# I/O Technology For Big Data

## Massively Parallel Data Access of Big Data Environments

D L Childs
Integrated Information Systems
Ann Arbor, Michigan

## ABSTRACT

*Traditional I/O technology is based on the storage and retrieval of records, records that are physically preserved in storage. Set processing I/O technology is based on the exchange of collections (sets) of records, records which may or may not physically exist in storage.*

*Given the advances in hardware platforms, set processing I/O technology can offer one to three orders of magnitude better system performance than traditional I/O technology.*

## 1. INTRODUCTION

Traditional I/O technology, initially developed in the '70s, was never intended for massively parallel I/O access to big data environments. There is a need today for an alternate I/O technology specifically intended to support massively parallel access to big data environments.

Instead of using application dependent organizations of data in storage to support the specific access needs of each individual application, an alternation would decouple all storage from all application needs and supply applications with relevant data as needed, when it is needed, and in the appropriate form as needed for each application.

This proposed structural disconnect between applications and storage is indeed antithetical to current practice and requires an explanation as to how data would be exchanged. Data would be exchanged based on data relevance instead of on data location. Data relevance would be conveyed through a mathematical interface between applications and storage.

Such an interface requires three parts: a mathematical description of physically stored data, well-defined operations to manipulate and access the stored data representations, and a mapping strategy for equating physical application data with abstract mathematical definitions.

This paper will describe a mathematical interface between applications and storage based solely on data defined in terms of set-theoretic membership conditions.

## 2. SET-THEORETIC DATA STRUCTURES

In 1968 a paper[3] was published extolling the idea that data, physically stored as well-defined mathematical objects, could then be manipulated and accessed by a small collection of mathematical operations. The idea of treating data as a mathematical operand was not new[2], but the approach was.

The reasonable approach to modeling objects of any kind was to use classical set theory as a base. However in 1957 Skolem[1] observed that the n-tuple was not 'defined in a suitable way'. Thus, using the n-tuple to model records as operands could produce 'unsuitable' results. For example: $< a, b > \cap < a, c > = < a, a >$ is not the expected behavior for determining the common component of two records,

Since records are a common form of representing data and since n-tuples are a conceptually convenient representation for records, the only reasonable approach seemed to be to discover a *suitable* definition for n-tuples.

### 2.1 Extensions to CST

The initial exploration into defining n-tuples so as to behave as *expected*, resulted in a very convoluted mutilation of classical set theory (CST). The results were not pretty, but they worked.

*Worked* in this context means that the use of the extended definitions of n-tuple give results that are expected. The intersection of $< a, b >$ with $< a, c >$ gives $< a >$, as it should. The intersection of $< a, b, c >$ with $< x, b, y >$ gives $<, b, >$, which is certainly an improvement over a null result.

### 2.2 STDS & MICRO

Given the extended 'more suitable' definition of n-tuples it seemed feasible to use extended n-tuples as a primitive construct for implementing a data structure.

The MICRO[7] project at the University of Michigan was established to implement a data analysis capability based on storing and accessing all record data as mathematically well-defined collections of n-tuples.

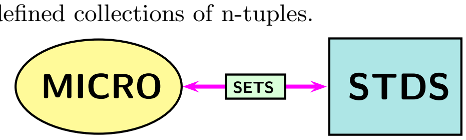


Figure 1: MICRO/STDS Set Accessing I/O Interface.

The resultant MICRO DBMS architecture consisted of two components: a user-friendly front-end language that accepted application queries phrased in constrained English, and a set-theoretic data structure (STDS) backend that stored all data as well defined set-theoretic collections of extended n-tuples. The key feature of this architecture was in how the two components communicated.

### 2.3 Set Accessing I/O

The application component and the storage component of the MICRO DBMS communicated using set operations and only set operations. Though possibly radical, viewed in perspective, this approach seemed quite natural at the time. What could provide more reliability and flexibility

Set-Processing Research & Development Series [IOTBD: 11/08/12]

than a mathematically sound data accessing architecture?

Armed with a new (and suitable) definition for n-tuples it was now possible to define new set operations that could manipulate and combine sets of extended n-tuples. These extended set processing (XSP) operations became the I/O interface between the MICRO applications and storage.

At the time MICRO and STDS were being developed in the early 1970s there was no formal mathematical model to support general set-theoretic I/O architectures. The formal foundation was a cobbled version of classical set theory, which though mathematically sound, was too limiting.

Though working on general foundations for an extended set theory was always a background task, it never seemed quite necessary given the sluggish I/O capabilities of the existing hardware platforms.

## 2.4 I/O Performance Potential

Today the story is quite different. The hardware advances in disk capacity and data transfer rates have provided an I/O performance potential that can be easily tapped by use of parallel data access techniques. Unfortunately, traditional I/O technologies thwart any hope of benefiting from this I/O performance potential.

