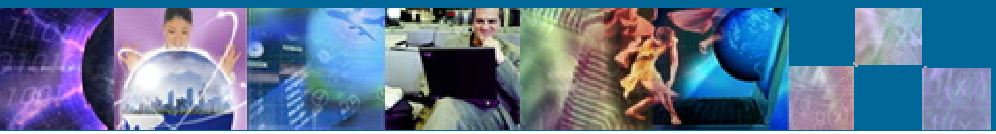


Deep Dive into *Deep* Compression

DB2 for Linux, UNIX, Windows

Chat with the Lab – Sept. 27, 2007

ON DEMAND BUSINESS™

Highlights

- DB2 9 (code named Viper) brought the initial functionality and capability of *Deep Compression*. Significant storage savings and in many cases, performance benefit, is possible. Viper 2 (DB2 9.5) brings additional enhancements to the compression.

- The goals of this presentation are:
  - To provide an overview and roadmap of Deep Compression (Viper & Viper II)
  - Relate best practices and guidelines for deploying compression
  - Discuss storage benefits and performance aspects associated with compression
  - Provide a competitive overview

Agenda

- Introduction
- DB2 9 (Viper) Deep Compression
  - “What” and How”
- Viper 2 (DB2 9.5) Compression enhancements
- Best practices
- References
- Competitive Comparison and Deep Compression Challenge

The ‘Nature’ of Data

- As data continues to grow, the cardinality of the data drops. As it turns out there just are not that many truly "unique" things in the world. Now, they may be unique when used in combination, like DNA, but the basic elements themselves are not all that varied. Consider the Periodic Table of Elements - everything in our world is made up of this rather small set of basic elements in combination. Apply the concept to data, and you find the same is true.

- For instance, there may be about 300M people in the US according to the last census, but there are only approximately 78,800 unique last names, producing very low cardinality, with huge "clumps" in certain name sets. First name is even worse, coming in at about 6,600 unique first names (4,400 for the females, 2,200 for males).

- The names of cities and streets and their "normalized", address corrected adornments (Street, Avenue, etc.) are also very low cardinality.

- Product names, descriptions and attributes ("Dolby Digital") also tend to be highly redundant and low cardinality.

➢ Hence, symbol based compression works even better over very large domains of data, since the data within the domain is statistically not that variant.

There is statistical redundancy in data – Zipf’s Law, Pareto Principle (80:20 rule), …

The Compression Value Proposition

Use less data pages to store table data.

What are the benefits?

- **Consume *less* storage**
- **Consume storage at a *slower* rate**
- **Possibly, *improve* your performance**

D

e

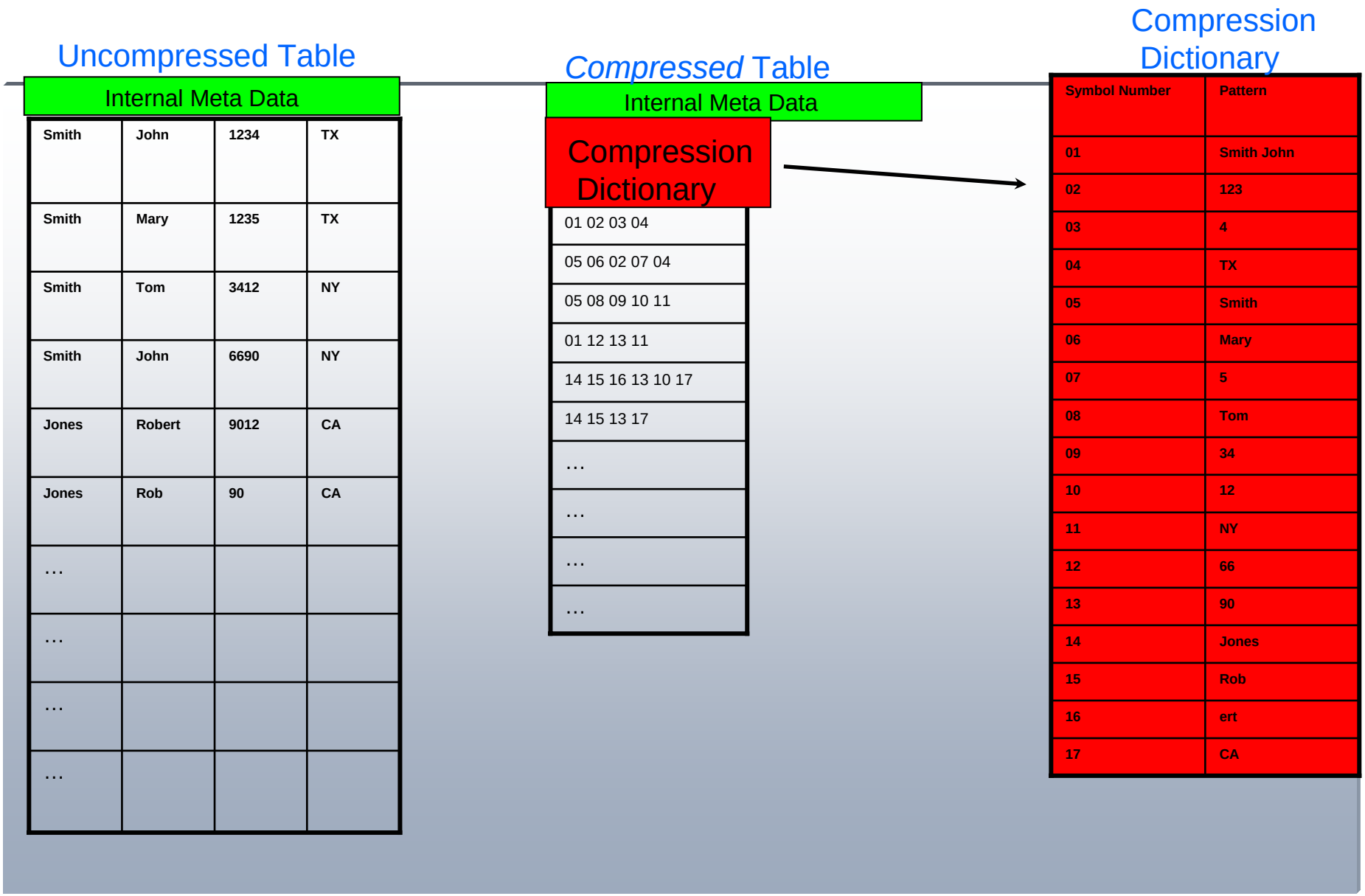
e

p Compression

Deep Compression in DB2 LUW

- ‘Static’ dictionary approach – repeating patterns replaced by smaller symbols
- The compression dictionary is physically stored within the data table – one dictionary per table object (DPF, Range Partition)
- Data exists in compressed format on disk and in memory (the bufferpool)
- User data is compressed in the logs
- There is a CPU cost associated with software compression

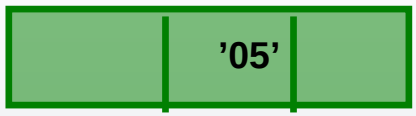
Pictorial Overview of Compressed Tables in DB2 LUW



DB2 Data Row Compression

Uncompressed Row

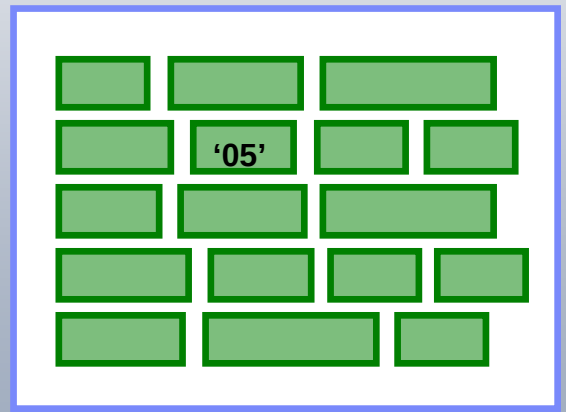
Compressed Row



Common sequences of consecutive bytes in row replaced with 12 bit symbol

Data page with uncompressed rows

Data page with compressed rows



How Do I Compress?

