D L Childs

[SPIO: 9/01/07]

SET-PROCESSING AT THE I/O LEVEL

A Performance Alternative to Traditional Index Structures

Abstract

It is generally believed that *index structures* are essential for high-performance information access. This belief is false. For, though indexing is a venerable, valuable, and mathematically sound identification mechanism, its logical potential for identifying unique data items is restricted by structure-dependent implementations that are extremely inefficient, costly, functionally restrictive, information destructive, resource demanding, and, most importantly, that preclude *data independence*. A low-level *logical* data access alternative to *physical* indexed data access is *set-processing*. System I/O level set-processing minimizes the overall I/O workload by more efficiently locating relevant data to be transferred, and by greatly increasing the information transfer efficiency over that of traditional indexed record access strategies. Instead of accessing records through imposed locations, the set-processing alternative accesses records by their intrinsic *mathematical identity*. By optimizing I/O traffic with *informationally dense* data transfers, using no physical indexes of any kind, low-level set-processing has demonstrated a substantial, scalable performance improvement over location-dependent index structures.

KEYWORDS: Extended Set Theory, Scopes, Set-Processing, Mathematical Identity, Informationally Dense I/O, Adaptive Data Restructuring, XSP Engine, XSP Information Access Accelerator, XSP I/O Subsystem

INTRODUCTION: Traditional I/O access strategies are based on a common principle: *preserve data-content integrity by preserving data-record integrity*. Thus the database record becomes the fundamental unit of data access and *index structures* are generated to facilitate locating records of application interest. Access strategies based on index structures make very inefficient use of a system's I/O capability. This paper will introduce the concept of system I/O level *set-processing* for maximizing a system's I/O efficiency by eliminating the use of index structures. Just as radio waves obviated the need for telegraph poles, set-processing obviates the need for index structures. By eliminating index structures, set-processing reduces data access maintenance and overhead, reduces required CPU processing, minimizes I/O volume, and improves overall system performance by:

- Locating data based on information content, not through artificially imposed index structures.
- Supporting informationally dense I/O data transfers within and between systems.
- Adaptively restructuring data for improved application processing,
- Accelerating application performance by accessing only successively relevant data.

With set-processing, *updates* are additive, non-destructive, so the unnecessary integrity checking overhead disappears. Data-content integrity preservation is a by-product of set-processing and not an imposed management concern. Set-processing reduces meta-management overhead and allows broadcasting operations to remote platforms. The extraction of information in parallel is a natural act with set-processing. Most importantly, set-processing imposes a truly *data independent* architecture that frees applications from any concern with the internal representation of the data.

Since implementations of set-processing require mathematical objects not definable under classical set theory, extensions to set theory are required to provide mathematical identities for modeling physical data representations. Resulting extended set processing, XSP, operations allow the implementation of mathematically well-defined necessary transformations of physical data representations. XSP systems have proven commercially productive for many years, and recent independent benchmarks have demonstrated orders of magnitude improvement in rapid information access when an XSP Information Access Accelerator is coupled with existing index structure based systems.

Copyright © 2005 INTEGRATED INFORMATION SYSTEMS ▪ ANN ARBOR MI ▪ iis@umich.edu

A NEED FOR INTEGRATED INFORMATION ACCESS SYSTEMS

Integrated information access is a high priority concern of both government and industry. Data is being accumulated faster than information can be accessed. The greatest technical challenge today is being able to extract information from data in the shortest possible time. Traditional data access methods are not up to the challenge. Thirty year old structure-dependent data access technologies were never intended for use on today's distributed high performance hardware platforms.

The government has recently announced critical concern with the deficiency of existing data access technologies and the urgent need to develop '*revolutionary advances* in technology' and not just '*evolutionary improvements* to the existing state of practice'.

XSP: A Revolutionary Technology: Since all data access technologies rely on *pointers*, what could be more revolutionary than a data access technology that did not require pointers? Extended Set Processing (XSP) is just such a technology, relying on pre-defined generic data access *operations* instead of depending on a constant regeneration of data access *structures*.

Future Integrated Information Access Systems (IIASs), supported by an *operation-centric* data access technology instead of a *structure-dependent* data access technology, can provide:

- **RAPID INFORMATION ACCESS OF LIVE DATA**
- **RAPID INFORMATION ACCESS OF LEGACY DATA**
- **RAPID INFORMATION ACCESS OF DISPARATE DATA**
- **RAPID INFORMATION ACCESS OF XML & RDM DATA**
- **RAPID INFORMATION ACCESS OF DISTRIBUTED DATA**

Clearly, existing Database Management Systems (DBMSs) can not support these capabilities.

Revolution vs. Evolution: If IIASs are expected to ever replace existing DBMSs then any potentially qualifying *revolutionary technology* must provide for *evolutionary advances* to existing systems; and future IIASs must provide all the capabilities of current DBMSs. There is even a precedence for a revolutionary technology allowing evolutionary advances to prior systems. Specifically, the revolutionary technology of the Relational Data Model (RDM) allowed Transaction Data Processing Systems (TDPS) to evolve into Relational Database Management Systems (RDBMSs). Similarly, any potential candidate for the role of 'revolutionary technology' must enable the evolution of today's DBMSs into tomorrow's IIASs. Some of the intrinsic *operation-centric* advantages of IIASs over *structure-dependent* DBMSs are:

EXISTING DBMS	FUTURE IIAS
Significant Data Structure Loading Costs	No Data Structure Loading Costs
New Access Structures For New Data	Same Access Operations For All Data
Performance: A Function Of Total Data	Performance: A Function Of Relevant Data
Data Updates Destroy Information	Data Updates Add Information
Only Predefined Data Relationships Are Accessible	All Derivable Data Relationships Are Accessible
Storage Structuring Frozen For All Data Access	Storage Restructuring For Optimized Data Access
New Data Disrupts Query Process	New Data Assimilated Into Query Process
Disparate Data Treated As Disjoint Structures	Disparate Data Integrated By Access Operations
Inhibits Real-Time Processing Of Live Data	Enables Real-Time Processing Of Live Data
Structure-Dependent Native XML Support	Operation-Centric Native XML Support
Restricted Interoperability Between XML Systems	Broad Interoperability Between XML Systems

XSP technology enables *structure-dependent* DBMSs to evolve into *operation-centric* IIASs.

INFORMATION ACCESS SYSTEMS: An Information Access System, IAS, is the coupling of an *external* environment of people with an *internal* environment of computers to process data. System architectures have to address the functional requirements of the external environment, the performance requirements of the internal environment, and the interfacing of these two environments to productively and efficiently extract information from data.

System Performance: System performance is a measure of the overall productivity and efficiency of a system. The productivity and efficiency of today's IASs can be greatly improved by replacing structure-dependent indexed-record architectures with operation-centric set-processing architectures. There are basically two ways with which overall performance can be achieved:

- **NOT DOING AT ALL, THAT WHICH DOES NOT NEED DOING.**
- **DOING NEAR OPTIMALLY, THAT WHICH DOES NEED DOING.**

Operation-centric set-processing architectures allow both of the above by eliminating processing and I/O that is not necessary; and by improving the efficiency of the processing and I/O that is necessary. Traditional system architectures use the *data-record* as a basis for data representation, access, and manipulation whereas, set-processing uses the *data-content* as a basis for data representation, access, and manipulation. This simple distinction greatly impacts the management overhead, processing requirements, and I/O transfer efficiencies of each architecture.

The argument to be presented is predicated on the fact that records are chosen to represent data and that the choice of data representation is preserved during I/O transfers in order to ensure that the data content is also preserved. By being able to ensure data content integrity without preserving the data representation allows representations to be dynamically manipulated in many ways in order to consume less physical space when being transferred and to be more amenable to application processing requirements. This space savings and application friendliness translates directly into improved performance. Since data content is an intrinsic property of data and not an artificially imposed wrapping, mathematical properties of the data content can be used to reduce data management overhead that are incurred when artificial wrappings are imposed to reflect data content.

System I/O Interfaces: *I/O* is the physical process of moving something from one location to another location. At the internal level, IAS architects have to contend with moving data between four distinct environments. The *real world* enterprise, or external environment of applications; the CPU/RAM processing environment of the computer platform; the platform's secondary storage environment for persistent data; and between remote platforms. These four major I/O interfaces, respectively, are:

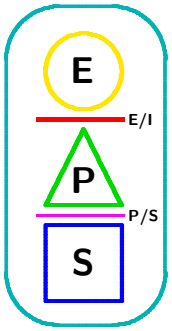
- CIO: Conceptual I/O Interface [data presentation]
- LIO: Logical I/O Interface [data processing]
- PIO: Physical I/O Interface [data preservation]
- NIO: Network I/O Interface [data propagation]

The 'something' being moved is different at each I/O interface, but in each case it qualifies as *data*. The CIO moves data between people and machines. The LIO moves data between memory and cache locations. The PIO moves data between the processor and storage. The NIO moves data between platforms. The individual and overall performance of each of these interfaces can be improved using operation-centric system architectures. Data ideally represented for one is unlikely to be the ideal representation for another. The external architecture of an IAS is only concerned with feeding and receiving data at the CIO interface. The only I/O demand of the external architecture is that the data received from the internal level be meaningful, efficiently processable, and with data content integrity preserved. The primary I/O concern of the internal architecture is preserving data content integrity while moving data representations across the four I/O interfaces. The choice of data representation also effects resources required to process and manage the data.

The most architectural freedom is available when two data content sharing I/O environments are *data independent*. This means that one or both environments has absolutely no knowledge whatsoever of the choice of data representation in the other environment. The best that traditional systems can offer is *data isolation*, which obscures most details of data representation but still shares influential knowledge.

Data independence at any I/O interface requires that data content be shared without sharing any knowledge about data representation. This can not be achieved with architectures that share data records. The sharing component has to be intrinsically non-physical. The mathematical identity of data content is the shareable I/O component used by set-processing architectures. By relying on the mathematical identity of data content, all four I/O interfaces supported by internal architectures can be mutually data independent. This in turn gives internal architects the greatest freedom for managing performance related resources without compromising the functionality provided to external environments.

SYSTEM ARCHITECTURES: Every IAS combines two very distinct and orthogonal architectures. One is user-friendly, addressing conceptual relationships in the external environment. The other is machine-friendly, addressing the movement and processing of data in the internal environment. The user-friendly side, or external architecture, is concerned with application functionality by supporting the *logical* representation, access, and manipulation of data. The machine-friendly side, or internal architecture, is concerned with system performance by supporting the *physical* representation, access, and manipulation of data. The visual aid to the right diagrammatically depicts the separation between the external environment of logical elements and the internal environment of physical elements. The E/I interface, or CIO, communicates data between people and machine. At this interface both the representation and content of the data are meaningful and processable by both sides. Once data is accepted by the processing component, **P**, of the internal environment it is processed and manipulated within **P** through LIO interfaces. Data is preserved by the secondary storage component, **S**, of the internal environment after being transferred through PIO interfaces. Data is transferred between platforms through NIO interfaces. Though data is the commodity being transferred through all four of these I/O interfaces the form and content of the data varies greatly and has a direct bearing on the overall performance of an IAS platform



IAS PLATFORM

OPERATION-CENTRIC ARCHITECTURES: Prior to the early 1970s both the external and internal architectures were *structure-dependent* in nature. That is, the data was not operated on directly by a knowledge of how data components were logically related, but operated on indirectly through a knowledge of how the data components were stored. At this time, external IAS architectures offered little more than a WYSIWYG reflection of the physical representation of data. More complex architectures were developed, all of which required an application user to navigate through storage. As architectures matured, commercial database applications became progressively more ambitious and maintenance costs rose precipitously. The interfaces between user-friendly external architectures and machine-friendly internal architectures became so intricate and fragile, that the slightest modification to a database structure could disable the entire system.

The navigation burden was also becoming quite severe. In 1970 E. F. Codd introduced the idea of an *operation-centric* architecture, one that relied on operations that treated data as an operand by relying only on a knowledge of data component relationships. Thus, the Relational Model of Data, RDM, as proposed allowed operations on data supported by the external architecture to be independent of the representation of the data supported by the internal architecture. This introduction of an operation-centric external architecture changed the landscape of the entire computer industry. Though both architectures have greatly matured since the mid 1970s, the predominant design for systems today is still an operation-centric external architecture supporting application functional-

ity, coupled with a structure-dependent internal architecture supporting system performance. The ideal system would be to have an operation-centric external architecture supporting application functionality, coupled with an operation-centric internal architecture supporting system performance. It is interesting to note that the RDM deals exclusively with the external environment.

> **The internal level will not be 'relational', because the objects at that level will not be just (stored) relational tables - instead, they will be the same kinds of object found at the internal level of any other kind of system (stored records, pointers, indexes, hashes, etc.). In fact, relational theory as such has nothing whatsoever to say about the internal level; it is concerned with how the database looks to the user. - Date [Da95] p.31**

Ideal Internal Architectures: An essential requirement for any internal architecture is that it provide all the functionality that existing internal architectures already provide, though it is not required that the functionality be accomplished by the same means or with the same strategies. Ideally an internal architecture should provide a *data independent* interface to its associated external architecture. This of course was a major potential offered by the RDM. It has never been commercially realized. Many advocates of RDBMSs will insist that SQL provides such data independence, but of course they will always add that one has to be careful how queries are expressed and how indices are declared, for they affect performance - and nobody laughs.

Thus, an internal architecture of any IAS should be totally transparent to the application environment, and in addition, provide independent control over the physical representation of data, support any mix of data representations, adaptively restructure data representations to accommodate any dynamic mix of application requirements, be able to avoid locking application access to data, offer non-destructive updating, share access to data representations on remote platforms, and ensure the integrity of data under any and all operational conditions. Such internal architectures are not prevalent in today's IASs, but they could be. The mathematics for doing so exists.

ROLE OF MATHEMATICS: The role of mathematics in system design is to provide rules for construction that ensure consistent behavior as designs become functionally and conceptually complex. Mathematical foundations can provide the difference between a program that works and one that does not fail. For simple tasks, mathematics can add more confusion than clarity, but when we run out of fingers and toes, mathematics can be of value. The value added is to provide simple rules for managing the conceptually complex.

For example, the simplest computer is a light switch. The logic is a binary conditional. The hardware will fail before the on-off logic will. The next largest computer is a pair of light switches. Again the hardware will fail before the 4-condition intent does. Since any digital computer is logically equivalent to a string of 'n light switches' ('n' being somewhat large), and since by induction for a 'n+1 collection of light switches' the hardware will fail before the software logic does, there is no reason why any software program written today should fail before the hardware does. The logic of this argument fails only because we have no mathematical control over conceptual complexity as 'n' grows larger.

