J. M Bishop
24/3/91

South African Computer Journal
Number 4
March 1991

Suid-Afrikaanse Rekenaar-tydskrif
Nommer 4
Maart 1991

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

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

Die Suid-Afrikaanse Rekenaartydskrif
'n Amptelike publikasie van die Suid-Afrikaanse Rekenaarvereniging en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

Editor
Professor Derrick G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083

Assistant Editor: Information Systems
Dr Peter Lay
P. O. Box 2142
Windmeul 7630

Subscriptions

	Annual	Single copy
Southern Africa:	R32-00	R8-00
Elsewhere:	$32-00	$8-00

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Computer Science Department, University of Pretoria, Pretoria 0002
Phone: (012) 420-2843 Fax: (int) + (012) + 43-6454

Sigma Press

Guest Editorial

Does Today's Industry Need Qualified Computer Scientists?

This guest editorial consists of two contrasting views on the value to industry of a professional degree in computer science. Both authors, one local and one from Germany, are managing directors of well-respected software houses. (Editor)

Viewpoint I

Hans G Steiner
MBP Software and Systems GMBH
Semerteichestrasse 47-49
D4600 Dortmund 1

I would like to begin by recounting from my student days a story that I consider to be relevant. While attending a career forum for computer scientists, mathematicians and physicists, the personnel officer from IBM Germany was asked if he would consider taking on mathematicians. The gist of his answer was as follows: "Of course I must admit that I could just as well give the mathematician's job to a theologian. What is important is the ability to think logically. It is only there, on the job, that he learns how to become productive for us."

This episode occurred 14 years ago at a time when graduating mathematicians did not necessarily learn programming and when computer scientists were few and far between. The situation has improved immensely since then. Mechanical engineers, electrical engineers and physicists, all with programming knowledge, have for the most part taken over many programming jobs. This shows industry that, as time goes by, the answer to the opening question is becoming an ever-louder and more frequent "NO".

I support this opinion and in the remainder of this essay I will expand on my reasons, as well as highlight some exceptions.

An employee who is recruited directly from a university should possess the following four capabilities:
1. An ability to think logically: One of the basic requirements in our business is the ability to recognise, analyze, structure, break down and solve a problem as well as to fully synthesize the solution. The important thing is to break down the problem in such a way that the individual components can feasibly be solved. This is what distinguishes an engineer/scientist from an arts scholar. The latter usually concentrates on the complete problem and tends to settle for a contentious, complex and partially non-feasible solution. In our business, it is not enough to merely ask the "right" questions.

This ability to think logically may be accentuated in computer science; however engineers/scientists will generally possess the ability to an equal extent.

2. Programming skills: Our employees' prime tool of the trade is their ability to encode solutions to problems. Ideally, this ability ought to be held as abstract as possible. In other words, the further away from the "bit", the better. FORTRAN programmers who, for example, concentrate on the multiple use of memory space of all variables will never be successful programmers in an object-oriented programming language.

The difference can be seen, even in today's universities. For example, one only has to read a PROLOG program from a student who learned PASCAL in his first semester and PROLOG in his fifth. On average, this is always a "PASCAL program in PROLOG". The various possibilities offered by a predicate calculus language are only recognised and used by the best students. Again, we do not need the average computer science scholar who has spent between six and eight years writing complicated PASCAL programs, but rather the "thinker" with basic programming knowledge who is capable of abstracting the task. Once again, the ability is independent of faculty.

3. Teamwork skills: Working successfully in a team

This SACJ issue is sponsored by
Department of Computer Science
Rhodes University

requires assertiveness, tolerance, stability and one's own ideas. Very few problems have solutions that can be managed by one person successfully in the allocated time. Out of 700 employees, we can only afford approximately five "lone warriors" who are, in turn, the leading specialists in a wide field. They have a strategic vision which we follow. All remaining employees are evaluated, for better or worse, on their team performance. Some people have an in-built ability to work in teams. A few universities - unfortunately not enough - encourage this team-thinking. Again we see that the ability is independent of university faculty.

4. *Motivation:* The ability to enjoy one's particular job is a major driving force in every employee. Whereas in the sixties everything had to be "bigger, faster and better" and in the eighties "things had to be meaningful to society", the theme for the nineties is self-realization. Those companies who succeed in incorporating different employees (ie employees with different driving forces) into the company culture and who motivate each employee optimally will be successful in the nineties and beyond. There are huge productivity gains to be had from motivating employees. Compared with this, the possibilities offered by CASE tools pale into insignificance.

One basic requirement is thus the recruitment of a self-motivated employee who should at no stage become demotivated, whether it be by company culture, superiors or working conditions.

Again, this is not linked to a specific university faculty and is independent of know-how.

As none of these four capabilities are necessarily restricted to studies in computer science, the technical/-scientific background of new employees who are being recruited is largely irrelevant.

I would now like to point out a few exceptions which might give a computer scientist the upper hand in an interview. I refer exclusively to our own company and our specific company tasks.

1. Porting our COBOL Compiler onto the latest UNIX machine from the manufacturer XY. Knowledge of the UNIX operating systems could be very valuable and enable the new employee to rapidly become productive.
2. Programming the 37th interface (special customer request) for our ISDN card. Knowledge of interface protocols or experience with protocol conversions would be very useful and could be a decisive factor. Such specialized knowledge is usually very rare.
3. Adapting our integrated office automation system to the 17th foreign language. The employee must command the language perfectly. Simply outsourcing the translation would mean that this language version could not be maintained or supported. From this example one can see that specialized knowledge not only refers to knowledge gained from computer science studies.