In order to compress data, ***two*** pre-requisites must be satisfied:

1) **The table COMPRESS attribute must be set to YES**
2) **A Compression Dictionary must exist for the table object**

Once *both* step 1) and step 2) are satisfied, all data subsequently populated into the table is subject to being compressed.

STEP 1. Enablement - Table DDL

- **CREATE TABLE <table name> --->**

```
          |---COMPRESS NO---|
     ----+-----------------+---->
          |---COMPRESS YES--|
```

- **ALTER TABLE <table name> --->**

```
       --+-----------------------+---->
         |--COMPRESS--+-YES--+--|
                      |--NO--|
```

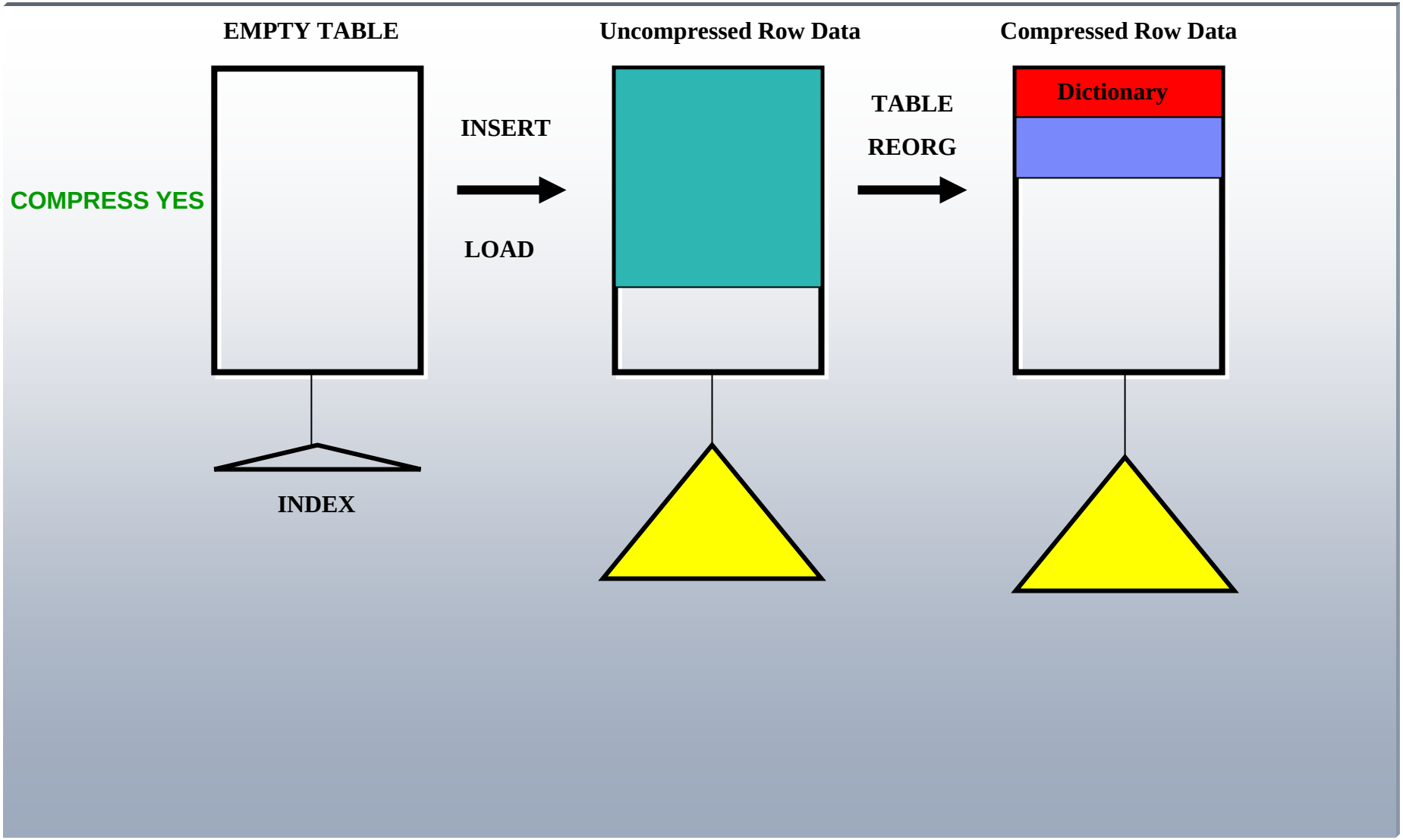
- Compression is *enabled* at the table level via either the CREATE or ALTER TABLE statements
- Compression will be in *effect* once a table dictionary is built

STEP 2. Dictionary Building – Classic or 'Offline' Table REORG

- When the compression dictionary is being built a temporary in-memory buffer of 10MB is required
  - This memory will be allocated from the utilities heap
- All the data rows that exist in a table will participate in the building of the compression dictionary

```
>--REORG--<table name>--+-----------------------+---->
                        '--INDEX--<index name>--'
   .-ALLOW READ ACCESS-.
>--+-------------------+--+---------------+--+-----------+->
   '-ALLOW NO ACCESS---'  '-USE-<tbspace>-'  '-INDEXSCAN-'
                  .-KEEPDICTIONARY---.
>--+-------------+-+------------------+-+-->
   '-LONGLOBDATA-' '-RESETDICTIONARY---'
```

Compression Dictionary Build in DB2 9 (Viper)



Compression Estimator – DB2 INSPECT

- Looks at all the rows of the table data and builds a compression dictionary from it. This dictionary will then be used to test compression against the records contained in the sample.

  INSPECT ROWCOMPESTIMATE TABLE

  Results include:
  - Estimate of compression savings
  - Dictionary size
  - Will insert the dictionary if COMPRESS YES is set
    - Allows for **online dictionary creation/insertion**
    - Future rows inserted/updated are compressed
    - Does not address existing rows (REORG to be used)

