IMPLEMENTATION OF DATA ABSTRACTION IN THE RELATIONAL DATABASE SYSTEM INGRES

James Ong
Bell Laboratories
Crawfords Corner Road
Holmdel, New Jersey 07733

Dennis Fogg
University of California
Berkeley, California 94720

Michael Stonebraker
University of California
Berkeley, California 94720

ABSTRACT

This paper discusses the design and implementation of an *abstract data type* (*ADT*) *facility* which was added to the INGRES database manager. Our implementation of ADTs allows a user to register ADTs and ADT operators with the run-time database manager, declare column values of relations to be instances of ADTs, and formulate queries containing references to ADTs and ADT operators. The user view, implementation, performance, and possible extensions to this new facility are described.

1. INTRODUCTION

In recent years, much attention has been paid to the feasibility of adding data abstraction capabilities to a database manager. Many database query languages such as RIGEL [ROWE 79,81], PLAIN [WASS 81], DAPLEX [SHIP 81], FQL [BUNE 79], and INGRES [OVER 81] already support some type of data abstraction facility.

This paper will discuss the design and implementation of an *abstract data type (ADT) facility* which was added to the INGRES database manager. ADTs have been explored extensively in a programming language context [LISK74, GUTT77]. Basically, an ADT is an encapsulation of a data structure and a set of associated operators that may access the data structure. Because the implementation details of the data structure and the operators are hidden, the user may write implementation-independent code which references high-level, user-defined data objects that are specific to the application.

Our implementation of ADTs allows a user to register ADTs and ADT operators with the run-time database manager, declare column values of relations to be instances of ADTs, and formulate queries containing references to ADTs and ADT operators. This paper will discuss the user view, implementation, performance, and possible extensions to this new facility.

2. USER VIEW

Before a user can enter queries containing ADTs and ADT operators, he must first:

- Register the ADTs and ADT operators with the database manager,
- Code and compile the user-written functions and operators associated with the ADTs, and
- Declare columns of a relation to store instances of the defined ADTs.

2.1 Registration of ADTs

Each ADT possesses an *internal representation* and an *external representation*. The internal representation is a user-defined data structure that represents the ADT value when it is stored and manipulated in the database system. The external representation is a character string that represents the ADT value in a form that is easily understood by a person. INGRES displays ADT values to the user in this form. Conventional data types also possess two representations: for example, the internal representation of an integer on a VAX is a 32-bit two's complement format while the external representation is a character string of digits.

As an example, we created an ADT called "complex" so that complex numbers could be treated as a data type. The internal representation was sixteen bytes long; eight bytes each were used to store the real and imaginary parts in double precision floating point format. The external representation consisted of two character strings representing the two values, separated by a comma. Therefore, a valid external representation of the complex value 3+4i could be "3,4".

To convert between internal representations and external representations, the definer of an ADT must supply two conversion routines. INGRES invokes the external-to-internal conversion routine to translate ADT values entered by the user into internal representations. In addition, INGRES invokes the internal-to-external conversion routine when displaying the results of a query to a user.

To register an ADT with INGRES, the names of these two routines must be supplied in a DEFINE ADT command. The complete specification of an ADT includes the following:

- The name of the ADT (TYPENAME),
- The maximum length of the internal (byte) representation in bytes (BYTESIN),
- The maximum length of the external (string) representation in bytes (BYTESOUT),
- The name of the user-written C function that converts external representations to internal representations (INPUTFUNC), and
- The name of the user-written C function that converts internal representations to external representations (OUTPUTFUNC),
- The name of the file that contains the conversion functions (FILENAME).

Type "complex" was defined by entering:

```
DEFINE ADT (TYPENAME   IS "complex",
            BYTESIN    IS 16,
            BYTESOUT   IS 27,
            INPUTFUNC  IS "tointernal",
            OUTPUTFUNC IS "toexternal",
            FILENAME   IS "/ja/guest/fogg/complex")
```

An UNDEFINE ADT command was also added to unregister an ADT from the database manager.

2.2 Declaration of ADT Attributes

Once the ADT has been registered, a user may define columns of a relation to contain ADT values. For example, to CREATE a relation called "ComplexNum" with fields of type "complex" and float, one may enter :

```
CREATE ComplexNums (field1 = ADT:complex, field2 = f4)
```

where "ADT:" specifies that the field is an instance of an abstract data type, and "complex" specifies the name of the ADT.

