Turning Null Responses into Quality Responses

by

Mimi A. Kao

BMath(Hons.), University of Waterloo, 1984

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

© Mimi A. Kao 1986
SIMON FRASER UNIVERSITY
November 1986

All rights reserved. This thesis may not be
reproduced in whole or in part, by photocopy
or other means, without the permission of the author.

Approval

Name: Mimi A. Kao

Degree: Master of Science

Title of Thesis: Turning Null Responses into Quality Responses

Lou Hafer
Chairman

Nick Cercone
Senior Supervisor

Wo Shun Luk
Senior Supervisor

Jim Delgrande

Gordon McCalla
External Examiner

August 15, 1986
Date Approved

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

→ Turning Null Responses into Quality Responses

Author:

(signature)

MIMI KAO

(name)

NOV 21, 1986

(date)

Acknowledgements

I would like to express my deepest gratitude to my two supervisors, Prof. Nick Cercone and Prof. Wo-Shun Luk, for their devoted support and guidance during the course of this thesis. Their friendly and earnest advice and criticisms are invaluable to the realization of this thesis.

I am very grateful to Dr. Jim Delgrande for acting in my examining committee, and for always being a helpful teacher. I am also indebted to Dr. Gordon McCalla for being my external examiner. Constructive comments and suggestions from both Dr. Jim Delgrande and Dr. Gordon McCalla have helped a lot to improve the original version of this thesis.

I would also like to acknowledge the financial support received from Simon Fraser University and the National Sciences and Engineering Research Council of Canada.

Finally, I would like to take this opportunity to express my love to my parents and my family. Their immeasurable love and care has always been a source of confidence for me in my life.

Abstract

Natural language interfaces to database systems free the user from the undue formalism of learning a query language. However, the more important issue from the system's point of view, besides correctly interpreting the natural language query, is to respond correctly and cooperatively. In particular, the generation of quality responses has proven problematical in situations when null values arise. If a user's query cannot be answered by the system because of an incorrect assumption that the user has made, it would be appropriate for the system to inform the user what has gone wrong. The query, "*What grade did John Doe get in MATH100?*", might result in a null answer because John is not enrolled in MATH100. It is important for a system to realize these kinds of complementary issues in order to be truly "natural".

Most recent work on cooperative responses has relatively overlooked null value events. In this thesis, we present an initial classification of null event problems in natural language database systems, and methods for responding with appropriate answers to some classes of null events. Assuming that the database is relational and that the database query language is SQL, we develop and incorporate a knowledge base into the system based on the RM/T model, an extended relational model proposed by E. F. Codd, to furnish information for diagnosis of failed queries. The knowledge base, which is also in the relational model, consists of meta-level data information. This knowledge base provides information such as: entity concept hierarchies; generalization and aggregation hierarchies; entity relationships and entity relationship constraints; event precedence in the database domain; and knowledge about null values existing in the database. Algorithms which further explicate the function of the knowledge base model are given to demonstrate the kind of quality responses we can obtain instead of a simple null answer.

Table of Contents

List of Figures

Chapter 1
Introduction

Natural language interfaces were introduced into database management systems primarily to relieve users of the necessity to learn a formal query language, for example, SQL, in order to access data in the database. In this way, a database system could be valuable to non-expert users.

Initial work on natural language interfaces addressed problems of natural language understanding and the linguistic coverage of the query[1]. Representative example systems include the PLANES system [Waltz 78], and the LIFER system [Hendrix, Sacerdoti, Sagalowicz, and Slocum 78]. The present generation of natural language database systems emphasizes portability [Kaplan 78], [Davidson 82] and cooperative responses [Webber, Joshi, Mays, and McKeown 83].

Using natural language to access the database has freed the user from learning a specialized formal database language. However, since natural language is less restrictive, queries may be rendered to the system less precisely and responses generated may be misleading. To illustrate this point, consider the following example:

```
Q1: Did anyone get a grade of E in CMPT 101 last semester?
R1: No.

Q2: Did anyone fail CMPT 101 last semester?
```

[1]By linguistic coverage we mean the breadth of natural language expressions that can be "correctly" parsed and analysed for meaning content.

```
R2: Yes.
```

If the user believes that "E" is the failing grade, then the user might become very confused with the two different responses, R1 and R2, he received. Moreover, if the second question is not asked at all, then the user might be misled into the belief that no one failed in CMPT 101 last semester.

According to Grice's four principles of cooperative human conversation [Grice 75], a conversation is cooperative only if the speaker's responses incorporate:

1. the maxim of **quantity**: be as informative as required;
2. the maxim of **quality**: contribute only when an adequate amount of evidence is present;
3. the maxim of **relation**: be relevant;
4. the maxim of **manner**: avoid obscurity of expression, avoid ambiguity, be brief.

A natural language database (NLDB) system should not be misleading and should also provide responses according to Grice's four maxims of cooperative response. Furthermore, we want a natural language database system to be able to answer queries concerning the database structure as well. Hence, in our earlier example, the system should respond that "E" is not a "known" or "valid" value for grade.

Research interest in generating cooperative responses has taken different forms. Kalita concentrated on giving summary responses of short non-enumerative answers, which under certain circumstances are more appealing and more desirable [Kalita 84], [Kalita, Jones, and McCalla 86]. For example,

```
Q3: Which students have completed less than 5 courses?
R3: All first year students.
```

Schank and Lehnert have worked on extended responses when the user's question tends to be vague and does not reflect a very clear intention [Schank and Lehnert 79]. For example,

```
Q4: What is the pattern of withdrawals from computing science courses
    over the past 3 years?
```

In this example, "pattern of withdrawals" can mean many different things, including: the perspective on the reasons why students have chosen to withdraw from a computing science course; a description of the trends, showing the increases or decreases in the number of withdrawals; a report on the number of withdrawals in each computing science course, etc.

McCoy and McKeown attempted to generate answers to requests on the database structure and show how it is possible to automatically generate this kind of meta-knowledge[2] in a generalization hierarchy from the data itself [McCoy 82], [McKeown 82] .

All the systems mentioned above require a knowledge base of some sort[3]. Kaplan, however, worked on cooperative responses which correct user's misconceptions without using an separate knowledge base [Kaplan 78]; and Motro worked on the method for interpreting null answers by assuming the existence of necessary knowledge in the database [Motro 86]. It is our contention that the need for a general knowledge base to support the generation of a wider variety of cooperative responses is essential.

[2]Meta-level data information is information pertaining to the data in the database. Thus, beside information about how the data are organized and related, information concerning what a null value in the database mean is also a kind of meta-level data information, see chapter 4.

[3]Please refer to chapter 4 for the distinction between a knowledge base and a database.

A brief survey of appropriate natural language database systems which generate cooperative responses is presented in chapter 2; these systems demonstrate the kind of problems that are subjected to my investigation in the subsequent chapters (especially null value events). Motro's recent approach to interpreting null answers using query generalization [Motro 86] is also discussed in the last section of chapter 2 together with some other related systems. A preliminary classification of null events is developed and presented in chapter 3. This classification provides suggested focusses for the development of a general knowledge base in natural language database systems. In chapter 4, the relational knowledge base model is described for relational databases, and the advantages of this approach are discussed. To illustrate the utility of the knowledge base model, two algorithms are introduced in chapter 5 along with illustrative examples for providing quality responses to user's misconceptions. In chapter 6, the concluding chapter, some of the contributions in this research are discussed and some directions for future research are outlined.

Chapter 2
Natural Language Database Systems

We introduce several examples of natural language database systems in this chapter. These systems include PLANES [Waltz 78] and LADDER [Hendrix, Sacerdoti, Sagalowicz, and Slocum 78] in the first group of the early "tout ensemble" systems. Summary responses [Kalita 84], [Kalita, Jones, and McCalla 86], ENHANCE [McCoy 82], TEXT [McKeown 82], monitor offers [Mays 82], and CO-OP [Kaplan 78] make up the second group of cooperative response systems. Query generalization [Motro 86], [Janas 79], conceptual coverage [Finin, Goodman, and Tennant 79], discourse focus [Davidson 82], and troubleshooting [Dankel 79] are some other related systems. From this brief survey, we can observe how research interest in natural language database systems has shifted its focus from treating the total natural language interface problem to a concentration on a particular aspect of the system, that is, towards providing better quality responses.

2.1. "Tout Ensemble"

We examine two early natural language database systems that attempt to develop an inclusive natural language interface to a database system. These systems do not address particular issues in building "complete" systems, but try to demonstrate that workable natural language interfaces can be implemented. As a result, they emphasize problems of natural language understanding and the linguistic coverage of the query.

2.1.1. PLANES

PLANES which stands for Programmed LANguage-based Enquiry System [Waltz 78] was developed at the University of Illinois as a natural language interface to a large relational database of aircraft flight and maintenance information. It is one of several natural language systems that came out in the 1970's.

The natural language processing portion of PLANES comprises a number of augmented transition networks. Each transition network matches phrases for a particular meaning along with context registers and concept case frames. Context registers are history keepers; they help resolve pronoun reference and ellipsis of future requests. Concept case frames enumerate patterns of questions understood by the system to infer missing information. So in cases when some useful information is missing, matching these concept case frames with the rest of the sentence suggests what type of phrase is necessary to complete the concept. The matched patterns, which are transformed into unordered sets of semantic constituents with canonical phrases substituting for the user's terms, and with pronoun references and ellipsis resolved, are then translated into the formal query language, DSL Alpha.

Other features that PLANES incorporates include: providing a browsing ability; tolerating vague and poorly defined questions; handling ungrammatical inputs; correcting simple spelling mistakes; and paraphrasing and generating dialogue for clarifying partially understood questions.

To generate meaningful and cooperative responses, PLANES usually returns more fields than what are asked for. To decide which data fields to return for an answer, PLANES uses a simple rule: all variables and sets of constants are presented in the answer. For example,

```
Q5: Which planes had 20 or more flight hours in May?
```

Here "planes" is considered as a variable and "20 or more flight hours" is a set of constants. Therefore PLANES will return a list of flight hours and plane numbers, even though only the planes are explicitly requested. For example,

```
R5:    BUSER        TOTHRS
        67            21
        68            28
        76            27
        81            24
```

Here BUSER stands for BUreau SERial number - a plane identification number, and TOTHRS is the total number of flight hours for that plane.

2.1.2. LADDER

LADDER, for **Language Access to Distributed Data with Error Recovery** [Hendrix, Sacerdoti, Sagalowicz, and Slocum 78], is a natural language system developed at SRI International. It provides an intelligent interface for natural language access to a large body of data distributed over a computer network, the ARPA net. The user is buffered from the actual database management systems by three layers of insulating components: INLAND, IDA and FAM. These layers operate in series, converting natural language queries into actual calls to the DBMSs.

The first component, the natural language component, INLAND, is constructed within the framework of a language processing package called LIFER. LIFER uses a semantic grammar and makes use of production rules, lexical entries and subgrammars to interpret the user's query. It generates a sequence of queries to the VLDB (Very Large Data Base) in a LISP internal language. Queries from INLAND are then passed to IDA (Intelligent Data Access) which in turn breaks down the queries against the entire distributed database into a sequence of queries against individual files. FAM, the file access manager, then performs the actual file access, which involves finding the location of generic files and managing the access to them.

In employing a semantic grammar, domain semantics can be easily embedded into the language. For example, semantic grammars use patterns like:

```
<present> the <attribute> of <ship>
```

instead of the more general (BNF formal) pattern such as:

```
<noun-phrase> <verb-phrase>.
```

Generalization of concepts and meta-knowledge about the data can be easily accomplished with production rules in LIFER. Other concerns that LIFER also focusses on include spelling correction, general elliptical input processing, redefinition of terms and paraphrase handling. However, LIFER still suffers from a limitation on its syntactic and semantic coverage: it can only recognize very simple straight forward questions. Resolving relative clauses which contain long distance dependencies, conjunctive and disjunctive sentences, and resolving definite noun phrases which depend on the context, are problems for LIFER.

PLANES and LADDER are among the earlier successful natural language database systems. In both systems considerable effort has been expended on the linguistic coverage and the natural language understanding of the query. These earlier systems were mainly directed towards concerns (i) and (ii) stated in [Cercone and McCalla 86, pg 4]:

i. the kinds of language used when interfacing with a database are usually constrained; ways must be found of expanding the linguistic coverage of natural language systems;

ii. techniques must be evolved to integrate syntax, semantics, and pragmatics so that whatever action is appropriate at a given time can be done.

iii. the separation of the linguistic component sets up an arbitrary barrier which may have become counterproductive; a means of re-integrating data and language must be found;

iv. traditional (relational) database structures are not necessarily conducive to promoting the kind of inferences which need to be made for the query to be comprehended or answered properly; more sophisticated structures must be devised;

v. the user's understanding of the capabilities of the linguistic and database components is an important aspect of the man-machine communication, which must be taken into account; the user cannot be ignored; and

vi. even in a restricted linguistic domain such as natural language database interfacing, many discourse phenomena arise which must be accounted for if the natural language system is to behave cooperatively.

PLANES did attempt to provide more informative responses by returning more fields than required, but this methodology does not always give positive results, for example,

```
Q6: How many planes of type A7 had 20 or more flight hours in May?
R6: 1000.
    BUSER      TOTHRS
     67          21
     68          28
     .           .
     .           .
     996 instances
     .           .
     .           .
     76          27
     81          24
```

In subsection 2.2.1 we will see how lengthy enumerative responses could give contrapositive results. In the following section, we consider some of the more recent natural language database systems which emphasize cooperative responses.

2.2. Cooperative Responses

In this section, we describe several systems that emphasize cooperative responses. Each of these systems tries to solve a particular problem, which is different from the others, in a natural language database system. However, all of them demonstrate the need to go beyond simple data access in a natural language database system, and the need to take the next step of cooperative responses.

2.2.1. Summary Responses

Kalita's summary response system was developed at the University of Saskatchewan [Kalita 84]. This system generated non-enumerative summary responses which were less verbose and could often avoid any misleading implicatures. According to Grice [Grice 75], an important convention of human conversation is to ensure that no participant monopolizes the discourse. Hence, a response with a lengthy list of data may not convey the salient point of the answer. Under certain circumstances, summary responses are more appealing and more desirable. Furthermore, extensional responses can sometimes mislead the user by generating false implications whereas summary responses would not. For example [Gallaire, King, Mylopoulos, Reiter, and Webber 83],

```
Q7  :  Which department managers earn over $40,000 per year?
R7.1:   Abel, Baker, Charles, Doug.
R7.2:   All of them.
```

By enumerating all of the managers who earn over $40,000 (R7.1), the system would imply that there are still managers who do not earn that much if the user does not know that R7.1 implies R7.2. This is called a *scalar implicature* according to [Grice 75]. A cooperative principle of conversation requires a speaker to say as much as he can and not say anything that is believed to be false. A responder could give R7.1 as the response if he could not say the more inclusive answer R7.2.

Turning Null Responses into Quality Responses

Kalita's system employs a knowledge base which consists of

1. **frames** that are used to store useful information about the relations and their attributes in the database and

2. **heuristics** that guide the search for "interesting" patterns in the data.

The system will undertake a heuristic search of the data that satisfies the user's query, and try to discover and respond with any underlying patterns or implicatures by consulting the information stored in the frames of the knowledge base. One basic problem with this approach is that not all underlying patterns are "applicable" or relevant. For example,

```
Q8 : Which students are accepted into the honors program of the
     School of Computing Science?
R8.1 : All undergraduate students with a GPA of 2.7 or higher.
R8.2 : All undergraduate students with a social insurance number.
```

For query Q8, response R8.2 is certainly not relevant or suitable, a more relevant response might be one such as R8.1. In fact, one of the main reasons for having the frames and the significant values defined in them is to try to cut down the possibility of discovering irrelevant regularities, like R8.2, from the enumerative result.

2.2.2. University of Pennsylvania Research Efforts

Research efforts into natural language database systems have been very active at the University of Pennsylvania under the supervision of Dr. Bonnie Webber. In this section, we examine the work of three of Webber's students, McKeown, McCoy, and Mays.

McKeown and McCoy have developed systems that are able to answer queries about the structure of a database [McKeown 82], [McCoy 82]. This area of concern is, in fact, very important in a natural language database system because answering queries about the database structure will mean that some cooperative problem-solving interaction can be made available.

Turning Null Responses into Quality Responses

McKeown's TEXT system [McKeown 82] was developed to respond dynamically to three types of database structure questions:

1. requests for definitions.
2. questions about the kind of information available in the database.
3. questions about the differences between entities existing in the database.

TEXT employs a knowledge base that is comprised of taxonomic, functional and attributive information about the concepts in the database. It is basically an annotated generalization hierarchy on the entities in the database.

McCoy's ENHANCE system [McCoy 82] generates part of TEXT's knowledge base automatically. The ENHANCE system assumes that the database is static in order to generate an annotated generalization hierarchy. This hierarchy defines entities in terms of their superordinates and their subclasses; it also contains information such as:

1. based database attribute lists - these lists indicate the reasons for splitting into subclasses.
2. Distinguishing Descriptive Attributes (DDA) - attribute-value pairs whose values will distinguish one subclass from others.
3. database attribute lists - names and values of all attributes which are constant within a subclass.