In the product business, it sometimes happens that computer scientists with specialized knowledge are sought. (This is almost impossible in the project business, due to the variety of tasks to be performed.) However such a "knowledge" advantage over others usually only lasts about a year. After that, the achievements of two different employees (one with specialized knowledge and the other without) tends to even out.

Most applicants who start out do not know our products, as the flow of employees in this industry is almost always from manufacturer to user. Hardware and software manufacturers often lose their products specialist to the products' users. Seldom do employees change in the other direction.

In my opinion, universities can learn two things from this essay:

1. Studies in computer science give basic knowledge that can be used in various jobs. The student should however be careful not to place all his eggs in one basket.
2. Teamwork should be encouraged more. Time allows for very few geniuses, acting as 'lone warriors', to initiate progress in our society.

I have taken the liberty of basing my interpretation and answer to the opening question on my own judgement and experiences. I would be grateful for other opinions and experiences on this topic.

I would like to conclude by expressing my gratitude for having had this opportunity to express my views.

Viewpoint II

Pierre Visser
Grinaker Informatics, P.O.Box 29818, Sunnyside, 0132

The title question currently generates as many viewpoints as a counterpart question: "What is the correct curriculum for a computer science qualification?" Such questions stem from the many and diverse requirements expected to be fulfilled by the still developing applied science. A basic assumption of this editorial is that we need to have an explosion and consolidation in computer science theory. Only after this has occurred will a more general consensus of opinion exist - as is the case in other matured sciences.

An argument is presented here for the current approach of striving towards a balance between immediate industry needs and long term perceived theoretical requirements of industry, even though the balance, as viewed from either side, will always be imperfect.

Industry can, of course, do without qualified computer scientists - that is how it was established. Dedicated mathematicians, physicists, engineers and other scientists will, as in the past, continue to effect improvements. However, as one of those scientists from the

early days, it is difficult for me to understand why one would choose to continue this way.

A computer science qualification is viewed here as a university education (4 years) into theory that is not obtainable otherwise. By definition, therefore, a qualified computer scientist is not trained to conform to specific job requirements. Rather, the computer scientist will possess knowledge that will serve him long past the present day's computing technology.

Whether industry needs qualified computer scientists depends on two issues. Firstly, can an education be provided for computing technology that will serve as a foundation for the student's next 45 years in industry; and secondly, can industry build upon this foundation to created wealth more effectively than without qualified computer scientists.

It is widely accepted that, in broad terms, the teaching of fundamental theory will serve the first purpose. However, what subject matter to include from the wealth of mathematics, physics, OR, and from computing fields such as networking, operating systems and others, remains the illusive issue. Universities can merely strive to select the right mix for the perceived future needs of industry. This requires insight into the evolution of computing technology. I will later discuss such insight as a basic requirement for a qualified computer scientist.

What is important in teaching is to focus on fundamental theory. Just as the natural science student needs to breed fruit flies in order to gain insight into the dynamics of inheritance, so too the computer science student needs to develop software. The purpose should be to create understanding and insight into fundamental theory, and, just as in the case of the breeder of fruit flies, the software developed should never be measured against efficiency requirements from industry.

The second issue is whether industry can build on this theoretical foundation to create wealth.

A depth of insight into computing technology, more so than with other training, can be identified as the focus of the potential value of a qualified computer scientist to industry. Three areas which require such insight are discussed below, namely organisation, product definition and the application of new computing technology in industry.

Computing products form an integral part of an organisation, and represent a significant capital investment aimed at increasing efficiency. These products are incorporated in an evolutionary way to match changing organisational requirements with improving product capabilities. Decisions to use products determine the long term efficiency and cost-effective replacement. Such decisions require insight into computing technology and its evolution. A qualified computer scientist can improve such decisions only if he gains enough insight into computing technology as well as its interaction with business through years of practice.

The success of products in some areas is dependent on market requirements which depend on computing technology and its evolution. The correct definition of characteristics of products that interface to computers is such an example. Insight into computing technology is able to create the versatility, simplicity or other improved selling features which can open new market segments.

The third area where insight into computing technology plays an important role is in the application of new computing technology (or a new trend) in an organisation. Examples include the introductory period for networking, DBMS-technology, distributed processing and document image processing. In areas such as these, the newly qualified computer scientist can be applied effectively and at the same time build up insight through experience which he will require for the other areas of organisational and product decisions mentioned above.

A major dilemma in the continuous development of insight into computing technology by qualified computer scientists is their correct application in industry. The identification of the opportunities within the three areas discussed above, requires insight into computing technology itself. Winning companies that depend on computing technology have this ability. In such companies the insight of the qualified computer scientist into computing technology as well as its contribution to the business is constantly stimulated, turning the qualified computer scientist into a valuable company resource.

What has been neglected in this whole discussion is the role of the "technician" and of the casual user of computing technology. Such personnel are required to implement selected computing technology of the day efficiently, whether in accounting, chemical engineering or other specialised disciplines. Their role and place is unquestioned. However, it cannot be expected of them to evaluate the potential of new computing technology, formulate algorithms from fundamental theory or any such decisions which require insight built upon a sound theoretical knowledge of the field.

The final aspect in answering the opening question is whether the qualified computer scientist can outperform other professionals who build up their own experience in computing technology. Many examples could be cited of improvement brought about by non-computer scientists in the past. However, these individuals formed part of the bootstrapping for computer science theory and education. We should have faith in this bootstrapping of computer science qualifications, because computing technology will increasingly diversify into many directions of specialisation in years to come, each requiring a body of fundamental theory.

This complexity cannot be left to a casual development of insight - industry requires qualified computer scientists to experience interaction with business objectives in order to cope successfully with future computing technology.

