WHITE PAPER

WinterCorp

SPONSORED
RESEARCH
PROGRAM

WINTER CORPORATION

ADVANCES IN DATA WAREHOUSE PERFORMANCE

I/O Elimination in DB2

WINTER CORPORATION

ADVANCES IN DATA WAREHOUSE PERFORMANCE

I/O Elimination in DB2

RICK BURNS
RICHARD WINTER
May 2007

411 WAVERLEY OAKS ROAD, SUITE 327
WALTHAM, MA 02452
781-642-0300

Executive Summary

TODAY, DATA WAREHOUSE USERS face novel and demanding challenges. Data is growing rapidly due to diverse market forces, including competitive pressures for increasingly detailed analysis and more stringent regulatory demands. User populations are rising sharply to support new activities like operational business intelligence and self-service applications. These demands not only push data warehouse scale to new data size frontiers, they also generate richer data relationships and increasingly dynamic and complex queries against them. These demands also produce shrinking data latency windows to support access to ever fresher data.

Along with the growing demand for better business intelligence, businesses face increasing competitive pressures to contain IT costs. The challenge is nothing short of improving the business value of data while simultaneously reducing IT spending.

These many faceted database requirements tax traditional performance and scalability techniques employed by data warehouse products. At the same time, hardware changes have opened new efficiency opportunities to support the data access needs of large-scale data warehousing. Data clustering strategies and intelligent software compression algorithms have emerged as highly effective techniques to enable database products to take advantage of these changes. These techniques improve the performance of business intelligence queries by radically reducing I/Os needed to resolve them.

With Version 9 of IBM's DB2 Warehouse on Linux, UNIX, and Windows, IBM offers an innovative set of features that work together to provide unmatched data clustering and storage density, to benefit many types of data warehouse queries. These features include robust data clustering on multiple data attributes, granular partition elimination, efficient data windowing on range-partitioned data, efficient data compression, scalable processing with high degrees of parallelism, and automated design tools to make it easier for database designers to make the best use of these features to meet their performance needs.

The innovative combination of capabilities found in IBM DB2 Warehouse work together synergistically to satisfy increasingly dynamic and demanding data warehouse requirements. These include high performance, highly selective data access across many types of queries, low latency data ingest to support real-time data warehousing, and storage optimization to lower operating costs. The performance gains provided by these features translate directly into improved business value for DB2 customers. With its unique and innovative combination of database design solutions, DB2 offers compelling options to many common, but complex, business intelligence requirements.

In short, IBM's DB2 Warehouse helps address the twin challenges facing enterprises today. Namely, improving the value they derive from the torrents of data they process every day, while lowering costs at the same time.

Table of Contents

Executive Summary 3
Table of Contents 4
1 Introduction 5
2 DB2: Architected for Performance 6
2.1 Architectural Keys to DB2 Data Warehouse Performance 6
2.2 Database Partitioning 7
2.3 Table Partitioning. 7
2.4 Multi-Dimensional Clustering. 9
2.5 Row Compression 10
2.6 Putting It All Together. 11
3 The DB2 Balanced Warehouse 12
4 DB2 Architecture in Practice 13
5 Conclusions. 16

1 Introduction

Today's data warehouse users face novel and demanding challenges. Data is growing exponentially[1] due to competitive pressures for increasingly detailed analysis and more stringent regulatory demands. User populations are rising sharply to support new activities like operational business intelligence and customer and supplier self-service. These demands not only push data warehouse scale to new data size frontiers, they also generate increasingly dynamic and complex queries and richer data relationships to support them. These demands also yield shrinking data latency windows to support access to ever fresher data.

Along with the growing demand for better business intelligence, businesses face increasing competitive pressures to contain IT costs. The challenge is nothing short of doing more with less, improving the business value of data while reducing IT spending at the same time.

These many faceted database requirements tax traditional performance and scalability techniques commonly employed by data warehouse products. Historically, transactional systems used indexes to retrieve individual rows or small result sets quickly from very large data repositories by dramatically reducing the number of I/Os needed to find them. Business intelligence queries typically exhibit different data access patterns, usually touching more data, grouping it by multiple attributes, and returning larger result sets. Extending indexed data access to support intersections and unions of multiple indexes to perform the complex restrictions required to slice data by multiple attributes has successful improved the performance of business intelligence queries. Parallelizing these data access algorithms has allowed this approach to scale as database size has grown. While improving performance however, these techniques in themselves do not solve the cost containment problem.

At the same time however, hardware changes have opened new efficiency opportunities to support the data access needs of large-scale data warehousing. The evolution of disk architectures has dramatically shifted the balance between performance of random and sequential I/O in favor of sequential I/O. Low memory costs have enabled larger data caches on disks and disk controllers to better support large sequential I/O, while 64-bit processing has enabled large main memories, supporting multi-block fetches initiated by application software. Finally, processing power has grown more rapidly that I/O bandwidth, enabling effective use of more compute cycles per data block.

