AN EXPERT VIEW CREATION SYSTEM FOR DATABASE DESIGN

By

VEDA CATHERINE STOREY

A.R.C.T. University of Toronto, 1978
B.Sc., Mount Allison University, 1978
M.B.A., Queen's University, 1980

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

in
THE FACULTY OF GRADUATE STUDIES
(Faculty of Commerce and Business Administration)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
October, 1986

© Veda Catherine Storey, 1986
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Commerce and Business Administration

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date October 14, 1986
ACKNOWLEDGEMENTS

I wish to sincerely thank the members of my dissertation committee, Dr. Robert C. Goldstein and Dr. Yair Wand of the Faculty of Commerce and Business Administration and Dr. Alan K. Mackworth of the Computer Science Department, all of the University of British Columbia, for their assistance in preparing this dissertation.

I am especially indebted to my research advisor, Dr. Robert C. Goldstein, for his patience and support and without whom this work would simply not have been possible.

This research was supported by fellowships from The Imperial Order of the Daughters of the Empire, The Natural Sciences and Engineering Research Council of Canada, Suncor Inc., and The University of British Columbia. This support is gratefully acknowledged.
Abstract

The process of generating user views during logical database design is formalized and expressed as a set of rules which comprise the knowledge base of an expert system. This system, called the View Creation System, engages the user in a dialogue to determine information requirements. These requirements are then translated into a set of Fourth Normal Form relations representing a view. The data model on which the system is based is the Entity-Relationship Model. Using this model, the system elicits entities, attributes and relationships while trying to detect and rectify inconsistencies and ambiguities in the user's input. With the aid of the user, functional dependencies are isolated and resolved before the final set of relations is produced.
Table of Contents

AN EXPERT VIEW CREATION SYSTEM FOR DATABASE DESIGN ........................................... 1
I.  INTRODUCTION .................................................................................................................. 1
II. VIEW MODELLING ............................................................................................................. 3
    ENTITY-RELATIONSHIP MODEL ...................................................................................... 4
    DATA ABSTRACTION ......................................................................................................... 6
    N-S MODEL ....................................................................................................................... 7
        User Views in the N-S Model ....................................................................................... 7
        Types of Associations ................................................................................................. 9
    BUBBLE CHARTING .......................................................................................................... 11
    INTERACTIVE SPECIFICATION ....................................................................................... 12
    SUMMARY OF VIEW MODELLING METHODOLOGIES ..................................................... 16
    DISCUSSION ...................................................................................................................... 17
        Ease of Use .................................................................................................................... 17
        Semantics ...................................................................................................................... 20
        Automation .................................................................................................................. 21
III. EXPERT SYSTEMS ........................................................................................................... 23
    TYPES OF EXPERT SYSTEMS .......................................................................................... 23
<table>
<thead>
<tr>
<th>KNOWLEDGE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain Knowledge</td>
<td>25</td>
</tr>
<tr>
<td>Hard-and-fast Rules or Procedures</td>
<td>26</td>
</tr>
<tr>
<td>Heuristics</td>
<td>26</td>
</tr>
<tr>
<td>Global Strategies</td>
<td>26</td>
</tr>
<tr>
<td>Theory of the Domain</td>
<td>27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KNOWLEDGE REPRESENTATION</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Production Rules</td>
<td>27</td>
</tr>
<tr>
<td>Logic</td>
<td>34</td>
</tr>
<tr>
<td>Semantic Nets</td>
<td>35</td>
</tr>
<tr>
<td>Frames</td>
<td>37</td>
</tr>
<tr>
<td>Summary</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INFERENC EMACHINE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction of Inference</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EXPLANATION OF REASONING</th>
<th></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>USER INTERFACE</th>
<th></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>EXPERT SYSTEMS AND DATABASE DESIGN</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure</td>
<td>46</td>
</tr>
<tr>
<td>Interfaces</td>
<td>46</td>
</tr>
</tbody>
</table>
Knowledge Base ........................................................................................................... 47
Discussion .................................................................................................................. 47

IV. RESEARCH DESIGN ................................................................................................. 49

VIEW CREATION SYSTEM CHARACTERISTICS ......................................................... 49

Underlying Theory: The Entity-Relationship Model .................................................... 50
Knowledge Acquisition ............................................................................................... 50
Knowledge Representation .......................................................................................... 52
Inference Machine ....................................................................................................... 53
Explanation of Reasoning ............................................................................................. 56

RESEARCH PLAN ......................................................................................................... 57

1. Prototype Development ............................................................................................ 57
2. Expansion of Knowledge Base .................................................................................. 58
3. System Testing and Modification ............................................................................ 59

V. VIEW CREATION SYSTEM ....................................................................................... 60

PROCEDURE ................................................................................................................. 60

INITIALIZATION ............................................................................................................. 63

ENTITIES ...................................................................................................................... 63

Attributes ..................................................................................................................... 63
Candidate Keys ............................................................................................................. 68
MODIFICATIONS - FIRST ................................................................. 179
SYNONYMS .............................................................................. 179
   Synonyms — Entities ......................................................... 179
   Synonyms — Verb Phrases ............................................. 181
   Synonyms — Relationships ........................................... 182
INHERITED ATTRIBUTES ......................................................... 182
MODIFICATIONS - LAST ............................................................. 182
PRIMARY KEYS ......................................................................... 184
   Rules for Choosing Primary Keys .................................... 184
   General Rules ................................................................. 185
   Is-a Relationships .......................................................... 185
      Adjust Attributes for Is-a Relationships .................... 186
   Instance-of Relationships ............................................. 186
      Adjust Attributes for Instance-of Relationships ........... 187
   Entities Requiring Other Entities for Identification .......... 187
REPRESENT ENTITIES and RELATIONSHIPS .......................... 188
   Entities ............................................................................. 188
   Relationships ................................................................. 188
      Construct Relations .................................................... 189

xii
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation</td>
<td>251</td>
</tr>
<tr>
<td>System Modification</td>
<td>251</td>
</tr>
<tr>
<td>USER#3</td>
<td>251</td>
</tr>
<tr>
<td>Application</td>
<td>251</td>
</tr>
<tr>
<td>Design Session</td>
<td>251</td>
</tr>
<tr>
<td>User#3’s View</td>
<td>252</td>
</tr>
<tr>
<td>Evaluation</td>
<td>252</td>
</tr>
<tr>
<td>System Modification</td>
<td>252</td>
</tr>
<tr>
<td>USER#4</td>
<td>252</td>
</tr>
<tr>
<td>Application</td>
<td>252</td>
</tr>
<tr>
<td>Design Session</td>
<td>252</td>
</tr>
<tr>
<td>User#4’s View</td>
<td>253</td>
</tr>
<tr>
<td>Evaluation</td>
<td>253</td>
</tr>
<tr>
<td>USER#5</td>
<td>253</td>
</tr>
<tr>
<td>Application</td>
<td>253</td>
</tr>
<tr>
<td>Design Session</td>
<td>253</td>
</tr>
<tr>
<td>User#5’s View</td>
<td>254</td>
</tr>
<tr>
<td>Evaluation</td>
<td>254</td>
</tr>
<tr>
<td>System Modification</td>
<td>256</td>
</tr>
</tbody>
</table>
USER#6 and USER#7 .......................................................... 257
Application ................................................................. 257
Design Session .......................................................... 257
User#6 and User#7's View ............................................. 257
Evaluation ................................................................. 257
USER#8 ................................................................. 258
Application ................................................................. 258
Design Session .......................................................... 258
User#8's View ............................................................ 258
Evaluation ................................................................. 259
System Modification ................................................... 259
List of Tables

TABLE I: View Modelling Approaches ................................................................. 18
TABLE II: View Creation System's Procedure ..................................................... 61
TABLE III: Test Sessions ...................................................................................... 132
I. INTRODUCTION

Database design is divided into two main parts – logical and physical. Logical database design is concerned with determining the actual contents of a database independent of implementation considerations while the physical design process concentrates on mapping the logical design into a form which is suitable for the given hardware and database management system (DBMS).

A number of methodologies exist for logical database design. Most of them assume that a statement of database requirements is available and expressed as a set of user views. Each view describes the database content and structure that is appropriate for a particular function that a user (or a group of users) performs. The process of designing user views is a difficult task which relies heavily on experience and judgment rather than mechanistic algorithms. Traditionally, it has involved an experienced database designer collecting information from users (or their surrogates, systems analysts) and then producing a view specification which is refined through an iterative process. Good database designers are both scarce and costly. Consequently, the number of users consulted and the number of design iterations are both usually less than what ideally would be desired. Therefore, the research question of this study is:

Can the process of generating user views be formalized and subsequently automated in order to increase the quality and consistency of database design while relying less on scarce expert human skills?

This question is addressed by actually formalizing, in a set of rules, a methodology for generating user views. These rules are encoded as the knowledge base of an expert system, called the View Creation System. The purpose of the system implementation is twofold. First, the View Creation System is primarily a vehicle for expressing the formalization. It thus allows the rules and control structures that comprise the
formalization to be tested experimentally for adequateness and completeness. Secondly, the development of the system guides the formalization process. Each version of the knowledge base is implemented and the system tested. The testing procedure assists in highlighting areas where the formalization is inadequate. In this sense, the formalization process is an iterative one.

This paper is divided into eight parts. Background literature and theory on view modelling and expert systems are presented in Chapters II and III, respectively, before outlining the research design in Chapter IV. The View Creation System is discussed in detail in Chapter V and the testing of the system in Chapter VI. Finally, the contribution of the current research is contained in Chapter VII and possible extensions to the work are described in Chapter VIII. A set of appendices presents the knowledge base of the expert system, a transcript of a design session using the View Creation System, documentation of consultation sessions with expert database designers and the results obtained when formally testing the system.
II. VIEW MODELLING

A user view can be defined as [Navathe and Elmasri, 1986]:

"the perception of users about what a proposed database (or an ideal database) should contain."

In essence, a user view is a representation of reality relevant to a particular user or group of users for a specific purpose. The set of all views used in an organization can be taken as a specification of the required contents of that organization’s database. Currently, most methodologies for database design assume the existence of a set of view definitions and are concerned primarily with integrating these into a unified whole [e.g. Martin, 1981; Raver and Hubbard, 1977].

An example of a simple user view for a payroll system is:

Employee:
[EMPLOYEE-NUMBER, Employee-Name, Salary]
(where EMPLOYEE-NUMBER is the key or identifier of Employee.)

Note that the information about an employee included in this user view would not be the only information required about an employee in an organization’s database. Rather, it is simply the information required for one specific application.

The process of eliciting a user’s view of the database is called view modelling and is defined formally by Navathe and Schkolnick [1978] as:

"the modelling of the usage and information structure of the real world from the point of view of different users and/or applications".

Navathe and Schkolnick describe the two major components of view modelling as:
1. Extracting from the user or from a person in charge of application development the relevant parts of real-world information.

2. Abstracting this information into a form which completely represents the user view so that it can be subsequently used in the design.

View representation has been addressed mainly as a by-product of data model development [Bracchi, 1981; Navathe and Schkolnick, 1978] According to Navathe and Schkolnick [1978], the most pertinent work done in this area has been the Entity-Relationship (E-R) data model of Chen [1976] and the Data Abstraction methodology of Smith and Smith [1977]. Navathe and Schkolnick also propose their own data model, the Navathe and Schkolnick (N-S) model, as a vehicle for modelling user views.

There are also two methodologies which have been explicitly developed for constructing user views. These are Bubble Charting [Martin, 1981] and the Interactive Specification methodology of Baldissera et al. [1979].

This section first summarizes without comment and then discusses these five main view modelling approaches (i.e. E-R Model, Data Abstraction methodology, N-S Model, Bubble Charting and Interactive Specification).

ENTITY-RELATIONSHIP MODEL

The Entity-Relationship model [Chen, 1976] is perhaps the best known of the view modelling approaches. The model is simple and easy to conceptualize, employing only two basic constructs – entities and relationships. An entity is a "thing" of interest in a database and something which can be clearly defined, e.g. student. A relationship is an association among entities, e.g. student takes courses is an association between student and courses. Attributes, i.e. properties or characteristics, can be identified for both entities and relationships. For example, student-number could be an attribute of the entity, student, and grade an attribute
of the relationship student takes courses.

Semantics are captured by the model in several ways. First, each entity is required to play a particular role in a relationship: travellers make reservations; reservations do not make travellers. Another way that semantics are captured is through the concepts of entity sets and relationship sets. Being a member of one of these sets indicates that the entity or relationship satisfies certain conditions which are characteristic of that set.

There are, however, situations where the Entity-Relationship model fails to capture semantics. The model does not allow for the representation of relationships between two relationships or between an entity and a relationship nor does it allow one to distinguish among different types of relationships [Navathe & Schkolnick, 1978].

Using the Entity-Relationship model as a logical design tool involves constructing an E-R model for each user view of the database. These models of the user views are then combined in a rather mechanical way to produce a unified E-R model which supports all of the requirements [Goldstein, 1985].

With respect to the process of generating user views, if the Entity-Relationship model is used as the underlying theory, then one must clearly identify and distinguish the main constructs. However, it may be difficult to determine whether something should be represented as an entity or a relationship (or as an attribute of an entity or a relationship). For example, should marriage be represented as an entity or as a relationship between two person entities? Fortunately, each view-level E-R model is relatively small and focused on a single application area which aids in reducing the complexity of the construction of a view [Goldstein, 1985].
DATA ABSTRACTION

In Smith and Smith's [1977] Data Abstraction methodology, both entities and relationships are represented as objects. Two different types of relationships are represented: association of similar entities and association of related entities. These are known as generalization and aggregation data abstraction types respectively and are used to build a multi-dimensional hierarchy of objects which represents a user view [Yao et al., 1978].

In general, an abstraction is a model of a system in which some details are purposely omitted in order to focus attention on the most significant features. An aggregation abstraction is an abstraction where a relationship between objects is made into a higher level object. For example [Smith and Smith, 1977], reservation is an abstraction of a particular relationship between a person, a hotel, and a date. One can think in terms of a reservation without being concerned with all the details of the underlying relationship (e.g. length of stay).

Generalization takes a class of similar entities and considers it as a generic object. For example, a set of people who drive trucks can be thought of as the generic object trucker. This new object is then a primitive for defining models of the real world. For instance, suppose one wished to model the abstract object employee. First, this abstract object would be decomposed along the generalization plane to (for example) the generic objects: trucker, secretary, and engineer. The decomposition of employee along the aggregation plane would serve to include attributes of employee such as: employee ID#, name, age, etc. Hence, the abstract object, employee, is at the intersection of a generalization and an aggregation hierarchy. Additional relationships can be modelled by further aggregation or generalization. (See Smith and Smith [1977].)
In terms of logical database design, Smith and Smith's approach has been described as conceptually elegant. However, when the approach is applied to the community view of data (as opposed to individual user views), it has been criticized for demanding a great deal of skill on the part of the designer when large-scale problems are involved [Navathe and Schkolnick, 1978].

**N-S MODEL**

The Navathe and Schkolnick (N-S) model [Navathe & Schkolnick, 1978; Navathe & Gadqil, 1982] draws on concepts of Smith and Smith's Data Abstraction model with an objective of modelling user views as explicitly as possible.

**User Views in the N-S Model**

User views are represented in the N-S model in terms of entities, associations, attributes and connectors and are presented in a view diagram. Associations are n-ary relationships among entities, among entities and associations, or among associations. They represent facts, ideas or specific aspects of a relationship. For example [Navathe and Schkolnick, 1978], advisor-of is an association between the entities professor and student.

A view is then a graph which has entities and associations as its nodes and edges (either directed or nondirected) linking the nodes. The edges are called connectors and are represented as two-tuples with an arrow over one of the members of a tuple indicating the direction of the association. Thus, Employee works-on Assignment would be represented as three nodes, one each for employee, works, and assignment. The connector between employee and works would be (employee, works) where an arrow over the word employee would represent the direction of the connector. Similarly, a connector between works and assignment would be (works, assignment) with an arrow over the word works indicating that the direction of the association is from works to assignment. An association is described by
an n-tuple which contains the names of the participating entities and attributes; e.g. Works (employee, assignment).

Both entities and associations are called objects in the model and have corresponding descriptor sets. Descriptor sets are lists of attributes, or groups of attributes in the case of repeating groups, which can be associated with an instance of an object in the database. For example, for the (entity) object employee in the example above, the corresponding descriptor set could be:

Employee = {EMPLOYEE#, Dep#, {Year, Grade}, {School,{Degree}}}

In this example there are two repeating groups: {Year, Grade} and {School, {Degree}}. For the (association) object assignment, the corresponding descriptor set could be:

Assignment = {ASSIGNMENT#, Assignment-Name}

Identifiers

Every entity in a view diagram must have an identifier. In this model, the type of identifier is explicitly defined. It may be internal or external with an internal identifier being either total or partial. When an element or several elements from the descriptor set are used to identify an entity, an internal identifier exists. A total internal identifier provides a full identification of the entity while a partial internal identifier needs some external identification to uniquely identify an instance of the entity. For example, Employee# is a total internal identifier for employee because there is a 1-1 correspondence between Employee# and employee. An external identifier may be needed for instances when the same value for an internal identifier stands for different, real objects as in the case of two instances of city requiring a corresponding state identifier.
Each entity must have either a total internal identifier or an augmented total identifier. The identifier associations aid in establishing rules for inserting and deleting instances of entities and identifier associations. For example, an instance of an entity with a partial identifier can only be inserted if it is provided with the appropriate external identification. This increases the semantics of the model.

*Types of Associations*

Associations (n-ary relationships defined over entity and association types) are further divided into simple and identifier associations. Three types of simple associations (categorization, selection, and subsetting) provide additional semantic information.

1. **Categorization**

   Categorization is an association between owner and member entities. This is closely related to Smith and Smith's [1977] generalization concept except it eliminates the mutually-exclusive category constraint. For example [Navathe and Schkolnick, 1978], a student may be categorized into undergraduate, graduate, special, resident, aid-recipient, etc. The mappings defined by the N-S model give each student a proper subset of categories so that many different categories do not have to be defined to ensure mutual exclusiveness. Categorization is used in a semantic context in connection with insertion / deletion activities. For example, inserting a student with a joint-major in Law and Business implies inserting two new instances under both those categories; insertion in Arts implies an insertion in Student.

2. **Selection**

   Selection is a binary association between entities. A selection association $A(X,Y)$ provides an association where an instance of $X$ is associated with a subset of instances
of \( Y \). For example [Navathe and Schkolnick, 1978], let \( A(X,Y) \) be the selection association: \( A(\text{country},\text{student}) \). This is interpreted as the selection of a student by country where country is the owner of the selection association.

Selection differs from categorization at the instance level. If the student-country example was modelled using categorization, for each student, an additional instance, such as U.S.-student or Canadian-student, would be required for each category.

In terms of semantic content, inserting an owner instance \( X \) for a selection \( A(X,Y) \) implies that the corresponding selection function must be modified, instances of \( Y \) must be repartitioned, and new instances of \( A \) created if necessary.

3. Subsetting

Subsetting is a unary association which provides a means of associating the name \( A \) with a subset of instances of \( B \). For example, \( \text{advisor-of} \) is an association between professors and students whereas \( \text{phd-advisors} \) is a subsetting association of which \( \text{advisors-of} \) is a component. (Other components of advisor-of could be: project-advisors and curriculum-advisors.) Subsetting allows a subset of association instances to be aggregated and named.

Summary

In summary, the N-S model has two type constructs: objects and connectors with object types being divided into entity and association types. Associations are n-ary relationships defined over entity and association types and are further divided into simple and identifier associations. Three types of simple associations (categorization, selection, and subsetting) provide additional semantic information. Connector types connect an association type to an
object (entity or association type). They may be directed or undirected with directed connector types implying certain rules of insertion and deletion for associations. A view is then represented by a graph having entities and associations at its nodes and connectors between nodes.

Some of the claimed advantages of the N-S model are [Navathe and Gadgil, 1982]:

1. Association types can be defined over entity types, association types or a combination of both (in contrast to the E-R model).

2. Existence-dependencies among entities are modelled separately from relationships among entities (unlike the Data Abstraction method) so identification paths are clearly defined.

3. Data relationships are modelled at both the schema and the instance levels. Properties of views with respect to insertion/deletion rules etc. are clearly distinguished (unlike the E-R model and the Data Abstraction method) by using different association subtypes and by using directed connector types.

**BUBBLE CHARTING**

Bubble Charting [Martin, 1981] is a method of representing user views whereby the users are taught to draw representations of data structures. Bubble Charts are claimed to provide a way of representing and thinking about data and their associations so simply that users can be taught to use them, draw them, and think about their data with them [Martin, 1983].

The fundamental piece of data which is represented by this technique is a data item—the smallest piece of data that is meaningful to a user (e.g. student-name). This is in contrast to approaching the development of a view from higher-lever concepts such as entities (e.g. student) and relationships (e.g. student is-a person).
Bubble charts are drawn by first representing individual data items by ellipses (or bubbles). Associations between the data items are represented by drawing links in the form of single and double-headed arrows between data items. Single-headed arrows between two data items imply that each value of the first item has one and only one value of the second item associated with it. Double-headed arrows indicate that one value of the first item may have zero, one, or many values of the second item associated with it. For example, in modelling a user view about employees a single-headed arrow would exist from employee-name to salary while a double-headed arrow would represent the appropriate association between employee-name and projects-worked-on.

A primary key is defined as a bubble with one or more single arrows leaving it while an attribute is a bubble with no single arrows leaving it. Similar definitions exist for secondary and concatenated keys. With these definitions, the user proceeds to draw his desired view. The rules that the user must follow are constructed to ensure that the resulting representation will be in third normal form (3NF) — e.g. "whenever a concatenated key is introduced, the designer should make sure that the items it identifies are dependent on the whole key, not on a portion of it only" [Martin, 1981].

INTERACTIVE SPECIFICATION

The DATAID project [Batini et al., 1984] at Politecnico di Milano has as its goal the development of a set of integrated computer-assisted tools which will aid in covering all aspects of database design from analysis of the database application through to physical design. The part of the project which deals with view modelling is based on the assumption that it is possible to aid a user in performing the view modelling function through an interactive system [Bracchi, 1981].
The Interactive Specification system [Baldissera et al., 1979] accepts as input elementary sentences describing application requirements. The system takes the responsibility of:

1. "linking" together different requirements that the user has so that the user need not be concerned with how he organizes his input sentences
2. checking that the representation is formally correct and detecting inconsistencies and redundancies
3. presenting to the user different ways in which the collected requirements could be interpreted and asking the user to resolve any ambiguities
4. producing a schema for the design phases which follow.

The user provides the system with some elementary sentences which express his requirements. The system then reorganizes these sentences to produce a canonical formal representation of the user's view. The aim of this process is to avoid some of the arbitrariness in perception which different designers have when modelling the same view. However, this is done on the basis of syntax only. Any assertions which refer to semantics must be interactively verified by the user.

**Model**

The underlying model which represents the views is an extension of the simple binary model [Bracchi, 1981]. The extensions are intended to provide greater semantic modelling capabilities. However, the complexity of the model is limited due to the problems which arise with natural language interfaces [Ceri et al., 1981].

The user provides the system with elementary sentences which are in binary form – i.e. two concepts or sets of concepts are linked through a verb or verbal construct. Three main kinds of elementary statements are allowed:

1. A verbal construct b.
2. A is b.
3. A has B.
where lower case-letters stand for simple concepts and upper case letters are used for sets of concepts.

For example [Baldissera et al., 1979]:

1. A "verbal construct" b: manager, employee works in department.
2. a verbal construct b: manager manages department.
3. A is b: manager, employee is dependent.
4. A has B: manager, employee has name, address.

In general, elementary sentences of the second and third type are the most desirable. Sentences (or relations) with the verb is imply that the user is expressing a hierarchy of concepts which provides a basic unit of information in the model. For sentences of the form A has b, the verbal construct is changed to has b. This type of sentence is used to avoid multiple names for the same relation.

The elementary sentences are represented by a graph where nodes correspond to concepts and arcs to verbal constructs. Thus, the elementary sentence person has address is represented by two nodes called person and address connected by the arc has address. The sentence person has name is integrated with the first sentence by adding another node called name and then connecting the person and name nodes with the arc has name. Other sentences are represented in a similar manner.

**Homonyms**

The input sentences (or relations) are analyzed to eliminate homonyms. Relations with the same name are divided into groups of relations which have the same semantic meaning and each group is assigned a different name.
For example [Baldissera et al., 1979]:

Manager manages activity
Employee manages activity
Servant manages hotel-room

is transformed to:

Manager, Employee manages activity
Servant keeps in order hotel-room.

Here, manages has two different meanings so, for the second meaning, a new verbal construct is created. Since the meaning of manages is the same for manager and employee managing activities, the two respective sentences are amalgamated. Note that this exercise deals with semantics, therefore the new set of elementary sentences must be obtained in cooperation with the user.

Hierarchies of Concepts

Since hierarchies of concepts provide the basic unit of information in the model, the elementary sentences are analyzed to test that:

1. hierarchies of concepts are explicitly defined and any redundancies eliminated
2. hidden hierarchies are detected and corresponding redundancies eliminated.

If a relation A is b is semantically correct, then all the relations applied to the superconcept, b, must also be applied to the a's which are members of the set of concepts, A.

E.g. [Baldissera et al., 1979]:

employee is person
employee, person has name name.
is converted to:

```
employee is person
person has name name.
```

**Synonyms**

Synonyms are detected by determining semantic equivalence which is done only through interaction with the user. When synonyms are detected, a common name is assigned to the relations involved and redundancies are eliminated.

E.g. [Baldissera et al., 1979]:

```
Person had department department
employee is person
employee works in department.
```

is converted to:

```
person has department department
employee is person.
```

Further testing and manipulation is performed (e.g. testing for and eliminating redundant concepts) before the final canonical representation of a view is obtained.

**SUMMARY OF VIEW MODELLING METHODOLOGIES**

The five view modelling approaches are very diverse in their focus. The Entity-Relationship data model has only two constructs – entities and relationships. It was not developed as a tool for view modelling but is the most common approach used in database design. The Navathe-Schkolnick data model, however, was developed specifically for modelling user views. Its objectives include capturing a great deal of semantic content and defining a user
view as explicitly as possible. It does so through the use of many different constructs. The Data Abstraction methodology, upon which the N-S model is based, employs only a single concept which is used to build hierarchies of objects that represent user views.

The other two approaches (Bubble Charting and Interactive Specification) were developed solely as methodologies for generating user views. Bubble Charting has no strong theoretical basis nor does it attempt to provide any semantic information. It focuses on each individual item of interest but does not consider higher-level concepts. The Interactive Specification methodology does have a theoretical basis which is a binary model. It classifies concepts into different types and constructs hierarchies of concepts which serve as basic units of information.

These view modelling approaches are summarized in Table I.

DISCUSSION

These approaches to view modelling are compared on three dimensions - ease of use, semantics and degree of automation.

Ease of Use

Two of the dimensions that Bracchi [1981] identifies for evaluating view modelling techniques are:

1. Naturalness – can the modelling be done in terms familiar to end-users?
2. Simplicity – are there too many constructs and too many ways to specify the same things?

Together, these could be considered as "ease of use".
### TABLE I: View Modelling Approaches

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inputs</strong></td>
<td>• Entities, Relationships, Attributes</td>
<td>• Abstract data types (objects)</td>
<td>• Objects (entities &amp; association types), Connectors</td>
<td>• Data items</td>
</tr>
<tr>
<td><strong>Processing Characteristics</strong></td>
<td>• Some semantic analysis</td>
<td>• Generalization and aggregation hierarchies representing abstract data types</td>
<td>• Semantic information from three types of simple associations</td>
<td>• User responsible for constructing views in 3NF</td>
</tr>
<tr>
<td></td>
<td>• Data grouping</td>
<td></td>
<td></td>
<td>• No semantic information</td>
</tr>
<tr>
<td><strong>Outputs</strong></td>
<td>• Hierarchical, DBTG-types and relational schemas</td>
<td>• Relational, DBTG-type schemas</td>
<td>• Relational &amp; DBTG-type schemas</td>
<td>• Canonical representation of user’s view (DBMS independent)</td>
</tr>
</tbody>
</table>

---

a. **Naturalness**

In general, data should be grouped into units which a user finds easy to think about although this grouping may be different for different users [Morgenstern, 1981]. In terms of modelling the real world, the E-R model, the Data Abstraction methodology and the N-S model can be considered as top-down in their approaches. They start with concepts such as entities or associations and use semantic information to organize data elements into relationships that represent these concepts.
In the Interactive Specification methodology, the user is allowed to express his requirements without concern for whether he is referring to data items or higher-level concepts. Certain forms of input are encouraged which allow concepts to be grouped into hierarchies. These hierarchies constitute the basic units of information in this approach. Hence, this methodology can also be classified as top-down.

In contrast, Bubble Charting is a bottom-up approach because it starts with individual data items without concern for formulating generic concepts.

A human is capable of aggregating entities within an abstract concept and is accustomed to thinking in this way [Baldissera et al., 1979]. This implies that a top-down approach to specifying information requirements\(^1\) is preferred. The Data Abstraction methodology appeals to the natural grouping (i.e. abstractions) that a human makes and concentrates on the different levels of detail that are of interest to the user. Therefore, the concepts of the Data Abstraction methodology are also desirable properties of a view modelling approach.

b. Simplicity

The E-R model is noted for being simple and can thus be understood by non-specialists [Konsynski, 1979]. Its two constructs, entities and relationships, are natural ones for modelling [Brodie, 1984].

The Interactive Specification methodology is also simple to use. The user is not required to distinguish among any constructs. He is only restricted to describing his application in the form \(A\) verbal construct \(B\) where the verbal constructs are simple and familiar (e.g. \(is\) or \(has\)).

\(^{1}\) For the purposes of this paper, the terms \(information\ requirements\) and \(user\ view\) are used interchangeably.
Bubble Charting requires that the user be familiar with only one construct - the data item. However, the user must learn the principles of 3NF because it is he who is responsible for ensuring that his view is constructed in this manner. Thus, this method is not as simple as it might initially seem. Also, it is not obvious that the user is the best person to be given this responsibility – he may or may not have an aptitude for or be interested in learning how to model his own views. Furthermore, it is easy to imagine that one user would construct more accurate views than another.

The Data Abstraction methodology also employs only one basic concept – an object which represents both entities and relationships. To use this approach, the user must be able to classify the objects of his application and group them into hierarchies. When the problem becomes large, this may be difficult to do, especially if the methodology is applied to the community view of data instead of an individual user view [Navathe and Schkolnick, 1978].

Of the five approaches, the N-S model is the most complicated and has the greatest number of constructs. Unlike the other methods, it concentrates on the instance level of data instead of the conceptual level. Due to the large number of concepts which must be understood and applied to an application domain, this model may be very difficult to use. Thus, the model sacrifices simplicity in order to capture more of the processes for which the data will be used.

Semantics

Bubble Charting is the only approach which makes no attempt to incorporate some degree of semantic information. The Interactive Specification method obtains its semantic information by querying the user. Semantics are captured in the E-R model through the concepts of roles and set membership. In the Data Abstraction approach, the semantic processes of
aggregation and generalization are used to build hierarchies of objects which represent views [Yao et al., 1978].

The N-S model makes the greatest attempt to capture semantic information through its various categories of object types and, in particular, its different kinds of simple associations (categorization, selection, and subsetting). In doing so, it captures more of the meaning of the data at a processing level (e.g. when data is inserted or deleted).

**Automation**

The issue of automation of the database design process has been raised by several researchers. Navathe and Gadqil [1982] argue that major problems in database design are due to the lack of a structured design methodology and the lack of automatic aids for developing complex databases. They present a methodology for automating the view integration process and illustrate using views which are based on the N-S model.

Bouzeghoub et al. [1985] call for automatic tools and human interaction to produce an internal schema from a conceptual schema. They approach this automation by applying expert systems technology. Mylopoulos [Reiter et al., 1983] also supports using expert systems to aid the database design process. He suggests that an expert system be provided with information on the characteristics of a database which it would use to generate automatically or semi-automatically a database design.

View modelling is currently being carried out in a manual manner with little discipline [Bracchi, 1981]. The E-R and N-S models and the Data Abstraction methodology all assume that the view modelling task is performed by a "database design expert" who obtains information about the user's need through interviewing, examining existing documents and other traditional means. To this extent, they suffer from three weaknesses:
1. they require the use of a scarce resource – the expert database designer;

2. the designer’s knowledge of an application is necessarily second-hand;

3. the quality of the design produced, regardless of the view modelling approach used, depends greatly upon the skill and experience of the designer.

If the methodology of the expert database designer could be codified in a computer program, it could be replicated and applied wherever needed – thereby alleviating the scarcity of human experts. Then the user, who best understands his application, could become the designer of his own view.

The Interactive Specification methodology is the only one which attempts to automate the generation of user views. Even this approach assumes that a system will only assist the user in performing view modelling (e.g. by making syntactic checks on the consistency of the user’s input). Hence, the need exists for automation of the view specification process.
III. EXPERT SYSTEMS

This section describes the relevant background literature and theory of expert systems and, where appropriate, applies them to the research question.

Expert systems are systems which attain expert-level performance in solving complex, real-world problems. Their power is obtained from large bodies of task-specific or domain knowledge, i.e. facts and procedures, gleaned from human experts, which have proven useful in solving problems in the application domain [Barr and Feigenbaum, 1982].

Clifford et al. [1983] describe the architecture of an expert system as being comprised of three main elements: a knowledge base consisting of domain knowledge; an inference engine which uses the facts and rules in the system’s knowledge base to reason about the application domain; and a user interface for knowledge acquisition and end-user consultation. A more detailed description of an ideal expert system is provided by Hayes-Roth et al. [1983]. It also includes a blackboard for storing intermediate results and a justifier for explaining the system’s reasoning to a user.

Specific characteristics of expert systems include performing tasks at expert levels, using domain specific problem-solving strategies, employing self-knowledge to reason about their own inferences and containing large amounts of domain specific knowledge to shorten the deduction process.

The major components of expert systems, including types of expert systems, knowledge representation, inference engines, explanations of reasoning and user interfaces, will be discussed.

TYPES OF EXPERT SYSTEMS

Stefik et al. [1982] identify six main types of expert systems: interpretation, diagnosis,
Interpretation systems infer situation descriptions from observables [Hayes-Roth et al., 1983]. They are required to find consistent and correct interpretations of data although the data they work with is often incomplete and unreliable. Consequently, interpretation systems must identify where they make assumptions which enable them to cope with partial information [Stefik et al., 1982]. These systems perform surveillance, speech understanding, image analysis, chemical structure eludication and signal interpretation [Hayes-Roth et al., 1983].

Diagnosis systems infer system malfunctions from observables. This category includes the diagnosis of infectious diseases in medical applications as well as electronic, mechanical and software diagnosis [Hayes-Roth et al., 1983]. Such systems often work with problems which have large amounts of data that need to be classified into categories. Many interacting events can obscure the data so statistics or fuzzy logic may be employed to provide a range of possibilities instead of a yes or no answer [Sowa, 1984].

Monitoring systems continuously interpret signals and set off alarms when intervention is required. When alarm conditions must be carried out, action is required on a real time basis. These systems must also avoid setting off false alarms [Stefik et al., 1982]. Monitoring systems are being studied for nuclear power plant, air traffic, disease and regulatory tasks [Hayes-Roth et al., 1983].

Prediction systems infer likely consequences from given situations and must often work with incomplete and diverse data. They should be able to account for multiple possible futures, thus generating a large number of possible scenarios [Hayes-Roth et al., 1983; Stefik et al., 1982]. These systems are useful for weather forecasting, demographic predictions, traffic predictions, crop estimations and military forecasting [Hayes-Roth et al.,
Planning systems solve problems which require a plan of action. The objective of this type of expert system is to construct a plan which achieves a goal without using too many resources or violating constraints. When goals are in conflict, priorities must be established [Stefik et al., 1982]. The problems these systems address include automatic programming, and robot, project, route, communication, experiment and military planning problems [Hayes-Roth et al., 1983].

Design expert systems search for a combination of structures to satisfy a particular goal, e.g. selecting components for a computer configuration or suggesting an experiment for molecular genetics. In contrast to diagnosis systems, design systems use exact reasoning on highly structured data [Sowa, 1984].

**KNOWLEDGE**

The preeminent aspect of any expert system is its inherent knowledge reflecting its encoded expertise. Different types of knowledge that a system may possess are [Barr and Feigenbaum, 1982]:

1. facts about the domain
2. hard-and-fast rules or procedures
3. heuristics or "rules of thumb"
4. global strategies
5. a "theory" of the domain.

The research question provides examples of each of these.

*Domain Knowledge*

In the research question, domain knowledge consists of knowledge about the database design
issue of view generation. This knowledge could refer to how entities, relationships, and attributes are identified and organized.

\textit{E.g.}

1. The view is modelled in terms of entities and relationships.
2. A relationship may exist either between entities or attributes.
3. The minimum and maximum mapping ratios for an entity \( A \) which is related to entity \( B \) are \((1,1)\) if there is one and only one value of \( B \) for each value of \( A \).

\textit{Hard-and-fast Rules or Procedures}

Examples of hard-and-fast rules or procedures are:

1. If an entity exists, then determine its corresponding attributes and candidate keys.
2. A transitive functional dependency exists when a non-key attribute depends on another non-key attribute instead of the key attributes.

\textit{Heuristics}

Heuristics are indeterminate pieces of knowledge about generating a user view that are gleaned from database design experts.

\textit{E.g.}

1. If an entity has an attribute of the form \textit{prefix\_id}, then \textit{prefix} may also be an entity.
2. When choosing a primary key from a set of candidate keys, give preference to the candidate keys which have the smallest number of elements (i.e. the least number of key attributes).

\textit{Global Strategies}

Examples of global strategies are:

1. A candidate key can always be comprised of all of the attributes of an entity.
2. A relationship may be represented by either: i) creating a relation or ii) adding the key attributes of one of the entities in the relationship as non-key attributes of the other entity.

Theory of the Domain

A theory of the domain corresponds to overall database design strategies.

E.g.
1. The Entity–Relationship model can be used to model a user’s information requirements.
2. Relationships can be expressed with a binary model.

KNOWLEDGE REPRESENTATION

An expert system must have knowledge of its domain before it can function effectively. In addition, this knowledge must be represented in a manner which lends itself to reasoning [Sowizral, 1985]. The basic problem in knowledge representation is the development of a sufficiently precise notation with which to represent knowledge [Mylopoulos and Levesque, 1984].

This section discusses four of the main approaches to knowledge representation – Production Rules, Logic, Semantic Nets and Frames – and where possible, relates them to the research question. For a more thorough overview of knowledge representation see Mylopoulos and Levesque [1984].

Production Rules

The most popular approach to representing knowledge in expert systems is by means of "IF–THEN" or production rules [Hatton, 1983]. A rule of this type usually corresponds to a particular "chunk" of knowledge about an application domain [Duda, 1981]. The "IF" or condition part of the rule (also called the premise or antecedent) is often a predicate or
conjunction of predicates which must be present before the "THEN" or action part can take place.

\[ \text{E.g. (from the research question)} \]
\[ \text{IF: entity}(A) \text{ and entity}(B) \text{ and } A \text{ is-a } B; \]
\[ \text{THEN: all the properties or characteristics of } B \text{ are attributable to } A. \]

This rule is interpreted as: *If both A and B are entities and A is a subset of B (condition); then one can attribute all the properties or characteristics of B to A (action).*

There are several distinguishing characteristics of expert systems (called *rule-based systems*) whose knowledge representation is based upon production rules.

1. **Modularity**

Rule-based systems are modular so individual rules, representing individual pieces of knowledge, can be added to, deleted from, or updated within the knowledge base without affecting other rules [Barr and Feigenbaum, 1981]. Hence, the modularity of these systems helps to reduce the programming effort required. Also, in some goal-oriented systems, the order in which the rules are placed in the knowledge base may be unimportant. If this is so, new rules can be easily added in a process which can sometimes be totally automated [Davis and King, 1977]. However, in large systems, the modularity may become difficult to maintain due to constraining interactions between rules which may result in inefficiencies [Barr and Feigenbaum, 1981].

2. **Uniformity**

Rule-based systems have a uniform structure (all the rules are encoded in IF-THEN form). This representation is therefore appropriate for systems which are designed incrementally [Barr and Feigenbaum, 1981; Davis and King, 1977].
3. **Naturalness**

According to Newell (see Davis and King, [1977]), rule-based systems provide a means of representation which is very similar in structure to fundamental mechanisms of human cognition. Statements about what to do in predetermined situations are naturally encoded as production rules. Furthermore, these are the kinds of statements that a human expert most frequently uses when explaining how he performs tasks requiring his expertise [Barr and Feigenbaum, 1981].

Rule-based systems have found applications in many different domains. Some of these systems are identified below and examples are provided from their knowledge bases.

1. **Medicine**

   a. MYCIN [Shortliffe, 1976], an interactive consultation system which performs diagnosis and recommends treatments for infectious blood diseases, is probably the best known rule-based system. Its knowledge base consists of a set of facts and production rules. The facts are represented as relations whose fields contain attributes, entities, attribute values and a certainty factor which indicates the strength of the evidence. For example [Sowa, 1984]:

   (SITE CULTURE-1 BLOOD 1.0)
   (IDENT ORGANISM-2 KLEBSIELLA .25).

   These facts are mapped to English sentences and read as:

   The site of culture-1 is blood.
   There is weakly suggestive evidence (.25) that the identity of organism-2 is klebsiella.