The Universal Relation as a Database Interface

[1]M J Philips and [2]S Berman
[1]*Peninsula Technikon, P O Box 1906, Bellville 7535*
[2]*Department of Computer Science, University of Cape Town*

Abstract

The universal relation is a way of overcoming users' prime database access difficulty, namely navigation through database structures. As a result there is reduced user involvement with the underlying logical database structure.
The universal relation is an imaginary relation that contains all the data in the database. Data element roles and 'flavours', weak universal relations, and cycles in database structures are important related issues. Practical universal relation implementations include the computational approach, where the universal relation is generated; the use of multiple predefined views, and the window generation method. The latter has been most popular, largely due to the demands made on hardware by the processing of user views.
Keywords: *universal relation, end-user systems, database navigation*
Computing Review Categories: *H.1.2, H.2.3.*

Received October 1989, Accepted October 1990

1. Introduction

Navigation through database structures by means of join operations, presents one of users' major difficulties in database access. The universal relation is one way of overcoming this. The logical structure of the database need not be considered by the end user at query time.

In this paper the concepts underlying the universal relation approach are described, together with related issues such as data element roles and 'flavours', weak universal relations, cyclic database structures, and the use of this theory in practice.

The various interpretations of views or windows on such a universal relation, as implemented in various systems, are described and contrasted. These include contexts, connections or windows, and representative instances.

2. Statement of problem

2.1 Data independence

The users' identification of information required is affected by the structure of the database. Many of their difficulties are a result of the inability to understand the database structure and contents [3]. This can result in incorrect responses (as compared with the user's requirement).

Apart from the simple matter of using the schema's own names for objects, complications may arise regarding entities upon which several meanings or roles may be imposed.

In any database implementation, there exist many alternative searching methods for obtaining required data. Great differences in efficiency may occur depending on the utilization of keys. The user may not be fully aware of the search options, or their effects.

Changes to complex database structures, together with outdated documentation, create an extra burden for the user.

2.2 User ergonomics

The specification of data requirements may be ambiguous, incomplete, or incompatible with the database itself, or simply be incorrect [3]. The scope of the database may be such that the user's query cannot be responded to as he would expect.

2.3 Features of a solution

The general requirements placed on a system in attempting to overcome some of these difficulties include the following:

The user must be required to supply the absolute minimum amount of information.

There must be logical data independence; i.e. the shielding of the user from details of logical data storage, over and above the issues relating to physical data storage. Examples of the former include which relations are to be used, their components and inter-relation join criteria.

3. Universal relation

3.1 Introduction

The universal relation r is an imaginary relation that contains all the data in the database. It is such that the tuples reflect all the predicates constraining

relationships between data elements. Stated formally, the universal relation must satisfy the join dependency below, i.e. the join of the projections of r onto the stored relation schemes R^1 ... R^n is equal to r.

$\pi R^1(r) \propto \pi R^2(r) \propto ... \propto \pi R^n(r) = r$

Relationships between attributes in a relation scheme may be defined in 'objects', which are 'minimal sets of attributes that have collective meaning' [28]. Such objects may often be equivalent to the stored relations, though they need not be.

The universal relation over the set of attributes (object) Ri must contain exactly the set of tuples that satisfy P (the predicate applicable to the object), and must satisfy the join dependency over all the relations for objects; this holds when a tuple exists which agrees with the attributes in Ri for all i (where Ri is an 'object'). That is, for each given object, and for each tuple within that object, there exists a tuple in the universal relation whose attributes' values agree with those in the relation describing the given object. This is the equivalent of a lossless join decomposition into the relations for the objects, and the relation can also be said to be join-consistent. The universal relation therefore could be expected to contain all, and only, the information in the decomposed relations. Such a relation is also said to satisfy the universal instance assumption [8,24].

The universal relation is a means of overcoming logical data dependence in that it goes one step further than the relational model. In this case, the organization of data over stored relations need not be known by the user [16,24].

An example is given of a universal relation over C(ourse), T(eacher), R(oom), H(our), S(tudent), and G(rade). These tuples are all those, and only those, which satisfy all the predicates applying to each of these entities individually. Examples of the tuples could be:

P (Course A, 10h00, Room 121) course A meets in room 121 at 10h00

P (Course B, Teacher Smith) course B is taught by teacher Smith

and the natural join of all these projections is the universal relation [7].

The universal relation scheme assumption (URSA) requires that an attribute refers to the same underlying class of entities wherever it occurs [17]. This means that a certain concept is intended with each attribute, and that concept is known by the same domain name; (eg Colour) wherever such an attribute occurs.

3.2 Attribute roles

Attributes included in a database scheme can play various roles, relating to the real world. This results in attributes being 'semantically overloaded', i.e. having different meanings when projected from different relations [16]. A common example is the attribute EMPLOYEE, which may represent MANAGER or STAFF-MEMBER. The relationship between attribute roles is described by the inclusion dependency, whereby the values of each distinct role (here MANAGER and STAFF-MEMBER) is a subset of those of the 'primary' attribute (here EMPLOYEE).

This means that every STAFF-MEMBER and MANAGER is an EMPLOYEE, but not necessarily the reverse. This role relationship can be shown as EMPLOYEE -> STAFF-MEMBER, i.e. STAFF-MEMBER is a role of EMPLOYEE. All of these manifestations would, however, be based on a single domain. In a database scheme with relations:

WORKS-FOR (EMPLOYEE,DEPT),

and MANAGES (EMPLOYEE,DEPT)

tuples in the first relation could be:

{<102,2 >, <104,2 >, <105,1 >, <106,3 >, <107,3 >, ...},

and the second could be:

{< 103,2 >,< 108,3 >, ...}

To disambiguate the roles in the relation scheme, the applicable relation name must be used.