MATHEMATICAL CONTROL: Even in the most advanced, most sophisticated systems today, there is no mathematical control over the definition, manipulation, and transformation of data. (The closest mathematical control available is the RDM, which only applies to specific external architectures.) Without mathematical control over the complex tasks required by an internal architecture, there needs to be control mechanisms added to the architecture to control the complexity. These control mechanisms are also complex and without mathematical control over their construction and operation. Thus these control mechanisms also require auxiliary control mechanisms. Such a system will never be mathematically sound, but will always be large, costly, inefficient, and error prone.

Mathematical control simply means that those items being controlled can be given a precise, non-ambiguous, definitive definition under the rules (or axioms) of some chosen mathematical discipline.

In terms of Classical Set Theory, CST, such items would have a mathematical identity. When being controlled, items with a mathematical identity are called *operands* and the controlling mechanisms are called *operations*. If an item has no mathematical identity, operations can not be applied and thus there is no mathematical control over the behavior of such items. In database management environments, the items of interest are representations of data.

Data representations like tables, files, records, lists, B-trees, sectors, tracts, arrays, clusters, XML-documents, memory blocks, are conceptually well understood and employed in most systems. Though these representations can be defined, manipulated, and transformed in a programming environment, they currently have no well-defined mathematical identities and therefore can not be subject to mathematical control. All data representations of interest must be mathematically well defined before any mathematical control can be exercised over data transformations of any kind. In practice, data representations are conceived to be conceptually convenient and thus do not respond well when treated as mathematical objects.

SET-PROCESSING: Set-processing provides mathematical control over data representation, data access, and data manipulation. Set theory, sets, and set operations are familiar terms to users and designers of database systems. *Set-processing* is not quite so familiar, but as the name implies it is a mathematically well defined process accepting sets as input and producing sets as output. Since set theory is mathematic's most powerful tool for mapping objects from one domain to another, it would seem to be a natural candidate for supporting data transformation interfaces within and between computer environments.

In fact, set-processing has a history of computing applications. The most familiar application is E. F. Codd's Relational Model of Data [Co70], that gave birth to the Relational Database Management System, RDBMS. Though RDBMS supports set-processing, the input sets (or RDM-operands) are confined to being *RDM-relations* and not sets in general. Set-processing was also employed by McCarthy [**?**] to support the foundations of LISP. Again, the operands were not general sets, but nestings of *ordered pairs*.

Though in both of these cases set theory provided many benefits, it also imposed severe restrictions. Codd was restricted to using sets of unnested, arbitrary length *tuples*, but no nested sets of any kind. McCarthy was restricted to arbitrary nestings of *ordered pairs*, but no arbitrary length tuples. Thus set theory constrained both uses and very early on earned the reputation of not being a general model for data representation and manipulation. The reason for set theory's inadequacy is quite simple, proper sets have no structure, yet all representations of data have a structure.

If set-processing is to be a viable candidate for supporting low-level I/O data access strategies, it will have to allow structured-sets as input and output. Extensions to the axioms of set theory provide the mathematical machinery to support these structured sets. With the ability to formally define structured sets, set-processing at the I/O level becomes possible.

Informationally Dense I/O: Traditional indexed-record architectures do not (and can not) support informationally dense I/O traffic. A *record* as a unit of access, may be only 5% to 20% useful to the application, the I/O transfer buffers may at most be 70% full of 20% useful data which are then accessed randomly which is 5 to 80 times slower than if accessed sequential. By using set-processing informationally dense transfer buffers and sequential access, the I/O times can be many times faster than currently available.

STRUCTURED SETS: In order to define a set-theoretic mapping between two domains, the objects in both domains need to be formally defined as sets. A mapping between a *logical* environment of conceptual objects and a *physical* environment of record representations requires that both domains be expressed set-theoretically. This presents a problem. Logical objects like SQL-tables, RDM-relations, and XML-structures are *structured sets* which, like proper sets, have elements, but, unlike

proper sets, have an extra condition (or structure) associated with each element.

The most familiar additional condition placed on elements of a set is that of order. The tuple $< a, b, c >$ and the tuple $< c, b, a >$ are familiar examples of structured sets. They both reflect the idea of sets having the same elements a, b, c, but with an ordering condition that makes $< a, b, c > \neq < c, b, a >$. Tuples even allow duplicate elements, $< a, a, c, c, c >$, which proper sets do not. Since, duplicate elements make no sense in set theory, modeling physical sets of non-uniquely identifiable elements can present a modeling problem for real world situations. Without unique names, the RDM models a swarm of bees as a swarm of bee. SQL tries to remedy this diminished modeling resolution with *duplicate rows*, (see Date & Darwin [DD97] p.438. As will be shown, an SQL-table with duplicate rows, does have a well-defined mathematical identity.

Since classical set theory, CST, supports the concept of a tuple, it would appear that CST supports structured sets. In fact, the tuple is universally used to formally model the concept of a record. The record-as-tuple is an essential concept in the RDM for defining data in terms of RDM-relations. Unfortunately, the CST support for tuples is an illusion.

Tuples & Records: There is no problem with records *per se* or even with using tuple notation to formally reflect an ordered sequence of objects. The problem arises when (and only when) tuples are treated as operands. They behave very badly. Even simple 2-tuples, or ordered pairs, behave unpredictably. For example, $< a, b >$ and $< a, c >$ are well-defined sets and therefore their behavior as operands under set operations is also well-defined. Thus, the expression $< a, b > \cap < a, c >$ is a well-defined set theoretic expression yielding a well-defined set result, which happens to be $< a, a >$ (*a la* Kuratowski). This is not necessarily an expected nor very useful result, but it is set-theoretically correct. It may come as a surprise to Relational advocates that the basic mathematical building block of the *Relational Model of Data*, the n-tuple, is mathematically unsound (*i.e.* it has no unique mathematical identity). This n-tuple anomaly still seems to be a well kept secret, even though it was well documented by Skolem in 1957, [Sk57].

To be a useful modeling medium, a set theory should support reasonable behavior of *n-tuples* under set-theoretic operations. The result of $< a, b > \cap < a, c >$ should be $< a >$, and the result of $< a, b, c > \cap < x, b, y >$ should be $< -, b, - >$. By extending the concept of set to equate $< a, b, c >$ to $\{a^1, b^2, c^3\}$ and $< x, b, y >$ to $\{x^1, b^2, y^3\}$, then $< a, b, c > \cap < x, b, y >$ would give $\{b^2\}$ as a result, which is more compatible with our expectations.

If the foundations of set theory were extended to supported structured sets, the expected behavior of records as tuples could be formally modeled and provide mathematical control for the design, implementation, and use of record-oriented system implementations. Since I/O interfaces typically map between structured set environments, an extended set theory could provide a formal model of structured set oriented I/O interfaces.

Extended Set Theory: Extended Set Theory, XST, simply extends the basic membership condition of CST to include a condition '**s**' for structure, or scope. In CST '**x**' is either a member of '**A**' or '**x**' is not a member of 'A'. In XST '**x**' is either a member of '**A**' under constraint '**s**' or '**x**' is not a member of '**A**' under constraint '**s**'. All CST sets are subsumed under XST when the constraint is 'NULL'. Under XST all data representations, both logical and physical, have mathematical identities, and are thus subject to mathematical control. Armed with this XST definition for n-tuple, all the intuitive notations about the behavior of records can be preserved when modeled by n-tuple notation. This also allows anomalies in the RDM to be corrected and would legitimize the bragging rights of the RDM as being mathematically sound. As is shown below and more extensively in [Ch00], a more exotic use of scopes also provides a formal foundation for processing XML-Structures. It should be stressed that the concept of scopes is not just a notational slight-of-hand that can be supported by any existing set theory. [Ch05].

Power of Scopes: Classical sets, by definition, are unstructured collections of elements, where the elements themselves can also be sets. By contrast, extended sets can be viewed as classical sets with *structure*. This concept of a 'structured set' is no more an oxymoron than is the term 'multi-valued-function'. So, in a naive way, scopes can be considered to add *structure* to sets. What that structure is depends on the choice of scope values. The choice of integers, though obvious, is not necessarily the only choice with practical ramifications for software development.

Any subset of a computer memory can be faithfully modeled by an extended set, where the elements of the set are byte values and the scopes of the elements are memory addresses. As simple as this example might be, it is an example of a component of the internal machine environment that now has a legitimate mathematical model. True that the notation is not new, but the underlying membership is now unambiguous. Thus two subsets of RAM can now be treated with mathematical rigor: their intersections, unions, symmetric differences, and relative complements are now well-defined.

By extending set operations beyond their usual reliance on elements only, and considering scope relationships (like every $i + 4$ value), then projections and selections can be legitimately defined on subsets of memory. Certainly, there are computer programs that already do this, but now there can be computer programs that have a rigorous mathematical model to ensure consistency and reliability.

Data & Extended Sets: Most people, familiar with SQL, know of its reliance on the RDM, and can recognize that the following twelve tables all represent exactly the same underlying RDM-Relation.

$$\mathbf{R}_1 = \begin{array}{|c|c|c|}\hline A & B & C \\ \hline a & b & c \\ \hline u & v & w \\ \hline \end{array} \qquad \mathbf{R}_2 = \begin{array}{|c|c|c|}\hline A & B & C \\ \hline u & v & w \\ \hline a & b & c \\ \hline \end{array} \qquad \mathbf{R}_3 = \begin{array}{|c|c|c|}\hline A & C & B \\ \hline a & c & b \\ \hline u & w & v \\ \hline \end{array} \qquad \mathbf{R}_4 = \begin{array}{|c|c|c|}\hline A & C & B \\ \hline u & w & v \\ \hline a & c & b \\ \hline \end{array}$$

$$\mathbf{R}_5 = \begin{array}{|c|c|c|}\hline B & A & C \\ \hline b & a & c \\ \hline v & u & w \\ \hline \end{array} \qquad \mathbf{R}_6 = \begin{array}{|c|c|c|}\hline B & A & C \\ \hline v & u & w \\ \hline b & a & c \\ \hline \end{array} \qquad \mathbf{R}_7 = \begin{array}{|c|c|c|}\hline B & C & A \\ \hline b & c & a \\ \hline v & w & u \\ \hline \end{array} \qquad \mathbf{R}_8 = \begin{array}{|c|c|c|}\hline B & C & A \\ \hline v & w & u \\ \hline b & c & a \\ \hline \end{array}$$

$$\mathbf{R}_9 = \begin{array}{|c|c|c|}\hline C & A & B \\ \hline c & a & b \\ \hline w & u & v \\ \hline \end{array} \qquad \mathbf{R}_{10} = \begin{array}{|c|c|c|}\hline C & A & B \\ \hline w & u & v \\ \hline c & a & b \\ \hline \end{array} \qquad \mathbf{R}_{11} = \begin{array}{|c|c|c|}\hline C & B & A \\ \hline c & b & a \\ \hline w & v & u \\ \hline \end{array} \qquad \mathbf{R}_{12} = \begin{array}{|c|c|c|}\hline C & B & A \\ \hline w & v & u \\ \hline c & b & a \\ \hline \end{array}$$

The purpose for tables is to provide a 'visual convenience' for determining the 'membership' of values in a RDM-Relation. The 'rows' of a table correspond to 'tuples' in a RDM-Relation. 'Column names' of a table correspond to 'Domain names' in a RDM-Relation. 'Values' in a row correspond to 'values' in a 'tuple'. Thus it can be visually determined that all twelve tables represent the same RDM-Relation, for by picking any value and column name of any row (along with all other members of the same row) of any table, then that same value with the same column name can be found in some row (along with all other members of the previous row) of any one of the other tables.

It is important to note that even though the above twelve tables have very different representations, they can all be recognized as having exactly the same mathematical identity (*i.e.* represent the same RDM-Relation). Just as '7-1', '4+2', and '24/4' can all be recognized as having the same mathematical identity.

As shown above, this equivalence of the twelve tables is readily established by observation. In fact, observation is the only way it can be established, for there is no formal deductive process available in SQL or through the RDM that can establish the common identity of these twelve tables. In short, the concept of table does not have a well defined mathematical identity.

That all twelve tables represent the same unique RDM-Relation can be expressed by:

$$For \;\; 1 \leq i \leq 12, \qquad \mathbf{R}i = \Big\{ \; \{a^A, b^B, c^C\}, \{u^A, v^B, w^C\} \; \Big\}.$$

The formal definition (though mathematically precise) is arcane and devoid of any recognizable intuitive meaning. However, the notation itself (while establishing mathematical rigor) also preserves the visual convenience which motivated the use of tables in the first place.

The following is a very simple XML-structure:

$$\begin{array}{l} < P > \\ \quad < p > \\ \quad\quad < n > Alan < /n > \\ \quad\quad < a > 42 < /a > \\ \quad\quad < e > agb@abc.com < /e > \\ \quad < /p > \\ \quad < p > \\ \quad\quad < n > Mary < /n > \\ \quad\quad < a > 29 < /a > \\ \quad\quad < e > mky@abc.com < /e > \\ \quad < /p > \\ < /P > \end{array}$$

Which could be represented by:

$$P \;=\; \Big\{ \; \{ \; Alan^{<1,n>}, \; 42^{<2,a>}, \; abc.com^{<3,e>} \; \}^{<1,p>}, \{ \; Mary^{<1,n>}, \; 29^{<2,a>}, \; mky.com^{<3,e>} \; \}^{<2,p>} \Big\}.$$

When XML tags are embedded in XST scopes, XML-structures become well-defined extended sets, [Ch05]. Since existing XSP software already relies on scopes to support structures internal to the machine environment, it is a relatively easy task to extend the existing software to support structures external to the machine environment, like XML-structures captured as extended sets.

DATA INDEPENDENT INTERFACES Another name for this kind of interface is 'data independent' which was introduced by the Relational Data Model. 'Data Independence' is generally understood to mean that no knowledge of how data is represented on one side of the interface need be known by the other side of the interface in order to effect the transfer of and manipulation of data. The term, more accurately, should be 'data-structure independent' since knowledge of the 'data-meaning' certainly needs to be known if predictable outcomes are desired.

In usage, 'Data Independence' is currently accepted to mean sharing knowledge of data-content but not knowledge of data-representation across an interface. Implementing such an interface requires knowledge of the data-content independent of the data-representation, since only the data-content can be known by both sides of the interface.

