On the implementation of a conceptual schema model within a three-level DBMS architecture*

by SHAMKANT B. NAVATHE
New York University
New York, New York

and

JOHANN LEMKE
Siemens AG
Munich, West Germany

INTRODUCTION

The ANSI/X3/SPARC study group on database management systems2 as well as some independent researchers17,10 have proposed a three-level "coexistence" architecture to database management systems. Under this approach (see Figure 1) a number of different users can be supported by means of different External Schemas, possibly with different data models and languages. It involves construction of a conceptual schema which completely represents the structure and semantics of a particular database. The underlying internal schema must provide a storage representation for the conceptual schema. Although the composition and scope of the conceptual schema is still a matter of controversy,1 several data models, semantic models (e.g., References 7, 8, 9, 13, 16, 20, 21) etc. could be considered as candidates for defining conceptual schema. Given a particular model for the conceptual schema, a number of problems arise in its implementation into a three-level DBMS, particularly regarding the design of an internal schema specification language to specify the mapping of the model into storage.

This paper discusses the general issues involved in implementing one specific model due to Falkenberg9 termed "the object-role model," as a conceptual schema model. Considerations in the design of a Conceptual Schema Language will be indicated. However, the emphasis will be on the issues of Internal Schema Language design. These languages are currently being developed and implemented at the Research Laboratories, Siemens AG, Munich, West Germany. Detailed analyses of these languages will be presented in forthcoming papers.4,15 Rather than present detailed language syntax, mathematical definitions of terms, operations, etc., this paper will highlight the issues and design decisions deemed essential for implementing a conceptual model.

A database administrator (DBA) in an organization is trusted with the task of defining a conceptual schema and an internal schema. In this paper wherever applicable, we point out that the DBA has to choose among alternatives or has to be aware of the implications of his decision.

The object-role (O-R) model

For the sake of completeness we will summarize the concepts from the object-role (O-R) model. This model is an ideal candidate for conceptual schema modeling since it has only a few basic concepts, which make it simple to use. It also has been shown to be evolvable and transformable.9,10 The model is used for modeling facts from a particular universe of discourse by means of objects, roles and associations. Objects are atomic, discrete elements in nature; the only information represented by them inherently is their existence. Facts concerning an object correspond to its association with one or more objects. An object performs a role in every association of which it is a part. Thus associations, which are n-ary in general, are composed of object-role pairs. Figure 2 gives an example of a model of a database in which the pairs (Doctor D, performs), (Surgical-procedure S, is-performed), (Patient P, is-operated) define the association "Surgical operation." An association may be "objectified" and may perform a role in another association. The latter is termed a nested association. In Figure 3 Miller's-salary is a binary association and (Miller's-salary, has-as-starting-date) is an object-role pair which is a component of the nested association "Miller's-salary-history."

The modeling concepts introduced thus far deal with modeling instances of objects, roles and associations. The type concept is introduced by which an association type refers to all associations with identical object-role pairs. Objects are pooled into object types such that objects under one type have at least one role in common. Figure 4 represents a database schema of which Figure 3 is an instance.

Objects and roles may be provided with significations.11

* This work was done while the first author was at Siemens AG, Munich, West Germany.
Structure of User Views

Structure of the Enterprise View

Structure of the Internal View

An object-type named Person may be signified by the person's First-name. A number of significations may be provided, e.g. First-name, Last-name and Date-of-birth for the object-type Person to make the signification unique. Frequency of occurrence of roles may be supplied as an additional information. In Figure 4 they are shown in parentheses. E.g., (0,1000) indicates that a Date may be the starting date in 0 to 1000 Salary-history associations whereas (1,1) indicates that a particular salary association starts on one and only one Date.

A conceptual schema language (CSL)

A CSL is being designed to define the conceptual schema of a database using the concepts from the object-role model. One of the early design decisions involved the definition of object-types.

As originally defined in the model, an object is atomic and has no information of its own. On principle, identification (unique significance) of an object x must be possible by associating x with objects of a different type(s)—usually
Figure 2—Objects and associations in a conceptual model (instance diagram).

Figure 3—A nested association (instance diagram).