The advantages of parallel data access techniques are well known. If one system access 10 I/O buffers serially and another access the same I/O buffers concurrently, then the second system is potentially an order of magnitude faster than the first.

Since data partitioning is requisite for parallel I/O and since partitioning is a basic operation, set-theoretic data structures provide all the basic ingredients for massively parallel I/O strategies.

# 3. XSP/STDS I/O ARCHITECTURES

For an architecture to support massively parallel I/O, storage organizations have to be structurally independent of all applications. Any reliance an application has on how data is represented in storage, severely restricts any ability to reorganize stored data into partitioned I/O packets.

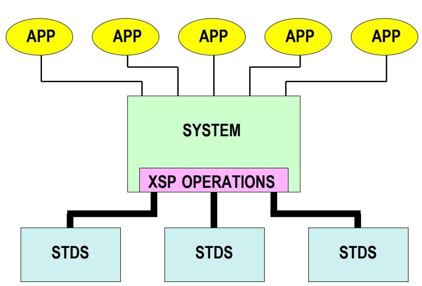


Figure 2: XSP/STDS Application Independent I/O Architecture

A most significant feature of a XSP/STDS architecture is the structural separation of application representations of data from the representations of data in storage.

Unlike traditional database architectures where stored data is tailored to fit application data access requirements, STDS modules are autonomous organizations of data, dynamically restructurable to meet the data access needs of any and all concurrently running applications.

The enabling feature for allowing structurally independent data access is a collection of extended set theory operations that process set transformations between application data representations and storage data representations. The only architectural requirement for a XSP/STDS architecture is that the interface extended set processing (XSP) operations have an explicit set-theoretic definition.

## 3.1 XSP Operations

Extended set processing (XSP) operations are operations defined to manipulate sets that satisfy axioms of extended set theory (XST)[32], These axioms differ from those of classical set theory (CST) in a number of significant ways.

The principle contribution of XST to computing is in being able to mathematically control the transformation between application data representations and storage data representations.

Since classical set theory does not support structured sets[1] physical representations of data can not be well-define as set-theoretic operands. XST resolves this with a definition of *tuplesets*[2], (see [33]).

STDS has supported commercial applications since 1972[7], but general development of XSP/STDS implementations was not feasible until the axiomatic foundation of XST had been completed. This axiomatic foundation is now completed and is available[32] for supporting general systems development.

## 3.2 STDS I/O Modules

On the application side, there are two different kinds of XSP sets: arbitrary abstract sets reflecting intrinsic data relationships, and structured sets representing application formatted data.

On the STDS storage side sets are tuplesets representing structured data in storage. There is no restriction on the physical representations of data in STDS storage. Any and all possible data representations and organizations have an XST formulation. Thus all currently implemented data structures can be represented in STDS storage, even indexed data structures

## 3.3 Massively Parallel I/O

Since amy implementation of concurrent I/O to a large block of data requires a partitioning of the block, then any block of data that can not be partitioned can only be accessed serially. Since partitioning is a basic concept of set theory, XSP/STDS provides mathematical control over dynamic partitioning of data and thus allows highly flexible control over parallel I/O.

This mathematical control over I/O performance allows developers to explore I/O performance strategies after data has already been loaded. This is a dramatic departure from traditional I/O technologies that require data I/O access strategies to be determined prior to data being loaded.

## 3.4 XSP Technology

Traditional data management technology relies on data structurings and data structure traversal to support optimal data access performance. In sharp contrast XSP technology relies on XSP operations and XSP optimization strategies to optimally derive relevant data, and only relevant data, from arbitrarily large collections of stored data.

The remainder of this paper will show how XSP/STDS architectures can employ XSP operations to provide optimal

[1]not just a suitable definition for n-tuples, but also the ordering and duplication of elements

[2]sets of n-tuple elements with matching n-tuple scopes

data delivery to any application requiring high performance data access to large pools of stored data.

## 4. I/O TECHNOLOGIES

Record accessing I/O technology is based on use of data access structures. Set processing I/O technology is based on use of data accessing operations. It is hard to imaging any two technologies that are more intrinsically antithetical.

Though set processing I/O technology predates[7] record accessing[17] technology for supporting relational database management systems, record access I/O technology is the current industry standard. Unfortunately, the limitations of structured data access are beginning to stress its utility[3].

### 4.1 I/O Throughput

For purposes of this paper only the differences between the technologies that impact I/O performance are of interest, where I/O performance is a measure of data throughput, or how long it takes data required by an application to be transfered to the application from storage. Just three parameters control data throughput:

1) I/O buffer size,
2) percent of relevant data buffer, and
3) concurrent I/O.

Structured data access I/O technologies inhibit all three, while set processing I/O technology offers at least an order of magnitude performance improvement on each.

### 4.2 I/O Buffer Size

Since the data transfer rate (DTR) on any given platform is a constant for both technologies, it is not of comparative interest. And though disk latency time is also a constant for both technologies, it is of great interest.

