ISSN 0254–2757

QI QUÆSTIONES INFORMATICÆ

Volume 5 • Number 1 April 1987

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereeneging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

QUÆSTIONES INFORMATICÆ

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Africa en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Editor

Professor G. Wiechers
INFOPLAN
Private Bag 3002
Monument Park 0105

Editorial Advisory Board

Professor D.W. Barron
Department of Mathematics
The University
Southampton S09 5NH, UK

Professor J.M. Bishop
Department of Computer Science
University of the Witwatersrand
1 Jans Smuts Avenue
2050 WITS

Professor K. MacGregor
Department of Computer Science
University of Cape Town
Private Bag
Rondebosch, 7700

Prof H. Messerschmidt
University of the Orange Free State
Bloemfontein, 9301

Dr P.C. Pirow
Graduate School of Bussiness Admin.
University of the Witwatersrand
P.O. Box 31170, Braamfontein, 2017

Professor S.H. von Solms
Department of Computer Science
Rand Afrikaans University
Auckland Park
Johannesburg, 2001

Professor M.H. Williams
Department of Computer Science
Herriot-Watt University, Edinburgh
Scotland

Circulation and Production

Mr C.S.M. Mueller
Department of Computer Science
University of the Witwatersrand
2050 WITS

Subscriptions

Annual subscription are as follows:

	SA	US	UK
Individuals	R10	$ 7	£ 5
Institutions	R15	$14	£10

Quæstiones Informaticæ is prepared by the Computer Science Department of the University of the Witwatersrand and printed by Printed Matter, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

SEMANTIC INFORMATION MANAGEMENT

Sonia Berman
Department of Computer Science,
University of Cape Town

ABSTRACT

An information management system with a semantic data model interface is currently being developed at UCT. This system provides high-level, non-procedural access to information and enforces powerful integrity and security constraints on stored data. The definition of tasks, routines which can subsequently be invoked by application programs, is included with the definition of the data itself.
KEY WORDS: information management, semantic model, database management system, integrity

1. INTRODUCTION

SDMS (the Semantic Data Management System) is being developed to improve database programmer productivity and to provide semantic data model facilities that capture more of the meaning of data in its definition and usage. This paper describes the data definition and manipulation facilities provided by SDMS and gives an outline of its implementation. In conclusion the system is evaluated and some suggestions for future work are made.

2. THE SDMS SCHEMA

An SDMS schema comprises a list of objects optionally followed by a list of tasks. Objects are entities of interest about which information will be stored; tasks define special operations to be performed when manipulating these objects and are based on a similar concept in the theoretical language TAXIS [9]. Properties can be specified for objects to describe their characteristics and behaviour. They can denote attributes (e.g. name of a person), relationship (e.g. courses taken by students), actions (e.g. calculate average mark) or tests (e.g. to ensure employed persons are of reasonable age.

To increase the semantic content of the schema, the following features of object properties are supported. They can be designated "id-properties" if their value uniquely distinguishes occurences of that object. They can be declared "class" [5] if they describe a class of object rather than particular occurences of an object. (Examples: average-course-size and maximum-course-length). Properties can be optional or mandatory, changeable or unchangeable (e.g. birthdate), single- or multivalued and weak or independent. A weak property is one whose existence depends on that of some other object: marks would be weak properties of students because a mark is of no interest if the student who attained it is deleted — when this occurs the student's marks must be deleted as well. Every property of objects has an associated valueclass [5] which indicates what type of value(s) that property will assume. A property can thus be seen as mapping from the object it describes to its valueclass. Name is a mapping from person to strings, courses-taken is a mapping from students to courses, and so on. This mapping can be defined as partial or total to indicate whether all objects of the valueclass must participate in such a mapping, or can optionally be exempt from this. From this it can be seen that for any property, both the mapping and its inverse are clearly described. Where applicable,i.e. where the valueclass is "string", "real" or "integer", properties can have their exact format (types and sizes) declared. It should be note that properties such as name represent ordinary printable attributes of an object; while properties such as course-taken represent a relationship between two types of object. One of the major reasons for the simplicity inherent in SDMS is its uniform treatment of attributes and associations.