McCoy also employs several world knowledge axioms in order to generate the hierarchy.

•

Mays worked on a system that has the ability to take the initiative and produce monitor offers of additional information that are both competent and relevant to the user's query [Mays 82]. For example,

```
Q9: Did John pass CMPT101?
R9: No, the semester hasn't ended, so he hasn't received a grade yet.
    Shall I let you know then when he passes the course?
```

This system deals with a dynamic database because it is able to reason about possible future states. To provide answers such as R9, a system must be able to recognize what events are actually possible, what additional information is relevant, and when it is appropriate to perform such services. Mays uses a branching-time temporal logic to achieve these goals. The system employs six composite temporal operators:

```
(Ex)P - holds iff P is true at some immediate future.
(Ax)P - holds iff P is true at every immediate future.
(EF)P - holds iff P is true at some time of some future.
(AF)P - holds iff P is true at some time of every future.
(EG)P - holds iff P is true at every time of some future.
(AG)P - holds iff P is true at every time of every future.
```

To illustrate how these operators are used to specify the system's view about the possibility of change in a dynamic database contents, consider the following example:

```
Let P stands for the predicate "student has passed course" and
    R stands for the predicate "student is registered for course",
then

(AG)[R → (EX)P] means if a student is registered for a course then
                   it is next possible that he/she has passed it.
(AG)[P → ¬R] means if a student has passed a course, then he/she
                is not registered for it.
```

Axioms such as the above would specify the relationship of the current state of the database to possible future states, and hence govern how the system views the possibility of whether the database contents may or may not be changed.

2.2.3. CO-OP

CO-OP [Kaplan 78], developed at the University of Pennsylvania by Jerrold Kaplan, is a natural language database system that specializes in the generation of cooperative responses. The motivation behind the development of such a cooperative response system is that if a natural language database system is capable of only providing direct responses to questions, then it will often result in some inappropriate or meaningless responses. Especially in the case of a null answer query, the appropriate response is rarely the direct answer to the question, but rather is an indirect answer. For example in [Kaplan 78, pg 2],

```
Q10:  Which students got a grade of F in CMPT100 last semester?
R10:  Nil [empty set].

Q11:  Did anyone fail CMPT100 last semester?
R11:  No.

Q12:  How many students passed CMPT100 last semester?
R12:  Nil.

Q13:  Was CMPT100 offered last semester?
R13:  No.
```

In these examples, it is inappropriate or uncooperative to give an answer such as R10. In the following example [Kaplan 78, pg12], an answer such as R14 will be meaningless if Bill was not at the banquet at all.

```
Q14:  How many Bloody Marys did Bill drink at the banquet?
R14:  0.
```

The cooperative responses that CO-OP can offer include: corrective indirect responses, suggestive indirect responses and supportive indirect responses. The mechanism that CO-OP employs to produce cooperative responses is domain transparent. The only domain specific knowledge that CO-OP needs can be derived from the information present in the database system if a suitably encoded lexicon is present. The cooperative response mechanism relies on language-driven inferences; that is, inferences concerning the user's presuppositions are driven from the particular phrasing of the user's input. Thus for Q14, inferences that can be made about the user's presuppositions are:

1. there is a Bill,
2. there is a banquet,
3. there is a liquor called Bloody Mary,
4. the liquor was available at the banquet,
5. Bill was at the banquet, etc.,

To accomplish this, CO-OP transforms the user's query into an intermediate representation called Meta-Query Language (MQL).

The MQL is a graph structure that encodes some of the syntactic relationships between entity sets present in the natural language query. Nodes in the graph represent entity sets, and edges represent binary relations defined on the connecting nodes. Each connected subgraph of the original graph corresponds to an assumption the user has made about the domain of discourse. Thus, if the initial query returns a null response, the system will check the user's assumptions by passing each connected subgraph, in turn, to be compared against the database to check for its non-emptiness. An empty set result for any one of the subgraphs will prove the falsity of the corresponding assumption, and thus an appropriate indirect response can be generated. For example, if the subgraph of inference 5 of Q14 returns a null answer, then the more appropriate response will be:

Bill was not at the banquet.

Kaplan's work on cooperative responses is closely related to my work, except that Kaplan's language-driven inference mechanism is very domain independent. This language-driven inference mechanism of CO-OP only enables it to handle requests for data retrieval, it cannot handle queries concerning the database structure or queries that require substantial inferences or comparisons of the data for generating an appropriate answer.[4]

2.3. Some Other Systems

2.3.1. Query Generalization

Motro has presented another approach to interpreting null answers using a technique called query generalization [Motro 86]. When a query returns with a null answer, an attempt is made to generalize the query, that is, to modify the query such that the answer set of the generalized query is a superset of that of the original query. This can be done by removing or substituting a condition in the search criteria. For example, consider the query to list all beer lovers:

$Q(x) = (x, \in, \text{PERSON})$ **and** $(x; \text{LOVES}, \text{BEER})$,

if this query results in a null answer, and there exists the fact:

(BEER, <[5] , ALCOHOLIC-BEVERAGE)

then the query generalizer will produce the following query:

$Q_1(x) = (x, \in, \text{PERSON})$ **and** $(x, \text{LOVES}, \text{ALCOHOLIC-BEVERAGE})$

The answer to this query results in a response to the user with a list of all persons that love alcoholic beverages. When the queries produced by the query generalizer

[4]I also believe that there are misconceptions that cannot be detected just from the surface language structure of the user's queries; this concern requires further investigation.

[5]< is a special entity that expresses the generalization relationship between 2 types. A concept described by the second type is more general than the concept described by the first type.

fail, the generalization process is applied again until it finds one query with a set of all successful generalized subqueries.

This approach is demonstrated with the author's Loose Structure Data Model, which overlays the underlying database. However, depending on the underlying database, the amount of necessary information for cooperative responses available in a traditional database system is questionable, especially meta-level data information. This process of query generalization for interpreting null answer from the database is similar to Kaplan's technique of passing each connected subgraph of the original graph, in turn, to be compared against the database to check for user's misconceptions.

Janas also took an approach similar to Motro and Kaplan to generate indirect answers to failed queries in database management systems [Janas 79]. From the formal representation of the user's failed query,[6] the system recursively substitutes the failed query by their failing predecessors until no more failed queries are found. This process results in a more informative answer than the user's original null response. A predecessor of a query is obtained by simply removing one term from the query expression without ended up with an unconnected expression, and which may have predecessors itself. For example, consider a relational database model:

$EMP(\underline{NAM}, SAL, AGE)$
$CAR(\underline{LIP}, OWN, COL)$

Q_3 is a predecessor of Q_2.

Q_2: $\{x \mid ((x.AGE < 30)$ **and** $\exists y((y.OWN = x.NAM)$ **and** $(y.COL = \text{'red'})))\}$
Q_3: $\{x \mid ((x.AGE < 30)$ **and** $\exists y((y.OWN = x.NAM))\}$

[6]A predicate calculus based query language.

Janas restricted his problem to query expressions which are connected. Q_2 is an example of a connected expression where variables x and y are connected by (y.OWN = x.NAM). A set of rules which specify how to obtain predecessors from a well formed query, and an algorithm using these rules to provide appropriate answers is also provided in [Janas 79].

2.3.2. PIQUE

In a natural language environment, a user will very often phrase his input with respect to the current perceived focus of the dialogue. Therefore, retaining a model of the user's current focus will assist the system to behave more appropriately and less erroneously.

Davidson has worked on modelling the user's focus during his/her interaction with the database [Davidson 82]. The user's focus is modelled by the segment of the database that the user's is currently accessing. The focus representation is just a DML (Data Manipulation Language) expression which can be viewed as an intensional description of that particular database segment that are on focus. The focus space determined by the DML might then provide the referent in subsequent dialogue. Interpretation-in-context is done via query modification. For example, consider the following two queries:

Q_4: "Who are the programmers?"
Q_5: "What is Jones' salary?"

a DML expression for Q_4 could be:

```
{x.name : x ∈ emps | x.occupation = 'programmer'}
```

and for Q_5, the DML interpretation without context could be:

```
{x.sal : x ∈ emps | x.name = 'Jones'}
```

but interpreting Q_5 in context of Q_4 using query modification could result in:

{x.sal : x ∈ emps | x.name = 'Jones' and x.occupation = 'programmer'}

Although there could be an inappropriate effect when evaluating the query with respect to a restricted focus space when the user didn't intend this restriction, Davidson tries to minimize this misuse of focus by restricting the use of focus to monotonic queries[7], and by employing several heuristic rules to determine whether a context-directed interpretation is appropriate.

2.3.3. JETS

JETS is a sequel of PLANES which was developed in the University of Illinois [Finin, Goodman, and Tennant 79]. JETS is centered on improving the conceptual coverage in natural language database systems. Often, users refer to concepts that are related to the activities in the domain, but are not actually represented in the database. Conceptual coverage of a system refers to the set of concepts that the system can deal with; concepts which are consistent with the domain, but which may not have been specifically forseen.[8]

The architecture of JETS is a network of frames. The generality/specificity links found in the system enables us to view the conceptual system as a directed tree of frames rooted at the most general concept with property inheritance. Interpretation of the conceptual frames is done by using a set of rules which are also frames themselves. The rules are basically pattern matchers. There are also a set of problem solving frames which help in the development of plans to extract required information

[7]Roughly speaking, monotonic queries are those which contain neither universial quatification, nor (certain forms of) negation [Davidson 82].

[8]According to the definition of closure in the manipulation of concepts by Woods [Woods 77].

from the database, and also generate dialogue with the user when ambiguous terminologies arise in the user's input.

In order to achieve closure in terms of semantic interpretation, JETS concentrates on conceptual modification. For example, to give an interpretation to the phrase "engine damage", where the ENGINE concept modifies the DAMAGE concept, JETS fill the damage-object slot in the DAMAGE concept with the ENGINE concept. However, solving the problem of conceptual modification in JETS does not imply that the system should be able to handle the concept in terms of answering questions about the concept in the database.

2.3.4. BROWSER

BROWSER is an automated system which searches a database for interesting patterns or configurations primarily for troubleshooting the database system [Dankel 79]. BROWSER explores the database using some special data models and a collection of data-dependent and data-independent heuristics. Interesting patterns occurring in the database are identified by using simple statistical techniques.

Representation of knowledge about the database is done via data-models and some data-dependent heuristics. There are five different types of models. *Data Base Models* contain descriptions of data fields in data files. *Data Specific Models* provide information on how the various data fields are related to each other. The *Specific Models* and the *Typical Models* provide the generalization and aggregation organisation, and the *General Models* provide information on typical data structures used within the data.

The controlled execution of tasks is done by using an agenda, which is an ordered list of tasks. The tasks are all weighted according to various considerations, for example, certain tasks must be executed before others, and certain tasks appear more important to perform than others because of the results of pervious tasks. The whole system is designed in a modified production system architecture. [Davis and King 75]

The system begins with a small basic set of concepts which describe the database. The execution of the agenda will allow the system to identify and define new significant subsets of data and explore them; thus, supplying a plausible explanation to certain outcomes in order to achieve troubleshooting.

Turning Null Responses into Quality Responses

Chapter 3
Classification of Null Events

In chapter 2, we briefly surveyed some previous work on natural language database systems. Natural language database systems have advanced beyond the stage of simple data access. There are systems that emphasize cooperative responses: [McCoy 82], [McKeown 82], [Kaplan 78], and [Mays 82]; summary responses: [Kalita, Jones, and McCalla 86]; system that keeps track of the focus of a discourse: [Davidson 82]; system that emphasizes conceptual coverage: [Finin, Goodman, and Tennant 79]; and system for troubleshooting: [Dankel 79]. However, most of these systems have overlooked the issue of null answers to queries. Kaplan, Janas, and Motro have worked on giving some treatment to null values. In particular, Kaplan considered the proper treatment of null answers to queries as very important to avoid any misleading implicatures.

Database systems rarely contain all of the information necessary to model their domain, hence null values arise in many database accesses. The ANSI/SPARC interim report [ANSI 75] lists 14 different manifestations of null values, for example:

1. not valid for this individual (for example, spouse of a bachelor)
2. valid, but does not yet exist for this individual (for example, a final grade of a course for a student before the final examination);[9] etc.

[9]Notice that Mays [Mays 82] has developed techniques to offer monitor responses whenever there is a null value of this type arises, see chapter 2.

Much research has attempted to give a proper semantic treatment to null values. I propose an initial classification of null events occurring in natural language database systems.

A typical natural language database system consists primarily of four major components as shown in the Figure 3-1.

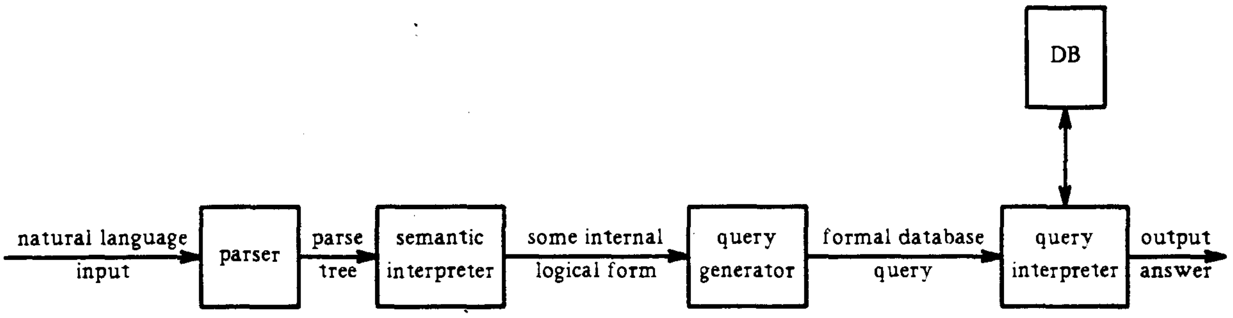


Figure 3-1: A typical NLDB system

We assume that the semantic interpreter is a very general semantic interpreter which can capture semantic information from the given database schema, and has its own knowledge about the linguistic aspect of natural language. Assume further that the semantic interpreter is as domain independent as possible, and does not carry detailed pragmatic concerns of the domain. An example of such a semantic interpreter can be found in [Cercone, Hadley, Martin, McFetridge, and Strzalkowski 84], where the natural language database system architecture is also similar to the one in Figure 3-1.

Before classifying null events, it is important to draw a clear distinction between a "No" and a "Null" answer to a database query. For a "closed" query, such as:

"Is John Doe a computing science student?"

which involves a "Yes/No" answer, a "Null" answer response should be interpreted as "I don't know whether John Doe is a computing science student" instead of being confused with the answer "No, John Doe is not a computing science student". In an "open" query, such as:

"Who is taking CMPT107 this semester?"

a "Null" answer should be interpreted as "I don't know who is taking CMPT107 this semester" instead of being confused with the empty set, {∅}, which means that "no one is taking CMPT107 this semester".

3.1. A Classification of Null Events

The classification scheme outlined below is very general and details five categories of null events. Not all categories may arise in every database and the extent to which domain information is incorporated into a semantic interpreter often will obviate some categories from further consideration. In the examples given below, **Q** represents the query, **DB** stands for a response from a system such as the one in Figure 3-1 where the response is the result of consulting only the knowledge that is available in the content of the database, **KB** assisted stands for a response that results from adding a knowledge base component to the system in Figure 3-1 which aids in providing more informative responses, and **Null** represents a null value response from the database.

Natural language database systems are designed to accommodate naive users. The more informal the query language is, the more sophisticated the system needs to be in order to comprehend and answer queries properly. Traditional database structures (for example, relational databases), however, are not conducive to providing the kind of inferences that are required. This gives rise to the first two classes of null event problems.

3.1.1. Null answer due to the need for general rules.

In this section, null responses due to the need for general rules will be discussed. Consider the following example:

```
Q : Does every graduate student have office space in the computing
    science department?
DB: Null.
________
KB assisted : Yes/No.
```

In this example, we require access to a general rule such as: "Every graduate student has an office space" in order to provide some definite answers to the given query.

Let us take a closer look at this problem. If the semantic interpreter can interpret the above query as:

```
"Do there exist graduate students that do not have office space?"
```

and respond to the original query by interpreting the answer it receives from the modified query, then we will not have a null value problem at all. However, if the semantic interpreter does not know how to interpret the original query because it does not know how to handle the universal quantifier, "every", then the semantic interpreter might reject the original query altogether. In this case, where we do not get any response from the system, we considered it as a null event[10] because the original query might have received an answer from the system with the help of a knowledge base. So, a null value problem is also a null event.