2.3 Registration of ADT Operators

The INGRES ADT facility supports unary and binary ADT operators. The operands and result may be standard INGRES data types (e.g., integers, floats, character strings) or user-defined data types (e.g., ADT "complex"). Thus, one may use the INGRES ADT facility to define new operators on standard INGRES data types as well as operators on new types. For example, INGRES does not support a built-in cube-root function, so a user could add one using the ADT facility.

Each ADT operator possesses a name and a precedence level. The name is used to reference an operator within a query. For instance, the addition operator has the name "+". The precedence level specifies how queries with multiple operators should be interpreted. For example, the precedence of "*" is automatically higher than that of "+", so the expression a + b * c is interpreted as a + (b * c). All ADT operators must be given a precedence level.

Each operator is implemented by a user-written C function. When INGRES needs to apply an ADT operator to its operands, it calls the function and passes pointers to the operands and to the buffer where the result value should be placed. The user-written function must be designed to accept this communication protocol.

Consequently, the following information must be entered into INGRES to define an ADT operator:

- The operator name as it appears in a query (OPNAME),
- The name of the user-written C function that implements the operator (FUNCNAME),
- The name of the file containing the C function (FILENAME),
- The types and lengths of the operands (ARG1 and ARG2) and the result (RESULT).
- The precedence of the operator - (PRECLEVEL) if binary. This may be any of the four precedence levels used by built-in

INGRES operators. Unary operators are assumed to have the highest precedence.

The DEFINE ADTOP command was added to allow a user to register an ADT operator with INGRES. For example, the declaration of a binary ADT operator that returns the complex product of two numbers might be written as:

```
DEFINE ADTOP (OPNAME    IS "*",
              FUNCNAME  IS "complexProduct",
              FILENAME  IS "/ja/guest/fogg/complex",
              RESULT    IS ADT:complex,
              ARG1      IS ADT:complex,
              ARG2      IS ADT:complex,
              PRECEDENCE LIKE "*")      /* like INGRES * */
```

The declaration of a unary ADT operator that yields the magnitude of a complex number might be written as:

```
DEFINE ADTOP (OPNAME    IS "Magnitude",
              FUNCNAME  IS "magnitude",
              FILENAME  IS "/ja/guest/fogg/complex",
              RESULT    IS f8,
              ARG1      IS ADT:complex)
```

The name of an ADT operator may be unique, or it may be the same as a built-in INGRES operator.

2.4 Use of ADT Expressions Within Queries

After the above steps have been completed, queries may be formulated to execute operations on ADT values. For example, a query which retrieves all complex numbers whose magnitude is greater than the magnitude of the complex number 3+4i may be written as :

```
RANGE OF C IS ComplexNums
RETRIEVE (C.field1)
        WHERE Magnitude C.field1 > Magnitude "3,4"
```

where "Magnitude" is the ADT unary operator defined above. Because "Magnitude" accepts an operand of type "complex", "3,4" is interpreted as the external representation of an ADT rather than as

a character string. Execution of the above query might yield:

```
|field1                          |
|--------------------------------|
| 3.000000e+00, 5.000000e+00     |
|-4.000000e+00,-4.000000e+00     |
|--------------------------------|
```

3. IMPLEMENTATION

A copy of INGRES was modified in fourteen man-weeks to support abstract data types. Changes were needed primarily in the scanner, the parser, and the One-Variable Query Processor (OVQP).

The scanner and parser accept a query entered by the user and generate a tree which INGRES uses internally to represent a parsed query. The scanner and parser modules were modified to generate trees for queries containing references to ADTs and ADT operators. A multi-relation query is decomposed into a sequence of one-relation commands by module DECOMP. This module required no changes. Module OVQP accepts a one-relation query tree and executes all retrievals and modifications to the relation specified. OVQP was modified to accept and execute query trees containing references to ADTs and ADT operators.

Moreover, two new system relations, "adt" and "adtoperator", were created to store information about user-defined ADTs and ADT operators. INGRES frequently accesses information stored in these relations when processing each query, so LRU software caches were created to speed up the access time. Four utilities were written to support the commands DEFINE ADT, UNDEFINE ADT, DEFINE ADTOP, and UNDEFINE ADTOP described in section 2.

Application of ADT operators requires INGRES to invoke the appropriate user-written C functions. Several possibilities for providing access to these C functions were considered. Static linking of the functions with the INGRES program was rejected because it would require INGRES to be re-linked each time an ADT operator is registered. The size of INGRES would also continue to grow with the addition of each new ADT. Running the user-written functions as separate processes was rejected because of the high overhead incurred when passing arguments and result values via inter-process communication. As a result, a simple dynamic linker was written to provide run-time access to user-written routines.

