TR-161

An Interactive Data Dictionary System to Support Logical Database Design

by

Piyush Gupta

Umeshwar Dayal

A. G. Dale

Department of Computer Sciences

The University of Texas at Austin

Austin, Texas 78712

October 1980

An Interactive Data Dictionary System to Support Logical Database Design

Piyush Gupta, Umeshwar Dayal and A. G. Dale
The University of Texas at Austin

1. Overview

This paper describes the structure of a language interface to an interactive data dictionary system to support a methodology for logical database design. The system is being implemented in the Department of Computer Sciences at The University of Texas at Austin.

The system fits the proposed ANSI/SPARC architecture [ANSI 77] by providing a capability for supporting conceptual schema design, storage of a global conceptual schema, and description of external views of the global database. Many of the ideas incorporated in the system implementation, and assumptions regarding the design methodology it will support, are based on recent work of A. P. Buchmann [BUCH 80] who has proposed a methodology for database design to support engineering design applications, and who used a binary associative system, CS-4 [BN 77], as a prototype environment to support data dictionary applications necessary to create a conceptual schema. The system described in this paper, however, uses a different data model (third normal form relations [CODD 76]) for the conceptual schema than the binary association model proposed by Buchmann, and incorporates a different algorithm for generating conceptual schema structures.

Design Methodology

The data dictionary system provides a software environment to support the following steps in the logical design of a database:

1. Generation of Local Views

Users (here, users refer to the end users of the database) express their 'views' of the applications of interest in terms of data objects and associations between data objects. If there are numerous data objects and associations in a user view, it is advisable to provide the database designer with system aids for generating consistent descriptions of data objects and associations. The techniques for requirements analysis are not of concern here. We assume the existence of a front-end analysis technique that permits the identification of data objects and relationships, as a preliminary to initial input to the data dictionary.

2. Generation of the Global View

The database designer uses the system interactively to integrate the various local views into a global, logical view of the database. This involves the resolution of homonyms and synonyms as the data objects are defined, resolution of homonyms and synonyms as the associations (dependencies) are being defined and, finally, the use of Bernstein's algorithm [Bern 76] to eliminate redundant and transitive associations and produce a set of Third Normal Form (3NF) relations. The conceptual schema data model in our system is thus a 3NF relational model. This normalization technique is described in Section 3.

3. Accommodation

This step deals with the transformation of the 3NF relations to a set of binary logical associations, as a preliminary to mapping the conceptual schema description into the data definition language, and data model, of a specific database management system.

Accommodation packages are specific to the target system, and so are not described in this paper. A subsequent paper will describe one package, now under development, to map from the data dictionary to System 2000 DDL statements to produce S2K database schemas consistent with a particular global schema in the data dictionary model.

Figure 1 summarizes the methodology. Sections 2 and 3 describe the Definition Phase and Normalization Phase in detail. It is important to note that the design proceeds interactively and iteratively: modelling errors discovered during the Normalization phase may entail a return to the Definition phase to change some of the data object or association definitions stored in the data dictionary. The language interface to the system is described in Appendix I. Appendix II illustrates the methodology with an example of the interaction between the designer and the system.