Notice here that separating the *extensional facts* and the *intensional facts* in our case (see chapter 4) can provide means to handle exceptional cases. For example:

[10] Notice the slight difference between *null value problem* and *null event* as used here.

```
KB assisted: Every graduate student has an office space in LB7602 except
             J.Smith who is on leave this semester. He can be contacted
             through his mail box.
```

By extensional facts, we basically mean the content of the database. Extensional facts are explicit information concerning the entities and relationships that exist at the current instant, they may change at relatively short intervals. Intensional facts are general facts about the world, constraints on how the world can be; for example, information about the structure of the database such as the database schema, is considered as intensional fact.

The following two examples further demonstrate the need for a knowledge base with general rules in order for the semantic interpreter and the database system to produce answers to the queries appropriately.

```
Q : Will Mark Johnson get an office space in the Computing Science
    department if he get accepted into the graduate program?
DB : Null.
```

This query could be appropriately answered if we have again the general rule: "Every graduate student has an office space".

Assuming the existence of the following two tuples in a STUDENT relation of (name, id, major, address):

```
(Joe, 111111111, Math, 1 First Ave Burnaby BC)
(John, 999999999, CMPT, 1 First Ave Burnaby BC)
```

then the question "Does Joe know John?" would produce "Null". However, we might be able to give an answer to this query with a general rule like:

$$\forall x \forall y (address(x) = address(y) \rightarrow know(x,y))$$

3.1.2. Null answer due to the lack of inferencing.

Assume the following relation:

CAR	COLOR
carA	white
carB	black

The question:

```
Q : Does carA have a lighter color than carB?
DB: Null.
```

results in a null answer since we require an inference rule which states what "lighter" means, in terms of color.

In the following example,

```
Q : Can I take Math102 next semester?
DB: Null. {Because Math102 is not an existing course.}
————
KB assisted: No, Math102 is not an existing course.
                                or
KB assisted: No, Math102 is not offered next semester.
```

note that the knowledge base might have an inference rule which states: *taking a course* means that *the course is in existence*, and *is offered next semester*. Thus a knowledge based natural language database system is able to give an answer to the above question instead of "I don't know". However, some databases might be able to respond with something like: "Math102 cannot be found" possibly with only a very slight modification in the retrieval routine. In chapters 4 and 5, we develop a knowledge base system to give quality responses to null events due to a user's misconceptions. The knowledge base system is made general enough that it is actually able to provide the more appropriate knowledge base responses seen in this example. However, to completely solve the problem in this category requires further research efforts.

SOURCE	RECIPIENT
NSERC	Cercone
...	...
...	...
Cercone	Kao

Consider another example, assuming the existence of the above relation: and assuming that the first tuple is the only tuple in the relation with Cercone as the recipient. Then, to answer the question "Is Kao doing any research work for NSERC?" correctly, we will need an inference rule like:

$\forall x \forall y \forall z(support(x,y)$ **and** $support(y,z) \rightarrow support(x,z))$[11]

3.1.3. Null answer due to the lack of knowledge about the database structure

Most existing formal query languages generally require the user to know what kind of information is stored in the database. In particular, considerable knowledge about how the database is actually structured is essential to construct the query. However, it is not essential in a natural language database system that the user need to have this pre-knowledge in order to pose some well-formed queries. Furthermore, it is not unusual for the user to want to know how the database is structured when he is engaged in solving problems related to the domain of discourse. The two examples below illustrate null responses due to a lack of knowledge about the database structure and suggest more appropriate knowledge base responses.

[11]Note the similarity between the methods for handling null responses due to the lack of general rules and those due to the lack of inferencing. By general rules, we mean to include the universal quantifier. Thus, depending on which kind of inference mechanism we have, the class of null event due to the need for general rules might be seen as a subclass of null events due to the lack of inferencing.

```
Q : What is the number of the phone in Rm1234?
DB: Null
______
KB assisted: Rm1234 is a theatre.
             No phone is present in a theatre in Simon Fraser University.


Q : How many different job classifications do we have in SFU?
DB: Null
______
KB assisted: There are 14 different academic job classifications in
             Simon Fraser University.
```

For the second query, if we have separate employee records for different departments in the university, then we will not be able to provide an appropriate answer to the query without knowing how the university is structured, unless, of course, the information about the different academic job classifications is explicitly expressed in a particular relation in the database. Thus, both of the above queries require some kind of hierarchies concerning the database structure in order to be able to give the equivalent kind of knowledge base assisted responses as shown above. For example, a facilities hierarchy, which tells us that theatre (including its location) is a kind of facility in the university, and which also tells us what kind of facilities are present in a theatre, is needed for the first query; and an employee's hierarchy is needed for the second query.[12]

3.1.4. Null answer due to the lack of ability to handle <u>partial</u> <u>information</u>.

Having partial information is not an uncommon real world situation. However, partial information has typically not been represented in traditional databases. If the partial information "teach(Joe, CMPT810) V teach(Art, CMPT810)" is known, this information may only be approximated in the database (say, relational database) as:

[12]Notice that, on the other hand, hierarchies can be expressed as inference rules too.

```
teach(CMPT810-spring,sect01,null)
```

In this case, we have information that either Joe or Art is teaching CMPT 810 next semester, but we don't know which. Then the questions

```
Q : Who is teaching CMPT 810 next semester?
DB: Null.
_______
KB assisted: Either Joe or Art is going to teach this course next semester.
```

or

```
Q : Is Joe teaching CMPT 810 next semester?
DB: Null.
_______
KB assisted: Either Joe or Art is going to teach this course next semester.
```

would have better responses from a knowledge-based natural language database system.

3.1.5. Null answer due to a misconception.

A user's query is usually loaded with assumptions. We have already considered examples of "loaded" queries in chapter 1 and chapter 2. One such loaded query is:

```
How many Bloody Marys did Bill drink at the banquet?
```

When one or more assumptions of a loaded query are incorrect, a null answer arises. While such null answers are correct from a technical point of view, very often they are unsatisfactory and abstruse. Null answers due to user's misconceptions can arise under several conditions:

1. a misconception that "fails intensionally" due to

    1. a missing relationship (example due to [Webber, Joshi, Mays, and McKeown 83]).

```
Q : Which graduate students have taught CMPT681?
DB: Null.
_______
KB assisted: CMPT681 is a graduate course.
             Only faculty can teach a graduate course.
```

2. or a missing attribute field.

```
Q : Is the king of France bald?
DB: Null.
────────
KB assisted: There is no king in France.

Q : Who is the dean of the computing science department?
DB: Null.
────────
KB assisted: There is no dean of a department, but you can
             talk about the dean of a faculty.
```

Note that in this subclass of null events, the semantic interpreter might reject the query initially, because it might not be able to find any interpretation of "dean of a department" or "king of France" from its knowledge base or from the database schema. This kind of null event can be resolved with the help of generalization hierarchies (see chapter 4), (and/or specialized inference rules) in a knowledge base. This class of null values is similar to *not valid for this individual* [ANSI 75].

2. a misconception that "fails extensionally" due to

   1. a missing tuple[13]

```
Q : Is CMPT107 offered this semester?
DB: Null. {Because CMPT107 is not an existing course}
────────
KB assisted: No, CMPT107 is not an existing course.
```

   2. a missing attribute

      1. temporal events[14]

```
Q : Who is teaching CMPT101 next semester?
DB: Null.
────────
KB assisted: The information is not known as yet.
             A course schedule for next semester will be
             ready by Dec4.
```

[13]Kaplan had worked on providing a similar result to the KB assisted response for this category of null event. A language-driven inference mechanism is used in his approach [Kaplan 78], see chapter 2.

[14]Note that, for this category, Mays had developed techniques to monitor possible changes in the database, and provide relevant information concerning these changes to the users. [Mays 82]

```
Q : What grade did John get in MATH100?
DB: Null.
_______
KB assisted: The grades for this semester are not in as yet.
             All grades will be in by Dec 3.
```

Note that this class of null values is similar to *valid, but does not yet exist for this individual* [ANSI 75].

2. always null

```
Q : Which text book is used for CMPT898?
DB: Null.
_______
KB assisted: CMPT898 is the course number for a masters thesis,
             there is no text involved here.
```

Note that this class of null events is very similar to the missing attribute field for a misconception that fails intensionally, as discussed earlier, depending on the structure of the database.

Figure 3-2 is a diagrammatic version of the different categories described above:

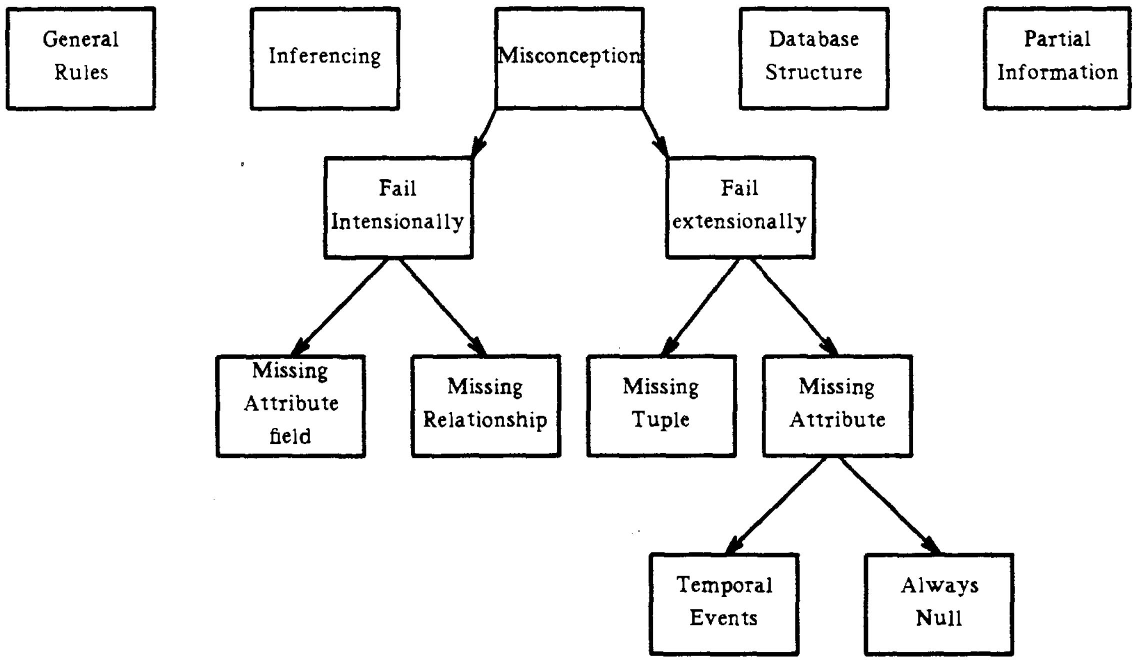


Figure 3-2: A Classification of Null Events

In the next two chapters, we further investigate quality responses to null answer queries mentioned in this last section, that is null answer queries due to user's misconceptions. The knowledge base model introduced in chapter 4 provides us with a general way of capturing information that is necessary and essential for generating the desired quality responses shown in this section. The two algorithms found in chapter 5 further explicate the knowledge base model, and demonstrate the kind of quality responses we can generate from this model.

Chapter 4
The Knowledge Base Model

Given a relational database, we have already seen (in chapter 3) that when a natural language front-end is introduced, we can only provide limited quality responses when a null value arises. This is mainly because when data retrieval ends up with an empty response, the meaning of this empty response is completely unknown to the user, and might even be inexplicable from the information found in a traditional relational database system. A knowledge base becomes important, at this point, in a natural language database system to provide information for the generation of quality responses to null answer queries. In particular, concept hierarchies, generalization and aggregation hierarchies need to be added to the relational database to facilitate the generation of quality responses. In this chapter, we will consider the structure of the knowledge base.

4.1. Knowledge Base vs Database

A database can be viewed as a repository of facts. Facts that describe some real world situation at a particular instance. Most of the time, a database is designed to tolerate changes, and is therefore may evolve over time. Thus, in a database we can find two types of information: the content of the database informs us of the way the world is modelled; and the structure of the database, specifically, the database schema, tells us the way the world can be modelled. We refer to these two types of information, as extensional facts and intensional facts in chapter 3.

Unfortunatly, traditional databases are considered to be quite limited in terms of expressive power. In a typical natural language database system as the one in Figure 3-1, the intensional facts available in the database are only used for giving plausible interpretations to the natural language inputs. There are other facts which are not present in the database management system, but are useful in informing us how the world can be modelled.

Hence, for any database management system used in a natural language database system, we want to add to it a knowledge base which contains more intensional information about how the world can be modelled, in particular, information that can help in the interpretation of null values in the database. Thus, a knowledge base can explicitly incorporate meta-data knowledge: knowledge about the database structure, such as entity concept hierarchies, generalization and aggregation hierarchies; knowledge about entity relationships, entity relationship constraints; knowledge about null values existing in the database; knowledge about event precedence in the database domain; etc. In the relational knowledge base model, which I am going to describe in detail later in this chapter, tuples entries will therefore have the function of informing us of the interrelationships between the different attribute fields, entity concepts, etc. present in the database.

We assume a relational database model, and that the query language is SQL. The knowledge base model is similar to the RM/T model[15], Codd's extended relational model [Codd 79]. Hence, the knowledge base model is also relational in nature. This approach has the following advantages:

---

[15]RM/T stands for Relational Model/Tasmania, where the ideas embedded in the model were first presented.

1. For relational database implementations, the required knowledge base is inexpensive to incorporate with only some additional information required.

2. A relational representation of a knowledge base does not require special operators to manipulate the knowledge base; the existing relational operators (with slight modifications) can be used. This renders our scheme conceptually simple for the database administrator, and immediately applicable as an add-on to any existing databases.

3. This approach makes it easy to implement a table-driven scheme, and, as a result, enhances portability.

4. The knowledge base model captures meta-data information at an intensional level, with the addition of a considerable static event calendar of the application domain. This makes the knowledge base independent of the data values in the database. Database updates will have no effects on the knowledge base unless there is a change in the database schema.

The RM/T model distinguishes *P-relations* (property relations) from *C-relations* (characteristic relations). In a P-relation only single-valued functional dependencies are considered; and in a C-relation only multi-valued dependencies are considered. Our model does not distinguish between these relations because we can always obtain all of the functional dependency information by consulting the database schema. Also, since we are mostly handling the database intensional facts in the knowledge base, the model is marginally different from the RM/T model, in which both intensional and extensional facts are handled.

4.2. The Knowledge Base Model

In this section, the relational knowledge base model is described. The knowledge base is considered to be a separate component in the natural language database system, although it is a relational model as is the database. Note that all examples illustrated in this section are based on the database schema shown in Figure 4-1. A brief overview of the knowledge base model is given first, and the full detailed description of each relation in the model is given later.

```
OFFERING(CNAME, SEMESTER#, CLASS#)
COURSE-PREREQUISITE(CNAME, PREREQUISITE)
COURSE-PREVIOUS-NAME(CNAME, PREVIOUS-NAME)
COURSE-DESCRIPTION(CNAME, DESCRIPTION)
SEMESTER(SEMESTER#, YEAR, SEASON)
CLASS-DEPT(CLASS#, DEPT, UNITS)
TEACH(CLASS#, SEC, INSTRUCTOR#, TEXT)
TAKE(CLASS#, SEC, STUDENT#, FINAL-GRADE)
CLASS-ROOM(CLASS#, SEC, TIME, ROOM#)
INSTRUCTOR(INSTRUCTOR#, NAME, OFFICE, SEX, STATUS)
STUDENT(STUDENT#, NAME, MAJOR, MINOR, SEX, STATUS)
DEPT-CHAIRMAN(DEPT#, CHAIRMAN, FACULTY)
DEPT-FACILITY(DEPT#, FACILITY)
FACULTY-DEPT(INSTRUCTOR#, DEPT)
```

Figure 4-1: Database Schema

There are basically two types of entity handled in our knowledge base model[16]: simple entities and associative entities. All simple entity types can be found in the ENT-relation, and the knowledge base is closed under all simple entity types, that is, we assume that all simple entities' identifiers are known by the world modelled by the database. For every entity type found in the ENT-relation, we assume that there exists a relation in the database that contains the list of all members of that entity type. We called this relation the complete list relation (CLR) of an entity type. The primary key attribute field, which we assume to not be a composite primary key, of a complete list relation (CLR) is naturally taken to be the identifying field of the corresponding simple entity type. Associative entity types are relations which we called AG-relations. Since we allow an associative entity type to be any n-ary relation, an AE-relation is introduced to keep track of all associative entity types that are defined in the domain. Both simple entity types and associative

[16]According to Ullman, in Principles of Database Systems [Ullman 82], there are basically two kinds of relations: (1) an entity set can be represented by a relation whose relation scheme consists of all the attributes of the entity set; and (2) a relationship among entity sets E_1, E_2, ..., E_k can be represented by a relation whose relation scheme consists of the attributes in the keys for each of E_1, E_2, ..., E_k.

entity types are allowed to have properties. The PG-relation is introduced to bind properties to different entity types.

