CHEOPS
A PROJECT FOR EVALUATING
ANSI/SPARC ARCHITECTURE

by

A. Klug

Computer Sciences Technical Report #349

April 1979

CHEOPS[1]

A Project for Evaluating ANSI/SPARC Architecture[2]

A. Klug

Computer Sciences Department

University of Wisconsin

Madison, WI 53706

---

[1] Cheops, Egyptian king of the 4th dynasty (~2900 B.C.), builder of the Great Pyramid at Giza. We chose this name because of the "double pyramid" structure of an ANSI/SPARC database management system[SeAl].

[2] This research was supported in part by the Mathematics Research Center of the University of Wisconsin-Madison.

Table of Contents

1. Introduction

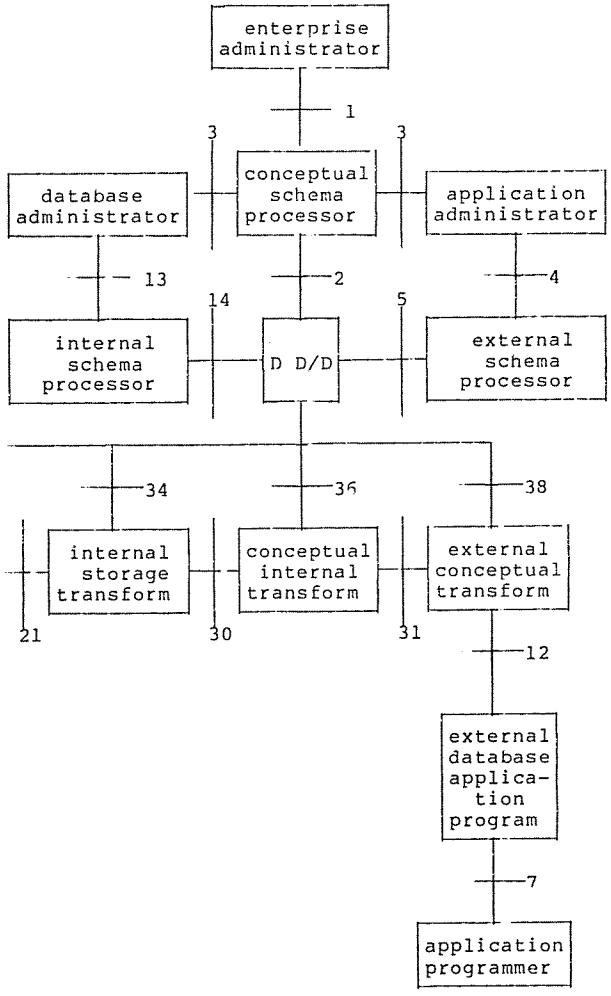
The standardization of database management systems is no longer a topic for idle speculation. Already subcommittees of the American National Standards Committee on Computers and Information Processing (ANSI/X3) are proceeding with standardization efforts for Codasyl database facilities. Back in 1975, the Standards Planning and Requirements Committee (SPARC) of ANSI/X3 established a study group to investigate the overall standardization problem for database management systems. This study group proposed a framework for database management systems within which the standardization of specific DBMS components could be discussed precisely[ANSI75,ANSI77]. In August of 1978, ANSI/X3/SPARC established a new database study group in response to the continued pressure for some guidelines and direction for database standards. One of the main responsibilities of this currently active study group is to 'develop further the concepts of the former SPARC Study Group Report". That is, the framework for database management systems will again be a focal point of interest. In view of this clear interest in the so-called ANSI/SPARC framework, it is important to understand the technical implications of basing DBMSs and standards for them on this framework. Up to the present time, there has been research only on isolated aspects of the framework an overall, comprehensive investigation including an implementation has been lacking. The Cheops project described in this document has as its goal the understanding of the unique technical implications of having DBMSs conform to the ANSI/SPARC framework.

This document consists of four parts. The first is this introduction which briefly motivates the need for the research. Section 2 gives a description of the DBMS framework as defined by the first study group. The third section discusses the Cheops project. Subsections include statements of overall project goals, necessary theoretical work completed and to be done, and a description of the interfaces and languages in Cheops. Section 4 will give a summary and conclusions.

2. The DMBS Framework

2.1 Description of the Framework

The framework proposed by the ANSI/X3/SPARC Database Study Group consists of a number human roles, processing functions and interfaces. The most important of these are represented in the following diagram:



The main components of the framework are a set of persons in roles, (enterprise administrator, database administrator, application administrator are the major ones) a set of processing

functions and a set of interfaces among elements of these sets. The framework is partitioned into three realms or levels: the internal, the conceptual and the external. At each of the internal and conceptual levels there is one schema describing the database. At the external level there are any number of external schemas, each describing some part of the database.