In the unique role assumption (URA), all attributes take on only one role in addition to the URSA restriction. In order to impose various roles on attributes, the naming of attributes must be unique for each role since equivalence is based on attribute names. The existence of such role-specific attributes means that they occur uniquely in relationships with other attributes. Using the example above, EMPLOYEE is either MANAGER or STAFF-MEMBER. STAFF-MEMBER and MANAGER would therefore become attribute names in addition to EMPLOYEE. Thus the access paths between relations are embedded in the attributes and their names [16].

The unique role assumption under the universal relation approach requires that all objects play unique roles, and that separate relations can only be linked on attributes with identical column names. There is therefore no need to specify access paths, but this restricts relationships between objects to only the most direct, due to these semantic facts being included in the overall database schema [2].

3.3 One flavour assumption

Attribute 'flavours' can also result in incorrect perceptions of data. Using the earlier example of Courses, Teachers, Rooms, Students and Grades, the relationship between Course - Teacher - Room implies Courses taught by a Teacher in a particular Room, whereas the relationship Course - Student implies any Courses taken by a particular Student. The domain Courses may be a superset of either, or their union. This generally occurs when multiple access paths reach the attribute in question. If three relations make up a database scheme, AB, BC, and AC, the set of attributes AB can be produced by two methods; either the existing relation AB, or the union of the projection on A,B of the natural join of AC and BC, and the relation AB:

$(AB) \cup \pi AB(AC \propto BC)$

These AB tuples might show a different relationship

depending on whether they were tuples from the relation AB, or projections of the join. In certain circumstances the relationship of A to B by way of the link on the third attribute may also be required.

3.4 Weak universal relations

For a given set of attributes R, there may exist a relation equivalent to the universal relation i.e. the universal relation is a superset of that relation [28].

A weak universal relation is a modified universal relation such that weak instances contain exactly those tuples which reflect the dependencies in a given set, or which can be inferred from that set, and not necessarily all the dependencies existing between all the data attributes. A weak universal relation satisfies only a given set of dependencies, and is a superset of the current relation R, for every R [16].

Weak equivalence can be considered as 'equivalence assuming no dangling tuples'; for example:

if dangling tuples are present, then there may not be a tuple in r relating b2 to c1, given

r1 = {< a1,b1 >,< a2,b1 >},

and

r2 = {< b1,c1 >,< b2,c1 >}

and such a projection would not have a value for b2,c1.

Since weak equivalence requires the involvement of only a subset of the relations making up the full database scheme, fewer joins result, leading to some optimization of queries, and it also provides results more likely to meet the intent of the inquiry; perhaps due to not wanting dangling tuples to be involved [24].

3.5 Cyclic database structures

Since they are of importance in using the universal relation, a description of cyclic database structures is given here, and certain implementations which address the topic are dealt with later.

A database hypergraph, which shows relationships by means of hyperedges between two or more relations, is acyclic if its 'Graham reduction' is empty (i.e. no edges), otherwise it is cyclic. The Graham reduction process is similar to the removal of leaves from trees

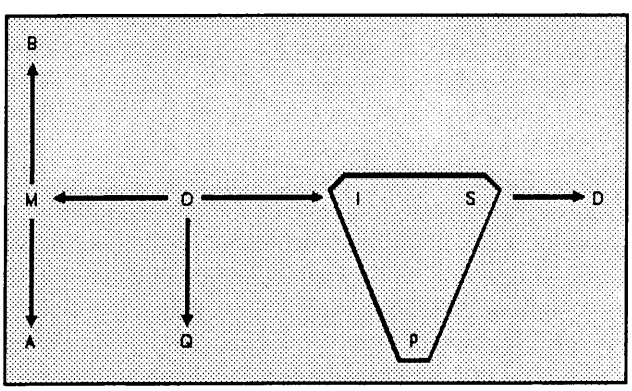

Figure 1

until nothing is left; here the process is of removing 'objects' (which may contain attributes occurring in at most one other object as well) from the hypergraph (known as 'ear removal'). The sequence of ear removal is not important because the same hypergraph will be obtained [24].

For example, using the hypergraph in figure 1,

i) remove MB in favour of MO, since B is in no object but MB,
ii) similarly, remove MA
iii) remove MO
iv) remove OQ
v) remove OI
vi) remove SIP
vii) remove SD

A cyclic database would therefore have objects which contain attributes common to more than that one object; eg. if customers can have both loan and deposit

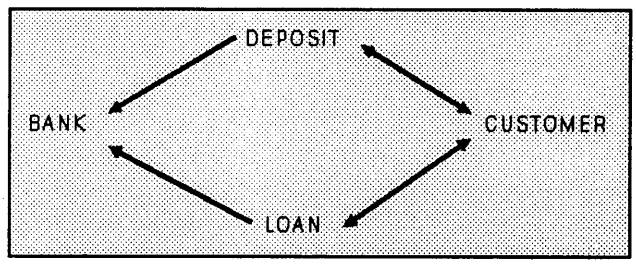

Figure 2

accounts with a bank, the following occurs: (from [14]).

Referring to the scheme in figure 2, not one of the objects DEPOSIT-BANK, BANK-LOAN, LOAN-CUSTOMER, or CUSTOMER-DEPOSIT can be removed, since attributes belong to other objects as well, and therefore the Graham reduction does not produce an empty set.

4. Features of the universal relation

4.1 Keys

Keys are likely to be very scarce in such a universal relation, due to a tuple in a normalized relation being present many times (as projected attributes of the greater n-ary universal relation). The example of an employee with multiple children shows that the employee's normal key attribute (e.g. an identification number) is not unique when a tuple exists for each child [9].

