

# Performance Study of Rollout for Multi Dimensional Clustered Tables in DB2

Bishwaranjan Bhattacharjee  
IBM T.J.Watson Research Center  
19 Skyline Drive  
Hawthorne, NY, 10598  
001-914-784-7605  
bhatta@us.ibm.com

## ABSTRACT

In data warehousing applications, the ability to efficiently delete large chunks of data from a table is very important. This feature is also known as Rollout. Rollout is generally carried out periodically and is often done on more than one dimension or attribute. DB2 UDB V8.1 introduced a new physical clustering scheme called Multi Dimensional Clustering (MDC) which allows users to cluster data in a table on multiple attributes or dimensions. This is very useful for query processing and maintenance activities including deletes. Subsequently, an enhancement was incorporated which allowed for more efficient rollout of data on dimensional boundaries. This paper details a performance study of MDC rollout and delete and compares it against the conventional delete mechanism of a regular DB2 table. We discuss some of the key points noticed and the lessons learnt.

## Categories and Subject Descriptors

H.2.4 [Systems]: *Relational databases*

## General Terms

Algorithms, Measurement, Performance, Design.

## Keywords

Rollout, Bulk Deletes, Mass Deletes, Multi Dimensional

## 1. INTRODUCTION

Data warehouse sizes have been growing in leaps and bounds. An important concern is the storage costs associate with it. This is addressed by the periodic archiving of old data which might be accessed less often or by its summary removal from the database. Both methods require the mass delete of data from the warehouse. This is also known as Rollout or as Bulk Delete. The space thus freed up is used to make way for new data that is available. For example, a company might have a warehouse of 5 years of data. At the end of every month they might delete the oldest month of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*Proceedings of the First International Workshop on Performance and Evaluation of Data Management Systems (EXPDB 2006)*, June 30, 2006, Chicago, Illinois, USA.

Copyright 2006 ACM 1-59593-463-4 ... \$5.00.

data and bring in data for the latest month.

In the past, such mass deletes were usually done in a maintenance window when the system load was low. Like after midnight. Recent trends indicate users are moving towards a shorter time frame to perform this type of maintenance activities. Customers want their systems to be available almost 24 X 7 - even for a warehouse. Also, the amount of data being rolled out is becoming smaller but it is being done more frequently. These factors make an efficient online rollout mechanism very important for a database engine. The efficiency can be measured by various parameters, like, response time of a rollout, the amount of log space used, the number of locks required, the response time of a rollback of the rollout, how quickly the space freed can be reused and what kind of concurrent access to the table is allowed when the rollout is going on.

Rollouts might happen on more than one dimension. For example, one might want to rollout data based on shipdate at one time and orderdate on some other instance on the same table. One might want to remove data pertaining to a particular product or region etc. Also there might be further restrictions on these rollouts. For example, a user might ask to "delete orders older than 6 months provided they have been processed". The multi dimensionality of rollouts is thus an important characteristic.

In DB2 UDB V8.1, a new data layout scheme called Multi Dimensional Clustering (MDC) [1], [2], [3], [4] was introduced. This allows a table to be clustered on one or more orthogonal clustering attributes (or expressions). MDC initially supported a deletion capability based on logging every row that was deleted and any indexes updated to reflect the delete. This delete works for mass deletes as well as single row deletes. Subsequently in DB2 UDB V8.2.2 Saturn [5], an enhancement was incorporated - known as MDC Rollout - which allowed a user to more efficiently purge data from a table on dimensional boundaries. This paper discusses a performance study of MDC rollout and delete and compares it against the conventional delete on a non MDC table in DB2. We present performance figures from our study and discuss some of the key points noticed and the lessons learnt.

The rest of the paper is structured as follows. Section 2 describes the MDC feature introduced in DB2 UDB V8, Section 3 describes the new MDC Rollout enhancement, Section 4 compares this against other rollout mechanisms and related work, and Section 5 discusses the performance results of MDC Rollout and delete and compares it against non MDC delete. In Section 6 we discuss the lessons learnt and conclude.

## 2. MULTI DIMENSIONAL CLUSTERING IN DB2

Multi Dimensional Clustering (MDC) in DB2 UDB V8.1, allows a user to physically cluster records in a table on multiple orthogonal attributes or dimensions. The dimensions are specified in an ORGANIZE BY DIMENSIONS clause on a create table statement. For example, the following DDL describes a Sales table organized by region, year(orderDate) and itemId.

```
CREATE TABLE Sales(
date orderDate,
int region,
int itemId,
float price,
int yearOd generated always as year(orderDate))
ORGANIZE BY DIMENSIONS (region, yearOd, itemId)
```

Each of these dimensions may consist of one or more columns, similar to index keys. In fact, a 'dimension block index' will be automatically created for each of the dimensions specified and will be used to quickly and efficiently access data. A composite block index will also be created automatically if necessary, containing all dimension key columns, and will be used to maintain the clustering of data over insert and update activity. For single dimensional tables since the dimension block index and composite block index will turn out to be identical, only one block index is automatically created and used for all purposes.