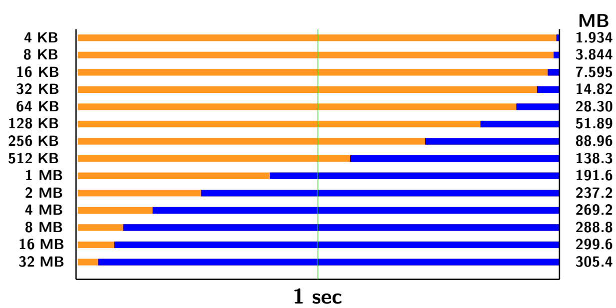


Figure 3: Buffer size impact on total data accessed.

Since there is search overhead for locating each buffer, minimizing the number of required buffers gives the best access times. When large amounts of data are required, the larger the buffer size the better the performance. Figure 3. compares the impact of buffer size choice on the amount of data that can be accessed in one second, given a disk environment with a sustainable DTR of 320 MB/sec.

In Figure 3. the cost (in elapsed time) for locating buffers to be transfered, is represented in orange. The actual data transfered is represented in blue. For any I/O transfer buffer under 1 MB, the system spends more time looking for data than in transferring data. A buffer size of less than 1 MB is clearly sub-optimal for accessing large volumes of data. Conversely, using a 32 MB allows a system to access a terabyte of data in less than an hour, using just one I/O port. Using forty[4] I/O ports (or 40 cores as in Figure 8) a terabyte of data can be accessed in 1.43 minutes.

Just by switching from 64 KB[5] I/O buffers to 32 MB I/O bufferers, I/O throughput can be improved by an order of magnitude.

### 4.3 Relevant Data Buffers

Structured data I/O technologies currently come in two flavors: row store records and column store records. Both are record oriented I/O technologies that rely on indexed data access structures, but column store implementations provide a higher percentage of relevant data per I/O buffer.

However neither row nor column storage strategies are very I/O efficient. A typical I/O buffer contains only 50% of actual data[21]. Of that 50% only 5%[26] of the data is application relevant. For column store the percentage is generally four times better than for row store systems.

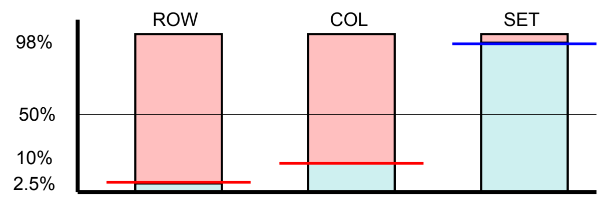


Figure 4: Percent of Relevant Data per Transfer.

In sharp contrast to record accessing implementations, set accessing implementations can tailor I/O buffers with nearly 100% of application relevant data.

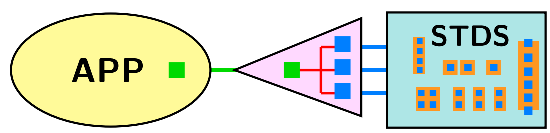


Figure 5: Mapping application relevant data to application format.

Set accessing systems, already operating in commercial environments [7], provide data transfers with 98% to 100% relevant data, requiring virtually no data to be actively rejected[6] by an application.

It should be noted that in MICRO this performance difference was due solely to improved relevant data capacity and not on parallel I/O, since only one disk was involved

### 4.4 Concurrent I/O

It is not unreasonable, given the capabilities of today's hardware, to assume that a platform could support as many concurrent I/Os as there were available CPU cores.

It also seems reasonable to assume that processor power is more than sufficient for applications to keep up with I/O data input. Then with just a 10 core platform, concurrent I/O performance could provide an order of magnitude better performance than an implementation limited to serial I/O.

When I/O buffer size is increased from 64 KB to 32 MB, when relevant data content is increase from 10% to near 100%, and when 10 or more concurrent I/O streams are employed, the accumulate system performance is improved by over three orders of magnitude.

[3] Vint Cerf: "It's like 1973 for moving data around.."

[4] a number compatible with TPC-H 1TB participants.

[5] DB2 uses 64 KB buffers, see [21] p.371

[6] studies have shown that nearly 90% of an application's processing time is spent rejecting non-relevant data[23]

# 5. SQL & XSP/STDS

SQL is a set-theoretic language that has been wrapped in a data access cocoon for over thirty years. The Select statement in SQL specifies the membership condition of the desired result. The conditions for specifying a membership condition are based on operations defined by the Relational Data Model, RDM, published 1n 1970 by E. F. Codd[5].

The RDM defines *logical* operations applied to *logical* operands. Since both operations and operands are *logical*, there is no issue of performance involved to influence the number nor order of operations. Nor is there any issue of performance with the size of any logical operands involved.