4. PERFORMANCE

In this section, the original copy of the INGRES program will be referred to as "Standard-INGRES", and the modified copy will be referred to as "ADT-INGRES."

4.1 Performance of ADT-INGRES When Running a Standard Query

For the first comparison, both programs executed:

```
Query-A:    RANGE OF R is RealpImagp
            RETRIEVE (SQRT(R.Realp**2 + R.Imagp**2))
               WHERE R.Realp = <const>
               AND R.Imagp = <const>
```

where relation "RealpImagp" contained 25,600 tuples, each with two 8-byte floating point attributes, "Realp" and "Imagp". This query computes the magnitude of a complex number when stored as two floating point numbers using standard INGRES data types. Table One presents the results.

Standard-INGRES/Query-A vs. ADT-INGRES/Query-A

Access Method	*Standard-INGRES Times (secs)*			*ADT-INGRES Times (secs)*		
	User	*System*	*Elapsed*	*User*	*System*	*Elapsed*
Hashed	0.67	0.16	0.85	1.10	0.36	1.57
Indexed	0.72	0.19	0.98	1.14	0.36	1.52
Heap	28.34	2.88	33.17	30.16	2.69	35.73

Table One

Table One shows that ADT-INGRES consumes nearly twice the CPU time required by STANDARD-INGRES when running QUERY-A for hashed relations, but consumes only about 5 percent more when running the same query for heap relations. Thus, ADT-INGRES appears to have a substantial per-query overhead incurred in the scanner and parser.

Even when running identical queries on identical relations, ADT-INGRES runs more slowly for several reasons. First, each time the scanner encounters an unquoted character string in a query, it must search the "adtoperator" relation to check if the string is actually the name of an ADT operator. The "adt" and "adtoperator" caches were designed to minimize the average successful search time for an "adt" or "adtoperator" tuple. Instead, the average search time for all retrievals, successful and unsuccessful, should have been minimized.

One solution would be to load the "adt" and "adtoperator" relations into a hashed table in the INGRES address space. Although this method would greatly speed access to ADT and ADT operator information, it would also limit the practical sizes of the "adt" and "adtoperator" relations. Another solution would be to restrict the set of names an ADT operator may legally have. This would allow INGRES to eliminate most character strings as potential ADT operators without accessing the "adtoperator" relation.

Second, ADT-INGRES runs more slowly because the overloading of operator names complicates the parsing of queries. The ADT facility allows names of arithmetic operators (e.g., "+", "*") and relational operators (e.g., "<", "!=") to be legal ADT operator names. To support this capability, the INGRES parser must be able to pattern match a query expression such as:

```
R.field1 * R.field2
```

to determine from context whether the "*" operator is a standard INGRES arithmetic operator or an ADT operator.

4.2 Performance of ADT-INGRES When Running an ADT Query

The second comparison measured the relative speeds of the programs when ADT-INGRES utilized ADTs to execute a simpler but functionally equivalent query. Thus, the executions times of Standard-INGRES running Query-A were compared with execution times of ADT-INGRES running Query-B:

```
QUERY-B:   RANGE OF C IS ComplexNums
           RETRIEVE (Magnitude (C.field1))
              WHERE C.Complex = "<const>,<const>"
```

Here, "Magnitude" is the ADT operator described above that returns the magnitude of a complex number.

Execution of queries containing ADTs and ADT operators causes ADT-INGRES to incur overhead when accessing user-written functions that support the ADT operators and ADT conversion functions. ADT-INGRES uses a simple but inefficient dynamic linker that overlays a user-written object file into a buffer area within the INGRES address space when needed. If a query references a function stored in a file not resident in the buffer area, INGRES must open the required object file, read it into the buffer area, read the file's symbol table to locate the needed function, and then close the file.

Thus, the performance of ADT-INGRES greatly depends upon whether or not the buffer already contains the needed user-written functions. We measured ADT-INGRES's execution time when running Query-B in two situations. Query-B(fast) is a best case where the buffer always has the required functions at the start of the query. Query-B(slow) is a pessimistic case where the object file must be read into the buffer area for each query. Tables Two and Three present these results.

