Section II

DB2 10: A Smarter Database

In this section, we take a detailed look at the major features of DB2 10 and the ways many of IBM's most innovative enterprise customers are intending to use them to deliver an enhanced IT service to the business.

Many of these enhancements can be used "out of the box" with little or no effort required to begin exploiting them, reducing the time to value for a DB2 10 upgrade.

This section is organized around the key DB2 10 themes:

- Efficiency: Reducing cost and improving productivity
- Resilience: Improving availability and data security
- Growth: Supporting new and expanding workloads
- Business Analytics: Enhanced query and reporting

Efficiency

Even in the most favorable economic climate, businesses need to control costs and increase efficiency to improve their bottom line. In today's more challenging business environment, this has become a key factor for the survival and success of enterprises of all sizes.

This section examines the major DB2 10 enhancements aimed at improving the efficiency of the IT systems that rely on DB2: a key design objective for the new release. These features can help to reduce ongoing operational costs, improve developer and DBA productivity, and enhance the customer's experience by increasing performance and delivering a more responsive application.

CPU Reductions

Most DB2 for z/OS customers operate on a CPU usage-based charging model, so increases or decreases in the amount of CPU required to run DB2 applications can have a direct and significant impact on overall operational costs. Traditionally, IBM has tried to limit the additional CPU cost of adding new functionality into each release, keeping the net CPU impact below 5 percent.

The move to a 64-bit computing platform in DB2 V8 was an exception to this rule, and it introduced some significant processing overheads that resulted

in many customers experiencing net CPU increases of 5 to 10 percent following the upgrade. DB2 9[2] helped to redress the balance somewhat by delivering modest CPU improvements for many large customers, but IBM was determined to deliver more significant cost reductions in DB2 10.

One of the fundamental design objectives of DB2 10 was to deliver a 5 to 10 percent CPU reduction "out of the box," with little or no change being required to applications and further savings being possible with some database and/or application changes. Figure 1 shows a pictorial representation of the typical CPU decrease seen in each release since V3.

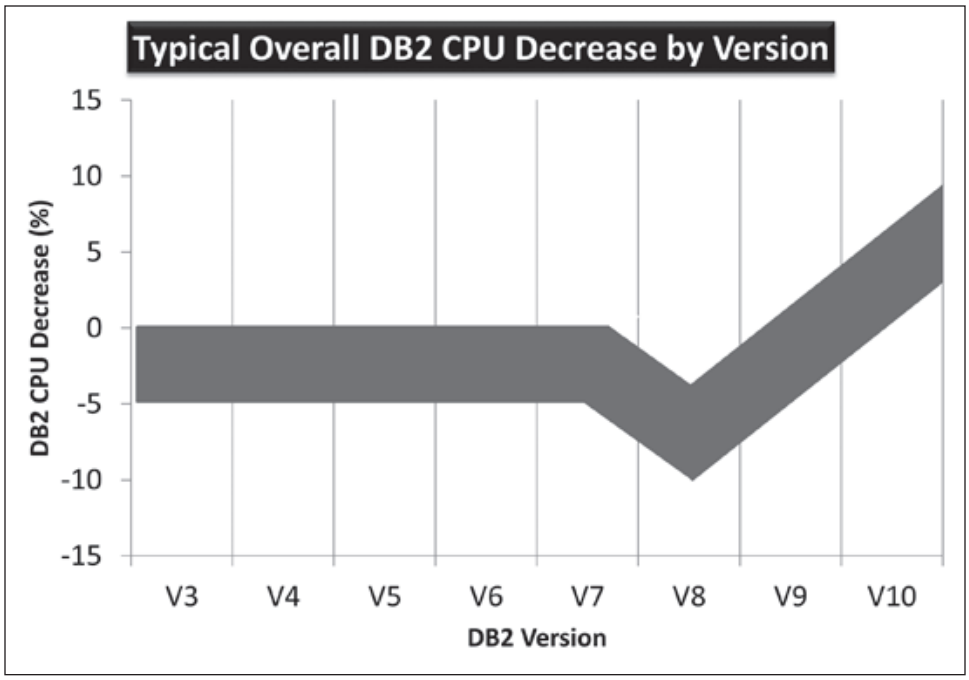

Figure 1: Typical overall CPU decrease by version

Based on IBM labs tests and some early beta customer experiences, IBM has exceeded this objective and delivered the most aggressive performance and CPU improvements of any DB2 release in the past 20 years. As many of these improvements are down to internal DB2 code optimization and exploitation of the latest System z hardware instructions, most customers can expect to see CPU savings of 5 to 10 percent in their traditional DB2 workload without any application changes[3] being required.