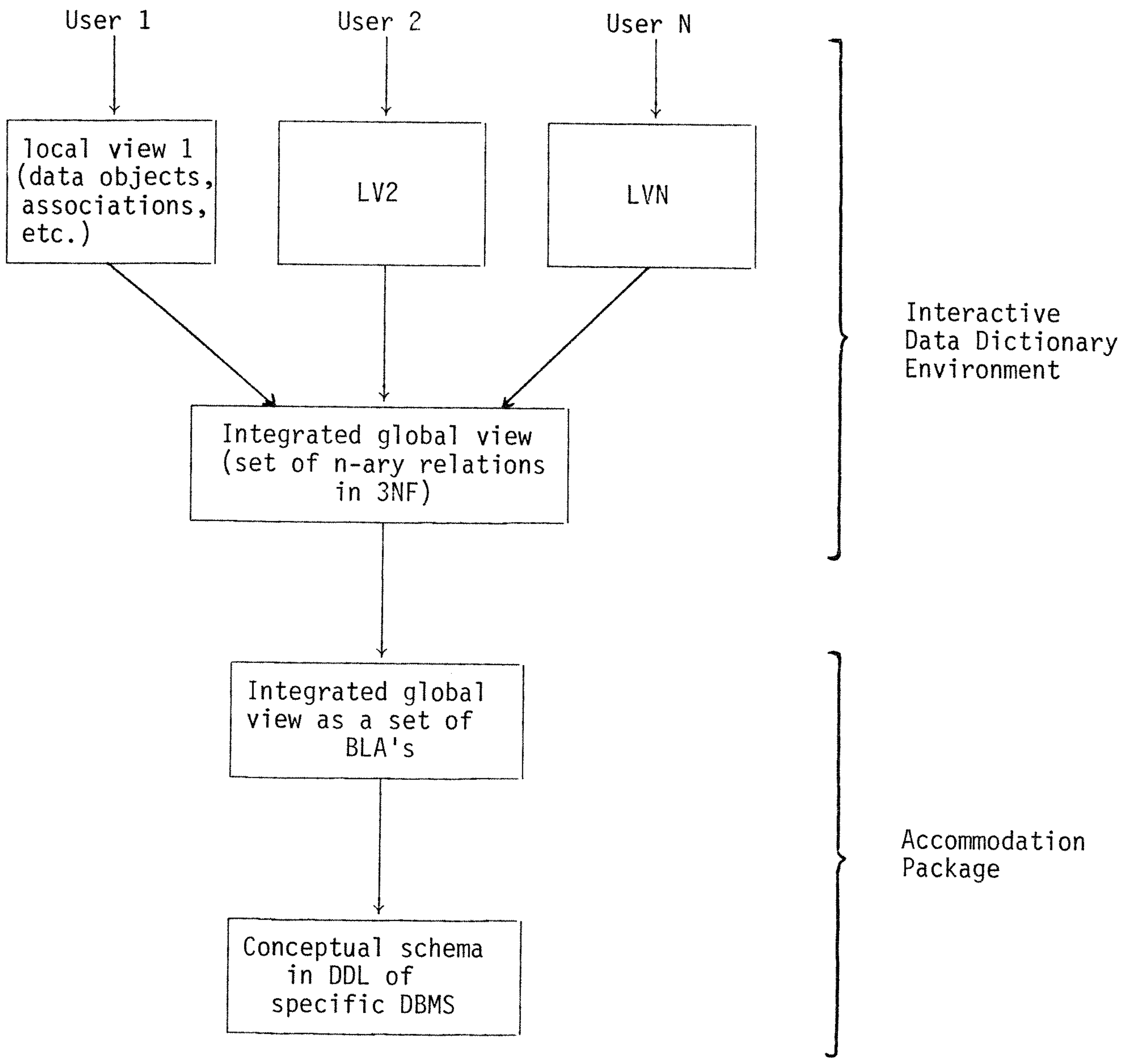
User 1
User 2
User N
local view 1 (data objects, associations, etc.)
LV2
LVN
Integrated global view (set of n-ary relations in 3NF)
Integrated global view as a set of BLA's
Conceptual schema in DDL of specific DBMS
Interactive Data Dictionary Environment
Accommodation Package

Fig. 1

2. The Definition Phase

2.1 Information Specification

As stated in Section 1, the user views are modelled in terms of data objects (entities) and their associations (relationships or dependencies). We now describe exactly how the data objects and associations are specified:

Object Definition

A data object is the smallest unit of interest in the data base. It could represent a physical object that we're interested in modelling or properties thereof, e.g. EMPLOYEE SS#, INCOME etc.

A data object is specified in terms of its name (a string of characters, e.g. employee), the application or view in which the data object is being defined (e.g. payroll), the source (which user is primarily responsible for input of instances of the data object), an abstract object type or generalization of the data object, and a brief description of the data object. Thus, each data object specification consists of a 5-tuple.

An essential element during data object and association definition is the identification of homonyms and synonyms and the resolution of such naming conflicts. The specification of data objects in the dictionary system lends itself to the identification of such conflicts.

In some applications, information on the 'source' of a data object helps reduce the search space during synonym detection (if

every data object has a unique source). We provide the designers with the option of specifying the 'source' in data object definitions to assist the system in synonym detection.

The 'abstract object type' is a generalization in the sense introduced by Smith & Smith [SS77]. It is also, like the source data, useful during synonym detection. The 'description' is a brief elaboration of the meaning of the data object and can assist the designer in confirming and resolving naming conflicts.

2.2 Association Definition

We are considering associations that are functional in nature [BBG 78]. An association consists of a name and two sets of functionally related data objects and may be represented as follows.

Association Name: Set of data objects (Subject) $\longrightarrow$
Set of data objects (Predicate)

The predicate is functionally dependent on the subject i.e. each subject value in the association has associated with it precisely one value of the predicate.

e.g. EMPLOYEE RECORD: EMP# $\longrightarrow$ EMPNAME, SALARY, AGE. Data objects EMPNAME, SALARY & AGE are each functionally dependent on the data object EMP# i.e. given a particular value of EMP#, there exists exactly one corresponding value of EMPNAME, SALARY & AGE.

e.g. ENROLLMENT: STUDENT#, COURSE# $\longrightarrow$ GRADE. Here, there exists a unique grade corresponding to each unique pair of values of STUDENT# & COURSE# and GRADE is functionally dependent on the STUDENT#, COURSE# pair.

In other words, the subject forms a key [CODD 76] of an association.

Each association is, therefore, specified in the form of a 3-tuple viz, <association name, set of data objects in the subject (Left Hand Side), set of data objects in the predicate (Right Hand Side)>. The data objects in the LHS & RHS are those that have been specified earlier during the data object specification phase. Also specified is the view in which the association is being defined.

2.2 System support during the definition phase

Data Object Specification

Data object 5-tuples from user views are entered into the system sequentially. It is possible that different end users could be referring to different data objects by the same name (homonyms) or the same data object by different names (synonyms). It is necessary, therefore, that as a data object is entered into the system, it be checked against the existing data objects for naming conflicts.

