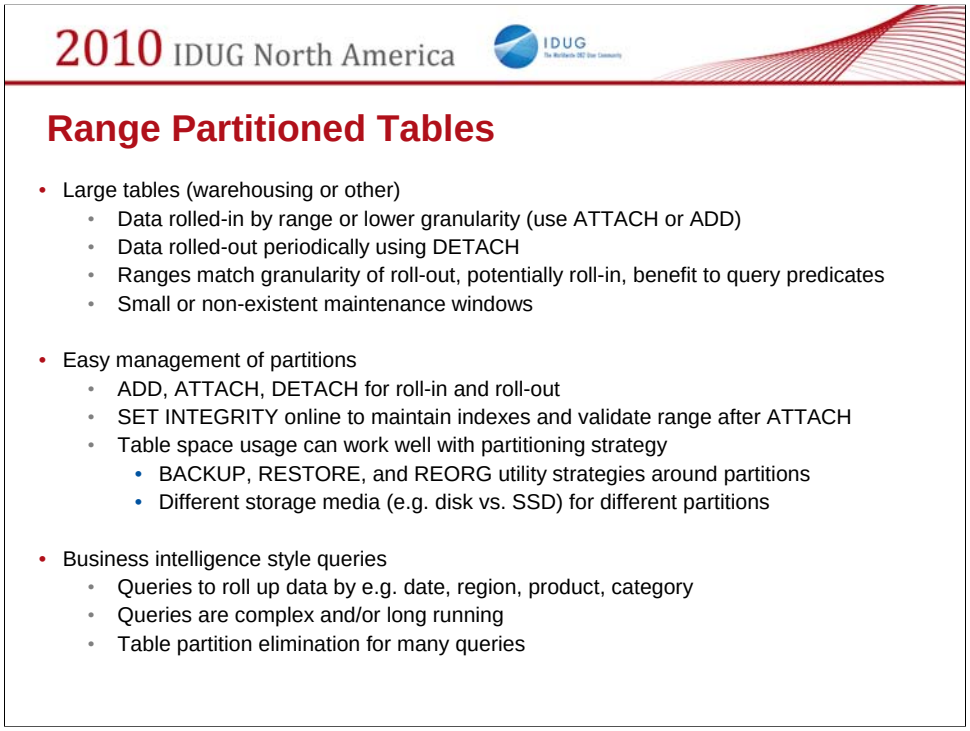
Data Lifecycle Management (DLM)

- In general, DLM is the practice or management of data from the point it is created and stored in the database until the time it is no longer needed and can be removed
- As databases continue to increase in size and at an ever increasing rate, it is becoming more and more important to decide what data is mission critical and most performance sensitive i.e. classify data by importance and priority (all data cannot be saved forever; data access performance cannot be maintained across growing data sets)
- As result, there is usually a time element and hence a lifecycle associated with data
- Correspondingly there are DB2 database features that can be leveraged to help facilitate the organization and management of data throughout it's life
- DLM will be discussed in the context of
  - Range Partitioned Tables
  - MDC Tables
  - 'Space' reclaim: Tables and table spaces

Obviously data cannot be kept forever as this would require ever increasing and unlimited storage capacity. Keep too much data increase administrative overhead and affects performance (more data to process/search through, more data to backup and store, ….)

With big databases (data sets), Data Lifecyle Management becomes essential. At the 30000ft level this is the practice of only keeping business critical data active and bascially archiving or removing data that no longer serves a business need.

Range Partitioned Tables are ideally suited for this purpose especially when dealing with very large Data Warehouses.



Let's discuss the details behind how RPTs are used to facilitate DLM. The primary concepts are "Data Roll-In" and "Data Roll-Out".

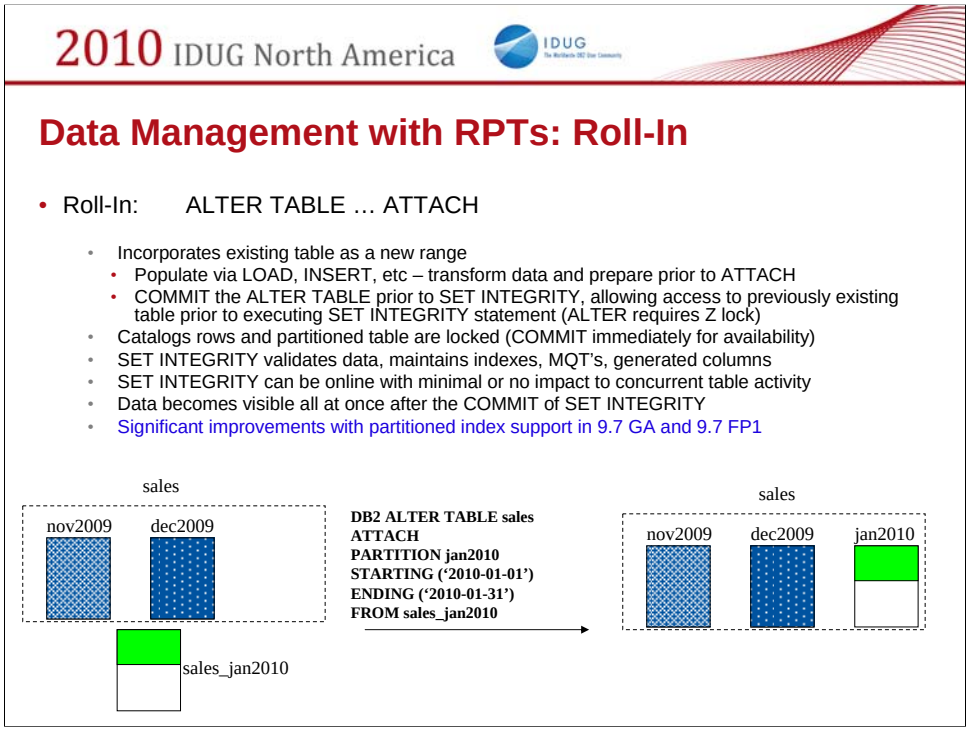
Value Proposition:

Scalability (Large Tables): RPTs can be virtually unlimited in size

Roll-in and Roll-out of ranges: easy addition and easy removal of table partitions with no data movement

Improved query performance through SQL Optimizer partition elimination

Granularity of Utility Maintenance



The most exciting feature within table partition is enhanced roll-in and roll-out. We have created two new operations to accomplish this.