Figure 2 shows the savings relative to DB2 9 that were achieved in internal IBM testing using the standard IBM Relational Warehouse Workload (IRWW). The first column shows a 3.7 percent CPU saving immediately following migration in DB2 10 compatibility mode (CM). The net saving increased to 7.4 percent following a **REBIND** of the affected packages with the same access path, and this remained the same when the system was placed in new function mode (NFM). Finally, a net saving of 17.4 percent was measured once the packages had been rebound to use the new **RELEASE** protocols described elsewhere in this document.

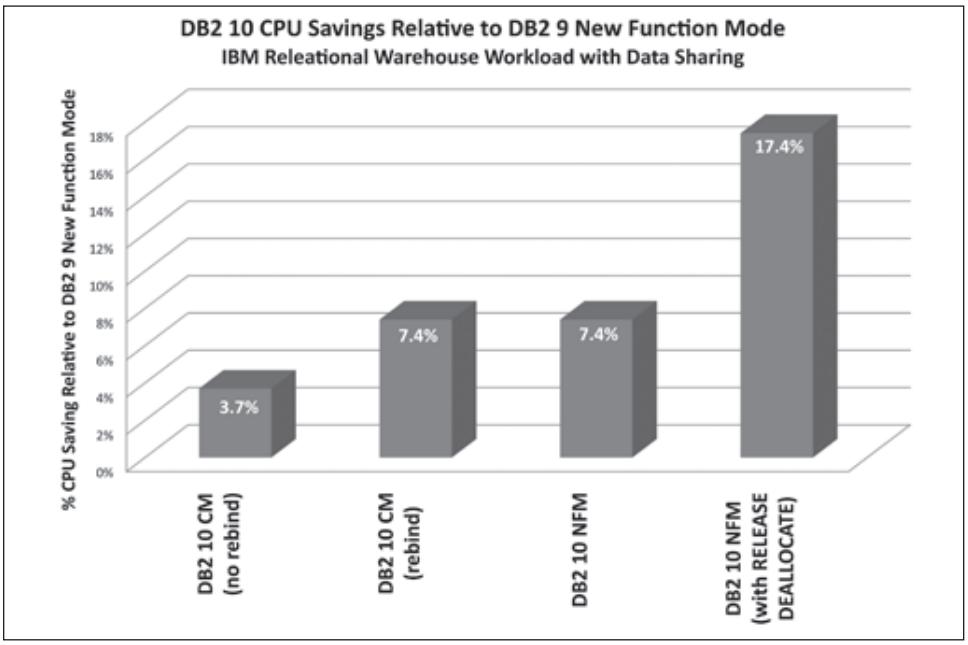

Figure 2: Sample CPU savings using IRWW

Customers running the following types of workload can expect even bigger CPU savings:

- Workloads previously constrained due to a lack of virtual storage in DB2 V8 or 9.
- Distributed applications connecting to DB2 via the DRDA protocol (e.g., SAP).

- Workloads using native SQL stored procedures. Efficiency enhancements with commonly used functions[4] have shown CPU reductions of up to 20 percent during initial IBM labs testing.
- Workloads with heavy concurrent insert activity, especially where rows are inserted sequentially, where savings of 5 to 40 percent have been observed in the labs.
- Complex query workloads, where up to 20 percent CPU reduction has been observed with no change to the access path. Greater savings are possible where a more efficient access path is selected.

All of these figures assume an upgrade from DB2 9 to DB2 10. For those customers considering a move directly from DB2 V8 to DB2 10,[5] the net impact could be even bigger.

Most of the performance measurements available at the time of writing are based on internal IBM lab workloads, but early indications from beta customers show CPU savings in line with the lab tests. The potential CPU savings made possible by DB2 10 are likely to be the single biggest factor in driving customers to upgrade to the new release—especially as many of the savings can be realized very quickly after the upgrade, and with few or no application changes.