(i) Homonym Resolution

If the system finds an existing data object with the same name as that of the object being entered it informs the user (user, in this report, refers to the Data Base Administrator and not to an end user of the data base unless stated otherwise. The end users only provide input at the requirements gathering stage) and displays a description of the existing data object. The user can, then, change the name of the data object being entered if the two are supposed to be different. These changes are recorded in the data dictionary.

(ii) Synonym Resolution

This is the more difficult task. In cases where every data object has a unique 'source', the system compares the 'source' and 'generalization' fields of the data object being defined with those of the existing data objects. If a match is found, the data objects for which the match occurred are potential synonyms and the user is informed. If the user responds (perhaps after reading their descriptions and conferring with the application administrator) by confirming that they are synonyms, this fact is recorded in the data dictionary.

In the more general case where a data object does not have a unique source, synonym detection is based on inspection of the 'generalization' field alone, and the display of objects with identical generalizations.

Association Specification

Next, the database designer enters the various associations (dependencies) into the system. Again, it is possible that different end users employ the same name for different associations or different names for the same association. Hence, the system helps resolve these naming conflicts in the following manner:

(i) Same Association Name

If another association with the same name is found to exist, the two are potential homonyms or synonyms. The system examines the subjects and predicates of these associations. If the data objects constituting the subjects and predicates are found to be identical (or synonymous) the two associations are identical (or synonymous) and the user is informed. If he agrees, the 'new' association he defined is just stored in his view. The other association (which was defined earlier) is retained in the central global schema and will be used for normalization purposes. If the user wants the two associations to be different and distinct, there has been an error in the understanding and definition of one of them. Accordingly, the user must change the name of one of the associations and update its subject and/or its predicate as well. (Could add more data objects and/or delete data objects and/or change the existing data objects).

If the subjects and predicates of the two associations (which have the same name) are neither identical nor synonymous, the two associations are homonymous and the user is asked to rename the association he is currently defining.

(ii) Different Association Name

Different association names do not guarantee that the associations are different. A search is conducted on the subject and predicate fields. If an association with subject and predicate identical or synonymous to those of the association being defined is found, the new association could be synonymous to it. Accordingly, the user is notified. He will then inform the system whether they are synonymous or different. If different, again, there has been an error in the understanding and definition of one of them. Accordingly, the user is asked to update the subject and/or the predicate of one of them.

If the user agrees that they are synonymous, this fact is recorded in the dictionary and the association being defined is stored only in his view.

3. The Normalization Phase

At the end of the definition phase, the collection of data objects and dependencies of interest to the enterprise have been identified. As an initial approximation, then, we may consider the global conceptual schema to be a single ("universal") relation (in First Normal Form) with the data objects as its attributes and the dependencies as constraints. First Normal Form relations, however, exhibit certain undesirable properties ("anomalies"). A series of "higher" Normal Forms that eliminate these anomalies have been proposed. Of these, the most popular is Codd's Third Normal Form [CODD76], which attempts to embody each independent relationship in a separate relation. Normalization, then, is a technique for improving an initial schema design by transforming the given First Normal Form relation into a set of relations in Third Normal Form that represent the same data objects and relationships. (The reader is referred to [CODD76, DATE77, BERN76, BBG78] for detailed discussions of normalization.) Our database design system uses Bernstein's synthesis algorithm for this purpose [BERN76]. In the sequel, we outline the various steps involved in this algorithm.

Let F be the given set of functional dependencies. Let F+ denote the closure of F i.e. the set of all functional dependencies that are implied by F. Let LHS(f) denote the set of data objects on the left hand side of a particular association $f \varepsilon F$. Let A represent a particular data object or attribute.

A functional dependency (FD), therefore, is of the form f:LHS(f)--->RHS(f).

Several steps of the synthesis algorithm require checking whether an FD f is in the closure of F, i.e., whether f is implied by F. A fast algorithm for this membership test is described in [BB79] and is repeatedly called as a subroutine by the synthesis algorithm.

Synthesis algorithm:

(i) Elimination of redundant attributes:

An attribute A is redundant in LHS(f) if LHS(f)-A--->RHS(f) is in F+ i.e. all the dependencies that can be inferred from LHS(f) can also be inferred from LHS(f)-A.

The following procedure implements this step:

```
do for each f$\varepsilon$F
     do for each A$\varepsilon$LHS(f)
          if (LHS(f)-A)--->RHS(f)$\varepsilon$F+
             then LHS(f)=LHS(f)-A
     enddo
enddo
```

This procedure takes time $O(|F|^2)$ where $|F|$ is the total number of attributes on the left hand sides of all the associations in F.

(ii) Elimination of Transitive dependencies (redundant associations):

Here we are trying to construct a non-redundant covering G of F. The procedure is:

```
G:=F
do for each f∈F
     if f∈(G-f)+ then G:=G-f
enddo
```

This procedure takes time $O(n|F|)$ where n is the number of associations in F.

(iii) <u>Partitioning of G</u>:

Partition the nonredundant covering of F constructed in step (ii) into groups such that each group has associations with identical left hand sides. This step takes $O(|F|^2)$ time.

