The business benefits of DB2 9

(formerly codenamed "Viper")

Author: Philip Howard
Date Published: July 2006

a white paper by

Bloor Research

...optimise your IT investments

Executive summary

The "Viper" release of IBM DB2, which is officially version 9, is the most important release of this database for many years. Indeed, IBM regards Viper as so significant that, at one time, it considered calling it DB3, on the basis that this represents the third generation of databases from IBM, following IMS and DB2.

As may be imagined, there are a large number of new features and capabilities, as well as incremental enhancements, in this release. This paper does not attempt to discuss all of these and concentrates, in particular, on those that have significant business benefit.

That begs the question of what we mean by business benefit. Since a database is inherently a part of a company's IT infrastructure, identifying its business benefits can be complex, because many of the advantages that a particular feature may bring are indirect. If new features of a database make it easier to manage, say, then the people responsible for administration need to spend less time doing that, which means that they can spend more time fulfilling other duties. In other words, they become more productive. So, for example, some other project may get completed more quickly than would otherwise be the case. However, this fact in itself may not be visible outside the IT department, let alone the fact that it derived from an easier to manage database.

Other benefits are, of course, more direct. Performance enhancements, for example, may delay a requirement to upgrade hardware, which has a direct impact on the bottom line. On the other hand, some features may open up new possibilities for the business that were not previously feasible: you may, for instance, now be able to develop applications that would not have been viable before, thereby opening up new business opportunities.

Another point to make is that business benefits may be competitive, in the sense that they enable capabilities that are not available from other vendors; or they may be comparative, where IBM is introducing enhancements that offer benefits when compared to previous versions of DB2 but do not necessarily provide competitive advantage relative to other database products.

In the discussions that follow we will identify the types of benefit that individual features of DB2 9 provide and whether or not they are competitive or merely comparative. In order to do this, and bearing in mind that databases are intrinsically technical products, in the sections that follow we will not merely be discussing DB2 9 in business terms but we will also explain how the various technologies work, and where they differ from that of IBM's competitors, in order to understand how these benefits are derived.

The format of this paper is that each of the major new features of DB2 9 has a section to itself, followed by a composite section that briefly discusses the remaining enhancements that have been introduced with this release. Finally, we summarise our conclusions.

XML

It is because of its XML support that IBM was considering calling this release of its database DB3. This is because DB2 9 is no longer just relational. Instead, it offers a hybrid relational/XML environment. Before we consider the implications of this we need to explain exactly what IBM has done and how it differs from the XML implementations of other database vendors.

There are various ways in which XML documents can be stored. The simplest method is to store a document as a single entity. This is done by treating it as a large object (LOB). However, the problem with this approach is that you can only access the XML in its entirety. You cannot, for example, access a particular detail within the document.

In order to enable the ability to access detail within an XML document an alternative approach is to parse the XML. That is, to split the document up into its constituent parts and then store those parts as relational data within tables in a conventional manner. However, this method has the drawback that this 'shredding' approach is slow. Moreover, if you want to inspect the whole document then it has to be re-constituted which, again, is a slow process. It is thus commonplace for vendors to offer this method in conjunction with LOB support, so that you do not have to re-combine tabular XML data if you want to view the original document in its entirety.

Performance is not the only problem with shredding: it is not generally a compliant approach. For example, it is sometimes possible that after de-composing an XML document, storing it and then putting it back together, you do not end up with what you started with. In particular, it does not support compliance in so far as digital signatures are concerned.

Another issue for relational environments when storing XML is that fields storing XML data are not natively understood by the database. This means, amongst other things, that columns with XML in them are not recognised by the database optimiser, which means further problems with performance. For this reason, a number of database vendors have introduced an XML datatype. This has the additional advantage that once you have an appropriate datatype you can define indexes against that column.

However, some suppliers, having introduced an XML datatype, have gone on to suggest that this therefore means that they have native support for XML storage. This is not the case—XML data is still either shredded or stored as a LOB—support for a datatype means that the database can natively understand that type of data but it does not mean that it is stored natively, so performance issues do not go away.