It is generally accepted that the Relational Data Model provides a means for supporting data independence at the user/host interface. It is also generally accepted that there exists no means for supporting data independence at the host/storage interface. This paper purports to provide a means for supporting both.

MAPPING SQL TO XST: It has been argued that any data independent interface must support the mathematical identity of objects on both sides of the interface. It has been shown that all SQL-Tables have a mathematical identity making them legitimate mathematical operands. What has not been show is that the SQL commands that manipulate SQL-Tables are well-defined mathematical operations.

SQL operations on data are defined by SELECT, FROM, and WHERE. These three clauses, with all their complex conditions, specify the membership condition of a result set. An XST

equivalent expression of this result set is the data content transferred back and forth across the CIO interface. As with all membership conditions, it makes no difference how the membership condition is expressed, it always represents the same set. This set-theoretic property ensures that no matter how a SQL statement is formed, it should not have any bearing on the internal level execution of that statement. This is not the case with current systems where SQL formulations do have an impact on performance. Though there are only three basic forms of queries, they may get very, very complicated.

$$\begin{array}{ll} \text{SELECT} & < a_1, a_2, .., a_n > \\ \text{FROM} & \mathsf{T}_1, \mathsf{T}_2, .., \mathsf{T}_k \\ \text{WHERE} & F \end{array} \qquad \boldsymbol{\pi}_{(a_1,a_2,\ldots,a_n)}\Big(\boldsymbol{\sigma}_F\big(\mathsf{T}_1 \times \mathsf{T}_2 \times \ldots \times \mathsf{T}_k\big)\Big).$$

$$\begin{array}{ll} \text{SELECT} & < a_1, a_2, .., a_n > \\ \text{FROM} & \mathsf{T}_1, \mathsf{T}_2, .., \mathsf{T}_k \end{array} \qquad \boldsymbol{\pi}_{(a_1,a_2,\ldots,a_n)}\big(\mathsf{T}_1 \times \mathsf{T}_2 \times \ldots \times \mathsf{T}_k\big).$$

$$\begin{array}{ll} \text{SELECT} & * \\ \text{FROM} & \mathsf{T}_1, \mathsf{T}_2, .., \mathsf{T}_k \\ \text{WHERE} & F \end{array} \qquad \boldsymbol{\sigma}_F\big(\mathsf{T}_1 \times \mathsf{T}_2 \times \ldots \times \mathsf{T}_k\big).$$

The above are not intended to be self-explanatory, but are included as an assertion of functional completeness in modeling SQL behavior by XST. Once captured by XST it becomes a programming effort to decide which operations are going to be defined and implemented.

MAPPING SQL FROM XST TO XSP: Queries like these allow the XSP engine to make best use of its underlying set theoretic capabilities. There is a great deal of potential parallelism to be exploited that other implementations just can not take advantage of.

XSP ENGINE: The problem of formally mapping SQL to the Relational Algebra, and formally mapping the Relational Algebra to a physical implementation requires the existence of a unifying mathematical framework that would be able to resolve all the ambiguities, anomalies, and assumptions currently being used as mortar in existing systems.

Extended set notation, XSN, at least in principal, provides the mathematical machinery to do the job. All that now needs to be done is define XSP equivalents to the RDM operations and show how they map to implementations.

Relational Completeness: Following are XSP operations more than sufficient to support Relational Completeness along with the arithmetic, conditional, and aggregation functionality necessary to support complex SQL SELECT statements. These operations are also platform independent.

XSP Union:

For $Rship(\mathbf{A})$, $Rship(\mathbf{B})$, $\mathbf{XUN}(\mathbf{A}, \mathbf{B}, \mathbf{C})$:

$\mathbf{C} = \mathbf{A} \cup \mathbf{B} \;\; = \Big\{ x : \; x \,\epsilon\, \mathbf{A} \text{ or } x \,\epsilon\, \mathbf{B} \Big\}.$

XSP Intersection:

For $Rship(\mathbf{A})$, $Rship(\mathbf{B})$, $\mathbf{XIN}(\mathbf{A}, \mathbf{B}, \mathbf{C})$:

$\mathbf{C} = \mathbf{A} \cap \mathbf{B} \;\; = \Big\{ x : \; x \,\epsilon\, \mathbf{A} \;\&\; x \,\epsilon\, \mathbf{B} \Big\}.$

XSP Relative Complement:

For $Rship(\mathbf{A})$, $Rship(\mathbf{B})$, $\mathbf{XRL}(\mathbf{A}, \mathbf{B}, \mathbf{C})$:

$\mathbf{C} = \mathbf{A} \sim \mathbf{B} \;\; = \Big\{ x : \; x \,\epsilon\, \mathbf{A} \;\&\; x \notin \mathbf{B} \Big\}.$

XSP Symmetric Difference:

For $Rship(\mathbf{A})$, $Rship(\mathbf{B})$, $\mathbf{XSD}(\mathbf{A}, \mathbf{B}, \mathbf{C})$:

$\mathbf{C} = \mathbf{A} \triangle \mathbf{B} \;\; = \Big\{ x : \; (\; x \,\epsilon\, \mathbf{A} \;\&\; x \notin \mathbf{B} \;) \text{ or } \; (\; x \,\epsilon\, \mathbf{B} \;\&\; x \notin \mathbf{A} \;) \Big\}.$

These four XSP operations provide the basic Boolean set operations. Notice that the commands are only dependent on the arguments being *Rships* and not dependent on scope compatibility. These four XSP operations are meaningfully defined no matter what the scope sets are.

XSP Relational Product:

For $Rship(\mathbf{Q})$, $Rship(\mathbf{R})$, $\mathbf{XRELP}(\mathbf{Q},\mathbf{R},\mathbf{C})$:

$$\mathbf{C} = \mathbf{Q} \times \mathbf{R} \;=\; \Big\{\, z : \; (\exists x, y)\big(x \in \mathbf{Q} \;\&\; y \in \mathbf{R}\big) \;\;\&\;\; \big(\; z \;=\; x \cup y\big)\Big\}.$$

XSP Project:

For $Rship(\mathbf{Q})$, $\mathbf{XPJ}(\mathbf{s},\mathbf{Q},\mathbf{C})$:

$$\mathbf{C} = \boldsymbol{\pi}_s(\mathbf{Q}) \;=\; \Big\{\, y : \; (\exists z)\big(z \in \mathbf{Q}\big) \;\;\&\;\; \big(y = z^{\prec\widehat{s}\succ} \neq \emptyset\;\big)\Big\}.$$

XSP Scope Restrict (Relational):

For $Rship(\mathbf{Q})$, $\mathbf{XSR}(\mathbf{s},\mathbf{Q},\mathbf{C})$:

$$\mathbf{C} \;=\; \boldsymbol{\sigma}_s(\mathbf{Q}) \;=\Big\{\, z : \; (\exists t)\big(t \in s \;\;\&\;\; z \in \mathbf{Q}\big) \;\;\&\;\; \big(\; t(z)\;\big)\Big\}.$$

XSP Restrict (classical):

For $Rship(\mathbf{Q})$, $Rship(\mathbf{A})$, $\mathbf{XRS}(\mathbf{Q},\mathbf{A},\mathbf{C})$:

$$\mathbf{C} \;=\; \mathbf{Q}\big\|A \;=\Big\{\, z : \; (\exists a)\big(a \in A \;\;\&\;\; z \in \mathbf{Q}\big) \;\;\&\;\; \big(\; a \,\dot{\subseteq}\, z\;\big)\Big\}.$$

XSP Domain:

For $Rship(\mathbf{Q})$, $\mathbf{XDOM}(\mathbf{s},\mathbf{Q},\mathbf{C})$:

$$\mathbf{C} \;=\; \boldsymbol{\text{Đ}}_s(\mathbf{Q}) = \Big\{\, x : \; (\exists z)\big(\; z \in \mathbf{Q}\big) \;\;\&\;\; \big(x = z^{[s]} \neq \emptyset\;\big)\Big\}.$$

XSP Range:

For $Rship(\mathbf{Q})$, $\mathbf{XRAN}(\mathbf{r},\mathbf{Q},\mathbf{C})$:

$$\mathbf{C} \;=\; \boldsymbol{\text{Ŕ}}_r(\mathbf{Q}) = \Big\{\, y : \; (\exists z)\big(\; z \in \mathbf{Q}\big) \;\;\&\;\; \big(y = z^{\prec\widehat{r}\succ} \neq \emptyset\;\big)\Big\}.$$

XSP Natural Join:

For $Rship(\mathbf{Q})$, $Rship(\mathbf{R})$, $\mathbf{XJOIN}(\mathbf{Q},\mathbf{R},\mathbf{C})$:

$$\mathbf{C} = \mathbf{Q} \bowtie \mathbf{R} \;=\; \Big\{\, z : \; (\exists x, y, \sigma)\big(x \in \mathbf{Q} \;\&\; y \in \mathbf{R}\big) \;\;\&\;\; \big(\;\; x \cup y \;\neq \emptyset \;\longrightarrow\; z = x \cup y\;\big)\Big\}.$$

XSP Theta-Join:

For $Rship(\mathbf{Q})$, $Rship(\mathbf{R})$, $\mathbf{XTJN}(\mathbf{T},\mathbf{Q},\mathbf{R},\mathbf{C})$:

$$\mathbf{C} = \mathbf{Q} \underset{\mathbf{T}}{\bowtie} \mathbf{R} \;=\; \Big\{\, z : \; (\exists x, y, t)\big(x \in \mathbf{Q} \;\&\; y \in \mathbf{R} \;\&\; t \in \mathbf{T}\big) \;\;\&\;\; \big(\;\; t(x,y) \;\longrightarrow\; z \;=\; x \cup y\;\big)\Big\}.$$

XSP Image:

For $Rship(\mathbf{Q})$, $Rship(\mathbf{A})$, $\mathbf{XIM}(\mathbf{r},\mathbf{Q},\mathbf{A},\mathbf{C})$:

$$\mathbf{C} \;=\; \mathbf{Q}[\![\mathbf{A}]\!]_r \;=\Big\{y : \; (\exists a, z)\big(a \in \mathbf{A} \;\&\; z \in \mathbf{Q}\big) \;\;\&\;\; \big(\; a \,\dot{\subseteq}\, z\big) \;\;\&\;\; \big(y = z^{\prec\widehat{r}\succ} \neq \emptyset\big)\Big\}.$$

XSP Divide:

For $Rship(\mathbf{Q})$, $Rship(\mathbf{R})$, $\mathbf{XDV}(\mathbf{Q},\mathbf{R},\mathbf{C})$:

$$\mathbf{C} \;=\; \mathbf{Q} \;\div\; \mathbf{R} \;=\; \Big\{\, z : \; (\exists x, y)\big(x \in \mathbf{Q} \;\&\; y \in \mathbf{R}\big) \;\;\&\;\; \big(\; \emptyset \;\neq\; z \;=\; x \sim y\big)\Big\}.$$

XSP Relative Product:

For $Rship(\mathbf{F})$, $Rship(\mathbf{G})$, $\mathbf{XRLP}(\mathbf{s},\mathbf{F},\mathbf{G},\mathbf{C})$:

$$\mathbf{C} \;=\; \mathbf{F} \,/_s\, \mathbf{G} \;=\Big\{\, z : \; (\exists f, g)\big(\; f \in \mathbf{F} \;\;\&\;\; g \in \mathbf{G}\;\big) \;\;\&\;\; \Big(\;\big(\; \mathcal{S}c(\mathbf{F}) \;\sim\; \breve{s}\;\big) \;\cap\; \big(\; \mathcal{S}c(\mathbf{G}) \;\sim\; \mathcal{S}(s)\big) \;=\; \emptyset\;\Big)$$
$$\&\;\; \Big(\; f^{\prec\widehat{s}\succ} \;\dot{\subseteq}\, g \;\longrightarrow\; z = f^{\llcorner s \lrcorner} \cup g^{\llcorner \widehat{s} \lrcorner}\Big)\Big\}.$$

It should be clear from the above list of operations, that an XSP-Engine that supports these operations is Relationally Complete, since the Relational Algebra is a proper subset of these operations.

Relational Extensions: Relational completeness, by itself, does not allow for the arithmetic, compare, and aggregation capabilities necessary to support practical SQL applications. These have to somehow be woven into the RDM operations. These are usually treated as implementation considerations and are not formally embedded into a set-theoretic framework.

Since more ambiguity and imprecision is not really necessary, the XSP approach is to embed these capabilities into a formal framework. This can be done, not directly as Xops, but more in the spirit of 'intention decelerations'. That is, commands that instruct subsequent commands, Xops, that while they are constructing a particular set, also perform this task.

These commands can allow the generation of *Bags*, 'sets with duplicate elements' (an oxymoron), cross tabulated fields, arithmetic calculations, and aggregations simulating 'repeating group' be-

havior.

XSP Calculation Conditions:

For $Rship(\mathbf{A})$, $\mathbf{XCALC}(\mathbf{s}, \mathbf{A})$:

This operation declares to the XSP-Engine that the set 's' contains conditional commands that are to be enforced when the 'set-to-be', **A**, gets generated. The set-theoretic details of this operation are a little much for this paper.

XSP Cross Tabulation:

For $Rship(\mathbf{A})$, $\mathbf{XCTAB}(\mathbf{s}, \mathbf{A})$:

Like the previous command, this operation declares to the XSP-Engine that the set 's' contains tabulation commands that are to be enforced when the 'set-to-be', **A**, gets generated.

XSP-Engine Additions:

The above XSP commands along with the set deceleration commands are more than adequate as a generic interface to an XSP-Engine for expressing all data manipulation requirements of SQL and other high level interfaces.

To be useful additional commands are needed for 'Universe Management'. These include all the usual routines for managing the creating, saving, and sharing data sets. These routines, though essential, do not impact the formal model and can be pretty much tailored to specific application needs. These routines will not be addressed in this paper.

For efficiency of operation, however, an additional collection of commands are sometimes more appropriate than the ones defined above. These additional operations take advantage of certain mathematical properties only available through a formal model that includes the ability to manipulate scope sets. XSN provides such a modeling, classical set theory does not.

INFORMATION DENSITY: In IASs, designed to rapidly extract relevant information from masses of disparate, distributed, dynamically changing data, the management of I/O traffic is a dominant performance concern. Ideal performance dictates transferring the minimum amount of data in the shortest possible time as infrequently as possible. This is sometimes translated (incorrectly) into 'get everything into RAM once'. The practical issues are actually much more complex, intertwined, and non-obvious. However, they all relate to a common unit of interest, the *Data Cluster*.