We use a CLASS-relation and a GH-relation to provide generalization and aggregation hierarchies for entity concepts. The GH-relation allows us to built hierarchical structures on an entity concept, and we can structure an entity concept with respect to different categories (classifications) with the CLASS-relation. Sometimes an entity type, simple or associative, or properties of an entity type might be time dependent. We use the EG-relation and the E-relation to facilitate event precedence relationships. An event can be represented by any attribute field defined in the database, or an E-relation. An E-relation is a relation which stores information about an event that has some predefined approximate (or exact) dates that are known to the knowledge base. Exceptional cases can also be modelled using the EXC-relation where exceptional case(s) can be attached to different concepts. Finally, the V-relation contains all the view definitions which are defined in the knowledge base model. The key way to tell whether an attribute concept participates in any of the event precedence relationships or exceptional facts is to put the corresponding attribute field as a value in the proper relation.

Our model consists of this package of relations added to the database, as illustrated in Figure 4-2. These relations' schemes are described in more detail below. Tuple entries in these knowledge base relations are yet to be handcoded by some appropriate person, like the system administrator.

- **ENT-relation** (ENTity relation) is a binary relation which contains all of the simple entity types of the application domain. The primary key of this relation is an entity type which will uniquely determine a relation, the complete list relation(CLR), in the database where we can get a complete list of the members of the corresponding entity type.

ENT-relation:

ENTITY	CLR
course	COURSE-DESCRIPTION
student	STUDENT
instructor	INSTRUCTOR
dept	DEPT
semester	SEMESTER
class	CLASS-DEPT

(a)

AE-relation:

RELATION
OFFERING
TEACH
TAKE
.
.
.

(c)

EG-relation:

SUP	SUB
final-exam	FINAL_GRADE
ROOM#	**class-begin**
.	.
.	.
.	.

(d)

PG-relation:

ENTITY	PROPERTY
course	DESCRIPTION
course	PREREQUISITE
course	PREVIOUS_NAME
course	CNAME
semester	SEMESTER#
semester	YEAR
semester	SEASON
class	CLASS#
class	DEPT
class	UNITS
dept	DEPT#
dept	CHAIRMAN
dept	FACULTY
dept	FACILITY
instructor	INSTRUCTOR#
instructor	NAME
instructor	DEPT
instructor	OFFICE
.	.
.	.
.	.
student	STUDENT#
student	NAME
.	.
.	.
.	.
OFFERING	CLASS#
TAKE	FINAL-GRADE
CLASSROOM	ROOM#

(b)

E-relation
final-exam:

SEASON	DATE
fall	10/12/86
spring	20/4/86
summer	22/8/86
.	.
.	.
.	.

(e)

E-relation
class-begin:

SEASON	DATE
fall	5/9/86
spring	9/1/86
summer	3/5/86
.	.
.	.
.	.

(f)

CLASS-relation:

SUP	SUB
course	course1
course	course2
.	.
.	.

(g)

GH-relation:

SUP	SUB
course1	graduate-course
course1	undergraduate-course
course2	art-course
course2	science-course
.	.
.	.
student	graduate-student
student	undergraduate-student
.	.
.	.

(h)

EXC-relation:

CONCEPT	EXCEPTION
TEXT	thesis-course
.	.
.	.
.	.

(i)

V-relation:

VIEW
thesis-course
.
.

(j)

Figure 4-2: Relations of the Knowledge Base Model except for AG-relations

For example, in the database schema of Figure 4-1, **course** can be considered as an entity type. There are four relations that are related to **course**: OFFERING, COURSE-PREREQUISITE, COURSE-PREVIOUS-NAME, and COURSE-DESCRIPTION. A designer might consider that COURSE-DESCRIPTION is the proper relation where he can get a complete list of all the known courses because of his knowledge about the database; or it might because of the fact that CNAME is the sole primary key attribute in COURSE-DESCRIPTION. Thus we get a tuple: (**course**, COURSE-DESCRIPTION) in the ENT-relation as shown in Figure 4-2 (a).

- **PG-relation** (Property Graph relation) is a binary relation. The main function of this relation is to connect properties to each entity type. The PROPERTY attribute field contains names of those attribute fields in the database that are considered to be properties of an entity type in the **ENT-relation** or properties of an AG-relation.

  For example, consider the entity type **course** and its four related relations; from COURSE-PREREQUISITE, COURSE-PREVIOUS-NAME and COURSE-DESCRIPTION, we can tell that every course has a CNAME, some PREVIOUS-NAMEs, some PREREQUISITEs, and a DESCRIPTION. Thus all of these are considered to be properties of the entity type **course**. The attribute fields in relation OFFERING serve more of the purpose of an associate to the other attributes[17], thus they do not appear as properties here, see Figure 4-2 (b).

- **AE-relation** (Associative Entity relation) is an unary relation which consists of the names of all associative entity types, that is, all AG-relations, see Figure 4-2 (c).

- **AG-relations** (Association Graph relations) are n-ary relations. This kind of relations interrelate entities of the ENT-relation, attribute concepts, or possibly AG-relations. Each AG-relation is an associative entity type that appears in the AE-relation. An associative entity type that an AG-relation represents is allowed to have properties as well. These properties can be found in the PG-relation. Since we are handling mostly intensional facts of the database in the knowledge base, and extensional facts concerning a particular association can be found in the database already, we associate tuple entries of an AG-relation to acknowledge the constraints we would like to assert on the particular association that the AG-relation represents.

  For example, assume we have the following **AG-relations**:

  $OFFERING(\underline{\textbf{course}}, \underline{\textbf{semester}})$
  $TEACH(\underline{\textbf{instructor}}, \underline{\textbf{course}})$
  $TEXT(\underline{\textbf{course}}, \underline{\text{TEXT}})$
  $TAKE(\underline{\textbf{student}}, \underline{\textbf{course}})$
  $CLASS\text{-}SEC(\underline{\textbf{class}}, \underline{\text{SEC\#}})$
  $CLASS\text{-}ROOM(\underline{CLASS\text{-}SEC}, \underline{\text{TIME}})$

  TEACH will be an associative entity type which informs us that instructors teach courses. A tuple entry in the TEACH AG-relation, say

[17] As we can see they are considered in the AE-relation.

(**graduate-student**, **undergraduate-course**), see Figure 5-2 (a), will inform us that graduate students are permitted to teach undergraduate courses. Also, we should try to put as high (in the hierarchy of an entity concept) a generic entity concept in the attribute field of an AG-relation as possible, so that constraints on the association can be specified as sub-concepts of those super-concepts. For example, we would have *OFFERING*(course, **semester**) rather than *OFFERING*(CNAME, **semester**), because CNAME is a properties of the concept **course**.

- **EG-relation** (Event Graph relation) is a binary relation which stores the precedence relationships of different events occurring in the application domain. There are two attribute fields in the EG-relation: the SUP and the SUB fields. An event that can appear in these two fields, for the time being, is just any attribute existing in the database or an E-relation in the knowledge base, see Figure 4-2 (d).

  For example, FINAL_GRADE is an attribute field in the TAKE relation in the database, and **final-exam** is an E-relation, see Figure 4-2 (e). So the tuple (**final-exam**, FINAL_GRADE) in the EG-relation means that the value of the FINAL_GRADE of the TAKE relation in the database will be known only after the **final-exam** event has happened.

- **E-relation** (Event-relation) is represented as a binary relation. Its primary key attribute is "seasons of the year", SEASON; and given the primary key, we can determine an exact date, or an approximate date, for the event in a particular season, see Figure 4-2 (e,f).

- **CLASS-relation** (CLASSification relation) is a binary relation. This relation informs us of the different possible taxomonic classifications we could have for an entity concept in the application domain. The SUP field will thus contain entity types, and the SUB field will contain concepts for the different classification categories, see Figure 4-2 (g). For example, concept **course** could be classified in two different ways, therefore we have two taxomonic classification concepts for **course**: **course1** and **course2**, see Figure 4-2 (g). **course1** is a classification with respect to academic levels, where the concept **course** is split into two subset: **graduate-course** and **undergraduate-course**; **course2** is a classification with respect to academic fields, where the concept **course** is split into **art-course** and **science-course**.

- **GH-relation** (Generalization Hierarchy relation) is a binary relation representing a directed graph. The two attributes of the GH-relation, SUP and SUB, indicate the superordinate or subordinate (superset/subset) role of the participating concept. A concept appearing in the GH-relation can either be an entity type in the ENT-relation, or a view in the V-relation, or a SUB in the CLASS-relation, see Figure 4-2 (h). For example, the tuple:

(**student**, **undergraduate-student**) represents that the concept **undergraduate-student** is a subset of the superset, **student**.

- **EXC-relation** (EXCeption relation) is a binary relation which informs us all the exceptional facts existing in the application domain. Entries for the CONCEPT attribute field, see Figure 4-2 (i), can be any attribute concept existing in the knowledge base/database[18]; and the entries for the EXCEPTION attribute field will mainly be either a view in the V-relation or a concept in the GH-relation or an entity type in the ENT-relation, see Figure 4-2 (i). For example, using our database schema outlined in Figure 4-1, TEXT is an attribute field of the TEACH relation in the database, and **thesis-course** is an entry in the V-relation. These facts inform us that there will be a text in the TEACH relation except for those courses which are considered as **thesis-course**, where **thesis-course** is defined in the view definition in the knowledge base's dictionary.

- **V-relation** (View relation) is an unary relation which contains all the views that are defined in the knowledge base's dictionary, see Figure 4-2 (j). The definition of a view is very similar to a view definition in system R. For example, a view definition for **graduate-student** can be derived from the base relation STUDENT in Figure 4-1 as follows:

```
DEFINE VIEW graduate-student
         AS SELECT SNAME
            FROM   STUDENT
            WHERE  STATUS = "GRAD"
```

For this model, we need to have all of the necessary hierarchies, information, rules, etc. to be hand-coded by some appropriate person, for example the database administrator, into the model.[19] The knowledge base will therefore require a dictionary to store all the view definitions of the V-relation, and to make the knowledge base adaptable to different environments.

With the introduction of a knowledge base into a natural language database

[18]We expect that they will be primarily non-key attributes of some relations.

[19]Also see comments in chapter 6 regarding the automatic generation of part of the knowledge base relations.

system, the natural language database system illustrated in Figure 3-1 evolves into a system as depicted in Figure 4-3. In the natural language database system shown in Figure 4-3, the knowledge base is interacting only with the database. However, a well-integrated natural language database system is anticipated which shares information among the semantic interpreter, the knowledge base, and the query generator as shown in Figure 4-4. The task of determining how to integrate these components together requires additional research effort (as discussed in chapter 6, section 6.1.2).

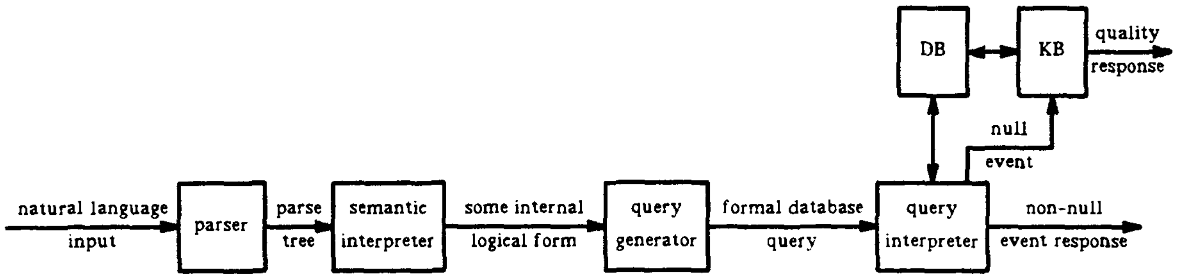


Figure 4-3: A NLDB system with a Knowledge Base component

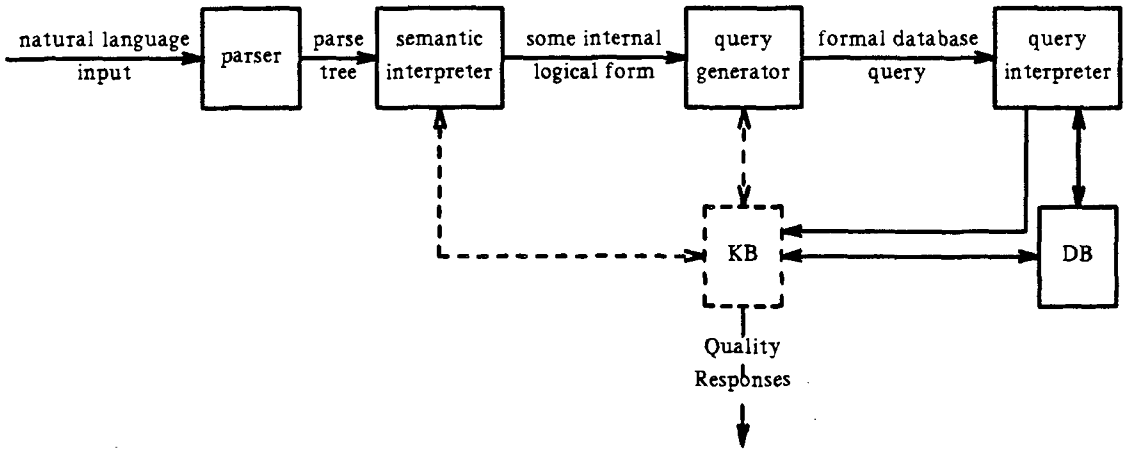


Figure 4-4: Integrating a knowledge base into a NLDB system

Turning Null Responses into Quality Responses

4.3. What if the Knowledge Base has null values?

A natural question to ask at this point is "*What if there are null values in the knowledge base?*" To answer this question, we would like to restrict the entries in the knowledge base to contain no null values for the sake of simplicity of the model. As a matter of fact, null values can only occur in the ENT-relation, and in the E-relations in the knowledge base, see Figure 4-2. However, another method to solve this problem is to have yet another meta-knowledge level of null values of the knowledge base[20], which allow us to know what to do if null values occur in the knowledge base. This approach will require multi-levels of meta-knowledge which will stop at some level where there are no more null values. In the worst case, unfortunately, this will render the system to have infinite meta-knowledge levels, which in any practical database management system one would try to avoid. However, these infinite meta-knowledge levels might be conceptually advantageous as it is similar to that espoused by Brian Smith of Xerox Parc in his work on 3-LISP. A further extension in the meta-knowledge direction is to "interject" the user at some point in the infinite meta-knowledge levels. When a null value occurs in a knowledge base relation, we can have the corresponding attribute field index some table which contains appropriate instructions to respond to the user, or an index to activate certain routine(s) that will interact with the user.

According to this model, the knowledge base is closed under all simple entities; thus, all simple entities' identifiers are supposed to be known in the application domain. Consider the entity **course** in the academic domain represented by the database schema in Figure 4-1. It is not unreasonable to assume that all the courses

[20]Say, another relation in the knowledge base that contains entries which are the potential knowledge base attribute fields that might have null values in there.

offered are known in the application domain. Also, the knowledge base is closed under all the AG-relations. A tuple in an AG-relation conveys the constraint that is being asserted to that particular relationship. This constraint is considered to be a positive constraint, in which we are informed what the legitimate participants are in that relationship. The model does not provide the representation of negative constraints; therefore, we require that all the AG-relations be closed in the knowledge base.

This relational knowledge base model emphasizes, what Brachman calls the epistemological (knowledge structuring) level of knowledge of the database [Brachman 79]. However, we are not concerning ourselves with giving a complete formalism on the epistemological level of knowledge representation as Brachman had done in describing KLONE. A taxonomy can be constructed through the CLASS-relation and the GH-relation; this is similar to the so-called IS-A hierarchy in the semantic networks representation, with a set/subset hierarchical structure. Concept hierarchies seem to be an indispensable element in knowledge structuring, as we can see it has been emphasized in different ways in the literature. In [Schubert, Goebel, and Cercone 79], classification of knowledge is done through topic hierarchies, with generalization and specialization being two of the significant topics. Concept generalization/specialization is used as a basis for conceptual modeling in TAXIS, this is done by a methodology called taxonomic specification, which combines the techniques of abstraction and stepwise refinement [Borgida, Mylopoulos, and Wong 84]. Stonebraker proposed to use abstract data types for adding semantic knowledge to relational database system [Stonebraker 84]. Reiter also illustrated how to provide a first order represention of generalization and aggregation hierarchies when he attempted to give a logical reconstruction of various aspects of the conventional relational database theory [Reiter 84].

Turning Null Responses into Quality Responses

Chapter 5 Algorithms for Detecting a User's Misconceptions

In this chapter, two algorithms for detecting user's misconceptions and providing quality responses are described. These algorithms illustrate how to provide quality responses for null response cases arising from those categories shown on the subtree of the "misconception" box in Figure 3-2. According to the classification in Figure 3-2, user's misconceptions can be classified into misconceptions that fail intensionally and misconceptions that fail extensionally. Extensional failure of a user's query occurs when the user has a misconception that there is some non-empty extension in the database that will satisfy his query description. While an intensional failure arises when the user has a misconception about some domain relationships, in particular, misconception about entity(ies) that can participate(s) in some relations. The main routine for detecting a user's misconception will check for both kinds of misconception when a null event arises.