Data clustering strategies and intelligent software compression algorithms have emerged as highly effective techniques to enable database products to take advantage of these hardware changes. These techniques further improve business intelligence query performance by dramatically reducing I/Os needed to resolve such queries. With Version 9 of IBM's DB2 Warehouse Edition on Linux, UNIX, and Windows, IBM offers an innovative set of features that work together to provide unmatched data clustering and storage density to offer dramatic performance benefits many types of data warehouse queries.

These features include robust data clustering on multiple data attributes, granular partition elimination, efficient data windowing on range-partitioned data, efficient data compression, scalable processing with high degrees of parallelism, and automated design tools to make it easier for database designers to make the best use of these features to meet their performance needs. Just as important, use of these features to improve performance is transparent to database users, including

---

[1] See WinterCorp 2005 TopTen Survey results in 2005 TopTen Program Summary. The largest data warehouse in the 2005 survey exceeded 100 TB, and was three times larger than the biggest data warehouse in the 2003 survey.

front-end business intelligence tools, so performance features can be added as needed without modifying applications or user interfaces.

While these features originated in several different parts of IBM's far flung operations – it extensive research labs, its deep mainframe experience, and its acquisition of companies like Informix and RedBrick – this unique combination of features now available in IBM DB2 Warehouse work together synergistically to satisfy increasingly dynamic and demanding data warehouse requirements. These include high performance data access across many types of queries, low latency data ingest to support real-time data warehousing, and storage optimization to lower operating costs. This paper examines the architecture of these key features and analyzes how they complement one another to provide significant customer benefits.

These performance gains translate directly into improved business value for DB2 customers. Besides improving the productivity of analytic workers, better query performance enables companies to pinpoint business problems and identify emerging business opportunities faster. Reduced data latency enables improved customer service and better overall business responsiveness. Reduced resource consumption can ea able broader and richer use of advanced business analytics at dramatically lower costs.

In short, IBM's DB2 Warehouse provides a robust database platform that helps address the twin challenges facing enterprises today. Namely, improving the value they derive from the torrents of data they process every day, while lowering costs at the same time.

2 DB2: Architected for Performance

2.1 ARCHITECTURAL KEYS TO DB2 DATA WAREHOUSE PERFORMANCE

IBM DB2 Warehouse attacks the performance of business intelligence queries from multiple angles, to shorten overall query response time and to reduce the system resources required to resolve the query. To shorten response time To improve query efficiency DB2 uses range partitioning and multi-dimensional clustering (MDC) to group data by shared attribute values, and data compression to reduce the volume of the data. These features provide targeted data access while minimizing I/O. Used separately or in combination these features can dramatically lower system costs and improve response time.

DB2's balanced data distribution via hash partitioning, and support for large-scale parallelism implements an effective divide-and-conquer strategy that equalizes work across many parallel query processors to reduce overall query response time. This work-splitting strategy, called database partitioning, is transparently coupled with DB2's unique ability to physically cluster data on multiple dimensions within each partition, and to group data by value range. Both techniques enable highly selective access to data, limiting query I/O to relevant table partitions, thereby dramatically reducing the work needed to resolve many queries.

Table partitioning also supports efficient partition roll-in and roll-out for data windowing applications. In addition, MDC's ability to group data by multiple attributes supplants the need for separate column indexes on those attributes, both reducing storage requirements and lowering update costs. Finally, good data compression techniques also reduce data volume, lowering storage costs and further reducing the I/Os needed to resolve queries.

To deliver the anticipated benefits however, these features must each be used and combined effectively (*Figure 1*). An enhanced automated database design wizard helps database architects to choose an optimized physical database design, based on their anticipated workload, faster and

with less effort. The database design wizard can also be used for ongoing recommendations as the data structure and workload evolves.

These features are key architectural elements of DB2 performance for business intelligence queries. The sections below first look at the architecture and implementation of each feature individually, and then examine the effects on performance and resource consumption of combining them.

2.2 DATABASE PARTITIONING

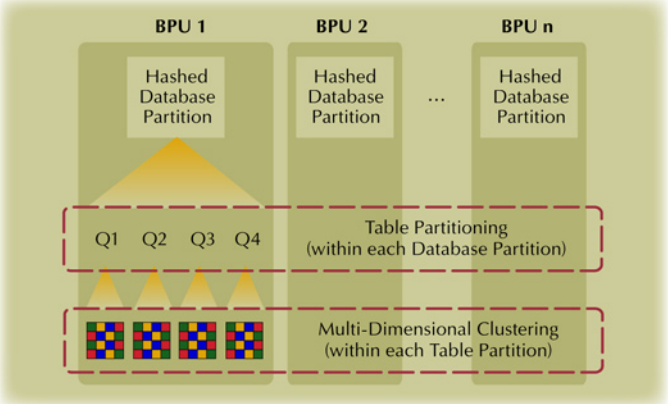
DB2 is a massively parallel processing (MPP) database system. DB2 divides system resources—processors, memory and storage –into equal sized segments, called data partitions, that are managed by an independent database process. DB2 calls this database partitioning, and it is fundamental to the DB2 architecture.

Each database partition and its associated memory and storage is called a balanced partition unit (BPU). A DB2 database is a collection of BPUs. BPUs usually contain 150-200 GB of user data, and typically consist of a processor with 4 GB of memory and 500 GB of attached storage. DB2 BPUs are usually deployed on clusters of small SMP servers with multiple partitions allocated per server, each logically separate from one another.