The semantic richness of the schema is especially evident in its facilities for defining integrity constraints and data derivations. The latter enable a property to be defined as the sum, minimum, average, maximum, count, union, intersection, subset or difference of other (multi-valued) properties. Examples: students-passed would be a "SUBSET OF students-registered

WHERE average-percent >= 50"; students failed could be "SUBTRACT students-passed FROM students-registered". Alternatively a derivation can specify an arithmetic expression or formula by which a single-valued property is to be computed from other single-valued properties. Integrity constraints allow for the declaration of any logical expression which must hold between properties and/or constants. Examples are "age <= 65 && age >= 16" and "tax <= 0.5 * salary". Finally it should be stressed that redundancy in the data description is encouraged in SDMS, as this increases the chances of obtaining a complete data definition and enhances its semantic content. To handle this " inverse" and "match" must be specified wherever one property represents the exact opposite, or exact same information as another property, respectively.

Security constraints can be specified for properties, objects or tasks and are enforced by means of passwords. Those defined for properties and objects can futher specify the operation(s) for which that password is required. Comments can also be defined for any property, object or task to clarify its exact meaning or purpose.

Object definitions can also include semantic information in the form of isa-hierarchies [11], groupings [5], functional dependencies between properties of an object [4] and list of the tasks that apply to each object. If "lecturer isa person" lecturers inherit all the properties of the object person in addition to the stated properties of their own. This facility considerably reduces the number of property declarations that have to be given; it also enables special constraints to be enforced when an inherited property is redefined for the specialised object to restrict its range of values (e.g. person property age might be redefined for lecturers to enforce "age > 21"). A predicate can be defined when an "isa" association is declared, to indicate what condition an object must satisfy to be classed as that special type of object. The predicate for lecturer would probably be based on the value of its career property. A grouping allows an object to be defined as a collection of other objects. Thus committee could be defined as a grouping of students and lecturers. A predicate defining how groupings are to be formed is also supported.

To provide database administrators with some control over the physical storage of the data, objects can have access methods defined for them, along with any keys to use. The options available are isam, hash, heap, compressed isam, compressed hash, compressed heap and truncated. These access methods are totally transparent to the programmer.

Tasks can be one of two types: base and auxiliary. The name of a base task is of the form operation_object or operation_object_property. This implies that the task must be invoked when the specified operation is performed on that perticular object or object-property, as some additional processing has to be carried out as specified in the task body. Auxiliary tasks are analogous to program modules; they can be called from any other task in the system. All tasks comprise actions optionally interspersed with tests. An action can be any SDMS or C [7] statement, or a task invokation. A test has associated with it some predicate which, if it evaluates to false, will raise an error and abort the task. Tests can optionally specify the name of an exception-handling (auxiliary) task to deal with the error.

3. THE DATA MANIPULATION LANGUAGE (DML)

The SDMS data manipulation language is described in detail in a companion paper [2]. To ensure ease of use there are few, simple commands. The language is non-procedural, relationally complete [4] and provides associative retrieval (viz. data is accessed by conditions on any of its properties). The SDMS data manipulation commands are GET, CREATE, DELETE, UPDATE, NULLIFY, INSERT and remove. NULLIFY alllows some (or all) of the values of a multivalued property to be erased. REMOVE and INSERT permit de-/classification of objects in a specialised object type or grouping. File input/output, aggregate functions (minimum, maximum, average, sum, count) and "mappings" [5] are supported. A mapping enables properties of properties to be referenced directly, as in "Thesis.Supervisor.Name". Any number of properties of different objects can be specified in any order in the commands, all of which operate on collections of objects rather than one at a time.