Temporal Tables

Many IT systems need to keep some form of historical information in addition to the current status for a given business object. For example, a financial institution may need to retain the previous addresses of a customer, as well as the current one, and may need to know which address applied at any given time. Equally, an insurance company may need to know what level of coverage was in place two months ago when a claim was made. Previously, these kinds of requirements would have required the DBA and application developers to spend valuable time creating and testing the code, as well as associated database design to support the historical perspective while minimizing any performance impact.

The new temporal data support in DB2 10 provides this functionality as part of the core database engine. The DBA indicates which tables/columns require temporal support when they are created, and DB2 maintains the history automatically whenever an update is made to the data. Elegant SQL support lets the developer query the database with an “as of” date, which will return the information that was current at the specified time.

As shown in Figure 3, DB2 maintains a separate "history table" for updated rows in a temporal table. This is completely transparent to the developer, who codes SQL against the main table as usual. When a row is updated (as shown at time T3 in the diagram), DB2 will store a version of the old row in the history table before updating the current row in the main table. Similarly, when a row is deleted, it is first copied to the history table before being removed from the main table. DB2 maintains system timestamps (the **SYS_START** and **SYS_END** columns shown) to record the period during which a given version of the row was current.

Finally, the new **AS OF** clause in SQL **SELECT** statements lets the developer see the data as it was at a given point of time. In the example, the policy information at time T2 is required, which will return the original address (**A3**) instead of the current address (**A4**).

With so many IT systems needing to accommodate a historical perspective and maintain audit logs of changes made to sensitive data, DB2's new temporal

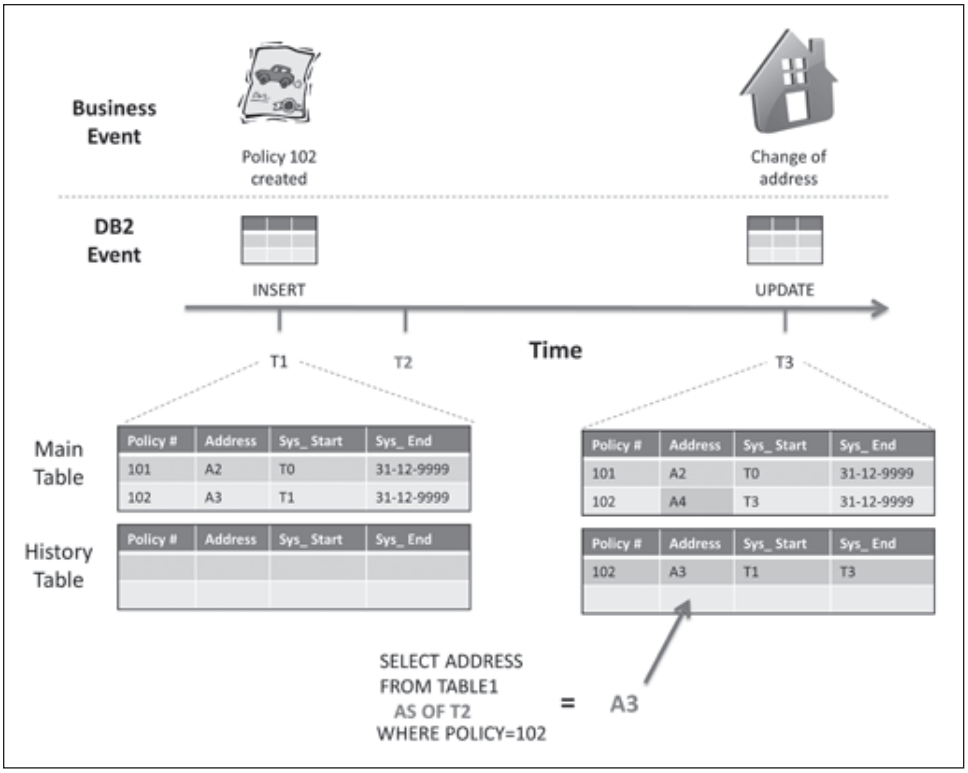

Figure 3: DB2 temporal data concepts

support promises to save many hundreds of hours of design, coding, and testing that would otherwise be required to build this function manually for each application. While the benefit for existing applications is limited, this feature promises to deliver major productivity savings for new developments.