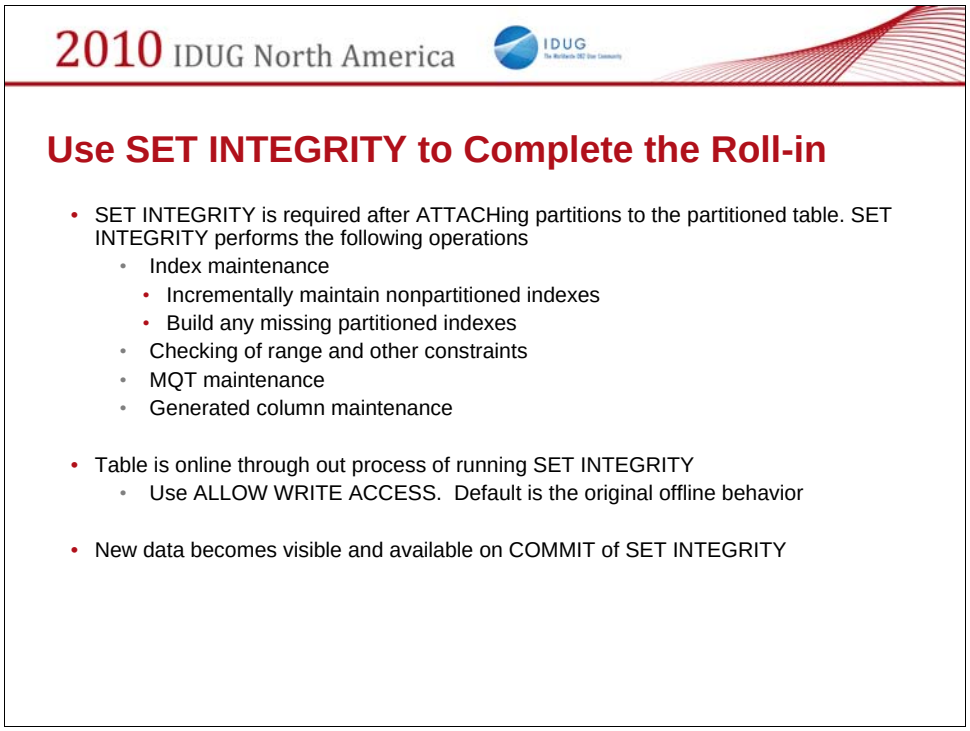
ALTER TABLE ... ATTACH takes an existing table and incorporates it into a partitioned table as a new range. Authority required for ATTACH:

- For the target table:
  - ALTER + INSERT
- For the source table, one of the following:
  - SELECT (table) and DROPIN (schema), or …
  - CONTROL privilege, or …
  - SYSADM or DBADM authority

ALTER TABLE ... DETACH is the inverse operation. It takes one range of a partitioned table and splits it off as a stand alone table. Authority required for DETACH:

- For the source table:
  - ALTER + SELECT + DELETE
- For the target table, one of the following:
  - SYSADM or DBADM or …
  - CREATETAB (database) and USE (tablespace) as well as one of
  - IMPLICIT_SCHEMA
  - CREATEIN

The key point about these operations is that there is no data movement. Internally, they are mostly just manipulating entries in the system catalogs. This means that the actual ATTACH or DETACH operations are very fast - on the order of a few seconds at most.



Without an exception table, any violation will fail the entire operation. Recommendation: provide an exception table for SET INTEGRITY

Example:

```
SET INTEGRITY FOR sales ALLOW WRITE ACCESS,
sales_by_region ALLOW WRITE ACCESS
IMMEDIATE CHECKED INCREMENTAL
FOR EXCEPTION IN sales USE sales_ex;
```

New behaviors specific to XML data with use of Exception Tables

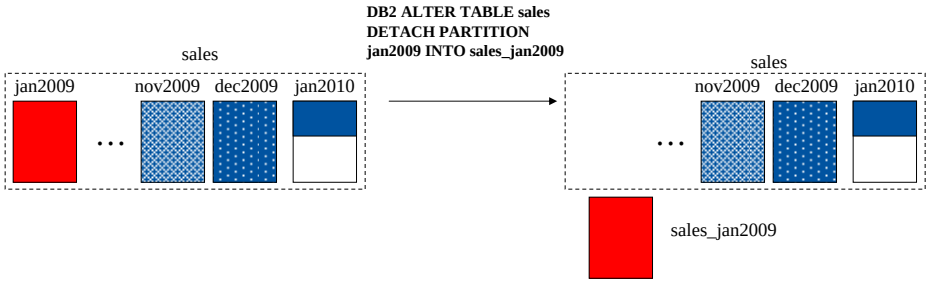
When an exception table is specified, it stores rows that violate constraints in the tables being checked. The table is taken out of set integrity pending state even if errors are detected. A warning message to indicate that one or more rows have been moved to the exception table is returned (SQLSTATE 01603).

The rows are moved out of the target table and into the exception table via DELETE and INSERT

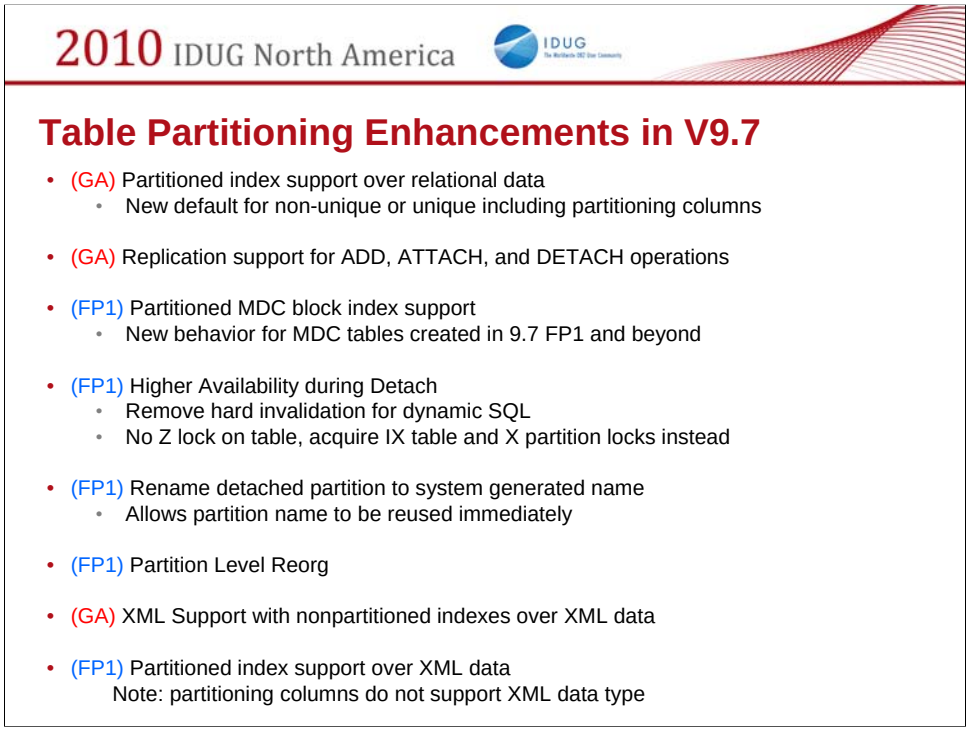
A new constraint violation type code 'X' in the message column of the exception table is introduced for SET INTEGRITY to denote an XML values index violation.