   An example of a MYCIN production rule is:

   **IF:**
   1) the site of the culture is blood, and
2) the identity of the organism is not known with certainty, and
3) the stain of the organism is gramneg, and
4) the patient has been seriously burned;
**THEN:** there is weak suggestive evidence (.4) that the identity is pseudomonas.

This rule is interpreted as: *If the above four symptoms hold; then this constitutes as weak evidence (.4 on a scale of -1 to +1) that the identity is pseudomonas.*

b. Other rule-based systems found in medical applications include: i) ONCOCIN [Shortliffe et al., 1981], a system designed to assist physicians in the treatment of cancer patients; and ii) PUFF [Feigenbaum, 1977], a system for diagnosis of pulmonary function diseases.

2. **Chemistry:**

DENDRAL [Buchanan et al., 1969] is a system which aids chemists in the identification of molecular structures of unknown compounds. An interesting feature of this system is that its development involved embedding a chemist's knowledge about mass spectrometry into rules usable by the program, without attempting to model the chemist's thinking [Davis and King, 1977].

DENDRAL's rules indicate how bonds will break and atoms migrate to produce data points. For example [Shortliffe, 1976]:

**Rule:** (CH)−CH2−NH−CH2−CH2−(CH) − > Breakbond (4,5)

This rule implies that a seven-member chain with characteristics as identified on the left-hand-side will have a bond break between atoms 4 and 5 when mass spectral bombardment occurs.

3. **Computer Design:**

R1 [McDermott, 1982] is a system which serves as a computer configurator. From a
customer's order for a VAX-11/780 computer system it determines what, if any, modifications need to be made to the order to ensure that the system would function properly. It produces as output a list of the components required and a diagram of the final configuration. When solving a design problem, it uses its domain knowledge to generate a single, acceptable solution instead of examining different hypotheses until an acceptable one is found.

Its rule base is a set of production rules, an example of which is given below.

**IF:**

a. the most current active context is distributing massbus devices  
b. and there is a single port disk drive that has not been assigned to a massbus  
c. and there are no unassigned dual port disk drives  
d. and the number of devices that each massbus should support is known  
e. and there is a massbus that has been assigned at least one disk drive  
   and that should support additional disk drives  
f. and the type of cable needed to connect the disk drive to the previous device on the massbus is known

**THEN:** Assign the disk drive to the massbus.

The above is an English translation of the rule. Unlike MYCIN, R1 does not map its rules to an English translation. It also does not use fuzzy logic. A component is either present in the configuration or it is not, so its certainty factors (when components are present) are 1.0 [Sowa, 1984].

4. **Meta-level Systems:**

Meta-level systems contain meta-level knowledge that is used for building or learning about other expert systems.

a. TEIRESIAS [Davis, 1977] is an interactive system which assists in the transfer of knowledge from a domain expert into a knowledge-based system. The human expert communicates with an object-level performance program (e.g. MYCIN)
via TEIRESIAS (the meta-level program). Using TEIRESIAS, the human expert tries to discover how the object-level program is performing and modifies its rules if necessary (e.g. by adding a new rule). Meta-rules are used in conjunction with object-level rules and provide a means for encoding problem solving strategies which occur in the domain of the performance program.

The knowledge base for the performance program is a collection of judgmental production rules which make inexact inferences. For an "investment" example, a typical rule is [Davis, 1977]:

RULE 27:
IF:
1) the time scale of the investment is long-term,
2) the desired return on the investment is greater than 10% and
3) the area of the investment is not known;
THEN: AT&T is a likely (.4) choice for the investment.

Meta-level knowledge is represented by rule models which are abstract descriptions of subsets of rules such as the one above. The rule models are built from empirical generalizations about the rules, and are used to characterize a typical subset member.

b. GUIDON [Clancey, 1979] is a tutorial system which uses the knowledge base of MYCIN (or MYCIN-like systems) to teach medical students facts and problem-solving strategies [Duda, 1981]. The system's teaching expertise contains capabilities both for measuring a student's competence and for using this measure as a basis for selecting knowledge to present to the student.

GUIDON contains two sets of rules - domain rules and tutorial rules. Tutorial rules represent the system's teaching strategy while the domain rules contain information from MYCIN's knowledge base. Meta-knowledge about the
representation and application of domain rules is contained in the tutorial rules and is used to examine the domain rules and reason about how they are used. This ability enables the system to make multiple use of any given domain rule. For example, a domain rule can be examined to determine if its subgoals are needed before it can be applied, to generate different questions for a student, or to ascertain whether it would be useful in understanding a student's hypothesis.

An example of a tutorial rule is given below. This rule indicates which domain rule the system should mention in order to help a student during a tutorial [Clancey, 1979]:

T-RULE 26.03
IF: The recent context of the dialogue mentioned either a "deeper subgoal" or a factor relevant to the current goal
THEN: Define the focus rule to be the domain rule that mentions this focus type.

c. Other examples of meta-level systems are: i) EMYCIN [van Melle, 1979], a problem-domain-independent version of MYCIN which contains a general-purpose scheme to aid in the construction and testing of expert systems for diagnosis and consultation; ii) EXPERT [Weiss et al., 1981], a general purpose system for building consultation systems; and iii) Meta-DENDRAL [Buchanan and Feigenbaum, 1978], an automatic theory formation program which produces rules that associate substructures of a molecule with fragmentation (migration) processes.

In terms of Stefik et al.'s categorization of types of expert system, the applications from medicine are diagnosis systems while the chemistry application is an interpretation system. R1 and the meta-level systems (with the exception of GUIDON which is a tutorial system) are design systems. Other rule-based systems are found in various application
domains (e.g. diagnosis and treatment of nuclear reactor accidents [Nelson, 1982] and engineering structural analysis [Bennett and Engelmore, 1979]).

Logic

In logical representation schemes the formal treatment of knowledge and thought, as developed in philosophy, has been applied to the development of computer programs which can reason [Barr and Feigenbaum, 1981]. A logical representation scheme uses constants, variables, functions, predicates, logical connectives and quantifiers to represent facts as logical formulas in some logic. This logic may be first or higher order, modal, fuzzy, etc. A knowledge base is then a collection of logical formulas that provide a partial description of a situation. Modifications to the description are made by adding or deleting logical formulas. In this manner, logical formulas become the atomic units for knowledge base manipulation [Mylopoulos and Levesque, 1984].

Logic is a natural way to express certain notions and is precise, flexible, and modular [Barr and Feigenbaum, 1981]. Inference rules allow one to operate on logical formulas to define proof procedures which can be used for information retrieval, semantic consistency checking, and problem solving [Mylopoulos and Levesque, 1984]. The major disadvantage of logic is that it separates the representation of knowledge from its processing [Barr and Feigenbaum, 1981].

The AI language, Prolog, has been developed as a practical logic programming language which is based on first-order logic but uses only a subset of it. The language combines the advantages of logical and procedural representation schemes and thus overcomes some of the problems of first-order logic. As a tool based on logic, it has been argued that Prolog is an appropriate one for developing expert systems [Coelho, 1983; Clark and McCabe, 1980; Hammond, 1982]. Examples of expert systems implemented in Prolog
may be found in Bouzeghoub et al. [1985], Pereira and Oliveira [1983], and Hammond [1982].

The following illustrates how a Prolog predicate could be used to express information about a user view.

*E.g.*

relationship(A,VP,B):- entity(A), entity(B), verb(VP).

"A VP B" is a relationship if both "A" and "B" are entities and "VP" is a verb.

Other forms of logical representation schemes are also found in expert systems. For example, PROSPECTOR [Duda et al., 1979] uses fuzzy-set formulas when evaluating how the change in the probability of one assertion affects the probability of another assertion. In fuzzy-set formulas, the probability of a hypothesis being true, where the hypothesis is defined as a logical conjunction (AND) of several pieces of evidence, is the minimum of the probability values of the individual pieces of evidence. Similarly, a hypothesis, defined as the logical disjunction (OR) of the evidence which supports it, is assigned a probability value equal to the maximum of the values assigned to each of the individual pieces of evidence. The underlying characteristic of this approach is that no partial credit is given, especially in conjunction cases where all but one of the pieces of evidence is known [Duda et al., 1979].

**Semantic Nets**

A semantic network represents knowledge as a collection of objects (nodes) and binary associations (directed labelled edges). The objects represent individuals or concepts while the binary associations represent binary relations over the concepts. A knowledge base is then a collection of objects and relations defined on them. Modifications to the knowledge base occur through insertion and deletion of the objects and manipulation of the relations [Mylopoulos
and Levesque, 1984]. Finally, a system's reasoning corresponds to traveling along the arcs in the network or pattern matching a problem description to sub-networks [Clifford et al., 1983].

Semantic nets appear in various forms and are used for many different purposes. They are popular because they provide a very natural approach which is reminiscent of human thinking [Gevarter, 1984]. Mylopoulos and Levesque [1984] provide several examples of different kinds of semantic nets: classification, aggregation, generalization and partitions. Generalization, for instance, relates one type to a more generic type. This relation is often called an *is-a* and types are organized into *generalization* or *is-a* hierarchies.

*E.g.*
Manager is-a Employee
Employee is-a Person.

Semantic networks are a natural knowledge representation scheme for domains where much of the reasoning is based on a very complicated taxonomy [Barr and Feigenbaum, 1981]. Such a situation is found in the expert system PROSPECTOR [Duda et al., 1979].

In PROSPECTOR's knowledge base, semantic nets, inference rules (which correspond to production rules) and fuzzy logic are combined to represent models of ore deposits. The semantic net is called an inference net and represents relations between field evidence and geological hypotheses. Both evidence and hypotheses are referred to as assertions in the network. For example: *The alterations suggest the potassic zone of a porphyry copper deposit* is an assertion which may be true, false, or known to have a certain probability of being true. Probability values are usually attached to each assertion. Most of the arcs in the inference net represent the inference rules that indicate how the probability of one assertion affects the probability of another. The other arcs identify situations where one assertion is a prerequisite for another; *e.g.* there must be *granitic intrusives in the region*
before there is a favourable regional environment.

The main or top-level assertion in the inference network corresponds to the available evidence matching a particular ore model. To establish the top assertion, various subhypotheses need to be established: e.g. the petrotectonic setting must be favourable for PCDA (porphyry copper deposit); the regional environment must be favourable for PCDA; and there must be an intrusive system that is favourable to PCDA before concluding that PCDA exists with a certain degree of confidence.

In general, the top-level assertion (e.g. PCDA) is determined by several second-level assertions, each of which may be determined by third-level assertions, etc. When the user provides evidence which corresponds to these assertions, the probabilities of various hypotheses involved will be affected. Probability formulas determine how the probabilities of related hypotheses are changed and, in this manner, the effect of the evidence is propagated through the net.

Although semantic nets tend to be found in AI applications other than expert systems, there are some very large expert systems, besides PROSPECTOR, which rely on semantic nets [Clifford et al., 1983]. Examples of such systems include: INTERNIST [Pople, 1977], a consultation system for internal medicine which contains one of the largest rule-bases (85% of internal medicine); and SOPHIE [Brown et al., 1981], a system which acts as an electronic laboratory instructor that assists a student in debugging a piece of malfunctioning equipment [Brachman, et al., 1983].

Frames

The idea of a frame representation was originally proposed by Minsky [1975]. A frame is a complex data structure for representing stereotype situations [Mylopoulos and Levesque, 1984]. The underlying assumption of a frame is that a great deal of information can be
obtained from applying general concepts to a specific problem where the general concepts have been gained from experience and expectations of previous situations [Clifford et al., 1983].

A frame data structure has slots which represent both the objects which can occur in a situation and the relations among the objects. Attached to each frame is information such as: how to use the frame; what to do if something unexpected happens; and default values for slots. The knowledge base for a system then becomes a collection of frames [Mylopoulos and Levesque, 1984].

An example from the field of database view creation is given below. It illustrates how a frame could be used to represent an entity in a database design.

\textit{E.g.}

Generic ENTITY Frame

- \textbf{Self}: a "thing" of interest in a database
- \textbf{Name}: a noun or group of nouns
- \textbf{Attributes}: properties or characteristics of the entity.
- \textbf{Key}: an attribute or group of attributes
  (default is entity's name-"id")

ENTITY-IN-DATABASE Frame

- \textbf{Self}: an Entity
- \textbf{Name}: LIBRARY
- \textbf{Attributes}: [library-id, address, director-id]
- \textbf{Key}: library-id

An example of an expert system based on the frame approach is found in WHEEZE [Smith and Clayton, 1980]. WHEEZE performs medical pulmonary function diagnosis based upon clinical test results. Assertions are represented as frames in the knowledge base.
with antecedent sub-assertions appearing in a Manifestation slot. Corresponding numbers indicate the assertion's importance while certainty factors and findings have separate slots in the assertions. Assertions in the SuggestiveOf and ComplementaryTo slots indicate those things that could be investigated if the original assertion is confirmed or denied, respectively.

A HowToDetermineBelief slot indicates how an assertion's belief is determined.

E.g. OADwithSmoking

Manifestation

(OAD-Present 10) (PatientHasSmoked 10)
(PatientStillSmoking)

SuggestiveOf

((SmokingExacerbatedOAD 5)
(SmokingInducedOAD 5))

ComplementaryTo

((OADwithSmoking-None 5))

Certainty

1000

Findings

"Discontinuation of smoking should help relieve the symptoms."

HowToDetermineBelief function for computing the minimum of the beliefs of the manifestations.

Possible interpretation:
The severity of the Obstructive Airways Disease (OAD) is more than mild. The patient has smoked and continues to do so. Therefore, discontinuing smoking should help relieve the symptoms.

Although frames have played a key role in knowledge representation research [Mylopoulos and Levesque, 1984] and the basic idea of a frame structure is promising, there are still many issues about the possible implementation of frame-based systems which are unresolved [Barr and Feigenbaum, 1981]. (See Mylopoulos and Levesque [1984] for examples of representation schemes which have been developed based on the frame concept.)
Summary

Of the four types of knowledge representation schemes discussed, production rules are the most common form of knowledge representation in expert systems. Semantic nets are more common in other AI applications (e.g. natural language processing) [Clifford et al., 1983]. However, they are appearing in some very large expert systems (sometimes in connection with other knowledge representation forms). Development of expert systems based on logic has been mostly through the programming language Prolog. Finally, frames still have unresolved issues which must be dealt with before they can be effectively used for knowledge representation in expert systems.

INFERENCE MACHINE

The quality of an expert system's reasoning depends on how well relevant facts and principles can be accessed and how complete the inference procedure is [Brachman et al., 1983]. In an expert system, an inference machine is used to relate a problem description to the system's knowledge in order to analyze the problem. The inference machine can be a pattern matcher, a theorem prover, a compiler or a knowledge representation language such as Prolog or EMYCIN [van Melle, 1979], etc. [Clifford et al., 1983].

Direction of Inference

There are two main approaches to solving problems or achieving goals in rule-based systems - forward chaining or bottom-up versus backward chaining or top-down reasoning. Combinations of these strategies are also possible.

A forward chaining system begins with available information and tries to draw conclusions. When the conditions required in the premises (or antecedents) of certain rules
can be satisfied, important conclusions can be drawn (the action parts of the rules). These conclusions may then be used to satisfy other premises which in turn cause other conclusions to be made, etc. until the overall goal can be reached.

A backward chaining system examines the action parts of the rules to find out which ones conclude the current goal. It considers the left-sides of those rules to determine which conditions invoke them and then seeks other rules whose action parts conclude these conditions, etc. until the overall goal can be solved [Barr and Feigenbaum, 1981].

For example, in MYCIN rules are invoked in a backward-chaining fashion that results in an exhaustive depth-first search of an AND/OR goal tree [Barr and Feigenbaum, 1982]. Suppose that MYCIN is trying to identify an organism. It considers all the rules that make the desired conclusion – that is, all the rules that mention *identity of bacteria* in their action clause, and looks at each one in turn to see if its premise clauses are met. For each premise, it may set up a subgoal with the system then searching for rules whose conclusion match that of the subgoal. The search is *depth-first* because each premise condition is considered in turn and it is also exhaustive because all the rules that are applicable are considered.

In an ideal expert system, the inference machine never needs to be modified. Instead, any change in the system's behavior is the result of a change in the rules that occur in the knowledge base [Barstow et al., 1983]. Barstow et al. provide the following example to illustrate a simple back-chaining inference machine.

*To test whether hypothesis X is true*

- if $X$ is stored in the global data base
  - then $X$ is true
- else if there are any rules whose consequents include $X$
  - then for each such rule:
    - if all antecedents are true
      - then add all consequents to the global data base
and $X$ is true
else if the user says that $X$ is true
then $X$ is true
else $X$ is false.

The antecedents, or premises, of a rule are checked which causes the inference engine to be invoked recursively. The recursion implements the back-chaining procedure.

EXPLANATION OF REASONING

The capability of a system to be able to explain its reasoning is very important, especially in consultation systems where the user is seeking advice. This capability allows the user to judge whether or not he wishes to follow the advice of the system or, alternatively, to detect flaws or omissions in the system's knowledge base and to update the knowledge base accordingly.

Most rule-based systems explain their reasoning by presenting the chain of rules which were activated by the inference engine in obtaining the given solution. A system is able to do so because its knowledge base is completely separated from its inference mechanism [Clifford, et al., 1983]. For expert systems which are based on probabilistic reasoning, a corresponding probability or certainty factor is provided indicating the degree of certainty the system has in its solution. An example of a system's explanation of its reasoning is taken from MYCIN [Shortliffe, 1980].

** Why did you treat E.coli?

ITEM-1 is E.coli associated with meningitis. I decided to treat ITEM-1 on the basis of clinical evidence only.

The following rules concluded about: the likelihood that e.coli is one of the organisms (other than those seen on cultures or smears) which might be causing the infection

RULE 45 was used to conclude that e.coli is one of the organisms (other than those seen on cultures or smears) which might be
causing the infection (.74).

RULE 557 was used to conclude that e.coli is one of the organisms (other than those seen on cultures or smears) which might be causing the infection (.93).

The parameters most frequently used in these rules:
the infection which requires therapy
whether a smear of the culture was examined
the type of the infection

The tutorial system GUIDON augments the explanation capabilities of MYCIN (or MYCIN-like expert systems) to teach a student about the expertise which is contained in MYCIN's knowledge base. The system uses both the knowledge base rules and its own knowledge of a student's competence and interests to explain various medical procedures. The following example illustrates how GUIDON tutors a student about treatment for a burn patient [Clancey, 1979]:

You should now proceed to ask questions that will enable you to make a hypothesis about the organisms (other than those seen on cultures or smears) which might be causing the infection.

** How is the fact that the patient is burned relevant?

The fact that patient-326 is seriously burned will be used to conclude that *Pseudomonas aeruginosa* is one of the organisms which might be present (in addition to those seen on cultures or smears) [Rule 578].

** Help

Try to determine the type of the infection: bacterial, fungal, viral, or TB.

** What is the patient's WBC?

With rule models, such as those found in TEIRESIAS, the system can explain its reasoning by supplying an overview of the knowledge in the relevant rules. This aids in
providing the user with some indication of the approach taken by the system [Davis, 1979].

\[E.g. \text{[Davis, 1979]}\]
\[How \ do \ you \ decide \ that \ the \ utilities \ group \ is \ an \ appropriate \ area \ for \ investment?\]

Rules which conclude that an appropriate area for investment is the utilities group generally use one or more of the following pieces of information:

- the desired rate of return of the investment
- the income-tax bracket of the client
- the time-scale of the investment

Furthermore, the following relationships hold:

If a rule mentions the income-tax bracket of the client, it also typically mentions the time scale of the investment as well.

RULE 247, RULE 45, RULE 123, and RULE 206 conclude that an appropriate area for investment is the utilities group. Which of these (if any) would you like to see? . . .

USER INTERFACE

Expert systems usually employ natural language parsers to interpret user inputs and use less sophisticated techniques with canned text to generate responses to the user [Hayes-Roth et al., 1983]. For example, in the expert system TEIRESIAS [Davis, 1977] questions and responses are either preformatted or obtained by filling in templates with appropriate words. Free text is analyzed based upon a simple keyword oriented approach.

\[E.g. \ (From \ TEIRESIAS, \ Davis \ [1977])\]

1) Investor's name?  
** FRED SMITH
2) Age?  
** 34
3) Present income?  
** 25,000
4) Number of dependents?
5) Occupation? ** ELECTRICIAN  
6) Amount of previous investment experience?  
(slight = less than one year; moderate =  
2 to 4 years; extensive = more than 4) ** MODERATE  
7) Does the investor subscribe to the Wall Street Journal, or any securities market newsletters? ** Y  

In PROSPECTOR, information provided by the user is restricted to simple statements in constrained English which give information on the names, ages, and forms of rocks and the types of minerals present. The user's statements are parsed by a natural-language interface facility called LIFER and represented as partitioned semantic networks [Barr and Feigenbaum, 1982].

Other approaches to the interface include graphics [Bouzeghoub et al., 1985] and natural language (e.g. ORBI [Pereira and Oliveira, 1983]).

EXPERT SYSTEMS AND DATABASE DESIGN

The most notable previous research conducted in the area of expert systems and database design has been reported by Bouzeghoub et al. [1985]. An expert system, SECSI (Systeme Expert en Conception de Systemes d'Informations), has been designed as an integrated intelligent tool for aiding users in database design. It is intended to aid, not replace, a human database designer. The system, which is developed in Prolog, starts from a naive description of the application and engages the user in a dialog to obtain the necessary input for a logical database design. The output produced is a semantic network representing the application. Design rules are applied to this representation and a set of normalized relations
is produced.

Procedure

The user provides an initial description of his application. This description includes: "is-a" relationships (e.g. employee is-a person); n-ary associations between entities (e.g. student is enrolled in one or more courses); names and types of attributes (e.g. "name" is an attribute of person and its type is "text"); and constraints indicating functional dependencies (e.g. the name of a department determines its address). From this initial description, the system performs a verification step which tries to detect and resolve (with the assistance of the user) inconsistencies such as synonyms, homonyms, etc. The next step, the relational step, interactively obtains constraints and choices for first normal form relations. The final step is a normalization procedure. Normalized relations are obtained using functional dependencies, which were identified by the user in the initial requirements specification, and cardinalities of associations. The cardinalities are either supplied by the user or elicited by the system and allow the system to make inferences about functional dependencies.

Interfaces

The system has two interfaces, one for a database design expert who wishes to specify his database expertise and one for the end-user. The end-user may use any, or a combination of, the following: i) a simple but formal declarative language (e.g. EMPLOYEE:PERSON represents the fact that an employee is a person); ii) a restricted subset of natural language (e.g. EMPLOYEES AND STUDENTS ARE PERSONS); or iii) a graphics interface.

For the database design expert, two types of interfaces are proposed. One is a graphical tool for expressing mappings between two types of semantic networks. The other
is a declarative language which allows an expert to express statements in the form of production rules. As these are only proposed forms of interfaces, the rules are currently represented as Prolog clauses.

**Knowledge Base**

The system's knowledge base uses a kind of semantic network which employs concepts from semantic data models such as aggregation, generalization and classification. It also contains three kinds of production rules: i) rules which ensure that the conceptual model, as described by the semantic network, is consistent and can be transformed into a normalized schema; ii) rules describing general knowledge such as the types of nodes and arcs in the semantic net; and iii) a hierarchy of meta-rules which control the sequence of the execution of the rules.

**Discussion**

Although the development of the system is not complete, the results of this research suggest that it is feasible to apply expert systems technology to the database design task. The completed version of the system should be flexible and possess all the major components of an expert system (e.g. provisions to modify the knowledge base; user-friendly interfaces; capabilities for explaining the system's reasoning; etc.)

In this research, the database design methodology is divided into three steps: i) view specification and integration; ii) logical schema design; and iii) physical design. SECSI addresses the second phase while a second and third version of the system are being specified for view integration and physical design. However, there are no provisions reported for future work on view specification which is the focus of this dissertation.
Although Bouzeghoub et al. claim that their system starts from a naive description of the application, it seems to assume that a user view is available. Hence, the View Creation System could be considered complementary to the SECSI system.
IV. RESEARCH DESIGN

The research question, "Can the process of generating user views be formalized and subsequently automated in order to increase the quality and consistency of database design while relying less on scarce expert human skills?", is addressed by formalizing, as a set of rules, a methodology for creating user views. The rules comprise the knowledge base of an expert system, the View Creation System, which automates the generation of user views. The validation of the formalization occurs when the expert system, and hence the rule set, is applied to selected database design problems and the results produced evaluated. Whereas the most visible product of the research is the system itself, the main objective is the formalization of the view specification process. The purpose of the expert system is to provide a vehicle for demonstrating that the set of rules produced is adequate and complete.

This section examines the underlying theory of the expert system and presents the system's main characteristics. The research plan for the construction of the View Creation System is then outlined.

VIEW CREATION SYSTEM CHARACTERISTICS

The function of the View Creation System is to act as a database designer in determining user information requirements for a database. It carries out its task, as a human would, by interactively querying a user about the application for which a database is needed. With the aid of the user, it employs normalization procedures to translate the user's requirements into a set of relations which represents a view. The choice of a relational representation is an arbitrary one. A network, or any other representation, including an Entity-Relationship representation, could have been selected instead with no additional difficulty.

In the terminology of expert systems, the application domain is database design. The domain-specific knowledge of the system pertains to the Entity-Relationship Model and
its use in developing user views. The View Creation System is application-independent since it does not contain any specific knowledge about a particular application.

**Underlying Theory: The Entity-Relationship Model**

The system employs the Entity-Relationship (E-R) approach to view modelling. The E-R model is well known and is an effective database design tool [Jajodia et al., 1983]. Its popularity for high-level database design is due to its economy of concepts and because entities and relationships are natural modelling concepts [Brodie, 1984].

In acting as a database designer, the View Creation System assumes that the user is familiar with his application domain but not with database design terminology or concepts. Hence, it is the system's responsibility to explain to the user any required design concepts. The E-R model is a natural choice from this perspective. It is simple and easily understood by non-specialists, which facilitates user involvement in the database design process [Konsynski, 1979].

The view modelling approach used by the system should also allow the user to express his requirements in a natural manner. The E-R model follows a top-down approach. It starts with entities and relationships which describe the fundamentals of a user's needs. It then obtains the more detailed attributes of these basic constructs. Since a human is capable of aggregating things and is accustomed to thinking in this manner [Baldissera et al., 1979], the model parallels a user's thought process. This enables the user to describe his application to the system in a natural way.

**Knowledge Acquisition**

A crucial task in designing any rule-based expert system is the acquisition of knowledge [Guida and Tass 1983]. Knowledge acquisition is the transfer and transformation of
problem-solving expertise from some knowledge source to a program. Potential sources of knowledge may take the form of human experts, textbooks, databases and one's own experience [Buchanan et al., 1983].

The knowledge incorporated in the View Creation System is obtained from various sources. These sources are listed below along with examples of the types of knowledge they provide. Further examples are found in Chapter V and in the system’s knowledge base (Appendix 1).

1. Database Design Theory
   a. procedures for converting an Entity-Relationship model into a relational one
   b. properties of is-a relationships
   c. alternative ways of obtaining mapping ratios and their function in a design
   d. candidate keys and their use as primary or foreign keys in a database design.

2. Entity-Relationship Model
   a. a set of constructs (entities, relationships and attributes) for modelling a user’s view
   b. a top-down approach to view modelling

3. Normalization Theory
   a. a means of determining whether or not a set of relations avoids certain anomalies
   b. rules for identifying and resolving violations of normalization principles.

4. Expert Database Designers
   a. heuristics (see Appendix 3)
   b. suggestions for improvements to the user interface.

5. Colleagues and other people knowledgeable in database design
   a. suggestions for improvements to the user interface
   b. rules for distinguishing entities and attributes
   c. rules for ascertaining whether an entity is a subset or a superset entity in a
relationship.

6. General Knowledge
   a. attributes names which are often used to identify entities.

7. Experience using and testing the system
   a. rules for identifying missing information
   b. rules for detecting inconsistencies
   c. system default values
   d. rules for allowing the key of one entity to be used in the identification of another
   e. improvements to the user interface.

Knowledge Representation

The formalization of the process of generating user views is embedded in a set of production rules for several reasons.

1. Production rules are the most popular means of knowledge representation in other expert systems and are a natural way for a human expert to explain how he performs tasks requiring his expertise [Barr and Feigenbaum, 1981].

2. The View Creation System application matches quite well Davis and King's [1977] criteria for appropriate domains for systems based on production rules. As outlined by Barr and Feigenbaum [1981], these three criteria are:
   a. domains where knowledge is diffuse as opposed to one in which there exist concise theories
   b. domains where the knowledge can be represented as a set of independent actions
   c. domains where knowledge can be separated from the manner in which it is used (e.g. in medicine) as opposed to one where the representation and control are combined (e.g. a recipe).

   The first two of these criteria apply directly to the View Creation System. There is no concise theory of database design which indicates how user views should be created. Production of user views tends to be done on an ad hoc basis with the quality
of the design produced being highly dependent upon the experience and insight of the
database designer [Bouzeghoub et al., 1985; Goldstein, 1985].

With the Entity-Relationship Model, most of the knowledge can be represented
as a set of independent modules. For example, distinct sets of rules can be defined for
obtaining entity definitions, mapping ratios, relationships, etc.

The View Creation System application does not quite fit the third criterion
listed above. Although the knowledge base is self-contained, a large number of rules
are procedure-oriented. This is due to the nature of the task. Many expert systems
have one major output (e.g. a recommendation of a treatment in a medical system or
an inference about the cause of a mechanical failure in a diagnosis expert system).
The View Creation System, however, is a design system. Such systems may have
more than one "correct" result. The output of the View Creation System includes sets
of entities, relationships and attributes which describe a user's information
requirements. Furthermore, the outputs must be produced in a certain order. It
would be meaningless for the system to ask the user to identify attributes of the major
things of interest in his database before those things (i.e. entities) have been defined as
such. Thus, the system is somewhat procedural. Even though the different modules
of the knowledge base are independent, the order in which they are combined and
executed must be controlled.

3. There was no particular evidence in the literature to strongly suggest a different
approach (e.g. semantic nets, frames) for this type of application.

4. The facilities for this type of approach were readily available.

*Inference Machine*

The View Creation System is developed in Prolog and thus employs predicate calculus for its
symbolic reasoning. This language has been advocated by various researchers as an appropriate one for defining and implementing expert systems [Coelho, 1983; Parsaye, 1983, Hammond, 1982; Clark and McCabe, 1980] It is chosen as the language for the implementation of the View Creation System because the constructs used in Entity-Relationship modelling can be easily represented [Parsaye, 1983]. For example,

```
entity(borrowers)
entity(books)
relationship(borrowers,borrow,books).
```

are three Prolog assertions indicating that "borrowers" and "books" are entities and that a relationship "borrowers borrow books" exists between these two entities.

Another reason for the choice of the Prolog language arises from the manner in which the system is constructed. The development of the View Creation System serves to drive the formalization of the view generation task. Testing different versions of the system aids in highlighting deficiencies in the system's knowledge base, and hence in the formalization. When deficiencies are located, appropriate modifications are made and the effects of the modifications analyzed. Prolog facilitates such a process in two ways. First, it allows the execution and, therefore, the immediate evaluation of partial programs. It also accommodates incremental additions or deletions [Coelho, 1983]. Thus, Prolog supports the development of the knowledge base in an iterative manner.

Production rules are easily represented and modified in a Prolog program [Hammond, 1982]. The system's knowledge base (see Appendix 1) consists of a set of procedural assertions and production rules. Procedural assertions guide the order of the system's execution while production rules indicate what actions can take place in a given set of conditions.
To illustrate the implementation of a production rule, consider a knowledge base rule that states:

IF: a relationship \( A \) is-a \( B \) exists where \( B \) is an entity, but \( A \) is undefined;
THEN: \( A \) is an entity.

This rule could be implemented in Prolog as:

\[ \text{Rule: entity}(A) : - \text{relationship}(A, \text{is-a}, B), \text{entity}(B). \]

If this rule is changed to:

IF: a relationship \( A \) associated-with \( B \) exists where \( B \) is an entity, but \( A \) is not an entity;
THEN: \( A \) is an attribute of \( B \).

only minor modifications are required to produce:

\[ \text{Modified Rule: attribute}(A, B) : - \text{relationship}(A, \text{associated-with}, B), \text{entity}(B). \]

New rules can simply be inserted, in any order, into an existing set of Prolog clauses.

Prolog also allows for quick and simple updating of acquired knowledge [Hammond, 1982]. As will be explained in the Research Plan, knowledge about the view generation task was acquired incrementally from various expert database designers. The augmented knowledge base may have required modifications to existing Prolog clauses or addition of completely new modules of Prolog clauses. Existing clauses are not difficult to modify in Prolog because they are relatively independent of each other. A module is added whenever a new concept is needed (e.g. the concept of a multivalued attribute). The new module is written independently of the existing modules and usually requires only a minimal amount of coupling to other modules. In this manner, Prolog easily accommodates additional knowledge.

Finally, Prolog is a goal-driven language which uses its rules in a backward-chaining manner. These characteristics match the application domain. Each goal consists of a series
of subgoals which can be solved in different ways. For example, the overall goal of the system is to develop a user view for a given application. This is solved by a series of subgoals such as: obtain entities and relationships, determine their representations in the database and resolve functional dependencies. These subgoals in turn require that other subgoals be satisfied before they can be, etc. Furthermore, the different subgoals may be obtained in more than one way so the system has to search through the various possibilities before arriving at a solution to the overall goal.

Similar to the example above, suppose that the system is trying to decide whether \( A \) or \( B \) values should be entities or attributes in the user's database where there are multiple ways that this could be determined. The following illustrates some possible rules for determining the identity of \( A \) and \( B \) as they are used in a relationship.

\[ \text{E.g.} \]

IF: relationship \( A \ has \ B \) where \( A \) and \( B \) are both attributes;
THEN: \( B \) is an attribute of (the attribute) \( A \).

IF: relationship \( A \ have/has \ B \) where \( A \) is an attribute and \( B \) is undefined;
THEN: \( B \) is an attribute of \( A \).

Etc.

Hence, the depth-first, backward-chaining strategy of Prolog is appropriate for this application.

**Explanation of Reasoning**

Many of the system's queries require a "yes" or "no" response from the user. The system also permits the user to ask "why?" before answering. When the system receives a "why" response, it simply pattern matches until it finds the appropriate form of a predicate, i.e. the one which contains the explanation. It provides values for the variables in the predicate based upon the current information in its database and then displays the precoded
explanation to the user.

*E.g.:* Entity: Book [CATALOG-NO, title, author-id, publisher]

*(Dialogue)*

For the entity "book" I have an attribute "author_id". It sounds like "author" should also be an entity in your database. Should I make it an entity?

("yes." / "no." / "why?")

| why?

An attribute that ends in "id" is often used to identify something. If that is the case here, then that something has not been explicitly defined. Should I make "author" an entity? . . .

**RESEARCH PLAN**

The research involved in the development of the expert system is divided into three main phases:

1. Development of a prototype.
2. Expansion of the knowledge base to include the expertise of "real world" database designers.
3. System testing and modification.

Each of these phases is described below.

1. *Prototype Development*

The first phase entailed the development of a prototype system which elicited user information requirements and translated these requirements into a set of normalized relations. The knowledge base for the prototype contained general knowledge about database design (i.e. from standard database textbooks) and knowledge about how concepts from the Entity-Relationship model can be used to describe user views. The purpose of this phase was to determine if the approach taken was a feasible one and to provide a basis for the more sophisticated contributions of later stages.
The prototype development phase showed that the major difficulty with the system was the user interface. The problems associated with the interface seem to be caused by the limitations of Prolog. In particular, Prolog does not have graphics capabilities, nor does it permit menu-driven responses.

2. Expansion of Knowledge Base

The system’s knowledge base was expanded to incorporate the expertise of human database designers. To accomplish this, consultation sessions were arranged with experienced designers from local business organizations. Of the four designers consulted, two were data administrators, one was a database design consultant and the other a database analyst. All have worked extensively in database design.

During these sessions, the human designer was requested to create a user view for a hypothetical problem. He was also asked to describe how and why he made certain decisions in solving the problem. Next, the designer was shown a demonstration of the current version of the View Creation System and asked to provide comments and suggestions for improvements. The rules and heuristics thus identified were transformed into a form suitable for implementation and incorporated into the system’s knowledge base. (Refer to Appendix 3 for documentation of the consultation sessions.)

Some general comments can be made on the database designers’ approaches to view generation. The most obvious characteristic is that the design process is an iterative one. During the consultation sessions, the designers began by obtaining an overall structure of the problem and successively refining it. Each indicated a preference for obtaining a general idea of the problem, completing the details by himself, and finally verifying the design with the user. All of the designers used diagramming techniques to aid in structuring the task as well as in communicating with the user.
The designers used a great deal of their own knowledge about the application domain (even when they said they did not know anything about the application – see Designer#3). For instance, the designers automatically generated attributes such as name, address, etc. for entities which correspond to people. They agreed that these attributes must be verified with the user. They also indicated that, in unfamiliar circumstances, they would be forced to rely on the user to provide all of the relevant attributes.

Each consultation session with a database design expert resulted in successive refinement of the knowledge base. This explains why the process was an iterative one.

3. System Testing and Modification

The final phase of the research concentrated on formally testing the system. The major reason for testing the system is to prove that the knowledge base is an adequate formalization of the view specification task. The general approach to testing the accuracy and completeness of the formalization required that the system’s output be examined and an evaluation made of the user view produced. This testing was done using real cases from participating corporations. Some gaps in the knowledge base were identified during these testing sessions and were filled prior to the next testing session. Thus, the system was not static across the entire set of tests.

The criteria used for testing the system and the results obtained are discussed in Chapter VI.
V. VIEW CREATION SYSTEM

This section describes, in detail, the View Creation System. The real content of the research is in the formalization of a process for generating user views. The expert system simply provides a means of expressing, in its knowledge base, this formalization. Although the knowledge base is documented in Appendix 1, it is presented more effectively by discussing the implementation. A complete transcript of a design session is included in Appendix 2.

The objective of the system is to emulate a database designer in the creation of a user’s view by interactively querying a user about an application. While doing so, the system tries to detect and resolve ambiguities and inconsistencies in the user’s input, identify places where there is redundant or missing information, and provide appropriate default values when the user is unable to respond to a request for input. The remainder of this section demonstrates how the system carries out these tasks.

PROCEDURE

The View Creation System’s procedure is summarized in Table 2.

The design session begins with the system identifying the user and instructing him how to respond. The user is then asked to provide an initial set of entities with their corresponding attributes and candidate keys. Relationships of the form $A \text{ verb phrase } B$, where $A$ and $B$ are entities, are elicited together with their corresponding attributes and mapping ratios. By this stage, the user has provided a preliminary version of his information requirements. The user is permitted to make modifications and the system performs validity checks to ensure that its internal database (representing the user’s requirements) remains consistent.
### TABLE II: View Creation System’s Procedure

<table>
<thead>
<tr>
<th>STEPS</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Session</td>
<td>• The System obtains the user’s id and describes the types of responses required.</td>
</tr>
<tr>
<td>Entities, Attributes, Candidate Keys</td>
<td>• The user provides an initial set of entities, attributes and candidate keys (combinations of attributes which uniquely identify entities). (Knowledge Source: E-R Model)</td>
</tr>
<tr>
<td>Missing Entities</td>
<td>• If an attribute is of the form <code>Term_id</code>, <code>Term_no</code>, or <code>Term_code</code>, etc., the system suggests that <code>Term</code> should be an entity. (Knowledge Source: Heuristic based on experience with the system.)</td>
</tr>
<tr>
<td>Relationships, Relationship Attributes, Mapping Ratios</td>
<td>• Relationships are restricted to the form A <em>verb phrase</em> B where A and B are entities. Mapping ratios indicate the minimum and maximum number of A values which can occur for each value of B and vice versa. Relationships may be represented by relations or by adding the primary key attributes of one entity as non-key attributes of another. The choice of representation depends on the mapping ratios and the types of queries that are of interest to the user. Assertions are added to the system’s database which indicate how each relationship will be represented after primary keys are chosen. (Knowledge Sources: E-R Model; database design principles; expert database designers.)</td>
</tr>
<tr>
<td>Special Kinds of Verb Phrases</td>
<td>• <em>Is-a</em> and <em>instance-of</em> verb phrases are important for inheriting attributes, determining primary keys and detecting inconsistencies. (Knowledge Sources: Data models; experience with the system.)</td>
</tr>
<tr>
<td>Have/Has Relationships</td>
<td>• In a relationship A <em>have/has</em> B, the <em>have/has</em> is ambiguous. The user is asked if: 1) A possesses B; 2) B component of A; 3) B instance/example of A; or 4) B associated with A and the relationship is modified accordingly. (Knowledge Source: Experience with the system.)</td>
</tr>
<tr>
<td>Modifications – First</td>
<td>• The user is allowed to modify his initial set of information requirements. (Knowledge Source: Experience with the system.)</td>
</tr>
<tr>
<td>Synonyms</td>
<td>• The system tries to detect entities and verb phrases which are synonyms and relationships which represent the same information in different ways. (Knowledge Source: Experience with the system.)</td>
</tr>
<tr>
<td>Inherited Attributes</td>
<td>• For relationships of the form A <em>is-a-instance-of</em> B, the user is asked whether it would be incorrect to attribute to A all of the attributes of B. If it would be incorrect to do so, an inconsistency exists which must be resolved. (Knowledge Sources: Data models; experience with the system.)</td>
</tr>
<tr>
<td>Modifications – Last</td>
<td>• The user is given a final opportunity to make modifications.</td>
</tr>
</tbody>
</table>
TABLE II (Cont’d): View Creation System’s Procedure

- **Primary Keys**
  - The system automatically determines primary keys from candidate keys. For relationships of the form A is-a B, the primary key of B is added to the set of candidate keys of A. For relationships of the form A instance-of B, the primary key of B is concatenated to each of the candidate keys of A before the primary key for A is chosen. (Knowledge Sources: Database design principles; experience with the system.)