The internal schema contains performance and other computer-oriented information. For example, an internal schema may have as objects direct access files, indices and pointer arrays. It is sometimes said that the internal database is what is stored on the computer. This may or may not be true, depending on one's point of view and also on the system architecture. For example direct access files may be implemented as B-trees which in turn reside on a paged data set which, on the disc, contains error checking bits. The framework draws the line (which is expected to move with time) between database system concerns and computer system concerns. In any case, the internal level is the lowest "abstract machine" level directly involved in database management. The database administrator, who manages the internal schema, can view the conceptual schema through interface 3 and manipulates the internal schema and the conceptual-internal mapping (for tuning) through interface 13.

If one considers the framework as describing an "onion" of nested machines, then the conceptual level will be the next layer out from the internal level. The objects declared in the conceptual schema model the entities of interest to the enterprise (the defined environment in which the database system will operate). These objects are not oriented towards efficiency or

towards user requirements. The objects declared in the conceptual schema are to give an overall, unbiased description of the enterprise. The imposition of the conceptual level between the external and internal levels also allows both of these latter levels to evolve, for their own reasons, without one unnecessarily affecting the other. The conceptual level provides a mechanism for the centralized control over the use and content of the database. The enterprise administrator manages the conceptual schema through interface 1.

The outermost, external level provides application-oriented views of the database. The role of this level is to provide each application (e.g., payroll, marketing, research and development) or application family with the portion of the database it needs in a form most suitable to it. The most suitable form, that is, data model, may in principle consist of Fortran arrays or complicated semantic networks or any other structure class. An application administrator manages external schemas and external-conceptual mappings through interface 4 and views parts of the conceptual schema through interface 3.

The proposed framework provides a structure within which multiple views and multiple data models can be supported. It allows a high degree of data independence and control over the database and facilitates dynamic reorganization (data translation). It also can form a basis for well-structured distributed databases.

2.2 Issues

It is hoped that the framework described above can provide a sound basis for DBMS standardization, for data independence and for DBMS control. However, numerous technical issues must be understood before the framework can be fully applied. This section describes some of these issues and notes what our research program will contribute towards their understanding.

2.2.1 Conceptual Level Constructs

The conceptual schema and conceptual level constructs in general have been the subject of numerous research reports and even an entire ISO committee (ISO/TC97/SC5/WG3). There has been debate on whether the objects defined in the conceptual schema should be binary relations[BrPP], irreducible relations[HaOT], n-ary relations, entities[Chen], roles[Bach] and so on. The Cheops project will show that this debate is irrelevant, just as the debate on which of the hierarchical, network and relational models is best for users is irrelevant. We will show that the only real requirement for a conceptual model is that it is precisely defined and that it has sufficient power to model the referenced subset of the real world. The Cheops conceptual data model will be relational. A conceptual schema will contain declarations of (normal form) n-ary relations, relation keys and foreign key constraints. If it is necessary, this model can be extended to include more constraint types. However, the constructs from the entity-relationship model, the role-model and

from other proposals will not be added directly to the model and will not be needed. If an enterprise administrator prefers to work with these concepts, it will be a simple matter to attach an interface to the conceptual schema processor which will present the conceptual schema in the desired format. We do not need to use binary or irreducible relations because by allowing null values in the ranges of the domains, our model will be mathematically equivalent to models with only binary relations or irreducible relations.

Another issue which has attracted attention is the question of what data manipulation operations need to be provided at the conceptual level. First of all, we note that there must be some data manipulation language at the conceptual level, since dynamic data independence[Melt] requires mapping through this level. We will show that a simple relational language consisting of retrieves, inserts, deletes and updates will be sufficient for a conceptual level data manipulation language.

Locking and access control at the conceptual level is also necessary. A predicate lock mechanism[EGLT] can operate using information in the structure mappings. That is, each application expresses the locks it needs in terms of its external view, and the schema mapping is used to translate these locks into locks on conceptual level objects. Deciding what access control primitives are necessary is a more involved problem (see next section).

There has been some concern expressed that multiple external data models will interfere with each other[PaPe], the problem

being called "cross data model interference". Some responsibility for this has been laid to the conceptual level constructs. Cheops will demonstrate that there is no such problem. In fact, since the different external data models are all defined by a translation to a canonical relational model (see Section 3.1.3), all models essentially look the same. There may be problems in verifying the correctness of external-conceptual and conceptual-internal mappings, but after this has been done, we need only maintain consistency through traditional locking protocols.

2.2.2 Access Control

Another class of problems within the framework relates to the use of mappings for database control.

Consider the following conceptual schema in which employees are assigned to projects which are associated with departments:

```
Emp(E#,Name,J#)
Proj(J#,Name,Security#,D#)
Dept(D#,Name,Mgr)
```

An external schema might contain the relation:

```
Employee(E#,Name,Mgr),
```

and users of this schema might be given read access to the E# and Name fields of the Emp conceptual relation and to the Mgr field of the Dept conceptual relation. Now consider the following mapping:

```
Employee <- select Emp.E#,Emp.Name,Dept.Mgr
                   where Emp.J# = Proj.J# and
                         Proj.D# = Dept.D#
```

According to this mapping, the query

```
select Employee.Mgr where Employee.Name='Wong'
```

would be translated as:

```
select Dept.Mgr
       where Emp.Name = 'Wong' and
             Emp.J# = Proj.J# and
             Proj.D# = Dept.D#
```

This query requires access to the Proj relation. Yet as long as the external queries are translated by the above mapping, the user will never see any part of the Proj relation. There is a problem, then, to devise the appropriate access control formalism which will allow properly translated queries and updates access to conceptual objects to which the user nominally has no access.