(iv) <u>Merging equivalent keys</u>:

Suppose G includes FDs of the form X-->Y and U-->W; at the end of step (iii) these will be contained in different groups. However, if U-->X and X-->U are in F+, U and X are functionally equivalent and these two groups should be merged. The following procedure implements this step:

```
J=0
do for each pair of groups Hi, Hj with LHS's Xi and Xj
     if Xi-->Xj and Xj-->Xi∈G+
          then
               begin
                    merge Hi and Hj
                    J:=J+[Xi-->Xj, Xj-->Xi]
                    G:=G-[Xi-->Xj, Xj-->Xi]
               end
enddo
```

This procedure runs in $O(n|F|)$ time.

(v) <u>Elimination of transitive dependencies generated during (iv)</u>:

The merging of groups might generate new transitive dependencies. Hence, another step is required to eliminate these transitive dependencies. Find a $G' \leqslant G$ such that $(G'+J)+=(G+J)+$ and no proper subset of G' has this property.

```
do for each Xi-->Xj∈J
     add it to the corresponding group in G'
enddo
```

This runs in $O(n|F|)$ time.

(vi) <u>Construction of 3NF relations</u>:

For each group construct a relation consisting of all the data objects appearing in that group. The LHS of a group is the key of the corresponding relation. If a group was formed by merging equivalent keys in step (iv), the corresponding relation will have several keys. The user is asked to designate one of these as the primary key for the relation. This set of 3NF relations represents the global view of the database.

(vii) To guarantee that each view is constructable as a set of relations from the conceptual schema generated, we add the following step to the synthesis algorithm:

(see [BDB79, OSBO78, LOZI80] for details).

For each association f(x) in each view check whether there is some relation R(y) in the conceptual schema generated with the property that $y \dashrightarrow x \in F^{+}$ (here, x and y are the set of attributes in f and R respectively); if there is no such relation then add a relation R(k) (to the conceptual schema) where k is a minimal subset of x with the property that k-->x.

This step takes $O(n|F|^2)$ time, where n is the number of relations synthesized.

Normalization is a purely syntactic procedure. It works under the assumption that the initial definition of the data objects (attributes) and dependencies is correct, and guarantees that all the associations specified by the users are represented in the synthesized conceptual schema, and also that each user view is constructable from the conceptual schema relations. This implies that testing or authentication is needed only at the requirements gathering and definition phases to ensure that the correct data objects and associations were specified. This can be done outside the system.

In our system, the normalization phase is not purely mechanistic. Rather, it proceeds interactively, and, hence, assists in catching some semantic errors, such as those pointed out in [BERN76]. In steps (i), (ii), and (iv), whenever a potentially extraneous attribute or redundant dependency is discovered, the user is asked to confirm that this is indeed the case; otherwise, a semantic error has occurred. For example, consider the FD's f_1: EMP#-->COMPANY, f_2: COMPANY--> ADDRESS, and f_3: EMP#-->ADDRESS. Syntactically, it appears as if f_3 is redundant, since it is implied by the composition of f_1 and f_2. However, the composition of f_1 and f_2 might actually give the corporate address of the company that an employee works in, whereas f_3 gives the employee's address. If these are different, then the FD inferred from f_1 and f_2 is different from f_3. This semantic distinction must be reflected as a syntactic distinction in the definitions of f_2 and f_3. The system then reverts to the definition phase, instructing the user to redefine the RHS's of f_2 and f_3; for example, thus:

f_2: COMPANY-->CORPORATE-ADDRESS,

f_3: EMP#-->ADDRESS-OF-EMPLOYEE.

An analogous situation might arise during the detection of extraneous attributes in step (i) of the synthesis algorithm (see [BERN76] for examples).

In conclusion, we make a passing reference to the ubiquitous "universal relation assumption". Normalization and data dependency theory is built on the assumption that at all points in time, all the relations in the database are projections of a single (universal) relation. The validity of this assumption in practice has been challenged [BBG78, BG80]. Yet, empirical evidence suggests that, even if the assumption is violated in practice during operational use of the database, schemas designed using the normalization technique are "good" (i.e., free of the update anomalies described in [CODD 76]). We reconcile ourselves to this seemingly paradoxical situation by suggesting that the theory be modified to drop the universal relation assumption. Since the normalization technique produces good schema designs, it must be more "robust" than the present theory suggests. In a forthcoming paper we describe how to modify the theory to accurately reflect the practice of normalization.

Conclusions

The commercial use of data bases has increased tremendously in the last few years with the result that databases have grown in size and complexity. Logical database design, therefore, is no longer a trivial problem that can be easily done by a designer using a paper and a pencil! In this paper we have described an interactive data dictionary system to support incremental logical database design. This system is designed to satisfactorily handle applications where the associations between the various data objects of interest are functional in nature.

However, not all associations in the world are functional!

Consider the association between

SUPPLIERS<--->PARTS.

A supplier could supply many different parts and a particular part could be stored by more than one supplier. Clearly, this is not a functional association. As a matter of fact, it is an m:n association.