4.2 Null values

Relations may exist where certain tuples are lost when a natural join is taken with another relation. An example:

R1 = AB; R2 = BC

r1 = {< a1,b1 >}

r2 = {< b1,c1 >,< b2,c2>}

s = {< a1,b1,c1 >}

πBC(s) = {< b1,c1 >},
which is not equal to r2

These 'lost' tuples are referred to as 'dangling tuples', and are retained when normalization is applied, or in the case of the universal relation, only by including null values [23].

This also has implications for keys, where null values must now be permitted. If a key may not take on a null value, then, using the example of employees, managers, departments, etc, a newly created department, as yet without employees assigned, will not have a key [9].

4.3 Roles

A consequence of the URSA is that the role information is lost; i.e. that STAFF-MEMBER is a role of EMPLOYEE; also many attributes will have the same domain, without this being obvious from the name. Further disadvantages are that the number of attribute names will be larger than necessary, increasing the complexity of the database scheme. Furthermore, the relatability of attributes requires higher level semantic information.

4.4 Naming

Columns with the same names must be defined on the same domain. Update dependencies are created between relations having columns in common. Given that column names imply identical domains, the definition of separate relations having the exact same set of attributes contradicts the universal relation concept, or else they would be identical.

4.5 Cyclic schemes

The universal instance assumption cannot hold in the case of a cyclic database scheme, and can only be solved by defining separate attributes, based on separate domains [22]. Using the employee - manager situation, the relation could be joined to itself many times by means of a recursive query to represent the manager of the manager of ... the employee. This clearly cannot be held in a (universal) relation, if only because it is potentially infinite.

5. Relational database query processing

A general theory of the logical environment in which queries to relational databases are processed is given by D'Atri in terms of "contexts". A context is said to "cover" a query if the attributes making up the query are equal to or are a subset of the set of attributes in the context. A minimal context is a context such that there is no subcontext covering the query in question.

Contexts are similar to views; however ambiguity may exist, and different answers given to a query depending on the subcontext. If a context covers n relation schemes, and a query covers (n-1), the context may provide the result using the join of the n relations, which may not be equal to the result of projecting from the join of (n-1) relations. Weak ambiguity occurs where a context is unambiguous for at least one query, but provides ambiguous results for others. An unambiguous context gives unambiguous results for every query which is covered by it. If a context is unambiguous, then all subcontexts will be unambiguous. Also, if the database scheme is unambiguous, then all queries will receive unique answers independent of the context used, in which case the database scheme satisfies relationship uniqueness. Cyclic schemas result in ambiguous contexts, and when present, there should be a unique interpretation (a unique context) defined by the system or the designer [6]. A context may also be termed an "object view" [19].

An example, where given the following relation shemes:

BOOKS(TITLE,AUTHOR,PNAME,LC-NO)
LOANS(CARD-NO,LC-NO,DATE)
BORROWERS(NAME,ADDR,CITY,CARD-NO)

and the required attribute set is:

{TITLE,AUTHOR,NAME}

(Title and author of books borrowed by a certain borrower) is:

(TITLE,AUTHOR,NAME) = (TITLE,AUTHOR,LC-NO) *
(CARD-NO,LC-NO) *
(CARD-NO, NAME)

5.1 Data independence under the universal relation

A system supporting a universal relation interface allows the user to imagine the existence of such a relation [23]. The main approaches are: (1) a user view is presented, (2) the universal relation is physically implemented, and (3) navigation between stored relations is inferred [16].

6. Universal Relation implementation

6.1 Physical universal relation approach

One implemented system based on the physical storage of the universal relation itself is ICL's FIDL (Flexible Interrogation and Declaration Language). In this system, which uses specialized storage media for fast scanning of a file, the relationships between entities are described by the designer as an 'implication network' in a DDL. This graphical representation of the dependencies between the relations indicates the allowable joins that can be made between relations, and the interpretation of the resulting relation. An example is the situation of multiple flavours, where the meaning of a certain attribute depends upon the derivation of the relation in which it appears; e.g. using the earlier example in section 3.3. If the relationship of A to B by way of the link on the third attribute is also required, then the implication network would indicate the join

between BC and AC; if not, then such a join would not be shown.

On update, this implication network is used to create tuples of the universal relation by performing all allowed joins on the relations, and augmenting this with dangling tuples [28].

6.2 User view approach

The entire universal relation may be presented as a user view (assuming unique attribute naming), or subsets of the universal set of attributes may be included in separate views covering only those attributes (i.e. a weak universal relation) [16].

Views are a method of binding structural details, generally they are derived or virtual relations. With this approach, roles of attributes can differ when they appear in different views since the navigation path may be critical in defining the attribute's role.

The number of views could become unmanageable in nontrivial schemes and furthermore the user must understand the semantic implications (in terms of attribute roles and flavours) of the many views, and must remember many view names [17]. Additionally, views are predefined by the database designer, and may not cover all the interpretations that may be required [19].

6.3 Computational approach

In this case, where the navigation between stored relations is inferred, two processes are generally involved; that of binding, and evaluation. The first is concerned with the logical data environment of the result, the latter with the actual operations for producing the result.

The attributes required must be accommodated in some relation (binding phase), and then operations such as selection, projection etc. are executed (evaluation phase). Binding has stages of navigation and computation; the former generates a context by mapping the set of attributes and the database schema to the context. The computation stage generates the actual relation using the context [19].

Generally, in finding appropriate joins, a context is defined, which provides the access path, but the means of establishing that context can differ between systems. This can vary from a special list that is manually established at design time for a given database (as used in the Q approach), through to the system/U approach where it is calculated from the functional dependencies and objects, and then designer-checked [24]. The evaluation phase is not dealt with to any great extent here, since the emphasis is on the logical environment for evaluation (i.e. the context).