Table data is distributed among the available BPUs based on a partitioning key that consists of one or more columns per table. Selecting partitioning keys with unique or nearly unique values produces a balanced distribution of data, and hashing the partitioning key allows DB2 to distribute data evenly across the available database partitions in a deterministic fashion. Each row in a partitioned table is hashed on its partitioning key to one of 4096 has buckets; each bucket is mapped to a specific BPU. In cases of heavily skewed data, a custom partition map can be used. In DB2, indexes are also partitioned.

Balanced data distribution is key to efficient parallel processing because it equalizes the work to retrieve data in each partition. This enables many concurrent query processes in each partition to complete in roughly equivalent time, optimizing query response time. It is also key to scalability, allowing data size to grow while maintaining roughly constant response times simply by adding the appropriate number of BPUs.

Figure 1: Tiered data clustering in DB2



2.3 TABLE PARTITIONING

Within each database partition (BPU), data can be sub-partitioned by range, clustered on multiple dimensions, or both (*Figure 1*). In range partitioned tables—called table partitioning in DB2 to distinguish it from database partitioning—data is clustered by range value, with every parallel BPU containing all data ranges. Data within a given range, say a time period or a sequence of zip codes, is stored as an unordered heap. Data that falls in different ranges is stored in separate heaps. Each data range can occupy its own tablespace, or multiple data range can share a tablespace. Indexes span table partitions; each index contains entries for rows in all range partitions, ordered by range. DB2 Version 9 introduced the table partitioning feature.

A key benefit of table partitioning is the elimination of partitions by the query optimizer for queries that use the partitioning key in the query predicate (*Figure 2*). For table scans, only the relevant subset of partitions are retrieved, dramatically reducing the I/O required to resolve the query. In the case of index scan retrievals, if range partitioning replaces an index that would otherwise be used, the query is resolved using fewer indexes, again eliminating I/O for the unneeded index. Further, since the row IDs (RIDs) contain the partition identifier, and are stored in order in the index, only the RIDs for the table partitions of interest in the remaining indexes need to be retrieved. In addition, using table partitioning in place of an index speeds up insert and update operations by reducing index maintenance.

Figure 2: Partition Elimination

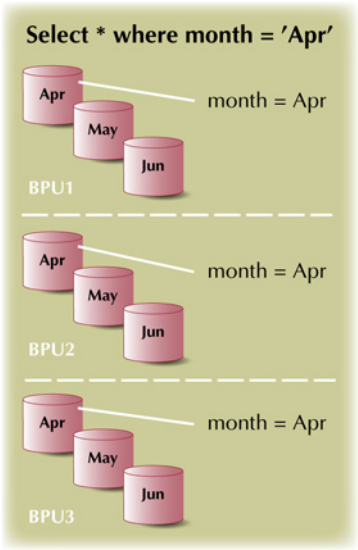
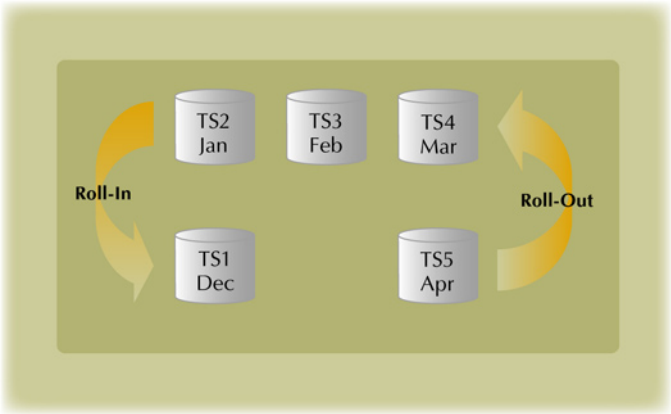


Table partitioning also supports roll-in and rollout operations on tables that maintain a fixed window of data, for example, the most recent 24 months of transactions (*Figure 3*). Windowing is a common data warehouse strategy that allows new data to be added to a table quickly, and obsolete data to be dropped from the table just as quickly.

Recent data may either be inserted into the latest range partition, or accumulated in a separate table until the end of the period. In the latter case, the table is then attached as a new range partition to the main table via the new *attach partition* option of the *alter table* statement. Attach involves no data movement, just system catalog updates. A follow-up operation is then required to validate the rows in the new partition and to update existing table indexes with data from the new partition. This operation is performed online, allowing earlier partitions to be fully accessible. Following this operation, the new partition is available for query.

Figure 3: Partition Roll-in and Roll-out



To remove, or rollout, an existing partition from the table, the *detach partition* option of the *alter table* statement is used. The detached partition becomes a separate table the can be archived, dropped or reused as desired. Data in the detached partition becomes unavailable to queries immediately upon completion of the detach operation, while data in the remaining partitions is accessible immediately following the detach operation. Index entries for rows in the detached partition immediately become invisible to queries, and will be automatically cleaned up to reclaim index space by an asynchronous background process.