DB2 INSPECT – Compression Evaluation

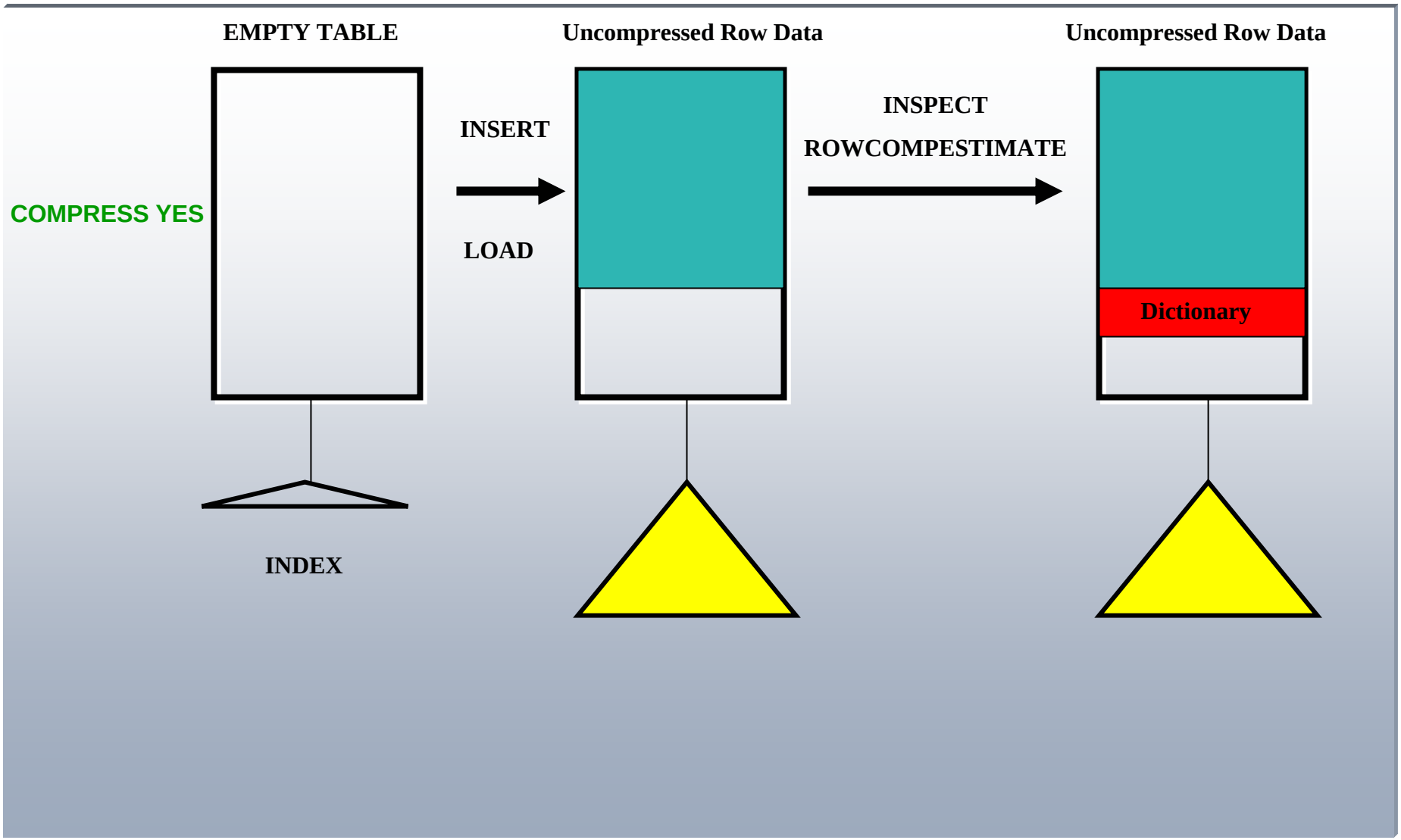
- **INSPECT ROWCOMPESTIMATE TABLE NAME <tbname>**
  SCHEMA <schema> RESULTS KEEP <filename>
- **../sqllib/db2dump/db2inspf <filename> <outfile>**

```
Action: ROWCOMPESTIMATE TABLE
Schema name: BILLM
Table name: EMPLOYEE
Tablespace ID: 2  Object ID: 6
Result file name: emp

    Table phase start (ID Signed: 6, Unsigned: 6; Tablespace ID: 2) : BILLM.EMPLOYEE

      Data phase start. Object: 6  Tablespace: 2
      Row compression estimate results:
      Percentage of pages saved from compression: 66
      Percentage of bytes saved from compression: 66
      Percentage of rows ineligible for compression due to small row size: 0
      Compression dictionary size: 13312 bytes.
      Expansion dictionary size: 10240 bytes.
      Data phase end.
    Table phase end.
```

DB2 9 INSPECT COMPRESSION DICTIONARY BUILD



Row Compression Example

- Create Table that is Eligible for Compression

```
CREATE TABLE Sales COMPRESS YES
```

- Get Representative Data Sample

```
LOAD FROM filesmall OF DEL REPLACE INTO Sales
```

- Creates dictionary on sample data

```
REORG TABLE Sales RESETDICTIONARY
```

- Load respects dictionary

```
LOAD FROM filerest OF DEL INSERT INTO Sales
```

Compression Catalog Information – syscat.tables

Column Name	Data Type	Description
COMPRESSION	Char(1)	B = Both value and row compression are activated N = No compression is activated; a row format that does not support compression is used R = Row compression is activated; a row format that supports compression might be used V =Value compression is activated; a row format that supports compression is used Blank = Not applicable
AVGROWSIZE	Smallint	Average length (in bytes) of both uncompressed and compressed rows in this table; -1 if statistics are not collected
PCTPAGESSAVED	Real	Approximate percentage of pages saved in the table as a result of row compression. This value includes overhead bytes for each user data row in the table, but does not include the space that is consumed by dictionary overhead; -1 if statistics are not collected.
PCTROWSCOMPRESSED	Real	Compressed rows as a percentage of the total number of rows in the table; -1 if statistics are not collected
AVGROWCOMPRESSIONRATIO	Smallint	For compressed rows in the table, this is the average compression ratio by row; that is, average uncompressed row length divided by the average compressed row length; -1 is statistics are not collected
AVGCOMPRESSEDROWSIZE	Smallint	Average length (in bytes) of compressed rows in this table; -1 if statistics are not collected

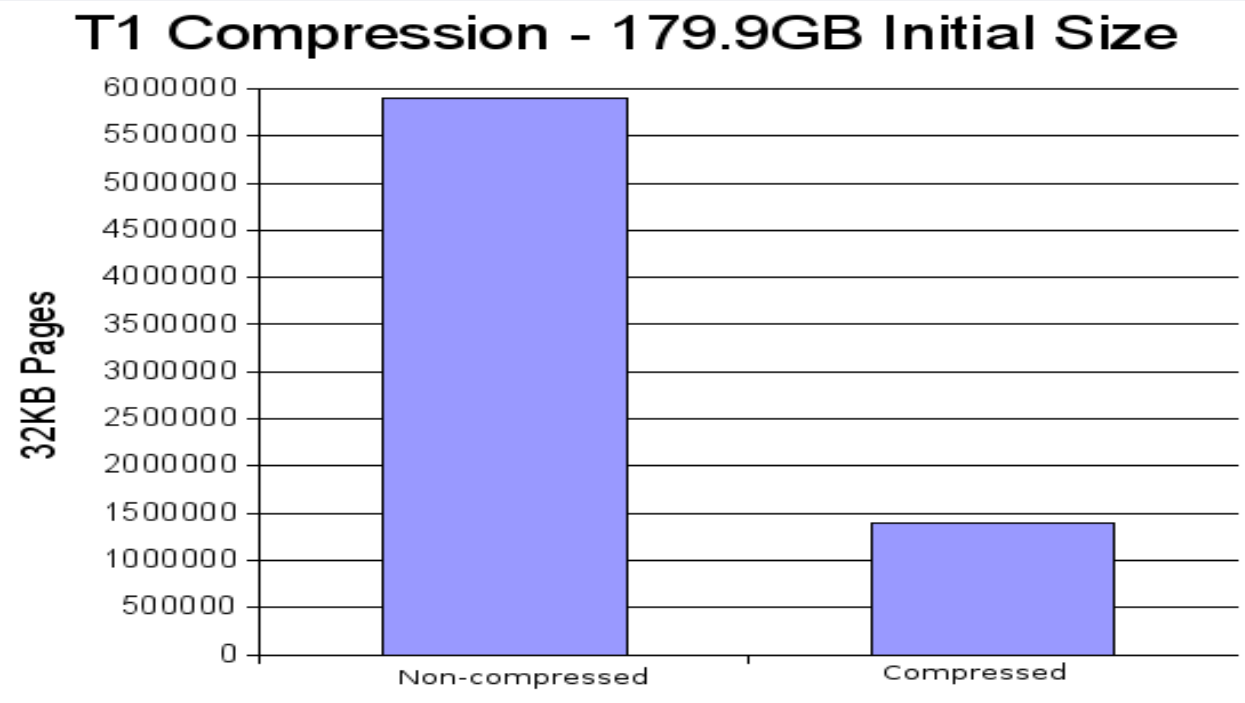
Limitations of Data Row Compression

- The following tables cannot be compressed:
  - Catalog tables
  - Declared global temporary tables
  - System temporary tables
- Row compression is only applicable to rows in data objects
  - Indexes are not compressed
  - Pure XML in separate object – XML data is not compressed**
  - LOBs are stored in separate object – LOB data is not compressed
  - LONGs are stored in separate object - LONG data is not compressed
- Row compression support and table data replication support will not be compatible.
  - DATA CAPTURE CHANGES option not compatible with the COMPRESS YES option

Compression Ratios (Customer Financial Data)

Compression Type	32KB Page Count	Space Required on Disk
No compression	5893888	179.9GB
Row compression	1392446	42.5GB