Our system will handle such non-functional associations satisfactorily as long as the data objects involved in these associations are also involved in some functional associations. e.g. consider an application where a construction project is being modelled. The user could specify the following associations:

SUPPLIES: SUPPLIER, PART --->QUANTITY

with the obvious meaning that QUANTITY stands for the quantity of a particular part supplied by a particular supplier. Thus, although there is an m:n relationship between SUPPLIER and PART, it will be correctly modelled because SUPPLIER and PART are also involved in a functional association, viz. SUPPLIES.

However, the system is not designed to handle "pure" non-functional associations where the data objects are not involved in other functional associations. e.g. let the m:n association

STOCK: SUPPLIER<--->PART

be the only association between the data objects SUPPLIER and PART. The system is incapable, at present, to handle such applications.

There exist techniques by which it can be extended to handle such applications. In particular, there is a technique involving the introduction of auxiliary variables [BERN 76] which can be used. We will consider this extension as a goal in the future.

We would like to add at this point that we have not considered multivalued dependencies [BBG 78] at all. This is primarily because experience has shown us that even highly technical users find it very difficult to identify and express their requirements in terms of MVD's.

Appendix A

Note:

In the following, system prompts and responses are enclosed within single quotes while user responses are enclosed within double quotes.

1.0 <u>View Definition</u>

The user initiates view definition by typing:

"DEFINE VIEW <viewname> *"

The system will respond with:

'READY'

All object and association definitions hereafter will be considered to belong to <viewname> until another view definition command is encountered.

2.0 <u>Object Definition</u>

To initiate object definition the user types:

"DEFINE OBJECT*"

The system responds with:

'NAME?'

in response to which the user should type in the name of the data object he is defining.

1. If the system detects a homonym, it responds with:

   'DATA OBJECT ALREADY EXISTS WITH FOLLOWING PARAMETERS'

   'SOURCE:----------'

   'GENERALIZATION:----------'

   'VIEW:----------'

   'DESCRIPTION:----------'

   'PLEASE CHANGE DATA OBJECT NAME IF DIFFERENT'

The user may respond with "SAME*" or "NEWNAME=-----------*".

If the system does not spot a homonym it asks for the rest of the information on the data object:

'SOURCE?' "<user responds>*"

'GENERALIZATION?' "<user responds>*"

'DESCRIPTION?' "<user responds>*"

2. If the system spots synonyms it responds with:

'POTENTIAL SYNONYMS FOUND'

and lists the synonyms (one representative element from each synonym class) in the following format:

```
'NAME'     'VIEW'     'DESCRIPTION'

 ____       ____       __________

 ____       ____       __________
```

'PLEASE CONFIRM'

The user can then respond with "NO*" in which case the object is stored uniquely or "YES <name>*" in which case the object is stored as a synonym of <name> in the alias table.

Object definition is terminated by typing "END*" in response to 'NAME?'.

3.0 Object Retrieval

The user may ask for the display of information on a particular data object, all objects in a particular view or all the objects in the entire system.

3.1 Particular Object

The user types

"SHOW OBJECT NAMED <name>*"

The system will output the corresponding tuple in the following form:

```
'SOURCE:----------'
'GENERALIZATION:----------'
'VIEW:----------'
'DESCRIPTION:----------'
```

3.2 Objects in a view

The user types "SHOW OBJECT IN VIEW <view>*". The system responds asking the user about the output format desired 'OUTPUT FORMAT?'. If the user types "BRIEF*" only the object names are listed. If the user types "DETAIL*" information is listed in the following format

```
'NAME:----------'
'GENERALIZATION:----------'
'DESCRIPTION:----------'
```

3.3 Objects in the system

The user types "SHOW OBJECT IN WHOLE DATABASE*" in response to which the system asks for the output format as before 'OUTPUT FORMAT?'.

If the user responds with "BRIEF*" the system lists the names of data objects in each view:

```
'VIEW <view1>'
'<data object1>, <data object2>----------'
```

'VIEW <view2>'

'<data object1>,----------'

If the user responds with "DETAIL*" the system asks for reconfirmation 'MASSIVE OUTPUT OK?'.

The user can now type "YES*" or "NO*" in response to which the system will output detailed information on all data objects in every view or abort the command.

4.0 Object update

The user types "UPDATE OBJECT<name>*". Thereafter, he could modify any part of the information on the specified object by typing one or more of the following:

"NEWNAME=----------*"

"NEWSOURCE=----------*"

"NEWGENERALIZATION=----------*"

"NEWVIEW=----------*"

"NEWDESCRIPTION=----------*"

Note:

Object updating may result in homonyms or synonyms again and the system could generate messages similar to those described in section 2.

5.0 Association Definition

The user types "DEFINE ASSOCIATION*".

The system responds with:

'NAME?' "<user response>*"

'LHS?' "<user response>*"

'RHS?' "<user response>*"

The system will then respond in one of the following ways:

1. If the association name already exists with different LHS and RHS the system will respond with:

   'NAME ALREADY USED PLEASE CHANGE'

   'NEWNAME?' "<user response>*"

2. If the association appears to be synonymous to some other association defined earlier i.e. they have the same or synonymous LHS and RHS, the system responds with:

   'POTENTIALLY SYNONYMOUS ASSOCIATION FOUND'

   It displays the synonymous association (the representative element of the synonym class detected) and seeks user confirmation.