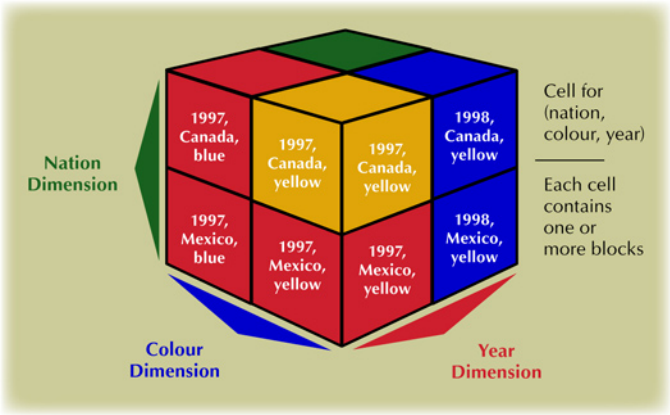
Any Materialized Query Tables (MQTs) which use the immediate refresh option that contain data from deleted partitions, are taken offline automatically until they are incrementally refreshed via a manual operation. This may affect query performance until the independent refresh completes, but does not affect access to base table data. If MQTs are based on the same time-cyclic roll-out

granularity as the base table(s), then table partitioning can be defined on them as well, and the maintenance is simple a *detach partition* of the effected rows in the MQT too.

2.4 MULTI-DIMENSIONAL CLUSTERING

DB2 offers a third form of physical data organization called multi-dimensional clustering (MDC). In MDC organized tables, data is clustered according to values of one or more columns (*Figure 4*). The table is independently searchable on each cluster dimension. While many database systems have the capability to order data on a single dimension, which may consist of multiple columns, DB2 is the only DBMS that has the ability to cluster data independently on multiple data columns. For example, a sales table may cluster on both store and day dimensions to keep each day's sales for each store together in a small series of data blocks called a cell. Each cell occupies one or more tablespace extents. Extents are the basic unit of storage allocation containing a specified number of contiguous pages on disk established at tablespace creation time.

Figure 4: Multi-dimensional Clustering



In an MDC table, each block contains a specific value or range of values for each dimension. DB2 maintains a block index for each dimension which points to data blocks associated with each dimension value. Because block IDs rather than row IDs are being indexed, block indexes are hundreds of times smaller than standard row indexes, and therefore, are usually memory resident during query processing. In addition, MDC tables maintain a composite block index, containing entries for every value combination of all dimensions. The composite block index is used during insert and load operations to quickly find the right block for the new row and also during select statements on MDC dimensions. During select processing, block indexes can be used in conjunction with standard row based indexes for further IO reduction.

MDC organized tables afford many benefits to DB2 users. First, MDC provides automatic, low maintenance data clustering. As rows are added to an MDC table, they are placed in a block of the current extent assigned to the cell matching the values of the dimension columns in the row being added. When that extent is full, a new one is allocated to the cell. Likewise on delete, when an extent is emptied it is returned to the free pool and no longer associated with the cell. In either case, the relevant block index are updated appropriately. This design means that clustering does not decay over time as rows are inserted and dropped. DB2 guarantees that these cells stay physically clustered such that no table reorganizations are ever required for clustering.

Clustering on multiple dimensions also provides a highly granular mechanism for cell elimination during query processing. If, as in the example introduced earlier in this section, sales data is clustered by store and day, queries about daily store sales reduce to a single MDC cell. Queries concerning weekly sales only need to read the seven daily cells comprising the inquiry period. MDC block indexes point directly to the blocks satisfying the query predicates, providing quick identification of the correct MDC cells. If the query selects on multiple dimensions, the blocks IDs for each

dimension are efficiently and-ed or or-ed as appropriate. DB2 also supports combining block index search results with the results obtained from searching traditional row indexes. Finally, since the rows of interest are stored on consecutive disk blocks, the relevant blocks can be scanned using sequential I/O, which provided higher performance than random I/O. MDC provides superior performance for queries with dimension-based predicates.

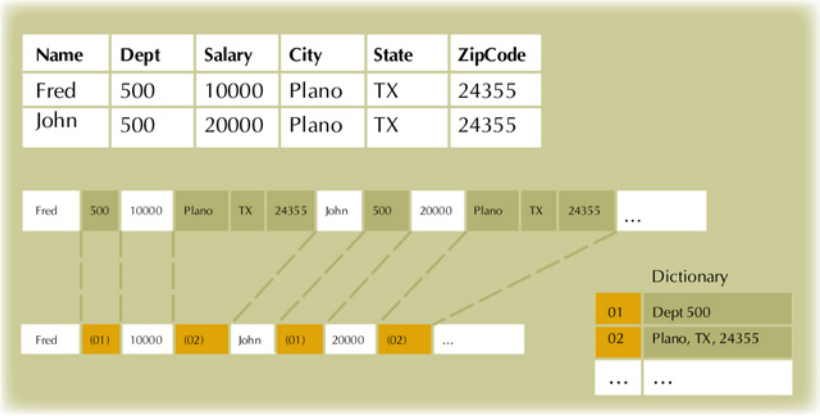
In addition, using MDC reduces the need for indexes, since MDC dimension columns provide automatic clustering and fast searching without indexes. This also yields reduced storage requirements and better performance during updates since fewer indexes need to be maintained. Thus, MDC enables high performance, low latency updates for near-real time data warehouses where data freshness is at a premium.