Just because implementors disregarded data independence, does not mean that they had to. In fact, the MICRO[7] DBMS provided a set-theoretic I/O interface between applications and storage. Thus ensuring the highest form of application independence.

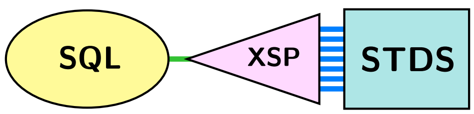


Figure 6: SQL Set-Membership Interface with XSP/STDS

Since all SQL implementations are built on a set-theoretic data specification model, there is no reason this specification could not be intercepted before being neutered by traditional data access strategies.

## 5.1 SQL Performance Comparisons

Set-accessing architectures can typically provide one to two orders of magnitude better performance than row-store architectures. A case in point is a performance comparison between row-store IBM and Oracle products and a set-access implementation, iXSP.

The comparison[19] was made on a very small[7], single CPU, single disk platform. Query was an SQL 5-way join on three different database sizes (1 GB, 2 GB, and 4 GB) each containing eight separate tables.

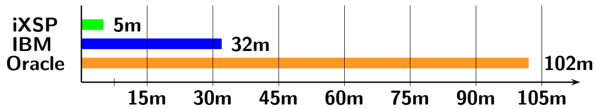


Figure 7: Average Load Time per Gigabyte.

There are a number of reasons why iXSP loaded six times faster than IBM and twenty times faster than Oracle. Though all three systems loaded the same *relevant* data, the IBM and Oracle row-store architectures required installation of access path data in order to support row-store data access, while only the relevant data was required for iXSP since the required set-accessing operations were already available.

In cases where data loading performance is not a critical concern but where repeated queries are accessing pre-loaded data, the performance shifts to pure execution comparisons.

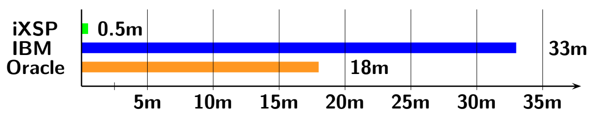


Figure 8 Average Query Time for SQL 5-Way Join.

[7]Intel 500MHz, 256 MB RAM, 30 GB IDE, Windows NT

# 6. TPC-H SYSTEMS COMPARISON

Given a platform with $n$-CORES and assuming enough processing power to keep up with I/O throughput, a lower bound on system performance can be established by the time it takes to fully load all relevant data and to execute applications of interest.

Using the TPC-H results[8] for data loading and Query 9 execution times for 10,000 GB of data, a hypothetical I/O performance comparison can be made between actual results and those possible by utilizing I/O throughput.

| TPC-H 1000GB 11/01/12 | | | |
|---|---|---|---|
| System | Data Load | Query 9 | TET |
| 32-CORES | 104.78 sec | 10.48 sec | 1.93 min |
| 40-CORES | 83.83 sec | 8.39 sec | 1.54 min |
| 64-CORES | 52.39 sec | 5.24 sec | 0.96 min |
| Oracle(32) | 1.38 hrs | 93.4 sec | 1.41 hrs |
| Microsoft(40) | 21.07 hrs | 162.2 sec | 21.12 hrs |
| DB2(64) | 0.82 hrs | 500.7 sec | 0.96 hrs |

Figure 9: Optimal I/O vs. Actual Indexed I/O

The above table shows actual Load, Query 9, and TET calculations for three commercially available products, all executing the same query on the same input data. Since real commercial installations are using computers to gain a competitive advantage, the real cost to a business is not the cost of equipment but the cost of lost business. Spending two or three million more dollars in three years on equipment to save from losing two or three million dollars a quarter is of more interest to business than a geometric mean of queries per hour. Only one real issue faces businessmen concerned with competitive advantage, the total elapsed time, TET, required to keep ahead of the competition.

It should be noted that the hypothetical results in Figure 9 are based solely on concurrent I/O and not on increased buffer size nor on high relevance of application data. Thus, the Q9 results can be improved and with fewer cores.

# 7. CONCLUSION

Given the advances in hardware platforms, set processing I/O technology can offer one to three orders of magnitude better system performance than traditional I/O technology.

# 8. REFERENCES

[1] Skolem, Thoralf: *Two Remarks on Set Theory,* Mathematica Scandinavica 5 (1957), p.43-46. - "Skolem concludes: I shall not pursue these considerations here, but only emphasize that it is still a problem how the ordered n-tuple can be defined in the most suitable way." http://www.mscand.dk/article.php?id=1481

[2] Lisp (programming language) From Wikipedia, http://en.wikipedia.org/wiki/Lisp_%28programming_language%29

[3] Childs, D. L.: *Feasibility of a set-theoretic data structure. A general structure based on a reconstituted definition of relation.* IFIP Cong., Edinburgh Scotland, August 1968 This obsolete paper proposed the thesis that mathematical control over the representation, management, and access of data was critical for the functional freedom of applications and I/O performance control of future system. Since classical set theory could not support meaningful and unambiguous operations on records represented as n-tuples,

[8]as of 11/01/12

extensions to set theory were proposed that would allow data representations to be recognized as mathematical objects.

