

Contents

Row Compression in DB2 9: Analysis of a DSS Database Environment	- 1 -
Executive Summary	- 1 -
Introduction	- 2 -
Experimental Design	- 3 -
Results: Compressed vs. Uncompressed	- 5 -
Throughput and Space/Cost Savings	- 5 -
CPU Overhead, I/O Utilization and Analysis	- 8 -
Best Practices for Row Compression through DB2 REORG	- 11 -
Full TABLE REORG Operation (re-clustering): Scan-Sort vs. Index Scan	- 12 -
Partial TABLE REORG Operation: REORG and LOAD	- 14 -
Conclusions	- 15 -
Appendix A:	- 16 -
Appendix B:	- 17 -
Appendix C:	- 18 -
Acknowledgments	- 19 -
References	- 19 -
®	- 20 -

Tables and Figures

Table 1 – Experimental Design Hardware Details	- 4 -
Table 2 – Disk Usage Details	- 4 -
Table 3 – Space Savings Compression Details per Table	- 5 -
Figure 1 – Database Space Savings from Backup Image Size	- 6 -
Figure 2 – Workload Throughput Comparison (Single-Stream)	- 6 -
Figure 3 – Workload Throughput Comparison (Multi-Stream)	- 7 -
Figure 4 – Single Stream CPU and Disk I/O Utilization per Query	- 9 -
Figure 5 – Multi-Stream CPU and Disk I/O Utilization	- 10 -
Table 4 – Default Full Table REORG	- 11 -
Table 5 – Re-clustered Full Table REORG using Scan-Sort	- 12 -
Table 6 – Re-Clustered Full Table REORG using Index-Scan	- 13 -
Figure 6 – The Effects of a Partial LOAD and REORG Operation	- 14 -
APPENDIX A – Row Size Compression Details per Table	- 16 -



Executive Summary

Every business model looks for the best way to minimize production costs in order to maximize profits. In the summer of 2006, IBM® released the latest version of its data server software technology, DB2® 9. One of the most prominent features introduced in this version is the ability to perform **row compression**. The goal of row compression is to reduce table storage. When using this feature, DB2 is able to shrink row sizes by building and utilizing a compression dictionary, which is used as an alphabet to compress the rows of data in the database tables. Since more data can reside on a data page, fewer data pages are allocated per table. The net effect is a reduced rate of storage consumption which may lead to fewer disks needed when allocating storage requirements. Since disk storage is one of the most expensive components on any data server, this feature can significantly reduce Total Cost of Ownership (TCO).

[DSS]

“Decision Support Systems” represent a specific class of computerized information system that supports business and organizational decision-making activities.

[Single-stream]

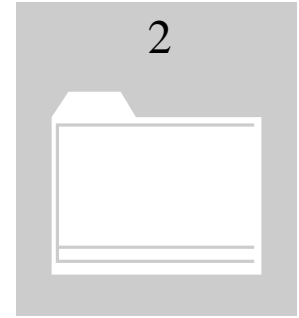
In terms of running our experiments, it consists of running each of the 22 workload queries individually in a specified sequence through a single database connection.

[Multi-stream]

In terms of running our experiments, N queries are run in parallel by N connections until all 22 queries are executed by each connection in a predetermined fashion.

Since row compression can reduce TCO, what effect does it have on performance? Due to the nature of compression and expansion algorithms, row compression naturally consumes additional CPU cycles beyond the equivalent uncompressed database activity. In systems with a typical amount of idle CPU, the performance effect of row compression may be negligible and in some cases be more efficient depending upon the compression ratio and the amount of data being retrieved from disk. However, for workloads that consume significant amount of CPU, it may be necessary to increase CPU capacity in order to maintain desirable performance results.

In a 362 GB **DSS** database, overall database storage consumption was reduced by 40% when row compression was enabled. As a result, the compressed database was restructured to use 50% fewer data disks than the original uncompressed database used. Regardless, the compressed database was able to achieve throughput similar to the uncompressed database when running a **single-stream** workload, and even experience a 25% throughput improvement in a **multi-stream** scenario. For row compression to perform efficiently, free CPU resources are indispensable because they are necessary to account for the overhead of compressing and expanding rows. Row compression should provide a performance enhancement to a database system if complex select queries that require mostly sequential data access create the bulk of its workload. If these requirements are met, the benefits of decreased production costs and increased process efficiency can both be attained through the use of this feature.



Introduction

In order to help its customers reduce TCO, IBM DB2 has introduced a row compression feature in DB2 9. Row compression uses dictionary-based lossless compression using a variant of the Lempel-Ziv algorithm to apply compression to each row of a data table. All rows are subject to compression but only those that exhibit storage savings will be compressed. Since it is likely that most rows will be compressed, fewer disks are generally sufficient to store the same data; this allows storage capacity savings which can lead to cost savings. The storage savings come at the cost of CPU cycles used to compress and expand the data in a row compressed table. Thus, there is a tradeoff between disk savings and CPU consumption, which may have a direct effect on overall system throughput.

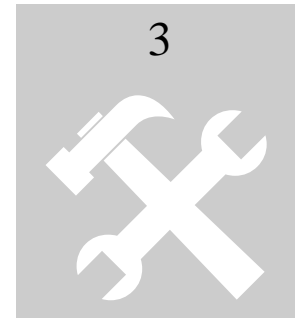
The objective of this whitepaper is to address the storage savings and performance impact of using row compression. The intent is to describe the workload characteristics and environments in which row compression will thrive, as well as provide general guidelines that can be followed to more efficiently use this feature.