A potential drawback of MDC is that it requires careful implementation to assure efficient storage utilization. Poor use of storage can result for specifying the dimensions of cells at too granular a level, resulting in many partially filled extents. Poor storage utilization will offset many of the I/O and storage savings promised by MDC. Using DB2s design advisor can help to avoid this problem.

2.5 ROW COMPRESSION

DB2's row compression feature can dramatically reduce the space occupied on disk by table data and correspondingly reduce the number of I/Os necessary to retrieve large row sets or partitions. Derived from the well-known Lempel-Ziv compression algorithm, DB2's compression technique replaces common repeating data patterns within a row with a pointer to a compression dictionary entry where a single copy of the pattern is stored (*Figure 5*). DB2 supports compression of all data types except longs, BLOBs and XML data. Although all major database products now support compression of table data, DB2 is unique in supporting compression of strings that span consecutive columns in a row. The compression dictionary itself is stored in hidden rows across several pages in the table. It is loaded into memory at first compressed field access.

Figure 5: Row Compression



In DB2, row data is compressed on disk, and remains compressed when data pages are read into memory buffers. It is only expanded when column data in the row needs to be accessed to satisfy a query. Since data decompression is a simple in-memory dictionary lookup, it requires modest CPU activity to perform. On I/O-bound systems, expansion of compressed data is usually not noticeable. In fact, expansion costs are commonly offset by larger CPU savings due to reduced I/O costs.

Compression is activated via the *compress* option of the *create table* or *alter table* statements. Once compression has been turned on, a static compression dictionary must be built. For a new table, a representative sample of the data is loaded. Next an offline table *reorg* is performed to build the compression dictionary. All rows in the table at *reorg* time participate in building the compression dictionary. For pre-existing, populated tables, current table data can be used to build the dictionary. Once the dictionary exists, any data subsequently loaded will be compressed automatically. The compression dictionary is static, meaning that it does not change dynamically as data is added to the table. If the common values of data added change over time, the compression ratio will suffer, and the dictionary will need to be rebuilt via another *reorg* process.

Figure 6: Row Compression Disk Storage Savings

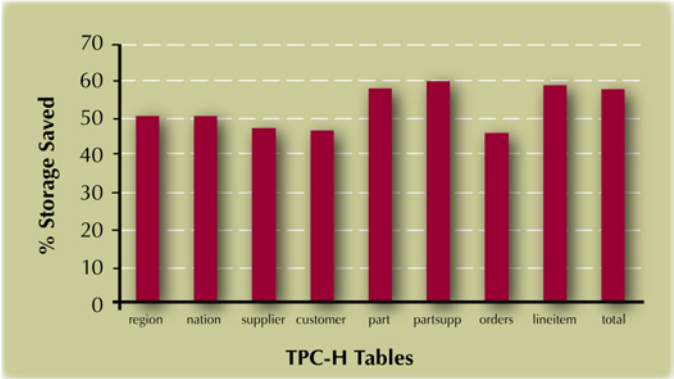


Table compression provides two principle benefits. First, compressed rows take less storage space, and because they are shorter, they waste less space at the end of disk pages. This results in significant storage savings. For example, *Figure 6* displays the storage savings from compressing the eight tables comprising the schema for the TPC-H benchmark. Since storage is commonly more than half the cost of a large data warehouse, this represents a major costs savings as well. Second, queries that need to touch a large set of data, a common property of data warehouse queries, benefit from dramatically reduced I/O, since the relevant rows occupy fewer disk pages. This translates into reduced CPU time to service I/Os, and into lower query response time.

A final note, DB2's row compression is currently incompatible with replication, a potentially significant limitation for continuously available data warehouses, since replication is often used to support them. In this case, other techniques like storage replication or dual load can be used to offset this limitation.

2.6 PUTTING IT ALL TOGETHER

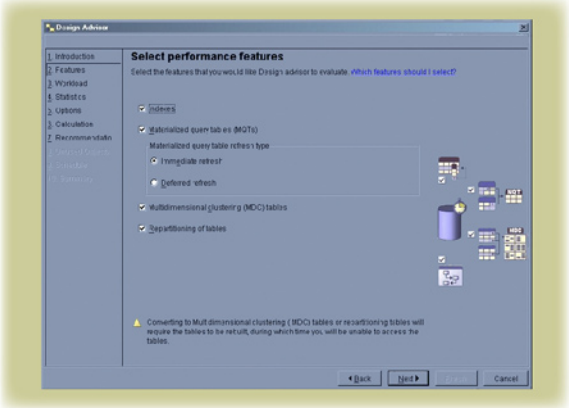
While each of these features provides significant individual benefits, when effectively combined the benefits may be substantially amplified. Database partitioning spreads data evenly to enable balanced division of labor among parallel query processes. Table partitioning clusters data by value ranges, and supports windowing applications, which is especially useful for time-based data like retail sales tables. MDC enables physical clustering of data across multiple dimensions, allowing grouping of data on disk by common search fields.