Data Clusters: Any collection of physically stored data is a *Data Cluster*. Any collection of physically stored data is also a structured set. Under XST, all structured sets have a mathematical identity. Therefore, under XST, all Data Clusters have a mathematical identity. Both OCAs and SDAs rely on the creation, manipulation, and transformation of Data Clusters to achieve their objectives. Since the objectives are the same for both, the major difference between OCAs and SDAs is in how they choose to achieve the objective. OCAs rely on the mathematical identity of Data Clusters, while SDAs rely on the physical structure of Data Clusters. (All index structures supported by SDAs are also Data Clusters, but their mathematical identities are ignored by SDA systems.)

LOADING DATA: Before any Data Cluster can be accessed, manipulated or transformed, it must be created or loaded into the system. Loading data is generally a very intensive and costly activity, even if it is done right, but especially costly if it is done incorrectly. Choosing exactly what data needs to be loaded, how it should be represented, determining if the data is complete or corrupt, and many other concerns make the loading of data a critical, resource-intensive activity. It is interesting how system vendors have conditioned customers into believing that loading costs are a necessary evil, and should best be ignored. A flagrant example of this is an industry standard benchmark that reports load-times, then fails to let them influence the performance results of the benchmark.

For RIA considerations load-time costs are crucial, time is money. For systems only interested in historic information, load-time completions are less critical, but for timely information access to fresh data, load-times are a dominant performance concern. Vendors of SDA systems do not seem to understand that loading data costs customers valuable time. Loading is not a no-op. If a daily data load takes 25 hours, and if it takes an additional 10 minutes per query, customers can not process many queries per day on the latest available data. To customers, total elapsed time, TET, includes loading times.

SDAs are very sensitive to the load process, since the overall system performance depends on the representation of data chosen at load-time. Loading with a SDA is euphemistically referred to as pouring concrete. Once loaded into their predefined structures, the data is "set" and cannot be easily restructured.

By contrast, OCAs are totally insensitive to the representation of data chosen at load-time. For, no matter what representation is chosen, it does not remain that way for very long. The performance of an OCA is directly dependent on the intrinsic relationships between data components, and the mathematical identity of a Data Cluster always preserves this, no matter what representation is chosen at load-time.

Locking & Updating: Since Data Clusters are sets, and since sets are immutable, it makes no more sense to update a Data Cluster than it does to update an integer (say from '6' to '7'). What does make sense is to replace a Data Cluster referenced by a specific name to a different Data Cluster now referenced by the same specific name. The same approach also works for integers.

Since records are Data Clusters, their mathematical identities can be used to achieve a non-destructive updating, which avoids the need for locking and allows internal system rollback (which in turn can support application transaction rollback) to any previous state maintained by available time-stamped Data Clusters. The non-destructive updating and application 'any-point-in-time' re-run strategies under OCAs are vastly different than the destructive updating and immediate rollback strategies under SDAs. (Their respective details are available in other papers.)

Performance Considerations: Following is a short review of considerations contributing to total performance. No one, individually, accounts for the dramatic performance difference, but their contributions are accumulative.

- More Informationally Dense I/O Transfers.
- Reduced Logical I/O (Memory Searches).
- Fewer Disk Seeks & Fewer Cache Misses.
- Informationally Dense Re-constituted Records.
- Less Total Data Transferred Over More Sustained I/Os.
- Less Meta-Data, Meta-Data Overhead, & Meta-Data Management.
- No Destructive Updates & No Locking Requirements.
- Reduced Loading Costs & No Re-Loading Required.
- No down Time for Re-Loading Required.
- All New Data Additive, No Re-Loading Required.
- Uses Data Access Operations, No Data Access Paths.
- Adaptive Data Restructuring As Applications Execute.
- No Index Rebuilding & Maintenance Costs.

The acceptance of their explanation depends very much on the concept of *mathematical identity* underlying the implementations of OCIA based systems. To paraphrase Sullivan, 'code follows concept'.

NO INDEX STRUCTURES: In *Through the Looking Glass* by Lewis Carroll, in response to the King's question about seeing his messengers, Alice replied "I see nobody on the road". To which the King replied "I wish I had such eyes to be able to see nobody!"

The following example from the TPC-H benchmark demonstrates using 'no index structures'. The execution results and further details of this example are contained in appendix-A. The highlights of why the use of scope driven set operations obviates the need for index structures will be summarized here.

TPC-H Query 9:

```
{ SELECT < NATION, o_year, sum_profit >
    FROM { SELECT < NATION, o_year, amount >
             FROM P, Q, L, O, S, N
           WHERE [P_P2] = LIKE '%[color]%'  &
           WHERE [P_P1] = [L_L2]            &
           WHERE [Q_Q1] = [L_L2]            &
           WHERE [Q_Q2] = [L_L3]            &
           WHERE [O_O1] = [L_L1]            &
           WHERE [S_S1] = [L_L3]            &
           WHERE [S_S4] = [N_N1]
         }
}
     With user defined functions:
          UDF1(O5,o_year)
              with V(o_year)     = Y(O5).
          UDF2(<S4,o_year>, amount, sum_profit)
              with V(sum_profit) = SUM(<S4,o_year>, amount)
          UDF3(L6,L7,Q4,L5,amount)
              with V(amount)     = (L6*(1-L7))-(Q4*L5).
```

WHERE statements are mapped to *scope conditionals* in XST. By [Q_Q2] = [L_L3], it is meant that the value of an element in an element of Q with a scope of Q2 is equal to the value of an element of an element of L with a scope of L3. Or, more simply, in $\mathsf{T} = \{\{a^A, b^B, a^C\}\}$ [T_A] = [T_C].

As was described earlier, in a generic SQL-friendly XSP engine, the interface algebra must provide a functional covering for SELECT statements. Recall that an XSP engine is any implementation of a data independent, set-processing algebra closed under extended set membership. Thus, the internal set representations can not be known and any interface collection of operations is admissible. Any two XSP engines can be interchanged as long as they are functionally equivalent and the interfaces have a known correspondence. The only allowable difference between two XSP engines is their performance. (An XSP engine could even have a structure-dependent indexed-record internal architecture. No one would ever be able to tell.)

For purposes of this example, it will be assumed that the XSP engine being used has the following two operations. (Even though these are not the exact command syntaxes used for the timings, they are mappable to ones that were.)

As was described earlier, the WHERE statement in SQL is mathematically ambiguous. In one case it specifies a Join condition between two SQL-tables, in the other it specifies a Restrict condition on a single SQL-table. Query 9 employs both types of WHERE statement. Both use scope conditionals to define the behavior of the operations.

First, a Restrict WHERE with a Project: RWHEREP(Scope_s, TSet_Q, Scope_r, TSet_C)

XSP-GENERIC *Restrict & Project:*

SELECT **r**
FROM **Q**
WHERE **s**

RWHEREP($\mathbf{s}, \mathbf{Q}, \mathbf{r}, \mathbf{C}$):

$$\mathbf{C} \;=\; (\mathbf{Q}\|\mathbf{s})^{\prec\mathbf{r}\succ} \;=\; \Big\{\, z^v :\; (z \in_v \mathbf{Q}) \;\;\&\;\; (\exists f)(\; f \in \mathbf{s} \;\;\&\;\; f(z))\Big\}^{\prec\mathbf{r}\succ}.$$

Next, a Join WHERE with a Project: JWHEREP(Scope_s, TSet_Q, TSet_R, Scope_r, Set_C)

XSP-GENERIC *Join & Project:*

SELECT $\mathbf{r}$
FROM $\mathbf{Q}, \mathbf{R}$
WHERE $\mathbf{s}$

JWHEREP($\mathbf{s}, \mathbf{Q}, \mathbf{R}, \mathbf{r}, \mathbf{C}$):

$$\mathbf{C} \;=\; \Big(\mathbf{Q} \;\bullet^s \mathbf{R}\Big)^{\prec\mathbf{r}\succ} \;=\; \Big\{\, z^v :\; (\exists x, y)(x \in_{\mathrm{v}} \mathbf{Q} \;\&\; y \in_{\mathrm{v}} \mathbf{R})$$
$$\&\;\; \big(\boldsymbol{\rho}_{s_1}(x).s_2.\boldsymbol{\rho}_{s_3}(y) \;\rightarrow\; z \;=\; x \cup y\big)\Big\}^{\prec\mathbf{r}\succ}. \;\; [s \;=< s_1, s_2, s_3 >]$$

Notice that for $s_2 \in \{=, \neq\}$, $\Big(\mathbf{Q} \;\bullet^s \mathbf{R}\Big)^{\prec\mathbf{r}\succ} \;=\; \Big(\mathbf{R} \;\bullet^s \mathbf{Q}\Big)^{\prec\mathbf{r}\succ}$, and

that for $u \;=< s_3, s_2, s_1 >$, $\Big(\mathbf{Q} \;\bullet^s \mathbf{R}\Big)^{\prec\mathbf{r}\succ} \;=\; \Big(\mathbf{R} \;\bullet^u \mathbf{Q}\Big)^{\prec\mathbf{r}\succ}$.

An XSP implementation consequence of the above set equivalence property is, depending on the respective cardinalities and record lengths of $\mathbf{Q}$ and $\mathbf{R}$, that the order of execution can make a performance difference. This decision would properly be made by the call to an XSP implementation, and not by the author of the SQL statement. An example of this follows.

These two generic XSP functions, like the ones described earlier, operate directly on physical set representations and produce physical set representations. How these functions are implemented has no impact on the set-theoretic behavior of the function. They are just like any other library routine, except they act directly at the I/O interface to secondary storage.

The notational convention 'Ai:Bj' means interchangeable names. When Ai and Bj are scope values it implies that their corresponding element values are identical, $\mathbf{V}(Ai) \;=\; \mathbf{V}(Bj)$, and that $\{\; a^{Ai},\; a^{Bj} \;\} \;\Rightarrow\; \{\; a^{Ai:Bj} \;\}$. Revisiting the SQL statement above and distinguishing the ambiguous WHERE statements and substituting appropriate scope conditionals, gives:

```
XSQL-1        Rwhere {P2=%green%}   in P      &
              Jwhere {P1:L2}        in PxL    &
              Jwhere {Q1:L2}        in QxL    &
              Jwhere {Q2:L3}        in QxL    &
              Jwhere {O1:L1}        in OxL    &
              Jwhere {S1:L3}        in SxL    &
              Jwhere {S4:N1}        in SxN
```

Since the following:

```
XSQL-2        Jwhere {Q1:L2}        in QxL    &
              Jwhere {Q2:L3}        in QxL
```

is set-theoretically equivalent to:

```
XSQL-3        Jwhere {L2:Q1, L3:Q2} in LxQ
```

Then, by substitution the following is equivalent to XSQL-1:

```
XSQL-4        Rwhere {P2=%green%}   in P      &
              Jwhere {P1:L2}        in PxL    &
              Jwhere {L2:Q1, L3:Q2} in LxQ    &
              Jwhere {O1:L1}        in OxL    &
              Jwhere {S1:L3}        in SxL    &
              Jwhere {S4:N1}        in SxN
```

Since the following are set-theoretically equivalent:

```
XSQL-5        Jwhere {P1:L2}        in PxL    &
              Jwhere {L2:Q1, L3:Q2} in LxQ

              Jwhere {Q1:P1}        in QxP    &
              Jwhere {L2:Q1, L3:Q2} in LxQ
```

Substitution of XSQL-5 into XSQL-4, gives:

```
XSQL-6        Rwhere {P2=%green%}   in P      &
              Jwhere {Q1:P1}        in QxP    &
              Jwhere {L2:Q1, L3:Q2} in LxQ    &
              Jwhere {L1:O1}        in LxO    &
              Jwhere {L3:S1}        in LxS    &
              Jwhere {N1:S4}        in NxS
```

XSQL-6 can be executed directly by an XSP engine after being coded in terms of generic XSP functions. Other set-theoretically equivalent command sequences are also possible, but with the physical data size information available, XSQL-6 will give the best performance when executed in the above order as successive I/O command sequences. If Table-Q were larger than Table-L, then XSQL-3 could give better execution performance (depending on the internal execution strategy). Both, of course, either would produce the same result set.

Since both the Generic XSP 'WHERE' commands provide for an embedded projection, performance enhancements can be affected just by reducing the physical size of a set representation by eliminating the size of its elements. Since subsequent operations are scope driven, only relevant scopes need be preserved. Set-theoretically, the answers will always be the same as long as all relevant scopes are available.

By adding the appropriate project conditions that reduce the result sets to having just those scope values that can contribute to the remainder of the query, the following executable command sequence will produce the result for query 9 for any size database. (TET for 1GB, 2GB, 4GB, and 8GB appear below.) As with SQL, '*' implies preserving all scopes in the result set.

```
XSQL-7   [1] RWHEREP( {P2=%green},    PP1,      {P1},                  RS1 )

         [2] JWHEREP( {Q1:P1},        QQ2, PP2,    *,                  RS2 )

         [3] JWHEREP( {L2:Q1, L3:Q2}, LL3, QQ3,    *,                  RS3 )

         [4] XSP_UDF( UDF3(L6,L7,Q4,L5,amount) )
         [4] JWHEREP( {L1:O1}, LL4, OO4, {L3, UDF1(O5,o_year), amount}, RS4 )

         [5] XSP_UDF( UDF2(<S4,o_year>, amount, sum_profit) )
         [5] JWHEREP( {L3:S1}, LL5, SS5, {S4, o_year, sum_profit},      RS5 )

         [6] JWHEREP( {N1:S4}, NN6, SS6, {NATION, o_year, sum_profit},  RS6 )

                      UDF1(O5,o_year)
                          with V(o_year)     = Y(O5)

                      UDF2(<S4,o_year>, amount, sum_profit)
                          with V(sum_profit) = SUM(<S4,o_year>, amount)

                      UDF3(L6,L7,Q4,L5,amount)
                          with V(amount)     = (L6*(1-L7))-(Q4*L5)
```

The above is all there is to a functionally complete XSP-executable version of Query 9. The next step is to optimize the performance of a specific instance of Query 9. Unlike indexed access optimization strategies there are no index structures to manage, not even any records to be indexed. There is not even a direct link to the internal representation of the data, since these I/O operations form a data independent interface to the actual data representations. Thus none of the familiar performance control mechanisms are available. With *scope driven* set operations performance is controlled by the choice of set operands and the order in which the operations are executed. The available optimizations are easily determined by a scope covering analysis.

SCOPE COVERING ANALYSIS: Once the functionality of an application, like Query 9, has been captured in terms of set operations, the mathematical identity of the result is known by the system.