Improved Scalability

The valuable scalability enhancements within DB2 10 are described elsewhere in this document. In addition to supporting workload growth and providing more flexibility, these enhancements can deliver some significant performance benefits, as follows:

Reduction in data sharing overhead. The virtual storage constraints within previous releases of DB2 imposed a practical limit of 400 to 500 concurrent active threads[6] within a single DB2 subsystem. As a result, many DB2 data sharing[7] customers were forced to use more DB2 members than otherwise necessary to support their workloads. Although DB2's industry-leading data sharing architecture minimizes the processing overheads, each additional member will impact overall performance and resource usage.

Figure 4 shows a typical scenario for an SAP environment. In this example, a data sharing group consisting of four DB2 members is used to support 1,600 concurrent threads from four SAP application servers.

DB2 10 introduces some dramatic scalability improvements that allow each system to handle five to 10 times the current number threads. This will allow

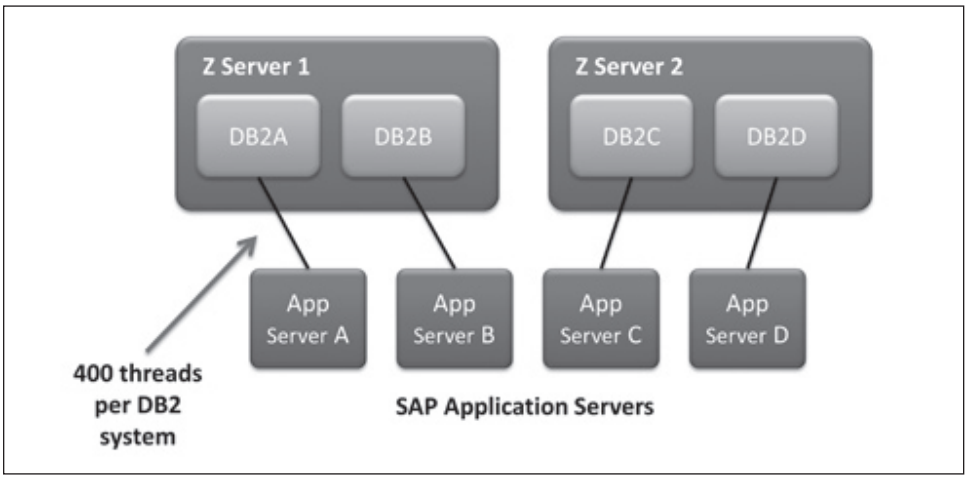

Figure 4: Typical SAP data sharing configuration

many customers to reduce the number of DB2 members needed to support their workloads, resulting in net CPU and memory savings and improving application performance.

This benefit is illustrated by Figure 5, which shows that the same 1,600-thread workload can be handled by just two DB2 subsystems, with significant scope for additional workload growth. (Initial SAP benchmarks show 2,500 threads per DB2 system is sustainable.) Productivity savings are also possible due to the reduced requirement to closely monitor available storage.

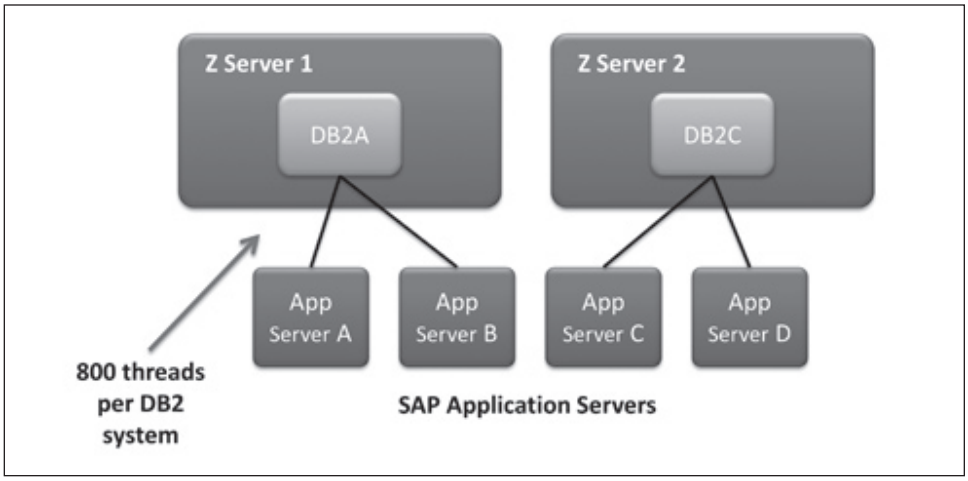

Figure 5: Potential DB2 10 SAP data sharing configuration

Improved dynamic statement caching. With the growing popularity of running Java™ and ERP workloads such as SAP on DB2 for z/OS, dynamic SQL[8] is becoming more and more prevalent. DB2 allows dynamic SQL statements to be cached in order to avoid most of the overheads usually associated with executing dynamic SQL, but the size of this cache was limited in previous releases due to the same virtual storage constraints described above. This in turn limited the effectiveness of the cache.