% Pages Saved: 76.4%



Compression Savings – Customer Data Warehouse

Table	% of Table Sampled	Number of Rows	Sample Size (GB)	Sample Size Compressed (GB)	Compression Ratio	Total Est. Disk Savings (GB)
T1	100	159259747	71.49	13.23	81.5%	58
T2	10	77390826	21.48	5.23	75.7%	162
T3	100	191362309	141.46	36.18	74.4%	105
T4	20	19474261	11.80	3.05	74.2%	43
T5	10	23370458	25.78	8.54	66.9%	172
T6	10	39573969	23.24	6.32	72.8%	169

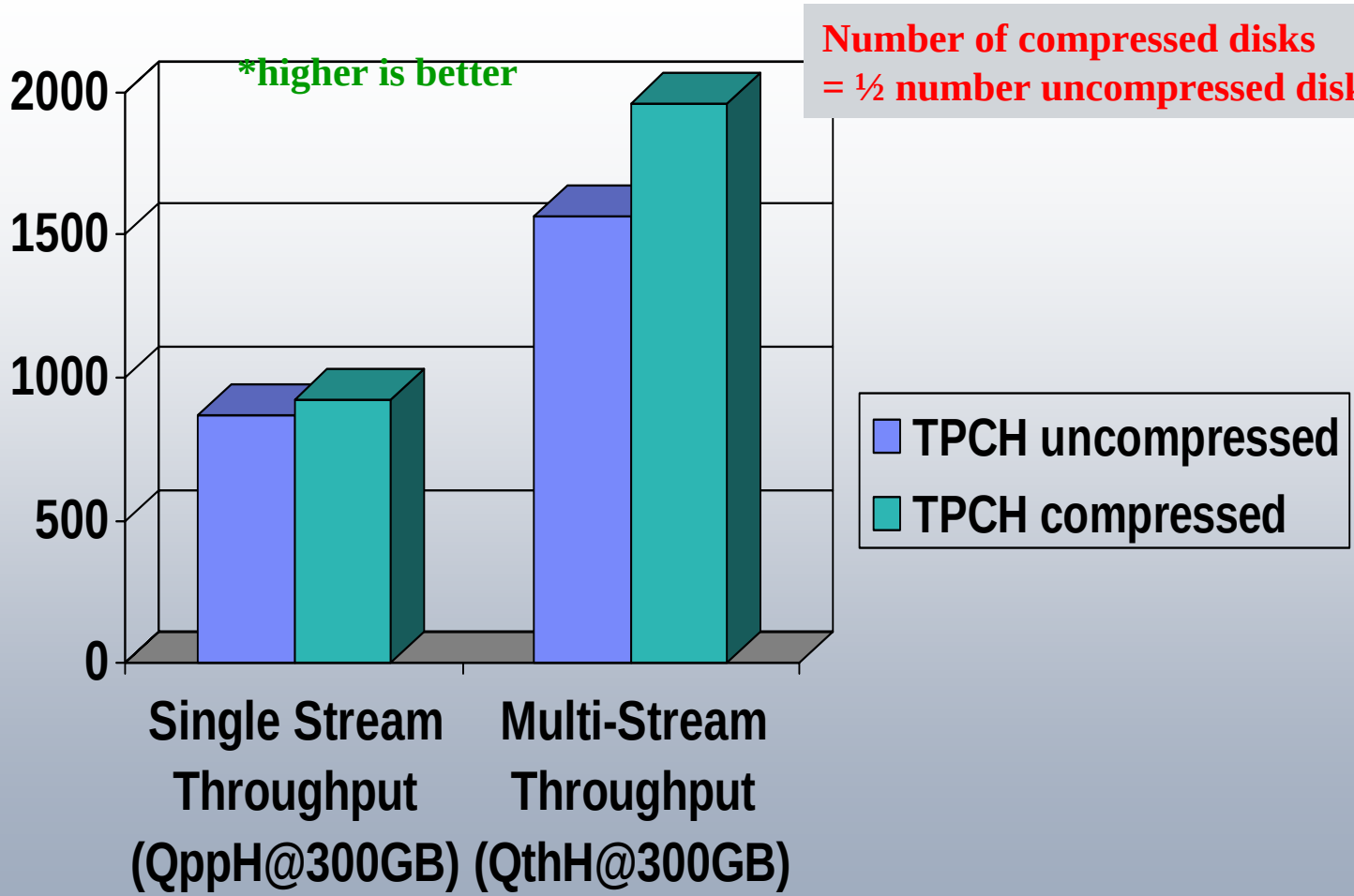
Data warehouse estimated reduction in size from 2.873TB to 1.453TB

- TPC-H 362GB Database
- System Configuration:
  - P5-570: 4-way (dual core) @ 1.65GHz
  - 128GB RAM
  - AIX 5.3 ML02
  - Storage:
    - 3 DS4500 Controllers
    - 8 VGs per DS4500 (24VGs in total)
    - 2 hdisks free each per VG (total 48 hdisks); each VG is 615GB
    - 13 disks per hdisk (624 physical disks available); disks setup in RAID5 using 1 hot-swap disk
    - 36.1GB available per disk

TPCH Space Savings per Table

TABLE	Number of 8K Pages	Number of 8K Pages with Compression	Total Space Saved
PART	1 118 524	401 494	5.9GB
SUPPLIER	60 201	33 250	220MB
PARTSUP	4 698 630	1 659 084	24.9GB
CUSTOMER	1 004 915	553 667	3.7GB
ORDERS	6 550 873	2 738 548	31.2GB
LINEITEM	30 701 523	13 532 605	140.6GB
TOTAL	**44 134 668**	**18 918 650**	**206.6GB**

TPCH Throughput Comparison



Query Performance

- Sample Warehouse Query Workload
  - I/O bound system: observed overall improvements in end-to-end workload execution time (20-30%)
  - CPU bound system: performance neutral or degradations on the order of 10% in end-to-end workload
- Compression trades I/O time for CPU time
  - If the system is CPU bound (more 70%), or if an agent is already consuming an entire CPU on its own, then compression might have a negative impact
  - If there is CPU headroom and I/O waits are happening compression can be beneficial

Backup Compression and Data Row Compression

Backup compression can be expensive and may not provide much added value in additional savings to backup image size

- **Time/size/value depends on the percentage of table space content with row compression. E.g. Are all tables compressed? Are indexes or long data stored in the same table space?**

Scenario	Total User Time (seconds)	Pages Used	Table Space Size (GB)	Backup Image Size (GB)
No compression	468	1510400	11.57	12
Backup Compression Only	1028	1510400	11.57	4.2
Data Row Compression Only	198	610816	4.68	4.7
Data Row and Backup Compression	662	610816	4.68	4.2

Best Practice:

LOAD 10%, REORG, Continue LOADing

TABLE	Time to Load 10% of Data (sec)	Time to Reorg (sec)	Time to Load 90% of Data (sec)	Compression Ratio	Total Time (sec): Load+Reorg
PARTSUPP	26.5	82.1	249.4	64%	358
ORDERS	36.4	86.4	411.9	57%	534.7
LINEITEM	163.5	330.4	1877.9	56%	2371.8

LOAD 100%, Full Table REORG

TABLE	Time to Load 100% of Data (sec)	Time to Reorg (sec)	Compression Ratio	Total Time (sec): Load+Reorg
PARTSUPP	260.9	559.6	65%	820.5
ORDERS	378.6	725.5	59%	1104
LINEITEM	1761.4	3572.8	57%	5334

Deep Compression Enhancements - Viper 2 (DB2 9.5)