4. IMPLEMENTATION

SDMS comprises three major subsystems: a schema processor, a DML processor and a browsing facility. It was decided to implement SDMS on top of a relational database system (viz. Ingres [10]) as this greatly simplified the development of an access method. A relation scheme is therefore designed from the given SDMS data definition and SDMS operations are executed by manipulating the underlying relational database. This does not involve extra overheads compared with Ingres, which also translates its DML into calls to Ingres routines; SDMS uses the same routines but simply maps from a higher-level language.

The schema processor uses Lex [8] and YACC [6] to parse the schema, performs all semantic checking and creates the internal representation of the schema. This metadata can subsequently be accessed by all users in exactly the same way as the data content. This processor also designs the relational scheme in which data will be stored. The design algorithm [1] essentially creates one relation for each object and then decomposes this into third normal form [4].

The DML processor performs syntactic and semantic checking of SDMS commands and generates appropriate calls to Ingres routines to perform the required data manipulation. It accesses the internal representation of the schema to check for program errors, to ascertain all actions necessary to execute the operation correctly, and to determine whether any special task must be invoked. In general one SDMS command is translated into several operations on the underlying database. This occurs for the following reasons: the properties of an object are scattered amongst several relations, an update or delete of an object must cascade through all relations where that object is referenced, derived data values must be computed and “inverse” and “match” properties must be correctly maintained.

The browsing facility guides the user through the schema to indicate what information is available and then assists him in formulating a query. Conditions on data values of interest can be specified using natural language. A query can be modified and rerun several times if desired.

5. CONCLUSION

SDMS enables information systems to be designed and used with less effort than would be the case using conventional database or filing systems. It should thus substantially improve programmer productivity. The non-proceduralarity of the access language provides navigational problems and its high-level primitives avoid the complexity of relational joins. The security and integrity of data is enhanced because of the rich semantic content of the data description. Complete data independence is achieved since the user's view of the information simply comprises a collection of objects and their properties and no internal data structuring. Maintaining tasks along with the data definition increases the semantic content as well as ease of use. The project has opened several avenues for future research. Incorporation of subschemas, database design tools, historic data and automatic program- and documentation- generation could be investigated.

APPENDIX - (PARTIAL) SCHEMA EXAMPLE

```
DATABASE uct
OBJECT person
ACCESS hash USING idnumber
 ID-PROPERTY
  firstname  : STRING C12 M:1 unchangeable compulsory;
  initials   : STRING A6  M:1 unchangeable;
  surname    : STRING C24 M:1 compulsory;
  birthdate  : date       M:1 unchangeable;
ID-PROPERTY
 idnumber : STRING D13 1:1 unchangeable compulsory;
PROPRTIES
 sex            : personsex     M:1 unchangeable compulsory;
 medicalaid     : boolean       M:1;
 pensionfund    : boolean       M:1
 maritial       : maritalstatus M:1;
 children       : person        2:M
 numdependents  : INTEGER       M:1 count children;
OBJECT lecturer ISA person
LOCK DELETE : "*(&^%"; UPDATE : "3y28#";
 PROPERTIES
  position     : academicstatus M:1 compulsory;
  dep          : department     M:1 compulsory;
  office       : STRING C4      2:1 compulsory;  /* max 2 per office */
  telnos       : telephonenum   4:2 total;       /* max 2 phones per chap */
  salary       : money          M:1 compulsory
  supervises   : student        1:M match with supervisor of project;
  lectures     : course         M:M compulsory total inverse of
                                    course-lecturers
  contacts     : student        M:M union of supervises and
                                    lectures.attendants
  subsidy      : boolean        M:1;
FDS telno -> office;
SPECIAL-TASKS
DELETE_lecturer; UPDATE_lecturer_marital;
...
...
 TASKS
   DELETE_lecturer
     precondition  : TEST COUNT (supervises) == 0;
     actiondone    : DELETE lecturer
/* add his position to "posts" - a multivalued property: */
     officefreed   : UPDATE dept (freerooms = freerooms + 1;
                     posts = lecturer.position) WHERE
                     name == lecturer.dep.name;
     phonefreed    : UPDATE university(freephones=freephones+COUNT(telnos));
     postcondition : TEST COUNT(dep.posts) !=dep.numposts ERROR emptydep();
 END
...
...
   UPDATE_lecturer_marital
/* a married female cannot legally be subsidised! */
     execute       : UPDATE person;
     checksubsidy  : TEST sex == "F" && marital == "M" && subsidy == "Y"
                     ERROR illegalsubsidy ();
 END
...
...
```