- **Represent Entities & Relationships**
  - An entity is represented by constructing a relation whose key and non-key attributes correspond to those of the entity. Each relationship is represented by: i) constructing a relation whose key is the concatenation of the primary keys of the relationship’s entities or ii) by adding the primary key attributes of one entity as non-key attributes of the other. (Knowledge Source: Database design principles.)

- **Functional Dependencies**
  - With the aid of the user, partial and transitive dependencies are identified and resolved. (Knowledge Sources: Database design principles; expert database designers.)

- **Output**
  - A set of Fourth Normal Form relations representing a user view is produced.

The system attempts to identify entities and verb phrases which are synonyms and relationships which represent the same information in different ways. "Inherited" attributes are identified and the system asks the user to indicate whether or not they are valid. The user is then allowed to make final modifications to his set of information requirements.

Primary keys are automatically determined by the system based on its internal rules. With the aid of the user, the system tries to identify and resolve functional dependencies before producing, as its final output, a set of Fourth Normal Form relations representing the user’s view. Throughout the dialogue the user may request a display of the current state of the design.

Each of these steps will now be discussed in more detail.
INITIALIZATION

The system assumes that the user is familiar with his application domain but not necessarily with database design concepts and terminology. Accordingly, it describes the types of responses required.

After requesting the user’s identity, the system explains the syntax of acceptable user responses.

E.g.
User? Enter your name followed by a dot (.) and return.
I have to ask you to always respond in this manner –
i.e. give your answer followed by a dot and then return.
Now enter your name please.
|: Veda.
Hello veda. For what do you wish to design a database?
(If you need more than one word to respond, connect the words with underscore characters.)
|: library_circulation.
.
.
.

ENTITIES

The concepts of entities and attributes are explained (see transcript – Appendix 2) and the user asked to identify each entity of interest along with its corresponding attributes. Entity names are requested in their singular form because the form of an entity (singular or plural) as it appears in a relationship will later be examined to infer mapping ratios. (See Mapping Ratios.)

Attributes

During the consultation sessions with database designers, the human experts usually
obtained an initial set of entities and possibly their keys. Detailed attributes were not dealt with until a later stage. The View Creation System differs in its approach by obtaining attributes as soon as an entity is identified. There are several reasons for proceeding in this way. First, requesting an entity's attributes whenever an entity is given, forces the user to think carefully about the entity. Second, it may be difficult for the user to distinguish between entities and attributes so the system tries to aid the user in making the distinction. Finally, proceeding in this manner enables the system to infer something about the types of relationships which will be needed at a later point. Examples of distinguishing between entities and attributes and the types of inferences the system makes are given below.

1. **Attribute Name is Entity Name**

   If the user provides an attribute name for one entity which is the name of another entity, then it would appear that the two entities must somehow be related (otherwise, the user should not have mentioned the attribute). In particular, it may be that one entity is needed to uniquely identify the other as in the case of a volume of a book requiring the book's identification before it can be unambiguously identified in a database.

   Therefore, if the user tries to add, as an attribute name for a new entity $E_{new}$, the name of a previous entity $E_{prev}$ (or $E_{prev}$ id, $E_{prev}$ name, etc.), the system first asks if the existing entity is needed to uniquely identify an instance of the new entity. If the user indicates that it is, the system "records" this fact by adding an assertion to its internal database which indicates that the existing entity name $E_{prev}$ should appear in at least one of the candidate keys of the new entity. (See Candidate Keys.) This assertion will be used at a later point to: 1) verify that the key of the previous entity is needed to identify the new entity or 2) to prompt the user for a relationship between the two entities. (See Relationships - Prompts for Missing Relationships and Primary Keys).
If the previous entity is not needed to uniquely identify an instance of the new entity, the system does not allow the previous entity name to be an attribute of the new entity. Instead, it informs the user that he is implying the existence of a relationship and adds an assertion which indicates that a relationship will be needed between the two entities. This assertion is later used to prompt the user for the relationship if it is not voluntarily provided. (See Relationships – Prompts for Missing Relationships.)

E.g.: \( E_{\text{New}} = \text{branch} \); \( E_{\text{Prev}} = \text{library} \)

Library: [id, name, address, director]

(Dialogue)
Enter next entity / "see." / "end."
|: branch
What are the attributes for "branch"?
|: name address library.
I already know that "library" is an entity. Does this mean that you will need to know "library" before you will be able to UNIQUELY identify an instance of "branch"?
("yes." / "no."")

When the user's response is affirmative:

|: yes.
Fine.
(The system adds an assertion which indicates that "library" may be needed to uniquely identify an instance of "branch").

When the user's response is negative:

|: no.
Then you are really telling me that a relationship exists between "library" and "branch". We'll get to that in a minute.
(The system adds an assertion which indicates that a relationship should exist between library and branch.)

2. Entity Name was Attribute Name

Analogous to the above, suppose the user adds, as the name of a new entity \( E_{\text{New}} \), (some form of) the name of an attribute \( A_{\text{Prev},j} \) of an existing entity \( E_{\text{Prev}} \). The system adds an
assertion which indicates that the new entity may be needed to uniquely identify an instance of the previous entity or that a relationship should exist between the two. This assertion will later be used to: 1) replace the attribute $A_{Prev,j}$ in the primary key of $E_{Prev}$ with the primary key of $E_{New}$ (after Primary Keys have been obtained) or 2) prompt the user for a relationship between $E_{New}$ and $E_{Prev}$. (See Relationships - Prompts for Missing Relationships and Primary Keys.)

$$E_{Prev} = \text{branch}; \ A_{Prev,j} = \text{library\_name}; \ E_{New} = \text{library}$$

Branch: [name, address, library\_name]

(Dialogue)
Enter next entity / "see." / "end."
|: library.
What are the attributes for "library"?
|: ...

(The system adds an assertion which indicates that "library" may be needed to uniquely identify "branch".)

3. Repeating Attributes

The system also tries to detect multiple attributes that seem to be referring to the same type of thing. If the user enters, as attributes of an entity $E_i$, two or more terms of the form $\text{Term\_1, Term\_2, etc.}$, the system infers that the user is referring to multiple instances of the same thing. Therefore, that thing should probably be an entity. When this happens, the system asks if $\text{Term}\_i$ should be an entity. If it becomes an entity, then a connection between $\text{Term}\_i$ and $E_i$ must exist (otherwise the user would not have originally provided $\text{Term\_1, Term\_2, etc. as attributes of } E_i$). To ensure that a connection between these entities will exist, the system adds an assertion to its database which indicates that a relationship between $\text{Term}\_i$ and $E_i$ will be required. If necessary, it will use this assertion to prompt the user for the relationship at a later point. (See Relationships - Prompts for Missing Relationships.)
E.g.
Enter the name of an entity.
|: borrower.
What are the attributes for "borrower"?
Are "book1" "book2" "book3" different occurrences of "book"?
("yes." / "no." / "why.")
|: yes.
Then we will make "book" an entity when we have finished with this one.
(System adds an assertion which indicates that a relationship should exist between borrower and book.)

4. Attributes with Multiple Values

One of the requirements of First Normal Form (see Discussion of System’s Correctness) is that each attribute in a relation be single-valued. Since each entity is eventually represented as a relation (see Represent Entities and Relationships), the system asks the user to identify any attributes which could have more than one value for an occurrence of a given entity. An attribute which meets this criterion is deleted from the original entity and converted to a new entity. An assertion is added to the system’s database which indicates that a relationship should exist between the original and the new entity.

E.g.: Book: [catalog_no, author, title, publisher]

(Dialogue)
Here are the attributes for:
book: [catalog_no, author, title, publisher]
Can a single "book" have more than one value for any of these attributes?
("no." / "why?" / attribute(s) which can have more than one value / "end." if you do not know.)
|: why?

If an attribute can have more than one value, then it is probably referring to a "thing" so it should really be an entity. Now consider book: [catalog_no, author, title, publisher]
Can any of these attributes have more than one value for each occurrence of "book"?
I will delete the attribute "author" and we will make it an entity later.

5. System Generated Attributes

If the user is not able to provide any attributes for an entity, the system first verifies that the entity should exist. Then, since an entity must be uniquely identified in a database, it generates as a default an attribute \textit{entity-name\_id}.

\textit{E.g.}

Enter next entity / "see." / "end."
|: library\_director.
Enter the attributes for "library\_director" / "end." if you do not know.
|: end.
Then should the entity "library\_director" exist?
|: yes.
Then I shall make an attribute "library\_director\_id".

\textit{Candidate Keys}

Once the system obtains the attributes for an entity, it asks the user to identify all the attributes or groups of attributes which uniquely identify an instance of the entity and will thus serve as candidate keys.

During the consultation sessions with database designers, some of the designers indicated that they prefer to use \textit{artificial}, system-generated keys (e.g. numeric codes) instead of \textit{normal} keys (consisting of attributes). (See Appendix 2 - Designer\#1 and Designer\#3.) The View Creation System is based on the assumption that the ultimate
database will use conventional keys which consist of single attributes or combinations of attributes. However, it also generates and uses internally artificial keys like those suggested by the designers – particularly when it has to refer to the key of a relation before the precise specification of the key has been completed. Thus, it would be a simple matter to make these artificial keys an explicit part of the database instead of trying to determine conventional keys for each relation. This approach was not adopted because it is not known how widely this strategy is followed and because the View Creation System should be capable of determining conventional keys for organizations that prefer them.

**Key Attributes**

The system contains, as general knowledge, a list of attributes which may be used as part, or all, of a candidate key. These "key indicator attributes" are *id, name, number*, etc. If the user’s initial set of attributes for an entity contains any of the ones on the list, the system suggests that the user consider them when providing candidate keys.

*E.g.*

Here are the attributes for borrower: [number, name, address]
Enter an attribute or combination of attributes which could be used to UNIQUELY identify an instance of the entity "borrower".
In particular, you may wish to consider:

1) number
2) name.

|: number.
Is there any other way that an instance of the entity "borrower" could be identified?
("see.", to see how I can identify it so far / "no." / "yes."")
|: . . .

Consistency checks are also performed on the candidate keys. If the user enters a set of attributes as a candidate key which is a subset of an existing candidate key, the system recognizes that the user is being inconsistent. It asks the user which set should be the candidate key. If the user does not know, it selects the larger set.
Example (from above)

Is there any other way that an instance of the entity "borrower" could be identified?
("see.", to see how I can identify it so far / "no." / "yes.")

|: name address
Is there any other way that an instance of the entity "borrower" could be identified?

|: name.
You already told me that the following are needed to UNIQUELY identify an instance of "borrower".
a) borrower: [NAME, ADDRESS]
Now it seems that I need only:
b) borrower: [NAME]
Which one should I use?
("a." / "b." / "no." if you do not know.)
|: a.
Is there any other way that an instance of the entity "borrower" could be identified?
|: no.

In the above example, the entity borrower has two candidate keys: [NUMBER] and [NAME, ADDRESS]. Hence, there are two alternative representations for borrower:

borrower: [NUMBER, name, address]
borrower: [NAME, ADDRESS, number]

Eventually, the system will select a primary key from the candidate keys in order to obtain the representation for borrower which will be used in the database. (See Primary Keys.)

One way the system tries to build up its knowledge of the application domain is by examining the attributes which the user provides as candidate keys. If an attribute is used
as part of a candidate key for at least two different entities, the system recognizes that this attribute serves as an identifier in the application domain. Therefore, it adds the attribute to its list of "key indicator attributes".

_E.g._ (Candidate keys given in UPPER CASE)
library: [LIBRARY_NAME, ADDRESS, . . . ]
borrower: [NUMBER, name, address]
borrower: [NAME, ADDRESS, number].

Since _address_ appears in the candidate key for library and in one of the candidate keys for borrower, it is added to the list of key indicator attributes.

**Modifications to the Initial Set of Entities**

Once the initial set of entities, attributes and candidate keys is obtained, the attributes are scanned. Any attribute names which occur for at least two distinct entities are prefixed by their entity names in an attempt to keep all attribute names unique. Attributes which are key indicator attributes are also modified in this way. The set of entities with their attributes and candidate keys is then displayed and the user given an opportunity to make modifications.

_E.g._
Here are the entities with their attributes and keys.
Note that I may have made a few modifications to the attribute names.
book: [CATALOG_NO, title, author_id, publisher]
book: [TITLE, AUTHOR_ID, catalog_no, publisher]
borrower: [BORROWER_NUMBER, borrower_name, address]
borrower: [BORROWER_NAME, ADDRESS, borrower_number]
volume: [VOLUME_NUMBER, date_due].

Do you wish to add, delete or change any of the entities?
("no." / "add." / "delete." / "change.")
|: . . .

The user is allowed to add, delete or rename complete entities. He may do the same for attributes of entities. If the user deletes an attribute which is part of a candidate key, the system will ask the user to identify candidate keys for the entity again.

*E.g.*: book: [CATALOG_NO, title, author_id, publisher]
    
    book: [TITLE, AUTHOR_ID, catalog_no, publisher]

*(Dialogue)*

Do you wish to add, delete or change any of the attributes?

|: delete.

Enter the attribute(s) you wish to delete / "end."

|: author_id.

Earlier you said that "author_id" was (part of) an identifier for "book" so we will now have to go back and reconsider the possible identifiers for "book".

**MISSING ENTITIES**

As previously mentioned, it may sometimes be difficult for the user to distinguish between entities and attributes. The system tries to detect attributes which should be entities and vice versa. Some examples of the system detecting such inconsistencies have already been discussed. *(E.g. if the user enters attributes of the form Term_1, Term_2, etc., the system will suggest that the user really has in mind multiple occurrences of an entity Term. See Entities.)*

Another way that the system tries to detect an attribute which should be an entity occurs when the user provides an attribute of the form Term_suffix where suffix is one of the "key indicator attributes". Since suffix is often used as an identifier for an entity, the system asks if Term should be an entity. If the user agrees that it should be, the system proceeds to query the user for corresponding attributes and candidate keys.
E.g.: Term_suffix = author_id
   book: [CATALOG_NO, title, author_id, publisher, author_bookno]
   volume: [COPY_NO, date_due, author_name]

(Dialogue)
For the entity "book" I have an attribute "author_id".
It sounds like "author" should also be an entity in your database.
Should I make it an entity?
("yes." / "no." / "why?"
|: why?
An attribute that ends in "id" is often used to identify something. If that
is the case here, then that something has not been explicitly defined.
Should I make "author" an entity?
("yes." / "no."
|: yes.
Enter the attributes for "author"?
(Else "end." if you do not know.)
|: id.

The system adds two assertions to its database which indicate the following:

1. A relationship will be required between the original entity and the new one, Term. Having Term_suffix as an attribute of an entity implies that Term and the entity must somehow be related; otherwise, the user should not have provided Term_suffix as an attribute of the entity. In the above example, a relationship will be required between book and author.

2. The attribute Term_suffix should later be deleted. If the relationship mentioned above is obtained, it will provide a connection between the two entities, so the attribute Term_suffix will no longer be needed. E.g. if a relationship exists between author and book then the attribute author_id can be deleted from book. Note that the ultimate representation of the relationship may involve inserting Term_suffix back into the relation for the original entity. However, this should not be assumed at this stage.

Next, the system examines other attributes of the form Term_suffix. If an attribute Term_suffix appears as part of the definition of any entity, but does not appear as an attribute of the new entity Term, the system asks the user if this attribute should be included.
E.g. (from above)

book: [CATALOG_NO, title, author_id, publisher, author_bookno]
volume: [COPY_NO, date_due, author_name]
author: [author_id]

(Dialogue)
Would you like to include the attribute "author_bookno"?
("yes."/"no."/"why?")
|: no.
Would you like to include the attribute "author_name"?
|: why?
I know "author_name" is an attribute of at least one other entity, e.g. "volume". Since it starts with "author", I thought that it could be an attribute of "author".
Do you wish to add "author_name" as an attribute of "author"?
("yes."/"no.")
|: yes.

For each attribute which is included, the system adds more assertions which indicate the following:

1. A relationship may be required between the entities where such attributes originally appeared and the new entity Term. E.g. a relationship may be required between volume and author.
2. The original attribute should eventually be deleted.

Both of these are conditional assertions and will depend on the existence of relationships among the original entities and Term. In the above example, the assertions will depend on the existence of relationships such as author writes books and books have volumes. (See Prompts for Relationships).

Finally, the system considers attributes of the form Term_suffix which occur in both Term and some original entity. As above, assertions are added which indicate that a relationship may may be needed between the original entity and Term and that the attribute Term_suffix should later be deleted from its original entity. There are no such attributes in the example above. Only one attribute (author_id) was initially provided by the user for the
entity author. If an attribute author_id existed for another entity (besides book), then the assertions would be added.

The objective of these assertions is to ensure that a connection will exist between the new entity and each original entity which has at least one attribute in common with the new entity. Four possible situations could arise after relationships have been obtained:

1. A relationship will exist between an original entity and Term. Then the attribute Term_suffix will be deleted because it will no longer be needed to represent the relationship. E.g. if a relationship author writes book exists, then the attribute author_id can (and should) be deleted from the entity book.

2. The connection between an original entity and Term will exist via a third entity. Then no direct relationship will be required between the two. The attribute Term_suffix will be deleted because it will no longer be needed to represent the relationship. E.g. if the relationships author writes books and books have volumes exist, then no direct relationship is required between volume and author.

3. No relationship will exist between an original entity and Term so one will be requested. If a relationship is provided, Term_suffix will be deleted from the original entity because it will no longer be needed to provide the connection between the two. E.g. if the relationship author writes books exists but there is no relationship connecting either volume and book or volume and author, the user will be asked for a relationship between volume and author. If one is provided, the attribute author_name will be deleted from the entity volume.

4. No relationship will exist between an original entity and Term (and, when asked, the user could not provide one). The user will then be asked if Term_suffix should really be an attribute of the original entity. E.g. suppose there is no relationship, either directly or indirectly between volume and author and, when asked, the user could not provide one. The user will be asked if author_name should really be an attribute of volume.

To summarize, in the above example, the system adds assertions to its database which indicate the following:

1. A relationship will be required between book and author.
2. The attribute author_id of the entity book should later be deleted.
3. A relationship may be required between volume and author.
4. The attribute author_name of the entity volume should later be deleted.
As will be seen in *Relationships – Prompts for Relationships*, ideally, the following should take place:

1. A relationship "author writes books" will eventually be obtained between author and book. "Author_id" will no longer be needed to indicate that a connection exists between the two entities, so it will be deleted.

2. A relationship "books have volumes" will be identified. Then the connection between volume and author will exist via the entity book (since books have volumes and a relationship exists between book and author). The attribute "author_name" will no longer be required as a connection between volume and author, so it will be deleted from the entity volume.

**RELATIONSHIPS**

Following Baldissera et al. [1979], relationships are restricted to the form \( A \; VP \; B \) where \( A \) and \( B \) are names of entities and \( VP \) stands for *verb phrase*. This relatively simple input form is taken from the literature on binary data models [Ceri et al., 1980; Baldissera et al., 1979; Bracchi et al., 1976]. Examples of input accepted by the system are:

- borrower borrows volumes
- library has branches
- volumes loaned to borrower.

The user is instructed to provide relationships using entity names which may be in either singular or plural form. As each relationship is entered, the system performs the following tasks.

1. It ascertains whether the \( A \) and \( B \) values are known to be entities. If either (or both) of the \( A \) or \( B \) values are unknown, they must be classified as entities or attributes. (See *Unidentified A's and B's.*)

2. If both \( A \) and \( B \) are entities, the system obtains the relationship’s mapping ratios and, where appropriate, the attributes of the relationship. (See *Mapping Ratios* and *Relationship Attributes.*)
Special Kinds of Verb Phrases

There are three main kinds of special verb phrases - i) is-a, ii) instance-of and iii) have/has. Is-a and instance-of are important in determining primary keys, "inheriting" attributes and detecting inconsistencies in the user's input as will be explained in later sections. Have/has verb phrases are classified into four different types, one of which is instance-of.

Is-a Verb Phrases

The is-a verb phrase corresponds directly to Smith and Smith's [1977] concept of generalization. A relationship A is-a B implies that one should be able to attribute to A all of the properties of B (but not vice versa). Here, there is at most one occurrence of A for each occurrence of B.

E.g.
"library-director is-a person"
A person may or may not be a library-director but all the attributes of person should be attributable to library-director.

Instance-of Verb Phrases

The instance-of verb phrase is similar to the is-a verb phrase in the sense that, for A instance-of B, one should be able to attribute properties of B to A (but not vice versa). However, the instance-of verb phrase differs from the is-a verb phrase because it allows for many occurrences of A for each occurrence of B.

E.g.
"volume instance-of book"
There can be one to many volumes for each book and all the attributes of book should be attributable to volume.
Unidentified A's and B's

The system requests relationships among entities. However, when the user provides a relationship \( A \ VP \ B \), an \( A \) or \( B \) value (or both) may not have been previously identified as an entity. The system's knowledge base contains a set of rules for dealing with such unidentified A's and B's. These rules were constructed by considering:

1. the possible values for \( B \) (entity, attribute, or unknown (due to a user's error)) when the value of \( A \) is known and vice versa;
2. the meaning of the verb phrase which appears in a relationship;
3. the information which already exists in the system's database about the \( A \) and \( B \) values.

Examples of how these rules are used in the system are given below.

First, consider a relationship \( A \ is-a B \). Assume that \( B \) is an entity but \( A \) is unidentified. Since \( B \) is a generalization of \( A \), \( A \) must also be an entity. Therefore, the system makes \( A \) a new entity and proceeds to query the user for corresponding attributes and candidate keys.

\[ \text{E.g.: Relationship "manager is-a employee"} \]
\[ \text{If "employee" is an entity but "manager" is undefined, then "manager"} \]
\[ \text{should also be an entity.} \]

There are rules for dealing with relationships where the verb phrase is \( have \) or \( has \) and there is some known information about \( A \) and \( B \). If, for example, \( A \ has B \), where \( A \) is an entity and \( B \) an attribute of \( A \), then there are two possibilities for \( B \). Either: 1) \( B \) should really be a separate entity and not an attribute of \( A \) or 2) \( B \) should remain an attribute of \( A \) in which case the relationship is redundant and can therefore be deleted.

\[ \text{E.g.: Relationship "book has author"} \]
\[ \text{book: [CATALOG_NO, author, ...]} \]
(Dialogue)
Should "author" be an entity?
("yes." / "no.")

If "yes", author will be deleted as an attribute of book and made a new entity.
(The key attributes of author may later be added as non-key attributes of book in order to represent a relationship between the two entities.)

If "no", the system informs the user that the relationship is redundant.
Fine. I no longer need the relationship so I will delete it.

Note that, in the above example, the system is: 1) giving the user an opportunity to correct a previous error (if author should be a separate entity instead of an attribute of book); or 2) detecting redundant information (if author remains an attribute of book, the relationship is redundant).

As another example, assume a relationship A have/has B is provided where A and B are both attributes. Here, the user is implying that B is an attribute of another attribute A. (It would be nonsensical to assume that B is an entity because one might say that an attribute is associated with an entity, not that an attribute has an entity.) Therefore, the relationship is deleted and an attribute A_B added which indicates that B is an attribute of A.

E.g.: Relationship "colour has intensity"
If an entity has "colour" as one of its attributes, then an attribute "colour_intensity" will be added.

Finally, the system uses this rule set to allow the user to provide missing information and to point out possible input errors. For example, if a relationship is of the form A VP B where VP is neither is-a nor have/has and B is an entity but A is undefined, the system asks if A should also be an entity. If A becomes an entity, corresponding attributes and candidate keys are obtained. Otherwise, the system suggests that the user made a mistake.
E.g.: Relationship "publisher publishes books"
("book" is an entity but "publisher" is unidentified)

(Dialogue)
Is "publisher" a new entity?
You haven’t mentioned it before.
("yes." / "no."

When the response is "yes", the system obtained corresponding attributes and candidate keys.

When the response is "no":
Then I think you made a mistake so I will delete the relationship.

Further details and examples are contained in the Knowledge Base – Appendix#1.

Prompts for Missing Relationships

The system tries to ensure that the view it produces completely represents the user’s information requirements. One of the ways that it does so is by verifying that the user has provided all of the relationships that it "knows" (i.e. that it can infer or has previously inferred) should exist. Therefore, after the initial set of relationships has been obtained, the system attempts to identify places where relationship information is missing. First, it searches for entities which do not appear in any of the existing relationships and asks the user to provide relationships involving these entities. If no relationships are provided, the entities are classified as "solo" entities.

Next, the system scans the assertions in its database which indicate that relationships are required between certain pairs of entities. For each assertion, it determines whether or not a relationship exists between the two involved entities. If one does exist, the assertion is deleted; if there is no such relationship, the system prompts the user to provide one. These assertions are based on inferences the system made when
acquiring the user’s initial set of entities and could have been obtained in five different ways:

1. The user originally entered, as the name of an attribute, the name of an existing entity. (See Entities).
2. The user originally entered, as the name of a new entity, the name of an attribute of another entity. (See Entities.)
3. The user tried to enter multiple occurrences of the same attribute when the attribute should have been an entity. (See Entities.)
4. The initial set of attributes contained attributes which had multiple values and were therefore converted to entities. (See Entities.)
5. The form of an attribute of one entity suggested that another entity was missing. (See Missing Entities.)

The following examples illustrate how the system deals with the first and last of these assertions. Further details can be found in the Knowledge Base – Appendix 1.

1. **Attribute Name was Entity Name**

The first type of prompt is a result of the user originally trying to add, as an attribute name for a new entity $E_{new}$, (some form of) the name of another entity $E_{prev}$. At that point, the system would have asked if $E_{prev}$ is needed to uniquely identify an instance of $E_{new}$. If the user indicated that $E_{prev}$ is needed to identify $E_{new}$, the system now determines if $E_{prev}$ occurs in at least one candidate key of $E_{new}$. If it does not appear, the system searches for a relationship between the two entities. If no relationship exists, it prompts the user for one. Alternatively, if the user indicated that $E_{prev}$ is not needed to identify $E_{new}$, and if there is no existing relationship between the two entities, the system prompts the user for one.

To illustrate, refer to the example in Entities – **Attribute Name was Entity Name** where the user tried to add the attribute library to the entity branch when library had been previously identified as an entity.
Case i)

E.g.
Library: [LIBRARY_ID, library_name, library_address, director]
Branch: [BRANCH_NAME, BRANCH_ADDRESS, library]
Branch: [BRANCH_NAME, LIBRARY, branch_address]

Assume the user indicated that library is needed to uniquely identify an instance of the entity branch. Then, since library appears in at least one of the candidate keys of branch, the system will continue to assume that this is the case. (However, if after obtaining primary keys, library does not appear as part of the key of branch, the system will prompt the user at that point for a relationship between library and branch. See Primary Keys.)

If library did not appear in any of the candidate keys of branch the system would proceed to prompt the user for a relationship as outlined in Case ii below.

Case ii)

Suppose the user originally indicated that library is not needed to uniquely identify an instance of branch. Then the current set of entities could be (library was not included as an attribute of branch):

E.g.
Library: [LIBRARY_ID, library_name, library_address, director]
Branch: [BRANCH_NAME, branch_address]

In this case, the system would have an assertion which indicates that a relationship should exist between library and branch (because the user originally tried to give library as an attribute of branch, thus implying that the two are related). The user is then prompted for a relationship as follows:
E.g.
Is there a relationship between "library" and "branch"?
("yes." / "no." / "why?")
:: why?

When you gave me the entity "branch" you said that "library" was one of its attributes. That tells me that "branch" and "library" could be related. Is there a relationship between "branch" and "library"?
("yes." / "no.")
:: yes.
Enter relationship(s). Enter "end." when you finish.
:: library has branches.

2. Missing Entities

The last type of prompt occurs when the user originally provided an attribute of the form Term_suffix where suffix is one of the "suggested key attributes". The system suggested that Term should really be an entity and it was converted to one.

Consider the example from Missing Entities which is outlined below.

E.g.: Book: [CATALOG_NO, title, author_id, author_bookno]
       Volume: [VOLUME_NO, date_due, author_name]

(Dialogue)
It sounds like "author" should be an entity in your database. Should I make it an entity?
("yes." / "no." / "why.")
:: yes.
What are the attributes for "author"?
id.
Would you like to include the attribute "author_bookno"?
:: no.
Would you like to include the attribute "author_name"?
:: yes...

The system added assertions to its database which indicated that:
1. a relationship is required between book and author.
2. the attributes author_id should eventually be deleted from book.
3. a relationship may be required between volume and author.
4. the attribute author_name of the entity volume should later be deleted.

If, at this point, there is no relationship between "book" and "author", the system prompts the user to provide one.

Do you wish to provide a relationship(s) between "book" and "author"?
("yes." / "no." / "why.")

|: why?
You have an attribute of "book" which contains some form of the term "author". Since "author" is also an entity, this suggests that there could be at least one relationship between "book" and "author".
Do you wish to provide a relationship between "book" and "author"?

|: author writes books.

I'm going to consider how to represent the relationship(s) later so for now I will delete the attribute "author_id" from "book".

(If the user was unable to provide a relationship between "book" and "author", the system would ask if "author_id" should really be an attribute of "book".)

To deal with the assertions involving the entity volume, the system first searches for either a direct or an indirect relationship between volume and author. If one is found, it deletes the attribute author_name from the entity volume (because the relationship provides the connection between the two entities). If there is no relationship, the system prompts the user to provide one and proceeds as above.
Ideally, there will be two relationships *author writes books* and *books have volumes*. Then the attributes *author_id* and *author_name* can be deleted from the entities *book* and *volume*, respectively. If the relationships are not provided, the user will be prompted for them.

**MAPPING RATIOS**

The View Creation System's output is a set of relations which represents a user’s view. There are two basic kinds of relations – entity relations and relations which represent relationships. Using the Entity-Relationship Model, one approach to representing relationships is simply to convert *all* relationships to relations. An alternative is to represent certain relationships by adding key attributes to existing (entity) relations and to construct new relations for the others [Goldstein, 1985]. In the system, the latter approach is adopted with the mapping ratios ensuring that the representations are valid. This approach is taken because, where there is a choice, it is more efficient to represent a relationship by adding the key attributes of one (entity) relation as non-key attributes of another instead of constructing a new relation. (For a discussion of the extra costs involved in representing relationships as separate relations, see Wilmot [1984]).

A key of one relation which appears in another relation is known as a *foreign key*. Wilmot [1984] argues that foreign keys decrease the adaptability of database designs and that a relationship should always be represented by constructing a new relation. Some of his arguments are based on the belief that the requirements of the database are not known in advance and, thus, there is a need for a great deal of flexibility in database design.

In the View Creation System, it is assumed that the user does know how the information will be used (i.e. what types of queries will be of interest). If the user is uncertain, the system, as a default, represents a relationship by a relation. (See examples
which follow.) Furthermore, many human designers also chose, for efficiency reasons, to use the method implemented in the system. Finally, representing each relationship by a separate relation is simply one of the options considered by the system. If necessary, the system could easily be instructed to use this one exclusively for any organization which prefers maximum flexibility over efficiency considerations.

Hence, each relationship $A \text{ VC } B$, has three possible representations in the View Creation System:

1. A relation can be constructed between $A$ and $B$ with relation name $A \text{ VC } B$ and relation key the concatenation of the keys of the $A$ and $B$ entities.
2. The key attributes of $B$ can be added as non-key attributes of $A$.
3. The key attributes of $A$ can be added as non-key attributes of $B$.

The choice of representation depends on the minimum and maximum number of $A$ values that can occur for each value of $B$ (and vice versa) and (perhaps) the type of query that is of interest to the user.

At this point in the procedure, the primary keys for each of the entities have not been determined. Therefore, although the rules for representing relationships dictate that one of the above representations is appropriate, the actual representation cannot be added to the system's database until the primary keys have been chosen from the candidate keys. To cope with this situation, the system adds assertions to its database which indicate how the relationships are to be represented. These assertions actually represent a set of uninstantiated keys which will be instantiated at a later point (i.e. after the primary keys are obtained – see Represent Entities and Relationships). Thus, in this section, references to adding key attributes of one entity to another entity or creating relations between two entities are to be interpreted as adding assertions which reflect the appropriate mode of representation.
Min/Max Values

Mapping ratios are based on both the minimum and maximum number of A values which can occur for each B value in a relationship A VC B and vice versa. Tsichritzis and Lockovsky, [1982] refer to this type of mapping ratio as the minimum and maximum cardinality of the mapping. For example, if each value of A can have from 0 to many corresponding values of B, then the min/max values for A are (0,N). Similarly, if each value of B has one and only one corresponding value of A, then the min/max values for B are (1,1). The system first tries to infer the min/max values for the entities and, when it is unable to do so, queries the user for them.

System Infers Min/Max Values

For a relationship, A VC B, the system tries to infer the min/max values for A and B by examining: 1) the verb phrase and 2) the form of the entities (singular or plural) as they appear in the relationship.

1. Is-a Verb Phrases

If the verb phrase is is-a, then a relationship A is-a B is interpreted as a connection between an individual A and a generic B, or A is a subset of the superset B [Brachman, 1983]. Therefore, each value of A can have one and only one corresponding value of B so the min/max values of A are defined to be (1,1). Since B is a generic term, for each value of B there may or may not be a corresponding value of A. Thus, the min/max values of B are (0,1).

E.g.
Enter next relationship / "see." / "end."
|: manager is_a employee.

The min/max values for manager are defined to be (1,1) because for each manager there can be one and only one employee. The min/max
values for employee are (0,1) because an employee may or may not be a manager.

2. Entities in Singular or Plural Form

The View Creation System can also infer something about the mapping ratios by examining the form (singular or plural) of the entities as they appear in a relationship.

First, consider a relationship A VC B where both A and B are singular. Recall that entity names are requested in their singular forms. The system compares an entity name as it appears in a relationship with the original entity name provided. If they are exactly the same, the system infers that the entity is used in its singular form; otherwise, it must be in its plural form. When both A and B are singular, the min/max values of A must be (1,1) because the user has implied, by using singularity for both entities, that there is one and only one B for each A. The inverse is not implied so the user must be queried for the min/max B values.

E.g.
Enter next relationship / "see." / "end."
|: book has publisher.

Since both book and publisher are in their singular forms, this implies that a book has one and only one publisher. Therefore the min/max values for book are (1,1). The inverse is not implied because, obviously, a publisher is not restricted to publishing only one book.

Another situation where the system can infer something about the mapping ratios occurs when A is singular and B is plural. Since B is plural this implies that, for a single value of A, there could be more than one value of B. Hence, the system infers that the maximum cardinality for A is N.
Query User for Min/Max Values

When the system cannot infer the min/max values for \( A \) and/or \( B \), it must query the user for them.

For a relationship \( A \ VC B \) the user is asked a series of questions which require a "yes" or "no" answer with the number of questions ranging from two to four. The user is first asked if there is at least one \( B \) value for each \( A \) value. If the answer to this question is "no", the min/max values for \( A \) are \((0,N)\) where the value of \( N \) can range from one to many. When the minimum value is zero, as is the case here, no further refinement of the maximum number of \( B \) values is necessary. As will be discussed later, when the min/max values are \((0,N)\), it is the minimum value which dictates what type of representation is appropriate for the corresponding relationship.

If the answer to the question "Is there at least one \( B \) value for every \( A \) value?" is "yes", the user is then asked if there is at most one \( B \) for each \( A \). If the answer to the latter question is also "yes", the min/max values of \( A \) are \((1,1)\); otherwise they are \((1,N)\).

Analogous questions are asked to determine the min/max values of \( B \) – i.e. the minimum and maximum number of \( A \) values which can occur for each value of \( B \). In the worst case, four questions must be posed and answered.

*E.g.:* Relationship "libraries have branches"

*(Dialogue)*

Is there at least one "branch" for every "library"?
("yes." / "no.")
|: y.
Is there at most one "branch" for every "library"?
("yes." / "no.")
|: n.
Is there at least one "library" for every "branch"?
("yes." / "no.")
\[ \text{y.} \]
Is there at most one "library" for every "branch?"
("yes." / "no.")
\[ \text{y.} \]

Here, the min/max values are: A (1,N) and B (1,1), i.e. each library can have 1 to N branches; each branch has one and only one library.

**Relationship Representation**

The next step is to determine an appropriate representation for the relationship. As will become apparent in the discussion which follows, the cardinalities which dictate what representations are valid are 0, 1, and N (many). This results in the following six combinations of min/max A and B values being sufficient to consider when deciding how to represent a relationship.

1) \( A (1,1) \) and \( B (1,1) \)
2) \( A (1,1) \) and \( B (0,N) \)
3) \( A (0,N) \) and \( B (1,1) \)
4) \( A (1,1) \) and \( B (1,N) \)
5) \( A (1,N) \) and \( B (1,1) \)
6) others.

**Case i: A (1,1) and B (1,1)**

There are three ways a relationship with \( A (1,1) \) and \( B (1,1) \) can be represented – i) the key attributes of \( A \) can be added as non-key attributes of \( B \); ii) the key attributes of \( B \) can be added as non-key attributes of \( A \); or iii) both. Since one has the option of using either the \( A \) or \( B \) entity to represent the relationship, there should never be a need to construct a new relation.
Consider first the entity $A$. Since each value of $A$ determines one and only one value of $B$, the key attributes of $B$ could be added as non-key attributes of $A$ (i.e. $B$ appears as a foreign key in the entity relation $A$) and a connection between the two entities (representing the relationship) would exist. One would then be able to navigate in the database from $A$ to $B$. Similarly, since each value of $B$ determines one and only one corresponding value of $A$, adding the key attributes of $A$ as non-key attributes of $B$ would ensure that a unique path exists in the database from $B$ to $A$.

Since there is one and only one $B$ value for each $A$ value and vice versa, adding the key attributes of $A$ as non-key attributes of $B$ and adding the key attributes of $B$ as non-key attributes of $A$ would ensure that a path in the database could be navigated from $A$ to $B$ or from $B$ to $A$. The only problem with this representation is that an update anomaly could exist when the relationship between $A$ and $B$ changes because the information would have to be updated in two places. The alternative representation is to construct a relation between the entities $A$ and $B$. However, adding non-key attributes is a simpler operation than constructing a relation. Also, an update anomaly, if one occurred, would happen in only two places (instead of the usual unknown number of places). Therefore, adding the key of $A$ to $B$ and adding the key of $B$ to $A$ is preferred over constructing a new relation.

Of the three possibilities, the actual choice of representation for a relationship is determined by the type of queries that are of interest to the user. The system simply asks the user which query he is most interested in:

a. the $A$ of $B$ or
b. the $B$ of $A$.

If the user is interested in the $A$ of $B$, then (since each value of $B$ corresponds to one and only one value of $A$) the key attributes of $A$ are added as non-key attributes of $B$. If the user is interested in the $B$ of $A$, the key attributes of $B$ are added as non-key attributes of $A$. 
(since each value of $A$ corresponds to one and only one value of $B$). If both types of queries are equally likely, or the user does not know, both of the above are done (i.e. the key attributes of $A$ added as non-key attributes of $B$ and the key attributes of $B$ added as non-key attributes of $A$).

E.g.: Relationship "library has director"
(Each library has one and only one director; each director corresponds to one and only one library.)

(Dialogue)
Which would you ask most often:
a) the "library" of "director" or
b) the "director" of "library"?
("a." / "b." / "no." – if you do not know or both are equally likely).

If: "a" add the key attributes of library as non-key attributes of director.
If: "b" add the key attributes of director as non-key attributes of library.
If: "no" add the key attributes of library as non-key attributes of director and add the key attributes of director as non-key attributes of library.

Case ii: $A$ (1,1) and $B$ (0,N)

Both this and the third case deal with optional relationships. When the min/max values of $A$ are (1,1), one and only one value of $B$ corresponds to each value of $A$. Therefore, as illustrated in the previous case, the relationship could be represented by adding the key attributes of $B$ as non-key attributes of $A$.

Now consider the $B$ entity in the relationship which has min/max values (0,N). The relationship from $B$ to $A$ is an optional one because, for each value of $B$, there may be from zero to $N$ values of $A$ where $N$ can range from one to many. In mathematical terms, the mapping from $B$ to $A$ is not a total mapping. (Refer to Tsichritzis and Lockovsky, 1982).
Therefore, since the mapping from \( B \) to \( A \) is not complete, one would not make provisions for navigating in the database from \( B \) to \( A \). Thus, in this case, a relationship is represented by adding the key attributes of \( B \) as non-key attributes of \( A \). Note that this representation is valid regardless of the exact value of \( N \). The minimum \( B \) value of zero implies an incomplete relationship which is the overriding factor in determining the representation.

E.g.: Relationship "manager is-a employee"

The min/max values of manager are \((1,1)\) while the min/max values of employee are \((0,1)\). The relationship is represented by adding the key attributes of employee as non-key attributes of manager.

Case iii: \( A \ (0,N) \) and \( B \ (1,1) \)

This case is the inverse the above and is handled analogously. For a relationship \( A \ VC B \), the mapping from \( A \) to \( B \) is incomplete but the mapping from \( B \) to \( A \) is a total mapping with each value of \( B \) corresponding to one and only one value of \( A \). Therefore, the relationship is represented by adding the key attributes of \( A \) as non-key attributes of \( B \).