[4] Childs, D. L.: *Description of a set-theoretic data structure* AFIPS fall joint computer conference San Fransico CA, December 1968 Abstract: The overall goal, of which this paper is a part, is the development of a machine-independent data structure allowing rapid processing of data related by arbitrary assignment such as: the contents of a telephone book, library files, census reports, family lineage, graphic displays, information retrieval systems, networks, etc. Data which are non-intrinsically related have to be expressed (stored) in such a way as to define the way in which they are related before any data structure is applicable. Since any relation can be expressed in set theory as a set of ordered pairs and since set theory provides a wealth of operations for dealing with relations, a set-theoretic data structure appears worth investigation.
http://dl.acm.org/citation.cfm?id=1476663

[5] Codd, E. F.: *A Relational Model of Data for Large Shared Data Banks*, CACM 13, No. 6 (June) 1970
http://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf
a) The term RELATION is used in its accepted mathematical sense. b) Expository arrays (TABLES) are not an essential part of the relational view.
http://xsp.xegesis.org/Relations_70.pdf[p. 379-380]

[6] Codd, E. F.: *Relational Completeness of Data Base Sublanguages*, Database Systems: 6598, 1970
http://classes.soe.ucsc.edu/cmps277/Fall08/Papers/Codd72a.pdf

[7] MICRO & STDS: MICRO was the first-large scale set-theoretic database management system to be used in production. MICRO applications accessed data from storage using Set-Theoretic Data Structure (STDS) software. The I/O routines treat the physical data as mathematical sets and perform set operations directly on the stored data. MICRO supported a natural language interface which allowed non-programmers to use the system. (1972-1998)
http://en.wikipedia.org/wiki/Micro_DBMS

[8] 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: 28-46 Abstract: Three distinct components comprise an information system: Information, Data, and Storage management. Until recently, all three have been subsumed under data management. As applications become more demanding, as support criteria become more complex, and as storage capacity becomes very large the need for functional independence of these three management areas has become more apparent. Recognition of this situation has been popularized through the phrase, "data independence", or more precisely, "data independence from information" and "data independence from storage".

The difficulty in achieving data independence arises through the incompatibility of a complex information space being supported by a simple storage space. The popular, but limiting approach, has been to force the information space into a restrictive record space. This achieves a deceptive compatibility allowing only the appearance of data independence at the user level. This record oriented approach has become pervasive for small databases even though it constrains user applications, requires substantial storage overhead, and imposes inherent processing inefficiencies.

As databases become very large and as distributed systems become desirable the need for inherent (not superficial) data independence becomes crucial. This paper is intended as a tutorial and will describe conditions for data independence and summaries the concepts of Extended Set Theory as a general model for expressing information systems embodying data independence. This generality will be demonstrated by considering some major problems pertinent to the design and support of very large, distributed, backend information systems.

It should be emphasized that Extended Set Theory is a formalism for expressing solutions and is not a specific solution in itself. Though "redundant membership condition", "distributed membership condition", and "set-theoretic interface" may be new concepts, Extended Set Theory does not preclude any current DBMS concepts, data structures, or existing implementations. Rather, Extended Set Theory embraces them all under a unifying model.
http://xsp.xegesis.org/VLDB_77abstract.pdf

[9] Boyce, R. F.; Chamberlin, D. D.; King, W. F.; Hammer, M. M.: *Specifying Queries as Relational Expressions: SQUARE*, IBM Technical Report RJ 1291, IBM Research Laboratory, San Jose, CA., 1973
http://dl.acm.org/citation.cfm?id=361221

[10] Chamberlin, D. D.; Boyce, R. F.: *SEQUEL: A Structured English Query Language*, IBM Research Laboratory, 1975
ABSTRACT: In this paper we present the data manipulation facility for a structured English query language (SEQUEL) which can be used for accessing data in an integrated relational data base. Without resorting to the concepts of bound variables and quantifiers SEQUEL identifies a set of simple operations on tabular structures, which can be shown to be of equivalent power to the first order predicate calculus. A SEQUEL user is presented with a consistent set of keyword English templates which reflect how people use tables to obtain information. Moreover, the SEQUEL user is able to compose these basic templates in a structured manner in order to form more complex queries. SEQUEL is intended as a data base sublanguage for both the professional programmer and the more infrequent data base user.
http://faculty.cs.tamu.edu/yurttas/PL/DBL/docs/sequel-1974.pdf