REFERENCES

1. Berman, S., [1983], ADD - The Automated Database Design Tool, *Quæstiones Informaticæ*, **2**, pp 25-27.
2. Berman, S. and Walker L., [1986], A High-Level Language Interface to a Relational Database, *Quæstiones Informaticæ*, **4**, pp 7-12.
3. Chen, P.P.S., [1976], The Entity-Relationship Model: Toward a Unified View of Data, *ACM Trans. on Database Syst.*, **1**, pp 9-36.
4. Codd, E.F., [1970], A Relational Model for Large Shared Data Banks, *Communications of the ACM*, **13**, pp 377-387,
5. Hammer, M. and McLeod, D., [1981], Database Description with SDM: a Semantic Datatbase Model, *ACM Trans. on Database Syst.*, **6**, pp 351-386.
6. Johnson, S.C., [1975], *Yacc: Yet Another Compiler Compiler*, Computer Science Technical Report No. 32, Bell Laboratories, New Jersey.
7. Kernighan, B.W. and Ritchie, D.M., [1978], *The C Programming Language*, Englewood Cliffs NJ, Prentice-Hall.
8. Lesk, M.E., [1975], *Lex - A Lexical Analyser Generator*, Computer Science Technical Report No. 39, Bell Laboratories, New Jersey.
9. Mylopolous, J., Bernstein, P.A. and Wong, H.K.T., [1980], A Language Facility for Designing Datbase-Intensive Applications, *ACM Trans. on Database Syst.*, **5**, pp 185-207.
10. NCR Corp., [1983], *TOWER Database Manager*, Relational Technology Inc.
11. Smith, J.M. and Smith, D.C.P., [1977], Database Abstractions: Aggregation and Generalization, *ACM Trans. on Database Syst.*, **2**, pp 105-133.

NOTES FOR CONTRIBUTORS

The purpose of the journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review artilces and exploratory articles of general interest to readers of the journal. The preferred languages of the journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to:

Prof. G. Wiechers
INFOPLAN
Private Bag 3002
Monument Park 0105
South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. Manuscripts produced using the Apple Macintosh will be welcomed. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name and affiliation and address. Each paper must be accompanied by an abstract less than 200 words which will be printed at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Tables and figures should not be included in the text, although tables and figures should be referred to in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Figures should also be supplied on separate sheets, and each should be clearly identified on the back in pencil and the authors name and figure number. Original line drawings (not photocopies) should be submitted and should include all the relevant details. Drawings etc., should be submitted and should include all relevant details. Photographs as illustrations should be avoided if possible. If this cannot be avoided, glossy bromide prints are required.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters; between the letter O and zero; between the letter I, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetic order of the author's name, and cited in the text in square brackets. Journal references should be arranged thus:

1. Ashcroft E. and Manna Z., The Translation of 'GOTO' Programs to 'WHILE' programs., *Proceedings of IFIP Congress 71,* North-Holland, Amsterdam, 250-255, 1972.
2. Bohm C. and Jacopini G., Flow Diagrams, Turing Machines and Languages with only Two Formation Rules., *Comm. ACM,* **9**, 366-371, 1966.
3. Ginsburg S., Mathematical Theory of Context-free Languages, McGraw Hill, NewYork, 1966.

Proofs

Proofs will be sent to the author to ensure that the papers have been correctly typeset and *not* for the addition of new material or major amendment to the texts. Excessive alterations may be disallowed. Corrected proofs must be returned to the production manager within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Only orginal papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.