Both table partitioning and MDC can eliminate the need for indexes on columns comprising cluster or partitioning keys, allowing faster updates. Row compression reduces both I/O and storage, regardless of the data organization. In combination, these features dramatically reduce the I/O required to resolve many business intelligence queries by grouping related data together and compressing it, and spread the remaining work evenly across all DB2 partitions (BPUs). They support reduced data latency by shrinking the number of indexes that need to be maintained as data is added or updated. They also support windowing applications on time-based data without sacrificing benefits of balanced data distribution or multi-dimensional clustering.

Deciding which feature to use where, and how to combine them, can be challenging. The choice of database partitioning key determines how well data is balanced across DB2 partitions and how fast certain common joins will be performed. The best partitioning key may not optimize join performance, and vice versa. The choice of MDC attributes and the range of values for each MDC cell often require subtle tradeoffs between cell granularity and efficient storage utilization. When to index versus when to cluster is not always obvious.

DB2's Design Advisor addresses this problem. The DB2 Design Advisor helps database designers select appropriate choices for physical database design elements, like database partitioning columns, MDC attributes and value ranges, indexing columns, and materialized query tables to generate and automatically maintain (*Figure 7*). The DB2 Design Advisor is intended to be used during initial database creation time, or when the data model or application workload changes significantly.

Figure 7: DB2 Design Advisor Selection Screen



The Design Advisor can be invoked through a GUI interface, or via the command line. In either case, it takes an existing database schema and a sample workload—a captured set of queries—as input. The capture option of DB2 Query Patroller can be used to collect sample queries. The sample workload should be representative of the overall database workload and should be numbered in the 10s to 100s of queries. The sample workload can be weighted to allow individual queries to have greater or lesser influence on the recommended physical design. The design advisor produces physical design recommendations and the appropriate DDL statements to implement them. Where relevant, it also indicates database objects, like indexes, which are not used to execute the sample queries, and which may be dropped.

Although the current release of the Design Advisor does not include a table partitioning advisor, in practice, choosing whether to add table partitioning and to which columns is usually straightforward. Windowing applications benefit from choosing table partitioning on the relevant date or time column at the granularity of the windowing interval. Continuous data frequently queried by range—like zip code, for example—may also benefit from table partitioning.

Given the wealth of DB2's physical design choices, DB2's Design Advisor plays an instrumental role in helping database designers and administrators navigate the complexities of feature selection and how to employ them

3 *The IBM Balanced Warehouse*

In response to market demand, IBM has started prescribing scalable packages servers, storage and database software, dubbed the Balanced Configuration Unit (BCU). The BCU concept has offered pre-tested combinations of hardware and software that provided easier system configuration and maintenance, and predictable database performance across a broad scale of system sizes.

The IBM Balanced Warehouse advances the BCU concept by offering pre-packaged data warehousing solutions, enabling faster implementation times with lower risk. The IBM Balanced Warehouse provides pre-configured, pre-tuned combinations of software, storage and hardware, built from a cluster of BCUs, each containing a subset of the BPUs of the database (*Table 1*).

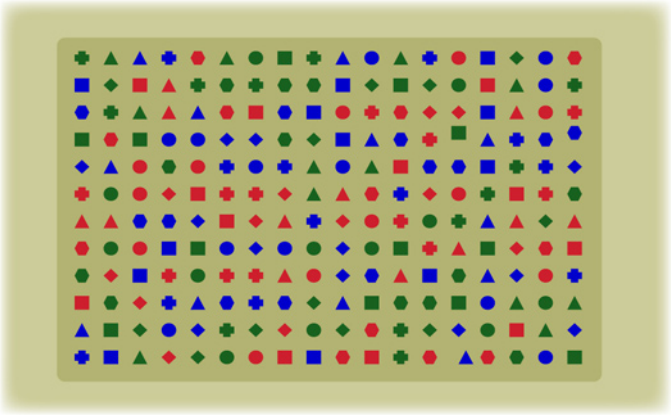
Table 1: DB2 BCU descriptions

AIX BCU	Linux BCU
— 8 processor IBM P5-575	— 2 Intel dual core processors (IBM x326)
— 1 DB2 partition / CPU	— 1 DB2 partition / core
— 4 GB RAM / partition	— 4 GB RAM / partition
— 2 Gbps Ethernet / node	— 2 Gbps Ethernet / node
— 10-14 x 73 GB 15K drives / partition	— 14 x 73 GB 15K drives / partition
— RAID 5 (4+p)	— RAID 5 or RAID 10
— 4 FC adapters / node	— .4 TB or .67 TB disk/ partition
— .5 TB disk / partition	— 150-200 GB data / partition
— 150-200 GB data / partition	

4 DB2 Architecture in Practice

To illustrate the magnitude of the benefit in I/O reduction obtained by each of these features, and how that benefit is amplified when the features are combined, we can examine a typical data warehouse scenario. Consider a national retail chain with 1000 stores, selling approximately 10,000 products to more than 10 million customers. The business achieves 100 million sales transactions per month, on average, and keeps 25 months of transaction data in its data warehouse for sales analysis and marketing promotions. This yields a sales activity table containing 2.5 billion rows. At 200 bytes per transaction, this transaction table will contain 500 GB of user data. If the DBA chooses a 32K page size for this data warehouse, the sales activity table occupies more than 500 GB and over 15 million pages on disk.