In our example, a dimension block index is created on each of the region, year(orderDate) and itemId attributes. An additional composite block index will be created on (region, yearOd, itemId). Each block index is structured in the same manner as a traditional B+ tree index except that at the leaf level the keys point to a block identifier (BID) instead of a record identifier (RID). Since each block contains potentially many records, these block indexes are much smaller than a corresponding RID index on a non MDC table. For some instances, block index could be of 71 pages and 2 levels whereas a corresponding RID index for a non MDC table would be of 222,054 pages and 4 levels [2].

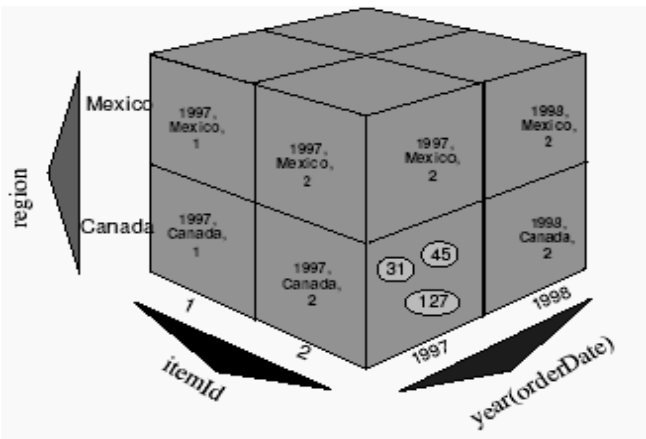


Figure 1: Logical view within a MDC table

Figure 1 illustrates these concepts. It depicts an MDC table clustered on the dimensions year(orderDate), region and itemId. The figure shows a simple logical cube with only two values for each dimension attribute. Logical cells are represented by sub-cubes in the figure and blocks by shaded ovals. They are numbered according to the logical order of allocated blocks in the table. We show only a few blocks of data for a cell identified by the dimension values <1997,Canada, 2>. Note that a cell without any records will not have any physical representation in the table.

A slice, or the set of blocks containing pages with all records having a particular key value as a dimension, will be represented in the associated dimension block index by a BID list for that key value. The following diagram illustrates slices of blocks for specific values of region and itemId dimensions, respectively.

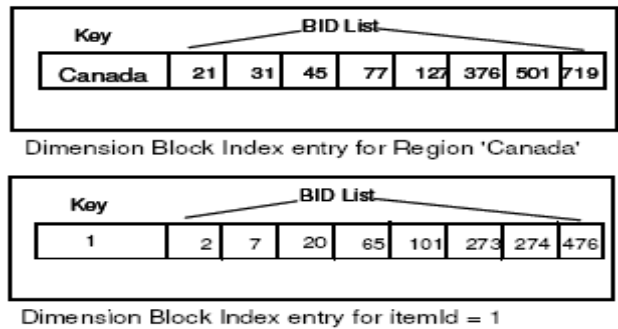


Figure 2: Logical view within a MDC table

In the example above, to find the slice containing all records with 'Canada' for the region dimension, we would look up this key value in the region dimension block index and find a key as shown in Figure 2(a). This key points to the exact set of BIDs for the particular value.

The DB2 UDB implementation was chosen by its designers for its ability to co-exist with other database features such as row-based indexes, table constraints, materialized query tables, high-speed load, mass delete, hash partitioned MPP as well as a SMP environment.

A delete of a record, entailed logging of the entire record and updating any record indexes defined on the table. The record index updates were logged too. The freed up space is available for reuse by the same unit of work even before the delete commits. After the commit, all transactions are free to reuse the space. If the delete ended up emptying the block in which the record resided, then the dimension block indexes were updated and logged. Thus a dimension block index is updated very few times compared to a corresponding record index on a similar non MDC table delete in DB2. This has a positive impact on response time of the delete and amount of logging needed.

MDC also introduced the concept of a Block Lock. The Block Lock is a locking mechanism which is between the Table Lock and a Record Lock in granularity. It allows for a block to be locked in various modes. Block Locks could escalate to Table Locks just like Record Locks do. However escalation of Record Locks to Block Locks is not currently supported.