The virtual storage constraint relief delivered in DB2 10 will enable most customers to dramatically increase the size of the dynamic statement cache, thereby allowing a greater proportion of their dynamic SQL to be cached and reducing CPU and elapsed times for these queries. Other important enhancements that will improve dynamic statement caching in DB2 10 are described elsewhere in this document.

Together, these scalability enhancements give DB2 customers more flexibility in the way they distribute their workload across the available System z servers, while reducing DB2 CPU usage and improving the performance of key application processes.

New Hash Access Method

Many high-volume OLTP applications need to efficiently access a single row via a fully qualified primary key, but most of the access paths available to DB2 today are optimized for accessing sets of rows. Previously, the most efficient access path for a single-row fetch would have been via a unique index on the table, as shown in Figure 6.

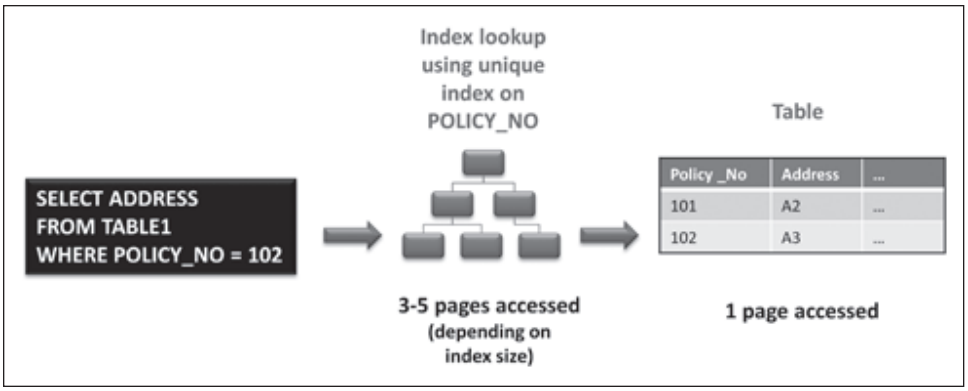

Figure 6: Single-row access via unique index

While this access path can be highly efficient if multiple rows need to be accessed sequentially, the overhead of navigating the index structure can be expensive for single-row access. Depending on the size of the data, the preceding example would typically require DB2 to access a total of 4 to 6 pages in the index and table, some of which might also require a physical I/O operation to pull the page into the buffer pool if it isn't already resident.

DB2 10 introduces a completely new access method, known as Hash Access. Where a table has been enabled for Hash Access, the vast majority of requests for a single row using the unique key will be satisfied with a single page access[9] because DB2 will use the key as input to a hashing algorithm that will produce the page number and row offset needed to directly access the given row (Figure 7).