The materialization of the result, for whatever purpose, at any time, and for any state of the internal representations, is just a matter of juggling the command sequence and plugging in appropriate set values. Both of these conditions can be determined automatically, adaptively, and instantaneously prior to the actual execution of the commands.

Scope Coverings: Scopes, in the context of Relational systems, are RDM-attributes. Just by looking at the specifications of Query 9, it can be determined that not all the scope values of all eight tables are required to execute Query 9. It will also be observed that not all the scope values required to execute Query 9 exist prior to the execution of Query 9. Thus, the order of execution of [1] to [6] may require a different ordering than presented in XSQL-7. (The actual TPC-H specification for Query 9 had [1] listed last, which is implementationally impossible.)

A simple way to determine the logical bounds on allowable execution sequences is a topological sort on *scope coverings*. A scope covering is a set of scope values that contains *at least* the scopes required by an operation. It is important to note that a scope covering is not necessarily a minimum covering. Minimal coverings always dictate the least amount of necessary data required, but may not be optimal for an aggregate collection of set operations (discussed later). As it turns out, only about 18% of the data in the six tables used for Query 9 is actually required for a scope covering of the complete query.

Topological Covering Hierarchy: The control of scope coverings can be done prior to a query or during a query. In the Query 9 example, the final scope covering is that of the final result. Which is $\{\ NATION,\ o_year,\ sum_(profit)\ \}$. Thus, the generation of both $\{o_year\}$ and $\{sum_(profit)\}$ are required prior to the execution of [6]. The obvious choice for the first scope partition is support for [1].

```
[1] RWHEREP( {P2=%green}, PP1,  {P1}, RS1 )
```

Here, scope set $\{\mathsf{P1},\ \mathsf{P2}\}$ is all that is required to generate RS1. Therefore, as long as PP1 is chosen such that $\{\mathsf{P1},\ \mathsf{P2}\} \subseteq \mathcal{S}c(\mathsf{PP1})$, [1] will produce RS1, such that $\mathcal{S}c(\mathsf{RS1}) = \{\mathsf{P1}\}$. The following excerpt from appendix-A is the actual executable version of Query 9, specifying the order in which the commands are executed and the specific set substitutions used.

```
[1] RWHEREP( {P2=%green},    P, {P1},    RS1 )                      RS1: 93% reduction

[2] JWHEREP( {Q1:P1},        Q,  RS1, *, RS2 )                      RS2: 20% reduction

[3] JWHEREP( {L2:Q1, L3:Q2}, L,  RS2, *, RS3 )                      RS3: 17% reduction

[4] XSP_UDF( UDF3(L6,L7,Q4,L5,amount) )
[4] JWHEREP( {L1:O1}, RS3, O, {L3, UDF1(O5,o_year), amount}, RS4 ) RS4: 68% reduction

[5] XSP_UDF( UDF2(<S4,o_year>, amount, sum_profit) )
[5] JWHEREP( {L3:S1}, RS4, S, {S4, o_year, sum_profit},      RS5 ) RS5: 42% reduction

[6] JWHEREP( {N1:S4}, N, RS5, {NATION, o_year, sum_profit},  RS6 ) RS6: 32% reduction
```

Notice that [1] has P substituted for PP1. This P is a minimal covering set for Query 9 and was generated by the command:

```
XSET( XP, { <P1, I,  4>,
            <P2, A, 55>} )
DT_LOAD( {|}, XP, P, .\PART.TBL )
```

Any set, XP, in the above command, that is a subset of $\{\mathsf{P1}, .., \mathsf{P9}\}$ and contained $\{\mathsf{P1}, \mathsf{P2}\}$ would work for PP1 in [1]. By choosing XP as above, the set P accesses only 36% of the data in the PART.TBL data. The output set, RS1, is even 93% smaller than this already data-reduced input set. These lopsided read/write ratios are listed in appendix-A.

As can be seen from [2] above, RS1 was substituted for PP2. It is the minimal covering set required for [2], though any larger covering would also have worked. The set Q substituted for QQ2 was generated by:

```
XSET( XQ, { <Q1, I, 4>,
            <Q2, I, 4>,
            <Q4, R, 8>} )
DT_LOAD( {|}, XQ, Q, .\PARTSUPP.TBL )
```

The choice of XQ in the above is a minimal scope covering for Query 9 and produces set Q which accesses only 7% of the total data in the PARTSUPP.TBL data. In [2] the output set, RS2, is 20% smaller than the combined input data read. RS2 is now substituted in [3] for QQ3. The set L substituted for LL3 was generated by:

```
XSET( XL, { <L1, I, 4>,
            <L2, I, 4>,
            <L3, I, 4>,
            <L4, I, 1>,
            <L5, R, 4>,
            <L6, R, 8>,
            <L7, R, 4>} )
DT_LOAD( {|}, XL, L, .\LINEITEM.TBL )
```

Again, the choice of XL in the above is a minimal scope covering for Query 9 and produces set L which accesses only 23% of the total data in the LINEITEM.TBL data. In [3] the output set, RS3, is 17% smaller than the combined input data read. RS3 is now substituted in [4] for LL4. The set O substituted for OO4 was generated by:

```
XSET( XO, { <O1, I, 4>,
            <O5, Z, 4>} )
DT_LOAD( {|}, XO, O, .\ORDERS.TBL )
```

Again, the choice of XO in the above is a minimal scope covering for Query 9 and produces set O which accesses only 8% of the total data in the ORDERS.TBL data. [4] is different than the previous XSP calls since it invokes a UDF that is called during the execution of the second part of [4]. Notice that the scope amount is both generated and used during the execution of [4]. Also during the execution of [4], the scope o_year and its respective values are extracted from scope O5 and its respective values. Thus the scope of the output set RS4 the set $\{\mathsf{L3}, \mathsf{o_year}, \mathsf{amount}\}$. RS4 is now substituted in [5] for LL5 and notice that RS4 does contain the required scope value L3. The set O substituted for OO4 was generated by:

```
XSET( XS, { <S1, I, 4>,
            <S4, I, 1>} )
DT_LOAD( {|}, XS, S, .\SUPPLIER.TBL )
```

Again, the choice of XS in the above is a minimal scope covering for Query 9 and produces set S which accesses only 3% of the total data in the SUPPLIER.TBL data. [5], like [4], also invokes a UDF that groups subsets of data (aggregates) and tabulates over a specific scope value. In this case sum_profit is generated from amount. The output set written to storage, RS5, is 42% smaller than the combined input data read. RS5 is now passed on to the last XSP operation that just replaces the NATION_ID value with the NATION_NAME value and writes the final result, RS6, to storage. This completes the execution of Query 9. The total elapsed times for 1GB, 2GB, 4GB, and 8GB databases are given in the next section.

An XSP Implementation: Appendix-A contains the complete calling sequence for executing Query 9 using the generic XSP operations already described. Any XSP-Engine supporting these commands will produce the same result sets. Following are results from using a specific XSP-Engine, iXSP, implemented by the author. (the iXSP implementation is by no means optimized for performance. It uses random access I/O, double buffers all I/O, uses a small 64Kb transfer buffer, does no

overlapping I/O, does not take advantage of high performance commercial sort packages, and uses no caching intelligence. It was implemented just as a proof of concept, yet its performance can give a feel for what set processing performance could be if implemented with performance as the objective.) Four separate tasks were run. Each started with the eight text-delimited tables generated DBGEN for 1GB, 2GB, 4GB, and 8GB. The available disk storage for usage by iXSP was just 13GB in all four cases. (It is worth noting that this would not likely be enough space for commercial indexed-record based systems.) The CPU was 2.8GHz and the available RAM was restricted to 64MB. The disk drive was a single Fujitsu SCSI 320 on a 32b PCI bus.

```
PLATFORM:   CPU 2.8GHz Pentium, Available RAM = 64MB
--------    SCSI 320 (75MB/s) Free Disk Space = 13GB

              1GB           2GB           4GB           8GB
              ---           ---           ---           ---
TOTAL LOAD:   57.33s    :  117.35s    :  237.87s    :  510.77s
   XSQL Q9:    5.47s    :   12.07s    :   24.57s    :   47.36s

  iXSP TET:   64.12s    :  130.14s    :  263.08s    :  558.13s
               1.07m         2.17m         4.38m         9.30m

             Figure 1: TOTAL ELAPSED TIME (TET) COMPARISON
             ----------------------------------------------
```

The above table shows the total load time, which includes all processing from the reading of the DBGEN tables to the beginning of the Q9 execution. This would include the creating of any index structures, if any index structures were to be created. The second line is the total time taken for executing XSQL-7 above. The complete executable script is contained in appendix-A. The third line is the total elapsed time that a business would actually experience in trying to answer Q9. The TET is a very practical measure of Rapid Information Access, RIA.

Improved Performance: The above XSQL Q9 times can probably be reduced by a factor of 3-4 given an implementation that was more oriented toward performance instead of just demonstrating what can be done without using index structures. Further reductions, for very large tables, can be achieved with multiple CPUs, multiple disks, and a parallel processing architecture.

Performance is not just enhanced by the improved I/O information density naturally allowed by set-processing, but also by relieving the internal architecture of unnecessarily consuming a very large amount of precious system resources. The system internal architecture only has to be concerned with the set-theoretic properties of an application. Management of set ownership, deleting old sets from a user's universe, adding new sets to a user's universe, and securing authorized access are the only management concerns. Improved performance by avoidance is achieved by what does not have to be managed by the internal architecture: no I/O caching concerns, no indexes to build, no indexes to access, no indexes to update, in fact there are not even any records visible to the internal architecture to index, even if there was a desire to. Improved performance by near optimality is achievable by the vendor of the set-processing implementation used by the internal architecture. The opportunity here is in the degree of information density and choice of transfers speed across the PIO and NIO interfaces.

Potential Applications: The intrinsic data independence that set-processing imbues all I/O interfaces can be utilized, not just at the PIO interface to bypass the system I/O, but also at the NIO to support high performance information exchange between data on disparate, but informationally homogeneous, platforms. XSP operations could be embedded in operating systems, implanted in chips, integrated into storage environments, and, in short, used anywhere that the data independent representation and access of data with unrestricted manipulation functionality is desirable.

Set-processing can be characterized as allowing only set-membership to be transferred between environments. Appendix-B refers to modeling data independent enterprise/machine mappings by XST as Gamma Interface Architectures, GIA. In a GIA there is no concept of a record at an I/O

interface, only set names. It would seem that general systems based on GIAs could be highly reliable, interoperable, high performance, low maintenance, scalable, functionally robust, in short, incarnations of whatever a mathematically sound foundation could support.

CONCLUSION: The pretext of this paper was that set-processing I/O strategies provide better performance than do traditional indexed-record access strategies. It has been argued that set-processing manipulates and preserves data content while indexed-record strategies manipulate and preserve data records. While true, and hopefully argued convincingly, it was also a vehicle for persuading the reader to think about manipulating data in terms of *data content* instead of in terms of *data records*.

The importance of manipulating data in terms of data content instead of data records was originally introduced by E. F. Codd in 1970. Though the practical advantage of manipulating data in terms of data content for external architectures has been proven by commercial RDBMSs, manipulating data in terms of data content has not yet permeated the thinking of developers of internal architectures.

Historically, preserving data record integrity was the only assurance for preserving data content integrity. Codd showed, with the relational calculus, that for certain forms of data records the data content could be captured by its mathematical identity. With the mathematical identity of data content available, operations could be defined directly in terms of data content independently of the specific representation of the data. This *data independence* is a characteristic property of operation-centric architectures based on mathematical identities.

The separation of functional capabilities from performance management through data independence is the essential difference between operation-centric architectures and structure-dependent architectures. There are very few systems left today that do not have operation-centric external architectures. There are fewer still that do have operation-centric internal architectures.

Data content integrity is still a concern for internal architectures. The mathematical identity for data content developed by Codd does not extend to the variety of physical data representations required by internal architectures. By extending the membership condition of set theory, the mathematical identity of data content of any and all physical data representations can be captured. By using extended set theory to capture the mathematical identity of data content of physical data, the integrity of data content is preserved and set-processing emerges as a viable candidate for operation-centric internal architectures.

Though the resource savings and performance benefits of set-processing can be easily verified by demonstration, conceptual appreciation requires a skewed intellectual slant, viewing system behavior in terms of operating on data content not in terms relocating data records. This appreciation was difficult to comprehend in 1970, when Codd introduced an operation-centric external architecture, and it is no easier today when trying to appreciate the performance benefits of operation-centric internal architectures. If set-processing achieves for internal architectures what relation-processing did for external architectures, this may all be obvious in thirty years.

References

[Bo75] Boyce, R. F.; Chamberlin, D. D.; King, F. W.; Hammer, M. M.: *Specifying Queries as Relational Expressions: The SQUARE Data Sublanguage*, Comm. of ACM, Vol. 18, No. 11, 1975.

[Ch68] Childs, D. L.: *Feasibility of a Set-Theoretic Data Structure: A General Structure Based on a Reconstituted Definition of Relation*, Proc. IFIP Congress 1968

[Ch77] Childs, D. L.: *Extended Set Theory: A General Model for Very Large, Distributed, Backend Information Systems*, Third International Conference On Very Large Databases, Tokyo, Japan, 1977

[Ch86] Childs, D. L.: *A Mathematical Foundation For Systems Development*, NATO ASI Series, Vol F24, Database Machines, Edited by A. K. Sood and A. H. Qureshi, Springer-Verlag, 1986

[Ch00] Childs, D. L.: *XSP Technology for XML Systems Design & Development*, IIS, 2000

[Ch05] Childs, D. L.: *Axiomatic Extended Set Theory*, IIS, 1968-2005

[Co70] Codd, E. F.: *A Relational Model of Data for Large Shared Data Banks*, CACM 13, No. 6 (June) 1970

[Da95] Date, C. J.: *An Introduction to Database Systems*, 6th. edition, Addison-Wesley, 1995

[DD97] Date, C. J.; Darwin Hugh: *A Guide to the SQL Standard*, 4th. edition, Addison-Wesley, 1997

[Sk57] Skolem, Thoralf: *Two Remarks on Set Theory,* Mathematica Scandinavica 5 (1957), p.43-46.

[Su60] Suppes, Patrick: *Axiomatic Set Theory*, Van Nostrand, 1960, p.20 & 141.