The TPC-H [1] schema, along with its workload queries, was selected to represent a typical DSS environment. Please note that, while the TPC-H data schema was used for this experiment, the workload was not executed in a way that is comparable to official TPC results. Comparisons with official results would be misleading and should not be considered. In this whitepaper, the throughput achieved by an uncompressed database is compared to a row compressed database running on fewer data disks. The decrease in the number of disks chosen is related to the compression ratio achieved when comparing the compressed and uncompressed backup image sizes. Following this discussion, the CPU utilization and I/O activity are studied in order to better understand the effects of row compression at a query level.

[REORG]

A DB2 command used to reorganize an index or a table. Since DB2 9, this command can also be used to create the dictionary for a compressed table.

Based on the analysis of the TPC-H data schema, the reader should have an adequate idea if he or she can use the row compression feature to their advantage. This technical whitepaper also provides some insight towards the best practices to build a database with row compression. The data dictionary used to compress and expand the rows in the table is built using the DB2 **REORG** TABLE command. This whitepaper will conclude its analysis with a few suggestions towards how to better utilize this utility for the purpose of row compression.



Experimental Design

[Geometric Mean]

The geometric mean of a set of 22 queries {q1, q2, ..., q22} is defined as the 22nd root of the product of all the members of the set.

TPC-H is an ad-hoc decision support workload. It has industry-wide relevance as it examines large volumes of data and executes highly complex queries that answer critical business questions. The DSS workload for this research project was modified from the TPC-H specification and results shown here are not comparable to official TPC-H results. In particular, we ran fewer than the required number of query streams for the size database that was exercised. This was done to create an environment where, on average, there was idle time in the CPUs of the system tested. We also altered the workload requirements to include only read-only queries. The complete suite of tests and audit procedures that are required of the benchmark were not executed. Results are not intended for comparison outside of the context of this whitepaper. Comparison outside of the data described in this whitepaper could lead to incorrect conclusions.

In this whitepaper, two performance metrics are used to report results. The *geometric mean* for all 22 queries ran in the workload was used to report the single-stream results because it provides equal weighting for all queries. The throughput of the multi-stream workload was measured in Queries per Hour:

$$\text{Throughput} = N_Q / T_s$$

Where

- N_Q represents the number of queries ran (4 streams * 22 queries per streams = 88 queries for multi-stream).
- T_s represents the measurement interval from the start of the first stream running the first query, to the end of the last query ran by the last outstanding stream.

This whitepaper will analyze storage savings and throughput measurements as well as individual query response times for the 362 GB DSS database.

[RID]

RID stands for Row Identifier. From DB2 9, users have to option to create regular 4-byte RID tablespaces or large 6-byte RID tablespaces in order to fit more rows per page, and more pages per tablespace.

The DSS workload consists of large queries which process hundreds of thousands or even millions of rows, typically executed by very few concurrent users. Sequential I/O access dominates query execution. An 8KB page size was used for both compressed and uncompressed databases because this design benefits from its sequential I/O characteristics. Large *RID* tablespaces were used, which is highly relevant to the row compression feature because it allows DB2 to fit more than 256 rows in a data page, a limit that is more likely to be reached when enabling row compression. Large RIDs are the default in DB2 9.

The hardware details of the machine used to conduct these experiments are shown in Table 1 below.

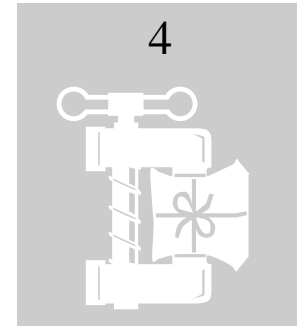
H A R D W A R E D E T A I L S F O R I B M P 5 - 5 7 0	
CPU	4-core Power5 at 1.65 GHz
OPERATING SYSTEM	AIX® 5.3
MEMORY	128 GB RAM available
STORAGE CONFIGURATION	3 IBM TotalStorage® DS4500 Controllers 1.6 TB of storage used for all tests

Table 1 – Experimental Design Hardware Details

While uncompressed, the DSS database was created using 32 hdisks for data and temporary tablespaces across two DS4500s, and the compressed DSS database was created across half as many hdisks and on only one DS4500. The reasoning behind this selection will be described in the results section. The transaction log configuration was held constant across both experiments. Table 2 below outlines the number of disks used for both compressed and uncompressed databases as well as the number of controllers used.

D I S K U S A G E D E T A I L S		
	362 GB DSS database	Compressed database
NUMBER OF DS4500 CONTROLLERS	2	1
DATA DISKS	384	192
DISKS FOR LOGS	12	12
TOTAL PHYSICAL DISKS	396	204

Table 2 – Disk Usage Details



Results: Compressed vs. Uncompressed

The use of row compression generally results in excellent storage savings but can sometimes have an impact on performance. This section starts off by highlighting the throughput characteristics and space savings obtained in both single-stream and multi-stream workloads when using row compression. Following this section, the CPU and disk I/O tradeoffs that result from using this feature are analyzed.

Throughput and Space/Cost Savings

The DSS database schema consists of eight tables, the largest three being the lineitem, orders and part-supplier tables. Table 3 shows the total number of pages used for the compressed and uncompressed database tables as well as the space savings attained per table. There is no need to compress smaller tables like nation and region, since a single page access already encompasses the entire uncompressed table. For details about the average row sizes for each case, please refer to Appendix A.