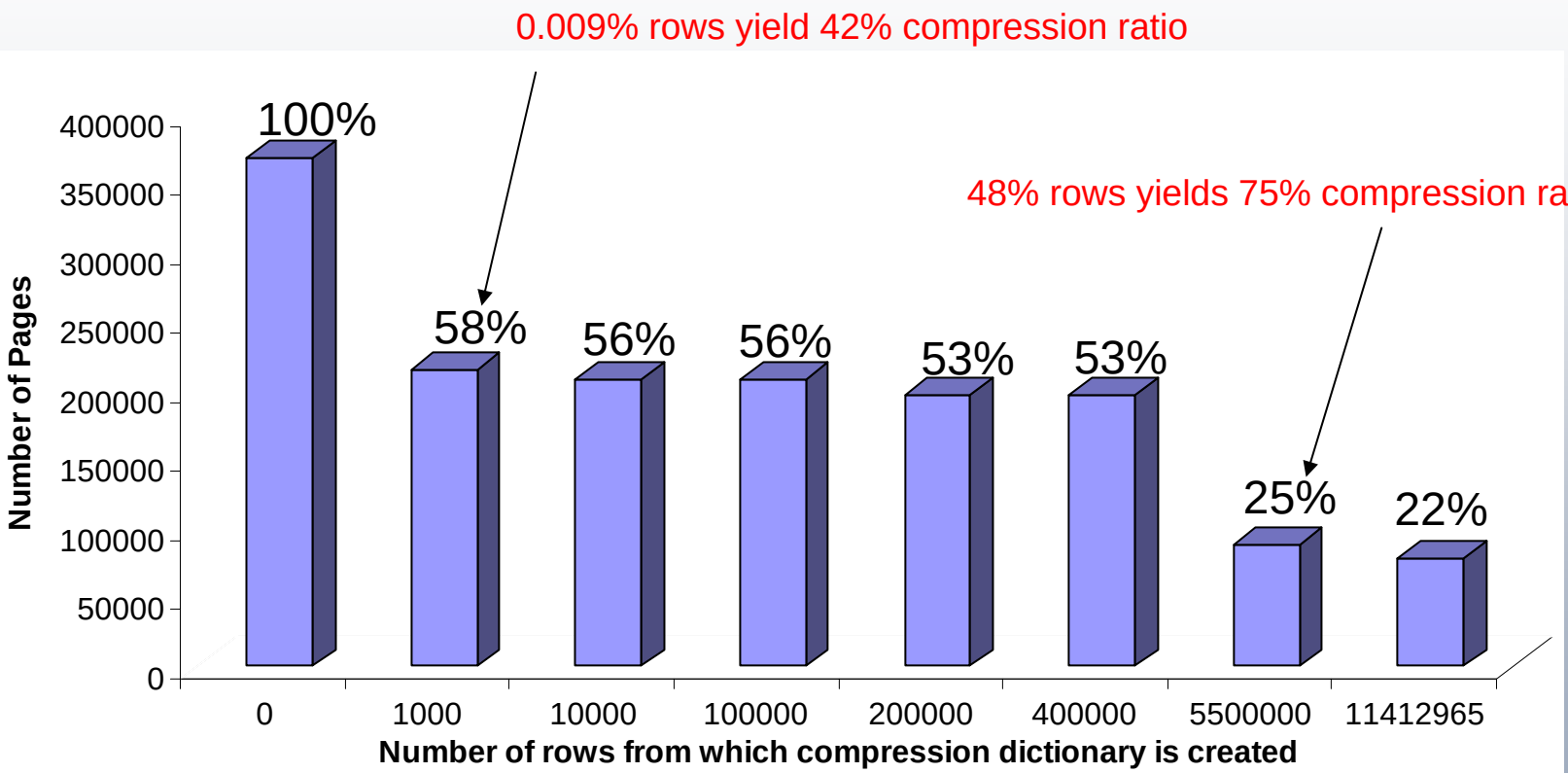
- What is new?
  - Automatic Dictionary Creation (ADC) with Table Growth
  - Dictionary build via the LOAD utility
  - New Compression Administrative Table Function
  - Compression Support for XML Data

Automatic Dictionary Creation (ADC) with Table Growth

- Optimal compression dictionaries and hence compression ratios, are achieved when the compression dictionary is built from an inclusive sample set. Table reorg (and inspect) builds a dictionary based on all the table data and thus produces the most optimal dictionary.
- The effectiveness of using small data subsets to build a compression dictionary is well demonstrated.
- This is the motivation behind Automatic Dictionary Creation (no table reorg) as part of table growth.

Table Size versus Size of Data Set used to Build Dictionary

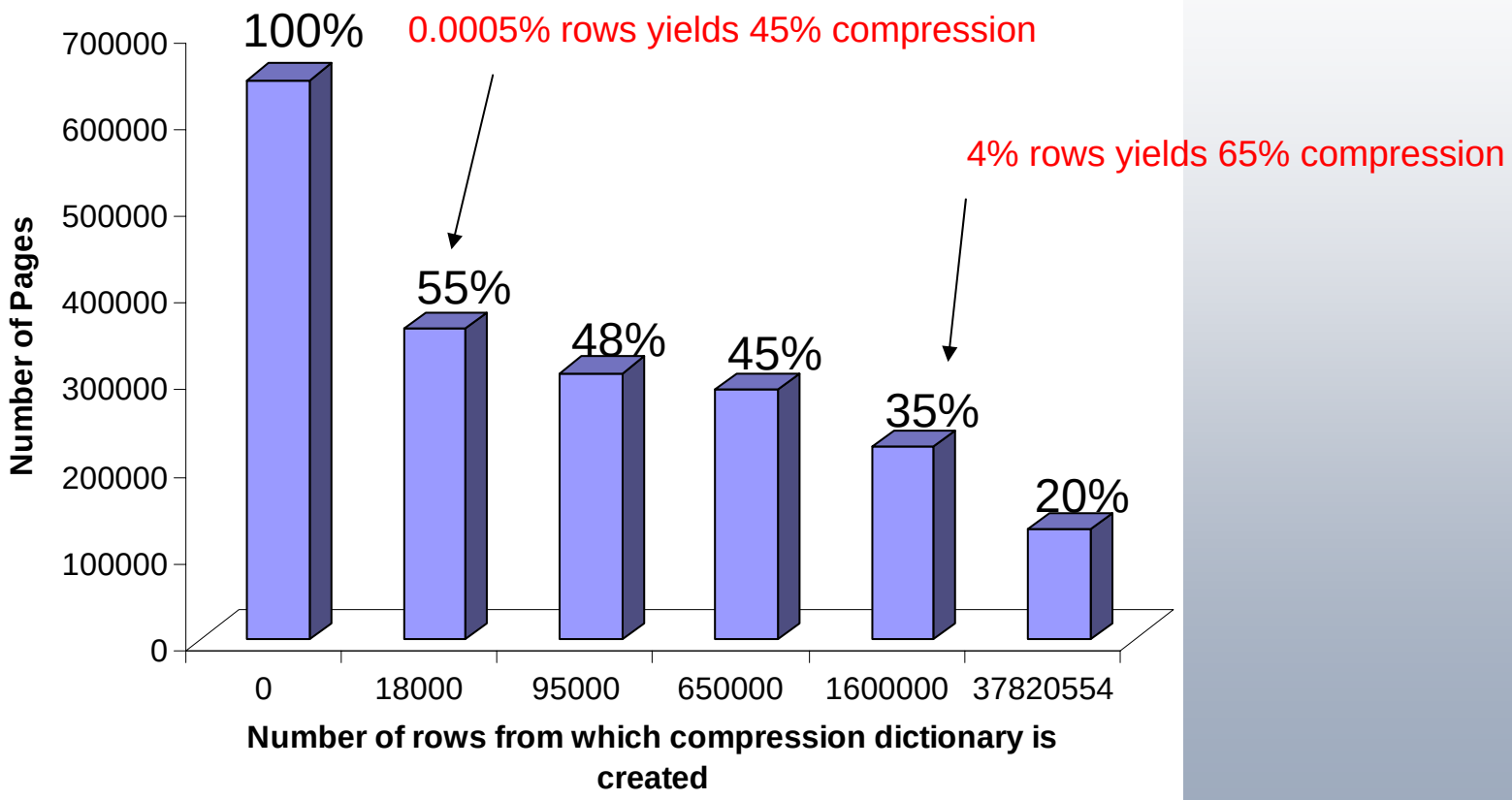
Column Data Type: VARCHAR
Total rows in table: 11,412,965