Figure 4—A database schema involving object-types and association-types (schema diagram).
strings over some alphabet. Associations serving this purpose are called name associations (also identifying associations) and an object (≠ x) participating in a name association for x is called a name object for x. E.g., in Figure 6 object type Dept is identified by association type Dnum, and Person by Pnum. In general, a hierarchy of associations may identify an object. However, if for each object x of a type X we know a string $s_x$ that can serve as a never-changing identifier for x, CSL allows for defining X as a self-identified object type. This simplifies the conceptual schema. CSL deals with object types as follows:

1. An object type has a name (e.g. Person).
2. An object type is devoid of any inner structure. An association participating in a nested association is treated as an association rather than as an object.
3. A non-self-identified object type is distinguished from an object type which stands for the names of the former, (e.g. Person is different from Alpha). It is identified by a number of name associations. (E.g. each Person object is identified by an object of type Empl-number.) This concept is parallel with Navathe’s identifying relations. 14
4. For a self-identified object type the distinction mentioned above is not made. The name of an object of such a type is treated as if it were the object it stands for.

Roles, object types and even (object type, role) pairs may occur in more than one association type and also several times within one association type, if this is necessary to describe the semantics of a conceptual model. The latter is a means for expressing symmetric relationships between objects (see Figure 5). However, the DBA must realize that the choice of several identical (object type, role) pairs affects possible manipulations. E.g., a query like “List all persons associated with person x by role is-friend-of” implies investigation of all (not only one) roles is-friend-of. This is a typical example of implementing a conceptual model by selecting a proper implementation strategy rather than by changing the model itself.

Semantic rules were postulated 9 as a means of providing additional constraints for consistency and integrity among data instances. The syntax of CSL can be designed to incorporate such rules to any degree of complexity. E.g., a very high-level semantic rule: no two names in the database can be the same; a very low-level semantic rule:

10,000≤Salary≤50,000: a complex semantic rule: salary raises do not apply to persons who earn more than their second-level managers. Another design consideration involves the setting up of triggering mechanisms to invoke procedures representing semantic rules. Currently CSL allows semantic rules like the following to be automatically invoked during the execution of corresponding update procedures:

1. Characteristics of association types, e.g. “An employee cannot work on more than two projects.”
2. Characteristics of object types, e.g. “10,000≤salary-amount≤50,000.”
3. Dependencies between associations, e.g. “Total salaries of employees on a project cannot exceed its budget.”
4. Characteristics of events, e.g., “A person’s marital status cannot change from ‘divorced’ to ‘single’.”

Most databases have transactions or updates which cause changes in the data instances over a period of time. Time is an important attribute of data and provides valuable information in any dynamic database. Incorporation of time in semantic models has not been addressed sufficiently, barring a few exceptions (References 3, 5 etc.). Objects with a time point as an attribute have been called events, while those with time interval as attributes have been called processes. 24 Incorporation of time into the above CSL is being investigated without classifying the object-type into subtypes. 4

AN APPROACH TO THE DESIGN OF AN INTERNAL SCHEMA LANGUAGE

In the discussion above we highlighted the features of the O-R model and the requirements of a corresponding CSL. After a particular database conceptual schema is expressed using CSL, the database must be populated with instances which are mapped into storage according to an internal schema specification. Any retrieval or updating transactions operating on an external view are mapped into transactions on the conceptual view and further into transactions on the internal view. The internal schema language (ISL) must be capable of expressing all manipulation.

Our ISL design is based on the premise that databases will be stored in electronic cyclic memories using quasi-associative addressing techniques. Existing implementations 18,19 have shown that these storages and accompanying processors can store and manipulate relations very efficiently. The relational data model 6 has also been shown to fit the bubble hardware well because of the intrinsic similarities between the two. A relational model-based ISL must allow for specifying the storage of relations corresponding to the conceptual schema in the CSL.

ISL structure

The language is divided into four levels along the logical to physical spectrum.
Level 1—At this level the logical structure of the stored data is specified. With the assumption that stored data is in the form of relations, one must specify what relations need to be stored and how they are populated. Two main operations termed aggregation and substitution are defined at this level.