TABLE	Number of 8K Pages	Number of 8K Pages with Compression	Percent Savings	Total Space Saved
NATION	1	1 (<i>compression not necessary</i>)	0%	0.0 GB
REGION	1	1 (<i>compression not necessary</i>)	0%	0.0 GB
PART	1,118,524	401,494	64%	5.9 GB
SUPPLIER	60,201	33,250	45%	0.2 GB
PARTSUP	4,698,630	1,659,084	65%	24.9 GB
CUSTOMER	1004915	553,667	45%	3.7 GB
ORDERS	6,550,873	2,738,548	58%	31.2 GB
LINEITEM	30,701,523	13,532,605	56%	140.6 GB
TOTAL	44,134,668	18,918,650	57%	206.6 GB

Table 3 – Space Savings Compression Details per Table

Since the indexes are not compressed (in order to retain maximum index access performance), the overall reduction in database size is somewhat lower than the average savings of all the tables. The size of the database backup image provides an appropriate indication of the size of the database itself, and in this case was reduced by 40% compared to the uncompressed version. Figure 1 shows the sizes of both the uncompressed and row compressed DSS databases. It is worth mentioning that DB2 also supports compression at a backup level. This feature can further improve the amount of space savings allocated to a backup database image, primarily due to a decrease in the size of the database's indexes. However, the details of this feature will remain outside the scope of this whitepaper.

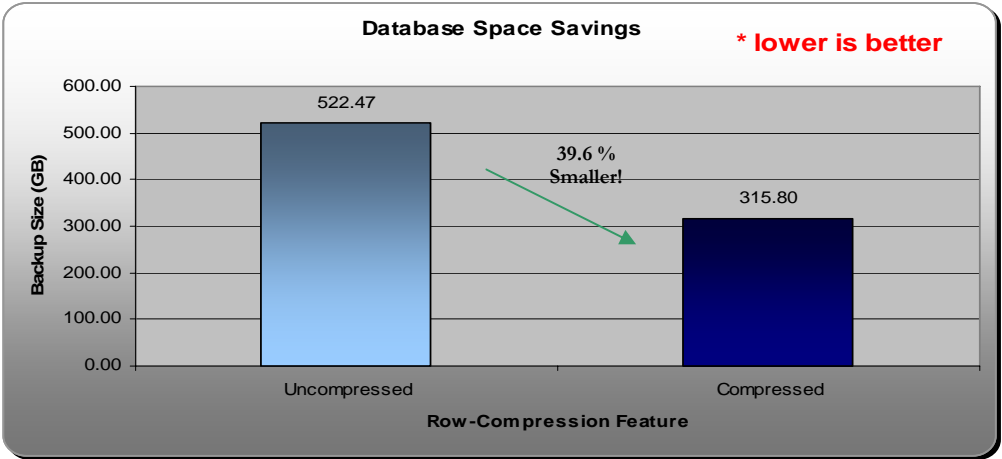


Figure 1 – Database Space Savings from Backup Image Size

In order to translate the resulting space savings into actual cost savings, it was decided to use only half as many disks for the tablespace data containers for the row compressed database in the experiments to follow. Figure 2 below compares the geometric mean response time of all workload queries between an uncompressed and a row compressed database setting.

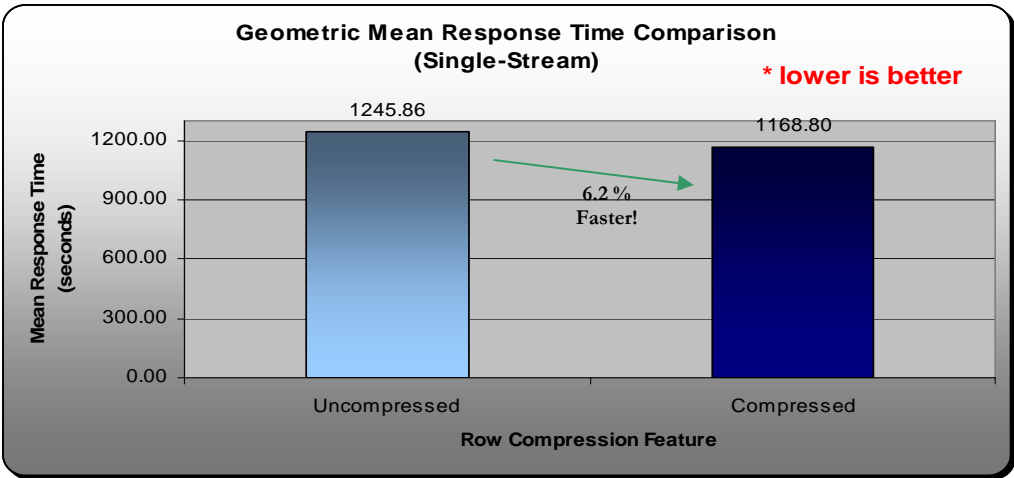


Figure 2 – Workload Throughput Comparison (Single-Stream)

Figure 3 below shows the throughput outcome (in queries per hour) of running the workload in 4 multiple concurrent streams.