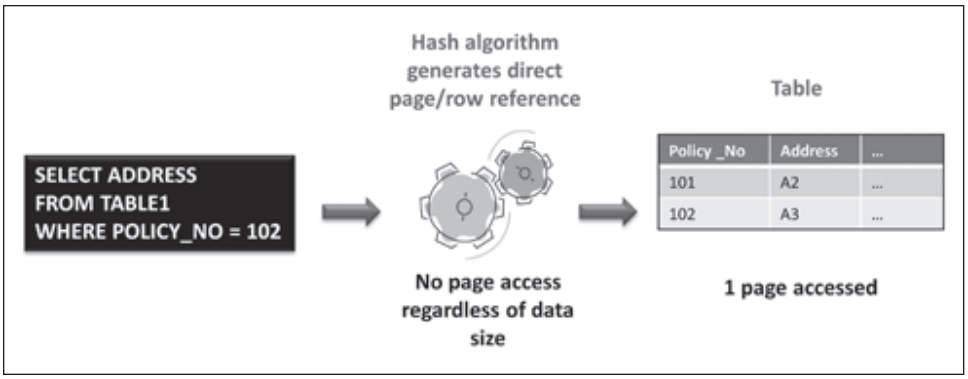

Figure 7: Single-row access via a hash

Hash access tables are not without their disadvantages: they will require 20 to 100 percent more disk space than traditional types and could be more expensive to access for multiple-row access. However, for many high-performance applications that predominantly use single-row access, these limitations could be an acceptable tradeoff for significantly reduced CPU (due to fewer pages accessed) and potentially lower I/O and elapsed times (if physical I/O operations are avoided for index page access).

Automated Statistics

One of the most important factors in DB2 query performance is the access path chosen by DB2, and that is heavily influenced by the table and index statistics gathered by the **RUNSTATS** utility. The old adage of "garbage in, garbage out" is very relevant to access path selection, so an important part of any DBA's job is to ensure that accurate, up-to-date statistics are available for critical tables.

RUNSTATS can be scheduled to run at fixed times, but this doesn't allow for ad hoc processes that can significantly change the table characteristics. A simple scheduled approach can result in statistics not being gathered often enough (leading to poor access paths and increased CPU/elapsed time) or too often (wasting the CPU used by the **RUNSTATS** utility).

A new automated statistics feature enables DB2 10 to dynamically monitor the currency of table and index statistics and automatically schedule the necessary **RUNSTATS** job when required. This frees the DBA to focus on more demanding activities, improving productivity and potentially reducing CPU requirements due to improved access paths and/or elimination of unnecessary **RUNSTATS** jobs.

Include Additional Columns in Unique Index

When a primary key is formally defined on a table, DB2 requires a unique index to be defined. In previous versions of DB2, that index could contain only the primary key columns. If additional columns were required to support specific SQL statements (such as the **SELECT** statement shown in Figure 8), it was necessary to create a separate additional index.

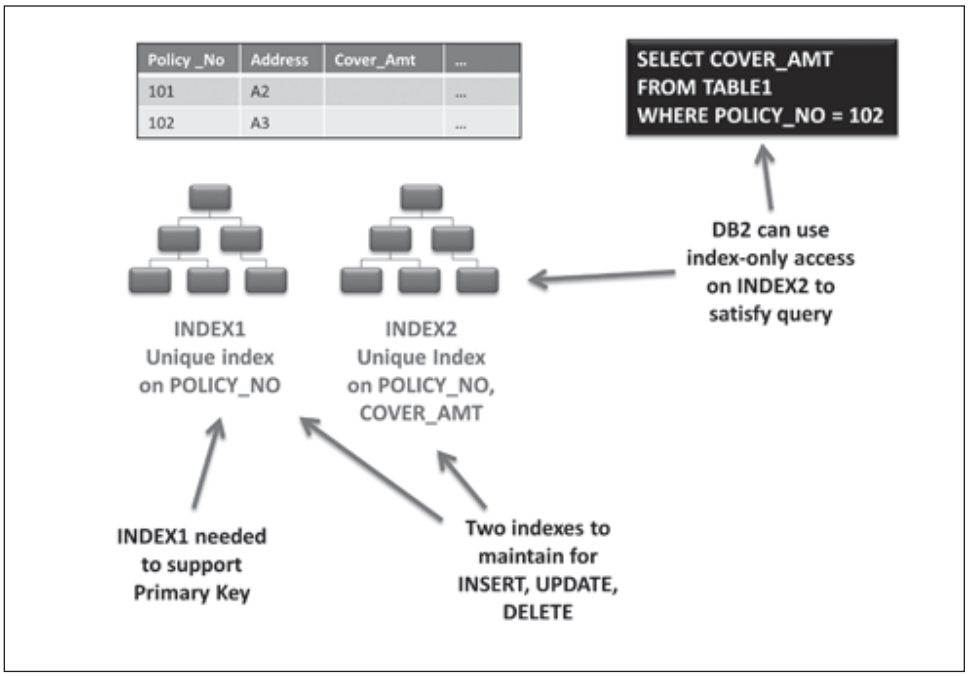

Figure 8: Multiple indexes to support index-only access

This approach allowed the SQL statement to be executed efficiently but added unnecessary overheads to **INSERT**, **UPDATE**, and **DELETE** operations as two indexes needed to be maintained rather than one.