Level 2—Provides for the specification of constraints on relations whereby they can be organized to support efficient logical access structures. E.g., tuples may be ordered, a relation may be partitioned horizontally by clustering tuples or vertically by reordering and grouping of domains.

Level 3—At this level a lot of information is supplied to define the mapping of relations into storage. It consists
of: the division of the target address space into areas (extents) and the allocation of relations to them; definition of the type of encoding, formatting information at the domain level such as length, type (picture), padding, justification; definition of storage policies dealing with tuple insertion, overflow handling, free space management, space reclamtion, relative placement of relations, etc. 

Level 4—This level is used to characterize the structure of the storage itself. It defines what a unit of storage is, how these units are grouped into higher-level storage structures, what types of access is supported, whether data compression algorithms are used and so on.

Levels 2 through 4 include the detailed specifications that any Internal Schema Language should be able to express (see Reference 2). If a model other than the relational were used for stored data, relations and tuples would be replaced by corresponding constructs from that model.

In the rest of this paper we propose to focus on the ISL at Level 1 because it is at this level that the conceptual model of a database must be mapped into and specified in terms of a corresponding storage counterpart. The choice of the O-R model for conceptual modeling and the relational model for internal schema modeling presents some interesting problems.

An internal schema "view"

An Internal Schema View defines one particular way that a DBA might choose to map a given conceptual model into storage. Some of the previous work on internal schema has dealt with the problem of coming up with optimal solutions to mapping, i.e. defining optimal views when the target storage model is known. We assume that by doing an analysis of the processing requirements and given the knowledge of occurrence frequencies in associations, the DBA would decide to aggregate associations in a particular way; the opposite possibility where the DBA would like to split associations does not exist since in the O-R conceptual model associations represent semantically-irreducible facts.

We first describe the IS structure as it is implied by the CS definition. This is the structure we store in case no structuring specification is given.

Representation of references to objects—Whenever an object of a self-identified object type is referred to, we store its name. For each instance \( y \) of a non-self identified object type or of an objectified association type we create an internal identifier \( i_y \) and store \( i_y \), wherever \( y \) is referred to.

\[ R_A \times O_1 \times O_2 \times \ldots \times O_n \]

with domain names \( O_1, O_2, \ldots, O_n \). The term "elementary" was chosen because these relations may be used to build larger relations, but cannot be split. Given an instance of type \( A \) that associates object instances \( x_1, x_2, \ldots, x_n \) such that for \( 1 \leq i \leq n \), \( x_i \) is of type \( O_i \) and plays role \( r_i \), we store the tuple \( (y_1, y_2, \ldots, y_n) \) in relation \( R_A \), where, according to (i), for \( 1 \leq i \leq n \),

\[ y_i = \begin{cases} \text{name of } x_i, & \text{if } O_i \text{ is a self-identified object type} \\ i_y, & \text{the internal identifier of instance } x_i, \text{which we create, if } O_i \text{ is a non-self-identified object type or an objectified association type.} \end{cases} \]

A key of relation \( R_A \) comprises a sublist of the domain list of \( R_A \).

E.g.: the Birth association is represented by elementary relation \( R_{Birth} \subset \text{Date} \times \text{Person} \) with domain names \( \text{Date-Birthdate-of} \), \( \text{Person-Born-on} \).

Let us consider Figure 4 as an example of a nested association:

We create elementary relations \( R_{Salary-History} \) and \( R_{Salary} \) with domain names \( \text{Salary-Starts}, \text{Date-Is-the-starting-date-of}, \text{Person-Earns} \) and \( \text{Salary-Amount-Is-earned-by} \). The instance of Figure 3 will be treated as follows:

1. Assuming Person, Salary-Amount are self-identified object types, the tuple (Miller, 20000) is stored in \( R_{Salary} \).
2. Since Salary is an objectified association type, we create an internal identifier, e.g. S1, for the tuple (Miller, 20000).
3. Assuming Date is a self-identified object type, the tuple (S1, 1-1-78) is stored in \( R_{Salary-History} \).