Another data structure introduced in MDC was the Block Map. This stores information on the state of the blocks in a table. The information includes if the block is free, if it has been recently

loaded, if it is a system block, requires Constraint enforcement etc. This information is used, among other things, during inserts and loads to select blocks to insert/load into. Figure 3 shows an example blockmap for a table. Element 0 in the block map represents block 0 in the MDC table. Its availability status is 'U', indicating that it is in use. However, it is a special block and does not contain any user records. Blocks 2, 3, 9,10,13,14 and 17 are not being used in the table and are considered 'F' or free in the block map. Blocks 7 and 18 have recently been loaded into the table. Block 12 was previously loaded and requires constraint checking to be performed on it.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
U	U	F	F	U	U	U	L	U	F	F	U	C	F	F	U	U	F	L	...

Figure 3: Block Map entries

A MDC dimension block index can be ANDed and ORed with other dimension block indexes as well as any record based index defined on the table. A full description of how they can be combined can be found in [1], [2].

### 3. MDC ROLLOUT

In DB2 UDB V8.2.2 Saturn, a new feature called MDC Rollout was introduced. This allows for a more efficient delete of data along cell boundaries for MDC tables and builds on the good points of MDC delete. The rollout is submitted via a conventional SQL Data Manipulation Language (DML) delete statement. Thus users don't have to change their applications to tap this new feature. The compiler, under the covers, decides if the delete statement can be executed using MDC Rollout. If it can be, then it generates a plan for its execution using MDC Rollout else it switches to conventional MDC delete for that statement.

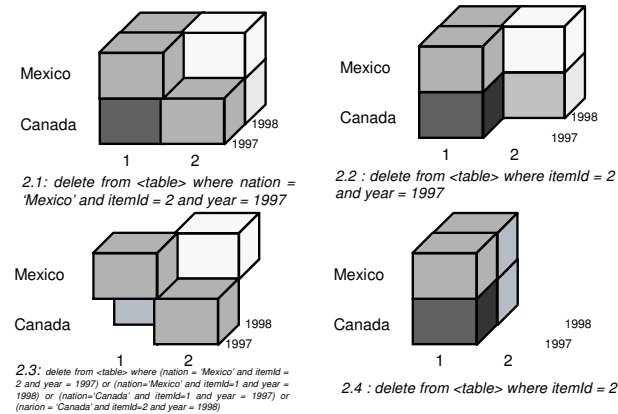


Figure 4: Example of rollout in a MDC table

Using this feature, multiple, full cells can be deleted in any combination as long as it can be described using delete DML statements. There are some restrictions but their description is beyond the scope of this paper. Figure 4 shows the result of 4 different Rollouts on the MDC table described in Figure 1. They depict the result of purging the table of individual cells to entire slices of data. While the rollout is executing, concurrent access to the table is permitted provided lock escalation to the table level has not occurred. The rollout itself acquires an intent exclusive Table Lock, and exclusive Block Locks on blocks being rolled out. It does not get any individual Record Locks on records being deleted. Thus the chances of lock escalation due to a rollout are

much reduced compared to non MDC and this has a positive impact on the concurrent access of the table when large rollouts occur.

In MDC Rollout, no record level logging is done as in conventional MDC delete. Instead, for all the records in the page, a single small log record is written. This indicates to the system that all records in the page have been deleted. Further Meta information stored in the page as well as the first page of the block is updated to indicate all records have been deleted and thus the pages of the block are free. This change is also logged.

MDC Rollout tends to process a block at a time as described above. When a block is rolled out, its corresponding entry in the Block Map is marked rolled out and the inuse bit is reset. This indicates that this block cannot be reused by the same transaction until the rollout is committed. All the Dimension Block Indexes are updated to reflect the fact that the block is no longer associated with its cell. It is to be noted that the block is still associated with the table after a commit and is reusable for any cell. It can be delinked from the table and returned to the tablespace by a table reorg.

Any record based indexes defined on the table are updated one record at a time. For each record, its entry in all the rid indexes is removed and this change is logged.

### 4. THE CURRENT STATE OF THE ART

The delete mechanism employed by database engines generally works horizontally, on a tuple at a time. In this a record is deleted and the defined indexes are updated one by one to reflect the delete of that record. For mass or multiple record deletes, one iterates over all records to be deleted in a similar fashion. The non MDC delete in DB2 UDB V8.1 is an example of that.