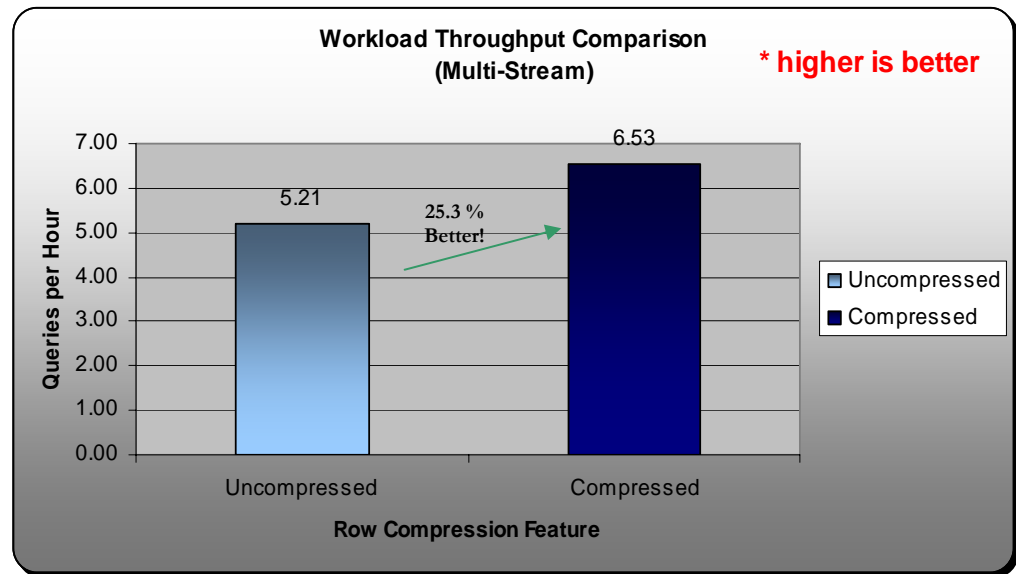


Figure 3 – Workload Throughput Comparison (Multi-Stream)

Even though the row compressed database was using 50% fewer data disks than the uncompressed database, its throughput increased more than 5% in the single-stream case and by 25% in the multi-stream case. In this scenario, not only were there cost savings from using fewer disks, but the system was able to achieve better DSS performance when using row compression. This feature effectively traded idle time spent waiting on I/O, for valuable CPU time used to compress/expand rows. Moreover, it also decreased the amount of I/O traffic in the disk subsystem. It is important to note that these results are dependent on factors such as the amount of available CPU power on the system.

It is important to mention that some query plans changed due to changes in statistics originating from the use of row compression. This is expected since the compressed tables use fewer pages, etc. – factors which the optimizer uses to generate the query access plan.

[intra_parallel]

A DBM configuration parameter that specifies whether the database manager can use intra-partition parallelism; the default is **not** to use intra-partition parallelism.

DB2 supports *intra-partition parallelism*, where query execution can be parallelized, typically on multiple CPUs, within one partition. Customers who are using INTRA_PARALLEL in a disk-bound environment may find additional benefit from row compression. This is because in a parallelized query execution, multiple DB2 subagents are executing concurrently, typically on different CPUs. When row compression is enabled, each of these subagents must then decompress the rows they want to process. In other words, decompression is essentially parallelized by the subagents, and is able to exploit idle processing power across multiple CPUs. Note that INTRA_PARALLEL can have a substantial impact on query access plans, so customers are recommended to carefully review the DB2 documentation [2] before enabling it.

Knowing that row compression may benefit from the use of intra parallel processing, we chose to perform some experiments enabling this feature. As expected, in our test system, there was a throughput increase of over 30% when using row compression (like the previous experiment, using only half the number of data disks) when INTRA_PARALLEL was enabled. The average mean response time for the individual queries executed with single stream, INTRA_PARALLEL ON are displayed in Appendix B.

The tested workload is composed of 22 complex select queries. Since there is no update activity in these queries, throughput improvement is achieved by decreasing the amount of I/O traffic and by exchanging useless machine cycles spent waiting on I/O for useful user CPU cycles to perform data expansion. SQL that modifies the database will likely require more CPU resources than queries of similar complexity, as rows must be uncompressed, modified and re-compressed before the changes can be written out. This extra amount of CPU work may exceed the amount of idle CPU capacity on the system. It is important to realize that the performance impact of row compression can vary depending on whether the workload is CPU or I/O bound. An analysis of CPU resources consumed and available in the system will be discussed in the next section.

Complex queries tend to manipulate rows sequentially. With row compression, every data page can now contain more rows. Thus, the workload queries in a row compressed database need fewer I/Os to access the same amount of data. If rows were accessed more randomly, rather than sequentially, the benefits of row compression might not be quite as good.

CPU Overhead, I/O Utilization and Analysis

When considering the use of row compression it is important to take the CPU and disk I/O utilization of the system into account. Since there is additional overhead when compressing and expanding rows in compressed tables, it is expected that row compression requires more CPU resources. However, since fewer disks are used in the compressed database setup, but at the same time, less I/Os are sufficient to run the same queries in an uncompressed scenario, the impact on I/O utilization needs to be examined. An analysis of the CPU and disk I/O resource utilization for both setups is given in this section.

Figure 4 below provides a graphical overview of the user CPU, system CPU and *I/O wait* utilization by each query with and without compression. Not all 22 of the workload queries are displayed in this graph. An analysis of queries Q3, Q5, Q8, Q10 and Q20 is not provided since after applying row compression, the plans of these queries changed. The CPU and I/O wait values of these queries cannot be compared side by side because they follow a different pattern of work that is not directly comparable between the uncompressed and compressed scenarios. This is not an issue to be concerned about since the DB2 query optimizer successfully provided better access plans which resulted in faster individual query response times.

[I/O Wait]

It consists of CPU idle time during which the system had outstanding disk/NFS I/O request(s).