For all of these reasons some companies have introduced specialised XML databases. Nevertheless, while this overcomes performance issues in storing and retrieving XML data it creates new problems when you want to combine relational and XML data. In a query-only environment these are not so much of a burden because

federated query products such as IBM's WebSphere Information Integrator can access separate XML and SQL databases within a single query without too much degradation in performance (though there will always be some slow down). However, in a transactional environment these problems are much more acute and are not limited to performance: for example, you may need to implement two-phase commit across these disparate databases in order to retain synchronicity, development will be much more complex, and management and administration will be more time consuming. For these reasons, pure XML databases are only really suitable for applications which are purely XML-based and do not require relational data as well.

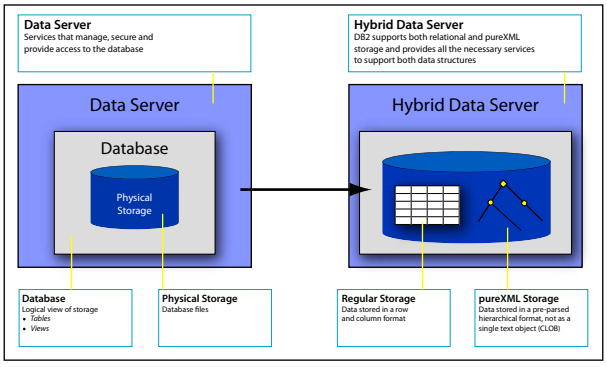

Figure 1: *Hybrid storage in DB2 9*

This brief outline describes the position prior to the introduction of DB2 9. What IBM has done is to appreciate that the only recourse is to have a database that can store both relational and XML data natively, which IBM is calling pureXML™, each with its own storage mechanisms (so that there is no performance hit when accessing XML data), as illustrated in Figure 1.

Note that IBM provides an environment that has a single management and administrative framework, as well as access capabilities that allow the co-mingling of relational and XML data. That is, you can use SQL to access relational data and XQuery to access XML data but you can also mix these (and IBM's version of SQL has long since been "XMLised" anyway) so that you can access both SQL and XML within a single query.

Development and app. performance re: XML data	with relational data server	with DB2 "Viper" hybrid data server
Development of search & retrieval business processes	LOB: 8 hrs Shred: 2 hrs	30 min.
Add field to schema	1 week	5 min.
Relative lines of I/O code (65% reduction)	100	35
Queries	24–36 hrs	20 sec–10 min
Query non-shredded XML element	1 week	½ day

Table 1: *Relative performance of different approaches to XML*

In terms of direct benefits there are two main advantages of this approach. The first of these is that, as discussed, it eliminates the performance hit that you get if XML is not stored natively. For example, Table 1 shows the relative performance of different approaches to XML, based on comparisons done at a beta customer of DB2 Viper. Note that performance benefits are not limited to the comparison between hybrid storage on the one hand and the use of LOBs or shredded approaches on the other, but also extends to database administration (adding a field to a schema, which can be done on the fly, thereby making the environment much more flexible) and to development. This leads on to the second major benefit offered thanks

to the hybrid storage method adopted by IBM, which is that it enables the development of applications that combine XML and relational data in ways that were not previously realistic.

Of course a lot of XML, especially in an SOA (service oriented architecture) environment, is transient. However, much of it is not. One good example is anything that involves contracts, ranging from consumer contracts in the financial services sector to service level agreements within the IT sector. The problem is that such documents typically include both structured (relational) and unstructured (XML) data and you not only want to be able to manipulate these separately (and extract the structured data from the contract in the first place) but also together. Moreover, in some instances, details change on a regular basis: for example, when it comes to service-level agreements these will be reviewed and amended on a periodic basis. Similarly, you may have milestones built into a contract that you need to monitor on an on-going basis. In such circumstances you only want to have to change one part of the data (relational or XML) and you want those changes to be reflected across the contract: thus you might change the price of a contract in your relational data but you want that to be reflected automatically in the relevant XML document.