Data Management with RPTs: Roll-Out

- Roll-Out: ALTER TABLE … DETACH
  - An existing range is split off (detached) as a standalone table
  - Data become invisible when partition is detached
  - Catalogs rows and partitioned table are locked (COMMIT immediately for availability)
  - Significant availability improvements in 9.7 FP1 make it less intrusive to concurrent access



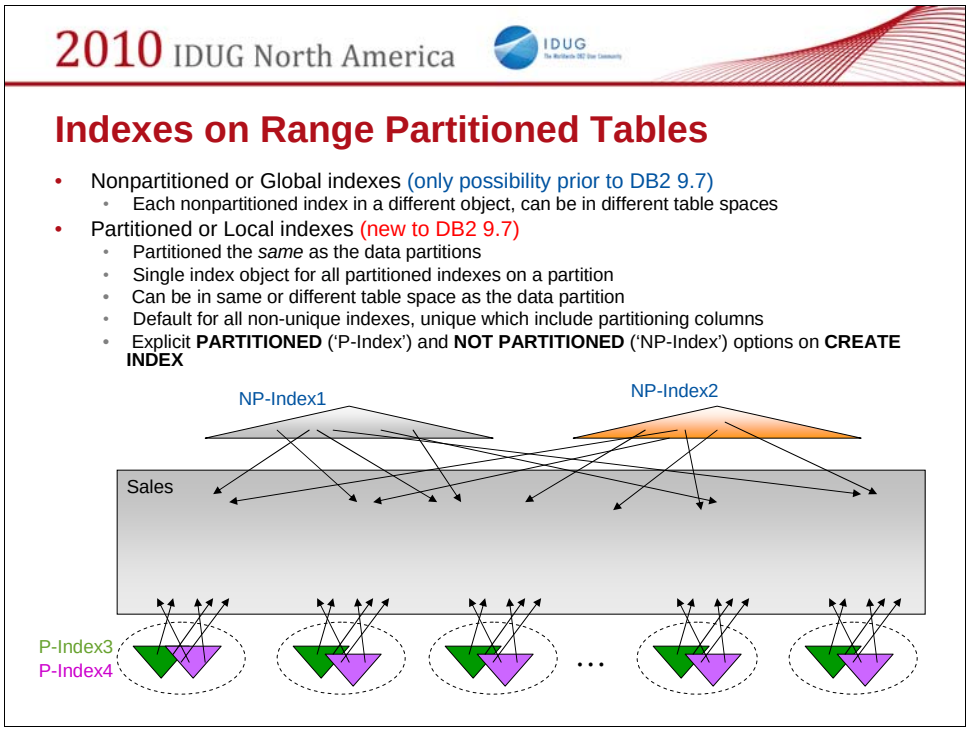
Roll-out with Range Partitioned Tables.



The table partitioning feature in DB2 9 provides a number of advantages, particularly to the data warehouse (DW) applications. DW systems will benefit from easier roll-in/roll-out of data and better query execution performance. DW systems currently using union all views are particularly well suited to make use of table partitioning.

The table partitioning enhancements in DB2 9.7 include partitioned indexes, higher availability during DETACH, partition level reorganization support on both data and indexes, XML columns, etc.

Some of these enhancements are provided in DB2 9.7 GA, while most have been delivered as part of DB2 9.7 FP1.



UNIQUE IDNEX:

A unique index (and therefore unique or primary key constraints being enforced using system generated unique indexes) cannot be partitioned unless the index key columns are a superset of the table partitioning key columns. That is, the columns specified for a unique key must include all the columns of the table partitioning key (SQLSTATE 42990).

DEFAULT:

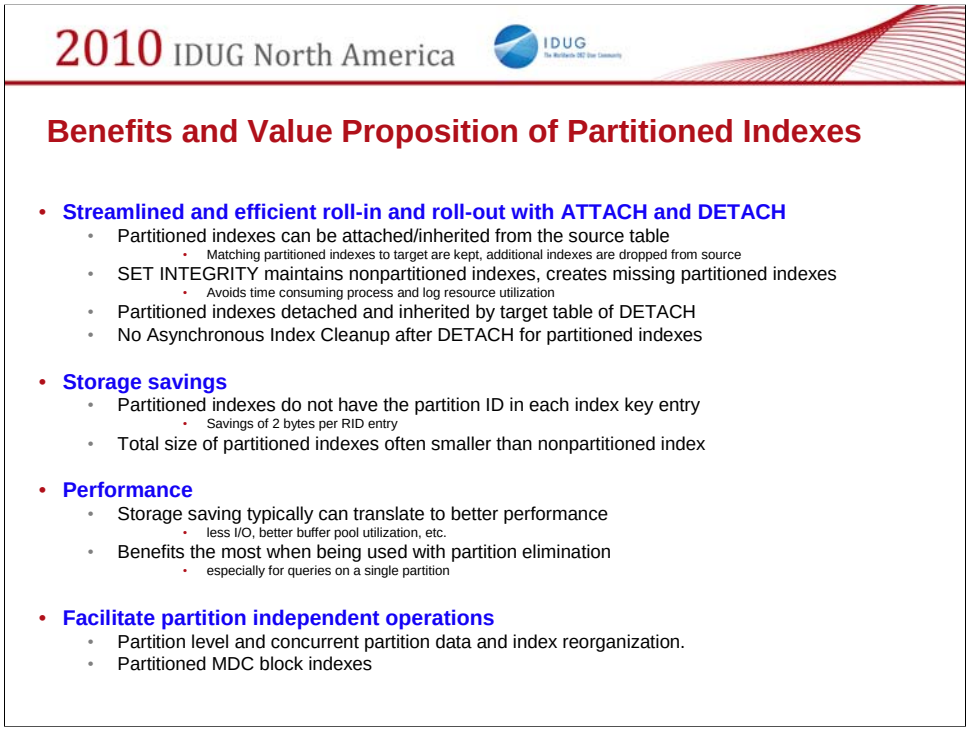
When the table is partitioned and CREATE INDEX does not specify either PARTITIONED nor NOT PARTITIONED keywords CREATE INDEX will create a partitioned index by default unless:

1. an unique index is being created *and* the index key does not include all the table partitioning key columns

2. a spatial index is being created

Index over XML data:

User created partitioned indexes over XML data only supported in 9.7 FP1.