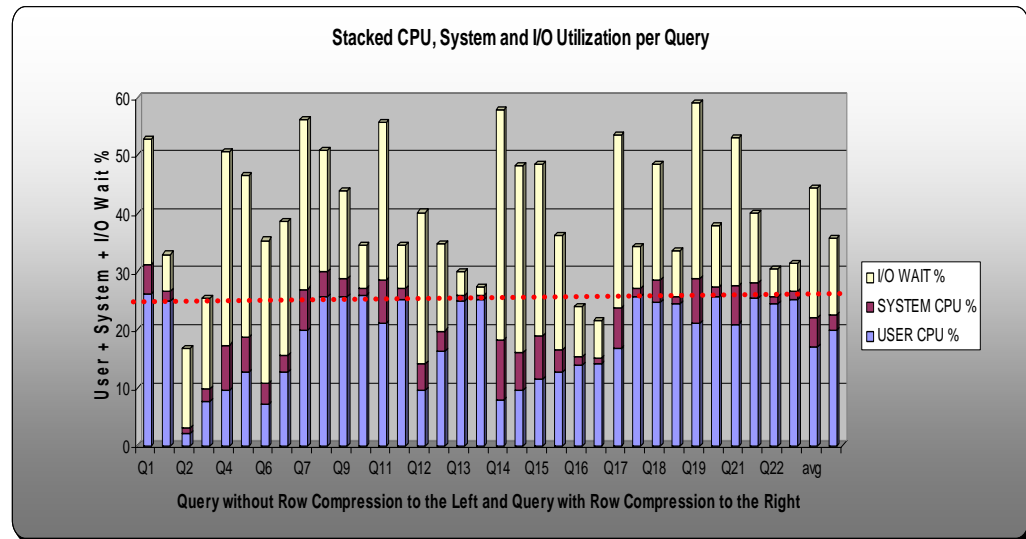


Figure 4 – Single Stream CPU and Disk I/O Utilization per Query

The figure shows that queries Q1, Q9, Q13, Q18 and Q22 are all CPU-bound since they consumed on average nearly 25% CPU of a 4-core machine in a single stream uncompressed database. Since there were no idle cycles available on the CPU executing the query in a single-stream, row compression consumed cycles that would have been used by the query itself, thus increasing the elapsed time.

Queries Q7, Q11, Q17, Q19 and Q21 were not CPU-bound when running against the uncompressed database (i.e., they consumed less than 25% CPU on a 4-core machine), but became CPU-bound when running against the compressed database. Row compression consumed the remaining free cycles on a single CPU, and possibly also some additional cycles that would have been used for the query itself. This may have slightly hurt the performance of these queries.

All other remaining queries had spare CPU cycles and spent significant time being idle waiting on I/O. Row compression allowed DB2 to fetch fewer pages from the disks, resulting in a decrease in I/O wait and a corresponding increase in user CPU consumption. Reducing the I/O bottleneck resulted in faster query execution times. On average, row compression used up 3% more CPU resources while the time spent waiting on I/O decreased by 9%.

When running the workload as a single stream, row compression can benefit from turning 'ON' the intra-parallelism setting because it takes advantage of the other otherwise unused 75% CPU. The throughput improvement of over 30% is tied to the result of using 13% more user CPU and spent 8% less time waiting on I/O.

Figure 5 below shows the average total CPU utilization for the multi-stream workload setup. When running this workload against the uncompressed database, there were spare CPU cycles since time was wasted waiting on I/O. In this case, row compression

increased CPU utilization by 17%, while time spent waiting for I/O decreased by 6%. The efficiency seen with these system statistics are in accordance to the 25% overall throughput improvement seen previously in Figure 3.

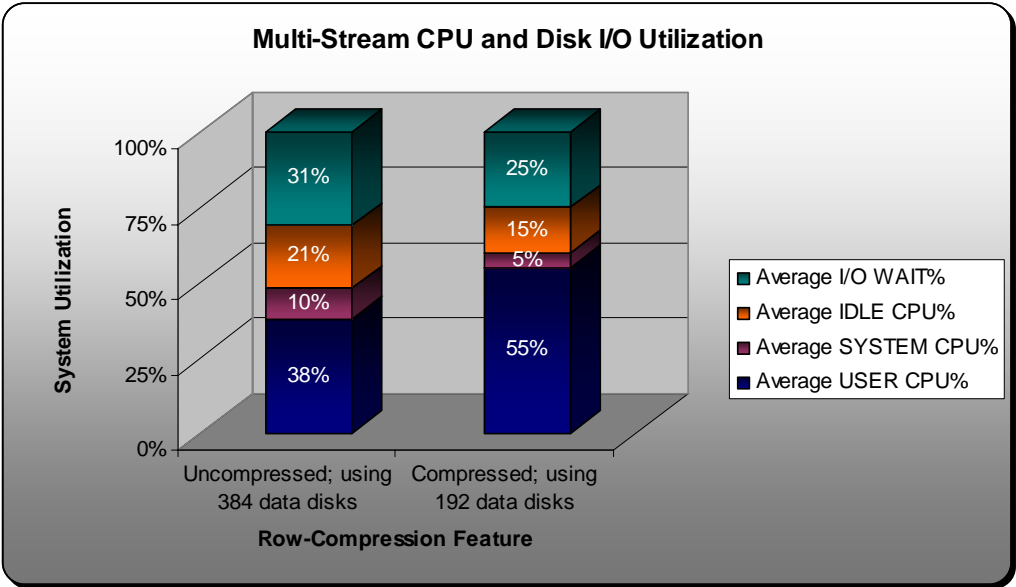
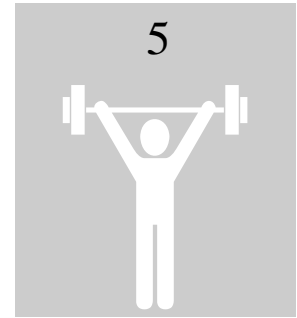