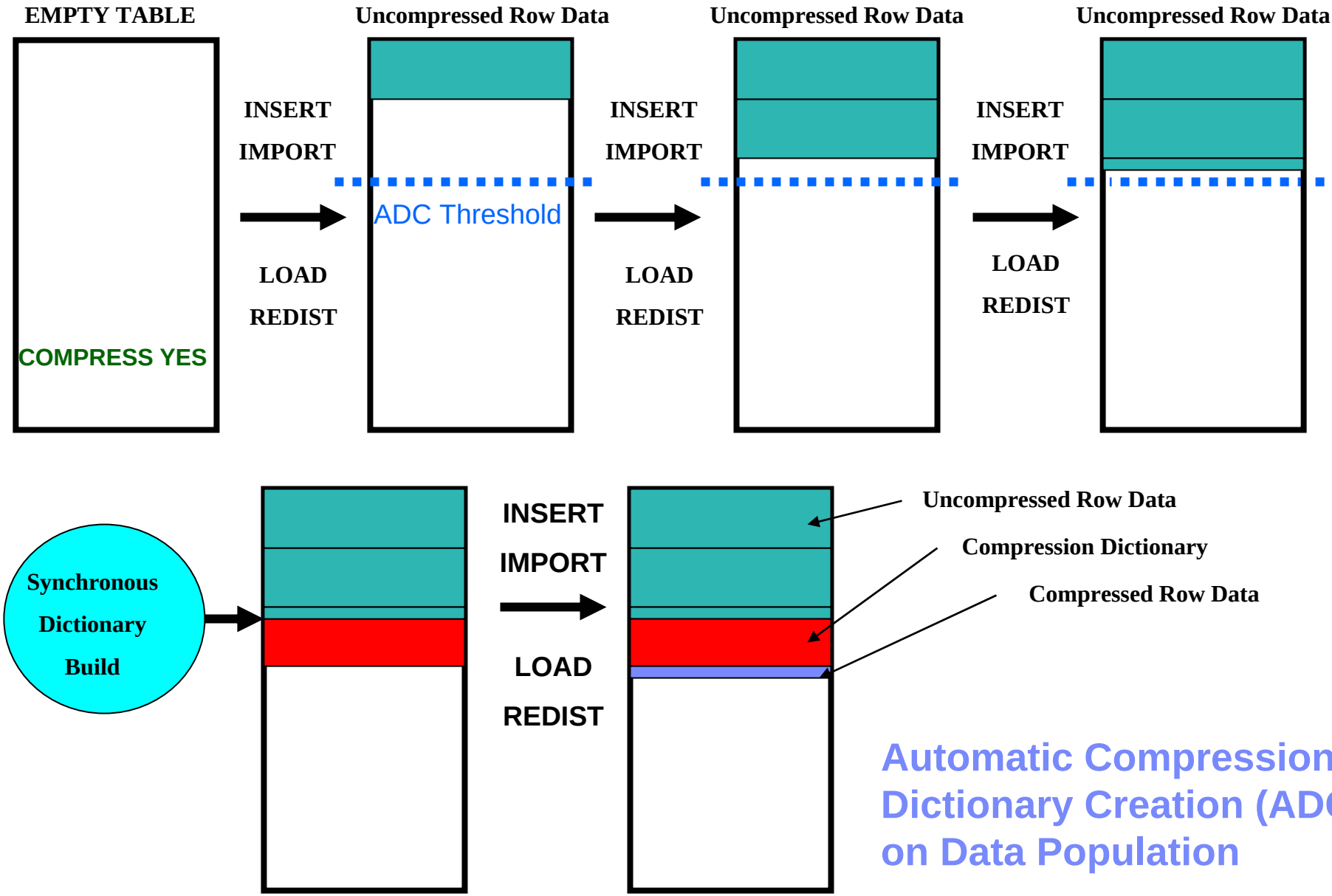


(Number of Sampled Rows)

Table Size versus Size of Data Set used to Build Dictionary

Column data types: (VAR)CHAR, INTEGER, DECIMAL
37,820,544 rows





Automatic Compression Dictionary Creation (ADC) on Data Population

Automatic Compression

- Compression automatically kicks-in as the table grows if COMPRESS attribute set
- The threshold at which ADC triggers is dependent on the size of the table and how much data exists within the table
  - Designed to happen when this makes sense:
    - ✓ Require decent compression
    - ✓ Don’t want to leave too much data in the table uncompressed
    - ✓ Don’t want to significantly impact the triggering transaction
- Applicable to growth operations: INSERT, IMPORT, LOAD, REDISTRIBUTE
- Reduces or eliminates need for table REORG
- Trade-offs:
  - Compression ratio can be less than optimal
  - Slight performance impact when threshold crossed

Automatic Compression – Hints/Tips

- If you don't want this behavior then don't set the table COMPRESS attribute until you are ready to compress the table
- Threshold is governed internally:
  - Table size must be on the order to 1 to 2 MB
  - There must be at least 700KB of data contained in the table
- If a large table is altered to have COMPRESS set ON, the next table growth action triggers ADC
  - Amount of table data scanned is limit to first portion of table

Compression Dictionaries and the *LOAD* Utility

- In DB2 9, *LOAD* will respect an existing compression dictionary but it could not create one

- In Viper II, the *LOAD* utility can now create a compression dictionary

  - **LOAD REPLACE RESETDICTIONARY**
    - Will unconditionally create a new dictionary or replace an existing one
    - A dictionary is built even if just 1 row of data is loaded (analogous to REORG TABLE RESETDICTIONARY)

  - **LOAD REPLACE KEEPDICTIONARY**
    - Will keep an existing dictionary but if one does not exist, will create one if sufficient data was loaded

  - **LOAD INSERT**
    - A dictionary can be automatically created if sufficient data is loaded or exists already in the table*

New Compression Administrative Table Function

ADMIN_GET_TAB_COMPRESS_INFO(tabschema, tabname, execmode)

- *execmode*
  - '**REPORT**'
    - Reports compression information at time table was compressed
  - '**ESTIMATE**'
    - Generates an estimate of new compression information based on current table data.
      - If *execmode* is empty ('') or NULL, the default value is 'REPORT'.

ADMIN_GET_TAB_COMPRESS_INFO ( )

Column	Data Type
TABSCHEMA	VARCHAR(128)
TABNAME	VARCHAR(128)
DBPARTITIONNUM	SMALLINT
DATA_PARTITION_ID	INTEGER
COMPRESS_ATTR	CHAR(1)
DICT_BUILDER	VARCHAR(30)
DICT_BUILD_TIMESTAMP	TIMESTAMP
COMPRESS_DICT_SIZE	BIGINT
EXPAND_DICT_SIZE	BIGINT
ROWS_SAMPLED	INTEGER
PAGES_SAVED_PERCENT	SMALLINT
BYTES_SAVED_PERCENT	SMALLINT
AVG_COMPRESS_REC_LENGTH	SMALLINT

Example: admin_get_tab_compress_info

```
select substr(TABNAME, 1, 10),
       COMPRESS_ATTR,
       DICT_BUILDER,
       DICT_BUILD_TIMESTAMP,
       PAGES_SAVED_PERCENT)
from table( sysproc.admin_get_tab_compress_info('BMINOR','', 'REPORT'))
as t;
```

```
1          COMPRESS_ATTR DICT_BUILDER   DICT_BUILD_TIMESTAMP       PAGES_SAVED_PERCENT
---------- ------------- -------------- -------------------------- --------------------
INVENTORY  N             NOT BUILT      -                          0
ORG2       Y             REORG          2007-05-24-15.31.23.000000 31
ORG4       Y             LOAD           2007-08-07-13.53.47.000000 71
ORGTAB     Y             NOT BUILT      -                          0
PRODUCT    N             NOT BUILT      -                          0
STAFF      Y             REORG          2007-08-06-16.59.56.000000 48
STAFFTAB   Y             TABLE GROWTH   2007-08-07-13.57.13.000000 66
```