For an extensional failure, it can be due to the non-existence of a certain object; or it can be due to the fact that the event which is responsible for the desired value (or set of values) has not been taken place as yet; or it might also because of some exceptional cases present in the domain. Utilising the CLR (complete list relation), we can, to certain extent, check for the existence of an object, in particular, the existence of known simple entities. If certain attribute values in the database are dependent on the occurrence of some other events in the domain, such information can be conveyed to us by the EG-relation, which tells us the precedence relationships of different events occurring in the domain. Furthermore, more informative responses can be supplied to the user if the unknown value is related to an event which is an E-relation, because some appropriate dates can be drawn from the related E-relation for the user's reference. If the user has specified an exceptional case in his input query, by checking with the EXC-relation, we can give a more informative explanation to the user's misconception.

An intensional misconception can be detected by checking all the related AG-relations that are presented in the user's query. If any constraint of a related AG-relation is not satisfied by the user's description set[21], the incorrect user's specification will be reported and the proper constraints for the misconceived relationship will also be reflected to the user.

A high-level description of the basic algorithms is presented, and the algorithms are further illustrated by a few examples employing specific SQL queries. The detailed (pseudo-code) algorithms are included at the end of this chapter.

5.1. Extensional Misconception

Procedure **Extensional_Misconception** provides quality responses to misconceptions which fail extensionally using the relational knowledge base model introduced in the last chapter. For this algorithm, we assume that the database will be able to supply information such as: "*x* cannot be found" in the case of a user's misconception that fails extensionally. We also assume that if *x* is a value then there

[21]By user's description set I mean the set of objects (or individuals) that satisfied the user's query description.

will be an X which is the corresponding attribute field of the value x that we can obtain from the *logical form* of the input query. A null value occurring in the database is given three possible interpretations by this algorithm: (1) a *valid* individual who does *not exist*; (2) a value that is *not valid* for an individual; and (3) a value that is *valid* but does *not yet exist* for an individual. **Check_Object_Existency**, **Check_Always_Null**, and **Check_Temporal_Event** are three procedures that test for these interpretations, respectively.

If x is a value, then this null event belongs to the category of extensional failure because of a missing tuple. In this case, we activate procedure Check_Object_Existency to ascertain, if possible, whether x is a known existing object in the database domain. If x is a non-existing object in the application domain, procedure Check_Object_Existency will generate an appropriate response to signify the non-existence of x, and no further investigation on any user's intensional misconception is necessary.

If x is an attribute field, this means that there is a *null* value in the database entry, and this null event belongs to the category of extensional failure because of a missing attribute. In this case, we call procedure Check_Always_Null and procedure Check_Temporal_Event, in turn, to attempt an appropriate interpretation of this *null* value. Again, no further investigation for intensional failure is necessary because encountering a *null* in the database is definitely an extensional failure.

Procedure Check_Always_Null ascertains whether the information specified in the logical form conforms to any exceptional case in the application domain, and gives an appropriate response if possible.

Turning Null Responses into Quality Responses

In procedure Check_Temporal_Event, if x is a temporal event, then relevant dates related to x will be drawn from the knowledge base in order to account for the *null* value which resulted from the database access.

More detailed explanations for each of the above procedures and one related subroutine are provided together with their algorithms in section 5.6 of this chapter.

5.2. Two Simple Examples of Extensional Misconceptions

We illustrate the algorithm for extensional misconceptions with two examples. The *logical form* depicted in the examples represents the output that will be generated from the semantic analyzer for the input natural language query, see [Cercone, Hadley, Martin, McFetridge, and Strzalkowski 84]. Subsequently, the logical form is translated into the SQL query as shown.

Example Query 1: *What grade did John Simpson get in CMPT106?*

Logical Form:

```
((((((WHICH sg v1) (v1 = (grade)))
    ((EXIST sg v2) (v2 = (student (studentname $John Simpson))))
    ((EXIST sg v3) (v3 = (course (cname $CMPT106))))
    ((EXIST nil v4) (v4 = (time (term=spring)(year=1986))))
    (TAKE (subj v2) (obj v1) (locative v3) (time v4))))))
```

SQL Query:

```
SELECT  FINAL_GRADE
FROM    TAKE
WHERE   STUDENT# IN
        SELECT  STUDENT#
        FROM    STUDENT
        WHERE   NAME = 'JOHN SIMPSON'
  AND   CLASS# IN
        SELECT  CLASS#
        FROM    OFFERING
        WHERE   CNAME = 'CMPT106'
          AND   SEMESTER = 'SPRING86'
```

If this query fails extensionally, then the possible database responses might be the following:

```
1. CMPT106 cannot be found
2. FINAL_GRADE cannot be found
              :
```

In this example we can demonstrate three possible valid interpretations which we may obtain using the algorithm for extensional misconception.

case 1: CMPT106 cannot be found.

```
x = CMPT106,     X = CNAME
```

In this case, CMPT106 might be a nonexisting course, see Figure 5-1 (a). Since x is a value; therefore procedure Check_Object_Existency is activated. If CMPT106 is not an existing course, the algorithm finds that CNAME is a property of the entity **course** from the P-relation, see Figure 4-2 (b); then it finds that CNAME is the entity identifier of **course**, see Figure 4-2 (a) and Figure 4-1. Therefore it checks if CMPT106 is one of those course identifiers. Since CMPT106 is not in the relation COURSE-DESCRIPTION, see Figure 5-1 (a), we obtain the response:

"CMPT106 is not an existing object for entity type course."

case 2: CMPT106 was not offered in the semester specified.

If CMPT106 was not offered in the specified semester, procedure Check_Object_Existency again finds that CNAME is a property of entity **course**; CNAME is the entity identifier of **course**. However, this time CMPT106 is an existing course, see Figure 5-1 (b); therefore the algorithm concludes that the relationship represented by the relation which contains x is false according to the world modelled by the database:

"CMPT106 is not offered in spring 1986."

case 3: FINAL_GRADE cannot be found.

The FINAL_GRADE cannot be found (with x = FINAL_GRADE). Since x is an attribute field, and x is not in the EXC-relation, procedure Check_Temporal_Event is activated. This procedure finds that x occurs in the SUB field of the EG-relation, see Figure 4-2 (d), the preceding event of FINAL_GRADE is **final-exam**, and the relevant preceding date is August 22, 1986, see Figure 4-2 (e). When the algorithm compares the current date with the preceding event date, the current date is smaller (earlier) than the preceding event date; therefore we obtain the response:

"Information about FINAL_GRADE is not known as yet; FINAL_GRADE will be known after final-exam."

Example Query 2: *Which text book is used for CMPT898?*

Logical Form:

```
((((((WHICH pl/sg v1) (v1 = (text)))
    ((EXIST sg v2) (v2 = (course (cname $CMPT898) (sec 01))))
    ((EXIST sg v3) (V3 = (time (term=spring) (year=1986))))
    (USE (subj v2) (obj v1) (time v3))))))
```

SQL Query:

```
SELECT  TEXT
FROM    TEACH
WHERE   CLASS# IN
        SELECT  CLASS#
        FROM    OFFERING
        WHERE   CNAME = 'CMPT898'
          AND   SEC = '01'
```

If this query fails extensionally, one possible response from the database might be:

```
TEXT cannot be found.
```

In this case, x is an attribute field, TEXT, so procedure Check_Always_Null is first activated. TEXT is found in the EXC-relation, see Figure 4-2 (i). The set of exceptions of x, except_set, is {**thesis-course**}. Since there is only one element in the exceptional set of TEXT, **thesis-course** is the only known exception to TEXT. **thesis-course** is a view in the V-relation, see Figure 4-2 (j), so the view definition of

thesis-course is evaluated, and variable S has the value {CMPT898, CMPT899,...}, the result of the evaluation. The attribute field name of the elements in S is CNAME. Since CNAME is present in the logical form with value CMPT898, and CMPT898 is in S, the procedure responds:

> **"CMPT898 is a thesis-course.**
> **The query relationship does not apply to concept of**
> **type thesis-course,**
> **that is, thesis-course does not have TEXT."**

COURSE-DESCRIPTION:

CNAME	DESCRIPTION
CMPT101	...
CMPT102	...
CMPT103	...
CMPT105	...
CMPT108	...
.	...
.	...

(a)

COURSE-DESCRIPTION:

CNAME	DESCRIPTION
CMPT101	...
CMPT102	...
CMPT103	...
CMPT105	...
CMPT106	...
CMPT108	...
.	...

(b)

Figure 5-1: Two Alternative Course Descriptions in the Database

5.3. Intensional Misconception

In this section, a second algorithm is described for handling intensional misconceptions. There are two kinds of relations in the knowledge base that are designed to provide more informative responses when a null answer arises due to a user's misconception that fails intensionally. These relations include the AE-relation and the AG-relations. When a query fails extensionally, we have only to look for relevant extensional facts to provide a quality response, as described in the previous section. However, when a query fails intensionally, we require the knowledge base to check the relevant relationships, in particular, any constraints on those relationships known in the knowledge base.

We begin with the logical form of a null answer query and AG-relations in the knowledge base. All of the relevant relations found in the logical form are initially preserved in the stack variable R, among which are those relations that are of an associated nature to be further examined in order to ascertain which relationship that the user has misconceived. For each relevant AG-relation, r, procedure Check_Intensional_Misconception first determines whether all concepts/values specified in the logical form for this relation are legitimate values. If there exist concepts/values that do not comply with the constraints specified in the AG-relation, the user's misconception will be corrected by certain responses. Otherwise, Check_Intensional_Misconception performs a divide and intersect action to determine the validity of the relationship in the user's specification, and the algorithm will stop as soon as the user's first intensional misconception is found. The divide part will be done by procedure Match_&_Extract, where for each participating concept,[22] constraints related to it that are found in the AG-relation are extracted from the relation resulting in a subrelation of r. Hence, if r represents a n-ary relationship, then we will have n subrelations of r for each participating concept, see Figure 5-3. These n subrelations are then intersected in Perform_Intersection. If there is a non-empty intersection, all of the participating concepts are valid values for r, and there is no explanation to the null response in terms of an intensional misconception on r. Otherwise, the ***maximal explanation set*** is acquired by procedure Check_Intensional_Misconception.

Consider the following hypothetical example: R_h is a hypothetical quaternary AG-relation, and suppose the logical form output specified an instance (a, b, c, d) of

[22]By participating concept, I mean the concept specified in the logical form for the relationship represented by the AG-relation.

Rh:

A	B	C	D
a	b	c	d1
a	b	c2	d2
a3	b	c	d

the relationship R_h. The largest correct combinations of instance i = {a, b, c, d} found in R_h are {a, b, c} and {b, c, d}. The algorithm will thus generate a set of explanatory responses for all of these correct combinations, the maximal explanation set. An explanatory response to a largest correct combination, *com*, includes those tuples from the AG-relation which contains *com* (these tuples inform us which is(are) the other proper value(s) that can be with *com*), and those tuples from the AG-relation which contain elements of i - *com* (these tuples determine which values should be with those elements in i - *com*). So an explanatory response for {a, b, c} would be Template(R_h, T_1), and Template(R_h, T_3)[23], where relation T_1 contains the tuple (a, b, c, d1), and relation T_3 contains the tuple (a3, b, c, d). Notice that in some situations, some instances of the AG-relation might be printed more than once. In the worst case situation we will then have (a lot of) superfluous duplicates. In the above example, we will print the entire relation R_h with the addition of the two duplicated tuples. It is, of course, not very informative or cooperative to have responses such as these. However, the reason for choosing this response strategy is that we want to generate enough informative responses so as to minimize the number of iterations that a user has to interact with the system before he/she can obtain any helpful information. Further investigation on how to build a more sophisticated responding routine for this purpose is desirable. Presumably, we would need a user model to achieve this, but this investigation is beyond the scope of this thesis.

[23]Template(r,s) is a template routine which will generate restricted natural language sentences for the relationship represented by r instantiated by the values found in relation s. It is used in procedure Respond_Intensional_Misconception to generate natural language responses.

Turning Null Responses into Quality Responses

Besides Check_Intensional_Misconception, Match_&_Extract, and Perform_ Intersection, there are six other small subroutines or procedures that help provide quality responses for intensional misconceptions. Documentation for these subroutines and procedures can be found with the complete algorithm, see section 5.7.

5.4. An Example for Intensional Misconception

We illustrate the handling of null responses due to intensional misconceptions with the following example, which was used earlier in chapter 3. This query produces a null response given the database in Figure 5-2. Again the input query is parsed, and subsequently transformed into SQL query as shown below:

Example Query 3: *Which graduate students have taught CMPT861?*

Logical Form:

```
((((((WHICH pl v1) (v1 = (student (status $grad))))
    ((EXIST sg v2) (v2 = (course (cname $CMPT861))))
    ((EXIST nil v3) (v3 = (time (before (term=spring) (year=1986)))))
    (TEACH (subj v1) (obj v2) (time v3))))))
```

SQL Query:

```
SELECT student.name
FROM   student
WHERE  student.status = 'grad'
  AND  student.name in
       SELECT instructor.name
       FROM   teach
       WHERE  class# in
              SELECT class#
              FROM   offering
              WHERE  cname = 'CMPT861'
                AND  semester = 'spring86'
```

Procedure Intensional_Misconception is activated. It initially stacks relations STUDENT, COURSE, TIME, and TEACH found in the logical form onto a stack R. The stack is processed in a last-in first-out order as usual, so relation TEACH is processed first. Since TEACH is an AG-relation in the knowledge base, see Figure 5-3 (a), procedure Check_Intensional_Misconception is thus invoked to see if any intensional misconception can be found related to this relation. The AG-relation TEACH is found in the knowledge base, then certain constraints imposed on it are found, therefore the divide and intersect process is carried out. First, the divide process is done by procedure Match_&_Extract. There are two attribute fields, **instructor** and **course**, found for the TEACH relation, see Figure 5-3 (a); therefore Match_&_Extract is called two times resulting in the two subrelations: S_1 and S_2, and variables e_1, e_2, val_1, and val_2 having the values shown in Figure 5-3 (b). e_1 (**graduate-student**) and e_2 (**graduate-course**) are both valid values in the AG-relation TEACH; therefore Perform_Intersection will find out what is(are) the largest possible combination(s) of e_1 (**graduate-student**) and e_2 (**graduate-course**) that can be found among the constraints of TEACH by intersecting the two subrelations S_1 and S_2. Since TEACH is only a binary relation, and there is no intersection between the two subrelations, S_1 and S_2, see Figure 5-3 (b), the recursive procedure Perform_Intersection terminates, returning an empty set as the result. The empty set that results from Perform_Intersection implies that the user's specification for TEACH does not meet the associative constraints that the knowledge base knows, so procedure Respond_Intensional_Misconception responds to the user with the largest possible set of maximal combinations of the e's, which is $\{\{e_1\}, \{e_2\}\}$ together with the appropriate values found in the constraints, and with some corrective responses attached as follows:

> **"Graduate students teach undergraduate courses.**
> **CMPT681 is a graduate course.**
> **Professors teach graduate courses."**

A user's intensional misconception is found, procedure Check_Intensional_Misconception sets the stop flag to true, and this tells procedure Intensional_Misconception to terminate.

Turning Null Responses into Quality Responses

OFFERING:

CNAME	SEMESTER#	CLASS#
CMPT861	spring86	s86CMPT861
...	...	...
...	...	...
...	...	...

TEACH:

CLASS#	SEC	INSTRUCTOR#	TEXT
s86CMPT861	01	01	...
...	...	...	...
...	...	...	...
...	...	...	...

INSTRUCTOR:

INSTRUCTOR#	NAME	OFFICE	SEX	STATUS
01	B. Becker	...	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...

STUDENT:

STUDENT#	NAME	MAJOR	MINOR	SEX	STATUS
...	H. Holmes	...	...	...	graduate
...	J. Jones	...	...	...	graduate
...	S. Smith	...	...	...	undergraduate
...	T. Thomas	...	...	...	undergraduate

Figure 5-2: A sample database

AG-relation

TEACH:

INSTRUCTOR	COURSE
professor	graduate-course
professor	undergraduate-course
graduate-student	undergraduate-course

(a)

S1:

INSTRUCTOR	COURSE
graduate-student	undergraduate-course

e1 = graduate-student val1 = 0

S2:

INSTRUCTOR	COURSE
professor	graduate-course

e2 = graduate-course val2 = CMPT861

(b)

Figure 5-3: An example to illustrate Procedure Intensional_Misconception

5.5. Selection Mechanism

Based on the two algorithms given in the previous sections, the main control routine for solving user's misconceptions follows:

```
PROCEDURE Misconception
BEGIN
    intension := true;
    IF (∃ database response which suggests information as to
        which kind of failure) THEN
       Extensional_Misconception(intension);
    IF (intension) THEN
       Intensional_Misconception;
END {Misconception};
```

In some cases of extensional failure, where the whole tuple is missing, procedure Check_Object_Existency signals the non-existence of the relationship representing that tuple by setting the variable intension to be true. This initiates a further investigation into whether this extensional failure is caused by any intensional misconception. The main routine, procedure Misconception, is thus very simple. Called in when null events arise, it calls the two procedures: Extensional_Misconception and Intensional_Misconception in turn to detect any user's misconceptions. If the database response can supply certain information as to which kind of extensional failure, then procedure Extensional_Misconception is activated. Otherwise, we continue to search for intensional misconception.