[TPC] TPC-H/R Benchmark Description, 2000

A TPC-H QUERY-9

Below is stylized version of Query 9 from the TPC-H benchmark. The long user-friendly names have been replaced with algebraically equivalent scope names. This is a functionally equivalent expression of Query-9. Its executable Generic SQL/XSP counterpart follows.

```
-------------------------------------------------------------------------------
EXAMPLE: TPC-H QUERY-9
-------------------------------------------------------------------------------
TABLES: { Scope_Name(Byte_Size) }  * = Key                [Total_Byte_Size]
*-----------------------------------------------------------------------------
     L: { *L1(4),  L2(4), L3(4), *L4(1), L5(4), L6(8), L7(4) }  [29]
     O: { *O1(4),  O5(4)  }                                     [ 8]
     Q: { *Q1(4), *Q2(4), Q4(8) }                               [16]
     P: { *P1(4),  P2(55) }                                     [59]
     S: { *S1(4),  S4(1)  }                                     [ 5]
     N: { *N1(1),  NATION(25) }                                 [26]
*-----------------------------------------------------------------------------
   { SELECT < NATION, o_year, sum_profit >

                     UDF2(<S4,o_year>, amount, sum_profit)
                         with V(sum_profit) = SUM(<S4,o_year>, amount)

       FROM { SELECT < NATION, o_year, amount >

                     UDF1(O5,o_year)
                         with V(o_year)     = Y(O5).

                     UDF3(L6,L7,Q4,L5,amount)
                         with V(amount)     = (L6*(1-L7))-(Q4*L5).

                FROM P, Q, L, O, S, N

               WHERE [P_P2] = LIKE '%[color]%'  &
                     [P_P1] = [L_L2]            &
                     [Q_Q1] = [L_L2]            &
                     [Q_Q2] = [L_L3]            &
                     [O_O1] = [L_L1]            &
                     [S_S1] = [L_L3]            &
                     [S_S4] = [N_N1]
            }
   }
         NOTATION:  [A]=[B]  -> Value at scope A equals value at scope B.
-----------------------------------------------------------------------------

-----------------------------------------------------------------------------
XSP ENGINE EXECUTION OF Q9
-----------------------------------------------------------------------------
TPC-H DBGEN TABLES - 1 GB
-----------------------------------------------------------------------------
 L-Table   765,884,367       .\LINEITEM.TBL  SCOPES: { L1,..,L16 }  [126]
 O-Table   173,416,292       .\ORDERS  .TBL  SCOPES: { O1,..,O9  }  [131]
 Q-Table   119,694,335       .\PARTSUPP.TBL  SCOPES: { Q1,..,Q5  }  [219]
 P-Table    24,180,198       .\PART    .TBL  SCOPES: { P1,..,P9  }  [164]
 S-Table     1,419,633       .\SUPPLIER.TBL  SCOPES: { S1,..,S7  }  [220]
 N-Table         2,214       .\NATION  .TBL  SCOPES: { N1,..,N4  }  [179]
-----------------------------------------------------------------------------
GIVEN: UDF1, UDF2, & UDF3
-------------------------

-------------------------------------------------
GENERIC XSP: LOAD, OPTIMIZATION, & EXECUTION OF Q9
```

```
-------------------------------------------------
-------------------------
BEGIN DELIMITED TEXT LOAD
-------------------------
---------------
LINEITEM.TBL: L with SCOPES: { L1,L2,L3,L4,L5,L6,L7 } [29]     23% of L-Table
   XSET( XL, { <L1, I, 4>,
               <L2, I, 4>,
               <L3, I, 4>,
               <L4, I, 1>,
               <L5, R, 4>,
               <L6, R, 8>,
               <L7, R, 4>} )
   DT_LOAD( {|}, XL, L, .\LINEITEM.TBL )

---------------
ORDERS  .TBL: O with SCOPES: { O1,O5 }                [ 8]      6% of O-Table
   XSET( XO, { <O1, I, 4>,
               <O5, Z, 4>} )
   DT_LOAD( {|}, XO, O, .\ORDERS.TBL )

---------------
PARTSUPP.TBL: Q   SCOPES: { Q1,Q2,Q4 }                [16]      7% of Q-Table
   XSET( XQ, { <Q1, I, 4>,
               <Q2, I, 4>,
               <Q4, R, 8>} )
   DT_LOAD( {|}, XQ, Q, .\PARTSUPP.TBL )

---------------
PART    .TBL: P  with SCOPES: { P1,P2 }               [59]     36% of P-Table
   XSET( XP, { <P1, I,  4>,
               <P2, A, 55>} )
   DT_LOAD( {|}, XP, P, .\PART.TBL )

---------------
SUPPLIER.TBL: S  with SCOPES: { S1,S4 }               [ 5]      3% of S-Table
   XSET( XS, { <S1, I, 4>,
               <S4, I, 1>} )
   DT_LOAD( {|}, XS, S, .\SUPPLIER.TBL )

---------------
NATION  .TBL: N  with SCOPES: { N1,N2 }               [26]     15% of N-Table
   XSET( XN, { <N1, I,  1>,
               <N2, A, 25>} )
   DT_LOAD( {|}, XN, N, .\NATION.TBL )

------------------------
END DELIMITED TEXT LOAD   TOTAL DATA REQUIRED = 203,684,446    18.78% TOTAL DATA
------------------------                                       -----------------
---------------------
BEGIN EXECUTION OF Q9
---------------------
   RWHEREP( {P2=%green},    P, {P1},    RS1 )                    RS1: 93% reduction

   JWHEREP( {Q1:P1},        Q,  RS1, *, RS2 )                    RS2: 20% reduction

   JWHEREP( {L2:Q1, L3:Q2}, L,  RS2, *, RS3 )                    RS3: 17% reduction

   XSP_UDF( UDF3(L6,L7,Q4,L5,amount) )
   JWHEREP( {L1:O1}, RS3, O, {L3, UDF1(O5,o_year), amount}, RS4 ) RS4: 68% reduction

   XSP_UDF( UDF2(<S4,o_year>, amount, sum_profit) )
   JWHEREP( {L3:S1}, RS4, S, {S4, o_year, sum_profit},      RS5 ) RS5: 42% reduction

   JWHEREP( {N1:S4}, N, RS5, {NATION, o_year, sum_profit},  RS6 ) RS6: 32% reduction

-------------------
END EXECUTION OF Q9
-------------------
*-------------------------------------------------------------------------------
OPERATION TIMES
```

Set	1GB	2GB	4GB	8GB	Scope Size
RS1	0.06 sec	0.11 sec	0.21 sec	0.42 sec	4 bytes
RS2	0.07 sec	0.10 sec	0.20 sec	1.56 sec	16 bytes
RS3	3.52 sec	7.16 sec	14.39 sec	29.89 sec	37 bytes
RS4	0.96 sec	2.15 sec	4.19 sec	8.71 sec	14 bytes
RS5	0.85 sec	1.62 sec	3.24 sec	6.77 sec	11 bytes
RS6	0.01 sec	0.01 sec	0.01 sec	0.01 sec	25 bytes
TOTAL	5.47 sec	11.15 sec	22.24 sec	47.36 sec	
	0.09 min	0.19 min	0.37 min	0.79 min	

*----------

LOPSIDED I/O [1GB]

Set	BYTES IN	BYTES OUT	WRITTEN/READ
RS1	11,800,064	46,612	0.40%
RS2	12,800,064 46,612*	744,832	5.80%
RS3	174,035,299 744,832*	7,324,024	4.19%
RS4	12,000,064 7,324,024*	4,882,704	25.27%
RS5	50,064 4,882,704*	1,989	0.04%
RS6	650 1,989*	4,439	168.21%
TOTAL	223,686,366	13,004,600	5.81%

* writing then reading unnecessary with RAM restrictions lifted and intelligent caching employed.

*----------

```
RESULTS FOR 1GB
----------------------
  TOTAL Query   Time                [   5.47 sec,   0.09 min ]
  TOTAL Load    Time                [  58.65 sec,   0.98 min ]
  TOTAL Elapsed Time                [  64.12 sec,   1.07 min ]

RESULTS FOR 2GB
----------------------
  TOTAL Query   Time                [  11.15 sec,   0.19 min.]
  TOTAL Load    Time                [ 116.93 sec,   1.95 min ]
  TOTAL Elapsed Time                [ 128.08 sec,   2.13 min ]

RESULTS FOR 4GB
----------------------
  TOTAL Query   Time                [  22.24 sec,   0.37 min.]
  TOTAL Load    Time                [ 240.34 sec,   4.00 min ]
  TOTAL Elapsed Time                [ 262.58 sec,   4.38 min ]

RESULTS FOR 8GB
----------------------
  TOTAL Query   Time                [  47.36 sec,   0.79 min.]
  TOTAL Load    Time                [ 510.77 sec,   8.51 min ]
  TOTAL Elapsed Time                [ 558.13 sec,   9.30 min ]
```

*----------

B MORE ON MATHEMATICAL IDENTITY

The preceding paper has stressed the importance of mathematical control for managing conceptual complexity. Adding two single digit integers is both a human-friendly and a mathematically-friendly task. Adding two 9,149 place integers is no longer a human-friendly task, but it is still a mathematically-friendly task. By capturing the mathematical identity of *integers* and by *abstracting* the task of adding, the task of adding any size integers together can be assigned to a *black box*. Feed it any two integers, kick it, and out comes a verifiable integer result. The black box does not have to be previously tested to assure that every result will in fact be an integer. The mathematics guarantees it. This abstraction of process is exactly what set processing provides at the I/O interface between any two set-theoretically definable environments. To be set-theoretically definable just means having mathematical identities for the inhabitants of the environments. One inhabitant of particular interest is the *TABLE*.

MATHEMATICAL IDENTITY: We are so familiar with the equals symbol in expressions like '$x = \sqrt{y}$' and '$7 - 1 = 4 + 2$' that we seldom recall that '=' just means that the expression on the left has the same mathematical identity as the expression on the right. The use of the '=' is so ubiquitous that we seldom think to challenge its usage. Even if one did think to challenge an expression like '$\mathbf{T}_i = \mathbf{T}_j$', or even '$\mathbf{T}_i \neq \mathbf{T}_j$', how would one do it?

For example, assume that '$\mathbf{T}_i$' and '$\mathbf{T}_j$' represent 'tables' as defined by the Relational Data Model, RDM. How do we verify that these two tables are either equal or not equal? Since the RDM is known to be mathematically well defined, it should not be too hard to compare the mathematical identities of any two tables, and resolve the issue. Consider the following RDM tables, does '$\mathbf{T}_1 = \mathbf{T}_3$'? Is '$\mathbf{T}_2 \neq \mathbf{T}_4$' false?

$$\mathbf{T}_1 = \begin{array}{|c|c|c|} \hline A & B & C \\ \hline a & b & c \\ \hline x & y & z \\ \hline \end{array} \qquad \mathbf{T}_2 = \begin{array}{|c|c|c|} \hline A & B & C \\ \hline x & y & z \\ \hline a & b & c \\ \hline \end{array} \qquad \mathbf{T}_3 = \begin{array}{|c|c|c|} \hline A & C & B \\ \hline a & c & b \\ \hline x & z & y \\ \hline \end{array} \qquad \mathbf{T}_4 = \begin{array}{|c|c|c|} \hline A & C & B \\ \hline x & z & y \\ \hline a & c & b \\ \hline \end{array}$$

The answers depend on the mathematical identities of the '$\mathbf{T}_i$'. Though it is clear by visual inspection what the answers are in this simple case. If the same question were asked with tables having 3,000,147 rows and 6,147 columns, visual inspection might be somewhat challenging, but mathematically the problem is no more complex than before.

RDM-TABLES & RDM-RELATION: Capturing the mathematical identities of RDM-tables is not as straight forward as it might at first seem. Most people who are familiar with the Relational Data Model know that it has a rigorous mathematical foundation. They also know that RDM-tables are just a representational convenience for expressing 'RDM-relations'. However, most of these same people do not know that the mathematical foundation of the RDM is based on Classical set theory, CST, which expresses RDM-relations as an equivalence class of 'CST-relations'.

What this means is that the unique mathematical identity, or *set-theoretic membership condition*, of a RDM-relation (and hence, a RDM-table) can not be expressed in CST. It must be clearly understood that though RDM-relations are conceptually considered to be mathematically well-defined as being individually unique, they are actually defined only as a collection, or 'equivalence class' which does have a unique mathematical identity, or set-theoretic membership condition.

This concept of equivalence is similar to the box of matches, the jar of bees, and the case of scotch. Each is analogous to an 'equivalence class of CST-relations', with the individual matches, the individual bees, and the individual bottles being analogous to individual CST-relations. All the individuals in each specific 'equivalence class' 'behave equivalently' and therefore have the same 'behavioral identity', even though the individual items (matches, bees, bottles, and relations) do not have the same 'individual identity'.

Mathematicians allude to this grouping of individual items (with the same 'behavioral identity') into a collection, or 'equivalence class', as an 'isomorphic equivalence'. Meaning, roughly, that since the individual items all behave in the same way, it does not matter which item is picked to do the job. What is left unsaid, is that it is often not possible to distinguish one individual item from the other, even if it were required. For example, the set $\{ < a, b, c >, \ < a, < b, c >>, \ << a, b >, c > \}$ appears to have a cardinality of 3, but it is actually just 2.

It is exactly this ability, of being able to distinguish one individual item form another, that is required to support the mathematical identity of individual items, like RDM-tables. Unfortunately, set theory is of no help in achieving this end. However, 'extensions' to classical set theory can be provided that do allow these individual items to have a unique mathematical identity.

XSN: EXTENDED SET NOTATION: This paper is not intended to be an exercise in set theory, classical or extended. It is intended to familiarize the reader with a method for exerting mathematical control over the storage, access, and manipulation of data. This method depends on capturing the data in a mathematical form, then any results obtained from manipulating the mathematical form of the data can be obtained by manipulating the actual data in the same way.