   'NAME <name> VIEW <view> LHS=--------- RHS=---------'

   'PLEASE CONFIRM'

If the user responds with "YES*", the association is appropriately stored in the association alias table, whereas if he responds with a "NO*" the system informs him that there is an inconsistency in one of the two definitions and requests him to modify one of them:

'SEMANTIC INCONSISTENCY. UPDATE ONE OF THE ASSOCIATIONS'

The user should then update the subject and/or the predicate of one of the associations. If the associations have the same name, one of the association names must be changed too.

Association definition is terminated by typing "END*" in response to 'NAME?'.

6.0 Association Retrieval

6.1 Named Association

The user types "SHOW ASSOCIATION NAMED <name>*" in response to which the user lists out all the information it has on that particular association in the following format:

'LHS=----------'

'RHS=----------'

'VIEW=----------'

6.2 Association in a view

The user types "SHOW ASSOCIATION IN VIEW <view>*". The system asks for the output format desired:

'OUTPUT FORMAT?'

If the user responds with "BRIEF*" the system just lists the names of all the associations in the specified view. If the user responds with "DETAIL*" the system lists each association in detail:

'NAME=----------'

'LHS=----------'

'RHS=----------'

6.3 Associations in the system

The user types "SHOW ASSOCIATION IN WHOLE DATABASE*" in response to which the system asks for the output format desired:

'OUTPUT FORMAT?'

If the user types "BRIEF*"

the system lists the associations in each view by name:

'VIEW <view1>'

'<association1 name><association2 name>----------'

'VIEW <view2>'

If the user types "DETAIL*" the system asks for reconfirmation:

'MASSIVE OUTPUT OK?'

The user can now type "YES*" in which case detailed information on all the associations in every view will be output or "NO*" in which case the command will be aborted.

7.0 Association Update

The user types "UPDATE ASSOCIATION*" and the system asks for its name and the view in which it has been defined:

'NAME?' "<user responds>*"

'VIEW?' "<user responds>*"

Thereafter, the user can make the desired modifications by typing one or more of the following:

"NEWNAME=----------*"

"NEWLHS=----------*"

"NEWRHS=----------*"

Note:

Association updating may result in new homonyms and synonyms due to which the system could generate messages similar to those generated during association definition.

8.0 Object Deletion

The user can delete a data object from the system by typing: "DELETE OBJECT <name>*" in response to which the system will delete the named data object from the data dictionary and issue the following message: '<name> DELETED'

Note:

It is the user's responsibility to ensure that associations using the deleted data object are modified or deleted.

9.0 Association Deletion

The user may delete an association from the system by specifying its name and the view in which it is defined in the following format:

"DELETE ASSOCIATION <name> FROM VIEW <view>*"

10.0 Miscellaneous

If the user types "NORMALIZE*" it will result in the execution of Bernstein's 3NF synthesis algorithm. During this execution, user intervention could be sought for primary key resolution:

```
'SPECIFY KEY IN ASSOCIATION'
'NAME:----------'
'LHS:----------'
'RHS:----------'
'POTENTIAL KEYS ARE'
     '1.----------'
     '2.----------'
'KEY?' "<user responds>*"
```

Also, whenever a redundant object or association is eliminated, the user will be informed and his confirmation sought:

```
'REDUNDANT OBJECT <name>'
'IN ASSOCIATION <association name>'
'IN VIEW <view>'
'PLEASE CONFIRM'
```

If the user types "REMOVE*" the object is eliminated, whereas if he types "RETAIN*" the system recognizes that there has been a mistake in defining the semantics of some associations and informs the user with an error message:

'SEMANTIC ERROR'

'PLEASE CHECK ASSOCIATIONS AND UPDATE APPROPRIATELY'

The user must then examine his associations, trace the error, make the necessary updates and restart the normalization process.

Redundant associations are treated in a similar manner. The system outputs the following messages:

'REDUNDANT ASSOCIATION <name>'

'IN VIEW <view>'

'PLEASE CONFIRM'

If the user responds with "REMOVE*" the association is deleted from the data dictionary. If he responds with "RETAIN*" the system detects that there are semantic errors in association definition. It informs the user and also suggests recovery techniques, e.g. suppose there are three associations:

1. DEPT OF EMPLOYEE: EMP--->DEPT

2. MANAGER OF EMPLOYEE: EMP--->MGR

3. DEPT OF MANAGER: MGR--->DEPT

modelling an enterprise where an employee may be managed by a manager from a different department. From associations 2 and 3 the system infers association 1 (transitive dependency) and requests permission to delete it:

'REDUNDANT ASSOCIATION NAMED DEPT OF EMPLOYEE'

'IN VIEW <view>'

The user, naturally, will type "RETAIN*". Now, the system realizes that the data objects DEPT in associations (1) and (3) should be distinct and issues appropriate error messages suggesting recovery techniques:

'SEMANTIC ERROR IN ASSOCIATION DEPT OF EMPLOYEE'

'IN VIEW <view>'

'DISTINCT DATA OBJECT DEPT REQUIRED'

The user can now define a new data object EMPDEPT and modify the association DEPT OF EMPLOYEE so that it looks like:

DEPT OF EMPLOYEE: EMP--->EMPDEPT

and issue the normalization command again.