Case iv: \( A \ (1,1) \) and \( B \ (1,N) \)

Since the min/max values for \( A \) are \((1,1)\), this implies that there is one and only one value of \( B \) for each value of \( A \). Therefore, as in the first and second cases, the relationship from \( A \) to \( B \) can be represented by adding the key attributes of \( B \) as non-key attributes of \( A \). However, since there could be one to many values of \( A \) for each value of \( B \), it would be inefficient to make provisions for an unknown number of key values of \( A \) to be represented as non-key attributes of \( B \). Update anomalies could be incurred because, every time any value of \( A \) was updated, \( B \) would also have to be updated. Thus, if one wishes to obtain the relationship from \( B \) to \( A \), the relationship should not be represented by adding the key attributes of \( A \) as non-key attributes of \( B \). Instead, a relation should be constructed between the two entities.
Hence, the two options available for representing a relationship where \( A \) (1,1) and \( B \) (1,N) are: i) add the key attributes of \( B \) as non-key attributes of \( A \) or ii) construct a relation between \( A \) and \( B \). The choice of representation depends on how the database is to be navigated and hence, on the types of queries that are of interest to the user. As in the first case, the user is asked whether he is interested in the \( A \) of \( B \) or the \( B \) of \( A \).

If the user indicates that he is interested in the \( B \) of \( A \), the key attributes of \( B \) are added as non-key attributes of \( A \).

\[ E.g.: \text{branch: [BRANCH\_NAME, branch\_address]} \]
\[ \text{library: [LIBRARY\_NAME, director\_id]} \]

(Dialogue)
For the relationship "branches belongs to libraries"
Where each branch has one and only one library, but each library can have one to many branches."

Which of the two queries would you ask most often?
- a) the "branch of library" or
- b) the "library of branch"?
("a." / "b." / "no." if both are equally likely or you do not know).

| b.

Represent as:
- branch: [BRANCH\_NAME, branch\_address, library\_name]
- library: [LIBRARY\_NAME, director\_id]

However, if the user is interested in the \( A \) of \( B \) or both are equally likely (or he does not know), a relation \( A\_VC\_B \) is constructed whose key is the combined key attributes of both the \( A \) and \( B \) entities (and there are no non-key attributes).

\[ E.g. \]
For the relationship: "branch belongs to library"
Which of the two queries would you ask most often?
- a) the "branch of library" or
- b) the "library of branch"?

---------------

2 For simplicity, assume that BRANCH\_NAME uniquely identifies a branch.
The rationale for constructing a new relation is as follows. Suppose that a relationship $A \ VP B$ where $A(1,1)$ and $B(1,N)$ is always represented by adding the key attributes of $B$ as non-key attributes of $A$. Then when the type of query that is of interest is the "A of B", selecting the relevant tuples would require scanning the entire $A$ relation for appropriate values of the non-key attributes. However, in the system, a relation is constructed specifically to deal with "A of B" queries, so that the key of $B$ will be in the key of the relation against which the query is processed. This is done to facilitate retrieval. Obviously, the key of $A$ also has to be included as part of the key of the relation because the key of $B$ does not uniquely identify the relationship.

The key of $B$ should not just be added to the $A$ entity because, clearly, this would create partial functional dependencies. If the key of $B$ is not part of the key, then the entity $A$ is not an efficient vehicle for dealing with the "A of B" type of queries. Hence, the justification for constructing a new relation to deal with this type of query.

**Case v: A (1,N) and B (1,1)**

This case is the inverse of Case iv and is handled analogously. For a relationship $A \ VC B$, if the user is interested in the $A$ of $B$, the key attributes of $A$ are added as non-key attributes of $B$. If the user is interested in the $B$ of $A$ or both are equally likely (or he does not know), a relation $A_{-}VC_{-}B$ is constructed whose key consists of the key attributes of $A$ concatenated with the key attributes of $B$. 

("a." / "b." / "no." if both are equally likely or you do not know).

|: a

**Represent as:**

branch_belongs_to_library: [BRANCH_NAME, LIBRARY_NAME]
branch: [BRANCH_NAME, branch_address]
library: [LIBRARY_NAME, director_id]
Case vi: Others

In all other situations, a relation $A_{VC}B$ is constructed whose key consists of the concatenation of the key attributes of $A$ and $B$. This is the appropriate representation because one or both of the following conditions hold:

1. neither entity uniquely identifies the other one
2. the relationship is optional.\(^3\)

The relationship may have corresponding relationship attributes. If so, it is converted to an entity. (See next subsection.)

\(E.g.:\) Relationship "borrowers borrow volumes"
(Where a borrower can borrow zero to many volumes and a volume can have zero or one borrowers.)

borrower: [BORROWER_ID, borrower_name, address]
volume: [VOLUME_NUMBER, date_due]

Represent as:
borrowers_borrow_volumes: [BORROWER_ID, VOLUME_NUMBER]
borrower: [BORROWER_ID, borrower_name, address]
volume: [VOLUME_NUMBER, date_due]

Relationship Attributes

As is the case with entities, relationships can have attributes — i.e. properties or characteristics of the relationship as a whole which are of interest to the user. However, unlike entities, which usually have corresponding attributes, only some types of relationships have attributes. These relationships are identified by examining the min/max mapping ratios of the $A$ and $B$ values. It will be shown that relationship attributes only exist when

\(\text{In the case of a relationship where the min/max values for } A \text{ and } B \text{ are both (0,1), the relationship is optional in both directions with maximum cardinality of one. Then either the key of } A \text{ or the key of } B \text{ is sufficient as the key of the relation. The choice should depend on usage. For an example of this type of relationship, see Howe [1983].}\)
the min/max values of $A$ are $(0,N)$ or $(1,N)$ and the min/max values of $B$ are not $(1,1)$ or vice versa.  

(Recall that, for the purposes of the View Creation System, $N$ means, zero, one or many, depending on the situation. Therefore, $(0,1)$ may sometimes be represented as $(0,N)$ – see Knowledge Base.)

Consider a relationship $A \ VP B$ where the min/max $A$ values are $(0,N)$ or $(1,N)$ and the min/max $B$ values are $(1,1)$. It must be shown that a relationship attribute cannot exist in such a situation. Suppose that such a relationship attribute, $R_{att}$, does exist. $R_{att}$ is a function of the relationship $A \ VP B$ and hence a function, $f$, of the entities $A$ and $B$. Formally this is represented as:

$$R_{att} = f(A,B) \quad (1)$$

However, $A$ is a function of $B$ since there is one and only one value of $A$ for each value of $B$. Therefore, (1) can be re-written as:

$$R_{att} = f(f_1(B),B) \quad (2)$$

or:

$$R_{att} = f(B)$$

Thus, if a relationship attribute did exist, for $A$ $(0,N)$ or $A$ $(1,N)$ and $B$ $(1,1)$, it would be a function of the entity $B$ only, and hence, would appear as an attribute of $B$. Analogously, it can be shown that, for $A$ $(1,1)$ and $B$ $(0,N)$ or $B$ $(1,N)$, a relationship attribute would be a function of the entity $A$ and would thus appear as an attribute of $A$.

Now consider a relationship $A \ VP B$ where $A$ $(1,1)$ and $B$ $(1,1)$. This case is easily shown to be an extension of the above. Since there is one and only one $A$ value for each $B$ value, then:

$$R_{att} = f(f_1(B),B)$$

or:

---------------------------

$^4$ This proof was suggested by Yair Wand.
\[ R_{\text{att}} = f'(B) \]

Similarly, since there is one and only one \( B \) value for each \( A \) value,

\[ R_{\text{att}} = g(A, g_1(A)) \]

or:

\[ R_{\text{att}} = g'(A) \]

Therefore:

\[ R_{\text{att}} = f'(B) \text{ and } R_{\text{att}} = g'(A). \]

i.e.

\[ f'(B) = g'(A). \]

This implies, that if a relationship attribute did exist, it could be expressed as an attribute of either the entity \( A \) or the entity \( B \).

Thus, it has been shown that relationship attributes do not exist for relationships where the min/max values of at least one of the entities are \((1,1)\). Alternatively, relationship attributes may exist if neither \( A \) nor \( B \) have min/max values of \((1,1)\). When these types of relationships occur, the user is asked to identify corresponding relationship attributes.

*E.g.*: Relationship "borrowers borrow volumes"
(where a borrower may borrow zero, one or many volumes; a volume may be borrowed by zero or one borrowers.)

(System instructs the user.)

An attribute of a relationship is a property or characteristic of the relationship which is of interest to you.

E.g. Relationship: "passengers reserve for flights"
Acceptable response: "class".

Note that "class" is not an attribute of "passenger" or "flight" but is specifically an attribute of the relationship:
"passengers reserve for flights".

When I add certain relationships I will ask for corresponding attributes. Relationship attributes may not always exist.

(System asks the user for relationship attributes.)

Enter attribute(s) for "borrowers borrow volumes" / "end."
|: date_due.

The system checks that a relationship attribute has not previously been identified as an attribute of one of the entities. If this situation occurs, the system informs the user of the inconsistency and asks whether the attribute should belong to the entity or the relationship.

E.g. (from above)

I already know that "date_due is an attribute of "volume". Think carefully. Is "date_due" an attribute (i.e. a property or a characteristic) of:
a) the entity "volume"
b) the relationship "borrowers borrow volumes"
("a." / "b." / "no." if you do not know.)
|: ...

Relationships Converted to Entities

When a relationship A VP B has at least one attribute, it is converted to an entity. The key attributes of the new entity are the key attributes of the A and B entities. (Since the primary keys for A and B have not yet been obtained, the system simply uses the names A and B as the key attributes of the new entity. After primary keys have been obtained, A and B will then be replaced by their primary key attributes. Refer to the example below.)

The non-key attributes of the new entity are the relationship attributes. Eventually, the entity will be represented as a relation as would the relationship if the conversion did not take place. Therefore, the conversion does not affect the final output.
The relationship is converted to an entity at this point because the system must verify that the relationship attributes are single-valued. If any are multivalued, a violation of First Normal Form would be incurred. By treating the relationship as an entity, the system is able to invoke its procedure which checks for multivalued attributes. This is a recursive procedure because, if an original relationship attribute is multivalued, it also becomes an entity. The system then determines if any attributes of this entity are multivalued. Any attributes which are in turn become new entities, etc. The system also makes certain that a relationship is provided between each entity which originally had a multivalued attribute and the new entity which resulted. (See Entities - Multivalued Attributes.)

A new entity created in this process can take part in relationships involving either another entity or another relationship which is converted to an entity. In this way, the system allows for relationships between certain relationships and between certain entities and relationships. Thus, the system is not bound by the original E-R limitations.

Before converting a relationship to an entity, the system first gives the user an opportunity to provide a shorter name for the concept being modelled.

E.g.: borrower: [BORROWER_ID, borrower_name, . . .]
volume: [VOLUME_NUMBER]
Relationship: borrowers borrow volumes

(Dialogue)
Enter relationship attribute(s) for "borrowers borrow volumes" / "end."
|: date_due.
A relationship of this type is often more conveniently dealt with as an entity so I am going to convert it to one.
Do you wish to provide a shorter name for "borrowers_borrow_volumes"? Enter new name in its SINGULAR form. / "end."
The name may be one word or two words connected with an underscore character.
Here are the attributes for:
loan: [borrower, volume, date_due]

Can a single "loan" have more than one value for any of these attributes?

In the above example, loan became a new entity and the system asked if any of its attributes could be multivalued. If a borrower is allowed to borrow a volume more than once, and the user wishes to maintain historical information, date_due is multivalued and will be converted to a new entity. The system "remembers" that a relationship will be needed between loan and date_due and will ensure that the user provides one at a later point. (See Entities - Multivalued Attributes.) Eventually, the attributes borrower and volume will be replaced by their primary keys to provide the key attributes for loan.

E.g.
borrower: [BORROWER_ID, borrower_name, ...]
volume: [VOLUME_NUMBER]
loan: [BORROWER_ID, VOLUME_NUMBER, ...]

Relationship Modification

At various points, the user is given an opportunity to add or delete relationships and add relationship attributes. If the user tries to add an attribute to a relationship where the min/max values of at least one of the entities are (1,1), the system will inform the user that he is really providing another entity attribute, not a relationship attribute. For example, suppose the min/max values of B are (1,1) and the min/max values of A are (1,N) and the user tries to add a relationship attribute. The system will suggest that the user is probably providing an attribute of the entity B although it could be an attribute of A.
E.g.: Relationship "books have volumes"
(Each book can have one to many volumes but each volume corresponds to one and only one book.)

(Dialogue)
Do you wish to add any relationship attributes (i.e. properties or characteristics of a relationship as a whole)?
("yes." / "no.")
|: yes.

Enter relationship / "end."
|: books have volumes.
Enter attribute(s) / "end."
|: prev_borrower.

Logically, "prev_borrower" should not be an attribute of the relationship. It should probably be an attribute of "volume" although it could be an attribute of "book". Should I make it an attribute of:
a) book
b) volume
("a." / "b." / "no." if you do not know.
|: ...

(When the user's response is "no", the system does not do anything. If the user cannot decide whether an attribute belongs to an entity or a relationship, the system cannot make the decision. In such a case, it is assumed that the user probably made a mistake.)

This is another way that the system endeavors to help the user understand the application.
It also provides the user with an opportunity to add information which had previously been missing (i.e. to add entity or relationship attributes which the user had failed to identify earlier).

HAVE/HAS RELATIONSHIPS

For relationships of the form A have/has B where A and B are both entities, the verb phrase is ambiguous. Four interpretations of have/has are possible:
1. \(A\) possesses \(B\)
2. \(B\) component-of \(A\)
3. \(B\) instance-of / example-of \(A\)
4. \(B\) associated-with \(A\) in some other way.

It is the third interpretation that is of most interest to the system. As previously mentioned, instance-of relationships are important in selecting primary keys and detecting inconsistencies in the user's input. The other three interpretations (possesses, component-of and associated-with) are employed because they reflect more of the semantics of the application than do the verbs have and has.

For each have/has relationship, the user is asked to identify which of these four is an appropriate interpretation and the relationship is modified accordingly. The rules for dealing with have/has relationships are outlined below. They were obtained by considering and testing the system with have/has relationships from different application domains.

1. **IF**: \(A\) have/has \(B\) where the have/has is possession;
   **THEN**: change the relationship from \(A\) have/has \(B\) to \(A\) possesses \(B\).
   E.g. library has branches is converted to library possesses branches.

2. **IF**: \(A\) have/has \(B\) where the have/has is component_of;
   **THEN**: change the relationship from \(A\) have/has \(B\) to \(B\) component_of \(A\).
   E.g. library has book_stacks is converted to book_stacks component_of library.

3. **IF**: \(A\) have/has \(B\) where the have/has is instance_of;
   **THEN**: change the relationship from \(A\) have/has \(B\) to \(B\) instance_of \(A\).
   E.g. books have volumes is converted to volumes instance_of books.

4. **IF**: \(A\) have/has \(B\) where the have/has is associated_with;
   **THEN**: change the relationship from \(A\) have/has \(B\) to \(B\) associated_with \(A\).
   E.g. book has publisher is converted to publisher associated_with book.
The following example illustrates the interaction with the user.

_E.g.: "books have volumes"

Which of the following is true?

a) "book" owns / possesses "volume"

b) "volume" is a component of "books"

c) "volume" is an instance / example of "book"

d) "volume" associated with "book" (in some other way)

"a." / "b." / "c." / "d." / "no." (if you do not know.)

|: c.

After categorizing the _have/has_ relationships, the system tries to ensure that _is-a_ and _instance-of_ verb phrases appear in conjunction with the most appropriate entities. For relationships _A is-a/instance-of B_, it searches for relationships involving _B_ and some other entity _X_. The system then asks whether the user's information requirements would be more accurately reflected if _X_ was associated with the subset entity _A_ instead of the superset entity _B_ or whether both associations are needed. The aim of this procedure is to ensure that the user's requirements are specified at the right level of detail.

_E.g._ Relationships:

volume instance of book

borrowers borrow books

_(Dialogue)_

I know that "volume instance of book" and "borrowers borrow books".

Which relationship(s) should I have:

a) borrowers borrow volumes

b) borrowers borrow books

c) both

"a." / "b." / "c." / "no." if you do not know.

|: ...

1. _If the response is "a", the relationship "borrowers borrow books" is changed to "borrowers borrow volumes"._

2. _If the response is "b" or "no", no change is made._
3. If the response is "c", the relationship "borrowers borrow volumes" is added.

MODIFICATIONS – FIRST

At this point in the View Creation System’s procedure, the user’s initial set of entities and relationships has been obtained. This set is displayed and the user given an opportunity to make modifications. If modifications are made, the system ensures that its database remains consistent. For example, if the user adds a new entity, corresponding attributes and candidate keys must be obtained and the user queried for relationships which include this entity. If a have/has relationship is obtained the appropriate type of have/has must be identified, etc.

SYNONYMS

The system attempts to detect redundant information by querying the user about possible synonyms in the entity names and the verb phrases. It also tries to identify relationships which capture the same information in different ways.

Synonyms – Entities

For relationships of the form $A_1 \ VP \ B_1, A_2 \ VP \ B_2$, etc. (where the VP’s are the same), the $A$ and $B$ entities are candidates to be either synonyms or related in some way that the system does not already know about.

The $A$ values are examined first and then the $B$ values. For each pair of entities, the user is asked whether the entities are synonyms or if one is a subset of the other. If synonyms are found, the user is asked which term should be used. When one entity is a subset of the other, an is-a relationship is created between the two.
E.g. Relationships:
student borrows volumes
borrower borrows volumes

(Dialogue)
Which of the following is true about a "student" and a "borrower"?

a) They refer to exactly the same thing.
b) "student" is a subset of "borrower"
   (i.e. for every "student" there is a "borrower" but for every
    "borrower" there may or may not be a "student")
c) "borrower" is a subset of "student"
   (i.e. for every "borrower" there is a "student" but for every
    "student" there may or may not be a "borrower")
d) None of the above or you do not know.

("a." / "b." / "c."/ "d.")

1. If the response is "a", the user is asked which entity should be
   used and the other is deleted.
2. If the response is "b", the relationship "student is-a borrower" is
   added.
3. If the response is "c", the relationship "borrower is-a student" is
   added.
4. If the response is "d", no change is made.

Synonyms – Verb Phrases

For relationships of the form $A\ VP_1\ B$, $A\ VP_2\ B$, etc., the verb phrases $VP_1$, $VP_2$, etc. are
candidates to be synonyms.

The system asks the user if any of the verb phrases are synonyms. If synonyms are
detected, the user is asked what term he would like to use. The relationship with the
replacement term is retained and the others deleted.
E.g. Relationships:
library lends volumes
library loans volumes

(Dialogue)
Are any of the following synonyms, i.e. different words for the same thing?

lends
loans

("yes." / "no." / "why.")
|: yes.
Which term would you like to use? It must be one of:

|: loans.
Fine, changed to "loans".

Synonyms – Relationships

If there are relationships of the form $A \text{ VP}_1 B$, $B \text{ VP}_2 A$, etc., the user is asked if he has represented the same information in different ways. If the user indicates that he has done so, he is further asked which relationship he wishes to use. The relationship chosen is retained and the others deleted.

E.g. Relationships:
library lends volumes
volumes on _loan_ from library

(Dialogue)
Are any of the following different ways of saying the same thing?

library lends volumes
volumes on _loan_ from library

("yes." / "no." / "why?")
|: yes.

Which one would you like to use? It must be one of:

library lends volumes
volumes on _loan_ from library
INHERITED ATTRIBUTES

For relationships of the form \(A\ is-a\ /\ instance-of\ B\), "inherited" attributes are examined. The system displays any attributes of the superset entity \(B\) which do not appear as attributes of the subset entity \(A\). The user is asked if it would be incorrect to attribute to entity \(A\) any of these attributes of \(B\). If it is incorrect to do so, the system suggests that there may be an inconsistency in the user's input. The user is given an opportunity to either delete the relationship (and possibly add a new relationship(s)) or to modify the attributes of the superset entity.

\(E.g.:\) Relationship "manager is-an employee"

Employee: [EMPLOYEE_NUMBER, employee_name, union]
Manager: [MANAGER_ID, title, department]

(Dialogue)
Since "manager is-a employee", we should be able to attribute to "manager" all of the attributes of "employee". Would it be incorrect to attribute any of the following attributes of "employee" to "manager"?
1) employee_number 2) employee_name 3) union
("yes." / "no.")
|: yes.

Then should the relationship "manager is-a employee" exist?
("yes." / "no." / "why?")
|: yes.

Then do you wish to modify any of the attributes of "employee"?
("yes." / "no.")
|: y.

(The attribute "union" is deleted from the entity "employee".)
MODIFICATIONS – LAST

If there were any synonyms or inherited attributes to query the user about, the user is again given an opportunity to make modifications to his entities and relationships. The system ensures that its database remains consistent.

The system also tries to build on what it has "learned". For instance, suppose the user enters a term which was previously identified as a synonym for another term. The system asks whether the user made a mistake by introducing the term again or if new information has been provided.

PRIMARY KEYS

Each entity must be uniquely identified in a database by a primary key. Primary keys are used in database management systems to determine how information will be stored and retrieved. In the View Creation System, primary keys are automatically (without interaction with the user) selected by the system from a set of candidate keys (i.e. from a set of attributes or groups of attributes which uniquely identify an occurrence of an entity).

A database designer usually selects a primary key from a set of candidate keys. Any of the candidate keys must, by definition, be acceptable. The ultimate choice depends on the designer's objective. In the View Creation System, the rules for choosing primary keys are heuristic ones. They concentrate first on obtaining the simplest possible key as the primary key, i.e. the candidate key which has the least number of elements (attributes). When this criterion does not result in a unique choice, the candidate key which appears most often as a candidate or primary key for other entities is selected. This criterion aims at enhancing retrieval efficiency by minimizing the number of join operations which may be required during the use of the database. Finally, if neither of these criteria are met (or there is a tie), the candidate key that was given first is chosen as it may be the most natural one.
When obtaining primary keys, the system considers three different kinds of entities –
i) those which occur in is-a hierarchies ii) those which occur in instance-of hierarchies and
iii) all others.

Is-a Relationships

An is-a hierarchy occurs when there are relationships of the form ... X is-a A, A is-a B. If
I is-a J, there is one and only one J for each I (and, at most, one I for each J). Since J is the
generic term for I, the key of J must uniquely identify an instance of I. Therefore, the key of
J can serve as a candidate key for the entity I. (E.g. if manager is-a employee and the key of
employee is employee_number, then employee_number also uniquely identifies manager.)

To obtain primary keys for entities in an is-a hierarchy, consider first the superset
entity, B. The system determines the primary key for this entity (based on its rules for
obtaining primary keys). The primary key for B is then added to the set of candidate keys
for its subset entity A. B’s primary key attributes are also added as non-key attributes in
the original set of candidate keys for A. A’s primary key is then chosen and this key in turn
added as a candidate key of the entity for which A is a superset. The process continues until
primary keys have been obtained for all the entities in the is-a hierarchy.

E.g.

Relationships in is-a hierarchy:
i) employee is-a person
ii) manager is-a employee.

Original set of candidate keys:
Person: [PERSON_NAME, ADDRESS, ... ]
Employee: [EMPLOYEE_NUMBER, ... ]
Manager: [TITLE, manager_name, department, ... ]
Manager: [MANAGER_NAME, DEPARTMENT, title, ... ]
The primary key \{PERSON\_NAME, ADDRESS\} is obtained for the entity, "person". It is then added as a candidate key for "employee":

Person: \{PERSON\_NAME, ADDRESS, \ldots \]
Employee: \{EMPLOYEE\_NUMBER, person\_name, address \ldots \]
Employee: \{PERSON\_NAME, ADDRESS, employee\_number, \ldots \]
Manager: \{TITLE, manager\_name, department, \ldots \]
Manager: \{MANAGER\_NAME, DEPARTMENT, title, \ldots \]

The primary key is chosen for "employee":
Person: \{PERSON\_NAME, ADDRESS, \ldots \]
Employee: \{EMPLOYEE\_NUMBER, person\_name, address \ldots \]
Manager: \{TITLE, manager\_name, department, \ldots \]
Manager: \{MANAGER\_NAME, DEPARTMENT, title, \ldots \]

The primary key for "employee" is added as a candidate key for "manager":
Person: \{PERSON\_NAME, ADDRESS, \ldots \]
Employee: \{EMPLOYEE\_NUMBER, person\_name, address \ldots \]
Manager: \{TITLE, manager\_name, department, employee\_number \ldots \]
Manager: \{MANAGER\_NAME, DEPARTMENT, title, employee\_number \ldots \]
Manager: \{EMPLOYEE\_NUMBER, manager\_name, department, title, \ldots \]

The primary key for "manager" is determined:
Person: \{PERSON\_NAME, ADDRESS, \ldots \]
Employee: \{EMPLOYEE\_NUMBER, person\_name, address \ldots \]
Manager: \{EMPLOYEE\_NUMBER, manager\_name, department, title, \ldots \]

At this point, attribute names may need to be adjusted. For example, if there is a non-key attribute in the superset entity of the form $\text{Ent}^{\text{super}}_{\text{supset}}$ -att and a non-key attribute in the subset entity of the form $\text{Ent}^{\text{sub}}_{\text{subset}}$ -att, the attribute $\text{Ent}^{\text{sub}}_{\text{subset}}$-att is deleted from the subset entity. This is done to avoid redundancy. Since $A$ is-a $B$, the key attributes of $B$ will either serve as the primary key of $A$ or have been added as non-key attributes for $A$. Therefore, the connection between the two entities will exist so the subset entity can inherit the non-key attributes of the superset entity.
E.g.
Relationship: employee is-a person

Person: [PERSON_NAME, ADDRESS, person_birthday...]
Employee:
[EMPLOYEE_NUMBER, employee_birthday, person_name, address...]

Becomes:
("employee_birthday" is deleted from "employee")

Person: [PERSON_NAME, ADDRESS, person_birthday...]
Employee: [EMPLOYEE_NUMBER, person_name, address]

Key attributes may also need to be adjusted. For subset entities which adopt the primary keys of their superset entities, the primary keys are prefixed by their entity names.

E.g.
Relationship: manager is_a employee

Employee: [EMPLOYEE_NUMBER, person_name, address]
Manager: [EMPLOYEE_NUMBER, department, title,...]

Becomes:
Employee: [EMPLOYEE_NUMBER, person_name, address]
Manager: [MANAGER_EMPLOYEE_NUMBER, department, title,...]

This adjustment allows other (hierarchical) relationships between a subset and a superset entity to be represented. In the above example, if one wished to model a relationship employees work_for managers, this relationship could eventually be represented as a relation:

employees work_for managers:
[MANAGER_EMPLOYEE_NUMBER, EMPLOYEE_NUMBER]

Prefixing the primary key with its entity name makes it possible for one to clearly distinguish what role each entity plays in the relationship.
**Instance-of Relationships**

*Instance-of hierarchies* are similar to *is-a hierarchies* – e.g. ... *X instance-of A, A instance-of B.* If *I instance-of J,* for each *I* there is one and only one *J* but for each *J* there could be many *I*’s. Therefore, the key of *J* does not uniquely identify an instance of *I* (as in the case of an *is-a* relationship). Since there could be many *J*’s in the database with each corresponding to multiple *I*’s, a unique identifier of *I* must include the key of *J.*

As in the case of *is-a hierarchies,* *instance-of* hierarchies are processed from the entity at the highest level downwards. The system determines the primary key for the superset entity which is at the top of the hierarchy. This primary key is concatenated to each of the candidate keys of its subset entity. The primary key for the subset entity is then determined based on the modified forms of the candidate keys. This primary key in turn is concatenated to the candidate keys of its subset entities and the process repeated until the primary keys for all the entities in the instance-of hierarchy have been selected.

The attributes of the entities are scanned. If there are attributes of the form $\text{Ent}_{\text{superset}}$-att for a superset entity and $\text{Ent}_{\text{subset}}$-att for a subset entity, the attribute $\text{Ent}_{\text{subset}}$-att is deleted from the subset entity. Since a connection between the two entities exists (in the key of the subset entity), the subset entity can inherit the attributes of the superset entity so it would be redundant to repeat them for the subset entity.

*E.g.:
Relationship "volume instance-of book"

*Candidate Keys:*
book: [CATALOG_NO, book_title, author, publisher]
book: [BOOK_TITLE, AUTHOR, catalog_no, publisher]
volume: [COPY_NO, volume_title].

*The system determines the primary key for "book":*
book: [CATALOG_NO, book_title, author, publisher]
The candidate key for "volume" becomes:
volume: [CATALOG_NO, COPY_NO, volume_title].

The attribute "volume_title" is deleted from "volume".
volume: [CATALOG_NO, COPY_NO].

Entities Requiring Other Entities for Identification

There could be assertions in the system's database which indicate that key attributes of one entity may be needed to uniquely identify another entity and should thus be incorporated as part of the latter entity's key. Two situations could have provided these assertions (see Entities and Relationships - Prompts for Missing Relationships):

1. The user provided an attribute Term for a new entity $E_{New}$, where Term was some form of the name of an existing entity $E_{Prev}$ (e.g. $E_{Prev}$, $E_{Prev}_id$, $E_{Prev}_name$, etc.)

2. The user provided, as the name of a new entity $E_{New}$, the name of an attribute $A_{Prev}$ of an existing entity $E_{Prev}$.

In the first case, the user was implying that $E_{Prev}$ may be needed to uniquely identify an instance of the entity $E_{New}$ while, in the second case, the user was implying that $E_{New}$ may be needed to uniquely identify an instance of $E_{Prev}$. Consider the first case. (The second is handled analogously.) If, at this point, the primary key of $E_{New}$ contains the attribute Term then Term must be replaced by the key attributes of $E_{Prev}$.

E.g.: $E_{New} = branch$; $E_{Prev} = library$; Term = library
Assertion: the primary key of library may be needed to uniquely identify branch.

Primary Keys:
Library: [LIBRARY_ID, library_name, library_address, director]
Branch: [BRANCH_NAME, LIBRARY, branch_address]

Since the primary key of branch contains the attribute LIBRARY, the attribute is replaced by the key attributes of library to obtain:
If there is an assertion which indicates that an entity $E_{prev}$ may be needed to uniquely identify an instance of the entity $E_{new}$, but the primary key of $E_{new}$ does not contain an attribute which refers to $E_{prev}$, the system prompts the user to provide a relationship between the two entities. The procedure, repeated below, is the same as that outlined in Relationships – Prompts for Missing Relationships.

E.g.

Assertion: the primary key of library may be needed to uniquely identify branch (because library was originally provided as an attribute of branch).

Primary Keys:
Library: [LIBRARY_ID, library_name, library_address, director]
Branch: [BRANCH_NAME, LIBRARY_ID, branch_address]

The primary key of branch does not contain the attribute library, so the system prompts the user for a relationship between library and branch.

(Dialogue)
Is there a relationship between "library" and "branch"?
("yes." / "no." / "why?")
|: why?
When you gave me the entity "branch" you said that "library" was one of its attributes. That tells me that "branch" and "library" could be related. Is there a relationship between "branch" and "library"?
("yes." / "no.")
|: yes.
Enter relationship(s). Enter "end." when you finish.
|: library has branches.

I'm going to consider how to represent the relationship(s) later so for now I will delete "library" as an attribute of "branch".

If the user was unable to provide a relationship between "library" and
"branch", the system would ask if "library" should really be an attribute of "branch" and delete or retain the attribute as indicated by the user's response.

REPRESENT ENTITIES and RELATIONSHIPS

Entities

Each entity is represented by a separate (entity) relation with the key and non-key attributes of the relation corresponding directly to those of the entity. When the relation is actually constructed, it may not be in its final form. Modifications will be required if the relation violates normalization principles. The necessary modifications are dealt with during the next stage.

Relationships

There could be two different types of assertions in the system's database which indicate how relationships are to be represented. These assertions deal with key values which, when the assertions were made, were uninstantiated because primary keys had not yet been obtained. Now that primary keys have been obtained, the system can complete the user's set of information requirements by incorporating the information contained in the assertions.

Each relationship can be represented in either of two ways (see Mapping Ratios):

1. by constructing a relation whose key is the concatenation of the key attributes of the two entities in the relationship; or
2. by adding the key attributes of one entity as non-key attributes of the other.
Construct Relations

Examples of relationships which are to be represented as relations can be found in *Mapping Ratios*. These relations are now added to the system's database.

*E.g.*
Relationship "author writes books"
Assertion: A relation should be constructed between author and book.

Entities:
- author: [AUTHOR_ID, author_name, ...]
- book: [CATALOG_NO, title, ...]

Relation:
- author writes book: [AUTHOR_ID, CATALOG_NO]

Add Key Attributes of one Entity to Another Entity

Examples of relationships which provided these types of assertions are also found in *Mapping Ratios*. The primary keys have been obtained so the entities can now be modified accordingly.

*E.g.*
Relationship "manager manages department"
Assertion: Add the key of "manager" to the entity "department".

Entities:
- Department: [DEPT_NO, dept_name]
- Manager: [MANAGER_ID, title]

Adding the key of manager to department results in:

- Department: [DEPT_NO, dept_name, manager_id]
- Manager: [EMPLOYEE_NO, title]
FUNCTIONAL DEPENDENCIES

The system deals with two types of undesirable functional dependencies – partial and transitive dependencies. In normalization theory, partial functional dependencies violate rules of Second Normal Form while transitive functional dependencies violate those of Third Normal Form. (See Discussion of System’s Corrections.) In both cases, the system explains the relevant concept and asks the user to identify any dependencies. In essence, the View Creation System acts only as a clerical system when identifying these dependencies. However, it is believed that this is the manner in which a human designer would interact with a user if the designer was not familiar with the details of the database application domain.

Partial Functional Dependencies

Consider a relation where there are two or three key attributes and at least one non-key attribute. For each non-key attribute, the user is shown all possible combinations of key attributes and asked which combination is necessary before a value of the non-key is known.

E.g.: flight: [FLIGHT_NO, DATE, city_from, city_to, #_passengers]

(Dialogue)
Which of the following do you need before you know the "city_from"?
   a) FLIGHT_NO
   b) DATE
   c) FLIGHT_NO & DATE
"a." / "b." / "no." if you do not know.
|: a.

Which of the following do you need before you know the "city_to"?
   a) FLIGHT_NO
   b) DATE
   c) FLIGHT_NO & DATE
"a." / "b." / "no." if you do not know.
|: a.

-----------

5 Several designers expressed this view.
For non-keys which depend on the same partial key, a new relation is constructed whose key is the partial key and whose non-keys are the attributes which are partially dependent. The partially dependent non-keys are then deleted from the original relation and the user asked for the name of the new relation.

_E.g. (from above)_

(Dialogue)
Please give me a name for:
[FLIGHT_NO, city_from, city_to]
|: timetable.

_New Relations:_
flight: [FLIGHT_NO, DATE, _#_passengers]
timetable: [FLIGHT_NO, city_from, city_to]

For relations where the number of key attributes is four or more and there is at least one non-key attribute, the relation is displayed. The user is asked to identify: i) any non-key attributes which depend on only part of the key as opposed to the complete key and ii) the partial key on which a partially dependent non-key attribute depends. Partial functional dependencies described in this way are treated as above.

_Transitive Functional Dependencies_

Each relation which has two or more non-key attributes is displayed. The user is asked to identify any non-key attribute(s) which depends on other non-key attributes instead of the primary key. A non-key attribute which meets this criterion has a transitive functional dependency and the user is asked to indicate the other non-key(s) upon which it depends.

When a transitive dependency is identified, a new relation is constructed. The transitivity dependent non-key attribute is a non-key in the new relation and the non-key attribute(s) upon which it is transitively dependent serves as the relation key. The name of
the new relation is the name of the non-key (or the names of the non-keys concatenated) upon which the original non-key is transitively dependent. The transitively dependent non-key is deleted from the original relation (but the non-keys upon which it depends are retained).

If the user indicates that there are no transitive dependencies but there are attributes of the form \(X, X_{\text{Suffix}1}, X_{\text{Suffix}2}, \text{etc.}\), the system prompts the user to consider these attributes.

\[E.g.: \text{Book: [CATALOG\_NO, title, author, publisher, publisher\_city]}\]

(\textit{Dialogue})

Enter any non-key attribute(s) which depend on other non-key attribute(s) / "end."

|: end.

What about "publisher" and "publisher\_city"?

|: publisher\_city depends\_on publisher.

\textit{Relations produced:}

Book: [CATALOG\_NO, title, author, publisher]
Publisher: [PUBLISHER, publisher\_city].

\textbf{OUTPUT}

The system produces, as its final output, a set of Fourth Normal Form relations\(^6\) which represents a user's view.

\[E.g.\]
That's all I have to ask you about.
Here is what I have:

\[-----------\]
\textit{See Discussion of System's Correctness.}\(^6\)
DISCUSSION OF SYSTEM'S CORRECTNESS

This section will attempt to formally show that the View Creation System is theoretically correct. Two general cases will be considered. The first case assumes the most appropriate conditions under which the system could operate while the second case concentrates on the system's performance when some of the assumptions made in the first case are relaxed.

System's Performance under Ideal Conditions

For an "ideal" design session, assume the following:

1. The user is familiar with his application domain but not necessarily with database design concepts and terminology.

2. The user can express his information requirements using the following constructs:
   a. entities
   b. entity attributes
   c. binary relationships
   d. relationship attributes.

3. The user understands and correctly follows the system's instructions.

Based on these assumptions, it must be shown that the View Creation System can elicit a user's view and represent the view as a set of normalized relations.
Given the assumptions, it is almost trivial to show that the desired output can be obtained. First, the Entity-Relationship Model is a well-known and established model for representing user views in terms of entities and relationships and their respective attributes [Goldstein, 1985, Navathe and Schkolnick, 1978]. The literature on binary data models and their use in view modelling provides support for employing binary relationships [Bracchi, 1981; Ceri et al., 1981; Baldissera et al., 1979].

Each entity must be uniquely identified in a database by a primary key. In general, the choice of a primary key from among several candidates is at the discretion of the designer. Any of the candidate keys must, by definition, be acceptable. In the View Creation System, primary keys are automatically selected (i.e. without interaction with the user) by the system from a set of candidate keys. If the heuristics that the system uses in the selection process are inadequate, the design obtained may be suboptimal but not incorrect.

Consider the representation of the main constructs (entities, relationships and attributes) in the database. Using the Entity-Relationship Model, relations can be defined for each entity and each relationship in the database. Alternatively, some relationships can be represented by adding key attributes of one relation to another [Goldstein, 1985]. The latter is used by the system when the mapping ratios ensure that such a representation is valid. All entities are expressed as relations.

Normalization

The process of converting a set of relations into one which avoids certain anomalies is called normalization. The objective of this procedure is "to produce a database design that can be manipulated in a powerful way with a simple collection of operations while minimizing data anomalies and inconsistencies" [Goldstein, 1985].
A normalized set of relations is obtained by adhering to a set of (normalization) rules which guarantee that the relations produced are in a certain normal form. The most common normal forms range from first to fifth normal form with each enhancing the integrity of the data by minimizing redundancy and inconsistency. A set of relations which is in third normal form guarantees that each attribute is either part of the relation key or provides a single-valued fact about exactly the whole key and nothing else. Fourth and fifth normal forms deal with multivalued facts [Kent, 1983]. The following briefly outlines each of these normal forms and discusses how they are dealt with in the View Creation System.

1NF

First Normal Form requires that all tuples (rows) in a relation have the same set of attributes and that the attributes be atomic, indivisible items. Any relation produced by the Entity-Relationship process will be in First Normal Form [Goldstein, 1985].

The View Creation System guarantees that each instance of a relation has the same set of attributes. It does so by asking the user to identify entity and relationship attributes which are of interest for an entity or a relationship as a whole and not for particular instances of either construct. Single-valued attributes are also guaranteed. The system explicitly queries the user to identify multivalued attributes. Such attributes are converted to entities and are later represented as separate relations.

2NF

Second Normal Form requires that each non-key attribute refers to the complete key. A non-key which refers to only part of the key violates 2NF and is said to have a partial functional dependency.
In the View Creation System violations of second normal form are discovered by asking the user to identify any non-key attributes which depend on part of the key as opposed to the complete key. For relations with two or three key attributes, this is accomplished by displaying all possible combinations of key attributes and asking the user to indicate which combination is needed to uniquely identify a non-key. When a relation has four or more keys, the user is asked the same type of question but without the convenience of a list displaying the various combinations of key attributes.

In general, violations of Second Normal Form are resolved by splitting a partially dependent non-key from the rest of its relation. This is exactly the procedure the View Creation System follows. For non-keys which depend on the same partial key, a new relation is constructed whose key is the partial key and whose non-keys are the attributes which are partially dependent. The partially dependent non-keys are then deleted from the original relation and the user asked to provide a name for the new relation. (See Partial Functional Dependencies.)

3NF

Third Normal Form requires that a relation be in 2NF and that each attribute depends directly on the key. When a non-key attribute depends directly on other non-key attributes instead of the key attributes, a transitive functional dependency exists. Transitively dependent attributes are detected in the system by asking the user to indicate which non-key attributes meet this criterion.