Aggregation

An IS View is defined by specifying an aggregation. E.g., to aggregate all associations in Figure 6, the following statements are used (An assumption could be made that each role name occurs in one and only one association type. However, for the sake of generality, we use role names qualified by association type names.):

<table>
<thead>
<tr>
<th>( V_1 )</th>
<th>Name</th>
<th>\begin{cases} \text{AGGR (Person \cdot P-has-name)} \ \text{AGGR (Person \cdot P-born-on)} \end{cases}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Birth</td>
<td>\begin{cases} \text{AGGR (Person \cdot P-born-on)} \ \text{AGGR (Person \cdot P-works-in)} \end{cases}</td>
<td></td>
</tr>
<tr>
<td>Work</td>
<td>\begin{cases} \text{AGGR (Person \cdot P-works-in)} \ \text{AGGR (Person \cdot P-earns)} \end{cases}</td>
<td></td>
</tr>
<tr>
<td>Salary</td>
<td>\begin{cases} \text{AGGR (Person \cdot P-earns)} \ \text{AGGR (Person \cdot P-is-member-of)} \end{cases}</td>
<td></td>
</tr>
<tr>
<td>Co-worker</td>
<td>\begin{cases} \text{AGGR (Project \cdot P-has-workers)} \ \text{AGGR (Project \cdot P-has-as-budget)} \end{cases}</td>
<td></td>
</tr>
</tbody>
</table>
The philosophy of the AGGR statement is to specify a relational join by selecting a joinable domain from a number of different associations. Two relations are joinable if they are coherent, i.e. if they share at least one object type. In V1, the domains on which join is performed are either Person-ids or Project-ids. Figure 7(a) shows the elementary relations and Figure 7(b) the result of aggregation which is a relation with domains containing identifiers, either internal identifiers or name values.

In aggregating associations we had to use a "modified join" as follows: "A modified join takes the union (as opposed to intersection) of values in the join-domain from the relations being joined. Null values are created under appropriate domains corresponding to relations where a domain value may be absent." This is necessary because no elementary facts may be lost in mapping from the conceptual to internal schema. Figure 8 shows a conceptual schema and corresponding elementary relations. Relation BIBLIO is the result of aggregation of the two associations by joining on the Book-ids. The tuples (B3,A2,-) and (B4,--,P1) would be absent under conventional join. However, in relation BIBLIO we must preserve the fact that book B3 has author A2 and book B4 has publisher P1.

Substitution

As shown in Figure 7(b) aggregation results in the definition of an aggregated relation where certain domain values are internal identifiers. A DBA is allowed to define relations by substituting some or all of the internal identifiers by name values.

---

**Elementary relations for identifying associations**

<table>
<thead>
<tr>
<th>R-Name</th>
<th>Person*</th>
<th>Enpl-number*</th>
<th>E-number-belongs-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Smith</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>Jones</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>Smith</td>
<td>200</td>
<td></td>
</tr>
</tbody>
</table>

**Elementary relations for associations**

<table>
<thead>
<tr>
<th>R-Name</th>
<th>Person*</th>
<th>Alpha-belongs-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Smith</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>Jones</td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>Smith</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R-Name</th>
<th>Person*</th>
<th>Money-belongs-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Smith</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>Jones</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R-Name</th>
<th>Person*</th>
<th>Money</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Smith</td>
<td>100K</td>
</tr>
<tr>
<td>P2</td>
<td>Jones</td>
<td>60K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R-Name</th>
<th>Person*</th>
<th>Money-belongs-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR1</td>
<td>Smith</td>
<td>100K</td>
</tr>
<tr>
<td>PR2</td>
<td>Jones</td>
<td>200K</td>
</tr>
<tr>
<td>PR3</td>
<td>Smith</td>
<td>1M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R-Name</th>
<th>Person*</th>
<th>Time-belongs-to</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Smith</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>Jones</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R-Name</th>
<th>Person*</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Smith</td>
<td>100K</td>
</tr>
<tr>
<td>P2</td>
<td>Jones</td>
<td>100K</td>
</tr>
<tr>
<td>P3</td>
<td>Smith</td>
<td>100K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R-Name</th>
<th>Person*</th>
<th>Dept*</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Smith</td>
<td>DP1</td>
</tr>
<tr>
<td>P2</td>
<td>Jones</td>
<td>DP2</td>
</tr>
<tr>
<td>P3</td>
<td>Smith</td>
<td>DP1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R-Name</th>
<th>Person*</th>
<th>Co-worker</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Smith</td>
<td>PR1</td>
</tr>
<tr>
<td>P2</td>
<td>Jones</td>
<td>PR2</td>
</tr>
<tr>
<td>P3</td>
<td>Smith</td>
<td>PR3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R-Name</th>
<th>Person*</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Smith</td>
<td>PR1</td>
</tr>
<tr>
<td>P2</td>
<td>Jones</td>
<td>PR2</td>
</tr>
<tr>
<td>P3</td>
<td>Smith</td>
<td>PR3</td>
</tr>
</tbody>
</table>