Figure 5 – Multi-Stream CPU and Disk I/O Utilization



Best Practices for Row Compression through DB2 REORG

Classic (offline) REORG of a compressed table creates a compression dictionary which is used for compressing and expanding. It is important to understand and distinguish different REORG commands as well as assess the impact of this additional work. In the following sections, we discuss different ways of executing the REORG command as well as analyze the tradeoff between enabling row compression by creating a dictionary based on an entire table versus creating a dictionary using only a subset of it.

The following experiments were performed on a smaller 12 GB DSS database to show the increase in time taken to do a REORG operation when using row compression. Table 4 below shows the amount of overhead required for the default REORG command to apply row compression. The overhead to create the compression dictionary is slightly less than 15 minutes for a 12 GB database.

TABLE	REORG – Without Row Compression (sec)	REORG – With Row Compression (sec)	Dictionary Creation Overhead (sec)
NATION	1.0	1.3	0.3
REGION	0.9	1.1	0.2
PART	35.6	56.0	20.4
SUPPLIER	3.6	67.7	64.1
PARTSUP	301.5	546.3	244.8
CUSTOMER	40.6	124.2	83.6
ORDERS	527.2	639.0	111.8
LINEITEM	2784.8	3113.6	328.8
<i>TOTAL</i>	3695.2	4549.2	854.0

Table 4 – Default Full Table REORG

Full TABLE REORG Operation (re-clustering): Scan-Sort vs. Index Scan

In the event that the user wishes to reorganize its database and re-cluster it according to a particular index, one can either perform a full REORG on a table by applying the scan-sort or the index-scan method. DB2 uses the scan-sort method by default when an index is specified in the REORG command. This involves scanning the table and sorting the results in memory (although it may spill to disk through a temporary tablespace). The index-scan REORG method requires the explicit use of the INDEXSCAN keyword; it does not use a sort because it follows the order of the index. However, the index-scan method needs to create a shadow copy of the table in the temporary tablespace by fetching the records in the original table for each RID in the index. For more details on REORG, please consult the IBM DB2 documentation pages [2].

Table 5 below shows the overall increase in time needed when using the scan-sort method. With additional work allocated towards re-clustering the table on an index, the total time to REORG and compress the table increased by about 5 minutes overall compared to the default REORG command. The overhead to create the row compressed dictionary decreased to less than 12 minutes for the 12 GB database.

TABLE	REORG – Without Row Compression (sec)	REORG – With Row Compression (sec)	Dictionary Creation Overhead (sec)
NATION	1.1	1.3	0.2
REGION	0.9	1.1	0.2
PART	44.2	63.9	19.7
SUPPLIER	4.0	68.3	64.3
PARTSUP	355.3	436.0	80.7
CUSTOMER	45.9	128.1	82.2
ORDERS	571.2	670.9	99.7
LINEITEM	3156.7	3509.4	352.7
<i>TOTAL</i>	4179.3	4879.0	699.7

Table 5 – Re-clustered Full Table REORG using Scan-Sort

Table 6 shows the results of using REORG through the index-scan method to build the row compressed table. Although the percent overhead to create the dictionary is slightly higher when completing a REORG through scan-sort, the overall time to perform the REORG operation and create the dictionary is faster. Because REORG through index-scan processing requires an extra pass of the table in order to build the compression dictionary, scan-sort processing usually completes the reorganization and dictionary creation more quickly. Index-scan REORG can be beneficial when indexes are highly clustered and there

are not enough sort resources available in the system (memory and temp space) in which a scan-sort would spill. Moreover, if the table is already compressed, an index-scan can potentially perform better since there are fewer pages being processed which increases the likelihood of an improved bufferpool hit ratio.

TABLE	REORG – Without Row Compression (sec)	REORG – With Row Compression (sec)	Dictionary Creation Overhead (sec)
NATION	1.4	1.3	-0.1
REGION	0.9	1.1	0.2
PART	74.6	51.3	-23.3
SUPPLIER	5.5	67.3	61.8
PARTSUP	481.8	523.7	41.9
CUSTOMER	76.2	120.8	44.6
ORDERS	1624.6	1490.2	-134.4
LINEITEM	8264.1	8761.3	497.2
<i>TOTAL</i>	10529.1	11017.0	487.9

Table 6 – Re-Clustered Full Table REORG using Index-Scan

Partial TABLE REORG Operation: REORG and LOAD

The previous section discussed REORG on pre-filled tables. In the event that the table to be compressed has not yet been completely loaded, the overhead of creating a dictionary for compression can be further reduced.

The idea here is to first load a portion of the data (typically 10%) into the table before compression is in effect. Once this data is loaded, the table can be reorganized in order to create a dictionary on the smaller portion of data loaded. It is assumed that the preloaded data is an adequate sample of the remaining data. The dictionary will then be created approximately in one tenth the time it would take to create the dictionary for the entire volume of data. Moreover, the remaining 90% of data that is to be subsequently loaded will require significantly less disk write activity (slightly increasing CPU utilization) than if the uncompressed data were to be loaded into the table. This is simply because fewer pages will be required to contain the data. Figure 6 below shows the cumulative time spent performing the LOAD and REORG operations through both approaches. For details on the LOAD and REORG time spent per table for both the full and partial LOAD/REORG methods, please refer to Appendix C.