Following normal database design practices, a violation of Third Normal Form is rectified by removing a transitively dependent attribute from its original relation and constructing a new relation. The new relation contains the transitively dependent attribute as its non-key and the original non-key(s) upon which it depends serves as the relation key.
The system does not query the user for the name of the new relation as it does when a violation of 2NF occurs. Instead, the name of the new relation is the name of the non-key attribute (or the names of the non-key attributes concatenated) upon which the transitively dependent non-key depends. (See Transitive Functional Dependencies.)

4NF

Although Third Normal Form provides a set of relations which is free from most cases of anomalous behavior, some complex conditions may still exist which can cause problems in the design. Fourth Normal Form requires that a relation should not contain two or more independent multivalued facts [Kent, 1983]. Stated alternatively, a relation should not represent more than one 1:N (one-to-many) relationship [Goldstein, 1985].

In the View Creation System, relationships are of the form $A \ VP B$. A 1:N relationship requires that there be one to many values of $B$ for each value of $A$, i.e. the min/max values for $A$ are (1,N).

A situation where a violation of 4NF exists would require, for example, that two independent relationships $A \ VP_{1} B_{1}$ and $A \ VP_{2} B_{2}$, with the min/max values of $A$ (1,N) be represented by the same relation. Following the rules outlined in Mapping Ratios, if the min/max values for $B_{1}$ and $B_{2}$ are (0,1), (0,N) or (1,N) two separate relations $A \ VP_{1} B_{1}$ and $A \ VP_{2} B_{2}$ will be constructed to represent the relationships. If the min/max values of $B_{1}$ and $B_{2}$ are both (1,1), then the relationships will be represented by either adding the key attributes of $A$ as non-key attributes of $B_{1}$ and $B_{2}$, respectively, or by constructing relations as in the case when the min/max values for $B_{1}$ and $B_{2}$ are (0,1), (0,N) or (1,N). (The choice of representation depends on the types of queries that are of interest to the user.) Situations where the min/max values of the $B$ entity in the relationship are (1,N) are handled analogously. Hence, the system will not represent two such independent relationships by
one relation. Thus, a violation of 4NF will not occur as a result of invoking the rules for relationship representation.

As will be discussed in Chapter VIII, the View Creation System can model a ternary relationship if the user provides an all-encompassing entity name which corresponds to the concept the user is trying to capture. To illustrate, suppose the user provides an entity name \( E_x \) and indicates that \( E_x \) is uniquely identified by three entities, \( E_1 \), \( E_2 \), and \( E_3 \). Eventually, this entity will be represented by a relation:

\[
E_x : [ E_1\text{-KEY}, E_2\text{-KEY}, E_3\text{-KEY}, \ldots ]
\]

If the three entities \( E_1 \), \( E_2 \), and \( E_3 \) are interdependent, the user has correctly specified the concept he wishes to model. However, if the three entities are independent then a violation of 4NF occurs (because two or more independent relationships are represented in the same relation). Therefore, the only situation where a violation of 4NF can occur is when the user assumes, and tries to model, a ternary relationship when in fact there is none.

Thus, it seems very unlikely that a violation of 4NF would occur in a set of relations produced by the View Creation System.

5NF

Fifth Normal Form deals with cases where information can be reconstructed from smaller pieces of information which can be maintained with less redundancy [Kent, 1983]. Since Fifth Normal Form is highly concerned with the semantics of the data and since the system does not attempt to deal with this form of normalization, it will not be discussed further.

The above remarks have relied heavily on the given assumptions and have tried to demonstrate that the View Creation System is capable of producing a normalized set of
relations which correctly represents a user's view. Obviously, the assumptions may not always hold. The system has some mechanisms for dealing with situations where the original assumptions may not be applicable.

**System's Performance Under Relaxed Assumptions**

Assume that the user is not always able to model his application in terms of entities, relationships and attributes or that the user does not completely understand either his application or the system's instructions. In these situations, the system strives to aid the user in determining the exact nature of his information requirements. The procedures that the system follows concentrate on identifying places where requirements are missing, detecting and resolving ambiguities and inconsistencies in the user's input, and relying on the system's defaults when the user is not able to provide the requested input. Although these procedures have just been discussed, they are categorized again below to highlight how they provide the system with some of its "intelligence".

**Entities, Relationships and Attributes**

The system aids the user in establishing whether something should be an entity, a relationship between two entities, or an attribute. It tries to ensure that each entity is related to at least one other entity. For certain types of relationships, the system makes inferences about the mapping ratios and how the relationships should be represented. (Refer to the sections *Entities and Relationships* for more details.)

1. When obtaining the user's initial set of entities, the system checks for attribute names which are entity names and vice versa. If either of these occur, the system verifies that the user has not made an input error and makes inferences about what relationships should exist in the database.

2. The system asks the user to identify multivalued attributes. A multivalued attribute is converted to an entity and an the system infers that a relationship will be required between the original entity where the multivalued attribute occurred and the new
entity.

3. Based on a minimal amount of general knowledge, the system is sometimes able to suggest which attributes the user should consider when providing candidate keys.

4. For relationships, the system infers some of the mapping ratios by examining the given form (singular or plural) of the involved entities.

5. After an initial set of relationships has been obtained, the system asks the user to provide relationships for those entities which do not appear in any of the existing ones.

6. The system prompts the user for relationships which should exist between entities based on the inferences it made when obtaining the original set of entities.

**Missing Information**

Along with helping the user categorize information requirements, the system tries to discover where the user has failed to provide necessary requirements.

1. The system attempts to identify missing entities by examining attribute names whose forms suggest that they may be referring to (unidentified) entities.

2. When the user provides a relationship $A \ VP \ B$ where either the $A$ or $B$ values have not been previously identified as entities or attributes, the system asks for the appropriate identification. For missing entities identified this way, the system obtains corresponding attributes and candidate keys.

**System Defaults**

The system has a number of defaults which it invokes when the user is unable to respond to a given question.

1. If the user is not able to provide an attribute for an entity, the system generates one which is the name of the entity suffixed with $id$.

2. The default for a candidate key is all the attributes of an entity.

3. When the user does not know what type of query is of interest when the system is trying to establish how a relationship should be represented, the relationship is represented by a relation.
Detection of Ambiguities

The system also attempts to detect ambiguities in the user’s input.

1. In a relationship A have/has B, the have/has is ambiguous. The user is asked if: 1) A possesses B; 2) B is a component of A; 3) B is an instance or an example of A; or 4) B is associated with A.

2. The system highlights entities and relationships which may be synonyms and asks the user to indicate whether or not they are.

Consistency Checks

The system tries to detect and resolve inconsistencies in the user’s input.

1. If an entity does not appear in any relationship (and when prompted for a relationship involving it, the user was unable to provide one), the system asks whether or not the entity should exist and deletes or retains it as indicated by the user’s response.

2. In a relationship A is-a/instance-of B, A is a subset entity while B is a superset entity. The system identifies entities which occur in other relationships involving the superset entity B. It then asks the user if the information requirements would be represented at a more appropriate level of detail if the other entities occurred in conjunction with the subset entity A instead of the superset entity or whether both types of relationships are required.

3. For a relationship A is-a/instance-of B, the system verifies that attributes which correspond to the superset entity B are also valid attributes of the subset entity A.

4. If, during a modification procedure, the user deletes an attribute of an entity which is part of a candidate key, the system asks the user to identify candidate keys again.

5. During the modification modules, the user is allowed to add relationship attributes. If the user tries to add attributes of a relationship which, logically, should not have attributes, the system informs the user that he has really provided entity attributes and asks the user to identify which entity the attributes correspond to.
VI. SYSTEM TESTING

As previously stated, the major reason for testing the system is to prove that the knowledge base is an adequate formalization of the view specification task. In the testing phase, the system was used to create user views and the resulting output – sets of relations representing both the entities and the relationships of the users’ applications – examined. When the system failed to perform as it should, an attempt was made to detect where the system’s knowledge was lacking and then codify the missing knowledge into the system.

TESTING CRITERIA

The following is a list of criteria used to judge whether or not the knowledge base formalization is both correct and "good".

A set of relations produced would not be correct if any of the following occur:

1. one of more relations violate normalization principles
2. there is an inappropriate division of entities, attributes, and relationships; e.g. an attribute appears in an entity relation which should be a separate entity (or vice versa).
3. relevant information about the application is missing.

When the output is correct in the above sense, an evaluation was made of whether or not it is "good" – i.e. whether the set of relations is free of undesirable properties such as:

1. extraneous or redundant relations.
2. extraneous or redundant attributes in relations. The occurrence of redundant attributes could, for example, indicate that a relationship which states that one entity is a subtype of another is missing. If this is so, attributes of the superset entity would also appear (redundantly) as attributes of the subset entity.

E.g. Suppose the following relations are obtained:

Person: \([SIN', person\_name, person\_address]\)

\(^{\text{SIN is an abbreviation for Social Insurance Number which is the Canadian equivalent of a Social Security Number.}}\)
Employee: [EMPLOYEE_NO, sin, employee_name, employee_address]

If the relationship employee is a person is known, it should be possible for the subset entity, employee, to "inherit" the attributes of the superset entity person. Then the more desirable set of relations would be produced:

Person: [SIN, person_name, person_address]
Employee: [EMPLOYEE_NO, sin]

3. relations are "over-normalized" resulting in many simple relations instead of one all-encompassing relation.

E.g. the entity book is represented as:
book: [CATALOG_NO, title]
book: [CATALOG_NO, publisher]
instead of:
book: [CATALOG_NO, title, publisher].

Such representations would lead to inefficiencies in query processing because extraneous join operations would be required.

4. "poor" choices of primary keys resulting in inefficient joins when processing queries. Joins on key attributes presumably are more efficient than joins on non-key attributes.

Note that some of these features are in conflict with each other. Although one of the criteria for a correct relation is normalization, a human designer might choose an unnormalized form for efficiency reasons. Then the system could only provide a suggested set of relations which a human may choose to modify.

TESTING SESSIONS

The system was tested for individual user views in different application domains by two types of users:
1. users who had some familiarity with database design concepts
2. naive users who had no prior exposure to database design.

The test sessions are summarized in Table III with further details provided in Appendix 4.
### TABLE III: Test Sessions

<table>
<thead>
<tr>
<th>USER &amp; APPLICATION</th>
<th>EVALUATION</th>
<th>SYSTEM MODIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User#1</strong>&lt;br&gt;Type: Systems Analyst&lt;br&gt;Application: Training Database</td>
<td>• The view contains one redundant relation and one relation where the information requirements are not represented at the correct level of detail.</td>
<td>• Two rules were added: 1) a new relation should not be constructed to represent an instance of relationship; and 2) for relationships A instance of B, determine whether any other entity appearing in a relationship with B should be associated with A instead.</td>
</tr>
<tr>
<td><strong>User#2</strong>&lt;br&gt;Type: Some familiarity with database concepts&lt;br&gt;Application: Student-Advisory Database</td>
<td>• The output is a normalized set of relations but does not totally represent the user's application because of the difficulty the user had when identifying entities and attributes.</td>
<td>• The system's instructions were modified to highlight some of the more subtle points.</td>
</tr>
<tr>
<td><strong>User#3</strong>&lt;br&gt;Type: Knowledgeable in data modelling&lt;br&gt;Application: Teaching Database</td>
<td>• The output is correct and free of any undesirable properties.</td>
<td>• Two minor modifications to the user interface were made based on the User#3's suggestions.</td>
</tr>
<tr>
<td><strong>User#4</strong>&lt;br&gt;Type: Naive User&lt;br&gt;Application: Origin-Destination database for movement of traffic</td>
<td>• The view produced is small but correct and free of any undesirable properties.</td>
<td>• No modification.</td>
</tr>
<tr>
<td><strong>User#5</strong>&lt;br&gt;Type: Naive User&lt;br&gt;Application: Equipment Database</td>
<td>• The view is correct and free of undesirable properties but does not reflect all of the user's requirements because the user failed to model one dimension of his application.</td>
<td>• A rule was added which allows one to distinguish what role subset and superset entities of is_a relationships play in other relationships when they both have the same primary keys.</td>
</tr>
<tr>
<td><strong>User#6 &amp; User#7</strong>&lt;br&gt;Type: Naive users who designed a single view.&lt;br&gt;Application: Database for insurance claims</td>
<td>• The view does not represent the users' information requirements because of the difficulty the users had in identifying entities in their application.</td>
<td>• No modification.</td>
</tr>
</tbody>
</table>
TABLE III (Cont'd): Test Sessions

| User#8 | Type: Learning database design | Application: Database for Program Specification Blocks. | The view appropriately reflects User#8's information requirements and is free of undesirable properties. | One rule was added. When a relationship $A$ have/has $B$ (with attributes) is converted to an entity $A$ have/has $B$ and a shorter name is not provided, the entity name should later be modified to reflect the appropriate interpretation of have/has. |

DISCUSSION

Users and Applications

Of the eight users who tested the system, four were naive users (Users 4, 5, 6 and 7) and four had some knowledge of database design concepts. The amount of knowledge the latter group had ranged from just beginning to learn about database design (User#8) to having done extensive work in data modelling (User#3). The design session covered a wide range of application areas.

Output and System Modification

Not surprisingly, the system performed best for users who knew something about database design. For these users, the design produced was either an accurate representation of the user's information requirements or highlighted a missing rule(s) in the system's knowledge base which was subsequently added.

Of the four sessions with naive users, one resulted in a good design (User#5), but failed to capture all of the user's requirements because the user omitted one dimension of his problem. For the other three sessions (User#3, User#6 and User#7), the major problem was
that the users found it difficult to identify and distinguish the entities and attributes of their applications.

In all cases, when an undesirable output was obtained, it was possible to detect where the system's knowledge was lacking and then codify the missing knowledge into the system. The system modifications were done immediately after each testing session so that the version given to the succeeding user included that knowledge.

Even in the final testing session, a situation occurred where a new rule was required. One potential criticism of the system is that it has not reached some form of a "steady state" where no further improvements are necessary. However, no major modifications occurred after the testing session with User#5. The changes (if any) to the system from the latter testing sessions were all refinements to the knowledge base. Using the system without these improvements did not invalidate the output. Rather, the refinements aided in capturing more of the semantics of the application. Therefore, the system did reach a reasonable degree of stability, at least as far as the major rules of the knowledge base are concerned.

User's Comments and Suggestions for Improvements to the System

Most of the user's comments and suggestions for improvements to the system have been documented in Appendix 4. The suggestions for improvements were usually incorporated if they pertained to the user interface. Some of the comments and suggestions were:

1. The system should be augmented with a tutorial (User#3 and User#4).

2. A rudimentary data dictionary, which would include a one or two sentence description of each entity, would help ensure that the user consistently uses the same terminology (User#3).

3. The system could be tedious to use for someone who knows a great deal about database design.

4. The system proved to have educational value for most of the users.
a. User#1 commented that it was beneficial because the queries posed by the system made him think very carefully about his database requirements.
b. Using the system helped User#4 understand and structure his application.
c. User#5, User#6 and User#7 wanted to try the system again because they thought they could more accurately capture their requirements a second time based on what they had learned the first time.
d. User#8, who was just beginning to learn about database design, expressed a desire to use the system for her next project.

CONCLUSIONS

Based on the above discussion, the following conclusions can be drawn.

Both classes of subjects – naive users and those with some knowledge of database design concepts – were able to successfully create views using the View Creation System. The two groups also found use of the system to be valuable in increasing their understanding of the database design process. Database designers with relatively little experience in the field expressed appreciation for the expert consultative role that the system played.

As previously discussed, the Entity-Relationship Model was used as the basis for the View Creation System because of claims in the literature that the basic concepts of this model provide a very natural way of representing a user’s information requirements. The testing sessions cast some doubt on the validity of these claims. Some naive users were unable to clearly identify and distinguish the entities and attributes of their applications. The system was able to aid the users in these tasks to a certain extent (e.g. User#1 and User#5) but unable to do so all of the time (e.g. User#6 and User#7). Augmenting the user interface to use natural language and/or graphics may assist in overcoming this difficulty. (These issues are peripheral to this research.)

Human designers consulted during this research expressed the view that it is both feasible and valuable to take some time at the beginning of a design project to educate the
users in relevant database concepts. The current View Creation System includes very minimal tutorial facilities. An expansion to this part of the system might eliminate part of the problem discussed above.
VII. SUMMARY AND CONTRIBUTION

SUMMARY

Recall the original research question: *Can the process of generating user views be formalized and subsequently automated in order to increase the quality and consistency of database design while relying less on scarce expert human skills?* This question has been addressed by actually formalizing, as a set of rules, a process for generating user views. The rule set, and hence the formalization, is found in the knowledge base of an expert View Creation System.

The View Creation System is based on the Entity-Relationship model and its later refinements incorporating greater semantic content. The system interacts with a user to determine information requirements. It guides the user in stating these requirements while trying to detect and resolve inconsistencies and ambiguities in the user’s input. Eventually, a set of Fourth Normal Form relations is produced which represents the user’s view.

The development of the system was an iterative process. It began with the implementation of a prototype based on knowledge about the database design process in general, and in particular, how the Entity-Relationship model can be used in modelling user views. The knowledge base was then augmented to include expertise from human database design experts and further refined through testing the system in various application domains.

As demonstrated both from a theoretical discussion of the system’s correctness and the experimental testing of the system, the answer to the research question is "yes".

CONTRIBUTION

The scholarly significance of this research lies in the insight it provides into the process of
database design. Specifically, a methodology for the part of logical database design which deals with the generation of user views has been formalized and expressed as the knowledge base of the View Creation System. In addition to providing a means for precisely expressing this formalization, implementation of the expert system has made it possible to experimentally validate its adequacy and completeness in the representation of the view specification task. However, the primary contribution of the research lies in the rules and procedures which comprise the system's knowledge base.

This research also makes a contribution to the domain of applied expert systems. Most of the well-known expert systems dealt with in the literature review have focused on diagnositic problems where the objective of the system is to work from given facts to determine their logical consequences. The View Creation System, however, addresses a design problem. In this class of problems, an overall objective is known, but there is, in general, not a well-specified starting point. This has required a somewhat different approach than that typically followed in diagnostic systems including, in particular, the introduction of an element of procedurality into the structure of the system.

PRACTICAL SIGNIFICANCE

The practical significance of the research is that the system has demonstrated the feasibility of alleviating a critical bottleneck in database technology. It has done so by putting some structure into a mixture of rules for view creation which were obtained from experience, database design techniques and semantic data models. The View Creation System, equipped with a more functional user interface, could provide the basis for the development of a commercially useful system. Then the user, or system analyst, who has a direct and comprehensive understanding of his application, could become the designer of his own database view. The system could also act as a consultant or trainer for relatively inexperienced database designers. However, it is not clear that the current version of the
View Creation System could fully replace a human designer when the end-user has no familiarity with database concepts.

Knowledgeable database people who participated in the testing of the system suggested the following additional applications:

1. The system could be used as an educational tool for training people in database design.

2. The system could be used by different users within the same application domain to jointly design a communal view. A somewhat extended version of the system could assist in identifying and resolving inconsistencies.

3. The system could be integrated with an organization's data dictionary to assist in creating user views into an already existing database. In this context, the system would attempt to relate the user's requirements to the description of the available data.

4. After information requirements have been obtained from one user, a second user in the same application domain could be allowed to build on the first user's view. This could shorten the time involved in obtaining information requirements as well as improve the quality of the resulting database.
VIII. EXTENSIONS TO THE RESEARCH

This section discusses possible extensions to the research as they relate to the user interface, the knowledge base and the system users.

USER INTERFACE

Since the contribution of the research lies in the formalization of the process of generating user views, the user interface was not given a high priority during the development of the View Creation System. There are many ways in which the interface would need to be improved in order to obtain a commercially successful version of the system. Some suggestions for improvements are outlined below.

Graphic Capabilities

A graphic interface would be a very desirable addition. Human database designers use diagramming techniques both for structuring and representing problems and as concise and effective tools for communicating with users. The advantage to the user is that he is able to observe all aspects of the design problem at once. Most formal methodologies for designing databases use graphic constructs. During the consultation sessions with human experts, the designers tended to use a variation of the Entity-Relationship diagram as their diagramming technique. Hence, if the View Creation System is to function as a human expert, it should also include graphics capabilities.

Menu-driven Responses

Another addition to the interface could be a menu-selection function. This would allow the system to automatically generate a set of alternatives from which the user could choose relevant information. This would decrease the number of user entries and consequently the length of the design sessions.
The expert designers often generated attributes for some of the entities based on their own knowledge of the application. For example, attributes such as *name*, *address*, *phone*, etc. were frequently generated for entities which referred to people while attributes such as *id*, *code*, etc. were generated as identifiers for other entities. The designers agreed that the generated attributes would require verification from the user before they could be incorporated into the design. The addition of menu-driven responses would become even more desirable if the system's knowledge base was augmented to include this type of real world knowledge. Then, as the human designers, the system could generate alternatives for the user to consider. Verification by the user would be facilitated using menus as the user could simply select the appropriate options from those suggested by the system.

The user is currently required to do a considerable amount of data entry. Menu-driven responses would decrease the amount of data-entry required as would modifications to the interface which would allow the user to refer to entities, relationships, etc., by a single number or letter instead of entering the complete name. For example, when entities or relationships which are synonyms are identified, the system asks which one should be used as the replacement term. The user responds by entering the appropriate name. (See examples in Chapter V or *Knowledge Base – Synonyms.*) The entry process could be facilitated if the user was required to indicate only a corresponding number or letter as an identifier.

E.g. Are any of the following different ways of saying the same thing?  
1. library lends volumes  
2. volumes on _loan_ from library  
("yes." / "no." / "why.")  
|: yes.

Which one would you like to use?  
("1." / "2." / "no." if you do not know)  
|: . .  

*Instead of re-entering the complete relationship, the user could enter the*
corresponding number.

Condensed Dialogue

A potential problem with the system is that, given its current lack of general knowledge, it may ask the user a large number of seemingly trivial questions about the application domain. As the user becomes familiar with the system's procedure, some of the system's questions could be bypassed or the responses could be entered in an abbreviated form (i.e. without going through all of the system's prompts).

E.g.:
Enter next entity / "see." / "end."
book: [CATALOG_NO, title, . . . ]
(Instead of waiting for the system's prompts, the user could enter a new entity along with its attributes and candidate keys.)

However, it does not seem desirable to permit a large-scale batch input mode because in the testing sessions even experienced designers felt that they benefited from the interactive dialogue with the system.

Natural Language

More natural language features could also be included.

1. The View Creation System recognizes simple plural forms and different tenses of verbs. This part of the program could be supplemented to allow the system to recognize all plural forms of entities and more complicated verb structures. Then the system would not have to verify with the user whether, for example, an entity name appears in its singular or its plural form. This extension could also provide a higher level of confidence in the inferences the system makes about mapping ratios.

2. Greater natural language capabilities could ensure that most of the questions the system asks are grammatically correct.
3. Enhancing the natural language capabilities might allow the system to use less
database design terminology which may be beneficial.

NON-BINARY RELATIONSHIPS

Currently the system accepts only binary relationships. As demonstrated during the testing
of the View Creation System, many real-world applications can be modelled this way.
However, not all relationships are binary.

The existing system allows the user to provide ternary relationships but doing so
requires considerable insight on the part of the user. A ternary relationship can be
represented by creating an entity whose key is the concatenation of the keys of the three
entities involved and whose non-keys correspond to the relationship's attributes. For
example, a discussion with User#3 indicated that the user might be interested in
representing a ternary relationship among the entities student, course and semester, namely,
that a student takes a course during a given semester and receives a corresponding grade.
This is handled in the system by creating an entity, student_course, whose initial attributes
are student, course, semester and grade as illustrated below. Eventually the entity will be
represented as a relation (i.e. an entity relation) in the final output.

E.g.
Create the entity:
student_course: [STUDENT, COURSE, SEMESTER, grade]

along with the entities:
Student: [STUDENT_ID, .... ]
Course: [COURSE_NUMBER, .... ]
Semester: [START_DATE, END_DATE, ... ]

As discussed in Chapter V, the system has mechanisms for dealing with a key
attribute of one entity which is the name of another entity. It does so by replacing the key
attributes with the primary keys of their corresponding entities after primary keys have
been established. (See Chapter V or Knowledge Base – Primary Keys.) In the above example, 
the representation in the final output would be:

```
student_course:
[STUDENT_ID, COURSE_NUMBER, START_DATE, END_DATE, grade]
```

In an extension to the system, the restriction of binary relationships should be 
relaxed to allow the user to directly enter non-binary relationships such as the one just 
described.

_E.g._

Enter next relationship / "see" . . .
|: students take courses during semesters.

In database design terminology, the above situation is known as the _connection trap_ 
problem. If the system was modified to deal with this problem (i.e. allow non-binary 
relationships), there would be less responsibility placed on the user when describing this type 
of relationship.

**CONSISTENCY CHECKS**

The knowledge base could be extended to include more rules for checking the consistency of 
the user’s input. One such extension involves using the mapping ratios to verify that the 
appropriate type of _have/has_ relationship has been identified or that possible synonyms have 
been detected. For example, if the user indicates that a relationship _A has B_ can be 
classified as _B instance_of A_, then the system could ascertain that this is correct by 
verifying that the min/max values for _B_ and _A_ are (1,1) and (1,N), respectively. (See 
_Mapping Ratios._)

As a second example, suppose the user identifies the entities _E_ \_1 and _E_ \_2 as synonyms 
because they occur in relationships of the form: _E_ \_1 VP _B_ and _E_ \_2 VP _B_. Then, as a
consistency check, the system could verify that the mapping ratios for $E_1$ and $E_2$ are the same. The system might also be able to identify similarities in entities by noting various patterns in the entities' attributes and the relationships in which they appear.

**ADDITION OF GENERAL KNOWLEDGE**

Another way to make the system behave more like a human designer is to include additional general knowledge which would increase the number of inferences that the system is capable of making. For example, if the system could recognize certain entities such as *employee*, *person*, *student*, *manager*, etc. as people, it could then infer that attributes such as *name*, *address*, etc. would be appropriate for them. As discussed earlier in conjunction with menu-driven responses, the system would only have to verify with the user that these attributes are appropriate instead of requiring that the user provide them.

**MULTIPLE USERS**

In real-world situations, there may be a number of users whose jobs are similar and who therefore share the same database view. Given resource limitations, it is likely that only one, or a very small number, is interviewed by a human database designer. With a computerized "designer", the resource constraint would disappear, and it should be possible to incorporate information from multiple users in the definition of a single view.

The inclusion of input from multiple sources would require that conflicts be resolved and choices be made among alternatives as the need arises. This could be a complex issue, which would require rules for dealing with many different possible scenarios. Therefore, the system should first be tested to determine whether or not the extension to incorporate multiple users is worthwhile. To do this, a group of homogeneous users could be selected and given a fundamental problem in their application domain. Each member of the group would be asked to use the View Creation System to design a user view. After completing the
design sessions, the resulting sets of relations would be compared.

If this experiment showed that the outputs are similar for all users, then it is likely that the system should work for any particular user. In that case, there would be no advantage in modifying the system to enable it to amalgamate information from multiple users. However, if the outputs differ substantially, then provisions should be made for incorporating information from multiple users when designing a single, communal view.

At the very least, the system could provide an impartial tool for experimentally testing whether or not this type of research should be undertaken. A human expert, on the other hand, could not provide such an objective measure because a human is capable of learning and remembering facts about an application as he moves from one design session to another.

INTEGRATION OF USER VIEWS

Each view that is produced by the View Creation System is a partial specification of the database that would be required by all the users in a specific application domain. A further modification of the system involves integrating the views of individual users to obtain a complete database design. Since algorithms already exist for the integration of a set of user views, this extension requires the addition of such an algorithm (e.g. Navathe and Elmasri, 1986).

GENERATION OF A DATABASE

An obvious extension to the existing system is to generate, as a Prolog program, an actual database. This would entail representing the database in Prolog and then populating it with actual data which would appear as a set of assertions. Integrity constraints would have to be obtained. For example, ranges of permissible values and the abstract data types of the
attributes would need to be established in consultation with the user. The addition of such
contraints is a major research topic in itself. Finally, a data dictionary would also have to be
created.

USER EDUCATION

One alternative to using the system as it is, or possibly with some extensions, is to educate
users in database design concepts, employing the Entity-Relationship constructs, before
exposing them to a design session using the system. The system itself could be augmented
to include a more extensive tutorial. Training a user before the design session is not unlike
something that a human designer would do. In fact, two of the designers of the consultation
sessions make explicit reference to teaching users the fundamentals of their approaches
before beginning to create user views. (See Appendix 3 – Designer#1 and Designer#4.) One
of these designers worked only with systems analysts while the other worked with both
systems analysts and (intelligent) end-users (e.g. engineers).
REFERENCES


APPENDIX 1 – KNOWLEDGE BASE

USER INTERFACE

1. Input can be given in upper or lower case letters. Upper case letters are converted to lower case to ensure that the internal representation is consistent.

2. Abbreviations of the form: "y"/"ok" for "yes", "n" for "no", "w" for "why", etc. are accepted.

3. Entity names are considered equivalent if they differ only in their plural forms (e.g. "s", "es" or "ies" appended to the same stem).

4. Simple verbs are considered equivalent if they differ only in tense (e.g. "s", "ed", or "ing" appended to the same stem).

VIEW CREATION

5. IF: *
   a. Start Session
   b. Entities
   c. Relationships
   d. Modifications – First
   e. Synonyms
   f. Inherited Attributes
   g. Modifications – Last
   h. Primary Keys
   i. Represent Entities and Relationships
   j. Functional Dependencies
   k. End Session

   THEN: a user view is created.

START SESSION

6. Instruct the user re: types of acceptable responses.
7. Obtain user’s id (e.g. user’s name).

* This rule controls the execution of the procedure for the design session.
ENTITIES

Procedure

8. Ask the user to provide entity names in their singular form.

9. For each entity:
   a. obtain attributes
   b. check for attribute names which should be entity names and vice versa
   c. convert multivalued attributes to entities
   d. identify candidate keys.

10. Check for missing entities.

Attributes

11. **IF**: the user cannot provide any attributes for an entity;
    **THEN**: generate an attribute entity-name_id.

Attribute Name is Entity Name

12. **IF**: the user provides an attribute Term for a new entity $E_{New}$ where Term is the name of an existing entity $E_{Prev}$ (or Term is of the form $E_{Prev_{-}Suffix}$ where Suffix is one of the "key indicator attributes" (see Candidate Keys));
    **THEN**: ask if $E_{Prev}$ will be needed to uniquely identify an instance of the entity $E_{New}$.

    **IF**: $E_{Prev}$ will be needed to uniquely identify an instance of $E_{New}$;
    **THEN**: accept the attribute Term and add an assertion (to the system’s database) which indicates that the key of $E_{Prev}$ may be needed to uniquely identify an instance of $E_{New}$. This assertion will be used later to either:

1) replace the attribute Term in the primary key of $E_{New}$ with the key attributes of $E_{Prev}$ (see Primary Keys); or
2) prompt the user for a relationship between $E_{New}$ and $E_{Prev}$ (see Relationships - Prompts for Missing Relationships and Primary Keys).

    **IF**: $E_{Prev}$ will not be needed to uniquely identify an instance of $E_{New}$;
THEN:
1) Do not include the attribute.
2) Inform the user that he is implying the existence of a relationship between $E_{New}$ and $E_{Prev}$ and that he will have an opportunity to provide the relationship at a later point.
3) Add an assertion which indicates that a relationship should exist between $E_{New}$ and $E_{Prev}$. This assertion will be used at a later point to prompt the user for a relationship if one is not voluntarily provided. (See Relationships – Prompts for Missing Relationships.)

E.g.
IF: the entity library already exists and, when adding the entity branch, the user provides library (or library_id, etc.) as an attribute;
THEN: ask if library will be needed to uniquely identify an instance of branch.

Entity Name was Attribute Name

13. IF: the user provides, as the name of a new entity $E_{New}$, some form of the name of an attribute $Att_{Prev,j}$ of another entity $E_{Prev}$;
THEN: add an assertion which indicates that $E_{New}$ may be needed to uniquely identify $E_{Prev}$. This assertion will be used at a later point to:

1) replace the attribute $Att_{Prev,j}$ in the primary key of $E_{Prev}$ with the primary key of $E_{New}$ (see Relationships – Prompts for Missing Relationships); or
2) prompt the user for a relationship between these two entities if one is not voluntarily provided. (See Relationships – Prompts for Missing Relationships and Primary Keys.)

E.g.
IF: the entity branch already exists and has an attribute library (or library_id, etc.) and the user adds the entity library;
THEN: add an assertion which indicates that library may be needed to uniquely identify an instance of branch.

Repeating Attributes

14. IF: the user adds multiple attributes of the form Term$_N$ where $N$ is numeric and Term is constant for an entity Ent;
THEN: ask if the user has entered multiple instances of the same attribute.

IF: multiple instances of the same attribute have been provided;
THEN:
1) Delete the attributes from the entity.
2) Make Term an entity (obtain corresponding attributes and candidate keys).
3) Add an assertion which indicates that a relationship should exist between the entity Ent and the new entity Term. This assertion will be used at a later point to prompt the user for a relationship between these two entities if one is not voluntarily provided. (See Relationships – Prompts for Missing Relationships.)

IF: the user has not entered multiple instance of the same attribute;
THEN: do nothing.

E.g.
IF: the user provides the following attributes for the entity student:
student [. . . course_1, course_2, course_3, . . . ];
THEN: ask if the user has really entered multiple occurrences of course.
If the user agrees that this is the case, delete the attributes course_1, course_2 and course_3 and add the entity course. Add an assertion to the system’s database which indicates that a relationship will be needed between student and course.

Attributes with Multiple Values

15. After obtaining the attributes for an entity, ask the user to identify any attributes which could have more than one value for each occurrence of the entity.

16. IF: an attribute can have more than one value for each occurrence of an entity;
THEN:
  a. Delete the attribute from the entity.
  b. Convert the attribute to an entity. (Obtain corresponding attributes and candidate keys.)
  c. Add an assertion which indicates that a relationship will be needed between the new entity and the original entity. This assertion will be used at a later point to prompt the user for a relationship if one is not voluntarily provided. (See Relationships – Prompts for Missing Relationships.)
Candidate Keys

17. "Key indicator attributes" are: i) id ii) name iii) number iv) no and v) code.

18. **IF:** attributes are provided for an entity $Ent$ which are "key indicator attributes" or of the form $Ent_X$ where $X$ is one of the "key indicator attributes"; **THEN:** suggest the user consider these as possible candidate keys.

19. **IF:** an attribute appears as part of a candidate key at least twice (for two different entities); **THEN:** add it to the list of "key indicator attributes".

20. **IF:** the user provides a candidate key which contains all the attributes of a previous candidate key (plus other attributes); **THEN:** ask the user which set of attributes should be a candidate key.
   **IF:** the user does not know; **THEN:** make the larger set of attributes the candidate key and delete any other candidate keys which are contained in the new (larger) candidate key.

   *E.g.* (UPPER CASE indicates candidate key attributes)
   New Candidate Key: person [NAME, ADDRESS, . . . ]
   Previous Candidate Key: person [NAME, address, . . . ]
   Ask the user whether i) NAME and ADDRESS or ii) NAME should be the candidate key. If the user does not know, make [NAME, ADDRESS] the new candidate key and delete the candidate key [NAME].

21. **IF:** the user provides, as a candidate key, a set of attributes which are part of another candidate key; **THEN:** ask which set of attributes should serve as a candidate key.
   **IF:** the user does not know; **THEN:** retain the larger set of attributes as the candidate key.

   *E.g.*
   New Candidate Key: person [NAME, . . . ]
   Previous Candidate Key: person [NAME, ADDRESS, . . . ]
   Ask the user whether i) NAME or ii) NAME and ADDRESS should be the candidate key. If the user does not know, retain [NAME, ADDRESS] as a candidate key and delete the candidate key [NAME].

22. **IF:** the user does not know which attribute(s) uniquely identify an instance of an entity; **THEN:** construct a candidate key which consists of all the entity's attributes.
Adjust Attributes

23. **IF:** an attribute name occurs for more than one entity;  
**THEN:** for each occurrence of the attribute, prefix the attribute name with its corresponding entity name.  
*E.g.*  
library [. . . address, . . .]  
borrower [. . . address, . . .]  
*Becomes:*  
library [. . . library_address, . . .]  
borrower [. . . borrower_address, . . .]

24. **IF:** an attribute $Att$ of entity $Ent$ is one of the "key indicator attributes";  
**THEN:** prefix the attribute by its entity name (i.e. convert $Att$ to $Ent_{-}Att$).

Entity Modification

25. After obtaining the initial set of entities, allow the user to:  
a. add, delete or rename entities  
b. add, delete or rename attributes  
   **IF:** an attribute is deleted which is part of a candidate key;  
   **THEN:** ask the user to identify candidate keys again.

MISSING ENTITIES

Let the set of all entities be represented by $E = \{E_i\}$ and the set of attributes by $A = \{A_{ij}\}$ where $A_{ij}$ is the jth attribute of the ith entity.

26. **IF:** entity $E_i$ has an attribute $A_{ij}$ which is of the form $Term_{-}suffix$ where $suffix$ is one of the "key indicator attributes" (e.g. id, name, number, etc.) and $Term$ is not an entity;  
**THEN:** ask if $Term$ should be an entity;  
   **IF:** $Term$ should be an entity  
   **THEN:**  
   1) Obtain its corresponding attributes.  
   2) Add an assertion which indicates that a relationship will be required between $E_i$ and $Term$.  
   3) Add an assertion which indicates that the attribute $A_{ij}$ may later be deleted. (See Relationships – Prompts for Missing Relationships.)
4) Scan the attributes of each entity.

**IF:** entity $E_k$ has an attribute $A_{kl}$ ($A_{kl} \neq A_{ij}$) which is also of the form Term_suffix but $A_{kl}$ is not an attribute of the new entity, Term;

**THEN:** ask if the user wishes to include attribute $A_{kl}$ as an attribute of Term.

**IF:** the user wishes to include $A_{kl}$ as an attribute of Term;

**THEN:**

a) include the attribute and add an assertion which indicates that $A_{kl}$ may later be deleted. (See Relationships - Prompts for Missing Relationships.)

b) add an assertion which indicates that a relationship may be required between the entities $E_k$ and Term.

**IF:** the user does not wish to include $A_{kl}$ as an attribute of Term;

**THEN:** do nothing.

**IF:** entity $E_k$ has an attribute $A_{kl}$ ($A_{kl} \neq A_{ij}$) which is also of the form Term_suffix and $A_{kl}$ is an attribute of the new entity, Term;

**THEN:**

a) Add an assertion which indicates that a relationship may be required between the entities $E_k$ and Term. (See Relationships - Prompts for Missing Relationships.)

b) Add an assertion which indicates that attribute $A_{kl}$ may later be deleted. (See Relationships - Prompts for Missing Relationships.)

5) Obtain candidate keys for Term.

**RELATIONSHIPS**

**Procedure**

27. Instruct the user to provide relationships of the form $A \ VP \ B$ where $A$ and $B$ are entities or attributes and VP stands for "verb phrase". Verb phrases may be multiple words concatenated, e.g. is_part_of.

28. For each relationship:

a. Determine if the $A$ and $B$ values are appropriately identified. (See Unidentified
A's and B's.)
b. Obtain mapping ratios – min/max values. (See Mapping Ratios.)
c. If appropriate, obtain relationship attributes. (See Relationship Attributes.)

29. Prompt the user for "missing relationships" based on previous input. (See Prompts for Missing Relationships.)

30. Allow the user to modify the relationships. (See Relationship Modification.)

31. For relationships of the form A have/has B, obtain an appropriate interpretation of have/has. (See Have/has Relationships.)

Special Verb Phrases

32. is__a verb phrases are: is, are, and is__a.
33. have/has verb phrases are: have and has.
34. instance verb phrases are: instance_of, example_of, and subset_of.
35. component_of verb phrases are: component_of, and part_of.
36. possesses verb phrases are: possesses and owns.

Relationship Attributes

37. IF: the min/max values of A are not (1,1) and the min/max values of B are not (1,1) (see Mapping Ratios);
THEN: ask the user to identify corresponding relationship attributes.

38. IF: the user provides a relationship attribute $R_{att}$ for a relationship $A \text{ VP } B$ where $R_{att}$ is an attribute of either entity A or entity B;
THEN: ask whether $R_{att}$ should be an attribute of the entity or the relationship.

  a. IF: $R_{att}$ should be an attribute of the entity;
   THEN: do nothing.

  b. IF: $R_{att}$ should be an attribute of the relationship $A \text{ VP } B$;
   THEN:
   1) delete $R_{att}$ from the entity where it appears as an attribute.
      IF: the attribute appeared in a candidate key for the entity;
      THEN: ask the user to identify the entity’s candidate keys again.
   2) add $R_{att}$ as an attribute of the relationship.
39. **IF**: a relationship $A \ VP \ B$ has at least one attribute;  
**THEN**: convert the relationship to an entity.

   a. Ask if the user wishes to provide a shorter name for $A \ VP \ B$.
      1) **IF**: a shorter name $R_{Name}$ is provided;  
         **THEN**: make $R_{Name}$ the name of the new entity.
      2) **IF**: a new name is not provided;  
         **THEN**: use $A \ VP \ B$ as the name of the new entity.

   b. **IF**: the verb phrase is *have/has* and a shorter name is not provided;  
      **THEN**: retain the relationship.  
      **OTHERWISE**: delete the relationship.

   c. Construct a candidate key for the new entity. The key attributes are the entity names which correspond to $A$ and $B$. Add an assertion which indicates that these key attributes will later be replaced by the primary keys of the entities after primary keys have been determined. (See *Primary Keys - Entities Requiring Other Entities for Identification.*) The non-key attributes for the new entity are the relationship attributes.

   d. Ask the user to identify an non-key attributes which could have more than one value for each occurrence of the new entity and proceed as outlined in *Entities - Attributes with Multiple Values*.