The top down, hierarchical program structure for detecting user's misconceptions is depicted in Figure A-1 in Appendix A. Knowing the basic structure of the knowledge base relations, the algorithm described above is able to retrieve knowledge from the knowledge base independent of the extensional values present. The algorithm requires: all attribute field names to be unique in the database, and that there are no duplicated attribute columns in the database relations; all views define only unary relation in the knowledge base, and views for the concepts that belong to the same tree in the GH-relation have the same identifying attribute; and the GH-relation contains only rooted trees. The algorithm tries to account for extensional failures which are due to the ones described at the beginning of this chapter, null values that are due to other reasons, see [ANSI 75], are not covered here. In interpreting the precedence relationships represented by the tuples of the EG-relation, the algorithm does not apply any branching time temporal logic, it assumes that the EG-relation contains only simple paths. The algorithm will detect, at most, one extensional misconception and one intensional misconception. For queries that imply multiple misconceptions, the algorithm will not detect all misconceptions.

5.6. Algorithm for Extensional Misconception

In this section, the algorithm for extensional misconception is given. In the algorithms given below, comments are placed between the delimiters { and }. Capitalized words are used to designate reserved words. To avoid ambiguity, all procedures assume that every attribute field name will be prefixed with the relation name over which the attribute ranges. For example **R.a** indicates the attribute field **a** of relation **R**.

Procedure Extensional_Misconception

```
PROCEDURE Extensional_Misconception(input  variable : intension;
                                    output variable : intension);

BEGIN

   IF (x is a value) THEN Check_Object_Existency(intension);

   ELSE {x is an attribute field}
   BEGIN
       intension := false;
       found := false;
       Check_Always_Null;
       Check_Temporal_Event;
       IF (not found) THEN
            response: "x cannot be found,";
                      "Sorry, I have no explanation for this!";
```

```
    END {else};
  END {procedure}.
```

Procedure Check_Object_Existency

This procedure determines the existence of a certain object in the database. If the attribute X is an entity identifier then the knowledge base will have a complete list of its members by consulting the ENT-relation[24]. In brief, this procedure proceeds as follows:

1. if X is an entity identifier then the algorithm will determine if x is a member of the known existing objects.

    1. if x is a known existing object (and it is not found in the query relationship), then the algorithm concludes that the relationship represented by the relation that contains x is false according to the "world" modelled by the database. Further investigation into any user's intensional misconception is necessary.

    2. otherwise x is not an existing object[25]. This is an extensional failure, so that no further investigation on intensional failure is necessary.

2. if X is not an entity identifier, no information concerning the existence of this object, x, can be found. There is no proof of any incorrect assumption about the existence of an object that the user has made. Further investigation into any intensional failure is necessary.

[24]Because the knowledge base is closed under all entities' identifiers.

[25]Thus the query relationship is incorrect because of the user's wrong presumption of the existence of x.

```
PROCEDURE Check_Object_Existency(input  variable : intension;
                                 output variable : intension);

BEGIN
  intension := true;
  P-set := [select Entity from PG-relation where Property = X];
  IF (P-set <> Ø) THEN
  BEGIN
    stop := FALSE;  {Note: An attribute can be a property of more than one
                           entity. We try to find the entity that has this
                           attribute as the identifier.}
    WHILE (~stop AND (P-set <> Ø)) DO
    BEGIN
      temp := first element of P-set;
      P-set := P-set - [temp];
      cr := [select CLR from ENT-relation where ENTITY = temp ];
                                   {Check if X is an entity identifier}
      IF (the primary key attribute field of CR = X) THEN
      BEGIN
        stop := TRUE;
                                   {Check if x is a known entity}
        IF (x ∈ [select X from CR]) THEN
          (conclude that the relationship represented by the relation that
           contains x is false according to the world modelled by the DB);
        ELSE BEGIN
          (signal x is not an existing object for entity type X to
           clear user misconception);
          intension := false;
        END {else};
      END {if};
    END {while};
    IF (~stop) THEN
      response: "Sorry! The DB doesn't know of any X with value x
                 in relationship R at this present time.";
                        {where R is the relation for X in the logical form.}
  END {if}
  ELSE
     response : "Sorry! The DB doesn't know of any X with value x
                 in the relationship R at this present time.";
END {Check_Object_Existency}.
```

Procedure Check_Always_Null

This procedure determines the case where a value is *not valid* for an individual. This procedure will ascertain exceptional cases from the EXC-relation, and decide if those exceptional cases occur in the information specified in the *logical form.*

```
PROCEDURE Check_Always_Null
BEGIN
   IF x ∈ [select concept from EXC-relation] THEN
   BEGIN
      except_set := [select EXCEPTION from EXC-relation where concept = x];
                    {Note: here we allow the exceptional set to be more than
                     one set. For example, all courses have a text except
                     thesis_course and some seminar_course.}
      WHILE (except_set <> ∅) DO
      BEGIN
      S := ∅;
           exception := first element of except_set;
           except_set := [except_set - exception];
                    {If the exceptions are specified as a generic concept in the
                     GH-relation then subroutine evaluate_exception will
                     recursively gather all the elements that belong to that
                     subtree.}
      IF exception is a SUP in the GH-relation THEN
           evaluate-exception(exception, S, attr)
      ELSE          {If the exceptions are specified as a V-relation, then we
                     just evaluate the view definition of that V-relation.}
         IF exception is a V-relation THEN
            S := evaluated set of the SQL query in the view definition of exception
            attr := the attribute field name of the select statement in the
                       view definition.
         ELSE    {If the exceptions are specified as an entity, then attr will
                  be the entity identifier, and S will be the element set of the
                  entity identifier.}
            IF (exception is an entity) THEN
            BEGIN
               cr := [select CLR from ENT-relation where ENTITY = EXCEPTION];
               S := [select primary key attribute from CR];
               attr := primary key attribute field name of CR;
            END {if};
                              {Check if these exceptional cases occur in the
                               information specified in the logical form}
      IF attr is one of the attribute fields specified in the logical form THEN
         IF there is a value of attr, say val, given in the logical form THEN
            IF val ∈ S THEN
               response: "val is a (an) exception.";
               response: "The query relationship does not apply to concept of
                          type exception,";
               response: "that is, exception does not have x.";
               found := TRUE;
      END {while};
   END {if};
END {Check_Always_Null}.
```

Procedure Check_Temporal_Event

This procedure checks for the case where a value is *valid* but does *not yet exist* for an individual. If x is a temporal event, then there are two relevant facts which can be extracted from the EG-relation: x's preceding event and its successor event. There are two cases in which this procedure can specify a definite response to the user: (1) if x's preceding event has not occurred yet, then x has not occurred either; and (2) if x's successor event has occurred, but the information is still not known as yet, then the information is not considered to be available. If neither of the above cases is true, then the algorithm attempts to respond with respect to relevant dates (either the preceding date or the successor date or both). This procedure can only handle events which have at most 1 preceding event, and at most 1 successor event at this stage.

```
PROCEDURE Check_Temporal_Event

BEGIN {If x is an E-relation then we can get the date information for x directly
        and give the appropriate response.}
   IF x is an E-relation THEN
   BEGIN
      found:= TRUE;
      date := [select date from x where season = CURRENT_SEASON];
      IF current date < date THEN
         response: "Information about x is not known as yet;
                    x will be known by date."
      ELSE response: "Information about X is not available.";
   END {if}
   ELSE
   BEGIN
      successor_date := 9999999;
      preceding_date := -9999999;
                                 {Get the relevant preceding event if possible.}
      IF x is a SUB in the EG-relation THEN
      BEGIN
         preceding_event := SUP of x;
         preceding_date := [select date from preceding_event
                            where season = current_season];
      END {if}
                                 {Get the relevant successor event if possible.}
      IF x is a SUP in the EG-relation THEN
      BEGIN
         successor_event := SUB of x;
         successor_date := [select date from successor_event
```

```
                              where  season = current_season];
        END {if}
                              {If the preceding event of x has not occurred, then
                               x has definitely not occurred.]
        IF current_date =< preceding_date THEN
        BEGIN
           response : "Information about x is not known as yet;
                       x will be known after preceding_data.";
           found := TRUE;
        END {if}
        ELSE       {If the successor event of x has occurred already, and the
                    information is still not known, then the information is
                    considered to be not available for x.}
           IF current_date >= successor_date THEN
           BEGIN
              response: "Information about X is not available.";
              found := TRUE;
           END {if}
           ELSE {Respond with respect to both the preceding event date and the
                 successor event date if they are available.}
              IF ((successor_date <> 999999) and (preceding_date <> -999999)) THEN
              BEGIN
                 response: "Information about X is not known, this information
                            should be available between preceding_date and
                            successor_date.";
                 found := TRUE;
              END {if}
              ELSE       {Respond with respect to the successor event date if the
                          information is available.}
                 IF (successor_date <> 9999999) THEN
                 BEGIN
                    response: "Information about X is not known.  But if the
                               information is available, it would be available
                               before successor_date.";
                    found := true;
                 END {if}
                 ELSE {Respond with respect to the preceding event date if the
                       information is available.}
                    IF (preceding_date <> -9999999) THEN
                    BEGIN
                       response: "Information about X is not known. But if the
                                  information is available, it would be available
                                  after preceding_date.";
                       found := TRUE;
                    END {if}
     END {else};
  END {Check_Temporal_Event}.[26]
```

[26]We assume that the SUB or SUP of an attribute field in the database has to be an E-relation, that is, there will be no tuple like (TEACH.instructor#, CLASS_ROOM.room#) in the EG-relation. Also we do not handle transitivity here.

Subroutine evaluate_exception

This subroutine recursively gathers all of the elements that belong to any subtree that are in the GH-relation. Evaluate_Exception makes certain assumptions:

- that there is only one attribute field specified in the select statement of the view definition.
- that all views of the children (SUB field) of a parent (SUP field) will have the same attribute field name in the select statement of their view definitions because they are all describing (a subset of) one super-concept.
- that all the hierarchies present in the GH-relation are trees, that is, no cycle is allowed.

```
SUBROUTINE evaluate_exception($v$, $s$, $a$)
BEGIN
   IF $v$ is not the SUB of the GH-relation THEN
      FOR all children (c) of $v$ DO
          IF c is not a SUP in the GH-relation THEN
             $s$ := $s$ $\cup$ evaluated set of the SQL query in the view definition of c;
          ELSE
             evaluate_exception(c, $s$, $a$);
   ELSE
      $s$ := $s$ $\cup$ evaluated set of the SQL query in the
           view definition of $v$;
      $a$ := the attribute field name of the select statement
           in the most recent view definition;
END {evaluate_exception};
```

5.7. Algorithm for Intensional Misconception

Procedure Intensional_Misconception

This procedure checks all the relations in the logical form that can fail intensionally, that is, all AE-relations that appear in the logical form, and calls procedure Check_Intensional_Misconception to identified any user's misconception as to the presence of a non-existing relationship.

```
PROCEDURE Intensional_Misconception

BEGIN

   Get the set of relation names from the logical form
   in the order they appear and put them in R
                                                  {where R is a stack.}

   stop := false;
   WHILE ((R <> Ø) AND (~stop)) DO
   BEGIN
      r := POP(R);
      IF (r ∈ AE-relation) THEN
         Check_Intensional_Misconception(r,stop);
      {If r is not an AG-relation, see if there are any relations of
       the form r.* that are AG- relations and process these
       relations as well.}
      IF ((~stop) AND
          (r is a prefix of some relations in the AE-relation)) THEN
         PUSH(R, the set of relation names in the AE-relation
                 with r as a prefix);
   END {while};
   IF (~stop) THEN
      response: "Sorry, I have no explanation for this null event!";

END {Intensional_Misconception}.
```

Procedure Check_Intensional_Misconception

This procedure tries to match the information specified in the logical form with the constraints found in the input AE-relation R. For each attribute field, A_i, of R, a relation S_i results from selecting those tuples of R with the value of A_i matching the information specified in the logical form. This procedure then invokes Perform_Intersection to intersect the S_i's to find out where the user's misconceptions about the non-existing relationships are, and then generates the appropriate response. Template_Wrong_Arg(r,n,arg) is a template that will give us the appropriate restricted natural language response if arg is the improper argument for the nth field of relation r. For example, if TEACH(**instructor**, **course**) is a relation that represents "instructor teaches courses", then Template_Wrong_Arg(TEACH, 2, secretary) will generate a response like: "Secretaries do not teach courses."

```
PROCEDURE Check_Intensional_Misconception(input  variable: R;
                                          output variable: stop);
BEGIN
   {If R has certain imposed constraints then check which
    constraints are violated by the users.                    }
   stop := true;
   IF (Size(R) <> 0) THEN
   BEGIN
      Num := the # of attribute field in R;
      {For each attribute field, A_i, of R, check if it appears
       in the logical form; if yes, select the tuples in R with A_i
       having the value specified in the logical form. $i means the
       ith attribute field.   }
      FOR i := 1 TO Num DO
          Match_&_Extract(R,$i,S_i,e_i,val_i);
      do_intersect := true;
      FOR i := 1 TO Num DO
          IF (e_i = 0) THEN
          BEGIN
             get the argument value, say arg, that is supposed to
             be in this attribute field of R from the logical
             form.

             Template_Wrong_Arg(R, i, arg);
             do_intersect := false;
          END {if};
      IF (do_intersect) THEN
      BEGIN
         {Do the intersection of all the relations resulting from
          Match_&_Extract, and give the appropriate response.       }
         whole_set := [$S_1$, $S_2$, ..., $S_{Num}$];
                                          {whole_set is a global variable.}
         intersect_set := Ø;
         Perform_Intersection(Num, whole_set, intersect_set);
         response_set := Ø;
         IF (intersect_set <> Ø) THEN
         BEGIN
            IF (whole_set ∈ intersect_set) THEN
               stop := false;
            ELSE BEGIN
               num := |largest element of intersect_set|;
               WHILE (intersect_set <> Ø) DO
               BEGIN
                  temp := first element of intersect_set;
                  intersect_set := intersect_set - [temp];
                  IF (|temp| = num) THEN
                     IF (temp ∉ response_set) THEN
                     BEGIN
                        response_set := response_set + [temp];
                        Respond_Intensional_Misconception(temp);
                     END {if};
               END {while};
            END {else};
         END {if}
         ELSE
            Respond_Intensional_Misconception(intersect_set);
      END {if};
   END {if};
END {Check_Intensional_Misconception}.
```

Procedure Match_&_Extract

This procedure tries to match certain information in the logical form with a concept in the given attribute column, attr, of a relation, r. If a concept, e, is identified to be present in the logical form, this procedure will select the tuples of r that have value e in the attribute column attr and return them in S. The concept e is returned to the calling procedure, and if e has a value, val, specified in the logical form, this value is also returned in variable val, or a 0 is returned in val otherwise.

```
PROCEDURE Match_&_Extract (input  variables: r, attr;
                           output variables: S, e, val);

BEGIN
   S := [select unique attr from r];
   e := 0;
   val := 0;
   Match_&_Extract_stop := false;
   WHILE ((S <> Ø) AND (~Match_&_Extract_stop)) DO
   BEGIN
      e1 := first element of S;
      S := S - [e1];
      concept_stack := Ø;
      Push_Sub_In_Stack(concept_stack, e1, CLASS-relation);
      Push_Sub_In_Stack(concept_stack, e1, GH-relation);
      IF (concept_stack = Ø) THEN
         PUSH(concept_stack, e1);
      WHILE ((concept_stack <> Ø) AND (~Match_&_Extract_stop))DO
      BEGIN
         e1 := POP(concept_stack);
         IF (In_V_Relation(e1)) THEN
         BEGIN
                      {Match concept appearing in the logical form.}
            Match_Logical_Form(e1,val,match);
            IF (match) THEN
            BEGIN
               Match_&_Extract_stop := true;
               {Extract the tuples of r with concept, e, in attribute
                field, attr.}
               S := [select * from r where attr = 'e1'];
               e := e1;
            END {if};
         END {if}
         ELSE BEGIN
            Push_Sub_In_Stack(concept_stack, e1, CLASS-relation);
            Push_Sub_In_Stack(concept_stack, e1, GH-relation);
         END {else};
      END {while};
   END {while};
END {Match_&_Extract}.
```