Q (UNIX)

A collection of contexts is maintained in a file (a 'rel' file) which is used to identify one which covers the query. This is a simple approach, where the database designer defines all the contexts, and where the first suitable one encountered at query time is used [27, 28].

Connections/Windows

Lossless joins are generally used as a basis for finding connections over the required attributes [16]. Connections are also referred to as "window functions" [18] and the mechanism for generating such windows the "window generator" [17]. Windows may be considered special cases of user views, however, as a result of the formal method of defining them, using objects, they provide semantic consistency. This is true since the semantic interpretations given to particular attributes occurring in two or more objects are consistent, if the objects are closed under nonempty intersection [19]. Further efforts may be made to find the 'optimal' joins by means of tableau optimization using weak equivalence [16].

The following algorithm may be used to define windows:

i) imagine the join of the universal relations
ii) replace the universal relation above with natural joins of all objects (a relation is imagined to be constructed from the actual relations by lossless joins.)
iii) construct an expression using the attributes required
iv) construct the tableau and optimize it for weak equivalence (a join of all the objects may not be the simplest or the most efficient join to take; a minimal weakly equivalent join can be used, since under those conditions, all relations are assumed to be projections of a universal relation, and objects not required for the attributes involved are eliminated from the join.)
v) build algebraic expression for the result of tableau optimization
vi) optimize for efficiency of evaluation [18,24]

Representative instances

Using a process of inferring values (from known dependencies, and existing tuples) known as 'chasing', tuples of the universal relation are completed (i.e. previous nulls may become non-nulls), and the required relation or representative instance is generated. For each relation scheme, each tuple is used to create a tuple in the universal relation, with unique nulls being inserted in the 'foreign' attributes. Then the known dependencies are used to chase the resulting relation so that previously-null attributes may receive non-null values, depending on the values for other attributes in the particular tuple. For example, using the scheme;

BOOKS(TITLE,AUTHOR,PNAME,LC-NO)
PUBLISHERS(PNAME,PADDR,PCITY)
BORROWERS(NAME,ADDR,CITY,CARD-NO)
LOANS(CARD-NO,LC-NO,DATE)

the tuples in **BOOKS**:

{<Computer Concepts, Smith, King Books, 12207>,
<Computer Advances, Smith, King Books, 12209>}

and, the tuples in LOANS :

{< 43567, 12209, 10/05/87 >, < 85632, 12207, 12/07/86 >}

would result initially in the tuples (for the relation : (TITLE,AUTHOR,PNAME,LC-NO,CARD-NO, DATE))

{< Computer Concepts, Smith, King Books, 12207, , >,
< Computer Advances, Smith, King Books, 12209, , >,
< , , , 12209, 43567, 10/05/87 >,
< , , , 12207, 85632, 12/07/86 >}

Using the known dependency:

LC-NO -> TITLE, AUTHOR, PNAME

the last two tuples become:

{<Computer Advances,Smith,King Books, 12209, 43567, 10/05/87 >,
<Computer Concepts,Smith,King Books, 12207, 85632, 12/07/86 >}

This chase process is completed when no new values are found.

The representative instance differs from the pure universal relation in that the values in the attributes are dependency-driven, and are not derived from joins of the individual relations, thereby possibly including more than the values justified by the known dependencies. This may be different when the set of attributes in question is a subset of the universal set of attributes.

Representative instances are always the intersection of all the weak instances, since each is a weak instance, i.e. the tuples satisfy a given set of dependencies, where distinct values are substituted for nulls [16].

Maximal objects

A solution to the problem of defining a relation upon which queries can be evaluated using the correct attribute interpretations, under conditions of cyclic structure, is presented in the form of "maximal objects" [24].

Maximal objects can be found by examining a list of sets of relations to join in order to cover all the attributes required, and furthermore, these joins can be constructed automatically, using functional dependencies and lossless joins. Such maximal objects are built from single objects, ensuring a lossless join at every step; this follows from the functional dependencies and join dependencies on the objects. The construction of maximal objects begins with a given object, and the largest set of objects containing it is formed such that each additional object is adjoined with a lossless join; then such sets of objects that are subsets of others are removed, leaving the maximal objects [24].

A join of two relations is lossless when the intersection of the relations functionally determines either source relation. Using MVD's, the join is found to be lossless if and only if the intersection multidetermines a set of attributes that includes the difference between the attribute sets of the relations in one direction but not the other. When both FD's and MVD's are used together, the MVD only rule must hold, and both set differences may not functionally determine the intersection. This rule is more stringent than the rule involving only MVD's, since 'connection traps' are avoided [14]. This situation can arise when embedded MVD's exist, such as in the example in section 3.5 and figure 2 where a connection from the Loan account to the Deposit account via the Bank is excluded by the set difference between the prospective addition (Bank-Deposit) and the candidate maximal object (Customer-Loan-Bank) not functionally determining the intersection between these objects. Simply stated, not adjoining this object to the maximal object excludes instances of all deposit accounts held by a particular customer at the bank where his loan account is held.

The algorithm is as follows:

For each candidate object (or relation scheme), another object is tested for losslessness with it; if this test is successful, the union becomes the new candidate object. When no further objects remain to be tested, the object is a maximal object.