Streamlined roll-in/roll-out with ATTACH/DETACH

Partitioned index can be attached along with the data

Partitioned index on the source can be kept and logically converted to the partitioned index on the newly ATTACHed partition.

Set Integrity doesn't need to maintain partitioned indexes

Strictly speaking, this is only true when the source table has all the indexes pre-created before ATTACH to match all the partitioned index on the target *(Best Practices).*

Partitioned index will be detached along with the data

In another word, they can be inherited by the target table of DETACH

No AIC is necessary after DETACH for partitioned indexes

The major drawbacks for partitioning indexes is it losts order for some queries when the partitioning column is not the leading column of index keys.

The indexes can not help the order requests in this case and extra sorts are required.

Reorg Table and Indexes

- REORG TABLE for a partitioned table is always offline
- REORG INDEXES ALL on a partitioned table is always offline
- REORG INDEX reorganizes a nonpartitioned index, supporting all access modes

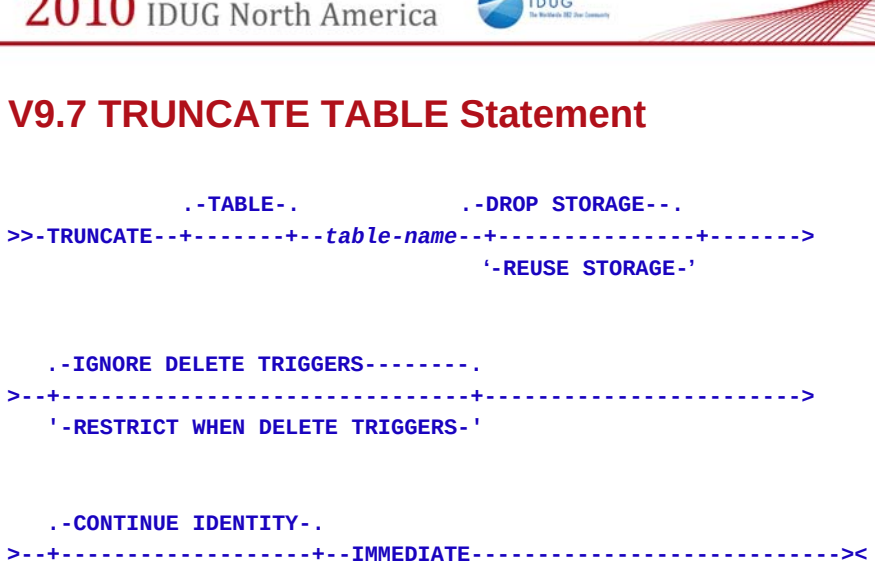
- Partition level reorg table and indexes are available in 9.7 FP1
  - REORG TABLE <t-name> **ON DATA PARTITION** <part-name>
    - **ALLOW NO/READ ACCESS** applies to part-name, not the entire partitioned table
  - **Without** nonpartitioned indexes (except xml path), the ALLOW READ ACCESS mode is the default behavior with full read/write access to all other partitions
  - **With** nonpartitioned indexes (except xml path), ALLOW NO ACCESS is the default and only supported mode.

  - REORG INDEXES ALL FOR TABLE <t-name> **ON DATA PARTITION** <part-name>
    - **ALLOW NO/READ/WRITE ACCESS** applies to part-name, not the entire partitioned table
    - **ALLOW WRITE ACCESS** is *not* supported for MDC tables (w/ mdc block indexes).

- *Concurrent partition reorg (TABLE and INDEXES ALL) supported when there are no nonpartitioned indexes and ALLOW NO ACCESS is specified*

With local indexes (partitioned indexes) reorg – both table reorg and index reorg – can now be executed at a finer granularity when working with RPTs, the partition level. Let's go through the details of reorg behaviour for RPTs in V9.7.

V9.7 TRUNCATE TABLE Statement



Partitioned tables: The table must not be in set integrity pending state due to being altered to attach a data partition. The table needs to be checked for integrity prior to executing the TRUNCATE statement. With DB2 Version 9.7 Fix Pack 1 and later releases, the table must not have any logically detached partitions. The asynchronous partition detach task must complete prior to executing the TRUNCATE statement.

Notes

Table statistics: The statistics for the table are not changed by the TRUNCATE statement.

Number of rows deleted: SQLERRD(3) in the SQLCA is set to -1 for the truncate operation. The number of rows that were deleted from the table is not returned.

Authorization

The privileges held by the authorization ID of the statement must include at least one of the following for the table, and all subtables of a table hierarchy:

DELETE privilege on the table to be truncated

CONTROL privilege on the table to be truncated

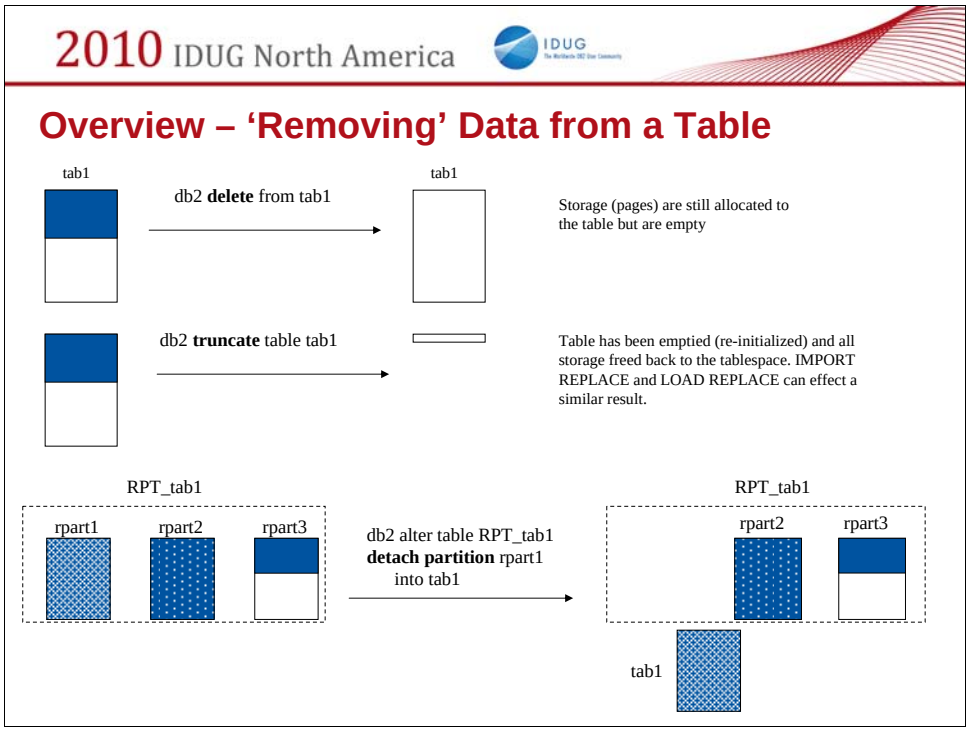
DATAACCESS authority

To ignore any DELETE triggers that are defined on the table, the privileges

TRUNCATE TABLE Statement *(continued)*

- *DROP STORAGE*
  - All storage allocated for the table is released and made available. If this option is specified (implicitly or explicitly), an on-line backup would be blocked.
- *REUSE STORAGE*
  - All storage allocated for the table will continue to be allocated for the table, but the storage will be considered empty. This option is only applicable to tables in DMS table spaces and is ignored otherwise.
- *IGNORE DELETE TRIGGERS*
  - Any delete triggers that are defined for the table are not activated by the truncation operation.
- *RESTRICT WHEN DELETE TRIGGERS*
  - An error is returned if delete triggers are defined on the table
- *CONTINUE IDENTITY*
  - If an identity column exists for the table, the next identity column value generated continues with the next value that would have been generated if the TRUNCATE statement had not been executed.
- *IMMEDIATE*
  - Specifies that the truncate operation is processed immediately and cannot be undone. The statement must be the first statement in a transaction

Detailed expalantion of all the TRUNCATE TABLE options – DROP STROAGE being the default.



Note that the default for TRUNCATE TABLE is DROP STORAGE. Optionally one can specify REUSE STORAGE and this would then provide the same end result as the first depiction i.e. equivalent to a fast delete of a table's contents (DMS tables only).

The Range Partitioned Table (RPT) example demonstrates 'roll-out' – a very quick means of removing data from an RPT. Partition rpart1 is no longer a data partition of RPT RPT_tab1. Rather it has been 'de-linked' from the RPT and exists as a separate standalone table tab1 (which can be subsequently dropped).

Space Reclaim

- After archiving, deleting, or compressing data, space reclamation may be necessary
  - Stale non-critical data is purged from system
  - After compression re-baseline overall storage requirements
- Will now discuss space reclamation features for
  - tables
  - tablespaces

Let's now table about space reclaim – specifically how to reclaim free space that exists within a table or tablespace. This is another fundamental practice for optimizing storage utilization, maintaining performance and facilitating DLM.

Tablespace growth - stating the obvious:

> As data is added to a tablespace it will grow over time and it will continue to grow if the rate of data ingest exceeds the rate at which data is purged from the tablespace
>
> Typically, however the rate of growth fluctuates as a function of time

With storage treated as a shared enterprise resource, there is a need to optimize storage usage: storage is under sized allocated; designed to meet average needs; if all request peek – requests cannot be serviced. (don't have capacity on demand)

Performance another reason to optimize storage

Minimizing storage charge backs

Space Reclaim – Fragmented Tables and Reorg

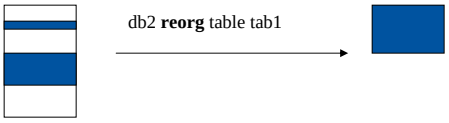


Table data has been compacted and free space within the table freed back to tablespace

Measuring Storage Utilization:
db2 select data_object_p_size from table(sysproc.**admin_get_tab_info**('BMINOR','TAB1'))
db2 select tbsp_used_pages, tbsp_free_pages from table(sysproc.**mon_get_tablespace**('USERSPACE1',0))

Consider the following:

- Data is LOADed into a table through weekly batch processing
- Data is purged monthly from the table via SQL DELETE
  - Even though this table will have free space created within it on a monthly basis, the table (and tablespace occupation) will continue to grow
  - Why? LOAD appends pages to the table

So we have such table state represented by the pictorial on the left – a table with empty blocks or free space interspersed with blocks of table data (the blue).

For the reorg scenario this could be the Classic/ 'Offline' table reorg or the 'Inplace' /Online table reorg; both table reorgs accomplish the same goals, table compaction and truncation of free space back to the table space (note that Inplace table reorg has the option NOTRUNCATE which would compact the table but leave all the collected free space still allocated to table as a contiguous empty block of pages at the end of the table).

Table Storage and Space Reclaim

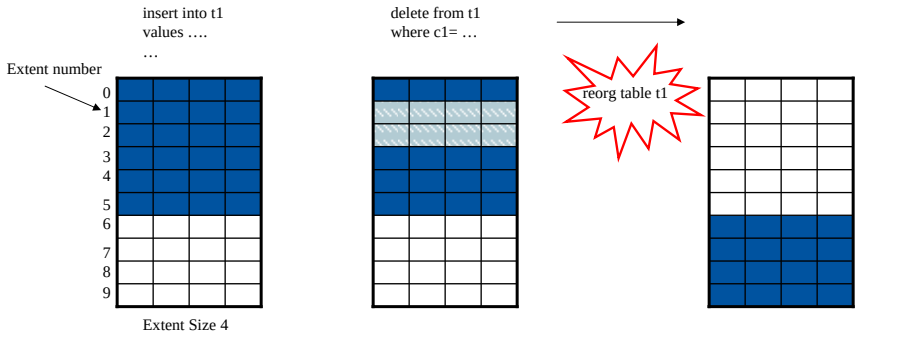


Table t1 occupies six extents (0 to 5) in this tablespace and the table is fully populated. Extents 6,7,8,9 do not belong to any object yet.

Table t1 has data deleted from it such that two of it's extents (1 and 2) are now empty. These empty extents still belong to table t1 however. Tablespace consumption has not changed (6 extents in use)

Table t1 is reorged to shrink the table and reclaim it's empty extents back to the tablespace. The table now only occupies four extents in the tablespace. Note that table reorg is a shadow copy approach so the location of the table has moved within the tablespace.