Procedure Match_Logical_Form

In this procedure, e is an input concept that is defined in a view definition. This procedure will match if this concept is being specified in the logical form or not; if yes, match will have the value true, and false otherwise. A value, v, is also returned if e has a value specified in the logical form, or v has value 0 otherwise.

```
PROCEDURE Match_Logical_Form (Input  variables: e;
                              Output variables: v, match);

{Let $R_e$ be the relation in the from statement of the view
 definition of e.}

BEGIN
   match := false;
   v := 0;
   IF ($R_e$ is in the logical form) THEN
   BEGIN
      IF ($\exists$ conditions in both the logical form and the
          view definition of e, say $c_{LF}$ and $c_e$
          respectively, for $R_e$) THEN
          IF ($c_e$ matches exactly as $c_{LF}$) THEN
             match := true
          ELSE
             IF (the evaluated set of e's view definition
                 contains the value(s) in $c_{LF}$) THEN
                 match := true;
       IF (one of the values in $c_{LF}$ is a primary key
           value of R) THEN
          v := the primary key value in $c_{LF}$;
   END {if}
END {Match_Logical_Form};
```

Push_Sub_In_Stack

This procedure checks if concept, c, is in the SUP field of relation, in_relation. If yes, it will push concepts that are SUB of c onto stack, in_stack.

```
PROCEDURE Push_Sub_In_Stack(input  variables: in_stack, c, in_relation;
                            output variable : in_stack);

BEGIN

   IF (c $\in$ [select SUP from in_relation]) THEN
      PUSH(in_instack, [select SUB from in_relation where SUB = c]);
END {Push_Sub_In_Stack}.
```

Procedure Perform_Intersection

This procedure does an intersection on the n relations given in the inset. In the first call to this routine inset will contains the n subrelations resulted from procedure Match_&_Extract. If there is no intersection between these relations in the first call, then the implication is that there is a missing relationship. In such case, this procedure will recursively handles the intersection of n relations, and returns all possible non-empty intersections of the n relations in outset. The join operator used in this procedure is the natural join operator.

```
PROCEDURE Perform_Intersection(input  variables : num, inset;
                               output  variable : outset);

BEGIN

   inset_1 join inset_2 join .... join inset_num over all
      attribute fields giving intersect;
   outset := Ø;
   {If there is no intersection in the relations in inset, then
    find out which is(are) the relation(s) that is(are) not
    with the other relations.}
   IF (Size(intersect) = 0) THEN
   BEGIN
      IF (num > 2) THEN
      BEGIN
         Choose(num-1, inset, choose_set);
         WHILE (choose_set <> Ø) DO
         BEGIN
            nextchoice := first element of choose_set;
            choose_set := choose_set - nextchoice;
            Perform_Intersection(num-1, nextchoice, outset);
         END {while};
      END {if};
   END {if}
   ELSE
     outset := outset U [inset];
END {Perform_Intersection}.
```

Procedure Choose

This is a recursive procedure which chooses all the possible n number of elements from inset and puts the result in outset.

```
PROCEDURE Choose(input  variables: n, inset;
                 output  variable: outset);

BEGIN

   outset := Ø;
   IF ((inset <> Ø) AND (n <> 0)) THEN
   BEGIN
      tempset := Ø;
      WHILE (|inset| > n-1) DO
      BEGIN
         temp := first element of inset;
         inset := inset - [temp];
         Choose(n-1, inset, chooseset);
         tempset := tempset U
                    Distribute_Union([temp], chooseset);
      END {while};
      outset := tempset;
   END {if};
END {Choose}.
```

Function Distribute_Union

This function performs a union of x to each element of inset, and returns the resulting set as the function value.

```
Function Distribute_Union(x, inset) : set;

BEGIN

   du := Ø;
   IF (inset = Ø) THEN
      du := [x];
   WHILE (inset <> Ø) DO
   BEGIN
      temp := first element of inset;
      inset := inset - [temp];
      du := du U [x U temp];
   END {while}
   Distribute_Union := du;
END {Distribute_Union}.
```

Respond_Intensional_Misconception

This procedure generates responses for the misconceptions found. It also generates the corrective information found in inset. For this procedure, all S_i's, val_i's, and e_i's resulting from Match_&_Extract are made accessible to it. Template(r, s) is a template routine that will generate restricted natural language responses with the relationship represented by r instantiated by the values found in relation s. For example, if TEACH(**instructor**, **course**) represents the relationship: "instructors teach courses", then Template(TEACH, S_1), see S_1 in Figure 5-3, will generate a response like: "Graduate students teach undergraduate courses". The join operator used in this procedure is again the natural join operator.

```
PROCEDURE Respond_Intensional_Misconception(inset);

BEGIN

   IF (inset = Ø) THEN
   BEGIN
      FOR i := 1 TO |whole_set| DO
      BEGIN
         IF ($val_i$ <> 0) THEN
            response: "$val_i$ is a/an $e_i$.";
         Template(R,$S_i$);
      END {for};
   END {if}
   ELSE BEGIN
      $inset_1$ join $inset_2$ ... join $inset_{|inset|}$
           over all attribute fields giving intersect;
      Template(R,intersect);
      inset := whole_set - inset;
      WHILE (inset <> Ø) DO
      BEGIN
         Si := first element of inset
         inset := inset - [Si];
         i := the subscript found in the name of relation Si;
         IF ($val_i$ <> 0) THEN
            response: "$val_i$ is a/an $e_i$.";
         Template(R,Si);
      END {while};
   END {else};
END {Respond_Intensional_Misconception}.
```

Function In_V_Relation

This is a boolean function that tests if e is in the V-relation or not.

```
FUNCTION In_V_Relation(e) : boolean;
BEGIN

   IF (e ∈ [select view from V_relation]) THEN
      In_V_relation := true
   ELSE
      In_V_relation := false;
END {In_V_Relation}.
```

Chapter 6
Concluding Remarks

When a query submitted by the user fails to elicit any answer, the user often finds the situation unsatisfactory and frustrating. The problem is more aggravating for database systems with a natural language front-end as users are not expected to have much knowledge about the database structure. Earlier research efforts in natural language database systems has concentrated mostly on natural language interfaces; more recent work in this area has concentrated on cooperative responses. In particular, Kaplan, Janas, and Motro (see chapter 2) have worked to provide quality responses for null answer queries [Kaplan 78], [Janas 79], [Motro 86]. However, domain specific knowledge is still indispensable for providing quality responses in a natural language database system. In this thesis, I have introduced **an initial classification of null event problems** in natural language database systems; this initial classification provides a structured way of approaching the problem of turning null responses into quality responses. As a partial solution to the complex problem of quality (cooperative) responses in natural language database systems, we **constructed a relational knowledge base model** which supplies information for diagnosis of failed queries. It provides meta-level data information to facilitate quality responses for correcting user's misconceptions. **Algorithms which provide quality responses** to null events arising from user's misconceptions (see Figure 3-2) further demonstrate the kind of quality responses we can obtain from the information supplied by the relational knowledge base model. Although, currently, the knowledge base model has to be hand-coded by the database administrator (or the system designer), the algorithms manipulating the knowledge base are domain independent. Extensional values in the knowledge base are all transparent to the algorithms, and thus enhance portibility.

We have examined the feasibility of providing correct interpretations of certain null values occurring in a database, and providing cooperative responses using the relational knowledge base model. This knowledge base model provides us a general scheme[27] for representing and organizing knowledge in a natural language database system without rendering the system ad-hoc, and, in principle, this model is transformable to other database models as well. Furthermore, the relational knowledge base model offers the following advantages:

1. inexpensive to incorporate into the database;
2. conceptually simple for database administrator to understand and design;
3. immediately applicable as an add-on to an existing databases;
4. portable;
5. independent of updates to the database.

There are, of course, disadvantages associated with this relational knowledge base. With a relational model, procedural and heuristic knowledge is difficult to represent. Also, the relational knowledge base model is good for representing homogeneous knowledge, but it will be cumbersome in representing heterogeneous knowledge, in particlar, inference rules cannot be easily put into a relational

[27]General, but possibly not complete. Realization of a complete general scheme requires further investigation into the other classes of null event problems as well.

representation. The relational knowledge base model will not be able to provide a visual immediacy of interrelationships between concepts. A cycle in the GH-relation will not be as easily detected as if the GH-relation is represented in a semantic network.

6.1. Future Research

6.1.1. Short term research directions

The initial classification is not complete and requires extension. Further research into other classes of null events problems, for example, null events due to the lack of knowledge about the database structure, null events due to the lack of ability to handle partial information, etc., will certainly amalgamate more detailed classification categories and eventually result in a more comprehensive classification.

The knowledge base model introduced in chapter 4 is designed specifically to incorporate information necessary to provide quality responses to null value problems. All of the information in the model is hand-coded at present. Automatic generation of the entire knowledge base model appears difficult because domain-specific knowledge needs to be acquired in some way. Automatic generation of some of the relations in the knowledge base model is possible nonetheless. Given a relational database, one can proceed to automatically generate the knowledge base model, to a certain degree, as follows:

step 1. locate major concepts[28] and their properties existing in the DB. (the ENT-relation and the PG-relation)

[28] By major, we mean the concepts that occur in the database, and only those concepts.

step 2. identify any association/relationship existing between concepts. (the AE-relation and the AG-relations)

step 3. generate hierarchies for all major concepts. (the CLASS-relation and the GH-relation)

I have developed a preliminary algorithm which performs steps 1 & 2 listed above. We will explain more about how to carry out these steps later.

In the relational database model:

- entities have properties - their attributes;
- each entity set has a key that uniquely identifies each entity in an entity set;
- there are also cases in which the entities of an entity set are not distinguished by their attributes (properties) but rather by their relationship to entities of another type.

Assuming that we have a database that is in 4th normal form[29], and based on the relational model outlined above, we make the following two observations:

1. relations which share some common attribute fields in their primary key are considered to be related to each other, and they are meant to describe certain concepts that those common attribute fields comprise. For example, relations like: OFFERING, COURSE-PREREQUISITE, COURSE-PREVIOUS-NAME, COURSE-DESCRIPTION, in Figure 4-1, are all related to the concept **course**, and **course** is made up of CNAME.
2. if a concept is made up of only one attribute field, a complete list of all[30] their members can be obtained from a relation whose primary key are the corresponding attribute field of that concept. For example, since

[29]Let R be a relation scheme and D the set of dependencies applicable to R. We say R is in fourth normal form if whenever there is a multivalued dependency $X \rightarrow\rightarrow Y$, where Y is not empty or a subset of X, and XY does not include all the attributes of R, then X is a superkey of R, where a superkey is any superset of a key. [Ullman 82]

[30]According to the closed world assumption that most databases adopt.

course is made up of CNAME only, COURSE-DESCRIPTION is assumed to contain the complete list of all the courses.

A high level description for part of the process for automatic hierarchy construction is outlined in Appendix B. The algorithm is illustrated using the sample database schema in Figure 4-1. The algorithm is incomplete and it requires some deeper analysis.

For the database schema illustrated in Figure 4-1, this algorithm will automatically generate the following:

PG-relation:

ENTITY	PROPERTY
course	DESCRIPTION
course	PREREQUITE
course	PREVIOUS_NAME
course	CNAME
semester	SEMESTER#
semester	YEAR
semester	SEASON
class	CLASS#
class	DEPT
class	UNITS
class	SEC
dept	DEPT#
dept	CHAIRMAN
dept	FACULTY
dept	FACILITY
instructor	INSTRUCTOR#
instructor	NAME
instructor	DEPT
instructor	OFFICE
.	.
.	.
student	STUDENT#
student	NAME
.	.
.	.
OFFERING.1	CLASS#
TAKE.1	FINAL-GRADE
CLASSROOM.1	ROOM#

(a)

ENT-relation:

ENTITY	CLR
course	COURSE-DESC
student	STUDENT
instructor	INSTRUCTOR
dept	DEPT
semester	SEMESTER
class	CLASS-DEPT

(b)

AE-relation:

RELATION
OFFERING.1
TEACH.1
TAKE.1
CLASS-ROOM.1

(c)

OFFERING.1(course, semester)

TEACH.1(class, instructor, TEXT)

TAKE.1(class, student)

CLASS-ROOM.1(class, TIME)

(d)

Notice that this algorithm cannot generate constraints for the AG-relations, as in Figure 5-3. This domain specific knowledge needs to be obtained from an external source. To automatically generate concept hierarchies, generalization and aggregation hierarchies from the database is more complicated. McCoy's ENHANCE system [McCoy 82] introduced in chapter 2 automatically generates an annotated generalization hierarchy on the entities in the database. This enables ENHANCE to provide responses to certain kinds of queries regarding the database structure. The knowledge base model introduced in this thesis also provides concept hierarchies, generalization and aggregation hierarchies. Combining these two techniques might provide a good basis for solving the null event problem due to the lack of knowledge about the database structure, and hence quality responses for this category of null event would follow. Also, performing a preprocessing of the database to discover exceptional cases[31] will also suggest some way of partitioning a hierarchy.

Partial information such as that illustrated in section 3.1.4 is not unusual in the real world; our ability to handle this kind of partial information is useful in an information system. As discussed in chapter 3, partial information such as: "*Joe will teach CMPT810 next semester or Art will teach CMPT810 next semester*" can only be approximated in the database as:

```
    teach(CMPT810-fall, sect01, null).
```

This is, in fact, a kind of extensional failure, if we try to find out "*Who is teaching CMPT810 next semester?*" If a system is able to handle partial information, then in the case of an extensional failure, the system should also check for the existence of any partial information, and the selection mechanism has to be further modified.

One possible way to handle this kind of partial information is to extend the

[31] that is, those entities that have null value all the time for a particular attribute field.

knowledge base model to incorporate one more relation, say the DIS-relation (DISjunction relation), to store partial information. This DIS-relation can be a binary relation where the CONCEPT field contains an index to any entry in the database. The index can be in the form: **attribute field name-primary key value**[32], see Figure 6-1. A POSS field contains the possible value for the corresponding value in the field CONCEPT. Elements in the POSS field, therefore, should have the same attribute domain as the attribute field domain for the corresponding elements in the CONCEPT field.

TEACH:

CLASS#	SEC	INSTRUCTOR	TEXT
f86CMPT810	01	null	...
...	...	...	...
...	...	...	...
...	...	...	...

DIS-relation:

CONCEPT	POSS
TEACH.INSTRUCTOR-f86CMPT81001	Art
TEACH.INSTRUCTOR-f86CMPT81001	Joe
...	...

Figure 6-1: Suggested DIS-relation to capture disjunctive information.

In this way, when there is an extensional failure, we can have some mechanism to index into the DIS-relation to find out whether there is(are) any possible value(s) known in the domain for the corresponding null value. However, this renders the knowledge base dependent on the data values in the database. The knowledge base will no longer be independent of updates to the database; whenever there are database updates, the knowledge base has to be updated as well. An input routine for this kind of partial information has to interact with the database. This

[32]or attribute field name concatenated with primary key values for composite primary key. Notice the concatenation of the composite primary key values has to follow some strict ordering; otherwise we cannot maintain the uniqueness of the index key.

reduces portability. Furthermore, disjunctive information such as: *P(a)* or *Q(b)* cannot be handled by the above DIS-relation. Further research is required on this issue.

In the knowledge base model, the database is not considered to be closed. Only simple entities are closed in the knowledge base, and all AG-relations are closed in the knowledge base. So far, the knowledge base model has only been used for the purpose of providing information which would be useful to correct any user's misconceptions.

It will be very interesting to determine how to extend the knowledge base model to incorporate negative facts concerning the database. Incorporating negative extensional facts again involves existing data values in the database. Nevertheless, an investigation on extending the knowledge base model to incorporate negative intensional facts of the database might prove fruitful.

The algorithms given in chapter 5 assume that there is no cycle in the CLASS-relation, and in the GH-relation. This is because they are considered to represent hierarchies in the application domain. Also for the EG-relation, the algorithm only handles events that have at most one preceding event and at most one successor event. Extending the algorithms to allow cycles and multiple events would be appropriate.[33]

[33]Notice how Mays can handle this kind of problem by using the branching time temporal logic technique. [Mays 82]

6.1.2. Long term research directions

In chapter 4, I discussed the problem of null values in the knowledge base. We can basically incorporate another meta-knowledge level of information regarding null values in the knowledge base. However, this results in infinite meta-levels of information in the worst case situation. The approach of having infinite meta-levels of information about the knowledge base's null values is another direction for future research since it might be conceptually advantageous as suggested by Brain Smith's work on 3-LISP.