The conditions for adding to M (where attr(M) is the set of attributes in the object M) are:

i) (P ∩ attr(M)) ➡ P, or
ii) (P ∩ attr(M)) ➡ attr(M), or
iii) P - attr(M) is disconnected from attr(M) - P when P attr(M) is deleted

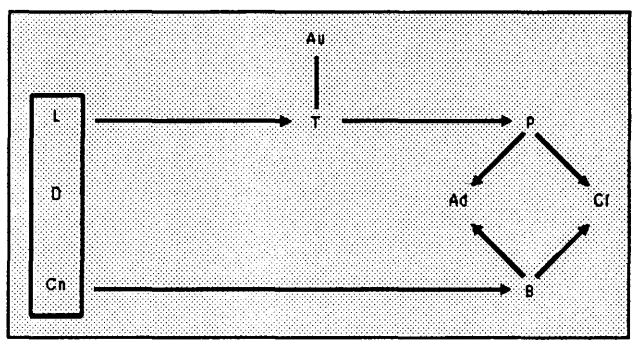

Figure 3

Referring to figure 3, start with TP; TAu can be added using iii) above since by deleting T, Au is disconnected from P; TL can be added under i) since the intersection of TL and TPAu (T) determines L.

A small restriction over what attributes can be included in a query using maximal objects, is that the objects over which they range have some 'strong' connection; e.g. that the joins are lossless.

Maximal objects do not affect acyclic databases in any way, because only one maximal object exists (the complete set of objects) [24].

Maximal objects must satisfy the containment condition which is:

where Y contains X, $\pi X(Y) \supseteq X$.

This ensures that the roles of the attributes in the two objects are consistent, and as a result only those objects containing the attributes required need be considered. This is a result of the tuples for the subset also

appearing in the superset, therefore the need to take the union of the maximal objects is removed [13, 19].

The generator should also be faithful, whereby any stored relation covering the set of attributes X is equal to the maximal object X. The condition is: r(X) = X where r(X) is the stored relation having the attributes in maximal object X [17].

Without this property, a tuple could be inserted into a relation, and not be identified (or retrieved again) using the window.

Objects are integral if windows on any subset are consistent. This means that any given attribute is contained in the object in question, without taking the union of projections onto it from many windows. An example, using the scheme described earlier, where some of the relation shemes are:

BOOKS(TITLE,AUTHOR,PNAME,LC-NO)
LOANS(CARD-NO,LC-NO,DATE)
BORROWERS(NAME,ADDR,CITY,CARD-NO)

the object (and relation scheme)

LOANS(CARD-NO,LC-NO,DATE)

will not return a set of all possible CARD-NO's. For this, the relation BORROWERS will have to be used.

System/U uses an algorithm for binding similar to that described for 'windows' above. The input is the query, the relation schemes, the objects, and the maximal objects and the output is an algebraic expression which is the union of terms. The difference is that instead of using the universal relation, the maximal objects are used [11]. The PIQUE system also uses this approach [19].

However, such automatically- constructed maximal objects may not give the intuitively correct answers in all cases; and the interpretation may not be what is required. If dependencies exist which are not immediately obvious from the given set of dependencies (e.g. embedded multivalued dependencies), the lossless joins would separate these attributes into separate sets, possibly contrary to the user's requirements.

Syntactically, queries on different roles will appear to be the same, and use the applicable maximal objects. They remove the requirement of explicit knowledge of the database structure from the user. Alternative maximal objects provide alternative roles for attributes. When no maximal object contains all the attributes of a query, a message should be produced; it could be that the relationship was not considered meaningful to the database designer. Maximal objects can be automatically constructed as mentioned above, but it is felt that the manual selection of maximal objects at design time is probably more useful, and should only change when the database schema is altered.

The algorithm for using the established maximal objects at query time is:

- find maximal objects which contain the attributes in the query,
- form the natural join of all of the relations in the maximal objects,
- construct the relation using the joins,
- apply selection, apply projection.

7. Conclusion

The algorithms discussed above in respect of representative instances and windows work satisfactorily for the simplest 'objects' (usually the fundamental relationships of the database). With a tree of objects, the algorithm finds the smallest subtree connecting all the attributes (the minimal connection) [24]. However with cyclic schemes, the maximal object approach is necessary, due to the inconsistency between contexts under such conditions.

The use of user views as predefined contexts for user selection has not been extensively applied, possibly due to the discretion allowed the user in choosing between potentially many views, and difficulties arising from having a large number of names.

The physical storage of the universal relation, as implemented in the FIDL system also has not been popular, mainly due to the computations at update time to produce tuples of the universal relation. Extra I/O operations, excessive storage requirements, and security maintenance difficulties also occur.

Operationally, the systems differ in complexity; particularly in the generation of contexts. None appear to perform computations for the binding step at query time, the definition of contexts being at database design time, and often with the intervention of the designer. Some systems rely on manual context definition (Q), while others, using inter-attribute dependencies, produce them automatically (System/U, PIQUE).

To dispense with the universal role assumption (where unique names are used for individual roles), it may be necessary to define these separate roles independently of the database scheme, i.e. use common names for the roles, but disambiguate them by means of the context in which they are used.This highlights one of the central problems with the use of a true universal relation system; that the semantic interpretation of attributes cannot be easily performed, and relies upon manual input, typically at design time.

References

[1] T R Addis, [1982], A Relation-Based Language Interpreter for a Content-Addressable File Store. *ACM Transactions on Database Systems*, July 1982, 125-163.

[2] S Berman, [1987], A semantic data model approach to logical data independence. in *4th South African Computer Symposium, 1987.* Proc., pp329-342.

[3] E F Codd, [1974], Seven steps to rendezvous with the casual user. *in* Klimbie JW and Koffeman KL (eds.) *Data Base Management.* Amsterdam, North-Holland, 1974.