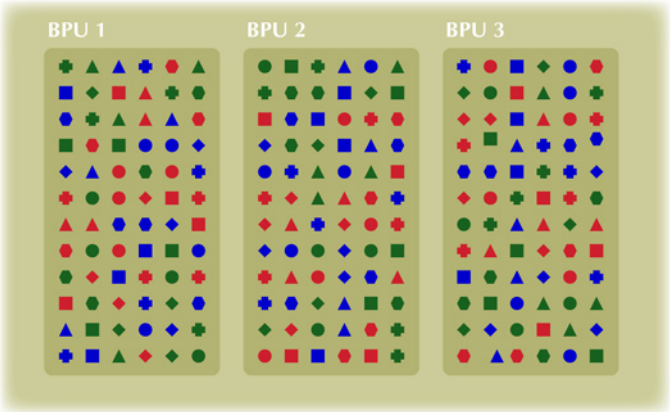
Figure 8: Data in a single partition table



To quantify the impact of various physical database design choices on query performance, let's look at the amount of I/O required to satisfy a typical sales reporting query that a marketing analyst or a store manager might ask (see *Table 2*). Consider a business query to determine total sales of a specific product group in a particular store last month. As a baseline, imagine that the sales activity data is stored all together in a single partition table with no clustering, no indexes, and no compression, as shown graphically in *Figure 8*. In this case, to satisfy the query, all 15 million pages will be sequentially scanned. At a sequential I/O rate of 50 MB per second, the 15 million pages will take 167 minutes to scan, so the query will take approximately 3 ½ hours to run.

Adding database partitioning allow us to split the data into multiple BPUs which can be scanned in parallel (*Figure 9*). With 10 BPUs, we can run the query 10-way parallel and complete it in 1/10th the time, or approximately 17 minutes, because each partition needs to scan only 1/10th of the

Figure 9: Table with hash database partitioning



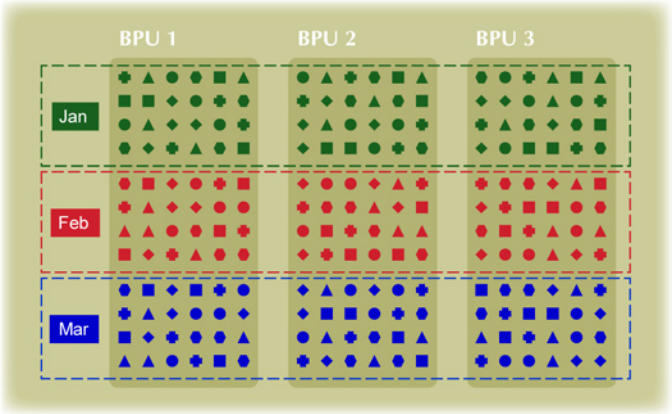
data—1.5 million pages. Note however, that while we reduced the response time by a factor of 10, running the query still required the same amount of total work by the database system as before—15 million 1/Os.

Other DB2 capabilities will help reduce the overall work required to resolve the query. Consider the effect of adding row compression to our partitioned table. Assuming a 2:1 compression ratio, which appears to be commonly achievable, based on early results reported by IBM, the size of the sales activity is cut by one half, so the table can be scanned with only ½ the I/O effort.—750 thousand I/Os per partition, which also lowers the query response time to about 8.5 minutes.

Both indexing and data clustering—table partitioning and/or MDC—will further reduce the work needed to satisfy the query and improve query response time. We can avoid the large table scans necessary to resolve the query by adding btree indexes on the query search attributes of store, date and product. Three indexes add an addition 150 GB to the storage requirements, but by reducing I/O required to 1,800 random I/Os per partition, can lower response time for the query to 36 seconds. Using compound indexes reduce this even more, to approximately 700 random I/Os, which can be performed in about 14 seconds.

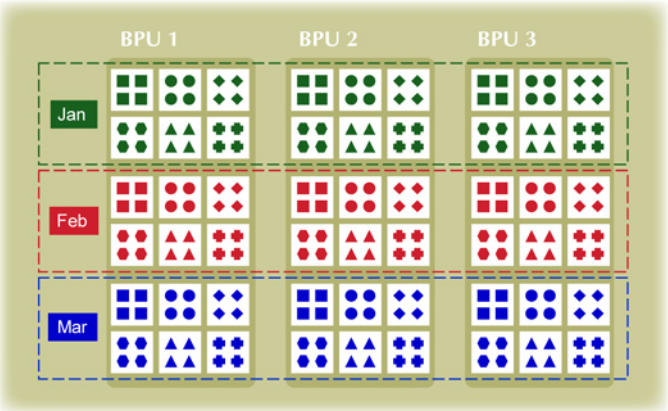
By contrast, table partitioning physically clusters data by a partition key (*Figure 10*). Replacing the date index with table partitioning by month keeps together data for each of the 25 months of data in the sales activity table. Without additional indexes, resolving the query would require scanning last month's data in each database partition, or 1 GB per partition, which yields a query response time of 20 seconds. On the other hand, adding indexes on product and store with table partitioning on month, yields about 800 random I/Os or 17 seconds response time. In either case, performance is on par with intelligent indexing, but this design has fewer indexes to maintain, yielding improved update performance. It also naturally supports the windowing design of the data warehouse.