Access control in Cheops is based on the idea of a conceptual subschema. The enterprise administrator defines a conceptual subschema by specifying which domains of which relations are to appear. In addition, horizontal subsetting can be specified by restriction clauses. Everything visible in a conceptual subschema has retrieval access. Modify access must be explicitly declared. Providing access to conceptual level objects only through conceptual subschemas yields some access control, but in some cases there must be access for mappings but not for the external application programs. This may require a general mapping capability from the conceptual schema to a conceptual subschema.

2.2.3 Mapping Languages

A mapping consists of two parts: a structure or schema mapping and an operation mapping. Structure mappings define the correspondence between database states, whether conceptual-external, internal-conceptual or internal-external. They provide the information needed for translating queries, that is, retrieval requests and for translating lock requests. Operation mappings define the interpretation of operations at one level in terms of operations at a lower level (external as conceptual, conceptual as internal, external as internal). Experience is needed in designing and implementing mapping languages. For external-conceptual structure mappings, the mapping language is similar to a query language. We intend to show further that there only needs to be one structure mapping language no matter how many external data models there may be. We need only one external-conceptual mapping language because we are defining the semantics of the three external data models being used in terms of a single "canonical" relational model. More details are given in Section 3.1.3. Our single structure mapping language will be based on relational algebra. For operation mappings, the situation should be much the same. As with structure mappings, we intend to define only one operation mapping language which will be used with all external data models. Inserts, deletes and updates based on relational algebra are the only operations that will be needed, but the language constructs needed to combine these operations with sufficient flexibility to support the various data model operations is not decided.

Our language will at least have conditional statements using some test on the database state or the operation to be translated. More details are given in Section 3.1.4.

2.2.4 Schema and Mapping Processors

Key elements in the framework are the processors whose input consists of schemas at the three levels and mappings between them. These processors must be able to tell when schemas are self-consistent. They must be able to tell when schema mappings will ensure that the constraints in the base schema will imply all constraints in the other schema. They must recognize when operations will be correctly interpreted by the operation mappings. These requirements imply that the processors must incorporate some sophisticated algorithms to make the necessary checks. The algorithms must be both sound and complete. Soundness is necessary, since otherwise incorrect results would ensue. Completeness is also necessary because we do not want correct mappings to be rejected by the system. As part of the Cheops project, algorithms are being developed for use in the schema and mapping processors. More details are given in Section 3.2.

2.2.5 Data Independence - Schema Changes

There are very interesting problems involved in providing data independence and in managing schema changes. An administration protocol must be developed for locking schemas, modifying schemas, locking and modifying mappings and for repopulating data

objects. For example, one question which must be considered is whether there is any difference between schema changes being effected by permutations to a single schema or by always defining a new schema. After a schema change there should be a processor which can decide which mappings do not need to be changed, which mappings can absorb the change with a suitable redefinition and which mappings cannot absorb the change, requiring the associated schema and/or application program also to be changed. There are several kinds of data independence a system can support[Melt]: static with early binding, static with late binding and dynamic. With Cheops' conceptual data language, even dynamic data independence can be provided. Although work has been done on how to modify application programs to accommodate a schema change[NaSu], the goal of the ANSI/SPARC framework is to avoid these changes whenever possible.

2.2.6 Efficiency Considerations

There is no doubt that interpretation through the conceptual level will adversely affect system performance. Whether or not a system with the flexibility of Cheops is ever commercially viable will depend on the ability to "compose" external-conceptual and conceptual-internal mappings. That is, the system will have to be able to take an external-conceptual mapping and a conceptual-internal mapping and automatically produce an efficient external-internal mapping. The processors for doing this would use the techniques already known for access path selection, but there would need to be development of ways to

communicate the needs of the application program to the DBA, who controls the internal schema. A long range goal of the Cheops project is to show that the only real additional cost in supporting a multiple-view, multiple-data-model DBMS with three schema levels is the cost of providing fast access paths — a cost which is not new to database management.

2.2.7 Other Data Models

The data models used at the external level (see Section 3.1) in Cheops were chosen as a result of a compromise: They faithfully represent the essential features of the three major approaches to data modelling, yet they have compatible data types and constraint types, and their query syntax is similar. We want to investigate the problems arising in supporting different data models, but we do not want to make the initial task any more difficult than necessary. However, the fact must be recognized that the data models in use in today's database management systems are not so compatible: There are disparate data types; constraint types may not be consistent; data languages are both record-oriented and set-oriented. Eventually, the Cheops system will be extended to accommodate more diverse data models, and we will be able to study how inconsistencies can be minimized or neutralized.