Dictionary Creation Messaging

- **Admin log i.e. /sqllib/db2dump/<instance_name>.nfy**

```
2007-08-06-16.43.46.704292   Instance:bminor   Node:000
PID:1016338(db2agent (SAMPLE))   TID:7198   Appid:*LOCAL.bminor.070806184623
data management  sqldReorgDictionaryDriver Probe:160   Database:SAMPLE

ADM5592I  A compression dictionary was built and inserted into object "15" in
tablespace "2" via "REORG" processing.
```

- **db2diag.log file**

```
2007-08-06-16.43.46.703344-240 E4687A547          LEVEL: Warning
PID     : 1016338              TID  : 7198        PROC : db2sysc
INSTANCE: bminor               NODE : 000         DB   : SAMPLE
APPHDL  : 0-12                 APPID: *LOCAL.bminor.070806184623
AUTHID  : BMINOR
EDUID   : 7198                 EDUNAME: db2agent (SAMPLE)
FUNCTION: DB2 UDB, data management, sqldReorgDictionaryDriver, probe:160
MESSAGE : ADM5592I  A compression dictionary was built and inserted into object
          "15" in tablespace "2" via "REORG" processing.
```

New to Viper 2 – XML Inlining

- What is (XML) inlining?
  - Support (XML) column data within the formatted data rows on the data pages

  (Index, LOB, LF data are still stored outside of base table and cannot participate in compression)

- So why is this relevant?
  - Inlining small to medium sized XML documents in the base table should give a big performance improvement for all operations on them: insert/update/delete/query
  - *Now that XML documents can exist within the base table (i.e. are inlined in the data row), XML data can be compressed!*

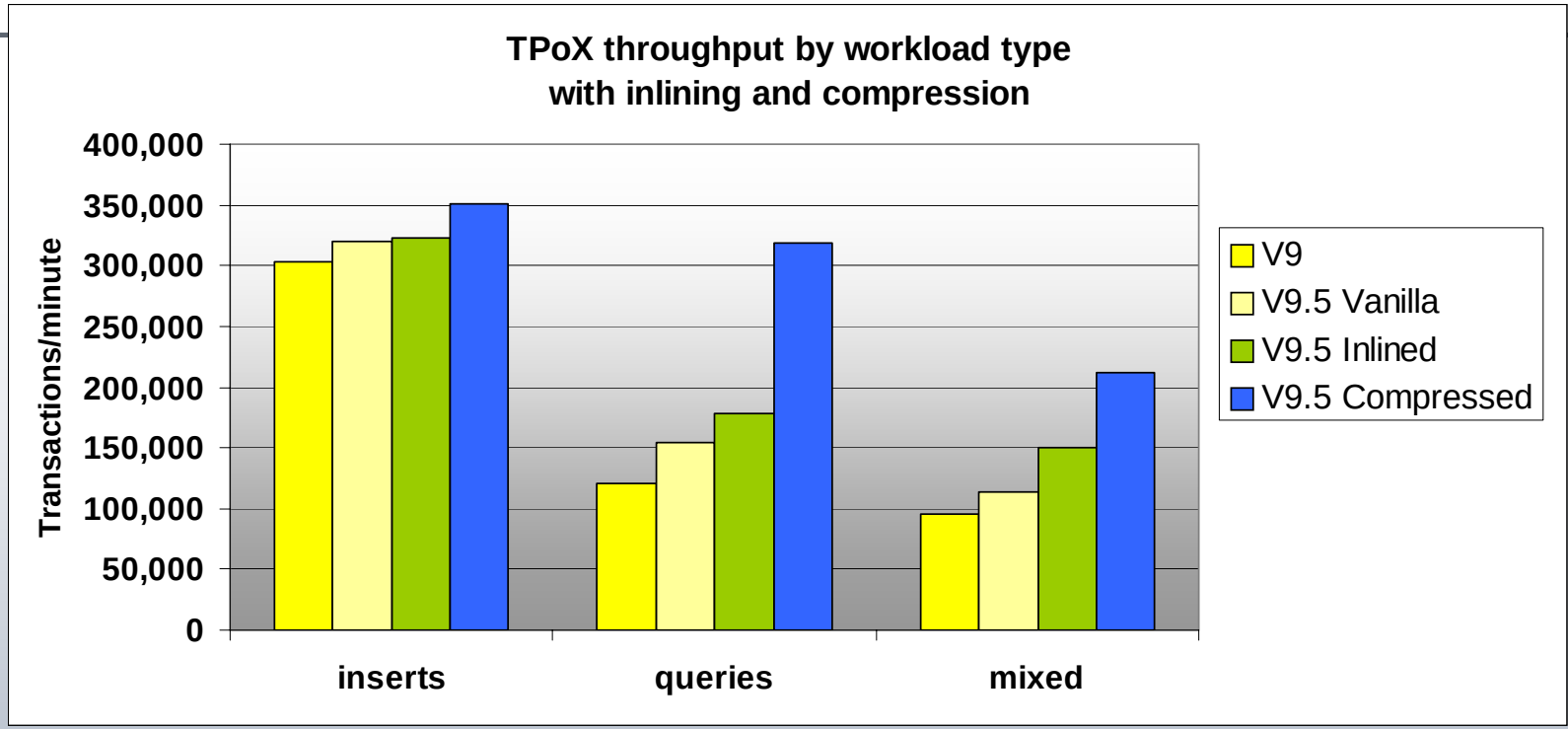
XML Document Inlining

```
ALTER TABLE <table name>
      ALTER COLUMN <column name>
      SET INLINE LENGTH <integer>

CREATE TABLE <table name>
      (<col name> XML INLINE LENGTH <integer>)
```

- Usage very similar to User-defined Structured Types
  - The inline length is limited by the page size
  - The inline length size can be increased but not reduced
- If documents are inserted that cannot be inlined, they will be inserted in the XDA object as if no inline length was specified

XML Inlining and Compression



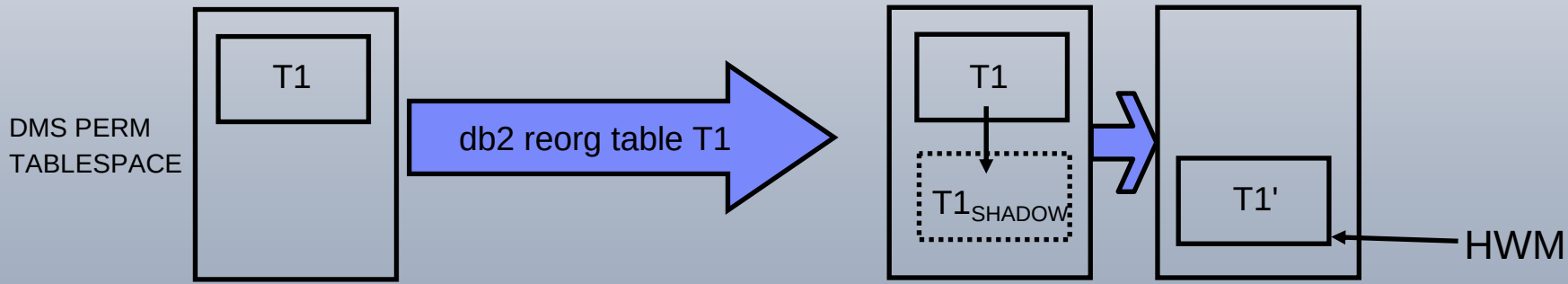
TPoX Database Number of 16K pages required

Page Type	*V9*	*Vanilla V9.5*	*Inlined V9.5*	*Compressed V9.5*
User Tables	21,178	21,178	628,518	259,706
XDA	955,600	632,448	250	250
User Indexes	88,388	88,233	88,122	88,312
Path/Region Indexes	17,961	15,458	6	6
Total	1,083,127	757,317	716,896	348,274
GB (= 1024**3)	16.5	11.6	10.9	5.3

100 Concurrent Users
12-way AIX

The "High Water Mark" (HWM)

- It is the page number of the highest allocated page in a *DMS* tablespace
- HWM is impacted by:
  - Offline' REORG of a table *within* the DMS tablespace that the table resides in
  - Index REORG with either ALLOW READ ACCESS or ALLOW WRITE ACCESS
- HWM affects:
  - Redirected Restore - redefinition of containers allowing tablespace to shrink in size; cannot be shrunk lower than HWM
  - Dropping or reducing the size of container via ALTER TABLESPACE only affects extents *above* the HWM