---

Figure 7a—Elementary relations for the conceptual model in Figure 6.
by using a SUBST specification. E.g., continuing with Figure 7(b).

\[
\text{PERSONNEL} = \text{VI} \quad \text{where} \quad \begin{aligned}
\text{Person.P-has-name} & \quad \text{SUBST} & \quad \text{Pnum} \\
\text{Dept.Employies} & \quad \text{SUBST} & \quad \text{Dnum}
\end{aligned}
\]

will produce a PERSONNEL relation in which Persons and Departments are represented with name values instead of by internal identifiers. See Figure 7(c).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Smith</td>
<td>1947 07 01</td>
<td>DP1</td>
<td>40K</td>
<td>PR1</td>
<td>50%</td>
<td>100K</td>
</tr>
<tr>
<td>P2</td>
<td>Jones</td>
<td>1930 01 15</td>
<td>DP2</td>
<td>60K</td>
<td>PR3</td>
<td>20%</td>
<td>1M</td>
</tr>
<tr>
<td>P3</td>
<td>Smith</td>
<td>-- -- --</td>
<td>DP1</td>
<td>--</td>
<td>PR2</td>
<td>100%</td>
<td>200K</td>
</tr>
<tr>
<td>P1</td>
<td>Smith</td>
<td>1947 07 01</td>
<td>DP1</td>
<td>40K</td>
<td>PR2</td>
<td>50%</td>
<td>200K</td>
</tr>
<tr>
<td>P2</td>
<td>Jones</td>
<td>1930 01 15</td>
<td>DP2</td>
<td>60K</td>
<td>PR1</td>
<td>80%</td>
<td>100K</td>
</tr>
</tbody>
</table>

Figure 7b—Aggregate relation VI for the conceptual model in Figure 6.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Smith</td>
<td>1947 07 01</td>
<td>801</td>
<td>40K</td>
<td>PR1</td>
<td>50%</td>
<td>100K</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Jones</td>
<td>1930 01 15</td>
<td>802</td>
<td>60K</td>
<td>PR3</td>
<td>20%</td>
<td>1M</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>Smith</td>
<td>-- -- --</td>
<td>801</td>
<td>--</td>
<td>PR2</td>
<td>100%</td>
<td>200K</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>Smith</td>
<td>1947 07 01</td>
<td>801</td>
<td>40K</td>
<td>PR2</td>
<td>50%</td>
<td>200K</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Jones</td>
<td>1930 01 15</td>
<td>802</td>
<td>60K</td>
<td>PR1</td>
<td>80%</td>
<td>100K</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7c—Aggregate relation PERSONNEL, derived from VI by substitution.
Manipulation of IS

Additional ISL statements have been defined for a further manipulation of the stored relations by the DBA. A DROP statement has the following syntax:

\[
\text{DROP (typename)}[, (typename)]
\]

\[
\text{(typename)} :: = \text{(objecttypename)} | \text{(associationtypename)}
\]

If the DBA visualizes that all the processing against a conceptual schema can be supported by the defined aggregations, he may proceed to drop certain object types and/or association types. The corresponding elementary relations would be
Given relation R with one:many relationships;