Other technologies in this area include the Detach mechanism for range partitioned tables. Range partitioning is available in some commercial database systems like DB2 zOS [5] and Oracle [6]. In this, a table is partitioned into ranges of values on a specified attribute. Detaching a partition would be the equivalent of delinking all the data of the partition from the table. Any local indexes on that partition are also thrown out. If there are global indexes defined, these will have to be updated. Detach tends to be a Data Definition Language (DDL) level command and application have to explicitly specify they want to detach. This will, in most implementations, result in getting an exclusive lock on the table for the duration of the Detach. Thus, during the Detach, concurrent access to the table is generally disallowed. Also, as explained in [7], partitioning for this purpose tends to be single dimensional and one cannot rollout on a granularity lower than a single partition or on an attribute not related to the partitioning attribute. For example, if a table described in Figure 1 is partitioned on year (orderDate) or region, then none of the 4 cases mentioned in Figure2 would qualify for a Detach. Further, if the table is partitioned on itemId, then except 2.4 none of the rest would qualify.

Some database engines implement the base table in the form of a B+ tree itself. The NonStop SQL [16], [17], [18] is an example of this. Here additional secondary indexes are allowed and will have to be updated on a delete. To speed this up, multiple indexes could be updated in parallel.

A mechanism for bulk deletes was explained in [7]. The aim of this method was to improve the response time of the delete. This is an important consideration for mass deletes. However, it did not address the issues of resource consumption for logging or locking or the response time of the rollback of the delete. It also assumed the base table would be exclusively locked and the indices would be offline for the duration of the delete. The method described, is based on vertical deletes of the base table and any rid indexes defined on it. This is to be contrasted with the conventional method of deleting the table record and updating the rid indexes iteratively for all qualifying records.

It is to be noted that while not directly related to rollouts, there has been a lot of work on analysis and implementation of deletes on indices and related issues [8],[9]. Bulk load (also know as Rollin) is the opposite of Rollout. This has also been studied in a number of papers [10],[11],[12]. Deleting records from tables and the management of free space has been discussed in [13]

## 5. PERFORMANCE EVALUATION OF ROLLOUT

There are various parameters on which a rollout could be evaluated. Clearly the response time of the rollout is very important. In addition, for any mass delete mechanism which allows for concurrent access to the table, parameters like the number and type of locks acquired and amount of logging is important. Also equally important, is the impact of record level indexes on all these parameters. These record level indexes could be of different clustering. Further the response time of the rollback of the rollout is also an interesting parameter. In this study all these parameters have been covered.

A 10GB TPCB [14] LINEITEM table was used for the experimental evaluations. The table consisted of approx. 60 million records with almost uniform distribution over a 7 years span on column L\_SHIPDATE. A basic non MDC version was loaded with data physically clustered on L\_SHIPDATE and with a record index defined on the same. This record index had 100% cluster ratio. A corresponding MDC table was created with L\_SHIPDATE as a single dimension. This resulted in the automatic creation of a Dimension Block Index on that column. Subsequently 3 different record indexes were created on the MDC table at different times for the experiments. Table 1 provides details on the columns on which the indexes were created and some of their important statistics. Both tables resided in the same tablespace. Table 2 provides details about the experimental setup used.

**Table 1. Details of indexes used in the evaluation**

RID INDEX NAME	NLEAFS	NLEVELS	CLUSTER RATIO
L_COMMITDATE	98640	4	13
L_RECEIPTDATE	98640	4	38
L_PARTKEY	102594	4	4

To study the impact of multi dimensions, a MDC table was created with dimensions on L\_SHIPDATE, L\_LINESTATUS and

L\_SHIPINSTRUCT and this was compared against a non MDC table with 3 record indexes on those individual columns.

**Table 2. Experimental setup details**

Operating System	64 bit AIX 5.2.0.0
DB2 Instance	Single node DB2 V82 FP9 (DB2 V8.2.2 Saturn) with MPP and SMP turned off
File System	JFS2 with CIO enabled
Disk Subsystem	Shark array with 4 disks
DB2 Tablespace Details	DMS FILE with "NO FILE SYSTEM CACHING"; Page size of 4KB; Extent size of 16 pages; Bufferpool of 30000 pages
Hardware System	IBM 7026-6M1 with 16GB of main memory
Processors	8 x 64 bit PowerPC_RS64_IV @ 752 MHz
DB2 Registry Variables	DB2_MDC_ROLLOUT=Y/N
TPCH Scale Factor	10

The experimental evaluation consisted of a study of how well the basic MDC delete performs in comparison to a delete on a non MDC table. This was followed by a comparison of the Rollout enhancement in DB2 V.8.2.2 with the basic MDC deletes.

