# SET THEORETIC DATA STRUCTURES

***A review paper by:***

1.Nishil Hoogar

2. Leo Barros

3.Varad Kelkar

4.Vaughan D’Souza

5. Isaiah D’costa

6. Joshua Coutinho

7.Duane Rodrigues

**ABSTRACT:**

The data base literature has paid attention to Extended Set Theory as a data base management discipline. Set Theoretic Information Systems Corporation has been selling a database system based on Extended Set Theory for some time. Set Theoretic Data Structures is the name of this system (STDS). STDS is similar to relational algebraic data base management systems, as demonstrated by a series of examples. The benefits of STDS include its simple data base architecture, compact data representation, and flexible, powerful data manipulation operators; nevertheless, the low-level basic user interface and incomplete implementation of Extended Set Theoretic principles are its drawbacks. A "user-friendly" interface and certain specific extended Set Theory capabilities should be created to make STDS particularly appealing.

*Keywords: Union,intersection,symmetric difference,domain,range,image,cardinality,converse image,concurrence:domain,range,set.*

## 1.INTRODUCTION:

The ultimate purpose of this project, of which this work is a part, is to create a machine-independent data structure that allows for quick processing of data linked by arbitrary relationships such as: such as: the contents of a telephone book, library files, census reports, family lineage, graphic displays, information retrieval systems, networks, etc. Non-intrinsically connected data must be expressed (stored) in such a way that the relationship between them can be defined before any data structure can be applied. A set-theoretic data structure looks to be worth investigating since each relation can be described in set theory as a set of ordered pairs, and set theory provides a richness of operations for dealing with relations.

A Set-Theoretic Data Structure (STDS) is any storage representation of sets and set operations that is isomorphic to **η** with S, given any family of sets t and any collection S of set operations. Any set-theoretic statement capable of being constructed from **η** and S may be used with an STDS. Every stored representation of a set must keep all of the set's properties, and every representation of a certain set must behave identically while performing set operations.

General storage representation

An STDS is made up of five structurally distinct components:

1. A collection of set operations S.

2. Asset of datum names **β**.

3 .The data: a collection of datum definitions, one for each datum name.

4. A collection of set names **η**.

5. A collection of set representations, each with a name in **η**.

The set operations must be executed early in order for an STDS to be effective. If any two sets can be well ordered (a linear order with a first element) and their union preserves this well ordering, then the set operations subroutines are simply a type of merge or, at worst, a binary search of just one of the sets. Another paper showed that any set defined over **η** may be ranked in this way. Sets are represented by contiguous storage locations in blocks, with t containing the names of all the sets. The set 0 is a contiguous block of storage locations that contains all datum names; the address of a location in the **β**-block is a datum name and an element of **β** (as shown in Fig 1). The address of a stored description of that datum is the content of a location in the **β**-block. The only pointers required for the execution of an STDS are the contents of the **β**-block and the **η**-block. Individual set storage representations do not carry pointers to other sets, but they do store information on datum names. Because each set representation is associated with only one pointer, it can be relocated throughout storage without affecting its contents or the contents of any other set representations—only the one pointer in **η** is affected. It's a simple task to maintain set representations current. The last element in the set is used to replace elements that are to be deleted.

As space allows, elements to be added are added to the end of the set representation. When contiguous locations become unavailable, a new set is created, and the element in **η** that previously referenced the set now references a location that signals that the set is now the union of two set representations. (In a paging structure such sets could be kept on the same page.) In the case of **η**, there are two types of sets: generator sets and composite sets. The composite sets are unions of generator sets, and the generator sets are mutually disjoint. Only the generator sets have storage representations. An STDS is intrinsically a minimal storage representation for arbitrarily connected data since no duplication of set storage is required and set representations are maintained to a minimum by having only the components of the sets and no pointers.

## 2.OPERATIONS OF AN STDS

Operation on an STDS can be done by means of set operations often encompassed in a subroutine (sequence of program instructions that performs a specific task). This set operation accesses the required sets through pointers. It should be stressed that no pointers exist between sets, hence the set operations act as the only structural ties between sets.