During the last stages of normalization, all the associations with the same LHS are merged to form a single association. The system will inform the user of this merger and ask him to name this association:

'ASSOCIATIONS BEING MERGED'

'A1 IN VIEW V1'

'A2 IN VIEW V2'

'NEWNAME?' "A3*"

The user can ask for a display of the final conceptual schema by typing "SHOW DESIGN*". The system will ask for 'OUTPUT FORMAT?'. If the user responds with "BRIEF*", only the names of the associations in the conceptual schema designed are listed, whereas if he responds with detail, the data objects forming the left and right hand sides of the associations are also listed.

An interactive design session is ended by typing in a period.

APPENDIX B

Here, we give an example of a logical database design, for a commercial organization, using our system. To keep the design simple and illustrative, we model only the Personnel, Accounting and Administration divisions of the organization.

The personnel division stores the names and addresses of all the employees in the organization. It also stores the name of an employee's manager and information on his insurance policy.

To focus attention on the system features and not burden the reader with unnecessary details of the model, we make some simplifications in it viz. every employee has exactly one insurance policy and can work in exactly one department. A department can have only one manager.

After the requirements analysis phase, the DBA has identified the following views of interest:

(i) Personnel

Data objects: Emp#, Empname, Address, Mgr#, Policy#, coverage

Associations:

Empname&address: Emp#--->name, address

Empinsurance: Emp#--->policy#, coverage

Mgr of emp: Emp#--->mgr#

(ii) Accounting

Dataobjects: employee#, salary, policy#, coverage premium, department, budget

Associations:

Payroll: Emp#--->salary

Empinsurance: policy#--->coverage, premium

Deptbudget: department--->budget

(iii) Administration

Dataobjects: department, manager#, emp#

Associations:

mgr of dept: department--->mgr#

Dept of emp: emp#--->department

The interactive design now proceeds as follows:

Note:

System prompts and responses are in upper case letters while user responses are in lower case.

```
define view personnel*

READY

define object*

NAME? emp#*

SOURCE? personnel*

GENERALIZATION? id*

DESCRIPTION? social security number of the employee*


NAME? empname*

SOURCE? personnel*

GENERALIZATION? name*

DESCRIPTION? name of the employee*


NAME? address*

SOURCE? personnel*

GENERALIZATION? address*

DESCRIPTION? permanent address of the employee*


NAME? mgr#*

SOURCE? administration*

GENERALIZATION? id*

DESCRIPTION? ss# of the manager of an employee*
```

```
NAME? policy#*

SOURCE? accounting*

GENERALIZATION? id*

DESCRIPTION? insurance policy number*


NAME? coverage*

SOURCE? accounting*

GENERALIZATION? cash amount*

DESCRIPTION? amount of policy coverage in dollars*


NAME? end*


define association*

NAME? empname & addr*

LHS? emp#*

RHS? name, address*


NAME? empinsurance*

LHS? emp#*

RHS? policy#, coverage*


NAME? mgr of emp*

LHS? emp#*

RHS? mgr#*


NAME? end*
```

```
define view accounting*

READY

define object*


NAME? employee#*

SOURCE? personnel*

GENERALIZATION? id*

DESCRIPTION? social security# of employee*

POTENTIAL SYNONYMS FOUND


NAME: EMP#

VIEW: PERSONNEL

DESCRIPTION: SOCIAL SECURITY NUMBER OF THE EMPLOYEE

PLEASE CONFIRM

yes*


NAME? salary*

SOURCE? accounting*

GENERALIZATION? income*

DESCRIPTION? monthly income of the employee*


NAME? p#*

SOURCE? accounting*

GENERALIZATION? id#*

DESCRIPTION? insurance policy#*

POTENTIAL SYNONYMS FOUND
```

```
NAME: POLICY#

VIEW: PERSONNEL

DESCRIPTION: INSURANCE POLICY NUMBER

PLEASE CONFIRM

yes*


NAME? coverage*

DATAOBJECT ALREADY EXISTS WITH FOLLOWING PARAMETERS

VIEW: PERSONNEL

SOURCE: ACCOUNTING

GENERALIZATION: CASH AMOUNT

DESCRIPTION: AMOUNT OF POLICY COVERAGE IN DOLLARS

PLEASE CHANGE DATAOBJECT NAME IF DIFFERENT

same*


NAME? premium*

SOURCE? accounting*

GENERALIZATION? cash amount*

DESCRIPTION? total premium on insurance policy*

POTENTIAL SYNONYMS FOUND

NAME: COVERAGE

VIEW: PERSONNEL

DESCRIPTION: AMOUNT OF POLICY COVERAGE IN DOLLARS

PLEASE CONFIRM

no*
```

```
NAME? department*

SOURCE? administration*

GENERALIZATION? department*

DESCRIPTION? department in which the employee works*


NAME? budget*

SOURCE? planning*

GENERALIZATION? cash amount*

DESCRIPTION? annual budget of the department*


NAME? end*


define association*

NAME? payroll*

LHS? emp#*

RHS? salary*


NAME? emp insurance*

LHS? policy#*

RHS? coverage, premium*

NAME ALREADY USED PLEASE CHANGE


NEWNAME? insurance*


NAME? dept. budget*

LHS? department*

RHS? budget*
```

```
NAME? end*


define view administration*

READY

define object*

NAME? department*

DATAOBJECT ALREADY EXISTS WITH FOLLOWING PARAMETERS

VIEW: ACCOUNTING

SOURCE: ADMINISTRATION

GENERALIZATION: DEPARTMENT

DESCRIPTION: DEPARTMENT IN WHICH THE EMPLOYEE WORKS

PLEASE CHANGE DATA OBJECT NAME IF DIFFERENT

same*


NAME? manager#*

SOURCE? administration*

GENERALIZATION? id*

DESCRIPTION? ss# of the manager*

POTENTIAL SYNONYMS FOUND

NAME: MGR#

VIEW: PERSONNEL

DESCRIPTION: SS# OF THE MANAGER OF AN EMPLOYEE

PLEASE CONFIRM

yes*
```

```
NAME? emp#*

DATAOBJECT ALREADY EXISTS WITH FOLLOWING PARAMETERS

VIEW: PERSONNEL

SOURCE: PERSONNEL

GENERALIZATION: ID

DESCRIPTION: SOCIAL SECURITY NUMBER OF THE EMPLOYEE

PLEASE CHANGE DATA OBJECT NAME IF DIFFERENT

same*


NAME? end*


define association*

NAME? mgr of dept*

LHS? department*

RHS? mgr#*


NAME? dept of emp*

LHS? emp#*

RHS? department*


NAME? end*


normalize*

REDUNDANT ASSOCIATION MGR OF EMP

IN VIEW PERSONNEL

PLEASE CONFIRM

remove*
```

```
REDUNDANT OBJECT COVERAGE
IN ASSOCIATION EMPINSURANCE
IN VIEW PERSONNEL
PLEASE CONFIRM
remove*

ASSOCIATIONS BEING MERGED
EMPINSURANCE IN VIEW PERSONNEL
EMPNAME&ADDR IN VIEW PERSONNEL
PAYROLL IN VIEW ACCOUNTING
DEPT OF EMP IN VIEW ADMINISTRATION
NEWNAME? employee info*

show design*
OUTPUT FORMAT? detail*
EMPLOYEEINFO
LHS: EMP#
RHS: NAME, ADDRESS, POLICY#, SALARY, DEPARTMENT
INSURANCE
LHS: POLICY#
RHS: COVERAGE, PREMIUM
DEPTBUDGET
LHS: DEPARTMENT
RHS: BUDGET
MGROFDEPT
LHS: DEPT
RHS: MGR#
```