2.2.8 Error Handling

It is generally recognized that external views should let the user see the database in the form most useful to him/her. A corollary of this, one which does not seem to have received much attention, is that error messages from the conceptual or lower levels should also be tailored to the particular view. For example, suppose a user issued the following statement (which uses a schema given in Section 3.1.3):

```
for each dept having name = 'comp sci'
   and ancestor school (having name = "UW"):
insert course having num = 401 and descr = 'op sys'
```

The "meaning" of this statement is the relational algebra statement:

```
insert course (school[1='UW'][1=1]course[2='comp sci'])
              [1,4] @ {<401,'op sys'>}
```

which would probably be simplified to:

```
insert course  <'UW','comp sci',401,'op sys'>
```

If there were no dept tuple with school-name = 'UW' and name = 'comp sci', then the system would return an error:

```
error -- violation of constraint:
         subset course(school-name,dept-name) in
           dept(school-name,name)
```

However, this message would not be meaningful to the user of the hierarchical view. The message this user should get is:

```
error -- no such dept parent
```

Thus we must have a methodology for translating errors just as we translate queries.

3. The Cheops Project

Having DBMSs conform to the ANSI/SPARC framework can provide many benefits, but as we have seen, there are problems on which we at least must have a hand-hold before these benefits accrue. The goal of the Cheops project is to help understand the above problems.

3.1 Language Descriptions

In this section we discuss the data and mapping models used in Cheops.

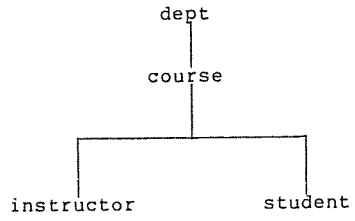
The external level in Cheops supports three different data models: NDL, which is a network model, RDL which is a relational model and HDL, which is hierarchical. All three models are set-oriented. The conceptual level supports a relational model and the external-conceptual mapping model is relational. A BNF syntax is given for these data models in the appendix.

3.1.1 The Hierarchical External Model

The hierarchical model is patterned after HQL[Fehd]. A sample schema is the following:

```
dept (dno:integer, name:string(20)) key dno
  (  course (cno:integer, name:string(25), credits:integer)
          key cno
       (  instructor (name:string(25), office:string(10)) key name
          student (sno:integer, name:string(25), major:string(4))
             key sno
       )
  )
```


This schema declares a hierarchy of four nodes with the structure:



A typical query against this schema is the following:

```
for each course having ancestor dept (having name='comp sci'):
   list cno, name, (name for each student having major='med')
```

Modification commands, e.g., deletions, have a similar format:

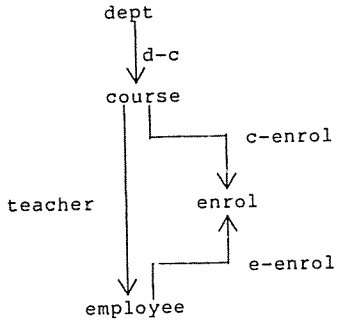
```
for each course having ≤ 5 student: delete
```

3.1.2 The Network External Model

The network model, NDL, has schemas such as the following one:

```
record dept (dno:integer, name:string(20)) key dno
record course (cno:integer, name:string(20)) key cno
record employee (eno:integer, name:string(25), sal:integer)
       key eno
record enrol ()
set d-c
        owner dept
        member course
        mandatory
set teacher
        owner course
        member employee
        optional
set c-enrol
        owner course
        member enrol
        mandatory
set e-enrol
        owner employee
        member enrol
        mandatory
```

The data-structure diagram for this schema is the following:



A typical query using this schema would be:

```
for each employee having owner teacher (having > 35 c-enrol):
    list name, dno of (teacher,d-c)
```

A modification command, for example, a deletion, has a similar format:

```
for each course having ≤ 30 c-enrol: delete permanent
```


3.1.3 The Relational External Model and Canonical Model

The relational external model RDL contains relation declarations, key declarations and subset constraints (foreign key constraints). This model is also the canonical relational model used to "define" HDL and NDL.

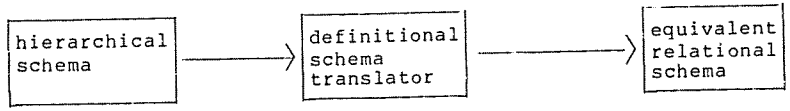
An example RDL schema is the following:

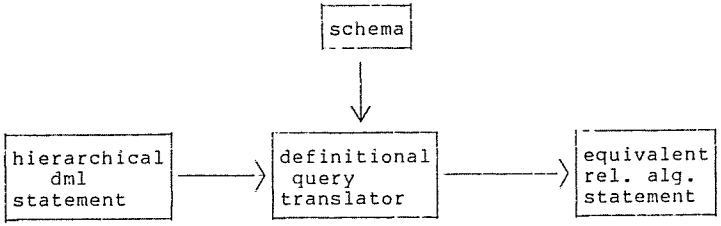
```
employee(name:string(25), sal:integer, dept:integer)
dept(num:integer, mgr:string(25), floor:integer)
key of employee is name
key of dept is num
subset employee(dept) in dept(num)
subset dept(mgr) in employee(name)
```