To be more specific consider the number of standardised XML variants:

- Financial—ACORD (insurance), FIXML (financial information), FPML (financial products), FUNDSML (Funds), XBRL (business reporting);
- Life Sciences—AGAVE (genomics), BSML (bioinformatics), CML (chemicals);
- Publishing—SportML, NewsML, XBITS (book industry), XPRL (public relations);
- Others—LandML (land development), MODA-ML (textile/clothing supply chain), MatML (materials property), JXDM (global justice), ebXML (electronic business).

And so on and so on. Huge amounts of information are captured using XML but it is difficult and unwieldy to extract structured information from those documents and to use that as, or combine it with, relational data, unless you use a solution such as DB2 9, which, at present, is one of a kind.

Another example is the use of XML for business intelligence purposes. In the past, you could not really use XML documents for conventional BI because it was impractical in terms of the time taken to shred relevant documents, unless the dataset under consideration was very small. Without that need, it makes it practical to query large XML document stores in a relatively short period of time, whereas previously this was only feasible for processes such as text mining.

Further, and in particular, IBM sees its hybrid support for XML as a cornerstone in the move towards an SOA environment. As an example, see Figures 2 and 3,

Figure 2: *Original order fulfilment system*

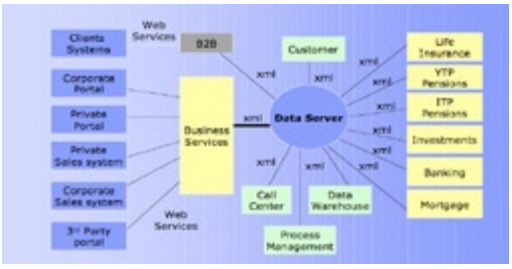

Figure 3: *DB2 9 powering the new SOA based solution via XML*

which show pre- and post-versions of the same application environment. The potential for using an XML-based approach should be obvious.

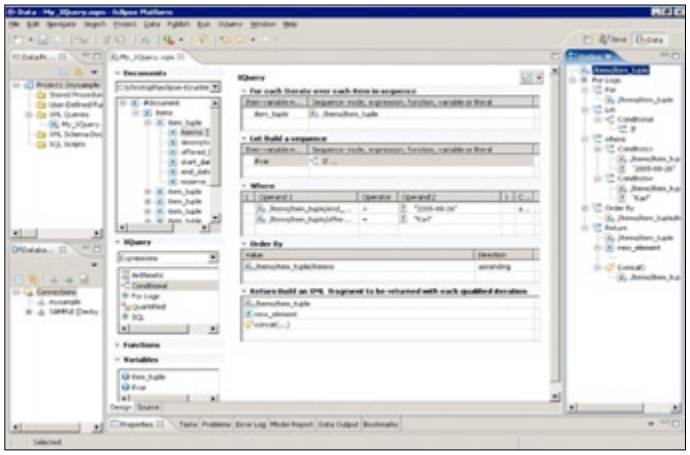

Figure 4: *Visual XQuery Builder*

Finally, it is worth remarking on IBM's support for XQuery, which is the standard method for addressing XML data and, in particular, its introduction of Visual XQuery Builder in this release, as illustrated in Figure 4. While the former is not out of the ordinary, the Visual XQuery Builder should certainly be a boon for developers. It is also worth noting that IBM has announced partnerships with a number of vendors, including Zend Technologies, Exegenix, Justsystem and ActiveGrid that have extended toolsets to help in the development of XML-based applications.