The STDS package itself consists of these set operations that we can readily use. The operations that manipulate these sets can be categorised as follows:

1. I/0 operations

2. Set operations

3. Arithmetic operations

4. Utility functions

Given below are examples of some functions that are available in the STDS package.

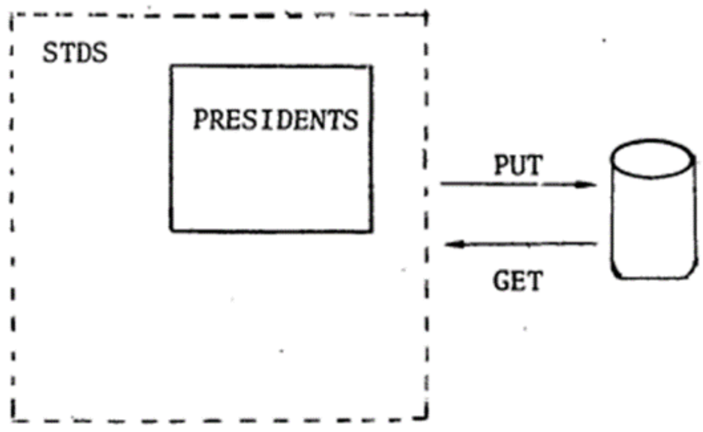
| I/O and Utility | Set Operations |
|---|---|
| GET | UN |
| PUT | XPAN8 |
| SETFMT | RMIX8 |
| LIST | XSET |
| QSFILE | RS8 |
| QRETURN | LEGL |

A few commonly used STDS functions have been illustrated below with an example of a database that stores a list of the Presidents of the United States.

## 2.1 GET and PUT - Retrieve Sets from Archival Storage (i.e., archive sets)

The GET operation retrieves an already stored set (table) from secondary storage. The set (table) must have previously been stored by a PUT operation. The PUT operation stores a set on secondary storage (hard drive).

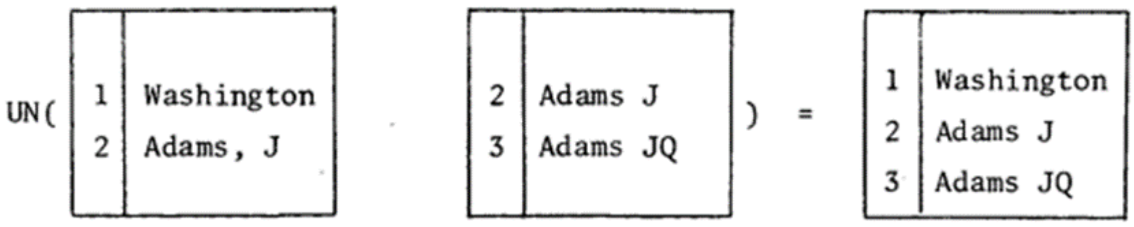
Example



## 2.2 UN - Produce the Union of Two Sets Given two sets

A resultant set is constructed that contains each row belonging to either input set. The resultant set does not contain duplicates.

Example


UN(

| | |
|---|---|
| 1 | Washington |
| 2 | Adams, J |

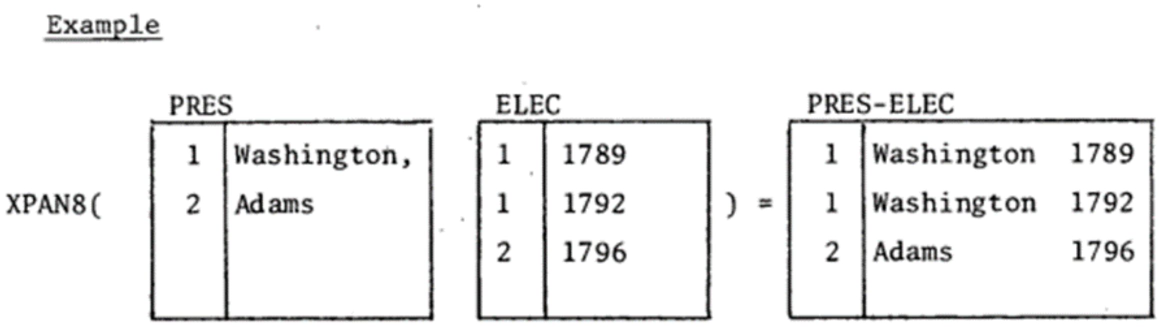
| | |
|---|---|
| 2 | Adams J |
| 3 | Adams JQ |