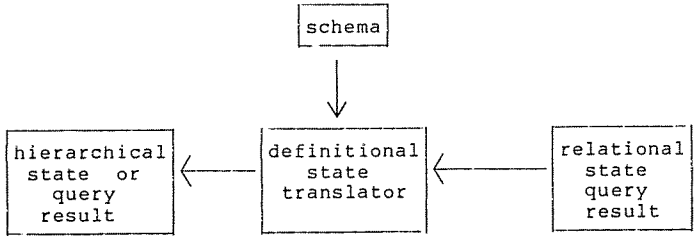
The data manipulation language for RDL is base on relational algebra[Codd]. Queries have the form:

```
retrieve  ((employee[dept=num]dept)[mgr=name]employee)
delete from employee  (employee[dept=name](dept[floor='2']))[1,2,3]
```

As the canonical model, RDL is used to give a common semantics to HDL and RDL. For example, the definition of the hierarchical model HDL is embodied by three algorithms as follows:







The following examples will illustrate these procedures. The languages used are HDL and RDL.

```
school(name,state) key name
  (  dept(name,chair) key name
       (  course(num,descr) key num
       )
  )
```

——————schema translator——————〉

```
school(name,state)
dept(school-name,name,chair)
course(school-name,dept-name,name,descr)
key of school is name
key of dept is school-name, name
key of course is school-name, dept-name, name
subset course(school-name,dept-name) in
       dept(school-name,name)
subset dept(school-name) in school(name)

for each dept having name='comp sci' and
        ancestor school (having state='calif'):
list chair, (descr for each course)
```


——————dml translator——————〉

```
retrieve  (dept[2='comp sci'][1=1]
           (school[2='calif'])[1,2=1,2]
           course)[3,9] order 3,9
```

```
for each school having state='wisc': delete
```

——————dml translator——————〉

```
       delete from course course[1,2=1,2](dept[1=1](school[2='wisc']))
delete from dept  dept[1=1](school[2='wisc'])
delete school[state='wisc']
```

chair	descr
smith	artificial intelligence
smith	data structures
smith	programming languages
wong	artificial intelligence
wong	differential equations
wong	programming languages

——————result translator——————〉

chair	descr
smith	artificial intelligence
	data structures
	programming languages
wong	artificial intelligence
	differential equations
	programming languages

Similar examples can be given for NDL.

3.1.4 External-Conceptual Mapping Model

As the previous section has shown, we need only one external-conceptual mapping model, and it is based on RDL. A schema mapping, which is similar to a collection of view definitions, is specified by associating with each relation in the canonical external schema a relational algebra expression over relations

in the conceptual schema. A BNF syntax is given in the appendix. Here we will give an example:

```
canonical external schema:
        employee(emp#,name,addr,sal,mgr)
        key of employee is emp#
        subset employee(mgr) in employee(emp#)

conceptual schema:
        emp(emp#,name,sal,dept)
        person(name,addr)
        dept(dno,mgr)
        key of emp is emp#
        key of person is name
        key of dept is dno

mapping:
        employee = ((emp[name=name]person)
                    [dept=dno]dept)[emp#,name,addr,sal,mgr]
```

Operation mappings associate with each operation on each canonical view relation a compound operation on the conceptual schema. Hence, an operation mapping will have at most 3*n entries if there are n relations in the canonical external schema. The syntax for operation mappings is given in the appendix. Here we will give an example:

```
canonical external schema:
        employee(emp#,addr,sal)
        key of employee is emp#

conceptual schema:
        emp(emp#,addr,sal,dept)
        dept(dno,loc,mgr)
        key of emp is emp#
        key of dept is dno
        subset emp(dept) in dept(dno)
        subset dept(mgr) in emp(emp#)

schema mapping:
        employee = (emp[dept=dno]dept[mgr='smith'])[emp#,addr,sal]

operation mapping:
        delete employee x  ⟶
           if x.sal > 10k  then delete emp x
           else replace emp x dept=null
```


3.1.5 The Internal Model

The internal model of Cheops will initially contain the basic file types. Eventually, the internal level will be an important focus of work, but for the present, we are concentrating efforts on the external and conceptual levels.

3.2 Processing Functions

As we noted in Section 2.2.4, the schema/mapping processors require algorithms to recognize consistent schemas and correct mappings. The conceptual level and the canonical form of the external level will both be based on the relational model, and the structure mappings will be defined by relational algebra. The external-conceptual mapping processor will receive statements identifying canonical external relations with relational algebra expressions involving conceptual relations. For each external relation and its associated relational algebra expression, the mapping processor must determine if the constraints on the relation declared in the external schema will be consequences of constraints in the conceptual schema over relations appearing in the expression. A very simple example is the following:

```
external schema:  emp(eno,addr,mgr)
                  key of emp is eno

conceptual schema:  empl(eno,addr deptno)
                    dept(deptno,loc,mgr)
                    key of empl is eno
                    key of dept is deptno

mapping:  emp = (empl[deptno=deptno]dept)[eno,addr,mgr]
```

The mapping processor must recognize that the key declaration in the external schema is a consequence of the key declarations in the conceptual schema the mapping.