Compression

Compression is the science of fitting a quart into a pint pot. Or, better yet, a half pint pot. In principle, compression is very valuable: if you have a 10 terabyte database and you can compress it by 50% then you can store it in 5 terabytes instead, thereby reducing both your storage costs and associated running costs. However, compression is difficult for conventional relational databases. This is because they are row-based and, because you store data by row, you have to compress it by row. (Note: this is not the case for column-based relational databases but as there are none of these that support transactional processing and only a few in the data warehousing arena, we will assume a row-based approach for the purposes of this discussion).

In a typical database row you might have one column with alphabetic data in it, another with alphanumeric information, several with numeric data, some with floating point decimals, one or two date fields and a variety of other exotica. The problem is, as far as compression is concerned, that the best way to compress an alphanumeric field is different from the best method of compressing floating point decimals. What this means in practice is that if you simply compress rows of data you have to select an algorithm that is based on the least common denominator: it compresses across the row, bearing in mind that there are multiple datatypes within a row, so no one column is compressed optimally.

Prior to this release IBM had already introduced null and default value compression, multi-dimensional clustering (which provides index compression through the use of block indexes) and database back-up compression, though only the first and third of these could be described as generic, with the multi-dimensional clustering being specific to data warehousing. It has also had value compression, which is effectively the lowest common denominator approach described above, for some time. However, in this release the company has added what it refers to as row compression but which is actually tokenisation which is, to our minds, not the same thing as compression, though it amounts to the same thing in the end: in that it means you need less storage space.

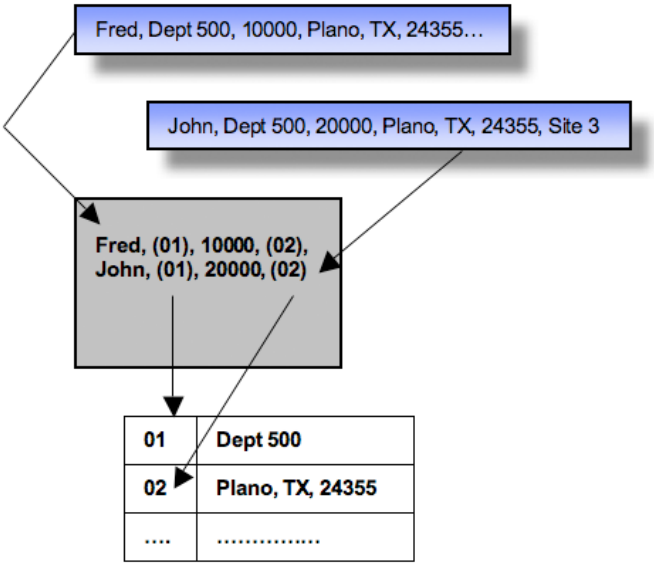

Figure 5: *Elements of tokenisation*

Put briefly, the aim of tokenisation is to separate data values from data use. This has the effect of reducing data requirements and improving performance. As a practical example, in a customer table you might have many customers in Michigan (or in the case of Figure 5 employees in Plano, Texas) and each of them would have "Michigan" stored as a part of their address. To store this several hundred, or even thousands of times is wasteful. Tokenisation aims to minimise this redundancy.

There is more than one way of implementing tokenisation but the method chosen by IBM is that the database software will look for repeated patterns within the data and, when these are found, it will create a relevant token

(usually a numerical value) that represents Michigan (or Plano, Texas, as illustrated) each time that it appears within a table and then have what is effectively a look-up table (held in the data dictionary) so that you can convert from the token to the data value. Note that for numeric values tokens are not required.

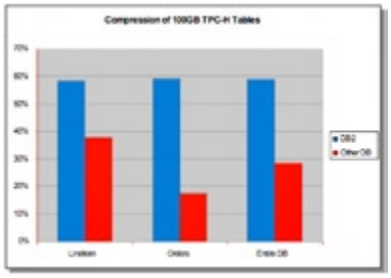

Figure 6: *Compression rates of DB2 9 compared to other approaches*

In terms of storage savings IBM estimates that use of this technique will produce savings of anywhere between 35% and 80% depending on how much repeated data is stored. Certainly, in customer and HR applications, for example, there are likely to be considerable savings. Further, we know of no other relational database vendor (excepting the aforementioned column-based products) that can offer this level of compression. While we tend to treat benchmarks with caution, Figure 6 illustrates the different compression rates achieved by DB2 9 when compared to more conventional approaches (which would also typify previous versions of DB2).