The first step is to capture the mathematical identity of the data. Since the mathematical identity of a set is equivalent to its set theoretic membership condition, all that is required to capture the mathematical identity of the data is to express the data as a well defined set. The development of Extended set theory was motivated and funded to perform exactly this task. The mathematical identity of data can be richly captured using extended set notation, XSN, as the following sections will illustrate. First, the notation will be introduced by examples, then explained in terms of the examples. Consider the following representations for RDM-tables:

$$\mathbf{R} = \begin{array}{|c|c|c|} \hline \multicolumn{3}{c}{A\ B\ C} \\ \hline a & b & c \\ \hline x & y & z \\ \hline \end{array} \qquad \mathbf{S} = \begin{array}{|c|c|c|} \hline \multicolumn{3}{c}{C\ A\ B} \\ \hline t & w & v \\ \hline c & a & b \\ \hline \end{array} \qquad \mathbf{Q} = \begin{array}{|c|c|c|} \hline \multicolumn{3}{c}{B\ C\ A} \\ \hline b & c & a \\ \hline \end{array} \qquad \mathbf{T} = \begin{array}{|c|c|c|} \hline \multicolumn{3}{c}{C\ B\ A} \\ \hline z & y & x \\ \hline c & b & a \\ \hline \end{array} \qquad \mathbf{P} = \begin{array}{|c|c|c|c|} \hline \multicolumn{4}{c}{D\ B\ A\ C} \\ \hline h & v & w & t \\ \hline d & b & a & c \\ \hline w & x & y & z \\ \hline \end{array}$$

Anyone familiar with the Relational Data Model will recognize that the above are RDM-tables that represent RDM-relations. It should also be clear that **R** and **T** are identical, that the intersection of **R** and **S** is equal to **Q**, that **R** is a sub-relation of **P**, and that **R**, **S**, **Q**, and **T** are domain compatible (*i.e.* having the same domains). All that remains to substantiate these observations, both structural and procedural, is to capture the mathematical identities in terms of extended set notation:

$$\mathbf{R} = \Big\{\{a^A, b^B, c^C\}, \{x^A, y^B, z^C\}\Big\}, \qquad \mathbf{S} = \Big\{\{t^C, w^A, v^B\}, \{c^C, a^A, b^B\}\Big\}, \qquad \mathbf{Q} = \Big\{\{b^B, c^C, a^A\}\Big\}$$

$$\mathbf{T} = \Big\{\{z^C, y^B, x^A\}, \{c^C, b^B, a^A\}\Big\}, \qquad \mathbf{P} = \Big\{\{h^D, v^B, w^A, t^C\}, \{d^D, b^B, a^A, c^C\}, \{w^D, y^B, x^A, z^C\}\Big\}$$

With these extended set definitions and the extended set operations defined for 'symmetric difference', 'intersection', and 'domain class', the following can be verified:

$\mathbf{R} \triangle \mathbf{T} = \emptyset$, therefore **R** and **T** are identical,

$\mathbf{R} \cap \mathbf{S} = \mathbf{Q}$, therefore **R** and **S** share **Q** as a sub-relation,

$\mathbf{R} \cap \mathbf{P} = \mathbf{R}$, therefore **R** is a sub-relation of **P**,

In the following example, '$\mathcal{S}(\mathbf{R})$' represents the *Scope*, or *Domain Class*, of '**R**'.

$\mathcal{S}(\mathbf{R}) = \mathcal{S}(\mathbf{S}) = \mathcal{S}(\mathbf{Q}) = \mathcal{S}(\mathbf{T}) = \{A, B, C\}$, therefore Domain Compatibility.

A number of clarifications might need to be made about this last section. The most obvious would be to provide more intuition behind the use of XSN. Less obvious would be to determine what, if anything, about the RDM was actually modeled using XSN.

LIMITATIONS OF THE RDM: Because of their universal familiarity, the RDM-tables and RDM-relations have been used to introduce the ideas and mechanisms for distinguishing 'data content' from 'data representation'. However, the use of RDM-tables and RDM-relations does not provide a complete answer to generalized data independence at the E/M interface and they provide no assistance at the I/O and inter-system interfaces. These limitations need to be delineated before presenting the requirements and mechanisms for providing total and complete 'data independence' at H/U interface, the I/O interface, and between systems.

As has been noted, the strength of the Relational Data Model is its use of mathematical identity for connecting the user environment with the host environment. However, there are two severe limitations with the Relational Data Model, both due to its classical set theoretic foundations and both correctable using extended set-theoretic foundations.

The first limitation is the lack of a rigorous *membership condition* for separately expressing RDM-tables and RDM-relations uniquely. The second limitation is also due to a weak *membership condition* that precludes the representation of general structures like: 'complex relations', 'un-normalized relations', 'objects', 'blobs', 'text data', 'SQML data', 'XML data', 'B-trees', 'linked lists', 'voice data', 'telemetry data', *etc.*, all of which diminishes the utility of the Relational Data Model for complex information systems. Both of these limitations will be addressed in subsequent sections.

MATHEMATICAL IDENTITY OF TABLES & RELATIONS: In the previous example, it was asserted that **R** and **T** were identical relations, but that they were different RDM-tables. This distinction is not formally supported by the Relational Data Model which uses mathematical descriptions based on CST notation. The use of extended set notation can resolve this distinction. For example, **T** can be expressed by the following twelve(12) distinct RDM-table representations:

$$\mathbf{T}_1 = \begin{array}{|c|c|c|}\hline A & B & C \\ \hline a & b & c \\ \hline x & y & z \\ \hline \end{array} \quad \mathbf{T}_2 = \begin{array}{|c|c|c|}\hline A & B & C \\ \hline x & y & z \\ \hline a & b & c \\ \hline \end{array} \quad \mathbf{T}_3 = \begin{array}{|c|c|c|}\hline A & C & B \\ \hline a & c & b \\ \hline x & z & y \\ \hline \end{array} \quad \mathbf{T}_4 = \begin{array}{|c|c|c|}\hline A & C & B \\ \hline x & z & y \\ \hline a & c & b \\ \hline \end{array} \quad \mathbf{T}_5 = \begin{array}{|c|c|c|}\hline B & A & C \\ \hline b & a & c \\ \hline y & x & z \\ \hline \end{array} \quad \mathbf{T}_6 = \begin{array}{|c|c|c|}\hline B & A & C \\ \hline y & x & z \\ \hline b & a & c \\ \hline \end{array}$$

$$\mathbf{T}_7 = \begin{array}{|c|c|c|}\hline B & C & A \\ \hline b & c & a \\ \hline y & z & x \\ \hline \end{array} \quad \mathbf{T}_8 = \begin{array}{|c|c|c|}\hline B & C & A \\ \hline y & z & x \\ \hline b & c & a \\ \hline \end{array} \quad \mathbf{T}_9 = \begin{array}{|c|c|c|}\hline C & A & B \\ \hline c & a & b \\ \hline z & x & y \\ \hline \end{array} \quad \mathbf{T}_{10} = \begin{array}{|c|c|c|}\hline C & A & B \\ \hline z & x & y \\ \hline c & a & b \\ \hline \end{array} \quad \mathbf{T}_{11} = \begin{array}{|c|c|c|}\hline C & B & A \\ \hline c & b & a \\ \hline z & y & x \\ \hline \end{array} \quad \mathbf{T}_{12} = \begin{array}{|c|c|c|}\hline C & B & A \\ \hline z & y & x \\ \hline c & b & a \\ \hline \end{array}$$

All twelve are distinct RDM-tables that represent the same unique RDM-relation. With the mathematical identity of the $\mathbf{T}_i$ expressed in the form of $\mathbf{T}_t$ in a previous section, then the underlying RDM-relation can be expressed formally by: for all 'i & j' ranging from 1 to 12:

$$\left(\mathbf{Đ}_M(\mathbf{T}_i)^{*\emptyset}\right)^{\prec \mathbf{Đ}_S(\mathbf{T}_i)\succ} = \left(\mathbf{Đ}_M(\mathbf{T}_j)^{*\emptyset}\right)^{\prec \mathbf{Đ}_S(\mathbf{T}_j)\succ} = \Big\{\{z^C, y^B, x^A\},\ \{a^A, c^C, b^B\}\Big\}.$$

This formally asserts that distinct, but equivalent RDM-tables can be mathematically transformed into their underlying unique RDM-relation. This is an example of using extended set notation to capture both the representation and content of data. More examples follow.

LIMITED GENERALITY OF THE RDM: Though the Relational Data Model provides data independence across the U/H interface, it is very restrictive on the complexity of structures that can be formally represented.

Originally RDM-relations were defined as homogeneous 'tabular-collections' that did not allow multiple entries, null entries, nor nestings of relations. Later works indicated a need for extending the original definition of the RDM-relation to include more complexity. The following are examples of 'extended relations', or Xrels, containing multiple entries, null entries, and other Xrels.

Xrel1 =

H	C	W
a	b, c	d
x	y	w, z

Xrel2 =

Q	E	F
c	b	A C B: x z y
z, w	y	x

Xrel3 =

A	E	M	P	T
	g		q	
b	i	n	r	
c		o	s	
d	k		t	v
	l			

$$\textbf{Xrel1} = \Big\{\{a^H, b^C, c^C, d^W\}, \{x^H, y^C, w^W, z^W\}\Big\},$$

$$\textbf{Xrel2} = \Big\{\{c^Q, b^E, \{x^A, z^C, y^B\}^F\}, \{z^Q, w^Q, y^E, x^F\}\Big\},$$

$$\textbf{Xrel3} = \Big\{\{g^E, q^P\}, \{b^A, i^E, n^M, r^P\}, \{c^A, o^M, s^P\}, \{d^A, k^E, t^P, v^T\}, \{l^E\}\Big\}.$$

Xrel4 =

D	B	A	C
h	Q E F: c b A C B: x z y; z, w y x	f	g
3	b	a	c
1 2 3: t w v; c a b	y	x	$< a, b, c, d, e >$

Xrel5 =

A B C	H U L	W P M
7 2 5	s 8 k	q h j
b	f	j
c	g	k
d	h	l

$$\textbf{Xrel4} = \Big\{\Big\{h^D, \Big\{\{c^Q, b^E, \{x^A, z^C, y^B\}^F\}, \{z^Q, w^Q, y^E, x^F\}\Big\}^B, f^A, g^C\Big\},$$
$$\Big\{3^D, b^B, a^A, c^C\Big\}, \quad \Big\{\Big\{\{t^1, w^2, v^3\}, \{c^1, a^2, b^3\}\Big\}^D, y^B, x^A, < a, b, c, d, e >^C\Big\}\Big\},$$

$$\textbf{Xrel5} = \Big\{\Big\{b^{\{\{7^A, 2^B, 5^C\}\}}, f^{\{\{s^H, 8^U, k^L\}\}}, j^{\{\{q^W, h^P, j^M\}\}}\Big\},$$
$$\Big\{c^{\{\{7^A, 2^B, 5^C\}\}}, g^{\{\{s^H, 8^U, k^L\}\}}, k^{\{\{q^W, h^P, j^M\}\}}\Big\}, \Big\{d^{\{\{7^A, 2^B, 5^C\}\}}, h^{\{\{s^H, 8^U, k^L\}\}}, l^{\{\{q^W, h^P, j^M\}\}}\Big\}\Big\}.$$

The above examples are not intended to be 'meaningful' in the sense that they necessarily model some real world situation, thought they might. These examples are only intended to show the modeling generality of extended set notation for arbitrarily complex structures that could possibly arise in the development of Information Access Systems.

C XML-STRUCTURES AS EXTENDED SETS

Given the following XML-Structures (ignoring the fact that they may be semantically vacuous in any known real world), can the structural relationship of the 'notational references' be preserved by a formal modeling as was done in the above RDM example?

```
<P>
  <p>
    <n>Alan</n>
    <a>42</a>
    <e>agb@abc.com</e>
  </p>
  <p>
    <n>Mary</n>
    <a>29</a>
    <e>mky@abc.com</e>
  </p>
</P>
```

This could be represented by:

$$\mathbf{P} = \Big\{ \ \{\{Alan^{<1>}\}^{<1,n>},\ \{42^{<1>}\}^{<2,a>},\ \{agb@abc.com^{<1>}\}^{<3,e>}\}^{<1,p>},$$
$$\{\{Mary^{<1>}\}^{<1,n>},\ \{29^{<1>}\}^{<2,a>},\ \{mky@abc.com^{<1>}\}^{<3,e>}\}^{<2,p>}\Big\}.$$

The use of 'scopes' and 'labeled elements' in the above example is intended only as introduction to the 'look and feel' of the notation. A formal explanation of the mathematical underpinnings is available in other papers.

For a more bizarre example, consider the following:

```
<P>
  <p>
    <n>
      Alan
      <a>42</a>
    </n>
    <e>agb@abc.com</e>
  </p>
  <p>
    <n>
    Mary<a>29<e>mky@abc.com</e></a>
    </n>
  </p>
</P>
```

This example is not presented as one that is likely to occur in the reader's reality, but as one that is intended to stress the notational strength of the formal notation.

This can easily be represented using labeled elements, as follows:

$$\mathbf{P} = \Big\{ \ \{\{Alan^{<1>},\ \{42^{<1>}\}^{<2,a>}\}^{<1,n>},\ \{agb@abc.com^{<1>}\}^{<2,e>}\}^{<1,p>},$$
$$\{\{Mary^{<1>},\ \{29^{<1>},\ \{mky@abc.com^{<1>}\}^{<2,e>}\}^{<2,a>}\}^{<1,n>}\}^{<2,p>}\Big\}.$$

The Tags of XML-Structures may also contain attributes expressed as name/value pairs. This information can also be precisely captured as part of the label or scope associated with the appropriate element as in the following semantically stressed example.

```
<P>
  <p>
    <n a='42' e='agb@abc.com' >
      Alan
    </n>
  </p>
  <p>
    <n a='29' e='mky@abc.com' >
    Mary
    </n>
  </p>
</P>
```

The following demonstrates capturing 'attribute information' as part of an element's label or scope:

$$\mathbf{P} = \Big\{ \ \{\{Alan^{<1>}\}^{<1,n,\{\{42^{<1>}\}^{<1,a>},\ \{agb@abc.com^{<1>}\}^{<2,e>}\}>}\ \}^{<1,p>},$$
$$\{\{Mary^{<1>}\}^{<1,n,\{\{29^{<1>}\}^{<1,a>},\ \{mky@abc.com^{<1>}\}^{<2,e>}\}>}\ \}^{<2,p>}\Big\}.$$

XML-Structures are not always as simplistic as these examples. However, more complex examples are just a matter of degree and do not depend on any new structural requirements. The following example includes some structural complexities that might naturally or theoretically arise in an XML-Structure, including embedded elements. Again the following example is intended to illustrate certain structural complexities and is not intended to be semantically justifiable.

```
<P>
  <p>
    <n>Alan<z>Z</z>G.<w>W<y d='H'>Q</y></w>Brown</n>
    <a>42</a>
    <e>agb@abc.com</e>
  </p>
  <p a="29" n="Mary" e="mky@abc.com" />
</P>
```

The following is one way to capture the mathematical identity of the above.

$$\mathbf{P} = \Big\{ \ \Big\{\{Alan^{<1>},\ \{Z^{<1>}\}^{<2,z>},\ G.^{<3>},\ \{W^{<1>},\ \{Q^{<1>}\}^{<2,y,\{\{H^{<1>}\}^{<1,d>}\}>}\}^{<4,w>},$$
$$Brown^{<5>}\}^{<1,n>},$$
$$\{42^{<1>}\}^{<2,a>},\ \{agb@abc.com^{<1>}\}^{<3,e>}\ \Big\}^{<1,p>},$$
$$\emptyset^{<2,p,\{\{Mary^{<1>}\}^{<2,n>},\ \{29^{<1>}\}^{<1,a>},\ \{mky@abc.com^{<1>}\}^{<3,e>}\}>}\ \Big\}.$$

These examples are not intended to be self-explanatory and will be treated in greater depth in other papers. The examples are presented here as a preview of how it is possible to formally transform the 'vertical' structure-centric notation of XML-Structures into an informationally equivalent 'horizontal' operation-centric notation in terms of extended sets.