Let's take a look at space reclaim (so table reorg) from the perspective of the tablespace storage.

Read left to right.

This is a tablespace with extentsize 4 (so 4 rectangles per row). Blue denotes extents allocated to a table – dark blue mena sthe extents have data in them light blue the extents are empty.

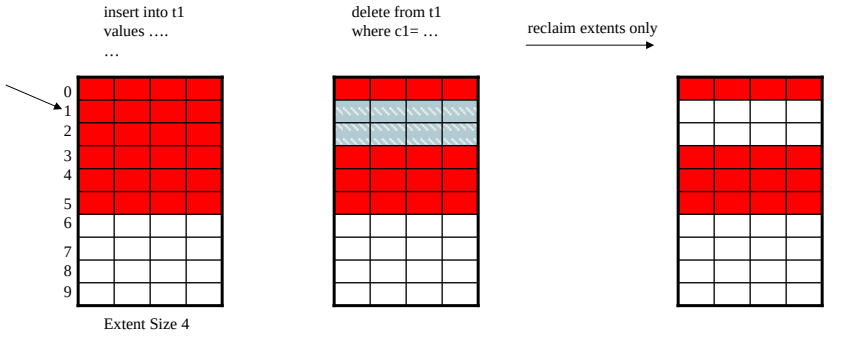
Storage for MDC Tables

- Storage in an MDC table is tracked through a 'block map': which extents have data and which don't (block == extent)
- When a block is emptied, the storage remains with the table and is available for later reuse
- Prior to V9.7, only Classic/'Offline' table reorg can free this storage
  - Does not allow write to the table
  - Shadow copy approach, approx. double the storage for the duration of the processing
  - Takes time as whole table is processed

Make sure 'sparse' is understood – empty blocks versus data pages which are predominantly empty (but not empty)

Contrast to reorg of an MDC table.

MDC Table Space Reclaim



MDC Table t1 occupies six extents (0 to 5) in this tablespace and the table is fully populated. Extents 6,7,8,9 do not belong to any object yet.

MDC Table t1 has data deleted from it such that two of it's extents (1 and 2) are now empty. These empty extents still belong to table t1 however. Tablespace consumption has not changed (6 extents in use)

Extents 1 and 2 are freed quickly online back to the tablespace and are available for any further tablespace growth.

In this case, let's look at space reclaim within an MDC table from the perspective of storage at the tablespace level (analogous to prior reorg table slide).

Read left to right.

V9.7 Sparse MDC Table Space Reclaim

- If MDC table has empty blocks, these can be reclaimed without performing a conventional table reorg
- New option on reorg table command to not reorg this table but reclaim these empty blocks/extents

```
REORG TABLE--<mdc-table-name>--RECLAIM EXTENTS ONLY-- . . .
```

- This storage is freed back to the tablespace and hence is available for use by any object in the tablespace
- Storage is freed incrementally during the processing
- During it's processing, this new capability allows concurrent write to the table and occurs with a minimum amount of time and resource
- Can be invoked at the data partition level
- Amount of space that can be reclaimed is reported through the column RECLAIMABLE_SPACE in the ADMIN_GET_TAB_INFO_V97 table function

Let's summarize MDC table space reclaim in V9.7.

REORG COMMAND as of V9.7

```
>>-REORG-------------------------------------------------------------------------------------->
>-+-TABLE--table-name-------------------------| Table Clause |-------------------------+--->
  |                                                                                       |
  +---+-INDEXES ALL FOR TABLE--table-name-------------+------------| Index Clause |-------+
  |   |-INDEX--index-name--+----------------------+--|                                    |
  |                        '-FOR TABLE--table-name-'                                       |
  |                                                                 |-ALLOW WRITE ACCESS-|  |
  +-TABLE—-mdc-table-name—-RECLAIM EXTENTS ONLY--------------+-------------------+-----+
                                                            |-ALLOW READ ACCESS--|
                                                            |-ALLOW NO ACCESS----|

>----+-----------------------------------+------------------------>
     |--| Table  Partitioning Clause |---+

>----+-----------------------------------+------------------------><
     |--| Database Partition Clause |----+
```

Table Partitioning Clause

>--ON DATA PARTITION--*partition-name*-->-

For partitioned tables, specifies a single data partition to reorganize.

Here is what the REORG command now looks like as v97. New data partition option and new option for ‘reorganzing’ MDC tables.

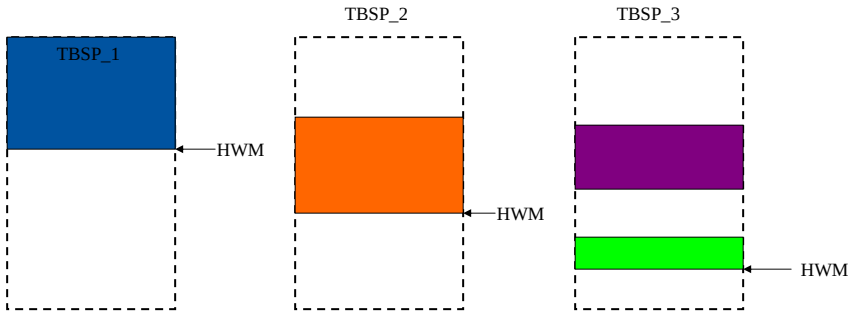
Tables - Space Reclaim

Table Type	CLASSIC OR 'OFFLINE' TABLE REORG	INPLACE OR 'ONLINE' TABLE REORG	SQL TRUNCATE	IMPORT OR LOAD REPLACE
'REGULAR'	✓	✓	✓	✓
MDC	* ✓	✗	✓	✓
RANGE PARTITIONED	✓	✗	✓	✓

NOTE:
Table reorganization of LOB/LONG/XML data can *only* be done via the Classic/'Offline' table reorg *and* requires specification of the LONGLOBDATA clause on the table reorg command.
*In V9.7, use REORG RECLAIM EXTENTS ONLY.

This chart summarizes the methods available for reclaiming space within a table.