There is, however, a potential downside: when you access tokenised data you have to read from the dictionary as well as the relational table so there is extra I/O involved in the process. However, this is offset by the fact that data is still in tokenised form when held in buffer pools in memory. This means that more data can be held in these buffers, thus reducing the amount of I/O required. In practice, IBM estimates that the advantage gained from holding more data in memory will usually outweigh the fact that you have to de-tokenise the data. For example, take this customer quote from the senior database administrator at AutoZone: "*with the new compression technology in DB2 Viper, we realized an 80 Percent improvement in space savings for our most critical tables in our Data Warehouse. We were even more pleased with this technology when we found that Viper's compression capability helped us process queries to the database an average of 40 Percent faster than before. We're looking forward to seeing the same results with our Operational Data Store and OLTP systems*." In other words, more often than not you should get a performance benefit as well as a reduced storage requirement.

To conclude this section, even if we assume the lowest possible factor for reducing storage requirements, database size should be reduced by a third while, at worst, performance should stay approximately the same. Thus there should be significant direct benefits in terms of cost of ownership. One point to note, however, is that compression is not available for XML (it is planned), LOBs or indexes (except when used with multi-dimensional clustering, as discussed), so the benefits outlined here apply specifically to relational data.

SAP optimisation

Given Oracle's status (subsequent to its acquisition of PeopleSoft, Siebel et al) as SAP's main competitor, it is perhaps not surprising that SAP and IBM have strengthened their partnership with respect to SAP and DB2. This started with the release, in 2005, of DB2 version 8.2.2, which was specifically optimised for SAP environments. With the Viper release of DB2 this integration has gone further and the two companies (all development is performed jointly) have an ongoing roadmap for continuing this partnership into the future.

The key to SAP optimisation is not so much that there are additional features for SAP environments but that DB2 understands the SAP environment within which it is working. For example, one of the new features of DB2 9 is for the software to auto-discover its environment upon installation and to automatically set default values based on that configuration. Normally speaking, this would include the hardware platform, operating system and other infrastructure details but, in the case of SAP, DB2 can now also recognise relevant details of the SAP configuration in use and set these defaults accordingly as well.

Of course, you could argue that you will install DB2 before you install SAP so that the Configuration Advisor (which provides the capabilities just discussed) won't know about the SAP deployment until later. However, there is also a "silent install" capability which means that you can install DB2 as a part of the SAP installation process—in effect, implementing both together, in which case the Configuration Advisor will indeed know about the SAP environment. Note that IBM is the only database vendor to have this silent install capability.

The other features that DB2 offers in conjunction with SAP are also based around the fact that the former knows about the latter. For example, DB2 is aware of SAP workloads and the database's built-in tuning capabilities can use this when it makes recommendations about building new indexes, materialised query tables and so on; and the same applies to troubleshooting, whereby diagnostics also understand the SAP environment. Features like multi-dimensional clustering can also be used specifically in conjunction with SAP.

The benefits deriving from the partnership between SAP and IBM are essentially about easier management and administration, and therefore reduced overhead in these areas. However, this is one of those cases when there are indirect advantages that accrue from these benefits. Both SAP and DB2 (and relational databases in general) are complex environments. Getting the best performance out of them on a consistent basis is not a trivial task: the integration of the two products means that the task of tuning, for example, is very much easier and, as a result, improved performance can also be expected. It is also worth noting that, although this is not technology specific, IBM and SAP have aligned their maintenance and product delivery plans. In the case of the former this means a single port of call in the event of a problem arising.