[11] Hardgrave, W. T.: *A technique for implementing a set processor*, Proceedings of the 1976 conference on Data : Abstraction, definition and structure archive ACM New York, NY, USA 1976 Abstract: Extended set theory is an effective method for describing the complex data structures needed to support large-scale data base applications. This paper describes the incorporation of the salient concepts of extended set theory into a general tool for the design and implementation of large-scale database systems. This tool is called a set processor.
http://dl.acm.org/citation.cfm?id=807126

[12] Codd, E. F.: *Relational Database:* A Practical Foundation for Productivity, 1981 ACM Turing Award Lecture, Comm. ACM vol 25, No. 2, 1982
http://classes.soe.ucsc.edu/cmps277/Fall08/Papers/Codd72a.pdf

[13] Maier, D.: *The Theory of Relational Databases*, Computer Science Press, Inc., 1983 One of the best books written on Relational Databases. Out of print, but available on line.
a) Relational algebra is a procedural system that gives a set of operations on relations and an **order** in which to perform them. A calculi expresses *what* a result should be, but *not how* to compute it. Results specified by a calculi tend to relieve applications from how to determine results, [p. 224].
http://www.cs.pdx.edu/~maier/TheoryBook/TRD.html

[14] Childs, D. L.: *VLDB Panel : Inexpensive Large Capacity Storage Will Revolutionize The Design Of Database Management Systems* Proceedings of the Tenth International Conference on Very Large Data Bases. Singapore, August, 1984 Abstract: As secondary storage devices increase in capacity and decrease in cost, current DBMS design philosophies become less adequate for addressing the demands to be imposed by very large database environments. Future database management systems must be designed to allow dynamic optimization of the I/O overhead, while providing more sophisticated applications involving increasingly complex data relationships. It is the purpose of this panel to present and discuss the potential impact that inexpensive large capacity storage devices will have on the design, implementation, management, and capabilities of future DBMSs. Specific attention will be directed to: Reducing the I/O Overhead and Increasing Application Functionality with discussion to include:
* Very large distributed databases
* Mixing storage devices with diverse performance/cost characteristics
* Transparent database management systems
* Continuous operation with dynamic modification to system support
* Integration of DBMS packages.
http://www.informatik.uni-trier.de/~ley/db/conf/vldb/vldb84.html

[15] 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 Abstract: A Hypermodel Syntax for Precision

Modeling of Arbitrarily Complex Systems: This paper focuses on resolving three specific system development issues. The approach is to introduce the concept of a Function Space Architecture as a new methodology to system design. The basic architectural unit of this new methodology is a Function Space which can provide as much or as little detail as a specific instance requires. Coverage will include: the Function Space aa a unit OF architecture For general communication and design detail; Structure Independent Architectures as an architectural design guide far reliable and productive systems; the Hypermodel to provide the Function Space continuum with explosive resolution; and Extended Set Notation to provide generality and rigor to the concept of a Hypermodel.
http://xsp.xegesis.org/Nato_asi.pdf

[16] DeWitt, D. J; Vanderberg, S. L. : *Algebraic Support for Complex Objects with Arrays, Identity, and Inheritance.*, University of Wisconsin Introduction: The relational model of data has been very successful both commercially and in terms of the research opportunities it has provided. One of the major reasons for this is that the model lends itself to an execution paradigm that can be expressed as an algebra. An algebraic execution engine is used to process queries and to optimize them by rewriting algebraic expressions into different algebraic expressions that produce the same answer in a (hopefully) more efficient manner. Algebraic implementation/optimization techniques are well-understood and algebraic specifications of data retrieval languages lend themselves to theoretical examination in terms of expressiveness and other issues. These features make algebraic specification a desirable thing for a data model/retrieval language. In recent years it has become apparent that the relational model is not always the right choice for a particular application, and many new data models have been proposed. Thus an important open [as of 1990] question is this: Exactly how far can the algebraic approach be taken? Is it a reasonable way to implement and optimize all data models, or is there some level of complexity of the data model at which the approach is no longer justified?
http://www.cs.wisc.edu/techreports/1990/TR987.pdf

[17] The 1995 SQL Reunion: People, Projects, and Politics [System R and Ingress]
http://www.mcjones.org/System_R/SQL_Reunion_95/SRC-1997-018.pdf

[18] Champion, M.: *XSP: An Integration Technology for Systems Development and Evolution*, Software AG - 2001 Abstract: Before the "relational revolution", developers had to understand the details of the data structures on which they operated in order to access the content. Relational theory introduced the concept of "data independence" in that the operations in the relational algebra do not need to know anything about the structure of the data in order to work. This allows analysts, designers, and programmers to do their jobs without knowledge of the structure the underlying physical data, and for database administrators to reorganize, optimize, and distribute physical databases without "breaking" applications
The mathematics of the relational model is based heavily on classical set theory, CST, and this is both its strength and its weakness. For example, limitations – such as the ability to talk meaningfully only about flat tables – stem from the fact that classical set theory blurs the distinction between sets with "ordered" elements and sets with \nested} elements. Thus, operations become ill defined when extended to represent and manipulate sets with both ordered elements and nested elements. D L Childs developed an "extended set theory", XST, more than 30 years ago that adds an additional parameter known as a "scope" to the membership condition of classical set theory. In CST, membership is based on only an element component; in XST membership is based on both an element component and a scope component. This extended membership condition can be used to model ordering and containment relationships that are simply too "messy" to handle in classical set theory and the formalisms (such as relational algebra) that are based on it.
http://xsp.xegesis.org/Xsp-uxr.pdf