REFERENCES

[ANSI 77] Tsichritzis, D. and A. C. Klug, "ANSI/X3/SPARC DBMS Framework. Report of the Study Group on Data Base Management Systems." AFIPS Press, Montvale, NJ, 1977.

[BB 79] Beeri, C. and P. A. Bernstein, "Computational Problems Related to the Design of Third Normal Form Relational Schemas," ACM trans. on Database Systems 4:1, March 1979, pp. 30-59.

[BBG 78] Beeri, C., P. A. Bernstein and N. Goodman, "A Sophisticate's Introduction to Database Normalization Theory," Proc. Fourth Intl. Conf. on Very Large Data Bases, Berlin, Sept. 1978, pp. 113-124.

[BDB 79] Biskup, J., U. Dayal and P. A. Bernstein, "Synthesizing Independent Database Schemas," Proc. ACM-SIGMOD Intl. Conf. on Management of Data, Boston, May-June 1979, pp. 143-151.

[BERN 76] Bernstein, P. A., "Synthesizing Third Normal Form Relations From Functional Dependencies," ACM Trans. on Database Systems 1:4, Dec. 1976, pp. 277-298.

[BG 80] Bernstein, P. A. and N. Goodman, "What Does Boyce-Codd Normal Form Do?", Proc. VLDB Conf., Oct. 1980, pp. 245-259.

[BN 77] Berild, S. and S. Wachmans, "Some Practical Applications of CSA - A DBMS for Associative Databases," in Architecture and Models in Data Base Management Systems, G. M. Nijssen, (ed.) North Holland Publishing Co., 1977.

[BUCH 80] Buchmann, A. P., "A Methodology for Logical Design of Databases for Project Engineering," Ph.D. Dissertation, Dept. of Chem. Engr., UT Austin, May 1980.

[CODD 72] Codd, E. F., "Further Normalization of the Database Relational Model," in Database Systems, Courant Comp. Sci. Symp. 6, (R. Rushi, ed.), Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 33-64.

[DATE 77] Date, C. J., "An Introduction to Database Systems, Addison-Wesley Pub. Co., Reading, MA, 1977.

[LOZI 80] Lozinskii, E. L., "Construction of Relations in Relational Databases," ACM TODS, Vol. 5, No. 2, June 1980, pp. 208-224.

[OSBO 78] Osborn, S. L., "Normal Forms for Relational Databases," Ph.D. Diss., Research Rep. CS-78-06, Dept. of CS, Univ. of Waterloo, Jan. 1978.