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
R & A & B & C1 & C2 & C3 \\
\hline
a1 & b1 & c1 & & & \\
a1 & b1 & c2 & & & \\
a1 & b1 & c3 & & & \\
a1 & b2 & c1 & & & \\
a1 & b2 & c3 & & & \\
a1 & b3 & c6 & & & \\
a2 & b1 & c2 & & & \\
a2 & b1 & c4 & & & \\
a2 & b1 & c6 & & & \\
a1 & b2 & c1 & & & \\
a1 & b2 & c2 & & & \\
a1 & b2 & c3 & & & \\
\hline
\end{array}
\]

R takes up 36 elements of storage

S = R REPEAT C 3 TIMES; gives

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
S & A & B & C(1) & C(2) & C(3) \\
\hline
a1 & b1 & c1 & c2 & c3 & \\
a1 & b2 & c1 & c3 & - & \\
a1 & b3 & c6 & - & - & \\
a2 & b1 & c2 & c4 & c6 & \\
a1 & b2 & c1 & c2 & c3 & \\
\hline
\end{array}
\]

S takes up 25 elements of storage

T = S REPEAT B 2 TIMES; gives

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
T & A & B(1) & B(2) & C1 & C2 & C3 \\
\hline
a1 & b1 & b2 & c1 & c2 & c3 & \\
a1 & b2 & - & c1 & c3 & - & \\
a1 & b3 & - & c6 & - & - & \\
a2 & b1 & - & c2 & c4 & c6 & \\
\hline
\end{array}
\]

T takes up 24 elements of storage

Figure 9—Use of REPEAT by DBA to adjust the breadth of relations.
automatically scratched. To scratch a specific relation from internal-schema storage, the DBA uses a SCRATCH statement.

(SCRATCHstatement) ::= SCRATCH(relationname) {, (relationname) }

When there is an association type which associates an object type with two or more object types in a one: many fashion, multiple values occur in certain domains for a given value in one domain. Figure 9 shows an example relation which results from one: many associations between A: B and A: C. To fit the storage-characteristics a DBA may want to adjust the ‘‘breadth’’ of a relation by allowing domains to be subscripted as shown in Figure 9. A REPEAT statement is used toward that purpose (see Figure 9). This idea is related to multi-valued dependencies.

(REPEATstatement) ::= (relationname) REPEAT (domainlist) n TIMES

To revert back from the use of SUBSTITUTE, DROP, SCRATCH and REPEAT commands, a DBA can use RESUBST, KEEP, CREATE and COLLAPSE commands.

One of the advantages of choosing a relational structure for the internal schema is that the manipulation operations can be derived from the relational algebra. Queries in external schemas will be mapped into a set of relational operations like Projection, Restriction, Natural join on the elementary and aggregate relations. The implementation will account for repeating domains with subscripts and null values.

Reorganization of IS

According to the procedure outlined previously, a given conceptual schema produces a set of relations in the internal schema. The composition of this set can be controlled to some extent by the DBA by using the manipulation operations. If relations need to be normalized, rearranged, decomposed or synthesized, the relational operators will be available to the DBA. It is conceivable in the future that certain sequences of operations will be made available in the form of IS reorganization verbs.

SUMMARY

This paper has highlighted some of the typical issues that arise during the implementation of a particular semantic model in a three-level DBMS architecture. The model chosen is the object-role model due to Falkenberg. With a given design decision to organize the internal schema using relations, considerations for the design of an internal schema language were discussed. A systematic approach to defining relations and populating them so that they are equivalent to a conceptual model was outlined. Of particular importance is the concept of aggregation using a modified join operation. IS manipulation verbs for the DBA were also indicated. The Conceptual and Internal Schema Languages are currently being implemented at the Research Laboratories, Siemens AG, Munich, West Germany.

ACKNOWLEDGMENTS

The authors would like to acknowledge the cooperation and valuable discussions with Mr. Breutmann, Dr. Falkenberg, Mr. Hahne and Mrs. Mauer in the development of the paper. The research contribution of Mr. Breutmann and Mrs. Mauer in the development of the Conceptual Schema Language is particularly recognized.

REFERENCES