DMS Tablespace Free Space (and the High Water Mark)



Notes:
HWM(TBSP_3) > HWM(TBSP_2) > HWM(TBSP_1)

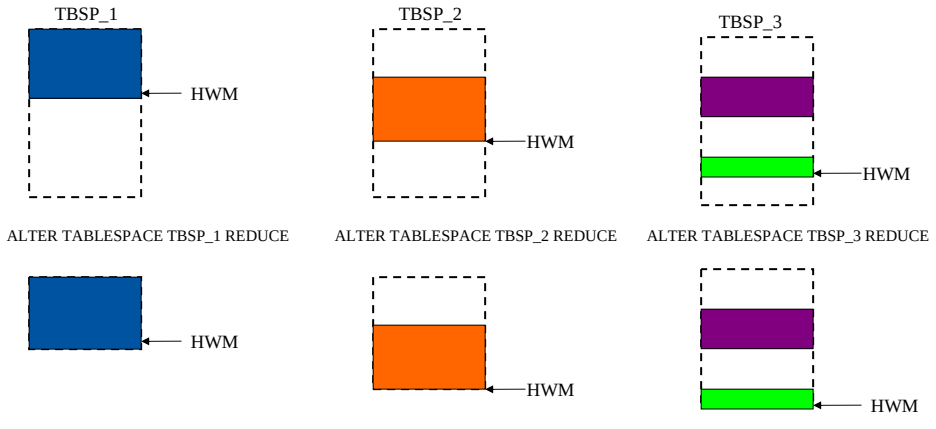
TotalPages(TBSP_3) = TotalPages(TBSP_2) = TotalPages(TBSP_1)
UsedPages(TBSP_3) = UsedPages(TBSP_2) = UsedPages(TBSP_1)

TBSP_1 does not have any free space below it's HWM, TBSP_2 and TBSP_3 do.

Let's now discuss space management for DMS tablespaces.

Three different tablespaces each of the same size and occupied (in this example) with the same amount of data but the location of used extents/pages in each of the tablespaces is different (hence the location of free space within the tablespaces is also different). As a result, the highest occupied page in the DMS tablespace – the Tablespace High Water Mark (HWM) – is different. The HWM for TBSP_3 is higher than that for TBSP_2 which is higher than that of TBSP_1. Note that TBSP_1 does **not** have any free space below it's HWM whereas both TBSP_2 and TBSP_3 do.

Shrinking DMS Tablespaces Prior to V9.7



Notes:
HWM(TBSP_3) > HWM(TBSP_2) > HWM(TBSP_1)

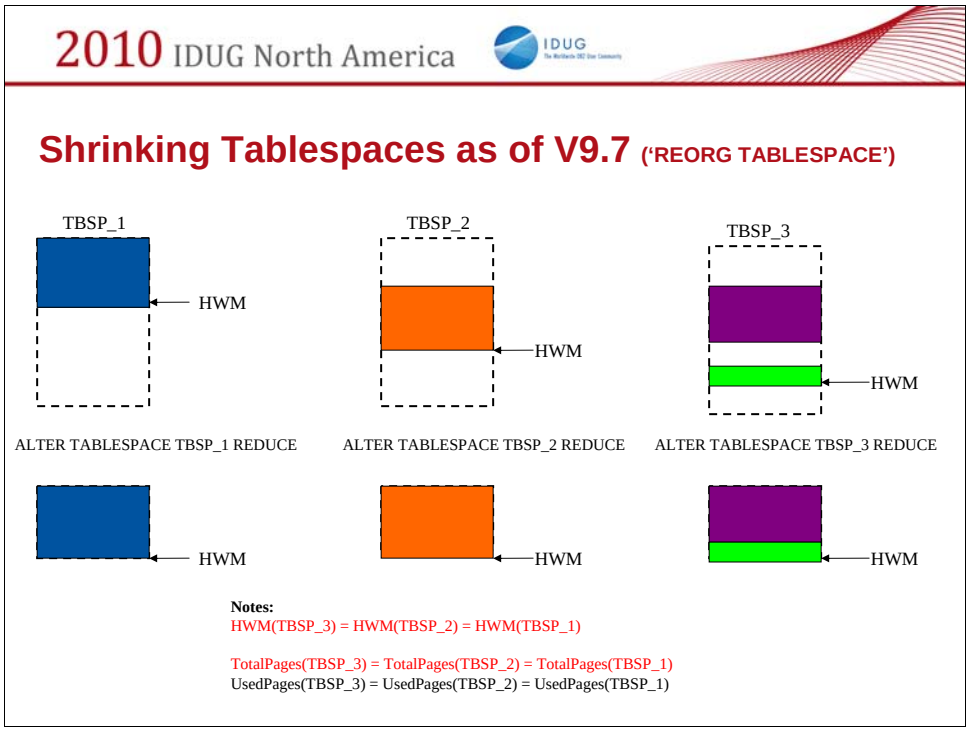
TotalPages(TBSP_3) > TotalPages(TBSP_2) > TotalPages(TBSP_1)
UsedPages(TBSP_3) = UsedPages(TBSP_2) = UsedPages(TBSP_1)

ALTER TABLESPACE REDUCE in V9.5 will only shrink the total space allocated to a tablespace down to the HWM. It cannot move the existing data in tablespace into lower positions in the tablespace where free space may exist.

V9.7 Reclaimable Storage Tablespaces :
Remedy for "HWM Issues"

- High Water Marks will always exist but
  - A HWM can become an 'issue' when there is free space below it *AND* no further growth with the tablespace is expected -> wasted space
- DB2 V9.7
  - All new DMS tablespaces created by default with a reclaimable storage attribute
  - New ALTER TABLESPACE options to lower the HWM and reclaim free space below it by transparently 'shuffling' the contents of the tablespace to lower positions in the tablespace (no db2dart operations)
  - Online to DML and DDL operations – one exception, Backup and Restore
  - Existing tablespaces need to be rebuilt to take advantage of this capability

New to V9.7, new DMS tablespace are created with an new underlying type 2 or 'reclaimable storage' infrastructure.



Continuing with the prior example of three different tablespaces, each with the same amount of data but located in different places within the tablespaces.

Using the new default DMS tablespaces in V9.7, an ALTER TABLESPACE REDUCE command can now, online and asynchronously, reclaim all the free space within a tablespace no matter where it resides – this is effectively a 'reorg tablespace' operation which reduces or shrinks the tablespace containers by compacting the contents of the tablespace.

Reclaimable Tablespace Storage- How?

- For Automatic Storage Tablespaces:

```
ALTER TABLESPACE <name> REDUCE --------------------------|
                                +--<size>--<units>--+
                                +--MAX--------------+
                                +--STOP-------------+
```

- Triggers asynchronous extent movement but iterative in the sense that space is gradually release back to the system
- If no option specified, abide by existing behaviour, viz. reclaim pending delete extents and reduce size of containers down to HWM
- MAX: reclaim as much storage as possible and reduce containers to new HWM
- <size>: to reclaim a user specified amount of space only
- STOP: to stop extent movement at the current point in time reclamation

Here is the command in V9.7 for reclaiming tablespace storage and lowering the tablespace HWM.