) =

| | |
|---|---|
| 1 | Washington |
| 2 | Adams J |
| 3 | Adams JQ |

## 2.3 XPAN8 - Expand to Sets (JOIN)

XPAN8 compares the first domains of two inputs, and where a match occurs, constructs a resultant set containing the concatenation of the data fields from the two input sets:


Example

XPAN8( PRES , ELEC ) = PRES-ELEC

PRES

| | |
|---|---|
| 1 | Washington, |
| 2 | Adams |

ELEC

| | |
|---|---|
| 1 | 1789 |
| 1 | 1792 |
| 2 | 1796 |

PRES-ELEC

| | | |
|---|---|---|
| 1 | Washington | 1789 |
| 1 | Washington | 1792 |
| 2 | Adams | 1796 |

It is to be noted that these operations are performed rapidly, thereby providing increased read and write speed of the data structure.

## 3. BLOCK DETAILS:

### 3.1 □-block:

The □-block is a section of contiguous storage locations with □o as the address of the head location. The first location containing a datum-pointer has the address □o+1, and the location of the i-th datum-pointer is □o+i. Let #□ be representing the total number of datum-pointers, then the last address of the □-block would be □o+#□. Since all datum-pointers are located between □o+1 and □o+#□, let □ be set of integers {1,2,3,...,#□}. Therefore any integer i such that 1<=i<=#□ is the datum-name for the i-th datum-pointer. The i-th datum-pointer locates a block of storage containing a description of the i-th datum and all the generator set names for which the i-th datum-name is constituent

### 3.2 η-block:

The η-block is similar to □-block with ηo and #η as the address of the head location and cardinality respectively. The contents of the η-block are pointers. These pointers are of two types and are distinguished by an integer η* such that 1<η*<=#η. For all 1<=i<η*, i is the name of a generator set, and for all η*<=i<=#η, i is a composite set. A generator set has a set representation while a composite set does not since it is the union of some generator sets. For i>=η* the pointer-in ηo+i locates a section of storage containing names of generator sets. For i<η* the pointer in ηo+i locates a section of storage containing all composite set names that use i, and a pointer to the set representation of i. Since all generator ste sare mutually disjoint and since only generator sets have a storage representation, there is no duplication of storage in an STDS.