To this end 2 delete statements, SR and LR, described in table 3 were used. SR represented a small rollout with 1 month of data being deleted. This corresponded to 1.3% of the total table. LR represented a large rollout with 3 years of data being deleted. This corresponded to 43.1% of the total table.

**Table 3. Details of the delete statements used in the evaluation**

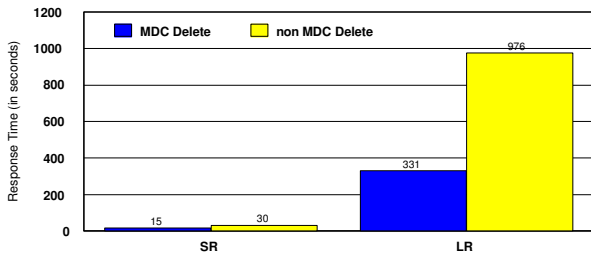
SR	<i>1 month out of 84 months = 1.3% of the table</i>	<i>delete from lineitem where L_shipdate between '01/02/1995' and '02/02/1995'</i>
LR	<i>36 months out of 84 months = 43.1% of the table</i>	<i>delete from lineitem where L_shipdate between '01/02/1992' and '01/02/1995'</i>

The deletes were run from the DB2 command line with the -c option and were timed using the AIX time command. The DB2 Monitor snapshots with the BUFFERPOOL, UOW and LOCK options were used to determine the locking, logging, physical and logical read statistics for the deletes. All deletes were preceded by a db2stop and db2start to clear the bufferpool of its contents.

### 5.1 Comparison of MDC and non MDC deletes

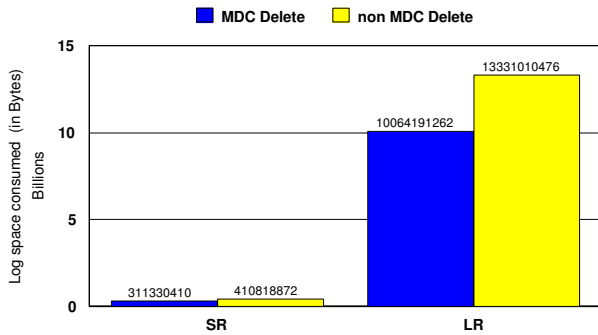
For this purpose we executed LR and SR on the MDC and non MDC tables. The non MDC table had a rid index on L\_SHIPDATE and the MDC table had just the automatically created dimension block index on L\_SHIPDATE defined on it. Figure 5 show the response time of LR and SR on both the tables.

As Figure 5 indicates, MDC delete performed 2 to 3 times better than a non MDC delete. The cause of this difference can be traced to the use of dimension block indexes by MDC in comparison to the record index by non MDC. While the amount of base table record processing (delete, logging etc) was identical in both cases, for MDC, the dimension block index had to be updated and logged only when a block became free and thus had to be removed from the index. In comparison, for non MDC, the corresponding record index had to be updated and logged for every record. With 16 pages to a block and approx. 25 records per page, the MDC block index had to be updated approx. 400 times less frequently compared to the non MDC table.



**Figure 5. Response Time of LR and SR on MDC and non MDC**

The impact of this on the amount of logging that is needed is visible in Figure 6. The non MDC deletes took about 30% more log space than the corresponding MDC delete. Occupying less log space increases the likelihood of a large rollout successfully finishing for a given log size. It also has an impact on the amount of concurrency possible in a system apart from resulting in the obvious disk space savings. It also helps a rollback run faster.

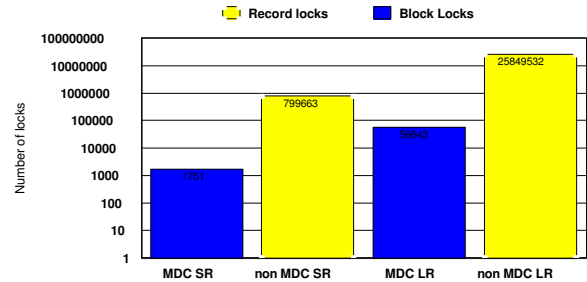


**Figure 6. Log space consumption for LR and SR on MDC and non MDC**

Figure 7 shows the number and type of locks acquired by both delete methods for deletes LR and SR. Both methods acquired an Intent Exclusive Table Lock. However the MDC delete acquired Exclusive Block Locks whereas the non MDC delete acquired Exclusive Record Locks. Numerically the number of locks acquired by MDC was about 450 times lower. This drastic reduction in the number of locks acquired has a positive impact on delete performance as well as on concurrency. The non MDC delete has a much higher chance of escalating into an Exclusive Table Lock with a similar lock list space.