The mapping processor must also be able to reject mappings when they will not result in valid external constraints. The next example illustrates this:

external schema: part(pno,wt,price)
key of part is pno

conceptual schema: car-part(cpno,wt,price)
boat-part(bpno,wt,price)
key of car-part is cpno
key of boat-part is bpno

mapping: part = car-part ⊔ boat-part

The key constraint in the external schema is not a consequence of constraints in the conceptual schema, and it therefore will not always hold.

We must not only have an algorithm for calculating the valid constraints on a relational algebra expression, we must also prove that the algorithm calculates exactly the valid constraints — that it misses none and that it does not calculate any false ones.

For the case of constraints consisting of functional dependencies we have developed an algorithm which does just this[Klug] We are investigating extensions to include subset constraints.

A complete mapping also has an operation part, and the mapping

processor must also test the correctness of operation mappings. Given a structure mapping already shown to be correct, the mapping processor must test a given operation mapping to see if it will always map inserts and deletes on the canonical external relation to operations on the conceptual relations such that the desired operation will appear to have occurred via the structure mapping. As in the case of structure mappings, there needs to be an algorithm which will recognize all and only those operation mappings which will correctly interpret the external operations according to the given structure mapping. We also have an algorithm for a weak form of the problem.

3.3 Human Interfaces

There are five major interfaces between human roles and the DBMS. Using any of these will, in general, require a password or an authorized user id. This section describes these interfaces and the commands available at each[3].

[0] Cheops <db-name>

This is the top-level command given to the operating system (unix). After this command is entered, the user may request access to any of the following six interfaces:

super

ea

dba

aa <appl>

[3] The number in brackets "[ ]" indicates the level of the command. Some commands have important parameters explicitly given.

ap <es>

usr <ap>

[1] super

This is a "super user" interface used to modify top-level authorizations and passwords and to modify system routines.

[1] ea

This is the enterprise administrator interface. The major commands at this interface are the following:

define-cs (conceptual schema)

edit-cs

compile-cs

establish-cs

disestablish-cs

define-css (conceptual subschema)

edit-css

permit-css

permit-cs

[2] define-cs, edit-cs

These commands manipulate the source forms of conceptual schemas, including displaying the conceptual schema.

[2] compile-cs

This command will produce the object form of a conceptual schema. Tests for self-consistency are performed.

[2] (dis)establish-cs

The establish-cs command makes the specified conceptual schema the current conceptual schema for the database.

Disestablish-cs removes the currency. These commands provide a tentative means for altering the current conceptual schema for a database. To modify the schema the following procedure can be used: Use the define-cs and edit-cs commands to produce a source form of the modified conceptual schema. Compile the new schema. Disestablish the old schema, and establish the new one. A more dynamic mechanism for modifying conceptual schemas will be one of the issues investigated in the Cheops project.

[2] define-css, edit-css, compile-css

These commands manipulate conceptual subschemas, which form windows for application administrators to view the conceptual schema. They also contain update access specifications. There will be one conceptual subschema for each application.

[2] permit-css

This command is used to give and revoke access to conceptual subschemas to specified application administrators.

[2] permit-cs

This command will allow access to a conceptual schema by a database administrator.

[1] dba

This is the database administrator interface. The major commands at this interface are the following:

display-cs

define-is (internal schema)

edit-is

compile-is

reorganize

define-map

edit-map

compile-map

establish-map

disestablish-map

[2] define-is, edit-is, compile-is

These commands have the same functions as their analogs under the ea interface.

[2] reorganize

This command essentially causes an internal application program to be run. (An internal application program is a program which references an internal schema.) For example, the reorganize command might transpose a file, build an index or compress a file. This command is used with the establish-map and disestablish-map commands to tune the database.

[2] define-map, edit-map

These commands are for manipulating the source form of conceptual-internal mappings.

[2] compile-map

This command involves several functions. First, the mapping is checked for correct syntax. The mapping is checked for consistency, that is, that every constraint in the conceptual schema will be maintained by the internal schema constructs according to the associations in the mapping. The mapping is checked to ensure that every relation in the conceptual schema which represents part of the computerized database is

actually mapped to some object or objects of the internal schema.

[2] establish-map, disestablish-map

These commands make a mapping current and remove a current mapping, respectively.

[1] aa <appl>

This is the interface through which an application administrator interacts with the database. The parameter to the command specifies which application is desired. (We assume this is the same as the name of a conceptual subschema.) An application consists of a conceptual subschema, a number of external schemas, a number of mappings and a number of application programs along with necessary authorization information. The major commands at this interface are the following:

display-css

define-cs <dm>

edit-cs

compile-cs

display-cs

define-map

edit-map

compile-map

establish-map

disestablish-map

permit-es

permit-ap

[2] display-css

This command causes the conceptual subschema to be displayed to the application administrator so that external schemas may be written against it.

[2] define-es <dm>

This command is analogous to the define-cs and define-is commands except that the parameter specifies which data model is being used. The data model is RDL, NDL or HDL.