E.g.

Relationship: students take courses  
(Assume that a single student can take the same course multiple times.)

Relationship attribute: grade

Shorter name for the relationship: enrollment

Construct new entity:

enrollment: [STUDENT, COURSE, grade]

**Prompts for Missing Relationships**

**Entities without Relationships**

40. **IF**: there is an entity which does not appear in any of the existing relationships;  
**THEN**: ask the user to provide a relationship(s) which involves this entity.

   **IF**: no such relationship is provided;  
   **THEN**: ask if the entity should be deleted.
IF: the entity should be deleted;
THEN: delete the entity.

IF: the entity should not be deleted;
THEN: classify the entity as a "solo" entity.

Attribute Name was Entity Name

41. IF: there is an assertion which indicates that an entity $E_2$ may be needed to uniquely identify an instance of another entity $E_1$ because the user originally provided Term as an attribute of $E_1$ where Term was some form of the entity name $E_2$ (see Entities);
THEN:
  IF: there is at least one candidate key of entity $E_1$ which contains Term;
  THEN: do nothing. (This will be dealt with when obtaining primary keys.)
 IF: none of the candidate keys of entity $E_1$ contain Term;
 THEN: delete the assertion and determine if a relationship exists between $E_1$ and $E_2$.
    IF: there exists a relationship between $E_1$ and $E_2$;
    THEN: delete Term as an attribute of $E_1$. (The relationship provides the connection between $E_1$ and $E_2$.)
    IF: no relationship exists between $E_1$ and $E_2$
    THEN: ask the user to provide one.
    IF: a relationship is provided;
    THEN: delete Term as an attribute of $E_1$.
    IF: no relationship is provided;
    THEN: ask the user if Term should really be an attribute of $E_1$.
      IF: the user indicates that Term should be an attribute of $E_1$;
      THEN: retain Term as an attribute of $E_1$.
      IF: the user indicates that Term should not be an attribute of $E_1$;
      THEN: delete Term.

E.g.
The user provided library (the name of an existing entity) as an attribute of a new entity branch and indicated that library would be needed to uniquely identify branch.

Library: [LIBRARY_ID, library_name, library_address]
Branch: [BRANCH_NAME, LIBRARY, branch_address]
Branch: [BRANCH_NAME, BRANCH_ADDRESS, library]

Since LIBRARY appears in at least one candidate key of branch, do nothing.
the first candidate key of branch is later chosen as the primary key, the attribute LIBRARY will then be replaced by the primary key of library. See Primary Keys.)

42. IF: there is an assertion which indicates that a relationship is needed between two entities, \(E_1\) and \(E_2\), because the user originally tried to provide some form of \(E_2\) as an attribute of \(E_1\) but \(E_2\) is not needed to uniquely identify an instance of entity \(E_1\) (see Entities);

   THEN:

   IF: there exists a relationship between \(E_1\) and \(E_2\);
   THEN: do nothing.
   IF: no relationship exists between \(E_1\) and \(E_2\)
   THEN: ask the user to provide one.

43. IF: there is an assertion which indicates that a relationship is needed between two entities, \(E_1\) and \(E_2\), because the user originally added \(E_2\) as an entity when an attribute \(Att\), which was some form of \(E_2\), already appeared as an attribute of \(E_1\) (see Entities);

   THEN:

   IF: \(Att\) appears in any of the candidate keys of \(E_1\)
   THEN: do nothing. (This will be dealt with when obtaining primary keys.)

   E.g.

   The user provided library as a new entity when library already appeared as an attribute of an existing entity branch.

   Branch: [BRANCH_NAME, BRANCH_ADDRESS, library, ...]
   Branch: [BRANCH_NAME, LIBRARY, branch_address, ...]
   Library: [LIBRARY_ID, library_name, ...]

   Since library appears in at least one of the candidate keys of branch, do nothing. (If the second candidate key of branch is later chosen as the primary key, the attribute LIBRARY will then be replaced by the primary key of library.)

   IF: \(Att\) does not appear in one of the candidate keys of \(E_1\);
   THEN:

   IF: there exists a relationship between \(E_1\) and \(E_2\);
   THEN: inform the user that \(Att\) is no longer needed as an attribute of \(E_1\)
and delete it. (The relationship provides the connection between the entities.)

IF: no relationship exists between \(E_1\) and \(E_2\)
THEN: ask the user to provide one.

IF: a relationship is provided;
THEN: inform the user that \(Att\) is no longer needed as an attribute of \(E_1\) and delete it.
IF: no relationship is provided;
THEN: ask if \(Att\) should really be an attribute of \(E_1\).
IF: \(Att\) should be an attribute;
THEN: do nothing.
IF: \(Att\) should not be an attribute;
THEN: delete \(Att\).

Entity was Repeating Attribute

44. IF: there is an assertion which indicates that a relationship is required between two entities, \(E_1\) and \(E_2\), because the user originally tried to add multiple occurrences of entity \(E_2\) as attributes of entity \(E_1\) (see Entities – Repeating Attributes);
THEN:
IF: no relationship exists between \(E_1\) and \(E_2\);
THEN: ask the user to provide one.

Entity was Multivalued Attribute

45. IF: there is an assertion which indicates that a relationship is required between two entities, \(E_1\) and \(E_2\), because the user originally provided \(E_2\) as a multivalued attribute of entity \(E_1\) (see Entities – Attributes with Multiple Values);
THEN:
IF: no relationship exists between \(E_1\) and \(E_2\);
THEN: ask the user to provide one.

Missing Entities

46. IF: there is an assertion which indicates that a relationship is needed between an original entity \(E_O\) and an entity \(E_{ME}\) which was identified as a missing entity (see
**Missing Entities**;

**THEN:**

**IF:** a relationship exists between $E_O$ and $E_{ME}$;  
**THEN:** delete any attributes $A_{Qi}$ of entity $E_O$ which have corresponding assertions indicating that they may later be deleted. (The relationship provides the connection between the two entities.)

**IF:** there is no relationship between $E_O$ and $E_{ME}$;  
**THEN:** ask the user to provide one.  
**IF:** a relationship is provided;  
**THEN:** delete any attributes $A_{Qi}$ of entity $E_O$ which have corresponding assertions indicating that they may later be deleted.  
**IF:** no relationship is provided;  
**THEN:**

**IF:** there are any attributes $A_{Qi}$ of $E_O$ which have corresponding assertions indicating that they may later be deleted;  
**THEN:** ask if the attributes should be deleted;  
**IF:** the user indicates that the attributes should be deleted;  
**THEN:** delete the attributes.  
**IF:** the user indicates that the attributes should not be deleted;  
**THEN:** do nothing.

47. **IF:** there is an assertion which indicates that a relationship may be needed between two entities, $E_I$ and $E_{ME}$, because $E_{ME}$ was identified as a missing entity (see **Missing Entities** and $E_I$ contained attributes of the form $A_{ij}$ which were or became attributes of $E_{ME}$;

**THEN:**

**IF:** a relationship already exists between $E_I$ and $E_{ME}$, either directly or indirectly (e.g. via another entity $E_K$);  
**THEN:** delete any attributes $A_{ij}$ of entity $E_I$ which have corresponding assertions indicating that they may later be deleted. (The relationship provides the connection between the two entities.)  
**IF:** there is no relationship between $E_I$ and $E_{ME}$;  
**THEN:** ask the user to provide one.  
**IF:** one is provided;  
**THEN:** delete any attributes $A_{ij}$ of $E_I$ which have corresponding assertions indicating that they may later be deleted.  
**IF:** no relationship is provided;  
**THEN:**

**IF:** there are any attributes $A_{ij}$ of $E_I$ which have corresponding assertions indicating that they may later be deleted;
THEN: ask if the attributes should really belong to $E_i$;

IF: the user indicates that the attributes should belong to $E_i$;
THEN: do nothing.

IF: the user indicates that the attributes should not belong to $E_i$;
THEN: delete the attributes.

**Relationship Modification**

48. Allow the user to add or delete relationships.

IF: a relationship is deleted;
THEN: delete the corresponding assertion which indicates how the relationship was to be represented.

49. Allow the user to add relationship attributes.

a. IF: the user tries to add an attribute $R_{att}$ to a relationship $A \ VP B$ where the min/max values of $A$ are (1,1) and the min/max values of $B$ are also (1,1) (see Mapping Ratios);
THEN: inform the user that $R_{att}$ is probably not an attribute of the relationship but should be an attribute of either $A$ or $B$. Ask which entity $R_{att}$ corresponds to and add it as an attribute of that entity.

IF: the user does not know;
THEN: do nothing.

b. IF: the user tries to add an attribute $R_{att}$ to a relationship $A \ VP B$ where the min/max values of $A$ are (1,1) and the min/max values of $B$ are not (1,1) (see Mapping Ratios);
THEN: inform the user that $R_{att}$ is probably not an attribute of the relationship but should be an attribute of $A$ although it could be an attribute of $B$. Ask which entity $R_{att}$ corresponds to and add it as an attribute of that entity.

IF: the user does not know;
THEN: do nothing.

c. IF: the user tries to add an attribute $R_{att}$ to a relationship, $A \ VP B$ where the min/max values of the $A$ entity are not (1,1) and the min/max values of $B$ are (1,1) (see Mapping Ratios);
THEN: inform the user that $R_{att}$ is probably not an attribute of the relationship but should be an attribute of $B$ although it could be an attribute of $A$. Ask which
entity $R_{att}$ corresponds to and add it as an attribute of that entity.

**IF:** the user does not know;  
**THEN:** do nothing.

d. **IF:** an attribute is added to a relationship $A \ VP \ B$ where the min/max values for $A$ are not $(1,1)$ and the min/max values for $B$ are not $(1,1)$ (see Mapping Ratios);  
**THEN:** convert the relationship to an entity. (See Relationship Attributes).

**UNIDENTIFIED A's & B's**

When a relationship $A \ VP \ B$ is obtained, determine if the $A$ and $B$ values are:  
1) entities or attributes that the system already knows about  
2) attributes which should be identified as such  
3) entities which should be added.

50. **IF:** $A$ is an entity and $B$ is also an entity;  
**THEN:** do nothing.

51. **IF:** $B$ is an entity and:  
   a. $A$ is an attribute of $B$; or  
   b. $A_{\text{Suffix}}$ is an attribute of $B$, where $\text{Suffix}$ is one of the "key indicator attributes"; or  
   c. $B_{\text{A}}$ is an attribute of $B$;  
**THEN:** ask if $A$ should be an entity.  
**IF:** $A$ should be an entity;  
**THEN:**  
1) ask for the singular form of the entity  
2) obtain corresponding attributes and candidate keys  
3) delete $A (A_{\text{Suffix}}, \text{or} B_{\text{A}})$ as an attribute of $B$.  
   **IF:** $A (A_{\text{Suffix}}, \text{or} B_{\text{A}})$ was part of a candidate key of $B$;  
   **THEN:** ask the user to identify candidate keys for $B$ again.  
**IF:** $A$ should not be an entity;  
**THEN:** delete the relationship.  

(E.g. if author is an attribute of the entity book and the relationship author writes book is provided, ask if author should be an entity.)

52. **IF:** $B$ is an entity, $A$ is unidentified, and the verb phrase is is\_a;  
**THEN:** make $A$ an entity also.
(E.g. if employee is an entity and the relationship manager is _a employees is provided, make manager an entity also.)

53. **IF:** B is an entity, A is unidentified, and the verb phrase is not is _a;
**THEN:** ask if A should be an entity.

  **IF:** A should be an entity;
  **THEN:**
  1) obtain the singular form of the entity name
  2) ask for corresponding attributes and candidate keys.

  **IF:** A should not be an entity;
  **THEN:** suggest that the user has made a mistake and delete the relationship.

54. **IF:** neither A nor B are entities, but there exist attributes of the following form:
   a. A and B; or
   b. Ent_A A and Ent_B B; or
   c. A_Suffix and B; or
   d. Ent_A and B.
   the verb phrase is have/has;
**THEN:** make B an attribute of A and delete the relationship.  (*See Rule "Make Y an attribute of X".*)

  E.g. Relationship publisher has address.
  **IF:** there are attributes publisher and address;
  **THEN:** make address an attribute of publisher.

  E.g. Relationship publisher has address.
  **IF:** there are attributes book__publisher and X_address;
  **THEN:** make address an attribute of book__publisher.

  E.g. Relationship publisher has address.
  **IF:** there are attributes publisher_name and address;
  **THEN:** make address an attribute of publisher.

  E.g. Relationship publisher has address.
  **IF:** there are attributes book__publisher and address;
  **THEN:** make address an attribute of book__publisher.

55. **IF:** neither A nor B are entities, but they are both attributes and the verb phrase is not have/has;
**THEN:** do nothing.

56. **IF:** neither A nor B are entities, B is an attribute but A is not an attribute;
**THEN:** ask whether A should be an entity or an attribute of B.
IF: A should be an "entity";
THEN: obtain corresponding attributes and candidate keys.

IF: A should be an "attribute";
THEN: make A an attribute of B and delete the relationship.  *(See Rule "Make Y an attribute of X").*

IF: the user does not know;
THEN: make A an entity.

57.  IF:
   a.  A is an attribute of an entity Ent; or
       A_Suffix is an attribute of an entity Ent; or
       Ent_A_Suffix is an attribute of an entity Ent;
   b.  the verb phrase is have/has;
   c.  it is not known what B is;

THEN: ask if the relationship is correct.
   a.  IF: the relationship is correct;
       THEN: make B an attribute of A *(see Rule "Make Y an attribute of X")* and
               delete the relationship.
   b.  IF: the relationship is not correct;
       THEN: delete the relationship.

58. Make Y an attribute of X.
   a.  IF: an entity has an attribute X, or X_Suffix;
       THEN: add attribute X_Y.
       *(E.g. If an entity has the attribute X=publisher, or X=publisher_name, and
       Y=address; then add the attribute publisher_address.)*
   b.  IF: an entity Ent has an attribute Ent_X;
       THEN: add attribute Ent_X_Y.
       *(E.g. If an entity book has an attribute X=book_publisher and Y=address,
       then add the attribute book_publisher_address.)*
   c.  IF: an entity Ent has an attribute of the form Ent_X_Suffix;
       THEN: add an attribute Ent_X_Y.
       *(E.g. If entity book has an attribute X=book_publisher_name and Y=address;
       then add the attribute book_publisher_address.)*
   d.  IF: an entity has attribute Y, but not attribute X;
       THEN: do nothing.
       *(E.g. an entity has the attribute Y=address only.)*
59. **IF**: neither A nor B are entities or attributes;  
**THEN**: ask if they are both entities.  
a. **IF**: they both should be entities;  
**THEN**: obtain corresponding attributes and candidate keys.  
b. **IF**: they are not both entities;  
**THEN**: delete the relationship.

60. **IF**: A is an entity, the verb phrase is *have/has* and;  
a. B is an attribute of A; or  
b. \(B_{\text{Suffix}}\) is an attribute of A, where \(Suffix\) is a "key indicator attribute" or  
c. \(A_{\_B}\) is an attribute of A;  
**THEN**: ask whether B should be an entity or an attribute.  
**IF**: B should be an entity;  
**THEN**:  
1) ask for the singular form of the entity name  
2) obtain corresponding attributes and candidate keys  
3) delete B (\(B_{\_\text{Suffix}}, \text{or} \ A_{\_B}\)) as an attribute of A.  
**IF**: B (\(B_{\_\text{Suffix}}, \text{or} \ A_{\_B}\)) was part of a candidate key of A;  
**THEN**: ask the user to identify candidate keys for A again.  
**IF**: B should be an attribute or the user does not know;  
**THEN**: delete the relationship.  
(E.g. if entity book has an attribute publisher and the relationship book has publisher is provided, ask whether publisher should be an entity or an attribute.)

61. **IF**: A is an entity and the verb phrase is not *have/has* and;  
a. B is an attribute of A; or  
b. \(B_{\_\text{Suffix}}\) is an attribute of A where \(Suffix\) is one of the "key indicator attributes"; or  
c. \(A_{\_B}\) is an attribute of A;  
**THEN**: ask if B should be an entity.  
**IF**: B should be an entity;  
**THEN**:  
1) ask for the singular form of the entity name  
2) obtain corresponding attributes and candidate keys  
3) delete B (\(B_{\_\text{Suffix}}, \text{or} \ A_{\_B}\)) as an attribute of A.  
**IF**: B (\(B_{\_\text{Suffix}}, \text{or} \ A_{\_B}\)) was part of a candidate key of A;  
**THEN**: ask the user to identify candidate keys for A again.  
**IF**: B should not be an entity or the user does not know;  
**THEN**: suggest that the user probably made a mistake and delete the
relationship.

(E.g. if an entity book has an attribute author and the relationship book written by author is provided, ask if author should be an entity.)

62. **IF:** A is an entity and B is neither an entity nor an attribute;  
**THEN:** ask if B should be an entity;  
**IF:** B should be an entity;  
**THEN:**  
1) ask for the singular form of the entity name  
2) obtain corresponding attributes and candidate keys.  
**IF:** B should not be an entity;  
**THEN:** suggest that the user probably made a mistake and delete the relationship.

### MAPPING RATIOS

**Procedure**

For each relationship A VP B:

a. Obtain the min/max values for A.  
b. Obtain the min/max values for B.  
c. Determine the appropriate representation for the relationship based on the min/max values for A and B and (perhaps) the types of queries that are of interest to the user.

**NOTE:** At this point in the procedure, the primary keys have not yet been determined. Some of the rules indicate that a relationship will be represented either by:

a. adding the key attributes of one entity as non-key attributes of another entity or  
b. constructing a relation whose key is the concatenation of the keys of the involved entities.

In the implementation these rules are represented by assertions (in the system's database) whose values are not yet instantiated. After primary keys are obtained, the system will use these assertions to produce appropriate representations for the relationships.  
(See Represent Entities and Relationships.)
Infer Min/Max Values

63. **IF:** A is-a B
   **THEN:**
   a. define the min/max values of A to be (1,1)
   b. define the min/max values of B to be (0,1)

64. **IF:** A VP B where both A and B are in their singular forms;
   **THEN:**
   a. define the min/max values of A to be (1,1)
   b. query the user for the min/max values of B.

65. **IF:** A VP B where VP is of the form "X_a";
   **THEN:**
   a. define the maximum value of A to be 1
   b. query the user for the minimum value of A and the min/max values of B.

66. **IF:** A VP B where A is in its singular form and B is in its plural form:
   **THEN:**
   a. define the maximum value of A to be N
   b. query the user for the minimum value of A and the min/max values of B.

Query User for Min/Max Values

67. When querying the user for min/max values, use the singular forms of the entity names.

Min/Max Values of A

68. Ask: "Is there at least one 'B' for every 'A'?
   **IF:** the user's response is "no";
   **THEN:** define the min/max A values to be (0,N).
   (Real values could be (0,1) or (0,N).)
   **IF:** the user's response is "yes";
   **THEN:** Ask: "Is there at most one 'B' for every 'A'?"
   **IF:** the user's response is "no";
   **THEN:** min/max A values are (1,N).
IF: the user’s response is "yes";
THEN: min/max $A$ values are (1,1).

Min/Max Values of $B$

69. Ask: "Is there at least one 'A' for every 'B'?'" 

IF: the user’s response is "no";
THEN: define the min/max $B$ values to be (0,N). 
(Real values could be (0,1) or (0,N).)
IF: the user’s response is "yes";
THEN: Ask: "Is there at most one 'A' for every 'B'?'";
IF: the user’s response is "no";
THEN: min/max $B$ values are (1,N).
IF: the user’s response is "yes";
THEN: min/max $B$ values are (1,1).

Relationship Representation

$A(1,1)$ and $B(0,N)$

70. IF: $A$ has min/max values (1,1) and $B$ has min/max values (0,N);
THEN: represent the relationship by adding the key attributes of $B$ as non-key attributes of $A$.

$A(0,N)$ and $B(1,1)$

71. IF: $A$ has min/max values (0,N) and $B$ has min/max values (1,1);
THEN: represent the relationship by adding the key attributes of $A$ as non-key attributes of $B$.

$A(1,1)$ and $B(1,1)$

For min/max ratios $A$ (1,1) and $B$ (1,1), a relationship can be represented by:

- a. adding the key attributes of $A$ as non-key attributes of $B$;
- b. adding the key attributes of $B$ as non-key attributes of $A$;
- c. adding the key attributes of $B$ as non-key attributes of $A$ and adding the key
attributes of $A$ as non-key attributes of $B$.

The representation chosen depends on the type of query that is of most interest to the user.

72. **IF:** the min/max values of $A$ are (1,1) and the min/max values of $B$ are also (1,1)

**THEN:** Ask: "For the relationship $A \ VP \ B$ which of the following two forms of queries would you ask most often?"

a) the $A$ of $B$
b) the $B$ of $A$

"a" / "b" / "no" if both are equally likely or you do not know.

a. **IF:** "a";
   **THEN:** add the key attributes of $A$ as non-key attributes of $B$.

b. **IF:** "b";
   **THEN:** add the key attributes of $B$ as non-key attributes of $A$.

c. **IF:** "no";
   **THEN:** add the key attributes of $A$ as non-key attributes of $B$ and add the key attributes of $B$ as non-key attributes of $A$.

E.g. **IF:** "library has director"

where the min/max values of $A$ (library) are (1,1) and the min/max values of $B$ (director) are (1,1);

i.e. for each library there is one and only one director; for each director there is one and only one library.

Question: For the relationship "library has director" which of the following two forms of queries would you ask most often:

a) the "library" of "director" or
b) the "director" of "library"?

**IF:** "a";
**THEN:** add the key attributes of library as non-key attributes of director.

**IF:** "b";
**THEN:** add the key attributes of director as non-key attributes of library.

**IF:** "no";
**THEN:** do both of the above — i.e. add the key attributes of library as non-key attributes of director and add the key attributes of director as non-key attributes of library.
A(1,1) and B(1,N)

For min/max ratios A (1,1) and B (1,N), a relationship can be represented by:

a. adding the key attributes of B as non-key attributes of A
b. constructing a relation between A and B whose key is the concatenation of the key attributes of A and B.

The representation chosen depends on the type of query that is of most interest to the user.

73. **IF:** the min/max values of A are (1,1) and min/max values of B are (1,N);
**THEN:** Ask: "For the relationship A VP B which of the following two forms of queries would you ask most often?"
   a) the A of B
   b) the B of A
   "a" / "b" / "no" if both are equally likely or you do not know.

   **IF:** "a" or "no";
   **THEN:** construct a relation A__VP__B whose key consists of the key attributes of A and the key attributes of B (and there are no non-key attributes).
   **IF:** "b";
   **THEN:** add the key attributes of B as non-key attributes of A.

E.g. "branch belongs_to library"
where the min/max values for A (branch) are (1,1) and the min/max values for B (library) are (1,N), i.e. each branch has one and only one library; each library may have one or more branches.

Question: For the relationship "branch belongs_to library" which of the following two forms of queries would you ask most often:
   a) the "branch" of "library" or
   b) the "library" of "branch" ?

   **IF:** "a" / "no";
   **THEN** construct a relation: "branch_belongs_to_library" whose key is the concatenation of the key attributes of branch and the key attributes of library.
   **IF:** "b";
   **THEN** add the key attributes of library as non-key attributes of branch.
A(1,N) and B(1,1)

For min/max ratios A (1,N) and B (1,1), a relationship can be represented by:
   a. adding the key attributes of A as non-key attributes of B
   b. constructing a relation between A and B whose key is the concatenation of the key attributes of A and B.

The representation chosen depends on the type of query that is of most interest to the user.

74. IF: A is (1,N) and B is (1,1);
THEN: Ask: "For the relationship A VP B which of the following two forms of queries would you ask most often?"
   a) the A of B
   b) the B of A
"a" / "b" / "no" if both are equally likely or you do not know.

IF: "a";
THEN: add the key attributes of A as non-key attributes of B.
IF: "b" or "no";
THEN: construct a relation A VP B whose key consists of the key attributes of A and the key attributes of B (and there are no non-key attributes).

E.g. "library owns branches"
where the min/max values for A (library) are (1,N) and the min/max values for B (branches) are (1,1), i.e. each library may have one or more branches; each branch has one and only one library.

Question: For the relationship "library owns branches" which of the following two forms of queries would you ask most often:
   a) the "library" of "branch" or
   b) the "branch" of "library"?

IF: "a";
THEN add the key attributes of library as non-key attributes for branch.
IF: "b" or "no";
THEN: construct a relation "library_owns_branches" whose key is the concatenation of the key attributes of library and the key attributes of branch (and there are no non-key attributes).
Others

For min/max ratios: \( A (0,N) \) or \( (1,N) \) and \( B (0,N) \) or \( (1,N) \), a relationship is represented by: constructing a relation between \( A \) and \( B \) whose key is the concatenation of the key attributes of \( A \) and \( B \). Relationship attributes may exist.

75. \textbf{IF}: Relationship \( A \ VP \ B \) where min/max values of \( A \) are \( (0,N) \) or \( (1,N) \) and the min/max values of \( B \) are \( (0,N) \) or \( (1,N) \); \textbf{THEN}:
   a. \textbf{IF}: there are no relationship attributes;
      \textbf{THEN}: construct a relation \( A\_VP\_B \) whose key consists of the concatenation of the key attributes of \( A \) and the key attributes of \( B \).
   b. \textbf{IF}: there are relationship attributes;
      \textbf{THEN}: convert the relationship to an entity.
         1) Ask the user if he wishes to provide a shorter name for the entity.
         2) The key of the new entity is the concatenation of the key attributes of \( A \) and the key attributes of \( B \). The relationship attributes serve as non-keys. (See \textit{Relationship Attributes}.)

\textbf{HAVE/HAS RELATIONSHIPS}

76. \textit{Have/has} relationships are relationships whose verb phrases are either \textit{have} or \textit{has} or are of the form \textit{have/has X}.

77. Ask the user to identify which of the following is the appropriate interpretation for each of the \textit{have/has} relationships:
   1) possession
   2) component
   3) instance\_of
   4) associated\_with

78. \textbf{IF}: \( A \ have/has \ B \) and the \textit{have/has} is \textit{possession}; \textbf{THEN}: delete the relationship \( A \ have/has \ B \) and add the relationship \( A \ possesses \ B \).
   (E.g. \textit{library has branches} is converted to \textit{library possesses branches}.)

79. \textbf{IF}: \( A \ have/has \ B \) and the \textit{have/has} is \textit{component\_of}; \textbf{THEN}: delete the relationship \( A \ have/has \ B \) and add the relationship \( B \ component\_of \ A \).
   (E.g. \textit{library has book\_stacks} is converted to \textit{book\_stacks component\_of library}.)
80. **IF:** A have/has B and the have/has is instance_of;  
**THEN:**  
   a. delete the relationship A have/has B and add the relationship B instance_of A.  
      (E.g. books have volumes is converted to volumes instance_of books.)  
   b. **IF:** there is an assertion which indicates that the relationship is to be represented as a relation;  
      **THEN:** delete the assertion.

81. **IF:** A have/has B and the have/has is associated_with;  
**THEN:** delete the relationship A have/has B and add the relationship B associated_with A.  
      (E.g. book has publisher is converted to publisher associated_with book.)

82. **IF:** the user is not able to classify the type of have/has;  
**THEN:** do nothing.

83. **IF:** there exists an entity A have/has B;  
**THEN:** change the name of the entity to reflect the appropriate interpretation of have/has and delete the relationship.  
      (The entity was constructed to represent the relationship because the relationship had attributes. See Relationship Attributes.)

84. **IF:** A is_instance_of B;  
**THEN:** for each relationship X VP B (or B VP X), ask the user which relationship(s) is required:  
   a) X VP A (or A VP X)  
   b) X VP B (or B VP X)  
   c) both of the above.  
      **IF:** "a";  
      **THEN:** change the relationship X VP B (or B VP X) to X VP A (or A VP X).  
      **IF:** "b" (or the user does not know);  
      **THEN:** do nothing.  
      **IF:** "c";  
      **THEN:** add the relationship X VP A (or A VP X).  
      *E.g.*  
      **IF:** volume instance_of course and there exists a relationship borrowers borrow books;  
      **THEN:** ask which relationship(s) is required.  
      a) borrowers borrow volumes  
      b) borrowers borrow books  
      c) both.
MODIFICATIONS – FIRST

The initial set of user inputs has been obtained and consists of:
   a. entities and their corresponding attributes and candidate keys
   b. relationships.

85. Ask if the user wishes to:
   a. modify entities
      1) add, delete or rename an entity
      2) add, delete, or rename an attribute
   b. add or delete relationships (see Relationship Modification);
   c. add relationship attributes (see Relationship Modification).

86. IF: entities are modified;
   THEN:
   a. check for missing entities
   b. IF: a new entity is added;
      THEN: ensure that it appears in at least one relationship or is identified as a "solo" entity.
   c. if appropriate, prompt the user for missing relationships (see Relationships).

87. IF: relationship are modified;
   THEN:
   IF: a have/has relationship is added;
   THEN: identify the appropriate type of have/has. (See Have/Has Relationships.)

SYNONYMS

Synonyms – Entities

For relationships of the form:
the $A$ and $B$ entities are candidates to be synonyms.

88. Do not consider relationships whose verb phrases are:
   a. is _a
   b. instance
   c. component _of
   d. possesses
   e. have/has

89. Examine pairs of $A$ values first and then pairs of $B$ values.

90. For each pair of entities, $E_j$ and $E_2$, ask which of the following is true:
   a. they are exactly the same.
   b. $E_j$ is a subset of $E_2$.
   c. $E_2$ is a subset of $E_j$.
   d. none of the above.

   **IF**: "a";
   **THEN**:
   1) Ask which entity name $E_i$, the user would like to use.
   2) Retain entity $E_i$ and delete $E_j$.
   3) **IF**: there are attributes of the form $A_{j,k}$ or $E_j\_A_{j,k}$ of entity $E_j$ which are not attributes of entity $E_i$;
      **THEN**: add attributes $A_{j,k}$ or $E_i\_A_{j,k}$, respectively, to entity $E_i$.
   4) For each occurrence of a $E_j$ either in a relationship or in an assertion which indicates how a relationship is to be represented (see Mapping Ratios), change it to $E_i$.
      **IF**: this results in a relationship of the form $A \ VP A$;
      **THEN**: delete the relationship and the assertion which indicates how the original relationship was to be represented.
   5) Construct a synonym table of entity names. Let the first entry in a row of
the table be \( E_i \) and the second \( E_j \).

**IF:** "b";
**THEN:** add a relationship \( E_i \) is _a \( E_2 \)

**IF:** "c";
**THEN:** add a relationship \( E_2 \) is _a \( E_1 \)

**IF:** "d";
**THEN:** do nothing.

---

**Synonyms – Verb Phrases**

91. For relationships of the form:

\[
A \ VP_1 \ B \\
A \ VP_2 \ B \\
\vdots \\
A \ VP_n \ B,
\]

the verb phrases \( VP_1, VP_2, \ldots, VP_n \) are candidates to be synonyms.

92. Do not consider verb phrases which are is-a or (unclassified) have/has.

93. Ask the user to identify which, if any, verb phrases are synonyms.

94. **IF:** synonyms are identified;
**THEN:**
   a. Ask the user which term should be used. The replacement term must be one of the original verb phrases. Retain the relationship with the term chosen as the replacement term. Delete the other relationships and their corresponding assertions which indicate how they were to be represented.

   b. Construct a synonym table of verb phrases based on the user’s responses. Let the first entry in a row of the table contain the replacement term and the other entries in the row the synonyms for the replacement term.
Synonyms – Relationships

95. IF: there are relationships of the form $A \ VP_1 \ B, B \ VP_2 \ A$ etc.;
    THEN: ask if the user has represented the same information in different ways.

    IF: the user has represented the same information in different ways;
    THEN: Ask which relationship the user would like to use. (The replacement relationship must be one of the original relationships.) Retain the relationship chosen. Delete the other relationships along with their corresponding assertions which indicate how they were to be represented.

    IF: the user has not represented the same information in different ways;
    THEN: do nothing.

INHERITED ATTRIBUTES

96. IF: A is-a / instance-of B;
    THEN: scan the attributes of B for any which do not appear as attributes of A.

97. IF: there are attributes of B which are not attributes of A;
    THEN: ask the user if it would be incorrect to attribute those attributes of B to A.

    IF: not be incorrect to do so;
    THEN: do nothing.
    IF: it would be incorrect to do so;
    THEN: ask if the relationship should exist.

    IF: the relationship should not exist;
    THEN: delete the relationship and ask if the user wishes to add a new relationship(s).

    IF: the relationship should exist;
    THEN: ask if the user wishes to modify any attributes of B.

MODIFICATIONS – LAST

The initial set of user inputs has been obtained and the user has had an opportunities to modify his set of entities and relationships. The system has tried to detect synonyms and
"inherited" attributes have been examined.

98. IF: there were possible synonyms or inherited attributes to query the user about;
   THEN: allow the user to:
   a. modify entities
      1) add, delete or rename an entity
      2) add, delete, or rename an attribute
   b. add or delete relationships (see Relationship Modification);
   c. add relationship attributes (see Relationship Modification);

99. IF: entities are modified;
    THEN:
    a. check for missing entities
    b. IF: a new entity is added;
       THEN:
       1) ensure that it appears in at least one relationship or is classified as a "solo" entity.
       2) IF: the entity appears in an is-a or instance-of relationship;
          THEN: examine inherited attributes.
       3) IF: the new entity name was previously identified as a synonym for another entity;
          THEN: ask if the user is really providing new information.
          IF: new information is being provided;
              THEN: add the entity name as a new entity.
              IF: new information is not being provided;
              THEN: do nothing.
    c. If appropriate, prompt the user for missing relationships. (See Relationships.)
    d. Check for entities which are synonyms.

100. IF: relationships are modified;
     THEN:
     a. IF: a have/has relationship is added;
        THEN: identify the appropriate type of have/has. (See Have/Has Relationships.)
     b. IF: a new relationship A VP B is added and the verb phrase is not have/has;
        THEN:
        1) IF: there are no entries in the synonym table;
           THEN: add the relationship.
2) **IF:** VP occurs in an existing relationship;  
**THEN:** do nothing.

3) **IF:** VP does not occur in another relationship;  
**THEN:** check the synonym table.

**IF:** VP occurs as an entry in row N;  
**THEN:** inform the user that VP was identified as a synonym for VP\textsubscript{N,1}, the first entry in row N. Ask if the user really wishes to include VP or whether the relationship should have VP\textsubscript{N,1} as its verb phrase.

---

**PRIMARY KEYS**

**Rules for Choosing Primary Keys**

101. **IF:** there is only one candidate key;  
**THEN:** make this candidate key the primary key.

102. The simplest possible keys are preferred.

103. **IF:** one and only one of the candidate keys is a single-element key;  
**THEN:** this candidate key becomes the primary key.

104. **IF:** there is more than one single-element key:  
**THEN:** the one which occurs most often in other entities becomes the primary key.  
**IF:** there is a tie  
**THEN:** the one which was entered first becomes the primary key.

105. **IF:** there is no single-element key;  
**THEN:** determine which candidate key (whole key) appears most often (as a candidate or primary key) in other entities.  
**IF:** there is a tie;  
**THEN:** the one which was obtained first becomes the primary key.

106. **IF:** none of the above rules succeed in selecting a primary key;  
**THEN:** the first candidate key obtained becomes the primary key.
General Rules

107. Relationships are divided into three types when choosing primary keys: a) instance-of relationships b) is-a relationships and c) others.

108. IF: an is-a relationship, A is-a B, exists;
    THEN: "inherit" the key of the superset entity B as a candidate key for the subset entity A.

109. IF: an instance-of relationship, A instance-of B, exists;
    THEN: concatenate the key of the superset entity B to each of the candidate keys of the subset entity A.

110. IF: a relationship is neither is-a nor instance-of;
    THEN: choose the primary keys for the entities based on the Rules for Choosing Primary Keys.

Is-a Relationships

111. IF: . . . Z is-a A and A is-a B;
    THEN:
    a. Determine the primary key for the highest B in the is-a hierarchy.
    b. Inherit this primary key as a candidate key of the entity A for which B is a superset entity.
    c. For each of the original candidate keys of A, add the key attributes of B as non-key attributes of A (if they are not already key or non-key attributes of A).
    d. Choose the primary key for A based on the Rules for Choosing Primary Keys.
    e. Continue until primary keys have been obtained for each entity in the is-a hierarchy.

E.g. If manager is-a employee and employee is-a person, the primary key for person is selected first. Employee inherits the primary key for person as one of its candidate keys. The primary key for employee is then chosen based on its new set of candidate keys. This primary key of employee is added as a candidate key for manager and the primary key for manager obtained.

112. The highest B in an is-a or instance-of hierarchy need not be unique. (E.g. library_director is-a person & author is-a person.) After obtaining the primary key for the highest B, obtain primary keys for the entities in each hierarchy which has B at the top.
Adjust Attributes for Is-a Relationships

113. **IF:** A non-key attribute $\text{Ent}_{\text{subset}}\text{-att}$ of a subset entity is the same as a key attribute, $\text{Ent}_{\text{superset}}\text{-att}$, of a superset entity;  
**THEN:** do nothing.

114. **IF:** A subset entity has two attributes of the form $\text{Ent}_{\text{superset}}\text{-att}$ and $\text{Ent}_{\text{subset}}\text{-att}$ where $\text{Ent}_{\text{superset}}\text{-att}$ is a key attribute of the superset entity;  
**THEN:** delete the attribute $\text{Ent}_{\text{subset}}\text{-att}$.

   E.g. author is-a person  
   If: `author_name` and `person_name` are both attributes of author and `person_name` is a key attribute of person;  
   Then: delete the attribute `author_name` from the entity author.

115. **IF:** $\text{Ent}_{\text{superset}}\text{-att}$ is a non-key attribute of a superset entity and $\text{Ent}_{\text{subset}}\text{-att}$ is a non-key attribute of the subset entity;  
**THEN:** delete the attribute $\text{Ent}_{\text{subset}}\text{-att}$.

   E.g.  
   If: `manager` is-a person and person has attribute `person_birthday` (non-key) and manager has attribute `manager_birthday`;  
   Then: delete the attribute `manager_birthday` from the entity manager.

116. **IF:** the primary key of a subset entity is the same as the primary key of its superset entity;  
**THEN:** concatenate the name of the subset entity to each of its key attributes.

   E.g.  
   If: `manager` is-a employee and the primary key of both manager and employee is `employee_number`,  
   Then: change the primary key of manager to `manager_employee_number`.

Instance-of Relationships

117. **IF:** $A$ instance-of $B$ and $B$ instance-of $Z$;  
**THEN:**
   a. Determine the primary key for the highest $Z$ in the instance-of hierarchy.  
   b. Concatenate this primary key to each candidate key of the subset entity $A$.  
   c. Choose the primary key for $A$ based on the Rules for Choosing Primary Keys.  
   d. Continue until primary keys have been chosen for each entity in the instance-of hierarchy.
Adjust Attributes for Instance-of Relationships

118. **IF:** a subset entity has a non-key attribute \( \text{Ent}_{\text{subset}}^{-\text{att}} \) and its superset entity has a non-key attribute \( \text{Ent}_{\text{superset}}^{-\text{att}} \);

**THEN:** delete the attribute \( \text{Ent}_{\text{subset}}^{-\text{att}} \).

E.g. **If:** volume instance of book and volume has a non-key attribute \( \text{volume}_{-\text{title}} \) and book has a non-key attribute \( \text{book}_{-\text{title}} \);

Then: delete \( \text{volume}_{-\text{title}} \) from the entity volume.

Entities Requiring Other Entities for Identification

119. **IF:** there is an assertion which indicates that the key of an entity \( \text{E}_2 \) may be needed to uniquely identify an instance of the entity \( \text{E}_1 \) (see Entities and Relationships – Prompts for Missing Relationships);

**THEN:**

**IF:** the primary key of \( \text{E}_1 \) includes an attribute \( \text{Term} \) which is some form of \( \text{E}_2 \) (e.g. \( \text{E}_2 \) or \( \text{E}_2 \_\text{Suffix} \) where \( \text{Suffix} \) is one or the "key indicator attributes", etc.)

**THEN:** replace \( \text{Term} \) with the primary key of \( \text{E}_2 \).

E.g. Candidate Keys for Library:
Library: [LIBRARY\_ID, library\_name, library\_address ... ]
Library: [LIBRARY\_NAME, LIBRARY\_ADDRESS, library\_id, ... ]
Candidate Key for Branch:
Branch: [BRANCH\_NAME, LIBRARY, ... ]

Primary Keys:
Library: [LIBRARY\_ID, library\_name, library\_address ... ]
Branch: [BRANCH\_NAME, LIBRARY, ... ]

Replace the attribute "LIBRARY" in the entity "branch" by the primary key of "library":
Library: [LIBRARY\_ID, library\_name, library\_address ... ]
Branch: [BRANCH\_NAME, LIBRARY\_ID, ... ]

**IF:** the primary key of \( \text{E}_1 \) does not include an attribute \( \text{Term} \) which is some form of \( \text{E}_2 \) and \( \text{Term} \) appears as a non-key attribute of \( \text{E}_1 \);

**THEN:** ask for a relationship between \( \text{E}_1 \) and \( \text{E}_2 \).
IF: Term should be an attribute of \( E_j \);
THEN: do nothing.