Transforming a non-mathematical notation into a very formal arcane mathematical notation does not, in general, achieve much more than would be achieved by translating Mother Goose into Latin. It may look more impressive and convince some that it is now more academically sound, but the only immediate effect is that all the friendly intuition of the old notation has been destroyed.

Mathematical notation of and by itself does not guarantee mathematical soundness of the represented concepts. Any formal description of software development proves this point. For a hostile mathematical notation to be of sufficient practical value as to warrant the time and discomfort required make it friendly, it has to provide benefits far beyond its notational compactness.

The benefit in capturing the mathematical identity of XML-Structures in terms of extended sets is that all extended sets are indeed mathematically sound operands that behave very predictably

under extended set operations, Xops. Since Xops are just extensions of set-theoretic operations that work on sets of 'labeled elements', all the mathematical power of set theory can now be harnessed to design, develop, implement, and use systems intended to process XML-Structures.

```
<P>
  <p a="42" n="Alan" e="agb@abc.com" />
  <p a="29" n="Mary" e="mky@abc.com" />
</P>
```

```
<P>
  <p>
    <n>
      Alan
      <a>42</a>
    </n>
    <e>agb@abc.com</e>
  </p>
  <p>
    <n>
    Mary<a>29<e>mky@abc.com</e></a>
    </n>
  </p>
</P>
```

D SUPPLEMENT

Following is a TPC-H Q9 load and execution demonstration of scalability as the input database grows from 1GB to 10GB.

The platform is a Windows 2000 Pro. with a 2.8GHz CPU, one disk drive with 7.6GB available for loading all required data for all 22 queries and for executing Q9 restricted to 64MB of RAM. (10GB is included, but there was not enough disk space for a full 10GB load.)

Base Q9: The time to execute Q9 on non-optimized data.

DATA OPT: The time required to construct optimized data for any Q9.

OPT Q9: The time to execute Q9 on optimized data.

PLATFORM: 2.8GHz Pentium, Available RAM = 64MB, Disk = 7.6GB

		1GB	2GB	4GB	8GB	10GB
[1]	Base Q9:	6.08s	12.07s	24.57s	55.60s	-
[2]	DATA OPT:	0.56s	0.55s	0.56s	0.55s	-
[3]	OPT Q9:	3.08s	10.07s	20.57s	40.60s	53.28s

A general business need is running repeat queries, once a "query class" is optimized. Following is a run of 10 random Q9 queries.

		1GB	2GB	4GB	8GB	10GB
[4]	10 Random Q9:	30.08s	100.07s	200.57s	400.60s	530.28s
		0.50m	1.50m	3.00m	6.50m	9.50m

Actual load and execution of Q9 for a 2GB database.

The platform is a Windows 2000 Pro. with a 2.8GHz CPU, one disk drive with 7.6GB available for loading all required data for all 22 queries and for executing Q9 using RAM restricted to 64MB. (I included 10GB, but did not have enough disk space for a full 10GB load.)

Original DBGEN generated 2GB TBL files

```
lineitem.tbl   1,508,188Kr...  8-18-04  3:25:48 am
orders  .tbl 348,725,075 r...  8-18-04  3:25:48 am
partsupp.tbl 240,558,676 r...  8-18-04  3:25:48 am
customer.tbl  49,127,194 r...  8-18-04  3:25:48 am
part    .tbl  48,926,277 r...  8-18-04  3:25:48 am
supplier.tbl   2,847,094 r...  8-18-04  3:25:48 am
nation  .tbl       2,128 r...  8-18-04  3:25:48 am
region  .tbl         401 r...  8-18-04  3:25:48 am
Files              8
Bytes  2,234,570,875
```

General XSP Files after full load for all Queries

```
L       .XHH 467,921,908 .a.. 12-16-04  2:04:41 am
LL      .XHH   1,218,547K.a.. 12-16-04  2:04:41 am

O       .XHH  63,000,064 .a.. 12-16-04  2:04:42 am
OO      .XHH 342,000,064 .a.. 12-16-04  2:04:42 am

Q       .XHH  32,000,064 .a.. 12-16-04  2:04:42 am
QQ      .XHH 331,200,064 .a.. 12-16-04  2:04:42 am

C       .XHH  66,000,064 .a.. 12-16-04  2:04:42 am

P       .XHH  23,600,064 .a.. 12-16-04  2:04:42 am
PP      .XHH  43,200,064 .a.. 12-16-04  2:04:42 am

S       .XHH     100,064 .a.. 12-16-04  2:04:42 am
SS      .XHH   3,880,064 .a.. 12-16-04  2:04:42 am

N       .XHH       4,539 .a.. 12-16-04  2:04:42 am

R       .XHH         954 .a.. 12-16-04  2:04:42 am
--------------------------------------------------
Files             13
Bytes  2,620,699,625
--------------------
```

```
GENERAL SCOPE COVERINGS FOR TPC-H QUERIES
-------------------------------------------------------------------------
 LINEITEM.TBL:< L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12,L13,L14,L15,L16 >
        L.XHH:< L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11 >
       LL.XHH:< L1,L2,L3,L4,L11,L12,L13,L14,L15,L16 >

   ORDERS.TBL:< O1,O2,O3,O4,O5,O6,O7,O8,O9 >
        O.XHH:< O1,O2,O3,O4,O5 >
       OO.XHH:< O1,O6,O7,O8,O9 >

 PARTSUPP.TBL:< Q1,Q2,Q3,Q4,Q5 >
        Q.XHH:< Q1,Q2,Q3,Q4 >
       QQ.XHH:< Q1,Q2,Q5 >

 CUSTOMER.TBL:< C1,C2,C3,C4,C5,C6,C7,C8 >
        C.XHH:< C1,C2,C3,C4,C5,C6,C7,C8 >

     PART.TBL:< P1,P2,P3,P4,P5,P6,P7,P8,P9 >
        P.XHH:< P1,P2 >
       PP.XHH:< P1,P3,P4,P5,P6,P7,P8,P9 >

 SUPPLIER.TBL:< S1,S2,S3,S4,S5,S6,S7 >
        S.XHH:< S1,S4 >
       SS.XHH:< S1,S2,S3,S4,S5,S6,S7 >

   NATION.TBL:< N1,N2,N3,N4 >
        N.XHH:< N1,N2,N3,N4 >

   REGION.TBL:< R1,R2,R3 >
        R.XHH:< R1,R2,R3 >
-------------------------------------------------------------------------
```

The scope values are taken directly from the TPC-H definitions. But this is not obvious since the mid-level system evoking the low level XSP I/O interface could have chosen any information-preserving representation during the load phase.

For example:

```
    NATION.TBL:< N1,N2,N3,N4 >
         N.XHH:< N1,N2,N3,N4 >

            N1: LOGICAL*1  - NATIONKEY
            N2: CHAR*25    - NAME       (CHAR*16 would work)
            N3: LOGICAL*1  - REGIONKEY
            N4: CHAR*152   - COMMENT


Non-optimized subset required for execution of Q9
-------------------------------------------------
L       .XHH 467,921,908 .a.. 12-16-04  2:04:41 am
O       .XHH  63,000,064 .a.. 12-16-04  2:04:42 am
Q       .XHH  32,000,064 .a.. 12-16-04  2:04:42 am
P       .XHH  23,600,064 .a.. 12-16-04  2:04:42 am
S       .XHH     100,064 .a.. 12-16-04  2:04:42 am
N       .XHH       4,539 .a.. 12-16-04  2:04:42 am
-------------------------------------------------
Files              6
Bytes    586,626,703
--------------------




First Q9 Result set
-------------------------------------------------
NYS_2GB .XSP       4,439 .a.. 12-16-04  2:07:52 am
-------------------------------------------------




Optimized pre-join for Q9 (obviates L & Q)
-------------------------------------------------
LXQA    .XHH 251,957,980 .a.. 12-16-04  2:08:51 am
-------------------------------------------------

      LXQA.XHH:< L1,L2:Q1,L3:Q2,L4,amount >

            L1: INTEGER*4  - ORDERKEY
         L2:Q1: INTEGER*4  - PARTKEY
         L3:Q2: INTEGER*4  - SUPPKEY
            L4: LOGICAL*1  - LINENUMBER
        amount: REAL*8     - UDF3(L5,L6,L7,Q4)

      LXQA.XHH is used with a slightly different QSF script than
      the one in the whitepaper. [2a] replaces [2] and [3]. [2a]
      accepts R1 and LXQA as inputs and produces R3 and the rest
      of the original QSF script continues on.




Semi-optimized covering for execution of Q9
-------------------------------------------------
LXQA    .XHH 251,957,980 .a.. 12-16-04  2:08:51 am
O       .XHH  63,000,064 .a.. 12-16-04  2:04:42 am
P       .XHH  23,600,064 .a.. 12-16-04  2:04:42 am
S       .XHH     100,064 .a.. 12-16-04  2:04:42 am
N       .XHH       4,539 .a.. 12-16-04  2:04:42 am
-------------------------------------------------
Files              5
Bytes    338,662,711
--------------------

Second Q9 Result set (optimized run)
```

```
-------------------------------------------------
NYS_2GB!.XSP       4,439 .a.. 12-16-04  2:10:10 am
-------------------------------------------------

  NYS_2GB-.XSP:< NATION, o_year, sum_profit >

        NATION: CHAR*25    - N_N2
        o_year: INTEGER*2  - UDF2(O5)
    sum_profit: REAL*8     - UDF1(amount)




Multi-Q9 Result sets
-------------------------------------------------
NYS_2GB1.XSP       4,439 .a.. 12-16-04  2:32:53 am
NYS_2GB2.XSP       4,439 .a.. 12-16-04  2:33:05 am
NYS_2GB3.XSP       4,439 .a.. 12-16-04  2:33:15 am
NYS_2GB4.XSP       4,439 .a.. 12-16-04  2:33:26 am
NYS_2GB5.XSP       4,439 .a.. 12-16-04  2:33:36 am
NYS_2GB6.XSP       4,439 .a.. 12-16-04  2:33:47 am
NYS_2GB7.XSP       4,439 .a.. 12-16-04  2:33:58 am
NYS_2GB8.XSP       4,439 .a.. 12-16-04  2:34:08 am
NYS_2GB9.XSP       4,439 .a.. 12-16-04  2:34:20 am
NYS_2GBa.XSP       4,439 .a.. 12-16-04  2:34:29 am
-------------------------------------------------


Total space used by iXSP
------------------------
Files             26
Bytes  2,872,710,873
------------------------




QSF Script for non-opt Q9
-------------------------

   [1] LWHEREP( {P2=%green},  P, {P1},                          R1 )

   [2] JWHEREP( {P1:Q1}, Q, R1,        *,                       R2 )

   [3] JWHEREP( {Q1:L2, Q2:L3}, L, R2, *,                       R3 )

   [4] XSP_UDF( UDF3(L6,L7,Q4,L5,amount) )
   [4] JWHEREP( {L1:O1}, R3, O, {L3, UDF2(O5,o_year), amount}, R4 )

   [5] XSP_UDF( UDF1(amount, sum_profit) )
   [5] JWHEREP( {L3:S1}, R4, S, {L3, o_year, sum_profit},      R5 )

   [6] JWHEREP( {N1:S4}, N, R5, {NATION, o_year, sum_profit},  R6 )



BEFORE EXECUTION
------------------
L.XHH  467,921,908
O.XHH   63,000,064
Q.XHH   32,000,064
P.XHH   23,600,064
S.XHH      100,064
N.XHH        4,539
------------------

BEGINNING OF Q9 EXECUTION
-------------------------
```

```
Files 6, Bytes  586,626,703

EXECUTION OF Q9 FROM GENERAL LOAD: HH_Q09.QSF
---------------------------------------------

                                                DISK USAGE
 HH_Q09                                         ----------
 ------

  XSP OPS    IN              OUT        READ_IN     WRITTEN_OUT
  -------    --              ---        -------     -----------

    [1]    P.XHH        ----> R1     23,600,064   =      92,796


    [2]    R1,                            92,796
           Q.XHH        ----> R2   + 32,000,064   =    1,483,77
                FREE(R1)


    [3]    R2,                         1,483,776
           L.XHH        ----> R3   +467,921,908   =  14,630,470
                FREE(R2)


    [4]    R3,                        14,630,470
           O.XHH        ----> R4   + 63,000,064   =   9,753,668
                FREE(R3)


    [5]    R4,                         9,753,668
           S.XHH        ----> R5   +    100,064   =       1,989
                FREE(R4)


    [6]    R5,                             1,989
           N.XHH        ----> R6   +      4,539   =       4,439
                FREE(R5)


END OF Q9 EXECUTION                               =       4,439
-------------------
Files 7, Bytes  586,631,142
[Elapse Time :   14.84 sec.,   0.25 min.] {HH_Q09.QSF}


GENERATION OF OPT FOR REPEATED Q9: H_OPT.QSF      = 251,957,980
----------------------------------
Files 8, Bytes  838,589,122
[Elapse Time :   37.24 sec.,   0.62 min.] {H_OPT.QSF}


EXECUTION OF Q9 USING OPTIMIZATION: HHH_Q09.QSF    =      4,439
-----------------------------------
Files 9, Bytes  838,593,561
[Elapse Time :    7.92 sec.,   0.13 min.] {HHH_Q09.QSF}


EXECUTION OF MULTIPLE Q9s: H_MULTI.QSF             =     44,390
--------------------------
Files 19, Bytes 838,637,951
[Elapse Time :  106.65 sec.,   1.78 min.] {H_MULTI.QSF}
```