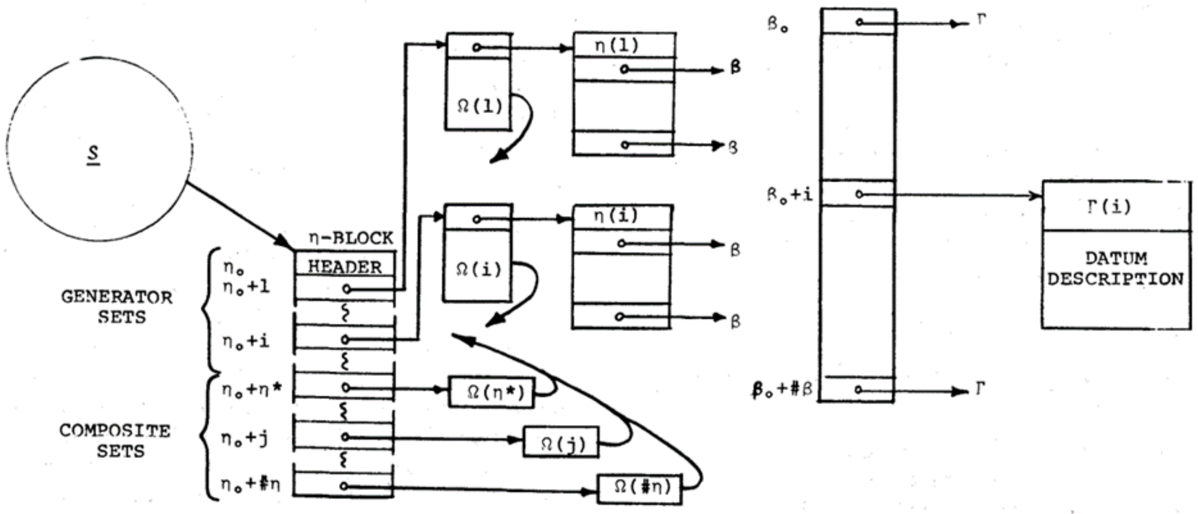


Figure 1:Block Representation

Ω(I) through Ω(ŋ*-I) are sets of pointers which point back to composite sets in η-block, while, Ω(ŋ*) through Ω(#ŋ) are sets of pointers which point to generator sets in η-block. *Γ*(i) are sets of pointers to generator sets in η-block

η-block is made up of generator sets [1<=i<η*] and composite sets [η*<=i<=#η]. Generator sets having set operations(S), while composite sets being the union of some generator sets
Set operations has pointers pointing to the datum names ☐. These datum names, a part of ☐-block points to datum description *Γ*(i). Where *Γ*(i) are sets of pointers to generator sets in η-block

## 4.SET REPRESENTATION

The Sets involved in set operations are isomorphic and have a unique linear representation of their elements in order to ensure faster execution speeds. Isomorphic Sets: a one-to-one mapping between two sets that preserves relationships between elements of the sets
Example:: (1,2)<---->(1) (2)

To Ensure efficient operation of an STDS: 1. The elements have some predefined well-ordering relation, such that independently of how the set is presented to a machine, the ordering of its elements will always be the same. 2.This ordering should be preserved under a union. As the Set Representatives are isomorphic,every set representation will reflect the rank of set and preserve the order of the set.
Let A = <a,b,c>, B = {a, b,c}, and C = {c,b,a} From above, we analyse that Sets B and C have the same set representation

while A is different.Hence we can conclude saying that B and C are the correct way of representing Sets.

## 5. COMPLEXES AND N-TUPLES

### 5.1 Complexity in set theory

What is complexity?
In its most general form, the notion of complexity seems to be
identifiable with that of a pre order (a reflexive, transitive relation):
$A \leq B$
Means: A is at most as complex as B (whatever A and B are)

The point is to identify a good way of comparing A and B, in order to
state that $A \leq B$

### 5.2 n-tuples in set theory

Extended Set Theory was developed by David L. Childs with support from the CONCOMP project at the University of Michigan.Childs realised that computer data structures did not have a rigorous mathematical formulation, and began to develop a definition that would ultimately lead to practical results when applied to the computer environment.

To achieve a mathematical definition of computer data structures,
In particular those used in databases, Childs investigated classical set theory.

The choice of classical set theory was a natural first step, because a database record might be viewed as an n-tuple where each field in the record represents a domain of the n-tuple. However, classical set theory has some definite shortcomings when applied to records and databases. These problems arise because of the definition of the n-tuple.

In mathematics, a tuple is a finite ordered list (sequence) of elements. An n-tuple is a sequence (or ordered list) of n elements, where n is a non-negative integer. There is only one 0-tuple, referred to as the empty tuple. An n-tuple is defined using the construction of an ordered pair.

A standard classical set theoretic definition of the ordered pair
(2-tuple) is:

$<x, y> = \{\{x\}, \{x, y\}\}$

This definition is extended to n-tuples in a straightforward manner:

$<x,y,z> \ = \ \{\{x\}, \{x,y\}, \{x,y,z\}....\}$

When this definition is applied 'to computer data structures, this definition quickly breaks down:

$<1,0,1>=\{\{1\},\{1,0\},\{1,0,1\}\}=\{\{1\},\{1,0\}\}=<1,0>$

Certain obvious classical set theoretic operations on n-tuples are undefined, largely because of the definition of the n-tuple.

To achieve a better foundation for computer data structures, Childs developed a new definition of a set. The set E is defined as: where $a_j$ is an atom of a set and $i_j$ is a position indicator.

$$S=\{ \begin{matrix} i_1,i_2,i_3.....i_n \\ a_1,a_2,a_3....a_n \end{matrix} \}$$

Note than an n-tuple is now just a special case of a set:

$<1,0,1,2>=\{1^1,0^{2,}1^3,2^4\}$

The advantages of the definition of the set:
1.There is no problem distinguishing between like elements with different positions.
2. n-tuples need not be considered specially; n-tuples are special cases of sets.
3. All classical set operations may be defined on these sets.
4. New operations on these sets may be defined.