Reclaimable Tablespace Storage - Monitoring

Example: MON_GET_TABLESPACE

```
db2 "select substr(tbsp_name,1,15) as tbsp_name,
            tbsp_type,
            tbsp_total_pages,
            reclaimable_space_enabled
    from table(sysproc.mon_get_tablespace('USERSPACE1', 0))"

TBSP_NAME       TBSP_TYPE  TBSP_TOTAL_PAGES RECLAIMABLE_SPACE_ENABLED
--------------- ---------- ---------------- -----------------------
USERSPACE1      DMS                  5000                       1
```

Other columns of interest:

- TBSP_USED_PAGES
- TBSP_FREE_PAGES
- TBSP_USABLE_PAGES
- TBSP_PENDING_FREE_PAGES
- TBSP_PAGE_TOP
- TBSP_MAX_PAGE_TOP

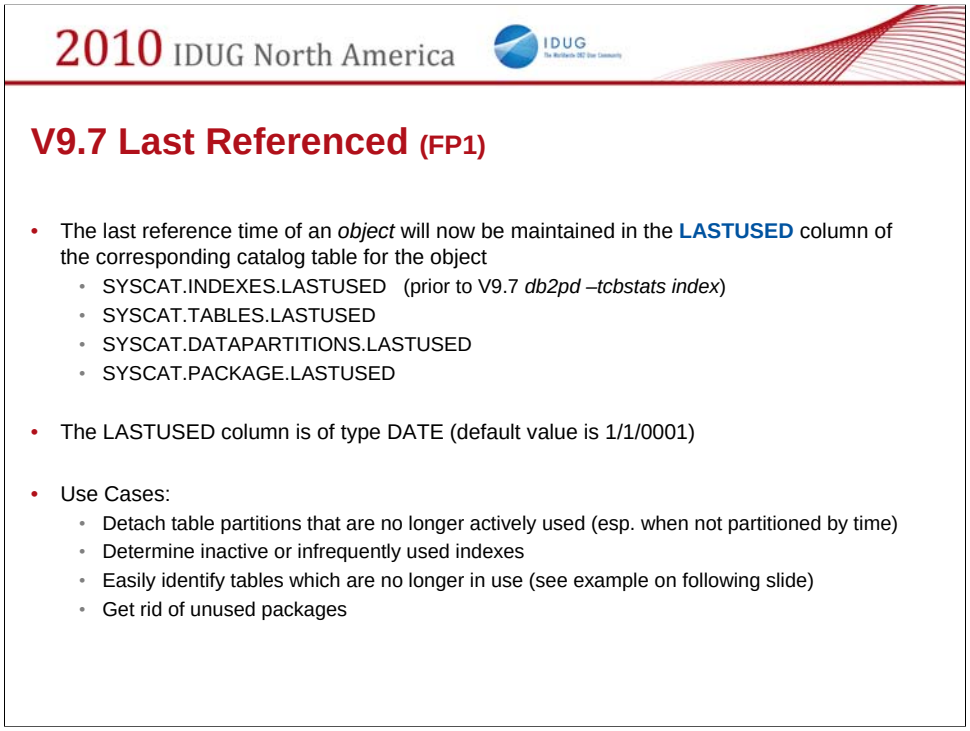
New administrative table function – SQL instead of tablespace snapshot.

Reclaimable Tablespace Storage - Monitoring

Example: MON_GET_EXTENT_MOVEMENT_STATUS

```
db2 "select substr(tbsp_name,1,15) as tbsp_name,
                                   current_extent,
                                      last_extent,
                                num_extents_moved,
                                 num_extents_left,
                                   total_move_time
 from
  table(sysproc.mon_get_extent_movement_status('TBSP_DMS1',0))"
```

New admin table function to assess and monitor tablespace space reclamation.



The last referenced time will be stored in the LASTUSED column (type DATE) of the corresponding catalog table for the object and accessible through the catalog view on the table. Note that the last referenced time is of interest when an object has not been used for an extended period of time (several months for example), hence a date is used rather than a timestamp. Coarser granularity (date rather than timestamp) permits less impact on the system when maintaining the last referenced time.

The LASTUSED column will be updated as follows:

SYSCAT.INDEXES.LASTUSED

The date when the index was used by any DML statement, such as SELECT or MERGE, for searched UPDATE, searched DELETE, or to used to enforce referential integrity constraints. The default value is 1/1/0001.

SYSCAT.TABLES.LASTUSED

The date when the table was used by any DML statements, such as SELECT, MERGE, INSERT, UPDATE, or DELETE, or the LOAD command. The default value is 1/1/0001. For aliases, views and nicknames, the LASTUSED column will remain set at the default. For partitioned tables the value in LASTUSED represents the last time any data partition was used. Refer to the LASTUSED column in SYSCAT.DATAPARTITIONS for the last use of a specific data partition.

SYSCAT.DATAPARTITIONS.LASTUSED

The date when the table partition was used by any DML statements, such as SELECT, MERGE, INSERT, UPDATE, or DELETE, or the LOAD command. The default value is 1/1/0001.

SYSCAT.PACKAGE.LASTUSED

Example:
When Did I Last Use a Table in a Particular Schema?

```
db2 "select substr(tabname,1,15) as tabname,
            create_time,
            lastused
  from syscat.tables where tabschema='BMINOR'"

TABNAME         CREATE_TIME                 LASTUSED
--------------- -------------------------- ----------
DEPARTMENT      2010-01-12-16.04.24.048940 01/01/0001
DEPT            2010-01-12-16.04.24.730248 01/01/0001
SALES_PART      2010-01-12-16.09.29.362305 01/12/2010
SMALLEY         2010-01-15-12.51.51.157979 01/15/2010
MYLOB           2010-01-15-15.04.37.135472 01/01/0001
MYLOB2          2010-01-15-15.05.35.377745 01/15/2010
STAFF3          2010-02-02-19.32.13.564964 01/01/0001
```

Example of last used.

Cast of DB2 Features (as they appeared in order)

- Table Compression
- Index Compression
- LOB Inlining
- Range Partitioned Tables and Local Indexes
- Truncate Table
- MDC Space Reclaim
- Tablespace Space Reclaim
- Last Referenced

Summarize the features we discussed … and there are many more in DB2 V9.7!

Bill Minor
bminor@ca.ibm.com