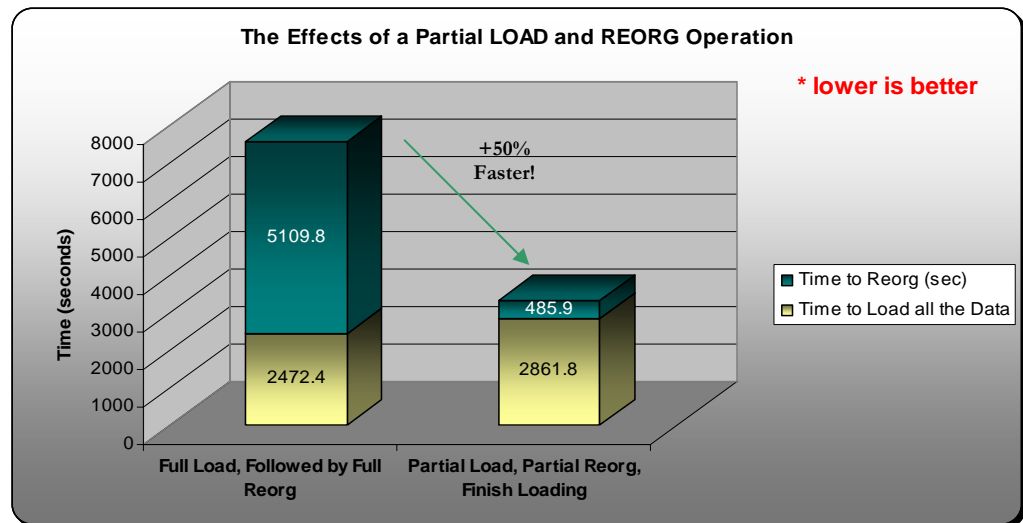


Figure 6 – The Effects of a Partial LOAD and REORG Operation

As expected the LOAD+REORG of 10% of the data is much faster. For our database setup, the first one tenth of the data is very similar to the rest of the data on all its tables. Thus, there is very little difference in space savings (just over 1%) when performing a REORG on 10% of the data vs. the entire data. Although loading the entire data set is slightly faster than loading 10% initially and 90% after the REORG (41 minutes vs. 47 minutes), we can see a tremendous benefit in time saved for REORG (nearly 10 times). This translates into an overall LOAD+REORG time savings of over 50% in exchange for an extra 1% of less efficiently compressed disk space.



Conclusions

DB2's row compression feature reduced storage requirements by 40% for the 362 GB DSS database we tested. Reduced storage requirements meant fewer disks could be used which in turn meant that the disk costs could be decreased by 50%.

When running our DSS workload (consisting of complex queries) a 6% throughput improvement was observed in the single stream case when using row compression. After analyzing CPU utilizations for the individual queries executed against the row compressed database, it was concluded that on average extra CPU cycles were used to perform useful compression and expansion work in exchange for ineffective idle cycles previously spent waiting for I/O. This improved efficiency of machine resources resulted in faster query execution times. However, in the relatively rare case when no idle CPU cycles are available, row compression may reduce performance somewhat, consuming otherwise useful CPU cycles as 'payment' for the reduction in storage requirements. In the multi-stream case, where idle CPU cycles were abundant, row compression consumed additional CPU cycles and turned them into a 25% overall throughput improvement, even though only half as many disks were used.

Row compression performs its best in I/O bound environments where there are free CPU cycles available that can be used to perform data compression and expansion. This feature works very well on DSS workloads composed of complex select queries where I/O access is mostly sequential and less random. It is important for the database design to use large RIDs because otherwise, the limit of 256 rows per page in regular RIDs can be more easily reached when row compression is enabled.

Because the compression dictionary is generated from the REORG command, it is important to understand the overhead implicated from creating the dictionary as well as assess which type of REORGs complete faster. In our experiments, it was more efficient to use scan-sort type REORG instead of index-scan REORG to create the compression dictionary. Moreover, we found that in the case of relatively homogeneous data, compression rates achieved by building the dictionary on the first 10% of the loaded data were almost as good as applying REORG to the entire table. Creating the dictionary based on the first 10% of the data saved over 50% of the total time required to load and reorganize the tables.

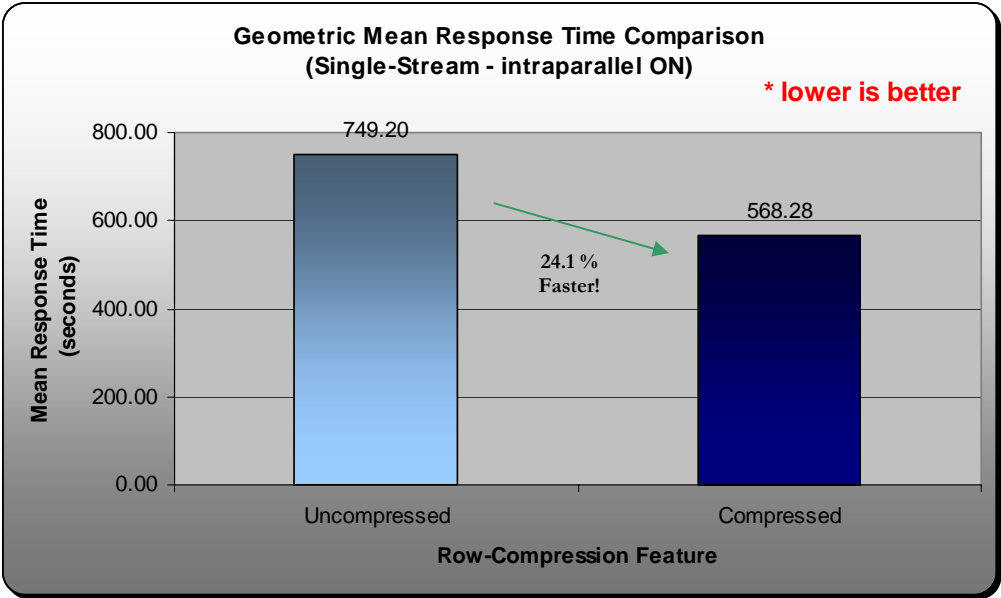
Through this whitepaper, we have shown that DB2 9's row compression feature can not only significantly reduce disk space cost, but can even improve performance, especially in the common case where the database system is highly disk bound.