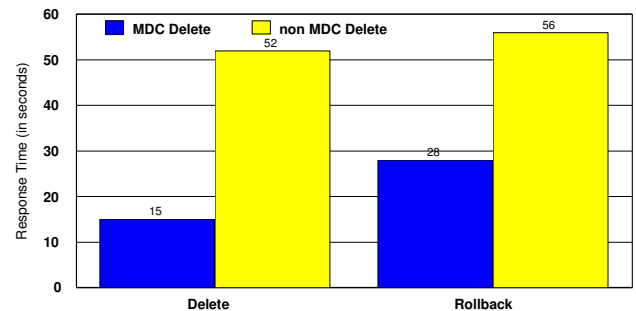
These figures indicate the inherent advantage the MDC architecture provides for deletes. With n MDC dimensions, one can do away with n record indexes that would have otherwise

been defined on the attributes. This would result in significant savings in response time of a delete, logging space consumed as well as locking resources. Figure 8 shows the response time of the delete and rollback of SR on the 3 dimensional MDC table with dimensions (L\_SHIPDATE, L\_LINESTATUS, L\_SHIPINSTRUCT) and the non MDC table with rid indexes on L\_SHIPDATE, L\_LINESTATUS and L\_SHIPINSTRUCT. The MDC delete was 3 times faster than the non MDC delete. Also the MDC delete with 3 dimensions performed as well as the delete with 1 dimension from Figure 5. Whereas the corresponding non MDC delete with 3 rid indexes took almost twice as long as with 1 rid index.



**Figure 7. Locking for LR and SR on MDC and non MDC**

An example where the value of MDC deletes can be seen in the benchmark outlined in the Winter Corporation's Whitepaper [15]. This benchmark required very high delete rates to be maintained while high volume record ingests were happening in parallel. Here, 3 dimensional MDC tables were used with no additional rid index defined on them. This had a very positive impact on the deletes that had to be done as part of the benchmark and helped DB2 meet the benchmark requirements for deletes.



**Figure 8. Response time for deletes and rollbacks for SR on 3D MDC and non MDC**

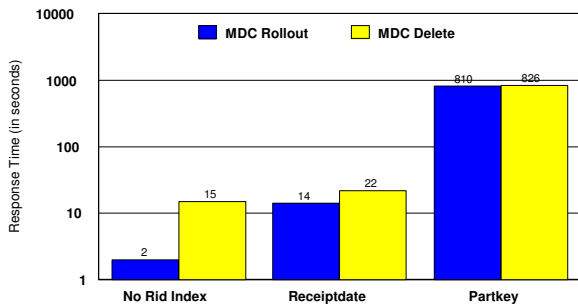
Note that any additional rid index created on a non dimension column for MDC and non MDC would add a similar overhead on these parameters for both cases. Studying the impact of these additional indexes is important and will be described in a later section.

The performance gain of MDC delete over a non MDC table would depend on the number of records that would fit in a block/extent. This will dictate the number of record locks which will be replaced by a block lock. It will also dictate the amount of index logging that is saved for the record indexes which are replaced by the Dimension Block Indexes. Other important factors which will also come into play are the number of MDC dimensions and their data types.

The next section discusses the performance evaluation of the MDC Rollout enhancement which went into DB2 V8.2.2 Saturn over the base MDC delete. In a comparison of MDC Rollout over the non MDC delete, all that was discussed in this section would hold and would generally be additive.

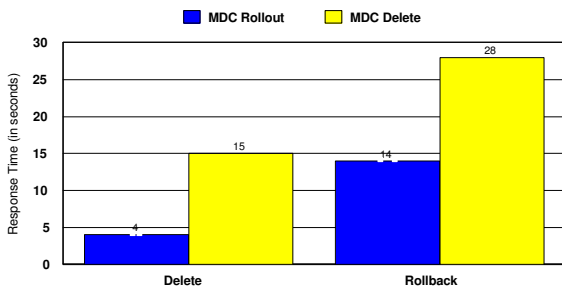
## 5.2 Comparison of MDC delete and MDC Rollout

The vanilla MDC delete is useful for all delete scenarios including delete of a subset of a cell or even a single record. MDC Rollout is useful when one wants to delete entire cells. It builds on the benefits of MDC delete and incorporates certain optimizations which are possible for the subset of deletes it handles. For the comparison of MDC delete and rollout, delete statements LR and SR were used on the single dimensional MDC LINEITEM table. These statements tend to delete entire cells.