[4] E F Codd, [1979], Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems* 4, 397-434.
[5] C J Date, [1982], *An Introduction to Database Systems* 3rd edition. Addison-Wesley, Reading, Mass., 1982.
[6] A D'Atri, M Moscarini, and N Spyratos, [1983], Answering queries in relational databases. in *ACM SIGMOD - Proceedings of Annual Meeting (Database Week), 1983.* edited by D J Dewitt and G Gardarin. ACM, 1983. 173-177.
[7] R Fagin, A O Mendelzon and J D Ullman, [1982], A simplified universal relation assumption and its properties. *ACM Transactions on Database Systems* 7(3), 1982, 343-360.
[8] P Honeyman, R E Ladner, and M Yannakakis, [1980], Testing the universal instance assumption. *Inf. Process. Lett.*, 10(1), 14-19.
[9] W Kent, [1981], Consequences of assuming a universal relation, *ACM Transactions on Database Systems* 6(4), 539-556.
[10] W Kent, [1983], The universal relation revisited (Technical Correspondence). *ACM Transactions on Database Systems* 8(4), 644-648.
[11] H F Korth, G A Kuper, J Feigenbaum, A Van-Gelder and J D Ullman, [1984], System/U: a database system based on the universal relation assumption. *ACM Transactions on Database Systems*, 9(3), 331-347.
[12] D Maier, [1983], *The theory of Relational Databases*. Computer Science Press, Rockville, Md., 1983.
[13] D Maier, S C Rozenshtein and D S Warren, [1983], Windows on the world. in *ACM SIGMOD - Proceedings of Annual Meeting (Database Week), 1983.* edited by D J Dewitt and G Gardarin. ACM, 1983. pp 68-78.
[14] D Maier and J D Ullman, [1983], Maximal objects and the semantics of universal relation databases. *ACM Transactions on Database Systems* 8(1), 1-14.
[15] D Maier, J D Ullman, and M Y Vardi, [1983], The revenge of the JD. in *ACM SIGACT-SIGMOD Second Symposium on Principles of Database Systems, 1983.* Proc. pp 279-287.
[16] D Maier, J D Ullman and M Y Vardi, [1984], On the foundations of the universal relation model. *ACM Transactions on Database Systems* 9(2), 283-308.
[17] D Maier, D Rozenshtein, and J Stein, [1985], Representing roles in universal scheme interfaces. *IEEE Trans. on Software Engineering* se-11(7), 644-652.
[18] D Maier, S C Rozenshtein and D S Warren, [1986], Window functions. *Advances in Computing Research* vol. 3. JAI Press, Greenwich, Conn.
[19] D Maier, D Rozenshtein, S Salveter, J Stein and D S Warren, [1987], PIQUE: A relational query language without relations. *Inform. Systems*, 12(3), 317-335.
[20] J Martin, [1983], *Managing the data-base environment.* Englewood Cliffs, Prentice-Hall, 1983.
[21] S L Osborn, [1979], Towards a universal relation interface. in *Proceedings Int. Conf. on Very Large Data Bases*, 52-60.
[22] Y Sagiv, [1983], A characterization of globally consistent databases and their correct access paths. *ACM Transactions on Database Systems*, 8, 266-286.
[23] J D Ullman, [1982], *Principles of Database Systems*, 2nd ed. Computer Science Press, Rockville, Md., 1982.
[24] J D Ullman. The U.R. strikes back, [1982], in *ACM Symposium on Principles of Database Systems, 1982.* Proc., 10-22.
[25] J D Ullman, [1983], Universal relation interface for database systems. in *Proceedings IFIP 1983*, edited by REA Mason, North-Holland, 243-252.
[26] J D Ullman, [1983], On Kent's 'Consequences of assuming a universal relation' (Technical Correspondence). *ACM Transactions on Database Systems* 8(4), 637-643.
[27] J D Ullman, [in pub.], *The Universal Relation as a User Interface*. Chapter 17 draft for new book. 1988
[28] M Y Vardi, [1988], The universal-relation data model for logical independence. *IEEE Software*, March 1988. pp 80-85.

Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research papers. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as a Communications or Viewpoints. While English is the preferred language of the journal papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted **in triplicate** to the editor.

Form of Manuscript

Manuscripts for review should be prepared according to the following guidelines.

- Use double-space typing on one side only of A4 paper, and provide wide margins.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be on separate sheets of A4 paper, and should be numbered and titled. Figures should be submitted as original line drawings, and not photocopies.
- Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin, if they are not clear in the text.
- References should be listed at the end of the text **in alphabetic order** of the (first) author's surname, and should be cited in the text in square brackets. References should thus take the following form:

  [1] E Ashcroft and Z Manna, [1972], The translation of 'GOTO' programs to 'WHILE' programs, *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.

  [2] C Bohm and G Jacopini, [1966], Flow diagrams, Turing machines and languages with only two formation rules, *Comm. ACM*, **9**, 366-371.

  [3] S Ginsburg, [1966], *Mathematical theory of context free languages,* McGraw Hill, New York.

Manuscripts *accepted* for publication should comply with the above guidelines, and may be provided in one of the following formats:

- in a **typed form** (i.e. suitable for scanning);
- as an **ASCII file** on diskette; or
- as a **WordPerfect**, **TEX** or **LATEX** or file; or
- in **camera-ready** format.

A page specification is available on request from the editor, for authors wishing to provide camera-ready copies. A styles file is available from the editor for Wordperfect, **TEX** or **LATEX** documents.

Charges

Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect scanning, typesetting, reproduction and other costs. Currently, the minimum rate is R20-00 per final page for camera-ready contributions and the maximum is R100-00 per page for contributions in typed format.

These charges may be waived upon request of the author and at the discretion of the editor.

Proofs

Proofs of accepted papers will be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Note that, in the case of camera-ready submissions, it is the author's responsibility to ensure that such submissions are error-free. However, the editor may recommend minor typesetting changes to be made before publication.

Letters and Communications

Letters to the editor are welcomed. They should be signed, and should be limited to about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement

Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.

South African Computer Journal

Number 4, March 1991
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 4, Maart 1991
ISSN 1015-7999

Contents