[2] edit-es

This command is analogous to the edit-cs and edit-is commands.

[2] compile-es

This command translates the source form of the external schema to an object form. This process includes checks for consistency of the schema. In addition, this interface will produce a canonical form of the external schema. For RDL schemas, the canonical form is the same as the original. For NDL and HDL schemas, the canonical form is a relational schema with the same information content.

[2] display-es

This command will display the canonical form of the specified external schema. This interface is needed when the application administrator writes a mapping from the external schema to the conceptual subschema.

[2] define-map, edit-map

These commands are used to manipulate the source form of

external-conceptual mappings.

[3] compile-map

This command translates the source form of a mapping to its object form. Besides the usual checks for syntactic correctness, this processor must ensure that all of the constraints in the external schema can be derived from constraints in the conceptual (sub)schema.

[2] establish-map, disestablish-map

These commands bind and unbind, respectively, an external schema to the conceptual schema with different mappings.

[2] permit-es

This command will allow application programmers to have access to an external schema in order to write application programs.

[2] permit-ap

This command gives access to application programs to users.

[1] ap <es>

This is the interface application programmers use for developing and testing application programs which run against the specified external schema. The major commands at this interface are:

define-ap

edit-ap

compile-ap

run-ap

These commands have the usual meaning.

[1] usr <ap>

This is the interface through which users invoke canned" application programs.

4. Summary and Conclusions

The ANSI/SPARC framework has potential for being a valuable tool in the future of database management. Before this value can be realized, we must understand the technological implications of the framework. This document has described some areas relative to the framework which need to be investigated. We have also described the Cheops project which will focus on these areas.

The Cheops database system will support network, hierarchical and relational external data models. The conceptual data model will be relational, and the external-conceptual mappings will be based on relational algebra. One of the functions of the external schema processor will be to translate from network and hierarchical external schemas to a relational form on which the mappings will be defined. Other processors will check the structure and operation mappings for correctness. If desired, there will be other interfaces to the conceptual schema so that it can be manipulated in terms of entities, roles and other constructs.

5. References

[Bach] Bachman C.W. "The Role Conceptin Data Models" 3rd International Conference on Very Large Databases, Tokyo, 1977

[BrPP] Bracchi G., Paolini P. and Pelagatti G. "Binary Logical Associations in Data Modelling", IFIP Working Conference on Modelling in Data Base Management Systems, North Holland, 1976

[Chen] Chen P.P.S. "The Entity-Relationship Model: Towards a Unified View of Data", TODS 1, pp. 9-36

[Codd] Codd E.F. "Relational Completeness of Data Base Sublanguages" Data Base Systems, R. Rustin (ed.), Prentice Hall, 1972

[EGLT] Eswaran K.P., Gray J.N., Lorie R.A. and Traiger I.L. "The Notions of Consistency and Predicate Locks in a Database System" CACM 19, 11, pp.624-633

[Fehd] Fehder P.L. "HQL: A Set-Oriented Transaction Language for Hierarchically-Structured Data Bases" ACM '74, Proceedings of the Annual Conference, 1974

[HaOT] Hall P., Owlett J and Todd S. "Relations and Entities" IFIP Working Conference on Modelling in Data Base Management Systems, North Holland, 1976

[Klug] Klug A. "Theory of Database Mappings", Ph.D. Thesis, University of Toronto, 1978

[Melt] Meltzer H.S. "Structure and Redundancy in the Conceptual Schema in the Administration of Very Large Data Bases", System for Large Data Bases (Lockeman and Neuhold, eds.), North Holland, 1976

[NaSu] Nations J. and Su Y.W. "Some DML Instruction Sequences for Application Program Analysis and Conversion", Proc. ACM-SIGMOD 1978

[PaPe] Paolini P. and Pelagatti G. "Formal Definition of Mappings in a Data Base" Proc.ACM-SIGMOD Conf. 1977

[SeAl] Senko M.E. and Altman E.B. "DIAM II: The Physical Device Level", System for Large Data Bases (Lockeman and Neuhold, eds.), North Holland, 1976