Finally, it is worth bearing in mind that IBM has never been the leading database vendor underpinning SAP solutions. However, there are signs that that could

change. Since DB2 version 8.2.2 came out with SAP integration, IBM has won a number of competitive SAP deals, some of which have migrated away from other database products.

Range partitioning

Unlike any of the other features we have so far discussed, range partitioning in itself (as we shall see, things are different when combined with multi-dimensional clustering) does not offer any competitive advantage, though it does offer comparative advantage with respect to earlier versions of DB2. We should, however, point out that while Viper applies to both the distributed and mainframe versions of DB2, in fact range partitioning has been available in DB2 for z/OS for some time.

Nevertheless, the introduction of range partitioning is late. Some other vendors have been able to offer it for a decade. What it does is allow you to partition data by range, such as a date range or geography or store. Thus you could have separate data partitions for January, February, March and so on; or for France, Germany, Brazil and so forth. Some vendors allow you to have multiple ranges in the sense that you can have sub-ranges. Thus you could have a partition by month, sub-partitioned by geography, say. In theory at least, there are products that allow you to have an infinite number of such sub-partitions. IBM, however, does not need to do this: instead it uses range partitioning in conjunction with its existing hash capabilities and the existing multi-dimensional clustering that we have already mentioned. The way that these work together is shown in Figure 7.

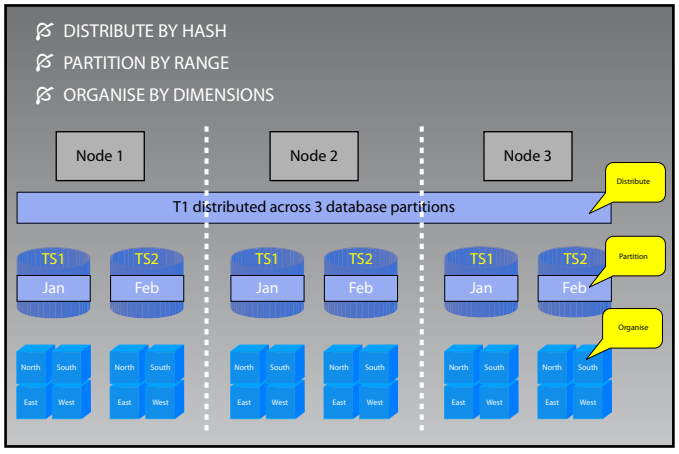

Figure 7: *DB2 data partitioning*

Now, IBM refers to this as distribution, partitioning and organisation but in practice these are all different methods of partitioning: use hash partitions to distribute the data in a way that the data does not become unbalanced, sub-partition using range techniques and then partition by dimension. Note that you can have any number of dimensions when using multi-dimensional clustering.

Now, in a great many instances a dimension could be a range: for example, in Figure 7, the data has been organised by geography. So, the question is why IBM opted for multi-dimensional clusters rather than support for sub-ranges? Moreover, why did it introduce the former before the latter? The answer is that multi-dimensional clustering is not just about how you store the data (close together so you get better performance) but is also intrinsically linked to the way that you want to do slice and dice and similar things within a query environment. The truth is that you could build an n-tier range partitioning model (where the database supports it—many do not) but it would be complex and it would mirror the work that you have to do when setting up relational cubes.

There is also the opposite question: if multi-dimensional clustering is so good, why do you need range partitioning? There are actually two answers to this question: the first is that you may want to segment data in a way that is not material to your needs for slicing and dicing and the second is that you may want to use range partitioning without multi-dimensional clustering. A relevant example that applies to both these cases is when partitioning is used in conjunction with Label-

based Access Control (see next section). This allows you to partition by security level so that you might have "top secret" information in one partition, say, and "confidential" information in another.

An associated feature that DB2 9 provides is roll in/roll out support for partitions. Specifically, you can attach or detach partitions using the ALTER TABLE statement. This ability can be particularly useful in a data warehouse environment where you often need to load or delete data to run decision-support queries.