[19] Stout, R.:*Information Access Accelerator:* Slide presentation explaining a 40 to 1 performance improvement over commercial DBMSs by having a set-access interface between applications and storage. (IBI 2005)
http://xsp.xegesis.org/Iaa_ibi.ppt
http://xsp.xegesis.org/Riam.pdf

[20] Stonebraker, M.:(many others): *C-Store: A Column-oriented DBMS (conference slides)* By. New England Database Group. M.I.T. (VLDB 2005)
slides: http://www.vldb2005.org/program/slides/thu/s553-stonebraker.ppt
paper: http://db.csail.mit.edu/projects/cstore/vldb.pdf

[21] Lightstone, S; Teorey, T.; Nadeau, T.: *Physical Database Design*, Morgan Kaufmann, 2007 A comprehensive analysis of how complicated the physical database design process can be and how the physical structures of databases affect performance when dealing with data volume.
p. 7) A file is a set of similarly constructed records of one type, and relational tables are typically stored as files.
p. 8) An index is data organization set up to speed up the retrieval (query) of data from tables. In database management systems, indexes can be specified by database application programmers...
p. 31) Chapt. 3: "... it is important to be able to analyze the different paths for the quality of the result, in other words, the performance of the system to get you the correct result and choose the best path to get you there."
p. 371) A block (or page) has been the basic unit of I/O from disk to fast memory (RAM), typically 4 KB in size. In recent years, prefetch buffers (typically 64 KB, as in DB2) have been used to increase I/O efficiency.
p. 372) The total I/O time for a full table scan is computed simply as the I/O time for a single block, or prefetch buffer, times the total number of those I/O transfers in the table.
http://lightstone.x10hosting.com/BookFlyer-physdb.htm

[22] Stonebraker, M.; Madden, S.; Abadi, D.; Harizopoulos, S.; Hachem, N.; Helland, P.: *The End of an Architectural Era (It's Time for a Complete Rewrite)*. Presents the position that separate PDMs (engines) are needed for separate LDMs (applications). 33rd International Conference on Very Large Data Bases, Vienna, Austria, 2007.
http://www.ahzf.de/itstuff/papers/vldb07hstore.pdf

[23] Harizopoulos, S.; Madden, S.; Abadi, D.; Stonebraker, M.: *OLTP Through the Looking Glass, and What We Found There*, SIGMOD'08, June 9-12, 2008, Vancouver, BC, canada
http://cs-www.cs.yale.edu/homes/dna/papers/oltpperf-sigmod08.pdf

[24] Larsen, S. M.: *The Business Value of Intelligent Data Access*, March 2009: Article provides an excellent description on how difficult it is to hand-craft data access paths.
http://www.ca.com/files/whitepapers/data-access-white-paper-final-030909_201548.pdf

[25] Philip A. Bernstein, Istvan Cseri, Nishant Dani, Nigel Ellis, Ajay Kalhan, Gopal Kakivaya, David B. Lomet, Ramesh Manne, Lev Novik, Tomas Talius: Adapting Microsoft SQL server for cloud computing. ICDE 2011: 1255-1263

[26] Teorey, T.;Lightstone, S; Nadeau, T.Jagadish, H. V.: *Database Modeling and Design*, Morgan Kaufmann, 2011, Fifth Edition. Many in the industry consider this to be the best book available on classic database design and for explaining how to build database applications
a) Chapt. 8: "In short, transferring data between a database and an application program is an onerous process, because of both difficulty of programming and performance overhead. This difficulty is attributed to an *impedance mismatch* between databases and programming systems."
http://mkp.com/news/database-modeling-and-design-5th-edition

[27] Childs, D. L.: *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.
http://xsp.xegesis.org/Spio.pdf

[28] Childs, D. L.: *Data Representations as Mathematical Objects, Considering Content Compatibility of Relational & XML Data Representations*, The theme of this paper is to treat all data representations as mathematical objects instead of as physical structures.
http://xsp.xegesis.org/Mi_data.pdf

[29] Childs, D. L.: *Set-Store data Access Architectures:* For High Performance Informationally Dense I/O Transfers
http://xsp.xegesis.org/SSDAA.pdf