**Figure 9. Response Time of SR with different Rid Index Clustering for MDC rollout and delete**

Figure 9 shows the response time of SR with indexes of different clustering defined on the MDC table. Details on these indexes can be found in Table 1. When we don't have any additional rid indexes defined on the table, rollout performs more than 7 times better than delete. The cumulative impact of these figures and those in Figure 3 means that MDC rollout is about 15 times faster than a non MDC delete in such scenarios. Figure 10 shows the impact when multi dimensions come into play and dimension block indexes replace corresponding rid indexes. We see that the gains are consistent and significant.

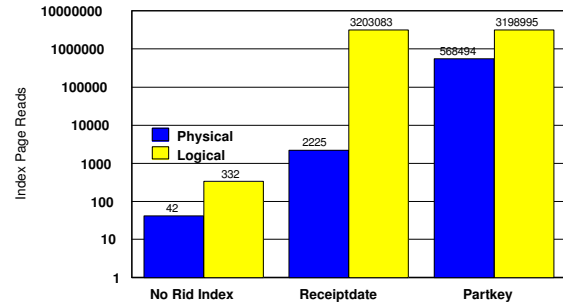


**Figure 10. Response time for deletes and rollbacks for SR on 3D MDC table using MDC delete and rollout**

If the situation requires additional rid indexes over and above the dimensional block indexes, then the response time of rollout would be a function of the cluster factor of these indexes and the number of rid indexes defined. In Figure 9, when the receiptdate rid index of 38% clustering was added to the table, the response time gains dropped to about 33% for rollout over delete. When

the receiptdate rid index was replaced by a partkey rid index of 4% clustering, the gains dropped to 2% for rollout over delete.

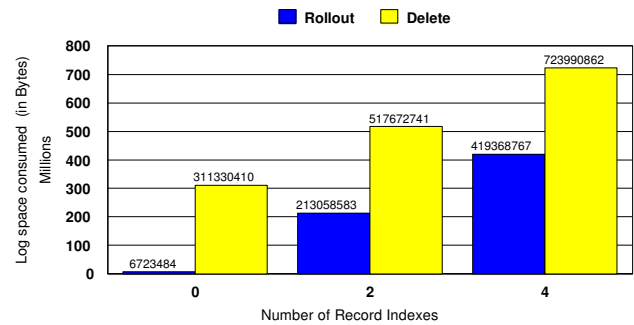
The reasons for this are visible in Figure 11. This shows the logical and physical index page reads that needs to be done as part of the index updates for rollout and delete. With the receiptdate index of 38% clustering, one ended up getting good bufferpool hit ratio for the index pages that were needed. This explains the substantial difference between the logical and physical index page reads for receiptdate. However, for the partkey index of 4% clustering, the amount of physical reads that needed to be done for almost the same number of logical reads was substantial. This accounted for the drop in response time for the partkey index.



**Figure 11. Index Page Reads For SR with various indexes**

A current way to tackle the impact of very badly clustered rid indexes like partkey on delete would be by selectively dropping and recreating those indexes. For large deletes, this will actually improve the overall response time of the delete. A recreate of the partkey index on the MDC table would take about 384 seconds and a subsequent runstats on the table and index about 514 seconds. So while for SR it would not result in a response time gain, for a large rollout like LR it certainly will.

Besides response time of the rollout, other parameters to consider are the logging, locking and rollback of the rollout. There is no improvement from the locking point of view for MDC rollout over MDC delete. However it does retain the same good characteristics of getting exclusive block locks instead of record locks. The advantages of this have been discussed in Section 5.1.

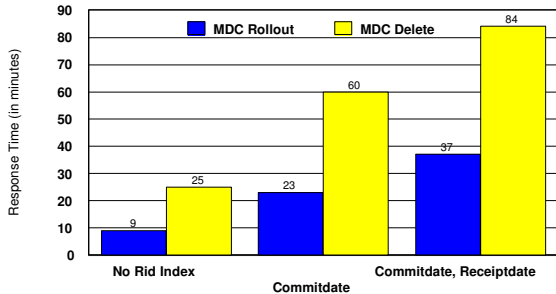


**Figure 12. Log spaced consumed for SR on MDC delete and rollout**

Figure 12 shows the log space consumption for MDC delete and rollout for SR. Since MDC rollout writes 1 log record for a page of data being deleted rather than 1 log record for a record, the total number of log records being written is significantly lower. With about 25 records for a page, it will be 25 times lower. Also the amount of data being logged will also be small. Every log

record for a MDC delete will contain the entire record. Whereas the MDC rollout log record will contain page level meta data only. Thus the performance gains of MDC rollout over delete would be a function of the number of record in a page as well as the record size. The logging savings of MDC rollout over a non MDC delete is a cumulative savings of Figures 6 and 10.