The "High Water Mark" (HWM) - Reducing via REORG

- If no free extents below the HWM then the only way to reduce the HWM is to drop the object holding it up

- db2dart option **/DHWM**
  - displays detailed tablespace information including which extents are free, which are in use and what object is using them as well as information about the object holding up the HWM
- db2dart option **/LHWM**
  - provides guidance as to how the HWM might potentially be lowered

- If DMS table data object holding up HWM then 'offline' REORG of table within the DMS tablespace that the table resides can be used to lower the HWM if enough free extents exist below the HWM to contain the shadow copy

- if DMS index object holding up HWM, index reorg may be able to reduce HWM

Some Best Practices

- Use INSPECT ROWCOMPESTIMATE to estimate compression ratios
- Use Automatic Compression to mitigate use of table reorg
- Collect stats and gauge compression effectiveness as data evolves
- Don’t specify index on reorg command unless required i.e. no reclustering reorg
- Lessen impact of DMS tablespace High Water Mark on tablespace space reclaim
  - Separate large tables into their own tablespace (same for index and lob data)
    - Reorg table using a tempspace for the shadow copy (specify USE clause)
  - Many tables in a tablespace
    - Reorg within tablespace and start with smallest table first
    - The HWM is grown least and the space freed within tablespace and below the HWM is available for the next table reorg to use
- To migrate big tables to compressed format
  - Consider building dictionaries online with INSPECT ROWCOMPESTIMATE and unload/reload utilizing High Performance Unload

References

- Chris Eaton blogs
  - http://blogs.ittoolbox.com/database/technology/
- developerWorks whitepaper
  - http://www.ibm.com/developerworks/db2/library/long/dm-0610chang/
- IBM
  - www.**ibm**.com/db2/**viper**
  - Search engine: ‘IBM Viper Compression’
- IDUG Solutions Journal
  - Spring 2007 Volume 14, Number 1

    “What Can DB2 9 Data Compression Do For You?”
- DB2 Magazine

Comparison of DB2 Viper Row Compression vs. the Competition

Comparison to Microsoft SQL Server 2005

- SQL Server 2005 has backup compression only available from 3rd parties
- They have no value compression
- They have no row compression

- In fact this note from Microsoft also recommends that you do not backup or store your database on compressed file systems
  - http://support.microsoft.com/kb/231347/en-us

- Microsoft claimed “Data Compression with SQL Server 2005 SP2”
  - What they delivered was a variable length Decimal data type
  - This is not what most people would call compression (simply a new data type that doesn’t waste as much storage as they did in the past)

Comparison to Teradata

- Teradata has dictionary based compression
- However,
  - Only at the column level for each column in the table
  - Only compresses 255 values per column
  - **DBA MUST SPECIFY THE VALUES TO COMPRESS ON CREATE TABLE**
  - Examples in their marketing material show 30% to 50% table compression

```
CREATE TABLE property (
   street VARCHAR(40),
   city CHAR(20) COMPRESS ('Chicago','Los Angeles','New York'),
   statecode CHAR(2)
)
```

Oracle Compression in 10g

- Oracle does allow tables or partitions of tables to be compressed
- However, Oracle compresses out common values **at the page level** (DB2 compression is at the table level)
  - This means that repeating values in a single page are replaced by a symbol in Oracle. The symbols are stored in the page header
  - Disadvantages of Oracle's approach
    - If there are consistent repeating values throughout the entire table, they will be stored multiple times in each page header
    - If the data is not sorted, there may be repeating patterns in the table but not on each page so Oracle will miss out on these compressions
    - Oracle only supports compression for bulk loads
      - DB2 supports compression for load, insert, and import

Key DB2 advantages over Oracle

- Non technically speaking
  - How many people in your department share the same birthdate?
  - How many people in your entire company share the same birthdate?
  - DB2 looks at a much large population of data and therefore finds more patterns to compress

- In a research paper on compression written by Oracle engineers they state:“Due to its global optimality of compression a **table-wide dictionary approach can result in high compression factors**.”

Table	Compression Ratio	
	Oracle	**DB2**
LINEITEM	38%	**58% (1.5x better)**
ORDERS	18%	**60% (3x better)**

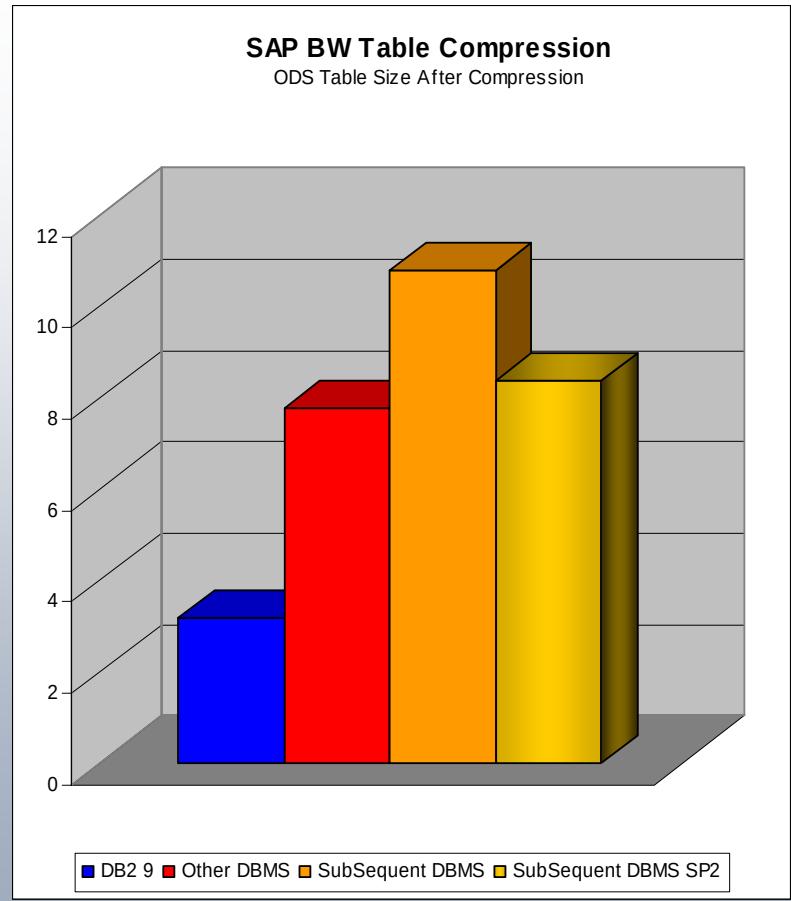
Advanced Compression Option in Oracle 11g

- In 11g Oracle has added the ability to compress data for insert/update operations
- Algorithm of page level compression only is still used
  - Would expect to get the same compression using this method as you would if you used load only in 10g (no better compression if data is loaded)
  - Previous example used LOAD -> therefore compression comparison is still valid
    - DB2 delivers between 1.5x and 3x better compression
  - Sorted data delivers better compression rates
    - Inserts typically do not arrive in sorted order therefore compression is likely to degrade over time (vs. initial load)

DB2 9 Compression for SAP BW Tables

- Size of 19 Million row FACT table (in GB) after compression

- DB2 9 delivers superior compression
  - 2.4x smaller than Other DBMS 1
  - 3.4x smaller than Other DBMS 2
  - 2.6x smaller than Other DBMS 2 with SP2



Estimating your compression benefits

- You have 4 days left to enter the IDUG Deep Compression Challenge
- Estimate your compression savings and enter the sweepstakes for your chance to win a free conference pass to IDUG (2008 NA, Europe or Australia conference)
- If you are currently running
  - DB2 9 – use the INSPECT command to evaluate compression benefits
  - DB2 V8 – use the Data Compression Estimation Calculator available on the IDUG website
  - Oracle or Microsoft – Use the Data Storage Analyzer Tool also available via the IDUG website
- www.idug.org/wps/portal/idug/compressionchallenge