IF: Term should not be an attribute of \( E_j \);
THEN: delete Term.

E.g.

**Primary Keys:**

Library: [LIBRARY_ID, library_name, library_address ... ]

Branch: [BRANCH_NAME, BRANCH_ADDRESS, library, ... ]

Ask for a relationship between branch and library. If one is provided, delete library as an attribute of branch. If a relationship is not provided, ask if library should be an attribute of branch.

**REPRESENT ENTITIES and RELATIONSHIPS**

**Entities**

120. Construct a relation (i.e. an entity relation) to represent each entity. The key and non-key attributes of an entity relation are the primary and non-key attributes, respectively, of the corresponding entity.

**Relationships**

At this point, there are assertions in the system’s database which indicate how the relationships should be represented. Each relationship can be represented in either of two ways (see *Mapping Ratios*):

a. by constructing a relation whose key is the concatenation of the keys of the two entities in the relationship.

b. by adding the key attributes of one entity as non-key attributes of the other. (See *Mapping Ratios*.)

The primary keys for the entities have been determined, therefore the values for the variables in these assertions can now be instantiated.
Construct Relations

121. IF: an assertion indicates that a relation should be constructed whose key is the concatenation of the keys of two entities;  
    THEN:  
        IF: the verb phrase is instance_of  
            THEN: do nothing. (The key of the subset entity already includes the key of the superset entity. See Primary Keys – Instance-of Relationships.)  
        IF: the verb phrase is not instance_of;  
            THEN: concatenate the keys of the involved entities and make the result the key of the relation.  
        IF: an attribute is common to the keys of both entities;  
            THEN: include it only once in the relation key.  

E.g.  
Relationship: students take courses  

Entities:  
Student: [STUDENT_ID, student_name, ...]  
Course: [COURSE_NO, description, ...]  

Assertion: Construct a relation students_takes_courses.  

Relation:  
students_take_courses: [STUDENT_ID, COURSE_NO]  

Add Key Attributes of One Entity as Non-key Attributes of Another Entity

122. IF: the key attributes of an entity $E_1$ are to be added as non-key attributes of another entity $E_2$;  
    THEN: add the attributes of $E_1$ as non-key attributes of $E_2$ if they do not already appear as either key or non-key attributes of $E_2$.  

E.g.  
Relationship: Manager is an Employee  

Entities:  
Manager: [MANAGER_NAME, DEPT, ... ]  
Employee: [EMPLOYEE_NO, employee_name, ...]
Assertion: Add the key attributes of employee as non-key attributes of manager.

Modified Manager Entity:
Manager: [MANAGER\_NAME, DEPT, employee\_no, ... ]

FUNCTIONAL DEPENDENCIES

Partial Functional Dependencies

123. IF: a relation has two key attributes, KeyAttribute\textsubscript{1} and KeyAttribute\textsubscript{2}, and at least one non-key attribute; 
THEN: for each non-key attribute ask which of the following is needed before the value of the attribute is known:

a. KeyAttribute\textsubscript{1}
b. KeyAttribute\textsubscript{2}
c. KeyAttribute\textsubscript{1} and KeyAttribute\textsubscript{2}.

IF: the user choses either of the first two options; 
THEN: a partial dependency has been identified.

124. IF: a relation has three key attributes, KeyAttribute\textsubscript{1}, KeyAttribute\textsubscript{2}, and KeyAttribute\textsubscript{3}, and at least one non-key attribute; 
THEN: for each non-key attribute ask which of the following is needed before the value of the attribute is known:

a. KeyAttribute\textsubscript{1}
b. KeyAttribute\textsubscript{2}
c. KeyAttribute\textsubscript{3}
d. KeyAttribute\textsubscript{1} and KeyAttribute\textsubscript{2}
e. KeyAttribute\textsubscript{1} and KeyAttribute\textsubscript{3}
f. KeyAttribute\textsubscript{2} and KeyAttribute\textsubscript{3}
g. KeyAttribute\textsubscript{1}, KeyAttribute\textsubscript{2} and KeyAttribute\textsubscript{3}.

IF: the user choses any of the first six options; 
THEN: a partial dependency has been identified.
125. **IF:** a relation has four or more key attributes and at least one non-key attribute;  
**THEN:** ask the user to identify any non-key attributes which depend on only part of the key instead of the complete key. Any non-key which meets this criteria has a partial dependency.

126. **IF:** an attribute has a partial dependency and no other attribute has the same partial dependency;  
**THEN:** construct a new relation whose non-key is the partially dependent attribute. The key of the new relation is the part of the original key upon which the attribute depends. Ask the user for the name of the new relation and delete the partially dependent attribute from the original relation.

127. **IF:** two or more attributes have the same partial dependency;  
**THEN:** construct a new relation whose non-keys are the partially dependent attributes. The key of the new relation is the part of the original key upon which the attributes depend. Ask the user for the name of the new relation and delete the partially dependent attributes from the original relation.

*E.g.*

flight: [FLIGHT_NUMBER, DATE, city_from, city_to, no_passengers]  
(City_from and city_to depend on flight_number only.)

**Construct new relations:**

Flight: [FLIGHT NUMBER, DATE, no_passengers]  
Timetable: [FLIGHT NUMBER, city_from, city_to]  
(where "timetable" is the relation name provided by the user.)

**Transitive Functional Dependencies**

128. For each relation which has two or more non-key attributes, ask the user to identify any non-key attributes which depend on other non-key attributes instead of the key attributes. Any attribute which meets this criteria has a transitive dependency.

129. **IF:** the user indicates that there are no transitive dependencies and there are attributes of the form $X, X_{Suffix_1}, X_{Suffix_2}, \text{etc.};$  
**THEN:** suggest that the user consider $X, X_{Suffix_1}, X_{Suffix_2}, \text{etc.}$ for transitive dependencies.

130. **IF:** a non-key has a transitive dependency;  
**THEN:** construct a new relation. The key of the new relation is the non-key(s) upon which the transitively dependent attribute depends. The non-key of the new relation is the transitively dependent attribute. Delete the transitively dependent attribute from the original relation. The name of the new relation is the name of the non-key
(or names of the non-keys concatenated) on which the original transitively dependent attribute depends.

**IF:** The name of a new relation is of the form \( Att \_Suffix \) where \( Suffix \) is one of the "key indicator attributes";

**THEN:** Change the name of the new relation to \( Att \).

*E.g.*

Book: \([\text{CATALOG\_NUMBER}, \text{title}, \text{author}, \text{publisher}, \text{publisher\_city}]\).
*(Publisher\_city is transitively dependent on publisher.)*

*Construct new relations:*

Book: \([\text{CATALOG\_NUMBER}, \text{title}, \text{author}, \text{publisher}]\)

Publisher: \([\text{PUBLISHER}, \text{publisher\_city}]\).
APPENDIX 2 – DESIGN SESSION

User? Please enter your name, followed by a dot (.) and return.

I have to ask you to always respond in this manner, i.e. give your answer followed by a dot and then return.

Now enter your name please.
|: Veda.
Hello veda .

For what do you wish to design a database?
(If you need more than one word to respond, connect the words with underscore characters.)
|: training.

--- ENTITIES ---

Tell me what things are of interest to you in a "training". I will refer to these "things" as "entities".

I also want you to give me the "attributes" of these "entities", i.e. the properties or characteristics that are of interest to you.

-------- E.g. Airline_Reservation --------
ENTITIES       ATTRIBUTES

flight         number city_from city_to no_of_passengers
passenger      name frequent_flyer_status
aircraft       id type max_passengers
person         name address phone

Note that you may only use one "word" for each attribute but the "word" may consist of several words connected by underscore characters.

-----------
9 This session is a variation of one done during the formal testing of the system. (See Appendix 4 – User#1.) The original session has been altered so that different aspects of the system's capabilities can be demonstrated. To make the log more readable, comments have been added and are found between the symbols "*/" and "/*".


If you enter something and then decide that it's wrong, you can change it later. Now please enter the name of an entity in its SINGULAR form (followed by a dot and return).

|: course.
What are the attributes for "course"?
Enter them separated only by a blank and with a period after the last one. Else enter "end." if you really do not know.
|: name number date session instructor.
Any more attributes for "course"?
("see."/"yes."/"no.")
|: yes.
Enter other attributes (or "end.")
|: duration.
Any more attributes for "course"?
("see."/"yes."/"no.")
|: no.
Good.

Here are the attributes for:
course: [name, number, date, session, instructor, duration]

Can a single "course" have more than one value for any of these attributes?
("no." / "why" / attribute(s) which can have more than one value / "end." if you do not know.)
|: date session.
I will delete the attribute "date"
and we will make it an entity later.

I will delete the attribute "session"
and we will make it an entity later.

Here are the attributes for:
course: [name, number, instructor, duration]

Enter an attribute or combination of attributes which could be used to UNIQUELY identify an instance of the entity "course".
I will call this attribute (or combination of attributes) a "key".
(If you really do not know, enter "end.")
In particular, you may wish to consider:

1) name 2) number

Now, enter identifying attribute(s).
Else enter: "see." / "end." if you really do not know.
   |
   : number.
Is there any other way that an instance of the entity "course"
could be identified?
("see.", to see how I can identify it so far / "no." / "yes."")
   |
   : yes.
Please enter the attribute(s).
   |
   : name.
Is there any other way that an instance of the entity "course"
could be identified?
("see.", to see how I can identify it so far / "no." / "yes."")
   |
   : no.
Fine.

/* The user originally thought that "date" was a multivalued attribute
   of "course" so the system tries to make it a new entity. */

Now let's deal with "date".
Is the term "date" in its SINGULAR form?
("yes."/"no.")
   |
   : y.
Fine.

What are the attributes for "date"?
Enter them separated only by a blank and with a period after
the last one. Else enter "end." if you really do not know.
   |
   : end.
Then should the entity "date" exist?
("yes."/"no.")
   |
   : no.
Then I shall delete this entity.

/* "Session was a multivalued attribute of "course" so it becomes
   a new entity. */

Now let's deal with "session".
Is the term "session" in its SINGULAR form?
("yes."/"no.")
What are the attributes for "session"?
Enter them separated only by a blank and with a period after
the last one. Else enter "end." if you really do not know.
[: yes.
Fine.

/* An attribute name is the name of an entity. */

I already know that "course" is an entity.
Does this mean that you will need to know "course"
before you will be able to UNIQUELY identify an instance of "session"?
("yes."/"no.")
[: yes.
Fine.

Any more attributes for "session"?
("see."/"yes."/"no.")
[: no.
Good.

Here are the attributes for:
session: [date, course]

Can a single "session" have more than one value for any of these attributes?
("no." / "why" / attribute(s) which can have more than one value /
"end." if you do not know.)
[: no.
Fine.

Here are the attributes for:
session: [date, course]

Enter an attribute or combination of attributes (i.e. a "key")
which could be used to UNIQUELY identify an instance of the
entity "session".
(If you really do not know, enter "end.")
[: date course.
Enter "see." if you wish to see what entities you have so far.
Otherwise enter the name of the next entity (or "end.").
[: see.
Here are the entities and attributes with identifying attributes (keys) written in UPPER CASE. If there are different candidates for identifying entity occurrences, all will be displayed.

course: [NUMBER,name,instructor,duration]
course: [NAME,number,instructor,duration]
session: [DATE,COURSE]
End entities.

Enter the name of the next entity (or "end.").
|: instructor.
Enter the attributes for: "instructor".
(Else "end." if you do not know.)
|: name employee _ number.
Any more attributes for "instructor"?
("see./""yes./""no.")
|: no.
Good.

Here are the attributes for:
instructor: [name, employee _ number]

Can a single "instructor" have more than one value for any of these attributes?
("no." / "why" / attribute(s) which can have more than one value / "end." if you do not know.)
|: no.
Fine.

Here are the attributes for:
instructor: [name, employee _ number]

Enter the attribute(s) which UNIQUELY identify an instance of "instructor".
(Or "end." if you really do not know.)

In particular, you may wish to consider:
1) name

Now, enter identifying attribute(s).
Else enter "see." / "end."
employee_number.
Is there any other way that an instance of the entity "instructor"
could be identified?
("see.", to see how I can identify it so far / "no." / "yes.")
|: no.
Fine.
Enter next entity / "see." / "end."
|: student.
Enter the attributes for: "student".
(Else "end." if you do not know.)
|: name employee_number.
Any more attributes for "student"?
("see."/"yes."/"no.")
|: no.
Good.

Here are the attributes for:
student: [name, employee_number]

Can a single "student" have more than one value for any of these attributes?
("no." / "why" / attribute(s) which can have more than one value /
"end." if you do not know.)
|: no.
Fine.

Here are the attributes for:
student: [name, employee_number]

Enter the attribute(s) which UNIQUELY identify an instance of "student".
(Or "end." if you really do not know.)

In particular, you may wish to consider:

1) name

Now, enter identifying attribute(s).
Else enter "see." / "end."
|: employee_number.
Is there any other way that an instance of the entity "student"
could be identified?
("see.", to see how I can identify it so far / "no." / "yes.")
|: no.
Fine.

Enter next entity / "see." / "end."
[:  see.

course: [NUMBER,name,instructor,duration]
course: [NAME,number,instructor,duration]
session: [DATE,COURSE]
instructor: [EMPLOYEE__NUMBER,name]
student: [EMPLOYEE__NUMBER,name]
End entities.

Enter the name of the next entity (or "end.").
[:  employee.
Enter the attributes for: "employee".
(Else "end." if you do not know.)
[:  employee__number name address job__title.
Any more attributes for "employee"?
("see."/"yes."/"no.")
[:  no.
Good.

Here are the attributes for:
employee: [employee__number, name, address, job__title]

Can a single "employee" have more than one value for any of these attributes?
("no." / "why" / attribute(s) which can have more than one value /
"end." if you do not know.)
[:  no.
Fine.

Here are the attributes for:
employee: [employee__number, name, address, job__title]

Enter the attribute(s) which UNIQUELY identify an instance of "employee".
(Or "end." if you really do not know.)

/* "Employee__number" is recognized as a "key indicator attribute" for this
 application. */

In particular, you may wish to consider:
1) employee_number  2) name

Now, enter identifying attribute(s).
Else enter "see." / "end."
|:  employee_number.
Is there any other way that an instance of the entity "employee"
could be identified?
("see.", to see how I can identify it so far / "no." / "yes.")
|:  no.
Fine.

Enter next entity / "see." / "end."
|:  see.

course: [NUMBER,name,instructor,duration]
course: [NAME,number,instructor,duration]
session: [DATE,COURSE]
instructor: [EMPLOYEE_NUMBER,name]
student: [EMPLOYEE_NUMBER,name]
employee: [EMPLOYEE_NUMBER,name,address,job_title]
End entities.

Enter the name of the next entity (or "end.").
|:  end.

Here are the entities, with their attributes and keys.
course: [COURSE_NAME,instructor,duration,course_number]
course: [COURSE_NUMBER,instructor,duration,course_name]
session: [DATE,COURSE]
instructor: [INSTRUCTOR_EMPLOYEE_NUMBER,instructor_name]
student: [STUDENT_EMPLOYEE_NUMBER,student_name]
employee: [EMPLOYEE_NUMBER,address,job_title,employee_name]
End entities.

Note that I may have made a few modifications to the attribute names.
Do you wish to add, delete or change any of the entities?

("no."/"add."/"delete."/"change.")
Enter the name of the entity to be changed (or "end.").

What would you like to change?

Enter:
"r." to rename the entity
"m." to modify attribute(s) (add/delete/rename)
"r." / "m." / "see." / "end."

Do you wish to add, delete, or rename an attribute?
"add." / "delete." / "rename." / "see." / "end."

Enter the attribute(s) you wish to add / "see." / "end."

Attribute "cost_material" added to "session".
Attribute "cost_labor" added to "session".
Attribute "man_hrs" added to "session".

Any other modifications?
("see."/"no."/"add."/"delete."/"change.")

Do you wish to add, delete or change any of the entities?

("no."/"add."/"delete."/"change.")

Fine.
--- RELATIONSHIPS ---

You have a training with the things you told me about. I need to know what happens with all of these things and how they interact with each other.

---------- E.g. AirlineReservation ----------

flights have aircraft
passenger is a person
passengers reserve for flights
etc.

----------------------------------

Please enter your information with three words (may use underscores) on each line, followed by a dot and return. Note that you can use both SINGULAR and PLURAL forms of the entities where appropriate.

If you make a mistake, you can change it later. From now on I will refer to the information that you are going to enter as "relationships".

First, would you like to see your entities again?
("yes."/"no.")
|-- no.
Fine. Enter a relationship of the form "A verb B".

|-- students attend sessions.

"students attend sessions"

Is there at least one "session" for every "student"?
("yes."/"no.")
|-- y.
Is there at most one "session" for every "student"?
("yes."/"no.")
|-- n.

Is there at least one "student" for every "session"?
("yes."/"no.")
|-- y.
Is there at most one "student" for every "session"?
("yes."/"no.")
|: n.

An attribute of a relationship is a property or characteristic of the relationship which is of interest to you.

E.g. Relationship: "passengers reserve for flights"
Acceptable attribute: "class".

Note that "class" is not an attribute of "passenger" or "flight" but is specifically an attribute of the relationship: "passengers reserve for flights".

When I add certain relationships I will ask for corresponding attributes. Relationship attributes may not always exist.

Enter attribute(s) for "students attend sessions" / "end."
|: grade attendance.
Attribute "grade" added.
Attribute "attendance" added.

/* A relationship with attributes is converted to an entity. */

A relationship of this type is often more conveniently dealt with as an entity so I am going to convert it to one.

Do you wish to provide a shorter name for "students attend sessions"?
Enter new name in its SINGULAR form. / "end."
The name may be one word or two words connected with an underscore character.
|: end.
Here are the attributes for:
students attend sessions: [student, session, grade, attendance]

Can a single "students attend sessions" have more than one value for any of these attributes?
("no." / "why" / attribute(s) which can have more than one value / "end." if you do not know.)
|: no.
Fine.
Enter next relationship / "see." / "end." / "see__ent." (to see entities)

| : courses have sessions.  
| : 
| : "courses have sessions"

Is there at least one "session" for every "course"?
("yes."/"no.")
| : y.
Is there at most one "session" for every "course"?
("yes."/"no.")
| : n.

Is there at least one "course" for every "session"?
("yes."/"no.")
| : y.
Is there at most one "course" for every "session"?
("yes."/"no.")
| : y.

For the relationship "courses have sessions"

Which of the following two forms of queries would you ask most often?

a) the "course" of "session"  
   or:
b) the "session" of "course"?

("a." / "b." / "no." if both are equally likely or you do not know)
| : b.

Enter next relationship / "see." / "end." / "see__ent." (to see entities)

| : teachers teach courses.
Is "teachers" a new entity?
You haven't mentioned it before.

("yes."/"no.")
|: n.
Then I think you made a mistake so I will delete the relationship.

Enter next relationship / "see." / "end." / "see_ent." (to see entities)
|: instructors teach courses.

"instructors teach courses"

Is there at least one "course" for every "instructor"?
("yes."/"no.")
|: yes.
Is there at most one "course" for every "instructor"?
("yes."/"no.")
|: no.

Is there at least one "instructor" for every "course"?
("yes."/"no.")
|: y.
Is there at most one "instructor" for every "course"?
("yes."/"no.")
|: n.
Enter attribute(s) for "instructors teach courses" / "end."
|: end.
Enter next relationship / "see." / "end." / "see_ent." (to see entities)
|: see.

RELATIONSHIPS
courses have sessions
instructors teach courses
End relationships.
Enter next relationship / "see." / "end." / "see_ent." (to see entities)

|: end.

/* Prompts for missing relationships. */

I have the entity "employee" but it does not occur in any relationship.
Would you like to add a relationship(s) which includes the entity "employee"?
("yes."/"no.")
|: yes.
Then add the relationships as you did above, i.e. give me relationships of the form "A verb B"
Enter "end." when you finish.
|: instructors are employees.
Enter next relationship / "see." / "end." / "see_ent." (to see entities)

|: students are employees.
Enter next relationship / "see." / "end." / "see_ent." (to see entities)

|: end.

/* "Instructor" was originally provided as an attribute of the entity course before instructor became an entity. */

I have a relationship between "course" and "instructor" so I no longer need "instructor" as an attribute of "course". I shall delete it.

Attribute "instructor" deleted from "course".
"courses have sessions"

Which of the following is true?

a) "course" owns / possesses "session"

b) "session" is a component of "course"

c) "session" is an instance / example of "course"

d) "session" is associated with "course" (in some other way)

"a." / "b." / "c." / "d." / "no." (if you do not know.)
\[\text{c.}\]

/* For an "instance_of" relationship, the system tries to ensure that the information requirements are captured at the correct level of detail. */

I know that "sessions instance_of courses" and "instructors teach courses"

Which relationship(s) should I have:

a) "instructors teach sessions"

b) "instructors teach courses"

c) both

"a." / "b." / "c." / "no." if you do not know.
\[\text{a.}\]

Fine, then I shall delete "instructors teach courses"
and add "instructors teach sessions"
Relationship "instructors teach courses" deleted.
"instructors teach sessions"

Is there at least one "session" for every "instructor"?  
("yes."/"no.")
|:  y.

Is there at most one "session" for every "instructor"?  
("yes."/"no.")
|:  n.

Is there at least one "instructor" for every "session"?  
("yes."/"no.")
|:  y.

Is there at most one "instructor" for every "session"?  
("yes."/"no.")
|:  y.

For the relationship "instructors teach sessions"

Which of the following two forms of queries would you ask most often?

a) the "instructor" of "session"
   or:

b) the "session" of "instructor"?

("a." / "b." / "no." if both are equally likely or you do not know)
|:  no.

--- MODIFICATIONS – FIRST ---

Here is your set of entities and relationships.

course: [COURSE_NAME,duration,course_number]
course: [COURSE_NUMBER,duration,course_name]
session: [DATE,COURSE,cost_material,cost_labor,man_hrs]
instructor: [INSTRUCTOR_EMPLOYEE_NUMBER,instructor_name]
student: [STUDENT_EMPLOYEE_NUMBER,student_name]
employee: [EMPLOYEE_NUMBER,address,job_title,employee_name]
students_attend_sessions: [STUDENT,SESSION,grade,attendance]
End entities.
instructors are employees
students are employees
sessions instance of courses
instructors teach sessions
End relationships.

Do you wish to modify any of the entities?
("see."/"yes."/"no.")
|:  n.
Do you wish to modify any of the relationships?
("see."/"yes."/"no.")
|:  n.
Do you wish to add any relationship attributes (i.e. properties or characteristics of a relationship as a whole)?
("see."/"yes."/"no.")
|:  n.
Fine.

--- SYNONYMS ---

/* The system checks for possible synonyms, but there are none to query the user about. */

--- INHERITED ATTRIBUTES ---

"instructors are employees"

Since "instructors are employees" we should be able to attribute to "instructor" all the attributes of "employee".
Would it be incorrect to attribute any of the following attribute(s) to "instructor"?
  1) employee_number  2) address  3) job_title  4) employee_name
"students are employees"

Since "students are employees" we should be able to attribute to "student" all the attributes of "employee". Would it be incorrect to attribute any of the following attribute(s) to "student"?
1) employee_number 2) address 3) job_title 4) employee_name

("yes."/"no.")
|:  n.

"sessions instance of courses"

Since "sessions instance of courses" we should be able to attribute to "session" all the attributes of "course". Would it be incorrect to attribute any of the following attribute(s) to "session"?
1) duration 2) course_name 3) course_number

("yes."/"no.")
|:  n.
Here is what I have so far.

course: [COURSE_NAME,duration,course_number]
course: [COURSE_NUMBER,duration,course_name]
session: [DATE,COURSE,cost_material,cost_labor,man_hrs]
instructor: [INSTRUCTOR_EMPLOYEE_NUMBER,instructor_name]
student: [STUDENT_EMPLOYEE_NUMBER,student_name]
employee: [EMPLOYEE_NUMBER,address,job_title,employee_name]
students_attend_sessions: [STUDENT,SESSION,grade,attendance]

End entities.

instructors are employees
students are employees
sessions instance_of courses
instructors teach sessions

End relationships.

Do you wish to modify any of the entities?
("see."/"yes."/"no.")
|: n.

Do you wish to modify any of the relationships?
("see."/"yes."/"no.")
|: n.

Do you wish to add any relationship attributes (i.e. properties or characteristics of a relationship as a whole)?
("see."/"yes."/"no.")
|: n.

Fine.
--- PRIMARY KEYS ---

/* The system determines primary keys without interaction with the user. */

course: [COURSE_NUMBER,duration,course_name]
session: [COURSE_NUMBER,DATE,cost_material,cost_labor,man_hrs]
instructor: [INSTRUCTOR_EMPLOYEE_NUMBER]
student: [STUDENT_EMPLOYEE_NUMBER]
employee: [EMPLOYEE_NUMBER,address,job_title,employee_name]

students attend sessions:
[DATE,COURSE_NUMBER,STUDENT_EMPLOYEE_NUMBER,grade,attendance]

End entities.

--- CONSTRUCT RELATIONS ---

instructors teach sessions:
[INSTRUCTOR_EMPLOYEE_NUMBER, COURSE_NUMBER, DATE]
course: [COURSE_NUMBER,duration,course_name]
session: [COURSE_NUMBER, DATE,cost_material,cost_labor,man_hrs]
instructor: [INSTRUCTOR_EMPLOYEE_NUMBER]
student: [STUDENT_EMPLOYEE_NUMBER]
employee: [EMPLOYEE_NUMBER,address,job_title,employee_name]

students attend sessions:
[DATE, COURSE_NUMBER, STUDENT_EMPLOYEE_NUMBER,grade,attendance]

End Relations.

--- FUNCTIONAL DEPENDENCIES ---

/* Partial Functional Dependencies. */

session: [COURSE_NUMBER, DATE,cost_material,cost_labor,man_hrs]

Which of the following do you need before you know the "cost_material"?
a) course_number
b) date
c) course_number & date
"a." / "b." / "c." / "no." (if you do not know)
$: c.

session: [COURSE_NUMBER, DATE,cost_material,cost_labor,man_hrs]

Which of the following do you need before you know the "cost_labor"?
a) course_number
b) date
c) course_number & date
"a." / "b." / "c." / "no." (if you do not know)
$: n.

session: [COURSE_NUMBER, DATE,cost_material,cost_labor,man_hrs]

Which of the following do you need before you know the "man_hrs"?
a) course_number
b) date
c) course_number & date
"a." / "b." / "c." / "no." (if you do not know)
$: c.

students_attend_sessions:
[DATE, COURSE_NUMBER, STUDENT_EMPLOYEE_NUMBER,grade,attendance]

Which of the following do you need before you know the "grade"?
a) date
b) course_number
c) student_employee_number
d) date & course_number
e) date & student_employee_number
f) course_number & student_employee_number
g) date & course_number & student_employee_number
"no." (if you do not know)
|: g.

students_attend_sessions:
[DATE, COURSE_NUMBER, STUDENT_EMPLOYEE_NUMBER, grade, attendance]

Which of the following do you need before you know the "attendance"?
  a) date
  b) course_number
  c) student_employee_number
  d) date & course_number
  e) date & student_employee_number
  f) course_number & student_employee_number
  g) date & course_number & student_employee_number
"no." (if you do not know)
|: g.

/* Transitive Functional Dependencies. */

For each of the following, identify any non-key attributes (written in lower case) which depend directly upon other non-key attribute(s) rather than the key attribute(s). Respond with: "non-key(s) depends_on non-key(s)"
Separate the non-key names by only a blank.

E.g. flight: [FLIGHT_NO, city_from, city_to, departure_airport]
Acceptable Response:
"departure_airport depends_on city_from city_to."
course: [COURSE_NUMBER, duration, course_name]

Enter any non-key attribute(s) which depend on other non-key attribute(s).
"end."
|: end.

session: [COURSE_NUMBER, DATE, cost_material, cost_labor, man_hrs]

Enter any non-key attribute(s) which depend on other non-key attribute(s).
"end."
|: cost_labor depends on man_hrs.

session:

Are there any other non-key attribute(s) which depend on other non-key attribute(s)?
Enter non-key attribute(s) / "end."

|: end.

employee: [EMPLOYEE_NUMBER, address, job_title, employee_name]

Enter any non-key attribute(s) which depend on other non-key attribute(s).
"end."
|: end.

students_attend_sessions:

[DATE, COURSE_NUMBER, STUDENT_EMPLOYEE_NUMBER, grade, attendance]

Enter any non-key attribute(s) which depend on other non-key attribute(s).
"end."
|: end.
--- END SESSION ---

That's all I have to ask you about.
Here is what I have:

instructors teach sessions:
   [INSTRUCTOR_EMPLOYEE_NUMBER, COURSE_NUMBER, DATE]
course: [COURSE_NUMBER,duration,course_name]
session: [COURSE_NUMBER, DATE,cost_material,man_hrs]
instructor: [INSTRUCTOR_EMPLOYEE_NUMBER]
student: [STUDENT_EMPLOYEE_NUMBER]
employee: [EMPLOYEE_NUMBER,address,job_title,employee_name]
students attend sessions:
   [DATE, COURSE_NUMBER, STUDENT_EMPLOYEE_NUMBER,attendance,grade]
man_hrs: [MAN_HRS,cost_labor]
End Relations.
APPENDIX 3 — CONSULTATION WITH DATABASE DESIGNERS

PROCEDURE

Each consultation session with an expert designer had three participants: i) the database designer; ii) one of the investigators who played the role of a hypothetical user for which a user view was to be created; and iii) a second investigator whose participation was in the role of an observer. The objectives of the research were explained and the database designer asked to carry out the design task for a hypothetical design problem. At various points the designer was asked how and why he was making certain decisions.

The following sections outlines each of the consultation sessions. Although the account is not verbatim, it serves to provide an overview of the sessions and to highlight the important points of the exercise.

DESIGNER#1

Design Session (Library Circulation Function)

User: I would like to design a database for the circulation function of a library.

Designer#1: The first thing I do is functional decomposition. Let me use your blackboard. I like to be able to erase as I go along. Now, what exactly are you interested in, in your library?

User: I want to keep track of what books are signed out - who has them at any particular time and when they are due back. I should mention that in our library system there is one central library and various branches. A user can borrow from any of the branches. I should also say that each book has different volumes and a user could borrow any one of the volumes.

Designer#1: I like to use Brown Diagrams when I work. So let me start. I'll explain the technique as I go along. I guess the first thing we need is a box for "book". There are three sections in each box. The first is the name of the entity so I'll put "BOOK" in it. The second is some identification for "BOOK". How do we identify it - ISBN?

User: Sure, that's fine. A volume of a book can belong to any branch.
Designer#1: Ok, so for book I need to identify it by concatenating "Branch", "Volume-No", and "ISBN". In the third section I put the things you need to know about book - I guess that’d be "title" and "author".

User: And the publisher.

Designer#1: ... And I have to account for the fact that a borrower could borrow more than one copy of the same book at any one time.

Designer#1: Oh, that won’t work, let me try this. See why I like to use a blackboard? (To observer) You’re not trying to take that down in pen, are you?

Designer#1: Well, I might want to try and change your opinion on that but I’ll take your word for it.

Designer#1: Usually I like to take a couple of hours and give the users an introduction to the techniques I use. Then, when I am actually doing the design, I find that they quickly understand what I am doing and can often point out mistakes that I make. Right now, I usually deal with systems analysts but I would like to be able to change that and deal directly with the users.
Observer: Why are you rubbing off that box?

Designer#1: I realize that there will never be anything in that link. Let me try like this... (Designer#1 draws a line around some other boxes.) There, that should take care of it.

Designer#1: How about historical information - would you like to be able to trace when and to whom a book has been loaned?

User: No, the most important thing is the date that the book is due.

Designer#1: What about "bad debt"?

User: Oh, I don't think we have to worry about that.

Designer#1: Do you have different classes of borrowers - for example, students and faculty members?

User: No.

Designer#1: Well if you did, this is how I would represent it...

Observer: What are you doing - just trying to see if everything is represented or are you considering the queries that might be asked?

Designer#1: Basically, just seeing that I have everything down and maybe giving a little consideration to the queries that might be asked.

User: I think that's about it. Can we think of anything else?
Observer: *Just the fact that the main library isn't represented.*

Designer#1: Oh, I missed that. *Then "Branch" gets represented like this.* (The diagram is changed to reflect the dependency of a branch on a main library . . .)

Designer#1: I didn't get a chance to show you how I would represent a relationship that is optional. I can't think of one for this example but here is one from outside . . .

General Questions Asked Of Designer#1 After Session

User: You didn't ask me what to put in, for instance, the "borrower" box. You just went ahead and put number, name, address, etc.

Designer#1: Well, I probably should have asked you but, at first, I like to get an overview of the problem. Although I could have asked you, I was relying on my own knowledge of how a library works. If it was an application area that I knew nothing about I would have to ask you. Also, eventually I document everything in a data dictionary so that I can check it with the user.

Observer: How do you chose the names you use - for example, why did you use "Borrowing-Class-Assignment" as one of the names of the boxes?

Designer#1: For that one I just knew that I needed something to indicate the link between "Borrowing-Class" and "Book-Class", so I made up the name (The user never explicitly used the word "assignment"). In general, I have a hard time making up names, especially for the relationships between the boxes. I sometimes ask the user. I also like to identify everything by numbers. That's why I automatically put "Library-Code" in as the identifier of library. I do it so that I can keep everything straight. The user usually does not see them.
User: How important is it to use a diagramming technique?

Designer#1: It's most important - it allows you to see the whole picture at once.

Designer#1: One thing that I have seen in another methodology, which is not in Brown's, is a representation for different alternatives from a certain box. I suspect that it is a deficiency in Brown's methodology. I think I've always been able to get around it by using categorization.

*Designer#1 Observes View Creation System for a Similar Task*

The designer observed a demonstration of the View Creation System. The system was used to develop a hypothetical view of a library-circulation function. The following is a list of the suggestions made by Designer#1 for improving the system.

1. When the system detects an attribute which suggests that an entity is missing (see Chapter V or Knowledge Base – Missing Entities), the system asks if an entity should exist. If the user indicates that one should exist, a new entity is created. Designer#1 suggested that this be flagged as a place where a relationship is required. He also said that he would immediately incorporate such a relationship into the design. (E.g. if author_id is an attribute of the entity book, the system suggests that author may also be an entity. If author becomes an entity, a flag is needed to indicate that a relationship is required between author and book.)

   **KNOWLEDGE BASE RULE:**
   **IF:** a "missing entity" is identified;
   **THEN:** "remember" (by adding an assertion to the system's database) that a relationship is required between the original entity whose attribute suggested a missing entity and the new entity.

   IF: the user does not voluntarily provide a relationship between the two entities when the initial set of relationships is obtained;
   THEN: prompt the user for one.

2. Many-to-many relationships are the only relationships which have attributes. They should be handled early in the design and represented as relations. This implies that mapping ratios must be determined as relationships are obtained.
KNOWLEDGE BASE RULES:

a. **Procedural Rules for Relationships**

1) Ask for relationships of the form \( A \ VP \ B \).
2) Determine if the \( A \) and \( B \) values are appropriately identified.
3) Obtain mapping ratios.
4) If appropriate, obtain relationship attributes.

b. **Mapping Ratio Rules**

1) **IF:** a relationship is a many-to-many relationship (i.e. the min/max values of \( A \) are not \((1,1)\) and the min/max values of \( B \) are not \((1,1)\));
   **THEN:**
   a) Obtain relationship attributes.
   b) Represent the relationship by a relation.

2) **IF:** a relationship is not a many-to-many relationship;
   **THEN:** depending on the types of queries that are of interest to the user, represent the relationship either by a relation or by adding the key attributes of one entity as non-key attributes of the other.

3. When detecting transitive functional dependences, there could be, for example, a non-key attribute \( Y \) which is transitively dependent on another non-key attribute \( Att \_X \). If \( X \) is one of the "key indicator attributes" (i.e. often used in identifying an entity), Designer\#1 suggested that the name of the new relation which is created to resolve the functional dependency be called \( Att \). (\( Att \_X \) was used as the relation name in the version of the system Designer\#1 observed.) (E.g. If \( publisher\_city \) is transitively dependent on \( publisher\_name \); then a new relation, called \( publisher \), should be constructed with key attribute \( publisher\_name \) and non-key attribute \( publisher\_city \).)

**KNOWLEDGE BASE RULE:**

**IF:** a non-key \( Y \) is transitively dependent on a non-key \( Z \);
**THEN:**

**IF:** \( Z \) is of the form \( Att \_X \) where \( X \) is one of the "key indicator attributes";
**THEN:** the name of the new relation is \( Att \).
**OTHERWISE:** the name of the new relation is \( Z \).
Comments

Besides the specific suggestions made for the View Creation System, the following points summarize the most important issues of the consultation session with Designer#1.

1. The designer feels that the users he deals with (although they are mostly systems analysts) can be taught the basics of database design which are needed for representing user views.
2. The design process is an interative one with an overview obtained first and successive refinements required.
3. The designer concentrates on obtaining the keys first. Non-key attributes are not obtained until a later iteration.
4. Diagrams appear to be very important.
5. Designer#1 uses internal codes for keys which are invisible to the user.
6. The designer is concerned with historical information. He asked, at least twice, if certain pieces of information were needed for historical purposes.
7. Designer#1 said he did not mind "overdesigning".
8. Many to many relationships were identified quickly and appropriate representations were constructed using the diagramming technique.
9. There may be some issues on which the designer disagrees with the user. Although the designer might try to persuade the user to make a change, he respects the fact that the user knows more about the problem than he does.
10. The designer used general knowledge to augment the basic design; e.g. "I have to allow for the same borrower having multiple copies of a book out at any one time".
11. The designer automatically generated some of the entity attributes – e.g. name, address, and phone_number for the entity borrower.

DESIGNER#2

Design Session (Library Circulation Function)

User: My position in life is that I work for a library. Each library has different branches and I work for one of the branches. I want to keep track of the books that are in the library - I want to keep track of my book inventory. I want to know who has a book and when it is due back. We usually refer to different copies of a book as volumes. I'd like to be able to ask a query such as: "Do you have a copy of a book?".

Designer#2: You want a database set up to support the things that you do at a
circulation function at a library. That sounds simple, but I have a hunch that it means more. It isn't just in and out. "Branch" seems like a strange concept to me.

User: We have to make sure that we have legal borrowers. A branch can issue cards to borrowers and then we can let them take out books. The date that a book is due depends on the card because the borrower must have one and the category of the book - not the category of the borrower. So the things we are interested in are: the category of the book and whether or not it is a recent acquisition. We have to make a record of the fact that a book is borrowed; we have to "undo" the operation when a book is returned; and we have to be able to trace an overdue book.

(Designer writes words and phrases on the blackboard - e.g. "A LOAN overdue"; "BORROW", "BORROWER card"; "BRANCH", "BOOK overdue"; "Category BOOK (fiction, etc., loan length)"; "Recent Acquisition"; "LOAN"; "book volume").

Designer#2: I have written down some things. "BOOK" seem to be the most important but you also say "copy".

User: A volume is the same as a copy of a book. We usually like to use "volume".

Designer#2: "Recent Acquisition" - that only relates to the first one?

User: Yes.

Designer#2: Now, the conceptual object "BOOK" - includes magazines?

User: That's a type of book.

Designer#2: I don't want someone to borrow more than one volume. A "BORROWER" is someone I know nothing about or is signed up already. Is there any other kind?
Designer#2: There is no such "real" information when a borrower gets together with a book. How about "LOAN" - is that a good word?

User: It's fine.

Designer#2: So, if we look at a book, we want to see if it's on loan and, if it's on loan, who has it.

User: Date due is very important - we're looking for overdue books.

Designer#2: By volume, not by book.

Designer#2: Is there anything else about a loan? I can bring a book back to any branch?

User: You mentioned the location of the book. . .

User: The first assumption is that it's in the branch where it belongs.

Designer#2: Could it be at another branch?

User: It's still considered on loan when it's at another branch. We have a shuttle that carries books back and forth between branches but we've never kept track of that in the past.

User: There could be another location then. A book could be in for repairs, at a branch, on loan or in transit (funny word for it). (Designer#2 writes this list on the board).
Designer#2: Let's go back. I'll draw a picture. We have categories of books. In these categories we could have lots of books. My symbol is like this. Could a book or a magazine be in multiple categories?

User: We could have one in juvenile or in adult or they all could be together. Juvenile is a special case. We could think of it as a branch within a branch.

Designer#2 Leave it for now - but it worries me a bit. Now, we'll see what we can do and how we'll change some things. One thing that we could add which shouldn't be tough is new categories, for example, "big" and "little" versus "fiction" and "nonfiction" and combinations of these. What we are saying now is that a book is in one and only one category.

User: Yes.

Designer#2: Anything else?

Designer#2: Back to the picture - which helps me a lot . . . You chose to call them volumes . . . Can you have a category with no book in it?

User: No.

Designer#2: What if someone asked for a book but you don't have it yet?

Designer#2: I'll just put a little zero down here (to indicate that it may or may not exist). . . . Can a borrower ask for any number of books or is he restricted?

User: No restrictions.

Designer#2: We'll come back to that again too.
Designer#2: (Summarizes)

What we are saying now is: a book may have copies and it may have a due date. If it has a due date, it has a borrower. I'm making some rules here - for example, a book can be borrowed by only one borrower at any given time. (Designer#2 points to his diagram and explains how he represents the rule.)

Designer#2: We have a branch out there. Maybe it has a phone-number, director, etc.