Figure 13 shows the response time of a rollback of delete LR for MDC rollout and MDC delete. With a much lower number of log records to undo and a much smaller log file to read, the rollback of a MDC rollout runs much faster. Like for logging, the gains for MDC rollout over a non MDC delete would be the cumulative. Note that one would get similar gains for a rollforward operation.



**Figure 13. Rollback time of MDC Rollout and Delete on LR**

## 6. CONCLUSION

The MDC rollout and delete mechanism consume significantly lower amount of system resources compared to a conventional non MDC delete in DB2. This includes locking and logging resources.

In situations where all record indexes are replaced by dimension block indexes, the response time of MDC rollout is an order of magnitude better than a non MDC delete. When multi dimensions are used in an MDC table, it results in multiple dimension block indexes being created. These would be a replacement for equivalent record indexes in non MDC. And in situations like this MDC rollout performs significantly better.

When additional rid indexes are created on a MDC table, it results in the response time gains being comparatively lower. This is especially true when there are very badly clustered rid indexes defined. Nevertheless one still continues to see significantly lower logging and locking resources being consumed.

Clearly, an important area to look into is ways and means of reducing the cost of updating these badly clustered rid indexes which might be defined on a MDC table. This has to be done without taking the table or the indexes off line.

## 7. ACKNOWLEDGMENTS

The author would like to acknowledge the help of Raj Kumar Rana of IBM T.J. Watson Research in generating some of the performance numbers.

## 8. REFERENCES

- [1] Padmanabhan, S., Bhattacharjee, B., Malkemus, T., Cranston L., Huras, M., "Multi-Dimensional Clustering: A New Data Layout Scheme in DB2", Proceedings of SIGMOD 2003.
- [2] Bhattacharjee, B., Padmanabhan, S., Malkemus, T., Lai, T., Cranston, L., Huras, M., "Efficient Query Processing for Multi-Dimensionally Clustering Tables in DB2", Proceedings of VLDB 2003
- [3] Lightstone, S., Bhattacharjee, B., "Automating the design of multi-dimensional clustering tables in relational databases", Proceedings of VLDB 2004
- [4] Malkemus, M., Padmanabhan, S., Bhattacharjee, B., Cranston, L., Lai, T., Koo, F., "Predicate derivation and Monotonicity detection in DB2 UDB", Proceedings of ICDE 2005
- [5] <http://www-306.ibm.com/software/data/db2>
- [6] <http://www.oracle.com>
- [7] Gartner, A., Kemper, A., Kossman, D., Zeller, B., "Efficient Bulk Deletes in Relational Databases", Proceedings of the ICDE 2001
- [8] Jannink, J., "Implementing deletion in B+trees", SIGMOD Record, Mar. 1995.
- [9] Johnson, T., Shasha, D., "B-trees with inserts and deletes: Why free-at-empty is better than merge-at-half", Journal of Computer and Systems Sciences, 1993
- [10] Van den Bercken, J., Seeger, B., Widmayer, P., "A generic approach to bulk loading multidimensional index structures.", Proceedings of the VLDB 1997
- [11] Wiener, J., Naughton, J., "Bulk loading into an OODB: A performance study", Proceedings of the VLDB 1994
- [12] Wiener, J., Naughton, J., "OODB bulk loading revisited: The partitioned-list approach", Proceedings of the VLDB 1995
- [13] McAuliffe, M., Carey, M., Solomon, M., "Towards effective and efficient free space management", Proceedings of the SIGMOD, 1988
- [14] <http://www.tpc.org/tpch>
- [15] Dorin, R., Winter, R., "Benchmarking an Extremely Large Decision Support requirement: Proof Point of Scalability with IBM and DB2", Winter Corporation White Paper 2005
- [16] <http://www.hp.com/go/nonstop>
- [17] Leslie, H., Jain, R., Birdsall, D., Yaghmai, H., "Efficient Search of Multi-Dimensional B-Trees", Proceedings of the VLDB 1995
- [18] Englert, S., Gray, J., Kocher, T., Shah, P., "A Benchmark of NonStop SQL Release 2 Demonstrating Near-Linear Speedup and Scaleup on Large Databases", Technical Report 89.4, Tandem Part No 27469, May 1989