6. Appendix

```
/*   H D L   B N F   */

/* schema */

hdl-schema  -->  DEFINE HDL SCHEMA ID : sch-def
sch-def  -->  seg
seg  -->  ID (field-list) key-def desc
key-def  -->  KEY (name-list)
name-list  -->  ID | name-list, ID
desc  -->  (descl) | empty
descl  -->  seg | descl seg
field-list  -->  field-def | field-list, field-def
field-def  -->  ID type
type  -->  INTEGER null-opt | STRING (NUMBER) null-opt
null-opt  -->  * | empty

  /*  dml  */

stmt  -->  for : action  ;
for  -->  FOR EACH seg-name qual
qual  -->  HAVING quall
quall  -->  val-qual | (quall and-or quall)
and-or  -->  AND | OR
val-qual  -->  val rel-op val | comp-val rel-op comp-val
comp-val --> (val-list) | (val-list OF seg-name)
val-list  -->  val | val-list, val
val  -->  field-name | value | field-name OF seg-name
          func ( val OF seg-name qual )
value  -->  NUMBER | STRINGCONST
func  -->  AVE | SUM | COUNT
action  -->  list | insert | update | delete
list  -->  LIST [val-list] [comp-val]
insert  -->  INSERT seg-name qual
update  -->  UPDATE up-list
up-list  -->  up-el | up-list, up-el
up-el  -->  field-name = up-expr
up-expr  -->  val | (up-expr arith-op up-expr)
arith-op  -->  + | - | * | /
delete  -->  DELETE
rel-op  -->  < | <= | > | >= | == | !=
seg-name -->  ID
field-name  --> ID

/*   N D L   S Y N T A X   */

/* schema */

ndl-schema  -->  name-part rec-part set-part
name-part  -->  NDL SCHEMA name:
rec-part  -->  RECORD SECTION rec-def-list
rec-def-list  -->  rec-def | rec-def-list, rec-def
rec-def  -->  name (dom-list) key-def
```



```
key-def  -->  KEY (name-list) | empty
dom_list  -->  dom_def | dom_list, dom_def
dom_def  -->  name : type
type  -->  INTEGER null-opt | STRING(NUMBER) null-opt
null  -->  * | empty
set-part  -->  SET SECTION set-def-list
set-def-list  -->  set-def | set-def-list set-def
set-def  -->  SET NAME : name owner-def member-def
              mand-opt-def key-def
owner-def  -->  OWNER IS name
member-def  -->  MEMBER IS name
mand-opt-def  -->  MANDATORY | OPTIONAL | empty
key-def  -->  KEY IS name | empty

/* dml */

stmt  -->  for : action
for  -->  forl rec-spec
forl  -->  FOR EACH | FOR UNIQUE
rec-spec  -->  rec-name qual
qual  -->  HAVING quall
quall  -->  val-qual | ( quall and-or qual2)
path  --> path-el | path, path-el
path-el  -->  name | UP name | DOWN name
val-qual  -->  val rel-op val | comp-val rel-op comp-val
comp-val  -->  (val-list) | (val-list OF path)
val  -->  field-name | value | field-name OF path
          func(val OF path qual)
func  -->  AVE | COUNT | SUM
action  -->  list | insert | update |
             delete | add | remove
list  -->  LIST [val-list] [comp-val]
val-list  -->  val | val-list, val
insert  -->  INSERT AND ADD add-list | INSERT
add-list  -->  add-el | add-list AND add-el
add-el  -->  IN set-name qual
update  -->  UPDATE up-list
up-list  -->  up-el | up-list, up-el
up-el  -->  field-name = up-expr
up-expr  -->  val | (up-expr arith-op up-expr)
arith-op  -->  + | - | * | /
delete  -->  DELETE [del-qual]
del-qual  -->  PERMANENT | SELECTIVE | ALL
add  -->  ADD IN set-name own-qual
remove  -->  REMOVE FROM set-name

/*   R D L   S Y N T A X   */

/* schema */

schema  -->  name-part rel-part constr-part
name-part  -->  RDL SCHEMA name :
rel-part  -->  RELATION SECTION rel-def-list
rel-def-list  -->  rel-def | rel-def-list, rel-def
rel-def  -->  RELATION name (dom-def-list)
dom-def-list  -->  dom-def | dom-def-list, dom-def
dom-def  -->  name : type
```

```
type  -->  INTEGER null-opt | STRING(NUMBER) null-opt
null-opt -->  * | empty
constr-part  -->  CONSTRAINT SECTION constr-def-list
constr-def-list  -->  constr-def | constr-def-list, constr-def
constr-def  -->  key-def | subs-def | fd-def
key-def  -->  KEY rel-name (dom-list)
subs-def  -->  SUBSET rel-name (dom-list) rel-name (dom-list)
fd-def  -->  FD rel-name dom-def-list

/* dml */

stmt  -->  retrieve | insert | delete | update
retrieve  -->  RETRIEVE expr ;
insert  -->  INSERT rel-name expr ;
delete  -->  DELETE rel-name expr ;
update  -->  REPLACE rel-name expr (update-list) ;
expr  -->  rel-name |  expr {rs-list} | expr (dom-list) |
           expr bin-op expr | agg-join | expr[dom-list = dom-list]expr
agg-join  -->  expr [dom-list; fn, dom-list; dom-list] expr
rs-list  -->  rs-item | rs-list, rs-item
rs-item  -->  dom relop val
dom-list  -->  dom | dom-list , dom
dom  -->  ID | NUMBER
name  -->  ID
val-list  -->  val | val-list, val
val  -->  sim-agg | dom | value
value  -->
sim-agg  -->  fn ( dom-list , expr )
fn  -->  AVE | SUM | COUNT
bin-op  -->  UNION | INTERSECT | DIFF
update-list  -->  dom = up-expr | update-list , dom = up-expr
up-expr  -->  arith-expr | value
arith-expr  -->  expr | arith-expr arith-op arith-expr |
                 - arith-expr
arith-op  -->  + | - | * | /
re-lop  --> > | >= | < | <= | == | !=
```