The bottom line is that, in competitive terms, range partitioning, in conjunction with the other features discussed, may provide a performance advantage (when compared to products that do not support unlimited range partitioning) and will offer an administrative advantage. One feature that we would like to see IBM introduce is support for replicated partitions, which can enhance query performance in large parallel environments. While we are not aware of any of its mainstream competitors having this facility, some of the data warehouse appliance vendors do.

Security

There are two major new security features in DB2 9: the introduction of Label-based Access Control (LBAC) and support for trusted contexts.

Label-based Access Control is an implementation of Mandatory Access Control, where the latter is a security system based on the principle that an administrator determines user access rights and that users may not assign less stringent access rights to any data that they have control over. This is in contrast to Discretionary Access Control (DAC) where, at least in principle, the owners of the data (users) determine who may or may not have access to it. What IBM has done is to implement LBAC as a complementary function to the DAC that has been the historic basis for security within DB2.

The point about the complementary nature of access security with LBAC is that the historical approach taken by IBM, with DAC, has been to apply this at the table level. In other words, a user could either look at data in a table or he or she could not. LBAC, on the other hand, is implemented at the row and column level, either individually or in conjunction. Thus, DAC is relatively coarse-grained and LBAC is much more granular, though you can use LBAC at the table level as well, if you wish to replace DAC. Note that you do not have to apply LBAC to all tables within a database.

In terms of the way that LBAC works it is not dissimilar from conventional access control: data is assigned a label and so is each user (using a hierarchy, group or tree-based approach, as required) and the two are compared at run time. However, it is not quite as simple as this. Simply comparing labels provides a very monolithic security structure, whereas you typically need something that is more flexible. So, LBAC also includes the concept of security policies, which are predefined rules, delivered with DB2, which you can apply whenever data is read or written. If these rules are not enough, it is also possible to assign 'exemptions' whereby particular individuals have special permission to access information that they would not normally be allowed to see.

The benefits of LBAC are two-fold: first, it provides more granular control over who can see and do what. Secondly, and perhaps more importantly, the combination of LBAC and DAC is much more flexible; in principle it allows you to implement the security policies that best suit your organisation, rather than being forced down a particular security path by the constraints of the database. This is both a comparative and competitive advantage.

The second aspect of security that is new in DB2 9 is that of trusted contexts. These are essentially a way of bridging between disparate systems and applications that have different security models. Trusted contexts are defined on the server and refer to the connections that exist between applications, databases or whatever. Connections may qualify as a trusted context based on one or more relevant attributes (userid, IP address and so on) and a context may also confer membership in a Role, though it cannot (in this release) assign a label. The big advantage of trusted contexts is that it avoids authentication costs within (especially) 3-tier systems.

Tuning

There are a number of different elements that relate to tuning that have been extended in this release, notably adaptive self-tuning memory, the design advisor, and automatic storage, though it is arguable that the last of these is more of an administrative function than a tuning one.

What IBM calls adaptive self-tuning memory is the ability of the database to detect the workload on the database (including SAP details if that solution is running) and to tune the memory available based on the needs of that workload, re-distributing memory between processes as required to optimise the workload.

In effect, automatic storage support does the same thing for storage, except that it is not dynamically based on workload but is instead based on defined policies (rules) that you set up for different storage types. It requires the use of the Data Managed Storage (DMS) model but what it allows you to do is to have faster and slower disks (or other storage media) on the same system and then allows you to allocate data that is, say, more than three months old to slower disks while keeping more up-to-date data on the fastest disks. The system can also allocate and grow storage on demand, which is a feature that specifically supports SAP (so policies might be based on information from within the SAP rather than the DB2 environment). In other words, it is providing at least a part of the solution for information lifecycle management (ILM) though IBM's offering here is not yet complete.

The Design Advisor itself has not been especially enhanced in this release. In other words it can still recommend the creation of indexes, materialised query tables and multi-dimensional clustering dimensions, and can then automatically create these if required. However, the previously available facilities for recommending partitions has been extended to include the newly introduced range partitions and, as with the self-tuning memory previously discussed, it now has knowledge of the workload on the system (including SAP).