DB2 10 allows columns other than the unique key to be specified in the index definition. As shown in Figure 9, this allows the second index to be dropped, removing the DASD, CPU, and I/O overheads associated with that index while continuing to support the efficient access path needed by the **SELECT** statement.

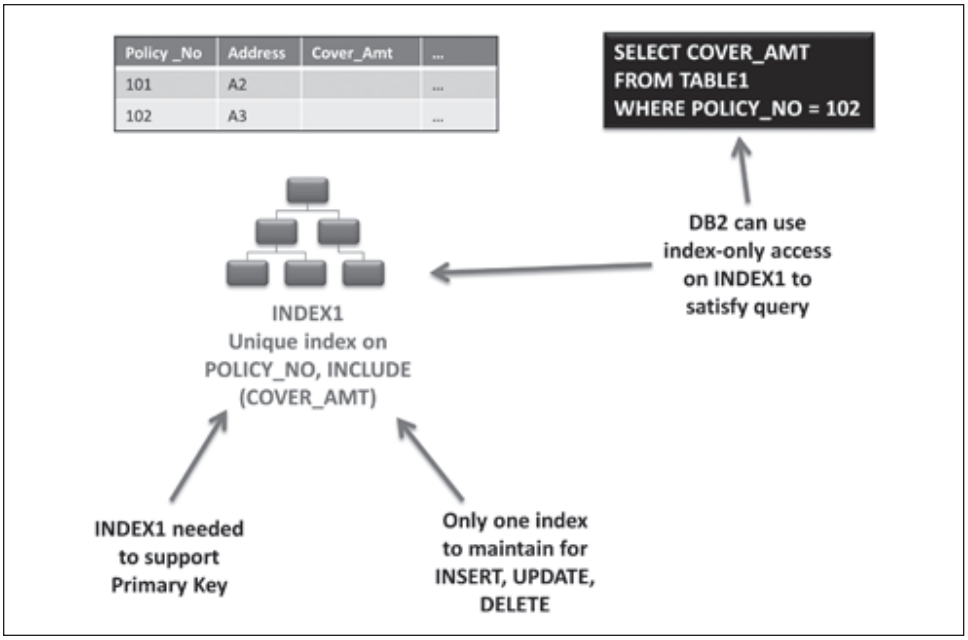

Figure 9: Single index with non-unique columns included

Where additional indexes have had to be created specifically to support this kind of access, removal of the redundant index will significantly reduce the cost of any update operations against the underlying table. Initial lab tests have shown up to 30 percent CPU reduction in **INSERT**, with identical query performance in one example where two indexes were replaced with a single one using **INCLUDE** columns.

Buffer Pool Enhancements

As processor speeds continue to increase at a faster rate than disk subsystems, the relative cost of performing random I/O operations is increasing, and minimizing I/O has become a major objective in improving performance. DB2's buffer pools cache frequently accessed data in memory, avoiding physical I/O activity and significantly improving performance.

DB2 10 introduces a number of new enhancements for buffer pools that should yield significant performance benefits.

Large page support. With the move to a 64-bit computing platform in DB2 V8, it became possible to define dramatically bigger buffer pools (up to 1TB). However, the size of each hardware "page" within the pool remained at 4KB (Figure 10), so large pools can have many millions of pages, leading to increased z/OS overheads.