[30] Blass, A.; Gurivitch, Y.: *Why Sets?*, 2004 Mathematics Department, University of Michigan, Ann Arbor, MI 48109-1043, U.S.A. Abstract: Sets play a key role in foundations of mathematics. Why? To what extent is it an accident of history? Imagine that you have a chance to talk to mathematicians from a far-away planet. Would their mathematics be set-based? What are the alternatives to the set-theoretic foundation of mathematics? Besides, set theory seems to play a significant role in computer science; is there a good justification for that? We discuss these and some related issues.
http://research.microsoft.com/en-us/um/people/gurevich/Opera/172.pdf

[31] Childs, D. L.: *Why Not Sets?*
Abstract: Sets are well defined collections of uniquely identifiable items. Data used by computers are well defined collections of items representing situations of interest. Computers themselves are just well defined collections of bits that change value over time. It would seem that all computer processing is highly set oriented. Why are sets not more widely used in modeling the behavior and assisting the development of computing systems? The following dialogue will attempt to amplify this question, though neither of the participants has a clue to the answer.
http://xsp.xegesis.org/WNSETS.pdf

[32] Blass, A.; Childs, D L: *Axioms and Models for an Extended Set Theory*, A Formal Foundation for Unified Modeling of Mathematical Objects Mathematics Department, University of Michigan, Ann Arbor, MI
Abstract: This brief paper introduces extensions to ZFC axioms intended to accommodate the representation, manipulation, and behavior of a larger body of mathematical objects then are currently allowed by ZFC axioms alone. ZFC extensions support: a Skolem-suitable definition for n-tuples, infinitely nested sets, an escalating hierarchy of arbitrarily large sets, constructive definitions for categories and functors, and functions defined as the behavior of interacting sets.
http://www.math.lsa.umich.edu/~ablass/xst11.pdf

[33] Childs, D. L.: *Functions Defined By Set Behavior* A Formal Foundation Based On Extended Set Axioms Abstract: The term *function* seems to connote a sense of action or process or behavior of something applied to something. Within the framework of extended set theory, XST, the concept of a *function* will be defined as a behavior of sets in terms of how specific sets react subject to their interaction with other sets. In particular, $\mathbf{f}_{(\sigma)} : \mathbf{A} \rightarrow \mathbf{B}$ will assert that the set '$\mathbf{f}$' behaves as a function under set '$\sigma$' in relating an individual member of the function domain, set '$\mathbf{A}$', to exactly one member of the function codomain, set '$\mathbf{B}$'. It will be shown that all Classical set theory, CST, graph based function behavior can be expressed in terms of XST function non-graph based behavior; that the behavior of functions applied to themselves is supported; and that the concepts of Category theory can be subsumed under XST. A notable consequence of this approach is that the mathematical properties of functions need no longer be dependent on the mathematical properties of a Cartesian product.
http://xsp.xegesis.org/Xfun.pdf

[34] Childs, D. L.: *Relations Recast for Cloud Computing Productivity* Abstract: When Relations for data modeling and data access were introduced in 1970 classical set theory was the only mathematical foundation available. This paper discusses why the Relational Data Model was so potent - not because of classical set theory, but inspire of it. Had Codd had the mathematical foundation he needed *null values*, *normalization* would not have been needed and data *updates* would have been procedural instead of structural.
http://xsp.xegesis.org/WNSETS.pdf

[35] North, K.: *Sets, Data Models and Data Independence*, 2010
A set-theoretic data structure (STDS) is virtually a 'floating' or pointer-free structure allowing quicker access, less storage, and greater flexibility than fixed or rigid structures that rely heavily on internal pointers.
http://drdobbs.com/blogs/database/228700616

[36] TPC-H Specifications & Top Ten
http://www.tpc.org/tpch/spec/tpch2.8.0.pdf
http://www.tpc.org/tpch/results/tpch_perf_results.asp

[37] TPC-H Microsoft
SQL Server 2008 R2 Enterprise Edition [12/07/11]
Executive Summary:
http://www.tpc.org/tpch/results/tpch_result_detail.asp?id=111120801&layout=
Full Disclosure:
http://c970058.r58.cf2.rackcdn.com/fdr/tpch/Cisco-MicrosoftC460TPC-H1000GB_FDR.pdf

[38] TPC-H Oracle
Database 11g R2 Enterprise Edition w/ Partitioning [09/26/11]
Executive Summary:
http://www.tpc.org/tpch/results/tpch_result_detail.asp?id=111092601&layout=
Full Disclosure:
http://c970058.r58.cf2.rackcdn.com/fdr/tpch/Oracle_T4-4_1TB_TPCH_FDR_092611.pdf

[39] TPC-H IBM
DB2 UDB 8.2 [02/14/05]
Executive Summary:
http://www.tpc.org/tpch/results/tpch_result_detail.asp?id=105021401&layout=
Full Disclosure:
http://c970058.r58.cf2.rackcdn.com/individual_results/IBM/x346.linux.1TB.051219.es.pdf