Standard-INGRES/Query-A vs. ADT-INGRES/Query-B(slow)

Access Method	*Standard-INGRES Times (secs)*			*ADT-INGRES Times (secs)*		
	User	*System*	*Elapsed*	*User*	*System*	*Elapsed*
Hashed	0.67	0.16	0.85	0.91	0.30	1.72
Indexed	0.72	0.19	0.98	0.98	0.36	2.04
Heap	28.34	2.88	33.17	35.66	2.98	44.40

Table Two

Standard-INGRES/Query-A vs. ADT-INGRES/Query-B(fast)

Access Method	*Standard-INGRES Times (secs)*			*ADT-INGRES Times (secs)*		
	User	*System*	*Elapsed*	*User*	*System*	*Elapsed*
Hashed	0.67	0.16	0.85	0.79	0.32	2.01
Indexed	0.72	0.19	0.98	0.81	0.36	1.52
Heap	28.34	2.88	33.17	35.46	2.96	41:40

Table Three

Query-B(slow) consumes an additional one or two tenths of a second of CPU time as compared with Query-B(fast). This difference is due to the additional work needed to load in a user-written object file and dynamically link the functions.

ADT-INGRES running Query-B(fast) is about 15 percent slower than Standard-INGRES running Query-A for heap relations. Since a query on a heap storage structure causes the "Magnitude" operator to be applied to every tuple in relation ComplexNums, we expect that this performance degradation is caused by inefficient access to dynamically-linked, user-written functions. Because there exist many possibilities for optimizing ADT-INGRES, we expect that judicious tuning can largely erase its performance penalty.

5. FUTURE WORK

This section proposes several extensions to the INGRES abstract data facility that would increase its power and performance.

5.1 Integrating ADTs into INGRES Query Processing

The INGRES ADT facility does not support the indexing of ADT fields except for the special case of the equality operator. Moreover, query processing heuristics cannot effectively optimize multi-relation commands containing ADT operators. Two classes of extensions to the ADT facility would overcome these drawbacks.

The first extension would allow the user to control the sorting of tuples in indexed relations. INGRES currently sorts tuples in

indexed relations into collating sequence. However, this ordering may or may not be useful. For example, consider a possible ADT called "timeOfDay" which stores precise time of day values using the following internal representation:

```
hour   -      2 byte integer
minute -      2 byte integer
second -      4 byte float
```

Sorting these fields in collating sequence will not result in chronological order for events occurring within the same second, so a user-supplied sorting function is needed.

More generally, it would be desirable to support the inclusion of new access methods appropriate for user-defined types. For example, in spatial applications, two-dimensional access methods such as KDB trees [ROBB 81] and bin structures may be reasonable.

The second class of extensions concerns multi-relation query processing heuristics. For example, consider a new operator "^" which operates on a pair of complex numbers and returns true if one is the negative of the other. For example, (3,4i) is the negative of (-3,-4i). One could find all pairs of numbers in ComplexNums which are negatives of each other as follows:

```
Range of C is ComplexNums
Range of C2 is ComplexNums
Retrieve into W (C.field1, C2.field1) where
     C.field1 ^ C2.field1
```

Two extensions are required to process multi-relation commands. First, one must make an estimate of the size of W. This is required for three-way joins in order to evaluate the cost of alternate processing orders. Current algorithms (e.g., [SELI 79]) have built-in functions to compute such sizes. One needs to call a user-defined function to obtain this information for ADT operators. A possible syntax would be to extend the DEFINE ADTOP command with an additional field:

```
statistics is my-stat
```

where the function my-stat could optionally be defined for any binary operator which had a boolean result type. When passed a constant, it would return the selectivity of the clause:

```
... where C.field op  constant
```

Alternatively when passed nothing, it would return the selectivity

of the clause

```
... where C1.field1 op C2.field1
```

The second required extension concerns merge-sort. It is always possible to process the above query by iterative substitution; it is sometimes possible to use merge-sort. To use merge-sort with the conventional equality operator, one must first sort each relation into collating sequence using the operator "<". For the operator "^" there is an ordering compatible with the merge-sort, and it is defined by the operator

```
C1.field1 << C2.field1 iff

     abs (C1.realPart)     abs (C2.realPart)
     ----------------- < -----------------
     abs (C1.imagPart)     abs (C2.imagPart)
```

Here "abs" takes the magnitude of a floating point number. The second extension to the DEFINE ADTOP command is an optional clause:

```
merge-sort is <<
```

With this additional information, a heuristic optimizer can make an intelligent choice of plans.

5.2 Hierarchies of ADTs

It would be very useful to be able to specify that a data type inherit the characteristics of a previously defined data type [STON 82]. This capability would allow ADT-A to be defined as a subset of ADT-B, thereby inheriting all of ADT-B's operators. ADT-B would then be the "superset ADT" of ADT-A, and ADT-A would be one of the "subset ADTs" of ADT-B. For example, assume that ADT "dog" has been defined to be a subset of the ADT "mammal." An ADT operator such as "age-of", originally defined upon ADT "mammal", would then be applicable to values of type "dog."