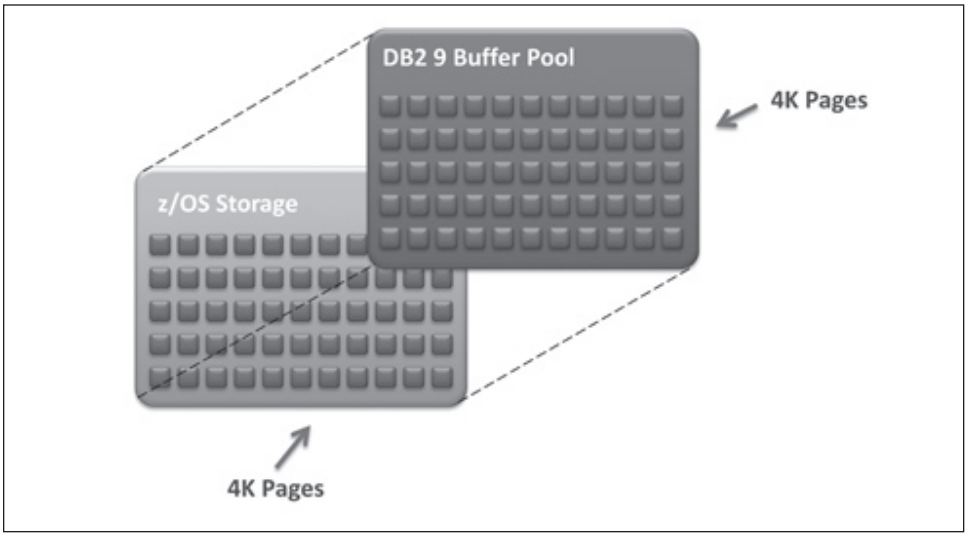

Figure 10: Buffer pool using 4KB z/OS page size

IBM's z10 and newer z196 servers are able to support 1MB page sizes within the hardware (Figure 11), which will result in fewer pages and more efficient access to data within the DB2 buffer pools. Internal IBM testing has shown CPU reductions of 1 to 4 percent with this feature enabled.

In-memory pagesets. Many DB2 applications make extensive use of "code tables": small, frequently referenced lookup tables. Such tables are often performance-critical and are placed in separate buffer pools that have been sized to ensure that all data remains in storage to avoid any I/O delays. A new attribute introduced in DB2 10 allows a given buffer pool to be marked as "in memory." DB2 will automatically read all data into this buffer pool at startup (avoiding I/O delays the first time a given data page is accessed), and the optimizer will assume a zero I/O cost when assessing the cost of accessing the table. This enhancement could further improve access times to these performance-critical tables.

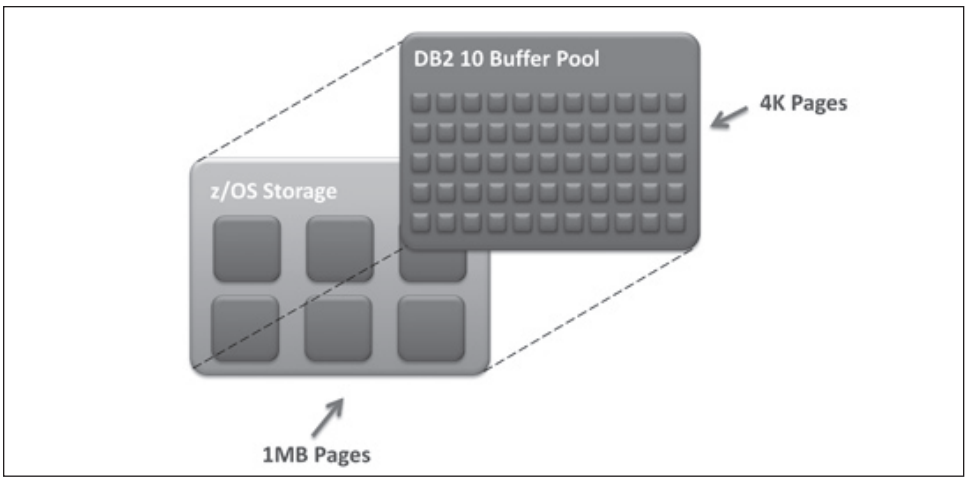

Figure 11: Buffer pool using 1MB z/OS page size

Memory allocation on demand. In previous versions of DB2, the full amount of storage was allocated as soon as the first table or index belonging to a given buffer pool was allocated. Oversized pools could therefore reserve storage that was never used. DB2 10 allocates storage as it is required, allowing more efficient use of available storage within the System z server.

Dynamic Statement Cache Enhancements

As mentioned elsewhere in this document, dynamic SQL is becoming more and more prevalent, and DB2 allows dynamic SQL statements to be cached to avoid most of the overheads usually associated with executing SQL in this way. However, the dynamic statement cache previously relied on SQL statements being identical to be able to re-use the cached statement.

In the example shown in Figure 12, the two SQL statements are different (they are selecting a different policy number), and therefore they will be separately cached even though the access path taken is likely to be identical for each one. This uses up valuable space in the dynamic statement cache and forces DB2 to fully re-prepare each statement, causing significant CPU and performance overheads for high-volume transactions.