**6.**

DEFINITION SCHEMA: $\{x^i{:}\Psi(x,i)\} = A$ iff $[(\forall x)$ $(\forall i \in N)\ (x \in_i A \leftrightarrow \Psi(x,i))$ & A is a complex$]$

These findings provide a theoretical basis for the following equivalent notations:

$$\begin{array}{lrcl}
\text{set} & \{a,b,c\} & = & \{a^1,b^1,c^1\} \\
\text{ordered pair} & <a,b> & = & \{a^1,b^2\} \\
\text{ordered triple} & <a,b,c> & = & \{a^1,b^2,c^3\} \\
\text{ordered quadruple} & <a,b,c,d> & = & \{a^1,b^2,c^3,d^4\} \\
\text{ordered pairs of ordered pairs} & & & \\
 & <<a,b>,<c,d>> & = & \{\{a^1,b^2\}^1,\{c^1,d^2\}^2\} \\
 & <a,<b,<c,d>>> & = & \{a^1,\{b^1,\{c^1,d^2\}^2\}^2\} \\
 & <a,<<b,c>,d>> & = & \{a^1,\{\{b^1,c^2\}^1,d^2\}^2\} \\
 & <<<a,b>,c>,d> & = & \{\{\{a^1,b^2\}^1,c^2\}^1,d^2\} \\
 & <<a,<b,c>>,d> & = & \{\{a^1,\{b^1,c^2\}^2\}^1,d^2\} \\
\multicolumn{2}{r}{\{<1,a>,<2,b>,<3,c>,<4,d>\}} & = & \{\{1^1,a^2,\},\{2^1,b^2\}, \\
 & & & \{3^1,c^2\};\{4^1,d^2\}\}
\end{array}$$

and from the beginning of this section,

$$W = \{a^1,b^1,\{\{c^1\}\},\{a^1,\{b^1,d^1\}^2,c^3\},\{\{a^1,b^2\},c^1\}\}$$

$$V = \{\{a^1,b^2,c^3\},\{\{\{a^1,b^2\},\{c^1,d^2\}\},\{d^1,a^2\}^2\},\{\{c^1\}\},b^1\}$$

Since for all a,{a1 } = {a}, the exponent '1 ' is optional. It is important to note that the sign 'x*' has no meaning except to be enclosed by set brackets. If A={$a^{6,}b^{8}$ },then a$\epsilon_6$A and b$\epsilon_8$A are true, but $a^6\epsilon$A is meaningless. Refer to figure 2 for more examples.

1) $<a,b,c>\cap<x,b,y> = \{b^2\}$

2) $<a,b,c>\cup<x,y> = \{a^1,x^1,b^2,y^2,c^3\}$

3) $\{a,b,c\}\cap<a,x,y> = <a> = \{a^1\} = \{a\}$

4) $\cup\{a^1,b^2,\{x^1,c^3\}^3,\{y^2,d^4\}^4\} = \{x^1,c^3\}\cup\{y^2,d^4\} = <x,y,c,d>$

5) $<a,b,z>\Delta<a,y,c>\Delta<x,b,c> = <x,y,z>$

6) $<a,b,c,d> \sim <x,y,c,d> = <a,b>$

FIGURE 2—Set operations between complexes

## 7.CONCLUSION

STDS provides a user with a very flexible and extremely powerful tool with which complex manipulation of data can be performed rapidly.

STDS-I:does not provide the user with a sufficiently"user-friendly" interface to allow easy work with a data base.
Reason:Because of its low-level nature

STDS-OS:User Friendly But currently has insufficient power to handle relationships between sets.

Features to be incorporated:
1. A "user-friendly" interface be developed
2. Distinctive features of the Extended Set Theory be implemented.

It is a system that appears to be of interest of relational database management systems that could possibly provide a more efficient means of implementing a relational system.

## 8.REFERENCES:

*Description of a set-theoretic data structure* by David L. Childs