Hierarchies of ADTs would require INGRES to maintain another system

relation that stores subset-ADT/superset-ADT pairs. For example:

Relation: ADT_Hierarchies

Superset-ADT	Subset-ADT
Mammal	Dog
Mammal	Cat
Dog	Poodle

This relation would be searched by the INGRES routines to type-check ADT operator expressions within queries. For example, if ADT operator-A has been defined on ADT-A, this operator may be applied to a value defined as an instance of ADT-B if and only if:

1. ADT-A is identical to ADT-B, or

2. ADT-B is in the transitive closure of SUBSET-OF ADT-A.

User-written functions that implement ADT operators could be correctly applied to subset-ADT values because INGRES calls these functions with *pointers* to the operands. To insure that the function operates correctly on the subset ADT values, the first n bytes of all subset-ADT internal representations must have the same format as the superset's n-byte internal representation. For example, assume that ADT "mammal" possesses an n-byte internal representation, ADT "dog" possesses an n+m byte representation, and ADT operator "age-of" has been defined to return a mammal's age given the mammal's n-byte internal representation. ADT operator "age-of" may be correctly applied to values of type "dog" if the first n bytes of types "dog" and "mammal" have the same format.

6. CONCLUSIONS

This paper has discussed the user-view, implementation, and performance of abstract data types within the relational database system INGRES. Our implementation allows a user to register abstract data types and abstract data type operators with the run-time database manager. Column values of a relation may be defined as instances of abstract data types, and ADT operators may be applied to these values. Some possible extensions were proposed to extend the power and performance of the INGRES ADT facility.

REFERENCES

[BUNE 79] Bune, P. et. al., *FQL - A Functional Query Language*, Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, PA. 1979.

[FOGG 82] Fogg, D., *Implementation of Domain Abstraction in the Relational Database System INGRES*, Master of Science

Report, Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA., September 1982.

[GUTT 77] Guttag, J., "Abstract Data Types and the Development of Data Structures," *CACM*, June 1977.

[HUNG 82] Hung, D., *Using a Relational DBMS to Store Symbol Table Information for Separate Compilation*, Master of Science Report, Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA., August 1982.

[LISK 74] Liskov, B. and Zilles, S., "Programming with Abstract Data Types," *ACD-SIGPLAN Notices*, April 1974.

[MUEL 82] Mueller, F., *Artificial Intelligence and Data Base Applications*, Master of Science Report, Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA., September 1980.

[ONG 82] Ong, J. C., *Implementation of Abstract Data Types in the Relational Database System INGRES*, Master of Science Report, Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA., September 1982.

[OVER 81] Overmyer, R., *A Time Expert for INGRES*, Master of Science Report, Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA., July 1981.

[ROBB 81] Robinson, J. T., "The K-D-B-Tree: A Search Structure for Large, Multi-dimensional, Dynamic Indices," *Proc. 1981 ACM SIGMOD Annual Conference on the Management of Data*, Ann Arbor, Mich., May 1981.

[ROWE 79] Rowe, L. et. al., *Data Abstraction, Views and Updates in RIGEL*, Electronics Research Laboratory, University of California, Berkeley, CA., Memo 79/5, January 1979.

[ROWE 82] Rowe, L., et. al., *Rigel Language Specification*, Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA., August 1981.

[SELI 79] Selinger, P. et. al. "Access Path Selection in a Relational Database Management System," *Proc. 1979 ACM SIGMOD Annual Conference on Management of Data*, Boston, Mass., May 1979.

[SHIP 82] Shipman, D., "The Functional Data Model and the Data Model DAPLEX," *ACM-TODS 6, 1*, pp. 140-173, March 1981.

[STON 76] Stonebraker, M. et. al., "The Design and Implementation of INGRES," *ACM-TODS 1,3*, September 1976.

[STON 82] Stonebraker, M., *Application of Artificial Intelligence Techniques to Database Systems*, Electronics Research Laboratory, University of California, Berkeley, CA., Memo 82/31, May 1982.

[STON 83] Stonebraker, M. et. al. Application of Abstract Data Types and Abstract Indices to CAD Data Bases, *Proc. 1983 ACM-IEEE Data Base Week*, San Jose, CA. May 1983.

[WASS 81] Wasserman, A. et. al., *Revised Report on the Programming Language PLAIN*, Laboratory of Medical Information Science, University of California, San Francisco, CA., Technical Report #42, January 1981.