All of these features lead directly to administrative improvements and, indirectly, to performance benefits in the case of self-tuning memory and the Design Advisor; and cost reductions in the case of automatic storage.

Other features

As one would expect from a major release of a major product, there are a large number of new and enhanced features and capabilities in DB2 Viper apart from those that we have discussed. These range from new on-line facilities such as dynamic bufferpool operations; online index creation and maintenance and online loading; to new automated features including backups and statistics collection, which will reduce administrative workload; and the removal of size limitations to Tablespaces.

There are also a number of new features to support developers, not least of which are the XML facilities already discussed. In addition to these there is also a new Eclipse-based developer workbench in place of the previous DB2 Development Center, there is a new stored procedure debugger and there are a number of other enhancements for developers. In particular, there is now support for online table reorganisations. Previously, using the ALTER TABLE command (used, for example, to drop or change a column) meant that you had to drop the table completely and re-create it, which was not just time consuming and complex but you also had to quiesce the database. This feature will be a boon to developers though it is not before time (competitive products have been able to do this for some years).

Conclusion

There are many new features in DB2 9 (Viper), of which we have highlighted some of the most important. Many of the benefits deriving from these new capabilities are conventional: reduced administration and management overhead leads to reduced cost and/or increased productivity; improved performance leads to better utilisation of existing hardware, which in turn means that upgrades and replacements can be put on hold, with a direct impact on the bottom line; enhanced security means that management can sleep easier in their beds at night and helps to support compliance requirements; in the case of SAP, understanding that environment leads to performance and administrative gains that lead to the benefits just outlined; and so on. However, there are two areas that we would particularly like to highlight.

The first of these derives from the row compression introduced in this release. Unlike performance, which has an indirect benefit in terms of hardware requirement, improved compression has a direct impact on your hardware needs and, therefore, related costs.

The final benefit that we believe derives from DB2 9 is arguably the most important but it is also the most intangible: it is the ability, provided by the new hybrid XML/relational storage (pureXML as IBM calls it), to create applications, combining these data types, that were not previously realistic possibilities. We do not know how many of these applications there are, nor what they might look like beyond the discussions in this paper. Nevertheless, we believe that they are there and that there are many of them. Thus the greatest benefit of DB2 9 is likely not to be performance or reduced cost or any of those things, but the new possibilities it opens up to companies that want to take advantage of the new facilities on offer.

To conclude: this release of DB2 introduces new capabilities that significantly exceed those of its competitors in a number of areas. In particular, its pureXML is a major advantage and in any company where XML is an important consideration (and this increasingly applies to all organisations), IBM should be at the head of the list when considering potential suppliers.

Bloor Research Overview

Bloor Research has spent the last decade developing what is recognised as Europe's leading independent IT research organisation. With its core research activities underpinning a range of services, from research and consulting to events and publishing, Bloor Research is committed to turning knowledge into client value across all of its products and engagements. Our objectives are:

- Save clients' time by providing comparison and analysis that is clear and succinct.
- Update clients' expertise, enabling them to have a clear understanding of IT issues and facts and validate existing technology strategies.
- Bring an independent perspective, minimising the inherent risks of product selection and decision-making.
- Communicate our visionary perspective of the future of IT.

Founded in 1989, Bloor Research is one of the world's leading IT research, analysis and consultancy organisations—distributing research and analysis to IT user and vendor organisations throughout the world via online subscriptions, tailored research services and consultancy projects.

Copyright & Disclaimer

Suite 4, Town Hall, 86 Watling Street East
TOWCESTER, Northamptonshire, NN12 6BS, United Kingdom

Tel: +44 (0)870 345 9911 – Fax: +44 (0)870 345 9922
Web: www.bloor-research.com – email: info@bloor-research.com

…optimise your IT investments