Appendix A:

Database Table	Total Number of Rows	Average Uncompressed Row Size (bytes)	Average Compressed Row Size (bytes)	Percent Savings (%)
NATION	25	117	n/a	n/a
REGION	5	111	n/a	n/a
PART	60,000,000	150	54	63% saved
SUPPLIER	3,000,000	161	89	44% saved
PARTSUP	240,000,000	157	55	64% saved
CUSTOMER	45,000,000	179	99	44% saved
ORDERS	450,000,000	117	49	57% saved
LINEITEM	1,799,989,091	137	60	55% saved

APPENDIX A – Row Size Compression Details per Table

Appendix B:



APPENDIX B –Geometric Mean Response Time Comparison for Queries (intra_parallel ON)

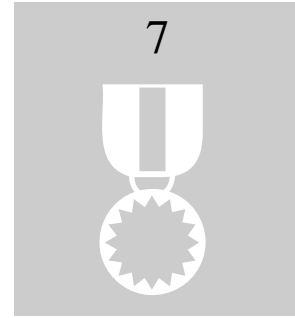
Appendix C:

TABLE	Time to Load 10% of Data (sec)	Time to REORG (sec)	Time to Load 90% of Data (sec)	Compression Ratio	Total Load + REORG Time (sec)
NATION	2.6	1.1	2.5	18%	6.2
REGION	2.7	1.3	2.6	5%	6.6
PART	5.1	37.7	32.3	67%	75.1
SUPPLIER	2.8	8.5	4.3	39%	15.6
PARTSUPP	26.5	82.1	249.4	64%	358.0
CUSTOMER	8.0	60.5	33.3	43%	101.9
ORDERS	36.4	86.4	411.9	57%	534.7
LINEITEM	163.5	330.4	1877.9	56%	2371.7
TOTAL	247.6	485.9	2614.2	57.0%	3469.6

Appendix C1 – Load 10%, Reorg table then Continue Loading

TABLE	Time to Load 100% of Data (sec)	Time to REORG (sec)	% Pages Saved	Total Load + REORG Time (sec)
NATION	2.7	1.1	54%	3.8
REGION	2.5	1.2	39%	3.8
PART	29.2	56.2	67%	85.4
SUPPLIER	4.2	68.1	44%	72.3
PARTSUPP	260.9	559.6	65%	820.5
CUSTOMER	32.9	125.2	45%	158.1
ORDERS	378.6	725.5	59%	1104.1
LINEITEM	1761.4	3572.8	57%	5334.2
TOTAL	2472.4	5109.8	58.09%	7582.2

Appendix C2 – Load 100% and REORG Entire Table



Acknowledgments

We are extremely thankful to Steve Rees (manager, DB2 Performance Quality Assurance) for his useful review comments and suggestions towards the betterment of this whitepaper. We would also like to thank Bill Minor (manager, DB2 DMS Development) and Mike Winer (STSM, DB2 DMS Development) for their ongoing support and focus on excellence during the development and planning of the row compression feature. Last, but not least, we would like to thank Sunil Kamath (manager, DB2 OLTP Performance) for his valuable input and for scheduling the availability of essential equipment to provide the test results for this technical whitepaper.

References

For more
information

- [1] DSS TPC-H Workload: <http://www.tpc.org/tpch/> TPC-H and the TPC-H benchmark specification are copyrighted by the Transaction Processing Performance Council and are used for this research paper according to the TPC's Policies for use of benchmark specifications. No data from this paper should be considered to be comparable to an official TPC-H result.
- [2] DB2 V9 Documentation
<http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp>



© Copyright IBM Corporation 2006
All Rights Reserved.
IBM Canada
8200 Warden Avenue
Markham, ON
L6G 1C7
Canada

Printed in United States of America
04-06

Neither this documentation nor any part of it may be copied or reproduced in any form or by any means or translated into another language, without the prior consent of all of the above mentioned copyright owners.

IBM makes no warranties or representations with respect to the content hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. IBM assumes no responsibility for any errors that may appear in this document. The information contained in this document is subject to change without any notice. IBM reserves the right to make any such changes without obligation to notify any person of such revision or changes. IBM makes no commitment to keep the information contained herein up to date.

The information in this document concerning non-IBM products was obtained from the supplier(s) of those products. IBM has not tested such products and cannot confirm the accuracy of the performance, compatibility or any other claims related to non-IBM products. Questions about the capabilities of non-IBM products should be addressed to the supplier(s) of those products.

IBM, the IBM logo, DB2, DB2 Universal Database, and AIX are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.