Figure 10: Table with database and table partitioning



Suppose our performance requirement for this class of queries demanded a 10 second maximum response time. None of the design choices examined so far produces results in this range, because they do not offer the granular clustering necessary to produce the locality of reference needed to limit I/Os sufficiently. In this case, multi-dimensional clustering is a natural fit (*Figure 11*). Enabling MDC on two dimensions, product group, and groups of 10 stores, within each month's table partition,

Figure 11: Table with MDC plus database and table partitioning



provides excellent locality of reference for a broad class of sales activity and marketing analysis queries, with good storage properties. The query needs to scan only a single four page MDC cell to satisfy the query, which takes only a fraction of a second. With MDC, the sales activity table also requires less storage than the indexing options, and, by further reducing index maintenance, it offers even better update performance.

In combination, database partitioning, table partitioning, MDC, and row compression offer a compelling design choice for this type of data warehouse. Since the data warehouse characteristics—large data volume, common search patterns, and windowing requirements—that make this design an excellent solution are commonplace in today's business environment, DB2's innovative database design options offer a widely applicable solution to everyday business intelligence needs.

Table 2: Effect of Physical Database Design on Storage and I/O Performance

Case	Table Size (in GB)	Total I/Os	I/O per Partition	Min. Response Time
Single partition table	500	15,000,000	15,000,000	167 min.
Table with 10 database partitions	500	15,000,000	1,500,000	17 min.
Partitioned Table with Row Compression	250	7,500,000	750,000	8.5 min.
Add Indexing (store, date, product)	400	17,800	1,780	36 sec.
Improved Indexing (month-store, product)	350	6,900	690	14 sec.
Table partitioning (replace date index)	350	8200	820	17 sec.
MDC (replace product & store indexes)	320	40	4	< 1 sec.

By dramatically reducing the I/O, CPU and memory resources necessary to solve broad classes of analytic queries, DB2 Warehouse not only improves business intelligence performance, it also enhances the productivity of analytic workers, enabling improved analytic capabilities, and dramatically lowers the cost of delivering better business intelligence. Reducing query response time from hours to seconds enables exploratory analysis that goes far beyond standard analytic reports. It allows analysts to not only describe what happened, but also explain why it happened, and to more accurately predict what will happen. At the same time, by reducing use of system resources by several factors, it can provide these added benefits at a fraction of previous costs. In

summary, this example illustrates how IBM's DB2 Warehouse V9 can truly help companies address the challenge of delivering better business intelligence at lower cost.

5 Conclusions

To meet the growing demands for enterprise-wide business intelligence, DB2 has developed an innovative set of database design options to support high performance data access, low latency data ingest for real-time data warehousing, and storage optimization to lower operating costs that are transparent to database users.

DB2 addresses the performance of business intelligence queries from two complementary angles. DB2's balanced data distribution via hash partitioning, and support for large-scale parallelism implements an effective divide-and-conquer strategy that equalizes work across many parallel query processors to reduce overall query response time. This work-splitting strategy, called database partitioning, is coupled with DB2's unique ability, via MDC, to physically cluster data on multiple dimensions, and to order data by value range, called table partitioning, both of which limit I/O to relevant data partitions, thereby dramatically reducing the work needed to satisfy many queries. Grouping data by multiple attributes supplants the need for indexes on those attributes, both reducing storage requirements and lowering update costs. Table partitioning also supports efficient partition roll-in and roll-out for data windowing applications. DB2's row compression also reduces data volume, lowering storage costs and further reducing the I/Os needed to resolve queries. Finally, DB2's database design wizard simplifies the choice of the best physical database design for a given workload.

The innovative combination of capabilities found in IBM DB2 Warehouse work together synergistically to satisfy increasingly demanding and dynamic data warehouse requirements. These include high performance data access across many types of queries, low latency data ingest to support real-time data warehousing, and storage optimization to lower operating costs. The performance gains provided by these features translate directly into improved business value for DB2 customers. With its unique and innovative combination of database design solutions, DB2 offers compelling options to many common but complex enterprise business intelligence requirements.

In short, IBM's DB2 Warehouse provides a robust database platform that helps address the challenges facing enterprises today to, improve the business intelligence value they derive from the torrents of data they process every day, while lowering costs at the same time.

WinterCorp is an independent consulting firm that specializes in the performance and scalability of terabyte- and petabyte-scale data management systems throughout their lifecycle.

Since our inception in 1992, we have architected many of the world's largest and most challenging databases in production today. Our consulting services help organizations define business-critical database solutions, select their platforms, engineer their implementations, and manage their growth to optimize business value.

With decades of experience in large-scale database implementations and in-depth knowledge of database products, we deliver unmatched insight into the issues that impede performance and the technologies that enable success.

411 WAVERLEY OAKS ROAD, SUITE 327
WALTHAM, MA 02452
781-642-0300

visit us at www.wintercorp.com