Designer#2: Say you had two copies of each book at each of seven branches. Do the numbers go 1, 2, 3, etc. up to 14 or does each branch have numbers 1 and 2.

User: Each branch has numbers 1 and 2.

Designer#2: So we have a little hierarchy like this with copies 1 and 2 for each branch.

User: True.

Designer#2: I am concerned with doing it that way. I see some problems with it.

Designer#2: One thing I wanted to come back to was the juvenile category. You said that it was like a branch within a branch . . .

Designer#2: So it's another special king of category - J!A. That's one way of looking at it. I'm not sure that it is right. It bothers me. Do you wany to worry about it some more? It still bothers me . . .
Designer#2: With this can we do the things you want? Can we list if its overdue and who has it? If a borrower comes in and asks for a book, can we see if it is in or out and in what particular location . . .

Any other functions?

Designer#2: Any historical information?

User: No.

Designer#2: Here's a problem . . . There's still a funny concept in there. It's not clear to me that a branch number which identifies a copy is clearly separated from the branch . . .

Ok, this line has to go back in. I'm happy now. A book does not have to have a branch yet or a copy. A branch even could have no books. So that's what was wrong.

Designer#2: Can I go away now and add some more details? I'll make a few changes here and there but I have a place to hang some things. As long as we have the structure, it's ok.

User: If I understand what you've done, it seems ok to me.
General Questions Asked Of Designer#2 After Session

User: What do you do?

Designer#2: If it's small, I put together some screens and let the user play with it. If it is a lot bigger, I do some traditional systems analysis things with it.

User: Here, we were only doing a view for me.

Designer#2: I write up textual descriptions. I design down and remove some boxes. I turn everything into entities. I'm going to end up with relations for them anyway, so I treat them all the same to begin with. A relationship is a "thing"; a "reserve" is a "thing" and it turns out that it is always a thing that the user has. Besides I only have one figure (in the diagram).

User: What about keys? You didn't ask me about them?

Designer#2: I have a good idea of a key but I'd look at some examples and see if I can determine the key. If I have all of your actual entities, I probably can do it. Then I'd check back.

User: What does your notation mean?

Designer#2: ... Zeros are generally costly things to write down ...

User: Some links do not have 1's in them.

Designer#2: I was careless, I should have put ...

Observer: Do you work with users or systems analysts?

Designer#2: With the user. I adopt to the user ... Generally, I spend some time explaining what data modelling is and give some examples, etc. I worry about possible future changes. In some cases, the users just pick it up as I go along. Sometimes they tell you where you need crows feet, etc.

I make lists of what is of interest and what I can identify. I consider future changes. Some questions I poked around but you said "no". I may disagree.

Definitions are difficult sometimes when things are new to me. It takes a lot of time. A lot of judgment is needed and a feel for whether or not you have it. I spend some time confirming.

User: You, the designer, would take a first cut and then come back and check with me?
Designer\#2: Yes.

**Designer\#2 Observes View Creation System for a Similar Task**

The following summarizes Designer\#2 comments on the current version of the View Creation System. The version he observed incorporated the rules that were obtained from Designer\#1.

1. Designer\#2 thought that the user would not understand the instructions for functional dependencies. He suggested that the questions to the user be of the form: *"If you wanted to know information about .... what must you have?".*

**KNOWLEDGE BASE RULES:**

a. **IF:** a relation has two key attributes, $\text{KeyAttribute}_1$ and $\text{KeyAttribute}_2$, and at least one non-key attribute;
   **THEN:** for each non-key attribute ask which of the following is needed before the value of the attribute is known:
   - a) $\text{KeyAttribute}_1$
   - b) $\text{KeyAttribute}_2$
   - c) $\text{KeyAttribute}_1$ and $\text{KeyAttribute}_2$.

   **IF:** the user chooses either of the first two options;
   **THEN:** a partial dependency has been identified.

b. **IF:** a relation has three key attributes, $\text{KeyAttribute}_1$, $\text{KeyAttribute}_2$, and $\text{KeyAttribute}_3$, and at least one non-key attribute;
   **THEN:** for each non-key attributes ask which of the following is needed before the value of the attribute is known:
   - a) $\text{KeyAttribute}_1$
   - b) $\text{KeyAttribute}_2$
   - c) $\text{KeyAttribute}_3$
   - d) $\text{KeyAttribute}_1$ and $\text{KeyAttribute}_2$
   - e) $\text{KeyAttribute}_1$ and $\text{KeyAttribute}_3$
   - f) $\text{KeyAttribute}_2$ and $\text{KeyAttribute}_3$
   - g) $\text{KeyAttribute}_1$, $\text{KeyAttribute}_2$ and $\text{KeyAttribute}_3$.

   **IF:** the user chooses any of the first six options;
   **THEN:** a partial dependency has been identified.

c. **IF:** a relation has four or more key attributes and at least one non-key attribute;
   **THEN:** ask the user to identify any non-key attributes which depend on
only part of the key instead of the complete key. Any non-key which meets this criteria has a partial dependency.

2. Designer#2 indicated that consistency checking is very important. In the View Creation System, this translates, for example, into using Synonym Tables to verify that the user is not using a term which has already been identified as a synonym for another term. (This issue had previously been discussed but not yet implemented).

   KNOWLEDGE BASE RULE:
   IF: the user provides a term which was previously identified as a synonym;
   THEN: ask whether the user made a mistake or if new information has been provided.

3. The designer was asked to consider how functional the system could be without making it application-specific. Designer#2 felt that, as long as the user understands the system’s questions, the system can serve its purpose.

Comments

1. Designer#2 recognized that the problem could be more complicated than it initially seemed.

2. He began with simply writing down a list of the terms he thought were important from listening to the user talk about the application. Later he used a diagram.

3. He treats both entities and relationships as entities.

4. He tried to understand how the user was managing his business. Sometimes he expressed concern for the way that current operations were being carried out and he made suggestions for improving them. In this sense, he acted more as a business consultant than a database designer.

5. His approach was not only iterative, but he purposely recycled. Several times he said: "We'll come back to that."

6. He didn’t explicitly ask how an entity could be identified. He preferred to identify the key himself and verify it later.

7. Attributes of entities were not obtained until one of the final iterations.

8. He automatically generated some of the attributes for an entity based on his own knowledge.

9. He deals directly with end-users.
10. He tried to prompt the user for other requirements by repeatedly asking: "What else would you be interested in?".

11. Designer#2 stressed that he worked in a "top-down" approach – obtaining an overview of the situation first and then later refining it.

12. He asked about the functions of other users in the application domain and whether or not they would be interested in some of the things that were not important to the user he was interviewing.

**DESIGNER#3**

*Design Session (Student Advisor Function)*

**User:** My role is as an advisor to students in a faculty. A student has a certain program and I must make sure that his program is viable and meets the requirements of the university. My view is that I need something to support my program approval task.

**Designer#3:** I am not familiar with anything that you are telling me about whereas I sometimes have a bias. You are interested in knowing about a student. You need to know his prerequisites, right?

(Designer#3 draws a box for "COURSES", a box for "STUDENT" and a line between the two boxes with "prerequisites" written by it.)

**User:** Every course can have prerequisites. It's not necessary that a course have a prerequisite but they often do.

**Designer#3:** Are you interested in knowing his objective?

**User:** A student is in a department and that department offers a few degrees so it could be bachelors, etc. There could be only a small number, say four or five degrees offered by the department that the student is in.

**Designer#3:** So the degree program has its course options.

**User:** Right - there could be, as requirements for a degree, three courses in one area or something like that.
Designer#3: I'm trying to find out what you are really trying to achieve because I need that to map back to you what I think you are trying to do.

User: I have to do a task - a program approval task. I have to find out if a student has the right courses.

Designer#3: So the end product is, for this student, a set of courses... The student probably has a student number or a student id. I need a picture to get some idea of what we are talking about. Are you familiar with my diagramming techniques?

User: What is it?

Designer#3: Bob Brown's ... I have to go into more detail then. For a student I need a student id. I have my own naming convention. I want to know the standard things - name, etc. You want to know the student number and the courses. Are you interested in what his grade is when he has taken a course? ... I assume you want to know what courses he can take given his history. I assume that he may not have ...

User: There are two reasons why I might want to know about the student's history: to see if he has taken the right courses in the past and to see if they are relevant to his program.

Designer#3: Is there any significance to the marks a student got in his courses or just whether or not he passed.

User: I'd like to know the mark.

Designer#3: Is it important when the courses are taken?

Designer#3: I tend to separate out the things that you are talking about until I am comfortable. Then I put them together.

Designer#3: If you look at the keys here they really are the same one. When you have the system, you might want to have the option of having the same keys corresponding to different relations.
User: I am not sure of the courses students want to take. A student may say that he wants to take a certain five courses so we'll look at them one by one.

Designer#3: So what you want to do is go into the course. Obviously you want to know the name and the prerequisite. Could there be more than one prerequisite?

User: There could be zero or more.

Designer#3: Are there other kinds of prerequisites (besides courses)?

User: Just limit it to the prerequisite courses. I know whether or not he is in a program and the other prerequisites are easy to deal with manually.

Designer#3: If I was an analyst, I would not accept that argument. I'd use free-form for that so you would have a prerequisite code. You want to be able to deal with these kinds of circumstances. So you have other "non-course" prerequisites.

User: Is this exclusive?

Designer#3: I am normalizing as I go. I take the view and go away with it. I then validate it with the system analyst...

When you have a course, you have a prerequisite which has a prerequisite type in it. This allows you to have an additional code or test to describe it. Here, you can deal with a situation where there are many courses...

Looking at the whole thing there might be a different solution to it. The other way is... How you solve the problem depends on your business needs. The real issue when you model is how often do you get text and whether or not it is only for two or three situations. The more you model, the more accurate and fail-safe the solution is.
Designer#3: The key to this thing is "Course-id" and "Course-relation-id". You will probably end up with more than one relation between courses. Another relation that you might have is a substitute for this course under this type of circumstance. What I am driving at is that you might have more in your key here. Another solution might be to split out the course type of things. You really need to look in more detail at which way you want to go. You must look at the attributes in the data group ... I use different data groups. When I look at two things, I look at the actual attributes of those data groups. I see if I can collapse them into one... I might use something that communicates better when I talk to the user. There is no sense in coming up with a smart construct if people who have to develop it can't deal with it...

One thing you might need is a minimum mark?

User: No, I don't need it.

Designer#3: You ask questions and try to think of things yourself which you verify with the user. Even if something happens in only five percent of the cases, you will end up having it.

I would stop here and look at attributes in the next iteration to see whether or not nulls are allowed.

Designer#3: One other thing. I want to find all the things that translate into attributes. I'll identify a few myself and fabricate a few situations and ask you about them.

Designer#3: Suppose each course has to have a department which is responsible for
it. This brings up another aspect. "Responsible department" is really a relation between course and department. It is part of the domain of department. You as a user want it as an attribute. I as an analyst have to worry about...

You have a many-to-many relationship here. Maybe you can't add a course without having a department responsible for it. Maybe course id's are assigned without having a department responsible for it. I want to capture whether or not it is an optional relation...

I also also want to be able to name the relations and define the cardinality between the boxes.

User: How do you decide on a key? You used "student-no". What about "name"?

Designer#3: I use numbers. When I create a box or a data group, I use computer-generated id's so there is no tampering with the numbers. Otherwise, when you look at an attribute, you have to say whether or not it is unique. Is it an alternative access path? Are nulls allowed?

Designer#3: I’m not sure where to go from here.

User: Students, courses, prerequisites, and student-history. What else?

Designer#3: First look at what a student has. Then look at the course. What happens when a student selects a course?

User: A student suggests a set of courses. I have to approve it. I have to verify that the rules are met.

Designer#3: Do you want to record what his program will be?
User: Yes.

Designer#3: So what you want to do is... I'm not sure what to do with the program yet. I have an attribute of the student which tells what program he is in. This is courses taken. This is courses planned. In the end this will be resolved. Now, for clarification, I have two boxes.

Designer#3: "Student-id" - what do you want to record about it?

User: Just that it exists.

Designer#3: Do you want to record what you advised?

User: No, I only advise a program. There are other things missing. The program has requirements. Somehow I want to show that the courses the student has taken and those he will take will somehow combine to meet the requirements of the degree.

(Designer#3 draws a Program box.)

Designer#3: For program you have total credits, etc. There is something about the courses, isn't there?

Designer#3: When you have a student and he is in one program, he has to have a certain amount of courses in one or another program?

User: Yes.
Designer#3: So, you have the administrative department which administers the course and the program which requires the course?

User: Every program has a department responsible for it and every program has courses which comprise it. There are requirements and electives - some within and some outside the department.

User: There are certain specific courses, certain courses within the department and some are unconstrained.

Designer#3: You are giving me information that is not captured. We need business rules. These dependencies are so complex that we can't express them yet.

Designer#3: You probably end up with some intersection data here which will give you... That's when you need the intersection. You have to see if "department" as you use it will not end up being referred to as something different. Look at the organization as a whole when you consider terminology.

User: "Department" is certainly an official term here.

Designer#3: The other thing is that you have this relationship between those two. You might have more than one relation between a student and a program.

User: No.

Designer#3: Yes, but what I implied is.....

Designer#3: Can a course be in more than one program?

User: Oh sure.
Designer#3: Then you need this distinction....

... 

User: I think that the overall structure is down.

Designer#3: Yes, but it might change. You must go into a lot of detail on that issue.

Designer#3 Observes View Creation System for a Similar Task

The following is a summary of the Designer#3’s comments and suggestions for improvements to the View Creation System.

1. Designer#3 thought that the system was not appropriate for an end-user because it uses too much database design terminology. He felt that the interface does not allow the user to express himself as he normally would when discussing his business application.

2. Designer#3 suggested that any time a "key indicator attribute" (see Knowledge Base – Entities) appears as an attribute of an entity, the attribute name should be prefixed by the entity name.

   KNOWLEDGE BASE RULE:
   IF: an attribute of an entity is a "key indicator attribute";
   THEN: prefix the attribute name with the entity name. (i.e. if attribute Att is a "key indicator attribute" then change it to EntityName_Att.)

3. Each entity must have a key. Designer#3 thought that the system should generate a unique number for a key instead of asking the user to identify key attributes. The generated number would be invisible to the user.

   KNOWLEDGE BASE RULE:
   IF: an entity Ent is identified;
   THEN: generate a unique number which will serve as the entity key.

4. Designer#3 stated that greater use should be made of class words such as name, percent, amount, etc. This implies that more general knowledge should be incorporated into the system’s knowledge base.

5. In order to obtain all the relationships, Designer#3 suggested that the user be asked
(ideally by using a menu) to consider each entity in relation to each other entity. The user should be asked if a relationship exists between each pair of entities and, if so, to identify the relationship.

KNOWLEDGE BASE RULE:
IF: entity, Ent_i;
THEN: ask if Ent_j is related to each Ent_i for i ≠ j. If they are related, obtain corresponding relationships.

This rule was not included in the system’s knowledge base because it was not supported by the other designers.

6. The system categorizes have/has relationships by querying the user. (See Knowledge Base – Have/Has Relationships). The designer suggested that, for a relationship of the form A have/has B, other relationships involving A and B should be examined to see if it is possible to deduce the type of have/has relationship without explicitly asking the user.

7. Designer#3 suggested that some consideration be given to the individual fields of the entities. For any field (e.g. address) which can be broken down into finer detail, the user should be asked whether he would like to provide the finer detail at the current point in the design session or at a later point. This idea was suggested because Designer#3 felt that it would force the user to give more consideration to an attribute early in the design exercise. This could avoid ambiguities at a later phase.

Comments

1. Designer#3 relied heavily on his knowledge of the application and used it to prompt the user for requirements.

2. The designer uses diagramming techniques (Brown Diagrams).

3. Designer#3 prefers to generate numerical codes to serve as unique identifiers for entities instead of asking the user for keys.

4. The designer decomposes a problem until he completely understands it. Then he aggregates parts of it.

5. The designer fabricated situations in an effort to prompt the user for information requirements.

6. The designer does not obtain attributes until one of the final iterations.

7. Designer#3 automatically generated some of the attributes for an entity based on his
own knowledge. He also made assumptions about the application which he confirmed with the user.

8. Designer#3 is concerned with how things relate to a business application. At one point he stated that business rules should be identified in order to obtain an adequate representation of the problem.

DESIGNER#4

Design Session (Student Advisor Function)

User: I am interested in a system to support my advising students in their programs. A student comes in to see me and he is in a degree program. The program has requirements. The student has taken some of the requirements and he has his projected courses for the year. I have to be able to tell if a student has the prerequisites he needs and if he meets meets the requirements of the program, etc.

Designer#4: My first move is to find out what is in place. Do you have a conceptual data model?

User: No.

Designer#4: What things are you looking for? What are the entities and relationships? You mentioned some entities already. The key one is student. And you are another one – an advisor. Another thing is a degree program. Is it a collection of courses?

User: It's a set of requirements.

Designer#4: The basic requirements are mapped through courses. Let me put courses in here... You have two relationships here... I have a one to many relationship. A program can have multiple courses and courses can have many programs so I have a many to many relationship.

User: Do you start out by giving the user a tutorial?

Designer#4 Yes.
Let me see ... When a student is in a program, is it one program?

User: Yes.

Designer#4: Not two programs?

User: No.

Designer#4: So we have a solid relation like this. The student can take many courses ... What else did you give me?

Designer#4: A student has one advisor; an advisor has multiple students ... Have we missed anything? ...

I am exploring your information requirements but not in detail. I have a specific query on a student and a non-specific query on courses. I can test the model by asking the queries. I am doing path analysis. Imagine these as relations ...

User: I wanted to see if a student has taken prerequisites of a course.

Designer#4: I believe I have the paths in place to get where I want. I can get what's been taken and what is projected. Anything else?

User: No, it seems that all the important relationships are connected...

That's the end of the conceptual model, so let's get into the views. A student comes in. What would be the key thing you are interested in?

User: I want to choose a set of courses for a student to take this year given the prerequisites and what he has taken.

Designer#4: You want to know what program he is in. What are the courses? Do you want to be able to look at it on a screen?

User: That's equivalent to what I do manually.

Designer#4: Scheduling, etc.?

User: That is someone else's problem.

Designer#4: You want to look at a student, his degree program, really degree courses, right? Once you have that, what else might you want?
User: I'd probably want to see his marks.

Designer#4: Do you want the system to do some "WHAT-IFing"? Do you want the system to do the prerequisite test?

User: That would be very nice.

Designer#4: There is a piece missing. We need course offering. This gets messy . . . A course can be offered at only one point in time. It gets into the scheduling question.

User: Sometimes the courses are not scheduled until later. We need to know whether or not a course is offered in a year.

Designer#4: Put in course availability. You want to know what courses are available before you assign a student to it and you do a prerequisites test. When you make a choice on selecting a course, you would like to know the mark?

User: Yes.

Designer#4: Ok, it looks like a single view. Anything else?

Designer#4: Do you need statistical information? For example, how many people in the program?

User: That is someone else's view.

Designer#4: I would now do a logical model. I want to refine from the entity down to the entity attributes. I'll use relational schema notation...
It is most important to establish the one to many relationships.

Let's start with the key entity – it's got to be student. How will we identify it? Student number? We have naming conventions. I'll put the name of the relation in front of the name of the attribute, except for the degree program so I can identify foreign keys.

User: I need a student name too. I don't remember student numbers.

Designer#4: I have composite keys. I'll choose this one because I know that it is easier.

Designer#4: This thing should come out normalized, or close to it. The problem is if we get some 4th Normal Form . . .

I talk to my users like this.

User: Systems analysts or end-users?

Designer#4: Both, end-users too.

User: Engineers?

Designer#4: Yes, intelligent users.

Designer#4: Next thing you need is to refer to course. Let's call it C-NO and probably C-Name. That's about all you need here. We need a relationship between student and course. I'm going to call it S-C. The link is the course grade. That's what you are interested in. I can't think of anything else. Leave it as it is for now.
User: What about advisor?

Designer#4: That's a real easy one ... There should be a security check. If you are not the advisor, you shouldn't be able to see the information on a student.

User: Just a field in student?

Designer#4: Yes, put the link in there.

Designer#4: One more and we are done. For course availability we need course number and course date. Let's make a business rule here: a course is offered only once during a semester.

Designer#4: For courses you have name, etc.

User: I'd include the units.

Designer#4: Call it C-Units, ok?

User: Ok.

Designer#4: Do you want to sum the units? Is that a meaningful measure?

User: Are you just checking that you have all the attributes?

Designer#4: Yes.
General Questions Asked of Designer#4 After Session

Designer#4: I start with entities. It gets us to normalization faster. I can draw a key table.

User: How did you decide on primary keys? You didn’t ask me.

Designer#4: It is not obvious. I’m thinking of the machine implementation.

User: What do you think of the idea of attaching to each relation an artificial key which is used exclusively within the system?

Designer#4: Here I have done that. I have a surrogate key. I only use it when I have problems with the data.

User: What would you do if I asked for course-name as an attribute of the relation student-course?

Designer#4: Course-name depends on student-number. . . . That will waste a lot of space so I’d ask if I can put it up here.

User: What about standing?

Designer#4: For a given grade, you have one standing?

User: Yes.

Designer#4: So in S–C (student–course) you want to put standing. It’s for ranges, therefore, you don’t want it in your logical model. Put it in a table where you can get it. In high volume situations, you may want to put it in the relation to save scanning each time. From a physical implementation, maybe it should be left in. It’s a complex transitive dependency. I don’t mind leaving it in there. That’s just a feeling.

I think we’ve got a clean model.

Designer#4 Observes View Creation System for a Similar Task

1. Designer#4 thought that the instructions for relationship attributes and functional dependencies were too difficult for a user to understand. (They were subsequently modified.)

2. The designer suggested that the system be used to inform the user what kind of information already exists in an organization’s database. This would be done by
performing string scans on the items of the user's information requirements and comparing them with those in the Data Dictionary. If items are similar, the user would be informed. Hence, a potential reason for using the system would be to ensure that a user is aware of existing data. This could assist in decreasing the amount of redundant information in an organization's database.

3. When detecting synonyms, Designer#4 suggested that the user be allowed to respond by entering a number which corresponds to the replacement term he would like to use instead of asking him to re-enter the name of the synonym.

4. Designer#4 liked the naming conventions that are used by the system. In his opinion, the naming of relations, entities and attributes is one of the most difficult parts of database design.

Comments

1. Designer#4 first obtained the entities and their key values. During the final iteration, he obtained attributes. This approach is not adopted in the View Creation System because the system can determine some of the relationships which will be required by examining the attributes as they are provided for the entities. (See Chapter V – Entities.)

2. Designer#4 gives his users a tutorial on database design before beginning a design session. He works with both systems analysts and end-users (who are usually engineers).

3. The designer uses attribute names as keys.

4. Designer#4 thought that eight relations was a reasonable size for a user view.

5. He was willing to violate normalization principles for efficiency reasons.

6. Designer#4 developed a conceptual model, then a more detailed logical model. He was also concerned with business rules.
APPENDIX 4 — TESTING SESSIONS

USER#1

User#1 was familiar with database design concepts and terminology with a level of knowledge comparable to that of a systems analyst.

Application

The application chosen by User#1 was a training database. His objective was to design a database which would "provide accurate and timely information regarding the training process for all employees" in the company he worked for. The user had written a proposal which outlined the various types of reports required from the database (e.g. individual employee report, supervisors report, master cost report, department report, etc.) The user chose one of these reports, the master cost report, to use as a basis for his view.

Design Session

1. Initially, the user had difficulty distinguishing between an entity and an attribute. For example, date was originally provided as an attribute of the entity course. When the system queried the user about multivalued attributes, User#1 identified date as multivalued so date became an entity. When asked for the attributes of date, the user provided year, month and day. The system then asked if any of the attributes of date were multivalued and the user indicated that they all were. Eventually, with the system's prompting, the user realized that the attributes of date were single-valued and made the appropriate adjustments.

2. The user was able to identify two transitive functional dependencies.

3. User#1 understood the system's instructions and responded appropriately to them.

4. User#1 commented that the session was beneficial because the queries posed by the system made him think very carefully about his database requirements.

User#1's View

student: [STUDENT_EMP_NUMBER, student_name]

instructor: [INSTRUCTOR_EMP_NUMBER, instructor_name]

session: [YEAR, MONTH, DAY, COURSE_NAME, cost_mat, man_hrs]

date: [DAY, MONTH, YEAR]

course: [COURSE_NAME, duration, course_course_number]

instructors_teach_courses: [INSTRUCTOR_EMP_NUMBER, COURSE_NAME, rating]
**Evaluation**

1. The system helped the user realize that *date* should not be an attribute of the entity *course*. However, *date* became a separate entity whereas a human would have made it an attribute of the entity *session*. The relation *date_associated_with_session* is redundant because *date* is needed to uniquely identify an instance of *session* and therefore is part of the key of *session*. This did not invalidate the output.

2. The relation *sessions_instance_of_courses* is redundant. Since *session* is an instance of *course*, the key of *session* includes the key of *course*. (See *Primary Keys - Instance_of Relationships*.) In general, for a relationship *A instance_of B*, there is one and only one *B* value for each value of *A* so the min/max values of *B* are (1,1). Thus, an *instance_of* relationship, and hence the relation representing it, has no relationship attributes. (See *Mapping Ratios*.) Therefore, since the relation *sessions_instance_of_courses* has the same key attributes as *course* and no non-key attributes, it is clearly redundant.

3. One of the relations is not quite correct. The relation *instructors_teach_courses* was constructed to represent the relationship *instructors teach courses*. Based on general knowledge of the application, the relationship *instructors teach sessions*, either as a replacement for the original relationship or in addition to the original relationship, would have been more appropriate. Then the representation for this relationship would have provided information about the application at a more appropriate level of detail.

**System Modification**

Two new rules were added to the system’s knowledge base.

1. **IF:** a relationship *A have/has_of B* is classified as an *instance_of* relationship and it has a corresponding assertion which indicates that it is to be represented as a relation; **THEN:** delete the assertion and do not construct a new relation. (See *Knowledge Base - Have/Has Relationships*.)
2. **IF:** a relationship is of the form \( A \) is a / instance of \( B \);  
**THEN:** for any other entity \( X \) which occurs in a relationship with the superset entity \( B \), ask whether \( X \) should appear in a relationship with the subset entity \( A \) or with the superset entity \( B \) (or both). (See *Knowledge Base - Have/Has Relationships.*)

(In the design session with User#1, the relationship sessions instance of courses was identified. Therefore, the system should have asked which relationship(s) the user required: 1) instructors instruct sessions or 2) instructors instruct courses (or both).)

**USER#2**

User#2 had some familiarity with database concepts.

**Application**

The user was employed by a university as a project leader for Student Systems. User#2 was working on a project for implementing a new database which would be used for advising students what courses to take in order to fulfill their degree requirements.

**Design Session**

1. The user did not appear to read the instructions carefully. For example, although the system asked for entity names in their singular form, the user entered them in their plural form.

2. User#2 had some difficulty understanding the application. For example, an entity name which the user later tried to add as the name of an attribute for another entity referred to two different concepts. Where the latter should have appeared in a relationship, the former was employed.

3. The user originally made an error when providing answers to the queries which obtained mapping ratios for a relationship. The error appeared to be a result of the user not reading or understanding the query. The relationship was deleted and added again during one of the modification modules.

4. User#2 identified both a partial and a transitive functional dependency.

5. User#2 suggested that the system include a small tutorial. It was also suggested that the system ask the user to provide one or two sentences which describe each entity as it is obtained. The user could refer to these descriptions to verify that the entity names are appropriately used in the relationships.
User#2’s View

students: [STUDENT_ID, address, yr, fac, program, students_name]
course: [CRSENO, SECT, COURSE_DEPT, max, obt]
requirements: [TABNO, TABNAM, TRANS, crscnt, deptcnt, rangcnt, crse, requirements_dept]
range: [FROM, TO, crseno, sect, course_dept]
siscode: [TABLENAM, KEY, desc]
students_take_courses: [STUDENT_ID, CRSENO, SECT, COURSE_DEPT]
ranges_component_of_requirements: [FROM, TO, TABNO, TABNAM, TRANS]
courses_map_into_requirements: [CRSENO, SECT, COURSE_DEPT, TABNO, TABNAM, TRANS]
table: [TABNO, tablenam]
max_obt: [MAX, OBT, stdg]

Evaluation

User#2’s view is a normalized set of relations but does not totally correspond to the user’s application. The undesirable features of the view reflect the difficulty the user had when identifying entities and attributes.

System Modification

Since the greatest problem with this session seemed to be the user’s lack of understanding of the system’s questions, the instructions were modified to highlight some of the more subtle points.

USER#3

User#3 had studied and done extensive work in data modelling.

Application

User#3 designed a user view for a teaching function.

Design Session

1. User#3 had no difficulty understanding the system and responding accordingly. In fact, he purposely made some errors so he could observe the system’s reaction.

2. The user made the following two suggestions for improving the interface.
   a. The system should display both the entity and attribute names when prompting the user for missing entities.
   b. When obtaining relationships, the user should be able to see the set of entities
before providing the next relationship.

User#3’s View

semester: [DATE]
course: [COURSE_NUMBER, description]
classroom: [CLASSROOM_NUMBER, seats]
student: [STUDENT_ID, address, programme, student_name]
teacher: [SIN, first_name, last_name]
students_take_courses: [STUDENT_ID, COURSE_NUMBER, grade]
teacher_teaches_courses: [SIN, COURSE_NUMBER]
courses_are_offered_in_classrooms: [COURSE_NUMBER, CLASSROOM_NUMBER, time]
courses_are_offered_during_a_semester: [COURSE_NUMBER, DATE, number_of_sessions]

Evaluation

The user view produced is correct and void of any undesirable properties.

System Modification

The suggested modifications to the interface were implemented.

USER#4

User#4 can be classified as a naive user.

Application

The user worked for a railway and wanted to be able to obtain information on the rail traffic which moved from an origin to a destination. For each origin there could be multiple destinations and vice versa.

Design Session

1. The user had some difficulty distinguishing whether something should be an entity or an attribute. The major problem was that, what the user initially thought were attributes of the entity origin, were really attributes of a relationship between the entities origin and destination. The system’s queries caused the user to realize that the attributes originally provided for origin were not attributes of that entity. However, the system was not able to help the user realize that they were attributes of the relationship.
2. User#4 had difficulty understanding how he was suppose to respond when the system asked him to identify transitive functional dependencies. Eventually, a transitive dependencies was identified.

3. The user said that the design session helped him to structure his information requirements.

4. User#4 suggested that the system be augmented with a tutorial.

**User#4’s View**

origin: [ORIGIN_NAME]
destination_territory: [DESTINATION_TERRITORY_NAME]
origins_move_traffic_to_destination_territories:
[ORIGIN_NAME, DESTINATION_TERRITORY_NAME,cars,tons,miles,bcdiv]
cars_tons: [CARS, TONS, revenue]

**Evaluation**

1. The user’s view is small.
2. The set of relations produced is correct and does not possess any undesirable properties.

**USER#5**

User#5 could be considered a naive user.

**Application**

The user wished to design a database which would provide information on equipment that his company configured and serviced.

**Design Session**

1. Initially, User#5 provided multivalued attributes which were converted to entities. When asked for the attributes of two of these new entities, the user indicated that he could not supply any and that the entities should be deleted. The information was later captured in a relationship and appropriately included in the design.

2. When the system asked the user to provide a unique identifier for one of the entities, User#5 gave an attribute which was not part of the initial set of attributes for that entity. The system added the attribute and the entity was correctly identified.
3. A partial functional dependency was identified and resolved. Manufacturer and unit_name were originally given as attributes of unit. Since they both depend only on model, a separate relation, product_name, was formed.

4. At the end of the design session, User#5 stated that the output captured all the information he had provided. However, it did not cover all aspects of his information requirements because he neglected to include one dimension of his design problem.

User#5’s View

configuration: [CONFIGURATION_ID]

service_event: [SERVICE_EVENT_NUMBER, action, date, service_person, comments, model, serial]

unit: [MODEL, SERIAL]

units_component_of_configurations: [MODEL, SERIAL, CONFIGURATION_ID]

product_name: [MODEL, manufacturer, unit_name]

Evaluation

1. User#5’s view is correct and free of any undesirable properties.

2. The view does not reflect all of the user’s requirements. In the user’s application, a configuration is composed of modules which in turn are composed of pairs of units (an inside_unit and an outside_unit). The user really wanted to capture a set of hierarchical relationships. The required relationships are listed below.

1) configurations have modules
2) modules have in_units
3) modules have out_units
4) in_unit is_a unit
5) out_unit is_a unit

The system was tried using these relationships and the following initial set of entities and attributes obtained. (Note that this time the user specified that unit is needed to uniquely identify service_event. The first time he specified a relationship units have service_events which was represented by adding the key attributes of unit as non-key attributes of service_event.)

module: [MODULE_ID]
in_unit: [IN_UNIT_ID]
out_unit: [OUT_UNIT_ID]

service_event: [UNIT, NUMBER, date, service_person, service_event_description]
configuration: [CONFIGURATION_ID]
unit: [MODEL, SERIAL, price, date_of_acquisition, unit_description]

After primary keys were incorporated and relationships represented, the following set of relations was produced.

module: [MODULE_ID, in_unit_id, out_unit_id]
in_unit: [IN_UNIT_ID, model, serial]
out_unit: [OUT_UNIT_ID, model, serial]
service_event: [SERIAL, MODEL, NUMBER, date, service_person, service_event_description]
configuration: [CONFIGURATION_ID]
unit: [MODEL, SERIAL, price, date_of_acquisition, unit_description]
modules_component_of_configurations: [MODULE_ID, CONFIGURATION_ID]

The results are not incorrect. However, the set of relations has some redundancy. In_unit and out_unit are identified by the attributes in_unit_id and out_unit_id, respectively. These were generated as identifiers because the user did not supply any attributes for either entity. Since in_unit and out_unit are both units, they could be uniquely identified by the key of unit. This was tested and the following initial set of entities and attributes obtained.

module: [MODULE_ID]
in_unit: [UNIT]
out_unit: [UNIT]
service_event: [UNIT, NUMBER, date, service_person, service_event_description]
configuration: [CONFIGURATION_ID]
unit: [MODEL, SERIAL, price, date_of_acquisition, unit_description]

The above indicates that unit is needed to uniquely identify both in_unit and out_unit instead of generating separate identifying attributes. Primary keys were then determined. This required that the attribute UNIT in in_unit, out_unit and service_event be replaced by the key attributes of unit. Relationships were represented and the following set of relations obtained.

module: [MODULE_ID, model, serial]
in_unit: [MODEL, SERIAL]
out_unit: [MODEL, SERIAL]
service_event: [MODEL, SERIAL, NUMBER, date, service_person, service_event_description]
configuration: [CONFIGURATION_ID]
unit: [MODEL, SERIAL, price, date_of_acquisition, unit_description]
modules_component_of_configurations: [MODULE_ID, CONFIGURATION_ID]
This result is incorrect because the fact that a module has two distinct units - 
in_unit and an out_unit - is not captured. The relationships modules have in_units and modules have out_units were represented by adding the key attributes of in_unit and out_unit, respectively, as non-key attributes of module. Since the key attributes (SERIAL and MODEL) are the same for both entities, each key attribute was only added once as a non-key attribute of module.

The desired output was not obtained because, when the subset entities, in_unit and out_unit in is-a relationships adopted the primary key of their superset entity, unit, the resulting primary keys of the subset entities were not distinguishable from that of the superset entity. (See Primary Keys – Is-a Relationships.) In other words, it is not possible to identify what role the subset and superset entities play in other relationships (besides the is-a relationships) involving the two entities. (These other relationships are modules have in_units and modules have out_units.) In order to make the distinction, the key attributes of in_unit and out_unit should be prefixed by their entity names. This rule was missing from the system’s knowledge base.

After making the appropriate changes to the knowledge base, the following set of relations was obtained which adequately captures the user’s information requirements.

module: [MODULE_ID, in_unit_model, in_unit_serial, out_unit_model, out_unit_serial]
in_unit: [IN_UNIT_MODEL, IN_UNIT_SERIAL]
out_unit: [OUT_UNIT_MODEL, OUT_UNIT_SERIAL]
service_event: [MODEL, SERIAL, NUMBER, date, service_person, service_event_description]
configuration: [CONFIGURATION_ID]
unit: [MODEL, SERIAL, price, date_of_acquisition, unit_description]
modules_component_of_configurations: [MODULE_ID, CONFIGURATION_ID]

System Modification

The following rule was added to enable the system to correctly model hierarchical relationships.

**IF:** a subset entity in an is-a relationship adopts the primary key of its superset entity;  
**THEN:** prefix the key attributes of the subset entity with the entity’s name.

**E.g.** Suppose manager is-a employee and manager adopts the primary key, employee_number of its superset entity, employee. Then if a relationship, employees work for managers is to be modelled, the desired output will be obtained:

employees_work_for_managers: [EMPLOYEE_NUMBER, MANAGER_EMPLOYEE_NUMBER].
USER#6 and USER#7

User#6 and User#7 were unfamiliar with database concepts and therefore can be classified as naive users.

Application

The users worked in an insurance company and wanted a database which would provide information on clients and the damages they incur when insurance claims are made.

Design Session

1. The two users worked together to develop one user view.
2. User#6 and User#7 had no difficulty communicating with each other and both used exactly the same terminology.
3. The users had a great deal of difficulty identifying the entities in their application.
4. When the system is checking for partial functional dependencies, it displays all possible combinations of key attributes and asks the user to indicate, for each non-key, which combination is needed before its value is known. For one of the entities the users said that none of the combinations were appropriate, implying that the non-key attributes were incorrectly specified.

User#6 and User#7's View

The view is not available because it contains confidential information.

Evaluation

1. The view does not accurately represent all of the information requirements because of the difficulty the users had in understanding the concept of an entity. For example, in the final output, one of the entities should have been classified as an attribute of another.
2. A second entity should also have been classified as an attribute. Because of this, the users did not know which attribute(s) uniquely identify it so all the attributes became part of a key (due to the system's default).
3. After the design session, the users said that they would like to try again because they felt that they could apply what they learned during the session to design a better view. This implies that, for these users, a more extensive tutorial (i.e. other than the system's current set of instructions) would be appropriate. Also it is felt that, if the
users had continued, they would have eventually obtained an acceptable view as a result of the system's corrections and the insight they gained while using the system.

USER#8

User#8 had just begun to work in data modelling and had some appreciation of database design concepts. For example, the user understood the concept of an entity.

Application

User#8 wanted a database which could be used to obtain information on program specification blocks.

Design Session

1. The user gave as a key an attribute which was not already part of the entity so it was appropriately added.

2. The system identified \textit{charge} as a possible missing entity because User#8 originally provided \textit{charge\_code} as an attribute of \textit{system}. The user agreed that \textit{charge} should be a new entity and it was added. Since \textit{charge\_code} was originally provided as an identifier for \textit{system}, the user was asked to identify candidate keys for \textit{system} again.

3. There were situations where the system checked for synonyms and missing subset/superset relationships but none were identified. (See \textit{Knowledge Base - Synonyms}.)

4. The user said that she would like to be able to use the system for her next data modelling project.

User#8's View

\texttt{psb: [PSB\_NAME,io\_error,type]}
\texttt{database: [DATABASE\_NAME,access\_method]}
\texttt{segment: [SEGMENT\_NAME,processing\_option]}
\texttt{system: [SYSTEM\_NAME,contact,charge\_project]}
\texttt{charge: [CHARGE\_PROJECT,charge\_code,system\_name]}
\texttt{psb\_contains\_databases: [PSB\_NAME, DATABASE\_NAME]}
\texttt{system\_has\_databases: [DATABASE\_NAME, SYSTEM\_NAME,access]}
\texttt{databases\_contain\_segments: [DATABASE\_NAME, SEGMENT\_NAME]}
\texttt{psbs\_associated\_with\_system: [PSB\_NAME, SYSTEM\_NAME]}
\texttt{type: [TYPE,language]}
Evaluation

1. The view is correct and appropriately reflects User#8's information requirements.

2. The relation system_has_databases would have represented more of the semantics of the application if the has was properly classified. (In general, a have or has verb can be categorized as possession, component_of, instance_of or associated_with (in some other way). See Have/Has Relationships.) The relationship which gave rise to this relation had a relationship attribute so it was converted to an entity. (See Knowledge Base – Mapping Ratios.) However, the conversion was done before have/has relationships were classified. Therefore, the conversion took place too early in the session's procedure because the entity name should have later been modified to reflect the classification of the verb.

Note that the relation system_has_databases could only have been changed to:

a. system possesses databases,
b. databases component_of system,
c. databases associated_with system (in some other way).

The classification databases instance_of system is not possible. The original relationship system has databases had a relationship attribute. As proved in Chapter V (Mapping Ratios), a relationship can only have attributes if the minimum and maximum cardinalities for one of the entities are both 1. In the case of a relationship A instance_of B, there can be one and only one value of A for each value of B so the min/max values of B are (1,1). (See Chapter V - Special Verb Phrases.) Therefore, an instance_of relationship does not have relationship attributes and thus would never be converted to an entity.

Hence, the relation system_has_databases is not incorrect, nor does representing the relationship in this way lose important information (as would be the case if the has could be classified as instance_of). It does fail to capture some semantic information.

System Modification

The following rule was added.

IF: a relationship A have/has B is converted to an entity A_have/has_B because the relationship has attributes and a shorter name is not provided for the entity name;

THEN: modify the entity name at a later point to reflect the appropriate classification of have/has.