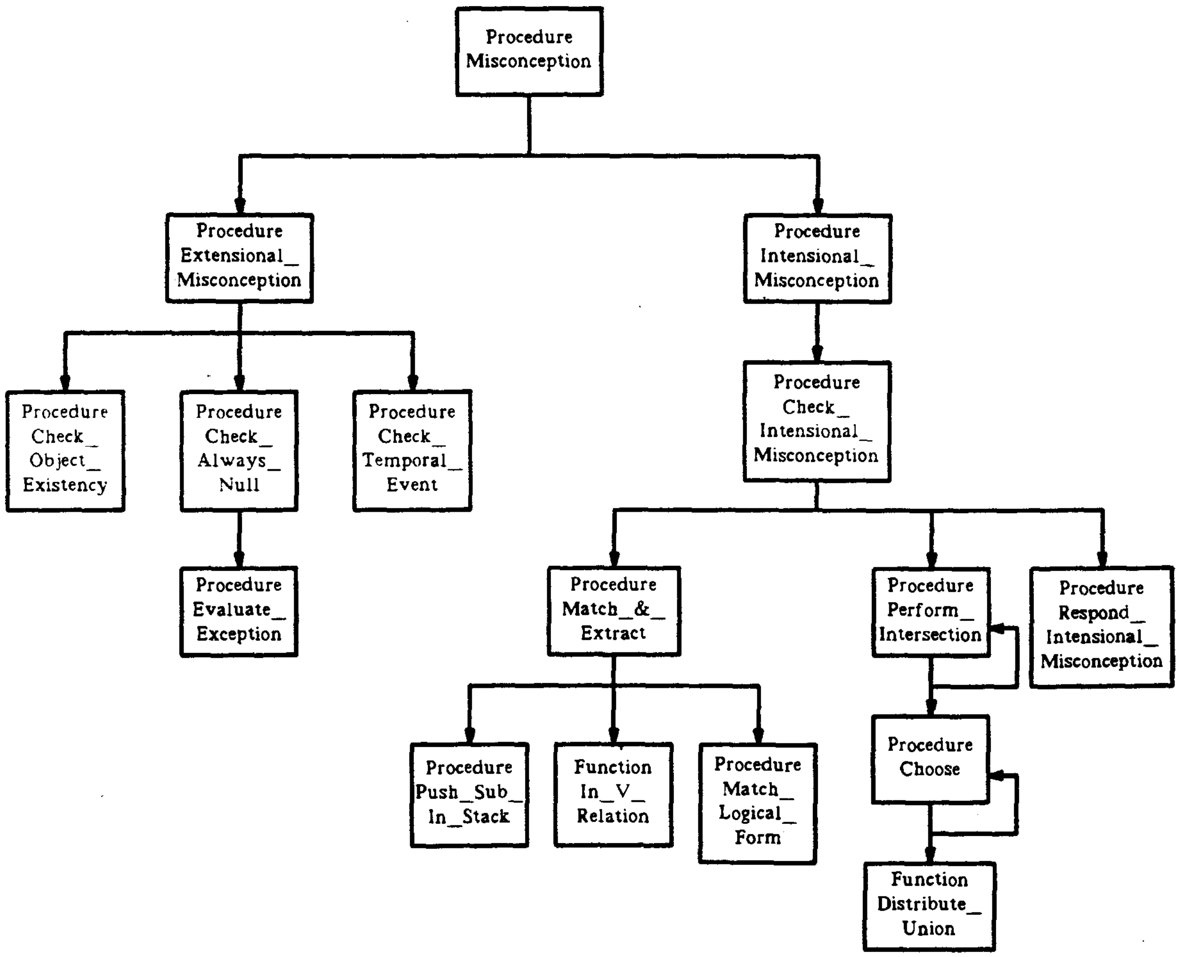
One important missing component in a natural language database system is the query generator which transforms the logical form from a semantic interpreter into a formal database query, say SQL. A lot of information in the knowledge base can be shared with the semantic interpreter and the query generator. Future research which bridges the gap between the semantic interpreter output and the actual database query might aid the knowledge base performance.

Finally, solving null event problems due to the lack of general rules and inferencing is, without doubt, essential to a natural language database system. When we must be concerned about computational complexity in a natural language database system, the problem becomes more difficult. I can imagine that it is possible to attach general rules to the entity concepts in the knowledge base model; however, this is not a "*quality response*" to such a profound question. More research effort is required.

It appears that the ideal natural language database system may still not be realized in the near proximity. Research work in this area has been incremental and approaching this ultimate goal. In essense, this thesis has partially demonstrated how a knowledge base can be represented in a relational model, which is homogeneous to the relational database model, and how this knowledge base can be utilized to give better quality responses to null event problem.

Appendix A

Control Flow Diagram of the Algorithm for Detecting a User's Misconception



Procedure Misconception
Procedure Extensional_ Misconception
Procedure Intensional_ Misconception
Procedure Check_ Object_ Existency
Procedure Check_ Always_ Null
Procedure Check_ Temporal_ Event
Procedure Evaluate_ Exception
Procedure Check_ Intensional_ Misconception
Procedure Match_&_ Extract
Procedure Perform_ Intersection
Procedure Respond_ Intensional_ Misconception
Procedure Push_Sub_ In_Stack
Function In_V_ Relation
Procedure Match_ Logical_ Form
Procedure Choose
Function Distribute_ Union

Figure A-1: Top Down Hierarchical Program Structure of Misconception Algorithm

Appendix B
Preliminary Algorithm for Hierarchy Construction

PHASE 1

- Initially let C = {Ø}, where C is a set of lists. Each list in C is in turn a list of two lists: the second list contains names of those relations that have some attribute field(s) in common; and the first list contains those common attribute fields' names. This set of lists in C contains information that will help in the identification of major concepts and their properties in the database.

- Names of those relations that share common attribute fields in their primary keys are grouped together as members of a list, and a list of the common attribute fields as members of a second list. These two lists are then grouped together to be members of another list, say c_i. c_i is then added to C. This procedure is repeated until no new list can be formed. As a result C will become a set of lists, where each list in C is in turn a list of two lists as described in the first step. From Figure 4-1, C will appear as follows:

```
C = {((CNAME)(OFFERING COURSE-PREREQUISITE
              COURSE-PREVIOUS-NAME COURSE-DESCRIPTION)),
     ((SEMESTER#)(OFFERING SEMESTER)),
     ((CLASS#)(CLASS-DEPT TEACH TAKE CLASSROOM)),
     ((INSTRUCTOR#)(INSTRUCTOR TEACH FACULTY-DEPT)),
     ((STUDENT#)(STUDENT TAKE)),
     ((DEPT#)(DEPT-CHAIRMAN DEPT-FACILITY))        }
```

- Notice that a relation can be a member of more than 1 list, like TEACH; and we want a minimal list of attribute fields to identify a concept (object). So in the case of:

```
CLASS-DEPT(CLASS, DEPT, UNITS)
TEACH(CLASS, SEC, INSTRUCTOR, TEXT)
    .
    .
    .
```

we want to have only

```
        ((CLASS)(CLASS-DEPT TEACH ... ))
```

instead of

```
        ((CLASS)(CLASS-DEPT TEACH ... ))
   and
        ((CLASS SEC)(TEACH TAKE ... ))
```

- At this point, we can clearly identify 6 concepts (objects), see Figure B-1.

- *Naming of Concepts*
  Concepts are named according to the domain name(s) of the attribute field(s) of each element of C. If there are more than one common attribute field, names of the domains are concatenated together to form the concept's name.

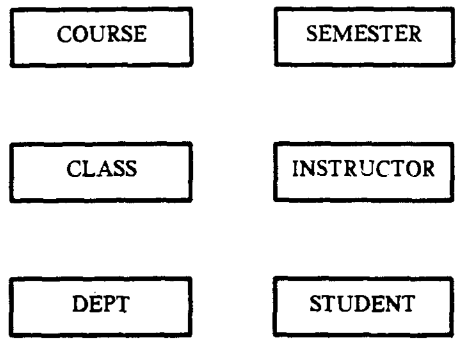


Figure B-1: 6 simple entities identified.

- *CLR relations*
  Each concept will have a CLR (Complete List Relation) name attach to it to indicate where we can get a complete list of the members of the concept. To determine which relation should be the CLR of a concept, we pick the relation for which the concept's identifying attribute field(s) is(are) the relation's only primary key(s).

- The concepts' names together with the corresponding CLR relation are then insert into the ENT-relation.

- *Properties Attachment and Relationship Linking*
  From the previous step, we know that for each concept, c_i, there is a set of relations, R_i, that are related to the concept, and a set of identifying attribute fields, A_i, of the concept. Let R be a relation in the set of relation R_i. This algorithm will distinguish associative entity types from simple entity types, and will attach properties to these two entity types in the following way: If A_i contains all the primary attribute fields of R,

then R is considerd to consist of property attributes of the concept c_i, therefore all the non-key attribute fields of R will be taken as properties of the concept. c_i. Otherwise, R represents a relationship among entity sets represented by its primary keys attribute fileds. This relationship is taken as an AG-relation, and the non-key attribute fields of R will become properties of this relationship in the P-relation.

```
notation: let ELEMENT_i be the ith element of the set C;
          let A_i be the attribute fields in the first list of ELEMENT_i;
          let c_i be the concept of ELEMENT_i identified by
             A_i as described in the previous step;
          let E be the set of concepts identified in the previous step;
          let concept(x) be a function which will return the concept
             corresponding to x, if x is a concept identifier. Otherwise,
             it will just return x.
FOR i FROM 1 TO |C| DO
   FOR (every relation, R, in the 2nd list of ELEMENT_i) DO
   BEGIN
      IF (A_i is not the whole primary key for R) THEN
         IF (there exist non-key attribute fields in R) THEN
            IF (there exist concept(primary key field(s) of R) which are
                not in E, or properties of the corresponding concepts
                in E) THEN
            BEGIN
               - create an AG-relation with a name prefixed by R,
                 and with attribute fields which are concept(A_i)
                 and the concept(the non-key attribute fields of R);
               - insert this AG-relation's name into the AE-relation;
               - insert tuple (c_i, dir-prop_j) into the
                 P-relation, where dir-prop_j is a key attribute of R
                 except A_i, for all non-key attributes of R except A_i;
            END
            ELSE BEGIN
               - create an AG-relation with a name prefixed by R, say R.*,
                 and with attribute fields which are same as
                 the concept(primary key attribute field(s) of R) -
                 those attribute field(s) that are properties of any
                 concept in E.
               - insert this AG-relation's name into the AE-relation;
               - insert tuple (R.*, non-key_j) into the P-relation,
                 where non-key_j is a non-key attribute field in R,
                 for all non-key attributes in R;
            END
         ELSE
            {Attach the rest of the primary key attribute field(s)/
             concept(s) as property(ies) of c_i if this does not
             create duplicate properties.}
            -insert tuple (c_i, prop_j) into the P-relation, where prop_j
             is a primary key attribute field of R that is not found
             in A_i, for all primary key attributes of R that are not
             in A_i;
         IF (there exist foreign key(s), say f, in the primary key) THEN
```

```
            –Remove R from the relation list of the other elements
             of C which have only f in its attribute list.
      ELSE
         {Attach the non-key attribute field(s)/concept(s) to $c_i$
          as properties.}
         –insert the tuple ($c_i$, non-key$_j$) into the P-relation,
          where non-key$_j$ is a non-key attribute field of R,
          for all non-key attribute of R;
      {Attach the attribute field(s) of ELEMENT$_i$ to be property(ies)
       of $c_i$ if it does not create duplicate properties.}
      –insert the tuple ($c_i$, key$_j$) into the P-relation, where
       key$_j$ is an element in $A_i$, for all elements in $A_i$;
   END {for};
```

References

[ANSI 75] Study Group on Data Base Management Systems.
Interim Report.
ANSI, New York, 1975.

[Borgida, Mylopoulos, and Wong 84]
Borgida, A., Mylopoulos, J., Wong, H. K. T.
Generalization/Specialization as a Basis for Software Specification.
On Conceptual Modelling.
Springer-Verlag, New York, 1984, pages 87-117.

[Brachman 79] Brachman, R.
On the Epistemological Status of Semantic Networks.
Associative Networks: The Representation and Use of Knowledge by Machine.
Academic Press, New York, 1979, pages 3-50.

[Cercone and McCalla 86]
Cercone, N., and McCalla, G.
Accessing Knowledge through Natural Language.
Advances in Computers, 25th Anniversary Issue .
Academic Press, New York, 1986, pages 1-99.

[Cercone, Hadley, Martin, McFetridge, and Strzalkowski 84]
Cercone, N., Hadley, R., Martin, F., McFetridge, P., and Strzalkowski, T.
Designing and Automating the Quality Assessment of a Knowledge-Based System: The Initial Automated Academic Advisor Experience.
In *IEEE, Proceedings of the Workshop on Principles of Knowledge-Based Systems*, pages 193-204. Denver, Colorado, 1984.

[Codd 79] Codd, E.
Extending the Database Relational Model to Capture More Meaning.
ACM Transactions on Database Systems 4(4):pages 397-434, 1979.

[Dankel 79] Dankel, D.D.
Browsing in Large Data Base.
In *IJCAI83, Proceedings of the 6th International Joint Conference on Artificial Intelligence*, pages 188-190. Tokyo, Japan, 1979.

[Davidson 82] Davidson, J.
Natural Language Access to a Database: User Modelling and Focus.
In *Proceedings of the 4th National Conference of the CSCSI/SCEIO*, pages 204-211. Saskatoon, 1982.

[Davis and King 75]
Davis, R., King, J.
An Overview of Production Systems.
Technical Report STAN-CS-75-524, Computer Science Department, Stanford University, Stanford, 1975.

[Finin, Goodman, and Tennant 79]
Finin, T., Goodman, B., Tennant, H.
JETS: Achieving Completeness Through Coverage and Closure.
In *IJCAI79, Proceedings of the 6th International Joint Conference on Artificial Intelligence*, pages 275-281. Tokyo, Japan, 1979.

[Gallaire, King, Mylopoulos, Reiter, and Webber 83]
Gallaire, H., King, J.J., Mylopoulos, J., Reiter, R., Webber, B.L.
A Panel on A.I. and Databases.
In *IJCAI83, Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 1199-1206. Karlsruhe, West Germany, 1983.

[Grice 75] Grice, H.P.
Logic and Conversation.
Syntax and Semantics.
Academic Press, New York, 1975.

[Hendrix, Sacerdoti, Sagalowicz, and Slocum 78]
Hendrix, G., Sacerdoti, E., Sagalowicz, D., and Slocum, J.
Developing a Natural Language Interface to Complex Data (LIFER).
ACM Transactions on Database Systems 3(2):pages 105-147, 1978.

[Janas 79] Janas, J.M.
How to not say 'NIL'.
In *IJCAI79, Proceedings of the 6th International Joint Conference on Artificial Intelligence*, pages 429-434. Tokyo, Japan, 1979.

[Kalita 84] Kalita, J.
Generating Summary Responses to Natural Language Database Queries.
Technical Report TR 84-9, Department of Computational Science, University of Saskatchewan, Saskatoon, 1984.

[Kalita, Jones, and McCalla 86]
Kalita, J. K., Jones, M. L., McCalla, G. I.
Summarizing Natural Language Database Responses.
Journal of Computational Linguistics 12:pages 107-124, 1986.

[Kaplan 78] Kaplan, J.
Cooperative Responses from a Portable Natural Language Data Base Query System.
PhD thesis, University of Pennsylvania, 1978.

[Mays 82] Mays, E.
Monitors as responses to questions: determining competence.
In *Proceedings of the 1982 National Conference on Artificial Intelligence.* 1982.

[McCoy 82] McCoy, K.
Augmenting a Database Knowledge Representation for Natural Language Generation.
In *Proceedings of the 20th ACL*, pages 121-128. Toronto, Ontario, 1982.

[McKeown 82] McKeown, K.
The TEXT System for Natural Language Generation: An Overview.
In *Proceedings of the 20th ACL*, pages 113-120. Toronto, Ontario, 1982.

[Motro 86] Motro, A.
Query Generalization: A Method or Interpreting Null Answers.
In *IEEE Proceedings of the Workshop on Expert Database Systems, L. Kerschberg editor*, pages 597-616. The Benjamin/Cummings Publishing Co., Columbia, South Carolina, 1986.

[Reiter 84] Reiter, R.
Towards a Logical Reconstruction of Relational Database Theory.
On Conceptual Modelling.
Springer-Verlag, New York, 1984, pages 191-238.

[Schank and Lehnert 79]
Schank, R., and Lehnert, W.
The Conceptual Content of Conversation.
In *IJCAI79, Proceedings of the 6th International Joint Conference on Artificial Intelligence*, pages 769-771. Tokyo, Japan, 1979.

[Schubert, Goebel, and Cercone 79]
Schubert, L., Goebel, R., and Cercone, N.
The Structure and Organization of a Semantic Network for Comprehension and Inference.
Associative Networks: The Representation and Use of Knowledge by Machine.
Academic Press, New York, 1979, pages 121-175.

[Stonebraker 84] Stonebraker, M.
Adding Semantic Knowledge to a Relational Database System.
On Conceptual Modelling.
Springer-Verlag, New York, 1984, pages 333-353.

[Ullman 82] Ullman, J. D.
Principles of Database Systems.
Computer Science Press, Inc., Maryland, 1982.

[Waltz 78] Waltz, D.
PLANES: An English Language Question-Answering System for a Large Relational Database.
CACM 21(7):pages 105-147, 1978.

[Webber, Joshi, Mays, and McKeown 83]
Webber, B., Joshi, A., Mays, E., and McKeown, K.
Extended Natural Language Database Interactions.
Computers and Mathematics Special Issue on Computational Linguistics.
Pergamon Press, London, 1983, pages 233-244.

[Woods 77] Woods, W. C.
A Personal View of Natural Language Understanding.
SIGART Newsletter :pages 17-20, February, 1977.